FEAST

Finite Element Analysis & Solution Tools

Technical Documentation

Ch. Becker, S. Buijssen, M. Grajewski,

S. Kilian, S. Turek, H. Wobker

March 9, 2006

Institut für Angewandte Mathematik & Simulation, Universität Dortmund, Germany
telephone: +49 231 755 5934 fax: +49 231 755 5933
WWW: http://www.featflow.de/feast, Email:feast@mathematik.uni-dortmund.de

Contents

Chapter 1

Introduction

Current trends in the software development for the numerical solving of Partial Differential Equations (PDEs), and here in particular for Finite Element (FEM) approaches, go clearly towards object oriented techniques and adaptive methods in any sense. Hereby the employed data and solver structures, and especially the matrix structures, are often in contradiction to modern hardware platforms. As a result, the observed computational efficiency is far away from expected peak rates of several GFLOP/s nowadays, and the "real life" gap will even further increase. Since high performance calculations may be only reached by explicitly exploiting "caching in" and "pipelining" in combination with sequentially stored arrays (using special machine optimised linear algebra libraries), the corresponding realisation seems to be "easier" for simple Finite Difference approaches. So, the question arises how to perform similar techniques for much more sophisticated Finite Element codes.

These discrepancies between modern mathematical approaches and computational demands for highly structured data often lead to unreasonable calculation times for "real world" problems, e.g. *Computational Fluid Dynamics* (CFD) calculations in 3D, as can be seen from recent benchmarks [14] for commercial as well as research codes. Hence, strategies for efficiency enhancement are necessary, not only from the mathematical (algorithms, discretisations) but also from the software point of view. To realise some of the aformentioned necessary improvements, we develope the new Finite Element package (**FEAST** – **F**inite **E**lement **A**nalysis & **S**olution **T**ools). This package is based on the following concepts:

- (recursive) "Divide and Conquer" strategies,
- hierarchical data, solver and matrix structures,
- ScaRC as generalization of multigrid and domain decomposition techniques,
- frequent use of machine optimised linear algebra routines,
- all typical Finite Element facilities included.

The result is a flexible software package with special emphasis on:

- (closer to) peak performance on modern processors,
- typical multigrid behaviour w.r.t. efficiency and robustness,
- parallelisation tools directly included on low level,
- open for different adaptivity concepts (r- and h-adaptivity),
- low storage requirements,
- application to many "real life" problems possible.

Figure 1 shows the general structure of the FEAST package:

As programming language Fortran (77 and 90) is used. The explicit use of the two Fortran dialects arises from following observations. For Fortran77 very efficient and well tried compilers are available which allow to exploit much of the machine performance. Furthermore, it is possible to reuse many reliable parts of the predecessor packages FEAT2D, FEAT3D and FEATFLOW [4]. On the other hand Fortran77 is not more than a better "macro assembler", the very limited language constructs make the project work very hard. In addtition to that, F77 is no longer the current standard, so support of this language will stop in near future. And which developer can be motivated to program in F77?

F90 on the other hand is the new standard and provides new helpful features like records, dynamic memory allocation, etc. But there are several disadvantages. The language is very overloaded and the realisation of some features like pointers is not succeeded. More severe, the compilers for Fortran 90 have not reached the amount of stability and robustness yet like their Fortran 77 counterparts.

The compromise is to implement the time critical routines from the numerical linear algebra in F77, while the administrative routines are based on F90.

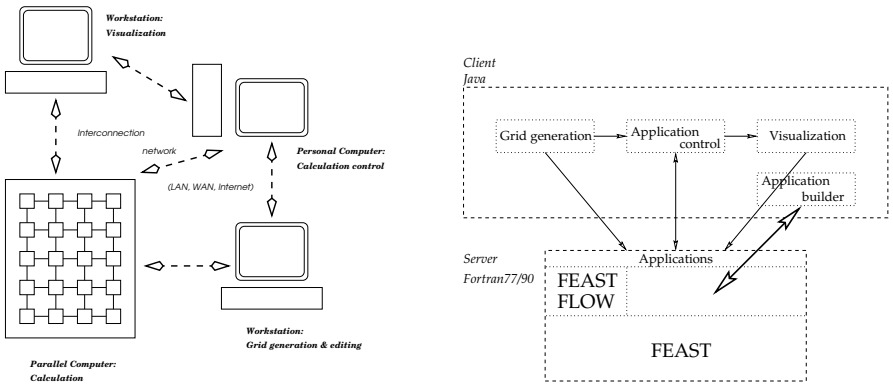

Figure 1.1: FEAST structure and configuration

The pre– and postprocessing is mainly handled by Java based program parts. Configuring a high performance computer as a FEAST server, the user shall be able to perform the remote calculation by a FEAST client.

In the following we give examples for "real" computational efficiency results of typical numerical tools which help to motivate our hierarchical data, solver and matrix structures. For a better understanding, we illustrate shortly in the subsequent chapter the corresponding solution technique ScaRC (**Sca**lable **R**ecursive **C**lustering) in combination with the overall "Divide and Conquer" philosophy. This solver concept forms an essential part of FEAST. We discuss how typical multigrid rates can be achieved on parallel as well as sequential computers with a very high computational efficiency.

The typical situation in high performance computing today shows the following situation:

Example: STAR-CD ($k-\epsilon$), 500 000 cells (by Daimer Chrysler), SGI Origin2000 (6 CPUs), 6.5 CPU h

Quantity	**Experiment**	**Simulation**	**Difference**
Drag ('c_w')	0.165	0.317	**92 %**
Lift ('c_a')	-0.083	-0.127	**53 %**

Table 1.1: Star-CD Experiment and Simulation

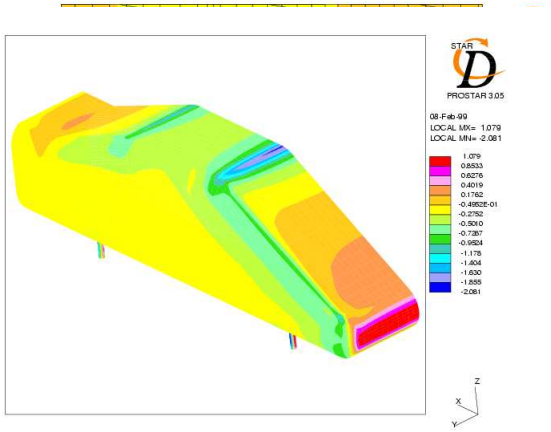

Figure 1.2: Star-CD Simulation

1.1 Main Principles in FEAST

1.1.1 Hierarchical data, solver and matrix structures

One of the most important principles in FEAST is to apply consequently a *(Recursive) Divide and Conquer* strategy. The solution of the complete "global" problem is recursively split into smaller "independent" subproblems on "patches" as part of the complete set of unknowns. Thus the two major aims in this splitting procedure which can be performed by hand or via self–adaptive strategies are:

- *Find locally structured parts.*
- *Find locally anisotropic parts.*

Based on "small" structured subdomains on the lowest level (in fact, even one single or a small number of elements is allowed), the "higher–level" substructures are generated via clustering of "lower–level" parts such that algebraic or geometric irregularities are hidden inside the new "higher-level" patch. Additional background information on this strategy is given in the following sections which describe the corresponding solvers related to each stage.

Figures 1.3 and 1.4 illustrate exemplarily the employed data structure for a (coarse) triangulation of a given domain and its recursive partitioning into several kinds of substructures.

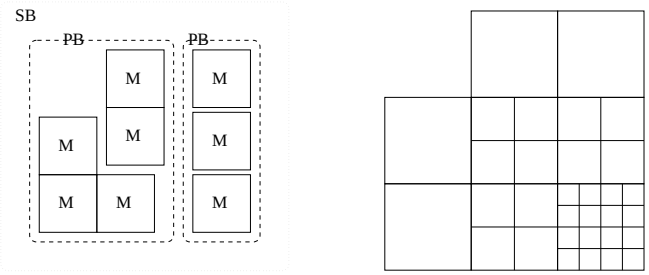

Figure 1.3: FEAST domain structure

According to this decomposition, a corresponding data tree – the skeleton of the partitioning strategy – describes the hierarchical decomposition process. It consists of a specific collection of elements, macros, matrix blocks (MB), parallel blocks (PB), subdomain blocks (SB), etc.

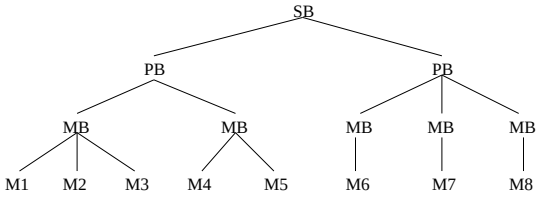

Figure 1.4: FEAST data tree

The *atomic units* in our decomposition are the "macros" which may be of type **structured** (as $n \times n$ collection of quadrilaterals (in 2D) with local band matrices) or **unstructured** (any collection of elements, for instance in the case of fully adaptive local grid refinement). These "macros" (one or several) can be clustered to build a "matrix block" which contains the "local matrix parts": only here is the complete matrix information stored. Higher level constructs are "parallel blocks" (for the parallel distribution and the realisation of the load balancing) and "subdomain blocks" (with special conformity rules with respect to grid refinement and applied discretisation spaces). They all together build the complete domain, resp. the complete set of unknowns. It is important to realise that each stage in this hierarchical tree can act

as independent "father" in relation to its "child" substructures. At the same time, it can act as a "child" another phase of the solution process (inside of the ScaRC solver, see below).

According to this data structure and with respect to the following solver engine Scarc in FEAST every vector belongs to one hierachical layer Subdomain (SD), Parallelblock (PB) and Matrixblock (MB). This concerns the way of data exchange and synchronisation. If you e.g. perform an exchange operation to a PB-vector, then the vector is modified only on the PB boundaries to have the same values, but not on the outer boundaries.

1.1.2 Generalised solver strategy ScaRC

In short form our long time experience with the numerical and computational runtime behaviour of typical multigrid (MG) and Domain Decomposition (DD) solvers can be concluded as follows:

Some observations from standard multigrid approaches:

While in fact the numerical convergence behaviour of (optimised) multigrid is very satisfying with respect to robustness and efficiency requirements, there still remain some "open" problems: often the parallelisation of powerful recursive smoothers (like SOR or ILU) leads to performance degradations since they can be realised only in a blockwise sense. Thus it is often not clear how the nice numerical behaviour in sequential codes for complicated geometric structures or local anisotropies can be reached in parallel computations. And additionally, the communication overhead especially on coarser grid levels dominates the total CPU time. Even more important is the "computational observation" that the realised performance on modern platforms is often far beyond (sometimes less than 1 %) the expected peak performance. Many codes often reach much less than 50 MFLOP/s, and this on computers which are said (by the vendors) to run with several GFLOP/s peak performance. The reason is simply that the single components in multigrid (smoother, defect calculation, grid transfer) perform too few arithmetic work with respect to each data exchange such that the facilities of modern superscalar architectures are poorly exploitable. In contrast, we will show that in fact 30 – 70 % are within reach with appropriate techniques.

Some observations from standard Domain Decomposition approaches:

In contrast to standard multigrid, the parallel efficiency is much higher, at least as long as no large overlap region between processors must be exchanged. While *overlapping* DD methods do not require additional coarse grid problems (however the implementation in 3D for complicated domains or for complex Finite Element spaces is a hard job), *non-overlapping* DD approaches require certain coarse grid problems, as the BPS preconditioner for instance which may lead again to several numerical and computational problems, depending on the geometrical structure or the used discretisation spaces. However the most important difference between Domain Decomposition and multigrid are the (often) much worse convergence rates of DD although at the same time more arithmetic work is done on each processor.

As a conclusion improvements are enforced by the facts that the **convergence behaviour** is often quite sensitive with respect to (local) geometric/ algebraic **anisotropies** (in "real life" configurations), and that the performed **arithmetic work** (which allows the high performance) is often restricted by (un)necessary **data exchanges**.

An additional observation which is strongly related to the previous data structure in combination with the specific hierarchical ScaRC solver is illustrated in the following figure. We show the resulting "optimal" mesh from a numerical simulation of R.Becker/R.Rannacher for "Flow around the cylinder" which was adaptively refined via rigorous a–posteriori error control mechanisms specified for the required drag coefficient ([10]).

As can be seen the adaptive grid refinement techniques are needed only locally, near the boundaries, while mostly regular substructures (up to 90 %) can be used in the interior of the domain. This is a quite typical result and shows that even for (more or less) complex flow simulations (here as a prototypical

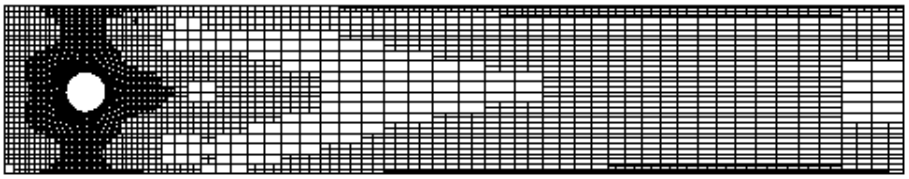

Figure 1.5: "Optimal grid" via a–posteriori error estimation

example) locally blockwise structured meshes can be applied: these regions can be detected and exploited by the given hierarchical strategies.

The ScaRC approach consists of a separated multigrid approach for every hierarchical layer, whereby the multigrid scheme on the outest layer (subdomain layer) gives the final result. The smoothing step of the multigrid method is based on the following notation:

Smoothing on level h for $A_h x = b_h$:

- **global** outer block Jacobi scheme (with averaging operator 'M')

$$x^{l+1} = x^l - \omega_g \tilde{A}_{h,M}^{-1}(A_h x^l - b_h)$$

  with $\tilde{A}_{h,M}^{-1} := M \circ \tilde{A}_h^{-1}$, $\tilde{A}_h^{-1} := \sum_{i=1}^{N} \tilde{A}_{h,i}^{-1}$, $\tilde{A}_{h,i} :=$ "$A_{h|\Omega_i}$"

- "solve" **local** problems $\tilde{A}_{h,i} y_i = def_i^l := (A_h x^l - b_h)_{|\Omega_i}$ via

$$y_i^{k+1} = y_i^k - \omega_l C_{h,i}^{-1}(\tilde{A}_{h,i} y_i^k - def_i^l)$$

  with $C_{h,i}^{-1}$ preconditioner for $\tilde{A}_{h,i}$, or employ a direct solver

The local smoothing operators can be a further multigrid scheme or any other scheme like Jacobi, Gauß–Seidel, ADI or ILU. The choice of the method depends on the local structure and the numerical difficulties caused by the given domain. In a first step this decision is taken by the user to choose explicitly the method but in future it is possible to use an "expert system" which makes this decision widely automatically.

There are several reasons why we explicitely use **this** *basic iteration*:

1. This general form allows the splitting into matrix–vector multiplication, preconditioning and linear combination. All 3 components can be separately performed with high performance tools if available.
2. The explicit use of the complete defect $A_h x^l - b_h$ is advantageous for certain techniques for implementing boundary conditions (see [12]).
3. All components in standard multigrid, i.e., smoothing, defect calculation, step–length control, grid transfer, are included in this *basic iteration*.

Figure 1.6 shows an example illustration of a ScaRC scheme.

The notation ScaRC stands for:

- **Scalable**, w.r.t. the number of global ('l') and local solution steps ('k'),
- **Recursive**, since it may be applied to more than 2 global/local levels,

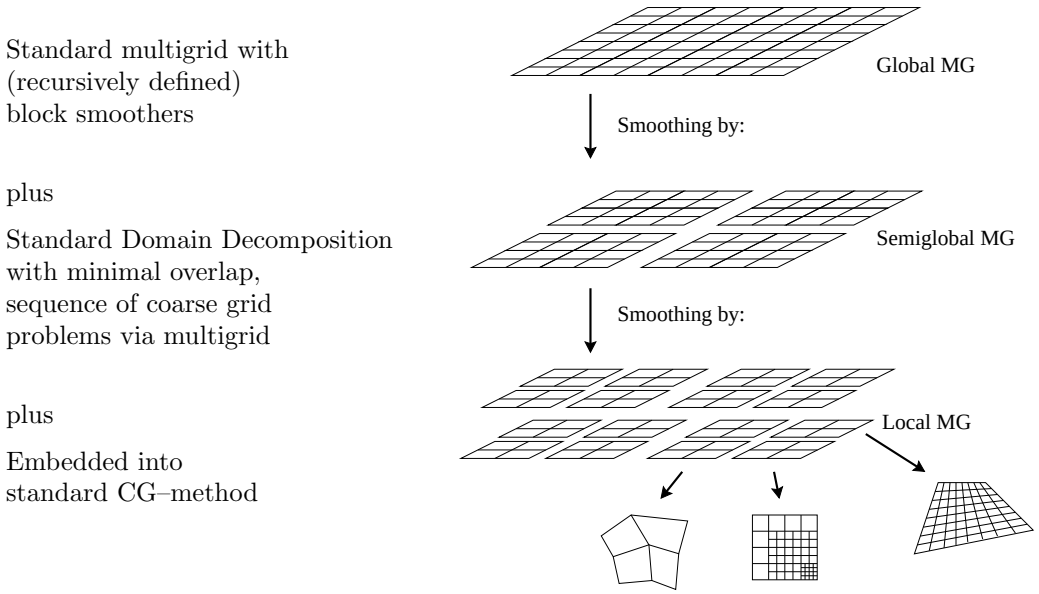

Figure 1.6: ScaRC scheme: example

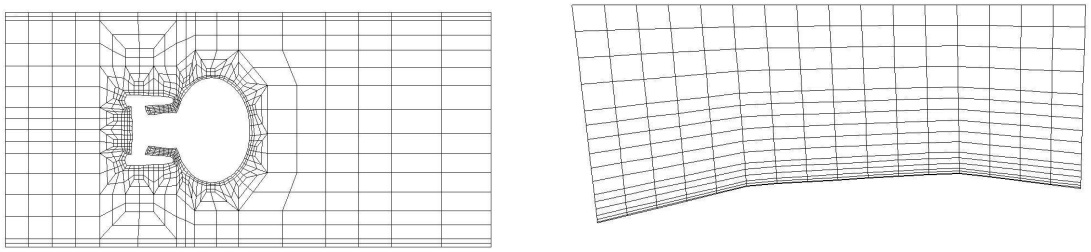

Figure 1.7: Example of ScaRC: 2D decomposition and zoomed (macro) element (LEVEL 3) with locally anisotropic refinement towards the wall

- **Clustering**, since fixed or adaptive blocking of substructures is possible.

Results for a ScaRC-CG solver (smoothing steps: 1 global ScaRC; 1 local 'MG-TriGS') for locally (an)isotropic refinement are shown in table 1.2.

$\#NEQ$	Dirichlet 'Velocity'		Neumann 'Pressure'	
	$AR \approx 10$	$AR \approx 10^6$	$AR \approx 10$	$AR \approx 10^6$
$210,944$	0.17 (**8**)	0.18 (**8**)	0.21 (**9**)	0.15 (**8**)
$843,776$	0.17 (**8**)	0.17 (**8**)	0.20 (**9**)	0.17 (**8**)
$3,375,104$	0.18 (**9**)	0.19 (**9**)	0.22 (**10**)	0.22 (**10**)
$13,500,416$	0.19 (**9**)	0.18 (**9**)	0.23 (**10**)	0.23 (**10**)

Table 1.2: Global (parallel) convergence rates for ScaRC on configuration shown in 1.7

For more information about ScaRC see [9] and [8].

1.1.3 High Performance Linear Algebra

One of the main ideas behind the described *(Recursive) Divide and Conquer* approach in combination with the ScaRC solver technology is to detect "locally structured parts". In these "local subdomains" we apply consequently "highly structured tools" as known from Finite Difference approaches: line– or rowwise numbering of unknowns and storing of matrices as sparse bands (however the matrix entries are calculated via the Finite Element modules). As a result we have "optimal" data structures on each of these patches (which often correspond to the former introduced "matrix blocks") and we can perform very powerful linear algebra tools which explicitly exploit the high performance of specific machine optimised libraries.

We have performed several tests for different tasks and techniques in numerical linear algebra on some selected hardware platforms. In all cases we attempted to use "optimal" compiler options and machine optimised linear algebra libraries.

Matrix–vector multiplication:

We examine more carefully the following variants which all are typical in the context of iterative schemes with sparse matrices. The test matrix is a typical 9–point stencil (emerging from a FE discretisation of the Poisson operator with bilinear Finite Elements). We perform tests for two different vector lengths N and give the measured MFLOP rates which are all calculated via $20 \times N/time$ (for MV), resp., $2 \times N/time$ (for DAXPY).

Sparse MV: SMV
The *sparse MV* technique (see 1) is the standard technique in Finite Element codes (and others), also well known as "compact storage" technique or similar: the matrix plus index arrays or lists are stored as long arrays containing the nonzero elements only. While this approach can be applied for arbitrary meshes and numberings of the unknowns, no explicit advantage of the linewise numbering can be exploited. We expect a massive loss of performance with respect to the possible peak rates since — at least for larger problems — no "caching in" and "pipelining" can be exploited such that the higher cost of memory access will dominate the resulting MFLOP rates. Results are shown in table 1.3.

```
                DO 10 IROW=1,N
                DO 10 ICOL=KLD(IROW),KLD(IROW+1)-1
10              Y(IROW)=DA(ICOL)*X(KCOL(ICOL))+Y(IROW)
```

Sample 1: Standard sparse matrix vector algorithm(DAXPY indexed)

Computer	#Unknowns	**CM**	**TL**	**STO**
	8,320	147	136	**116**
	33,280	125	105	**100**
Alpha ES40	133,120	81	71	**58**
(667 Mhz)	532,480	60	51	**21**
	2,129,920	58	47	**13**
	8,519,680	58	45	**10**

Table 1.3: Performance rates of the FEATFLOW code with different numbering schemes (Cuthill–McKee, TwoLevel, Stochastic) for matrix vector multiplication

Banded MV: BMV
A "natural" way to improve the *sparse MV* is to exploit that the matrix consists of 9 bands only. Hence the matrix–vector multiplication is rewritten such that now "band after band" are applied. The obvious

advantage of this *banded MV* approach is that these tasks can be performed on the basis of BLAS1–like routines which may exploit the vectorisation facilities of many processors (particularly on vector computers). However for "long" vector lengths the improvements can be absolutely disappointing: For the recent workstation/PC chip technology the processor cache dominates the resulting efficiency!

Banded blocked MV: BBMVA, BBMVL, BBMVC
The final step towards highly efficient components is to rearrange the matrix–vector multiplication in a "blockwise" sense: for a certain set of unknowns, a corresponding part of the matrix is treated such that cache–optimised and fully vectorised operations can be performed. This procedure is called "BLAS 2+"–style since in fact certain techniques for dense matrices which are based on ideas from the BLAS2, resp., BLAS3 library, have now been developed for such sparse banded matrices. The exact procedure has to be carefully developed in dependence of the underlying FEM discretisation, and a more detailed description can be found in [2].

While BBMVA has to be applied in the case of arbitrary matrix coefficients, BBMVL and BBMVC are modified versions which can be used under certain circumstances only (see [2] for technical details). For example PDE's with constant coefficients as the Poisson operator but on a mesh which is adapted in one special direction only, allow the use of BBMVL: This is often the case for the Pressure–Poisson problem in flow simulations on boundary adapted meshes. Additionally version BBMVC may be applied for PDE's with constant coefficients on meshes with equidistant mesh distribution in each (local) direction separately: This is typical for tensor product meshes in the interior domain where the solution is mostly smooth.

Performance results for banded techniques in comparation to standard techniques are shown in table 1.4.

As example the Poisson problem with multigrid with TriGS smoother on NCC-1701D grid is calculated, corresponding performance results are shown in table 1.5.

2D case	N	DAXPY(I)	MV		MG-TriGS	
			V	C	V	C
Sun V40z	65^2	1521 (422)	1111	1605	943	1178
(1800 MHz)	257^2	264 (106)	380	1214	446	769
'**Opteron**'	1025^2	**197 (54)**	**362**	**1140**	**333**	**570**
DEC 21264	65^2	205 (178)	538	795	370	452
(667 MHz)	257^2	224 (110)	358	1010	314	487
'**ES40**'	1025^2	**78 (11)**	**158**	**813**	**185**	**401**
NEC SX-6	65^2	1170 (422)	1204	1354	268	341
(500 MHz)	257^2	1100 (412)	1568	2509	316	459
'**Vector**'	1025^2	**1120 (420)**	**1597**	**3421**	**339**	**554**
IBM POWER4	65^2	1521 (845)	2064	3612	1386	1813
(1700 MHz)	257^2	1100 (227)	1140	3422	1048	1645
'**Power**'	1025^2	**390 (56)**	**550**	**2177**	**622**	**1138**

Table 1.4: Performance rates for the operations DAXPY, DAXPY-I, MV and MG-TriGS for different arhitectures

More information in available in [1], [3] and [13].

As a summary, we can draw the following conclusions:

- Sparse techniques are the basis for most of the recent software packages.

N	1p	2p	3p	4p
843,776	11.04(191)	5.72(368)	3.85(547)	3.45 (611)
3,381,507	30.36(271)	15.62(526)	10.73(766)	8.42 (976)
13,513,219	98.64(328)	51.04(634)	34.79(931)	27.80(1165)
54,027,267	367.85(301)	198.35(559)	129.07(859)	107.70(1029)

Table 1.5: CPU times and numerical MFlop rates for different numbers of CPUs (Sun V40z, four Opteron 844 CPUs with 1800Mhz, 16 GByte memory)

- Different numberings can lead to identical numerical results and work (w.r.t. arith.ops and data accesses) but to huge differences in CPU time.
- Sparse techniques are 'slow'. The computational speed depends on the problem size and the kind of data access.

1.1.4 Several adaptivity concepts

As typical of modern FEM packages, we directly incorporate certain tools for grid generation which allow easy handling of local and global refinement or coarsening strategies: **adaptive mesh moving**, **macro adaptivity** and **fully local adaptivity**.

Adaptive strategies for moving mesh points, along boundaries or inner structures, allow the same logic structure in each "macro block", and hence the shown performance rates can be preserved. Additionally, we work with adaptivity concepts related to each "macro block". Allowing "blind" or "slave macro nodes" preserves the high performance facilities in each "matrix block", and is a good compromise between fully local adaptivity and optimal efficiency through structured data. Only in that case, that these concepts do not lead to satisfying results, certain macros will loose their "highly structured" features through the (local) use of fully adaptive techniques. On these (hopefully) few patches, the standard "sparse" techniques for unstructured meshes have to be applied.

1.1.5 Direct integration of parallelism

Most software packages are designed for sequential algorithms to solve a given PDE problem, and the subsequent parallelisation of certain methods takes often unproportionately long. In fact it is easy to say, but hard to realise with most software packages. However the more important step, which makes parallelisation much more easier, is the design of the ScaRC solver according to the hierarchical decomposition in different stages. Indeed from an algorithmic point of view, our sequential and parallel versions differ only as analogously Jacobi– and Gauß–Seidel–like schemes work differently. Hence all parallel executions can be identically simulated on single processors which however can additionally improve their numerical behaviour with respect to efficiency and robustness through Gauß–Seidel–like mechanisms.

Hence we only provide in FEAST the "software" tools for including parallelism on low level, while the "numerical parallelism" is incorporated via our ScaRC solver and the hierarchical "tree structure". However what will be "non–standard" is our concept of (adaptive) parallel loadbalancing which is oriented in "total numerical efficiency" (that means, "how much processor work is spent to achieve a certain accuracy, depending on the local configuration") in contrast to the "classical" criterion of equilibrating the number of local unknowns (see [2] for detailed information and examples in FEAST).

1.2 Pre– and Postprocessing

1.2.1 General remarks

As remarked in the introduction the pre– and postprocessing should be realised in main tasks by a general framework of Java based programs called DeViSoR. DeViSoR means "Design & Visualisation Software Resource". This framework is intented to perform the main tasks grid generation and editing, control of the calculation and visualisation of the results. These main tasks use the same ground classes (called DFC - DeViSoR Foundation Classes) and the same user interface, so the access to the underlying numerical core parts are performed in the same manner.

As intented in the introduction the various subtasks can be performed on several machines which communicates over a network system. This allows the user to choose the suitable system for the corresponding task, e.g. a Silicon Graphics workstation for the visualisation. The access to a parallel computing system should also be performed by a Java program. This allows not only the developer of a numerical code to use a parallel computer.

DeViSoR is planned to be an "open system" for the developing of pre– and postprocessing tools for FEM packages. The DeViSoR foundation classes contain the basic tools to handle and administrate FEM typical structures. Further applications could realise e.g. further visualisation procedures and adaptions to several parallel computer systems.

For this project Java as implementation environment is been choosen. Though Java is a relative "young" programming language the advantages of this system are significant. The "write once, run anywhere" capability reduces the implementation effort widely against combinations like C/C++/OpenGL. It exist only one program which runs without any modification on several different configurations like Unix workstations, Linux PCs, Windows PCs, Macintoshs and many more. A further advantage is the core class library for various subareas like file handling, network functions, visualisation and user interface facilities. These classes are easy to use and produce an pleasing output. The use of additional tools like applications builders is not necessary. The most disadvantage of nowadays Java implementations is the relative low performance because of the fact that Java is an interpreted language. However further developments like more sophisticated interpreter with Just–In–Time compiling facilities and especially the native Java processor will hopefully close this performance leak.

More information you can find in [7].

1.2.2 Preprocessing: DeViSoRGrid

This subprogram should support the generation and editing of 2D domains. The two main parts are the description of the domain boundary and the generation of the grid structure. The program supports several boundary elements like lines, arcs and splines, further it is planned to add a segment which consists of an Fortran subroutine which describes a parametrisation. This allows to use an analytic description. Several triangular and quadrilateral elements are supported. Extensive editing possibilities allow the user to delete, move and adjust the boundaries and elements. For the future it is planned to implement simply automatic grid generators for producing coarse grids. As further tasks this program should be able to read many formats from other tools like CAD systems and professional grid generators and prepare this data for the use in the calculation process.

1.2.3 Processing: DeViSoRControl

DeViSoRControl enables the user to control the calculation und to follow the calculation progress. Main tasks of this program part are the distribution of the macros to the processing nodes (at the moment manually, in future automatically), the collecting and displaying of the log information of the processing nodes and finally the configuration of the ScaRC algorithm with respect to the selection of smoothing/preconditioning methods on a given hierarchical layer, the size of smoothing steps and the stopping

criterion. Furthermore this part builds the interface to the other DeViSoR parts for the pre– and post-processing. From the control part the grid program is invoked, a grid is editing. For this grid the user selects the desired solution method and visualise finally the results with the DeViSoRVision program.

1.2.4 Postprocessing: DeViSoRVision

The last part in the current project is the DeViSoRVision program which performs the visualisation task. The program offers several techniques to visualise the results of the calculation like shading techniques, isolines, particle tracing (planned). Furthermore it contains an animation module to create animations for nonstationary problems. The result of the animation can be stored in several formats like MPEG and AnimatedGIF.

Chapter 2

Installation

Please perform the following prelimarary installation steps:

get `feast_xx.xx.xx.tar.gz` from `www.featflow.de`

Untar the file with the command

```
gzip -d feast_xx.xx.xx.tar.gz
tar xvf feast_xx.xx.xx.tar
```

This creates the following directory structure

feast	feast	-FEAST program-
	doc.pdf	-This document-
	README	-latest information-
	CHANGES	-changes from previous versions-
	OPENBUGS	-known and open bugs-

The feast folder itself consists of the following subfolders

`applications`	application programs
`arch`	architecture dependent stuff
`bin`	shell scripts and binaries
`docs`	Documentation
`fbenchmark2`	benchmark program
`grids`	domain descriptions
`kernel`	FEAST kernel sources
`libraries`	external libraries
`object`	folder for object files
`outofdata`	stuff out of date
`sbblas`	SBBLAS folder
`sbblasbench`	SBBLAS benchmark folder
`scarc`	folder containing the scarc algorithm descriptions
`solver`	folder containing the solver algorithm descriptions

Get LAM-MPI, a message passing library, from

`http://www.lam-mpi.org/download`

and install it in accordance with the instructions found in the `LAM` folder (for lsiii users: already installed) [5, 11]

Make sure that the environment variables `MPI_INC` and `MPI_LIB` point to the directories for include files and the LAM/MPI libraries, e.g.

```
setenv MPI_INC /usr/local/lam/include
setenv MPI_LIB /usr/local/lam/lib
```

Make sure, that the path `/usr/local/lam/bin` is in your `PATH` variable.

FEAST has a global configure script located in the `bin` folder. Every application subfolder contains a local configure script which calls the global script with application specific options.

`configure` tries to detect the system architecture. The architecture is coded as following:

`arch-cpu-os-mpi-compiler-blas`

The sub specifications means as follows:

- `arch`: architecture, currently `alpha,ibm,pc,sun4u,sx6`
- `cpu`: processor type, currently `athlon,athlon64,athlonxp,ev6,opteron,pentium3,pentium4,pentium4m,powerpc_power4,sparcv8,sparcv9`
- `os`: operating system, currently `aix,linux32,linux64,osf1,sunos,superux`
- `mpi`: MPI environment, currently `dmpi,lammpi,mpich,optmpich,tsmpi`
- `compiler`: compiler, currently `cf90,ifort,g95,gfortran,pgi,sunf90,xlf`
- `blas`: used BLAS library, currently `atlas,blas,dxml,essl,goto,perf`

Currently, the following IDs are supported:

```
alpha-ev6-osf1-dmpi-cf90-atlas,
alpha-ev6-osf1-dmpi-cf90-blas,
alpha-ev6-osf1-dmpi-cf90-dxml,
alpha-ev6-osf1-lammpi-cf90-atlas,
alpha-ev6-osf1-lammpi-cf90-blas,
alpha-ev6-osf1-lammpi-cf90-dxml,
ibm-powerpc_power4-aix-poempi-xlf-essl,
pc-athlon64-linux32-lammpi-ifort-blas,
pc-athlonxp-linux32-lammpi-g95-blas,
pc-athlonxp-linux32-lammpi-gfortran-blas,
pc-athlonxp-linux32-lammpi-ifort-atlas,
pc-athlonxp-linux32-lammpi-ifort-blas,
pc-athlonxp-linux32-lammpi-pgi-blas,
pc-opteron-linux32-lammpi-g95-blas,
pc-opteron-linux32-lammpi-g95-goto,
pc-opteron-linux32-lammpi-gfortran-goto,
pc-opteron-linux32-lammpi-ifort-goto,
pc-opteron-linux32-lammpi-sunf90-blas,
pc-opteron-linux64-lammpi-g95-blas,
pc-opteron-linux64-lammpi-g95-goto,
pc-opteron-linux64-lammpi-gfortran-goto,
pc-opteron-linux64-lammpi-ifort-blas,
pc-opteron-linux64-lammpi-ifort-goto,
pc-opteron-linux64-lammpi-pgi-atlas,
pc-opteron-linux64-lammpi-pgi-blas,
pc-opteron-linux64-lammpi-pgi-goto,
pc-opteron-linux64-lammpi-sunf90-blas,
pc-opteron-linux64-mpich-g95-blas,
pc-opteron-linux64-mpich-psc-goto,
pc-opteron-linux64-optmpich-ifort-goto,
pc-opteron-linux64-tsmpi-ifort-goto,
pc-opteron_ia32-linux64-lammpi-g95-blas,
pc-opteron_ia32-linux64-lammpi-sunf90-blas,
pc-pentium3-linux32-lammpi-ifort-blas,
pc-pentium4-linux32-lammpi-g95-blas,
pc-pentium4-linux32-lammpi-ifort-atlas,
pc-pentium4-linux32-lammpi-ifort-blas,
pc-pentium4m-linux32-lammpi-g95-blas,
pc-pentium4m-linux32-lammpi-ifort-blas,
pc-pentium4m-linux32-lammpi-ifort-goto,
```

```
sun4u-sparcv8-sunos-lammpi-g95-blas,
sun4u-sparcv8-sunos-lammpi-g95-perf,
sun4u-sparcv8-sunos-lammpi-sunf90-blas,
sun4u-sparcv8-sunos-lammpi-sunf90-perf,
sun4u-sparcv9-sunos-lammpi-sunf90-blas,
sun4u-sparcv9-sunos-lammpi-sunf90-perf,
sx6-none-superux-mpi-f90-blas,
sx8-none-superux-mpi-f90-blas
```

The configure script supports the following options:

```
Usage: configure [options]
Default settings are taken from Makefile.inc based on a guessed host id.
Options specified here will override these settings.
Configuration:
--------------
--id=ARCH           Configure for ARCH.
                    For a complete list invoke 'configure --list-all-ids'
--force-id          Every id is accepted even if the system verify fails
--appname=ARG       Set application name to ARG (default: feastmain)
--srclist_app=LIST  Set application's source files to LIST
--mode=KEYWORD      where KEYWORD is one of PARALLEL or SERIAL.
                    Create Makefile for a parallel or serial application
                    (default: parallel).
--[no-]opt          Controls whether to use full compiler optimisation or none
                    at all (but including debugging symbols)
                    (default: full optimisation).
--objdir-prefix=DIR Prepend DIR to paths of object files (default: none)
--[no-]mpiwrappers  Controls whether or not to use MPI wrappers for compiling
                    and linking (default: no).
--[no-]monitor-compile-env
                    where KEYWORD is one of YES or NO.
                    Monitor modification dates of all compilers and compile
                    settings for changes in subsequent compilations. In case of
                    discrepancies, recompile application (default: yes)
--prec=KEYWORD      where KEYWORD is one of DOUBLE or SINGLE.
                    Create Makefile for double precision (default) or single
                    precision arithmetics.
--make=PROG         Set make program to PROG
--makefile=FILE     Name of makefile to create
--f77=PROG          Fortran 77 compiler
--f90=PROG          Fortran 90 compiler
--cc=PROG           C compiler
--cpp=PROG          C++ compiler
--ld=PROG           Linker
--cflagsf77=LIST    Set Fortran 77 compiler flags to LIST
--cflagsf90=LIST    Set Fortran 90 compiler flags to LIST
--cflagsc=LIST      Set C compiler flags to LIST
--cppflags=LIST     Set C preprocessor flags to LIST
--ldflags=LIST      Set linker flags to LIST
--modsuffix=SUFFIX  Specify SUFFIX Fortran 90 compiler uses for
                    module files
--movemod=ARG       If Fortran 90 compiler expects module files in $(OBJDIR) set
                    to YES, otherwise NO
--includedir=DIR    F77/F90/C/C++ header files in DIR
--buildlib=ARG      Specify libraries to build (like: gpu gpuemulate umfpack
                    blas lapack metis)
--libdir=DIR        Object code libraries in DIR
```

```
--libs=ARG          Libraries to include in build
--mpiinc=DIR        Look for MPI header files in DIR
--mpilibdir=DIR     Look for MPI libraries in DIR
--mpilibs=ARG       MPI libraries to link against
--ar=ARG            Path and options for creating archives
--ranlib=ARG        Path and options to create index for archives
--coprocessor=ID    Identifier for hardware coprocessors
--[no-]relink-always
                    Whether or not to relink the application every time make is
                    invoked.
                    (default: yes).
                    Be careful with --no-relink-always:
                    When developing on multiple platforms, your application
                    might not be relinked if no source file has changed since
                    you changed platform!
--debug             Show settings stored into Makefile which will override
                    defaults from Makefile.inc.
                    This flag is *not* an alias for --no-opt nor does it result
                    in a Makefile that creates code with debugging symbols!

Options which cause configure to print only information, while no Makefile will
be created:
--help              Print this message
--version           Show version information
--list-all-ids      Print a list of all valid build target IDs. Use any of them
                    as ARCH for --id option
--list-kernel-modules
                    Print all FEAST kernel files the application is built from
--list-app-modules  Print application's proprietary source files
```

After configure is successfully run a makefile is created. Type `make` to compile the package and the application.

Chapter 3

Tutorial

3.1 A first step to FEAST Finite Element & Solution Tools

In the first step we want to create our grid. For this task we use the DeViSoR program, which you have to load and install separately from `http://www.featflow.de`.

Start DeViSoR with the command

```
grid3
```

First we build a new domain. Go to

`Domain->New Domain`

type `1,0` for every size in the dialog.

then we want to build the boundary description. Press the space bar and select

`Mouse mode->Create SegmentList`,

then go to position (0,0) and klick the left button, go to (1,0),(1,1),(0,1) and klick again. To finish the boundary description press the space bar and select

`Done!`.

Then go to

`Domain->Save Domain As->FEAST V3`

and save the file under the name `~/feast/feast/grids/firstgrid.feast`.

In the next step we will create the nodes. First, we define the nodes on the boundary. This will be done automatically by the the function

`Coarse Mesh->Add multiple parameter nodes(2D)`.

Select this function and a dialog will appear, where you can input some parameters. Change the value of the `Number of Nodes` from 4 to 8 and click OK. This will create boundary nodes on the edge and midpoints. One interior point has to be defined, to do this, press space bar and select

`Mouse mode->Create Points`

and klick with the left button to the position (0.5,0.5).

In the next step you have to define the edges. Press space bar and select

`Select`.

Klick with the left button and draw the rubberband with pressed button so, that the nodes 1 and 2 are gathered. Then press the E key. A new edge is created. Repeat this for the node groups (2,9), (9,8), (8,1), (9,6), (6,7), (7,8), (5,6), (4,2), (9,4), (3,4) and (2,3)). Then gather the complete domain, press space bar and select

`Add Quads`.

Now the grid is ready and you can finally save it.

Now we can try to start a calculation run. Go to the folder

`~/feast/feast/applications/poisson`

and type: `./configure`

This creates the makefile.

Type `make` to build the executable.

After editing and building we have to modify the central data file called `master.dat`. In this file there are several parameters concerning the environment and simulation defined. The detailed format of this file is described later. Make sure that the lines with PROJECTDIR and LOGDIR contains the correct path to your FEAST folder.

Before you start the executable make sure that the LAM daemon is running. To achieve this type the following commands

```
wipe
lamboot
```

to start the program type

```
mpirun -c 4 doenerkebap
```

The program run should start with

```
* master: This is FEAST!  Version 0.9.0 (22.11.2005 RC1)
```

and end with

```
* master: Get finish message from    1 with               0
* master: Get finish message from    2 with               0
* master: Get finish message from    3 with               0
```

Now have a look to the solution files `sol_simple.gmv`. To view this file you will need the program *GMV*.

3.2 Sample program

The following sample program demonstrates the solving of the poisson problem with the FEAST library. This sample program is the slavemod_complex.f90 file in the `~/feast/feast/applications/poisson` folder.

```
!##############################################################################
!# FINITE ELEMENT ANALYSIS & SOLUTION TOOLS  F E A S T  (Release 1.0)   #
!#                                                                            #
!# Authors: Ch.Becker, S.Kilian, S.Turek                                      #
!#          Institute of Applied Mathematics & Simulation                     #
!#          University of Dortmund                                            #
!#          D-44227 DORTMUND                                                  #
!#                                                                            #
!##############################################################################
!#                                                                            #
!# sample program to solve the laplace equation                               #
!#                                                                            #
!#                   - d_xx x - d_yy x  = f                                   #
!#                                                                            #
!#                                                                            #
!##############################################################################

!# Current version: $Id: slavemod_complex.f90,v 1.34 2006/03/02 13:02:18 hwobker Exp $

!basic definitions
!<!--
#include <feastdefs.h>
! -->

module slavemod

  ! module import
  use boundarycondition
  use communication
  use errorcontrol
  use fsystem
  use hlayer
  use io
  use loadbal
  use masterservice
  use output
  use parallelblock
  use solver
  use storage
  use ucd

  implicit none

  ! data structures needed for serial version of feast
  type (t_masterDataParallelBlock), target :: rmdParBlock
  type (t_masterDataSolver), target        :: rmdSolver
!  type (t_masterDataLoadBal), target       :: rmdLoadBal

  !variables to save a time/operation count stamp
  type (t_stat) :: rtime
  type (t_stat) :: rstat

  !border save variable for internal purpose
  type (Tbordersave) :: rbs

  !saves the Scarc solver
  type (t_solver), dimension(:), pointer :: Rsolver
```

```
!save the discretization information
type (Tdiscretization) :: rdiscr

!stores the parallelblock structure
type (t_parBlock) :: rparBlock

!number of vertices
integer :: nvert

!number of elements
integer :: nelem

!handle for solution vector 1
integer :: h_Dsol

!handle for solution vector 2
integer :: h_Dsol1

!handle for solution vector 3
integer :: h_Dsol2

!handle for solution vector 4
integer :: h_Dsol3

!handle for new rhs vector
integer :: h_Drhs

!number of scarc solvers
integer :: nscarc

!aux variables
integer :: ntimeIter
integer :: i, j, k, l

!number of computations
integer :: ncomp

!variable contains the L2 error
real(DP) :: dl2error

!minimum grid size
real(DP) :: dhmin

!maximum aspect ratio
real(DP) :: daspectRatio

!aux variable for a real value
real(DP) :: dtimeStep

!variables that contains the assigment of gmv variables to feast
!variables if case of solution import
integer, dimension(4) :: igmvidx
integer, dimension(4) :: idxs

!discretization identifiers
type (Tdiscrid) :: rmatStiff
type (Tdiscrid) :: rmatWork
type (Tdiscrid) :: rmatWork1
```

```
  !aux string variable
  character(len = 256) :: stemp

  !macro index
  integer :: imacro

  !double precision access pointers
  real(DP), dimension(:), pointer :: dy1
  real(DP), dimension(:), pointer :: dy2

  !aux variables
  real(DP) :: dval1, dval2

  !node variables for defining boundary conditions
  integer :: inode1, inode2, inode0

  !constants for export and import feast solutions
  integer, parameter :: SOLIO_NONE = 0
  integer, parameter :: SOLIO_READ = 1
  integer, parameter :: SOLIO_WRITE = 2

  !mode for feast solution import/export
  integer :: csolio

  !format for feast solution import/export
  integer :: csolfmt

  !aux variables
  real(DP) :: dprobtime, dtmp
  integer, dimension(:), pointer :: iref
  integer, dimension(:), pointer :: idummy
  integer :: j1, j2, nstep, imode, inref

  !strings containing build information

  !build architecture
  character(len = 64) :: sarch

  !build cpu
  character(len = 64) :: scpu

  !build operating system
  character(len = 64) :: sos

  !use MPI environment
  character(len = 64) :: smpienv

  !build compiler
  character(len = 64) :: scompiler

  !used  BLAS
  character(len = 64) :: sblas


contains

  !main slave routine
```

```
  subroutine startslave

    !discritisation index
    integer :: idiscr

    !timing variable
    type (t_stat) :: rstat

    !Start section 'variables for UCD output'
    ! number of data vectors to output
    integer, parameter :: nmaxNumberOfDataVectors = 10
    integer :: nnumberOfDataVectors = 2

    ! array containing handles for all vectors to output
    integer, dimension(nmaxNumberOfDataVectors) :: H_allDataVectors

    ! array containing the descriptive names of all data vectors to output
    character (len=SYS_STRLEN), dimension(nmaxNumberOfDataVectors) :: SalldataVectorNames

    ! array describing the data type of all vectors to output:
    ! node-based or cell-based. Use UCD_VISDATATYPE_NODEBASED /
    ! UCD_VISDATATYPE_CELLBASED to set the type
    integer, dimension(nmaxNumberOfDataVectors) :: CallDataVectors

    !array describing what to do with data vectors:
    !export or not, agglomerate velocity components to a velocity field or not
    !Use flags like UCD_NOEXPORT, UCD_NO_AGGLOMERATE, UCD_AGGLO_VELOCITYFIELD etc.
    integer, dimension(nmaxNumberOfDataVectors) :: CtreatDataVectors

    !number of additional integer variables to send
    !(e.g. iteration number, parameters for user_additionalVisData)
    integer, parameter :: nnumberOfAddIntVars = 0

    !array containing additional integer variables to send
    integer, dimension(nnumberOfAddIntVars) :: IaddIntVars

    !number of additional double variables to send
    !(e.g. time step, parameters for user_additionalVisData)
    integer, parameter :: nnumberOfAddDblVars = 0

    !array containing additional double variables to send
    real(DP), dimension(nnumberOfAddDblVars) :: IaddDblVars

    !bit array describing what kind of grid statistics to export
    !Use flags like UCD_MINCONVRATE, UCD_HMIN, UCD_ASPECTRATIOS etc.
    integer :: cmiscData

    !number of data vectors that will be calculated on-the-fly
    !by user_additionalVisData when performing UCD output
    integer, parameter :: nnumberOfAddDataVectors = 0
    !End section 'variables for UCD output'

    !we save the current time
    rtime = stat_getSystemTime()

#ifdef SERIAL
    !code block to implement serial version of FEAST
    call msrv_initmaster(rmdParBlock)
```

```
#endif

    !we initialise the library concerning the information in master.dat
    call msrv_feastinit(rparBlock, rdiscr, rbs, rmdParBlock, &
                        Rsolver, nscarc, rmdSolver)

    !get build information and print it
    call sys_getBuildEnv(sarch, scpu, sos, smpienv, scompiler, sblas)
    write (OU_LOG, *) "Build: " // trim(sarch) // "-" // trim(scpu) // "-" // trim(sos)&
         // "-" // trim(smpienv) // "-" // trim(scompiler) // "-" // trim(sblas)

    !This is our matrix identifier, the laplacian operator, defining in the master.dat file
    rmatStiff = rdiscr%rdiscrid(1)
    rmatWork  = rdiscr%rdiscrid(2)
    rmatWork1 = rdiscr%rdiscrid(3)

    !Alternative:
    !   rmatStiff = form_rgetBLF(rdiscr, "Stiffness")
    !   rmatWork  = form_rgetBLF(rdiscr, "Work")
    !   rmatWork1 = form_rgetBLF(rdiscr, "Work1")


    !assembling all systems
    do idiscr=1,rdiscr%ndiscr

      !using wrapperroutines
      call as_set_rhs_vol(rparBlock, rdiscr%rdiscrinfo(idiscr), AS_ALL_BLF, prhs, &
                          AS_SAVE_RHS) !only volume forces are computed
      call as_set_matrix(rparBlock, rdiscr%rdiscrinfo(idiscr), AS_ALL_BLF, pcoeff)

      !...which internally call the following routines

!       call as_matBlockSet(rparBlock, rdiscr%rdiscrinfo(idiscr), MB_ALL_LEV, MB_ALL_MB, &
!                           AS_SETRHS_VOL, AS_ALL_BLF, pcoeff, prhs, pboundaryValue, &
!                           -1, -1, rdiscr%rbfs(idiscr)%ccubTypeRHS, AS_SAVE_RHS)
!
!       call as_matBlockSet(rparBlock, rdiscr%rdiscrinfo(idiscr), MB_ALL_LEV, MB_ALL_MB, &
!                           AS_SETMATRIX, AS_ALL_BLF, pcoeff, prhs, pboundaryValue, &
!                           -1, -1, rdiscr%rbfs(idiscr)%ccubTypeBLF, AS_DO_NOT_SAVE_RHS)

    enddo

    !The application specific values defined in master.dat are set
    call comm_getAddVar_double(rparBlock,"TIMESTEP", dtimeStep)
    call comm_getAddVar_int(rparBlock,"TIMEITER", ntimeIter)
    call comm_getAddVar_int(rparBlock,"ncomp", ncomp)
    call comm_getAddVar_string(rparBlock,"BC", stemp)
    call comm_getAddVar_string(rparBlock,"SOLUTIONIO", stemp)

    !maps the string values for solution import/export in symbolic values
    if (trim(stemp).eq."NONE") then
      csolio = SOLIO_NONE
    else if (trim(stemp).eq."READ") then
      csolio = SOLIO_READ
    else if (trim(stemp).eq."WRITE") then
      csolio = SOLIO_WRITE
    else
      csolio = SOLIO_NONE
```

```
endif

call comm_getAddVar_string(rparBlock,"SOLUTIONIOFMT", stemp)
if (trim(stemp).eq."BIN") then
  csolfmt = IO_BINARY
else
  csolfmt = IO_ASCII
endif

!reads the node indices defining the boundary condition
call comm_getAddVar_int(rparBlock,"NEUNODE1", inode1)
call comm_getAddVar_int(rparBlock,"NEUNODE2", inode2)
call comm_getAddVar_int(rparBlock,"STARTNODE", inode0)

!boundary values
call comm_getAddVar_double(rparBlock,"DIRVAL1", dval1)
call comm_getAddVar_double(rparBlock,"DIRVAL2", dval2)

!The variable OU_LOG is bind to the log file
write (OU_LOG, *) "TIMESTEP = ", dtimeStep
write (OU_LOG, *) "TIMEITER = ", ntimeIter
write (OU_LOG, *) "BC       = ", trim(stemp)
write (OU_LOG, *) "NEUNODE1 = ", inode1
write (OU_LOG, *) "NEUNODE2 = ", inode2
write (OU_LOG, *) "STARTNODE= ", inode0
write (OU_LOG, *) "DIRVAL1  = ", dval1
write (OU_LOG, *) "DIRVAL2  = ", dval2


!construct user defined boundary condition
!the default boundary condition has index one and consists of pure dirichlet
!condition

!this bc consists of 6 segments
call bc_reserve(rparBlock%rbcList, 2, 6, "U2")

!first segment is a dirichlet interval from node startnode to node inode1
call bc_addSegment_vertIdx_int(rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, 1, &
                               BC_TYPE_DIR, dval1, .FALSE., inode0, inode1)

!2nd segment is a dirichlet point at node inode1
call bc_addSegment_vertIdx_vert(rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, 2, &
                                BC_TYPE_DIR, dval1, .FALSE., inode1)

!3rd segment is a neuman interval from inode1 to inode2
call bc_addSegment_vertIdx_int(rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, 3, &
                               BC_TYPE_NEU, 0.0_DP, .FALSE., inode1, inode2)

!4th segment is a dirichlet point at node inode2
call bc_addSegment_vertIdx_vert(rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, 4, &
                                BC_TYPE_DIR, dval2, .FALSE., inode2)

!5th segment is a dirichlet interval from node inode2 to node startnode
call bc_addSegment_vertIdx_int(rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, 5, &
                               BC_TYPE_DIR, dval2, .FALSE., inode2, inode0)

!6th segment is a dirichlet point at node startnode
call bc_addSegment_vertIdx_vert(rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, 6, &
```

```
                                BC_TYPE_DIR, dval2, .FALSE., inode0)

!initialization of the bc
call bc_init(rparBlock%nmaxMgLevel, rparBlock%nmatBlocks, &
             rparBlock%RmatBlockBaseList, rparBlock%RmacroBaseList, &
             rparBlock%RmacroList, rparBlock%RmacroInMatBlock, &
             rparBlock%IedgeInfoSD, rparBlock%rbcList, 2, .FALSE.)

!The communication structures for this matrix and also the boundary
!conditions are set. You have to call this routine every time you make
!a change to the matrix to perform operations like matrix-vector-multiplication
!etc

call as_applyBoundCond(rparBlock, MB_ALL_LEV, rmatStiff, AS_BC_DIRICHLET, &
                       pboundaryValue)

!These calls reserves the vector handles for the solution vectors.
!They are global vectors on all multigrid levels
h_Dsol  = hl_reserve("slave", "h_Dsol", rparBlock, HL_SD, rparBlock%nmaxMgLevel, &
                     HL_ALL, 1, MB_ALL_MB, rmatStiff, -1)
h_Dsol1 = hl_reserve("slave", "h_Dsol1", rparBlock, HL_SD, rparBlock%nmaxMgLevel, &
                     HL_ALL, 1, MB_ALL_MB, rmatStiff, -1)
h_Dsol2 = hl_reserve("slave", "h_Dsol2", rparBlock, HL_SD, rparBlock%nmaxMgLevel, &
                     HL_ALL, 1, MB_ALL_MB, rmatStiff, -1)
h_Dsol3 = hl_reserve("slave", "h_Dsol3", rparBlock, HL_SD, rparBlock%nmaxMgLevel, &
                     HL_ALL, 1, MB_ALL_MB, rmatStiff, -1)
h_Drhs  = hl_reserve("slave", "h_Drhs", rparBlock, HL_SD, rparBlock%nmaxMgLevel, &
                     HL_ALL, 1, MB_ALL_MB, rmatStiff, -1)

!read in global sol. (pressure and streamfunction) from a featflow generated gmv file
igmvidx(1) = 4
idxs(1)    = h_Dsol

igmvidx(2) = 5
idxs(2)    = h_Dsol1

stemp = "sb25_l5.gmv.gs"

!call io_readglobalsol(stemp, rparBlock, 2, igmvidx, idxs, 2, rparBlock%nmaxMgLevel, &
!                      dprobtime)

write (OU_LOG, *) "Probtime = ", dprobtime

!This function calculates the min grid size and max aspect ratio
call tools_getGridMass(rparBlock, HL_PB, rparBlock%nmaxMgLevel, MB_ALL_MB, dhmin, &
                       daspectRatio)

!This call starts the global time measurement
call comm_sendglobaltime(rparBlock,0, 0, t_stat(0.0_DP, 0.0_DP, 0_I64), 0, 0.0_DP, &
                         0.0_DP, 0.0_DP, 0.0_DP, dhmin, daspectRatio)

!limits the output level
if (rparBlock%ioutputLevel .gt. rparBlock%nmaxMgLevel) then
  rparBlock%ioutputLevel = rparBlock%nmaxMgLevel
endif

!start iteration
do k = 1, ntimeIter
```

```
!read solution if desired
if (csolio .eq. SOLIO_READ) then
  call io_readSolution(rparBlock, "solution1", h_Dsol, rparBlock%nmaxMgLevel, &
                       csolfmt)
endif

!calculation with boundary condition 1
call as_applyBoundCond(rparBlock, MB_ALL_LEV, rmatStiff, &
                       AS_BC_DIRICHLET + AS_BC_NEUMANN, pboundaryValue,1)

!initialise the solver, this has to be done every time the matrix was changed
call solver_reinit(rparBlock, Rsolver(rmatStiff%rform%iscarc), rmatStiff, HL_ALL, &
                   rparBlock%nmaxMGLevel, -1)

!initialise the solution vector with the right boundary values
call solver_initDirichValues(rparBlock, HL_ALL, rparBlock%nmaxMgLevel, &
                             rmatStiff, h_Dsol, pboundaryValue)

!start the solver
call solver_perform(rparBlock, Rsolver(rmatStiff%rform%iscarc), &
                    rparBlock%nmaxMgLevel, h_Dsol, MB_ALL_MB, &
                    SOLV_NOT_PURE_NEUMANN, SOLV_THIS_START_VEC, rstat)

!matrix copy stiffness to work
call as_matBlockCopy(rparBlock, MB_ALL_LEV, rmatStiff, rmatWork, YES)

!Index of boundary condition is NOT set in as_matBlockCopy or as_matBlockLinComb, so
!do it now.
rmatWork%rform%ibc = rmatStiff%rform%ibc
rmatWork%rform%sbc = rmatStiff%rform%sbc

!write the solution to a file if desired
if (csolio.eq.SOLIO_WRITE) then
  call io_writeSolution(rparBlock, "solution1", h_Dsol, rparBlock%nmaxMgLevel, &
                        csolfmt)
endif

!Start section UCD output
!exported to AVS/GMV format
nnumberOfDataVectors   = 1
H_allDataVectors(1)    = h_Dsol
SalldataVectorNames(1) = "solution"
CallDataVectors(1)     = UCD_VISDATATYPE_NODEBASED
CtreatDataVectors(1)   = UCD_NO_AGGLOMERATE

if (Rsolver(rmatStiff%rform%iscarc)%H_Daux(PREC_AXVDEF) .gt. 0) then
  !defect sent, too
  nnumberOfDataVectors   = 2
  H_allDataVectors(2)    = Rsolver(rmatStiff%rform%iscarc)%H_Daux(PREC_AXVDEF)
  SalldataVectorNames(2) = "defect"
  CallDataVectors(2)     = UCD_VISDATATYPE_NODEBASED
  CtreatDataVectors(2)   = UCD_NO_AGGLOMERATE
endif

!Export information on
!- minimal convergence rate per parallel block
!- maximal convergence rate per parallel block
```

```
!- average convergence rate per parallel block
!- minimal grid size per parallel block
!- maximal aspect ratio per parallel block
cmiscData = UCD_MINCONVRATE + UCD_MAXCONVRATE + UCD_AVGCONVRATE + &
            UCD_HMIN + UCD_ASPECTRATIOS

call comm_sendVisData(&
     rparBlock, nnumberOfDataVectors, nnumberOfAddDataVectors, &
     H_allDataVectors, SalldataVectorNames, CallDataVectors, CtreatDataVectors,&
     nnumberOfAddIntVars, IaddIntVars, nnumberOfAddDblVars, IaddDblVars, &
     cmiscData, trim(sys_sgetOutputFileName()), &
     rparBlock%ioutputLevel, rmatStiff)
!End section UCD output

!calculation with boundary condition 2, set as last parameter of as_applyBoundCond

!clear RHS
call as_matBlockSet(rparBlock, rmatStiff%rdiscrinfoPtr, MB_ALL_LEV, MB_ALL_MB, &
                    AS_CLEARRHS, rmatStiff%matid, pcoeff, prhs, pboundaryValue, &
                    -1, -1, cub_igetID("G3X3"), AS_SAVE_RHS)

!assemble the RHS (only volume integrals)
call as_matBlockSet(rparBlock, rmatStiff%rdiscrinfoPtr, MB_ALL_LEV, MB_ALL_MB, &
                    AS_SETRHS_VOL, rmatStiff%matid, pcoeff, prhs, pboundaryValue, &
                    -1, -1, cub_igetID("G3X3"), AS_SAVE_RHS)

!Compute RHS contributions coming from Neumann boundary and apply
!Dirichlet boundary conditions
call as_applyBoundCond(rparBlock, MB_ALL_LEV, rmatStiff, &
                       AS_BC_DIRICHLET + AS_BC_NEUMANN, pboundaryValue,2)

!initialise the solver, this has to be done every time the matrix was changed
call solver_reinit(rparBlock, Rsolver(rmatStiff%rform%iscarc), rmatStiff, &
                   HL_ALL, rparBlock%nmaxMGLevel, -1)

!initialise the solution vector with the right boundary values
call solver_initDirichValues(rparBlock, HL_ALL, rparBlock%nmaxMgLevel, &
                             rmatStiff, h_Dsol1, pboundaryValue)

!start the solver
call solver_perform(rparBlock, Rsolver(rmatStiff%rform%iscarc), &
                    rparBlock%nmaxMgLevel, h_Dsol1, MB_ALL_MB, &
                    SOLV_NOT_PURE_NEUMANN, SOLV_THIS_START_VEC, rstat)

!Start section UCD output
!solution is sent
nnumberOfDataVectors   = 1
H_allDataVectors(1)    = h_Dsol1
SalldataVectorNames(1) = "solution"
CallDataVectors(1)     = UCD_VISDATATYPE_NODEBASED
CtreatDataVectors(1)   = UCD_NO_AGGLOMERATE

if (Rsolver(rmatStiff%rform%iscarc)%H_Daux(PREC_AXVDEF) .gt. 0) then
  !defect sent, too
  nnumberOfDataVectors   = 2
  H_allDataVectors(2)    = Rsolver(rmatStiff%rform%iscarc)%H_Daux(PREC_AXVDEF)
  SalldataVectorNames(2) = "defect"
  CallDataVectors(2)     = UCD_VISDATATYPE_NODEBASED
```

```
  CtreatDataVectors(2)   = UCD_NO_AGGLOMERATE
endif

call comm_sendVisData(&
     rparBlock, nnumberOfDataVectors, nnumberOfAddDataVectors, &
   H_allDataVectors, SalldataVectorNames, CallDataVectors, CtreatDataVectors,&
   nnumberOfAddIntVars, IaddIntVars, nnumberOfAddDblVars, IaddDblVars, &
   cmiscData, "solution_bc2.gmv", &
   rparBlock%ioutputLevel, rmatStiff)
!End section UCD output


!connect the BLF to the first boundary condition again

!second matrix assembly via calculation of form, these 4 steps
!(clear, assembly, set rhs, set bc are essential!!!)

!copy the information from the BLF of rmatStiff to that of rmatWork1
rmatWork1%rform = rmatStiff%rform
call as_matBlockSet(rparBlock, rmatWork1%rdiscrinfoPtr, MB_ALL_LEV, MB_ALL_MB, &
                    AS_CLEARMATRIX, rmatWork1%matid, pcoeff, prhs, pboundaryValue, &
                    h_Dsol, h_Dsol, rdiscr%rbfs(rmatStiff%discrid)%ccubTypeBLF, &
                    AS_SAVE_RHS)

call as_matBlockSet(rparBlock, rmatWork1%rdiscrinfoPtr, MB_ALL_LEV, MB_ALL_MB, &
                    AS_CLEARRHS, rmatWork1%matid, pcoeff, prhs, pboundaryValue, &
                    h_Dsol, h_Dsol, rdiscr%rbfs(rmatStiff%discrid)%ccubTypeBLF, &
                    AS_SAVE_RHS)


call as_matBlockSet(rparBlock, rmatWork1%rdiscrinfoPtr, MB_ALL_LEV, MB_ALL_MB, &
                    AS_SETMATRIX, rmatWork1%matid, pcoeff, prhs, pboundaryValue, &
                    h_Dsol, h_Dsol, rdiscr%rbfs(rmatStiff%discrid)%ccubTypeBLF, &
                    AS_SAVE_RHS)

call as_matBlockSet(rparBlock, rmatWork1%rdiscrinfoPtr, MB_ALL_LEV, MB_ALL_MB, &
                    AS_SETRHS_VOL, rmatWork1%matid, pcoeff, prhs, pboundaryValue, &
                    h_Dsol, h_Dsol, rdiscr%rbfs(rmatStiff%discrid)%ccubTypeBLF, &
                    AS_SAVE_RHS)

call as_applyBoundCond(rparBlock, MB_ALL_LEV, rmatWork1, &
                       AS_BC_DIRICHLET + AS_BC_NEUMANN, pboundaryValue,1)


!perform two tests

!initialise the solver, this has to be done every time the matrix was changed, rmatWork
!is used
call solver_reinit(rparBlock, Rsolver(rmatWork%rform%iscarc), rmatWork, HL_ALL, &
                   rparBlock%nmaxMGLevel, -1)

!initialise the solution vector with the right boundary values
call solver_initDirichValues(rparBlock, HL_ALL, rparBlock%nmaxMgLevel, &
                             rmatWORK1, h_Dsol2, pboundaryValue)

!start the solver
call solver_perform(rparBlock, Rsolver(rmatWork%rform%iscarc), &
                    rparBlock%nmaxMgLevel, h_Dsol2, MB_ALL_MB, &
```

```
                    SOLV_NOT_PURE_NEUMANN, SOLV_THIS_START_VEC, rstat)

!initialise the solver, this has to be done every time the matrix was changed, rmatWork1
!is used
call solver_reinit(rparBlock, Rsolver(rmatWork1%rform%iscarc), rmatWork1, &
                   HL_ALL, rparBlock%nmaxMGLevel, -1)

!initialise the solution vector with the right boundary values
call solver_initDirichValues(rparBlock, HL_ALL, rparBlock%nmaxMgLevel, &
                             rmatWork1, h_Dsol3, pboundaryValue)

!start the solver
call solver_perform(rparBlock, Rsolver(rmatWork1%rform%iscarc), &
                    rparBlock%nmaxMgLevel, h_Dsol3, MB_ALL_MB, &
                    SOLV_NOT_PURE_NEUMANN, SOLV_THIS_START_VEC, rstat)

!To access the data vector, you have to loop over all matrixblock
!in this parallel domain
do i = 1, rparBlock%nmatBlocks

  !This functions lets the given double pointers to the piece of the
  !global vectors point resides on matrixblock i
  call hl_select(dy1, rparBlock%nmaxMgLevel, HL_SD, h_Dsol3, i)
  call hl_select(dy2, rparBlock%nmaxMgLevel, HL_SD, h_Dsol2, i)

  !Now you can perform normal fortran array operations with this pointers

  if (minval(abs(dy2)) .gt. 0.0_DP) then

    dy1 = dy1/dy2

    print *, "sol3/sol2 (min=1, max=1):  min=", minval(dy1), " max=", maxval(dy1)

  endif

enddo

!perform a matrix linear combination without modification of the RHS (last parameter)
call as_matBlockLinComb(rparBlock, MB_ALL_LEV, -1.0_DP, rmatStiff, 2.0_DP, rmatWork1, NO)
!Index of boundary condition is NOT set in as_matBlockCopy or as_matBlockLinComb, so
!do it now.
rmatWork1%rform%ibc = rmatStiff%rform%ibc
rmatWork1%rform%sbc = rmatStiff%rform%sbc

call as_applyBoundCond(rparBlock, MB_ALL_LEV, rmatWork1, &
                       AS_BC_DIRICHLET + AS_BC_NEUMANN, pboundaryValue,1)

!initialise the solver to solve the combined matrix
call solver_reinit(rparBlock, Rsolver(rmatWork1%rform%iscarc), rmatWork1, &
                   HL_ALL, rparBlock%nmaxMGLevel, -1)

!initialise the solution vector with the right boundary values
call solver_initDirichValues(rparBlock, HL_ALL, rparBlock%nmaxMgLevel, &
                             rmatWork, h_Dsol2, pboundaryValue)

!start the solver
call solver_perform(rparBlock, Rsolver(rmatWork1%rform%iscarc), &
                    rparBlock%nmaxMgLevel, h_Dsol2, MB_ALL_MB, &
```

```
                    SOLV_NOT_PURE_NEUMANN, SOLV_THIS_START_VEC, rstat)

  !Start section UCD output
  !solution is sent
  nnumberOfDataVectors   = 1
  H_allDataVectors(1)    = h_Dsol2
  SalldataVectorNames(1) = "solution"
  CallDataVectors(1)     = UCD_VISDATATYPE_NODEBASED
  CtreatDataVectors(1)   = UCD_NO_AGGLOMERATE

  if (Rsolver(rmatWork1%rform%iscarc)%H_Daux(PREC_AXVDEF) .gt. 0) then
    !defect sent, too
    nnumberOfDataVectors   = 2
    H_allDataVectors(2)    = Rsolver(rmatWork1%rform%iscarc)%H_Daux(PREC_AXVDEF)
    SalldataVectorNames(2) = "defect"
    CallDataVectors(2)     = UCD_VISDATATYPE_NODEBASED
    CtreatDataVectors(2)   = UCD_NO_AGGLOMERATE
  endif

  call comm_sendVisData(&
       rparBlock, nnumberOfDataVectors, nnumberOfAddDataVectors, &
       H_allDataVectors, SalldataVectorNames, CallDataVectors, CtreatDataVectors,&
       nnumberOfAddIntVars, IaddIntVars, nnumberOfAddDblVars, IaddDblVars, &
       cmiscData, "solution_linComb.gmv", &
       rparBlock%ioutputLevel, rmatWork1)
  !End section UCD output

  !The timings are printed out
  call comm_writeCompTime()

  !To access the data vector, you have to loop over all matrixblock
  !in this parallel domain
  do i = 1, rparBlock%nmatBlocks

    !This functions lets the given double pointers to the piece of the
    !global vectors point resides on matrixblock i
    call hl_select(dy1, rparBlock%nmaxMgLevel, HL_SD, h_Dsol, i)
    call hl_select(dy2, rparBlock%nmaxMgLevel, HL_SD, h_Dsol2, i)

    !Now you can perform normal fortran array operations with this pointers


    if (minval(abs(dy2)) .gt. 0.0_DP) then

      dy1 = dy1/dy2

      print *, "sol/sol2 (min=1, max=1):  min=", minval(dy1), " max=", maxval(dy1)

      dy1 = dy2

    endif

  enddo

enddo

call hl_clear(rparBlock, HL_SD, rparBlock%nmaxMgLevel, h_Drhs, HL_ALL, -1, rstat)
```

```
!attach a new vector as rhs
call hl_attachRHS(rparBlock, HL_SD, rparBlock%nmaxMgLevel, h_Drhs, rmatStiff)

do i = 1, rparBlock%nmatBlocks
  call hl_select(dy1, rparBlock%nmaxMgLevel, HL_SD, h_Drhs, i)
  print *, "RHS attachtest (min=0, max=0):  min=", minval(dy1), " max=", maxval(dy1)
enddo

!The difference time from now to timestamp are calculated
rtime = stat_diffTime(rtime)

!... and printed
write (OU_LOG, '(A, 2F6.2)') "  mes. sum                    : ", &
       rtime%dtimeCpu, rtime%dtimeReal

!the total number of vertices and elements are determined ....
call tools_getNVert(rparBlock, HL_SD, rparBlock%nmaxMgLevel, MB_ALL_MB, rmatWork1, &
                    nvert)
call tools_getNElem(rparBlock, HL_SD, rparBlock%nmaxMgLevel, MB_ALL_MB, nelem)


!This function calculates the min grid size and max aspect ratio  ...
call tools_getGridMass(rparBlock, HL_SD, rparBlock%nmaxMgLevel, MB_ALL_MB, &
                       dhmin, daspectRatio)

dtmp = tools_dnorm(rparBlock, HL_SD, rparBlock%nmaxMgLevel, h_Dsol, &
                   MB_ALL_MB, rstat, rmatWork1)
if (dtmp .gt. 0.0) then
  dl2error = tools_dl2errorDiscr(rparBlock, HL_SD, rparBlock%nmaxMgLevel, h_Dsol, &
                                 MB_ALL_MB, rstat, Rsolver(rmatWork1%rform%iscarc)%rdiscrId, &
                                 pexactSol)  /  dtmp
  call output_line(OL_MSG, "global result", &
                   "global rel. L2 error = " // trim(sys_sdEL(dl2error, 8)) // &
                   "   L2/Hmin^2 =" // trim(sys_sdEL(dl2error / (dhmin * dhmin), 8)) // &
                   " L2f=" // trim(sys_sdL(dl2error, 12)))
else
  dl2error = tools_dl2errorDiscr(rparBlock, HL_SD, rparBlock%nmaxMgLevel, h_Dsol, &
                                 MB_ALL_MB, rstat, rmatWork1, pexactSol)
  call output_line(OL_MSG, "global result", &
                   "global abs. L2 error = " // trim(sys_sdEL(dl2error, 8)) // &
                   "   L2/Hmin^2 =" // trim(sys_sdEL(dl2error / (dhmin * dhmin), 8)) // &
                   " L2f=" // trim(sys_sdL(dl2error, 12)))
endif


!... and printed out
call output_line(OL_MSG, "global result", &
                 "global NVT = " // trim(sys_siL(nvert, 10)) // &
                 "   global nelem = " // trim(sys_siL(nelem, 10)))


!... and prints out
call output_line(OL_MSG, "global result", "hmin = " // trim(sys_sdEL(dhmin, 8)) // &
                 "  hmin^2 = " // trim(sys_sdEL(dhmin * dhmin, 8)) // &
                 "   AR = " // trim(sys_siL(int(daspectRatio), 8)))


!this sequence generates the stop message to the master process
```

```
    call par_finish()

    !information about the storage usage is printed out
    call storage_info

  end subroutine startslave

end module slavemod
```

Chapter 4

File formats used by FEAST

4.1 FEAST file format

Extension: **.feast**

Preliminaries

FEAST files are plain ASCII files in a line-based format, using shared-vertex data representations. Empty lines are not allowed. Comment lines can be arbitrarily placed, the special character # marks any such line. If the comment character # occurs in a line, the rest of this line will be ignored by FEAST.

Each FEAST *project* consists of six separate files:

1. header file (filename.feast)
2. static geometry file (filename.fgeo)
3. fictitious boundary file (filename.ffgeo)
4. mesh file (filename.fmesh)
5. partitioning file (filename.fpart)
6. boundary condition file (filename.fbc)

Header file

The header file lists those files that are included in the current project.

```
FEAST              # ID-tag
2D                 # mode
3                  # version major
0                  # version minor
description        # single line description of the project
# file locations
filename.fgeo      # relative path to static geometry file
filename.ffgeo     # relative path to fictitious geometry
filename.fmesh     # relative path to mesh file
filename.fpart     # relative path to partitioning file
filename.fbc       # relative path to boundary condition file
```

Sample 2: FEAST 2D – Header

The keyword `NONE` instead of a path indicates that no such file is used. Note that FEAST and the DeViSoR might crash if you exclude vital files.

Paths are relative to the location of the header file.

Geometry files

Both static and fictitious geometry files have the same format.

Each geometry file starts with the following header:

```
FEAST_GEOM          # ID-tag
2D                  # mode
3                   # version major
0                   # version minor
6  # number of boundary components
```

Sample 3: FEAST 2D – geometry

Supported boundary types are:

- segment lists containing segments (single type or mixed) of the following types
  - line segments defined by cartesian start coordinates and vector from start to end point
  - circle segments defined by midpoint cartesian coordinates, radius plus dummy, start and end angle in radians (angles are measured counterclockwise starting from the 3 o'clock position)
  - NURBS segments are defined by the degree n, a flag to distinguish between open and closed curves, $m+1$ control points, $m+1$ weights and a node vector of length $m+1$.

  For each segment, the parameter interval is stored additionally, including the starting point, excluding the endpoint.
- analytic (via reference to some object file which returns cartesian coordinates for each parameter value and maximum parameter value)

All boundary types are editable in the DeViSoR , for analytic boundaries, this means editing and recompiling some Fortran or C or whatever code. These functions have to provide three calls: `PARXC(t)`, `PARYC(t)`, `TMAXC` returning x and y coordinates for a given parameter value and the length of the parameter interval (implicitly starting at parameter value 0).

Note that closed circles and closed NURBS curves must be the only boundary type in each segment list!

```
4         # number of segments on first boundary
0         # type of first segment: line
-1.0 1.0  # cartesian coordinates of start point
0.0 -2.0  # vector from start to end point
0.0 0.25  # parameter interval
0         # type of second segment: line
-1.0 -1.0 # cartesian coordinates of start point
2.0 0.0   # vector from start to end point
0.25 0.5  # parameter interval
0         # type of third segment: line
1.0 -1.0  # cartesian coordinates of start point
0.0 2.0   # vector from start to end point
0.5 0.75  # parameter interval
0         # type of 4th segment: line
1.0 1.0   # cartesian coordinates of start point
-2.0 0.0  # vector from start to end point
0.75 1.0  # parameter interval

1         # number of segments on second boundary
1         # type of first (only) segment: circle
1.0 1.0   # cartesian coordinates of center
1.0 0.0   # radius and some dummy variable
0.0 6.28... # start and end angles
0.0 1.0   # parameter interval

1         # number of segments on third boundary
2         # type of third boundary: open nurbs curve
3         # degree n
6         # number of control points m+1
0.0 1.0 0.25 # X,Y,weight of first control point
...
1.0 1.0 0.25 # X,Y,weight of last control point
0.0       # node vector entry 0 to n
0.1       # node vector entry n+1
...
0.9       # node vector entry m
1.0       # node vector entry m+1 to n+m+1
0.0 1.0   # parameter interval

1         # number of segments on fourth boundary
3         # type of third boundary: closed nurbs curve
3         # degree n
6         # number of control points m+1
0.0 1.0 0.25 # X,Y,weight of first control point
...
1.0 1.0 0.25 # X,Y,weight of last control point
0.0       # node vector entry 0
...
0.1       # node vector entry m+1
0.0 1.0   # parameter interval

1           # number of segments on this boundary
4           # type: analytic
source file # path to source file
```

Sample 4: FEAST 2D – Boundary components

Mesh file

Each mesh file starts with the following header:

```
FEAST_MESH          # ID-tag
2D                  # mode
3                   # version major
0                   # version minor
0                   # type of mesh (0=quad,1=tri,2=hybrid)
25                  # number of macro nodes
16                  # number of macro elements
40                  # number of macro edges
```

Sample 5: FEAST 2D – mesh

Nodes FEAST2D supports three kind of nodes: inner nodes (cartesian nodes), parameterized nodes and fictious boundary nodes. Parameterisation is equal to FEAT2D, fictitious boundary nodes are used in moving boundary scenarios.

```
# Nodes
0.0  0.0 0  # X,Y,0 for inner node
0.0 0.0 -5  # X,Y,nodal property for inner nodes carrying an
            # additional nodal property
0.25 0.0 1  # parameter value 0.25, dummy, boundary number 1
# ...
```

Sample 6: FEAST 2D – Nodes

As the boundary list is indexed from 1 upwards, inner nodes can be distinguished from boundary nodes unambiguously by testing the third value of each line against zero.

Edges Edges are defined by giving the numbers of start and end nodes.

```
# Edges
1 2  # index of start and end node
# ...
```

Sample 7: FEAST 2D – Edges

Elements Elements are defined using tons of information: First up the numbers of the nodes that make up the macro, listed counterclockwisely for both quads and triangles. Next the numbers of the edges, from first node to second node, from second to third and so on. The next couple of lines contain neighbourhood information: First up the numbers of those elements sharing a common edge, then the numbers of those elements sharing a common node but not an edge. Number 0 is used to mark that there is no neighbour. There can be at most one edge neighbour per edge, but an arbitrary number of node neighbours. The following trick is used: instead of printing out the number of the neighbour element, the count of neighbours is printed out as a negative integer. For all such negative tags an additional line is inserted directly after the node neighbour line containing the corresponding element neighbours. The third part of the macro definition lists anisotropic refinement data. This is coded into five numerical values:

1. Initial refinement level both in x and in y direction
2. Refinement control is coded into four bits. The LSB controls refinement in x direction, the LSB+1 controls direction (left or right). The MSB and MSB-1 contain this information for the y direction. The integer representing this information is stored.

3. The last three double values give the refinement factors the underlying algorithm uses.

```
# Macro number 1
0        # type (0=quad, 1=tri)
1 2 7 6  # indices of nodes
1 2 3 4  # indices of edges
0 2 5 0  # Indices of edge neighbours
0 6 0 0  # Indices of node neighbours
1 0 0.5 0.5 0.5   # refinement level, refinement mode
                  # three factors
# cell number 2
1        # type (0=quad, 1=tri)
1 2 7    # indices of nodes
1 2 3    # indices of edges
0 2 5    # Indices of edge neighbours
0 6 0    # Indices of node neighbours
1 2      # refinement level and subdivision mode
# ...
```

Sample 8: FEAST 2D – Elements

Partition file

For each element, the partition file contains one single line containing the indices of the matrix and parallel blocks for each element. Element numbering is based on the `fmesh` file.

```
FEAST_PART         # ID-tag
2D                 # mode
3                  # version major
0                  # version minor
47                 # number of elements
4                  # number of parallel blocks
1 4                # parallel and matrix block for element 1
...
```

Sample 9: FEAST 2D – partition

Boundary condition file

This file defines the boundary conditions. The user can extend the default boundary conditions dirichlet and neumann with self defined condition tags (user defined boundary status). Also he can add additional information to each boundary specification (user defined boundary information). Several boundary condition building blocks can be defined. Each block has as default zero dirichlet condition. Each line in such a block defines a certain condition on a given interval or point. The interval can be defined via parameter value (`PDEFINT` with `PAR1 PAR2`) or via node indices (`NDEFINT` with `NODE1 NODE2`) (note that parameter intervals are only unique for a single boundary, thus, boundary references are given if parameters are used), a point can also be defined via parameter value (`PDEFNODE` with `PAR1`) or via node index (`NDEFNODE` with `NODE1`). The boundary value can be a float constant or a tag, telling the library to call a user defined function.

```
# FEAST2D boundary condition file
#
FEAST_BC           # ID-tag
2D                 # mode
3                  # version major
0                  # version minor
#
2 # number of user defined boundary status (UDBS)
SUPERSONICIN
SUPERSONICOUT
#
2     # number of user defined boundary information
# name INT|FLOAT
USERVALUE1 INT
USERVALUE2 FLOAT
#
3     # number of bcbb (boundary condition building blocks)
#
# bcbb 1
U1    # name (string constant)
3     # number of bcbbd (bcbb definitions)
PDEFINT DIR 0.0 0.0 1.0 1 2.0
# bcbbd :  PDEFINT|PDEFNODE|NDEFINT|NDEFNODE
#          DIR|NEU|UDBS1|..|UDBSn
#          BVALUE|FUNC
#          BDIX PAR1 [PAR2] | NODE1 [NODE2]
#          USERVALUE1|FUNC..USERVALUEn|FUNC
#
#    if FUNC is given, for parameter evaluation a user
#    given routine
#    pboundaryvalue(bs,bcbb,bcbbd,par1,par2,par,valueindex)
#    is called
#
PDEFNODE NEU FUNC 2.0 1 2.0
NDEFINT SUPERSONICIN 0.0 1 2 1 2.0
#
# bcbb 2
U2
1
PDEFINT DIR 0.0 0.0 1.0
#
# bcbb 3
P
2
NDEFINT NEU 0.0 2 4
NDEFNODE DIR 10.0 3
```

Sample 10: FEAST 2D – boundary conditions

4.2 Data file

Extension: **.dat**

At the moment the entries in this file must have the same sequence and arrangement. The entries are the following

`PROJECTID:` deprecated

`PROJECTDIR:` This defines the complete absolute path without ~ to the feast subfolder

LOGDIR: This defines the complete absolute path without ~ to the folder, where the log files should be saved. This could be a tmp file area.

MSG_LEVEL: level of verbosity (0 = none, 9 = all messages)

GRIDFILE: This is a path relatively to PROJECTDIR to the grid file describing the domain

LOADBALANCE: deprecated, should be 0

LOADBALFILE: deprecated

AUTOPARTITION: YES/NO This switch defines, if the auto partition is used. If it is activated, then the number of started processes is used to get the number of partitions, otherwise the information in the FEAST file is used.

WEIGHTEDPART: YES/NO This switch defines if in case of auto partititon load balancing is used.

CONSTMALMUL: YES/NO This switch determines if the faster constant matrix multiplication is used, if possible.

FASTMATASS: deprecated

CALCSMOOTHNORM: YES/NO This switch defines if after a smoothing operation in the multigrid driver the norm of the defect is calculated. This is not necessary for the algorithm, only for debugging purposes. For production runs you should set this to NO.

LONGFORM: This defines if the output of the master program is in long or short format.

The next area describes the discretisation of the problem. A discretisation consists of a finite element and a cubature formula. For efficency reasons you have the possibilty to define more than one bilinear forms ('matrix') over one discretisation.

The entry NDISCR defines the number of discretisation. For each discretisation, the entries ELE, NBF and BF have to be defined. The ELE identifier defines the type of the element used for test and ansatz functions, and the used quadrature formulas for building the matrix, the right hand side and for incorporating Neuann boundary conditions. Therefore, the last cubature formula maut be a 1D cubature formula. Currently only the quadraliteral bilinear Finite Element with the id Q1 is fully supported. For the quadrature formulas have a look to the description of the cubature module. The entry NBF determines the number of bilinear forms defined in this discretisation. The last entry BF describes the bilinear form itself. As example let us have a look to the discretisation of the Poisson equation

$$-\Delta u = f$$

with the weak formulation

$$c_1(a_x, b_x) + c_2(a_y, b_y) = (f, b).$$

The first integer of the BLF entry describes the number of terms in the bilinear form, here two.

Every following line in the BLF entry describes one term as follows: coefficient, ansatz function, test function. If the coefficent is set to VAR, then the function pcoeff from the userdef module is used to calculat the value of the coefficient. For the ansatz and test functions φ the following values can be used:

FUNC – φ
DERX – φ_x
DERY – φ_y
DERXX – φ_{xx}
DERXY – φ_{xy}
DERYY – φ_{yy}

The next parameter defines the number of terms of the right hand side, in this case one term, the function itself.

The last parameter is an string identifier for the matrix in short and long form.

The next section describes the assignment of the ScaRC algorithms to the matrices. The entry NSCARC defines the number of ScaRC files to be included. The next entries consists of two parameters each line, first parameter the relative path to the ScaRC algorithm file (format described later) and the identifier of the matrix, the algorithm should solve. This assignment can be changed later in the code by API routines. See the description of the sample program.

`OUTPUTFILE:` name of the gmv/avs file without suffix
`OUTPUTLEVEL:` the multigrid level, on which the solution shall be written in the output file
`LASTSOLUTION:` deprecated
`LASTMATRIX:` deprecated
`STORAGE_DES:` number of storage descriptors, problem size dependent `STORAGE_SIZE:` heap size
`MAXMGLEVEL:` maximum multigrid level
`ANISOREFMODE:` deprecated, should be MODE2

The following entries allows the setting of a certain SBBLAS implementation.

`SBBLASMV:` implementation of the MV algorithm
`SBBLASPRSLOWL:` implementation of the TriGS algorithm
`SBBLASPRSLINE:` implementation of the MVline algorithm
`SBBLASPRSLINE:` implementation of the Tri algorithm
`SBBLASOVERRIDE:` is it is set to YES, the above defined values are used
`SBBLASCONF:` path to the sbblas.conf file

The further entries in this file are application dependent.

4.3 ScaRC application file

extension **.scarc**

This file describes the structure of a ScaRC algorithm over the various hierachical layers.

The definition starts with the keyword SOLVER.

At the moment, two main solvers are supported, multigrid (MG) and conjugate gradient (CG/BICG) schemes. When the latter are intendend to work as smoothers in an outer multigrid scheme (e.g. when solving multidimensional problems) then this can be indicated by CG_SMOOTH and BICG_SMOOTH, respectively. (The effect is that the necessary structures are allocated on all multgrid levels). The direct solvers are intented for solving the coarse grid problem in the multigrid scheme.

```
SOLVER=[CG|CG_SMOOTH],[iter],[exit-crit],[Hlayer]
  PREC=ALL,0,[omega],[scarcflag],[Hlayerflag]
  [SOLVER|LSMOOTHER]

SOLVER=[BICG|BICG_SMOOTH],[iter],[exit-crit],[Hlayer]
  PREC=ALL,0,[omega],[scarcflag],[Hlayerflag]
  [SOLVER|LSMOOTHER]

SOLVER=MG,[iter],[exit-crit],[cycle],[npresmooth],[npostsmooth],[Hlayer]
  SMOOTH=ALL,0,[omega],[scarcflag],[Hlayerflag]
  [SOLVER|LSMOOTHER]
  COARSE=[SOLVER]
```

```
SOLVER=GLOBAL

SOLVER=PGLOBAL

SOLVER=MGLOBAL

LSMOOTHER =  JACOBI|TRIGS|ADITRIGS|ILU
```

- `[iter]` : iteration count
- `[exit-crit]` : exit criterion absolute (`ABS`) or relative (`REL`). `ABS:1E-6` e.g. indicates, that the iteration process shall end, if the absolute residual is smaller than 10^{-6}.
- `[omega]` : damping parameter for this level
- `[Hlayer]` : hierachical layer this scheme works on
- `[scarcflag]` : If this parameter is 0, the next scheme is a simple local smoother LSMOOTHER, otherwise a further ScaRC solver.
- `[Hlayerflag]` : if this parameter is 0, the next scheme works on a different hierachical layer, otherwise on the same.
- `[cycle]` : multigrid cycle type V,W or F
- `[npresmooth],[npostsmooth]` : number of pre- and postsmoothing steps

Lets have a look to the following example:

`SOLVER=CG,32,REL:1.0E-6,HL_SD`	The global scheme is a CG scheme which stops after 32 iterations or if a relative accuracy of 10^{-6} is reached.
`PREC=ALL,1,1.0,HL_SD,1,1`	The preconditioning is done with damping factor 1.0. The precondiotioner is of ScaRC type and acts on the same hierachical level.
`SOLVER=MG,1,REL:1.0E-6,V,1,1,HL_SD` `SMOOTHER=ALL,0,0.8,HL_SD,1,0`	As preconditioner acts a global multigrid scheme which performs one global step. The exit criterion has in this case no meaning. The global multigrid performs 1 pre- and 1 postsmoothing step. The preconditioner is of ScaRC type and works not on the same hierachical level.
`SOLVER=MG,1,REL:1.0E-6,F,4,4,HL_PB` `SMOOTHER=ALL,0,1.0,HL_PB,0,0`	As smoother works a local multigrid scheme, which works seperatly on the parallel subdomains It performs one step and 4 pre- and 4 postsmoothing steps. The smoother for this scheme is a local smoother. In this case the last parameter has no meaning.
`LSMOOTHER=TRIGS`	Local smoother triGaussSeidel

`COARSE=CG,512,REL:1.0E-6,HL_PB`
`PREC=ALL,1,1.0,HL_PB,0,0`
`LSMOOTHER=JACOBI`
	The coarse grid solver for the local multigrid scheme It performs up to 512 steps till an accuracy of relatively 10^{-6} is reached. It is preconditioned with the Jacobi method.

`COARSE=CG,512,REL:1.0E-6,HL_SD`
`PREC=ALL,1,1.0,HL_SD,0,0`
`LSMOOTHER=JACOBI`
	The coarse grid solver for the global multigrid scheme. It has the same parameters as the local scheme.

Chapter 5

Maintainance

5.1 Style guidelines

- The leading letter of the variable name describes the type of the variable.
  scalar variables:
  - `n` - integer variables for maximum or total numbers (e.g. `nmacros` : total number of macros)
  - `c` - integer variables with fixed value set (e.g. `csolIO` : number of output channel)
  - `i` - all other integer variables
  - `d` - double precision variables
  - `f` - single precision variables (float...)
  - `b` - boolean variables
  - `p` - functions in mathematical sense (e.g. `pmyExactSol`)
  - `r` - record variables (e.g. `rtime` is variable of type `t_time`)
  - `s` - characters (strings)
- Arrays, handles and types:
  - Names of array identifiers start with capital letters, using the same convention described above (e.g. `Dsolution` : double prec. vector)
  - Names of handles are indicated by "`h_`" (e.g. `h_Dsolution` : handle of vector `Dsolution`)
  - Names of types begin with "`t_`" (e.g. `t_time`)
  - Self-defined pointers in a routine are indicated by "`p_`" (e.g. `p_rmakro => rparBlock%whatsoever%...`).
- The second letter in the variable name is always a small letter (e.g. `isteps` instead of `iSteps`). Following name parts starts with a capital letter (e.g. `rparBlock`.
- Comments for constucts like variables or code samples start before the construct with the samt indent, e.g.

```
! string to be scanned
character(len=*) :: sbuffer

!find the index istart of the first non-blank in the string sbuffer
do i=1,len(sbuffer)
  if (sbuffer(i:i).ne.' ') exit
enddo
```

- Constants are written in block letters and start with (an abbreviation of) the name of the module where the constant is defined. Separated by "_", the actual name follows (e.g. `COMM_MYCONST` indicating that this constant is declared in `communication.f90`). In contrast to variables, the type of the constant is not declared in the name, as constants are used as switches for function calls often, where the actual value and type of the constant are not important (e.g. `AS_SETRHS`).
- The boolean values `.TRUE.` and `.FALSE.` are written in capital letters, as they are constants.
- It is highly recommended to use speaking variables. It is not to expect that this goal can be achieved using variables consisting of 2 letters or less.
- It is not admissible to use index variables like `i`, `j` except in very short loops consisting of at most 5-10 lines of code or in direct implementations of mathematical algorithms like e.g. operating in a matrix, where the indices `i,j` are standard in all textbooks. In this case, the type convention above may be violated in favor of variables like `i,j,k,l`.
- The number of columns in the source code must not exceed 90.
- All constructs are indented by 2 blanks.
- If calling subroutines or functions and the parameter list exceeds one line, then the parameter list is indented, e.g.

```
call assembly_setMatrixBlock(rparBlock, -1, -1, AS_SETRHS,&
                             rparBlock%rmatrixBlockBaseList&
                               (rmatStiff%idiscrId,1)%&
                               rmatrixBlockLevel(1)%ra%rform(1),&
                             rmatStiff, prhs1, 0, 0, cub_getId("G3X3"))
```

- The parameters in the parameter list of a function call are separated by blanks (e.g. see above).
- Functions and subroutines are named by a prefix indicating the name of the module where the function is defined followed by "_" and the actual name of the function / subroutine (e.g. `solver_reinit(...)`). The naming scheme for the following subroutine or function name is the same as for variable names.
- If there are several variants of a function which differ, e.g., in the type of the returned value, these variants may be distinguished by adding appendices separated by underscores from the actual name (e.g. `comm_askAddVar_int(...)`, `comm_askAddVar_double(...)` .

5.2 Documentation

To keep the documentation up-to-date, the documentation for the library functions and data structures is direct included in the source code. This information is structered by special tags which are comments for the fortran compiler. The documentation can be produced in two formats, HTML and TeX. To the TeX output there are some chapters added which contains basic information concerning theoretical background, data formats, tutorials, benchmark etc. To generate the HTML-documentation, run the script

`~/feast/feast/docs/html/generate_html`

to generate the TeX-documentation run the script

`~/feast/feast/docs/tex/generate_tex`

5.2.1 Header

The header information is tagged by the `<name>` tag, which defines the name of the module, and the `<purpose>` tag, which describes the global purpose of the module.

```
!##############################################################################
!# FINITE ELEMENT ANALYSIS & SOLUTION TOOLS  F E A S T  (Release 1.0)   #
!#                                                                            #
!# Authors: Ch.Becker,S.Kilian,S.Turek                                        #
!#          Institute of Applied Mathematics & Simulation                     #
!#          University of Dortmund                                            #
!#          D-44227 DORTMUND                                                  #
!#                                                                            #
!##############################################################################
!#                                                                            #
!# <name> fsystem </name>                                                     #
!#                                                                            #
!#                                                                            #
!# <purpose>                                                                  #
!# System routines like time, string/value conversions etc.                   #
!# </purpose>                                                                 #
!#                                                                            #
!##############################################################################
```

Sample 11: Header documentation

5.2.2 Global constants

Global constants are documented by the tags `<constants>` and `<constantblock>`, this tag supports the `description` attribut for further explanation. The comments for every constant definition must be placed above the corresponding declaration.

```
!<constants>

!<constantblock description="constants for logical values">

  ! logical value 'true'
  integer, parameter :: YES = 0

  ! logical value 'false'
  integer, parameter :: NO = 1

!</constantblock>


!<constantblock description="kind values">

  ! kind value for double precision
  integer, parameter :: DP = selected_real_kind(13,307)

  ! kind value for single precision
  integer, parameter :: SP = selected_real_kind(6,63)

  ! kind value for 32Bit integer
  integer, parameter :: I32 = selected_int_kind(8)

  ! kind value for 64Bit integer
  integer, parameter :: I64 = selected_int_kind(10)

!</constantblock>
!</constants>
```

Sample 12: Global constants documentation

5.2.3 Global types

Global types are documented by the tags `<types>` and `<typeblock>`, this tag supports the `description` attribut for further explanation. The comments for every definition must be placed above the corresponding declaration.

```
!<types>

!<typeblock>

  ! iteration descriptor
  type t_iterator

    ! start value
    integer :: istart

    ! stop value
    integer :: istop

    ! step value
    integer :: istep

  end type
!</typeblock>


!<typeblock>

  ! simulation of an array of double pointers
  type t_realPointer
    real(DP), dimension(:), pointer :: ptr
  end type t_realPointer

!</typeblock>

!</types>
```

Sample 13: Global types documentation

5.2.4 Global variables

Global variables are documented by the tag `<globals>`. The comments for every definition must be placed above the corresponding declaration.

```
!<globals>

  ! global system configuration
  type (t_sysconfig) :: sys_sysconfig

  ! parallel mode 0 = single, 1=double binary (deprecated)
  integer :: sys_parmode

  ! output format
  integer :: coutputFormat

  ! output format
  character(len=256) :: soutputFileName

!</globals>
```

Sample 14: Global variables documentation

5.2.5 Functions

The documentation for a function is tagged by the `<function>` tags, which must include the function definition. The `<function>` tag can contain the following tags for additional documentation:

- `<description>`: contains description of the function
- `<result>`: contains description of the result of the function
- `<input>`: contains description of the input parameters of the function

```
!<function>
  character (len=32) function sys_sd(dvalue, idigits) result(soutput)

    !<description>
    ! This routine converts a double value to a string with idigits
    ! decimal places.
    !</description>

    !<result>
    ! String representation of the value, filled with white spaces.
    ! At most 32 characters supported.
    !</result>

    !<input>

      ! value to be converted
      real(DP), intent(in) :: dvalue

      ! cont of digits
      integer
                    :: idigits
    !</input>
!</function>
```

Sample 15: Function documentation

5.2.6 Subroutines

The documentation for a subroutine is tagged by the `<subroutine>` tags, which must include the subroutine definition. The `<subroutine>` tag can contain the following tags for additional documentation:

- `<description>`: contains description of the function
- `<result>`: contains description of the result of the function
- `<input>`: contains description of the input parameters of the function
- `<output>`: contains description of the output parameters of the function
- `<inputoutput>`: contains description of the input/output parameters of the function

```
!<subroutine>
    subroutine sys_parInElement(Dcoord, dxpar, dypar, dxreal, dyreal)

    !<description>
    !This subroutine is to find the parameter values for a given point (x,y) in real
    !coordinates.
    !
    !Remark: This is a difficult task, as usually in FEM codes the parameter values are
    !known and one wants to obtain the real coordinates.
    !</description>

    !<input>

      !coordinates of the evaluation point
      real(DP),intent(in) :: dxreal,dyreal

      !coordinates of the element vertices
      real(DP), dimension(2,4),intent(in) :: Dcoord

    !</input>

    !<output>

      !parameter values of (x,y)
      real(DP), intent(out) :: dxpar,dypar

    !</output>

!</subroutine>
```

Sample 16: Subroutine documentation

5.3 FEAST Benchmark v2

Since FEAST is a library that is developed and enhanced by several users, a Version Management system called CVS [6] has been used right from the start. Such a system, however, does not suffice to detect regressions. Changes made to the FEAST library to support a requested feature for application A may lead to the situation that another FEAST function gets broken unperceivedly. Such an unintentionally introduced error may even remain undetected for quite some time only to affect, finally, the application B of a person completely uninvolved in the changes having been made to that particular part of FEAST.

FBenchmark2 is a FEAST application designed to detect such regressions. Furthermore, it validates the correctness of the FEAST library. Everyone who has wet his feet in computational mathematics and/or computer science will probably agree that faultless compilers are not a matter of course. Switching to a new parallel platform, upgrading to a new compiler version or simply experimenting with other optimisation flags can result in erroneous binaries.

FBenchmark2 tries to abridge the tedious and time-consuming task of tracing such errors by testing every relevant feature provided by the library, among them:

- Sparse Banded BLAS operations for different number of macros and degree of parallelism, for various grid refinement levels, for the case of variable and constant matrix entries, for different boundary conditions etc.;
- Tests for different kind of ScaRC solvers (i.e. 1-level-ScaRC, 2-level-ScaRC, 3-level-ScaRC, each with different smoothing algorithms and coarse grid solvers);
- Test cases where several macros are clustered to a single matrix block;
- Tests for different kind of multidimensional solvers (ScRich1, Rich-ScRich1, BiCG-ScRich1, BiCG-ScBiCG etc.;
- Test cases that incorporate dynamic grid refining strategies;
- Poisson problems with different kind of boundary conditions;
- Parameter studies for stationary and generalised Stokes problems.

This test suite is continually extended to keep up with new features in the FEAST library or FEAST applications. Currently, the complete test suite features 161 independent test cases.

Additionally, FBenchmark2 ships with a complete set of scripts that take care of retrieving the most recent version of FEAST from CVS, configure the benchmark for a given platform, compile and run the benchmark and finally compare the results with some reference solution. So, you can set up a cron job in order to have an automatic test to keep things safe. At the Chair of Mathematics III, changes to the FEAST library trigger these scripts at night. A meaningful subset of all tests is run weekdays while once every week all tests are performed. This is done for all platforms available at at chair: Linux 32bit and 64bit architectures, Sun Solaris and DEC/HP Alpha. The results from such a benchmark are sent to the FEAST development team via e-mail.

5.3.1 Directory structure

The FBenchmark2 directory contains the following files and subdirectories:

`bin`

A directory containing several scripts. Among them are the cron job scripts that perform daily and weekly regression tests. Furthermore, this directory contains scripts to start benchmark tests on systems using the LoadLeveler queuing system (like IBM p690 (JUMP) at NIC Jülich).

`grids`

A directory containing all grids used for the tests

`include`

A directory containing sub-scripts used by the benchmark control script (intuitively called `runtests`).

All scripts within the FBENCHMARK2 directory are written in TCSH syntax. In TCSH, however, one cannot define functions. As a workaround, these sub-scripts are included in another script whereever this script would normally call a function.

`refsol`

A directory containing reference solutions.

The directory contains subfolders for several build target ID. Due to different SBBLAS routines being used on different architectures as well as due to different BLAS implementations reference solutions for different build target IDs may differ slightly. So, a single reference solution for every test ID can simply not be provided.

If a reference solution for the particular build target ID you want to test is missing, use those from a different build target ID as initial setting. This is feasible as aberrations are usually small. Erroneous code will be definitely not go undetected this way.

In each of the subfolders the reference solution for a single test ID is stored in a separate file.

`results`

A directory containing the solutions obtained from a run of the benchmark control script.

Initially, this directory is missing. It is created by the benchmark control script as soon as the first benchmark test has completed.

It has the same substructure as the directory `refsol`. This facilitates comparison of current results with their corresponding reference solutions. (see also script `bin/manuallycheck_feastresults.sh`).

`scarc`

A directory containing all solver algorithms used for the tests

`src_*`

Directories containing the source code of the different benchmark applications

`tests`

A directory containing files (with extension `fbdef`) that code the different FEAST benchmarks.

`*.fbconf`

Files that contain a list of IDs of benchmark tests. Every ID should match a benchmark coded in one of `fbdef` files in directory `tests`.

Calling `make` with the name of one of the `fbconf` files (omitting the extension, i.e. `make alltests`) will create a benchmark control script that, when invoked, will run exactly those tests coded with their IDs in this very same file.

FBENCHMARK2 currently provides the following `fbconf` files:

`alltests.fbconf`

A list of all test IDs. The control script generated from it will test every relevant FEAST feature incorporated into the benchmark application so far. Such a "all" benchmark will run for at least six hours.

`dailytests.fbconf`

A meaningful subset of all test IDs. The control script generated from it will test the most relevant FEAST features; such a "daily" benchmark will run for one to two hours.

`serialtests.fbconf`

A list of all test IDs that code tests that can be run serially. As the automatic partition feature is usually turned off in FBENCHMARK2's configuration file (`master.dat*`), this typically means that these list consists of all tests that use a grid with a single parallel block.

`singletests.fbconf`

A small set of test IDs, typically used to debug a particular problem.

`Makefile`

Makefile to compile the benchmark application and create a benchmark control script from one of the files with extension `fbconf`.

For any `fbconf` file there is automatically a corresponding make target. Bearing in mind the four `fbconf` files listed above, it is hence valid to invoke any of

```
% make alltests
% make dailytests
% make serialtests
% make singletests
```

If you add a new `fbconf` file to the repository, e.g. `griddeform.fbconf` (containing IDs of tests that deal with grid deformation), then there will be immediately the corresponding make target:

```
% make griddeform
```

which will create a script that will only perform those grid deformation tests.

5.3.2 Sample run

A typical run consists of the following commands:

```
% make clean
% make configure
% make compile
% make [dailytests|alltests|singletests]
% make run
```

Agglomeration of the different `make` targets into a single command is possible:

```
% make clean
% make configure compile dailytests run
```

Figure 17 shows how a (non)typical output looks like. “Nontypical” because deviations in the results are detected. This means that either the underlaying code has changed or compiler or compiler settings.

The first part of the result output shows the result of the actual computation, the second part shows the reference result.

```
ID     = SCARC0001
CLASS  = SCARC
GRID   = grids/cyl/cyl_24m_24mb_4p_aniso.feast
SCARC  = scarc/CG.scarc
MGLVLS = 3,4,5

START:Thu Jun 2 17:31:28 CEST 2005

Running 3....Done
Running 4....Done
Running 5....Done

Storing results.
Comparing current with reference solution...
TEST FAILED

LEVEL   #NEQ        ||L2err||      c    #NIT         ||RES||        hmin         AR
1,2c1,2
< 3      1536    8.29885308E-03  0.830   75      1.90171268E-04  3.74643252E-05  174
< 4      6144    2.34384979E-03  0.905   141     8.72524109E-04  3.74643252E-06  869
---
> 3      1536    8.29885302E-03  0.830   75      1.90054245E-04  3.74643252E-05  174
> 4      6144    2.34384978E-03  0.905   141     8.72461370E-04  3.74643252E-06  869

Difference:
LEVEL   #NEQ        ||L2err||      c    #NIT         ||RES||        hmin         AR
  -        -         7.2299e-09      -    -        6.1573e-04                -    -
  -        -         4.2665e-09      -    -        7.1910e-05                -    -
  -        -                  -      -    -                 -                -    -
Note: Values for ||L2err|| and ||RES|| are relative, not absolute!

FBMARK l2error matmod solution =   0.63992618D-03
FBMARK sol/sol2 (min=1/max=1):  min= 1.000000 max= 1.000000

FINISH:Thu Jun 2 17:37:57 CEST 2005

---------------------------------------


SUMMARY:

tests           : 1
tests failed    : 1
tests unverified: 0
tests passed    : 0


The tests that failed are coded in the following files:
SCARC0001
```

Sample 17: Fbenchmark2 sample run

5.3.3 Available scripts

The `bin` subdirectory contains several scripts to facilitate the use of FBENCHMARK2. Their use and purpose is explained in the following:

`create_script.pl`

A Perl script used by the Makefile when it creates the benchmark control script (intuitively called `runtests`). It parses a given ASCII file containing a list of test IDs, looks up the settings associated

with these test IDs (stored in one of the `fbdef` files in subdirectory `tests`) and creates instruction blocks (TCSH syntax) for all the tests requested. These instructions are integrated into `runtests`.

If a test ID is found for which there is no definition in any of the `fbdef` files, it will report a warning and ignore the test ID.

`ll_*` A set of scripts written for IBM p690 (JUMP) at NIC Juelich, Germany. There, a queuing system is used; jobs have to be submitted via IBM LoadLeveler (a tool for managing serial and parallel jobs over a cluster of servers).

`ll_results`
: A TCSH script that creates a brief report on successful and failed benchmark tests.

`ll_runtime`
: A TCSH script that parses the output of a benchmark run and calculates the aggregated runtime of all tests performed.

`llschedule_alltests`
: A TCSH script that creates a sequence of benchmark control scripts to be submitted to the IBM LoadLeveler one by one.
Version to perform all tests coded in `alltests.fbconf`.

`llschedule_dailytests` A TCSH script that creates a sequence of benchmark control scripts to be submitted to the IBM LoadLeveler one by one.
Version to perform all tests coded in `dailytests.fbconf`.

`llschedule_singletests`
: A TCSH script that creates a sequence of benchmark control scripts to be submitted to the IBM LoadLeveler one by one.
Version to perform all tests given as test IDs in argument list.
Example usage:

```
% bin/llschedule_singletests BC0001 SCARC0012 STOKES0004
```

`manuallycheck_feastresults.sh`

A TCSH script that compares FEAST benchmark result files in two directories with each other. It takes two directory paths as input and will generate a report like `runregressiontest` and `runregressiontest_child`.

Remember that the results of every benchmark test is stored in a single file. The script will compare the contents of files with matching names and list any difference for every test found in the first directory.

Example usage:

```
% bin/manuallycheck_feastresults.sh results/pc-opteron-linux64-lammpi-ifort-goto \
                                    refsol/pc-opteron-linux64-lammpi-ifort-goto
```

The script is particularly useful

- if you lost the screen output of a benchmark run. For instance, when the terminal history does not provide enough lines.
- if you want to *manually* compare results. For instance, when checking the results of a benchmark run that is still running or when comparing results from a benchmark run with reference solutions for a different build target ID:

  ```
  bin/manuallycheck_feastresults.sh results/some-obscure-target \
                                      refsol/pc-opteron-linux64-lammpi-ifort-goto
  ```

`runregressiontest`

A TCSH script that anonymously checks out the HEAD revision of FEAST from CVS to directory `$HOME/nobackup/feast/feast`. Any existing (previous checkout) directory of that name is renamed to `$HOME/nobackup/feast/feast.old`. If differences between these two checkouts are found, FBENCHMARK2 is initiated: The FEAST benchmark application is cloned to `$HOME/nobackup/feast/feast/`

`fbenchmark2_$HOSTNAME`, configured for the current build target ID, compiled and run. A report is sent upon completion of the test to the FEAST development team.

The script takes two arguments: The first describes the test set to run. It simply is the name of one of the `fbconf` files (omitting the extension). The second argument is a description of this test set to be used be used in the e-mail subject.

Example usage:

```
% bin/runregressiontest 'dailytests' 'daily run'
```

The script is the "master" instance of two fraternal twin scripts to detect regressions in the FEAST library: `runregressiontest` and `runregressiontest_child`.

`runregressiontest_child`

A TCSH script that is the "slave" instance of the two fraternal twin scripts to detect regressions in the FEAST library. It does not retrieve a recent copy of FEAST from CVS, but waits if the "master" instance decides to initiate a FEAST benchmark. If so, the "slave" instance also clones the FEAST benchmark application to `$HOME/nobackup/feast/feast/fbenchmark2_$HOSTNAME`, configures it for the current build target ID, compiles and runs it. A report is sent upon completion of the test to the FEAST development team.

Invocation of the script is identical to that of `runregressiontest`.

5.3.4 Test classes

The benchmark consists of the following test classes:

bc.fbdef:	tests for checking the handling of boundary conditions
dynref.fbdef:	tests for checking the dynamic refinement
fb.fbdef:	tests for checking the handling of fictitious boundaries
sbblas.fbdef:	tests for checking the SBBLAS library
scarc.fbdef:	tests for checking several scarc solvers
stokes.fbdef:	tests for solving stationary and instationary Stokes problems, these tests check in particular the solution of multidimensional problems

Every `fbdef` file contains definitions blocks of the following structure:

```
id        = SBBLAS0002
class     = SBBLAS
descr     = sbblas tests
grid      = macrotest/mt_a_0.01_2x2
appl      = sbblas,MATCONST:NO,AUX1:MV
bc        = DIR
scarc     = CG_MG_Z,LSMOOTHER:TRIGS,MAXITER:3
mglevel   = 5
rhs       = ZERO_ON_EHQ
masterdat = master.dat
```

Explanation of the keywords:

`id`

Unique identifier of the test.

`class`

Class identifier of the test. The value should identical for all tests in a single `fbdef` file. Typically, this file bears the class name.

`descr`

Short description of the test. This text is printed to screen when the test is started and serves as a hint what this test is about.

`grid`

Subpath of the grid file to use for this test.

As we learnt in section 5.3.1, all grids are stored in subdirectory `grids`. This directory name is automatically added as prefix to the value given here.

`appl`

Name of the test application to use for this test.

Additional parameters can be defined as comma separated key:value lists.

To add a benchmark application, create a subdirectory with prefix `src_<appl>`. For more details, see section 5.3.5.

Currently, the following test applications are integrated into the benchmark:

disk

[Description still missing...]

dynref

[Description still missing...]

fibound

[Description still missing...]

matmod

[Description still missing...]

poisson

[Description still missing...]

sbblas

[Description still missing...]

stokes

The application solves the generalised Stokes equation

$$\begin{aligned} \gamma\mathbf{u} - \nu k\Delta\mathbf{u} + \nabla p &= \mathbf{f} && \text{in } \Omega, \\ \nabla\cdot\mathbf{u} &= 0 && \text{in } \Omega, \\ \nu\tfrac{\partial\mathbf{u}}{\partial n} + p\cdot n &= 0 && \text{on } \Gamma_{\mathrm{N}}, \\ \mathbf{u} &= \tilde{\mathbf{g}} && \text{on } \Gamma_{\mathrm{D}}, \\ \mathbf{u} &= \mathbf{u}_0 && \text{in } \Omega, \end{aligned} \tag{5.1}$$

where $\mathbf{u}$ and p denote velocity and pressure, respectively. n is the outer normal vector and Γ_{D} and Γ_{N} the boundary parts with, respectively, Dirichlet and Neumann boundary conditions (i.e. inflow, outflow and adhesion conditions). The kinematic kinematic viscosity ν, finally, is assumed constant and positive: $\nu > 0$, $\nu \neq \nu(p, c_p)$.

Test cases include

- stationary Stokes problems;
- generalised Stokes problems, with time steps in the range $[10^{-3}, 10^{3}]$;
- grids with a single and multiple macros;
- grids with a single and multiple parallel blocks,;
- isotropic and anisotropic grids.

`bc`

Definition of the boundary conditions used, given as comma separated list.

Valid values are `DIR`, `MIXED` and `DIRZERO`.

`scarc`

Definition of the ScaRC solver used. Additional parameters can be defined as comma separated key:value lists.

`mglevel`

Definition of the multi grid levels on which the test should be performed. The levels are given as comma separated list.

Example: Execution on multi grid level 3 and 4 is given via

```
mglevel = 3,4
```

`rhs`

Definition of the right hand side for the test problem. Valid values are `ZERO_ON_EHQ`, `ZERO`, `MINUS8DX` and `n.a.`.

`masterdat`

Name of the configuration file to use.

Simple matrix-vector multiplication tests take a simpler configuration file than adaptive refinement tests with Poisson problem or Stokes problem tests.

Every keyword found in an `fbdef` file is exported as environment variable. By this mechanism, it is usually sufficient to provide a single configuration file (the value of `masterdat`) for one class of tests. Instead of using fixed values for certain keywords, environment variables are used within the configuration file. The benchmark application will automatically replace them with the appropriate value at runtime.

The list of variables presented above can be extended. This is, for instance, done for the `stokes` application. Tests defined in `tests/stokes.fbdef` additionally define the keywords `mdsfile`, `gamma`, `k` and `nu`. The corresponding configuration file `master.dat.stokes`, hence, contains the following lines

```
gamma           $GAMMA    # parameter for switching on and off the reactive part
k               $K        # parameter indicating the time step size
nu              $NU       # kinematic viscosity
mdsFile         $MDSFILE  # name of file coding the solver for multidimensional problems
```

Sample 18: Excerpt from `master.dat.stokes`

5.3.5 Extending the benchmark application

As mentioned in section 5.3.4, the FBenchmark2 application features seven different test classes so far: SBBLAS tests, tests for matrix operations, tests for dynamic refinement and fictitious boundary conditions and tests involving the solution of Poisson, Stokes and elasticity problems.

Every class is coded as a separate application in subdirectories with prefix `src_`. The variable part of an application subdirectory must coincide with the value given as `appl` in the related `fbdef` file.

Inside, an application is set up like every other in `feast/feast/applications`: There have to be at least two modules called `userdef.f90` and `slavemod.f90`. The first provides routines that influence assemblation of matrices and right hand sides, namely coefficient functions and a function returning information on boundary values. Furtheron, it may provide exact solutions for test problems and, finally,

it provides facilities to postprocess data when exporting visualisation data to file as well as an interface to export user-defined visualisation output (other than AVS or GMV format).

The latter implements program control: reading of configuration files, assembling of matrices, the solving process and possibly the evaluation of results as well as visualisation output.

You can add additionally applications-specific modules if you like. Do not forget to include them into your personal `configure` script, then. If your application can get along with the two standard modules `userdef.f90` and `slavemod.f90` you might opt to choose the master copy of `configure`. Create a symbolic link to `../master.configure`.[1]

So, if you plan to extend the benchmark application, you will have to do the following:

1. Add a subdirectory `src_<appl>` where `<appl>` coincides with the value given for the keyword `appl` in `fbdef` file (see next item).

   Add at least two modules called `userdef.f90` and `slavemod.f90` which code your application.

2. Create a new `fbdef` file in directory `tests`.

3. Add the test IDs you defined in the newly created `fbdef` file to `alltests.fbconf` (at least) and any other appropriate `fbconf` file (e.g. the `dailytests.fbconf`). Otherwise your test will not be executed by the nightly FBenchmark2 regression tests.

5.3.6 Solver algorithms

BICG_MG_MG_ZZ.scarc:
BICG with 2-Level-SCARC-MG as preconditioner and direct coarse grid solvers,
supported options are: MAXITER, STEPS, OMEGA, LOCALITER, LSMOOTHER

CG_MG_MG_ZZ.scarc:
CG with 2-Level-SCARC-MG as preconditioner and direct coarse grid solvers,
supported options are: MAXITER, STEPS, OMEGA, LOCALITER, LSMOOTHER

CG_MG_Z.scarc:
CG with MG as preconditioner and direct coarse grid solver,
supported options are: MAXITER, STEPS, OMEGA, LOCALITER, LSMOOTHER

CG.scarc:
plain CG with local preconditioner,
supported options are: MAXITER, LSMOOTHER

BICG.scarc:
plain BICG with local preconditioner,
supported options are: MAXITER, LSMOOTHER

MG_MG_MG.scarc:
3-Level-SCARC-MG with BICG schemes as coarse grid solvers,
supported options are: MAXITER, OMEGA, LSMOOTHER

MG_MG_MG_ZZZ.scarc:
3-Level-SCARC-MG with direct coarse grid solvers,
supported options are: MAXITER, OMEGA, LSMOOTHER

[1] Do not forget to extend `feast/feast/Makefile.symlinks` accordingly afterwards. The most easy way to do this is invoking `bin/symlinks2Makefile` in directory `feast/feast`. This script will create a Makefile to restore every symbolic link found underneath `feast/feast`.

MG_MG.scarc:
    2-Level-SCARC-MG with BICG schemes as coarse grid solvers,
    supported options are: MAXITER, OMEGA, LSMOOTHER

MG_MG_ZZ.scarc:
    2-Level-SCARC-MG with direct coarse grid solvers,
    supported options are: MAXITER, OMEGA, LSMOOTHER

MG.scarc:
    MG with BICG scheme as coarse grid solver,
    supported options are: MAXITER, OMEGA, LSMOOTHER

MG_Z.scarc:
    MG with direct coarse grid solver,
    supported options are: MAXITER, OMEGA, LSMOOTHER

Chapter 6

Module overview

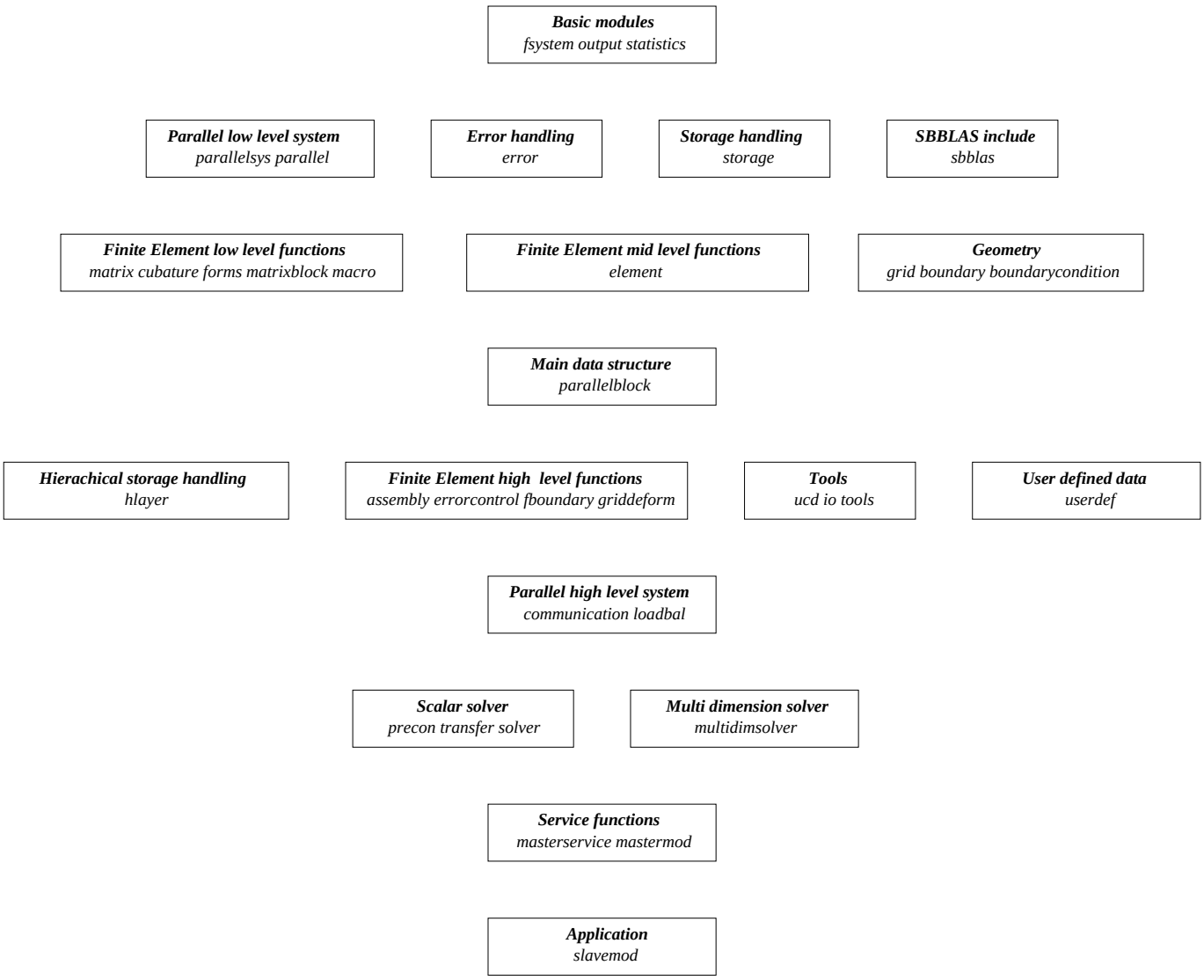

Figure 6.1: FEAST module overview

This chapter gives a short overview of the structure of the FEAST library and the purposes of the single modules. Figure 6.1 shows the structure.

Module auxiliary

Auxiliary routines like analytic functions.

Module error

This module contains the definition of the error codes and the routine which prints the output text if an error has ocuured.

Module fsystem

This module contains system routines like time measurement, string/value conversions, auxiliary routines and several sorting routines.

Module io

This module contains several routines for input/output purposes.

Module output

Handling the output of messages to terminals and logfiles

Module storage

This module contains the memory management of the FEAST package. For performance reasons, we implemented our own memory management for the data vectors of FEAST and decided not to use the one provided by Fortran. When starting the program, every process calls a initialisation routine which reserves a heap of adjustable size for every data type. The routine storage_new reserves a vector. This routine returns a vector handle. The routine storage_getbase creates a pointer to this handle, which can be used like a normal array. (DOCU IMCOMPLETE)

Module sbblas

Driver routines for SBBLAS package

Module statistics

This modules contains several routines for calculating computation and MFLOP/sec rates.

Module ucd

This module covers the job of exporting simulation results to several formats. Export of node-based and cell-based data to AVS and GMV documents is fully supported.

Module hlayer

This module implements the hierachical layer structure of the vector management. It contains routines for reservation and freeing of vectors, further access functions and inquiry functions.

Module loadbal

This module contains the definition of the loadbalancing strategies.

Module macro

This module defines the data structure for the basic data element macro. Further it contains routines for sending/receiving and copying macros.

Module mastermod

This modules contains the master part of the main program. Every request from a slave to the master is handled in the main routine startmaster. In this routine, there exists an endless loop, in which all messages are recieved. Depending on the message header, the program goes in an appropriate branch and fulfills the request of the calling slave(s).

Module masterservice

This module provides all the routines which are called by the endless loop in the module mastermod.f90.

Module matrix

This modules contains several control routines for the matrix multiplication. It acts as an frontend for the according SBBLAS routines. Further it defines the matrix types and toolkit routines for converting and output.

Module matrixblock

This module contains the basic atomic unit called matrixblock.

Module parallelblock

(yet to come)

Module grid

This module contains grid handling routines for generating and refining routines. There are two types of grids, direct stored for not uniform and analytic calculated for uniform grids. One grid is assigned to a matrixblock object. The active grid is selected by setting the global variable gr_rwgrid. The communication with the other functions are also handled by global variables gr_.... THE COMMENTS HAVE STILL TO BE IMPROVED AND CORRECTED.

Module fboundary

In this module, several routines for modifying fictitious boundaries are gathered.

Module assembly

This module contains routines for the assembly of FEM matrices and right hand sides. Furthermore, it contains routines for the setting of boundary conditions and values. Moreover, there are routines for matrixblock object scaling, multiplication and adding.

Module boundary

This module implements the handling of (curved) boundaries and their parametrisation. It provides the basic segment types line and circle as well as open and closed NURBS. Also analytically defined boundaries are supported (WIP). Furthermore, the module contains routines for creating, reading and transferring boundary components.

Module boundarycondition

This module contains several routines to describe boundary conditions. Each boundary condition can consist of several description blocks. Each block describes an interval or a point with Neumann or Dirichlet boundary condition. The current version allows the definition per node index, the definition per parameter value is not implemented yet.

Module cubature

This module provides several cubature schemes with their nodes and and weights. These values refer to the reference element

Module element

This module contains the generation routines for the finite element shape functions. Currently, only the bilinear element Q_1 (Elem_Q1) is fully supported.

Module errorcontrol

This module contains routines which are related to error estimation procedures. This contains the routines ec_compSPR and ec_compPPR for obtaining recovered gradients, which are used in ec_h1Est to estimate the H1-error. Furthermore, there are and will be routines to enable dual weighted residual based error control.

Module forms

This module contains the description of the discretisation in form of the weak formulation. For further information see the example in an earlier section in this manual.

Module griddeform

This module contains all routines regarding grid distortion, grid smoothing and grid deformation.

Module structmech

This module contains different routines for structural mechanics. Especially it realises a consequent element-wise assembling strategy which is commonly used in the structural mechanics community and thus should make the extension to more complicated "elements" more convenient. (In the structural mechanics community the corresponding element routines are simply called "elements" which should not be mistaken for the "mathematical" usage of this term.)

Module tools

The module contains routines for calculating norms and scalar products on different hierachical layers.

Module precon

This module contains several routines for preconditioing schemes like Jacobi, Gauss-Seidel, Tri-Gauss-Seidel. The routines are divided in two groups. the first group initialises some structures like factorization, the other group performs the actual preconditiong to a given vector. The comments have to be improved.

Module solver

This module contains the FEAST solver engine ScaRC, multigrid solvers, CG and BiCGStab solvers as well as routines for setting of boundary conditions and values. Moreover, there are routines for matrixblock object scaling, multiplication and adding.

Module multidimsolver

This module contains numerous solver routines for vector-valued problems.

Module transfer

This modules contains routines to perform the prolongation and and restriction within the multigrid cycle. These routines are suitable for the conforming bilinear finite element.

Module communication

This module contains the basic routines for exchanging matrices and vectors and for communication for gathering operations like scalar products.

Module parallel

This module defines the basic low level communication routines. It provides routines for exchanging integer and double values. Further it contains mechanisms for synchronisation of parallel processes.

Module parallelsys

This module implements the parallel initialisation routines.

Chapter 7

Module reference for application programmers

7.1 Module auxiliary

Purpose: Auxiliary routines like analytic functions.

7.1.1 Function aux_danalyticFunction

Interface:

`aux_danalyticFunction(x, y, cderiv, cselect, dparam)`

Description:
This function provides some analytic functions, which can be used for validating your FE code.

Input variables:

Name	Type	Rank	Description
`x, y`	`real(DP)`		the x,y coordinates
`cderiv`	integer		derivative of the function to be calculated
`cselect`	integer		selector for the desired function
`dparam`	`real(DP)`		optional parameter to influence the solution

Result:
real(DP) : (The result of the function calculation) dval

7.2 Module error

Purpose: This module contains the definition of the error codes and the routine which prints the output text if an error has ocuured.

7.2.1 Subroutine error_print

Interface:

`error_print(icode, sroutine, bcritical, iarg1, iarg2, darg1, darg2, sarg1, sarg2)`

Description:
This routine prints the error message for the given error code. If the occured error is critical so that the further program execution is impossible the program terminates.

Input variables:

Name	Type	Rank	Description
sroutine	c (len = *)		name of the calling routine
bcritical	logical		flag if the error is critical, then terminate the program
icode	integer		error code
ireturnCode	integer		
iarg1	integer		integer argument 1 (optional)
iarg2	integer		integer argument 2 (optional)
darg1	real(DP)		double argument 1 (optional)
darg2	real(DP)		double argument 2 (optional)
sarg1	c(len = *)		string argument 1 (optional)
sarg2	c(len = *)		string argument 2 (optional)

7.2.2 Function error_askError

Interface:

```
error_askError()
```

Description:
This function returns the error constant of the last occurred error. If no error has occurred ERR_NO_ERROR will be given back. Naturally, this works only for noncritical errors!

Result:

integer : error code

7.2.3 Subroutine error_clearError

Interface:

```
error_clearError()
```

Description:
This routine resets the internal error memory to ERR_NO_ERROR.

7.3 Module fsystem

Purpose: This module contains system routines like time measurement, string/value conversions, auxiliary routines and several sorting routines.

Constant definitions:

Name	Type	Purpose
Purpose: constants for logical values		
YES	integer	logical value 'true'

NO	integer	logical value 'false'

Purpose: kind values for floats

DP	integer	kind value for double precision
SP	integer	kind value for single precision

Purpose: kind values for integers

I32	integer	kind value for 32Bit integer
I64	integer	kind value for 64Bit integer

Purpose: sort algorithms

SORT_HEAP	integer	heap sort (default); reliable all-rounder
SORT_QUICK	integer	quicksort (cutoff = 50), then insert-sort
SORT_INSERT	integer	insertsort; for small arrays

Purpose: system flags

BEEP	c(1)	constant for a system beep
NEWLINE	c(1)	constant for breaking line in a string (useful for comm_sendmasterstring(...))
SYS_STRLEN	integer	standard length for strings in FEAST
SYS_MAXNADDVARS	integer	maximum number of additional variables in the master file
SYS_MAXNSCARC	integer	maximum number of scarc solver definition files
SYS_MAXNSCARCTOKENS	integer	maximum number of tokens per line in scarc solver definition file
SYS_FEAT	integer	deprecated
SYS_PI	real(DP)	mathematical constant Pi
SYS_DEBUG	integer	flag for debug mode
SYS_MAXREAL	real(DP)	maximal values for real variables
SYS_MAXINT	integer	maximal values for integer variables
INCX	integer	increment value = 1
SYS_TIMERSTART	integer	control value for sys_deltatime
SYS_TIMERSTOP	integer	control value for sys_deltatime
SYS_APPEND	integer	flag for appending data to a file (used in io)

SYS_REPLACE	integer	flag for replacing a file (used in io)
SYS_GMV	integer	flag for GMV ouput
SYS_AVS	integer	flag for AVS output
SYS_GMV_SUFFIX	c(6)	file extension for GMV files
SYS_AVS_SUFFIX	c(6)	file extension for AVS files
SYS_MAXGLOABLSOL	integer	maximum number of global solution in one gmv file
GRID_GETLEVMIN	integer	???
GRID_GETMINLEV	integer	
GRID_GETMAXLEV	integer	

Purpose: system signals

SIGILL	integer	
SIGTRAP	integer	
SIGABRT	integer	
SIGEMT	integer	
SIGFPE	integer	
SIGBUS	integer	
SIGSEGV	integer	

Purpose: communication message tags

COMM_MASTERFIN	integer	finish master process
COMM_MASTER_RECEIVEVISDATA	integer	receive visualisation data sent from comm_sendVisData
COMM_MASTERSYNC	integer	
COMM_MASTERTIMER	integer	
COMM_MASTERPRINT	integer	
COMM_MASTERPRINTTAG	integer	
COMM_MASTER_PREPREF	integer	tell the master to organise adaptive refinement
COMM_SLAVE_PREPREF	integer	send information to perform adaptive refinement to slave
COMM_FREE_GRID_STORAGE	integer	
COMM_MASTERAVSTAG	integer	
COMM_MASTERSYNCTAG	integer	
COMM_MASTERTIMERTAG	integer	
COMM_MASTERCGRIDTAG	integer	
COMM_MASTERADDBOUNDARY	integer	
COMM_MASTERADDBOUNDARYTAG	integer	

COMM_MASTERMOVEBOUNDARY	integer
COMM_MASTERMOVEBOUNDARYTAG	integer
COMM_MASTERROTATEBOUNDARY	integer
COMM_MASTERROTATEBOUNDARYTAG	integer
COMM_TOPDOWNTAG	integer
COMM_DOWNTOPTAG	integer
COMM_MAXDB	integer
COMM_MSGIDDIAG	integer
COMM_MSGIDEDGE	integer
COMM_MSGIDFIN	integer
COMM_MSGIDCGRID	integer
COMM_MASTERSOLVE	integer
COMM_MASTERSOLVEINIT	integer
COMM_MASTERSOLVEINITTAG	integer
COMM_MASTERSOLVEPREPARE	integer
COMM_MULTIDIM_INIT_DIRECT	integer
COMM_MULTIDIM_SOLVE_DIRECT	integer
COMM_MASTERADDVAR	integer
COMM_MASTERADDVARTAG	integer
COMM_MASTERREADFEAT	integer
COMM_MASTERREADFEATTAG	integer
COMM_MASTERAVSUSER	integer
COMM_EXCMACRO	integer
COMM_EXCMACROTAG	integer
COMM_DYNADAP	integer
COMM_DYNADAPTAG	integer
COMM_LOADBAL	integer
COMM_LOADBALTAG	integer
COMM_EXCMAT	integer
COMM_EXCVEC	integer
COMM_MIDEXT	integer
COMM_MIDINT	integer
COMM_MIDREST	integer

Purpose: ucd output flags describing the data type

UCD_VISDATATYPE_NODEBASED	integer	Flag stating that the associated data is node-based
UCD_VISDATATYPE_CELLBASED	integer	Flag stating that the associated data is cell-based

Purpose: ucd output flags coding what to do with data fields

UCD_NOEXPORT	integer	Flag stating that the associated data should not be exported
UCD_NO_AGGLOMERATE	integer	Flag stating that the associated data should be treated stand-alone
UCD_AGGLO_VELOCITYFIELD	integer	Flag stating that the associated data should be agglomerated to a velocity field (if possible within concerning UCD data format)
UCD_ADD_TO_NODES_X_COORDS	integer	Flag stating that the associated data should be added to the x-coordinates of the nodes.
UCD_ADD_TO_NODES_Y_COORDS	integer	Flag stating that the associated data should be added to the y-coordinates of the nodes.

Purpose: ucd output flags coding which grid statistics to export

UCD_MINCONVRATE	integer	Flag stating that the minimal convergence rate per parallel block should be exported
UCD_MAXCONVRATE	integer	Flag stating that the maximal convergence rate per parallel block should be exported
UCD_AVGCONVRATE	integer	Flag stating that the average convergence rate per parallel block should be exported
UCD_HMIN	integer	Flag stating that the minimal grid size per parallel block should be exported
UCD_ASPECTRATIOS	integer	Flag stating that the maximal aspect ratio per parallel block should be exported

Type definitions:

Type name	component name	Type	Rank	Purpose
t_sysconfig	global configuration type			
	sprojectID	c(SYS_STRLEN)		project id
	sprojectDir	c(SYS_STRLEN)		project directory
	slogDir	c(SYS_STRLEN)		log directory
	coutputLevel	integer		output level
	sgridFile	c(SYS_STRLEN)		name of feast grid file
	scfg_gridfile_fbc	c(SYS_STRLEN)		
	scfg_gridfile_fpart	c(SYS_STRLEN)		
	scfg_gridfile_fmesh	c(SYS_STRLEN)		

scfg_gridfile_fgeo	c(SYS_STRLEN)		
scfg_gridfile_ffgeo	c(SYS_STRLEN)		
sabsolutePath	c(SYS_STRLEN)		
cloadBalanceMode	integer		loadbalancing mode
sloadBalanceFile	c(SYS_STRLEN)		file name for loadbalancing
canisoRefMode	integer		anisotropic refinement mode
cmatConstMul	integer		use constant matrix vector multiplication if possible
cfastMatAssembly	integer		use fast matrix assembly if possible
ccalcSmoothNorm	integer		calculate norm inside smoother
clongOutputFormat	integer		long output format
nscarc	integer		number of scarc definitions
sscarcFileNames	c(SYS_STRLEN)	SYS_MAXNSCARC	scarc file names
soutputDataFormat	c(SYS_STRLEN)		data output file format as string
coutputDataFormat	integer		data output file format as constant
soutputDataFileName	c(SYS_STRLEN)		data output file name
ioutputDataLevel	integer		data output level: l .gt. 0 : perform advanced output on level l l .lt. 0 : perform standard output (with duplicate vertices) on level -l l=0 : perform no output
badvancedOutput	logical		flag if advanced output shall be used (advanced means: on macro boundaries there are no vertex duplicates anymore!)
cuseStoredSolution	integer		use stored solution
cuseStoredMatrix	integer		use stored matrix
idefaultStorageHeapSize	integer		default size of storage heaps
idefaultNStorageDescr	integer		default number of storage descriptors

	imaxMGLevel	integer		maximum multigrid level
	isbblasMV	integer		sbblas mv implementation id
	isbblasPRSLOWL	integer		sbblas prslowl implmentation id
	isbblasMVTRI	integer		sbblas mvtri implementation id
	isbblasPRSLINE	integer		sbblas prsline implementation id
	iaddvaridx	integer		index of additonal config variables
	saddVarName	c(SYS_STRLEN)	SYS_MAXNADDVARS	name of additional config parameters
	saddVarValue	c(SYS_STRLEN)	SYS_MAXNADDVARS	value of additional config parameters
	cautoPartition	integer		autopartition
	cautoPartitionWeighted	integer		autopartition with weighting
	cuseExtraStorageMatrixEdges	integer		use extra storage for matrix edges

t_iterator	iteration descriptor			
	istart	integer		start value
	istop	integer		stop value
	istep	integer		step value

t_realPointer	simulation of an array of double pointers			
	ptr	real(DP)	:	

Global variable definitions:

Name	Type	Rank	Purpose
sys_dtimeMax	real(DP)		
sys_sysconfig	t_sysconfig		global system configuration
sys_parmode	integer		parallel mode 0 = single, 1=double binary (deprecated)

7.3.1 Subroutine sys_throwFPE

Interface:

sys_throwFPE()

Description:
This routine throws a floating point exception for debugging purposes to prevent the debugger to exit the program.

7.3.2 Function sys_sgetOutputFileBody

Interface:

`sys_sgetOutputFileBody()`

Description:
This routine provides the body of the outputfile name defined in the master.dat. Example : "solution.gmv", the routine provides "solution".

Result:

character(len=SYS_STRLEN) : filename body

7.3.3 Function sys_sgetOutputFileSuffix

Interface:

`sys_sgetOutputFileSuffix()`

Description:
This routine provides the body of the outputfile name defined in the master.dat. Example : "solution.gmv", routine provides ".gmv".

Result:

character(len=6) : filename extension

7.3.4 Function sys_sgetOutputFileName

Interface:

`sys_sgetOutputFileName()`

Description:
This routine composes the complete name of the visualisation output file from basename specified in a configuration file (master.dat) and the appropriate suffix based on the requested visualisation output format.

Result:

character(len=SYS_STRLEN) : filename

7.3.5 Subroutine sys_version

Interface:

`sys_version(ifeastVersionHigh, ifeastVersionMiddle, ifeastVersionLow, sreldate)`

Description:
This subroutine returns the library version information.

Output variables:

Name	Type	Rank	Description
ifeastVersionHigh	integer		high version number
ifeastVersionMiddle	integer		middle version number
ifeastVersionLow	integer		low version number
sreldate	c(*)		release date

7.3.6 Subroutine sys_getBuildCPU

Interface:
`sys_getBuildCPU(scpu)`

Description:
This subroutine returns informations about the built cpu.

Output variables:

Name	Type	Rank	Description
sarch	c(SYS_STRLEN)		build architecture
scpu	c(*)		build cpu
sos	c(SYS_STRLEN)		build operating system
smpienv	c(SYS_STRLEN)		use MPI environment
scompiler	c(SYS_STRLEN)		build compiler
sblas	c(SYS_STRLEN)		used BLAS

7.3.7 Subroutine sys_getBuildCompiler

Interface:
`sys_getBuildCompiler(scompiler)`

Description:
This subroutine returns informations about the build architecture.

Output variables:

Name	Type	Rank	Description
sarch	c(SYS_STRLEN)		build architecture
scpu	c(SYS_STRLEN)		build cpu
sos	c(SYS_STRLEN)		build operating system
smpienv	c(SYS_STRLEN)		use MPI environment
scompiler	c(*)		build compiler
sblas	c(SYS_STRLEN)		used BLAS

7.3.8 Subroutine sys_getBuildArch

Interface:

`sys_getBuildArch(sarch)`

Description:
This subroutine returns informations about the build architecture.

Output variables:

Name	Type	Rank	Description
`sarch`	`c(*)`		build architecture
`scpu`	`c(SYS_STRLEN)`		build cpu
`sos`	`c(SYS_STRLEN)`		build operating system
`smpienv`	`c(SYS_STRLEN)`		use MPI environment
`scompiler`	`c(SYS_STRLEN)`		build compiler
`sblas`	`c(SYS_STRLEN)`		used BLAS

7.3.9 Subroutine sys_getBuildEnv

Interface:
`sys_getBuildEnv(sarch, scpu, sos, smpienv, scompiler, sblas)`

Description:
This subroutine returns informations about the built environment.

Output variables:

Name	Type	Rank	Description
`sarch`	`c(*)`		build architecture
`scpu`	`c(*)`		build cpu
`sos`	`c(*)`		build operating system
`smpienv`	`c(*)`		use MPI environment
`scompiler`	`c(*)`		build compiler
`sblas`	`c(*)`		used BLAS

7.3.10 Subroutine system_init

Interface:
`system_init()`

Description:
This subroutine initialises some internal data structures.

Global variables:
`sys_dtimeMax, SYS_MATCONST, SYS_FASTASS, SYS_CALCSMOOTHNORM, SYS_PI`

7.3.11 Function sys_sd

Interface:

`sys_sd(dvalue, idigits)`

Description:
This routine converts a double value to a string with idigits decimal places.

Result:
character (len=32) : String representation of the value, filled with white spaces. At most 32 characters supported.

Input variables:

Name	Type	Rank	Description
`dvalue`	`real(DP)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.12 Function sys_sdE

Interface:
`sys_sdE(dvalue, idigits)`

Description:
This routine converts a double value to a string with idigits decimal places in scientific notation.

Result:
character (len=24) : String representation of the value, filled with white spaces. At most 24 characters supported.

Input variables:

Name	Type	Rank	Description
`dvalue`	`real(DP)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.13 Function sys_si

Interface:
`sys_si(ivalue, idigits)`

Description:
This routine converts an integer value to a string of length idigits.

Result:
character (len=32) : String representation of the value, filled with white spaces. At most 32 characters supported.

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer`		value to be converted
`idigits`	`integer`		number of decimals

7.3.14 Function sys_si0

Interface:

```
sys_si0(ivalue, idigits)
```

Description:
This routine converts an integer value to a string of length idigits.

Result:
character (len=32) : String representation of the value, filled with zeros. At most 32 characters supported.

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer`		value to be converted
`idigits`	`integer`		number of decimals

7.3.15 Function sys_sli

Interface:

```
sys_sli(ivalue, idigits)
```

Description:
This routine converts a long integer value to a string of length idigits.

Result:
character (len=32) : String representation of the value, filled with white spaces. At most 32 characters supported.

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer(I64)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.16 Function sys_sli0

Interface:

```
sys_sli0(ivalue, idigits)
```

Description:
This routine converts a long integer value to a string of length idigits.

Result:

character (len=32) : String representation of the value, filled with zeros. At most 32 characters supported.

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer(I64)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.17 Function sys_sdL

Interface:
`sys_sdL(dvalue, idigits)`

Description:
This routine converts a double value to a string with idigits decimal places.

Result:
character (len=32) : String representation of the value (left-aligned), fixed length of 32 characters

Input variables:

Name	Type	Rank	Description
`dvalue`	`real(DP)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.18 Function sys_sdEL

Interface:
`sys_sdEL(dvalue, idigits)`

Description:
This routine converts a double value to a string with idigits decimal places in scientific notation.

Result:
character (len=32) : String representation of the value (left-aligned), fixed length of 32 characters

Input variables:

Name	Type	Rank	Description
`dvalue`	`real(DP)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.19 Function sys_siL

Interface:
`sys_siL(ivalue, idigits)`

Description:
This routine converts an integer value to a string of length idigits, filled up with white spaces.

Result:
character (len=32) : String representation of the value (left-aligned), fixed length of 32 characters

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer`		value to be converted
`idigits`	`integer`		number of decimals

7.3.20 Function sys_si0L

Interface:
`sys_si0L(ivalue, idigits)`

Description:
This routine converts an integer value to a string of length idigits, filled up with zeros.

Result:
character (len=32) : String representation of the value (left-aligned), fixed length of 32 characters

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer`		value to be converted
`idigits`	`integer`		number of decimals

7.3.21 Function sys_sliL

Interface:
`sys_sliL(ivalue, idigits)`

Description:
This routine converts a long integer value to a string of length idigits.

Result:
character (len=32) : String representation of the value (left-aligned), fixed length of 32 characters

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer(I64)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.22 Function sys_sli0L

Interface:

`sys_sli0L(ivalue, idigits)`

Description:
This routine converts a long integer value to a string of length idigits.

Result:

character (len=32) : String representation of the value (left-aligned), fixed length of 32 characters

Input variables:

Name	Type	Rank	Description
`ivalue`	`integer(I64)`		value to be converted
`idigits`	`integer`		number of decimals

7.3.23 Function sys_inumberOfDigits

Interface:

`sys_inumberOfDigits(inumber)`

Description:
This function determines the number of digits a given number has.

Input variables:

Name	Type	Rank	Description
`inumber`	`integer`		name of the master file

Output variables:

Name	Type	Rank	Description
`sys_inumberOfDigits`	`integer`		the number of digits of the given number

7.3.24 Subroutine sys_getMasterFile

Interface:

`sys_getMasterFile(smaster)`

Description:
This subroutine returns the name of the master .dat file. The default dat-file is "master.dat". If a different file shall be used as dat-file, its name has to be appended to the program call:
Example: mpirun -np x program-name [dat-file]
x - number of processors to be used
dat-file - name of the dat-file to be used instead of master.dat

Output variables:

Name	Type	Rank	Description
`smaster`	`c (*)`		name of the master file

7.3.25 Subroutine sys_tokeniser

Interface:

`sys_tokeniser(sbuffer, nmaxtokcount, stokens, itokcount)`

Description:
This routine splits a string into substrings, divided by blank, comma or equation sign.

Input variables:

Name	Type	Rank	Description
`sbuffer`	`c(*)`		string to be scanned
`nmaxtokcount`	`integer`		maximal number of substrings

Output variables:

Name	Type	Rank	Description
`stokens`	`c(*)`	:	array of substrings
`itokCount`	`integer`		number of tokens

7.3.26 Function sys_getenv_int

Interface:

`sys_getenv_int(svar,ivalue)`

Description:
This functions returns the integer value of a given enviroment variable. The routine returns .TRUE., if the variable exists, otherwise .FALSE. .

Result:
logical : exit status

Input variables:

Name	Type	Rank	Description
`svar`	`c(*)`		name of the enviroment variable

Output variables:

Name	Type	Rank	Description
`ivalue`	`integer`		value of the enviroment variable

7.3.27 Function sys_getenv_real

Interface:

`sys_getenv_real(svar,dvalue)`

Description:
This functions returns the real value of a given enviroment variable. The routine returns .TRUE., if the variable exists, otherwise .FALSE. .

Result:

logical : exit status

Input variables:

Name	Type	Rank	Description
`svar`	`c(*)`		name of the enviroment variable

Output variables:

Name	Type	Rank	Description
`dvalue`	`real(DP)`		

7.3.28 Function sys_getenv_string

Interface:

`sys_getenv_string(svar,sresult)`

Description:
This functions returns the string value of a given enviroment variable. The routine returns .TRUE., if the variable exists, otherwise .FALSE. .

Result:

logical : exit status

Input variables:

Name	Type	Rank	Description
`svar`	`c(*)`		name of the enviroment variable

Output variables:

Name	Type	Rank	Description
`sresult`	`c(*)`		

7.3.29 Function sys_readpar_int

Interface:

`sys_readpar_int(svalue)`

Description:
This routine checks, if the given string starts with an $-character and returns in this case the integer value of the enviroment variable. Otherwise it returns the integer value of the string itself.

Result:

integer : integer value of the string or environment variable

Input variables:

Name	Type	Rank	Description
`svalue`	`c(*)`		string with value or variable name

7.3.30 Function sys_readpar_real

Interface:

`sys_readpar_real(svalue)`

Description:
This routine checks, if the given string starts with an $-character and returns in this case the integer value of the enviroment variable. Otherwise it returns the real value of the string itself.

Result:

real(DP) : integer value of the string or environment variable

Input variables:

Name	Type	Rank	Description
`svalue`	`c(*)`		string with value or variable name

7.3.31 Function sys_readpar_string

Interface:

`sys_readpar_string(svalue)`

Description:
This routine checks, if the given string starts with an $-character and returns in this case the string value of the enviroment variable. Otherwise it returns the string value of the string itself.

Result:

character(len=256) : integer value of the string or environment variable

Input variables:

Name	Type	Rank	Description
`svalue`	`c(*)`		string with value or variable name

7.3.32 Function sys_upcase

Interface:

```
sys_upcase(sinput)
```

Description:
This routine converts a given string to its uppercase version.

Input variables:

Name	Type	Rank	Description
`sinput`	`c(*)`		input string

Output variables:

Name	Type	Rank	Description
`soutput`	`c(len(sinput))`		output string

7.3.33 Function sys_stringToInt

Interface:

```
sys_stringToInt(svalue)
```

Description:
This routine tries to convert a string to an integer value. If the conversion fails, the return value is set to SYS_MAXINT.

Input variables:

Name	Type	Rank	Description
`svalue`	`c(*)`		string to be converted

Result:
integer : resulting value (= SYS_MAXINT if conversion fails)

7.3.34 Function sys_stringToReal

Interface:

```
sys_stringToReal(svalue)
```

Description:
This routine tries to convert a string to a real value. If the conversion fails, the return value is set to SYS_MAXREAL.

Input variables:

Name	Type	Rank	Description
`svalue`	`c(*)`		string to be converted

Result:
real(DP) : resulting value (= SYS_MAXREAL if conversion fails)

7.3.35 Subroutine sys_getNextEntry

Interface:
`sys_getNextEntry(iunit, sbuffer)`

Description:
This routine reads an item from the file connected to unit iunit and writes it into the buffer sbuffer. Items are assumed to be separated by spaces, comma, tabs. Anything from hash character # aka Lattenkreuz till the EOL is ignored (assuming fortrans eor=EOL). The escape symbol is the backslash.
Author: Jaroslav

Input variables:

Name	Type	Rank	Description
`iunit`	`integer`		unit connected to the file to read from

Output variables:

Name	Type	Rank	Description
`sbuffer`	`c(*)`		output string

7.3.36 Function sys_getFreeUnit

Interface:
`sys_getFreeUnit()`

Description:
This routine tries to find a free unit (for file input/output). If a free unit is found, it is returned, otherwise -1 is returned.

Result:
integer : number of free unit (-1 if no free unit available)

7.3.37 Function sys_fileExists

Interface:
`sys_fileExists(iunit,sname)`

Description:
This function checks if there is a file connected to unit iunit, which we can access for reading.

Input variables:

Name	Type	Rank	Description
`iunit`	`integer`		unit the file shall be attached to
`sname`	`c (*)`		name of the file to look at

Result:
logical : .TRUE. if the file is accessable for reading, .FALSE. otherwise

7.3.38 Subroutine sys_flush

Interface:
`sys_flush(iunit)`

Description:
This routine flushes the buffers associated with an open output unit. This normally happens when the file is closed or the program ends, but this routine ensures the buffers are flushed before any other processing occurs.

Input variables:

Name	Type	Rank	Description
`iunit`	`integer`		unit connected to the file to write to

7.3.39 Subroutine sys_int_sort

Interface:
`sys_int_sort(Iarray, csortMethod, Imapping)`

Description:
Sorting routine for integer data type. If more than one vector must be sorted, or if several vectors may be sorted the same way, the mapping array Imapping can be computed. This array must be created outside the routine and must have the same length as Iarray. Then, the other vectors are computed by Vec(Imapping(i)). The array Imapping is an optional argument. The sorting method used is told by the parameter csortMethod. If this optional argument is not used, the routine performs heapsort.

Input variables:

Name	Type	Rank	Description
`csortMethod`	integer		sort algorithm: SORT_HEAP, SORT_QUICK, SORT_INSERT

Input/Output variables:

Name	Type	Rank	Description
`Iarray`	`integer`		integer array to be sorted
`Imapping`	`integer`		optional mapping vector (if more than 1 vector may be sorted)

7.3.40 Subroutine sys_i32_sort

Interface:

`sys_i32_sort(iarray, csortMethod, Imapping)`

Description:
Sorting routine for i32 data type. If more than one vector must be sorted, or if several vectors may be sorted the same way, the mapping array Imapping can be computed. This array must be created outside the routine and must have the same length as Iarray. Then, the other vectors are computed by Vec(Imapping(i)). The array Imapping is an optional argument. The sorting method used is told by the parameter csortMethod. If this optional argument is not used, the routine performs heapsort.

Input variables:

Name	Type	Rank	Description
`csortMethod`	integer		sort algorithm: SORT_HEAP, SORT_QUICK, SORT_INSERT

Input/Output variables:

Name	Type	Rank	Description
`Iarray`	`integer(i32)`		integer array to be sorted
`Imapping`	`integer`		optional mapping vector (if more than 1 vector may be sorted)

7.3.41 Subroutine sys_i64_sort

Interface:

`sys_i64_sort(Iarray, csortMethod)`

Description:
sort routine for integer64 type

Input variables:

Name	Type	Rank	Description
`csortMethod`	integer		sort algorithm: SORT_HEAP, SORT_QUICK, SORT_INSERT

Input/Output variables:

Name	Type	Rank	Description
`Iarray`	`integer(i64)`		integer array to be sorted

7.3.42 Subroutine sys_sp_sort

Interface:

`sys_sp_sort(Darray, csortMethod)`

Description:
sort routine for single precision

Input variables:

Name	Type	Rank	Description
`csortMethod`	integer		sort algorithm: SORT_HEAP, SORT_QUICK, SORT_INSERT

Input/Output variables:

Name	Type	Rank	Description
`Darray`	`real(sp)`		singe precision array to be sorted

7.3.43 Subroutine sys_dp_sort

Interface:
`sys_dp_sort(Darray, csortMethod, Imapping)`

Description:
Sorting routine for double precision arrays. If more than one vector must be sorted, or if several vectors may be sorted the same way, the mapping array Imapping can be computed. This array must be created outside the routine and must have the same length as Iarray. Then, the other vectors are computed by Vec(Imapping(i)). The array Imapping is an optional argument. The sorting method used is told by the parameter csortMethod. If this optional argument is not used, the routine performs heapsort.

Input variables:

Name	Type	Rank	Description
`csortMethod`	integer		sort algorithm: SORT_HEAP, SORT_QUICK, SORT_INSERT

Input/Output variables:

Name	Type	Rank	Description
`Darray`	`real(dp)`		double precision array to be sorted
`Imapping`	`integer`		optional mapping vector (if more than 1 vector may be sorted)

7.3.44 Function sys_triArea

Interface:
`sys_triArea(dx1,dy1,dx2,dy2,dx3,dy3)`

Description:

Compute the oriented area of the triangle spanned by (dxi, dyi).

Result:
real(DP) : area of the triangle

Input variables:

Name	Type	Rank	Description
`dx1,dy1,dx2,dy2,dx3,dy3`	`real(DP)`		coordinates of the three points

7.3.45 Function sys_quadArea

Interface:
`sys_quadArea(dx1,dy1,dx2,dy2,dx3,dy3,dx4,dy4)`

Description:
Compute the oriented area of the quadrilateral spanned by (dxi, dyi).

Result:
real(DP) : area of the quadrilateral

Input variables:

Name	Type	Rank	Description
`dx1,dy1,dx2,dy2,dx3,dy3,dx4,dy4`	`real(DP)`		coordinates of the four points

7.3.46 Subroutine sys_parInElement

Interface:
`sys_parInElement(DcoordX, DcoordY, dxpar, dypar, dxreal, dyreal)`

Description:
This subroutine is to find the parameter values for a given point (x,y) in real coordinates.
Remark: This is a difficult task, as usually in FEM codes the parameter values are known and one wants to obtain the real coordinates. inverting the bilinear trafo in a straightforward manner by using pq-formula does not work very well, as it is numerically unstable. For parallelogram-shaped elements, one would have to introduce a special treatment. For nearly parallelogram-shaped elements, this can cause a crash as the argument of the square root can become negative due to rounding errors. In the case of points near the element borders, we divide nearly 0/0.
Therefore, we have implemented the algorithm described in Introduction to Fineite Element Methods, Carlos Felippa, Department of Aerospace Engineering Sciences and Center for Aerospace Structures, http://titan.colorado.edu/courses.d/IFEM.d/ .

Input variables:

Name	Type	Rank	Description
`dxreal, dyreal`	`real(DP)`		coordinates of the evaluation point
`DcoordX`	`real(DP)`	4	coordinates of the element vertices
`DcoordY`	`real(DP)`	4	

Output variables:

Name	Type	Rank	Description
`dxpar,dypar`	`real(DP)`		parameter values of (x,y)

7.3.47 Function ipointInElement

Interface:
`ipointInElement(DcoordX, DcoordY, dx,dy)`

Description:
This function is to detect wether a given point (dx, dy) is inside the macro given by the vertices coordinates in Dcoord. We devide the macro in 4 triangles. The point P lies in the quadrilateral ABCD, if the oriented areas of ABP,BCP,CDP,DAP all are lower than 0. The area is computed in the function area. This requires the counterclockwise numeration of the quad.

Input variables:

Name	Type	Rank	Description
`DcoordX`	`real(DP)`	4	coordinates of vertices of element
`DcoordY`	`real(DP)`	4	
`dx, dy`	`real(DP)`		coordinates of test point

Result:
integer : -i, if point is vertex i
0, if point is outside the macro
1, if point is in macro or on a macro edge or is a vertex
2, if point is in the interior of the macro

7.4 Module io

Purpose: This module contains several routines for input/output purposes.

Constant definitions:

Name	Type	Purpose
Purpose: device number to use for I/O		
`IO_CHANNEL1`	`integer`	device number for I/O
Purpose: flags for output format		
`IO_BINARY`	`integer`	output format binary
`IO_ASCII`	`integer`	output format ascii

7.4.1 Subroutine io_readSolution

Interface:
`io_readSolution(rparBlock, sname, h_Dsol, imglevel, cform)`

Description:
This routine reads a solution from a distributed file structure.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block structure
`imglevel`	`integer`		multi grid level of the solution
`sname`	`c(*)`		file name
`cform`	integer		binary or ascii format of the solution, IO_BINARY: binary format, IO_ASCII: ascii format
`h_Dsol`	`integer`		handle to solution vector

7.4.2 Subroutine io_writeSolution

Interface:
`io_writeSolution(rparBlock, sname, h_Dsol, imglevel, cform)`

Description:
This routine writes a solution to a distributed file structure.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block structure
`h_Dsol`	`integer`		solution variable
`sname`	`c(*)`		file name
`imglevel`	`integer`		multi grid level of the solution
`cform`	integer		binary or ascii format of the solution, IO_BINARY: binary format, IO_ASCII: ascii format

7.4.3 Subroutine io_openFileForReading

Interface:
`io_openFileForReading(sfilename, iunit)`

Description:

This routine tries to open a file for reading. If succesful, on can read from it via unit "iunit". Otherwise, iunit is -1.

Input variables:

Name	Type	Rank	Description
`sfilename`	`c(*)`		filename

Output variables:

Name	Type	Rank	Description
`iunit`	`integer`		number of unit

7.4.4 Subroutine io_openFileForWriting

Interface:

`io_openFileForWriting(sfilename, iunit, cflag, bfileExists)`

Description:
This routine tries to open a file for writing. If succesful, one can write to it via unit "iunit". Otherwise, iunit is -1. cflag specifies if an already existing file should be replaced or if the output should be appended. bfileExists is an optional parameter, which will be set to true, if the file already existed, otherwise false.

Input variables:

Name	Type	Rank	Description
`sfilename`	`c(*)`		filename
`cflag`	integer		mode: SYS_APPEND or SYS_REPLACE

Output variables:

Name	Type	Rank	Description
`iunit`	`integer`		unit of the opened file
`bfileExists`	`logical`		optional parameter (see description)

7.4.5 Subroutine io_writeMatrix

Interface:

`io_writeMatrix(rparBlock, rdiscrId, sname, idigits)`

Description:
This routine writes the matrices of rdiscrId corresponding to the macros of the parallelblock into files 'sname_pb(p)_mac(m).dat' (with (p) = number of parallelblock and (m) = number of macro). The values

are exported with idigits precision. Existing files are replaced. The output can be viewed and treated, e.g., by Matlab. Be aware, that full (not sparse) matrices are written! The files become large very easily.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock
`rdiscrId`	`type(Tdiscrid)`		matrix identifier
`sname`	`c(*)`		string defining the (beginning of the) filename
`idigits`	`integer`		number of digits of the matrix entries in the file

7.4.6 Subroutine io_printVector

Interface:

`io_printVector(rparBlock, h_Dvec, imgLevel, sname)`

Description:
This routine simply prints a vector to the screen.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock
`h_Dvec`	`integer`		handle of the vector
`imgLevel`	`integer`		MG level (MB_ALL_LEV for all levels)
`sname`	`c(*)`		name to be printed

7.4.7 Subroutine io_writeBlockMatrix

Interface:

`io_writeBlockMatrix(rparBlock, rdiscrId, sname, idigits, bseparateMatrices)`

Description:
This routine writes the block matrices of rdiscrId corresponding to the macros of the parallelblock into files 'sname_pb(p)_mac(m).dat' (with (p) = number of parallelblock and (m) = number of macro). The values are exported with idigits precision. The value of bseparateMatrices controls whether the block matrices are stored as one big matrix (.FALSE.) or each matrix block in a separate file. Existing files are replaced. The output can be viewed and treated, e.g., by Matlab. Be aware, that full (not sparse) matrices are written! The files become large very easily.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock
rdiscrId	type(Tdiscrid)	:, :	matrix identifier
sname	c(*)		string defining the (beginning of the) filename
idigits	integer		number of digits of the matrix entries in the file
bseparateMatrices	logical		flag whether block matrices are exported as one big matrix (.FALSE.) or each matrix block in a separate file.

7.4.8 Subroutine io_printBlockVector

Interface:
io_printBlockVector(rparBlock, H_Dvec, imgLevel, sname)

Description:
This routine simply prints a block vector (i.e. the vectors belonging to the handle array H_Dvec) to the screen.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock
H_Dvec	integer	:	handle array of the block vector
imgLevel	integer		MG level (MB_ALL_LEV for all levels)
sname	c(*)		name to be printed

7.5 Module output

Purpose: Handling the output of messages to terminals and logfiles

Constant definitions:

Name	Type	Purpose
OU_LOG	integer	device number for logfile
OU_CHANNEL1	integer	device number for I/O
OU_CHANNEL2	integer	device number for I/O
OU_CHANNEL3	integer	device number for I/O
OU_CHANNEL_FEAST	integer	
OU_CHANNEL_FGEO	integer	

OU_CHANNEL_FFGEO	integer	
OU_CHANNEL_FMESH	integer	
OU_CHANNEL_FPART	integer	
OU_CHANNEL_FBC	integer	
OU_CHANNEL_RESULT	integer	device number for result files
OM_LOG	integer	output_cmode, trace info in log file
OM_TERM	integer	output_cmode, trace info on terminal
OM_BOTH	integer	output_cmode, trace info in file/term
OM_NONE	integer	output_cmode, no trace information
OL_NONE	integer	output_clevel, no trace information
OL_ERROR	integer	output_clevel, only output of error messages
OL_MSG	integer	output_clevel, only system messages
OL_TIMER	integer	output_clevel, only system/timer messages
OL_TRACE1	integer	output_clevel, trace info level 1
OL_TRACE2	integer	output_clevel, trace info level 2
OL_TRACE3	integer	output_clevel, trace info level 3

Global variable definitions:

Name	Type	Rank	Purpose
output_clevel	integer		level of tracing information
output_numberResultFile	integer		number of actual result file

7.5.1 Subroutine output_openResultFile

Interface:

```
output_openResultFile()
```

Description:
This subroutine opens the result file for global result data. Called on the master process the name of the result file is masterresult.X, on the slave process slaveresult.X, X gives the number of calls of this routine to generate a sequence of result files

7.5.2 Subroutine output_closeResultFile

Interface:

```
output_closeResultFile()
```

Description:
This routine closes the result file.

7.5.3 Subroutine output_writeIntResult

Interface:
`output_writeIntResult(stag, ivalue)`

Description:
This routine writes an integer formatted result item to the result file

Input variables:

Name	Type	Rank	Description
`stag`	`c(*)`		tag name of the result data
`ivalue`	`integer`		result data

7.5.4 Subroutine output_writeDoubleResult

Interface:
`output_writeDoubleResult(stag, dvalue)`

Description:
This routine writes an double formatted result item to the result file

Input variables:

Name	Type	Rank	Description
`stag`	`c(*)`		tag name of the result data
`dvalue`	`real(DP)`		result data

7.5.5 Subroutine output_writeFloatResult

Interface:
`output_writeFloatResult(stag, dvalue)`

Description:
This routine writes an single formatted result item to the result file

Input variables:

Name	Type	Rank	Description
`stag`	`c(*)`		tag name of the result data
`dvalue`	`real(DP)`		result data

7.5.6 Subroutine output_writeStringResult

Interface:
`output_writeStringResult(sstring)`

Description:
This routine writes a string to the result file

Input variables:

Name	Type	Rank	Description
`sstring`	`c(*)`		result data

7.5.7 Function output_do

Interface:
`output_do(cpriorityLevel)`

Description:
This function returns the do state of a pending output operation

Input variables:

Name	Type	Rank	Description
`cpriorityLevel`	integer		priority level, OL_

7.5.8 Subroutine output_line

Interface:
`output_line(cpriorityLevel, sroutine, smessage)`

Description:
This routine writes a message with priority cpriorityLevel to log device

Input variables:

Name	Type	Rank	Description
`cpriorityLevel`	integer		priority level, OL_
`sroutine`	`c (*)`		name of calling routine
`smessage`	`c (*)`		message which shall be printed

7.5.9 Subroutine output_init

Interface:
`output_init(shome, ipbIdx, cpriorityLevel)`

Description:
This subroutine initializes the log files.

Input variables:

Name	Type	Rank	Description
ipbIdx	integer		index of calling parallel block
cpriorityLevel	integer		priority level
shome	c (*)		path to home directory

7.5.10 Subroutine output_close

Interface:
output_close()

Description:
This routine closes the log device.

7.6 Module storage

Purpose: This module contains the memory management of the FEAST package. For performance reasons, we implemented our own memory management for the data vectors of FEAST and decided not to use the one provided by Fortran. When starting the program, every process calls a initialisation routine which reserves a heap of adjustable size for every data type. The routine storage_new reserves a vector. This routine returns a vector handle. The routine storage_getbase creates a pointer to this handle, which can be used like a normal array. (DOCU IMCOMPLETE)

Constant definitions:

Name	Type	Purpose
ST_NOHANDLE	integer	defines an non-allocated storage block handle
ST_SINGLE	integer	storage block contains single floats
ST_DOUBLE	integer	storage block contains double floats
ST_INT	integer	storage block contains ints
ST_NEWBLOCK_ZERO	integer	init new storage block with zeros
ST_NEWBLOCK_NOINIT	integer	no init new storage block

Type definitions:

Type name	component name	Type	Rank	Purpose
Tarray	record with array of integer pointers			
	a	integer	:	access array

`Tdarray`	record with array of double pointers			
	`a`	`real(DP)`	:	access array

7.6.1 Subroutine storage_initarray

Interface:
`storage_initarray(ils, iar)`

Description:
This routine sets the pointers of the array to the corresponding memory handles.

Input variables:

Name	Type	Rank	Description
`ils`	`integer`		memory handles

Output variables:

Name	Type	Rank	Description
`iar`	`Tarray`	dim(:, :)	array of integer pointer

7.6.2 Subroutine storage_initarrayEntry

Interface:
`storage_initarrayEntry(i, j, ils, IarrayHandles)`

Description:
This routine sets the pointers of the array to the corresponding memory handles of the specific array/edge set. To do so, ils is the handle of the handle array IarrayHandles, wich contains the handles for the different arrays in this structure.

Input variables:

Name	Type	Rank	Description
`i`	`integer`		array index
`j`	`integer`		edge index
`ils`	`integer`		memory handle

Output variables:

Name	Type	Rank	Description
`IarrayHandles`	`integer`	:	integer pointer

7.6.3 Function storage_initarrayOneEntry

Interface:

`storage_initarrayOneEntry(i,j,ils,il)`

Description:
This routine sets the pointers of the array to the corresponding memory handles of the specific array/edge set

Input variables:

Name	Type	Rank	Description
`i`	`integer`		array index
`j`	`integer`		edge index
`ils`	`integer`		memory handles
`il`	`integer`		offset

7.6.4 Subroutine storage_initarrayline

Interface:

`storage_initarrayline(idy, ils, iar)`

Description:
This routine sets the pointers of the array to the corresponding memory handles on line idy (second dimension).

Input variables:

Name	Type	Rank	Description
`idy`	`integer`		index of second dimension
`ils`	`integer`		array of memory handles

Output variables:

Name	Type	Rank	Description
`iar`	`Tarray`	dim(:)	array of integer pointer

7.6.5 Subroutine storage_initdarray

Interface:

`storage_initdarray(nx, ny, ils, dar)`

Description:
This routine sets the pointers of the array to the corresponding memory handles.

Input variables:

Name	Type	Rank	Description
nx	integer		array dimension 1
ny	integer		array dimension 2
ils	integer	:, :	array of memory handles

Output variables:

Name	Type	Rank	Description
dar	Tdarray	dim(:, :)	array of double pointer

7.6.6 Subroutine storage_initdarrayidx

Interface:
storage_initdarrayidx(Iheader, Ddata, idx, dar)

Description:
This function sets the pointers of the array to the corresponding memory handles on line idx.

Input variables:

Name	Type	Rank	Description
idx	integer		index of dimension 1
Iheader	integer	:	
Ddata	real(DP)	:	

Output variables:

Name	Type	Rank	Description
dar	Tdarray	dim(:, :)	array of double pointer

7.6.7 Subroutine storage_cleardarrayidx

Interface:
storage_cleardarrayidx(idx, ils)

Description:
This function sets the pointers of the array to the corresponding memory handles on line idx.

Input variables:

Name	Type	Rank	Description
idx	integer		index of dimension 1
ils	integer	2	array of memory handles

7.6.8 Subroutine storage_initdarray2

Interface:
`storage_initdarray2(nx, ils, dar)`

Description:
This function sets the pointers of the array to the corresponding memory handles.

Input variables:

Name	Type	Rank	Description
`nx`	`integer`		dimension
`ils`	`integer`	:	array of memory handles

Output variables:

Name	Type	Rank	Description
`dar`	`Tdarray`	dim(:)	array of double pointer

7.6.9 Subroutine storage_initdarray2idx

Interface:
`storage_initdarray2idx(idx, nx, ils, dar)`

Description:
This routine sets the pointers of the array to the correspondig memory handles on line idx (first dimension).

Input variables:

Name	Type	Rank	Description
`nx`	`integer`		array dimension 2
`idx`	`integer`		index of first dimension
`ils`	`integer`	:, :	array of memory handles

Output variables:

Name	Type	Rank	Description
`dar`	`Tdarray`	dim(:)	array of integer pointer

7.6.10 Subroutine storage_link

Interface:
`storage_link(h_sourceDescriptor, h_destDescriptor)`

Description:
This subroutine sets a link from one handle to another handle

Input variables:

Name	Type	Rank	Description
`h_sourceDescriptor`	`integer`		source handle

Output variables:

Name	Type	Rank	Description
`h_destDescriptor`	`integer`		destination handle

7.6.11 Function storage_getheaplen

Interface:
`storage_getheaplen(ityp)`

Description:
This function returns the length of the specified heap

Input variables:

Name	Type	Rank	Description
`ityp`	`integer`		heap type (ST_INT,ST_DOUBLE)

Result:
integer : heap length

7.6.12 Subroutine storage_init

Interface:
`storage_init(idesk, iheaplen, dratio)`

Description:
This routine initializes the storage management and allocates memory. If not enough memory is available, the program terminates immediately.

Input variables:

Name	Type	Rank	Description
`idesk`	`integer`		count of descriptor entries
`iheaplen`	`integer`		DOUBLE heaplen
`dratio`	`real(DP)`		INT heaplen = dratio times heaplen

7.6.13 Subroutine storage_setStoragePar

Interface:

`storage_setStoragePar(iblk, ides, isingleheap, idoubleheap, iintheap)`

Description:
This routine sets the default allocation sizes for the heaps and for the descriptors. This routine has to be called before storage_init.

Input variables:

Name	Type	Rank	Description
`iblk`	`integer`		allocation size for the allocation block table
`ides`	`integer`		allocation size for the descriptor table
`isingleheap`	`integer`		allocation size for the single float heap
`idoubleheap`	`integer`		allocation size for the double float heap
`iintheap`	`integer`		allocation size for the integer heap

7.6.14 Subroutine storage_freeSpaceOnTop

Interface:

`storage_freeSpaceOnTop(ctype)`

Description:
This routine prints the free storage areas on top of every heap

Input variables:

Name	Type	Rank	Description
`ctype`	integer		type of the heap

7.6.15 Subroutine storage_checkConsistency

Interface:

`storage_checkConsistency(ctype)`

Description:
This routine checks consistency of the storage structures (only for debugging purposes)

Input variables:

Name	Type	Rank	Description
`ctype`	integer		type of the heap

7.6.16 Subroutine storage_compress

Interface:

`storage_compress(ctype)`

Description:
This routine compresses a complete heap

Input variables:

Name	Type	Rank	Description
`ctype`	integer		type of the heap

7.6.17 Subroutine storage_printDescriptor

Interface:

`storage_printDescriptor(idescr, sprefix)`

Description:
This routine prints the given descriptor

Input variables:

Name	Type	Rank	Description
`idescr`	`integer`		storage descriptor
`sprefix`	`c(*)`		

7.6.18 Subroutine storage_printStorageContent

Interface:

`storage_printStorageContent(idescr, sprefix)`

Description:
This routine prints the current storage content belonging to idescr

Input variables:

Name	Type	Rank	Description
`idescr`	`integer`		storage descriptor
`sprefix`	`c(*)`		

7.6.19 Subroutine storage_new

Interface:

`storage_new(scall, sname, isize, ctype, idescr, cinitNewBlock)`

Description:

This routine reserves a memory block of desired size and type.

Input variables:

Name	Type	Rank	Description
`isize`	`integer`		requested storage size
`ctype`	integer		data type
`sname`	`c(*)`		clear name of data field
`scall`	`c(*)`		name of the calling routine
`cinitNewBlock`	integer		init new storage block (ST_NEWBLOCK_ZERO,ST_NEWBLOCK_NOINIT)

Output variables:

Name	Type	Rank	Description
`idescr`	`integer`		memory descriptor

7.6.20 Subroutine storage_yield

Interface:

```
storage_yield()
```

Description:
This routine initiates the storage compression if neccessary.

7.6.21 Subroutine storage_free

Interface:

```
storage_free(idescr)
```

Description:
This routine deletes a memory block with a given descriptor.

Input/Output variables:

Name	Type	Rank	Description
`idescr`	`integer`		memory descriptor

7.6.22 Function storage_size

Interface:

```
storage_size(ides)
```

Description:
This function returns the length of the specified memory block.

Result:
integer : length of the memory block

Input variables:

Name	Type	Rank	Description
`ides`	`integer`		memory handle

7.6.23 Subroutine storage_resize

Interface:
`storage_resize(ides, incount, sCall)`

Description:
This routine shrinks a memory block to a new length.

Input variables:

Name	Type	Rank	Description
`ides`	`integer`		memory descriptor
`incount`	`integer`		new length of the memory block
`sCall`	`c(*)`		

7.6.24 Subroutine storage_clear

Interface:
`storage_clear(ides)`

Description:
This routine clears a memory block with a given descriptor.

Input variables:

Name	Type	Rank	Description
`ides`	`integer`		memory descriptor

7.6.25 Subroutine storage_info

Interface:
`storage_info()`

Description:
This routine gives a summary about the heap allocation.

7.6.26 Subroutine storage_getbase_single

Interface:

`storage_getbase_single(idescr, fdruf)`

Description:
This subroutine returns the pointer associated with a memory block. Alternatively the call storage_getbase_double can be used.

Input variables:

Name	Type	Rank	Description
`idescr`	`integer`		storage block handle

Output variables:

Name	Type	Rank	Description
`fdruf`	`real(SP)`	:	pointer to storage block

7.6.27 Subroutine storage_getbase_double

Interface:

`storage_getbase_double(idescr, ddruf)`

Description:
This subroutine returns the pointer associated with a memory block. Alternatively the call storage_getbase_double can be used.

Input variables:

Name	Type	Rank	Description
`idescr`	`integer`		storage block handle

Output variables:

Name	Type	Rank	Description
`ddruf`	`real(DP)`	:	pointer to storage block

7.6.28 Subroutine storage_getbase_int

Interface:

`storage_getbase_int(idescr, idruf)`

Description:
This subroutine returns the pointer associated with a memory block. Alternatively the call storage_getbase_double can be used.

Input variables:

Name	Type	Rank	Description
idescr	integer		storage block handle

Output variables:

Name	Type	Rank	Description
idruf	integer	:	pointer to storage block

7.7 Module sbblas

Purpose: Driver routines for SBBLAS package

Constant definitions:

Name	Type		Purpose
Purpose: SBBLAS routines IDs			
SBBLAS_ID_MV_V	integer		
SBBLAS_ID_MV_C	integer		
SBBLAS_ID_TRIGS_C	integer		
SBBLAS_ID_TRIGS_V	integer		
SBBLAS_ID_TRIS_V	integer		
SBBLAS_ID_TRIS_C	integer		
SBBLAS_ID_TMV_V	integer		
SBBLAS_ID_TMV_C	integer		
Purpose: SBBLAS configuration parameter			
SBBLAS_NLEVEL	integer		number of levels
SBBLAS_NALGS	integer		number of algorithms
SBBLAS_IDS	integer	SBBLAS_NALGS	array of IDs
SBBLAS_KEYS	c(8)	SBBLAS_NALGS	array of string keys
SBBLAS_CONST	logical	SBBLAS_NALGS	constant/variable version of the routines
SBBLAS_MAX_MV	integer		upper boundary for SBBLAS*-parameters in master.dat
SBBLAS_MAX_MVTRI	integer		
SBBLAS_MAX_PRSLOWL	integer		

SBBLAS_MAX_PRSLINE	integer	
SBBLAS_DEFAULT_MV	integer	default SBBLAS*-parameters used in error case
SBBLAS_DEFAULT_MVTRI	integer	
SBBLAS_DEFAULT_PRSLOWL	integer	
SBBLAS_DEFAULT_PRSLINE	integer	

Global variable definitions:

Name	Type	Rank	Purpose
sbblas_Iconf	integer	SBBLAS_NLEVEL*SBBLAS_NALGS*2	SBBLAS configuration array

7.7.1 Subroutine sbblas_init

Interface:

`sbblas_init(sconffile, sys_sbblasmv, sys_sbblasprslowl, sys_sbblasmvtri, sys_sbblasprsline)`

Description:
This routine reads the given sbblas.conf file and initializes the configuration variable, if the build version is found in the configuration file. Otherwise, a default selection is made.

Input variables:

Name	Type	Rank	Description
sconffile	c(*)		file name of the configuration file
sys_sbblasmv	integer		SBBLAS routine ID
sys_sbblasprslowl	integer		SBBLAS routine ID
sys_sbblasmvtri	integer		SBBLAS routine ID
sys_sbblasprsline	integer		SBBLAS routine ID

Global variables:

7.7.2 Subroutine sbblas_override

Interface:

`sbblas_override(sys_sbblasmv, sys_sbblasprslowl, sys_sbblasmvtri, sys_sbblasprsline)`

Description:
This routine sets the SBBLAS configuration variable to the given default values. This routine is mainly intented for debugging purposes.

Input variables:

Name	Type	Rank	Description
sys_sbblasmv	integer		SBBLAS routine ID
sys_sbblasprslowl	integer		SBBLAS routine ID
sys_sbblasmvtri	integer		SBBLAS routine ID
sys_sbblasprsline	integer		SBBLAS routine ID

Global variables:

7.7.3 Subroutine sbblas_setinfo

Interface:

`sbblas_setinfo(id, n, iimpl, iwindow)`

Description:
This routine sets to a given SBBLAS algorithm ID and vector length the given implementation version and window size.

Input variables:

Name	Type	Rank	Description
id	integer		SBBLAS_ID
n	integer		vector length
iimpl	integer		selected implementation of the algorithm
iwindow	integer		selected window size

7.7.4 Subroutine sbblas_getinfo

Interface:

`sbblas_getinfo(id, n, iimpl, iwindow)`

Description:
This routine returns to a given SBBLAS algorithm ID and vector length the suitable implementation version and window size.

Input variables:

Name	Type	Rank	Description
id	integer		SBBLAS_ID
n	integer		vector length

Output variables:

Name	Type	Rank	Description
iimpl	integer		selected implementation of the algorithm
iwindow	integer		selected window size

7.7.5 Subroutine sbblas_getnimpl

Interface:
`sbblas_getnimpl(id, nimpl)`

Description:
This routine returns to a given SBBLAS algorithm ID the number of available implementations

Input variables:

Name	Type	Rank	Description
`id`	`integer`		SBBLAS_ID

Output variables:

Name	Type	Rank	Description
`nimpl`	`integer`		number of available implementations

7.7.6 Subroutine sbblas_getnwindows

Interface:
`sbblas_getnwindows(id, n, Iwindow, nwindows)`

Description:
This routine returns to a given SBBLAS algorithm ID and vector length the all available window sizes.

Input variables:

Name	Type	Rank	Description
`id`	`integer`		SBBLAS_ID
`n`	`integer`		vector length

Output variables:

Name	Type	Rank	Description
`nwindows`	`integer`		number of available windows sizes
`Iwindow`	`integer`		selected window size

7.8 Module statistics

Purpose: This modules contains several routines for calculating computation and MFLOP/sec rates.

Constant definitions:

Name	Type	Purpose
Purpose: constants for operation numbers		
STAT_DSCAL	integer	
STAT_DAXPY	integer	
STAT_MVBAND2DE11	integer	
STAT_MVTRI	integer	
STAT_PRECSTRI	integer	
STAT_PRECSLOWERLINE	integer	
STAT_COPY	integer	
STAT_LINCOMB	integer	
STAT_PRECSMATGB	integer	
STAT_RESTE11	integer	
STAT_PROLE11	integer	
STAT_PRECSJAC	integer	
STAT_PRECSTRIGS	integer	
STAT_GAUSSELE	integer	
STAT_DH1ERROR	integer	
STAT_SCALEVECTOR	integer	analysed tools_dH1error and counted the number of basic operations there

Type definitions:

Type name	component name	Type	Rank	Purpose
t_stat	Statistics about runtime and number of operation			
	nop	integer(I64)		number of operations
	dtimeCpu	real(DP)		CPU time
	dtimeReal	real(DP)		Real time

7.8.1 Subroutine stat_clearOp

Interface:
`stat_clearOp(rstat)`

Description:
This routine clears the nop statistics in rstat

Input variables:

Name	Type	Rank	Description
rstat	t_stat		

7.8.2 Subroutine stat_clearTime

Interface:

`stat_clearTime(rstat)`

Description:
This routine clears the time statistics in rstat

Input variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		

7.8.3 Subroutine stat_clear

Interface:

`stat_clear(rstat)`

Description:
This routine clears the statistic in rstat

Input variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		

7.8.4 Subroutine stat_copyOp

Interface:

`stat_copyOp(rstat1, rstat2)`

Description:
This routine copies the nop statistics in rstat1 to rstat2

Input variables:

Name	Type	Rank	Description
`rstat1`	`t_stat`		

Output variables:

Name	Type	Rank	Description
`rstat2`	`t_stat`		

7.8.5 Subroutine stat_copyTime

Interface:

`stat_copyTime(rstat1, rstat2)`

Description:
This routine copies the time statistics in rstat1 to rstat2

Input variables:

Name	Type	Rank	Description
`rstat1`	`t_stat`		

Output variables:

Name	Type	Rank	Description
`rstat2`	`t_stat`		

7.8.6 Subroutine stat_copyStat

Interface:
`stat_copyStat(rstat1, rstat2)`

Description:
This routine copies the statistics in rstat1 to rstat2

Input variables:

Name	Type	Rank	Description
`rstat1`	`t_stat`		

Output variables:

Name	Type	Rank	Description
`rstat2`	`t_stat`		

7.8.7 Subroutine stat_addOp

Interface:
`stat_addOp(rstat1, rstat2)`

Description:
This routine adds the nop statistics in rstat1 to rstat2

Input variables:

Name	Type	Rank	Description
`rstat1`	`t_stat`		

Input/Output variables:

Name	Type	Rank	Description
rstat2	t_stat		

7.8.8 Subroutine stat_addTime

Interface:
stat_addTime(rstat1, rstat2)

Description:
This routine adds the time statistics in rstat1 to rstat2

Input variables:

Name	Type	Rank	Description
rstat1	t_stat		

Input/Output variables:

Name	Type	Rank	Description
rstat2	t_stat		

7.8.9 Subroutine stat_add

Interface:
stat_add(rstat1, rstat2)

Description:
This routine adds the statistics in rstat1 to rstat2

Input variables:

Name	Type	Rank	Description
rstat1	t_stat		

Input/Output variables:

Name	Type	Rank	Description
rstat2	t_stat		

7.8.10 Subroutine stat_linCombOp

Interface:

`stat_linCombOp(dcoeff1, rstat1, dcoeff2, rstat2, rstatDest)`

Description:
This routine computes an linear combination of two operation counters

Input variables:

Name	Type	Rank	Description
`rstat1, rstat2`	`type(t_stat)`		
`dcoeff1, dcoeff2`	`real(DP)`		

Output variables:

Name	Type	Rank	Description
`rstatDest`	`type(t_stat)`		

7.8.11 Subroutine stat_getNumOp

Interface:
`stat_getNumOp(ctype, n, rstat)`

Description:
This routine returns for a specific operation and number of unknowns the number of operations.

Input variables:

Name	Type	Rank	Description
`ctype, n`	integer		

Output variables:

Name	Type	Rank	Description
`rstat`	`type(t_stat)`		

7.8.12 Subroutine stat_addNumOp

Interface:
`stat_addNumOp(ctype, n, rstat)`

Description:
This routine adds for a specific operation and number of unknowns the number of operations to rstat.

Input variables:

Name	Type	Rank	Description
`ctype, n`	integer		

Output variables:

Name	Type	Rank	Description
rstat	type(t_stat)		

7.8.13 Function stat_sprint

Interface:
stat_sprint(rstat)

Description:
This routine prints and returns the information stored in rstat

Input variables:

Name	Type	Rank	Description
rstat	type(t_stat)		

7.8.14 Subroutine stat_MFLOPs

Interface:
stat_MFLOPs(rstat, iiter, dmfr1, dmfr2)

Description:
This routine calculates the MFLOP per second rates for the statistics stored in rstat.

Input variables:

Name	Type	Rank	Description
rstat	type(t_stat)		
iiter	integer		

Output variables:

Name	Type	Rank	Description
dmfr1, dmfr2	real(DP)		

7.8.15 Function stat_getSystemTime

Interface:
stat_getSystemTime()

Description:
This routine stores the system time in rstat%dtimeReal

Result:

type(t_stat) : rstat : current time

7.8.16 Function stat_addTimes

Interface:

```
stat_addTimes(rstat1, rstat2)
```

Description:
This function adds the time statistics of rstat1 and rstat2 and returns the sum in a new t_stat object rstatDest.

Input variables:

Name	Type	Rank	Description
`rstat1, rstat2`	`t_stat`		

Result:
type(t_stat) : rstatDest

7.8.17 Function stat_subTimes

Interface:

```
stat_subTimes(rstat1, rstat2)
```

Description:
This function substracts the time statistics of rstat2 from rstat1 and returns the difference in a new t_stat object rstatDest.

Input variables:

Name	Type	Rank	Description
`rstat1, rstat2`	`t_stat`		

Result:
type(t_stat) : rstatDest

7.8.18 Function stat_diffTime

Interface:

```
stat_diffTime(rstat)
```

Description:
This function stores the difference between rstat and the current system time in a new object rstatDest and returns this.

Input variables:

Name	Type	Rank	Description
rstat	t_stat		

Result:
type(t_stat) : rstatDest

7.8.19 Function stat_sgetTime

Interface:
`stat_sgetTime(rstat)`

Description:
This function returns the passed time variable as string

Input variables:

Name	Type	Rank	Description
rstat	type(t_stat)		time variable

Result:
character(len=15) : stime

7.8.20 Function stat_sgetTimeRatio

Interface:
`stat_sgetTimeRatio(rstatP, rstatG)`

Description:
This function returns the ratio between part time in rstatP and complete time in rstatG

Input variables:

Name	Type	Rank	Description
rstatP	type(t_stat)		part time
rstatG	type(t_stat)		complete time

Result:
character(len=19) : stimeRatio

7.9 Module output

Purpose: This module covers the job of exporting simulation results to several formats. Export of node-based and cell-based data to AVS and GMV documents is fully supported.

7.9.1 Subroutine ucd_exportAVS

Interface:

ucd_exportAVS(nparBlock, sfilename, nvertVis, nelemVis, NvertPerPb, NelemPerPb, nnumberOfDataVectors, nnumberOfAddDataVectors, H_DdataVis, SdataVectorName, CdataVectorType, CtreatDataVectors, nnumberOfAddIntVars, IaddIntVars, nnumberOfAddDblVars, IaddDblVars, cmiscData, h_DvertCoordsVis, h_IelemInfoVis, h_IparBlockIdxVis, dlar, dlhmin, DconvRateMin, DconvRateMax, DconvRateAvg)

Description:
This routine is called from the master process and covers the export of calculated data to AVS format. If you want to postprocess calculated data, provide in your userdef module a routine named user_exportVisData, perform the operations there and set the optional last argument to comm_sendVisData to .TRUE. An arbitrary number of data vectors of arbitrary size is supported. You can give instructions how to deal with each vector: allow or deny export (the latter when you only need the data to derive other data in a postprocessing step), agglomerate some data vectors to a velocity field, add data vector to coordinates (to visualise calculated displacements) etc. Under normal circumstances the routine should not be called directly from an application, but from comm_receiveVisData. Description of UCD file format: http://help.avs.com/Express/doc/help/reference/dvmac/UCD_Form.htm

Input variables:

Name	Type	Rank	Description
nparBlock	integer		number of parallel blocks
sfilename	c(*)		output file name
nvertVis, nelemVis	integer		number of vertices and elements on visualisation output level
NvertPerPb	integer	:	number of vertices per parallel block
NelemPerPb	integer	:	number of elements per parallel block
nnumberOfDataVectors	integer		number of solution vectors to be exported to visualisation output file
nnumberOfAddDataVectors	integer		number of data vectors that will be calculated on-the-fly by user_additionalVisData when performing UCD output
H_DdataVis	integer	:	array of handles to all data vectors
SdataVectorName	c(*)	:	name of all (fixed) solution vectors, to be displayed in visualisation program
SdataVectorName	c(SYS_STRLEN)	:	
CdataVectorType	integer	:	type of data of all (fixed) solution vectors: node-oriented, cell-oriented, ...
CtreatDataVectors	integer	:	array describing what to do with (fixed) data vectors: export or not, agglomerate velocity components to a velocity field or not Use flags like UCD_NOEXPORT, UCD_NO_AGGLOMERATE, UCD_AGGLO_VELOCITYFIELD etc.
nnumberOfAddIntVars	integer		number of additional integer variables (e.g. iteration number, parameters for user_additionalVisData)
IaddIntVars	integer	:	array containing additional integer variables
nnumberOfAddDblVars	integer		number of additional double variables (e.g. time step, parameters for user_additionalVisData)
IaddDblVars	real(DP)	:	array containing additional double variables
cmiscData	integer		bit array describing what kind of grid statistics to export Use flags like UCD_MINCONVRATE, UCD_HMIN, UCD_ASPECTRATIOS etc.
h_DvertCoordsVis	integer		handle for coordinates of all nodes
h_IelemInfoVis	integer		handle for array containing the four node numbers that build up each element
h_IparBlockIdxVis	integer		handle for array telling for each element from which parallel block it is received + access pointer
dlar	real(DP)	:	Vectors containing aspect ratios and convergence rates
dlhmin	real(DP)	:	
DconvRateMin	real(DP)	:	

7.9.2 Subroutine ucd_exportGMV

Interface:

`ucd_exportGMV(nparBlock, sfilename, nvertVis, nelemVis, NvertPerPb, NelemPerPb, nnumberOfDataVectors, nnumberOfAddDataVectors, H_DdataVis, SdataVectorName, CdataVectorType, CtreatDataVectors, nnumberOfAddIntVars, IaddIntVars, nnumberOfAddDblVars, IaddDblVars, cmiscData, h_DvertCoordsVis, h_IelemInfoVis, h_IparBlockIdxVis, dlar, dlhmin, DconvRateMin, DconvRateMax, DconvRateAvg)`

Description:
This routine is called from the master process and covers the export of calculated data to GMV format. If you want to postprocess calculated data, provide in your userdef module a routine named user_exportVisData, perform the operations there and set the optional last argument to comm_sendVisData to .TRUE. An arbitrary number of data vectors of arbitrary size is supported. You can give instructions how to deal with each vector: allow or deny export (the latter when you only need the data to derive other data in a postprocessing step), agglomerate some data vectors to a velocity field, add data vector to coordinates (to visualise calculated displacements) etc. Under normal circumstances the routine should not be called directly from an application, but from comm_receiveVisData. The first additional integer is always treated as solver iteration number, the first additional double is always treated as time iteration.

Input variables:

Name	Type	Rank	Description
nparBlock	integer		number of parallel blocks
sfilename	c(*)		output file name
nvertVis, nelemVis	integer		number of vertices and elements on visualisation output level
NvertPerPb	integer	:	number of vertices per parallel block
NelemPerPb	integer	:	number of elements per parallel block
nnumberOfDataVectors	integer		number of solution vectors to be exported to visualisation output file
nnumberOfAddDataVectors	integer		number of data vectors that will be calculated on-the-fly by user_additionalVisData when performing UCD output
H_DdataVis	integer	:	array of handles to all data vectors
SdataVectorName	c(*)	:	name of all solution vectors, to be displayed in visualisation program
SdataVectorName	c(SYS_STRLEN)	:	
CdataVectorType	integer	:	type of data of current solution vector: node-oriented, cell-oriented, ...
CtreatDataVectors	integer	:	array describing what to do with data vectors: export or not, agglomerate velocity components to a velocity field or not Use flags like UCD_NOEXPORT, UCD_NO_AGGLOMERATE, UCD_AGGLO_VELOCITYFIELD etc.
nnumberOfAddIntVars	integer		number of additional integer variables (e.g. iteration number, parameters for user_additionalVisData)
IaddIntVars	integer	:	array containing additional integer variables
nnumberOfAddDblVars	integer		number of additional double variables (e.g. time step, parameters for user_additionalVisData)
IaddDblVars	real(DP)	:	array containing additional double variables
cmiscData	integer		bit array describing what kind of grid statistics to export Use flags like UCD_MINCONVRATE, UCD_HMIN, UCD_ASPECTRATIOS etc.
h_DvertCoordsVis	integer		handle for coordinates of all nodes
h_IelemInfoVis	integer		handle for array containing the four node numbers that build up each element
h_IparBlockIdxVis	integer		handle for array telling for each element from which parallel block it is received + access pointer
dlar	real(DP)	:	Vectors containing aspect ratios and convergence rates
dlhmin	real(DP)	:	
DconvRateMin	real(DP)	:	
DconvRateMax	real(DP)	:	

7.9.3 Subroutine ucd_calcGridStatistics

Interface:

`ucd_calcGridStatistics( nparBlock, nvertVis, nelemVis, h_DvertCoordsVis, h_IelemInfoVis, NelemPerPb, dlhmin, dlar, daspectRatioGlobal, dhminGlobal)`

Description:
This routine calculates maximal aspect ratio and minimal grid size.

Input variables:

Name	Type	Rank	Description
`nparBlock`	`integer`		number of parallel blocks
`nvertVis, nelemVis`	`integer`		number of vertices and elements on visualisation output level
`h_DvertCoordsVis`	`integer`		handle for coordinates of all nodes
`h_IelemInfoVis`	`integer`		handle for array containing the four node numbers that build up each element
`NelemPerPb`	`integer`	:	number of elements per parallel block

Output variables:

Name	Type	Rank	Description
`dlar`	`real(DP)`	:	Vectors containing aspect ratios and convergence rates
`dlhmin`	`real(DP)`	:	
`daspectRatioGlobal`	`real(DP)`		maximal aspect ratio
`dhminGlobal`	`real(DP)`		minimal grid size

7.10 Module hlayer

Purpose: This module implements the hierachical layer structure of the vector management. It contains routines for reservation and freeing of vectors, further access functions and inquiry functions.

Constant definitions:

Name	Type	Purpose
`HL_ALL`	`integer`	reserve for all multigrid levels
`HL_ONE`	`integer`	reserve for one multigrid
`HL_SD`	`integer`	hierachical layer subdomain
`HL_PB`	`integer`	hierachical layer parallelblock

HL_MB	integer	hierachical layer matrixblock
HL_SUBEXT	integer	
INCXX	integer	

HL_MAXMBPPB	integer	maximal number of MB per ParBlk
HL_MAXVECPMB_SD	integer	maximal number of vecs per MB
HL_MAXVECPMB_PB	integer	
HL_MAXVECPMB_MB	integer	
HL_MAXVECPMB_SD_INC	integer	increment values for increasing vector storage
HL_MAXVECPMB_PB_INC	integer	
HL_MAXVECPMB_MB_INC	integer	
HL_MAXMGLEVEL	integer	maximal number of mg levels

Type definitions:

Type name	component name	Type	Rank	Purpose
Thlayer				
	ilsd_vec	integer	:, :	
	ilpar_vec	integer	:, :	
	ilmb_vec	integer	:, :	
	imlsd_vec	integer	:	
	imlpar_vec	integer	:	
	imlmb_vec	integer	:, :	
	itmpsd_vec	integer	:	
	itmppar_vec	integer	:	
	itmpmb_vec	integer	:, :	
	snamesd_vec	c(32)	:	
	snamepar_vec	c(32)	:	
	snamemb_vec	c(32)	:, :	
	rnext	Thlayer		
Thlayer_access				
	ra	Thlayer		

7.10.1 Function hl_igetHLayer

Interface:

```
hl_igetHLayer(shl)
```

Description:
This functions returns the internal values for the string identifiers HL_SD,HL_PB and HL_MB.

Input variables:

Name	Type	Rank	Description
`shl`	`c (5)`		string identifier

Result:
integer : internal value of hierachical layer

7.10.2 Function hl_sgetHLayerName

Interface:
`hl_sgetHLayerName(chLayer)`

Description:
This function returns the string identifier for the internal constants HL_SD,HL_PB and HL_MB.

Input variables:

Name	Type	Rank	Description
`chLayer`	integer		hierachical layer constant

Result:
character (len=5) : string identifier

7.10.3 Function hl_igetHLayerAbove

Interface:
`hl_igetHLayerAbove(ilayer)`

Description:
This function returns the hierachical layer up to the given one.

Input variables:

Name	Type	Rank	Description
`ilayer`	`integer`		given layer

Result:
integer : the layer one above of the iven one

7.10.4 Function hl_igetHLayerBelow

Interface:
`hl_igetHLayerBelow(ilayer)`

Description:

This functions returns the hierachical layer below the given one.

Input variables:

Name	Type	Rank	Description
`ilayer`	`integer`		given layer

Result:
integer : the layer below the given one

7.10.5 Subroutine hl_init

Interface:
`hl_init(nmb)`

Description:
This functions initializes the data structure.

7.10.6 Subroutine hl_select

Interface:
`hl_select(dy, imglevel, ilayer, h_Dvec, imacroidx)`

Description:
This subroutine returns to a given selection (imglevel, ilayer, h_Dvec, imacroidx) the pointer to the vector dy.

Input variables:

Name	Type	Rank	Description
`imglevel`	`integer`		multigrid level
`ilayer`	`integer`		hierachical layer (HL_SD,HL_PB,HL_MB)
`h_Dvec`	`integer`		vector handle
`imacroidx`	`integer`		matrixblock index, only relevant for HL_MB

Output variables:

Name	Type	Rank	Description
`dy`	`real(DP)`	:	access pointer

7.10.7 Subroutine hl_attachRHS

Interface:
`hl_attachRHS(rparBlock, chLayer, iMGLevel, h_Drhs, rdiscrId)`

Description:
This function attaches the given vector handle as right hand side to the given matrixblock object. But be aware, what happens to the former RHS vector! If you do not know its handle, then the vector cannot be accessed anymore while its memory is still allocated!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
h_Drhs	integer		new rhs variable
chLayer	integer		hierachical layer have to be HL_SD
iMGLevel	integer		multigrid level on which to exchange if iMGLevel .gt. 0: exchange level iMGLevel only = MB_ALL_LEV : exchange all levels .lt. 0 : exchange all levels up to -iMGLevel

Input/Output variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		matrixblock object

7.10.8 Subroutine hl_copy

Interface:
`hl_copy(rparBlock, imglev, chLayer, imatBlock, h_DvecSrc, h_DvecDest, rstat)`

Description:
This functions copies a vector of the given level for the given matrix blocks.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imglev	integer		multigrid level, MB_ALL_LEV means all multigrid levels
chLayer	integer		hierachical layer
imatBlock	integer		matrix block index
h_DvecSrc	integer		source variable

Output variables:

Name	Type	Rank	Description
h_DvecDest	integer		destination variable
rstat	t_stat		structure for storing runtime statistics

7.10.9 Subroutine hl_add

Interface:

`hl_add(rparBlock, imglev, chLayer, imatBlock, h_Dx, h_Dy, dcoeff1, dcoeff2, rstat)`

Description:
This functions adds a vector to another vector of the given level for the given maxtrix block, h_Dy = dcoeff1 * h_Dx + dcoeff2 * h_Dy.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`chLayer`	integer		hierachical layer
`h_Dx`	`integer`		first vector
`dcoeff1`	`real(DP)`		coefficient 1
`dcoeff2`	`real(DP)`		coefficient 2
`imglev`	`integer`		multigrid level, MB_ALL_LEV means all multigrid levels
`imatBlock`	`integer`		maxtrix block index

Input/Output variables:

Name	Type	Rank	Description
`h_Dy`	`integer`		second vector (which will contain the result)

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		structure for storing runtime statistics

7.10.10 Subroutine hl_copyrhs

Interface:

`hl_copyrhs(rparBlock, rdiscrId, imglev, chLayer, imatBlock, h_Dy, cflag, rstat)`

Description:
This functions copies the rhs vector to another vector of the given level for the given maxtrix block.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
chLayer	integer		hierachical layer
imglev	integer		multigrid level, MB_ALL_LEV means all multigrid levels
imatBlock	integer		maxtrix block index
rdiscrId	Tdiscrid		discretisation id
cflag	integer		0=get rhs, 1=set rhs

Input/Output variables:

Name	Type	Rank	Description
h_Dy	integer		second vector (which will contain the result)

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.10.11 Subroutine hl_scale

Interface:

hl_scale(rparBlock, imglev, chLayer, imb, h_Dvec, dcoeff, rstat)

Description:
This functions scales a vector 'in situ' by dcoeff, idvaridx = dcoeff*h_Dvec.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
chLayer	integer		hierachical layer
h_Dvec	integer		vector handle
dcoeff	real(DP)		scaling coefficient
imglev	integer		multigrid level, MB_ALL_LEV means all multigrid levels
imb	integer		matrix block index

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.10.12 Subroutine hl_print

Interface:

`hl_print(rparBlock, imglev, chLayer, isvaridx)`

Description:
This functions prints the given vector.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imglev`	`integer`		multigrid level, MB_ALL_LEV means all multigrid levels
`isvaridx`	`integer`		source variable
`chLayer`	integer		hierachical layer

7.10.13 Function hl_reserve

Interface:

`hl_reserve(scall, sname, rparBlock, clayer, imglevel, cflag, imult, imacroidx, imatidx, itemplidx, copt)`

Description:
This functions reserves a vector structure and returns a variable identifier (interface has to be reworked).

Input variables:

Name	Type	Rank	Description
`scall`	`c(*)`		name of the calling subroutine
`sname`	`c(*)`		name of the vector
`rparBlock`	`t_parBlock`		parallelblock object
`clayer`	integer		hierachical layer
`imglevel`	`integer`		multigrid level maximum
`cflag`	integer		multigrid level flag HL_ONE,HL_ALL

Result:
integer : variable identifier

7.10.14 Function hl_icountFree

Interface:

`hl_icountFree(clayer)`

Description:
This function returns the number of free variable slots for the given hierachical layer.

Input variables:

Name	Type	Rank	Description
`clayer`	integer		hierachical layer

Result:
integer : number of free slots

7.10.15 Subroutine hl_free

Interface:
`hl_free(rparBlock, clayer, imglevel, h_Dvec, cflag, imacroidx)`

Description:
This function frees the given variable.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`clayer`	integer		hierachical layer
`imglevel`	`integer`		multi grid layer
`cflag`	integer		switch flag HL_ONE, HL_ALL
`imacroidx`	`integer`		macro index, only for HL_MB
`h_Dvec`	`integer`		vector handle

7.10.16 Function hl_igetLength

Interface:
`hl_igetLength(clayer, imglevel, h_Dvec, imacroidx)`

Description:
This function returns the length of the given variable.

Input variables:

Name	Type	Rank	Description
`clayer`	integer		hierachical layer
`imglevel`	`integer`		multigrid level
`h_Dvec`	`integer`		vector handle
`imacroidx`	`integer`		matrixblock index

Result:
integer : length of the given vector

7.10.17 Subroutine hl_clear

Interface:

`hl_clear(rparBlock, clayer, imglevel, h_Dvec, cflag, imacroidx, rstat)`

Description:
This function clears the given vector or vectors.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`clayer`	integer		hierachical layer
`imglevel`	`integer`		multigrid level
`h_Dvec`	`integer`		vector handle
`cflag`	integer		access flag: HL_ONE, HL_ALL
`imacroidx`	`integer`		matrixblock index

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		structure for storing runtime statistics

7.10.18 Subroutine hl_set

Interface:

`hl_set(rparBlock, clayer, imglevel, h_Dvec, cflag, imacroidx, dvalue, rstat)`

Description:
This function sets the given vector to dvalue.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`clayer`	integer		hierachical layer
`imglevel`	`integer`		multigrid level
`h_Dvec`	`integer`		vector handle
`cflag`	integer		access flag: HL_ONE, HL_ALL
`imacroidx`	`integer`		matrixblock index
`dvalue`	`real(DP)`		value to be set

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.10.19 Subroutine hl_setRandom

Interface:

hl_setRandom(rparBlock, clayer, imglevel, h_Dvec, cflag, imacroidx)

Description:
This function sets the given vector to random values between 0.0 and 1.0.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
clayer	integer		hierachical layer
imglevel	integer		multigrid level
h_Dvec	integer		vector handle
cflag	integer		access flag: HL_ONE, HL_ALL
imacroidx	integer		matrixblock index

7.10.20 Subroutine hl_restoreVector

Interface:

hl_restoreVector(rparBlock, h_Dvec, imglev)

Description:
This function restores a saved vector from a file to the given variable.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
h_Dvec	integer		vector handle
imglev	integer		multigrid level

7.10.21 Subroutine hl_storeVector

Interface:

hl_storeVector(rparBlock, h_Dvec, imglev)

Description:
This function stores a given vector from to a file.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
h_Dvec	integer		vector handle
imglev	integer		multigrid level

7.11 Module loadbal

Purpose: This module contains the definition of the loadbalancing strategies.

7.12 Module macro

Purpose: This module defines the data structure for the basic data element macro. Further it contains routines for sending/receiving and copying macros.

Constant definitions:

Name	Type	Purpose
MAC_BDX_IDX	integer	index for boundary nodes per macro
MAC_BDX_BELE	integer	index for boundary elements per macro
MAC_BDX_BC	integer	???
MAC_BC_LENGTH	integer	entry length of BC entry per node
MAC_BC_GLENGTH	integer	number of bands affected by incorporating Dirichlet boundary conditions for Q1
MAC_BC_GLENGTHQ1	integer	
MAC_BC_GLENGTHQ2	integer	
MAC_BC_GLENGTHQ3	integer	
MAC_BC_LENPOS	integer	position of length entry
MAC_BDX_MAT	integer	index for boundary matrix entries (baseline)
MAC_EPM	integer	number of edges per macro
MAC_DIM	integer	dimensions of the problem
MAC_DIAGDEPTH	integer	maximum number of elements meeting in one vertex
MAC_EXCDEPTH	integer	maximum number of matrix lines for exchanging
MAC_SRVDEPTH	integer	number of service lines in the iborderidx structure

MAC_MAXMGLEVEL	integer	maximum multigrid level
MAC_INACTIVE	integer	identifier macro edge inactive (formerly 2 !!!)
MAC_ACTIVE	integer	identifier macro edge active
AS_EDGENO	integer	edge type: inner edge, no parallelblock border
AS_EDGEOUTER	integer	edge type: boundary edge, no parallelblock border
AS_EDGEINNER	integer	edge type: inner edge, on parallelblock border
AS_EDGEMIRROR	integer	edge type: explicit mirror boundary
AS_EDGEBC_DIR	integer	edge status: Dirichlet boundary condition (in the case of boundary edge)
AS_EDGEBC_NEU	integer	edge status: Neumann boundary condition (in case of for boundary edge)

Type definitions:

Type name	component name	Type	Rank	Purpose
t_macroLevData	level dependent macro information			
	ilborderidx	integer		exchange information, see assembly
	ilmatsave_neu	integer	MB_MAXDA	matrix entry for neumann boundaries save variable
	ilmatsave	integer	2	matrix entry save variable (handle array)
	ilmatsave_sg	integer	2	
	ilmatsave_g	integer	2	
	ilcorrmatentry	integer	MB_MAXDA, 3	correction entries
	ilcorrmatentry2	integer	MB_MAXDA, 3	correction entries indices
	icorrmatidx	integer	MB_MAXDA, 3	index for CONSTANT correction
	icorrmatidx0	integer	MB_MAXDA, 3	index for CONSTANT correction
	ilbsave	integer	MB_MAXDA, MAC_EPM	rhs entry save variable
	dvecsave	real(DP)	MAC_EPM	save area for vector corner points

	ndofPerEdge	integer		save area for vector edge points integer , dimension(MAC_EPM) :: ilvecsave number of degrees of freedom per edge
	lmapping	integer		handle for mapping vector
	nmex	integer		number of macro vertices in x direction
	nmey	integer		number of macro vertices in y direction
	nme	integer		number of vertices in macro
	ilmb_edge	integer	2	save area for overlapping edges
	dmb_diag	real(DP)	MB_MAXDA, MAC_DIAGDEPTH	save area for corner points
	fastass	logical		switch fast matrix assembly (DEPR)
	bmatConstMul	logical		switch constant matrix multiplication
	h_NadjMacros_SD	integer		handle, number of adj vertices SD-layer
	h_NadjMacros_PB	integer		handle, number of adj vertices PB-layer
	rnext	t_macroLevData		

t_macro level independent macro information

	iidxSD	integer		global index of the macro
	imatBlockIdxSD	integer		global index of the matrixblock this macro belongs to
	imatBlockIdxPB	integer		index of the matrixblock this macro belongs to in parallelblock
	bmarker	logical		???
	nin	integer	:	???
	caniso	integer		type of anisotropy represented by a bit array [b3, b2, b1, b0] b0 = 0 –¿ no anisotropy in x-direction, b0 = 1 –¿ anisotropy in x-direction b1 = 0 –¿ no anisotropy in y-direction, b1 = 1 –¿ anisotropy in y-direction b2 = 0 –¿ anisotropy to the left, b2 = 1 –¿ anisotropy to the right b3 = 0 –¿ anisotropy to the top, b3 = 1 –¿ anisotropy to the

dvecsave	real(DP)	MAC_EPM	??? save area for vector corner points
irefLevCoarse	integer		refinement level of the macro on coarse mesh
DanisoFactor	real(DP)	3	anisotropy factors
IvertIdxSD	integer	MAC_EPM	global indices of macro nodes
IedgeIdxSD	integer	MAC_EPM	global indices of macro edges
dnodes	real(DP)	MAC_DIM, MAC_EPM	boundary vertex: parameter value in first component, second comp. empty inner vertex: cartesian coordinates ??? Eigentlich waere eine eindimensionale Struktur ausreichend! Dann muesste in der mastermod.f90 das Einlesen anders gestaltet werden...
dcnodes	real(DP)	MAC_DIM, MAC_EPM	cartesian coordinates of vertices (in case of inner vertices: dcnodes = dnodes) ??? siehe dnodes
IvertBCompIdx	integer	MAC_EPM	status of macro vertices: 0 = inner vertex, n = bound. vert. on component n
iparBlockIdx	integer		number of the parallel block this macro belongs to
IadjMacIdx	integer	MAC_EPM	macro indices of the edge neighbours
IdiagAdjMacIdx	integer	MAC_EPM, MAC_DIAGDEPTH	macro indices of the diagonal neighbours
IadjMbIdx	integer	MAC_EPM	matrix block indices of the edge neighbours
IdiagAdjMbIdx	integer	MAC_EPM, MAC_DIAGDEPTH*2	matrix block indices of the diagonal neighbours
IadjPbIdx	integer	MAC_EPM	parallel block indices of the edge neighbours

IdiagAdjPbIdx	integer	MAC_EPM, MAC_DIAGDEPTH	parallel block indices of the diagonal neighbours
CcommStatEdge	integer	MAC_EPM	edge communication status binary coded: 0 = 0+0 not sent + not received, ..., 3 "=" 1+1 sent + received
CcommStatEdgeDefault	integer	MAC_EPM	default for edge communication status
CedgeStatus	integer	MAC_EPM	status of the edges (0 = inner edge, 1 = boundary edge, 2 = parallelblock border)
inodebc	integer	MAC_EPM	boundary condition if outer edge (only for old file format)
CpriorEdgeSD	integer	MAC_EPM	edge priority (SD-layer) (0 or 1)
CpriorEdgePB	integer	MAC_EPM	edge priority (PB-layer) (0 or 1)
CpriorVertSD	integer	MAC_EPM	vertex priority (SD-layer) (0 or 1)
CpriorVertPB	integer	MAC_EPM	vertex priority (PB-layer) (0 or 1)
CcommStatVert	integer	MAC_EPM, MAC_DIAGDEPTH	vertex communication status binary coded: 0 = 0+0 not sent + not received, ..., 3 "=" 1+1 sent + received
CcommStatVertDefault	integer	MAC_EPM, MAC_DIAGDEPTH	default for vertex communication status
brect	logical		flag if macro is rectangular
CblindNodes	integer	MAC_EPM	0, if there are blind nodes, otherwise 1
IvertIdxMB	integer	MAC_EPM	indices of the macro vertices in matrix block
clsmootherprof	integer		local smoother selector via profiling
rnext	t_macro		pointer to next macro

t_macroAccess	record of macros			
	ra	t_macro		access variable
t_macroLevDataAccess	record of macros			
	ra	t_macroLevData		access variable
t_macroBase	record of macros			
	rmakrolevel	t_macroLevDataAccess	dim(:)	access variable

7.12.1 Subroutine macro_clearEdgeFlags

Interface:
`macro_clearEdgeFlags(rm)`

Description:
This routine clears the edge and diag communication flags.

Input variables:

Name	Type	Rank	Description
rm	t_macro		macro object

7.12.2 Subroutine macro_send

Interface:
`macro_send(idx, rm, imacroidx)`

Description:
This routine sends a macro to the specified process.

Input variables:

Name	Type	Rank	Description
rm	t_macro		macro object
idx	integer		destination process index
imacroidx	integer		local index of the macro on the process

7.12.3 Subroutine macro_receive

Interface:
`macro_receive(idx, rm, rmb, imacroidx)`

Description:
This routine received a macro from the specified process.

Input variables:

Name	Type	Rank	Description
rm	t_macro		macro object
rmb	t_matrixBlockbase		macro base object
idx	integer		destination process index
imacroidx	integer		local index of the macro on the process

7.12.4 Subroutine macro_copy

Interface:
macro_copy(rms, rmbs, rmd, rmbd, i, j)

Description:
This routine copies a macro to another.

Input variables:

Name	Type	Rank	Description
rms	t_macro		source macro object
rmd	t_macro		destination macro object
rmbs	t_matrixBlockbase		source macro base object
rmbd	t_matrixBlockbase		destination macro base object
i	integer		
j	integer		

7.13 Module mastermod

Purpose: This modules contains the master part of the main program. Every request from a slave to the master is handled in the main routine startmaster. In this routine, there exists an endless loop, in which all messages are recieved. Depending on the message header, the program goes in an appropriate branch and fulfills the request of the calling slave(s).

7.13.1 Subroutine master_start

Interface:
master_start()

Description:
This is the main master routine, called by the program masterslave in the module masterslave.f90.

7.14 Module masterservice

Purpose: This module provides all the routines which are called by the endless loop in the module

mastermod.f90.

7.14.1 Subroutine msrv_feastinit

Interface:

`msrv_feastinit(rparBlock, rdiscr, rbsave, rmdParBlock, Rsolver, nscarc, rmdSolver)`

Description:
This routine inits the main data structures like coarse grid solver, communication structures, boundary conditions, grid and macro structure.

Input/Output variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`rdiscr`	`Tdiscretization`		discretisation object
`rbsave`	`Tbordersave`		
`rmdParBlock`	`t_masterDataParallelBlock`		
`Rsolver`	`t_solver`	dim(:)	
`nscarc`	`integer`		
`rmdSolver`	`t_masterDataSolver`		

7.14.2 Subroutine msrv_readmasterfile

Interface:

`msrv_readmasterfile(smaster, rdiscr)`

Description:
This routine reads the configuration file for an application and stores the data in global variable sys_sysconfig.

Input variables:

Name	Type	Rank	Description
`smaster`	`c(*)`		

Input/Output variables:

Name	Type	Rank	Description
`rdiscr`	`Tdiscretization`		

7.14.3 Subroutine msrv_showconfig

Interface:

`msrv_showconfig(rdiscr)`

Description:

Input variables:

Name	Type	Rank	Description
rdiscr	Tdiscretization		

7.14.4 Subroutine msrv_readFEASTv3

Interface:

msrv_readFEASTv3(rmasterParBlock, rdiscr, nparBlock, nmacroEdges, nmacros, nmacroVerts, h_ImacroEdgeInfo, h_DmacroVertInfo, NmacrosPerPb)

Description:
Output variables:

Name	Type	Rank	Description
rmasterParBlock	t_parBlock		parallelblock structure
rdiscr	Tdiscretization		discretisation structure

7.14.5 Subroutine msrv_calcedges

Interface:

msrv_calcedges(rmasterParBlock, nmacros, nmacroEdges, h_ImacroEdgeInfo)

Description:
Input/Output variables:

Name	Type	Rank	Description
rmasterParBlock	t_parBlock		

Input variables:

Name	Type	Rank	Description
nmacroEdges	integer		
nmacros	integer		
h_ImacroEdgeInfo	integer		

7.14.6 Subroutine msrv_mainloop_masterfin

Interface:

msrv_mainloop_masterfin(nparBlock, bmasterFinish)

Description:
This routine handles what to do for the master process when it receives the message identifier COMM_MASTERFIN. This message is sent from a slave process, preferably via routine par_finish. Upon completion of this routine, a flag is set that causes the master process to end.

Input variables:

Name	Type	Rank	Description
nparBlock	integer		

Output variables:

Name	Type	Rank	Description
bmasterFinish	logical		

7.14.7 Subroutine msrv_partdomain

Interface:

msrv_partdomain(rmasterParBlock, nparBlock, NmacrosPerPb, nupart)

Description:
This routine distributes automatically the coarse grid macros to the parallel blocks. For the trivial cases one parallel process and as many parallel processes as macros the mapping is done autonomously. For all other cases METIS partitioning algorithms are called.

Input variables:

Name	Type	Rank	Description
nupart	integer		number of parallel blocks to distribute the macros among

Input/Output variables:

Name	Type	Rank	Description
rmasterParBlock	t_parblock		parallel block of master process (administering the coarse grid)

Output variables:

Name	Type	Rank	Description
NmacrosPerPb	integer	:	number of macros per parallel block
nparBlock	integer		number of parallel blocks

7.14.8 Subroutine msrv_exportpartition

Interface:

msrv_exportpartition(sfilename)

Description:

This routine writes a partition to disk.

Input variables:

Name	Type	Rank	Description
`sfilename`	`c(*)`		name of file to write to

7.15 Module matrix

Purpose: This modules contains several control routines for the matrix multiplication. It acts as an frontend for the according SBBLAS routines. Further it defines the matrix types and toolkit routines for converting and output.

Constant definitions:

Name	Type	Purpose
`MA_SPARSE`	`integer`	matrix type sparse
`MA_BAND`	`integer`	matrix type banded
`MA_BANDCONST`	`integer`	matrix type banded const
`MA_VAR`	`integer`	matrix coefficient variabel
`MA_BLKCONST`	`integer`	matrix coefficient blockwise constant
`MA_CONST`	`integer`	matrix coefficient constant in a FE sense
`MA_SYM`	`integer`	matrix symmetric (yet not supp)
`MA_NSYM`	`integer`	matrix non symmetric
`MA_MAXBAND`	`integer`	maximum number of matrix bands (49 for Q3)

Type definitions:

Type name	component name	Type	Rank	Purpose
`Tmatrix`	matrix description type			
	`ctype`	integer		type of matrix (MA_SPARSE, MA_BAND)
	`neq`	`integer`		number of equations
	`cstruct`	integer		structure of the matrix (MA_VAR, MA_BLKCONST, MA_CONST)

`nnonZero`	`integer`	number of non-zero entries
`ndiag`	`integer`	number of bands/diagonals How the bands are enumerated can be seen in the following scheme of the matrix structure 1 6 7 8 9 5 * * * * * * * * * * * * * * * * * 4 * * * * * 9 3 * * * * * 8 2 * * * * * 7 * 6 2 3 4 5 1 The bands are also denoted by 2, 3, 4 = LL, LD, LU 5, 1, 6 = DL, DD, DU 7, 8, 9 = UL, UD, UU
`csym`	integer	flag if the matrix is symmetric (MA_SYM, MA_NSYM)
`h_Da`	`integer`	h_da is the handle for the array which stores the matrix entries in the case of variable band entries. Its length is ndiag * neq. since only the main diagonal DD band is of size neq, there are some "gaps" in the array subdiagonal bands have the gap at the beginning superdiagonal bands have the gap at the end Example: with 25 nodes the array h_Da looks like this (actually, the array is one-dim.!!) 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 1: * 2: 0 0 0 0 0 0 * * * * * * * * * * * * * * * * * * * 3: 0 0 0 0 0 * 4: 0 0 0 0 * 5: 0 * 6: * 0 7: * 0 0 0 0 8: * * * * * * * * * * * * * * * * * 0 0 0 0 0 9: * * * * * * * * * * * * * * * * * * * 0 0 0 0 0 0 In matrix form

	h_DaConst	integer		h_daConst is the handle for the array which stores the matrix entries in case of constant band entries
	h_Ioff	integer		handle of the array which stores for each band its start position in the vector h_Da. So in the example above: h_Ioff(1) = 1, h_Ioff(2) = 25+6+1 = 32, h_Ioff(2) = 50+5+1 = 56 ...
	h_Iidx	integer		handle for the array which stores for every matrix band its "distance" to the main diagonal (subdiagonals have negative, superdiagonals positive entries) So in the example above: h_Iidx(1) = 0, h_Iidx(2) = -sqrt(25) - 1 = -6, h_Iidx(3) = -sqrt(25) = -5, ... h_Iidx(6) = 1, h_Iidx(7) = sqrt(25) - 1, ... With the help of this information bands can be identified and correctly accessed
	ccubType	integer		cubature id
	dmajcoeff	real(DP)	9	coefficients of the diagonals if the matrix is constant
	rconst	type(Tconst)		pointer to const delta stucture

Tconst

	nblk	integer	
	lblk1	integer	3
	lblk2	integer	3
	lblk3	integer	3

7.15.1 Subroutine matrix_mmmult_coo

Interface:

matrix_mmmult_coo(neq, na1, da1, ix1, iy1, na2, da2, ix2, iy2, na3, da3, ix3, iy3, ierr)

Description:
This routine performs a matrix matrix multiplication A*B=C in the COO format.

Input variables:

Name	Type	Rank	Description
neq	integer		number of equations
na1	integer		number of nonzero entries in A
da1	real(DP)	:	nonzero entries in A
ix1	integer	:	x position of matrix entries in A
iy1	integer	:	y position of matrix entries in A
na2	integer		number of nonzero entries in B
da2	real(DP)	:	nonzero entries in B
ix2	integer	:	x position of matrix entries in B
iy2	integer	:	y position of matrix entries in B

Output variables:

Name	Type	Rank	Description
na3	integer		number of nonzero entries in C
da3	real(DP)	:	nonzero entries in C
ix3	integer	:	x position of matrix entries in C
iy3	integer	:	y position of matrix entries in C
ierr	integer		return status of operation

7.15.2 Subroutine matrix_getBand

Interface:
matrix_getBand(rmat, nsub, nsuper)

Description:
This routine returns the number of sub- and superdiagonals of the given matrix.

Input variables:

Name	Type	Rank	Description
rmat	Tmatrix		matrix

Output variables:

Name	Type	Rank	Description
nsub	integer		number of subdiagonals
nsuper	integer		number of superdiagonals

7.15.3 Subroutine matrix_print

Interface:
`matrix_print(rmat)`

Description:
This routines prints the matrix, only useful for small ones.

Input variables:

Name	Type	Rank	Description
`rmat`	`Tmatrix`		matrix

7.15.4 Subroutine matrix_linePrint

Interface:
`matrix_linePrint(rmat)`

Description:
This routines prints the matrix, only useful for small ones.

Input variables:

Name	Type	Rank	Description
`rmat`	`Tmatrix`		matrix

7.15.5 Subroutine matrix_matVecMult

Interface:
`matrix_matVecMult(dalpha, dbeta, rmat, dx, dy, iops) subroutine matrix_matVecMult(dalpha, dbeta, rmat, dx, dy, rstat)`

Description:
This routine performs the operation $y = \alpha * A * x + \beta * y$ The operation count is returned in iops.

Input variables:

Name	Type	Rank	Description
`dalpha`	`real(DP)`		coefficient alpha
`dbeta`	`real(DP)`		coefficient beta
`dx`	`real(DP)`	:	vector x
`rmat`	`Tmatrix`		matrix A

Input/Output variables:

Name	Type	Rank	Description
`dy`	`real(DP)`	:	vector y

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		operation count

7.15.6 Subroutine matrix_release

Interface:
`matrix_release(rmat)`

Description:
This routine releases the storage vectors of the matrix.

Input variables:

Name	Type	Rank	Description
`rmat`	`Tmatrix`		matrix A

7.15.7 Subroutine matrix_transfer_b2s

Interface:
`matrix_transfer_b2s(rmat1, rmat2)`

Description:
This routine converts a matrix from band to sparse typ. The storage for the new vector is allocated automatically.

Input variables:

Name	Type	Rank	Description
`rmat1`	`Tmatrix`		matrix band typ

Output variables:

Name	Type	Rank	Description
`rmat2`	`Tmatrix`		matrix sparse typ

7.15.8 Function matrix_getPosForRowDof

Interface:
`matrix_getPosForRowDof(idofRow, ibandIdx, ioff, neq)`

Description:
This function returns the global matrix position (in array Da) of the entry given by ibandIdx (defines the band) and idofRow (defines the row number). If this DOF does not appear in the band, the return

value ipos is set to -1. It is assumed that the arrays rmatrix%h_Ioff and rmatrix%h_Iidx have already been accessed outside this function such that it does not have to be done in every call of this function.

Input variables:

Name	Type	Rank	Description
`idofRow`	`integer`		row index
`ibandIdx`	`integer`		index of the band (negative –¿ subdiagonal, positive –¿ superdiagonal)
`ioff`	`integer`		offset which indicates the start of the band in the matrix array
`neq`	`integer`		number of equations/rows in the matrix

Result:

integer : !position in the band integer :: ipos

7.15.9 Function matrix_getPosForColDof

Interface:

`matrix_getPosForColDof(idofCol, ibandIdx, ioff, neq)`

Description:
This function returns the global matrix position (in array Da) of the entry given by ibandIdx (defines the band) and idofCol (defines the column number). If this DOF does not appear in the band, the return value ipos is set to -1. It is assumed that the arrays rmatrix%h_Ioff and rmatrix%h_Iidx have already been accessed outside this function such that it does not have to be done in every call of this function.

Input variables:

Name	Type	Rank	Description
`idofCol`	`integer`		column index
`ibandIdx`	`integer`		index of the band (negative –¿ subdiagonal, positive –¿ superdiagonal)
`ioff`	`integer`		offset which indicates the start of the band in the matrix array
`neq`	`integer`		number of equations/columns in the matrix

Result:

integer : !position in the band integer :: ipos

7.16 Module matrixblock

Purpose: This module contains the basic atomic unit called matrixblock.

Constant definitions:

Name	Type	Purpose
MB_MAXMACPMB	integer	maximum number of macros per matrix block
BC_MAX_BDB	integer	maximum number of boundary conditions per matrix block
MB_ALL_MB	integer	Flag to apply operation to all matrix blocks
MB_ALL_LEV	integer	Flag to apply operation to all mg levels

Type definitions:

Type name	component name	Type	Rank	Purpose
t_matrixBlockbase	sequence of matrixblocks			
	rmblevel	t_matrixBlock_access	dim(:)	access variable
t_matrixBlock_access	sequence of matrixblocks			
	ra	t_matrixBlock		access variable
t_matrixBlock	matrix block structure			
	h_IfictBoundNodes	integer		storage for fictious boundary nodes
	iidxSD	integer		global index of the matrixblock
	iidxPB	integer		local index of the matrixblock
	nBLF	integer		number of bilinear forms in this matrix block
	Rmatrix	Tmatrix		matrices of this block
	cstatus	integer	dim(:)	handle for RHS vector integer, dimension(:),pointer :: H_Drhs Controls wether the edges of the matrixblock lie on PB-(AS_SBSEMILOCAL) or SD-border (AS_SBGLOBAL) (otherwise AS_SBLOCAL) The first two correspond to averaged matrix entries, the last to full entries.
	lprpoints	integer		??? handle, priority vector, PB-layer
	lprpointsg	integer		??? handle, priority vector, SD-layer
	iref	integer		??? refinementlevel of the matrixblock
	lbc	integer		
	ibccidx	integer		
	lbccoeff	integer		
	rnext	t_matrixBlock		

Tmpmb	mapping macros/matblock		
	inmacro	integer	number of macros
	imacroid	integer MB_MAXMACPMB	index of the macros belonging to this matrix block
	rnext	Tmpmb	

Tmpmb_access	mapping macros/matblock	
	ra	Tmpmb

7.16.1 Subroutine matrix_getLinePos

Interface:
`matrix_getLinePos(rmatrix, iline, nentries, Ientries)`

Description:
This routine returns the indices of the matrix entries in the array Da on the given line

Input variables:

Name	Type	Rank	Description
rmatrix	Tmatrix		matrix object
iline	integer		index of the matrix line (corresponds to node number)

Output variables:

Name	Type	Rank	Description
nentries	integer		number of matrix entries
Ientries	integer	:	array with matrix entry offsets

7.16.2 Subroutine matrixblock_init

Interface:
`matrixblock_init(rblock, q, nmat)`

Description:
This routine initializes the matrix block.

Input variables:

Name	Type	Rank	Description
q	integer		matrix block index
nmat	integer		number of matrices

Output variables:

Name	Type	Rank	Description
rblock	t_matrixBlock_access		matrixblock to init

7.16.3 Subroutine matrixblock_reinit

Interface:
matrixblock_reinit(rblock, q, nmat)

Description:
This routine reinitializes the matrix block.

Input variables:

Name	Type	Rank	Description
q	integer		discretization index
nmat	integer		number of matrices

Output variables:

Name	Type	Rank	Description
rblock	t_matrixBlock_access		matrixblock to init

7.17 Module parallelblock

Purpose: (yet to come)

Type definitions:

Type name	component name	Type	Rank	Purpose
Tloadbal	load balancing info structure			
	idx_xadj	integer		
	idx_adj	integer		
	lpartxadj	integer		
	lpartadj	integer		
	lpartload	integer		
	lpartload1	integer		
	lpart	integer		
	lpart1	integer		

	lpartcount	integer		
	lactload	integer		
	nparts	integer		
	lcx	integer		
	lcy	integer		
	lcidx	integer		
	lcx1	integer		
	lcy1	integer		
	lcidx1	integer		

t_parBlock description of parallel block

	nedgesSD	integer		global number of macro edges on grid
	IedgeInfoSD	integer	:, :	global edge info (node1, node2, status)
	rparBlockCoarse	t_parBlock		pointer to parallel block on coarsest level
	nmacros	integer		number of macros in parallel block
	nmacroVertSD	integer		global number of macro vertices
	nmacroVert	integer		number of macro vertices in parallel block
	nmatBlocks	integer		number of matrix blocks in parallel block
	nmaxMgLevel	integer		maximum multi grid level
	nlevelsFineGrid	integer		number of different levels on finest grid
	ioutputLevel	integer		level of output
	ilastsolution	integer		???
	ilastmatrix	integer		???
	RmacroList	t_macroAccess	dim(:)	access pointer to macro list
	rmacroListRoot	t_macro		pointer of root of macro list
	RmacroListGround	t_macro	dim(:)	static storage for macro list
	RmacroBaseList	t_macroBase	dim(:, :)	access pointer to macro base list

RmacroBaseListRoot	t_macroLevDataAccess	dim(:, :)	root pointer of makro base list
RmacroBaseListGround	t_macroLevData	dim(:, :)	static storage for makro base list
RmatBlockBaseList	t_matrixBlockbase	dim(:, :)	access pointer to matrix block list
RmatBlockBaseListRoot	t_matrixBlock_access	dim(:, :)	pointer to root of matrix block list
RmatBlockBaseListGround	t_matrixBlock	dim(:, :)	static storage for matrix block list
RmacroGrid	Tmakrogrid	dim(:)	access pointer to macro grid structure list
RmacroGridRoot	Tgrid_access	dim(:)	pointer to root of grid structure list
RmacroGridGround	Tgrid	dim(:, :)	static storage for grid stucture list
RmacroInMatBlock	Tmpmb_access	dim(:)	access pointer to macros in matrixblock structure
rmacroInMatBlockRoot	Tmpmb		pointer to root of macros in matrixblock structure
rmacroInMatBlockGround	Tmpmb		static storage for matrixblock structure
H_Drhs	integer	:, :	access handle to RHS, length: (number of discretisations, maximum number of bilinear forms in discretisation)
rboundary	Tboundary		boundary structure
rbcList	t_bcList		boundary condition list
rloadBal	Tloadbal		load balancing structure
inpp	integer	2, PAR_MAXPP	???
idxnpp	integer		???
ipborder	integer	2, 2*PAR_MAXPP	???
iexcycles	integer		???
h_Iheader	integer		handle for communication definition vector for header
h_Iintern	integer		handle for communication definition vector for complete internal communication

	h_Iextern	integer		handle for communication definition vector for external edge communication
	h_IexternDiag	integer		handle for communication definition vector for external diagonal communication
	h_Dbuffer	integer		handle for communication buffer
	rmasterData	t_masterDataParallelBlock		
	h_Ihnc	integer		handles for hanging node correction
	h_Dhnc	integer		
	h_ilvecsave	integer	:, :	workspace for edge communication, holds the original values

t_masterDataParallelBlock

	rtimeStamp	type(t_stat)		global time measurement
	rdiscr	Tdiscretization		discretisation structure
	rbs	Tbordersave		
	ImatBlockIdx	integer	PAR_MAXPP	array with matrix indices
	rmasterParBlock	t_parBlock		
	bfakrec	logical		
	bmasterFinish	logical		
	nmacros	integer		
	nmacroVerts	integer		
	ipb	integer		
	h_IadjParBlock	integer		
	bgcs	logical		if global coarse grid solver applied or not
	ipborder	integer	:	
	iexcycles	integer	2	
	idxnpp	integer	2, PAR_MAXPP	idxnpp(1,j) : number of different parallel blocks which are edge-adjacent to parblock j idxnpp(2,j) : " " " diagonal-adjecent "

IadjParBlock	integer	:	pointer array of length 2*nparBlock*PAR_MAXPP, Per parallelblock the indices of edge-adjacent parallelblocks are stored in the first half of this array, and the indices of vertex-diagonal parallelblocks in the second half: [en of pb1 — ... — en of last pb —— dn of pb1 — ... — dn of last pb]. en:=edge neighbours, dn:= diagonal neighbours
NmacrosPerPb	integer	PAR_MAXPP	number of macros per parallel block
h_ipborder,nparBlock	integer		integer :: ioffset !offset for DYNADAP-case muss vielleicht global
RtimeStart	type(t_stat)	PAR_MAXPP	
dlhmin,dlar	real(DP)		
DconvRateMin	real(DP)		array with minimal convergence rate per parallel block (minimal means best conv rate)
DconvRateMax	real(DP)		array with maximal convergence rate per parallel block (maximal means worst conv rate)
DconvRateAvg	real(DP)		array with average convergence rate per parallel block
smaster	c (len = 256)		contains name of gmv output file
Rstat	type(t_stat)	PAR_MAXPP	statistics per parallelblock
RstatCurr	type(t_stat)	PAR_MAXPP	statistics of current iteration per parallelblock
ImacroRefIdx	integer	:	array with indices of all macros to be refined

h_ImacroRefIdx	integer		
inumberMacroRef	integer		
imodus	integer		
h_DmacroGlobContrib	integer		
h_NmacroPerParBlock	integer		
h_InumMacroRef	integer		
h_IparBlockIdx	integer		
h_ImacroIdx	integer		
h_H_IivertIdx, h_IlevGlob	integer		
i	integer		
isyncs	integer		
imasterServiceId	integer		switch variable to the master services
partload1	integer	:	********** variables not commented or still to be renamed **********

7.17.1 Subroutine parblock_initFictitiousBounds

Interface:

`parblock_initFictitiousBounds(rparBlock)`

Description:
This routine inits the node indices to realise fictitious boundaries. Every time a fictitious boundary is manipulated this routine has to be called.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block structure

7.17.2 Subroutine parblock_filterFictiousBoundary

Interface:

`parblock_filterFictiousBoundary(rparBlock, rmatBlockId, imgLevel, imatBlockIdx, Dx, Dy, Dval)`

Description:
This routine filters the given vectors according the given list of fictitious boundary node indices

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block structure
rmatBlockId	Tdiscrid		blf identifier
imgLevel	integer		multi grid level of the vectors
imatBlockIdx	integer		index of the matrixblock where the vectors come from
Dval	real(DP)		optional set value

Output variables:

Name	Type	Rank	Description
Dx	real(DP)	:	data vector one
Dy	real(DP)	:	optional data vector two

7.17.3 Function parblock_igetNeq

Interface:
parblock_igetNeq(rparBlock,imbidx,imglevel,rdiscrId)

Description:
This routine returns the number of unknowns of the selected matrixblock.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block record
imbidx	integer		index of matrixblock
imglevel	integer		multigrid level
rdiscrId	Tdiscrid		matrix identifier

Result:
integer : number of unknowns of the matrixblock

7.17.4 Subroutine parallelblock_init

Interface:
parallelblock_init(rparBlock, rdiscr)

Description:
This routine initialises the parallelblock structure rparBlock.

Output variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rdiscr	Tdiscretization		discretisation structure

7.17.5 Subroutine parallelblock_setup

Interface:
parallelblock_setup(rparBlock, idesk, isize, nmakro, nvt, nmb, maxmglevel, rdiscr)

Description:
This routine initialises the parallel block rparBlock with the given parameters.

Input variables:

Name	Type	Rank	Description
idesk	integer		count of descriptors
isize	integer		memory size
nmakro	integer		number of macros
nvt	integer		number of vertices
nmb	integer		number of matrix blocks
maxmglevel	integer		maximum multi grid level

Output variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
rdiscr	Tdiscretization		discretisation structure

7.17.6 Subroutine parallelblock_copy

Interface:
parallelblock_copy(rpbs, rpbd, imglevel, rdiscr)

Description:
This routine copies a parallelblock to another.

Input variables:

Name	Type	Rank	Description
rpbs	t_parBlock		source parallelblock structure
imglevel	integer		maximum multi grid level

Output variables:

Name	Type	Rank	Description
rpbd	t_parBlock		destination parallelblock structure
rdiscr	Tdiscretization		discretisation structure

7.17.7 Subroutine getmacro_p_xy

Interface:
getmacro_p_xy(rmasterParBlock, dx, dy, k)

Description:
Ths routine

Input variables:

Name	Type	Rank	Description
rmasterParBlock	t_parBlock		
dx, dy	real(DP)		

Output variables:

Name	Type	Rank	Description
k	integer		

7.18 Module grid

Purpose: This module contains grid handling routines for generating and refining routines. There are two types of grids, direct stored for not uniform and analytic calculated for uniform grids. One grid is assigned to a matrixblock object. The active grid is selected by setting the global variable gr_rwgrid. The communication with the other functions are also handled by global variables gr_.... THE COMMENTS HAVE STILL TO BE IMPROVED AND CORRECTED.

Constant definitions:

Name	Type	Purpose
GR_DIRECT	integer	structure is directly stored
GR_REG	integer	structure calculated in situ, regular grids
GR_ANISO	integer	structure calculated in situ, grid with anisotropic refinement
GR_LOGISO	integer	logical enumeration of nodes, row-wise

GR_LOGIRR	integer	irregular enumeration of nodes, FEAT-like
GRD_REFINEALL	integer	logical enumeration of nodes, row-wise
GRD_REFINEONE	integer	irregular enumeration of nodes
GR_MAXADAPRANGE	integer	maximum refinement range

GR_ANISOREFMODE1	integer	classic mode
GR_ANISOREFMODE2	integer	Susi compatibility mode (s.Kilian2002:56)

Type definitions:

Type name	component name	Type	Rank	Purpose
Tmakrogrid	multi level structure for grids			
	rgridlevel	Tgrid_access	dim(:)	access variable
Tgrid_access	grid structure			
	ra	Tgrid		
Tgrid	grid structure			
	dar	real(DP)		aspect ratio of the grid
	ctype	integer		type of the grid, GR_DIRECT,GR_REG,GR_ANISO
	clogstruct	integer		logical structure of the grid GR_LOGISO,GR_LOGIRR
	lvert	integer		FEAT enumeration fields, handle for vector DVERT
	ladj	integer		FEAT enumeration fields, handle for vector KADJ
	ladjs	integer		integer :: ladj1 = -1 !handle for vector KADJ integer :: ladj2 = -1 !handle for vector KADJadap1 integer :: ladj3 = -1 !handle for vector KADJadap2 integer :: ladj4 = -1 !handle for vector KADJadap3 handle for vector KADJadap

ifnmakros	integer	count of macros on refined levels
h_DvertCoords	integer	handle for vector DvertCoords (vertex coordinates)
lnpr	integer	handle for vector KNPR (vertex status, 0:inner node)
lref	integer	handle for vector KREF fuer Adaptivitaet
lmid	integer	handle for vector KMID
lbs	integer	handle for border status of the according node ! DEPRICATED
dx1,dy1,dx2,dy2	real(DP)	coordinates of macro vertices, GR_REG
dx3,dy3,dx4,dy4	real(DP)	coordinates of macro vertices, GR_REG
nel	integer	number of elements in macro
nvt	integer	number of vertices in macro
nmt	integer	number of midpoints in macro
nvel	integer	number of corners per element
nbct	integer	reserved
nvbd	integer	reserved
nblindnodes	integer	
lblindnodes	integer	
nx,ny	integer	number of nodes in x and y direction
nelx, nely	integer	number of elements in x and y direction
dhax,dhay	real(DP)	stepsizes in x and y direction for the equidistant case
ilhx	integer	handle for stepsize vector in x direction for the non-equidistant case

ilhy	integer		handle for stepsize vector in y direction for the non-equidistant case
imacnodes	integer	4	indices of the macro nodes
imacelem	integer	4	indices of the macro elements
rnext	Tgrid		pointer to next grid structure

Global variable definitions:

Name	Type	Rank	Purpose
gr_rwgrid	Tgrid		working grid
gr_iwve	integer		working vertex per element
gr_iwel	integer		working element
gr_ivert	integer		working node per element indices
gr_iwv	integer		working node
gr_dx	real(DP)		working coordinates, output
gr_dy	real(DP)		working coordinates, output
gr_dxs	real(DP)	4	complete, working coordinates, output
gr_dys	real(DP)	4	complete, working coordinates, output
gr_iverts	integer	4	complete indices of a element
gr_anisoMode	integer		anisotropic refinement mode

7.18.1 Subroutine grid_setAnisotropicRefMode

Interface:
grid_setAnisotropicRefMode(cmode)

Description:
This routine sets the mode for the anisotropic macro refinement

Input variables:

Name	Type	Rank	Description
cmode	integer		mode to be set

7.18.2 Subroutine grid_calcInlinedNodes

Interface:
grid_calcInlinedNodes(rboundary, rgrid, h_Inodes)

Description:
This routine calculates for all fictitious boundaries the indices of the nodes which have to set to zero of the given grid object.

Input variables:

Name	Type	Rank	Description
`rboundary`	`type(Tboundary)`		boundary structure
`rgrid`	`type(Tgrid)`		grid structure

Output variables:

Name	Type	Rank	Description
`Inodes`	`integer`	:	node indices of fictitious boundaries

7.18.3 Subroutine grid_freeStorage

Interface:
`grid_freeStorage()`

Description:
This subroutine frees the storage of the vectors in Tgrid. It has to be called everytime the grid has changed (e.g. due to adaptive refinement).

7.18.4 Subroutine grid_freeStorageMaster

Interface:
`grid_freeStorageMaster(rgrid)`

Description:
This internal subroutine is called by the master via the routine grid_freeStorage(). It releases all vectors of the grid structure.

Input variables:

Name	Type	Rank	Description
`rgrid`	`Tgrid`		grid structure

7.18.5 Subroutine getVertCoords

Interface:
`getVertCoords(DvertCoords, ivertIdx, dx, dy)`

Description:
This subroutine returns the coordinates (dx,dy) for a given vertex with index ivertIdx.

Input variables:

Name	Type	Rank	Description
Dvertcoords	real(DP)	:	array with vertex coordinates
ivertIdx	integer		index of the node

Output variables:

Name	Type	Rank	Description
dx, dy	real(DP)		node coordinates

7.18.6 Subroutine setDvertCoords

Interface:
setDvertCoords(DvertCoords, ivertIdx, dx, dy)

Description:
This routine sets the coordinates (dx,dy) for a given node with index ivertIdx.

Input variables:

Name	Type	Rank	Description
DvertCoords	real(DP)	:	array with coordinates
ivertIdx	integer		index of the node
dx,dy	real(DP)		node coordinates

7.18.7 Subroutine getivert

Interface:
getivert(IvertIdx, ielemIdx, ivertIdxLoc, ivertIdxGlob)

Description:
This routine returns for the given element with index ielemIdx and and the local vertex index ivertIdxLoc the global node index ivertIdxGlob.

Input variables:

Name	Type	Rank	Description
IvertIdx	integer	:	array with indices of the vertices
ielemIdx	integer		element index
ivertIdxLoc	integer		local vertices index

Output variables:

Name	Type	Rank	Description
ivertIdxGlob	integer		global index of the node

7.18.8 Subroutine setivert

Interface:

`setivert(IvertIdx, ielemIdx, ivertIdxLoc, ivertIdxGlob)`

Description:
This routine sets for the given element with index ielemIdx and and the local vertex index ivertIdxLoc the global node index ivertIdxGlob.

Input variables:

Name	Type	Rank	Description
`IvertIdx`	`integer`	:	array with vertices
`ielemIdx`	`integer`		element index
`ivertIdxLoc`	`integer`		vertices index
`ivertIdxGlob`	`integer`		global index of the node

7.18.9 Subroutine getiadj

Interface:

`getiadj(iadj, ielemIdx, iedge, iadjElemIdx)`

Description:
This routine returns for the given element with index ielemIdx and and the local edge index iedge the index of the element which has edge iedge in common with element ielemIdx.

Input variables:

Name	Type	Rank	Description
`iadj`	`integer`	:	array with neighbour information
`ielemIdx`	`integer`		element index
`iedge`	`integer`		edge index

Output variables:

Name	Type	Rank	Description
`iadjElemIdx`	`integer`		element index of the specified neighbour

7.18.10 Subroutine setiadj

Interface:

`setiadj(iadj, ielIdx, iedge, iadjElemIdx)`

Description:
This routine sets for the given element with index ielIdx and and the local edge with index iedge the index of the element which has edge iedge in common with element ielIdx.

Input variables:

Name	Type	Rank	Description
`iadj`	`integer`	:	array with neighbour information
`ielIdx`	`integer`		element index
`iedge`	`integer`		edge index
`iadjElemIdx`	`integer`		element index of the specified neighbour

7.18.11 Subroutine grid_replace

Interface:

`grid_replace(rgrid, rboundary)`

Description:
This routine replaces the parameter description of the boundary nodes by their cartesian coordinates in the vector DvertCoords.

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary object

Input/Output variables:

Name	Type	Rank	Description
`rgrid`	`Tgrid`		grid object

7.18.12 Subroutine grid_refindfak

Interface:

`grid_refindfak(rboundary, iboundIdx, duzfak, idxzfak, dparStart, dparEnd, bflag)`

Description:
This routine changes adaptively the refinement factors to archieve a equidistant refinement on boundary edges with strong differences in the parametrization.

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary object
`idxzfak`	`integer`		number of refinement factors
`iboundIdx`	`integer`		boundary index
`bflag`	`logical`		
`dparStart,dparEnd`	`real(DP)`		start and end parameter of boundary macro edge

Input/Output variables:

Name	Type	Rank	Description
duzfak	real(DP)		refinement factors

7.18.13 Subroutine grid_setreg

Interface:

grid_setreg(ilevelAbs, rmacrou, rgrid, rboundary, cdynAdap)

Description:
This routine initializes the grid object rgrid according to the given macro and macrolevel objects with a regular grid structure.

Input variables:

Name	Type	Rank	Description
ilevelAbs	integer		absolute level of refinement
rmacrou	t_macroAccess		macro object
rgrid	Tgrid		grid object
rboundary	Tboundary		boundary object
cdynAdap	integer		mode of adaptive refinement, 0: no level change, 1: refine, -1:coarsen

7.18.14 Subroutine getCartCoords

Interface:

getCartCoords(DvertCoords, IvertStatus, rboundary, ivertIdx, dx, dy)

Description:
This routine returns for the specified vertex index ivertIdx the cartesian coordinates (dx,dy). For boundary nodes the calculation is automatically done.

Input variables:

Name	Type	Rank	Description
rboundary	Tboundary		boundary object
DvertCoords	real(DP)	:	array of node coordinates
IvertStatus	integer	:	array of node stati (0:inner point)
ivertIdx	integer		index of the specified node

Output variables:

Name	Type	Rank	Description
dx, dy	real(DP)		cartesian coordinates

7.18.15 Subroutine grid_refine

Interface:

`grid_refine(iref, rmakrolist, rmakrobaselist, rgrid, rboundary, imacs, imglevel, cregRefine, tlb_nel, cmode)`

Description:
This routine refines the given grid for the given times. The grid has to be stored in direct format.

Input variables:

Name	Type	Rank	Description
`iref`	`integer`		number of refinements
`imglevel`	`integer`		current multigrid level
`imacs`	`integer`	:	index of defining macros
`rmakrobaselist`	`t_macroBase`		dimension of temporary aux vector integer :: idim macro base list
`rmakrolist`	`t_macroAccess`		macro base list
`rboundary`	`Tboundary`		boundary object
`cmode`	integer		refinement mode GRD_REFINEONE,GRD_REFINEALL HACK

Input/Output variables:

Name	Type	Rank	Description
`rgrid`	`Tgrid`		grid object

7.18.16 Subroutine grid_shrinkgrid

Interface:

`grid_shrinkgrid(nmakro, rmakrolist, itmp)`

Description:
This routine shrinks the global grid to the grid resides on the current parallel block.

Input variables:

Name	Type	Rank	Description
`nmakro`	`integer`		number of macros of the global grid
`itmp`	`integer`	:	

Input/Output variables:

Name	Type	Rank	Description
`rmakrolist`	`t_macroAccess`		list of macros

7.18.17 Subroutine grid_getlocalgridsize

Interface:

`grid_getlocalgridsize(ngnvt, nlmakro, rmakrolist, nlnvt, itmp)`

Description:
This routine returns the local mapping according to global nodes.

Input variables:

Name	Type	Rank	Description
`ngnvt`	`integer`		global number of nodes
`nlmakro`	`integer`		local number of macros
`rmakrolist`	`t_macroAccess`		makro list local

Output variables:

Name	Type	Rank	Description
`nlnvt`	`integer`		local number of nodes
`itmp`	`integer`	:	mapping vector

7.18.18 Subroutine grid_calcAspRat

Interface:

`grid_calcAspRat(rgrid)`

Description:
This routine computes the aspect ratio of the given grid. The ratio is stored in the structure.

Input/Output variables:

Name	Type	Rank	Description
`rgrid`	`Tgrid`		grid structure

7.18.19 Subroutine grid_setdirect

Interface:

`grid_setdirect(irefLevCoarse, imglevel, nmacroVert, nmacrosInPb, RmacroList, RmacroBaseList, rgrid, rboundary, ndiscr, cmode, imacs, boversize)`

Description:
This routine initialises the direct grid structure for a given macro list and refines this structures with a certain refinement parameter.

Input variables:

Name	Type	Rank	Description
irefLevCoarse	integer		refinement level of coarse grid
ndiscr	integer		number of discretisations
cmode	integer		if cmode = GRD_REFINALL: uniform refinement
imacs	integer		
imglevel	integer		multigrid level
nmacroVert	integer		number of macro vertices
nmacrosInPb	integer		number of macros in the parallel block
RmacroList	t_macroAccess		macro list
RmacroBaseList	t_macroBase		macro base list
rboundary	Tboundary		description of the domain boundary
boversize	logical		if flag is set, additional lmapping storage space is allocated (Jens mode)

Output variables:

Name	Type	Rank	Description
rgrid	Tgrid		grid structure

7.18.20 Subroutine grid_setdirect_select

Interface:

grid_setdirect_select(iref, imglevel, rmakrolist, RmacroBaseList, rmacrogrid, rboundary, ndiscr, imacs, imacidx, cmode)

Description:
This routine initializes the direct grid structure for a given makro list and refines this structures with a certain refinement parameter.

Input variables:

Name	Type	Rank	Description
iref	integer		refinement level
ndiscr	integer		
imglevel	integer		multigrid level of this grid
nvt	integer		number of vertices of the coarse grid
nmakro	integer		number of macros of the coarse grid
rmakrolist	t_macroAccess		macro list
RmacroBaseList	t_macroBase		macro base list
rboundary	Tboundary		description of the domain boundary
imacidx	integer		
imacs	integer	:	
cmode	integer		

Output variables:

Name	Type	Rank	Description
rmacrogrid	Tgrid		grid structure

7.18.21 Subroutine grid_getivert

Interface:
grid_getivert()

Description:
This routine returns the specified vertex id of a selected element.

7.18.22 Subroutine grid_getallivert

Interface:
grid_getallivert()

Description:
This routine returns all vertices of a certain element.

7.18.23 Subroutine grid_allgetcorvg

Interface:
grid_allgetcorvg()

Description:
This routine returns the cartesian coordinates of a node.

7.18.24 Subroutine grid_getcorvg

Interface:

grid_getcorvg()

Description:
This routine returns the cartesian coordinates of a node.

7.18.25 Subroutine grid_outputavsgmv

Interface:

grid_outputavsgmv(sfilename, rgrid, rbound, iflag)

Description:
This routine produces an AVS or GMV file of the given grid.

Input variables:

Name	Type	Rank	Description
sfilename	c (*)		AVS/GMV file name
rgrid	Tgrid		grid to be processed
rbound	Tboundary		boundary of the grid
iflag	integer		=0 only cartesian coordinates, =1 boundary nodes with parametrization

7.18.26 Subroutine grid_chModDirect

Interface:

grid_chModDirect(Rmacrogrid, imacroIdx, nmaxMgLevel)

Description:
This routine is to change the way how the coordinates are obtained in a macro from computation in situ to the access of a local array. This is done for all levels.

Input variables:

Name	Type	Rank	Description
RmacroGrid	Tmakrogrid	dim(:)	structure with the grid information of the macro
imacroIdx	integer		local index of macro
nmaxmglevel	integer		maximum multigrid level

7.18.27 Subroutine deconstruct_Tgrid

Interface:

deconstruct_Tgrid(grid)

Description:
This routine clears all references of a given Tgrid data structure

Input variables:

Name	Type	Rank	Description
grid	type(Tgrid)		grid to be deconstructed

7.18.28 Subroutine grid_getElementCoords

Interface:

grid_getElementCoords(ielIdx, DvertCoord)

Description:
This subroutine computes the coordinates of the vertices of the element ielIdx and writes them into DvertCoord. warning!!!!! gr_rwgrid must be set outside

Input variables:

Name	Type	Rank	Description
ielIdx	integer		element index in macro

Output variables:

Name	Type	Rank	Description
DvertCoord	real(DP)	2, MAC_EPM	array with vertex coordinates

7.19 Module fboundary

Purpose: In this module, several routines for modifying fictitious boundaries are gathered.

7.19.1 Subroutine fboundary_moveFictitious

Interface:

fboundary_moveFictitious(rparBlock, rboundary, dmx, dmy, ibdry)

Description:
This routine moves a given fictitious boundary

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rboundary	Tboundary		boundary structure
dmx	real(DP)		move offset in x direction
dmy	real(DP)		move offset in y direction
ibdry	integer		number of the boundary

7.19.2 Subroutine fboundary_rotateFictitious

Interface:

`fboundary_rotateFictitious(rparBlock, rboundary, dmx, dmy, dualpha, ibdry)`

Description:
This routine moves a given fictitious boundary

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`rboundary`	`Tboundary`		boundary structure
`dmx`	`real(DP)`		x rotation center
`dmy`	`real(DP)`		y rotation center
`dalpha`	`real(DP)`		rotation angle
`ibdry`	`integer`		number of the boundary

7.19.3 Subroutine fboundary_getPerimeter

Interface:

`fboundary_getPerimeter(rboundary, ibdry, dmx, dmy, dpxmin, dpymin, dpxmax, dpymax)`

Description:
This routine moves a given fictitious boundary

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary structure
`ibdry`	`integer`		number of the boundary

7.19.4 Subroutine fboundary_addFictitiousCircle

Interface:

`fboundary_addFictitiousCircle(rparBlock, rboundary, dmx, dmy, dmr, ibdry)`

Description:
This routine adds a fictitous boundary with shape circle to the boundary

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rboundary	Tboundary		boundary structure
dmx	real(DP)		x coordinate of circle midpoint
dmy	real(DP)		y coordinate of circle midpoint
dmr	real(DP)		radius of circle

Output variables:

Name	Type	Rank	Description
ibdry	integer		index of created boundary

7.19.5 Subroutine fboundary_addFictiLineSegs

Interface:
fboundary_addFictiLineSegs(rparBlock, rboundary, nlineseg, Dsegment, ibdry)

Description:
This routine adds a fictitous boundary with shape circle to the boundary

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rboundary	Tboundary		boundary structure
Dsegment	real(DP)		coordinate and grades of line segments
nlineseg	integer		number of lines

Output variables:

Name	Type	Rank	Description
ibdry	integer		index of created boundary

7.20 Module assembly

Purpose: This module contains routines for the assembly of FEM matrices and right hand sides. Furthermore, it contains routines for the setting of boundary conditions and values. Moreover, there are routines for matrixblock object scaling, multiplication and adding.

Constant definitions:

Name	Type	Purpose
Purpose: general flags		
AS_INITASS	integer	reserve and assembly
AS_INITONLY	integer	reserve only
AS_SETMATRIX	integer	Flag, set matrix bilinear form
AS_INITMATRIX	integer	Flag, reserve matrix storage
AS_SETRHS_BOTH	integer	Flag, set RHS for volume and boundary forces
AS_SETRHS_VOL	integer	Flag: set RHS for volume forces
AS_SETRHS_BOUND	integer	Flag, set RHS for boundary forces
AS_CLEARMATRIX	integer	Flag, clear matrix
AS_SHOWMATRIX	integer	Flag, clear matrix
AS_CLEARRHS	integer	Flag, clear matrix
AS_ALL_BC	integer	Compute Neumann contributions for all boundary conditions
AS_SBGLOBAL	integer	Flag for full matrix entries on macro edges, scarc level subdomain
AS_SBNEUMANN	integer	Flag for default matrix entries on macro edges
AS_SBLOCAL	integer	Flag for half matrix entries on macro edges, scarc level matrix block
AS_SBSEMILOCAL	integer	Flag for half or full matrix entries on macro edges, scarc level parallel block
AS_BORDERPRESET	integer	???
AS_ASSEMBLY_BLF	integer	traditional assembly mode using a BLF definition
AS_ASSEMBLY_ELEM	integer	element-wise assembly mode (common in structural mechanics)
Purpose: flags for treatment of boundary conditions		
AS_BC_NONE	integer	Flag, set no boundary condition in matrix (restore original matrix)
AS_BC_DIRICHLET	integer	Flag for telling assembly_applyBoundCond to treat Dirichlet bc
AS_BC_NEUMANN	integer	Flag for telling assembly_applyBoundCond to treat Neumann bc

AS_BC_NO_DIRICH_IN_MATRIX	integer	Flag for telling assembly_applyBoundCond not to modify the matrix due to Dirichlet bc
AS_BC_NO_DIRICH_IN_RHS	integer	Flag for telling assembly_applyBoundCond not to modify the RHS due to Dirichlet bc
AS_BC_DIRICH_IN_RHS_ZERO	integer	Flag for telling assembly_applyBoundCond to set the entries of the RHS corresponding to Dirichlet nodes to zero (useful for defect correction or Newton Raphson method)
AS_BC_MAINDIAG_ZERO	integer	Flag for telling assembly_applyBoundCond to set the main diagonal values corresponding to Dirichlet nodes to zero (necessary for a system matrix consisting of several blocks where off-diagonal blocks need complete zero "Dirichlet lines"
AS_BC_DO_NOT_RESTORE_RHS	integer	Flag for telling assembly_applyBoundCond *not* to restore RHS values on macro borders. While restoring the values makes sense in some situations it is annoying when the RHS has been changed (e.g. in a time-dependent computation) and then the wrong values are restored. The default behaviour is to restore the default values which have been saved while assembling the RHS vector or with help of the routine as_saveCurrentRHS(...).

Purpose: constants for mass matrix scaling

AS_DO_NOT_INVERT_MASS	integer	flag that a vector shall be scaled with the mass matrix itself (simple matrix vector multiplication)
AS_INVERT_MASS	integer	flag that a vector shall be scaled with the inverse mass matrix

Purpose: constants for grid adaptation

DYNADAP_MODE_SELF	integer	self defined adaptation process
DYNADAP_MODE_DEMO1	integer	grid adaptation demo 1
DYNADAP_MODE_DEMO2	integer	grid adaptation demo 2

Purpose: constants for matrix dumping

AS_DADUMPMODE_NONE	integer	no matrix dumping, calculation of matrix entries
AS_DADUMPMODE_WRITE	integer	matrix entries calculation, then writing
AS_DADUMPMODE_READ	integer	matrix entries reading, no calculation
AS_DADUMPFORMAT_BIN	integer	matrix entries dump, binary format
AS_DADUMPFORMAT_ASCII	integer	matrix entries dump, ascii format

7.20.1 Subroutine as_setDumpMatrixParameter

Interface:

`as_setDumpMatrixParameter(cmode, cformat, sfilename)`

Description:
This routine sets the control parameters for matrix dumping. The call of this routine has to be done before call of as_assMatrix_band.

Input variables:

Name	Type	Rank	Description
cmode	integer		mode for matrix dumping: AS_DADUMPMODE_NONE: no matrix dumping, calculation of matrix entries AS_DADUMPMODE_WRITE: matrix entries calculation, then writing (inside as_assMatrix_band) AS_DADUMPMODE_READ: matrix entries reading, no calculation (inside as_assMatrix_band)
cformat	integer		format for matrix dumping: AS_DADUMPFORMAT_ASCII: dump in ascii format AS_DADUMPFORMAT_BIN: dump in bin format
sfilename	c(*)		file name template for the matrix entries files

7.20.2 Subroutine as_dumpMatrix

Interface:

`as_dumpMatrix(rparBlock, rdiscrid, sname, h_exactSol, h_compSol)`

Description:
This routine dumps the entries of the given matrix to files.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		Parallel block object
rdiscrid	type(Tdiscrid)		matrix identifier
sname	c(*)		file name
h_exactSol	integer		vector with exact solution
h_compSol	integer		vector with computed solution

7.20.3 Subroutine as_dynRefine

Interface:

as_dynRefine(rparBlock, rdiscr, rbs, cmode, ncycle, istep, nmacroRef, ImacroRefList, nmacroRefActual, ImacroRefIdx, nmaxBoundCond)

Description:
This routine updates the grid level information in the grid objects for dynamic grid refinement. After calling this routine all matrix and solver objects have to be updated.

Input variables:

Name	Type	Rank	Description
cmode	integer		mode selection (DYNADAP_MODE_SELF, DYNADAP_MODE_DEMO1, DYNADAP_MODE_DEMO2)
ncycle	integer		parameter for demo refinement, number of cycles
istep	integer		parameter for demo refinement, current cycle
nmacroRef	integer		number of macros to be refined on current parallel block
ImacroRefList	integer	:	contains list with global number of the macros that have really been locally refined. This may differ from the list of the macros to be refined e.g. because of macros which were refined to ensure the level difference between two edge-adjecent macros to be at most 1.
nmacroRefActual	integer		number of the macros really refinened or coarsened
ImacroRefIdx	integer	:	local indices of macros really refined (.gt. 0 refined, .lt. 0 derefined)
nmaxBoundCond	integer		maximum number of boundary conditions
imacroIdxInPb	integer		local position of macro in parallel block
rdiscr	Tdiscretization		discretisation structure
rbs	Tbordersave		DEPR????

Input/Output variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object

7.20.4 Subroutine as_parBlockReAssemble

Interface:

as_parBlockReAssemble(rparBlock, imgLevel, rdiscr, rbs, imatBlockIdx, ilevDiff)

Description:
This routine reassembles the matrix structures after dynamic adaptive refinement. For the right hand side, only volume forces are computed and saved as default. If one wants the right hand side with Neumann contributions as default rhs, one has to call as_set_rhs_bound(..., AS_SAVE_RHS) after the dynamic refinement and then as_applyBoundCond(...AS_DIRICHLET...).

Input variables:

Name	Type	Rank	Description
imgLevel	integer		maximum multigrid level
rdiscr	Tdiscretization		discretisation of the domain
rbs	Tbordersave		bordersave object
imatBlockIdx	integer		index of the matrixblock to be refined
ilevDiff	integer		difference of old and new level of refinement

Input/Output variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object

7.20.5 Subroutine as_initParBlockCoarse

Interface:

as_initParBlockCoarse(rparBlock, rdiscr)

Description:
This routine initializes the data structures for the parallelblock coarse grid solver.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rdiscr	Tdiscretization		discretisation object

7.20.6 Function as_igetHLayer

Interface:

`as_igetHLayer(ias)`

Description:
This function returns to a given boundary set flag (AS_SBxxx) the corresponding hierachical layer.

Input variables:

Name	Type	Rank	Description
`ias`	`integer`		boundary set flag

Result:

integer : hierachical layer

7.20.7 Function as_igetBorderFlag

Interface:

`as_igetBorderFlag(ihlayer)`

Description:
This function returns to a given hierachical layer the corresponding boundary set flag (AS_SBxxx).

Input variables:

Name	Type	Rank	Description
`ihlayer`	`integer`		hierachical layer

Result:

integer : boundary set flag

7.20.8 Subroutine as_assRhs

Interface:

`as_assRhs(rparBlock, rmacroIter, rmgLevIter, rdiscrInfo, ccubType, celType, plocGlobMapping, ndofLoc, prhs, iblfIdx, cmode, cdynAdap, csaveRhs)`

Description:
This routine assembles the right hand side of the given matrix objects of the same discretisation.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rmacroIter	type(t_iterator)		matrixblock index
rmgLevIter	type(t_iterator)		multigrid level
ccubType	integer		selected cubature formula
celType	integer		given FEM basic function from the element module include "pele.h"
ndofLoc	integer		subroutine for computing the mapping between local and global DOFs number of local degrees of freedom
iblfIdx	integer		function for calculating entries of the RHS selected matrices of the same discretisation, -1 means all matrices
cmode	integer		mode of assembly, AS_INITASS, AS_INITONLY
cdynAdap	integer		dynamic refinement control
csaveRhs	integer		if RHS should be saved

Input/Output variables:

Name	Type	Rank	Description
rdiscrInfo	type(Tdiscrinfo)		matrixblock objects of the same discretisation

7.20.9 Subroutine as_assRhsNeumann

Interface:

as_assRhsNeumann(rparBlock, rmbIter, rmglevIter, rdiscrInfo, iblfIdx, iboundCond, pboundaryValue, csaveRhs)

Description:
This routine calculates the Neumann contributions and adds it to the RHS of the given matrix objects of the same discretisation.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rmbIter	t_iterator		matrixblock index
rmglevIter	t_iterator		multigrid level
iblfIdx	integer		selected bilinear form of the same discretisation, -1 means all bilinear forms
iboundCond	integer		index of the boundary condition (-1 means all)
csaveRhs	integer		user defined function for calculations on Neumann boundary if RHS should be saved

Input/Output variables:

Name	Type	Rank	Description
rdiscrInfo	type(Tdiscrinfo)		matrixblock objects of the same discretisation

7.20.10 Subroutine as_saveCurrentRhs

Interface:

as_saveCurrentRhs(rparBlock, rdiscrId)

Description:
This routine saves the current RHS which is attached to the matrix rdiscrId as default RHS. This means that in the routine as_applyBoundCond(...) the border values of this current RHS are restored! This routine is similar to hl_attachRhs. But while the latter is meant to attach a *new* RHS vector, this one here is meant to re-attach the already existing RHS.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rdiscrId	type(TdiscrId)		matrix object

7.20.11 Subroutine as_assMatrix

Interface:

as_assMatrix(rparBlock, imatBlock, imgLevel, rdiscrinfo, ccubType, celType, ndofLoc, pcoeff, absym, iblfIdx, cmode, cdynAdap)

Description:
This routine assembles the FEM matrices of the given matrix objects of the same discretisation.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imatBlock	integer		matrixblock index
imgLevel	integer		multigrid level
ccubType	integer		selected cubature formula
celType	integer		FEM function from the element module include "pele.h"
iblfIdx	integer		subroutine for computing the connection between local and global DOFs include "plocGlobMapping.h" selected BLF of the same discretisation, -1 means all BLFs
absym	logical		flag if operator is symmetric
cmode	integer		assembly mode AS_INITASS, AS_INITONLY
cdynAdap	integer		type of adaptive level change 0: no change, 1: refine, -1: coarsen
ndofLoc	integer		

Input/Output variables:

Name	Type	Rank	Description
rdiscrInfo	type(Tdiscrinfo)		matrixblock objects of the same discretisation

7.20.12 Subroutine as_exchangeMatBlock

Interface:

as_exchangeMatBlock(rparBlock, imgLevel, rdiscrId)

Description:
This routine exchanges the matrix entries and stores the received entries in internal structures.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imgLevel	integer		multigrid level

Input/Output variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		matrixblock object

7.20.13 Subroutine as_matBlockLinComb

Interface:

`as_matBlockLinComb(rparBlock, imgLevel, dcoeff1, rdiscrIdSource, dcoeff2, rdiscrIdDest, csetrhs)`

Description:
This function calculates the linear combination $y = c1 * x + c2 * y$ of two matrixblock objects. After calling, the user has to set the boundary conditions and to init the internal data structures. If rdiscrIdDest shall use the boundary conditions of rdiscrIdSource, this has to be set manually since this is not always wanted and thus cannot be the default behaviour. rdiscrIdDest%rform%ibc = rdiscrIdSource%rform%ibc rdiscrIdDest%rform%sbc = rdiscrIdSource%rform%sbc The destination matrix will have full entries (AS_SBGLOBAL) at the end due to the call of as_initborder.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imgLevel`	`integer`		multigrid level
`rdiscrIdSource`	`Tdiscrid`		matrixblock object 1
`dcoeff1`	`real(DP)`		coefficient 1
`dcoeff2`	`real(DP)`		coefficient 2
`csetrhs`	integer		if YES then modify RHS too

Input/Output variables:

Name	Type	Rank	Description
`rdiscrIdDest`	`Tdiscrid`		matrixblock object 2

7.20.14 Subroutine as_matBlockCopy

Interface:

`as_matBlockCopy(rparBlock, imgLevel, rdiscrIdSource, rdiscrIdDest, csetrhs)`

Description:
This function copies the first matrixblock object to the second matrixblock object. After calling, the user has to set the boundary conditions and to init the internal data structures. If rdiscrIdDest shall use the boundary conditions of rdiscrIdSource, this has to be set manually since this is not always wanted and thus cannot be the default behaviour. rdiscrIdDest%rform%ibc = rdiscrIdSource%rform%ibc rdiscrIdDest%rform%sbc = rdiscrIdSource%rform%sbc

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imgLevel`	`integer`		multigrid level
`rdiscrIdSource`	`Tdiscrid`		matrixblock object 1
`csetrhs`	integer		if YES, then modify rhs too

Input/Output variables:

Name	Type	Rank	Description
`rdiscrIdDest`	`Tdiscrid`		matrixblock object 2

7.20.15 Subroutine as_matBlockScaledCopy

Interface:

`as_matBlockScaledCopy(rparBlock, imgLevel, rdiscrIdSource, dcoeff, rdiscrIdDest, csetrhs)`

Description:
This function stores a scaled copy of the first matrixblock object to the second matrixblock object. After calling, the user has to set the boundary conditions and to init the internal data structures. If rdiscrIdDest shall use the boundary conditions of rdiscrIdSource, this has to be set manually since this is not always wanted and thus cannot be the default behaviour. rdiscrIdDest%rform%ibc = rdiscrIdSource%rform%ibc rdiscrIdDest%rform%sbc = rdiscrIdSource%rform%sbc The destination matrix will have full entries (AS_SBGLOBAL) at the end due to the call of as_initborder.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imgLevel`	`integer`		multigrid level
`rdiscrIdSource`	`Tdiscrid`		matrixblock object 1
`dcoeff`	`real(DP)`		scaling factor
`csetrhs`	integer		if YES, then modify rhs too

Input/Output variables:

Name	Type	Rank	Description
`rdiscrIdDest`	`Tdiscrid`		matrixblock object 2

7.20.16 Subroutine as_matBlockScale

Interface:

`as_matBlockScale(rparBlock, imgLevel, rdiscrId, dcoeff, csetrhs)`

Description:
This function scales the matrix rdiscrId with the factor dcoeff. If csetrhs is .TRUE. then also the RHS is scaled. The destination matrix will have full entries (AS_SBGLOBAL) at the end due to the call of as_initborder.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imgLevel	integer		multigrid level
dcoeff	real(DP)		scaling factor
csetrhs	integer		if YES then modify RHS too

Input/Output variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		matrixblock object

7.20.17 Subroutine as_scaleLumpMass

Interface:

as_scaleLumpMass(rparBlock, imgLevel, chLayer, rdiscrId, ivaridx, cinverse, rstat)

Description:
This function scales the given vector with the given mass matrix. If cinverse is AS_INVERT_MASS, then the vector is scaled with the inverted mass matrix, otherwise (cinverse = AS_DO_NOT_INVERT_MASS) with the mass matrix itself (which means a matrix vector multiplication)

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imgLevel	integer		multigrid level
chLayer	integer		hierachical layer
rdiscrId	Tdiscrid		matrix block object mass matrix
cinverse	integer		flag if the vector shall be scaled with the inverse of the mass matrix or with the mass matrix itself (so, simple matrix-vector multiplication)

Input/Output variables:

Name	Type	Rank	Description
ivaridx	integer		handle of the scaled vector

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.20.18 Function as_igetRhsHandle

Interface:

`as_igetRhsHandle(rparBlock, rdiscrId)`

Description:
This functions returns the handle of the current right hand side vector, defined by the given parameters.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`rdiscrId`	`type(Tdiscrid)`		matrix block object

Result:
integer : handle of the rhs vector

7.20.19 Subroutine as_matBlockSet

Interface:

`as_matBlockSet(rparBlock, rdiscrInfo, imgLevel, imacroIdx, cmode, iblfIdx, pcoeff, prhs, pboundaryValue, ivar, ivar1, ccubType, csaveRhs)`

Description:
This functions sets some properties of the given matrix block object, controlled by the subtype cmode. After assembling the matrix will have full entries (AS_SBGLOBAL) due to the call of as_initborder.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rdiscrInfo	Tdiscrinfo		discretisation information
imgLevel	integer		multigrid level
imacroIdx	integer		macro index
cmode	integer		submode AS_CLEARMATRX - clears given matrix AS_SETMATRIX - sets form and assembles given matrix AS_INITMATRIX - reserves only storage for the matrices AS_CLEARRHS - clears RHS AS_SETRHS_BOTH - sets forms and assembles RHS (volume and boundary forces) AS_SETRHS_VOL - sets forms and assembles RHS (only volume forces) AS_SETRHS_BOUND - sets forms and assembles RHS (only boundary forces)
ivar	integer		function for calculating the coefficients of the Matrix function for calculating the coefficients of the RHS function for calculating the Neumann contribution to the RHS handle for data vector 1 for access for coefficient function
ivar1	integer		handle for data vector 2 for access for coefficient function
ccubType	integer		selected cubature formula
csaveRhs	integer		flag, if the assembled RHS shall be saved as default RHS

Input/Output variables:

Name	Type	Rank	Description
iblfIdx	integer		index of blf in the discretisation

7.20.20 Subroutine as_set_matrix

Interface:

as_set_matrix(rparBlock, rdiscrInfo, iblfIdx, pcoeff)

Description:
This functions assembles the matrix for the bilinear form with index iblfIdx.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rdiscrInfo	Tdiscrinfo		discretisation information

Input/Output variables:

Name	Type	Rank	Description
`iblfIdx`	`integer`		index of bilinear form in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.21 Subroutine as_clear_matrix

Interface:

`as_clear_matrix(rparBlock, rdiscrInfo, iblfIdx)`

Description:
This functions clears the matrix for the bilinear form with index iblfIdx.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`rdiscrInfo`	`Tdiscrinfo`		discretisation information

Input/Output variables:

Name	Type	Rank	Description
`iblfIdx`	`integer`		index of blf in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.22 Subroutine as_init_matrix

Interface:

`as_init_matrix(rparBlock, rdiscrInfo, iblfIdx)`

Description:
This functions clears the matrix for the bilinear form with index iblfIdx.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`rdiscrInfo`	`Tdiscrinfo`		discretisation information

Input/Output variables:

Name	Type	Rank	Description
iblfIdx	integer		index of blf in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.23 Subroutine as_set_rhs_both

Interface:

as_set_rhs_both(rparBlock, rdiscrInfo, iblfIdx, prhs, pboundaryValue, csaveRhs)

Description:
This functions assembles the RHS for the bilinear form with index iblfIdx. Volume and boundary contributions are calculated.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rdiscrInfo	Tdiscrinfo		discretisation information
csaveRhs	integer		function for calculating the coefficients of the RHS function for calculating the Neumann contribution to the RHS Flag, if calculated RHS shall be saved as default RHS. If csaveRhs is AS_SAVE_RHS, then the RHS with Neumann contribution will be saved as default RHS. Otherwise the RHS only with volume contributions is saved as default RHS.

Input/Output variables:

Name	Type	Rank	Description
iblfIdx	integer		index of blf in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.24 Subroutine as_set_rhs_vol

Interface:

as_set_rhs_vol(rparBlock, rdiscrInfo, iblfIdx, prhs, csaveRhs)

Description:
This functions assembles the RHS for the bilinear form with index iblfIdx (volume contributions only).

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rdiscrInfo	Tdiscrinfo		discretisation information
csaveRhs	integer		function for calculating the coefficients of the RHS flag, if calculated RHS shall be saved as default RHS

Input/Output variables:

Name	Type	Rank	Description
iblfIdx	integer		index of blf in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.25 Subroutine as_set_rhs_bound

Interface:

as_set_rhs_bound(rparBlock, rdiscrInfo, iblfIdx, pboundaryValue, csaveRhs)

Description:
This functions assembles the RHS for the bilinear form with index iblfIdx (only Neumann boundary contributions).

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
rdiscrInfo	Tdiscrinfo		discretisation information
csaveRhs	integer		function for calculating the Neumann contribution to the RHS flag, if calculated RHS shall be saved as default RHS

Input/Output variables:

Name	Type	Rank	Description
iblfIdx	integer		index of blf in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.26 Subroutine as_clear_rhs

Interface:

as_clear_rhs(rparBlock, rdiscrInfo, iblfIdx)

Description:
This functions clears the RHS for the bilinear form with index iblfIdx.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`rdiscrInfo`	`Tdiscrinfo`		discretisation information

Input/Output variables:

Name	Type	Rank	Description
`iblfIdx`	`integer`		index of blf in the discretisation (AS_ALL_BLF for all BLFs of the discretisation)

7.20.27 Subroutine as_parBlockInit

Interface:

`as_parBlockInit(rparBlock, imgLevel, rdiscr, rbs)`

Description:
This routine initialises the complete data structures for a given discretisation for a complete parallel block for all multigrid levels.

Input variables:

Name	Type	Rank	Description
`imgLevel`	`integer`		maximum multigrid level
`rdiscr`	`Tdiscretization`		discretisation of the domain
`rbs`	`Tbordersave`		bordersave object DEPR????

Input/Output variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object

7.20.28 Subroutine as_matVecMult

Interface:

`as_matVecMult(rparBlock, imgLevel, chLayer, rdiscrId, h_Dx, h_Dy, dalpha, dbeta, rstat, imac_start, imac_stop)`

Description:
This routine performs a matrix vector multiplication of the form $y = \alpha Ax + \beta y$ and a matrix correction in case of constant matrix multiplication.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
imgLevel	integer		multigrid level of the vectors
chLayer	integer		hierachical layer of the vectors
rdiscrId	type(Tdiscrid)		identifier of the matrix A
h_Dx	integer		handle of vector x
dalpha	real(DP)		coefficient α
dbeta	real(DP)		coefficient β
imac_start	integer		start matrixblock (wird noch geaendert)
imac_stop	integer		end matrixblock (wird noch geaendert)

Output variables:

Name	Type	Rank	Description
h_Dy	integer		handle of vector y
rstat	type(t_stat)		operation count

7.20.29 Subroutine assembly_distribute

Interface:
assembly_distribute(rparBlock, ilayer, imglev, ivaridx, rdiscrid)

Description:
This subroutine converts a vector from a given hierachical layer to another layer.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
ilayer	integer		given hierachical layer
rdiscrid	Tdiscrid		id of the discretisation

Input/Output variables:

Name	Type	Rank	Description
ivaridx	integer		

7.20.30 Subroutine assembly_distribute_sparse

Interface:

`assembly_distribute_sparse(rparBlock, dy1)`

Description:
This subroutine modifies the right hand side for computations with hanging nodes.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object

7.20.31 Subroutine assembly_interpolate_sparse

Interface:
`assembly_interpolate_sparse(rparBlock, dy1)`

Description:
Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object

7.20.32 Subroutine assembly_zero_sparse

Interface:
`assembly_zero_sparse(rparBlock, dy1)`

Description:
Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object

7.20.33 Subroutine assembly_interpolate

Interface:
`assembly_interpolate(rparBlock, ilayer, imglev, ivaridx, rdiscrid)`

Description:
This subroutine converts a vector from a given hierachical layer to another layer.(Kommentar falsch)

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`ilayer`	`integer`		given hierachical layer
`rdiscrid`	`Tdiscrid`		id of the discretisation

Input/Output variables:

Name	Type	Rank	Description
ivaridx	integer		

7.20.34 Subroutine assembly_initborder

Interface:

assembly_initborder(rparBlock, imgLevel, rdiscrid)

Description:
This routine inits the boundary structures and, if possible, calculates the delta entries for the constant matrix multiplication. At the end the matrix will have full entries as as_setBorder is called with AS_SBGLOBAL last.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imgLevel	integer		

Input/Output variables:

Name	Type	Rank	Description
rdiscrid	Tdiscrid		matrix object

7.20.35 Subroutine as_applyBoundCond

Interface:

as_applyBoundCond(rparBlock, imgLevel, rdiscrId, cboundCond, pboundaryValue, ibcNew)

Description:
This routine initialises the boundary conditions and internal data structures. If cboundCond contains AS_BC_NEUMANN, also the RHS contributions coming from Neumann boundaries are calculated and added to the RHS (but not saved as default!). At the end the matrix will have full entries (AS_SBGLOBAL) due to the call of as_initborder.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imgLevel	integer		multigrid level
cboundCond	integer		Decide how to deal with boundary conditions. Following flags can be used ADDITIVELY: AS_BC_NONE, AS_BC_DIRICHLET, AS_BC_NEUMANN, AS_BC_NO_DIRICH_IN_MATRIX, AS_BC_NO_DIRICH_IN_RHS, AS_BC_DIRICH_IN_RHS_ZERO, AS_BC_MAINDIAG_ZERO, AS_BC_DO_NOT_RESTORE_RHS (of course not every combination makes sense!)
ibcNew	integer		function for boundary values With this optional parameter the bound. cond. with number ibcNew will be connected to BLF rdiscrId.

Input/Output variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		matrix block object

7.20.36 Subroutine as_getVal

Interface:

as_getVal(iDofIdx, idxbcs, cstatus, ibc, celType, l, ndofPerEdge, nvertMacroX, dcoeff)

Description:
This subroutine computes for DOF idofIdx the coefficient for the right hand side when applying Dirichlet boundary conditions.

Input variables:

Name	Type	Rank	Description
iDofIdx	integer		index of the DOF
cstatus	integer		
ibc	integer		
idxbcs, celType,l	integer		
ndofPerEdge	integer		number of degrees of freedom in macro edge
nvertMacroX	integer		number of vertices on macro edge

Output variables:

Name	Type	Rank	Description
dcoeff	real(DP)		the desired coefficient

7.20.37 Subroutine as_setZeroDirichValues

Interface:

as_setZeroDirichValues(rparBlock, chLayer, imgLevel, rdiscrId, h_Dvec)

Description:
This routine sets the entries of the vector h_Dvec corresponding to Dirichlet boundary nodes to Dvec(j) = dalpha * Dvec(j) + dbeta * Dir(j) + dgamma * Du(j) + dvalue. The routine is merely a wrapper for as_setDirichValuesAux which does the actual work. Its purpose are performance reasons: We determine here the finite element used on every macro to not having to do this within a loop later on.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level (MB_ALL_LEV means all levels)
rdiscrId	Tdiscrid		matrix
h_Dvec	integer		handle array for vector to be set (i.e. initialised or altered etc. depending on the values of dalpha, dbeta, dgamma)

7.20.38 Subroutine as_setDirichValues

Interface:

as_setDirichValues(rparBlock, chLayer, imgLevel, rdiscrId, dalpha, h_Dvec, dbeta, pboundaryValue, dgamma, h_Du, dvalue, baverage)

Description:
This routine sets the entries of the vector h_Dvec corresponding to Dirichlet boundary nodes to Dvec(j) = dalpha * Dvec(j) + dbeta * Dir(j) + dgamma * Du(j) + dvalue. The routine is merely a wrapper for as_setDirichValuesAux which does the actual work. Its purpose are performance reasons: We determine here the finite element used on every macro to not having to do this within a loop later on.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level (MB_ALL_LEV means all levels)
rdiscrId	Tdiscrid		matrix
dalpha	real(DP)		scaling factor for current values of Dvec
h_Dvec	integer		handle array for vector to be set (i.e. initialised or altered etc. depending on the values of dalpha, dbeta, dgamma)
dbeta	real(DP)		scaling factor for Dirichlet values (which in turn are queried from pboundaryValue or set using the constant value saved with the corresponding boundary object)
dgamma	real(DP)		user defined function for calculations on Neumann boundary scaling factor for additional source term vector Du
h_Du	integer		handle array for the vector whose Dirichlet values are added to h_Dvec It is explicitly assumed that the entries of this vector already have the same state (full or averaged) as the output vector.
dvalue	real(DP)		constant value Dvec can be set to
baverage	logical		flag if the values should be averaged

7.20.39 Subroutine as_setZeroDirichValuesAux

Interface:

`as_setZeroDirichValuesAux( rparBlock, chLayer, rdiscrId, h_Dvec, imac, imgStart, imgEnd)`

Description:
This routine sets the entries of the vector h_Dvec corresponding to Dirichlet boundary nodes to Dvec(j) = dalpha * Dvec(j) + dbeta * Dir(j) + dgamma * Du(j) + dvalue. where Dir(j) is a (not physically allocated) vector containing a) the values defined by the function pboundaryValue (if the corresponding boundary object is defined as having variable function values) or b) the constant values saved with the corresponding boundary object, Du(j) is an additional source term vector (e.g. incremental loading (e.g. within a Newton-Raphson scheme) or steeming from a time stepping scheme) dvalue is a constant value to eventually set a vector to constant (even homogeneous) Dirichlet boundary values. baverage indicates whether the final vector Dvec should be returned averaged or should contain full entries. In case it is set to SOLV_AVERAGE and dalpha .ne. 0 (dgamma .ne. 0), Dvec (Du) is assumed to already have averaged entries.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
rdiscrId	Tdiscrid		matrix
h_Dvec	integer		handle array for vector to be set (i.e. initialised or altered etc. depending on the values of dalpha, dbeta, dgamma)
imac	integer		matrix block number
imgStart, imgEnd	integer		coarsest and finest multigrid level

7.20.40 Subroutine as_setDirichValuesAux

Interface:

as_setDirichValuesAux(rparBlock, chLayer, rdiscrId, dalpha, h_Dvec, dbeta, pboundaryValue, dgamma, h_Du, dvalue, baverage, pgetValue, imac, imgStart, imgEnd)

Description:
This routine sets the entries of the vector h_Dvec corresponding to Dirichlet boundary nodes to Dvec(j) = dalpha * Dvec(j) + dbeta * Dir(j) + dgamma * Du(j) + dvalue. where Dir(j) is a (not physically allocated) vector containing a) the values defined by the function pboundaryValue (if the corresponding boundary object is defined as having variable function values) or b) the constant values saved with the corresponding boundary object, Du(j) is an additional source term vector (e.g. incremental loading (e.g. within a Newton-Raphson scheme) or steeming from a time stepping scheme) dvalue is a constant value to eventually set a vector to constant (even homogeneous) Dirichlet boundary values. baverage indicates whether the final vector Dvec should be returned averaged or should contain full entries. In case it is set to SOLV_AVERAGE and dalpha .ne. 0 (dgamma .ne. 0), Dvec (Du) is assumed to already have averaged entries.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
rdiscrId	Tdiscrid		matrix
dalpha	real(DP)		scaling factor for current values of Dvec
h_Dvec	integer		handle array for vector to be set (i.e. initialised or altered etc. depending on the values of dalpha, dbeta, dgamma)
dbeta	real(DP)		scaling factor for Dirichlet values (which in turn are queried from pboundaryValue or set using the constant value saved with the corresponding boundary object)
dgamma	real(DP)		user defined function for calculations on Neumann boundary scaling factor for additional source term vector Du
h_Du	integer		handle array for the vector whose Dirichlet values are added to h_Dvec It is explicitly assumed that the entries of this vector already have the same state (full or averaged) as the output vector.
dvalue	real(DP)		constant value Dvec can be set to
baverage	logical		flag if the values should be averaged
imac	integer		finite element evaluation function to use matrix block number
imgStart, imgEnd	integer		coarsest and finest multigrid level

7.21 Module boundary

Purpose: This module implements the handling of (curved) boundaries and their parametrisation. It provides the basic segment types line and circle as well as open and closed NURBS. Also analytically defined boundaries are supported (WIP). Furthermore, the module contains routines for creating, reading and transferring boundary components.

Constant definitions:

Name	Type	Purpose
PAR_MAXNURBSDEGREE	integer	maximal degree of NURBS
Purpose: types of boundary segments		
BOUNDARY_TYPE_LINE	integer	boundary segment type line
BOUNDARY_TYPE_CIRCLE	integer	boundary segment type circle

BOUNDARY_TYPE_OPENNURBS	integer	boundary segment type open nurbs
BOUNDARY_TYPE_CLOSEDNURBS	integer	boundary segment type closed nurbs
BOUNDARY_TYPE_ANALYTIC	integer	boundary segment analytic

Purpose: kinds of boundary segments

BOUNDARY_KIND_FICTITIOUS	integer	boundary kind fictitious
BOUNDARY_KIND_GEOMETRIC	integer	boundary kind geometric

Purpose: boundary segment header definition

BOUNDARY_SEGHEADER_TYPE	integer	boundary segment header offset for type
BOUNDARY_SEGHEADER_OFFSET	integer	boundary segment header offset for offset in the data vector
BOUNDARY_SEGHEADER_NURBSDEGREE	integer	boundary segment header offset for nurbs degree
BOUNDARY_SEGHEADER_NCNTRLPNTS	integer	boundary segment header offset for number of control points
BOUNDARY_SEGHEADER_LENGTH	integer	boundary segment header length

Type definitions:

Type name	component name	Type	Rank	Purpose
Tboundary	boundary structure of the domain			
	iboundarycount_f	integer		number of fictitious boundary components
	iboundarycount_g	integer		number of geometric boundary components
	iboundarycount	integer		total number of boundary components
	h_iboundary_type	integer		handle for type vector of boundary component(s) (0=fictitious or 1=geometric)
	h_istartnode	integer		handle for index vector of starting point(s) of boundary component(s)

h_isegcount	integer	handle for a vector containing the number of segments per boundary component
h_idbldatavec_handles	integer	contains handles for data vectors of boundary components
h_iintdatavec_handles	integer	contains handles for offset vectors of boundary components

7.21.1 Function boundary_igetNBoundComp

Interface:

`boundary_igetNBoundComp(rboundary)`

Description:
This function returns the total number of boundary components.

Result:

integer : total number of boundary components

Input variables:

Name	Type	Rank	Description
rboundary	Tboundary		boundary structure

7.21.2 Function boundary_igetStartVert

Interface:

`boundary_igetStartVert(rboundary, iboundCompIdx)`

Description:
This function returns the index of of the starting node of the boundary component with index iboundCompIdx.

Result:

integer : index of start node of boundary component iboundCompIdx

Input variables:

Name	Type	Rank	Description
rboundary	Tboundary		boundary structure
iboundCompIdx	integer		index of boundary component

7.21.3 Subroutine boundary_init

Interface:

`boundary_init(rboundary, DmacroVertInfo, nnodes)`

Description:
This subroutine is to compute the index of the macro vertex with the minimal parameter value, i.e. the starting vertex, for every boundary component. These indices are stored in an array accessible via rboundary%h_istartnode.

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary structure
`DmacroVertInfo`	`real(DP)`	:	coordinates + status of macro vertices (see mastermod.f90)
`nnodes`	`integer`		number of macro vertices on coarse grid

7.21.4 Function boundary_dgetLength

Interface:
`boundary_dgetLength(rboundary, iboundCompIdx, dt1, dt2)`

Description:
This function returns the euclidian length between the point dt1 on the boundary curve and the point dt2.

Result:
real(DP) : real euclidian length

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary structure
`iboundCompIdx`	`integer`		boundary index
`dt1`	`real(DP)`		start parameter value
`dt2`	`real(DP)`		end parameter value

7.21.5 Function boundary_dgetMaxParVal

Interface:
`boundary_dgetMaxParVal(rboundary, iboundCompIdx)`

Description:
This function returns the parametric length of the boundary component iboundCompIdx

Result:
real(DP) : parametric length of boundary component iboundCompIdx

Input variables:

Name	Type	Rank	Description
rboundary	Tboundary		boundary structure
iboundCompIdx	integer		index of boundary component

7.21.6 Subroutine boundary_send

Interface:

boundary_send(idx, rboundary)

Description:
This routine sends the boundary structure rboundary to the selected process with index idx.

Input variables:

Name	Type	Rank	Description
idx	integer		index of the receiving process
rboundary	Tboundary		boundary structure

7.21.7 Subroutine boundary_rec

Interface:

boundary_rec(idx, rboundary)

Description:
This routine receives a boundary object to the given process. As the boundary objects are objects with dynamically created arrays, we cannot just receive the handles of the vectors, because they are valid on the source processor block only. Instead of this, we have to allocate the arrays on the destination process again and then we can read all the stuff from the message buffer. Therefore the same handle can have different numbers on source and destination process, but it references the same data.

Input variables:

Name	Type	Rank	Description
idx	integer		index of the source process
rboundary	Tboundary		boundary structure

7.21.8 Subroutine boundary_read

Interface:

boundary_read(iversion, rboundary, ichannel_fgeo, ichannel_ffgeo)

Description:
This routine reads the description of the geometric and fictitious boundary components from the file units ichannel_fgeo and ichannel_ffgeo. The files have to be opened before calling this routine. The format of the files is described in the manual. All information (both geometric and fictitious) is put into the structure rboundary.
The channel ichannel_ffgeo is optional.

Except of the Flag iboundary_type, we do not seperate the information for geometric and fictitious boundaries. The information regarding geometric boundaries is stored first. All information for one boundary component is stored in the arrays intdatavec and dbldatavec, which handles are accessible via the arrays iintdatavec_handles and idbldatavec_handles. Because of possible open NURBS segments with arbitrary number of control points, it is not a priori clear how long the vector dbldatavec has to be. Therefore we read the files twice : the first time to calculate the required length irequiredLength for this vector, and the second time to fill all vectors and structures. In the case of the old FEAST format (version 2) things are easier, as at most 8 entries per segment are enough (no NURBS). Therefore we do not have to read the file twice. The vector intdatavec has 4 entries per boundary segment with:
1) type of boundary segment
2) offset of the corresponding part in the dbldatavec
3) NURBS degree (0, if no NURBS)
4) number of control points (0, if no NURBS)

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary structure to be filled
`iversion`	`integer`		version of grid file (2 if created with DEVISOR 2 or 3 if created with DEVISOR 3)
`ichannel_fgeo`	`integer`		unit of geometric boundary description file
`ichannel_ffgeo`	`integer`		unit of fictitious boundary description file

7.21.9 Subroutine boundary_getcoords

Interface:

`boundary_getcoords(rboundary, iboundCompIdx, dt, dx, dy)`

Description:
This routine returns for a given parameter value dt the cartesian coordinates of the point on the boundary component iboundCompIdx

Input variables:

Name	Type	Rank	Description
`rboundary`	`Tboundary`		boundary structure
`iboundCompIdx`	`integer`		index of boundary component
`dt`	`real(DP)`		parametric value of boundary point

Output variables:

Name	Type	Rank	Description
`dx`	`real(DP)`		x-coordinate of boundary point
`dy`	`real(DP)`		y-coordinate of boundary point

7.22 Module boundarycondition

Purpose: This module contains several routines to describe boundary conditions. Each boundary condition can consist of several description blocks. Each block describes an interval or a point with Neumann or Dirichlet boundary condition. The current version allows the definition per node index, the definition per parameter value is not implemented yet.

Constant definitions:

Name	Type	Purpose
BC_DEF_PINT	integer	interval definition per parameter value (yni)
BC_DEF_PNODE	integer	point definition per parameter value (yni)
BC_DEF_NINT	integer	interval definition per node index
BC_DEF_NNODE	integer	point definition per node index
BC_TYPE_DIR	integer	Dirichlet boundary condition
BC_TYPE_NEU	integer	Neumann boundary condition
BC_VAL_FUNC	integer	parameter calculation via function
BC_VAL_CONST	integer	parameter calculation via constant
BC_USERVALUE_TYPE_INT	integer	parameter for defining an user defined boundary value as integer
BC_USERVALUE_TYPE_FLOAT	integer	parameter for defining an user defined boundary value as real

7.22.1 Subroutine bc_show

Interface:
`bc_show(rbcList)`

Description:
This routines prints out a summary of the given bc structure for debugging purposes

Input variables:

Name	Type	Rank	Description
rbcList	t_bcList		boundary condition structure

7.22.2 Function bc_getIndex

Interface:
`bc_getIndex(rbcList,sname)`

Description:
This function returns the index of the named boundary condition

Input variables:

Name	Type	Rank	Description
`rbcList`	`t_bcList`		list of boundary conditions
`sname`	`c(*)`		name of the boundary condition

Result:
integer : index of the boundary condition, if the name was not found -1 is returned

7.22.3 Subroutine bc_read

Interface:
`bc_read(IedgeInfoSD, rbcList, ichannel, bfoundBC)`

Description:
This routine reads the description of the boundary conditions from the open file channel ichannel. The file format is specified in the FEAST File Format Specification. (WIP)

Input variables:

Name	Type	Rank	Description
`IedgeInfoSD`	`integer`	:, :	global edge info (node1, node2, status)
`rbcList`	`t_bcList`		boundary condition structure
`ichannel`	`integer`		file channel

Output variables:

Name	Type	Rank	Description
`bfoundBC`	`logical`		flag if a boundary condition is defined

7.22.4 Subroutine bc_send

Interface:
`bc_send(iprocess, rbcList)`

Description:
This routine is called by the master to send bc information stored in rbcList to the slave iprocess.

Input variables:

Name	Type	Rank	Description
`iprocess`	`integer`		number of the process receiving the bc information
`rbcList`	`t_bcList`		boundary condition structure

7.22.5 Subroutine bc_receive

Interface:

`bc_receive(isourceProcess, rbcList)`

Description:
This routine is called by a slave to receive bc information from the process isourceProcess (usually the master) and to store it in rbcList.

Input variables:

Name	Type	Rank	Description
`isourceProcess`	`integer`		number of the source process, which sends the bc information
`rbcList`	`t_bcList`		boundary condition structure

7.22.6 Subroutine bc_reserve

Interface:

`bc_reserve(rbcList, ibidx, nbcseg, sname)`

Description:
This routine reserves the memory for a given boundary condition list.

Input variables:

Name	Type	Rank	Description
`rbcList`	`t_bcList`		boundary condition structure
`ibidx`	`integer`		selected boundary condition
`nbcseg`	`integer`		number of boundary condition segments in selected boundary condition structure
`sname`	`c(*)`		name of the boundary condition

7.22.7 Subroutine bc_addSegment_vertIdx_int

Interface:

`bc_addSegment_vertIdx_int(IedgeInfoSD, rbcList, ibidx, isegidx, cstatus, dval, bflag, in1, in2)`

Description:
This routine adds a boundary condition segment of type BC_DEF_NINT to a given boundary condition.

Input variables:

Name	Type	Rank	Description
IedgeInfoSD	integer	:, :	global edge info (node1, node2, status)
rbcList	t_bcList		boundary condition structure
ibidx	integer		index of selected boundary condition
isegidx	integer		segment index within the boundary condition
cstatus	integer		type of boundary condition (BC_TYPE_DIR,BC_TYPE_NEU)
dval	real(DP)		constant for bc parameter
bflag	logical		if true then bc parameter is variable
in1	integer		first interval definition node
in2	integer		second interval definition node

7.22.8 Subroutine bc_addSegment_vertIdx_vert

Interface:

bc_addSegment_vertIdx_vert(IedgeInfoSD, rbcList, ibidx, isegidx, cstatus, dval, bflag, in1)

Description:
This routine adds a boundary condition segment of type BC_DEF_NNODE to a given boundary condition structure.

Input variables:

Name	Type	Rank	Description
IedgeInfoSD	integer	:, :	global edge info (node1, node2, status)
rbcList	t_bcList		bondary condition structure
ibidx	integer		selected boundary condition
isegidx	integer		segment index within the bc
cstatus	integer		bc status (BC_TYPE_DIR,BC_TYPE_NEU)
dval	real(DP)		constant for bc parameter
bflag	logical		if .TRUE. then bc parameter is variable
in1	integer		segment definition node

7.22.9 Subroutine bc_init

Interface:

bc_init(nmaxMgLevel, nmatBlocks, RmatBlockBaseList, RmacroBaseList, RmacroList, RmacroInMatBlock, IedgeInfoSD, rbcList, ibc, brelease)

Description:

This subroutine initializes the internal data structures. This subroutine has to be called after the definition of a boundary condition structure.

Input variables:

Name	Type	Rank	Description
nmaxMgLevel	integer		max. MG level
nmatBlocks	integer		number of matrix blocks
RmatBlockBaseList	t_matrixBlockbase	dim(:, :)	matrix block base list
RmacroList	t_macroAccess	dim(:)	macro list
RmacroBaseList	t_macroBase	dim(:, :)	macro base list
RmacroInMatBlock	Tmpmb_access	dim(:)	macros per matrix block
IedgeInfoSD	integer	:, :	global edge info (node1, node2, status)
rbcList	t_bcList		boundary condition structure
ibc	integer		selected boundary condition
brelease	logical		if true release node vector

7.23 Module cubature

Purpose: This module provides several cubature schemes with their nodes and and weights. These values refer to the reference element

Constant definitions:

Name	Type	Purpose
Constants for ccubType		
Purpose: 1D formulas		
CUB_G1_1D	integer	1-point Gauss formula, 1D, degree = 2, ncubp = 1
CUB_TRZ_1D	integer	trapezoidal rule, 1D, degree = 2, ncubp = 2
CUB_G2_1D	integer	2-point Gauss formula, 1D, degree = 4, ncubp = 2
CUB_G3_1D	integer	3-point Gauss formula, 1D, degree = 6, ncubp = 3
CUB_G4_1D	integer	4-point Gauss formula, 1D, degree = 8, ncubp = 4
CUB_G5_1D	integer	5-point Gauss formula, 1D, degree = 10, ncubp = 5
CUB_SIMPSON_1D	integer	Simpson-rule, 1D, degree = 4, ncubp = 3

CUB_G6_1D	integer	6-point Gauss formula, 1D, degree = 12, ncubp = 6
Constants for ccubType		
Purpose: 2D formulas, quad		
CUB_G1X1	integer	1x1 Gauss formula, degree = 2, ncubp = 1
CUB_TRZ	integer	trapezoidal rule, degree = 2, ncubp = 4
CUB_MID	integer	midpoint rule, degree = 2, ncubp = 4
CUB_G2X2	integer	2x2 Gauss formula, degree = 4, ncubp = 4
CUB_NS1	integer	Newton formula 1, degree = 4, ncubp = 4
CUB_NS2	integer	Newton formula 2, degree = 5, ncubp = 6
CUB_NS3	integer	Newton formula 3, degree = 6, ncubp = 7
CUB_G3X3	integer	3x3 Gauss formula, degree = 6, ncubp = 9
CUB_G	integer	Gauss formula, degree = 7, ncubp = 12
CUB_G4X4	integer	4x4 Gauss formula, degree = 8, ncubp = 16
CUB_G5X5	integer	5x5 Gauss formula, degree = 10, ncubp = 25
CUB_PG1X1	integer	piecewise 1x1 Gauss formula, degree = 2, ncubp = 4
CUB_PTRZ	integer	piecewise trapezoidal rule, degree = 2, ncubp = 9
CUB_PG2X2	integer	piecewise 2x2 Gauss formula, degree = 4, ncubp = 16
CUB_PG3X3	integer	piecewise 3x3 Gauss formula, degree = 6, ncubp = 36
Constants for ccubType		
Purpose: 2D formulas, tri		
CUB_G1_T	integer	1-point Gauss formula, triangle, degree = 2, ncubp = 1
CUB_TRZ_T	integer	trapezoidal rule, triangle, degree = 2, ncubp = 3
CUB_G3_T	integer	3-point Gauss formula, triangle, degree = 3, ncubp = 3
CUB_Collatz	integer	Collatz formula, degree = 3, ncubp = 3

CUB_VMC	integer	vertices, midpoints, center, degree = 4, ncubp = 7
Constants for ccubType		
Purpose: 3D formulas		
CUB_G1_3D	integer	1-point Gauss formula, 3D, degree = 2, ncubp = 1
CUB_MIDAREA_3D	integer	midpoints of areas, 3D, degree = 2, ncubp = 6
CUB_TRZ_3D	integer	trapezoidal rule, 3D, degree = 2, ncubp = 8
CUB_G2_3D	integer	2-point Gauss formula, 3D, degree = 4, ncubp = 8
CUB_G3_3D	integer	3-point Gauss formula, 3D, degree = 6, ncubp = 27

Global variable definitions:

Name	Type	Rank	Purpose
cub_ncubp	integer		number of cubature points of the selected cubature scheme
cub_dxi	real(DP)	dim(CUB_MAXCUBP, 3)	coordinates of the cubature points (format: icub,xyz)
cub_domega	real(DP)	dim(CUB_MAXCUBP)	weights of the cubature nodes on reference element

7.23.1 Function cub_igetID

Interface:

```
cub_igetID(scubName)
```

Description:
This routine returns the cubature id to a given cubature formula name. It is case-insensitive.

Result:
integer : id of the cubature formula

Input variables:

Name	Type	Rank	Description
scubName	c (*)		cubature formula name

7.23.2 Subroutine cub_getCubPoints

Interface:

```
cub_getCubPoints(ccubType)
```

Description:

This routine initializes the coordinates and weight fields according to the selected cubature formula. The integration domain is $[-1,1]^n$. In the case of one-dimensional integration, only cub_dxi(i,1) is used. The coordinates of the cubature points on triangles are given in barycentric coordinates.

Input variables:

Name	Type	Rank	Description
`ccubType`	integer		id of the cubature formula to be set

7.24 Module element

Purpose: This module contains the generation routines for the finite element shape functions. Currently, only the bilinear element Q_1 (Elem_Q1) is fully supported.

Constant definitions:

Name	Type	Purpose
`EL_UNSET`	`integer`	indicator for a not set element type (used in tools.f90)
`EL_Q1`	`integer`	ID of bilinear conforming quadrilateral FE
`EL_E11`	`integer`	ID of bilinear conforming quadrilateral FE (just for the FEAT-users...)
`EL_Q2`	`integer`	ID of biquadratic conforming quadrilateral FE
`EL_Q2L`	`integer`	ID of biquadratic conforming quadrilateral FE (Lagrangian basis functions)
`EL_E13`	`integer`	ID of biquadratic conforming quadrilateral FE (just for the FEAT-users...)
`EL_Q3`	`integer`	ID of bicubic conforming quadrilateral FE
`EL_Q3L`	`integer`	ID of bicubic conforming quadrilateral FE (Lagrangian basis functions)
`EL_E14`	`integer`	ID of bicubic conforming quadrilateral FE (just for the FEAT-users...)
`EL_CALC`	`integer`	If the element is called with parameter EL_CALC, the function values and the desired derivatives are computed.

EL_SETID	integer	If the element is called with parameter EL_SETID, only the element ID is returned. Nothing is computed.

Purpose: maximal values

EL_MAXNBAS	integer	maximum number of basic functions
EL_MAXNDER	integer	maximum number of derivatives (length of el_bder, explanation below)
EL_MAXNVE	integer	(maximum) number of vertices per element

Global variable definitions:

Name	Type	Rank	Purpose
el_dbas	real(DP)	EL_MAXNBAS, EL_MAXNDER	The following variables refer to the currently considered element (have to be recomputed when stepping to the next element) values of the basic functions
el_djac	real(DP)	4	values of the Jacobian matrix 1 1 - 1 2 1 - 2 1 2 - 3 2 2 - 4
el_ive	integer	EL_MAXNVE	global indices of vertices
el_detj	real(DP)		determinant of the Jacobian matix
el_bder	logical	EL_MAXNDER	The entries in el_bder corresponding to the values which shall be computed have to be set to .TRUE. 1 - function value 2,3 - x- and y-derivative 4,5,6 - xx,xy,yy-derivative

7.24.1 Subroutine elem_Q1

Interface:

elem_Q1(dx, dy)

Description:
This subroutine calculates the values of the basic functions of the conforming bilinear finite element at the specified point (dx,dy) on the reference element.

Input variables:

Name	Type	Rank	Description
dx, dy	real(DP)		cartesian coordinates of the evaluation point on reference element

7.24.2 Subroutine elem_Q2

Interface:

`elem_Q2(dx, dy)`

Description:
This subroutine calculates the values of the basic functions of the conforming biquadratic finite element at the specified point (dx,dy) on the reference element. The distribution of the local degrees of freedom is as follows:
4—5—3
— — —
— — —
6—7—8
— — —
— — —
1—9—2
The global numbering is strictly rowwise. This strange numbering has two reasons: At first, we use hierachical basis functions. Therefore it seems favourable, that the local degrees of freedom are distributed such that the first degrees of freedom correspond with the degrees of freedom from the lower element (here: Q_1). Second, we do not use local ordering. Therefore, we should order the basis functions such that the number of jumps between rows is minimized when looping over the local degrees of freedom. Because of the rowwise numbering, such a jump results in a big jump in the global vector which may cause cache misses. Despite of this, jumps in a row are not critical, as physically these entries are close together (maximum distance: 3). In this implementation, the number of critical jumps is 3 and therefore optimal.

Input variables:

Name	Type	Rank	Description
`dx, dy`	`real(DP)`		cartesian coordinates of the evaluation point on reference element

7.24.3 Subroutine elem_Q2L

Interface:

`elem_Q2L(dx, dy)`

Description:
This subroutine calculates the values of the basic functions of the conforming biquadratic finite element at the specified point (dx,dy) on the reference element. The distribution of the local degrees of freedom is as follows:
7—8—9
‖ ‖ ‖
— — —
4—5—6
— — —
— — —
1—2—3
The global numbering is strictly rowwise. In contrast to elem_Q2, here Lagrangian basis functions are used.

Input variables:

Name	Type	Rank	Description
`dx, dy`	`real(DP)`		cartesian coordinates of the evaluation point on reference element

7.24.4 Subroutine elem_Q3

Interface:

`elem_Q3(dx, dy)`

Description:

This subroutine calculates the values of the basic functions of the conforming bicubic finite element at the specified point (dx,dy) on the reference element. The distribution of the local degrees of freedom is as follows:

4—5—16—-3 — — — — — — — — — 12—-13—-14—-15 — — — — — — — — — 6—7—11—-8 — — — — — — — — — 1—9—10—-2

The global numbering is strictly rowwise. This strange numbering has two reasons:

At first, we use hierachical basis functions. Therefore it seems favourable, that the local degrees of freedom are distributed such that the first degrees of freedom correspond with the degrees of freedom from the lower element (here: Q_2).

Second, we do not use local ordering. Therefore, we should order the basis functions such that the number of jumps between rows is minimized when looping over the local degrees of freedom. Because of the rowwise numbering, such a jump results in a big jump in the global vector which may cause cache misses. Despite of this, jumps in a row are not critical, as physically these entries are close together (maximum distance: 4). In this implementation, the number of critical jumps is 6. The optimal number is 3, but this kind of numbering does not permit p-hierarchy.

Input variables:

Name	Type	Rank	Description
`dx, dy`	`real(DP)`		cartesian coordinates of the evaluation point on reference element

7.24.5 Subroutine elem_Q3L

Interface:

`elem_Q3L(dx, dy)`

Description:

This subroutine calculates the values of the basic functions of the conforming bicubic finite element at the specified point (dx,dy) on the reference element. The distribution of the local degrees of freedom is as follows:

13—-14—-15—-16 — — — — — — — — — 9—10—-11—-12 — — — — — — — — — 5—6—7—8 — — — — — — — — — 1—2—3—4

The global numbering is strictly rowwise. In contrast to elem_Q3, here Lagrangian basis functions are used.

Input variables:

Name	Type	Rank	Description
`dx, dy`	`real(DP)`		cartesian coordinates of the evaluation point on reference element

7.24.6 Subroutine elem_generic

Interface:
`elem_generic(dx, dy, celType)`

Description:
This is a wrapper routine for the specific element routines.

Input variables:

Name	Type	Rank	Description
`dx, dy`	`real(DP)`		coordinates of evaluation point on reference element
`celType`	integer		ID of finite element type

7.24.7 Function elem_igetNDofLoc

Interface:
`elem_igetNDofLoc(celType)`

Description:
This function returns the number of local DOFs for the element used.

Input variables:

Name	Type	Rank	Description
`celType`	integer		ID of finite element type

Result:
integer : number of local degrees of freedom

7.24.8 Function elem_igetNDofGlob

Interface:
`elem_igetNDofGlob(rgrid, celType)`

Description:
This function returns the number of global DOFs for the element chosen.

Input variables:

Name	Type	Rank	Description
`rgrid`	`Tgrid`		grid object
`celType`	integer		ID of finite element type

Result:
integer : global number of equations on current grid

7.24.9 Subroutine elem_locGlobMapping

Interface:

`elem_locGlobMapping(ielIdx, celType, rgrid, IdofGlob)`

Description:
This subroutine calculates the global indices in the array IdofGlob of the degrees of freedom of the element ielIdx. It is a wrapper routine for the corresponding routines for a specific element type.

Input variables:

Name	Type	Rank	Description
`ielIdx`	`integer`		element index
`celType`	integer		ID of finite element type
`rgrid`	`Tgrid`		grid object

Output variables:

Name	Type	Rank	Description
`IdofGlob`	`integer`	EL_MAXNBAS	array of global indices

7.24.10 Subroutine elem_locGlobMapping_Q1

Interface:

`elem_locGlobMapping_Q1(ielIdx, rgrid, IdofGlob)`

Description:
This subroutine calculates the global indices in the array IdofGlob of the degrees of freedom of the element ielIdx in the case of the conforming bilinear element Q_1 .

Input variables:

Name	Type	Rank	Description
`ielIdx`	`integer`		element index
`rgrid`	`Tgrid`		grid object

Output variables:

Name	Type	Rank	Description
`IdofGlob`	`integer`	EL_MAXNBAS	array of global indices

7.24.11 Subroutine elem_locGlobMapping_Q2

Interface:

`elem_locGlobMapping_Q2(ielIdx, rgrid, IdofGlob)`

Description:
This subroutine calculates the global indices in the array IdofGlob of the degrees of freedom of the element ielIdx in the case of the conforming biquadratic element Q_2 .

Input variables:

Name	Type	Rank	Description
`ielIdx`	`integer`		element index
`rgrid`	`Tgrid`		grid object

Output variables:

Name	Type	Rank	Description
`IdofGlob`	`integer`	EL_MAXNBAS	array of global indices

7.24.12 Subroutine elem_locGlobMapping_Q2L

Interface:
`elem_locGlobMapping_Q2L(ielIdx, rgrid, IdofGlob)`

Description:
This subroutine calculates the global indices in the array IdofGlob of the degrees of freedom of the element ielIdx in the case of the conforming biquadratic element Q_2 (Lagrangian basis functions).

Input variables:

Name	Type	Rank	Description
`ielIdx`	`integer`		element index
`rgrid`	`Tgrid`		grid object

Output variables:

Name	Type	Rank	Description
`IdofGlob`	`integer`	EL_MAXNBAS	array of global indices

7.24.13 Subroutine elem_locGlobMapping_Q3

Interface:
`elem_locGlobMapping_Q3(ielIdx, rgrid, IdofGlob)`

Description:
This subroutine calculates the global indices in the array IdofGlob of the degrees of freedom of the element ielIdx in the case of the conforming bicubic element Q_3 .

Input variables:

Name	Type	Rank	Description
ielIdx	integer		element index
rgrid	Tgrid		grid object

Output variables:

Name	Type	Rank	Description
IdofGlob	integer	EL_MAXNBAS	array of global indices

7.24.14 Subroutine elem_locGlobMapping_Q3L

Interface:

`elem_locGlobMapping_Q3L(ielIdx, rgrid, IdofGlob)`

Description:
This subroutine calculates the global indices in the array IdofGlob of the degrees of freedom of the element ielIdx in the case of the conforming bicubic element Q_3 (Lagrangian basis functions).

Input variables:

Name	Type	Rank	Description
ielIdx	integer		element index
rgrid	Tgrid		grid object

Output variables:

Name	Type	Rank	Description
IdofGlob	integer	EL_MAXNBAS	array of global indices

7.24.15 Subroutine elem_calcJacPrepare

Interface:

`elem_calcJacPrepare()`

Description:
This subroutine prepares the computation of the element Jacobi matrix. It calculates some values which do not change when looping over cubature points and thus saves some operations. The routine has to be called per element before the loop over the cubature points is started. It is assumed that the coordinates of the current element are already stored in the global arrays gr_dxs and gr_dys.

7.24.16 Subroutine elem_calcJac

Interface:

`elem_calcJac(dxi, deta)`

Description:
This subroutine calculates the element Jacobi matrix and its determinant in the current cubature point (dxi, deta). It is assumed that the routine elem_calcJacPrepare() has already been called for the current element such that the array el_calcJacPrep is correctly set.

Input variables:

Name	Type	Rank	Description
`dxi, deta`	`real(DP)`		the coordinates of the current cubature point

7.24.17 Subroutine elem_calcRealCoords

Interface:
`elem_calcRealCoords(dxi, deta, dxReal, dyReal)`

Description:
This subroutine computes the real coordinates of a point which is given by parameter values (dxi, deta) on the reference element. It is assumed that the subroutine elem_calcJacPrepare has been clled before.

Input variables:

Name	Type	Rank	Description
`dxi, deta`	`real(DP)`		parameter values on reference element

Output variables:

Name	Type	Rank	Description
`dxReal, dyReal`	`real(DP)`		real coordinates of this point

7.25 Module errorcontrol

Purpose: This module contains routines which are related to error estimation procedures. This contains the routines ec_compSPR and ec_compPPR for obtaining recovered gradients, which are used in ec_h1Est to estimate the H1-error. Furthermore, there are and will be routines to enable dual weighted residual based error control.

Constant definitions:

Name	Type	Purpose
Constants for cerrorQuantity		
Purpose: target functionals		

EC_NO	integer	no error control
EC_H1	integer	control of gradient error
EC_POINT	integer	control of point error
EC_LINE	integer	control of integral mean value along a line
EC_LIFT	integer	control of lift
EC_DRAG	integer	control of drag
Constants for cadapMethod Purpose: method of adaptation		
EC_H	integer	h-adaptivity
EC_R	integer	r-adaptivity
EC_P	integer	p-adaptivity
EC_RH	integer	rh-adaptivity

7.25.1 Subroutine ec_compSPR_scalar

Interface:

ec_compSPR_scalar(rparBlock, rdiscrId, h_Dsol, h_DrecGradX, h_DrecGradY, imgLevel)

Description:

Compute the recovered gradient of the solution vector using the superconvergent patch recovery technique by Zienkiewicz and Zhu. The gradient is recovered by evaluating the gradient in the Gauss points (for Q1: in the element center) and then computing a bilinear interpolation of the values in these points. This is evaluated in the vertex, defining the bilinear recovered gradient, which is superconvergent on regular meshes.
In this routine, we have to use local coordinates for two reasons:
1) If the interpolation points are extremely close, the matrix for bilinear interpolation has a condition number of O(1/patchsize**2), as we have a row with terms (x-x_i)* (y-y_i) = O(distance-2). Therefore, we introduce scaled coordinates ensuring the patchsize (in these coordinates) to be O(1).
2) In the special case of a patch consisting of 4 identical quads, but rotated by 45 degrees, we have the situation that all the interpolation points are situated on the coordinate axes. As the function xy and 0 both interpolate these 4 points, the interpolation is not unisolvent (see Rannacher-Turek-element). Therefore, we rotate the local coordinates, too, to avoid this special case. Remarks: 1) The boundaries of the macros are treated like domain boundaries. 2) The recovered gradient vectors have to be allocated before! 3) The subroutine SPR_Q2L only provides the structure for gradient recovery in the quadratic case. Therefore, it serves for both Q2 AND Q2L. Which element is used, is decided by the choice of the routines pcomputepatchQ2 and compSPRonBDryQ2. 4) For the same reason, SPR_Q1 and SPR_Q2L work for the scalar and for the vector-valued case.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		structure describing the parallel-block belonging to the calling slave
h_Dsol	integer		handle of solution
h_DrecGradX	integer		handle of recovered x-derivative of solution
h_DrecGradY	integer		handle of recovered y-derivative of solution
rdiscrId	Tdiscrid		discretisation ID (determines element type)
imgLevel	integer		multigrid level for which the recovered gradient shall be computed

7.25.2 Subroutine ec_compSPR_vector

Interface:

ec_compSPR_vector(rparBlock, rdiscrId, h_Dsolx, h_Dsoly, h_DrecGradXX, h_DrecGradXY, h_DrecGradYX, h_DrecGradYY, imgLevel)

Description:
Compute the recovered gradient of the solution vector using the superconvergent patch recovery technique by Zienkiewicz and Zhu. The gradient is recovered by evaluating the gradient in the Gauss points (for Q1: in the element center) and then computing a bilinear interpolation of the values in these points. This is evaluated in the vertex, defining the bilinear recovered gradient, which is superconvergent on regular meshes.
In this routine, we have to use local coordinates for two reasons:
1) If the interpolation points are extremely close, the matrix for bilinear interpolation has a condition number of O(1/patchsize**2), as we have a row with terms (x-x_i)* (y-y_i) = O(distance-2). Therefore, we introduce scaled coordinates ensuring the patchsize (in these coordinates) to be O(1).
2) In the special case of a patch consisting of 4 identical quads, but rotated by 45 degrees, we have the situation that all the interpolation points are situated on the coordinate axes. As the function xy and 0 both interpolate these 4 points, the interpolation is not unisolvent (see Rannacher-Turek-element). Therefore, we rotate the local coordinates, too, to avoid this special case. Remarks: The boundaries of the macros are treated like domain boundaries. The recovered gradient vectors have to be allocated before!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		structure describing the parallel-block belonging to the calling slave
h_Dsolx, h_Dsoly	integer		handles of solution vectors (x-and y-component)
h_DrecGradXX, h_DrecGradXY	integer		handle of recovered x- and y-derivative of x-component of the solution
h_DrecGradYX, h_DrecGradYY	integer		handle of recovered x- and y-derivative of y-component of the solution
rdiscrId	Tdiscrid		discretisation ID (determines element type)
imgLevel	integer		multigrid level for which the recovered gradient shall be computed

7.25.3 Subroutine ec_compPPR_scalar

Interface:

ec_compPPR_scalar(rparBlock, rdiscrID, h_Dsol, h_DrecGradX, h_DrecGradY, imgLevel)

Description:
Compute the recovered gradient of the solution vector using the polynomial preserving recovery technique by Zhang. The gradient is recovered computing a biquadratic interpolation on a patch of four elements using its nine vertices. Evaluating the gradient of this interpolation in the vertex in the center of the patch, we obtain the values of the recovered gradient in each inner vertex. This defines a continuous bilinear recovered gradient, which is superconvergent on arbitrary meshes (for details, see Zhimin Zhangs work).
In this routine, we have to use local coordinates for two reasons:
1) If the interpolation points are extremely close, the matrix for biquadratic interpolation has a condition number of O(1/patchsize**4), as we have a row with terms (x-x_i)**2 * (y-y_i)**2 = O(distance**-4). Therefore, we introduce scaled coordinates ensuring the patchsize (in these coordinates) to be O(1).
2) In the special case of a patch consisting of 4 identical quads, but rotated by 45 degrees, we have the situation that all the interpolation points are situated on the coordinate axes. As the function xy and 0 both interpolate these 4 points, the interpolation is not unisolvent (see Rannacer-Turek-element). Therefore, we rotate the local coordinates, too, to avoid this special case. This is actually not necessary for the biquadratic interpolation here, but performed for commonality with the SPR-variant of this routine in the same module. Remarks: The boundaries of the macros are treated like domain boundaries. The recovered gradient vectors have to be allocated before!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		structure describing the parallel-block belonging to the calling slave
rdiscrId	Tdiscrid		discretisation ID (determines element type)
h_Dsol	integer		handle of solution
h_DrecGradX	integer		handle of recovered x-derivative of solution
h_DrecGradY	integer		handle of recovered y-derivative of solution
imgLevel	integer		multigrid level on which the gradient recovery shall take place

7.25.4 Subroutine ec_compPPR_vector

Interface:

ec_compPPR_vector(rparBlock, rdiscrId, h_Dsolx, h_Dsoly, h_DrecGradXX, h_DrecGradXY, h_DrecGradYX, h_DrecGradYY, imgLevel)

Description:
Compute the recovered gradient of the solution vector using the polynomial preserving recovery technique by Zhang. The gradient is recovered computing a biquadratic interpolation on a patch of four elements using its nine vertices. Evaluating the gradient of this interpolation in the vertex in the center of the patch, we obtain the values of the recovered gradient in each inner vertex. This defines a continuous bilinear recovered gradient, which is superconvergent on arbitrary meshes (for details, see Zhimin Zhangs work).
In this routine, we have to use local coordinates for two reasons:
1) If the interpolation points are extremely close, the matrix for biquadratic interpolation has a condition number of O(1/patchsize**4), as we have a row with terms (x-x_i)**2 * (y-y_i)**2 = O(distance**-4). Therefore, we introduce scaled coordinates ensuring the patchsize (in these coordinates) to be O(1).
2) In the special case of a patch consisting of 4 identical quads, but rotated by 45 degrees, we have the situation that all the interpolation points are situated on the coordinate axes. As the function xy and 0 both interpolate these 4 points, the interpolation is not unisolvent (see Rannacer-Turek-element). Therefore, we rotate the local coordinates, too, to avoid this special case. This is actually not necessary for the biquadratic interpolation here, but performed for commonality with the SPR-variant of this routine in the same module. Remarks: The boundaries of the macros are treated like domain boundaries. The recovered gradient vectors have to be allocated before!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		structure describing the parallel-block belonging to the calling slave
rdiscrId	Tdiscrid		discretisation ID (determines element type)
h_Dsolx, h_Dsoly	integer		handles of solution components
h_DrecGradXX, h_DrecGradXY	integer		handle of recovered x- and y-derivative of x-component of the solution
h_DrecGradYX, h_DrecGradYY	integer		handle of recovered x- and y-derivative of y-component of the solution
imgLevel	integer		multigrid level on which the graddient recovery shall take place

7.25.5 Subroutine ec_h1Est_scalar

Interface:

ec_h1Est_scalar(rparBlock, rdiscrId, h_DelemContrib, h_Ilocalised, h_DmacroContrib, dh1Error, h_Dsol, h_DrecGradX, h_DrecGradY)

Description:
Compute the elementwise contributions of the H1-error estimation using a recovered gradient technique. This suhbroutine applies to the scalar case. Remark: The recovered gradient has to be computed outside this routine.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock identifier
rdiscrId	Tdiscrid		discretisation ID (determines element type)
h_Dsol	integer		handle of solution
h_DrecGradX	integer		handle of recovered x-derivative of solution
h_DrecGradY	integer		handle of recovered y-derivative of solution

Output variables:

Name	Type	Rank	Description
h_DelemContrib	integer		handle of vector with element contributions to the H1-error of the solution
h_Ilocalised	integer		handle of the vector indicating if the error is localised inside the macro
h_DmacroContrib	integer		handle of vector with macro-wise error contributions
dh1Error	real(DP)		estimated gradient error

7.25.6 Subroutine ec_h1Est_vector

Interface:

ec_h1Est_vector(rparBlock, rdiscrId, h_DelemContrib, h_Ilocalised, h_DmacroContrib, dh1Error, h_Dsolx, h_Dsoly, h_DrecGradXX, h_DrecGradXY, h_DrecGradYX, h_DrecGradYY)

Description:
Compute the elementwise contributions of the H1-error estimation using a recovered gradient technique. This function is also applicable to vector-valued solutions. Remark: The recovered gradient has to be computed outside this routine.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock identifier
rdiscrId	Tdiscrid		discretisation ID (determines element type)
h_Dsolx, h_Dsoly	integer		handles of solution components
h_DrecGradXX	integer		handle of recovered x-derivative of solution
h_DrecGradXY	integer		handle of recovered y-derivative of solution
h_DrecGradYX	integer		handle of recovered x-derivative of the second solution component
h_DrecGradYY	integer		handle of recovered y-derivative of the second solution component

Output variables:

Name	Type	Rank	Description
h_DelemContrib	integer		handle of vector with element contributions to the H1-error of the solution
h_Ilocalised	integer		handle of the vector indicating if the error is localised inside the macro
h_DmacroContrib	integer		handle of vector with macro-wise error contributions
dh1Error	real(DP)		estimated gradient error

7.25.7 Subroutine ec_buildRhsLine

Interface:

ec_buildRhsLine(rparBlock, dstartX, dstartY, dendX, dendY, rdiscrId, h_DdualRhs, h_DprimalSol, dtargetVal, h_DprimalSol2, dtargetVal2)

Description:
Build the dual rhs for the estimation of the error of the integral over a line in the domain and compute the target value of the primal solution(s), i.e. compute $\int_\Gamma (u) ds$. In the case of vector-valued problems, 2 target values are computed, but only one dual rhs, as for vector-valued problems, the block matrices are treated as different single matrices. Therefore, the 2 dual rhs are identical. The line exceeds the start and end points. The line does not have to match the edges of the grid.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		structure describing the parallel-block belonging to the calling slave
rdiscrId	Tdiscrid		discretization structure
dstartX	real(DP)		x-coordinate of the first point defining the integration line
dstartY	real(DP)		y-coordinate of the first point defining the integration line
dendX	real(DP)		x-coordinate of the second point defining the integration line
dendY	real(DP)		y-coordinate of the second point defining the integration line
h_DprimalSol	integer		handle of solution vector
h_DprimalSol2	integer		handle of second solution component (optional)

Output variables:

Name	Type	Rank	Description
h_DdualRhs	integer		handle for dual right hand side
dtargetVal	real(DP)		value of the integral over the line of the primal solution
dtargetVal2	real(DP)		value of the integral over the line of the second component of the primal solution (optional)

7.25.8 Subroutine ec_compContribIntp

Interface:

ec_compContribIntp(rparBlock, celtype, rmatPrimal, h_DprimalSol, h_DdualSol, h_DelemContrib, h_clocalised, h_DmacroContrib, prhs, pcoeff, dtargetErr)

Description:
Compute the elementwise contributions of the error estimation, i.e. for the primal problem $a(u, \varphi) = F(\varphi)$, compute the error estimation $J(u - u_h) = F(z - z_h) - a(u_h, z - z_h)$.
The definitions for a(.,.) and F(.) are read from the forms structure asociated with the primal matrix. This routine works for LINEAR equations and target functionals only.
The dual solution is approximated by biquadratic reconstruction. To perform this, we compute the biquadratic interpolation on every patch. As every inner element is covered by four patches, we take as the reconstruction the arithmetic mean of the four interpolations available. The elementwise error contributions are stored in the array DelemContrib (length: number of elements in parblock), the macrowise error contributions are stored in DmacroContrib. If the error inside the macro is localised, i.e. if the largest elementwise error contribution is XXX times larger than the average elementwise error contribution on this macro, clocalised(el_idx) is set to YES.
The error contributions from boundary approximation errors and the error on the boundary due to non-homogenous boundary data is not taken into account.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		the parallelblock structure of the parallel block
rMatPrimal	Tdiscrid		matrix description for the primal problem (this is to evaluate!!!!)
h_DprimalSol	integer		handle for primal solution
h_DdualSol	integer		handle for dual solution

Output variables:

Name	Type	Rank	Description
h_DelemContrib	integer		handle for vector with elementwise error contributions
dtargetErr	real(DP)		error of target functional
h_clocalised	integer		handle for the vector indicating if the error is localised in the macro
h_DmacroContrib	integer		handle for the vector with macrowise error contributions

7.25.9 Subroutine ec_prepareAdapRef

Interface:

`ec_prepareAdapRef(rparBlock, h_DmacroContrib, h_imacroRefIdx, inummacroRef)`

Description:
This subroutine steers the adaptive grid refinement process. The fixed fraction strategy is applied, ie.e the macrowise error contributions are sorted and the worst 33
Input variables:

Name	Type	Rank	Description
`h_DmacroContrib`	`integer`		corresponding handles

Input/Output variables:

Name	Type	Rank	Description
`h_ImacroRefIdx`	`integer`		parallel block structure

Output variables:

Name	Type	Rank	Description
`inummacroRef`	`integer`		number of macros to be refined an current parblock

7.25.10 Subroutine ec_buildrhs_dudn_direct

Interface:

`ec_buildrhs_dudn_direct(rparBlock, rmatPrimal, rmatDual, h_DprimalSol, h_DdualRhs, iboundarycomp, dtargetVal)`

Description:
his subroutine builds the dual rhs for the estimation of the error of the integral of the normal derivative over a part of the domain boundary. It is currently impossible to choose only a part of a boundary component, only complete boundary components are admissible. In the same computation, the subroutine computes the current approximation to the value of the target functional.

Input variables:

Name	Type	Rank	Description
`iboundarycomp`	`integer`		number of boundary component where the integral shall be computed
`rmatDual`	`Tdiscrid`		primal discretisation
`rmatPrimal`	`Tdiscrid`		dual discretisation
`h_DprimalSol`	`integer`		handle of primal solution

Input/Output variables:

Name	Type	Rank	Description

Output variables:

Name	Type	Rank	Description
h_DdualRhs	integer		handle for dual rhs
dtargetVal	real(DP)		approximate value of target functional

7.26 Module forms

Purpose: This module contains the description of the discretisation in form of the weak formulation. For further information see the example in an earlier section in this manual.

Constant definitions:

Name	Type	Purpose
Purpose: maximal values		
MB_MAXDA	integer	maximum number of BLFs per discretisation
F_MAXAB	integer	maximum number of terms per form object
F_MAXDER	integer	maximum number of derivative types
F_MAXLENBLFNAME	integer	maximal length of the BLF name
Purpose: bilinear form description		
F_FUNC	integer	function value in term
F_DERIV_X	integer	x derivative in term
F_DERIV_Y	integer	y derivative in term
F_DERIV_XX	integer	2nd x derivative in term
F_DERIV_XY	integer	xy derivative in term
F_DERIV_YY	integer	2nd y derivative in term
F_Q1	integer	Q1 quadr., bilinear, conforming element
F_Q2	integer	Q2 quadr., biquadratic, conforming element, hierarchic basis functions
F_Q2L	integer	Q2 quadr., biquadratic, conforming element, Lagrangian basis functions

F_Q3	integer	Q3 quadr., bicubic, conforming element, hierarchic basis functions
F_Q3L	integer	Q3 quadr., bicubic, conforming element, Lagrangian basis functions

Purpose: assembly information

AS_ALL_BLF	integer	Assemble all BLFs of a discretisation
AS_DO_NOT_SAVE_RHS	integer	Flag that assembly routine should NOT save the calculated RHS as default
AS_SAVE_RHS	integer	Flag that assembly routine should save the calculated RHS as default

7.26.1 Function forms_sgetDerivName

Interface:

```
forms_sgetDerivName(cder)
```

Description:
This function returns the derivative id in clear text.

Input variables:

Name	Type	Rank	Description
cder	integer		derivative identifier

Result:
character (len=5) : derivative in clear text

7.26.2 Function forms_igetDeriv

Interface:

```
forms_igetDeriv(sder)
```

Description:
This routine returns the derivative ID for a given derivative in clear text.

Input variables:

Name	Type	Rank	Description
sder	c (5)		derivative in clear text

Result:
integer : integer derivation identifier

7.26.3 Subroutine forms_readDef

Interface:
`forms_readDef(ichan, rdiscr)`

Description:
This routine reads a form description from an open file channel ichan. The format of the form description is described in an earlier section of this manual.

Input variables:

Name	Type	Rank	Description
`ichan`	`integer`		file channel

Output variables:

Name	Type	Rank	Description
`rdiscr`	`Tdiscretization`		discretisation variable

7.26.4 Subroutine forms_writeDef

Interface:
`forms_writeDef(rdiscr)`

Description:
This subroutine prints a form description in clear text.

Input variables:

Name	Type	Rank	Description
`rdiscr`	`Tdiscretization`		discretisation variable

7.26.5 Subroutine forms_send

Interface:
`forms_send(idx, rdiscr)`

Description:
This routine sends a form description to the specified process.

Input variables:

Name	Type	Rank	Description
`rdiscr`	`Tdiscretization`		discretisation variable
`idx`	`integer`		index of the destination process

7.26.6 Subroutine forms_receive

Interface:
`forms_receive(idx, rdiscr)`

Description:
This routine receives a form description from the specified process.

Input variables:

Name	Type	Rank	Description
`rdiscr`	`Tdiscretization`		discretisation variable
`idx`	`integer`		index of the sending process

7.26.7 Function form_sgetBLFname

Interface:
`form_sgetBLFname(rdiscrId)`

Description:
This function returns the name of the BLF corresponding to the given discretisation ID and BLF ID.

Input variables:

Name	Type	Rank	Description
`rdiscrId`	`type(Tdiscrid)`		discretisation identifier

7.26.8 Function form_rgetBLF

Interface:
`form_rgetBLF(rdiscr, sname, bfound)`

Description:
This function returns the BLF corresponding to the given name specified in master.dat.

Input variables:

Name	Type	Rank	Description
`rdiscr`	`type(Tdiscretization)`		main discretisation object
`sname`	`c(*)`		name of the BLF specified in master.dat

Output variables:

Name	Type	Rank	Description
`bfound`	`logical`		

Result:
type(Tdiscrid) : type(Tdiscrid) :: rdiscrId

7.27 Module griddeform

Purpose: This module contains all routines regarding grid distortion, grid smoothing and grid deformation.

Constant definitions:

Name	Type	Purpose
Constants for cmethod		
Purpose: ODE solvers		
GRIDDEF_EXPL_EULER	integer	explicit Euler method
GRIDDEF_AB2	integer	Adams-Bashforth method (linear 2-step method), 2nd order, starting method: Heun
GRIDDEF_AB2EE	integer	Adams-Bashforth method (linear 2-step method), 2nd order, starting method: Euler
GRIDDEF_AB3	integer	Adams-Bashforth method (linear 3-step method), 3rd order, starting method: RK3
GRIDDEF_HEUN	integer	Runge-Kutta method of 2nd order (Heuns method)
GRIDDEF_RK3	integer	Runge-Kutta method of 3rd order
GRIDDEF_RK4	integer	(classical) Runge-Kutta method of 4th order
Constants for csearchmethod		
Purpose: search methods		
GRIDDEF_VERYSLOW	integer	brute-force searching
GRIDDEF_FAST	integer	fast raytracing search method
GRIDDEF_SLOW	integer	brute force searching with heuristical improvement
Constants for crecMethod		
Purpose: recovery in deformation		
GRIDDEF_SPR	integer	use SPR for gradient recovery
GRIDDEF_PPR	integer	use PPR for gradient recovery
GRIDDEF_INTPOL	integer	use plain interpolation for gradient recovery

7.27.1 Subroutine griddef_distort

Interface:
`griddef_distort(rparBlock, imacroIdx, dalpha, rdiscrId)`

Description:
this routine distorts the grid (e.g. for test purposes). Hanging nodes are allowed, even in this case the grid remains admissible, i.e. there is no overlapping on the macro boundaries with hanging nodes. If on the macro to be distorted the element vertices are computed in situ, this is changed by calling the subroutine grid_chModDirect. After griddef_distort, there is no rechange.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`type(t_parBlock)`		the parallelblock structure of the parallel block belonging to the calling slave
`dalpha`	`real(DP)`		parameter for strength of distortion
`imacroIdx`	`integer`		global index of macro to be distorted, -1: all macros
`rdiscrId`	`Tdiscrid`		discretisation identifyer (only needed to be able to use the hlayer-routines)

7.27.2 Subroutine griddef_LaplacianSmooth

Interface:
`griddef_LaplacianSmooth(rparBlock, imacroIdx, dalpha, ncycle, rdiscrId, ImacroList)`

Description:
This routine performs Laplacian smoothing to the current grid, i.e.

$$p_{new} = (1-\alpha)p + \frac{\alpha}{|Adj(p)|} \sum_{q_j \in Adj(p)} q_j.$$

In the case that only one macro may be smoothed, the vertices on the macro boundary are not modified to guarantee the consistency of the grid. Currently, one can choose between a single macro or all macros (imacroIdx = MB_ALL_MB) to be smoothed.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		the parallelblock structure of the parallel block belonging to the calling slave
`rdiscrId`	`Tdiscrid`		discretisation identifyer
`dalpha`	`real(DP)`		parameter for smoothing procedure
`ncycle`	`integer`		number of smoothing cycles
`imacroIdx`	`integer`		global index of the macro to be smoothed, if MB_ALL_MB: all macros smoothed
`ImacroList`	`integer`	:	list with the indices of the macros to be smoothed (optional)

7.27.3 Subroutine griddef_getLevelMin

Interface:

`griddef_getLevelMin(rparBlock, h_Iivertidx, h_IlevMin)`

Description:
This subroutine is to find out for all macro vertices the minimal level of refinement of all macros sharing a vertex. The information is stored in the array IlevMin. The information is sorted macro-wise, i.e. IlevMin(MAC_EPM*(imacro-1) + ivert) is the minimal level of the macros which share the vertex ivert on macro imacro.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		

Output variables:

Name	Type	Rank	Description
`h_IivertIdx`	`integer`		handle of index vector
`h_IlevMin`	`integer`		handle of the vector containing the level information

7.27.4 Subroutine griddef_el2node

Interface:

`griddef_el2node(rparblock, rdiscrId, imgLevel, chlayer, imacroIdx, h_DelemVec, h_DFEMVec)`

Description:
This routine computes a representation of an element-wise defined vector in the FEM space. Both vectors have to be allocated outside this routine. This subroutine does not work with Q3-elements.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`type(t_parBlock)`		the parallelblock structure of the parallel block belonging to the calling slave
`rdiscrId`	`Tdiscrid`		discretisation identifyer
`h_DelemVec`	`integer`		handle of the vector with element-wise defined vector
`imgLevel`	`integer`		multigrid level of the vectors
`chlayer`	integer		hierarchical layer
`imacroIdx`	`integer`		macro on that the grid shall be deformed

Output variables:

Name	Type	Rank	Description
h_DFEMVec	integer		handle of FEM vector

7.27.5 Subroutine griddef_normaliseFctsNum

Interface:

griddef_normaliseFctsNum(rparBlock, rdiscrId, h_Df1, h_Df2, imgLevel, chlayer, imacroIdx)

Description:
This subroutine is to normalize two given FEM functions f_1 and f_2 , i.e. multiply the both functions by suitable constants to achieve

$$\int_\Omega f_1(x)dx = \int_\Omega f_2(x)dx = |\Omega|.$$

The scaling constants are computed by

$$dscale1 = \frac{|\Omega|}{\int_\Omega f_1 dx}, \quad dscale2 = \frac{|\Omega|}{\int_\Omega f_2 dx}.$$

For numerical integration, the routine uses the 2x2-Gauss formula.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block...
rdiscrId	Tdiscrid		discretisation identifyer
h_Df1, h_Df2	integer		handles of the vectors containing the 2 functions
imgLevel	integer		multigrid level
chlayer	integer		hierarchical layer (whole domain, parallel block, single matrix block)
imacroIdx	integer		index of the macro where the grid shall be deformed

7.27.6 Subroutine griddef_normaliseFctsExact

Interface:

griddef_normaliseFctsExact(rparBlock, rdiscrId, rgriddefInfo, h_Df, fmon, imgLevel, chlayer, dscalefAnalytic)

Description:
This subroutine is to normalise a given FEM function f and an analytically given function f_{mon} , i.e. multiply the FEM-function by a certain constant to achieve

$$\int_\Omega f(x)dx = \int_\Omega f_{mon}(x)dx = |\Omega|.$$

and to compute the appropriate scaling constant for the analytically given function. The scaling constants are computed by

$$dscalef = \frac{|\Omega|}{\int_\Omega f dx}, \quad dscalefAnalytic = \frac{|\Omega|}{\int_\Omega f_{mon} dx}.$$

For numerical integration, the routine uses the 2x2-Gauss formula.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block...
rdiscrId	Tdiscrid		discretisation identifyer
rgriddefInfo	type(t_griddefInfo)		
h_Df	integer		handle of the vector containing the FEM functions
imgLevel	integer		multigrid level
chlayer	integer		hierarchical layer (whole domain, parallel block, single matrix block)
dx, dy	real(DP)		monitor function x- and y-coordinates
cderiv	integer		flag indicating whether to return function values or derivatives (One of the constants F_FUNC, F_DERIV_X, F_DERIV_Y etc. defined in forms.f90)
icomp	integer		The component number of the solution vector the query regards (for multidimensional problems)
dregpar	real(DP)		regularisation parameter
dresult	real(DP)		

Output variables:

Name	Type	Rank	Description
dscalefAnalytic	real(DP)		scaling parameter for analytically given function

7.27.7 Subroutine griddef_normaliseFctsInv

Interface:

griddef_normaliseFctsInv(rparBlock, rdiscrId, rgriddefWork, imgLevel, chlayer, imacroIdx)

Description:
This subroutine is to normalize the reciprocals of two given FEM functions f_1 and f_2 , i.e. multiply the both functions by certain constants to achieve

$$\int_\Omega \frac{1}{f_1(x)} dx = \int_\Omega \frac{1}{f_2(x)} dx = |\Omega|.$$

The scaling constants are computed by

$$dscale1 = \frac{|\Omega|}{\int_\Omega \frac{1}{f_1} dx}, \quad dscale2 = \frac{|\Omega|}{\int_\Omega \frac{1}{f_2} dx}.$$

For numerical integration, the routine uses the 2x2-Gauss formula.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block...
rdiscrId	Tdiscrid		discretisation identifyer
rgriddefWork	type(t_griddefWork)		structure containing all vector handles for the deformation algorithm
imgLevel	integer		multigrid level
chlayer	integer		hierarchical level (whole domain, parallel block , single matrix block)
imacroIdx	integer		index of the maccro which shall be deformed

7.27.8 Subroutine griddef_normaliseFctsInvExact

Interface:

griddef_normaliseFctsInvExact(rparBlock, rdiscrId, rgriddefInfo, rgriddefWork, imgLevel, dblendpar, chlayer, dscalefmon, dscalefmonInv)

Description:
This subroutine is to normalize the reciprocals of a given FEM function f_1 and an analytically given function f_2 , i.e. multiply the first function by a constant to achieve

$$\int_\Omega \frac{1}{f_1(x)} dx = \int_\Omega \frac{1}{f_2(x)} dx = |\Omega|.$$

For the analytically given function, the necessary scaling constant is computed. The scaling constants are computed by

$$dscalef = \frac{|\Omega|}{\int_\Omega \frac{1}{f_1} dx}, \quad dscalefmonInv = \frac{|\Omega|}{\int_\Omega \frac{1}{f_2} dx}.$$

As this routine is to be used in grid deformation, the analytic monitor function f can be blended with the area distribution. Therefore, we need the blending parameter in this subroutine. For numerical integration, the routine uses the 2x2-Gauss formula.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block
rdiscrId	Tdiscrid		discretisation identifyer
rgriddefWork	type(t_griddefWork)		structure containing all vector handles for the deformation algorithm
rgriddefinfo	type(t_griddefInfo)		
imgLevel	integer		multigrid level
chlayer	integer		hierarchical level (whole domain, parallel block , single matrix block)

Output variables:

Name	Type	Rank	Description
dscalefmonInv	real(DP)		

7.27.9 Subroutine performDeformation

Interface:

`performDeformation(rparBlock, rsolver, rmatDeform, rgriddefInfo, rgriddefWorkDef, rgriddefWorkCorr, h_Dcontrib, psearchroutine, dsearchtimetotal, dQL2, dqLinfty, bstartNew, bterminate, iiteradapt, ibcIdx)`

Description:
This subroutine is the main routine for the grid deformation process, as all necessary steps are included here. For performing grid deformation, it is sufficient to define a monitor function and call this routine.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block which belongs to the deformation domain
`rsolver`	`t_solver`		saves the Scarc solver
`rMatDeform`	`Tdiscrid`		discretisation for deformation process
`rgriddefInfo`	`type(t_griddefInfo)`		this structure contains all information for grid deformation
`bstartNew`	`logical`		searching method chosen if true, reset all vectors
`bterminate`	`logical`		
`iiterAdapt`	`integer`		number of adaptive iteration
`ibcIdx`	`integer`		index of boundary condition related to the deformation PDE

Input/Output variables:

Name	Type	Rank	Description
`h_Dcontrib`	`integer`		handle of vector with elementwise error contributions

Output variables:

Name	Type	Rank	Description
`dQL2`	`real(DP)`		quality measure Q (L_2)
`dqLinfty`	`real(DP)`		quality measure Q (L_oo)
`dsearchtimetotal`	`real(DP)`		total search time for deformation

7.27.10 Subroutine griddef_moveMesh

Interface:

`griddef_moveMesh(rparBlock, rdiscrId, rgriddefInfo, rgriddefWork, h_IboundDescr, h_Dsteps, h_Devals, psearchRoutine, benforceVerts, ilevelODE, ilevelPDE, dsearchtimetotal)`

Description:
This subroutine performs the actual deformation of the mesh. To do this, the grid points are moved in a vector field represented by DphiX and DphiY, the monitor function Dfmon and the area distribution Darea. The following ODE is solved:

$$\frac{\partial\varphi(x,t)}{\partial t} = \eta(\varphi(x,t),t), \quad 0 \le t \le 1, \quad \varphi(x,0) = x$$

$$\eta(y,s) := \frac{v(y)}{s\tilde{f}(y) + (1-s)\tilde{g}(y)}, \quad y \in \Omega, s \in [0,1].$$

To solve the ODE which describes the movement of the grid points, several ODE solver are avialable. The type of solver is chosen by codeMethod. Currently, supported are:
1) explicit Euler (GRIDDEF_EXPL_EULER)
2) Adams-Bashforth 2 with Heuns method as starting method (GRIDDEF_AB2)
3) Adams-Bashforth 2 with explicit Euler as starting method (GRIDDEF_AB2EE)
4) Heuns method (GRIDDEF_HEUN)
5) RK3 (GRIDDEF_RK3)
6) RK4 (GRIDDEF_RK4)
7) Adams-Bashforth 3 with Heun as starting method (GRIDDEF_AB3)
Formula for AB3 with variable stepsize:
h_k := t_k+1 - t_k
dhelp1 := 0.5 - h_n/(6(h_n-2 + h_n-1))
dhelp2 := h_n/h_n-1 * (h_n + h_n-1)/(h_n-1 + h_n-2)
dhelp3 := 0.5 * h_n/h_n-1
y_n+1 = (1+dhelp3 + dhelp1*dhelp2)f_n-2
- (dhelp3 + dhelp1*dhelp2*(1+ h_n-1/h_n-2) f_n-1
+ dhelp1*dhelp2 hn-1/h_n-2* f_n
Input variables:

Name	Type	Rank	Description
`rparBlock`	`type(t_parBlock)`		the parallelblock structure of the parallel block belonging to the calling slave
`h_IboundDescr`	`integer`		handle of boundary description vector
`rgriddefInfo`	`type(t_griddefInfo)`		grid deformation description structure
`rgriddefWork`	`type(t_griddefWork)`		structure containing all vector handles for the deformation algorithm
`rdiscrId`	`Tdiscrid`		
`benforceVerts`	`logical`		if true, then the corner vertices of the domain are enforced to be represented in the grid
`ilevelODE`	`integer`		searching routine level for solving the ODEs in deformation process
`ilevelPDE`	`integer`		grid level for grid deformation

7.27.11 Subroutine griddef_createRhs

Interface:
`griddef_createRhs(rparBlock, rmatDeform, rgriddefWork, imgLevel, chlayer, imacroIdx)`

Description:

This subroutine creates the right hand for the deformation system side from the (normalized) monitor function and area distribution. The vector of the right hand side has to be created outside this routine. The handle of this vector is stored in in the rgriddefWork structure.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`rmatDeform`	`type(TdiscrId)`		
`rgriddefWork`	`type(t_griddefWork)`		structure containing all vector handles for the deformation algorithm
`imgLevel`	`integer`		multigrid level
`chlayer`	integer		hierarchical level (whole domain, parallel block , single matrix block)
`imacroIdx`	`integer`		index of the macro on which the rhs shall be created (all: MB_ALL_MB)

7.27.12 Subroutine griddef_createRhsExact

Interface:

`griddef_createRhsExact(rparBlock, rmatDeform, rgriddefInfo, rgriddefWork, imgLevel, chlayer, dblendpar, dscalefmon, dscalefmonInv, fmon, imacroIdx)`

Description:
This subroutine creates the right hand for the deformation system side from the (normalised) analytically given monitor function and area distribution. In contrast to the subroutine griddef_createRhs, no interpolation of the monitor function is used, but the monitor function itself. The vector of the right hand side has to be created outside this routine. The handle of this vector is stored in in the rgriddefWork structure.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rmatDeform	type(TdiscrId)		
rgriddefInfo	type(t_griddefinfo)		grid deformation description structure
rgriddefWork	type(t_griddefWork)		structure containing all vector handles for the deformation algorithm
imgLevel	integer		multigrid level
chlayer	integer		hierarchical level (whole domain, parallel block , single matrix block)
imacroIdx	integer		index of the macro on which the rhs shall be created (all: MB_ALL_MB)
dblendpar	real(DP)		current blending parameter
dscalefmon, dscalefmonInv	real(DP)		scaling factors for monitor function and the reciprocal of the blended monitor function
dx, dy	real(DP)		monitor function x- and y-coordinates
cderiv	integer		flag indicating whether to return function values or derivatives (One of the constants F_FUNC, F_DERIV_X, F_DERIV_Y etc. defined in forms.f90)
icomp	integer		The component number of the solution vector the query regards (for multidimensional problems)
dregpar	real(DP)		regularisation parameter
dresult	real(DP)		

7.27.13 Subroutine griddef_evalPhi_VERYSLOW

Interface:

griddef_evalPhi_VERYSLOW(rparBlock, rdiscrId, ilevDefPDEAbs, dxEval, dyEval, bsearchFailed, h_Dfmon, h_DareaVert, dresultX, dresultY, dresultfmonInv, dresultAreaInv, dsearchtime, ielIdx, imacIdx, h_DphiX, h_DphiY)

Description:
This routine performs the evaluation of the vector field (DpihX, DphiY) as well as the evaluation of the monitor function Dfmon in the point (dxEval, dyEval). As these functions are computed by FEM and are elementwisely defined therefore, this evulation requires searching, because it is not known which element element the evaluation point contains. This is done with brute force searching, i.e. starting from the first element, every element is tested until the right one is found.

Input variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		discretisation identifyer
dxEval, dyEval	real(DP)		coordinates of the point to evaluate the function and its gradients at
h_DphiX	integer		handle for x-component of vector field (optional)
h_DphiY	integer		handle for y-component of vector field (optional)
h_Dfmon, h_DareaVert	integer		handle for monitor function
ielIdx	integer		element index of (dxEval,dyEval)
imacIdx	integer		macro index of (dxEval,dyEval)
ilevDefPDEAbs	integer		level for grid deformation process

Input/Output variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block which belongs to the deformation domain
bsearchFailed	logical		true, if the point is not inside the domain

Output variables:

Name	Type	Rank	Description
dresultx	real(DP)		x-component of deformation vector field in (dxEval, dyEval)
dresulty	real(DP)		x-component of deformation vector field in (dxEval, dyEval)
dresultfmonInv, dresultAreaInv	real(DP)		fmon(dxEval, dyEval)
dsearchtime	real(DP)		average searchtime per time step

7.27.14 Subroutine griddef_evalPhi_SLOW

Interface:

griddef_evalPhi_SLOW(rparBlock, rdiscrId, ilevDefPDEAbs, dxEval, dyEval, bsearchFailed, h_Dfmon, h_DareaVert, dresultX, dresultY, dresultfmonInv, dresultAreaInv, dsearchtime, ielIdx, imacIdx, h_DphiX, h_DphiY)

Description:
This routine performs the evaluation of the vector field (DpihX, DphiY) as well as the evaluation of the monitor function Dfmon in the point (dxEval, dyEval). As these functions are computed by FEM and are elementwisely defined therefore, this evulation requires searching, because it is not known which element element the evaluation point contains. This done by the improved brute force approach: the element index where the point belonged to in the last time step is tested first, after this we performe a brute force sarch.

Input variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		discretisation identifyer
dxEval, dyEval	real(DP)		coordinates of the point to evaluate the function and its gradients at
h_DphiX	integer		handle for x-component of vector field (optional)
h_DphiY	integer		handle for y-component of vector field (optional)
h_Dfmon, h_DareaVert	integer		handle for monitor function
ielIdx	integer		element index of (dxEval,dyEval)
imacIdx	integer		macro index of (dxEval,dyEval)
ilevDefPDEAbs	integer		

Input/Output variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block which belongs to the deformation domain
bsearchFailed	logical		true, if the point is not inside the domain

Output variables:

Name	Type	Rank	Description
dresultx	real(DP)		x-component of deformation vector field in (dxEval, dyEval)
dresulty	real(DP)		x-component of deformation vector field in (dxEval, dyEval)
dresultfmonInv, dresultAreaInv	real(DP)		fmon(dxEval, dyEval)
dsearchtime	real(DP)		average searchtime per time step

7.27.15 Subroutine griddef_evalPhi_FAST

Interface:

griddef_evalPhi_FAST(rparBlock, rdiscrId, ilevDefPDEAbs, dxEval, dyEval, bsearchFailed, h_Dfmon, h_DareaVert, dresultX, dresultY, dresultfmonInv, dresultAreaInv, dsearchtime, ielIdx, imacIdx, h_DphiX, h_DphiY)

Description:
This routine performs the evaluation of the vector field (DpihX, DphiY) as well as the evaluation of the monitor function Dfmon in the point (dxEval, dyEval). As these functions are computed by FEM and are elementwisely defined therefore, this evulation requires searching, because it is not known which element element the evaluation point contains. This is done by fast raytracing searching (explanation see paper and diss).

Input variables:

Name	Type	Rank	Description
rdiscrId	Tdiscrid		discretisation identifyer
dxEval, dyEval	real(DP)		coordinates of the point to evaluate the function and its gradients at
h_DphiX	integer		handle for x-component of vector field (optional)
h_DphiY	integer		handle for y-component of vector field (optional)
h_Dfmon, h_DareaVert	integer		handle for monitor function
ielIdx	integer		element index of (dxEval,dyEval)
imacIdx	integer		macro index of (dxEval,dyEval)
ilevDefPDEAbs	integer		

Input/Output variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block which belongs to the deformation domain
bsearchFailed	logical		true, if the point is not inside the domain

Output variables:

Name	Type	Rank	Description
dresultx	real(DP)		x-component of deformation vector field in (dxEval, dyEval)
dresulty	real(DP)		x-component of deformation vector field in (dxEval, dyEval)
dresultfmonInv, dresultAreaInv	real(DP)		fmon(dxEval, dyEval)
dsearchtime	real(DP)		average searchtime per time step

7.27.16 Subroutine griddef_discrProject

Interface:

griddef_discrProject(rparBlock, imgLevel, dx, dy, h_IboundCompDescr_handles, imacroIdxNew, IelIdxNew, iboundIdx)

Description:
This subroutine is to ensure that a boundary grid point remains on the boundary of the discrete domain during the deformation process. For the boundary point (dx,dy), we search the whole boundary, which consists for a discrete domain of line segments. If a projection on a line segment is possible, we calculate the distance between the point and the line segment. After this, we project the point onto the admissible line segment with the minimal projection distance. It may happen, that a vertex is in a gap between to line segements: / / segment2/ domain / / segment1 / _____/ normal-= — —-vector —=— normal ——=— — vector — in this — — area: — in this area: — x projection — projection

— possible — possible — !gap! To handle this, the routine tryProjection detects, if (dx,dy) is located too far left (tryProjection returns 1) or too far right (tryProjection returns -1). If in two following line segments 1 and -1 are returned, the point is in a gap. In this case, we set the point to the endpoint of line segment 1. (to be improved...)

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallelblock structure
imacroIdxNew	integer		index of the macro, to which the element belongs, onto that the point (dx,dy) is projected
IelIdxNew	integer		index of the element, onto that the point (dx,dy) is projected
h_IboundCompDescr_handles	integer		
imgLevel	integer		multigrid level on which the projection shall take place
iboundIdx	integer		boundary index of the point to project

Input/Output variables:

Name	Type	Rank	Description
dx, dy	real(DP)		coordinates of the point to project on discrete boundary

7.27.17 Subroutine griddef_contProject

Interface:

griddef_contProject(rparblock, ilevDefPDEAux, h_DcoordAuxX, h_DcoordAuxY)

Description:
This subroutine projects the moved boundary vertices on the boundary of the exact domain after the deformation process. Currently, only line segments and circles are supported. In the case of NURBS, nothing happens, so that the routine does not harm if applied to grids with NURBS boundaries.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallelblock structure

Input/Output variables:

Name	Type	Rank	Description
h_DcoordAuxX	integer		handle of the coordinate vector for the deformed grid, x-coords
h_DcoordAuxY	integer		handle of the coordinate vector for the deformed grid, y-coords

7.27.18 Subroutine griddef_enforceVerts

Interface:

`griddef_enforceVerts(rparBlock, imgLevel, h_DcoordAuxX, h_DcoordAuxY)`

Description:
This routine is to enfore that the geometric boundary points, e.g. the points a piecewise linear domain boundary consists of, are represented in the grid. To achieve this, the routine loops over all these geometric boundary points and seeks the nearest boundary grid point. Then, this point gets the coordinates of the geometry point.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`type(t_parBlock)`		parallel block data structure
`imgLevel`	`integer`		multigrid level

Input/Output variables:

Name	Type	Rank	Description
`h_DcoordAuxX, h_DcoordAuxY`	`integer`		handles for vectors containing the deformed grid

7.27.19 Subroutine griddef_transferMonFct

Interface:

`griddef_transferMonFct(rparBlock, rmatDeform, rgriddefInfo, rgriddefWork, chlayer)`

Description:
This subroutine has the task to tranfer one FEM-function to another grid. This is used e.g. for monitor functions made from error distributions. These functions have to be transferred to the deformed grid, if another deformation step is desired.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block which belongs to the deformation domain
`rMatDeform`	`Tdiscrid`		discretisation for deformation process
`rgriddefinfo`	`type(t_griddefInfo)`		grid deformation description structure
`rgriddefWork`	`type(t_griddefWork)`		structure containing all vector handles for the deformation algorithm
`chlayer`	integer		hierarchical layer for transfer

7.27.20 Subroutine griddef_blendmonitor

Interface:

`griddef_blendmonitor(rparblock, rdiscrId, rgriddefWork, imgLevel, dblendPar, chlayer, imacroIdx)`

Description:
This subroutine performs the bleding between the monitor function and the area distribution function during the grid deformation process. The blending performed is a linear combination of these two functions: fmon → t * fmon + (1-t)*darea. If the deformation is too harsh, the resulting grid can become invalid due to numerical errors. In this case, one performs several deformations with the blended monitor function, such that every single step results in relatively mild deformation.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		stores the parallelblock structure
`rdiscrId`	`Tdiscrid`		discretisation identifyer
`rgriddefWork`	`type(t_griddefWork)`		structure containing all handles for vectors necessary for deformation
`dblendPar`	`real(DP)`		blending parameter
`imgLevel`	`integer`		multigrid level on which the blending may take place
`chlayer`	integer		hierarchical layer
`imacroIdx`	`integer`		

7.27.21 Subroutine griddef_gradient

Interface:

`griddef_gradient(rparBlock, rdiscrId, imgLevel, h_Dsol, h_DrecGradX, h_DrecGradY, chlayer)`

Description:
This subroutine computes a suitable interpolation of the gradient of the function stored in Dsol into the FEM space. For the bilinear case, the value of the gradient in a vertex is the arithmetic mean of the values of the gradients in this point in the elements surrounding this point. In the Q2/Q3-case, additionally to this, the values of DOS on edges are the arithmetic means of the two corresponding gradient values on the elements sharing this edge. For Dofs inside element, the values of the gradient are taken. Gradient vectors have to be allocated before!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		structure describing the parallel-block belonging to the calling slave
h_Dsol	integer		handle of solution
h_DrecGradX	integer		handle of x-derivative of solution
h_DrecGradY	integer		handle of y-derivative of solution
celType	integer		element type
imgLevel	integer		multigrid level
chlayer	integer		hierarchical layer

7.27.22 Subroutine griddef_performOneDefStep

Interface:

griddef_performOneDefStep(rparBlock, rmatDeform, rsolver, rgriddefInfo, rgriddefWork, chlayer, imacroIdx, fmon, dblendpar, ilevelODE, ilevelPDE, rstat, h_iboundCompDescr_handles, benforceVerts, dsearchTimeTotal, psearchroutine, h_Dtimesteps, h_Devals, h_Dcontrib, bstartNew, iiterAdapt, ibcIdx)

Description:
This subroutine performs one deformation step of the enhanced deformation method.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block which belongs to the deformation domain
rMatDeform	Tdiscrid		discretisation for deformation process
rgriddefinfo	type(t_griddefInfo)		grid deformation description structure
rgriddefWork	type(t_griddefWork)		structure containing all vector handles for the deformation algorithm
chlayer	integer		
imacroIdx	integer		index of the macro where to perform deformation
benforceverts	logical		
ilevelODE	integer		absolute level on which the points are moved
ilevelPDE	integer		absolute level on which the deformation PDE is solved
dblendpar	real(DP)		blending parameter for monitor-function sf + (1-s)g
h_Dcontrib	integer		
bstartNew	logical		if true, rebuild all vectors from scratch
iiterAdapt	integer		number of adaptive iteration

7.27.23 Subroutine griddef_deformationInit

Interface:

`griddef_deformationInit(rparBlock, rgriddefInfo)`

Description:
This subroutine initialises the rgriddefinfo structure which describes the grid deformation algorithm. The values for the certain parameters defining the deformation process are read in from the master.dat file. This routine has to be called before strating grid deformation.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		stores the parallelblock structure

Output variables:

Name	Type	Rank	Description
`rgriddefInfo`	`t_griddefInfo`		grid deformation information structure

7.27.24 Subroutine griddef_createMonitorFunction

Interface:

`griddef_createMonitorFunction(rparBlock, rgriddefWork, rgriddefInfo, rmatDeform, chlayer, iiterAdapt)`

Description:
This subroutine has the delicate task to create a suitable monitor function from a given error distribution.

Input variables:

Name	Type	Rank	Description
`rparBlock !parallelblock`	`t_parBlock`		
`rgriddefWork`	`type(t_griddefWork)`		structure containing all vector handles for the deformation algorithm
`rgriddefInfo`	`type(t_griddefInfo)`		
`chlayer`	integer		
`iiterAdapt`	`integer`		

7.27.25 Subroutine griddef_writeCoords

Interface:

`griddef_writeCoords(rparBlock, rgriddefInfo, rgriddefWork)`

Description:
This subroutine serves as auxiliary subroutine for griddef_performDeformation. Here, the coordinate vectors of all levels are written.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parblock)		parallel block which belongs to the deformation domain
rgriddefInfo	type(t_griddefInfo)		grid deformation information structure
rgriddefWork	type(t_griddefWork)		structure containing all vector handles for the deformation algorithm

7.27.26 Subroutine griddef_mon2aux

Interface:

griddef_mon2aux(rparBlock, rgriddefInfo, rgriddefWork, ilevelPDE, chlayer)

Description:
This subroutine transfers the FEM monitor function to an aux vector to avoid the loss of the monitor function by overwriting the corresponding vector. The monitor function itself is always defined on the finest multigrid level. In contrast to this, the aux vector is defined on the multigrid level on which the solution of the deformation PDE is to take place.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parblock)		the parallelblock structure of the parallel block belonging to the calling slave
rgriddefinfo	type(t_griddefInfo)		grid deformation description structure
rgriddefWork	type(t_griddefWork)		structure containing all vector handles for the deformation algorithm
ilevelPDE	integer		multigrid level for deformation PDE
chlayer	integer		hierarchical layer for deformation

7.27.27 Subroutine griddef_LaplacianSmoothVec

Interface:

griddef_LaplacianSmoothVec(rparBlock, imacroIdx, dalpha, ncycle, imgLevel, rdiscrId, h_Dvec, ImacroList)

Description:
This routine performs Laplacian smoothing to a given FEM-function, i.e.

$$p_{new} = (1-\alpha)p + \frac{\alpha}{|Adj(p)|} \sum_{q_j \in Adj(p)} q_j.$$

In the case that the function may be smoothed on one macro only, the values on the macro boundary are not modified to guarantee the consistency of the function. Currently, one can choose between a single macro or all macros (imacroIdx = MB_ALL_MB) to be smoothed.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		the parallelblock structure of the parallel block belonging to the calling slave
rdiscrId	Tdiscrid		discretisation identifyer
dalpha	real(DP)		parameter for smoothing procedure
ncycle	integer		number of smoothing cycles
imacroIdx	integer		global index of the macro to be smoothed, if MB_ALL_MB: all macros smoothed
ImacroList	integer	:	list with the indices of the macros to be smoothed (optional)
h_Dvec	integer		handle for the FEM-function
imgLevel	integer		multigrid level for smoothing

7.27.28 Subroutine griddef_checkGrid

Interface:

griddef_checkGrid(rparBlock, rgriddefWork, bdegenerated, chlayer)

Description:
This subroutine checks after one deformation step, if the new grid, which is stored in DcoordAux, is admissible. To do so, we compute the Jacobian determinant of the transformation from the reference element in the four vertices of every element. If this determinant is lower than 0, the element is not convex any more and therefore, the deformation has failed. If so, the routine is terminated and the flag bgenerated is set to .TRUE.. Furthermore, this routine sends a message to the master indicating the index tangled element and the corresponding macro.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block structure
rgriddefWork	type(t_griddefWork)		handle strucxture for deformation
chlayer	integer		hierarchical layer

Output variables:

Name	Type	Rank	Description
bdegenerated	logical		if true, there is a tangled element

7.28 Module structmech

Purpose: This module contains different routines for structural mechanics. Especially it realises a consequent element-wise assembling strategy which is commonly used in the structural mechanics community

and thus should make the extension to more complicated "elements" more convenient. (In the structural mechanics community the corresponding element routines are simply called "elements" which should not be mistaken for the "mathematical" usage of this term.)

Constant definitions:

Name	Type	Purpose
Purpose: array sizes		
STRUC_MAX_NUM_MAT_PAR	integer	maximal number of material parameters
Purpose: formulations		
STRUC_PLANE_STRAIN	integer	
STRUC_PLANE_STRESS	integer	
STRUC_DISPL	integer	
STRUC_DISPL_PRESS	integer	
Purpose: material models		
STRUC_STVENANT	integer	
STRUC_NEOHOOKE	integer	
STRUC_NEOHOOKE2	integer	
Purpose: deformation		
STRUC_SMALL_DEFORM	integer	
STRUC_FINITE_DEFORM	integer	
Purpose: flags		
STRUC_DO_NOT_CALC_U	integer	
STRUC_CALC_U	integer	
Purpose: derivatives		
STRUC_FU	integer	
STRUC_DX	integer	
STRUC_DY	integer	

7.28.1 Subroutine struc_init

Interface:

`struc_init(rparBlock, nspatDim, nvtEl, nnodEl, ndofNode, Rblmat, H_Duvec, DmatPar, cform, cdeform, cmatMod, bnumTang, dpertFact, dstab, boutElData)`

Description:
This routine initialises some variables and allocates necessary storage. The routine has to be called before the assembly process.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
nspatDim	integer		spatial dimension of the problem
nvtEl	integer		number of vertices per element (e.g. 4 for quadrilaterals)
nnodEl	integer		number of nodes per element (e.g. 9 for Q2)
ndofNode	integer		number of degrees of freedom per node (=dimension of the block structures)
Rblmat	Tdiscrid	dim(ndofNode, ndofNode)	system block matrix
H_Duvec	integer	ndofNode	solution vector handle
DmatPar	real(DP)	STRUC_MAX_NUM_MAT_PAR	material parameters
cform	integer		formulation (pure displacement or displacement-pressure)
cdeform	integer		flag if small deformation
cmatMod	integer		material model
bnumTang	logical		flag if analytical or numerical shall be used
dpertFact	real(DP)		perturbation factor for the numerical computation of the consistent tangent
dstab	real(DP)		stabilisation parameter alpha for the stabilised mixed formulation
boutElData	logical		flag if element stiffness matrices / residua shall be displayed on the screen

7.28.2 Subroutine struc_setParameters

Interface:

struc_setParameters(DmatPar, cform, cdeform, cmatMod)

Description:
This routine sources out some parameter settings. The reason is that these parameters have to be reset on the master processor when visual output is performed.

Input variables:

Name	Type	Rank	Description
DmatPar	real(DP)	STRUC_MAX_NUM_MAT_PAR	material parameters
cform	integer		formulation (pure displacement or displacement-pressure)
cdeform	integer		flag if small deformation
cmatMod	integer		material model

7.28.3 Subroutine struc_calcInnerForces

Interface:

`struc_calcInnerForces(rparBlock, imacroIdx, imgLevel, rdiscrInfo, H_DuVec, H_Dres)`

Description:
This routine calculates the inner forces $\int_O megaS(u_n) : E$ for the Newton-Raphson scheme. The computed values are added to the vector H_Dres.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imacroIdx`	`integer`		selected macro(s)
`imgLevel`	`integer`		selected mg level(s)
`rdiscrInfo`	`type(Tdiscrinfo)`		Matrixblock objects of the same discretisation. Needed for cubature formula.
`H_Duvec`	`integer`	ndofPerNode	solution vector handle

Output variables:

Name	Type	Rank	Description
`H_Dres`	`integer`	ndofPerNode	residuum vector handle

7.28.4 Function struc_calcVolume

Interface:

`struc_calcVolume(rparBlock, imacroIdx, imgLevel, rdiscrInfo, H_DuVec)`

Description:
This function calculates the volume of the deformed body on mg level imgLevel.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imacroIdx`	`integer`		selected macro(s)
`imgLevel`	`integer`		selected mg level(s)
`rdiscrInfo`	`type(Tdiscrinfo)`		Matrixblock objects of the same discretisation. Needed for cubature formula.
`H_Duvec`	`integer`	ndofPerNode	solution vector handle

Output variables:

Name	Type	Rank	Description

7.28.5 Subroutine struc_elementRoutine

Interface:

`struc_elementRoutine(imacro, imgLevel, nnodes, IdofGlob, nvert, isw, ccalcElemU)`

Description:
This is a routine for elementwise assembly of the global stiffness matrix and the residual vector for the Newton-Raphson scheme. It is called PER ELEMENT during the global assembly process. Instead of considering every scalar BLF (matrix) seperately (as it was done by now), the whole system is treated - elementwise! So, for example in the 2D linear elasticity case, one does not have to define four BLFs for the Lame equation (one uncoupled for every component and two coupled equations), but one has to define the element matrix and the residual for the Newton Raphson scheme here. For the 2D linear elasticity case with Q1 this is a (8x8)-matrix and a 8-vector, respectively (8 = number of nodes per element * DOF per node). These entities have to be ordered by DOF first, then by node number. So it is not Dres = (ux_1, uy_1, ux_2, uy_2, ux_3, uy_3, ux_4, uy_4), but Dres = (ux_1, ux_2, ux_3, ux_4, uy_1, uy_2, uy_3, uy_4) where in ux_e e is the local number of the node and x stands for x-direction. The global matrix is then block-structured accordingly. At the end, the results for the "BLF-ansatz" and the "element ansatz" are, of course, the same, but the latter is commonly used in the structural mechanics community. That is why the calling structure of the FE program FEAP is resembled: existing "elements" (as this element assembly subroutines are usually called - not to be mistaken for the element Q1!) can be simply "recoded" here. As this routine is called per element in the global assembly routine, it is assumed that the global variables in element.f90 are already set.

Input variables:

Name	Type	Rank	Description
`imacro`	`integer`		index of the macro for which the matrix has to be assembled
`imgLevel`	`integer`		multigrid level
`nnodes`	`integer`		number of nodes in the element
`IdofGlob`	`integer`	:	global indices of the element nodes
`nvert`	`integer`		number of vertices in the element
`isw`	`integer`		switch for steering the calculation
`ccalcElemU`	integer		flag if DuEl has to be calculated

7.28.6 Subroutine struc_disp2dSmall

Interface:

`struc_disp2dSmall(Du, Dcoord, nnodes, isw, Dmat, Dres)`

Description:
Element routine for a 2D small deformation plane strain formulation with pure displacement element.

Input variables:

Name	Type	Rank	Description
Du	real(DP)	:, :	Element solution parameters (Du(*,1) = displacements)
Dcoord	real(DP)	:, :	Element nodal coordinates
nnodes	integer		number of nodes (Q1: 4, Q2: 9)
isw	integer		switch

Output variables:

Name	Type	Rank	Description
Dmat	real(DP)	:, :	element stiffness matrix
Dres	real(DP)	:	element residual vector

7.28.7 Subroutine struc_disp2dFinite

Interface:
struc_disp2dFinite(Du, Dcoord, nnodes, isw, Dmat, Dres)

Description:
Element routine for a 2D plane strain finite deformation formulation with pure displacement element.

Input variables:

Name	Type	Rank	Description
Du	real(DP)	:, :	Element solution parameters (Du(*,1) = displacements)
Dcoord	real(DP)	:, :	Element nodal coordinates
nnodes	integer		number of nodes (Q1: 4, Q2: 9)
isw	integer		switch

Output variables:

Name	Type	Rank	Description
Dmat	real(DP)	:, :	element stiffness matrix
Dres	real(DP)	:	element residual vector

7.28.8 Subroutine struc_mixed2dSmall

Interface:
struc_mixed2dSmall(Du, Dcoord, nnodes, isw, Dmat, Dres)

Description:
Element routine for a 2D plane strain formulation with mixed displacement/pressure element.

Input variables:

Name	Type	Rank	Description
`Du`	`real(DP)`	:, :	Element solution parameters (Du(*,1) = displacements)
`Dcoord`	`real(DP)`	:, :	Element nodal coordinates
`nnodes`	`integer`		number of nodes (Q1: 4, Q2: 9)
`isw`	`integer`		switch

Output variables:

Name	Type	Rank	Description
`Dmat`	`real(DP)`	:, :	element stiffness matrix
`Dres`	`real(DP)`	:	element residual vector

7.28.9 Subroutine struc_mixed2dFinite

Interface:

`struc_mixed2dFinite(Du, Dcoord, nnodes, isw, Dmat, Dres)`

Description:
Element routine for a finite 2D plane strain formulation with mixed displacement/pressure element.

Input variables:

Name	Type	Rank	Description
`Du`	`real(DP)`	:, :	Element solution parameters (Du(*,1) = displacements)
`Dcoord`	`real(DP)`	:, :	Element nodal coordinates
`nnodes`	`integer`		number of nodes (Q1: 4, Q2: 9)
`isw`	`integer`		switch

Output variables:

Name	Type	Rank	Description
`Dmat`	`real(DP)`	:, :	element stiffness matrix
`Dres`	`real(DP)`	:	element residual vector

7.28.10 Subroutine struc_calcStabCoeffs

Interface:

`struc_calcStabCoeffs(DcoeffStab)`

Description:

Calculate coefficients needed for enhanced stabilisation. All needed variables are globally known in this module.

Output variables:

Name	Type	Rank	Description
`DcoeffStab`	`real(DP)`	4	array which contains the 4 coefficients

7.28.11 Subroutine struc_calcValContEq

Interface:

`struc_calcValContEq(DvalU, DcoeffStab, DvalContEq)`

Description:
Calculates the values needed for the continuity equation in the current cub. point.

Input variables:

Name	Type	Rank	Description
`DvalU`	`real(DP)`	:, :	Values of u in the current cub. point
`DcoeffStab`	`real(DP)`	:	coefficients needed for stabilisation enhanced stab.: DcoeffStab(1:3) coefficients for directional derivatives DcoeffStab(4) = -dalpha/(4**imgLevelDiff * 2*mu)

Output variables:

Name	Type	Rank	Description
`DvalContEq`	`real(DP)`	:	values needed for element assembly (DvalContEq(1)*el_dbas(i,F_FUNC), DvalContEq(2)*el_dbas(i,F_DERIV_X), DvalContEq(2)*el_dbas(i,F_DERIV_Y))

7.28.12 Subroutine struc_calcU

Interface:

`struc_calcU(Du, nnodes, DvalU, CactiveDeriv)`

Description:
Calculates values of u in the current cubature point.

Input variables:

Name	Type	Rank	Description
Du	real(DP)	:, :	Element solution parameters (Du(*,1) = displacements)
nnodes	integer		number of nodes (Q1: 4, Q2: 9)
CactiveDeriv	integer	:	information per DOF which derivatives are needed

Output variables:

Name	Type	Rank	Description
DvalU	real(DP)	:, :	

7.28.13 Subroutine struc_calcStrainTensor

Interface:
struc_calcStrainTensor(DvalU, DstrainTens)

Description:
Calculates the strains in one point using the current displacements in this point stored in DvalU.

Input variables:

Name	Type	Rank	Description
DvalU	real(DP)	:, :	Values of u in the current cub. point

Output variables:

Name	Type	Rank	Description
DstrainTens	real(DP)	:	

7.28.14 Subroutine struc_calcSecondPK

Interface:
struc_calcSecondPK(DvalU, DstressTens)

Description:
Calculates the second Piola-Kirchhoff stress tensor in the current cubature point using the current displacements stored in DvalU.

Input variables:

Name	Type	Rank	Description
DvalU	real(DP)	:, :	Values of u in the current cub. point

Output variables:

Name	Type	Rank	Description
`DstressTens`	`real(DP)`	:	the stresses to be computed

7.28.15 Subroutine struc_calcFirstPK

Interface:
`struc_calcFirstPK(DvalU, DfirstPKstress)`

Description:
Calculates the first Piola Kirchhoff stress tensor in the current cubature point using the current displacements stored in DvalU.

Input variables:

Name	Type	Rank	Description
`DvalU`	`real(DP)`	:, :	Values of u in the current cub. point

Output variables:

Name	Type	Rank	Description
`DfirstPKstress`	`real(DP)`	:, :	the stresses to be computed

7.28.16 Subroutine struc_calcMaterialTensor

Interface:
`struc_calcMaterialTensor(DvalU, DmatTensor)`

Description:
Calculates the incremental material tensor.

Input variables:

Name	Type	Rank	Description
`DvalU`	`real(DP)`	:, :	Values of u in the current cub. point

Output variables:

Name	Type	Rank	Description
`DmatTensor`	`real(DP)`	:, :	the material tensor to be computed

7.28.17 Subroutine struc_calcBmatrix

Interface:

`struc_calcBmatrix(DvalU, ibas, DB)`

Description:
Routine for computing the "B-matrix" corresponding to local basis function ibas. The basis functions, Jacobian etc. are assumed to already be computed. The "B-matrix" represents the strain tensor eps(phi_ibas) computed for the current basis function phi_ibas in the current cubature point.

Input variables:

Name	Type	Rank	Description
`DvalU`	`real(DP)`	:, :	values of u in the current cubature point
`ibas`	`integer`		local index of current basis function

Output variables:

Name	Type	Rank	Description
`DB`	`real(DP)`	3, 2	matrix B

7.28.18 Subroutine struc_calcNumericalTangent

Interface:

`struc_calcNumericalTangent(Du, nnodes, bsymmetric, CactiveDeriv, dcubWeight, DmatElem)`

Description:
This routine numerically computes the consistent tangent for the Newton method by applying a finite difference scheme to the stress tensor.

Input variables:

Name	Type	Rank	Description
`Du`	`real(DP)`	:, :	Element solution parameters (Du(*,1) = displacements)
`nnodes`	`integer`		number of nodes (Q1: 4, Q2: 9)
`bsymmetric`	`logical`		flag if matrix is symmetric
`CactiveDeriv`	`integer`	ndofPerNode	information per DOF which derivatives are needed
`dcubWeight`	`real(DP)`		cubature weight

Input/Output variables:

Name	Type	Rank	Description
`DmatElem`	`real(DP)`	:, :	element stiffness matrix

7.28.19 Subroutine struc_calcNumericalTangentMixed

Interface:

`struc_calcNumericalTangentMixed(Du, nnodes, bsymmetric, CactiveDeriv, dcubWeight, DcoeffStab, DmatElem)`

Description:
This routine numerically computes the consistent tangent for the Newton method for mixed formulation by applying a finite difference scheme to the stress tensor.

Input variables:

Name	Type	Rank	Description
`Du`	`real(DP)`	:, :	Element solution parameters (Du(*,1) = displacements)
`nnodes`	`integer`		number of nodes (Q1: 4, Q2: 9)
`bsymmetric`	`logical`		flag if matrix is symmetric
`CactiveDeriv`	`integer`	ndofPerNode	information per DOF which derivatives are needed
`DcoeffStab`	`real(DP)`	:	coefficients needed for stabilisation enhanced stab.: DcoeffStab(1:3) coefficients for directional derivatives DcoeffStab(4) = -dalpha/(4**imgLevelDiff*2*mu)
`dcubWeight`	`real(DP)`		cubature weight

Input/Output variables:

Name	Type	Rank	Description
`DmatElem`	`real(DP)`	:, :	element stiffness matrix

7.28.20 Subroutine struc_printDisplacements

Interface:

`struc_printDisplacements(rparBlock, ioutputLevel, H_Du, iunit)`

Description:
This routine outputs the displacements u on grid level ioutputLevel for each macro of the parallel block. iunit can be connected to a file or to the standard output.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		
`ioutputLevel`	`integer`		
`H_Du`	`integer`	:	
`iunit`	`integer`		

7.28.21 Subroutine struc_getDisplacements

Interface:

`struc_getDisplacements(rparBlock, isearchLevel, H_Du, Dpoints, Dresults)`

Description:
This routine returns the displacements of the points in Dpoints. The points have to be given in cartesian coordinates. The routine runs through ALL points on grid level isearchLevel. If a point is not found in the grid, this information is displayed and the corresponding displacements in Dresults are set to SYS_MAXREAL. The array Dresults is allocated here, so do not allocate before! The search can be optimised by first looking for the macro which contains the point (will be done later).

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
isearchLevel	integer		
H_Du	integer	:	
Dpoints	real(DP)	:, :	
Dresults	real(DP)	:, :	

7.29 Module tools

Purpose: The module contains routines for calculating norms and scalar products on different hierachical layers.

Constant definitions:

Name	Type	Purpose
Purpose: Norm and error constants		
TOOLS_L2_ERR	integer	
TOOLS_L2_NORM_FE	integer	
TOOLS_L2_NORM_ANALYT	integer	
TOOLS_GRAD_ERR	integer	Gradient error and norm
TOOLS_GRAD_NORM_FE	integer	
TOOLS_GRAD_NORM_ANALYT	integer	
TOOLS_LINF_ERR	integer	
TOOLS_LINF_NORM_FE	integer	
TOOLS_LINF_NORM_ANALYT	integer	
Purpose: Position indicators in result array of norm/error calculation		
TOOLS_IDX_L2_ERR	integer	

TOOLS_IDX_L2_NORM_FE	integer	
TOOLS_IDX_L2_NORM_ANALYT	integer	
TOOLS_IDX_GRAD_ERR	integer	Gradient error and norm
TOOLS_IDX_GRAD_NORM_FE	integer	
TOOLS_IDX_GRAD_NORM_ANALYT	integer	
TOOLS_IDX_LINF_ERR	integer	
TOOLS_IDX_LINF_NORM_FE	integer	
TOOLS_IDX_LINF_NORM_ANALYT	integer	
TOOLS_NORM_ERR_ARRAY_SIZE	integer	length of the result array

7.29.1 Subroutine tools_init

Interface:

`tools_init(rparBlock, imglev, rdiscr, bUseScaledNorm)`

Description:
This routine initialises the data stuctures necessary to calculate norms and scalar products. It has to be called once at program start.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
imglev	integer		maximum multigrid level
rdiscr	Tdiscretization		discretization structure
bUseScaledNorm	logical		flag for using scaled norm

7.29.2 Subroutine tools_calcNormError

Interface:

`tools_calcNormError(rparBlock, chLayer, imgLevel, imatBlock, Rmatrix, H_Dsol, pexactSol, Cselected, Dparam, Dresults)`

Description:
This routine calculates L_2 -errors and norms, gradient errors and norms and L_∞ -errors and norms.
For errors the difference between the FEM solution and the exact solution in cubature points of the 3x3 Gauss cubature rule is computed. Norms are computed the same way for both the FEM solution or the exact solution. These operations are supported for every component given in H_Dsol. Use a bit field deploying the constants defined above (TOOLS_L2_*, TOOLS_GRAD_*, TOOLS_LINF_*) to instruct the routine which operations shall be done for which solution component. The results are stored in the matrix Dresults: Per component you can address a particular result using one of the constants TOOLS_IDX_L2_*, TOOLS_IDX_GRAD_*, TOOLS_IDX_LINF_*

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallelblock object
chLayer	integer		hierachical layer
imgLevel	integer		multigrid level
imatBlock	integer		matrixblock index
Rmatrix	Tdiscrid	dim(:)	System matrix, used to determine element type per macro
H_Dsol	integer	:	handle array for vectors containing the FEM solution
Cselected	integer	:	function for evaluating the exact solution array containing the information which norms/errors shall be computed (per component)
Dparam	real(DP)	:	parameters for pexactSol

Output variables:

Name	Type	Rank	Description
Dresults	real(DP)	:, :	array containing the desired norms and errors per solution component

7.29.3 Function tools_dl2errorDiscr

Interface:

```
tools_dl2errorDiscr(rparBlock, chLayer, imgLevel, h_Dvec, imatBlock, rstat, rdiscrid, pexactSol, doptpar)
```

Description:
This function computes the l_2 -error for the given vector and hierachical layer. In the case of Q2 and Q3, this routines works for one macro only. The l_2 -error is regarded to be the error in the vertices only.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rdiscrid	Tdiscrid		matrix identifier
chLayer	integer		hierachical layer
imgLevel	integer		multigrid level
h_Dvec	integer		handle of vector containing the FEM function
imatBlock	integer		matrixblock index
doptpar	real(DP)		function for evaluating exact solution optional parameter for pexactSol

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

Result:
real(DP) : l2-error of the given vector

7.29.4 Subroutine tools_interpolateAnalyticFct

Interface:
`tools_interpolateAnalyticFct(rparBlock, chLayer, imgLevel, h_Dvec, imacroIdx, rstat, rdiscrid, pexactSol, doptPar)`

Description:
This subroutine returns in the given vector the given exact solution. It works for Q2 on 1 macro only, and for Q3 it does not work at all.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rdiscrid	Tdiscrid		matrix identifier
chLayer	integer		hierachical layer
imgLevel	integer		multigrid level
h_Dvec	integer		handle of the vector containing the interpolant
imacroIdx	integer		matrixblock index
doptPar	real(DP)		function for evaluating (to save an addtional interface definition, it is called pexactSol here) optional parameter

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

Result:
: FEM-interpolant of the analytically given function

7.29.5 Function tools_dnorm

Interface:
`tools_dnorm(rparBlock, chLayer, imglev, h_Dvec, imatBlock, rstat, rdiscrid)`

Description:
This function computes the l_2 -norm for the given vector and hierachical layer.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`rdiscrid`	`Tdiscrid`		matrix identifier
`chLayer`	integer		hierachical layer
`imglev`	`integer`		multigrid level
`h_Dvec`	`integer`		vector handle
`imatBlock`	`integer`		matrixblock index

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		number of operations performed + time needed

Result:
real(DP) : l2 norm

7.29.6 Function tools_dnormEl

Interface:
`tools_dnormEl(rparBlock, chLayer, imglev, h_Dvec, imatBlock, rstat, rdiscrid)`

Description:
This function computes the l_2 -norm for the given vector and hierachical layer. The vector is assumed to be defined elementwise.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`rdiscrid`	`Tdiscrid`		matrix identifier
`chLayer`	integer		hierachical layer
`imglev`	`integer`		multigrid level
`h_Dvec`	`integer`		vector handle
`imatBlock`	`integer`		matrixblock index

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

Result:
real(DP) : l2 norm

7.29.7 Function tools_dscalProd

Interface:
`tools_dscalProd(rparBlock, chLayer, imglev, ivaridx1, ivaridx2, imatBlock, rstat, rdiscrid)`

Description:
This function computes the l_2 -scalar product for the given vectors and hierachical layer.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rdiscrid	Tdiscrid		matrix identifier
chLayer	integer		hierachical layer
imglev	integer		multigrid level
ivaridx1	integer		vector handle 1
ivaridx2	integer		vector handle 2
imatBlock	integer		matrixblock index

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

Result:
real(DP) : l2 scalar product

7.29.8 Function tools_dsum

Interface:
`tools_dsum(rparBlock, chLayer, imgLevel, h_Dvec, imatBlock, rstat, rdiscrId)`

Description:
This function computes the sum of the components of the given vector on given multigrid level and hierachical layer.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
chLayer	integer		hierachical layer
imgLevel	integer		multigrid level
h_Dvec	integer		vector handle
imatBlock	integer		matrixblock index (-1 means: all matrix blocks)
rdiscrId	Tdiscrid		matrix identifier

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

Result:
real(DP) : sum of components

7.29.9 Function tools_dmeanValue

Interface:
`tools_dmeanValue(rparBlock, chLayer, imgLevel, h_Dvec, imatBlock, rstat, rdiscrId, bscaleToZero)`

Description:
This function computes the mean value of the components of the given vector on given multigrid level and hierachical layer. If bscaleToZero is true, then the vector is scaled to mean value of zero. The result of the function will be the mean value of the non-scaled vector! For Q2/Q3, we shift only the DOFs in the vertices, as when shifting the represented function, the quadratic/cubic corrections are not changed. BUG!!!!!!!!!!!!!! muss noch implementiert werden!!!!!!!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
chLayer	integer		hierachical layer
imgLevel	integer		multigrid level
h_Dvec	integer		handle of the vector to be normalised
imatBlock	integer		matrixblock index (-1 means: all matrix blocks)
rdiscrId	Tdiscrid		matrix identifier
bscaleToZero	logical		if true, then the vector h_Dvec will be scaled such that the sum is zero

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

Result:
real(DP) : mean value of the unscaled vector

7.29.10 Subroutine tools_getNElem

Interface:
`tools_getNElem(rparBlock, chLayer, imglev, imatBlock, nel)`

Description:
This routine computes the number of elements for the given hierachical layer.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
chLayer	integer		hierachical layer
imglev	integer		multigrid level
imatBlock	integer		matrixblock index

Output variables:

Name	Type	Rank	Description
nel	integer		number of elements

7.29.11 Subroutine tools_getNVert

Interface:
`tools_getNVert(rparBlock, chLayer, imglev, imatBlock, rdiscrid, nvt)`

Description:
This routine computes the number of vertices for the given hierachical layer.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
chLayer	integer		hierachical layer
imglev	integer		multigrid level
imatBlock	integer		matrixblock index
rdiscrid	Tdiscrid		

Output variables:

Name	Type	Rank	Description
`nvt`	`integer`		number of vertices

7.29.12 Subroutine tools_getGridMass

Interface:
`tools_getGridMass(rparBlock, chLayer, imglev, imatBlock, dhmin, dar)`

Description:
This routine calculates key units like min(h) and the aspect ratio of the underlying grid.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`chLayer`	integer		hierachical layer
`imglev`	`integer`		multigrid level
`imatBlock`	`integer`		matrixblock index

Output variables:

Name	Type	Rank	Description
`dhmin`	`real(DP)`		minimal grid size
`dar`	`real(DP)`		maximal aspect ratio

7.29.13 Subroutine tools_getGridMassPartition

Interface:
`tools_getGridMassPartition(rparBlock, dhmin, dar)`

Description:
This routine calculates key units like min(h) and the aspect ratio of the underlying grid.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object

Output variables:

Name	Type	Rank	Description
dhmin	real(DP)		minimal grid size
dar	real(DP)		maximal aspect ratio

7.29.14 Subroutine tools_getGridMassPartitionAll

Interface:
tools_getGridMassPartitionAll(rparBlock, nparblock, dhmin, dar)

Description:
This routine calculates key units like min(h) and the aspect ratio of the underlying grid.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
nparblock	integer		number of parallel blocks

Output variables:

Name	Type	Rank	Description
dhmin	real(DP)		minimal grid size
dar	real(DP)		maximal aspect ratio

7.29.15 Function tools_dintVal

Interface:
tools_dintVal(rparBlock, chLayer, imgLevel, h_Dfmon, imatBlock, rstat, rdiscrId, bscaleToZero)

Description:
This subroutine is to normalise the given monitor function f_{mon} i.e. compute $f_{normMon}$ ith

$$\int_{\Omega} (f_{normMon} - 1)\, dx = 0.$$

This is achieved by scaling the function f_mon with

$$dscale = \frac{|\Omega|}{\int_{\Omega} f_{mon} dx}$$

For numerical integration, the routine uses the 2x2-Gauss formula. The area is computed

Input variables:

Name	Type	Rank	Description
chLayer, imacStart, imacStop, imgLevel, imatBlock	integer		
rdiscrid	Tdiscrid		matrix identifier
bscaletoZero	logical		
rparBlock	type(t_parBlock)		parallel block...
h_Dfmon	integer		handle of monitor function
rstat	t_stat		number of operations performed + time needed

7.29.16 Subroutine tools_getArea

Interface:

tools_getArea(h_Darea, rparBlock, chlayer, imacroIdx, imgLevel)

Description:
This subroutine computed the element area distribution of the current grid. The area of the elements is stored in the vector Darea, which has to be created outside this routine. This vector is elementwise organised (length: number of elements on parallel block), therefore it may be created by a call of storage_new, and not by a call of hl_reserve.

Input variables:

Name	Type	Rank	Description
rparBlock !parallelblock	t_parBlock		
h_Darea	integer		handle for area distribution vector
chlayer	integer		hierarchical layer
imacroIdx	integer		index of the macro on which the area distribution shall be computed (if MB_ALL_MB or (not HL_MB), all macros are taken)
imgLevel	integer		multigrid level on which the compuation shall be performed

7.30 Module precon

Purpose: This module contains several routines for preconditioing schemes like Jacobi, Gauss-Seidel, Tri-Gauss-Seidel. The routines are divided in two groups. the first group initialises some structures like factorization, the other group performs the actual preconditiong to a given vector. The comments have to be improved.

Constant definitions:

Name	Type	Purpose
Constants for imode		
Purpose: mode of prec_init		

Constants for ctypPrec

Purpose: types of preconditioners

PREC_TYPJAC	integer	Jacobi preconditioner
PREC_TYPGAUSS	integer	Gaussian elimination as preconditioner
PREC_TYPTRIDI	integer	TriDi preconditioner
PREC_TYPNO	integer	no preconditioning at all
PREC_TYPTRIGS	integer	TriGS (Gauss-Seidel) preconditioner
PREC_TYPTRIGS1	integer	aux- TriGS for ADITriGS
PREC_TYPADITRIGS	integer	ADITRiGS (alternating directions Guass-Seidel) preconditioner
PREC_TYPILU	integer	ILU preconditioner
PREC_TYPADITRIGSJAC	integer	ADITRiGS (alternating directions Guass-Seidel) preconditioner with JACOBI on regular grids
PREC_TYPTRIGSJAC	integer	TRiGS preconditioner with JACOBI on regular grids
PREC_TYPGPU	integer	GPU-based preconditioner
PREC_TYPGPUMG	integer	GPU-MG preconditioner (2level-scarc, inner level GPU-based MG)

Purpose: defect correction

PREC_CORRECT_DEF_VECTOR	integer	
PREC_CORRECT_ITER_VECTOR	integer	

7.30.1 Function prec_permindex

Interface:

`prec_permindex(inode, neqSqrt)`

Description:
This routine permutes macro node indices as needed for ADITRIGS. In case of having nine nodes/equations, the mapping is 7 8 9 3 6 9 4 5 6 –¿ 2 5 8 1 2 3 1 4 7 So, the routine mirrors the macro nodes at the diagonal going from down left to top right.

Input variables:

Name	Type	Rank	Description
inode	integer		node number (between 1 and neq (=number of nodes))
neqSqrt	integer		square root of neq (=number of nodes in one row/column)

7.30.2 Subroutine prec_init_jacobi

Interface:

prec_init_jacobi(rparBlock, chlayer, H_Daux, imglev, cflag, imacroidx, imatidx)

Description:
This function reservers and/or initializes the structures for Jacobi preconditioning.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
imglev	integer		multigrid level
cflag	integer		multigrid level flag HL_ONE, HL_ALL
imacroidx	integer		matrixblock id (important only if HL_MB)
imatidx	Tdiscrid		matrix id

Output variables:

Name	Type	Rank	Description
H_Daux	integer	PREC_MAXAUXV	init data

7.30.3 Subroutine prec_init_ilu

Interface:

prec_init_ilu(rparBlock, H_Daux, imglev, cflag, imacroidx, imatidx, ciluType, dilualpha, H_Iidx, H_Ioff)

Description:
This function reservers and/or initializes the structures for ILU preconditioning for sparse matrices.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
imglev	integer		multigrid level
cflag	integer		multigrid level flag HL_ONE, HL_ALL
imacroidx	integer		matrixblock index (important only if HL_MB)
imatidx	Tdiscrid		matrix id
ciluType	integer		???
dilualpha	real(DP)		???
H_Iidx	integer	:	???
H_Ioff	integer	:	???

Output variables:

Name	Type	Rank	Description
H_Daux	integer	PREC_MAXAUXV	init data

7.30.4 Subroutine prec_init_tridi

Interface:

prec_init_tridi(rparBlock, chlayer, H_Daux, imglev, cflag, imacroidx, imatidx, dswitch, cuse)

Description:
This function reservers and/or initializes the structures for TriDiag and TriGS preconditioning.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
chlayer	integer		hierachical level
imglev	integer		multigrid level
cflag	integer		multigrid level flag HL_ONE, HL_ALL
imacroidx	integer		matrixblock id (important only if HL_MB)
imatidx	Tdiscrid		matrix id
dswitch	real(DP)		if aspect ratio of the grid is less than dswitch then use Jacobi instead of TRIGS if dswitch = -1.0 then the Jacobi/TRIGS switching option is deactivated
cuse	integer		use for TriDiag or TriGS preconditioning

Output variables:

Name	Type	Rank	Description
H_Daux	integer	PREC_MAXAUXV	init data

7.30.5 Subroutine prec_init_trigs1

Interface:

prec_init_trigs1(rparBlock, chlayer, H_Daux, imglev, cflag, imacroidx, imatidx, dswitch)

Description:
This function reservers and/or initializes the structures for trigs1 preconditioning, which simply is TRIGS preconditioning applied to the matrix which results from mirroring the macro node numbers at the diagonal going from down left to top right. Example for macro with 25 nodes:
21 22 23 24 25 5 10 15 20 25 16 17 18 19 20 4 9 14 19 24 11 12 13 14 15 $\mapsto$ 3 8 13 18 23 6 7 8 9 10 2 7 12 17 22 1 2 3 4 5 1 6 11 16 21 The matrix is permuted accordingly by exchanging entries of the

corresponding bands. If we write the band indices into the macro structure with the node corresponding to main diagonal into the center
7 8 9 4 6 9 UL UD UU LU DU UU 5 1 6 ↦ 2 5 8 or DL DD DU ↦ LD DD DU 2 3 4 2 5 7 LL LD LU LL DL UL it can be seen that the following exchanges have to be done band index in orig matrix band index in permuted matrix 1 1 2 2 3 5 4 7 5 3 6 8 Bands 7 - 9 do not have to be treated as for the TRIGS preconditioner only the first six bands are necessary (LL, LD, LU, DL, DD, DU).

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`chlayer`	integer		hierachical level
`imglev`	`integer`		multigrid level
`cflag`	integer		multigrid level flag HL_ONE, HL_ALL
`imacroidx`	`integer`		matrixblock id (important only if HL_MB)
`imatidx`	`Tdiscrid`		matrix id
`dswitch`	`real(DP)`		if aspect ratio of the grid is less than dswitch then use Jacobi instead of TRIGS if dswitch = -1.0 then the Jacobi/TRIGS switching option is deactivated

Output variables:

Name	Type	Rank	Description
`H_Daux`	`integer`	PREC_MAXAUXV	init data

7.30.6 Subroutine prec_init_gauss

Interface:

`prec_init_gauss(rparBlock, H_Daux, imglev, imacroidx, imatidx)`

Description:
This function reserves and/or initializes the structures for direct elem preconditioning.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallelblock object
`imglev`	`integer`		multigrid level
`imacroidx`	`integer`		matrixblock id (important only if HL_MB)
`imatidx`	`Tdiscrid`		matrix id

Output variables:

Name	Type	Rank	Description
H_Daux	integer	PREC_MAXAUXV	init data

7.30.7 Subroutine prec_init_gpu

Interface:

prec_init_gpu(rparBlock, H_Daux, imglev, cflag, imacroidx, imatidx, rdataGPU)

Description:
This function reserves and/or initializes the structures for GPU preconditioning.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
imglev	integer		multigrid level
cflag	integer		multigrid level flag HL_ONE, HL_ALL
imacroidx	integer		matrixblock id (important only if HL_MB)
imatidx	Tdiscrid		matrix id
rdataGPU	t_dataGPU		GPU parameters

Output variables:

Name	Type	Rank	Description
H_Daux	integer	PREC_MAXAUXV	init data

7.30.8 Subroutine prec_init_gpumg

Interface:

prec_init_gpumg(rparBlock, H_Daux, imglev, cflag, imacroidx, imatidx, rdataGPU)

Description:
This function reserves and/or initializes the structures for GPU-MG preconditioning.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
imglev	integer		multigrid level
cflag	integer		multigrid level flag HL_ONE, HL_ALL
imacroidx	integer		matrixblock id (important only if HL_MB)
imatidx	Tdiscrid		matrix id
rdataGPU	t_dataGPU		GPU parameters

Output variables:

Name	Type	Rank	Description
H_Daux	integer	PREC_MAXAUXV	init data

7.30.9 Subroutine prec_perform_jacobi

Interface:

prec_perform_jacobi(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, rstat)

Description:
This subroutine performs Jacobi preconditioning to the given vector. The subroutine works in two modes:
1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		handle for iteration vector
imatidx	Tdiscrid		matrix id

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.10 Subroutine prec_perform_trigs

Interface:

prec_perform_trigs(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, dswitch, rstat)

Description:
This subroutine performs TriGS preconditioning to the given vector. The subroutine works in two modes:
1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		vector handle
imatidx	Tdiscrid		matrix id
dswitch	real(DP)		if aspect ratio of the grid is less than dswitch then use Jacobi instead of TRIGS if dswitch = -1.0 then the Jacobi/TRIGS switching option is deactivated

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.11 Subroutine prec_perform_ilu

Interface:

prec_perform_ilu(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, H_Iidx, H_Ioff, imatidx, rstat)

Description:

This subroutine performs TriGS preconditioning to the given vector. The subroutine works in two modes:
1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		vector handle
imatidx	Tdiscrid		matrix id
H_Iidx	integer	:	???
H_Ioff	integer	:	???

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.12 Subroutine prec_perform_aditrigsjac

Interface:

prec_perform_aditrigsjac(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, iter, dswitch, rstat)

Description:
This subroutine performs ADITriGS preconditioning to the given vector. The subroutine works in two modes: 1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		vector handle
imatidx	Tdiscrid		matrix id
iter	integer		
dswitch	real(DP)		if aspect ratio of the grid is less than dswitch then use Jacobi instead of TRIGS if dswitch = -1.0 then the Jacobi/TRIGS switching option is deactivated

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.13 Subroutine prec_perform_aditrigs

Interface:

prec_perform_aditrigs(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, iter, rstat)

Description:
This subroutine performs ADITriGS preconditioning to the given vector. The subroutine works in two modes: 1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		vector handle
imatidx	Tdiscrid		matrix id
iter	integer		

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.14 Subroutine prec_perform_trigs1

Interface:

prec_perform_trigs1(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, dswitch, rstat)

Description:
This subroutine performs a trigs preconditioning to the given vector. The subroutine works in two modes:
1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		vector handle
imatidx	Tdiscrid		matrix id
dswitch	real(DP)		if aspect ratio of the grid is less than dswitch then use Jacobi instead of TRIGS if dswitch = -1.0 then the Jacobi/TRIGS switching option is deactivated

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.15 Subroutine prec_perform_tridi

Interface:

prec_perform_tridi(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, dswitch, rstat)

Description:
This subroutine performs tridi preconditioning to the given vector. The subroutine works in two modes: 1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		vector handle
imatidx	Tdiscrid		matrix id
dswitch	real(DP)		if aspect ratio of the grid is less than dswitch then use Jacobi instead of TRIGS if dswitch = -1.0 then the Jacobi/TRIGS switching option is deactivated

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.16 Subroutine prec_perform_gpu

Interface:

prec_perform_gpu(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, rstat)

Description:
COMPLETELY OUTDATED COMMENT!!! This subroutine performs Jacobi preconditioning to the given vector. The subroutine works in two modes: 1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		handle for iteration vector
imatidx	Tdiscrid		matrix id

Output variables:

Name	Type	Rank	Description
rstat	t_stat		operation count

7.30.17 Subroutine prec_perform_gpumg

Interface:

prec_perform_gpumg(rparBlock, H_Daux, chLayer, domega, cdefcor, imglev, imacStart, imacStop, h_Dx, imatidx, rstat)

Description:
COMPLETELY OUTDATED COMMENT!!! This subroutine performs Jacobi preconditioning to the given vector. The subroutine works in two modes: 1) cdefcor = PREC_CORRECT_ITER_VECTOR: it computes $x = D^{-1}x$ 2) cdefcor = PREC_CORRECT_DEF_VECTOR: it computes $x = x + \omega D^{-1}(def)$

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
H_Daux	integer	PREC_MAXAUXV	init data
chLayer	integer		hierachical layer
domega	real(DP)		omega parameter
cdefcor	integer		defect correction mode (PREC_CORRECT_ITER_VECTOR or PREC_CORRECT_DEF_VECTOR)
imglev	integer		multigrid level
imacStart	integer		start matrixblock
imacStop	integer		end matrixblock
h_Dx	integer		handle for iteration vector
imatidx	Tdiscrid		matrix id

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		operation count

7.31 Module solver

Purpose: This module contains the FEAST solver engine ScaRC, multigrid solvers, CG and BiCGStab solvers as well as routines for setting of boundary conditions and values. Moreover, there are routines for matrixblock object scaling, multiplication and adding.

Constant definitions:

Name	Type	Purpose
Purpose: constants for solver types		
SOLV_TYPSCARC	integer	ScaRC solver
SOLV_TYPCG	integer	CG method
SOLV_TYPMG	integer	multigrid method
SOLV_TYPRICH	integer	Richardson scheme
SOLV_TYPGLOBAL	integer	SD coarse grid scheme
SOLV_TYPPGLOBAL	integer	PB coarse grid scheme
SOLV_TYPBICG	integer	BiCGStab method (with right preconditioning)
SOLV_TYPMGLOBAL	integer	MB coarse grid scheme
SOLV_TYPRICHARDSON	integer	Richardson method (working as stand alone solver)
SOLV_TYPBICGLEFT	integer	BiCGStab method (with left preconditioning)
Purpose: misc. solving parameters		
SOLV_SMOOTHDIRECT	integer	direct smoothing via precon
SOLV_SMOOTHINDIRECT	integer	indirect smoothing via ScaRC
MG_MAXLEVEL	integer	maximum multigrid level
MG_VCYCLE	integer	Do not change these three constants!!! V multigrid cycle
MG_FCYCLE	integer	F multigrid cycle
MG_WCYCLE	integer	W multigrid cycle
SOLV_PURE_NEUMANN	logical	Do not change these three constants!!! for problems with Neumann boundary conditions only (in this case, special treatment required)
SOLV_NOT_PURE_NEUMANN	logical	
SOLV_ZERO_START_VEC	logical	clear the solution vector before starting the iteration
SOLV_THIS_START_VEC	logical	use the given solution vector as starting vector
Purpose: solving mode		
SOLV_MODESOLVER	integer	
SOLV_MODESMOOTHER	integer	
SOLV_MODECOARSE	integer	

Type definitions:

Type name	component name	Type	Rank	Purpose

t_umfpackInfo	Control record fuer UMFPACK			
	Control	real(DP)	20	*** direct UMFPACK variables, documentation see UMFPACK manual no FEAST-styleguide applied here to maintain the consistency to UMFPACK control records
	Info	real(DP)	90	
	symbolic	integer		handle for symbolic factorisation
	numeric	integer		handle for numeric factorisation
	h_Dsol	integer		*** END direct UMFPACK variables style guide ON handle for solution vector
	h_Drhs	integer		handle for right hand side
	neq	integer		number of eqations
	nnonZero	integer		number of non-zero elements
	h_da	integer		handle for direct matrix
	h_ipivot	integer		handle for pivot element vector

t_solver	solver object		
	rmasterData	t_masterDataSolver	
	ctype	integer	identifier SOLV_TYPxx
	nmaxIter	integer	maximum number of iterations
	chLayer	integer	hierachical layer
	dtol	real(DP)	stopping criterion (tolerance)
	brel	logical	flag if relative or absolute tolerance
	iactualIter	integer	actual number of iterations needed for achieving tolerance

dres	real(DP)		residuum (norm of defect)
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
dconvRateMin	real(DP)		minimum convergence rate ScaRC
dconvRateMax	real(DP)		maximum convergence rate ScaRC
dconvRateAvg	real(DP)		average convergence rate ScaRC
rstat	t_stat		solver time and operations counter
H_Daux	integer	PREC_MAXAUXV	handle array for auxiliary vectors
bresBelowTol	logical		flag if residuum is below chosen tolerance
cprecond	integer		type of preconditioner SOLV_SMOOTHxxx
bisGlobal	logical		flag if solver is global
rschlumpf	t_umfpackInfo		??? UMFPACK solver record
rschlumpf1	t_umfpackInfo		??? UMFPACK solver record
rdataMG	t_dataMG		additional data for multigrid
rdataCG	t_dataCG		additional data for cg
rdataRich	t_dataRich		additional data for richardson
rdataGPU	t_dataGPU		additional data for GPU-based solvers/smoothers
rdiscrId	type(Tdiscrid)		discretisation this solver works for
retrhs	integer		handle for saving the RHS connected to the solver
H_DdefCorr	integer		handle vector for defect correction vectors ScaRC (size: nmacros in rparBlock)

	H_Iidx	integer	GR_MAXADAPRANGE+HL_MAXMGLEVEL	master index for converting to sparse matric
	H_Ioff	integer	GR_MAXADAPRANGE+HL_MAXMGLEVEL	master offset for converting to sparse matric
	iglobalUmfpackIndex	integer		index of UMFPACK structure t_masterDataSo for the case solver is a g coarse grid solve
	rnext	t_solver		pointer to solve ject responsible next matrix blo

t_dataMG structure for additional multigrid data

	nlevelMin	integer		start multigrid l
	nlevelMax	integer		end multigrid le
	ctypeCycle	integer		multigrid schem be used (V, W,
	CtypeCycleLocal	integer	MG_MAXLEVEL	For realising th cycle and W- a control is ne which decides cally" whether or a W-cycle is cuted.
	h_Db	integer		??? handle for tion vector
	nsmoothPre	integer		number of moothing steps
	nsmoothPost	integer		number postsmoothing steps
	rsmoother	t_solver		solver object smoothing
	rsolverCoarse	t_solver		solver object coarse grid solve
	rstatSmooth	t_stat		time and oper counter for sm ing
	rstatTransfer	t_stat		time and oper counter for tra operations

	rstatCoarse	t_stat		time and operation counter for coarse grid solving
	rstatDaxpy	t_stat		time and operation counter for vector operations

t_dataCG structure for additional CG data

	rprecond	t_solver		solver object for preconditioning
	nnumVec	integer		number of aux. vectors
	H_Dvec	integer	:	array of vector handles (length nnumVec)
	cmodus	integer		flag for use of cg solver (SOLV_MODESOLVER, SOLV_MODESMOOTHER or SOLV_MODECOARSE)

t_solverAccess access type for solver

	ra	t_solver		access variable

t_dataRich structure for additional Richardson data

	ctypePrec	integer		preconditioner type
	ddamping	real(DP)		damping parameter (omega)
	nprecondSteps	integer		number of smoothing steps
	Rprecond	t_solverAccess	dim(:)	solver objects for preconditioning (size = nmacros in rparBlock)
	rprecondRoot	t_solver		dynamic solver object for preconditiong
	rprecondBase	t_solver	dim(:)	static solver object for preconditiong
	ciluType	integer		type of ILU preconditioner
	diluAlpha	real(DP)		
	dswitch	real(DP)		switch parameter

t_masterDataSolver	Structure storing the UMFPACK info objects for all global coarse grid solvers. This structure lives on the m
	nnumCoarseSolvers
	RumfpackInfo

7.31.1 Subroutine solver_initCoarseSD

Interface:
solver_initCoarseSD(rparBlock, rsolver, imatidx)

Description:
This routine initializes the data structures for the global coarse grid solver.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallelblock structure
imatidx	Tdiscrid		

Output variables:

Name	Type	Rank	Description
rsolver	type(t_solver)		solver structure

7.31.2 Subroutine solver_initCoarseMB

Interface:
solver_initCoarseMB(rparBlock, rsolver, imacroIdx, rdiscrId)

Description:
This routine initializes the data structures for the matrixblock coarse grid solver. The routine is inefficient and needs rework (convert from BAND to UMFPACK2 to CSR...).

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		parallel block structure
imacroIdx	integer		index of the macro the solving shall take place on (-1: all macros)
rdiscrId	Tdiscrid		

Output variables:

Name	Type	Rank	Description
rsolver	type(t_solver)		

7.31.3 Subroutine solver_reinit

Interface:
solver_reinit(rparBlock, rsolver, rdiscrId, cflag, imglevel, imacroIdx)

Description:
This subroutine reinits the solver data structures like factorizations etc. for the given matrixblock.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock structure
rdiscrId ! matrix object	type(Tdiscrid)		
cflag	integer		HL_ALL or HL_ONE
imacroIdx	integer		index of the macro the solving shall take place on (-1: all macros)
imgLevel	integer		

Output variables:

Name	Type	Rank	Description
rsolver	t_solver		reinitialized solver object

7.31.4 Subroutine solver_initDirichValues

Interface:
solver_initDirichValues(rparBlock, chLayer, imgLevel, rdiscrId, h_Dvec, pboundaryValue)

Description:
This routine sets the entries of the vector h_Dvec corresponding to Dirichlet boundary nodes either to the values defined by the function pboundaryValue (if the corresponding boundary object is defined as having variable function values) or to the constant values saved with the corresponding boundary object. No averaging will be done! The vector h_Dvec will contain full entries!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level (-1 means all levels)
rdiscrId	Tdiscrid		matrix
h_Dvec	integer		handle array for vector to initialised

7.31.5 Subroutine solver_setDirichValues

Interface:

solver_setDirichValues(rparBlock, chLayer, imgLevel, rdiscrId, h_Dvec, dvalue)

Description:
This routine sets the entries of the vector h_Dvec corresponding to Dirichlet boundary nodes to the passed value dvalue. No averaging will be done! The vector h_Dvec will contain full entries!

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level (-1 means all levels)
rdiscrId	Tdiscrid		matrix
h_Dvec	integer		handle array for vector to initialised
dvalue	real(DP)		

7.31.6 Subroutine solver_copyDirichValuesScaled

Interface:

solver_copyDirichValuesScaled(rparBlock, chLayer, imgLevel, rdiscrId, h_Dsource, dscale, h_Ddest, baverage)

Description:
This routine copies the entries of the vector h_Dsource corresponding to Dirichlet boundary nodes to the entries of h_Ddest (scaled by dscale). If baverage is .TRUE. the values in h_Ddest are averaged afterwards.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level (-1 means all levels)
rdiscrId	Tdiscrid		matrix
h_Dsource	integer		handle array for the source vector
dscale	real(DP)		value by which the Dirichlet entries have to be scaled
h_Ddest	integer		handle array for the destionation vector
baverage	logical		optional flag if the values should be averaged

7.31.7 Subroutine solver_parseEps

Interface:

`solver_parseEps(shl, dtol, brel)`

Description:
This subroutine parses a given string exit condition and returns the type of the stopping criterion as well as the numerical value of the tolerance.

Input variables:

Name	Type	Rank	Description
shl	c (*)		type of stopping condition: REL or ABS

Output variables:

Name	Type	Rank	Description
brel	logical		type of condition, true: size relative residuum as stopping criterion
dtol	real(DP)		tolerance value

7.31.8 Function solver_ges

Interface:

`solver_ges(brel)`

Description:
This functions returns the string representation of the type of a exit condition.

Input variables:

Name	Type	Rank	Description
`brel`	`logical`		type of exit condition, true: relative residuum

Result:
character(len=3) : string exit condition representation

7.31.9 Subroutine solver_perform

Interface:
`solver_perform(rparBlock, rsolver, imgLevel, h_Dx, imacroIdx, bpureNeumann, bclearIterVec, rstat)`

Description:
This routines solves the problem given in rsolver and returns the solution in h_Dx.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block structure
`rsolver`	`t_solver`		solver structure
`imgLevel`	`integer`		multigrid level of the solution
`imacroIdx`	`integer`		Index of the macro where solving shall take place (only for HL_MB)
`bpureNeumann`	`logical`		if true, the problem has no Dirichlet boundary
`bclearIterVec`	`logical`		if true, the iteration vector is initially cleared

Output variables:

Name	Type	Rank	Description
`h_Dx`	`integer`		handle of the solution vector
`rstat`	`t_stat`		stat record

7.31.10 Subroutine solver_info

Interface:
`solver_info(rsolver, soffset, bscarc)`

Description:
This routine prints some information about the given solver.

Input variables:

Name	Type	Rank	Description
rsolver	t_solver		solver structure
soffset	c (*)		string offset, first call: null string
bscarc	logical		if true: full scarc information is printed

7.31.11 Subroutine solver_precondInfo

Interface:
solver_precondInfo(rsmoother, soffset)

Description:
The routine prints information about the given smoother.

Input variables:

Name	Type	Rank	Description
rsmoother	t_solver		smoother object
soffset	c (*)		string offset, first call: null string

7.31.12 Function solver_btolReached

Interface:
solver_btolReached(brel, dtol, dres, dresInitial)

Description:
This function checks if a given exit condition is fulfilled.

Input variables:

Name	Type	Rank	Description
brel	logical		if true: relative residuum used, otherwise: aboslute residuum
dtol	real(DP)		given accuracy
dres	real(DP)		norm of the defect
dresInitial	real(DP)		norm of the initial defect

Result:
logical : true, if exit condition is fulfilled

7.31.13 Subroutine solver_initGlobalCoarseSlave

Interface:
solver_initGlobalCoarseSlave(rparBlock, rsolver)

Description:

Subroutine, which sends the parallel block contribution of the global coarse matrix to the master. The receiving routine is solver_initGlobalCoarseMaster.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallelblock object
rsolver	t_solver		global coarse grid solver

7.31.14 Subroutine solver_cg

Interface:
solver_cg(rparBlock, rres, imgLevel, h_Dx, imacroIdx, bpureNeumann, bclearIterVec, rstat)

Description:
This routine implements the preconditioned conjugate gradient method.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
rres	t_solver		
imgLevel	integer		level of grid refimenent
h_Dx	integer		handle for iteration vector
imacroIdx	integer		index of the macro on which has to be solved
bpureNeumann	logical		true, if there is no Dirichlet boundary part
bclearIterVec	logical		true, if the iteration vector shall be set to 0 before iteration starts

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

7.31.15 Subroutine solver_bicgleft

Interface:
solver_bicgleft(rparBlock, rres, imgLevel, h_Dx, imacroIdx, bpureNeumann, bclearIterVec, rstat)

Description:
This routine performs the left preconditioned variant of the BiCGStab-algorithm, i.e. we solve $C^{-1}Ax = C^{-1}b$. This method has the disadvantage, that we are not able to control the residual itself, but only

the preconditioned one. Therefore, the routine is suitable as inner routine to gain e.g. two digits accuracy, but is not suitable as outer method, when the absolute size of the residuum has to be controlled. The alternative (not implemented in FEAST) is right preconditioning: solve $Ay = b, y = C^{-1}x$. The differences in the algorithm are: 1) The initial residuum is not preconditioned any more. 2) The order of MV-multiplication and preconditioning is changed. In the left-preconditioned variant, we apply MV first and apply preconditioning thereafter, in the right-preconditioned variant, it is vice versa. Remark: In the implementation, we reuse the right hand side vector as aux vector. By this, we destroy the right hand side, but save one vector.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		
`rres`	`t_solver`		
`imgLevel`	`integer`		level of grid refimenent
`h_Dx`	`integer`		handle of iteration vector
`imacroIdx`	`integer`		index of the macro on which has to be solved
`bpureNeumann`	`logical`		true, if there is no Dirichlet bondary part
`bclearIterVec`	`logical`		true, if the iteration vector shall be set to 0 before iteration starts

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		number of operations performed + time needed

7.31.16 Subroutine solver_bicg

Interface:

`solver_bicg(rparBlock, rres, imgLevel, h_Dx, imacroIdx, bpureNeumann, bclearIterVec, rstat)`

Description:
This routine performs the right preconditioned variant of the BiCGStab-algorithm.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
rres	t_solver		
imgLevel	integer		level of grid refimenent
h_Dx	integer		handle of iteration vector
imacroIdx	integer		index of the macro on which has to be solved
bpureNeumann	logical		true, if there is no Dirichlet bondary part
bclearIterVec	logical		true, if the iteration vector shall be set to 0 before iteration starts

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

7.31.17 Subroutine solver_rich

Interface:

solver_rich(rparBlock, rsmoother, imgLevel, h_Dx, imacroIdx, cdefcor, bpureNeumann, bclearIterVec, rstat)

Description:
This routine implements the preconditioned Richardson method.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
rsmoother	t_solver		
imgLevel	integer		level of grid refimenent
h_Dx	integer		handle for iteration vector
imacroIdx	integer		index of the macro on which has to be solved

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

7.31.18 Subroutine solver_mg

Interface:

`solver_mg(rparBlock, rsolver, h_Dx, imacroIdx, bpureNeumann, bclearIterVec, rstat)`

Description:
This routine implements the preconditioned multigrid method.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		
`rsolver`	`t_solver`		
`h_Dx`	`integer`		handle of solution vector
`imacroIdx`	`integer`		index of the macro on which has to be solved
`bpureNeumann`	`logical`		true, if there is no Dirichlet part
`bclearIterVec`	`logical`		true, if the iteration vector shall be set to 0 before iteration starts

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		number of operations performed + time needed

7.31.19 Subroutine solver_richardson

Interface:
`solver_richardson(rparBlock, rres, imgLevel, h_Dx, imacroIdx, bpureNeumann, bclearIterVec, rstat)`

Description:
This routine implements a preconditioned Richardson method

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		
`rres`	`t_solver`		
`imgLevel`	`integer`		level of grid refimenent
`h_Dx`	`integer`		handle for iteration vector
`imacroIdx`	`integer`		index of the macro on which has to be solved
`bpureNeumann`	`logical`		true, if there is no Dirichlet boundary part
`bclearIterVec`	`logical`		true, if the iteration vector shall be set to 0 before iteration starts

Output variables:

Name	Type	Rank	Description
rstat	t_stat		number of operations performed + time needed

7.31.20 Subroutine solver_readScarcClient

Interface:

solver_readScarcClient(rparBlock, rdiscr, Rsolver, nscarc, rmdSolver)

Description:
This routine inits the main data structures like coarse grid solver, communication structures, boundary conditions, grid and macro structure.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
rdiscr	Tdiscretization		
nscarc	integer		

Input/Output variables:

Name	Type	Rank	Description
rmdSolver	t_masterDataSolver		

7.31.21 Subroutine solver_prepareGlobalCoarseSolver

Interface:

solver_prepareGlobalCoarseSolver(rmdParBlock, rmdSolver)

Description:
!This routine (RUNNING ON THE MASTER PROCESS) prepares the direct global coarse !grid solvers by allocating the necessary structures. It is called directly after !all *.scarc files have been read in and the number of coarse grid solvers needed !is known. !

Input variables:

Name	Type	Rank	Description

7.31.22 Subroutine solver_BlindNodeMatMod_VerySlow

Interface:

solver_BlindNodeMatMod_VerySlow(neq, nnonZero, rparBlock, DAorig, KCOLorig, KLDorig)

Description:

This subroutine performs the matrix modification necessary for treating hanging nodes with a global solver. The matrix A is modified to $M^T AM$ where M is a projection matrix described in "G Kanschat, F.T. Suttmeier:Finite Element Techniques on locally refined grids". This matrix M ensures the continuity of the solution. The matrix M does not have full rank, and therefore $M^T AM$ s not suitable for direct solvers. In $M^T AM$ there are 0-rows and columns for the hanging nodes, i.e. they are not considered in the computation at all. Therefore, we modify these columns and rows to be unit-columns and -rows. As this filtering technique does not work for Dirichlet DOFs, we have to restore these rows in the matrix after the matrix multiplication is performed. However, after the computation of the solution, one has to correct the values of the hanging nodes manually! This subroutine is very inefficient and memory-wasting, but obtains correct results unlike the previous version.

Input variables:

Name	Type	Rank	Description
`neq`	`integer`		number of equations
`nnonZero`	`integer`		number of non-zero matrix entries
`rparBlock`	`t_parBlock`		
`DAorig`	`real(DP)`	:	vectors describing the matrix (CSR-format)
`KCOLorig`	`integer`	:	
`KLDorig`	`integer`	:	

7.31.23 Subroutine solver_BlindNodeMatMod

Interface:

`solver_BlindNodeMatMod(neq, nnonZero, rparBlock, DAorig, KCOLorig, KLDorig)`

Description:
This subroutine performs the matrix modification necessary for treating hanging nodes with a global solver. The matrix A is modified to $M^T AM$ where M is a projection matrix described in "G Kanschat, F.T. Suttmeier:Finite Element Techniques on locally refined grids". This matrix M ensures the continuity of the solution. The matrix M does not have full rank, and therefore $M^T AM$ s not suitable for direct solvers. In $M^T AM$ there are 0-rows and columns for the hanging nodes, i.e. they are not considered in the computation at all. Therefore, we modify these columns and rows to be unit-columns and -rows. As this filtering technique does not work for Dirichlet DOFs, we have to restore these rows in the matrix after the matrix multiplication is performed. However, after the computation of the solution, one has to correct the values of the hanging nodes manually!

Input variables:

Name	Type	Rank	Description
`neq`	`integer`		number of equations
`nnonZero`	`integer`		number of non-zero matrix entries
`rparBlock`	`t_parBlock`		
`DAorig`	`real(DP)`	:	vectors describing the matrix (CSR-format)
`KCOLorig`	`integer`	:	
`KLDorig`	`integer`	:	

7.31.24 Subroutine solver_solveGlobalCoarseMaster

Interface:

`solver_solveGlobalCoarseMaster(RumfpackInfo, rparBlock)`

Description:
This routine (running on the master process) receives the right hand side vector contributions from all slaves, puts them together and performs the UMFPACK solving process. The solution is then sent back to the slaves. The counterpart of this routine is solver_solveGlobalCoarseSlave.

Input variables:

Name	Type	Rank	Description
`RumfpackInfo`	`t_umfpackInfo`		!!TEMPORARILY DEACTIVATED!!! (reason: see solver.f90, routine solver_prepareGlobalCoarseSolver) type (t_umfpackInfo),dimension(:),pointer :: RumfpackInfo !!TEMPORARILY DEACTIVATED!!!
`rparBlock`	`t_parBlock`		

7.31.25 Subroutine solver_solveGlobalCoarseSlave

Interface:

`solver_solveGlobalCoarseSlave(rsolver, rparBlock, h_Dsol, imgLevel, rstat)`

Description:
This routine sends the contributions of the current parallel block to a global problem to the master process. When the master has solved this global problem, the routine receives the part of the global solution which belongs to the current parallel block. The counterpart of this routine is solver_solveGlobalCoarseMaster.

Input variables:

Name	Type	Rank	Description
`rsolver`	`t_solver`		solver structure
`rparBlock`	`t_parBlock`		parallelblock structure
`h_Dsol`	`integer`		handle of solution vector
`imgLevel`	`integer`		level of refinement

7.31.26 Subroutine solver_initGlobalCoarseMaster

Interface:

`solver_initGlobalCoarseMaster(mdparBlock, rmdSolver)`

Description:
In this subroutine (RUNNING ON THE MASTER), the assembly of a global matrix from the contributions given by the parallel blocks is performed. For the collection of the matrix from the slaves, the UMFPACK2-format is employed, as it is very difficult to build up a CSR-matrix on the fly. After the collection process, the matrix is converted to CSR.

Input variables:

Name	Type	Rank	Description
mdParBlock	t_masterDataParallelBlock		master parallel block
rmdSolver	t_masterDataSolver		master solver structure

7.32 Module multidimsolver

Purpose: This module contains numerous solver routines for vector-valued problems.

Constant definitions:

Name	Type	Purpose
Purpose: solver flags		
MDS_DIRECT	integer	
MDS_SCARC	integer	
MDS_RICH	integer	
MDS_CG	integer	
MDS_BICGSTAB	integer	
MDS_MG	integer	
MDS_SCHURCOMPL_RICH	integer	
MDS_SCHURCOMPL_CG	integer	
MDS_SCHURCOMPL_BICGSTAB	integer	
MDS_SCHURCOMPL_MG	integer	
MDS_TEST_SOLVER	integer	
Purpose: preconditioner flags		
MDS_PREC_NONE	integer	
MDS_PREC_JACOBI	integer	
MDS_PREC_SOR	integer	
MDS_PREC_SSOR	integer	
MDS_PREC_SCBLOCK	integer	
MDS_PREC_GENERIC	integer	
MDS_PREC_EXT_GAUSS_SEIDEL	integer	
Purpose: communication		
MDS_DO_EXCHANGE	logical	
MDS_DO_NO_EXCHANGE	logical	

MDS_SCARCINFOLENGTH	integer	

Purpose: divergence detection

MDS_DIV_INDICATOR	integer	maximum number of monotonously increasing residua
MDS_DIV_FOR_SURE	real(DP)	maximum value for acceptable relative residuum

Purpose: storage organisation

MDS_NO_FREE_HANDLE	integer	constant indicating that there is no free vector handle anymore
MDS_NOTHING_ALLOCATED	integer	constant indicating that no vectors are allocated

Type definitions:

Type name	component name	Type	Rank	Purpose
t_multidimSolver	structure for multidimensional solver			
	ctype	integer		identifier MDS_xx
	nmaxIter	integer		maximum number of iterations
	drelax	real(DP)		relaxation parameter omega for SOR and SSOR
	ddamp	real(DP)		damping parameter for defect correction
	dtol	real(DP)		stopping criterion (tolerance) Special handling for CG- and BiCG Schur complement solvers: If the tolerance of the u-solver is set to a negative value this indicates the usage of a relaxation strategy which should be applied when the SCCG / SCBiCG solver is used as stand alone sovler.

	brel	logical
	bzeroStartVector	logical
	cprecond	integer
	nmaxIterExtGS	integer
	rprecond	t_multidimSolver
	rdataSC	t_schurComplement
	rdataMG	t_multigridData
	bstoreDefInRhs	logical
	rinfo	t_multidimSolverI

t_schurComplementData structure containing data for a multidimensional solver of type Schur complement

	rmdSolver_u	t_multidimSolver

	rmdSolver_p
	rscPrecMatDiffusive
	rscPrecMatReactive
	dcoeffDiffusive
	dcoeffReactive
t_schurComplementData	RHS vector which is passed to the smoother. By setting the flag to .TRUE., the smoother knows that it
t_multigridData	structure containing data for a multidimensional solver of multigrid type cmgCycle
	npreSmooth
	npostSmooth
	iminLevel
	rmdSolver_coarse
t_directSolverData	structure containing data for a direct solver

	imgLevel
	bfactorised
	rumfpackInfo
t_multidimSolverInfo	structure for storing results of the solving process (to be able to do some statistics or whatever after the s
	niter
	dresAbs
	dresRel
	dconvRate
	nop
	niterScarc
	niterSchur

7.32.1 Subroutine mdsolv_solve

Interface:

`mdsolv_solve( rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, rstat, rmdSolverInfo)`

Description:
Starts the solving process for a blockstructured system. The block structured matrix RblockMat has to be averaged while the right hand side H_Df must have averaged entries. The solution vector H_Dsol has to be initialised with the Dirichlet boundary values and will contain full entries on return. This routine has the purpose to simplify the starting of the solving process for the user. He does not have to think about t_multdimSolver structures or recursion depth.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	blockstructured solution vector
H_Df !averaged	integer	:	blockstructured rhs vector
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for output

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics
rmdsolverInfo	type(t_multidimSolverInfo)		info about the solving process

7.32.2 Subroutine mdsolv_perform

Interface:

mdsolv_perform(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, rstat, rmdSolverInfoOpt)

Description:
Performs the solving process for a blockstructured system.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	blockstructured solution vector
H_Df !averaged	integer	:	blockstructured rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for output
irecursionDepth	integer		recursion depth of recursive calls

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics
rmdSolverInfoOpt	type(t_multidimSolverInfo)		info about the solving process

7.32.3 Subroutine mdsolv_direct

Interface:

mdsolv_direct(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured system with a direct solver.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer	MDS_SCARCINFOLENGTH	number of ScaRC calls in preconditiong length of this vector has to be 2 * nBlocks - 1 (5 is enough, acutally!)
DscarcConvRates	real(DP)	MDS_SCARCINFOLENGTH	accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)	MDS_SCARCINFOLENGTH	accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.4 Subroutine mdsolv_directMaster

Interface:

`mdsolv_directMaster(rmasterParBlock)`

Description:
This routine (running on the master process) receives the right hand side vector contributions from all slaves, puts them together and performs the UMFPACK solving process. The solution is then sent back to the slaves. The counterpart of this routine is solver_solveGlobalCoarseSlave.

Input variables:

Name	Type	Rank	Description
rmasterParBlock	t_parBlock		

7.32.5 Subroutine mdsolv_scarc

Interface:

mdsolv_scarc(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured system with ScaRC

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer	MDS_SCARCINFOLENGTH	number of ScaRC calls in preconditiong length of this vector has to be 2 * nBlocks - 1 (5 is enough, acutally!)
DscarcConvRates	real(DP)	MDS_SCARCINFOLENGTH	accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)	MDS_SCARCINFOLENGTH	accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.6 Subroutine mdsolv_rich

Interface:

mdsolv_rich(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured system with the Richardson method.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong length of this vector has to be 2 * nBlocks - 1 (5 is enough, acutally!)
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.7 Subroutine mdsolv_cg

Interface:

mdsolv_cg(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured system with the CG method. (see Templates for the solution of linear systems: Building blocks for iterative methods)

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.8 Subroutine mdsolv_bicgstab

Interface:

mdsolv_bicgstab(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured system with the BiCGstab method.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.9 Subroutine mdsolv_mg

Interface:

mdsolv_mg(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured system with multigrid.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong length of this vector has to be 2 * nBlocks - 1 (5 is enough, acutally!)
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.10 Subroutine mdsolv_schurcompl_rich

Interface:

mdsolv_schurcompl_rich(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Drhs, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:

++++++++++++++++ Under Construction ++++++++++++++++++ Richardson Schur complement method to solve a blockstructured saddlepoint system +++++++++++++++ Under Construction ++++++++++++++++++

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Drhs !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.11 Subroutine mdsolv_schurcompl_cg

Interface:

mdsolv_schurcompl_cg(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Drhs, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
++++++++++++++++ Under Construction ++++++++++++++++++ CG Schur complement method to solve a blockstructured saddlepoint system ++++++++++++++++ Under Construction ++++++++++++++++++

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Drhs !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.12 Subroutine mdsolv_schurcompl_bicgstab

Interface:

mdsolv_schurcompl_bicgstab(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Drhs, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:

++++++++++++++++ Under Construction ++++++++++++++++++ BiCGstab Schur complement method to solve a blockstructured saddlepoint system ++++++++++++++++ Under Construction ++++++++++++++++++

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Drhs !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.13 Subroutine mdsolv_schurcompl_mg

Interface:

mdsolv_schurcompl_mg(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:
Solves a blockstructured saddle point system with multigrid.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong length of this vector has to be 2 * nBlocks - 1 (5 is enough, acutally!)
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.14 Subroutine mdsolv_testSolver

Interface:

mdsolv_testSolver(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Drhs, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, dres, dresInitial, dconvRate, NscarcCalls, DscarcConvRates, NscarcIters, niterActual, rstat)

Description:

++++++++++++++++ Under Construction ++++++++++++++++++ Generic solver for testing purposes ++++++++++++++++ Under Construction ++++++++++++++++++

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Drhs !averaged entries	integer	:	handle array for rhs vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Output variables:

Name	Type	Rank	Description
dres	real(DP)		residuum
dresInitial	real(DP)		initial residuum
dconvRate	real(DP)		convergence rate
NscarcCalls	integer		number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)		accumulated convergence rates of the ScaRC solvers in preconditiong
NscarcIters	real(DP)		accumulated number of ScaRC iterations in preconditioning
niterActual	integer		actual number of iterations
rstat	t_stat		structure for storing runtime statistics

7.32.15 Function mdsolv_calcNorm

Interface:

```
mdsolv_calcNorm(rparBlock, chLayer, imgLevel, RblockMat, H_Dvec, bexchange, rstat)
```

Description:

Calculates norm of a blockstructured vector.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dvec !full entries	integer	:	handle array for vector
bexchange	logical		flag if vector shall be exchanged before calculation

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

Result:
real(DP) : !norm of the vector real(DP) :: dnorm

7.32.16 Subroutine mdsolv_calcDefect

Interface:
mdsolv_calcDefect(rparBlock, chLayer, imgLevel, RblockMat, H_Dsol, H_Df, H_Ddef, rstat)

Description:
Calculates the defect Ddef = Df - RblockMat * Dsol

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dsol !full entries	integer	:	handle array for solution vector
H_Df	integer	:	handle array for rhs vector
H_Ddef	integer	:	handle array for defect vector to be calculated

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.17 Subroutine mdsolv_precondition

Interface:

mdsolv_precondition(rparBlock, chLayer, imgLevel, RblockMat, H_Dc, H_Ddef, rmdSolver, bpureNeumann, cverbosity, ifileUnit, sls, irecursionDepth, NscarcCalls, DscarcConvRates, NscarcIters, rstat)

Description:
Calculate the defect correction by applying a preconditioner

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dc !full entries	integer	:	handle array for correction vector to be calculated
H_Ddef !averaged entries	integer	:	handle array for defect vector
rmdSolver	t_multidimSolver		
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform

Input/Output variables:

Name	Type	Rank	Description
NscarcCalls	integer	MDS_SCARCINFOLENGTH	number of ScaRC calls in preconditiong
DscarcConvRates	real(DP)	MDS_SCARCINFOLENGTH	accumulated convergence rates of the ScaRC solvers in preconditioning
NscarcIters	real(DP)	MDS_SCARCINFOLENGTH	accumulated number of ScaRC iterations in preconditioning

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.18 Subroutine mdsolv_scPrecond

Interface:

mdsolv_scPrecond(rparBlock, chLayer, imgLevel, H_Dc, H_Dr, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, rstat, RblockMat)

Description:
++++++++++++++++++ Under Construction ++++++++++++++++++++ Preconditioner for the different Schur complement solvers. ++++++++++++++++++ Under Construction ++++++++++++++++++++

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
H_Dc !full entries	integer	1	handle for correction vector
H_Dr !averaged entries	integer	1	handle for defect vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.19 Subroutine mdsolv_scPrecondElman

Interface:

mdsolv_scPrecondElman(rparBlock, chLayer, imgLevel, h_Dc, h_Dr, rmdSolver, bpureNeumann, caddOutput, cverbosity, ifileUnit, sls, irecursionDepth, rstat, RblockMat)

Description:
++++++++++++++++ Under Construction ++++++++++++++++++ Preconditioner based on Elman's paper on block commutators ++++++++++++++++ Under Construction ++++++++++++++++++

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
h_Dc !full entries	integer		handle for correction vector
h_Dr !averaged entries	integer		handle for defect vector
rmdSolver	t_multidimSolver		data about the multidim solver to be used
bpureNeumann	logical		flag, if a pure Neumann problem has to be solved
caddOutput	integer		level of additional file output
cverbosity	integer		level of verbosity
ifileUnit	integer		file unit for additional file output
sls	c(*)		leading string for outputs
irecursionDepth	integer		recursion depth of recursive calls of mdsolv_perform
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.20 Subroutine mdsolv_assignLaplaceMatrixU

Interface:
mdsolv_assignLaplaceMatrixU(RblockMat)

Description:
Auxiliary routine for mdsolv_scPrecondElman: Unveil a Laplace matrix of the velocity space to the modul multidimsolver. This way it is not necessary to pass the Laplace matrix through all subroutine calls.

Input variables:

Name	Type	Rank	Description
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix

7.32.21 Subroutine mdsolv_assignMassMatrixU

Interface:

`mdsolv_assignMassMatrixU(RblockMat)`

Description:
Auxiliary routine for mdsolv_scPrecondElman: Unveil a mass matrix of the velocity space to the modul multidimsolver. This way it is not necessary to pass the mass matrix through all subroutine calls.

Input variables:

Name	Type	Rank	Description
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.22 Subroutine mdsolv_clearVector

Interface:

`mdsolv_clearVector(rparBlock, chLayer, imgLevel, cmgFlag, H_Dvec, rstat)`

Description:
Setting a blockstructured vector to zero

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`cmgFlag`	integer		multigrid level flag HL_ONE,HL_ALL
`H_Dvec`	`integer`	:	handle array for vector to be cleared

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		structure for storing runtime statistics

7.32.23 Subroutine mdsolv_copyVector

Interface:

`mdsolv_copyVector(rparBlock, chLayer, imgLevel, H_Dvec1, H_Dvec2, rstat)`

Description:
Copy blockstructured vector Dvec1 to Dvec2 (only on level imglevel!)

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
H_Dvec1	integer	:	handle array for source vector
H_Dvec2	integer	:	handle array for destination vector

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.24 Subroutine mdsolv_scaleVector

Interface:

`mdsolv_scaleVector(rparBlock, chLayer, imgLevel, H_Dvec, dscaleFactor, rstat)`

Description:
Scale blockstructured vector Dvec with dscaleFactor

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
H_Dvec	integer	:	handle array for vector to be scaled
dscaleFactor	real(DP)		scale factor

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.25 Subroutine mdsolv_addVector

Interface:

`mdsolv_addVector( rparBlock, chLayer, imgLevel, dcoeff1, H_Dvec1, dcoeff2, H_Dvec2, rstat)`

Description:
Compute Dvec2 = dcoeff1*Dvec1 + dcoeff2*Dvec2

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
H_Dvec1	integer	:	handle array for first vector
H_Dvec2	integer	:	handle array for vector to be overwritten
dcoeff1, dcoeff2	real(DP)		coefficients

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

7.32.26 Function mdsolv_scalarProd

Interface:

```
mdsolv_scalarProd(rparBlock, chLayer, imgLevel, RblockMat, H_Dvec1, H_Dvec2, bexchange, rstat)
```

Description:
Calculate the scalar product of two vectors. If the two vectors have to be exchanged, they are averaged afterwards again.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
chLayer	integer		hierarchical layer
imgLevel	integer		multigrid level
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix
H_Dvec1	integer	:	handle array for first vector
H_Dvec2	integer	:	handle array for second vector
bexchange	logical		flag if vectors have to be exchanged before calculation (The used FEAST routine for calculating scalar product requires vectors with full entries!)

Output variables:

Name	Type	Rank	Description
rstat	t_stat		structure for storing runtime statistics

Result:
real(DP) : !scalar product of the two vectors real(DP) :: dscalProd

7.32.27 Subroutine mdsolv_matVecMult

Interface:
`mdsolv_matVecMult( ** rparBlock, chLayer, imgLevel, RblockMat, H_Dx, H_Dy, rstat) ** **`

Description:
* ! Perform the matrix vector multiplication ** ! Dy = RblockMat * Dx ** ! Dx must have full entries, Dy will be averaged on return. ** !

Input variables:

Name	Type	Rank	Description

Output variables:

Name	Type	Rank	Description

7.32.28 Subroutine mdsolv_matVecMult

Interface:
`mdsolv_matVecMult( rparBlock, chLayer, imgLevel, RblockMat, H_Dx, H_Dy, rstat)`

Description:
Perform the matrix vector multiplication Dy = RblockMat * Dx where RblockMat does not contain everywhere the proper Dirichlet boundary conditions. (The matrix blocks that do not incorporate them properly have rform%bemulateBC set.) Dx must have full entries, Dy will be averaged on return.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix
`H_Dx`	`integer`	:	handle array for vector Dx
`H_Dy`	`integer`	:	handle array for solution vector

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		structure for storing runtime statistics

7.32.29 Subroutine mdsolv_matVecMultAdd

Interface:

`mdsolv_matVecMultAdd( rparBlock, chLayer, imgLevel, dc1, RblockMat, H_Dx, dc2, H_Dy, rstat)`

Description:
Compute matrix vector multiplicaton y = c1 * RblockMat * x + c2 * y x must have full entries, y will be averaged afterwards.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`dc1`	`real(DP)`		first coefficient
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix
`H_Dx`	`integer`	:	handle array for vector Dx
`dc2`	`real(DP)`		second coefficient
`H_Dy`	`integer`	:	handle array for solution vector

Output variables:

Name	Type	Rank	Description
`rstat`	`t_stat`		structure for storing runtime statistics

7.32.30 Subroutine mdsolv_exchAverageVector

Interface:

`mdsolv_exchAverageVector(rparBlock, chLayer, imgLevel, H_Dvec, RblockMat)`

Description:
Exchange and average a blockstructured vector

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`H_Dvec`	`integer`	:	handle array for vector to be exchanged and averaged
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.31 Subroutine mdsolv_exchVector

Interface:

`mdsolv_exchVector(rparBlock, chLayer, imgLevel, H_Dvec, RblockMat)`

Description:
Exchange a blockstructured vector

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`H_Dvec`	`integer`	:	handle array for vector to be exchanged
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.32 Subroutine mdsolv_averageVector

Interface:

`mdsolv_averageVector(rparBlock, chLayer, imgLevel, H_Dvec, RblockMat)`

Description:
average a blockstructured vector

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`H_Dvec`	`integer`	:	handle array for vector to be exchanged and averaged
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.33 Subroutine mdsolv_init

Interface:

`mdsolv_init( rparBlock, RblockMat, sMDSfile, ndof, RscarcSolver, rscPrecMatDiffusive, rscPrecMatReactive, dcoeffDiffusive, dcoeffReactive, nadditionalVectors)`

Description:
Reads solver definition for multidimensional problems and allocates necessary auxiliary vectors.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix (needed for vector allocation)
sMDSfile	c(*)		name of file which contains the solver definition
ndof	integer		Number of degrees of freedom per node (e.g. space dimension + 1 in case of a mixed u/p formulation). It is needed to determine the amount of vectors which have to allocated in advance.
RscarcSolver	t_solver	dim(:)	structure containing all ScaRC solver objects defined in master.dat
rscPrecMatDiffusive, rscPrecMatReactive	Tdiscrid	dim(:, :)	Matrices for preconditioning the diffusive and reactive part
dcoeffDiffusive, dcoeffReactive	real(DP)		Coefficients to weight the preconditioning matrices
nadditionalVectors	integer		If there is some testing phase and you need some additional vectors than usual then you can allocate some more. So, if this argument is present, nadditionalVectors additional HL_ALL-vectors will be allocated. (For example, the vectors needed for the Elman or the extended Gauss-Seidel preconditioner have not been considered.)

7.32.34 Subroutine mdsolv_initDirect

Interface:

`mdsolv_initDirect(rparBlock, RblockMat)`

Description:
This routine initialises the direct solver for the given block matrix. It has to be called before the solving process.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
RblockMat	Tdiscrid	dim(:, :)	blockstructured matrix (needed for vector allocation)

7.32.35 Subroutine mdsolv_initDirectMaster

Interface:

`mdsolv_initDirectMaster(rmdParBlock)`

Description:

This routine (RUNNING ON THE MASTER) applies the UMFPACK factorisation for the block matrix which is received from mdsolv_initDirect.

Input variables:

Name	Type	Rank	Description
rmdParBlock	t_masterDataParallelBlock		master parallel block

7.32.36 Subroutine mdsolv_readSolverDefinition

Interface:

mdsolv_readSolverDefinition(rparBlock, ifileUnit, ndof, RscarcSolver, rmdSolver, rscPrecMatDiffusive, rscPrecMatReactive, dcoeffDiffusive, dcoeffReactive)

Description:
This routines reads the multidim solver definition from the file connected to ifileUnit. While reading it also determines how many auxiliary vectors are needed.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
ifileUnit	integer		File unit for file containing definition of solver for multidimensional problems
ndof	integer		Number of degrees of freedom per node (e.g. space dimension + 1 in case of a mixed u/p formulation). It is needed to determine the amount of vectors which have to allocated in advance.
RscarcSolver	t_solver	dim(:)	structure containing all ScaRC solver objects defined in master.dat
rscPrecMatDiffusive, rscPrecMatReactive	Tdiscrid	dim(:, :)	Matrices for preconditioning the diffusive and reactive part
dcoeffDiffusive, dcoeffReactive	real(DP)		Coefficients to weight the preconditioning matrices

Output variables:

Name	Type	Rank	Description
rmdSolver	t_multidimSolver		data about the multidim solver to be used

7.32.37 Subroutine mdsolv_allocateAllVectors

Interface:

mdsolv_allocateAllVectors(rparBlock, RblockMat)

Description:
This routine allocates all auxiliary vectors needed for the defined solver. It uses the globally defined variables nnumVectorsAllMG and nnumVectorsMaxMG which have been determined while reading the solver definition file.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.38 Subroutine mdsolv_deallocateAllVectors

Interface:

```
mdsolv_deallocateAllVectors(rparBlock)
```

Description:
This routine deallocates all vectors that were allocated. When vectors are needed again, the routine mdsolv_init(...) has to be called again.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block

7.32.39 Function mdsolv_getHandle

Interface:

```
mdsolv_getHandle(cflagMG)
```

Description:
This routine finds a free handle of desired type (HL_ALL, HL_ONE) among the allocated vectors. The free handle pointer always points to the last free handle in the list. This is then removed and the pointer decreased by one until there are no free handles anymore.

Input variables:

Name	Type	Rank	Description
`cflagMG`	integer		flag HL_ALL or HL_ONE

Result:
integer : free handle integer, intent(out) :: h_Dvec

7.32.40 Subroutine mdsolv_releaseHandle

Interface:

```
mdsolv_releaseHandle(cflagMG, h_Dvec)
```

Description:
This routine releases the given handle of desired type (HL_ALL, HL_ONE). The released handle is appended to the list of free handles. (Consequently the order of handles will be mixed up, which is no problem, though.) WARNING! There is no check if the handle (=integer) is really associated with an allocated vector, since it is simply appended to the list of free handles. So, if it is not valid (e.g. if it some telephon number) a serious crash will likely occur! Be careful! (A validity check would, of course, be possible, but probably too time consuming or quite complicated to implement (linked list!). It is simply not worth the effort!)

Input variables:

Name	Type	Rank	Description
`cflagMG`	integer		flag HL_ALL or HL_ONE
`h_Dvec`	`integer`		handle to be released

7.32.41 Subroutine mdsolv_testConvergence

Interface:

`mdsolv_testConvergence(dres, dresInitial, dprevRes, dtol, brel, iiter, caddOutput, cverbosity, ifileUnit, sls, dconvRate, dconvRate1, btolReached)`

Description:
Test if iterative solver converged and output solver statistics

Input variables:

Name	Type	Rank	Description
`dres`	`real(DP)`		residuum
`dresInitial`	`real(DP)`		initial residuum
`dprevRes`	`real(DP)`		residuum from previous iteration
`dtol`	`real(DP)`		tolerance to be reached
`brel`	`logical`		relative or absolute tolerance
`iiter`	`integer`		number of iteration
`caddOutput`	integer		level of additional file output
`cverbosity`	integer		level of verbosity
`ifileUnit`	`integer`		file unit for additional file output
`sls`	`c(*)`		leading string for outputs

Output variables:

Name	Type	Rank	Description
`dconvRate`	`real(DP)`		convergence rate
`dconvRate1`	`real(DP)`		decrease of residuum from last iteration to current one
`btolReached`	`logical`		flag if tolerance is reached and iteration can be stopped

7.32.42 Function mdsolv_testDivergence

Interface:

`mdsolv_testDivergence(dvalue, icounter)`

Description:
Counts how many times the relative residuum increases monotonously. When a limit is reached, divergence of the solver method is indicated.

Result:

logical : Boolean value indicating divergence.

Input variables:

Name	Type	Rank	Description
`dvalue`	`real(DP)`		Value to test for monotonous increase (indicator for divergence)

Input/Output variables:

Name	Type	Rank	Description
`icounter`	`integer`		Counter for subsequent increases of relative residuum in current solver scheme

7.32.43 Subroutine mdsolv_deallocAuxVecFilteredMVM

Interface:

`mdsolv_deallocAuxVecFilteredMVM(rparBlock, chLayer, imgLevel, RblockMat)`

Description:
Deallocate an auxiliary vector to use whenever mdsolv_matVecMultFiltered is invoked. Call this routine when you have finished solving block-structured systems.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.44 Subroutine mdsolv_allocAuxVecFilteredMVM

Interface:

`mdsolv_allocAuxVecFilteredMVM(rparBlock, chLayer, imgLevel, RblockMat)`

Description:

Allocate an auxiliary vector to use whenever mdsolv_matVecMultFiltered is invoked. Call this routine once before invoking mdsolv_matVecMultFiltered.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block
`chLayer`	integer		hierarchical layer
`imgLevel`	`integer`		multigrid level
`RblockMat`	`Tdiscrid`	dim(:, :)	blockstructured matrix

7.32.45 Subroutine mdsolv_replaceTolerance

Interface:

`mdsolv_replaceTolerance(dtol)`

Description:
This routine overwrites the tolerance of the global multidimsolver. It has to be called after mdsolv_init(...). It can be useful when tests with several solvers are performed, where a special tolerance shall be reached. Thus, not all the *.mds files have to be modified. (An alternative is, of course, to use global shell variables as it is done in the benchmark tests.)

Input variables:

Name	Type	Rank	Description

7.32.46 Subroutine mdsolv_change2Abs

Interface:

`mdsolv_change2Abs()`

Description:
This routine changes the stopping criterion of the global multidimsolver to absolute. This is necessary for adaptive computations, where one starts with the solution from the previous iteration step. In this case, the initial residual is very small, and therefore the multidimsolver trie to reach ridicoluos accuracies.

7.33 Module transfer

Purpose: This modules contains routines to perform the prolongation and and restriction within the multigrid cycle. These routines are suitable for the conforming bilinear finite element.

7.33.1 Subroutine transfer_prolong

Interface:

`transfer_prolong(rparBlock, imacroIdx, chLayer, h_Dxc, h_Dxf, ilevelc, imatidx, rstat)`

Description:

This routine performs the prolongation operation from a vector on level ilevelc to level ilevelc+1. The coarse level vector must have FULL entries while the prolongated vector will have averaged entries.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
imacroIdx	integer		matrixblock id on HL_MB level
chLayer	integer		hierachical layer
h_Dxc	integer		handle of the coarse vector (must have FULL entries)
ilevelc	integer		level of the coarse grid
imatidx	Tdiscrid		discretization identifier

Output variables:

Name	Type	Rank	Description
h_Dxf	integer		handle of the fine vector (will have averaged entries)
rstat	t_stat		number of operations performed + time needed

7.33.2 Subroutine transfer_restrict

Interface:

transfer_restrict(rparBlock, imacroIdx, chLayer, h_Dxf, h_Dxc, ilevelc, imatidx, rstat)

Description:
This routine performs the restriction operation from a vector on level ilevelc + 1 to level ilevelc. The fine level vector must have FULL entries while the restricted vector will have averaged entries.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
imacroIdx	integer		matrixblock id on HL_MB level
chLayer	integer		hierachical layer
h_Dxf	integer		handle of the fine vector (must have FULL entries)
ilevelc	integer		level of the coarse grid
imatidx	Tdiscrid		discretization identifier

Output variables:

Name	Type	Rank	Description
h_Dxc	integer		handle of the coarse vector (will have averaged entries)
rstat	t_stat		number of operations performed + time needed

7.34 Module communication

Purpose: This module contains the basic routines for exchanging matrices and vectors and for communication for gathering operations like scalar products.

Constant definitions:

Name	Type	Purpose

7.34.1 Subroutine comm_mainloop_print

Interface:
`comm_mainloop_print()`

Description:
This routine writes strings it receives from the master process to the screen and into the log file iff the priority level of the message is high enough (i.e. higher than the minimum level specified configuration file of the application)

7.34.2 Subroutine comm_sendmasterstring

Interface:
`comm_sendmasterstring(smessage, clevel)`

Description:
This routine sends the given string to the master process and this process will print it out.

Input variables:

Name	Type	Rank	Description
clevel	integer		priority of message
smessage	c (*)		message string

7.34.3 Subroutine comm_exchangeMacros

Interface:
`comm_exchangeMacros(rparBlock, rtrf)`

Description:
This routine performs a macro exchange.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block

Output variables:

Name	Type	Rank	Description
rtrf	Tloadexc		structure with exchange information

7.34.4 Subroutine comm_getAddVar_int

Interface:
comm_getAddVar_int(rparBlock, svar, ivalue, bfound)

Description:
This routine asks the masterprocess for the integer value of the given variable, stored in the master data file.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block
svar	c(*)		name of the variable

Output variables:

Name	Type	Rank	Description
ivalue	integer		return value of the variable
bfound	logical		set to .TRUE., if the variable svar has been found in the master data file.

7.34.5 Subroutine comm_getAddVar_double

Interface:
comm_getAddVar_double(rparBlock, svarName, dvalue, bfound)

Description:
This routine asks the masterprocess for the double value of the given variable, stored in the master data file.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
svarName	c(*)		name of the variable

Output variables:

Name	Type	Rank	Description
dvalue	real(DP)		return value of the variable
bfound	logical		set to .TRUE., if the variable with name svarName has been found in the master data file.

7.34.6 Subroutine comm_getAddVar_string

Interface:

comm_getAddVar_string(rparBlock, svarName, svalue, bfound)

Description:
This routine asks the masterprocess for the string value of the given variable, stored in the master data file.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		
svarName	c(*)		name of the variable

Output variables:

Name	Type	Rank	Description
svalue	c(*)		return value of the variable
bfound	logical		set to .TRUE., if the variable svarName has been found in the master data file.

7.34.7 Subroutine comm_adjustClusterComm

Interface:

comm_adjustClusterComm(rparBlock)

Description:
This routines modifies the communication exchange structure with respect to clustered macros

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object

7.34.8 Subroutine comm_exchange

Interface:

comm_exchange(rparBlock, chLayer, imgLevel, h_Dvec, rdiscrId)

Description:
This routines performs a vector exchange for the given variable. The routine adds the exchanged values to the corresponding boundary nodes.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
chLayer	integer		hierachical layer
imgLevel	integer		multigrid level
rdiscrId	Tdiscrid		id of the discr the vector belongs to

Input/Output variables:

Name	Type	Rank	Description
h_Dvec	integer		handle of the vector to be exchanged

7.34.9 Subroutine comm_exchangeModifier

Interface:

comm_exchangeModifier(rparBlock, chlayer, imgLevel, ivarIdx, rdiscrId, csubmodus)

Description:
The aim of this routine is to compute priorisation information for the macro edges. To do so, the vector dvecsave, which is intended for the averaging and exchange of global vectors, is abused: Before the call of this subroutine, it is filled with the index of the matrixblock the current macro belongs to. Here, the values in the macro edges, which occur twice, are overwritten with the smaller value. After this (in tools_init), by comparing the value in this vector and the matrix block index the macro belongs to, one obtains the desired priority information (priority, if these values coincide). This information is needed e.g. in global scalar products, where the macro boundary values must not be count twice. Here, the use of a whole vector for storing just one number seems ineffective, but remember that the vector dvecsave has to be allocated anyway, if global averaging or exchanging occurs, which is nearly unavoidable.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
chlayer	integer		hierachical layer
imgLevel	integer		multigrid level
csubmodus	integer		submodus 0 = min
rdiscrId	Tdiscrid		id of the discr the vector belongs to

Input/Output variables:

Name	Type	Rank	Description
ivarIdx	integer		variable index (ist handle!!!!)

7.34.10 Subroutine comm_route

Interface:
`comm_route(rparBlock)`

Input variables:

Name	Type	Rank	Description
rparBlock ! parallel block object	t_parBlock		

7.34.11 Subroutine comm_average

Interface:
`comm_average(rparBlock, chlayer, imgLevel, h_Dvec, cflag, rdiscrId)`

Description:
This subroutine performs averaging to a vector with handle h_Dvec on the borders of macros and matrix blocks, respectively. This is necessary to ensure the consistency of the vector in the boundary nodes of a macro and matrix block, respectively. Rolle von cflag???

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
chlayer	integer		given hierachical layer
cflag	integer		destination hierachical layer
rdiscrId	Tdiscrid		id of the discretisation

Input/Output variables:

Name	Type	Rank	Description
h_Dvec	integer		handle of the vector to average.

7.34.12 Subroutine comm_average_single

Interface:

`comm_average_single(rparBlock, i, chlayer, imgLevel, h_Dvec, cflag, rdiscrId)`

Description:
This subroutine performs averaging to a vector with handle h_Dvec on the borders of macros and matrix blocks, respectively. This is necessary to ensure the consistency of the vector in the boundary nodes of a macro and matrix block, respectively. Rolle von cflag???

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`chlayer`	integer		given hierachical layer
`cflag`	integer		destination hierachical layer
`rdiscrId`	`Tdiscrid`		id of the discretisation

Input/Output variables:

Name	Type	Rank	Description
`h_Dvec`	`integer`		handle of the vector to average.

7.34.13 Subroutine oldcomm_average_single

Interface:

`oldcomm_average_single(rparBlock, i, chlayer, imgLevel, h_Dvec, cflag, rdiscrId)`

Description:
This subroutine converts a vector on a single matrixblock from a given hierachical layer to another layer.

Input variables:

Name	Type	Rank	Description
`i`	`integer`		index of the matrixblock
`rparBlock`	`t_parBlock`		parallel block object
`chlayer`	integer		given hierachical layer
`cflag`	integer		destination hierachical layer
`rdiscrId`	`Tdiscrid`		id of the discretisation

Input/Output variables:

Name	Type	Rank	Description
`h_Dvec`	`integer`		

7.34.14 Subroutine comm_exchangeMatrix

Interface:

`comm_exchangeMatrix(rparBlock, imgLevel, ihlayer, imatidx)`

Description:
This function performs an exchange of the matrix entries at the parallel block boundaries.

Input variables:

Name	Type	Rank	Description
`rparBlock`	`t_parBlock`		parallel block object
`imgLevel`	`integer`		multigrid level
`ihlayer`	`integer`		hierachical layer of the exchange HL_SD

Input/Output variables:

Name	Type	Rank	Description
`imatidx`	`type(Tdiscrid)`		discretisation id

7.34.15 Subroutine comm_globalMin

Interface:

`comm_globalMin(dmin)`

Description:
This routine computes the minimum of all variables dmin from the different processors. Afterwards all variables contain this minimum.

Input/Output variables:

Name	Type	Rank	Description
`dmin`	`real(DP)`		

7.34.16 Subroutine comm_globalMax

Interface:

`comm_globalMax(dmax)`

Description:
This routine computes the maximum of all variables dmax from the different processors. Afterwards all variables contain this maximum.

Input/Output variables:

Name	Type	Rank	Description
dmax	real(DP)		

7.34.17 Subroutine comm_sum_double

Interface:

comm_sum_double(dsum)

Description:
This routine sums up the value dsum coming from the single processors. The sum is then written into the same variable, such that all variables contain this sum afterwards.

Input/Output variables:

Name	Type	Rank	Description
dsum	real(DP)		

7.34.18 Subroutine comm_sum_int

Interface:

comm_sum_int(isum)

Description:
This routine sums up the value isum coming from the single processors. The sum is then written into the same variable, such that all variables contain this sum afterwards.

Input/Output variables:

Name	Type	Rank	Description
isum	integer		

7.34.19 Subroutine comm_sum_int64

Interface:

comm_sum_int64(isum)

Description:
This routine sums up the value isum coming from the single processors. The sum is then written into the same variable, such that all variables contain this sum afterwards.

Input/Output variables:

Name	Type	Rank	Description
isum	integer(I64)		

7.34.20 Subroutine comm_sendVisData

Interface:

`comm_sendVisData( rparBlock, nnumberOfDataVectors, nnumberOfAddDataVectors, H_DdataVector, SdataVectorName, CdataVectorType, CtreatDataVectors, nnumberOfAddIntVars, IaddIntVars, nnumberOfAddDblVars, IaddDblVars, cmiscData, svisFilename, ivisOutputLevel, rdiscrId, buseUserVisExport)`

Description:
This subroutine sends data for visualisation on specified output level to the master process.

Input variables:

Name	Type	Rank	Description
rparBlock	t_parBlock		parallel block object
nnumberOfDataVectors	integer		number of solution vector to be exported to visualisation output file
nnumberOfAddDataVectors	integer		number of data vectors that will be calculated on-the-fly by user_additionalVisData when performing UCD output
H_DdataVector	integer	:	handle array for solution vectors to be exported to visualisation output file
SdataVectorName	c (*)	:	name of solution vector, to be displayed in visualisation program
CdataVectorType	integer	:	type of data of solution vector: node-oriented, edge-oriented, ...
CtreatDataVectors	integer	:	array describing what to do with data vectors: export or not, agglomerate velocity components to a velocity field or not Use flags like UCD_NOEXPORT, UCD_NO_AGGLOMERATE, UCD_AGGLO_VELOCITYFIELD etc.
nnumberOfAddIntVars	integer		number of additional integer variables to send (e.g. iteration number, parameters for user_additionalVisData)
IaddIntVars	integer	:	array containing additional integer variables to send
nnumberOfAddDblVars	integer		number of additional double variables to send (e.g. time step, parameters for user_additionalVisData)
IaddDblVars	real(DP)	:	array containing additional double variables to send
cmiscData	integer		bit array describing what kind of grid statistics to export Use flags like UCD_MINCONVRATE, UCD_HMIN, UCD_ASPECTRATIOS etc.
svisFilename	c (*)		output file name
ivisOutputLevel	integer		visualisation output level
rdiscrId	Tdiscrid		matrixblock object
buseUserVisExport	logical		optional flag whether to a user-specified function to export the data to disk. If unspecified or set to .FALSE. a kernel routine will be used.

7.34.21 Subroutine comm_receiveVisData

Interface:

comm_receiveVisData(mdParBlock)

Description:
This routine is invoked by the master process as soon as a slave process has sent a message with the identifier COMM_MASTER_RECEIVEVISDATA in routine comm_sendVisData. All data vectors from all parallel blocks are received and memory as needed to store them is allocated. The actual export to file in some UCD format is done afterwards. The decision whether to use a kernel routines like master_writeAVS / master_writeGMV / etc. or a user-specific routine is made in routine user_writeVisData which is called at the end of this routine.

Input variables:

Name	Type	Rank	Description
mdParBlock	t_masterDataParallelBlock		

7.34.22 Subroutine comm_sendglobaltime

Interface:
comm_sendglobaltime(rparBlock, giter, modus, rstat, iter, dlcappamin, dlcappamax, dlcappaavg, dres, dresInitial, dgcappa)

Description:
This client subroutine sends several status information to the server process.

Input variables:

Name	Type	Rank	Description
rparBlock	type(t_parBlock)		
modus	integer		mode variable(0=starts global time meas., .ne.0=delta time meas.
rstat	t_stat		count of operations
iter	integer		type (Tops) :: iops iteration number
dlcappamin	real(DP)		minimal convergance rate
dlcappamax	real(DP)		maximal convergance rate
dlcappaavg	real(DP)		average convergance rate
dres	real(DP)		global residuum
dresInitial	real(DP)		initial global residuum
dgcappa	real(DP)		global convergance rate

7.35 Module parallel (MPI version)

Purpose: This module defines the basic low level communication routines. It provides routines for exchanging integer and double values. Further it contains mechanisms for synchronisation of parallel processes.

Global variable definitions:

Name	Type	Rank	Purpose
PSYMODE_NORMAL	integer		buffered transfer mode
PSYMODE_SYNC	integer		syncr buffered transfer mode
PSYMODE_IMM	integer		immdiatly transfer mode
PARSYNC	integer		default transfer mode

7.35.1 Subroutine par_senderror

Interface:

```
par_senderror()
```

Description:
This client subroutine sends the error signal to the master process. After receiving this message the master cancels the other client processes.

7.35.2 Subroutine par_errorcancel

Interface:

```
par_errorcancel(index)
```

Description:
This server subroutine sends the cancelation signal to all clients except the process which caused the error.

Input variables:

Name	Type	Rank	Description
index	integer		index of process caused the error

7.35.3 Subroutine par_masterwait

Interface:

```
par_masterwait(index)
```

Description:
This server subroutine waits for a synchronisation signal from the specified client process.

Input variables:

Name	Type	Rank	Description
index	integer		index of process waiting for

7.35.4 Subroutine par_sendack

Interface:

```
par_sendack()
```

Description:
This client subroutine sends the synchronisation signal to the server process.

7.35.5 Subroutine par_waitack

Interface:

```
par_waitack()
```

Description:
This client subroutine waits for the acknowledge signal from the server process.

7.35.6 Subroutine par_doack

Interface:

```
par_doack()
```

Description:
This master subroutine sends the ackowledge signal to all client processes.

7.35.7 Subroutine par_initmsg

Interface:

```
par_initmsg(id)
```

Description:
This routine initialises the message buffer for a new message.

Input variables:

Name	Type	Rank	Description
`id`	`integer`		identifier of the new message

7.35.8 Subroutine par_recmsg

Interface:

```
par_recmsg(index, iutag, id)
```

Description:
This routine waits for a message from the specified process or from all processes and returns the message in the internal message buffer and the message identifier.

Input variables:

Name	Type	Rank	Description
`index`	`integer`		destination index (-1= all processes)
`iutag`	`integer`		message tag

Output variables:

Name	Type	Rank	Description
`id`	`integer`		identifier of the received message

7.35.9 Subroutine par_writemsgstring

Interface:
`par_writemsgstring(sdata)`

Description:
This subroutine writes a string in the message buffer.

Input variables:

Name	Type	Rank	Description
`sdata`	`c (*)`		string to be put in the message buffer
`stmp`	`c (1024)`		

7.35.10 Subroutine par_readmsgstring

Interface:
`par_readmsgstring(sdata)`

Description:
This routine reads a string from the message buffer.

Output variables:

Name	Type	Rank	Description
`sdata`	`c (*)`		string to get from the message buffer

7.35.11 Subroutine par_readmsgint

Interface:
`par_readmsgint(idata)`

Description:
This routine reads an integer from the message buffer.

Output variables:

Name	Type	Rank	Description
`idata`	`integer`		integer value to be read in from the message buffer

7.35.12 Subroutine par_readmsglint

Interface:
`par_readmsglint(idata)`

Description:
This routine reads an long integer from the message buffer.

Output variables:

Name	Type	Rank	Description
`idata`	`integer(I64)`		long int to be read from the message buffer

7.35.13 Subroutine par_readmsgints

Interface:
`par_readmsgints(idata, ilen)`

Description:
This routine reads an integer array from the message buffer.

Input variables:

Name	Type	Rank	Description
`ilen`	`integer`		length of the array

Output variables:

Name	Type	Rank	Description
`idata`	`integer`	:	integer array read from the msg buffer

7.35.14 Subroutine par_readmsgints2

Interface:
`par_readmsgints2(idata, ilen1, ilen2)`

Description:
This routine reads an integer array from the message buffer.

Input variables:

Name	Type	Rank	Description
`ilen1`	`integer`		length dimension 1
`ilen2`	`integer`		length, dimension 2

Output variables:

Name	Type	Rank	Description
`idata`	`integer`	:, :	integer array read from the msg buffer

7.35.15 Subroutine par_readmsgdouble

Interface:
`par_readmsgdouble(data)`

Description:
This routine reads an double from the message buffer.

Output variables:

Name	Type	Rank	Description
`data`	`real(DP)`		double to be read from the message buffer

7.35.16 Subroutine par_readmsgdoubles

Interface:
`par_readmsgdoubles(data, ilen)`

Description:
This routine reads an double array from the message buffer.

Input variables:

Name	Type	Rank	Description
`ilen`	`integer`		length of the array

Output variables:

Name	Type	Rank	Description
`data`	`real(DP)`	:	double array read from the msg buffer

7.35.17 Subroutine par_readmsgdoubles2

Interface:
`par_readmsgdoubles2(data, ilen1, ilen2)`

Description:
This routine reads an double array from the message buffer.

Input variables:

Name	Type	Rank	Description
`ilen1`	`integer`		length dimension 1
`ilen2`	`integer`		length dimension 2

Output variables:

Name	Type	Rank	Description
`data`	`real(DP)`	:, :	double array read from the msg buffer

7.35.18 Subroutine par_readmsglogical

Interface:

`par_readmsglogical(bdata)`

Description:
This routine reads a logical value from the message buffer.

Input variables:

Name	Type	Rank	Description
`bdata`	`logical`		boolean value to be read from the message buffer

7.35.19 Subroutine par_writemsgint

Interface:

`par_writemsgint(idata)`

Description:
This routine writes an integer to the message buffer.

Input variables:

Name	Type	Rank	Description
`idata`	`integer`		int to be put in the message buffer

7.35.20 Subroutine par_writemsglint

Interface:

`par_writemsglint(idata)`

Description:

This routine writes an long integer to the message buffer.

Input variables:

Name	Type	Rank	Description
idata	integer(I64)		int to be put in the message buffer

7.35.21 Subroutine par_writemsgints

Interface:
par_writemsgints(idata, ilen)

Description:
This routine writes an integer array to the message buffer.

Input variables:

Name	Type	Rank	Description
idata	integer	:	integer array to write in the msg buffer
ilen	integer		length of the array

7.35.22 Subroutine par_writemsgints2

Interface:
par_writemsgints2(idata, ilen1, ilen2)

Description:
This routine writes an integer array to the message buffer.

Input variables:

Name	Type	Rank	Description
idata	integer	:, :	integer array write to the msg buffer
ilen1	integer		length dimension 1
ilen2	integer		length dimension 2

7.35.23 Subroutine par_writemsgdouble

Interface:
par_writemsgdouble(data)

Description:
This routine writes an double to the message buffer

Input variables:

Name	Type	Rank	Description
data	real(DP)		double to be put in the message buffer

7.35.24 Subroutine par_writemsgdoubles

Interface:

par_writemsgdoubles(data, ilen)

Description:
This routine writes an double array to the message buffer

Input variables:

Name	Type	Rank	Description
data	real(DP)	:	double array written to the msg buffer
ilen	integer		length of the array

7.35.25 Subroutine par_writemsgdoubles2

Interface:

par_writemsgdoubles2(data, ilen1, ilen2)

Description:
This routine writes an double array to the message buffer.

Input variables:

Name	Type	Rank	Description
data	real(DP)	:, :	double array written from the msg buffer
ilen1	integer		length dimension 1
ilen2	integer		length dimension 2

7.35.26 Subroutine par_writemsglogical

Interface:

par_writemsglogical(bdata)

Description:
This routine writes a logical value to the message buffer.

Input variables:

Name	Type	Rank	Description
bdata	logical		logical to be put in the message buffer

7.35.27 Subroutine par_sendmsg

Interface:

par_sendmsg(index, itag)

Description:
This routine sends a new message from the actual buffer to the specified process.

Input variables:

Name	Type	Rank	Description
index	integer		index of destination process
itag	integer		message tag

7.35.28 Function par_getNumberOfParProcesses

Interface:

par_getNumberOfParProcesses()

Description:
Returns the number of parallel processes the program consists of

7.35.29 Function par_maxSizeMessageChunkInt

Interface:

par_maxSizeMessageChunkInt(icount)

Description:
Returns the maximum number of allowed subsequent calls of par_writemsgint before a call of par_send() is mandatory to prevent a send buffer overflow. The message buffer should be empty when relying on this function on both master and slave processes. So, it is strongly advised to call par_msginit before using the data from this function to divide messages into chunks!

7.35.30 Function par_maxSizeMessageChunkDouble

Interface:

par_maxSizeMessageChunkDouble(icount)

Description:
Returns the maximum number of allowed subsequent calls of par_writemsgdouble before a call of par_send() is mandatory to prevent a send buffer overflow. The message buffer should be empty when relying on this function on both master and slave processes. So, it is strongly advised to call par_msginit before using the data from this function to divide messages into chunks!

7.35.31 Subroutine par_finish

Interface:
`par_finish()`

Description:
This subroutine needs to be called by all slaves to indicate that they are about to exit.

7.36 Module parallelsys (MPI version (for all systems))

Purpose: This module implements the parallel initialisation routines.

Constant definitions:

Name	Type	Purpose
PAR_MAXBUFFER	integer	MPI buffer size
PAR_MAXPP	integer	maximal number of parallel processes
PAR_MAXPP_NDIGITS	integer	number of digits the maximal number of parallel processes has
PVMENCODING	integer	
PAR_ANSWERRES	integer	
PAR_ANSWEROPC	integer	
PAR_ANSWERFIN	integer	
PAR_EOMESSAGEFRAGMENT_REAL	real(DP)	
PAR_EOMESSAGEFRAGMENT_INT	integer	
PAR_IDINT	integer	
PAR_IDINTS	integer	
PAR_IDDOUBLE	integer	
PAR_IDDOUBLES	integer	
PAR_IDMACRO	integer	
PAR_IDCMD	integer	
PAR_IDACK	integer	
PAR_IDROUTE	integer	
PAR_IDEDGE	integer	
PAR_IDROUTEDIAGREQ	integer	
PAR_IDROUTEDIAGANS	integer	
PAR_IDDIAG	integer	
PAR_IDBOUNDARY	integer	

PAR_IDSCARCDEF	integer
PAR_IDCOARSEGRID	integer
PAR_IDGLBVECTOR	integer
PAR_ERROR	integer
PAR_FATALERROR	integer
PAR_IDFORM	integer
PAR_IDBC	integer

Global variable definitions:

Name	Type	Rank	Purpose
par_shome	c(256)		working directory
iparself	integer		own parallel index id of the process
nmaxproc	integer		number of parallel processes

7.36.1 Subroutine par_exit

Interface:

```
par_exit()
```

Description:
This subroutine closes the parallel communication library.

7.36.2 Subroutine par_abort

Interface:

```
par_abort()
```

Description:
This subroutine aborts the application properly.

7.36.3 Subroutine par_init

Interface:

```
par_init(shome, slog, iueb)
```

Description:
This subroutine initialises the parallel communication library.

Input variables:

Name	Type	Rank	Description
shome	c (*)		path to the working directory
slog	c (*)		path to the log directory
iueb	integer		reserved (number of parallel blocks)

7.36.4 Subroutine par_sendfatalerror

Interface:

`par_sendfatalerror(ierror)`

Description:
This routine sends the error code and the FATAL signal to the master process.

Input variables:

Name	Type	Rank	Description
`ierror`	`integer`		error variable

7.36.5 Subroutine par_preinit

Interface:

`par_preinit(irank)`

Description:
Initialises MPI environment and gets number of current process

Output variables:

Name	Type	Rank	Description
`irank`	`integer`		number of the calling process

Bibliography

[1] M. Altieri, C. Becker, and S. Turek. On the realistic performance of linear algebra components in iterative solvers. In H.-J. Bungartz, F. Durst, and C. Zenger, editors, *High Performance Scientific and Engineering Computing: Proceedings of the International FORTWIHR Conference on HPSEC, Munich, March 16–18, 1998*, volume 8 of *Lecture Notes in Computational Science and Engineering*, pages 3–12. Springer, Berlin, 1999. ISBN 3-540-65730-4.

[2] C. Becker. *FEAST – The Realisation of Finite Element Software for High–Performance Applications*. PhD thesis, Universität Dortmund, 2004. to appear.

[3] C. Becker, S. Kilian, and S. Turek. Consequences of modern hardware design for numerical simulation and their realization in feast, Aug. 1999. Vortrag, EuroPar99 Parallel Processing, Toulouse, France.

[4] C. Becker and S. Turek. Featflow – finite element software for the incompressible Navier–Stokes equations. User manual, Universität Dortmund, 1999.

[5] G. Burns, R. Daoud, and J. Vaigl. LAM: An Open Cluster Environment for MPI. In J. W. Ross, editor, *Proceedings of Supercomputing Symposium*, pages 379–386. nn, 1994. `http://www.lam-mpi.org/download/files/lam-papers.tar.gz`.

[6] Cederquist. *Version Management with CVS*. Cederqvist et al, 2003. `http://www.cvshome.org/docs/manual/cvs.html`.

[7] P. DeViSoR. Endbericht der Projektgruppe DeViSoR. Ergebnisberichte des Instituts für Angewandte Mathematik, Nr. 240t, FB Mathematik, Universität Dortmund, 2003.

[8] S. Kilian. *Ein verallgemeinertes Gebietszerlegungs-/Mehrgitterkonzept auf Parallelrechnern*. PhD thesis, Universität Dortmund, 2001.

[9] S. Kilian and S. Turek. An example for parallel scarc and its application to the incompressible Navier–Stokes equations. Preprints SFB 359, Nr. 98-06, Universität Heidelberg, 1998.

[10] R. Rannacher and R. Becker. A feed–back approach to error control in finite element methods: Basic analysis and examples. *East-West Journal of Numerical Mathematics*, 4:237–264, 1996.

[11] J. M. Squyres and A. Lumsdaine. A Component Architecture for LAM/MPI. In J. Dongarra, D. Laforenza, and S. Orlando, editors, *Recent Advances in Parallel Virtual Machine and Message Passing Interface, 10th European PVM/MPI Users‘ Group Meeting, Venice, Italy, September 29 - October 2, 2003, Proceedings*, number 2840 in Lecture Notes in Computer Science, pages 379–387. Springer, Berlin, Sept. 2003.

[12] S. Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, Berlin, 1999.

[13] S. Turek, C. Becker, and S. Kilian. Hardware–oriented numerics and concepts for PDE software. Ergebnisberichte des Instituts für Angewandte Mathematik, Nr. 235, FB Mathematik, Universität Dortmund, 2003. to be published in a special issue on PDE Software in the Elsevier journal Future Generation Computer Systems.

[14] S. Turek, M. Schäfer, and R. Rannacher. Evaluation of a CFD benchmark for laminar flows. In H. G. Bock, Y. A. Kuznetsov, R. Glowinski, J. Periaux, and R. Rannacher, editors, *ENUMATH 97: Proceedings of the 2nd European Conference on Numerical Mathematics and Advanced Applications*, pages 549–563. World Science Publishing, 1998.