Implementation and Performance Analyses of a Highly Efficient Algorithm for Pressure-Velocity Coupling

Implementierung und Untersuchung einer hoch effizienten Methode zur Druck-Geschwindigkeits-Kopplung

Master-Thesis von Fabian Gabel

Tag der Einreichung:

1. Gutachten: Prof. Dr. rer. nat. Michael Schäfer
2. Gutachten: Dipl.-Ing Ulrich Falk

TECHNISCHE UNIVERSITÄT DARMSTADT

Studienbereich CE
FNB

Implementation and Performance Analyses of a Highly Efficient Algorithm for Pressure-Velocity Coupling
Implementierung und Untersuchung einer hoch effizienten Methode zur Druck-Geschwindigkeits-Kopplung

Vorgelegte Master-Thesis von Fabian Gabel

1. Gutachten: Prof. Dr. rer. nat. Michael Schäfer
2. Gutachten: Dipl.-Ing Ulrich Falk

Tag der Einreichung:

Erklärung zur Master-Thesis

Hiermit versichere ich, die vorliegende Master-Thesis ohne Hilfe Dritter nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus Quellen entnommen wurden, sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, den 7. März 2015

(Fabian Gabel)

Contents

List of Figures

List of Tables

List of Algorithms

TODO LIST

- Bilder zum Konvergenzverlauf sind gut
- wall boundary condition treatment could improve convergence
- say that viscous dissipation is not accounted for in the energy balance
- midpoint integration rule or midpoint rule of integration
- remove equation numbering when there is no reference
- change the matrix coefficient indexing to $a_P^{u_i,p}$ like in [**?**].
- extend nomenclature
- align equations using the **alignat** environment
- check pressure correction equation on iteration indices (gradients)
- check simple chapter, since the right hand side lacks of deferred corrector and under-relaxed velocities
- mention the boundary conditions for the pressure correction
- check all headings for correct spelling
- mention that for an unknown velocity field the partial differential equation for the temperature is non-linear as well
- consistent use of either temperature or energy equation
- check the signs in the Boussinesq approximation
- pressure weighted interpolation method for large body forces?
- not only speed but also improvement of robustness
- instationary flows?
- align exponents in the equation for the Newton-Raphson linearization
- define the term consistent
- add citations to clipper, opencascade and maple, ICEM CFD
- a priori exact solution use this formulation
- Falsche Wahl der problem domain, führt zu global nicht erfüllter kontinuitätsgleichung. residuum der druckkorrectur entspricht dem massenfluss
- reference Comparison of finite-volume numerical methods with staggered and colocated
- read klaij again and use some of its arguments grids
- emphasize the flexibility of the implementation
- This is often referred to as "matrix-free", though it is still a Mat in PETSc (Mat in PETSc means "finite-dimensional linear operator").

1 CAFFA Framework

This section presents the CAFFA framework that has been extended for the present thesis. *CAFFA* is an acronym and stands for *Computer Aided Fluid Flow Analysis*. The solver framework is based on Perić's CAFFA code [**?**,**?**] that has been modified to handle three-dimensional problem domains and block structured grids with non-matching blocks. Furthermore, the framework has been parallelized making extensive use of the PETSc library. One characteristic of the solver framework is, that arbitrary block boundaries are handled in a fully implicit way. Details on how those transitional conditions are handled are located in section **??**. The framework consists of two different solver algorithms, a segregated solver and a fully coupled solver, which have been introduced in sections **??** and **??** respectively. In addition, the framework features an own implementation of a grid generator for block structured grids as well as a mapper program to convert grids created with ICEM CFD [**?**]. To prepare the grid data for the later use in the solvers, a preprocessor program has been integrated into the framework. Furthermore the solver programs exhibit different export functionalities to visualize the numerical grid and the results within ParaView [**?**] and the export of binary vectors and matrices to MATLAB ® [**?**].

FIGURE

The following subsections will now briefly introduce the PETSc framework, present the components of the CAFFA framework and finally sketch the program flow.

1.1 PETSc Framework

PETSc is an acronym for *Portable Extensible Toolkit for Scientific Calculations* and is a software suite that comprises data structures and an extensive set of linear solver routines [**?**,**?**]. A great part of the data structures is designed to work in parallel on high performance computers, also the solver routines have been parallelized using the MPI standard for message passing. PETSc provides the tools that can be used to build large-scale application codes for scientific calculations. The great advantage of PETSc is that adds multiple additional layers of abstraction that make its use in applications straightforward and allow the users to focus on the essential implementations of their own code instead of having to deal with parallel data structures and algorithms, which represents an additional source for programming errors. PETSc provides different interfaces through which the user can parallelize his code, what furthermore increases readability and maintainability of the developed code. Last but not least, the implementations used for data structures and solvers in PETSc have are not only verified and updated on a regular basis, they have furthermore proven to be highly efficient in the creation of large-scale computer applications on high performance clusters REFERENCE.

The data structures PETSc offers reflect the commonly used elements in scientific codes. Vectors, matrices and index sets are among this basic data structures. PETSc offers a variety of options for its data structures. Vectors can either be used sequentially or parallelly in which case PETSc distributes the vector components to the processes inside a MPI communicator. Furthermore parallel matrix formats are available that allow the processes to assemble different parts of one global matrix simultaneously. PETSc contains other data structures that can be used to fetch and distribute either vector or matrix elements from remote processes.

On the other side PETSc contains a large collection of parallel solvers for linear systems. The major part comprises Krylov subspace methods which have been introduced in section **??**. Among the variety of solver options that are available for further optimization of the performance, the Krylov subspace methods can be combined with elements of another thorough collection of preconditioners. For further details on the available data structures and subroutines the reader is referred to [**?**,**?**].

In addition to the mentioned tools PETSc also comes with an internal profiler tool that collects a variety of information on the used PETSc objects and bundles them into a human readable log file. The log file and other dynamic output which can be triggered by command line arguments facilitate the optimization process of developed applications and allow the accurate determination of performance measures.

FIGURE

1.2 Grid Generation and Preprocessing

A grid generator has been developed to generate hexahedral block structured grids. To provide flexibility for testing locally refined and skewed grids the grid generator deploys a random number generator which moves grid points within the domain and varies the number of grid cells of each block optionally. Figure REFERENCE shows an example grid, that was generated with the developed grid generator.

FIGURE

Key feature of the developed framework is the handling of block structured locally refined grids with non-matching block interfaces. To neighbouring relations are represented by a special type of boundary conditions; Random number generator to move grid points within an epsilon neighbourhood while maintaining the grid intact.

1.3 Preprocessing

Matching algorithm – the idea behind clipper and the used projection technique; alt.: Opencascade. Efficient calculation of values for discretization. Important for dynamic mesh refinement, arbitrary polygon matching, parallelizable due to easier interface

1.4 Implementation of CAFFA

This section provides an overview of the implementation aspects of the developed CAFFA framework. Since applications for modern computers have to be able to efficiently use the provided resources the concept for the parallelization of the application is presented. A separate section presents how convergence is monitored and controlled. Later on, this section presents ho to use the data structures that store the variables and how to realize the domain composition as central part of the parallelization process.

1.4.1 The Message-Passing Model

The PETSc framework makes thoroughly use of the message-passing computational model [**?**]. This model assumes that a set of processes with local memory is able to communicate with other processes by passing, i.e. sending and receiving messages. *MPI* is an acronym for *Message Passing Interface* and is a software library that collects features of message-passing systems. More importantly it provides a widely used standard that assures portability of application that make use of this library as does PETSc.

The message-passing model is not only reflected in the application as a programming paradigm, but also in the design of parallel computers and their communication network. Under this perspective the message-passing model is complementary to the distributed memory design, which encapsulates the idea of local memory and the exchange of data through a network. High performance computers are seldom designed exclusively as a distributed memory system, rather they incorporate features of another model, namely the shared memory model. This memory model is characterized by a common address space, which each of the processes can access uniformly, i.e. with a location independent latency, or non-uniformly. Latter systems are also called *NUMA* systems (*Non-Uniform Memory Access*). A widely used system design, as for the system the performance tests of section REFERENCE are performed on, comprises features of both models by using NUMA computing nodes that share local memory and are able to exchange messages with other NUMA nodes via an interconnect.

One important advantage of using software that uses the message passing model and hence applications that were parallelized within this programming paradigm, is their compatibility with computers that use combinations of distributed and shared memory.

1.4.2 Convergence Control

One part of the PETSc philosophy is to provide great flexibility in choosing solvers and parameters from within the command line. However, the developed framework implements a non-linear iteration process without using the SNES objects provided by PETSc. This creates the need for a hard-coded implementation of convergence control. Convergence control for the used Picard iteration method comprises two parts. One part controls the overall convergence, and determines when the calculations have finished. The other part controls convergence on an outer iteration basis. The convergence criteria are met if the actual residual falls below the upper bound for the residual r_{final}. The bound for final convergence is hereby determined to

$$r_{final} = r_{initial} * 10^{-8},$$

which states that a decrease of the initial residual of 10^{-8} indicates convergence. This criterion will be fixed for all further analyses since it controls the overall accuracy of the solver results and hence maintains comparability of solver performance.

The other convergence criterion can be handled with more flexibility, since it controls the relative decrease of the residual in each outer iteration. This parameter should not affect the overall accuracy but instead convergence speed, within the individual bounds on the solver algorithm. This means that, depending on the coupling algorithm, low relative solver tolerances will not always benefit convergence speed. On the other hand low relative solver tolerances will affect the number of inner iterations and hence the execution time of the solver program. The convergence criterion for each outer iteration is implemented as

$$r^{(k)}_{final} = r^{(k)}_{initial} * \text{rtol},$$

where rtol indicates the relative decrease of the residual in each outer iteration.

1.4.3 Indexing of Variables and Treatment of Boundary Values

All variables needed by the CAFFA solver are represented by one-dimensional arrays. To establish a mapping between a location (i, j, k) of the numerical grid and the respective variable values location (ijk) inside the array the following formula is used

$$(ijk) = N_i N_j (k-1) + N_j (i-1) + j,$$

where N_i and N_j are the Number of grid cells in the respective coordinate direction plus two additional boundary cells. The inclusion of the boundary cells circumvents the need to provide an additional data structure for boundary values. Furthermore the same mapping rule can be used for the grid vertex coordinates. This variable indexing will be used to assemble the matrices surging from the discretization process. This will lead to rows for the boundary values, that that are decoupled from the rest of the linear system. PETSc provides two different approaches to handle boundary values. In the first approach another abstraction layer is introduced via an object, that redistributes the data and removes the rows containing the boundary values. This approach has been tested with the conclusion that the redistribution not only causes significant overhead but also hides important data for debugging solver convergence. In the present work a second approach is used, that retains boundary values and adapts the right hand side accordingly. This approach has proven to be more efficient with the drawback of memory overhead and the need to reset the boundary values after a linear system has been solved with a high relative tolerance.

This simple indexing model is not capable to handle neither block structured grids nor grids that have been distributed across various processors within the MPI programming model. This characteristic is implemented through the use of additional mappings that return a constant offset $(ijk)_b$) for each block and $(ijk)_p$ for each processor

$$(ijk) = (ijk)_p + (ijk)_b + N_i N_j (k-1) + N_j (i-1) + j.$$

The introduction of the processor offset is needed to provide a mapping between locally and globally indexed values. Within a process only the local indexing

$$(ijk)_{loc} = (ijk)_b + N_i N_j (k-1) + N_j (i-1) + j.$$

will be used, since these indexes can be mapped directly to memory addresses. For inter process communication however, global indices have to be provided to the respective PETSc subroutines by adding the processor offset

$$(ijk)_{glo} = (ijk)_{loc} + (ijk)_p. \tag{1}$$

It should be noted that the presented mappings do not include the index mappings from FORTRAN applications, which use 1-based indexing, to the PETSc subroutines which use 0-based indices.

As presented in section **??**, the use of a coupling algorithm leads to a global linear system that contains the linear algebraic equations for different variables. The implementation used for the present thesis interlaces this values, which increases data locality and hence improves memory efficiency. The presented mapping is easily extended to handle arrays of values that contain n_{eq} variables and use an interlaced storing approach by multiplying (1) with the number of interlaced values

$$(ijk)_{interglo} = n_{eq} * \big((ijk)_{glo} - 1\big) + c_{eq},$$

where $c_{eq} = 1, \ldots, n_{eq}$.

1.4.4 Domain Decomposition, Exchange of Ghost Values and Parallel Matrix Assembly

The developed CAFFA framework is able to solve a given problem across different processors. For this, before each solve, the relevant data has to be distributed among all involved processes. Within the solver parallelization three types of this data distribution are considered with respect to their frequency of communication and redistribution among all or only a part of the processes involved in the solution process.

The first type of data refers to the distribution of stationary data that will not change throughout the solution process. This refers mostly to geometry related data as the coordinates of the grid points. The distribution of the data is already part of the domain composition process and hence can be done even before the solver starts as it is implemented in the developed solver framework. It should be noted that in the case that local dynamic refinement strategies are used the geometric data at block boundaries has to be redistributed. If additionally dynamic load balancing is used even geometric data from the inside of the problem domain might be redistributed. This however imposes other design guidelines for the involved processes which is not inside the scope of the present thesis.

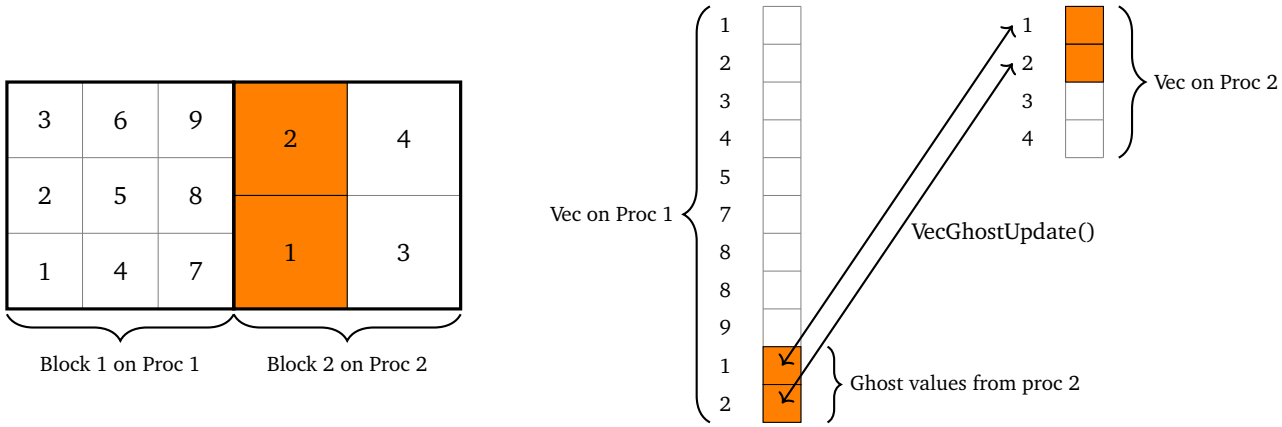

Figure 1: Non-zero structure of the linear systems used in the SIMPLE algorithm for a block structured grid consisting of one 3×3 and one 2×2 block

The second type of data has to be distributed in a regularly since this data changes at least every outer iteration. The necessity to interchange values, also known as *ghosting*, between processors surges when calculations of variable gradients or coefficients are made for block boundary control volumes. PETSc provides special interfaces for ghosted vectors that allow to perform different types of vector updates through top-level subroutines. The central part are the involved vector scatter and gather routines which allow to calculate contributions to source terms of control volumes located at block boundaries accurately or update ghost values for the control volumes that rely on variable values from neighbouring blocks to calculate fluxes or gradients. It should be noted that the modelling approach for ghosting values is not symmetric in the sense that each block saves ghost values of the neighbouring blocks. On the contrary, for each block boundary only one of the neighbored blocks is responsible for the calculation of the fluxes and gradient contributions. This maintains the same amount of data communicated through the interconnect while almost splitting the computational effort into half.

Another advantage of the PETSc environment for ghosting values that reside in the memory space of other processes is the unified approach of data coherence, since on the top level only one vector object to store all data related to a variable, e.g. the velocity u_1 has to be created. Through precise information provided in the creation step of these vector objects PETSc distributes the vector object to all involved processes and uses the provided information to realize ghosting through the internal vector scatter and gather routines. The parallel vector layout and the process of ghosting are schematically shown in figure 1 for a two dimensional domain consisting of two grid blocks.

The last type of data distribution refers to global reduce operations, in which a single processes gathers data from all other processes and after some modification redistributes the data. A common scenario for global reduce operations is the calculation of the mean pressure throughout the domain which has been introduced in section **??**.

The distribution of stationary data takes place throughout the preprocessing program within the developed framework before the CAFFA solver is launched. Each processor is assigned a binary file with the respective data. Since the solver is not able to handle adaptive grid refinement or dynamic load balancing this approach is straightforward.

The present solver framework treats the calculation of matrix coefficients for block boundary control volumes in a special way. To reduce communication overhead and redundant data only one of the two involved processors calculates this coefficients and then sends them to the respective neighbour. The data needed to calculate the matrix coefficients of a neighbouring processor embraces not only the values of the dominant variables from the last outer iteration but also the values of the cell center gradients. For the ghosting of variable values PETSc offers special vector types, that comprise a user friendly interface to use the gather and scatter routines provided by PETSc. During the creation of these vectors a set of global indices referring to the values that are to be ghosted has to be provided. After this, the interchange of values is realized through a single subroutine call, since PETSc handles the concrete communication process necessary to accomplish the value interchange. The local representation of a ghosted vector hence not only comprises the data local to one processor but provides space for ghosted values that are repeatedly updated. The layout for one ghosted vector is shown for a simple two-dimensional solution domain that consists of two grid blocks.

FIGURE