

# FEEL: A Problem-Solving Environment for Finite Element Analysis

By Hidehiro FUJIO\* and Shun DOI\*

**ABSTRACT** We have developed a PSE (Problem Solving Environment) named FEEL (Finite Element Equation Language) for finite element analysis. One feature of FEEL is its ability to generate a program from a user-defined formulation of a finite element calculation scheme. Another feature of FEEL is its function to generate a program suitable for HPC (High Performance Computing) hardware platforms, especially distributed memory parallel computers. FEEL will generate a non-overlapping Domain Decomposition parallel program that will make use of parallel mathematical libraries.

**KEYWORDS** Problem solving environment, Parallel computing, Finite element method

## 1. INTRODUCTION

General-purpose software tools that solve various partial differential equations are called PSE (Problem Solving Environment). ELLPACK [1] is one of the earliest PSEs. It is an extension of Fortran and designed to solve elliptic partial differential equations with the finite difference method. Modulef [2] is a collection of finite element program modules. There were other PSEs developed in the 1980s such as DEQSOL [3] and FIDISOL. Except for DEQSOL, these PSEs are primarily for academic and research oriented use.

In the past decade, computer performance has been improved partly due to the parallelization of computer architecture. Many different computer architectures have been developed to increase computer speed. This progress requires the user's knowledge of computer architecture and programming techniques. An objective of PSE is to free users from having to learn the underlying computer architectures and programming techniques, and to let them concentrate their attention on productive work such as the development of physical and mathematical models of PDEs (Partial Differential Equations). The efficient use of HPC (High Performance Computing) systems, such as distributed-memory parallel computers, is an important and practical objective in recent PSEs, e.g., Diffpack [4], //Ellpack [5], PETSc [6], and UG [7]. It is also important to cope with distributed network environments including the Internet. An example may be the //Ellpack WWW system. The authors have been developing FEEL (Finite Element Equation Language) as a PSE for

finite element analysis [8,9]. The main goal is to provide a high-level programming interface for researchers and engineers. The FEEL engine will generate a finite element program from a user-defined formulation of a finite element calculation scheme. A currently available system is a 2D research prototype. This prototype is also able to generate an MPI (Message Passing Interface)-based parallel program, which employs the non-overlapping domain decomposition technique. We will first introduce an outline of the FEEL system configuration, language specification, and its parallel program generation function. The latter part of this paper will be devoted to a discussion dealing with how FEEL could be used in conjunction with other PSE tools, including various parallel mathematical libraries.

## 2. SYSTEM OVERVIEW

### 2.1 Objectives

One of the main objectives of FEEL is to provide users with a high-level programming interface for numerical analysis by the finite element method. A user may desire to apply a state-of-the-art numerical algorithm to their problem, and confirm whether the method works for it. A user may also want to develop a new method, such as a new type of element, a new linearization scheme, etc., and to check to see if the method works for a certain problem. Another important objective is to provide a means to easily access advanced HPC resources. In particular, the functionality to generate an MPI-based portable program for a distributed memory parallel computer is important.

### 2.2 System Configuration

Figure 1 shows the configuration of the FEEL

---

\*C&C Media Research Laboratories

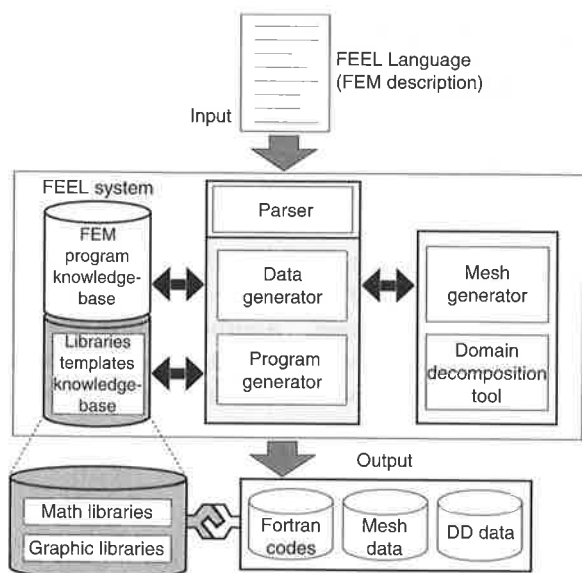


Fig. 1 FEEL system configuration.

system. The system is composed of a FEEL language processor, a mesh generator, a domain decomposition tool, several libraries, and templates. Libraries include shape functions, quadratures, linear solvers, run-time graphics, and communication libraries. Each template corresponds to a different type of computer architecture. At this moment, templates for a sequential computer and for a distributed memory parallel computer are available. Input into the system is a user's description of the governing equation and a computational scheme. Output includes a Fortran program and mesh data. If a parallel computer is specified, a parallel program which uses the non-overlapping domain decomposition parallel solver will be generated. Since the parallel program generated is written in Fortran with MPI, the program will retain high portability. In the generation of mesh data, domain decomposition control data will be also generated if the target machine is a parallel machine.

## 2.3 Outline of Language Specification

FEEL language is composed of a number of functional blocks [8]. This section introduces an outline of FEEL language. Figure 2 shows a typical FEEL program. It is a program for the solution of 2D incompressible Navier-Stokes equations by the penalty method [10].

### (1) Domain Definition

In the current version, the domain is defined in a hierarchical way by specifying *points*, *edges*, and

*regions*. The overall domain, specified by the *domain* statement, is defined as a set of regions. The *nodes* statement is to specify an expected number of nodes, which is input into the mesh generator.

### (2) Element Definition

In the system, common elements, such as linear, and quadratic triangle elements (P1, P2), or bilinear, and 8-node serendipity rectangle elements (Q1, Q2), etc., are pre-defined as built-in functions. A user can also define a new element by using the *element* statement if its explicit expression is known. In Fig. 2, the P1-bubble element is defined as *P1b*.

### (3) Quadrature Definition

Some quadratures are also incorporated in the system. A user can also define a new quadrature by using the *quadrature* statement.

### (4) fem and ewise Variables Definition

In addition to Fortran compatible integer and real variables, a user can define *fem* and *ewise* variables. For example, a definition "fem u[P1]" indicates that the variable *u* is an *fem* variable, and is based on the P1 basis function. Once a user defines a new element, such as *P1b* in Fig. 2, one can use the new element just by writing "fem u[P1b]". The *fem* variables are assumed to have some continuity between elements and used to approximate unknown functions, while the *ewise* variables are defined within an individual element and it may be discontinuous between elements. The *ewise* variables are mainly used for physical parameters.

### (5) Scheme Definition

For scheme description, several commands are prepared such as *goto*, *label*, *if*, and so on. In a scheme block, the user's scheme is described by using one or more solve statements. Output commands, such as *fwrite*, *plot-file*, *contour*, etc., can be used in the scheme block to save calculation results on files or to draw graphical pictures on an X Window\* display.

### (6) Solve Statement

In a *solve* statement, the partial differential equations in their weak forms, starting with the label '*weq*,' are specified. Several '*weq*' specifications in a single *solve* statement are simultaneously

\*X Window is a trademark of X Consortium, Inc.

```

                                Navier-Stokes equation (Cavity flow)

/* Region Definitions (Cavity) */
mesh {
  point a(0,0),b(1,0),c(1,1),d(0,1);
  edge  bottom(d,a,b,c),upper(c,d);
  refine c(0.5),d(0.5);
  region reg[tri](a,b,c,d);
  domain doml(reg);
  nodes doml(3800);
}

/* Element Definition (P1-bubble element) */
element Plb[tri] {
  (0,0):1-r-s;
  (1,0):r;
  (0,1):s;
  (1/3,1/3):(1-r-s)*r*s;
}

/* Variable Definitions */
var {double re,e,e2,ee;
  fem u[Plb],v[Plb]; /* velocity */
  fem p[P1],p0[P1]; /* pressure */
  fem w[Plb],s[Plb]; /* vorticity, stream func */
}

/* Scheme Definitions (Iterative Solution for Navier-Stokes Eq.) */
scheme {
  re =1.0/400.0 ; /* reciprocal of R-Number */
  e = 0.0; e2 = 0.0001; p0 = 0;
  u0 = 0; v0 = 0; ee = e - e2;
  iter: /* label for iteration */
  solve[u,v,p]{
    linear method: skyline; quadrature method: tri4;
    set-eps 1.0E-9;
    initial u = u0; initial v = v0; initial p = p0;
    nonlinear;
    weq:re*dx(u)*dx($)+re*dy(u)*dy($)+u*dx(u)*$+v*dy(u)*$+dx(p)*$;
    weq:re*dx(v)*dx($)+re*dy(v)*dy($)+u*dx(v)*$+v*dy(v)*$+dy(p)*$;
    weq:-dx(u)*$ - dy(v)*$ - e2 * p * $ - ee * p0 * $;
    dbc: u = 1, on upper;
    dbc: u = 0, on bottom;
    dbc: v = 0, on bottom, upper;
  }
  /* Convergence judgment */
  pmax = MAXnorm(p0 - p); pgmax = MAXnorm(p);
  umax = MAXnorm(u - u0); ugmax = MAXnorm(u);
  vmax = MAXnorm(v - v0); vgmax = MAXnorm(v);
  u0 = u; v0 = v; p0 = p;
  if(pmax/pgmax > 1.0e-8) goto iter;
  if(pmax/pgmax > 1.0e-8) goto iter;
  if(pmax/pgmax > 1.0e-8) goto iter;

  /* Calculation of vorticity */
  solve[w]
    weq: w *$ - dx(v)*$ + dy(u)*$;
  }
  /* Calculation of stream function */
  solve[s]{
    weq: dx($)*dx(s) + dy($)*dy(s) + w * $,<g * $>;
    dbc: s = 0, on bottom;
    nbc: g = 1, on upper;
  }
  /* Graphical output */
  contour[s] (winsiz=600,mesh=off);
  showvec-file 'velocity'[u,v];
}

```

Fig. 2 Typical FEEL program (Navier-Stokes equations).

discretized to generate a single system of linear equations. This means that the mixed method is supported. The *nonlinear* statement is prepared for the generation of a program for the Newton-Raphson iteration scheme. A *solve* statement has several parameters, such as linear algebra library, quadrature method, coordinate transformation function (to specify sub-parametric or super parametric elements), and initial values for nonlinear problems. By using more than two *solve* statements, the user can define his/hers own segregated iteration scheme for multi-valuable nonlinear equations.

These statements enable the specification of various finite element formulations. For example, the reduced integration method is obtained with a *quadrature* statement and individual quadrature specifications in the *solve* block. FEEL can also deal with a stream-upwind formulation if it can be explicitly expressed in weak formulations. One interesting example of the element definition is the modified 8-node serendipity element developed by Kikuchi [11]. These formulations can be efficiently defined, and then used easily by a user.

### 3. PARALLEL PROGRAM GENERATION

#### 3.1 Non-Overlapping Domain Decomposition

FEEL is able to generate an MPI-based parallel program that uses the non-overlapping domain decomposition method. In the domain decomposition method, the set of elements is divided into disjoint subsets, where the number of subsets is equal to the number of processors, and each subset is assigned to a corresponding processor. The assignments of nodes to processors are based on the element assignment. That is, a node will be assigned to processor  $i$  if the constituent element is assigned to processor  $i$ . It is noted that a node may be owned by more than two processors if the elements (which include the

node) are on an artificial inner boundary, and are assigned to more than two processors. The coefficient matrix for the "global" linear system is divided into "local" matrices. Individual "local" matrices are assigned to parallel processors based on the nodal assignment.

#### 3.2 Parallel Execution

Based on the above assignment, element-by-element calculations, such as the element matrix calculation and the assembly of a "local" matrix, are performed independently of others. A generated parallel program makes a call to a parallel linear system solver. In default, a preconditioned iterative method is prepared.

#### 3.3 Parallel Performance

Performance evaluation was carried out on the distributed memory parallel computer Cenju-3.

In the following, the speedup  $S_n$  is defined by  $S_n = (\text{Computational time by a single PE}) / (\text{Computational time by } n \text{ PEs})$ .

The communication ratio  $R_{\text{comm}}$  is defined by  $R_{\text{comm}} = (\text{Communication time}) / (\text{Time for computation and communication})$ .

Here, it is assumed that communications among processors are performed synchronously. In fact, this assumption can be proved in the execution of the generated programs. The plane stress problem is used to evaluate the performance of the generated program. The bilinear rectangular (Q4) element is used. Two sample problems are used. The small sample includes 1,785 elements and 5,494 nodes, while the large sample includes 3,656 elements and 11,161 nodes. For the parallel solution of linear systems, the Conjugate Gradient method with diagonal scaling is used. The iterations were terminated when the relative  $L^2$  norm reached  $10^{-12}$ .

Table I shows the results of the experiments,

Table I Computational time (in sec) and speedup ratio.

Example	Processor elements	1	2	4	8	16	32	64
Small	Time	151.7	72.0	39.8	15.5	9.9	6.2	4.5
	$S_n$	1.0	2.1	3.8	9.8	15.3	24.6	31.9
	$R_{\text{comm}}$ (%)	0.0	1.3	3.9	11.6	22.3	42.4	63.1
Large	Time	440.3	219.8	108.3	56.3	22.2	14.4	9.0
	$S_n$	1.0	2.0	4.1	7.8	19.8	30.6	48.8
	$R_{\text{comm}}$ (%)	0.0	0.6	2.3	4.8	14.5	25.6	46.9

including computational time, speedup ratio, and communication overhead versus processors. A reasonable speedup is observed in the large sample problem (11,161 nodes). In fact, the speedup reaches 49 with 64 processors. If the number of processors is fixed, a large problem always indicates (except for the case of 8 PEs) higher speedup than a small problem. This is because the calculation/communication ratio becomes higher as the size of the problem becomes larger (when the number of PEs is fixed). Therefore we can expect higher speedup if the problem is larger. Super-linear speedup has been observed when the numbers of PEs are 8 and 16. This well-known phenomenon results from the less cache miss-hit due to the smaller size of the problem (compared with a single processor execution). When 64 PEs are used, the communication overhead reaches more than 60% for the smaller problem and 40% for the larger problem. This illustrates the importance of the reduction of communication overhead.

#### 4. DISCUSSION

Developing practical PSE is still challenging. PSE needs to cover various fields. They are physical and mathematical modeling, geometrical modeling, discretization, parallel computation, visualization, and so on. Several software tools are required to work cooperatively with others to complete such a problem-solving task. A single tool can hardly cover all PSE functionality. It is important to understand the positioning or the role of each PSE components when using it. This section will discuss how FEEL could cooperate with other tools, and what is needed to interface with them. A particularly important factor is the efficient use of parallel and distributed platforms.

##### 4.1 Functional Hierarchy in PSE

Figure 3 gives a hierarchical view of PSE functions. The low-level layer shows a function close to the hardware, such as the MPI and the parallel mathematical library. A higher-level layer may correspond to PDE problem specification. The higher the level, the more abstract the input into the layer becomes. The right hand side blocks in Fig. 3 show the names of software tools. FEEL corresponds to the middle layer in this PSE hierarchy map. Input into FEEL is the schematic formulation for finite element analysis expressed by FEEL language. The program generated by FEEL will utilize lower level libraries without burden on users. The essence of

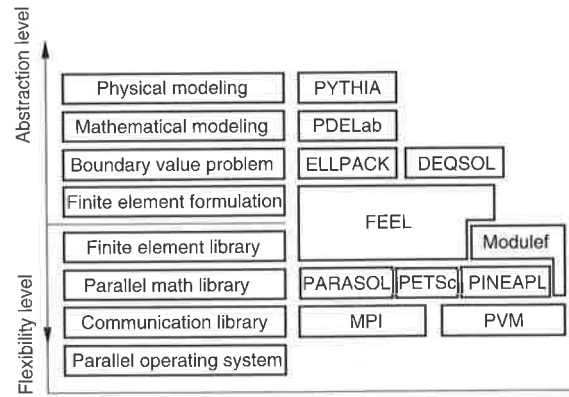


Fig. 3 PSE components.

FEEL is a programmed knowledge base of various coding techniques of the FEM which connects the library layer and the more abstract layer. A software tool in a higher-level layer could generate inputs for FEEL system, in this case, users could specify their problem in more abstract manner.

##### 4.2 Interface to a Parallel Mathematical Library

The obviously important low-level layer is the parallel mathematical library that is used for the solution of sparse linear systems and eigenvalue problems. Actually, many research projects are being carried out in order to develop message passing parallel libraries, e.g., PETSc, PARASOL [12], and PINEAPL [13], etc. The objective of these libraries is to extract a high performance from underlying hardware platforms. More attention focuses on the performance, and it tends to sacrifice usability. This causes the calling sequence and data structures in these libraries to be complicated and difficult to use. An objective for program generation systems like FEEL is to hide such low-level library interfaces from users. Actually, FEEL generates a parallel program which makes a call to a native parallel library for the solution of linear systems, and the user does not need to take care of such a low-level interface. The next version of FEEL will be able to generate a library interface for the other standard libraries named above.

##### 4.3 PSE in the WWW Environment

Another influential factor on PSE is the Internet. Cooperative work over the Internet could increase productivity and promote knowledge sharing with team members. Developments for this direction can be seen in the //ELLPACK WWW System [14] and

the FEELWEB (FEEL on the WWW Environment). FEELWEB enables users to access the FEEL server by using PC browser as a GUI terminal via the Internet. FEELWEB is now available at the NEC Parallel Processing Center [15].

## 5. CONCLUSION

The authors have been developing FEEL, a PSE for finite element analysis. A feature of FEEL is its ability to generate a program from a user's description of a finite element calculation scheme. Another feature of FEEL is its function to generate a program suitable for HPC hardware platforms, especially for distributed memory parallel computers. FEEL is currently able to generate a non-overlapping domain decomposition parallel program which calls to a parallel mathematical library, thus allowing easy access to other parallel mathematical libraries. Another important factor is efficient use of the Internet. We have been developing FEELWEB in this context, and it is accessible at the NEC Parallel Processing Center. The current version of FEEL is a research prototype. Further enhancements are needed to promote FEEL as a commercially valuable system. We are planning to enhance its language specification and language processor so that, ideally, it could deal with any finite element formulations. Such enhancements may include the space-time integration method, discontinuous Galerkin method, and vector elements, etc., but is not limited to them. It is also important to increase the ability to interface easily with low-level libraries. For this objective, we are designing a library interface macro with which an interface for a new library could be easily added to FEEL by end-users. Efficient use of the Internet is still a key factor. In this context, a prototype version of FEELWEB is now available in the

Internet.

## REFERENCES

- [1] E. Gallopoulos, et al., *Future Research Directions in Problem Solving Environments for Computational Science*, CSRD Report No. 1259, 1991 ([http://www.csr.d.uiuc.edu/tech\\_reports.html](http://www.csr.d.uiuc.edu/tech_reports.html)).
- [2] *Modulef*, <http://www-rocq.inria.fr/modulef/modulef.html>.
- [3] Y. Umetani, et al., "Visual PDEQSOL: A Visual and Interactive Environment for Numerical Simulation," *Programming Environment for High-Level Scientific Problem Solving*, P. Gaffney and E. N. Houstis ed., North-Holland, pp. 259-267, 1992.
- [4] *Diffpack*, <http://www.nobjects.com/prodserve/diffpack>.
- [5] E. N. Houstis and J. Rice, "Parallel ELLPACK: a Development and Problem Solving Environment for High Performance Computing Machines," *Programming Environments for High-Level Scientific Problem Solving*, T. Gaffney and E. N. Houstis ed., North-Holland, pp. 229-241, 1992.
- [6] *PETSc*, <http://www.mcs.anl.gov/Projects/petsc/petsc.html>.
- [7] *UG*, <http://www.ica3.uni-stuttgart.de/~ug/index.html>.
- [8] H. Fujio and K. Hayami, "FEEL — A Tool for Finite Element Analyses," *Proc. Numer. Anal. Symp. Structural Anal.*, 17, pp. 489-497, 1993 (in Japanese).
- [9] S. Doi, et al., "Automatic Parallel Program Generation for Finite Element Analysis," *Quality of Numerical Software*, R. F. Boisvert ed., IFIP, pp. 255-266, 1997.
- [10] F. Kikuchi, "Application of an Iteration Scheme to the Analysis of Incompressible or Nearly Incompressible Media," *Patterns and Waves-Qualitative Analysis of Nonlinear Differential Equations*, pp. 445-458, 1986.
- [11] F. Kikuchi, "Explicit Expressions of Shape Functions for the Modified 8-Node Serendipity Element," *Com. Num. Meth. Eng.*, 10, pp. 711-716, 1994.
- [12] *PARASOL*, <http://www.genias.de/parasol/>.
- [13] *PINEAPL*, <http://www.nag.co.uk/projects/PINEAPL.html>.
- [14] *Ellpack*, <http://www.cs.purdue.edu/research/cse/ellpack/>.
- [15] *FEELWEB*, <http://www.ppc.nec.co.jp/feelweb/>.
- [16] J. Rice and R. Boisvert, "From Scientific Software Libraries to Problem-Solving Environments," *IEEE CS&E*, 3, 3, pp. 44-53, 1996.

Received July 1, 1998

\* \* \* \* \*



Hidehiro FUJIO received his B.E. degree in mathematics from the University of Tokyo in 1990. He joined NEC Corporation in 1990, and is now a member of the Computational Engineering Technology Group, C&C Media Research Laboratories. He is engaged in the development of a numerical simulation language system for Finite Element Analysis.



Shun DOI received his B.E., M.E. and Ph.D. degrees in engineering from Hokkaido University in 1979, 1981 and 1984, respectively. He joined NEC Corporation in 1984, and is now Senior Manager of the Computational Engineering Technology Group, C&C Media Research Laboratories. He was a visiting scholar at INRIA (Institut National de Recherche en Informatique et en Automatique, France) from 1988 to 1989. He has been engaged in the research and development of sparse linear solvers, scientific visualization, and problem solving environments for large-scale numerical simulations.

\* \* \* \* \*