A Model for Parallel Adaptive Finite Element Software

Krzysztof Banaś

Cracow University of Technology, Section of Applied Mathematics ICM

Summary. The paper presents a conceptual model and details of an implementation for parallel adaptive finite element systems, particularly their computational kernels. The whole methodology is based on domain decomposition while message passing is used as a model of programming. The proposed finite element architecture consist of independent modules, most of them taken from sequential codes. The sequential modules are only slightly modified for parallel execution and three new modules, explicitly aimed at handling parallelism, are added. The most important new module is the domain decomposition manager that performs most tasks related to parallel execution. An example implementation that utilizes 3D prismatic meshes and discontinuous Galerkin approximation is presented. Two numerical examples, the first in which Laplace's equation is approximated using GMRES with multi-grid preconditioning and the second where dynamic adaptivity with load balancing is utilized for simulating linear convection, illustrate capabilities of the approach.

1 Introduction

Parallelization of adaptive finite element systems is a complex and complicated task. There are only few systems (Bastian et al. [1997], Beall and Shephard [1999], Bangerth and Kanschat [1999]) combining adaptivity and parallelism within a comprehensive finite element environment. Comprehensive is understood as offering such capabilities as 2D and 3D meshes of various types, continuous, discontinuous and higher order approximations, multi-level iterative solvers for linear systems, handling of coupled, possibly multi-physics, non-linear problems. On the other hand there is a growing interest in using principles of software engineering and object orientedness for design of scientific codes (Bruaset and Langtangen [1997], Beck et al. [1997]). The goal of the research is to combine flexibility and maintainability of object oriented codes with efficiency of monolithic Fortran or C programs.

The stress in the present paper is put on the modular structure of codes, design of modules' interfaces and fundamental principles for parallelizing adaptive finite element codes. The starting point is a modular finite element frame-

work for sequential computations, designed for extendibility, reusability and efficiency. The goal of the parallelization process is to preserve the modular structure of the framework and add efficient mechanisms for parallel execution. Additionally parallelization is designed to change sequential modules as little as possible and to be conceptually simple. The aim of these last features is to offer an easy way for parallelizing existing legacy finite element programs.

2 A model architecture

It is assumed that the whole code is built of independent modules having their own data structures and communicating through interfaces accessible using the main programming languages of scientific computing (Fortran90, C, C++) (Banaś [2002]). There are four fundamental sequential modules:

- mesh manipulation module - that provides the other modules with all topological data concerning elements and other entities (faces, edges, vertices) present in the mesh; mesh manipulation module performs refinements/derefinements of individual mesh entities
- approximation module - that performs all tasks related to approximation fields defined for finite element meshes; main tasks include numerical integration, finite element interpolation and different kinds of projections
- linear solver module - the module may form a solver itself or serve as an interface for some external solver
- problem dependent module - the rest of the code that includes, among others, submodules specifying the solved PDE problem and driving the process of adaptation (the latter involves error estimation)

The mesh manipulation module is the simplest to design. It possess its own data structure and is organized as a set of services, a library of functions returning data concerning mesh entities. It does not use data from other finite element modules, although it can, and sometimes should, interact with external modules for mesh generation and geometry modeling.

The approximation module handles all tasks related to approximation fields that are defined in terms of finite element shape functions. Since definitions of shape functions are specific to different types of elements, approximation modules are strongly related to particular mesh manipulation modules. Besides the dependence on a specific mesh, the approximation module is also strongly coupled with the problem dependent module. The problem of efficient realization of numerical integration in flexible, multi-purpose codes is solved using an interface with few well defined call-backs (Banaś [2002]).

The mechanism of call-backs is also utilized in the design of the interface between the problem dependent module and the linear solver module. It is assumed that the problem dependent module calls the linear solver to perform basic steps of the solution procedure, but it is the linear solver that gathers all data (on mesh entities, approximation fields and particular entries to the

system matrix and the load vector) necessary to perform multi-level solution of linear equations.

2.1 Parallel execution modules

The proposed model of parallelization is based on domain decomposition as an algorithmic foundation and message passing as a programming technique. One of reasons for such a choice is the possibility of reusing much of sequential modules for parallel codes.

It is assumed that sequential modules are included into parallel codes without substantial modifications. There are three new modules added to handle parallel execution. The two first are simple interface modules. The first, named parallel execution interface, gathers the main calls related to parallel execution within the problem dependent module. These calls are then passed to the main parallel module, the domain decomposition manager, or left with no effect in case of sequential runs.

The second simple module connects the finite element program to a parallel execution environment. It consist of a set of generic send/receive and group operations, that have to be implemented for various communication libraries.

The domain decomposition manager performs the following tasks:

- interfacing an external mesh partitioner (and, possibly different, repartitioner)
- distributing the mesh among processors
- creating necessary overlap and managing all requests related to overlap entities
- implementing domain decomposition algorithm
- adapting mesh in parallel
- load balancing and data transfer

The domain decomposition manager is composed of several submodules, responsible for parallel execution of different tasks. In such a way it is possible to parallelize only parts of the code (e.g. linear solver) while the rest remains sequential.

3 Implementation

Based on the proposed architecture (see Fig. 1) a prototype implementation has been created that uses 3D prismatic meshes and discontinuous Galerkin approximation.

The basis for implementation of the domain decomposition manager is formed by the assumption that every mesh entity and every set of approximation data present in the data structure is equipped with a global (interprocessor) identifier (IPID). This identifier can be understood as a substitute

for a global address space used in sequential codes and is composed of a processor (subdomain) number and a local (to a given processor) identifier. IPIDs are not known to sequential modules of the code and all situation where the access to non-local data is necessary are handled by the domain decomposition manager.

4 Numerical examples

Two numerical examples showing capabilities of the described approach and the prototype implementation are presented in this section. The example problems are very simple from mathematical point of view. However, they show the effect of practical realization of two important and technically difficult, from implementation point of view, phases of simulation: parallel multilevel solution of linear equations and parallel adaptivity combined with transfer of mesh entities to maintain load balance.

The computational environment for both examples consist of a set of Linux workstations connected using a standard 100 Mbit Ethernet network. The results in tables have been obtained using computers equipped with 1.6 GHz Pentium IV processor and 1 GByte memory.

4.1 Simulating diffusion

The first example is Laplace's equation

$$\Delta u = \Delta u_{ex}$$

where u_{ex} is the known exact solution:

$$u_{ex} = \exp\left(-x^2 - y^2 - z^2\right)$$

The computational domain consist of the box $[0, 0.1] \times [0, 1] \times [0, 10]$ and boundary conditions are chosen to match the exact solution. Discontinuous Galerkin approximation (Oden et al. [1998]) and the preconditioned GMRES method are used for solving the problem.

Table 1 presents results for a series of computations corresponding to the described problem. Two preconditioners are employed, both use the combination of additive Schwarz preconditioning for the whole problem and multiplicative Schwarz within subdomains. The first is single level preconditioner and the second uses three consecutive mesh levels to achieve multigrid preconditioning. For each preconditioner problems of different sizes, corresponding to subsequently uniformly refined meshes, are considered. For each combination preconditioner/problem size results of computations using 1, 2, 4 and 8 workstations are shown. For the largest problem the reference number of processors to compute speed up and efficiency is two, since the problem did not

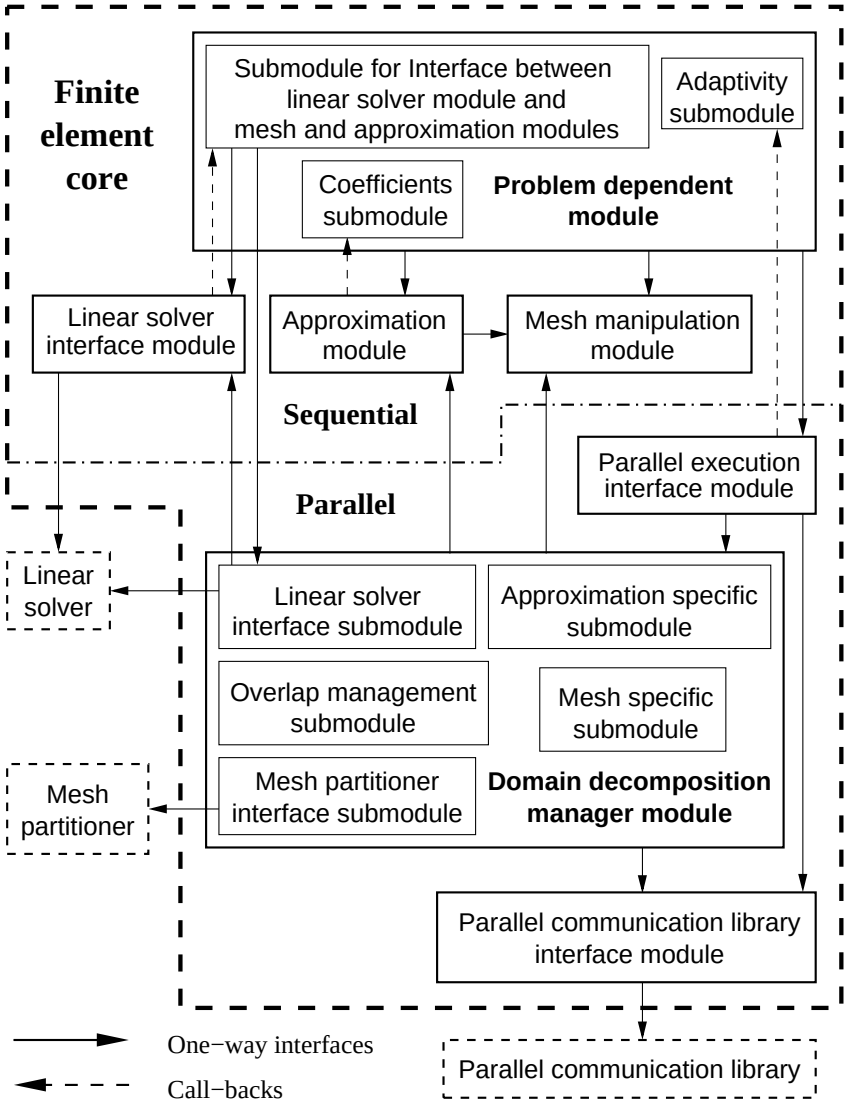

Fig. 1. Diagram of the proposed modular architecture for computational kernels of parallel adaptive finite element codes

fit into a memory of a single computer. For the smallest problem the number of mesh levels was equal two and the only possible preconditioner was single level.

Results are reported for 10 iterations of the preconditioned GMRES method to focus on the efficiency of parallel implementation, not considering

the influence of parallelization on the convergence of GMRES (nevertheless the latter is reported for completeness). Subsequent meshes are obtained by uniform refinements and for each mesh N_{DOF} is the number of degrees of freedom. N_{proc} is the number of workstations solving the problem. *Error* is the norm of residual after 10 GMRES iterations (within a single restart) and *Rate* is the total GMRES convergence rate during solution. Execution time $Time$ is a wall clock time. Speed-up and efficiency are computed in the standard way.

Table 1. Results for 10 iterations of the preconditioned GMRES method and discontinuous Galerkin approximation used for solving Laplace's equation in a box domain (description in the text).

Single level preconditioner						
N_{DOF}	N_{proc}	Error*10^9	Conv. rate	Exec. time	Speed up	Efficiency
48 896	1	0.288	0.443	2.26	1.00	100%
	2	0.328	0.448	1.16	1.94	97%
	4	0.340	0.450	0.61	3.70	92%
	8	0.361	0.452	0.36	6.28	78%
391 168	1	9.313	0.626	17.85	1.00	100%
	2	10.173	0.632	8.93	1.99	100%
	4	10.252	0.633	4.53	3.94	98%
	8	11.183	0.638	2.34	7.63	95%
3 129 344	2	48.041	0.738	70.76	1.00	100%
	4	47.950	0.738	35.63	1.98	99%
	8	48.748	0.739	17.71	3.99	100%
Three level preconditioner						
N_{DOF}	N_{proc}	Error*10^9	Conv. rate	Exec. time	Speed up	Efficiency
391 168	1	0.018	0.335	26.18	1.00	100%
	2	0.017	0.334	14.18	1.85	92%
	4	0.018	0.335	9.08	2.88	72%
	8	0.024	0.346	7.60	3.44	43%
3 129 344	2	0.027	0.350	111.16	1.00	100%
	4	0.027	0.350	57.76	1.92	96%
	8	0.027	0.348	33.15	3.35	84%

4.2 Simulating convection

The second example is a simple convection problem in the box $[0, 38] \times [0..1000] \times [0..18]$. A rectangular pattern is traveling from left to right (along

the y-axis). GMRES with single level Schwarz preconditioning is used, once again with discontinuous Galerkin approximation. The only interesting process for this example, that will be described in more detail, are the subsequent parallel mesh adaptations and load balancing achieved through transfer of mesh entities. There are four workstations used for simulation and the computational domain is divided into four subdomains. Subdomains have two element overlap to enable mesh adaptations and overlapping Schwarz preconditioning. After each time step (in the example run there were 120 time steps) the mesh is adapted in parallel.

After each mesh adaptation, the number of degrees of freedom in each subdomain is checked against the average (it is assumed that processors are of the same speed). If imbalance larger than 10% is encountered, mesh repartitioner is called, to provide new domain decomposition. According to the new assignment of elements to processors and two element overlap requirements, mesh entities are marked respectively, and the transfer between subdomains takes place. To enable clustering, mesh transfers consider always whole element families - initial elements that are marked for a transfer and all their antecedents.

Table 2 presents characteristics of mesh transfers for five subsequent time steps, from 100 to 104. The average number of DOFs in a subdomain remains constant since the same number of elements appears due to refinements and disappears due to derefinements. Since refinements and derefinements takes place in different regions the difference between the subdomain with the greatest number of DOFs and the subdomain with the smallest number of DOFs grows after each time step. The numbers of mesh entities reported in the table concern the total number of entities effectively transferred between all the subdomains. The numbers do not include entities for which IPIDs only are exchanged.

For the whole simulation, the speed up obtained using 4 processors was equal to 2.67, giving the efficiency of 67%. For the overhead that includes mesh repartitioning, mesh transfers and the fact that, according to the overall strategy, the load for processors is not perfectly balanced, the results appear to be reasonable.

5 Conclusions

The new architecture proposed for parallel adaptive finite element codes fulfills the requirement of combining execution efficiency and code modularity. Further improvements of the prototype implementation concerning efficiency and the creation of new specialized modules that would increase code's flexibility are under way.

Acknowledgement. This work has been supported by the Polish State Committee for Scientific Research under grant 7 T11F 014 20

Table 2. Characteristics of mesh transfers during parallel simulation for the convection problem.

	Time step number				
	100	101	102	103	104
Average number of DOFs	5086	5086	5086	5086	5086
Maximal number of DOFs	5636	5120	5372	5596	5120
Minimal number of DOFs	4468	5012	4732	4508	4996
Number of transferred vertices	300	0	0	390	0
Number of transferred edges	1212	0	0	1671	0
Number of transferred faces	1284	0	0	1863	0
Number of transferred elements	438	0	0	657	0

References

K. Banaś. Finite element kernel modules for parallel adaptive codes. Report 4/2002, Section of Applied Mathematics ICM, Cracow University of Technology, Warszawska 24, 31-155 Kraków, Poland, 2002. *submitted to Computing and Visualization in Science.*

W. Bangerth and G. Kanschat. Concepts for object-oriented finite element software - the deal.II library. Preprint SFB 359, Universitat Heidelberg, 1999.

P. Bastian, K. Birken, K. Johannsen, S. Lang, N. Neuss, H. Rentz-Reichert, and C. Wieners. UG - a flexible software toolbox for solving partial differential equations. *Computing and Visualization in Science*, 1(1):27–40, 1997.

M. Beall and M. Shephard. An object-oriented framework for reliable numerical simulations. *Engineering with Computers*, 15:61–72, 1999.

R. Beck, B. Erdman, and R. Roitzsch. An object-oriented adaptive finite element code: design issues and application in hyperthermia treatment planning. In E. Arge, A. Bruaset, and e. H.P Langtangen, editors, *Modern software tools for scientific computing*, pages 105–123. Birkhauser Press, 1997.

A. Bruaset and H. Langtangen. A comprehensive set of tools for solving partial differential equations - Diffpack. In M. Daehlen and A. Tveito, editors, *Numerical Methods and Software Tools in Industrial Mathematics*, pages 63–92. Birkhauser, 1997.

J. Oden, I. Babuska, and C. Baumann. A discontinous *hp* finite element method for diffusion problems. *Journal of Computational Physics*, 146: 491–519, 1998.