Agent Architecture for Mesh Based Simulation Systems

K. Banaś

Cracow University of Technology
Warszawska 24, 31-155 Kraków
kbanas@pk.edu.pl

Abstract. The paper presents an analysis of requirements for building simulation systems with tightly coupled components, such as typical mesh based PDE approximation software. The considered systems are characterized as having high communication to computation ratio. When designing architectures for such systems the hardware and middleware capabilities for providing communication links between processes have to be investigated and fully exploited. This is the place where agent technology perfectly fits the requirements. In the whole system, the capabilities of agents should be complemented with less flexible but more efficient software organization.

As an example a framework for finite element simulations, employing a modular architecture (described in [1]), is considered. Communication requirements for typical computations are estimated and evaluated in view of possible inter-process communication. The role of agents in setting up the execution structure of simulations is described.

1 Agent Based Simulation Systems

Agent technology is often used to add flexibility to classical computational systems that are executed on distributed hardware resources [2, 3]. Agents can be used to query static information on hardware resources, as well as to monitor their workload. Based on the data the optimal mapping of computations on processing elements together with proper communication patterns can be selected [4]. Autonomous agents can additionally gather necessary information and take decisions: concerning e.g. splitting computations, migrating or stopping. The current paper describes a setting where the above mentioned possibilities of agents are crucial for achieving the assumed goal. The setting consist of a classical computational science system, a mesh based PDE solver, for which a new flexible architecture, that can take advantage of modern hardware resources is sought.

2 Mesh Based Computations with Tightly Coupled Components

Mesh based computations, using the finite difference, finite element and finite volume methods, are common in scientific computing. Such computations contain, as an important ingredient, matrix operations, and therefore the focus of

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part III, LNCS 3993, pp. 743–750, 2006.

research on such systems, in view of achieving high performance, is often put on developing linear algebra packages. However, the overall efficiency of such codes may depend on the architecture of the whole system, including the interaction of algorithms and data structures that deal with mathematical entities appearing in the problem domain: discretized computational domains, discrete fields and systems of algebraic linear equations. It is possible to develop software architectures that use different modules to deal with the above mentioned entities and to combine the resulting flexibility in setting up the computational system with high performance of execution. Such architectures can be useful when complex data structures and algorithms are used for e.g. adaptive mesh refinement, projections of complex discrete fields or multigrid solvers and preconditioners.

The agent technology for modular mesh based solvers can be employed through linking each module (component) with an agent that serves as a control and steering unit for the module. Such an agent has to know the specifications of hardware resources as well as the detailed characteristics of the modules it controls, including the required communication with other modules. This knowledge is used to properly configure a computational system, exploiting the capabilities of hardware and software components.

Next sections describe an example architecture for a mesh based solver that uses the adaptive finite element method. From the three, mentioned previously, numerical methods used in mesh based simulations, the finite element method is theoretically the most complex: it can use unstructured adaptive meshes and higher order approximation. It is also the one, that usually suffers from the most severe performance penalty due to the need to deal with the above mentioned complexities. Therefore it is important to design architectures for FEM systems that can take advantage of different, flexible hardware environments.

3 Modular Architecture for Sequential Finite Element Systems

Classical architectures for finite element codes usually distinguish between the two levels of coupling among their building components. At the loosely coupled level there are modules for three phases of simulations: pre-processing (creating the description of a computational domain, generating a mesh for the domain), processing (actual calculations) and post-processing (calculation of derived quantities, visualization). At that level the requirements for communication between modules are low and the modules are often organized into different programs exchanging data by means of files that are produced by one module and then subsequently read by another.

At the tightly coupled level there are submodules that handle specific data structures and computations within the simulation phases. In the sequel, only the phase of actual calculations will be considered in detail, as usually the most time consuming. For the component that realizes this phase, the finite element computational kernel, a modular architecture has been proposed [5], that aims at creating software more flexible and easier to modify and extend. This architecture,

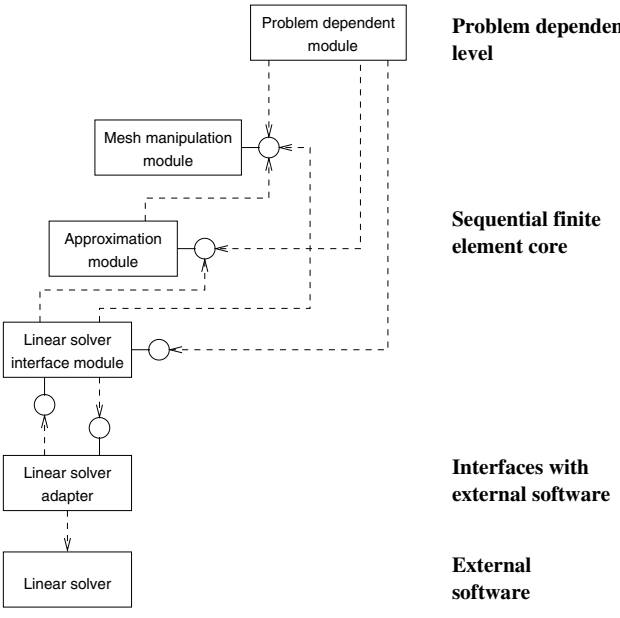

Fig. 1. The proposed architecture for sequential finite element codes

illustrated by a UML diagram in Fig. 1, consist of several modules, that are distinguished based on the data structures they operate on. The reason for this is that classical structural analysis of FEM computations, that finds modules by looking at the flow of computations, is no longer the only possible and even not the most appropriate, as the computations become more and more interactive. Hence there are the following modules and data structures in the proposed architecture:

- problem dependent module with algorithms and data structures particular to a problem solved
- mesh manipulation module with a data structure that holds all topological and geometric data on a mesh; algorithms in the module perform e.g. mesh modifications
- approximation module; the module responsible for storing discrete data on physical fields and for realizing the necessary operations on the fields (like e.g. integration, differentiation, projection)
- interface with linear solvers module that translates information expressed in terms particular to the PDE approximation methods into information expressed in terms of numerical linear algebra entities like vectors and matrices; this module can be equipped with additional data structures, e.g. for reordering linear equations
- linear solver which is considered an external software in the proposed architecture

4 Domain Decomposition for FEM Core

For parallel execution the extension shown in Fig. 2 has been proposed [1], that uses a centralized management based on domain decomposition and message passing. This architecture enables achieving high performance of execution for static parallel hardware configurations. However, from the software engineering point view it goes against the design goals of the sequential version. The idea to create independent components for dealing with fundamental data structures of the code does no longer holds.

Therefore in the current paper a natural next step in development of modular design for computational kernels is presented. This next step consist in equipping each sequential module of the code with its own domain decomposition manager. The architecture is shown in Fig. 3. The parallel modules, consisting of pairs sequential module–domain decomposition manager, can now be developed and tested separately. Additional flexibility is brought by the possibility of using the same domain decomposition managers for different sequential modules of the same kind and exchanging in a particular code only one parallel module at a time.

The standard run-time structure of a parallel code built according to the architecture consist of a set of processes executed on different processing nodes of a computational system. The processes correspond to compiled FEM programs, each of which contains all modules from Fig. 3.

It is known that with the increasing number of subdomains (processes) the parallel efficiency of fundamental algorithms, such as computing vector norms

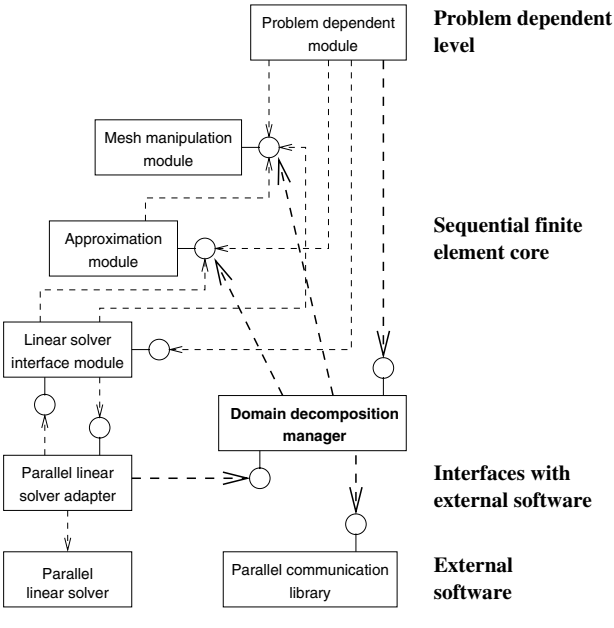

Fig. 2. The centralized architecture for parallel finite element codes

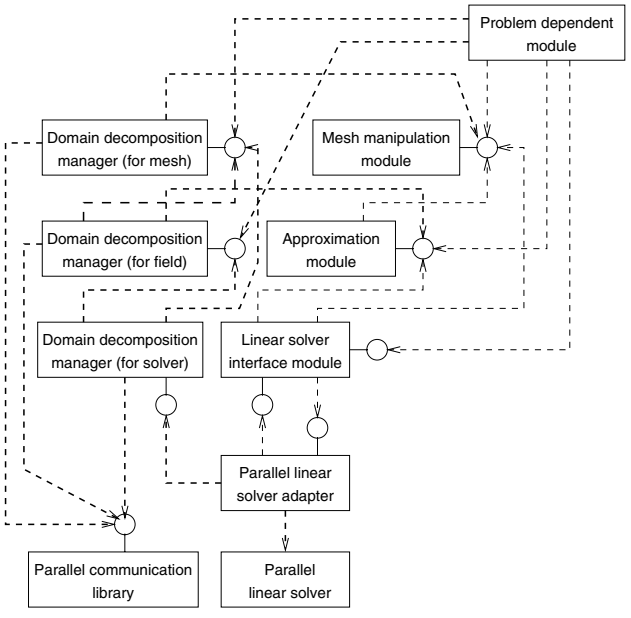

Fig. 3. The proposed modular architecture for parallel finite element codes

and scalar products or matrix-vector products, that form building blocks of mesh based simulation systems, diminishes. Moreover, the convergence of iterative solvers usually slows down. In order to maintain algorithmic complexity of sequential computations and to maximize utilization of computing systems comprising of more and more processors, different parallelization strategies, beyond simple domain decomposition and message passing, have to employed. One of such strategies is to combine message passing with multi-threading [6]. Still the processes correspond to programs consisting of all modules, the difference lying in the multi-threaded mode of execution, using compiler directives or OpenMP.

5 Functional Decomposition for FEM Core

Further increase in flexibility of setting up a run-time structure of codes can be obtained by investigating the possibility of using separate processes for different parallel modules of the code. In such a case each dependence between modules should be now understood as requiring communication: each inter-module function call involves sending a message. This creates two kinds of communication links for each module: horizontal – among modules of the same kind managing a distributed data structure and vertical – between different modules.

It is interesting to estimate what are the communication requirements for different modules in the above mentioned situation. The analysis below uses simple and typical settings, more detailed investigations can be found in [7].

The main steps of the typical FEM solution process comprise numerical integration of terms from the finite element formulation of the problem, assembling of the linear system matrix and solving the system of linear equations. In the parallel execution model with domain decomposition and message passing the first two phases can be performed perfectly or almost perfectly in parallel. When typical iterative solvers are used for linear system solution phase, communication requirements are mainly related to matrix-vector products.

The communication complexity of performing matrix-vector products for FEM matrices depends on their non-zero structure. This structure reflects the whole finite element setting: the type of problem solved, the dimension of the physical space, the mesh employed, the kind of approximation. The most important ingredient (the other change the communication cost by constant factors) is domain decomposition that determines the ratio of the number of unknowns (degrees of freedom) inside domain to the number of unknowns on the boundary of domains. When performing matrix-vector products data corresponding to intersubdomain boundary has to be exchanged among processes. In the simple case of very regular 3D domains with perfect domain decomposition the number of exchanged unknowns per subdomain can be estimated as being in order of $(N/N_{\mathrm{S}})^{2/3}$, where N is the total number of unknowns and N_{S} is the number of subdomains. During the whole solution process the number of data exchanges is usually equal to the number of iterations, N_{it}, which practically is of order 10–1000.

Among the parallel modules exchanging data vertically the pair approximation module–linear solver interface module requires the least communication. The minimal estimate for this communication comprises sending all entries of the system matrix (this neglects, among others, sending an initial guess vector, a right hand side vector and the information on the structure of the matrix). The number of entries per subdomain, in the same case as considered for the communication analysis, is in the order of $(N/N_{\mathrm{S}})N_{\mathrm{n}z}$ where $N_{\mathrm{n}z}$ is the average number of non-zero entries in a single row of the system matrix. The last number for typical problems, meshes and approximations is of order 10–1000.

Hence the amount of vertical communication is, in typical configurations, at least one order of magnitude greater than the amount of horizontal communication (the ratio of vertical communication to horizontal communication in the considered particular case is equal at least to $(N/N_{\mathrm{S}})^{1/3}N_{\mathrm{n}z}/N_{\mathrm{it}}$). Taking this into account the question arises, whether the vertical splitting of modules can bring performance advantages? This may take place for hardware systems consisting of several SMP nodes or, the solution that will become more and more popular in the future, multi-core processors. Fast communication using shared memory can be used to execute modules in parallel, keeping hardware resources busy, and not increasing the number of subdomains.

6 Agent Based Architecture

For the described architecture with horizontal and vertical splitting of execution components the proper mapping of computations on the hardware become a

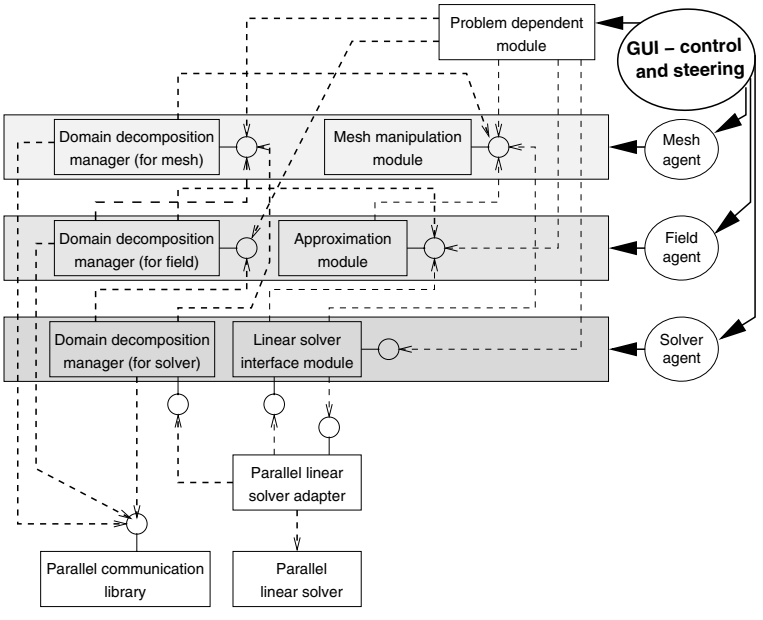

Fig. 4. The proposed agent based modular architecture for parallel finite element codes

complex issue. To resolve this issue an agent based architecture is introduced in Fig. 4. Each horizontal layer is equipped with an agent that is used to discover possible hardware-software setting for the simulation and to make decisions aiming at achieving the best performance. The information considered by agents includes data on simulation (problem, mesh, approximation, linear solver) as well as on computing system (the number of available computers/processors/cores, the speed of communication links – latency and bandwidth, current load etc.). A control-and-steering unit (possibly distributed) activates agents and provides them with user data.

7 Conclusion

The final agent architecture for computational kernels of mesh based solvers consists of several layers dealing with fundamental data structures of the code. The exchange of data and services between layers is done through fast communication links, preferably using shared memory. Each layer is composed of three components:

- the sequential part unaware of possible parallel execution
- domain decomposition manager enabling the module to communicate with other modules of the same kind using message passing and to work with them in parallel
- agent – intelligent component discovering resources and other modules, used for setting up the whole simulation environment

The proposed architecture brings additional flexibility to the process of forming hardware-software environments for performing computations. The flexibility comes with a price of much more complex management of different modules for parallel and distributed execution. The role of agents in the architecture is to provide intelligence necessary to unify all components and achieve high performance realization.

References

1. Banaś, K.: A modular design for parallel adaptive finite element computational kernels. In Bubak, M., van Albada, G., Sloot, P., Dongarra, J., eds.: Computational Science — ICCS 2004, 4th International Conference, Kraków, Poland, June 2004, Proceedings, Part II. Volume 3037 of Lecture Notes in Computer Science., Springer (2004) 155–162
2. Szymanski, B., Varela, C., Cummings, J., Napolitano, J.: Dynamically reconfigurable scientific computing on large-scale heterogeneous grids. In Wyrzykowski, R., Dongarra, J., Paprzycki, M., Wa'sniewski, J., eds.: Parallel Processing and Applied Mathematics, Proceedings of Vth International Conference, PPAM 2003, Częstochowa, Poland, 2003. Volume 3019 of Lecture Notes in Computer Science., Springer (2004) 419–430
3. Kisiel-Dorohinicki, M.: Agent-based models and platforms for parallel evolutionary algorithms. In Bubak, M., van Albada, G., Sloot, P., Dongarra, J., eds.: Computational Science — ICCS 2004, 4th International Conference, Kraków, Poland, June 2004, Proceedings, Part III. Volume 3037 of Lecture Notes in Computer Science., Springer (2004) 646–653
4. Grochowski, M., Schaefer, R., Uhruski, P.: Diffusion based scheduling in the agent-oriented computing system. In Wyrzykowski, R., Dongarra, J., Paprzycki, M., Wa'sniewski, J., eds.: Parallel Processing and Applied Mathematics, Proceedings of Vth International Conference, PPAM 2003, Częstochowa, Poland, 2003. Volume 3019 of Lecture Notes in Computer Science., Springer (2004) 97–104
5. Banaś, K.: On a modular architecture for finite element systems. I. Sequential codes. Computing and Visualization in Science **8** (2005) 35–47
6. Płażek, J., Banaś, K., Kitowski, J.: Comparison of message passing and shared memory implementations of the GMRES method on MIMD computers. Scientific Programming **9** (2001) 195–209
7. Banaś, K.: The application of the adaptive finite element method in large scale computations (in Polish). Wydawnictwo Politechniki Krakowskiej, Kraków (2004)