Improving productivity for parallel finite element codes through software engineering

D.R. Shires*, B.J. Henz

US Army Research Laboratory, Aberdeen Proving Ground, MD 21005, USA

Abstract

Software development remains a central problem for large-scale parallel computing systems and as a result productivity for these systems is continuously hampered. Developing the codes to run on these systems requires new approaches. This paper discusses a software development environment that is focused on the domain of computational engineering problems that employ the finite element method. The Simple, Parallel, Object-Oriented Computing Environment for the Finite Element Method (SPOOCEFEM) is an active development environment designed to achieve the goals of a productive library while delivering required parallel performance. It incorporates all the benefits of object-oriented design and has proven utility in several applications. This paper describes the design and implementation of SPOOCEFEM, provides metrics on its utility, and discusses how the system is used in practice.

Keywords: Object-oriented design; Finite element method; Parallel programming

1. Introduction

Today's large-scale computing systems are complex architectures. A myriad of parallel computing architectures exist with each subsequent release being somewhat faster than the one before. However, while hardware continues to improve, software development for these complex machines has continued to lag. While a petaflop system may not be too far in the future, a serious concern is whether or not the software will be in place to actually use such a powerful machine. Without a concerted effort to address this issue, it is quite probable that only the most trivially parallel applications could be developed for such a machine in a short period of time.

Revolutionary software engineering practices and techniques will only become reality through a concerted effort to incorporate more complex and intelligent compiling environments coupled with adaptable code that can learn from its runtime implementation. While this is some way off, the object-oriented design (OOD) paradigm has matured to the point where computational science needs to embrace its concepts to provide the required functionality in current productive software systems. Our first-hand experiences in software development for parallel computers have shown how beneficial a change in mindset can be to efficient code development.

The remainder of this paper discusses our efforts to construct an object-oriented development library that brings parallelism to the field of finite element analysis. The Simple, Parallel, Object-Oriented Computing Environment for the Finite Element Method (SPOOCEFEM) was constructed to facilitate code development for scientific engineering applications that use the finite element method. Object-oriented programming (OOP) is not widespread in computational science but it has been successfully used. Following is a discussion of our OOP methodology for this class of problem and how we were able to successfully increase our productivity through this approach.

2. The SPOOCEFEM goals and structure

SPOOCEFEM had several goals. First and foremost it had to balance overall code execution speed with development time. Since we had past experience with programming parallel finite element codes, we had a good understanding of how to adequately manage this. Other goals were to create a system that would promote code reuse, limit testing, and incorporate parallelism

* Corresponding author. Tel.: +1 410 278 5006; Fax: +1 410 278 4983; E-mail: dale.shires@us.army.mil

Computational Fluid and Solid Mechanics 2005
K.J. Bathe (Editor)

with the library managing and controlling all inter-processor communication.

SPOOCEFEM is written mainly in ANSI-standard C++ and is built from many different components, as shown in Fig. 1. The most basic components are located at the bottom of the stack with each subsequent layer representing additional functionality. The SPOOCEFEM library has standardized around the eXtensible Data Model and Format (XDMF) for all data storage [1]. This format provides the required large-scale binary storage needed for large data sets, contains the functionality required to represent finite element data, provides for communication with interdisciplinary applications, and is supported by the open source Paraview scientific visualization package from Kitware. A general-purpose customizable graphical user interface (GUI) is provided that supports geometry rendering along with dialog boxes to set various model parameters. Parallelism is realized by a domain decomposition approach for the finite element meshes coupled with the message-passing interface (MPI).

3. SPOOCEFEM in practice

SPOOCEFEM development began in the early part of 2002. It has continued to mature as applications are constructed within its framework. Not including all of the packages that it incorporates, SPOOCEFEM currently stands at roughly 20 000 lines of code. About 17% of this includes the code to implement the SPOOCEFEM GUI.

SPOOCEFEM has successfully been used in three applications. The first is a composite manufacturing process model known as the Composite Manufacturing Process Simulation Environment (COMPOSE) [2]. COMPOSE uses unstructured finite element meshes to simulate resin flow in various composite manufacturing processes. COMPOSE uses SPOOCEFEM by extending the base classes through inheritance (additional element capabilities required by COMPOSE are implemented this way). All of the mesh partitioning for multi-processor execution is handled by SPOOCEFEM.

Two additional codes were also written using SPOOCEFEM following the success of the COMPOSE developmental effort. PhoenixFlow is a multidisciplinary application encompassing fluid flow, heat transfer, and resin cure. PhoenixFlow utilizes OOD to inherit the fluid flow model from COMPOSE and expand it. Finally, a multi-scale residual thermal stress analysis code (MSStress) has been written using SPOOCEFEM. While no direct inheritance is used, it is constructed in this framework to utilize the GUI and for parallelization.

4. SPOOCEFEM benefits

In the circles of scientific computing, the mention of OOD and OOP are often met with disdain. The belief that poor performance will result from class member data layout causing poor cache performance or from the runtime overhead for pointer chasing arising from polymorphism is well known [3]. However, we feel that belief to be a bit too far reaching. Since most of the time spent in scientific codes is on number crunching or solving systems of equations, careful choices and design can be made to still provide for good performance. Table 1 shows the time required for the optimized Fortran 90 version of COMPOSE versus the optimized SPOOCEFEM version of COMPOSE. The three

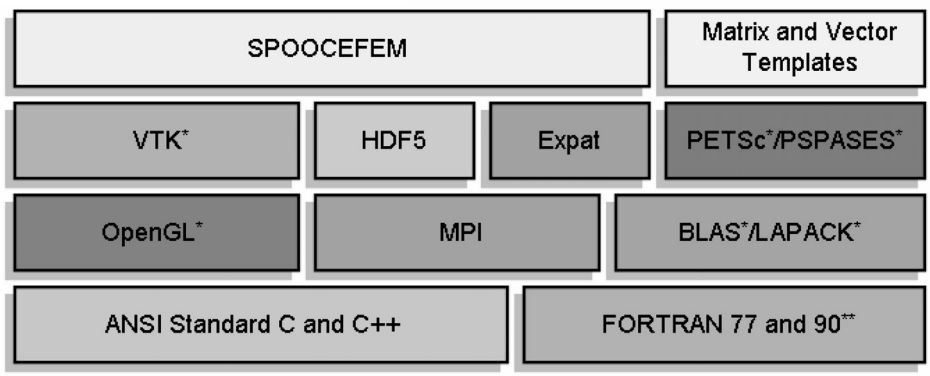

Fig. 1. The SPOOCEFEM building block framework.
Key
*Optional
**Optional, required by PSPASES

Table 1
COMPOSE breakdown of subroutine runtimes for Fortran 90 and C++ versions

Operation	Fortran 90	C++ (SPOOCEFEM)
Linear solver	2766	2694
Calculate pressure	114	128
Update	22	71
Core total time	2902	2893

routines listed comprise over 96% of the total execution time of the code. While the update and pressure routines do require more time to complete due to the OOD, the combined runtime of these two modules is only a small fraction of the overall time.

Since we initially constructed the COMPOSE code in Fortran 90, we have some idea about the amount of time required to complete the code development comparing the two different approaches. It took approximately 4 man years of development time from the start to completion of testing for the Fortran 90 parallel version of COMPOSE. In contrast, the SPOOCEFEM version of PhoenixFlow (which incorporates the functionality of COMPOSE), required only 2.5 man years of development time and is inclusive of the growing capabilities of SPOOCEFEM during the code development (a savings of roughly 38%).

Also, we have noted a marked decrease in the number of source lines of code (SLOCs) required to achieve the same results. The initial Fortran 90 COMPOSE code that provided support for three different element types required about 5700 SLOCs. In contrast, the SPOOCEFEM COMPOSE version took only 2300 SLOCs to provide the same functionality. In the older version of the code, roughly 7000 SLOCs were required to provide the mesh reading, converting, and partitioning tasks. This functionality is now part of the SPOOCEFEM library and included in the 16 000 lines of SPOOCEFEM implementation. The SPOOCEFEM implementation has expanded the number of supported element types, increased the number of solvers available, and can be used again and again. In contrast, the earlier developments, due to the nature of the Fortran 90 environment, are more-or-less limited to a one-time use.

By factoring out as much as possible into libraries, developers are also saved the trouble and effort of repeatedly testing and debugging. The testing process is often overlooked or given a low priority by computational scientists but, in fact, can take considerable time. Software validation will constantly be changing as it usually involves testing against known experimental values. However, software verification, which usually means checking serial and parallel runs against themselves or analytical solutions, can be greatly reduced through the use of a library that has been thoroughly tested. Furthermore, problems such as memory leaks and un-optimized code can be corrected once and never have to be touched again.

5. Conclusions

Only through maximizing productivity can large-scale parallel computers truly address the scientific computing problems of today and the future. The computer hardware provides the tool to arrive at a solution but the software still has to be constructed. That process still takes time, and will likely continue to be outpaced by hardware developments for some time. However, approaches like OOD, while not new, are now proving their worth to the scientific programming community. We have shown the application of these approaches for the finite element methodology. As hopefully demonstrated by this paper, the results can be dramatic and help position software development teams for continued success as the parallel computing landscape continues to expand into the future.

Acknowledgments

This research was supported in part by a grant of computer time and resource by the Department of Defense High Performance Computing Modernization Program. Additional support was provided by the US Army Research Laboratory's Major Shared Resource Center.

References

[1] Clarke JA, Namburu RR. A distributed computing environment for interdisciplinary applications. Concurrency and Computation: Practice and Experience 2002;14:11161–1174.

[2] Henz BJ, Mohan RV, Shires DR. Large-scale integrated process modeling simulations enabling composite material developments and applications. Proceedings of the 2004 National Space and Missile Materials Symposium, Seattle, WA, 2004.

[3] Todd I, Veldhuizen TL, Jernigan ME. Will C++ be faster than Fortran? Lecture notes in computer science, vol. 1343, Springer-Verlag, London, 1997, 49–56.