**Toward Object-Oriented Parallel Finite Element Computations**

Shang-Hsien Hsieh,¹ Elisa D. Sotelino,² Donald W. White,³ and Wai-Fah Che⁴

School of Civil Engineering
1284 Civil Engineering Building, Purdue University
West Lafayette, IN 47907-1284, U. S. A.

**Abstract**

Research on an object-oriented application framework for parallel finite element computations is currently underway in the School of Civil Engineering at Purdue University. The main goal of this research is to facilitate reuse and rapid prototyping of a wide variety of parallel finite element software in Structural Engineering applications. The framework is being developed by taking advantage of the object-oriented methodologies supported by C++ as well as recent advances in parallel computing. This paper gives an overview of the ongoing research effort by highlighting several reusable components that have been or are being developed within the target framework.

1. Introduction

The finite element method has been widely utilized to solve engineering applications because of its ability to model complex structural geometries and material properties, and its adaptability to high-performance computation. However, the timely development of high-quality finite element software is still a
challenging task. Many existing finite element analysis systems are one-of-a-kind software built by components which are designed and developed only for a specific application. The extension of such systems for solving different types of applications is usually difficult if not impossible. Code duplication is common within and across many types of systems developed for solving different types of applications because of difficulties in reusing existing software. As a result, the software development efforts often cannot be accumulated, and it is not uncommon for researchers to spend an inordinate amount of their time in the development and maintenance of their finite element software.

In addition, the advancement of the parallel processing technology has made the finite element method an increasingly powerful tool for solving computationally intensive engineering applications. However, the parallelization of existing sequential finite element software can be a tedious task that often requires careful modifications and extensions of the program’s underlying data structures. Moreover, the portability of the parallel finite element software among the various existing parallel machines is a common and difficult issue that needs to be addressed.

The research effort described in this paper is an attempt to shed some light on the above issues related to the development of parallel finite element software. The objective of the work is to improve the reusability, quality, productivity, maintainability, and portability of parallel finite element software by applying the object-oriented technology [1] and parallel programming techniques. The focus is on the distributed-memory parallel computing environments. An object-oriented parallel finite element framework, called PFE++, is currently under development. PFE++ is a major component of the Structural Engineering Concurrent Software Development Environment (SECSDE) [2], which is currently being developed in the School of Civil Engineering at Purdue University. The main goal of the SECSDE
research is to create a software development environment to facilitate reuse and rapid prototyping of high-performance computing applications in Structural Engineering. Object-oriented methodologies, as well as parallel programming and computer graphics techniques, are being investigated and integrated to build the target environment. The C++ language [3] is the major programming language being used.

The remainder of this paper describes the components that have been or are being developed that are a part of or are being used by PFE++. In Section 2, a parallel portability interface for providing message-passing operations required in parallel computations is discussed. Section 3 presents a parallel matrix library that supports a set of high-level parallel matrix abstractions to ease the parallel programming involving parallel matrix operations often encountered in parallel finite element computations. A sequential object-oriented finite element framework, called FE++, is described in Section 4. This framework serves as the basis for the development of PFE++. The tasks required in the ongoing effort to parallelize FE++ into PFE++ are discussed in Section 5. Finally, Section 6 provides some closing remarks.

2. Parallel Portability Interface

To provide a stable (unchanging) interface between the client parallel programs and the rapidly evolving distributed computing environments, an object-oriented parallel portability library called PPI++ has been developed [4,5]. By taking advantage of object-oriented techniques, PPI++ also supports a clean, consistent, and easy-to-use C++ interface for message-passing programming. The programming semantics supported by the interface help improve the clarity and
expressiveness of client parallel codes. The implementation details and complexity are encapsulated by the interface to ease parallel programming tasks.

The design of PPI++ takes advantage of the object-oriented model used by MPI [6]. MPI is a new de facto message-passing interface standard with either C or FORTRAN language binding, recently proposed to promote portability and efficiency of parallel codes as well as to provide better support for development of parallel libraries [7]. To benefit fully from the functionality, portability, and efficiency provided by MPI, the present implementation of PPI++ uses MPI functions to achieve interprocess communication and synchronization. However, the C++ user interface of PPI++ is designed to be independent of the use of MPI. This separation of the interface and implementation ensures that no change in the client code is required if there is a need to change the implementation.

It has been demonstrated in Ref. [8] that with the aid of PPI++, the porting of parallel client codes to a variety of parallel systems can be greatly simplified. In addition, the study in Ref. [4] has shown that the overhead introduced by PPI++ on top of MPI is very small.

3. Parallel Matrix Library

Parallel finite element computations usually involve several types of parallel matrix operations, such as parallel matrix-vector multiplication, parallel vector dot-product, and parallel solution of system of equations. To provide an object-oriented, semantically clean and consistent support for the development of PFE++, a parallel matrix class library has been developed [9]. The library is designed to encapsulate as much as possible the complexity of both the data management and the message passing involved in parallel matrix algorithms using the object-oriented approach.
The library uses two orthogonal sets of base class abstractions to capture: (1) the data distribution characteristics among parallel processes and (2) matrix sparsities for different types of parallel matrices commonly encountered in parallel computational mechanics. All other library classes are then derived from these two sets of base classes using multiple inheritance to ensure consistent semantics. For example, the \textbf{PScaMatrix} class is a base class used to model parallel matrices that have their data scattered among parallel processes while each data entry (coefficient) is managed uniquely by an individual process. The \textbf{PSymMatrix} class is a base class used to model parallel symmetric matrices. The \textbf{PScaSymMatrix} class is then derived from both the \textbf{PScaMatrix} and \textbf{PSymMatrix} classes to model parallel scattered symmetric matrices. More detailed description of class abstractions used in the library can be found in Ref. [9].

At the present time (March 1995), both the LU decomposition solver and the diagonally preconditioned conjugate gradient solver have been implemented for the parallel scattered symmetric profile matrices in the library. Some preliminary studies on the efficiency of these two solvers on the Intel Paragon parallel computer at Purdue University have been reported in Ref. [9]. Work is underway to further improve the parallel efficiency of the library and to implement the above two solvers for other types of parallel matrices.

In addition, the library uses PPI++ for the implementation of all the required message passing operations. With the aid of PPI++, not only the implementation is greatly simplified but also the library enjoys the same portability as PPI++.

4. A Framework for Finite Element Programming

To support the development of a wide variety of finite element application software, an object-oriented framework for finite element programming called FE++
has been created in the SECSDE. FE++ provides a set of high level programming abstractions that model closely the components and concepts in the finite element computations to facilitate rapid prototyping of finite element applications. It also defines a set of collaborative mechanisms among FE++ objects to allow for easy use of a variety of finite element formulations and solution algorithms with more specific procedures.

The key class abstractions in FE++ include the Node, Element, Assemblage, Material, AnalysisDriver, TimeVariable, and Matrix classes. They are briefly discussed below [11]:

- The Node class and its derived classes maintain nodal information of a finite element mesh, such as degrees of freedom (DOF’s), the coordinate system at a node, the list of adjacent elements, and nodal displacement vectors, etc. There is no upper limit imposed on the number of DOF’s per node in FE++.

- The Element class hierarchy is a collection of various finite elements formulated on the basis of the displacement method. The common behavior of different finite elements is captured in the Element base class by defining a set of protocols, including the operations for querying the element stiffness, mass, and damping matrices, the internal force vector of the element, and for updating element internal state during finite element analysis.

- The Assemblage class is derived from the Element class to support structuring of the elements. It hides solution domain details, such as assignment of equation numbers and assembling element quantities (e.g., element stiffness matrices), from the client programmer. It is capable of handling heterogeneous elements with variable number of DOF’s per node.
• The **Material** classes are used to encapsulate the constitutive relationships of the physical materials.

• An **AnalysisDriver** is an active object that drives the progress of analysis. Various static and dynamic solution schemes can be easily accommodated in FE++.

• The **TimeVariable** class models quantities that are a function of time (in the case of static analysis, the pseudo-time concept is used).

• The **Matrix** classes provide support for matrix operations in FE++. More detailed discussions of the matrix library are provided in Ref. [12].

5. From FE++ to PFE++

One popular approach to parallelizing finite element computations in Structural Engineering is the use of the substructuring technique. In this approach, the finite element mesh is partitioned into a number of subdomains. The substructure condensation for each subdomain is performed on a separate process independently and concurrently with no interprocess communication required. The solution of the unknowns along the subdomain interfaces is then carried out using either parallel direct (e.g., parallel LU decomposition) or parallel iterative (e.g., parallel preconditioned conjugate gradient) algorithms. Once the interface unknowns are solved, subsequent solution of the unknowns within each subdomain can be computed independently and concurrently without the need for interprocess communication. This is the major approach that has been chosen to parallelize FE++ into PFE++.

To support the substructuring method, a **SuperElement** class abstraction has been derived in FE++ [13]. Similar to an **Assemblage**, a **SuperElement** consists
of a collection of elements and supports the assembly of the elements. However, unlike the Assemblage, the SuperElement does not apply the boundary constraints and it supports the substructure condensation within the SuperElement. In addition, the present design of the SuperElement class allows for multi-level substructuring [13].

To support the parallel assembly of the interface unknowns, PAssemblage is another important class abstraction being derived in FE++ [13]. A PAssemblage object (constructed in a local process) refers conceptually to the assemblage of the complete finite element mesh although it contains only elements within its local subdomain. Using the PPI++ discussed in Section 2 and the parallel matrix library discussed in Section 3, the PAssemblage class encapsulates the complexity and details involved in the domain of parallel solution, such as assembling condensed superelement stiffness matrices into a global stiffness matrix in the form of a parallel matrix. For parallel solution of the interface unknowns, the parallel solvers supported in the parallel matrix library can then be used.

FE++ was originally developed without the consideration of parallel computing. However, with the help of object-oriented technology and with the support of the PPI++ and the parallel matrix library, its extension to PFE++ only requires some minor modifications in its existing classes besides the addition of the SuperElement and PAssemblage classes. In addition, research on the development of an object-oriented domain partitioning framework is currently underway to support the automatic generation of subdomains for parallel analysis.
6. Closure

Ongoing research toward an object-oriented application framework for parallel finite element computations has been described in this paper. By taking advantage of the object-oriented technology as well as recent advances in parallel processing, it is expected that the target framework will facilitate reuse and rapid prototyping of a wide variety of parallel finite element application software. The key goals are improved software quality, reliability, maintainability, extensibility, and portability.

Acknowledgments

This research is sponsored by the U.S. Department of Energy under Award No. DE-FG02-93ER25169. The support from the U.S. Department of Energy and the specific contributions by Dr. J. Lu, Mr. G. R. Mukunda, and Mr. S. Modak to this work are gratefully acknowledged. Any opinions expressed in this work are those of the authors and do not necessarily reflect the views of the sponsor.

References


