A Data Management and Communication Layer for Adaptive, Hexahedral FEM

Judith Hippold* and Gudula Rünger

Chemnitz University of Technology, Department of Computer Science
09107 Chemnitz, Germany
{juh,ruenger}@informatik.tu-chemnitz.de

Abstract. The parallel realization of adaptive finite element methods (FEM) has to deal with several irregular and dynamic algorithmic properties caused by adaptive mesh refinement (AMR). For an implementation on distributed memory machines irregular communication behavior results from dynamically growing data structures and statically unknown communication partners. An efficient parallel implementation has to provide appropriate mechanisms to cope with the flexibility of the adaptive finite element approach at runtime. We have implemented a data management and communication layer for an adaptive, 3-dimensional, hexahedral FEM on distributed memory machines and use it to parallelize an existing sequential code. The data management and communication layer realizes duplicated data structures for boundaries of distributed data, additional hierarchical data structures to deal with uneven refinement, and coherence protocols to guarantee correctness of communication partners and messages. An easy to use interface provides access to the functionality of the layer.

1 Introduction

The discretization of the physical domain into a mesh of finite elements and the approximation of the unknown solution function by a set of shape functions on those elements makes the finite element method flexible and applicable for a wide range of different applications. For the simulation of large problems adaptive mesh refinement and parallel execution can help to reduce runtime. Especially for 3-dimensional problems, an efficient parallel realization is difficult to achieve and mostly interferes with the algorithmic structure. Parallelization modules accessible by interface functions can provide easy to use parallel functionality for application programmers, maintain the algorithmic structure, and offer reusability for a wide range of different finite element implementations.

Adaptive FEM belongs to the class of irregular algorithms. Irregularity is caused by AMR and hanging nodes: The adaptive refinement process with computations on different refinement levels creates hierarchies of data structures and requires the explicit storage of these structures including their relations. Hanging nodes can result from different refinement levels and are mid-nodes of faces

* Supported by DFG, SFB393 *Numerical Simulation on Massively Parallel Computers*.

or edges which are vertices of finite elements at the same time. Such nodes require projections on nodes of other refinement levels during the solution process. Both characteristics lead to irregular communication behavior. Thus the efficient parallel implementation is difficult, especially for distributed memory.

The contribution of this paper is the design and realization of a data management and communication layer for parallel, adaptive FEM with emphasis on hexahedral finite elements. Our layer encapsulates communication and the management of distributed data which allows easy extensibility and does not require modifications in the algorithmic structure of the existing sequential FEM code. The data management concept provides duplicated data storage with fast access and modification functions and also supports hanging nodes. A special communication mechanism reduces the number of messages considerably, decreases the overhead for managing asynchronous communication behavior, and allows SPMD programming style despite of irregular characteristics. Furthermore it offers wide facilities for optimization like additional overlapping of communication and computation phases, software caching, and monitoring of program behavior. We have incorporated the data management and communication layer into an existing sequential, adaptive, 3-dimensional, hexahedral FEM program package which was developed and implemented at the *SFB 393: Numerical Simulation on Massively Parallel Computers* at the Chemnitz University of Technology. However, the module is designed to be integrated into further FEM packages.

The paper is organized as follows: Section 2 summarizes the given adaptive FEM implementation. The concepts of our data management and communication layer are introduced in Section 3. Section 4 presents experimental results and Section 5 concludes.

2 The FEM Implementation

SPC-PM3AdH [1] is a 3-dimensional, adaptive finite element software package suitable to solve 2nd order elliptic partial differential problems like the Poisson equation or the Lamé system of linear elasticity. [2–4] present FE-software which differ from SPC-PM3AdH, e. g. in finite elements or solvers implemented or parallel realization. SPC-PM3AdH implements hexahedral elements with linear and quadratic shape functions as shown in Figure 1.

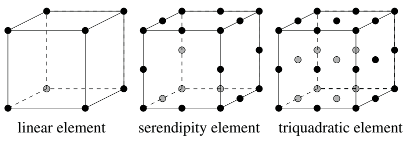
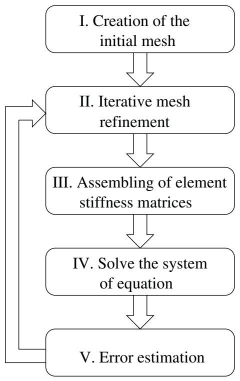


Fig. 1. Finite elements implemented in SPC-PM3AdH.

Each hexahedron has the maximum number of 27 nodes. If only linear or serendipity elements with 20 nodes are used, the remaining nodes actually exist but have no associated shape functions. A finite element is implemented through a hierarchy of data structures: *Volumes* are the most coarse-grained structure and represent the finite elements. Volumes are composed of faces and each face consists of four *edges*. The smallest unit is the *node*. Nodes have three coordinates and, when they are associated with a shape function, a solution vector. Nodes can be vertices or mid-nodes of volumes, faces, and edges.

The program structure of SPC-PM3AdH consists of five phases as illustrated on the right:


I. Creation of the initial mesh
II. Iterative mesh refinement
III. Assembling of element stiffness matrices
IV. Solve the system of equation
V. Error estimation

- I First the initial mesh is created from an input file.
- II Volumes are subdivided into 8 children according to the estimated error and geometrical conditions like hanging nodes. The difference of refinement levels between two neighboring volumes is restricted to one.
- III Each volume is associated with an element stiffness matrix. The matrices for new volumes, the right hand side vectors, and the global main diagonal are assembled in the third phase.
- IV The system of equations is solved with the preconditioned conjugate gradient method. A Jacobi, Yserentant [5], or BPX [6] preconditioner can be selected for preconditioning.
- V The error is estimated with a residual based error estimator [7]. Volumes are labeled for refinement if their estimated error is close to the maximum error.

3 The Data Management and Communication Layer

The basis of the data management and communication layer is a specific approach for distributed data storage and administrational data structures containing distribution information. The entire information concerning the distribution of data is hidden to the user but can be accessed and modified by functions supplied by the layer. The layer encapsulates the actual data exchange via MPI and communication optimizations.

3.1 Distribution of Data Structures

The parallel implementation assigns volumes to processors which leads to a partitioning of the stiffness matrix represented by local element stiffness matrices. The faces, edges, and nodes shared between neighboring volumes in different address spaces exist duplicated within the memory of each owning processor. This allows fast computations for each volume with minimal communication. Processors sharing duplicates are called *owners* of duplicates. The approach induces that the solution and auxiliary vectors are spread over the address spaces of the different processors. The entries for duplicated nodes exist redundantly and contain only subtotals which have to be accumulated to yield the total result. For duplicated hanging nodes *worker* processors can be selected which are allowed to perform the necessary projections exclusively. This avoids multiple accumulations.

Refinement of distributed data. The adaptive refinement process creates a hierarchy of faces and edges. Due to hanging nodes processors may perform computations on different refinement levels of the same edge or face. To gain fast access on the different levels faces are organized in a quadtree data structure and edges as binary trees. Refinement on duplicated faces and edges subdivides data structures only in local memory. Thus the remote hierarchy trees have to be kept consistently by a two-phase iterative refinement cycle:

The *first phase* iteratively subdivides local volumes according to the estimated error and geometrical demands, like unacceptable hanging nodes. In the *second phase* the remote refinement of subdivided duplicated faces and edges is done. A synchronization step ensures that the entire process is performed until no further subdivision of volumes is done on any processor.

Coherence lists. To keep data consistent and to support remote refinement the fast identification and localization of duplicates is necessary. For that reason the tuple *Tup(identifier, processor)* is introduced. *Identifier* denotes a data structure of type face, edge, or node and serves for local identification. The parameter *processor* allows global identification of a duplicate. This parameter is assigned after distributing a data structure and is the number of the processor where the duplicate is situated. The combination of both parameters allows unique identification.

The tuple *Tup* is used to implement *coherence lists*. Each duplicated data structure is tagged with a coherence list which contains information about the remote identifiers and the location of *all* existing duplicates. Figure 2 illustrates such lists for the edges *e1* and *e2* in the address space of processor *P2*. To access remote duplicates the owners and the corresponding remote identifiers can be extracted and used to send a message. A receiving processor is able to

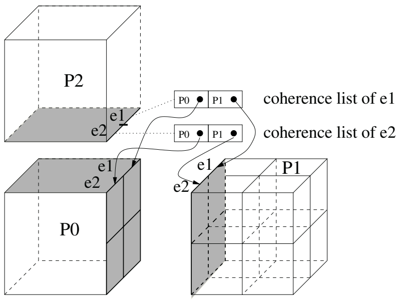


Fig. 2. Illustration of the coherence lists for the edges *e1* and *e2* in the address space of processor *P2* and illustration of remote refinement necessary for processors *P0* and *P2* after local refinement on *P1*.

determine a local structure by the received identifier and to perform actions like remote refinement. Coherence lists are built up and updated within the data management layer. The local identifier of a data structure serves as input parameter to find the corresponding list. Data structures without duplicates in remote address spaces have no lists associated.

Data consistency of refined volumes. Adaptive refinement on volumes distributed over different address spaces requires to maintain consistency for correct program behavior. This includes the remote refinement of duplicated faces and edges and the creation or update of coherence lists for new data structures. Figure 2 also illustrates remote refinement: The subdivision of the volume in the address space of processor *P1* requires the refinement of one face and of four edges located at *P0* and of one edge located at *P2*. The entire process can be generalized and structured into four steps. We refer to the data structures on higher levels of the hierarchy trees as *parents* and to the corresponding new structures arising from subdivision as *children*.
(1) To identify data structures which have to be remotely subdivided each parent volume is investigated for duplicated faces and edges. For each found duplicate, a data package has to be sent. It contains the remote identifier of the duplicate for identification and the local identifiers of the newly created data structures (e. g. nodes, children) for updating the coherence lists.
(2) The receiving owners refine the parent data structure identified by the received package and create and update the coherence lists of the new structures with the information of the message package. If the parent data structure is already subdivided, only the update of coherence lists is necessary.
(3) To update the lists of the remotely created, new structures in the address spaces of the remaining owners the identifiers of the new duplicates have to be sent back according to the package structure described in item (1).
(4) The other owners receive the messages and update their coherence lists.

3.2 Communication Approach

During a program run different situations of communication occur. They can be classified into three types:

Synchronization and exchange of results: Communication is necessary to synchronize the refinement process and to determine the maximum error. This communication takes place globally between all processors.

Accumulation of subtotals: For duplicated nodes only subtotals are computed by the different owners. To yield the total result data exchange is necessary.

Exchange of administrational information: In order to keep data consistent administrational information has to be exchanged in each program iteration step. This comprises for example global procedures like remote subdivision, the generation and update of coherence lists, and the identification of hanging nodes.

The last two communication situations have irregular characteristics. That means the communication partners, the number of messages, and the message sizes vary and depend on the specific program run. Furthermore the exact communication time is not known in advance. [8, 9] present a communication concept for irregular algorithms. However, the communication requirements for adaptive FEM are slightly different and suggest the following realization:

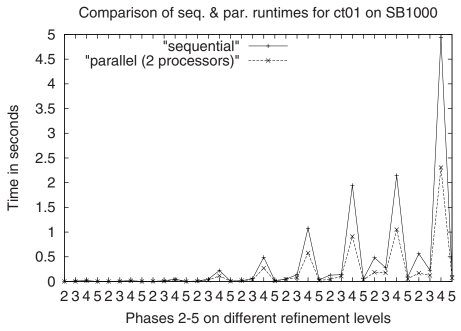
Communication protocol. Due to the special duplicated data storage of the data management layer and the program structure of SPC-PM3AdH the exchange of data within a computational phase can be delayed. This allows to displace the communication at the end of a computational phase and to separate computation from communication. The resulting communication mechanism can be described as follows:
(A) During computation each processor collects information about necessary data exchanges with different collect functions. These functions can be chosen adapted to the algorithmic needs and detect duplicates for a given local data structure. If there are duplicates, the remote identifiers and additional information are stored in send buffers for later exchange. During the computational phase a collect function can be called several times.
(B) After the local computations the gathered values are sent to the corresponding processors extracted from the coherence lists. This process is initialized by the first call of a get function. The application programmer uses get functions to obtain data from the receive buffer. Each function call returns an identifier of a local data structure and the received additional information for this structure.
(C) Afterwards specific actions can be performed. The return of an invalid identifier denotes an empty receive buffer.

The usage of the described collect&get communication mechanism is illustrated by the following pseudo-code for the parallel labeling of edges with hanging nodes.

```
for each local volume V { for each edge E of V {
            if(E is subdivided) {
                  label children S1 and S2 of E as hanging;
                  collect(S1, S2);   /* (A) */
}     }     }
while(get(&id)) /* (B) */ { label the edge identified by id; /* (C) */ }
```

4 Experiments

This section shows some measurement results demonstrating the usability of the layer within the program SPC-PM3AdH. To gain experimental results two platforms are used: XEON, a 16x2 SMP cluster of 16 PCs with 2.0 GHz Intel Xeon processors running Linux and SB1000, a 4x2 SMP cluster of 4 SunBlade 1000 with 750 MHz UltraSPARC3 processors running Solaris. The MPI implementation is ScaMPI using an SCI network. We consider three examples: *layer3* a boundary layer for the convection-diffusion equation, *ct01* solving the Lamé equation, and *torte4d* a layer near a non-convex edge [1]. For parallel and sequential measurements linear finite elements and the Jacobi preconditioner are used.



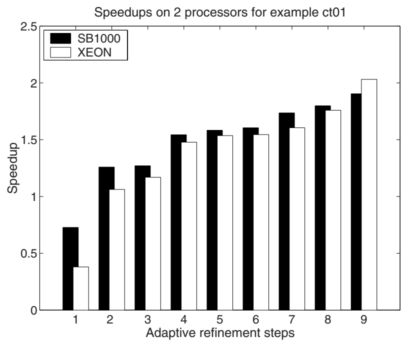


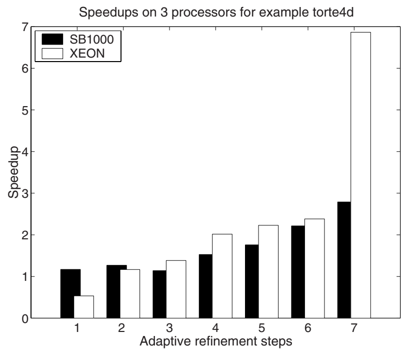
Fig. 3. Left: Execution times for the different algorithmic phases on different refinement levels for example *ct01* on SB1000 (2: AMR, 3: assembling element stiffness matrices, 4: solving the system of equations, 5: error estimation). Right: Speedups for example *ct01* on 2 processors.

Small numbers of processors are sufficient to achieve efficiency for the available examples.

The algorithmic phase consuming most of the sequential and parallel calculation time is to solve the system of equations (see Figure 3, left, phase 4). During this phase also the main portion of communication takes place in the parallel version. Figure 3 (right) shows speedup values for the example *ct01* on SB1000 (black) and XEON (white) after different adaptive refinement steps. Speedups improve with increasing refinement level because the computational effort compared to the communication overhead as well as the vector size and the number of volumes grow, e. g. for the specific example the ratio of the number of volumes to the number of sent messages is 8/100, 742/260, and 1884/1544 after the 1st, 5th, and 9th refinement step. For examples *torte4d* and *layer3* (Figure 4) we get superlinear speedup after 7 refinement steps, especially on XEON. This is mainly caused by the growing length of data structure lists and the operations on these lists which are distributed among multiple address spaces in the parallel version. Cache effects might have further influence.

5 Conclusion and Future Work

We have presented a data management and communication layer for adaptive, 3-dimensional, hexahedral FEM on distributed memory including a management for duplicated data storage and a special communication mechanism. Both are completely hidden to the user and accessible by functions of the layer. The advantages of the modular structure are the easy extensibility which does not require modifications in the algorithmic structure of the FEM code and the possibility for internal communication optimizations. First tests of the layer demonstrate usability and deliver good speedup results.



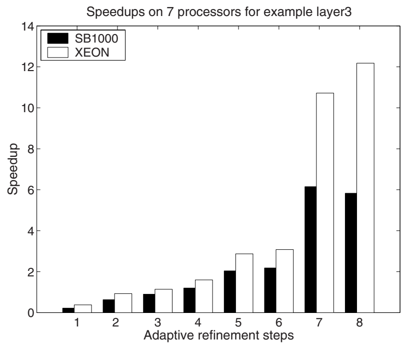


Fig. 4. Speedups for varying numbers of adaptive refinement steps on SB1000 and XEON. Left: example *torte4d* on 3 processors. Right: example *layer3* on 7 processors.

Acknowledgement

This work is supported by DFG, *SFB393: Numerical Simulation on Massively Parallel Computers*. Special thanks to S. Beuchler, A. Meyer, and M. Pester, Chemnitz University of Technology, Department of Mathematics, for their support and many interesting discussions.

References

1. Beuchler, S., Meyer, A.: SPC-PM3AdH v1.0, Programmer's Manual, Technical Report SFB393/01-08, Chemnitz University of Technology (2001)
2. Bangerth, W., Kanschat, G.: Concepts for Object-Oriented Finite Element Software - the deal.II Library, IWR Heidelberg, SFB359 Preprint 99-43 (1999)
3. Blazy, S., Kao, O., Marquardt, O.: padfem2 - An Efficient, Comfortable Framework for Massively Parallel FEM-Applications. In Dongarra, J., Laforenza, D., Orlando, S., eds.: Proc. of EuroPVM/MPI, LNCS 2840. Springer (2003) 681–685
4. Diekmann, R., Dralle, U., Neugebauer, F., Roemke, T.: PadFEM: A Portable Parallel FEM-Tool. In Liddell, H., Colbrook, A., Hertzberger, B., Sloot, P., eds.: Proc. of HPCN-Europe, LNCS 1067. Springer (1996) 580–585
5. Yserentant, H.: On the Multi-level-splitting of the Finite Element Spaces. Numer. Math. **49** (1986) 379–412
6. Bramble, J., Pasciak, J., J.Xu: Parallel Multilevel Preconditioners. Math. Comp. **55** (1991) 1–22
7. Kunert, G.: A Posteriori Error Estimation for Anisotropic Tetrahedral and Triangular Finite Element Meshes, PhD Thesis, TU-Chemnitz (1999)
8. Hippold, J., Rünger, G.: A Communication API for Implementing Irregular Algorithms on Clusters of SMPs. In Dongarra, J., Laforenza, D., Orlando, S., eds.: Proc. of EuroPVM/MPI, LNCS 2840. Springer (2003) 455–463
9. Hippold, J., Rünger, G.: Task Pool Teams for Implementing Irregular Algorithms on Clusters of SMPs. In: Proc. of the 17th IPDPS, CD-ROM. (2003)