An Adaptive, 3-Dimensional, Hexahedral Finite Element Implementation for Distributed Memory

Judith Hippold[⋆] [a], Arnd Meyer [b], and Gudula Rünger [a]

Chemnitz University of Technology,
[a] Department of Computer Science, [b] Department of Mathematics
09107 Chemnitz, Germany
{juh,ruenger}@informatik.tu-chemnitz.de,
a.meyer@mathematik.tu-chemnitz.de

Abstract. Finite elements are an effective method to solve partial differential equations. However, the high computation time and memory needs, especially for 3-dimensional finite elements, restrict the usage of sequential realizations and require efficient parallel algorithms and implementations to compute real-life problems in reasonable time. Adaptivity together with parallelism can reduce execution time significantly, however may introduce additional difficulties like hanging nodes and refinement level hierarchies. This paper presents a parallel adaptive, 3-dimensional, hexahedral finite element method on distributed memory machines. It reduces communication and encapsulates communication details like actual data exchange and communication optimizations by a modular structure.

1 Introduction

Finite element methods (FEM) are popular numerical solution methods to solve partial differential equations. The fundamentals are a 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. The numerical simulation of real-life problems with finite elements has high computation time and high memory needs. Adaptive mesh refinement has been developed to provide solutions in reasonable time. However, there is still need for parallel implementations, especially for 3-dimensional problems as considered in this paper.

The basis of an efficient parallel implementation is a sophisticated algorithmic design offering a trade-off between minimized data exchange, computation overhead due to parallel realization, and memory needs. The actual parallel implementation furthermore requires optimized communication mechanisms to achieve good performance. The main problems to address for adaptive, hexahedral FEM are irregular structured meshes and hanging nodes: An adaptively refined mesh spread out across the address spaces of several processes requires

[⋆] Supported by DFG, SFB393 *Numerical Simulation on Massively Parallel Computers* to keep information about the different refinement levels of neighboring volumes owned by different processes. Furthermore, hanging nodes caused by hexahedral finite elements require several projections during the solution process. Both characteristics lead to high communication needs with irregular behavior. The parallelization approach presented in this paper reduces the number of sent messages by a special numerical design for solving the system of equation which we adopt from [1], [2], and [3] and by a specific communication mechanism. The advantages of the proposed parallel realization are: (a) a reduced number of messages due to the separation of communication and computation and duplicated data storage and (b) the possibility for internal optimizations without modifying the original FEM implementation which is reached by a modular structure. An interface for using the communication mechanism is provided.

The paper is organized as follows: Section 2 gives a brief overview of the FEM implementation. The parallel numerical and implementation approaches are introduced in Section 3. Section 4 presents our parallel realization in detail. Experimental results are given in Section 5 and Section 6 concludes.

2 Adaptive, 3-Dimensional Finite Element Method

The software package SPC-PM3AdH [4] implements the adaptive, 3-dimensional finite element method with hexahedral elements and solves 2nd order elliptic partial differential problems like the Poisson equation (1) or the Lamé system of linear elasticity (2).

$$Lu := -\nabla \cdot (A(x)\nabla u) + cu = f \;\; in \;\; \Omega \subset I\!R^3, \qquad A(x) = diag(a_i)_{i=1}^{3} \tag{1}$$

$$u = u_0 \quad on \;\; \partial\Omega_1, \qquad n^t A(x)\nabla u = g \quad on \;\; \partial\Omega_2$$

$$-\mu\Delta\underline{u} - (\lambda+\mu)\, grad\, div\, \underline{u} = \underline{f} \quad in \;\; \Omega \subset I\!R^3, \qquad \underline{u} = (u^{(1)}, u^{(2)}, u^{(3)})^t \tag{2}$$

$$u^{(i)} = u_0^{(i)} \quad on \;\; \partial\Omega_1^{(i)}, \qquad t^{(i)} = g^{(i)} \quad on \;\; \partial\Omega_2^{(i)}, \quad i = 1,2,3$$

The program uses h-version finite element analysis where refinement of the elements is done according to the estimated error per hexahedron. Finite elements with linear, quadratic, and tri-quadratic shape functions are realized. The finite element method implemented by SPC-PM3AdH is composed of 5 phases:

Phase I: The first phase creates the initial mesh from an input file. A mesh consists of a hierarchy of structures. The most coarse-grained structure is the *volume* which represents a hexahedral finite element. Volumes are geometrically formed by 6 *faces* and each face is composed of 4 *edges*. Edges connect two vertices and a mid-node. *Nodes* are the most fine-grained data structure. They store information about coordinates and the solution vector. To keep track of the development of the adaptively refined mesh there is an additional hierarchy implemented for faces and edges to express the *parent-child* relation.

Phase II: Volumes are subdivided into 8 children according to the estimated error and geometrical conditions. Adaptive refinement may lead to different subdivision levels. The difference of those levels for neighboring volumes is restricted to one which causes additional iterative refinement.

Phase III: To facilitate a parallel implementation the global stiffness matrix

is subdivided and an element stiffness matrix is assigned to each volume. The element stiffness matrices are assembled for newly created volumes by the third phase of the program.
Phase IV: The system of equations is solved with the preconditioned conjugate gradient method (PCGM). For preconditioning a Jacobi, an Yserentant [5], or a BPX [6] preconditioner can be selected.
Phase V: In the last phase the error is estimated with a residual based error estimator [7]. If the error for a volume deviates within a predefined threshold value from the maximum error, it is labeled for refinement.

3 Parallelization Approach

The parallelization approach assigns finite elements to processes. Thus the corresponding data for each volume representing a finite element are distributed among the address spaces of the different processes. For the parallel realization three main problems have to be solved: the management of shared data structures, the minimization of communication needs, and the consistency of different refinement levels.

3.1 Shared Data Structures

Neighboring volumes share faces, edges, and nodes. If two neighboring volumes are situated in different address spaces, the shared data structures are duplicated and exist within the memory of each owner process, which allows fast computation with minimal communication. Vector entries for duplicated nodes exist several times (see Figure 1) and contain only subtotals which have to be accumulated to yield the total result. [8] presents an approach distributing the nodes exclusively over the address spaces.

Computations on duplicated data structures require the unique identification of the different duplicates. For that reason we introduce the tuple *Tup(Identifier, Process)*. *Identifier* denotes a local data structure of type face, edge, or node and *Process* denotes the number of the process that owns the duplicate. The tuple *Tup* is used to implement *coherence lists*. Each duplicated data structure is tagged with a coherence list which contains the identification tuples of all existing duplicates of that structure.

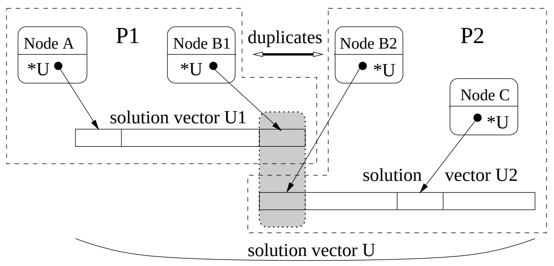


Fig. 1. Solution vector $\underline{u}$ spread over the address spaces of processes P1 and P2. Entries for the node B shared by P1 and P2 are duplicated and contain only subtotals after a computational phase.

3.2 Minimization of Communication Needs

Numerical approach: The discretization of Formulas (1) or (2) with nodal shape functions yields the linear system $V\underline{u} = \underline{b}$ where the global stiffness matrix V and the global right-hand-side vector $\underline{b}$ contain problem describing data and the solution vector $\underline{u}$ has to be calculated. Each process owns only element stiffness matrices V_s and element right-hand-side vectors $\underline{b}_s$ of its volumes, parts of the solution vector, and parts of the main diagonal which is necessary for applying the preconditioners. As introduced in Subsection 3.1 data shared by different processes require global accumulation of partial results. To keep the communication overhead low especially during solving the system of equation we distinguish between data requiring accumulation, e. g. the main diagonal and the solution vector, and data which can be used for independent calculations performed by the distinct processes as V_s and $\underline{b}_s$ (see also [2]). Our parallel preconditioned conjugate gradient algorithm working mainly on unaccumulated data and therefore reducing communication is described in the following:

```
Start:
    Produce $\underline{u}_s$ from $\underline{u}$,
    $\underline{r}_s := V_s\underline{u}_s - \underline{b}_s$,
    $\gamma_{old} := 0.0$
Iterate until convergence:
    (1) $\underline{w} := C^{-1}\underline{r}$
    (2) $\gamma_s := \underline{r}_s^T\underline{w}_s,\ \gamma := \sum_{s=1}^{p}\gamma_s,\ if\ \gamma_{old} > 0\ :\ \beta := \gamma/\gamma_{old},\ \gamma_{old} := \gamma$
    (3) $\underline{s}_s := \underline{w}_s + \beta\underline{s}_s$
    (4) $\underline{w}_s := V_s\underline{s}_s$
    (5) $\alpha_s := \underline{w}_s^T\underline{s}_s \qquad \alpha := \sum_{s=1}^{p}\alpha_s, \quad \alpha := -\gamma/\alpha$
    (6) $\underline{u}_s := \underline{u}_s + \alpha\underline{s}_s$
    (7) $\underline{r}_s := \underline{r}_s + \alpha\underline{w}_s$
```

Communication mechanism: Due to the special algorithmic design the exchange of data within a computational phase can be delayed and performed at the end of that phase, thus separating computation from communication. The resulting collect&get communication mechanism is the following: During computation each process collects information about necessary data exchanges with different *collect* functions which are adapted to the algorithmic needs. Such a function examines the coherence list for a given local data structure and in case of duplicates it stores the remote identifiers and additional values in a send buffer for later exchange. After the computations the gathered values are sent to the corresponding processes extracted from the coherence lists. This data exchange is initialized by the first call of a *get* function. Further calls return an identifier of a local data structure and the received values for this structure from the receive buffer in order to perform specific actions.

3.3 Consistence of Refinement Levels

Adaptivity causes irregularly structured meshes with different refinement levels for neighboring volumes. Thus *hanging nodes* arise for hexahedral volumes (see Figure 2). Hanging nodes need several projections during the solution process which requires accesses to the parent-child hierarchy of the corresponding faces and edges. If the parent and child data structures are situated in different address spaces, as illustrated in Figure 2 for parent face F, the projections either

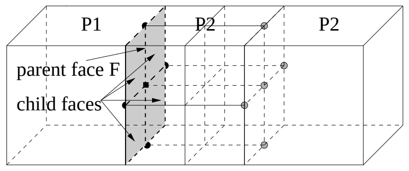


Fig. 2. Illustration of hanging nodes. Projections for grey-shaded hanging nodes access local data in the address space of process P2. Black hanging nodes require duplicated storage of face and edge parent-child hierarchies.

require explicit communication for loading the remote data or the duplicated storage of face and edge hierarchies. Our parallelization approach stores the face and edge hierarchies in the address space of each owner process because this reduces communication and improves performance. For this reason the explicit refinement of duplicated faces and edges within the refinement phase and the creation of coherence lists for these data structures is necessary to keep data consistent. (see Section 4, Phase II).

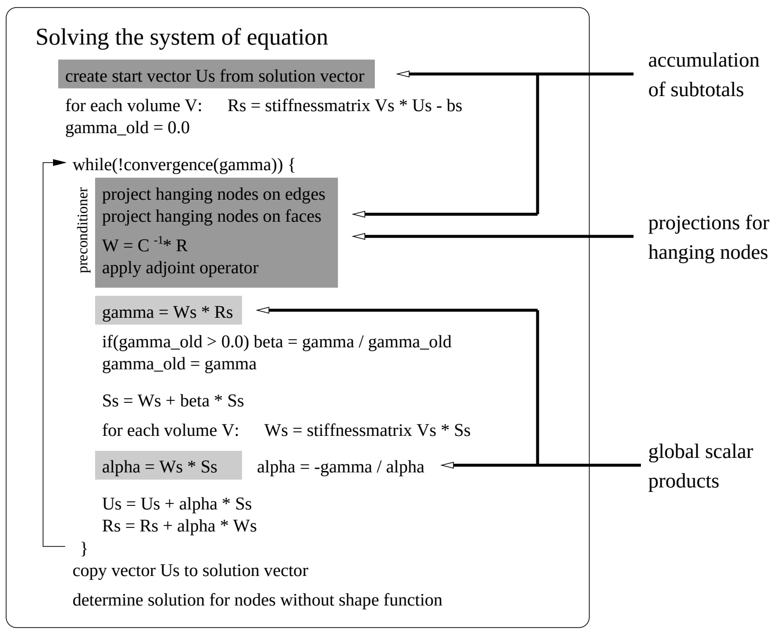
4 Parallel Implementation

This section describes the parallel realization with regard to the necessary data exchanges within the different algorithmic phases.

Phase I - Creation of the Initial Mesh In the first phase the initial mesh is read from an input file. The distribution of data structures is done according to a computed initial partitioning. First the entire mesh exists on each cluster node in order to reduce the communication effort necessary to create the coherence lists. The functions collect_dis and get_dis are provided to determine the duplicated structures and their remote identifiers and owner processes.

Phase II - Iterative Mesh Refinement The parallel execution of the iterative refinement process requires the remote subdivision of duplicated faces and edges in order to keep data structures und coherence lists consistent. For that reason the refinement process is split into 2 steps: The first step iteratively subdivides local volumes and investigates them for duplicated faces and edges. For these faces and edges the identifiers of the children and the identifiers of the connected, newly created edges and nodes are collected with the function collect_ref. The remotely subdivided faces and edges are received using the function get_ref. In the second step the local refinement of those faces and edges and the creation of coherence lists is done. To update the coherence lists at the process initiating the remote refinement the collection and exchange of identifiers is necessary again. Refinement is performed until no further subdivision of volumes is done by any process. A synchronization step ensures convergence.

Projections of hanging nodes during the solution process require to access the corresponding faces and edges. Parallel execution needs explicit communication because processes do not have information about the current refinement levels of neighboring volumes. We reduce the number of sent messages by extracting the necessary information for faces during remote refinement and by using our collect&get communication mechanism for edges.



```
Solving the system of equation
  create start vector Us from solution vector
  for each volume V:     Rs = stiffnessmatrix Vs * Us - bs
  gamma_old = 0.0

  while(!convergence(gamma)) {
      project hanging nodes on edges
      project hanging nodes on faces
      W = C^-1 * R
      apply adjoint operator

      gamma = Ws * Rs
      if(gamma_old > 0.0) beta = gamma / gamma_old
      gamma_old = gamma

      Ss = Ws + beta * Ss
      for each volume V:     Ws = stiffnessmatrix Vs * Ss

      alpha = Ws * Ss     alpha = -gamma / alpha

      Us = Us + alpha * Ss
      Rs = Rs + alpha * Ws
  }
  copy vector Us to solution vector
  determine solution for nodes without shape function
```

Fig. 3. Solving the system of equation with the parallel PCGM. Shaded areas indicate global data exchange. Capital letters denote vectors.

Phase III - Assembling the Element Stiffness Matrices The entire main diagonal and the global right-hand-side vector are re-computed after assembling the element stiffness matrices for the new volumes. For the main diagonal, containing accumulated values, the global summation of subtotals for duplicated nodes is necessary and is supported by the functions collect_val and get_val.

Phase IV - Solving the System of Equation Figure 3 outlines the conjugate gradient method for solving the system of equation in parallel. There are 3 communication situations to distinguish: calculation of scalar products, accumulation of subtotals, and projections for hanging nodes.

To determine the global scalar product each process computes subtotals which have to be accumulated. Duplicated nodes do not require special consideration because computation is done on unaccumulated vectors only. To create a uniform start vector $\underline{u}$ and to provide a uniform residual vector $\underline{r}$ for the preconditioner, partial results for duplicated nodes have to be accumulated using collect_val and get_val. Hanging nodes require several projections. If there are modifications of values for duplicated nodes, communication can be necessary to send the results to the other owner processes. To perform this the functions collect_own and get_own are provided.

Phase V - Error Estimation The parallel error estimator determines the global maximum error by investigating the set of volumes with the maximum local error. To determine the error per volume calculations for the faces of the volumes are necessary. If a face is shared between two volumes, the overall result for this face is composed of the partial results computed by the different owners.

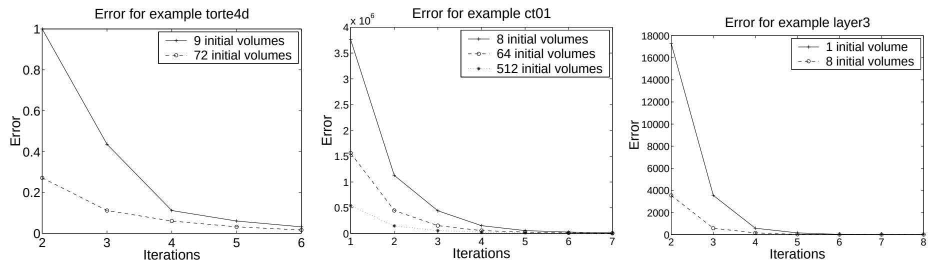


Fig. 4. Error for *torte4d*, *ct01*, and *layer3* using different initial numbers of volumes.

5 Experimental Results

To gain experimental results two platforms have been 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. One process is assigned to each cluster node which enforces network communication. For parallel and sequential measurements linear finite elements and the Jacobi preconditioner have been used. We consider three examples: *layer3* a boundary layer for the convection-diffusion equation ($-\epsilon\Delta u + u = 1 \quad in\ \Omega = (0,1)^3; \quad u = 0\ for\ x,y,z = 0; \quad \frac{\partial u}{\partial n} = 0\ for\ x,y,z = 1$), *ct01* representing the Lamé equation (2) ($\Omega = (0,2)\times(0,1)\times(0,4); \quad \partial\Omega_1^{(i)} = (0,1)\times(0,1)\times\{0\},\ i = 1,2,3; \quad \partial\Omega_2^{(3)} = (0,2)\times(0,1)\times\{4\},\ g^3 = 1000$), and *torte4d* a layer near a non-convex edge ($-\Delta u = 0\ in\ \Omega; \quad \Omega = (0,3)\times(0,\frac{3}{2}\Pi)\times(0,1); \quad u = 100\ on\ \Gamma_1 = \partial\Omega$).

The advantages of adaptivity illustrate the volume refinement histories for adaptive and regular subdivision: e. g. 36 vs. 521; 554 vs. 262,144; 1884 vs. 134,217,728 volumes after $3, 6, 9$ program iterations for *ct01.* The number of initial volumes might be less than the number of parallel processes. Therefore regular refinement is performed at program start until a satisfying number of volumes is reached. Figure 4 compares the development of the maximum error for different initial numbers of volumes.

Figure 5 and Figure 6 (left) depict speedups on SB1000 and XEON for the examples *ct01* and *torte4d* using different initial numbers of volumes. In general speedups increase with growing number of program iterations because the communication overhead compared to the computation effort is reduced. For larger initial numbers of volumes, speedups are in most cases better than for smaller numbers. This is caused by the better computation-communication ratio and by cache effects due to the huge amount of data to process by the sequential program. If the initial number of volumes is too high and many nodes are shared between the processors, speedup decrease is possible with proceeding refinement (see example *ct01* on SB1000).

On the right of Figure 6 sequential and parallel runtimes on XEON are compared for *layer3.* After 6 iterations runtimes increase extremely due to a rapid increase of volumes. Thus cache effects largely influence the achievable speedups (strongly superlinear). Speedups with different calculation bases (sequential, 2, 3 processors) are shown in the middle of Figure 6.

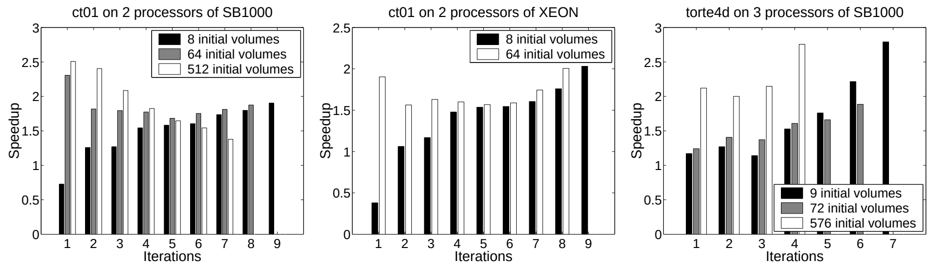


Fig. 5. Speedups for example *ct01* on 2 processors of SB1000 and XEON and for example *torte4d* on 3 processors of SB1000.

6 Conclusion

We have presented a parallel implementation for adaptive, hexahedral FEM on distributed memory. The numerical algorithm and the parallel realization have been designed to reduce communication effort. The modular structure of the implementation allows internal optimizations without modifying the original algorithm. Tests for three examples deliver good speedup results.

References

1. Meyer, A.: A parallel preconditioned conjugate gradient method using domain decomposition and inexact solvers on each subdomain. Computing **45** (1990) 217–234
2. Meyer, A.: Parallel Large Scale Finite Element Computations. In Cooperman, G., Michler, G., Vinck, H., eds.: LNCIS 226. Springer Verlag (1997) 91–100
3. Meyer, A., Michael, D.: A modern approach to the solution of problems of classic elasto–plasticity on parallel computers. Num. Lin. Alg. with Appl. **4** (1997) 205–221
4. Beuchler, S., Meyer, A.: SPC-PM3AdH v1.0, Programmer's Manual. Technical Report SFB393/01-08, Chemnitz University of Technology (2001)
5. Yserentant, H.: On the multi-level-splitting of the finite element spaces. Numerical Mathematics **49** (1986) 379–412
6. Bramble, J., Pasciak, J., J.Xu: Parallel multilevel preconditioners. Mathematics of Computation **55** (1991) 1–22
7. Kunert, G.: A posteriori error estimation for anisotropic tetrahedral and triangular finite element meshes., Phd Thesis, TU-Chemnitz, Logos Verlag Berlin (1999)
8. Gross, L., Roll, C., Schoenauer, W.: Nonlinear Finite Element Problems on Parallel Computers. In: Proc. of PARA'94. (1994) 247–261

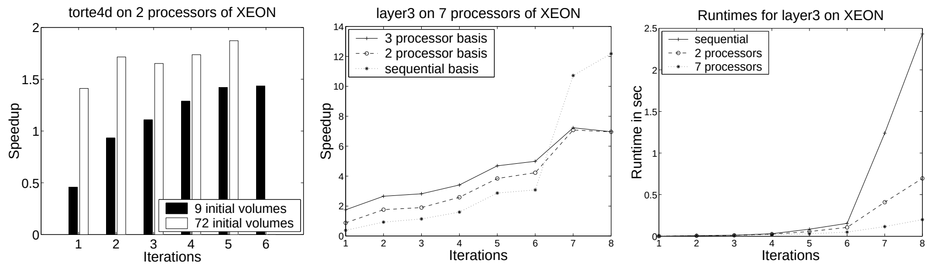


Fig. 6. Speedups for example *torte4d* on 2 and *layer3* on 7 processors of XEON. Comparison of runtimes for example *layer3*.