AMDiS - Adaptive Multidimensional Simulations: Parallel Concepts

Angel Ribalta, Christina Stoecker, Simon Vey, and Axel Voigt

Crystal Growth Group, Research Center Caesar, Ludwig-Erhard-Allee 2, 53175 Bonn, Germany. {ribalta,stoecker,vey,voigt}@caesar.de

Summary. We extend the parallel adaptive meshing algorithm by [2] to further reduce communication requirements in solving PDEs by parallel algorithms. Implementation details and first results for time dependent problems are given.

1 Introduction

In this work we describe the parallelization concepts of our finite element software AMDiS [7], which is a C++ toolbox for solving systems of partial differential equations. It uses adaptive mesh refinement based on local error estimates, to keep the number of unknowns as small as possible. Bank and Holst [2] introduced a new approach for parallelizing such adaptive simulation software, which can be summarized in the following steps:

1. Solve the problem on a relative coarse mesh. Estimate element errors and create partitions with approximately equal error.
2. Each processor solves the problem on the whole domain Ω, but does adaptive refinements only within its assigned local domain (including some overlap).
3. Construct the global solution out of all local solutions by a partition of unity method.

This idea differs from the classical domain decomposition approach in two main points. First, the load balancing is not done before every iteration, but only once at the beginning of computation. Second, the processors do not restrict computations to their local partitions, but refine only within these partitions. Both points lead to reduced communication needs between the processors, and mainly the second point makes it relatively easy to port a sequential software into a parallel one, because necessary code changes are reduced to a minimum.

In section 2 we explain how the domain decomposition is realized in AMDiS, followed by a description of necessary changes in the parallel adaptation loop in section 3. The construction of the global solution is subject of section 4. In section 5 we show some numerical results also for time dependent problems. And finally section 6 contains some conclusions and an outlook to further work.

2 Domain Decomposition

To prepare parallel computations, first the decision must be made, how the work is distributed amongst the given number of processors. Like already mentioned in section 1, the main idea is to do a domain decomposition on a relative coarse mesh, such that the sum of estimated errors in the different domains Ω_i is approximately the same. The coarse mesh can be constructed by a given number of adaptive refinement steps starting from a (very coarse) macro triangulation. These initial refinement steps can be done by one processor which sends the result to all other processes, or they can be done by all processors in parallel. The second approach has the benefit, that no communication has to be done after this phase, because all processors come to the same result in nearly the same time. After domain decomposition each processor i does its calculation on the whole domain, but refines the mesh only within its partition Ω_i^+, which is the domain Ω_i including some overlap with neighboring partitions.

The domain decomposition is computed with help of the parallel graph partitioning and sparse matrix ordering library ParMETIS [5]. ParMETIS first creates the dual graph of the mesh and then partitions this graph considering the error estimates as element weights. When constructing the dual graph, the degree of connectivity among the vertices in the graph can be given, by specifying the number of nodes that two elements at least have to share, so that the corresponding vertices are connected by an edge in the dual graph. For the partitioning of AMDiS meshes this number is set to the dimension of the mesh, because this is the number of common vertices of two neighboring simplices.

Setting this number to one, a dual graph with a higher connectivity is constructed, which can be used for an efficient overlap computation. The overlap is computed on the coarse mesh which is used for the domain decomposition. An overlap of Ω_i of size k includes all elements of the coarse mesh, that have a distance of at most k to any element within Ω_i. Two elements have a distance of 1, if they have at least one common node. A breadth-first search on the dual graph with higher connectivity can be used to determine all elements with distance $d \leq k$ to domain Ω_i.

In Figure 1 the overlap of size 1 for domain Ω_1 on the coarse two dimensional mesh is shown (left hand side). In the middle one can see the global fine mesh after parallel computations, and on the right hand side the corresponding fine mesh of rank 1 is shown. The dashed line at the overlap boundary indicates, that Ω_i^+ is an open domain. The finite element basis functions that are located at this boundary do not belong to partition i, which is important for the partition of unity method.

3 Parallel Adaptation Loop

The adaptation loop keeps nearly the same as in sequential computation. The only thing to do here, is to ensure, that error estimations and refinements are done only on the local partition (including overlap). One goal for the parallel adaptation is to achieve a mesh, which is as similar as possible to that of the sequential computation. To reach this goal, the right refinement strategy has to be chosen. If the strategy depends on global values like the maximal error estimated on Ω, synchronization

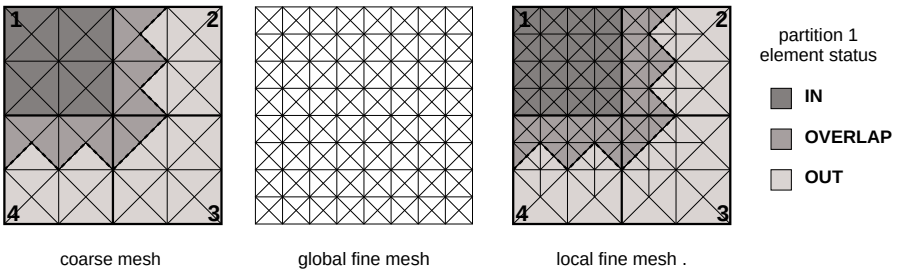


Fig. 1. Overlap of partition 1 on the coarse mesh (left), global fine mesh (middle), and fine mesh of rank 1 (right)

after each iteration is needed to determine and communicate this value. This synchronization could slow down the parallel computation drastically. So we use the equidistribution strategy described in [6] where the refinement depends only on the total error tolerance which is known in advance.

4 Building the Global Solution

After the parallel adaptation loop, every process has computed the solution on the whole domain Ω. But refinements for process $rank$ was only done in Ω^+_{rank}. Out of this region the solution was computed on a very coarse mesh, and so it may not be very accurate. In this section we describe the construction of a final global solution out of the local rank solutions by a parallel partition of unity method. Here each process $rank$ computes a weighted sum of all local solutions on its domain Ω_{rank} using the other solutions u_i on Ω^i_{rank} for all $i \neq rank$. In section 4.1 the concept of *mesh structure codes* is described, which provides an efficient way to exchange information about the binary mesh structure at different processes. Using these information, it is easy to synchronize meshes and exchange local solutions in the overlap regions, what is explained in section 4.2. Finally section 4.3 shows, how the parallel partition of unity is built locally by the single processes.

4.1 Mesh Structure Code

To be able to construct a global solution using the partition of unity method, first the local solutions in overlap regions must be exchanged between the different processes. An easy way to do this, is first to synchronize the meshes within these regions, and than exchange the values of corresponding nodes. The concept of mesh structure codes allows to synchronize the meshes in a very efficient way using the MPI communication protocol.

Refinement in AMDiS is done by bisectioning of simplicial elements. The two new elements are stored as children of the bisected element. So a binary tree arises, where each element has either two children or no children. An inner element is represented by a 1 in the mesh structure code, a leaf element is represented by a 0. Furthermore a unique order of tree elements must be given, which is independent

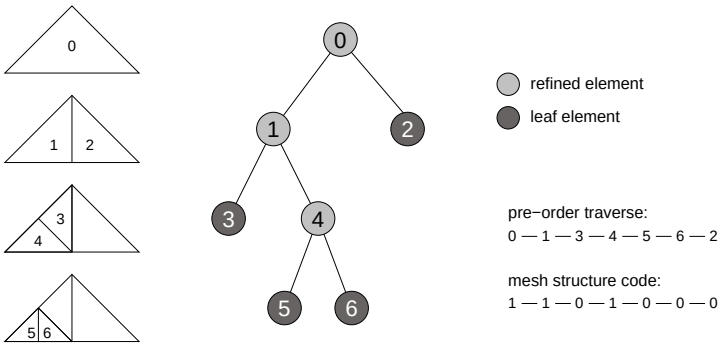


Fig. 2. Mesh structure code of an adaptively refined triangle

of the sequence of adaptive refinements and coarsenings. This order is defined by a pre-order traversal of the tree elements. In Figure 2 the construction of the mesh structure code for an adaptive refined triangle is shown. The resulting binary code in this example can be represented by the decimal value 104. Depending on the size of the binary tree and of the internal integer representation, not one but a vector of integers is necessary to represent the whole tree. To exchange the mesh structures between different processors only one or a few integer values for each macro element has to be sent over MPI. The goal of mesh synchronization is to create the composition of all meshes in overlap regions. To do this, mesh structure codes can be merged very efficiently at binary level. Then the local mesh can be adjusted to this composite mesh structure code.

4.2 Mesh Synchronization and Value Exchange

The process with rank *rank* will construct the global solution within its domain Ω_{rank}. Here fore it needs the local solution u_i on Ω^i_{rank} of every other process $i \neq rank$ which has an overlap with Ω_{rank}. After adapting the local mesh according to the composite mesh structure code the different meshes have common nodes at least in the overlap region. But due to different adaptation sequences in the meshes, these nodes can have different indices, which makes the value exchange difficult. A unique node order can be constructed by sorting the nodes lexicographically in ascending order by their coordinates. So for every other rank i a sending order is created on Ω_i^{rank} and a receiving order on Ω^i_{rank}. To avoid serialization in the MPI communication, first a process sends in a non blocking way to all other processes, and after that receives in a blocking way from the other processes.

4.3 Parallel Partition of Unity

Now process *rank* has all relevant informations to construct the global solution on Ω_{rank} by a partition of unity (see [1]), which is defined by a weighted sum over all local solutions u_i:

$$u_{PU}(x) := \sum_{i=1}^{numRanks} \gamma_i(x) u_i(x) \quad \forall x \in \Omega \tag{1}$$

where $\sum_{i=1}^{numRanks} \gamma_i(x) = 1$ for all $x \in \Omega$. We set $\gamma_i(x) := \frac{W_i(x)}{\sum_{j=1}^{N} W_j(x)}$ with $W_i(x) := \sum_{\phi \in \Phi_i^c} \phi(x)$ where Φ_i^c is the set of linear coarse grid basis functions within Ω_i^+. The partition of unity now is evaluated at all coordinates where fine grid basis functions of Ω_i are located. In [3] an upper bound for the error in H^1 semi norm resulting from the partition of unity is given. Assume $u \in H^2(\Omega)$, then $\|u - u_{PU}\|_{H^1} \leq C(h + H^2)$, where h is the maximal edge size of mesh i in Ω_i and H is the maximal edge size of mesh i in $\Omega \setminus \Omega_i$. In particular, if $h \leq \sqrt{H}$:

$$\|u - u_{PU}\|_{H^1} \leq C(h)\,. \tag{2}$$

5 Numerical Results

In this chapter we present two examples that demonstrate the functionality of the parallelization approach. In section 5.1 the Poisson equation is solved on the unit square. Section 5.2 shows an application in the field of image processing and provides an extension of the approach to time dependent problems.

5.1 Poisson Equation

We solve the Poisson equation

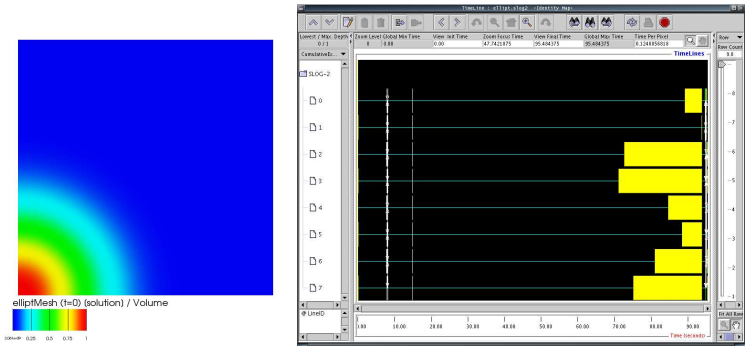
$$-\nabla u(x) = f(x) \quad \forall x \in \Omega \tag{3}$$

$$u(x) = g(x) \quad \forall x \in \partial\Omega \tag{4}$$

with $\Omega = [0,1] \times [0,1]$, $f(x) = -(400x^2 - 40)e^{-10x^2}$, and $g(x) = e^{-10x^2}$. The analytical solution is $u(x) = e^{-10x^2}$, shown in Figure 3 a). In Figure 3 b) the time lines of a parallel computation with eight processors are shown. In this case a speedup of 5.8 to the serial computation was reached. The optimal speedup is not reached due to the overhead of parallel initialization at the beginning and the partition of unity at the end, as well as by a certain load imbalance during the parallel adaptation loop. To get an impression of the error that arises due to the partition of unity, we compared the solution after parallel computation with the true analytical solution and computed the pointwise error, the error in L^2 norm, and the error in the H^1 semi norm. Afterwards we synchronized the meshes on the whole domain Ω and solved the problem on this global mesh again on one single processor. In Table 1 the measured errors for the single cases are listed. The final solve step reduced the pointwise and L^2 error by about one order of magnitude. However the error in H^1 semi norm is reduced only by about ten percent.

Table 1. Errors for a computation with 4 processors before and after a final solve step

	pointwise error	L^2 error	H^1 error
before final solve	$2.726 \cdot 10^{-2}$	$6,207 \cdot 10^{-5}$	$3.508 \cdot 10^{-3}$
after final solve	$1.120 \cdot 10^{-3}$	$6.418 \cdot 10^{-6}$	$3.327 \cdot 10^{-3}$



a) b)

Fig. 3. Analytical solution of the Poisson equation and time lines of a parallel computation with 8 processors

5.2 Perona-Malik Denoising

We now extend the approach to time dependent problems. For this we use the Perona-Malik equation (see [4]) to reduce the noise level in a monochrome image. The gray values of the image are interpreted as height field. A dogma in image processing is that images are of high interest where the gradient of this height field is large. So the Perona-Malik equation smooths regions with a small gradient and sharpens regions with a large gradient. The equation reads

$$u_t = div(g(|\nabla u|)\nabla u) \tag{5}$$

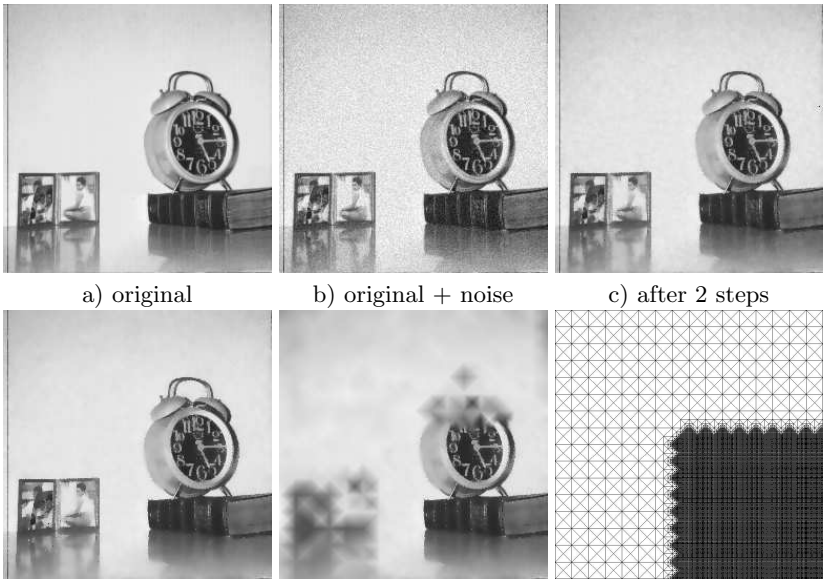
with

$$g(s) := e^{-(\frac{s}{2\lambda})^2} \quad . \tag{6}$$

The parameter λ determines the smoothing/sharpening properties. The time step size determines the degree of denoising. In this example we do not use adaptive mesh refinement, but use a fixed hierarchical mesh, which on the finest level represents the resolution of the image. The domain decomposition and overlap computation is done on a certain lower level of the mesh. After each time step a partition of unity of the local rank solutions was computed. As initial solution we added random noise of the interval $[-30, 30]$ to a picture with gray values between 0 (black) and 255 (white). The picture domain is $\Omega = [0, 1] \times [0, 1]$. The parameter λ was set to 3000 and the time step size is 10^{-4}. In Figure 4 a) the original in b) the noised picture is shown. Figure 4 c) shows the denoising result after two time steps, 4 d) after four time steps. To illustrate the view of one processor, in Figure 4 e) the local solution of rank 4 after time step 4 is shown. In Figure 4 f) the corresponding local mesh is presented.

6 Conclusions and Outlook

The concepts presented in this paper, allowed a parallelization of our finite element software AMDiS with a comparatively small amount of redesign and reimplementation. With the ParMETIS library domain decomposition was done efficiently and



a) original b) original + noise c) after 2 steps

d) after 4 steps e) view of rank 4 f) mesh of rank 4

Fig. 4. Parallel denoising of a monochrome picture

the concept of mesh structure codes enabled an easy way of mesh synchronization. An aspect of future work is the treatment of time dependent problems with adaptive refinements.

References

[1] I. Babuska and J. M. Melenk. The partition of unity method. *Internat. J. Numer. Methods Engrg.*, 40:727–758, 1997.

[2] R.E. Bank and M. Holst. A new paradigm for parallel adaptive meshing algorithms. *SIAM Rev.*, 45(2):291–323, 2003.

[3] M. Holst. Applications of domain decomposition and partition of unity methods in physics and geometry. *Proceedings of the Fourteenth International Conference on Domain Decomposition Methods*, pages 63–78, 2002.

[4] P. Perona and J. Malik. Detecting and localizing edges composed of steps, peaks and roofs. Technical Report UCB/CSD-90-590, EECS Department, University of California, Berkeley, 1990.

[5] K. Schloegel, G. Karypis, and V. Kumar. Parallel static and dynamic multiconstraint graph partitioning. *Concurrency and Computation: Practice and Experience*, 14:219–240, 2002.

[6] A. Schmidt and K.G. Siebert. *Design of Adaptive Finite Element Software*, volume 42 of *LNCSE*. Springer, 2005.

[7] S. Vey and A. Voigt. ADMiS: adaptive multidimensional simulations. *Comput. Vis. Sci.*, 10(1):57–67, 2007.