Automatic Domain Decomposition for a Black-Box PDE Solver

Torsten Adolph and Willi Schönauer

Institute for Scientific Computing, Forschungszentrum Karlsruhe,
Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany
{torsten.adolph,willi.schoenauer}@iwr.fzk.de

Summary. We want to develop a tool to distribute in a black-box environment automatically a given arbitrary 2-D or 3-D mesh equally among a given number of processors. First we sort the nodes by their x-coordinate locally on each processor, afterwards globally by a sophisticated algorithm where we make use of the message passing paradigm. This results in a one-dimensional domain decomposition that may also run over dividing lines. The elements are sent around in a ring shift afterwards, where each processor takes the necessary element information out of the current basket in each tact. To be able to set up the linear system of equations resulting from the discretization purely local without communication, we also create an overlap, i.e. we also store on each processor the necessary node and element information of neighbouring processors. The re-sorting of refined meshes serves the purpose of load balancing, and for the resulting matrix it also serves as bandwidth optimizer.

1 Introduction

Basically, there are three main methods for the numerical solution of nonlinear PDEs, cf. [Langtangen, 2003, Larsson and Thomée, 2003]: the finite difference method (FDM) that dominated the early development of numerical analysis of PDEs, the finite element method (FEM) that has been introduced by engineers in the 1960s and that has become the mostly used numerical method for PDEs over the last decades, and the finite volume method that is between the FDM and the FEM (widely used in CFD). Furthermore, there is the boundary element method, but this method is applicable only to linear systems of PDEs. We combine the advantages of the first two methods as we use a FDM on an unstructured FEM mesh which we call Finite Difference Element Method (FDEM). The FEM mesh is only needed for the structure of the space, after we have collected the neighbour nodes for each grid node, we do not need the elements any more, and thus we have a meshfree method.

The organization of this paper is as follows: In Sect. 2 we give a short survey of FDEM. There we describe the generation of the difference and error formulas and the computation of the error estimate. The parallelization of

the FDEM progam package necessitates the re-sorting of the mesh which results in an automatic domain decomposition that we explain in Sect. 3. To demonstrate the usefulness of our method, we give a 2-D and a 3-D example in Sect. 4, before we finally conclude.

2 The Finite Difference Element Method

We want a robust black-box solver to solve nonlinear systems of elliptic and parabolic PDEs in 2-D and 3-D with arbitrary nonlinear boundary conditions (BCs) where we use an unstructured mesh on an arbitrary domain. The domain may be composed of several subdomains with different systems of PDEs. Then the solutions of the subdomains are coupled by coupling conditions over the dividing lines. Together with the solution we compute a reliable error estimate that we also use for the order control and for a local mesh refinement. Considered together, these properties cannot be found in any other code for the numerical solution of PDEs.

We discuss the solution method in 2-D, the extension to 3-D is straightforward, see [Schönauer and Adolph, 2005]. We abbreviate the PDE and BC operator for the unknown solution $u(t, x, y)$ as follows:

$$Pu \equiv P(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{yy}, u_{xy}) = 0 \ , \tag{1}$$

where u and Pu are vectors with l components (system of l PDEs). If we include t and u_t the system is parabolic, otherwise it is elliptic.

A basic paper on FDEM is [Schönauer and Adolph, 2001], a progress report is [Schönauer and Adolph, 2003]. A detailed report is available online, see [Schönauer and Adolph, 2005].

2.1 The Generation of Difference and Error Formulas

For the generation of the difference and error formulas we make a polynomial approach in x,y of order q which then will be the consistency order:

$$P_q(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + \cdots + a_{m-1} y^q \ . \tag{2}$$

We need $m = (q+1)(q+2)/2$ nodes to determine the m unknown coefficients a_0 to a_{m-1}.

In order to get explicit difference formulas we make use of the principle of the influence polynomials. For a node i the influence polynomial $P_{q,i}$ of order q is defined by

$$P_{q,i}(x, y) = \begin{cases} 1 \text{ for } (x_i, y_i) \\ 0 \text{ for } (x_j, y_j),\ j \neq i \ . \end{cases} \tag{3}$$

In the FEM, the basis functions are always defined on a unit element. The shape of an element is prescribed, and on this shape the nodes are prescribed

according to the order. In FDEM there is no shape. The nodes are arbitrarily distributed in the space. When the nodes are selected, it is a meshfree method.

With the influence polynomials the discretized solution which we denote by u_d (the index d means "discretized") can be represented by

$$u_d(x,y) := P_q(x,y) = \sum_{i=0}^{m-1} u_i P_{q,i}(x,y) \ . \tag{4}$$

By the evaluation of $P_{q,i}$ for a grid point $x = x_j, y = y_j$, we obtain the coefficients of an interpolation polynomial at a node j. The difference formulas are the partial derivatives of (4). E.g. for u_x we get

$$u_{x,d} := \frac{\partial P_q(x,y)}{\partial x} = \sum_{i=0}^{m-1} u_i \frac{\partial P_{q,i}(x,y)}{\partial x} \ . \tag{5}$$

One of the most critical sections is how we choose the m nodes on an unstructured FEM mesh. The nodes are collected in rings (2-D) or balls (3-D), respectively, around the central node, see Fig. 1. For reasons of simplicity, we also speak of "rings" in 3-D. We use logical masks to get the next neighbour ring of a given ring from the element list (gives nodes of an element) and the inverted element list (gives elements in which a node occurs). We do not only

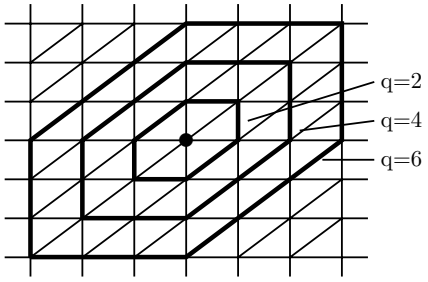

Fig. 1. Illustration for ring search

collect m nodes up to the consistency order q but $m + r$ nodes up to order $q + \Delta q$ because there may be linear dependencies on straight lines. A second criterion is that we collect at least $q + 2$ rings (because of the error formula). This results in $m + r$ equations for the m coefficients. We want to have nodes in the difference stars that are close to the central node. Therefore, we arrange the equations according to the ring structure and allow the crossing of a ring limit only if the current pivot element $|pivot| \leq \varepsilon_{pivot}$. The parameters Δq and ε_{pivot} determine the quality of the difference and error formulas and therefore are the key for the whole solution process. As we must determine m influence polynomials that generate the unit matrix as the right hand sides, we must invert the matrix in reality, see [Schönauer and Adolph, 2001, (27), (28)].

2.2 The Discretization Error Estimate and the Error Equation

As we have formulas of arbitrary order q, we get an easy access to the estimate of the discretization error, e.g. for the derivative u_x

$$d_x = u_{x,d,q+2} - u_{x,d,q} \tag{6}$$

$$d_{exact} = u_x \qquad - u_{x,d,q} \tag{7}$$

where $u_{x,d,q}$ denotes the difference formula of order q, i.e. the discretization error is defined by the difference to the order $q+2$. In (7) we show the exact discretization error and we see that the derivative is replaced by a "better" formula for the estimate which holds only for sufficiently fine grid. Equation (6) is the key for our explicit error access. This estimate has a built-in self-control: If the higher order formula is not essentially better, we get a large error estimate that shows the failure of the method.

Pu (1) is an arbitrary nonlinear function of its arguments. Therefore, we linearize system (1) with the Newton-Raphson method and then discretize the resulting linear Newton-PDE by replacing e.g. for the derivative u_x

$$u_x \Leftarrow u_{x,d} + d_x \ , \tag{8}$$

and analogously for the other derivatives. After linearizing also in the discretization errors, we finally get the error equation for the overall error Δu_d:

$$\begin{aligned} \Delta u_d = & \quad \Delta u_{Pu} + \Delta u_{D_t} + \Delta u_{D_x} + \Delta u_{D_y} + \Delta u_{D_{xy}} \text{ (level of solution)} \\ & = Q_d^{-1} \left[(Pu)_d \ + \ D_t \ + \ \{ D_x \ + \ D_y \ + \ D_{xy} \} \right] \ . \text{ (level of equation)} \end{aligned} \tag{9}$$

Here Q_d denotes the large sparse matrix resulting from the discretization. The inverse Q_d^{-1} is never explicitly computed as it is a full matrix. Instead, the system is solved iteratively. $(Pu)_d$ is the discretized Newton residual and the D_μ are discretization error terms that result from the linearization in the d_μ, e.g.

$$D_x = \frac{\partial Pu}{\partial u_x} \, d_x + \frac{\partial Pu}{\partial u_{xx}} \, d_{xx} \ . \tag{10}$$

The Newton correction Δu_{Pu} is computed from

$$Q_d \Delta u_{Pu} = (Pu)_d \ . \tag{11}$$

The other error terms in the first row of (9) are only used for the error control. If we applied these terms, we had no error estimate any more. This approach also implies that we can explicitly follow the effect of a discretization error to the level of solution.

3 Automatic Domain Decomposition

We have a black-box for the PDEs and for the domain, i.e. the user may solve any system of elliptic or parabolic PDEs in 2-D or 3-D on any unstructured

FEM grid that may also consist of several subdomains coupled by coupling conditions. Then the subdomains are separated by internal boundaries (dividing lines) over which we must not differentiate, see Fig. 2(a). However, the program code must be efficiently parallelized for distributed memory parallel computers with MPI because the numerical solution of large PDE problems needs much computation time and memory. This must even be fulfilled after the mesh has been refined locally several times, so that there are elements of many different refinement stages, see Fig. 2(b) for a part of a refined mesh with elements of 4 different refinement stages.

The solution is as simple as effective: we re-sort the nodes and elements stored on the processors globally by x-coordinate, and we also store on the processors necessary node and element information of nodes and elements that are owned by neighbour processors, which results in an automatic 1-D domain decomposition with overlap, see Fig. 2(a) and Fig. 5 below.

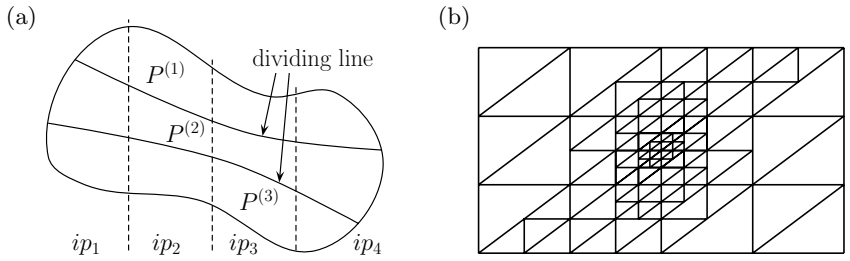

Fig. 2. (a) Domain of solution with 2 dividing lines and illustration of 1-D domain decomposition on $np = 4$ processors; **(b)** grid after 3^{rd} refinement

3.1 Global Mesh Re-sorting Algorithm

Why is it necessary to re-sort the mesh? From Fig. 3 we learn that we need nodes of all 4 processors for the difference star of node 25_4 (Fig. 3(a)), i.e. we either need much communication to generate the difference formulas or we have to store nodes of all processors on each processor. As both is not an efficient way to parallelize a program code we re-sort the nodes so that we only need nodes of the neighbour processor(s) for the difference star of this node (now node 22_3), see Fig. 3(b). The node re-sorting algorithm also performs an efficient load balancing across the processors, especially after the mesh has been refined locally.

We use a parallel global re-sort algorithm where we first sort the nodes locally on each processor by x-coordinate. For the global re-sorting we need $2(np-1)$ steps on np processors, for illustration see Table 1, and there are up to $np/2$ processors active in parallel.

In step 1 processor 1 sends its sorted nodes to the right neighbour processor 2. Processor 2 merges the two sorted node lists in step 2 and sends the first

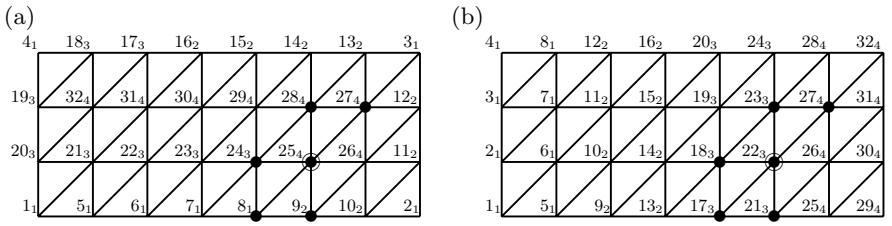

Fig. 3. Illustration of node numbering on $np = 4$ processors, index 1–4 means processor number: **(a)** numbering by mesh generator; **(b)** numbering after global re-sorting of the nodes by x-coordinate

part to processor 3. In step 3 processor 3 merges the nodes and sends the first part to processor 4, whereas processor 2 sends the second part of its nodes to processor 3. This is continued up to step $np - 1$ when processor $np = 8$ receives new nodes for the first time. In the second half of the algorithm the first $np - 1$ processors proceed the same way as before, and processor np merges the received nodes with its own nodes and sends the part with the lowest x-coordinates to the current target processor (from 1 to $np - 1$) in each step.

Table 1. Illustration of the global node re-sorting algorithm on $np = 8$ processors

	Step													
Proc.	1	2	3	...	7	8	9	...	13	14				
1	S					R								
2	R	M S	S				R							
3		R	M S R											
4			R		S									
5					M S R	M S	S							
6					M S R	M S R	M S R		R					
7					M S R	M S R	M S R		S	R				
8					R	M S R	M S R		M S R	M S				

M: Merge, S: Send, R: Receive

The elements and boundary nodes are read from the mesh file in parallel, and each processor puts its data into a basket. These baskets are sent around in a ring-shift in np tacts, and in each tact each processor takes out the data it needs, see Fig. 4. For the elements we need 2 ring-shifts as we first have to determine the owning processor of each element (an element is owned by the processor that owns the leftmost node of this element).

For the boundary nodes we compare the received boundary node numbers to the node numbers of the own nodes in each tact. Matching node numbers are stored on the processor, non-matching nodes are sent to the next neighbour processor. For the (sliding) dividing line nodes we proceed the same way.

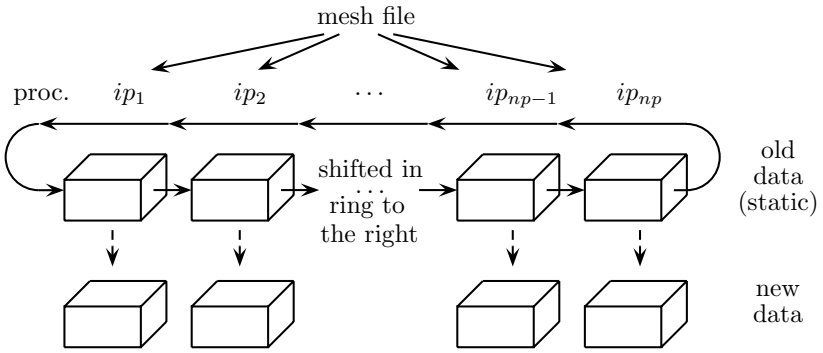

Fig. 4. Distribution of the elements and boundary nodes

3.2 Generation of Overlap

As we want to execute the computation of the large sparse matrix Q_d and the r.h.s. $(Pu)_d$ purely local without communication, we also have to store on a processor ip the necessary node and element information of its left and right neighbour processor(s). This is indicated as overlap, see Fig. 5.

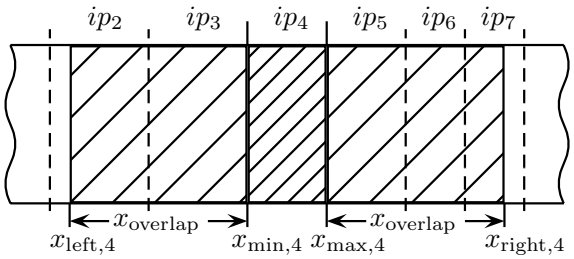

Fig. 5. Illustration of the overlap

The width of the overlap $x_{overlap}$ depends on the consistency order q and the mean edge length of the grid and is computed so that two criterions are fulfilled: we must collect enough nodes and enough rings. Therefore, an overlap may also extend over several processors but we do not necessarily store the information of all nodes and elements of an overlap processor. Furthermore, the number of overlap processors on the left and right side may be different, only the width of the overlaps is always the same, see Fig. 5.

When the data with the overlap has been distributed to the processors, each processor generates its part of the matrix Q_d and of the r.h.s. $(Pu)_d$ without communication. The global dependency is made by the numerical solution of the linear system of equations for which we need message passing again.

4 Numerical Examples

To demonstrate the usefulness of the FDEM, we present two examples in this section. Both are problems for which we got the PDEs from an academic partner. The computations have been carried out on the HP XC4000 with 2.6 GHz AMD Opteron processors and InfiniBand 4X interconnect at the University of Karlsruhe, Germany.

4.1 The Numerical Simulation of a Microreactor

We simulate numerically the mixing and the chemical reactions in a microreactor. Here a laminar jet enters from a pipe, perpendicular to the main flow in a channel. One chemical component enters through the main channel, another one through the pipe. They react and produce a third chemical component.

We have six variables, as there are two velocity components, the pressure and three chemical components. So we need a system of six PDEs: global continuity equation, two momentum equations for the mixture, two continuity equations for the entering chemical components, and Dalton's law.

In order to get a mean relative estimated error in the 1% region, we have to use a very fine grid. We use a grid with 2561×641 nodes in the channel and 161×321 nodes in the pipe, resulting in 1,693,121 nodes. We compute with consistency order $q = 4$, and we use 128 processors.

The most interesting result is the distribution of the reaction product. The maximum of the global relative estimated error is 0.75 (75%), the mean error is 0.19%, i.e. the maximum error only occurs locally. The CPU time for the master processor 1 is 17.6 h. Due to the limited space, we refer to [Adolph and Schönauer, 2007b] for further details.

An interesting by-product of the global node re-sorting is that it works as a bandwidth optimizer. For this example, we have 10,158,726 unknowns which therefore is the bandwidth of the full matrix if we define the bandwidth as the difference from the main diagonal to the outermost non-zero element. Before the node re-sorting, the bandwidth of the resulting large sparse matrix Q_d is 10,096,384, after the re-sorting the bandwidth is 24,697, i.e. about 0.24% of the original bandwidth. By the SSP bandwidth optimizer [Zundel and Schönauer, 2001] that has been developed by ourselves and that is an improved Cuthill-McKee algorithm, we get a bandwidth of 9,663 so that it is safe to say that our "global node re-sorting" bandwidth optimizer is quite good.

4.2 The Heat Conduction in a Power Module with 6 Power Chips

The simulation of the temperature in a power semiconductor module is a 3-D parabolic problem with two subdomains that are coupled by a sliding dividing line (interior boundary over which we must not differentiate, non-matching grids, see [Schönauer and Adolph, 2001]). In the upper subdomain, we have six power chips that have the same power dissipation of 250 W/Chip. On the

bottom surface, cooling is applied, either by a convective liquid or gas (air) stream. We want to compute the temperature distribution on the top surface after 50 sec.

As we have only the temperature as variable, we only need one PDE, the heat conduction equation. This is a linear PDE, but the boundary condition for the cooling at the bottom side is nonlinear.

We use a grid with $237 \times 117 \times 9$ nodes in the upper subdomain, and $119 \times 59 \times 9$ nodes in the lower subdomain. We compute with consistency order $q = 4$, and we use 32 processors.

The maximum of the global relative estimated error is 0.78% in the upper subdomain and 0.22% in the lower subdomain. The mean errors in the two subdomains are 0.02% (upper) and 0.004% (lower), respectively. The CPU time for master processor 1 is 42.0 h for 73 time steps. Again, we refer to [Adolph and Schönauer, 2007a] for further details due to the limited space.

For this example, we have 312,750 unknowns which is the bandwidth of the full matrix. Before the node re-sorting, the bandwidth of the resulting large sparse matrix Q_d is 306,502, after the re-sorting the bandwidth is 11,037, i.e. about 3.6% of the original bandwidth. By the SSP bandwidth optimizer, we get a bandwidth of 5,607 (1.8% of 306,502).

We also carried out a scalability test for this problem. We repeated the computation with the same problem size on 64, 128, 256 and 512 processors. This is not the usual way to examine the scalability of a code but because of the mandatory LU preconditioning it is impossible to perform scalability tests by increasing the problem and the number of processors uniformly as usual, see [Adolph and Schönauer, 2007a].

The CPU time on 64 processors is 24.2 h, on 128 processors it is 11.0 h, on 256 processors it is 5.9 h, and on 512 processors it is 6.4 h. The computation time is reduced roughly by the factor 2 if we double the number of processors. For 512 processors, the communication overhead strongly affects the computation time. As more than 99% of the computation time is comsumed by the linear solver LINSOL, there is still space for improvement.

5 Concluding Remark

We have developed a black-box PDE solver that is able to solve any nonlinear system of elliptic or parabolic PDEs on any unstructured domain in 2-D and 3-D that may even consist of several subdomains with different PDEs. Together with the solution we compute an error estimate which is a unique feature for such a general black-box. To set up the resulting linear system of equations purely local, we re-sort the nodes globally by x-coordinate. Furthermore, we also store necessary node and element information of neighbour processors on each processor which results in an automatic 1-D domain decomposition with overlap and built-in bandwidth optimizer. Above all, the

FDEM program package is efficiently parallelized on distributed memory parallel computers by the means of MPI.

As the use of such a code is only possible by well-trained experts, we offer a service: In academic or industrial cooperations, the partner gives us his PDEs, and we, the experts, solve them for him. Up to now, we were able to solve all systems of PDEs we were confronted with.

References

Torsten Adolph and Willi Schönauer. The snuffle problem Denev for the numerical simulation of a microreactor. Forschungszentrum Karlsruhe, 2007a. URL `http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/snuffle-denev.pdf`.

Torsten Adolph and Willi Schönauer. The snuffle problem Gerstenmaier for the heat conduction in a power module with 6 power chips. Forschungszentrum Karlsruhe, 2007b. URL `http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/snuffle-gerstenmaier.pdf`.

Hans Petter Langtangen. *Computational Partial Differential Equations*, volume 1 of *Texts in Computational Science and Engineering*. Springer-Verlag, Berlin Heidelberg New York, 2nd edition, 2003.

Stig Larsson and Vidar Thomée. *Partial Differential Equations with Numerical Methods*, volume 45 of *Texts in Applied Mathematics*. Springer-Verlag, Berlin Heidelberg New York, 2003.

Willi Schönauer and Torsten Adolph. How WE solve PDEs. *Journal of Computational and Applied Mathematics*, 131:473–492, 2001.

Willi Schönauer and Torsten Adolph. FDEM: How we make the FDM more flexible than the FEM. *Journal of Computational and Applied Mathematics*, 158:157–167, 2003.

Willi Schönauer and Torsten Adolph. FDEM: The evolution and application of the Finite Difference Element Method (FDEM) program package for the solution of partial differential equations. Universität Karlsruhe (TH), 2005. URL `http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/fdem.pdf`.

Detlev Zundel and Willi Schönauer. A fast "parallelized" single pass bandwidth optimizer for sparse matrices. In *Proceedings of the International Conference on Numerical Algorithms, Marrakech, Morocco, October 1–5, 2001*, 2001. URL `http://www.rz.uni-karlsruhe.de/rz/docs/LINSOL/Literatur/bo-nasv.ps.gz`.