THE FINITE DIFFERENCE ELEMENT METHOD (FDEM) WITH EXAMPLES AND ERROR ESTIMATES

TORSTEN ADOLPH† AND WILLI SCHÖNAUER†

Abstract. FDEM is a black-box solver that solves by a finite difference method arbitrary nonlinear systems of elliptic and parabolic PDEs (partial differential equations) on an unstructured FEM grid in 2-D or 3-D. The FEM grid serves only for the structuring of the space, i.e. the determination of the neighboring nodes. In 2-D we use triangles, in 3-D tetrahedrons. For each node we generate difference formulas of consistency order q with a sophisticated algorithm. By the use of formulas of order $q+2$, an estimate of the discretization error is obtained. An unprecedented feature for such a general black-box is the error estimate that is computed together with the solution. We present four applications for the FDEM program package. They arise in different fields of applications, but for all problems the error estimate shows the quality of the solution. For all these examples it would be very difficult to obtain a quality control of the solution by conventional grid refinement tests.

Key words. finite difference method, unstructured grid, nonlinear PDEs, error estimate, mesh refinement, black-box solver, parallelization

AMS subject classifications. 65M06, 65N06, 65G99

1. Introduction. The most natural way to solve PDEs numerically is the FDM (finite difference method). Here, derivatives are simply replaced by difference formulas, i.e. by a polynomial approach. Therefore, this was the preferred and original method. However, the classical difference formulas are 1-D formulas in x, y, z and therefore need a rectangular grid that restricts essentially the geometrical flexibility. For this reason, the FEM (finite element method) has become the most popular method. Here, by the "triangularization" of the space, full geometrical flexibility is obtained. However, the PDEs are not solved directly, but in a form where the PDE is multiplied by a test function and integrated over the domain. In the integral, derivatives are shifted from the solution to the test function by partial integration. This results in a "weak" solution. The FEM needs a complicated theory for each type of PDE.

The dream is to have a FDM with the full geometrical flexibility of the FEM. This dream has become true in the FDEM (finite difference element method) that is a FDM on an unstructured FEM mesh. From the element list and its inverted list, we select nodes in rings around the central node of the difference formula. Because there are linear dependencies on straight lines, we select more nodes than are needed for the chosen polynomial order q. A sophisticated algorithm has been developed to select from this set of nodes those nodes that result in the best difference formulas of consistency order q. Once the nodes are selected, our method could be called "mesh-free" because then the mesh that has only been necessary for finding neighboring nodes is no longer used. From the difference of formulas of order q and $q+2$, we get an estimate of the discretization error. The explicit character of the FDM allows the discretization including the error estimates which results in the error equation that is solved together with the computation of the solution.

Up to now PDEs have not been solved by black-box solvers that include an error estimate: Happy are those people that do not see the errors. In the FEM an explicit

†Karlsruhe Institute of Technology, Forschungszentrum Karlsruhe GmbH, Steinbuch Centre for Computing, Hermann-von-Helmholtz-Platz 1, 76344 Eggenstein-Leopoldshafen, Germany ({torsten.adolph, willi.schönauer}@iwr.fzk.de).

error estimate for a black-box solver is not possible. Therefore, only mesh refinement can be used to test the accuracy which is not possible if the problem itself goes to the limit of the available computer. Our error estimate makes directly visible the problems of the numerical solution. The examples below will demonstrate the invaluable benefit and insight that results from the error estimate. Therefore, FDEM is a unique method for the numerical solution of arbitrary nonlinear systems of PDEs.

Other useful properties of the FDEM program are: The knowledge of the error allows a local mesh refinement to obtain a requested accuracy, it allows to balance all errors in space and time and it is used for the stopping of the Newton-Raphson iteration. FDEM is a monolithic code that is efficiently parallelized on distributed memory parallel computers.

However, the use of such a code is only possible by well-trained experts. Therefore, our intention is not to sell the code but to offer a service where the partner gives us his PDEs and we, the experts, solve them for him, much better than his own people could do it. The ideal numerical simulation is the cooperation of the engineer or physicist at the one side and the computer scientist at the other side. This is our philosophy.

2. The FDEM Code. We want to solve nonlinear elliptic and parabolic systems of 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. The solutions of the subdomains are coupled by coupling conditions (CCs). We want a robust black-box solver with a reliable error estimate that we also use for the order control and for a local mesh refinement.

We discuss the solution method in 2-D; the extension to 3-D is too extensive so that we refer to [8] for details. The most general operator that we admit for the PDEs and BCs in 2-D, with the unknown solution $u(t, x, y)$, has the form

$$Pu \equiv P(t, x, y, u, u_t, u_x, u_y, u_{xx}, u_{yy}, u_{xy}) = 0 \tag{2.1}$$

where u and Pu are vectors with l components (system of l PDEs), if the variables are e.g. u, v, p, we have $l = 3$. If we include t and u_t, the system is parabolic, otherwise it is elliptic.

A basic paper on FDEM is [6], a progress report is [7]. A detailed report is available online, see [8].

2.1. The Generation of Difference and Error Formulas. For the generation of the difference and error formulas, we make use of a finite difference method of consistency order q which means local approach of the solution u by a polynomial of order q. The 2-D polynomial of order q is

$$P_q(x, y) = a_0 + a_1 x + a_2 y + a_3 x^2 + a_4 xy + a_5 y^2 + a_6 x^3 + \cdots + a_{m-1} y^q. \tag{2.2}$$

This polynomial has m coefficients a_0 to a_{m-1} where $m = (q + 1)(q + 2)/2$. For the determination of these m coefficients, we need m nodes with coordinates (x_0, y_0) to (x_{m-1}, y_{m-1}). For example, for $q = 2$ we need $m = 6$ nodes.

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

$$P_{q,i}(x, y) = \begin{cases} 1 \text{ for } (x_i, y_i) \\ 0 \text{ for } (x_j, y_j),\ j \neq i. \end{cases} \tag{2.3}$$

This means that the influence polynomial $P_{q,i}$ has function value 1 in node i and 0 in the other $m-1$ nodes. Then the discretized solution u which we denote by u_d (the index d means "discretized") can be represented by

$$u_d(x,y) := P_q(x,y) = \sum_{i=0}^{m-1} u_i \cdot P_{q,i}(x,y). \tag{2.4}$$

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 the interpolation polynomial P_q, i.e. we have to differentiate (2.4). For example, for the difference formula for u_x which we denote by $u_{x,d}$ we get

$$u_{x,d} := \frac{\partial P_q(x,y)}{\partial x} = \sum_{i=0}^{m-1} u_i \cdot \frac{\partial P_{q,i}(x,y)}{\partial x}. \tag{2.5}$$

The other difference formulas are computed analagously, so we use $\partial^2 u_d/\partial x^2$ for $u_{xx,d}$ or $\partial^2 u_d/\partial x \partial y$ for $u_{xy,d}$.

In order to use these formulas, we have to determine the coefficients of the influence polynomials $P_{q,i}$. Therefore, we put into (2.2) the coordinates (x_j, y_j) of the m surrounding nodes of the central node. So each of these m nodes creates one equation and one r.h.s., and we get m linear systems of equations for each node of the mesh. If we denote by M the coefficient matrix and by A the matrix where we have in the i^{th} column the coefficients of the i^{th} influence polynomial $P_{q,i}$, we can write

$$M \cdot A = I \tag{2.6}$$

with

$$M = \begin{pmatrix} 1 & x_0 & y_0 & x_0^2 & x_0 y_0 & y_0^2 & \cdots & y_0^q \\ 1 & x_1 & y_1 & x_1^2 & x_1 y_1 & y_1^2 & \cdots & y_1^q \\ \vdots & & & & & & & \vdots \\ 1 & x_{m-1} & y_{m-1} & & \cdots & & & y_{m-1}^q \end{pmatrix}. \tag{2.7}$$

We get this relation by writing out (2.2) for the m influence polynomials. The solution of (2.6) is

$$A = M^{-1} \tag{2.8}$$

which means that the coefficients of the i^{th} influence polynomial are in the i^{th} column of the matrix M^{-1}.

But in order to be able to form the matrix M we have to choose m nodes out of all surrounding nodes of the evaluation node. As we want to have good difference and error formulas to get a good solution and error estimate, this is one of the most critical sections in the whole solution process. On the one hand, we want the m nodes to be as close as possible around the evaluation node because we want to have local information in the interpolation polynomial. Wider difference stars would introduce false information if the function values change rapidly. Furthermore, they would increase the bandwidth of the resulting large sparse matrix Q_d for the solution of our PDEs what would lead to a larger storage requirement and higher computing time. On the other hand, we also need nodes that are farther away in order to get information about the solution that closer nodes cannot give.

The FEM mesh is generated by a commercial mesh generator. When the grid file is read in, the node numbers of the elements, which consist of three nodes in 2-D, are stored in the element list (gives nodes of an element). The nodes are collected in rings around the central node. The inverted element list (gives elements in which a node occurs) gives us the elements the central node belongs to. From the element list we get all nodes that belong to these elements. With the help of a logical mask (to exclude already collected nodes) we get the first ring. For the following rings, this procedure is repeated for the nodes of the current ring until we have collected enough nodes.

As we have seen above, we only need m nodes for the determination of the matrix M (2.7) because we have m coefficients in each of the m influence polynomials. But then there is the risk that the matrix could become singular if some of the m nodes are linearly dependent, e.g. on a straight line. This occurs easily for a rectangular grid. So we do not search only for nodes up to order q. Instead we search for nodes up to order $q+\Delta q$ where Δq is the so-called surplus order. Δq must be at least equal to 2 because the error formulas we generate are formulas of order $q+2$. So usually we set $\Delta q = 4$ because we also need additional nodes for the error formulas in order to avoid linear dependencies.

Another criterion that the collected nodes must fulfil is that we have to collect enough rings. Therefore, we collect $q+2$ rings around each node because on a rectangular grid we must have $q+2$ rings for the order of the error formula. The number of surplus nodes we have collected is denoted by r.

To get "normalized" equations, the $m+r$ nodes are transformed to the square between -1 and $+1$ in x- and y-direction. 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 afterwards we normalize to absolute row sum equal to 1. We execute the Gauss-Jordan algorithm with row pivoting for the computation of the inverse M^{-1} (2.8) 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.

2.2. The Estimate of the Discretization Error. In Subsect. 2.1 we explained how we generate difference formulas of arbitrary consistency order q in space. By these formulas, we get an easy access to the estimate of the discretization error. If we denote e.g. the difference formula of order q for the derivative u_x by $u_{x,d,q}$ (index d means "discretized"), it holds

$$u_x = u_{x,d,q} + \bar{d}_{x,q} = u_{x,d,q+2} + \bar{d}_{x,q+2}, \tag{2.9}$$

where $\bar{d}_{x,q}$ and $\bar{d}_{x,q+2}$ denote the exact discretization errors for the formulas of order q and $q+2$, respectively. If we resolve the second and third part of (2.9) for the error of the order q and neglect the error of the higher order formula $u_{x,d,q+2}$, the error estimate d_x is defined by

$$d_x = u_{x,d,q+2} - u_{x,d,q}, \tag{2.10}$$

i.e. by the difference to the order $q+2$. If we resolve the first and the second part of (2.9) for the exact error $\bar{d}_{x,q} = \bar{d}_x$, we get

$$\bar{d}_x = u_x - u_{x,d,q}, \tag{2.11}$$

If we compare (2.11) to (2.10), we see that for the estimate the derivative is replaced by a "better" formula which holds only for sufficiently fine grid. Equation (2.10) is the

key for our explicit error access. If the higher order formula is not a better formula, we get large errors. So we have a built-in self-control of the error estimate.

2.3. The Error Equation. Pu (2.1) is an arbitrary non-linear function of its arguments. Therefore, we linearize system (2.1) with the Newton Raphson method by the approach

$$u \Leftarrow u^{(\nu+1)} = u^{(\nu)} + \Delta u^{(\nu)} \tag{2.12}$$

where we immediately drop the iteration index ν, and we linearize (2.1) in the Newton correction function Δu, i.e. we also get the corresponding derivatives of Δu, e.g. Δu_{xx}. So we get a linear PDE for the Newton correction function Δu:

$$\begin{aligned} Q\Delta u &\equiv -\frac{\partial Pu}{\partial u}\Delta u - \frac{\partial Pu}{\partial u_t}\Delta u_t - \frac{\partial Pu}{\partial u_x}\Delta u_x - \cdots - \frac{\partial Pu}{\partial u_{yy}}\Delta u_{yy} \\ &= P(t,x,y,u,u_t,u_x,u_y,u_{xx},u_{yy},u_{xy}) \end{aligned} \tag{2.13}$$

where the r.h.s. $P(\ldots) \equiv Pu \equiv Pu^{(\nu)}$ is the Newton residual for $u^{(\nu)}$. The $\partial Pu/\partial u_{\ldots}$ are the $l \times l$ Jacobian matrices. For a scalar PDE with only one unknown variable, it is a scalar value. If we have a system of l PDEs where u and Pu have l components, the Jacobian matrices look like this, e.g.:

$$\frac{\partial Pu}{\partial u} = \begin{pmatrix} \frac{\partial P_1 u}{\partial u_1} & \cdots & \frac{\partial P_1 u}{\partial u_l} \\ \vdots & & \vdots \\ \frac{\partial P_l u}{\partial u_1} & \cdots & \frac{\partial P_l u}{\partial u_l} \end{pmatrix}, \quad \frac{\partial Pu}{\partial u_x} = \begin{pmatrix} \frac{\partial P_1 u}{\partial u_{1,x}} & \cdots & \frac{\partial P_1 u}{\partial u_{l,x}} \\ \vdots & & \vdots \\ \frac{\partial P_l u}{\partial u_{1,x}} & \cdots & \frac{\partial P_l u}{\partial u_{l,x}} \end{pmatrix}. \tag{2.14}$$

Now we discretize (index d) the linear Newton-PDE (2.13) by replacing function values by their value on the grid and derivatives by difference formulas:

$$\Delta u \Leftarrow \Delta u_d, \quad \Delta u_t \Leftarrow \Delta u_{t,d}, \quad \Delta u_x \Leftarrow \Delta u_{x,d}, \ldots \tag{2.15}$$

where e.g. $\Delta u_{x,d}$ is the difference formula for Δu_x. However, the derivatives of the function $u = u^{(\nu)}$ in $P(\ldots)$ are replaced by difference formulas plus their error estimates:

$$u \Leftarrow u_d, \quad u_t \Leftarrow u_{t,d} + d_t, \quad u_x \Leftarrow u_{x,d} + d_x, \ldots \tag{2.16}$$

where e.g. d_x is the discretization error estimate for the difference formula $u_{x,d}$ (see Subsect. 2.2).

Then we linearize also in the discretization errors which again introduces Jacobian matrices (2.14). These additional error terms on the "level of equation", i.e. on the consistency level where we approximate a differential equation by a difference equation, create corresponding error terms on the "level of solution". If we arrange all these error terms on the l.h.s., we finally get the error equation for the overall error Δu_d:

$$\begin{aligned} \Delta u_d &= \Delta u_{Pu} + \Delta u_{D_t} + \Delta u_{D_x} + \Delta u_{D_y} + \Delta u_{D_{xy}} \quad \text{(level of solution)} \\ &= Q_d^{-1}\left[(Pu)_d + D_t + \{D_x + D_y + D_{xy}\}\right]. \quad \text{(level of equation)} \end{aligned} \tag{2.17}$$

Q_d denotes the large sparse matrix resulting from the discretization. It is the Q of (2.13) where all derivatives have been replaced by difference formulas. The inverse

Q_d^{-1} is never explicitly computed as it is a full matrix. $(Pu)_d$ is the "discretized" Newton residual where all derivatives have been replaced by difference formulas that are evaluated for $u_d = u_d^{(\nu)}$. The D_μ are discretization error terms that result from the linearization in the d_μ, e.g.

$$D_x = \frac{\partial Pu}{\partial u_x} d_x + \frac{\partial Pu}{\partial u_{xx}} d_{xx}. \tag{2.18}$$

In the brackets of the second row of (2.17) we have error terms that can be computed "on the level of equation" and that are transformed by Q_d^{-1} to the "level of solution". These corresponding errors on the solution level are arranged above their source terms. So the overall error Δu_d has been split up into the parts that result from the corresponding terms on the level of equation.

The only correction that is applied is the Newton correction Δu_{Pu} that results from the Newton residual $(Pu)_d$. It is computed from

$$Q_d \Delta u_{Pu} = (Pu)_d \tag{2.19}$$

by the linear solver program package LINSOL [10]. The Newton-Raphson iteration is stopped, if Δu_{Pu} is small enough, see Subsect. 2.4. The other error terms in the first row of (2.17) 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.

2.4. The Selfadaptation Process. For the determination of the optimal space order, we compute the space key error $\|D_x + D_y + D_{xy}\|_i$ for each node i for the orders $q = 2, 4, 6$ and select the order with the smallest value. Thus each node gets an individual order.

Before we discuss the selfadaptation in space direction, we need a scale for the accuracy on the level of equation in the sense of the error equation (2.17). Therefore, the user prescribes a global relative tolerance *tol* for the solution, and the refinement process is stopped if

$$\|\Delta u_d\|_{rel} := \frac{\|\Delta u_d\|}{\|u_d\|} \le tol \tag{2.20}$$

holds where $\|\cdot\|$ denotes the maximum norm. For the control of the solution process, we need a corresponding value on the level of equation. So we use the argument that the tolerances on the level of equation and on the level of solution behave like the norms of the errors on the level of equation ($\|(Pu)_d\|$, Newton residual) and on the level of solution ($\|\Delta u_{Pu}\|_{rel}$, relative Newton correction). So we get the following relation for *tolg* on the level of equation:

$$\frac{tolg}{tol} = \frac{\|(Pu)_d\|}{\|\Delta u_{Pu}\|_{rel}}. \tag{2.21}$$

With the relative Newton correction

$$\|\Delta u_{Pu}\|_{rel} = \frac{\|\Delta u_{Pu}\|}{\|u_d\|} \tag{2.22}$$

it holds for *tolg*

$$tolg = tol \cdot \|u_d\| \cdot \frac{\|(Pu)_d\|}{\|\Delta u_{Pu}\|}. \tag{2.23}$$

A node i for which $\|D_x + D_y + D_{xy}\|_i > s_{grid} \cdot tolg$ holds (s_{grid} is a tuning factor) is a refinement node, and the triangles (or tetrahedrons) the node belongs to are refined by halving the edges. For resaons of data organization we admit three nodes on an edge at most which may induce a refinement cascade. Space order control and mesh refinement are optional features of FDEM. For details, we refer to [1].

The tolerance on the level of equation $tolg$ is also used for the stopping of the Newton-Raphson iteration:

$$\|(Pu)_d\| < f \cdot \max\left(\frac{1}{2}\, tolg,\ \|D_x + D_y + D_{xy}\|\right) \tag{2.24}$$

where f is a safety factor. The smaller we set f the smaller $\|(Pu)_d\|$ must be to stop the Newton iteration, i.e. there will be executed more Newton steps. The iteration is also stopped if the relative Newton correction is small enough:

$$\|\Delta u_{Pu}\|_{rel} < \frac{1}{10}\, tol. \tag{2.25}$$

2.5. Parallelization. For the numerical solution of large PDE problems, we need much computation time and memory. Therefore, we need an efficiently parallelized program that is executed on distributed memory parallel computers with message passing (MPI). For the generation of the difference and error formulas, we need rings of neighbored nodes. Therefore, we re-sort the nodes for their x-coordinate, at first locally on each of the np processors, then globally by a special algorithm by which up to $np/2$ processors are active in parallel. By the sorting the nodes are distributed with increasing x-coordinate in np nearly equal parts on the np processors which results in a 1-D domain decomposition [8, §2.8], see Fig. 2.1. Note that FDEM is also a black-box solver concerning the domain: We do not know which 2-D or 3-D domain the user will deliver.

The elements are distributed correspondingly: An element is owned by the processor that owns its leftmost node. Here we make use of the "basket principle": Each processor puts its old, static data in a basket that is sent around in a nearest neighbor ring in np tacts. The np processors are conceptually treated as a linear arrangement with wrap-around, or more precisely as a ring. In each tact the processors take the information they need out of the current basket.

If we want to execute the generation and evaluation of the difference and error formulas and the computation of the matrix Q_d and the r.h.s. $(Pu)_d$ purely local without communication, we also have to store on processor ip node and element information of its left and right neighbor processor(s) which is indicated as overlap, see Fig. 2.1.

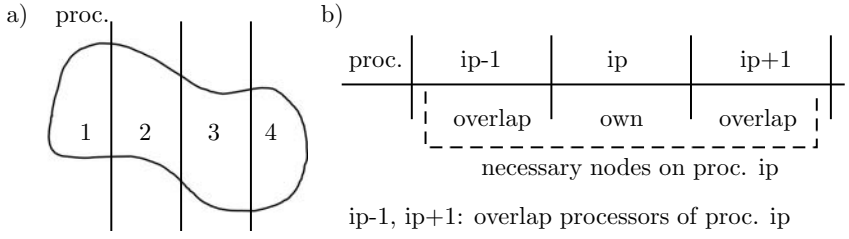

FIG. 2.1. *Illustration of the distribution of the data to the processors.*

The re-sorting of the nodes and elements onto the processors must be repeated after each mesh refinement to guarantee an efficient load-balancing.

The solution process of the PDEs starts with the data distributed onto the processors where on each processor a local numbering over all own and overlap data is used. So each processor can compute its part of the matrix Q_d and the r.h.s. $(Pu)_d$ completely independent of the other processors without communication as if it was a single processor and not only one processor in a parallel computer. Each processor calls the linear solver LINSOL with its part of the linear system of equations as parameter. The key for this seemingly quite simple procedure is the overlap and the local numbering. To solve the linear system of equations, we need communication between the processors in LINSOL, and finally each processor gets its part of the Newton correction Δu_{Pu}. Quite naturally, after each Newton step the values of u_d for the overlap nodes must be exchanged between the processors in FDEM.

2.6. Academic Examples. The purpose of these academic examples is to check the quality of the error estimate (which is one level higher than the usual check for the quality of the solution), see [8, §2.10]. We define the global relative error for a component j of the solution and the global relative error by

$$\frac{\|\Delta u_{d,j}\|}{\|u_{d,j}\|}, \quad \frac{\|\Delta u_d\|}{\|u_d\|} = \max_j \frac{\|\Delta u_{d,j}\|}{\|u_{d,j}\|}, \tag{2.26}$$

where $\Delta u_{d,j}$ is computed from the error equation (component j of (2.4.8) in [8]). The norm $\|\cdot\|$ is the maximum norm. For the test of our program, we use a PDE of which we know the exact solution. So we prescribe the test solution $\bar{u}(x, y)$ and generate from the original PDE $Pu = 0$ a "test PDE"

$$Pu - P\bar{u} = 0 \tag{2.27}$$

that has $\bar{u}$ as solution. $P\bar{u}$ is our problem with the known function $\bar{u}(x, y)$ instead of the unknown function u. Note that $P\bar{u}$ is a given function of x and y which is an absolute term in the test PDE that contains no variables. This prescription holds also for the BCs. The exact global relative error then is

$$\frac{\|\bar{u} - u_d\|}{\|u_d\|}. \tag{2.28}$$

We compute by FDEM the estimated relative error $\|\Delta u_d\|_{rel}$ (2.20) and compare this error to the exact relative error and thus get information about the quality of our error estimate.

Another important question is if the Newton-Raphson method converges. Newton's method converges quadratically if we are close enough to the solution. If not, anything may happen. Therefore, we introduce the damped Newton method with a relaxation factor that controls if the Newton residual $(Pu)_d$ decreases in the Newton step, see [8, §2.5].

We compute on the HP XC6000 with 1.5 GHz Intel Itanium2 processors and Quadrics interconnect (University of Karlsruhe, Germany). As exact solution $\bar{u}$ we select either a polynomial of a given order or a sugar loaf type function (2.10.16) in [8]. We solve the Navier-Stokes equations in velocity/vorticity form (2.10.13) in [8] with the unknown functions velocity components u, v and vorticity ω, and Reynolds number $Re = 1$. We solve on a circle with radius$= 1$ on a grid with 751 nodes,

TABLE 2.1
Results for the solution of the Navier-Stokes type equations on a circle with 751 nodes for different consistency orders q and test function $\bar{u}$.

	order $q=2$		order $q=4$		order $q=6$	
type $\bar{u}$	error exact error estim.	CPU sec.	error exact error estim.	CPU sec.	error exact error estim.	CPU sec.
pol. order 6	0.154 0.155	0.158	$0.914 \cdot 10^{-2}$ $0.367 \cdot 10^{-1}$	0.175	$0.108 \cdot 10^{-10}$ $0.109 \cdot 10^{-8}$	2.131
sugar loaf	$0.694 \cdot 10^{-1}$ $0.642 \cdot 10^{-1}$	0.168	$0.238 \cdot 10^{-1}$ $0.220 \cdot 10^{-1}$	0.184	$0.457 \cdot 10^{-2}$ 0.736*	1.853

* here the order 8 for the error estimate is overdrawn (too coarse grid)

1410 elements that has been generated by the commercial mesh generator I-DEAS. We compute with 8 processors. The given CPU time is that of the master processor 1.

Table 2.1 shows the results. Here are two remarks: For $\bar{u}$ polynomial of order 6 and consistency order $q = 6$ we should reproduce $\bar{u}$ exactly which is expressed by the small errors. For the sugar loaf function and consistency order $q = 6$, we get a large error estimate. This shows the built-in self-control: Near the top of the sugar loaf the grid is too coarse for the consistency order $q + 2 = 8$ that is used for the error estimate, the order 8 is "overdrawn" (higher order may not be better).

For the demonstration of the self-adaptation, we solve the same problem with the sugar loaf function again with 8 processors, but now we switch on the mesh refinement and order control for a global relative error of 0.25%. The results are shown in Table 2.2. The requested accuracy needs three refinement cycles. "no. of nodes ref." is the number of refinement nodes that determine the refinement elements from which then follows the new number of nodes. Observe the excellent error estimate that results from the optimal local order. Figure 2.2 shows the initial mesh and the refined mesh in the 3^{rd} cycle, the refinement is clearly visible.

TABLE 2.2
Results for the self-adaptation of mesh and order for sugar loaf test function for prescribed global relative error $0.25 \cdot 10^{-2}$ *(0.25%).*

cycle	no. of nodes	no. of elem.	no. of nodes ref.	no. of nodes with order 2	4	6	global relat. error exact	estimated	sec. for cycle
1	751	1410	132	427	320	4	$0.305 \cdot 10^{-1}$	$0.280 \cdot 10^{-1}$	1.021
2	1332	2493	345	180	1144	8	$0.109 \cdot 10^{-1}$	$0.950 \cdot 10^{-2}$	3.604
3	2941	5469	–	360	2556	25	$0.179 \cdot 10^{-2}$	$0.174 \cdot 10^{-2}$	10.086

As mentioned above we wanted to demonstrate the quality of the error estimate. The estimate is the better the smaller the error is, this is a natural consequence of (2.10). What we have seen in Table 2.1 is also part of our test technique for each new problem: At first we create from the new problem a test PDE $Pu - P\bar{u} = 0$ and check with polynomial test solutions $\bar{u}$ the error estimate like in Table 2.1.

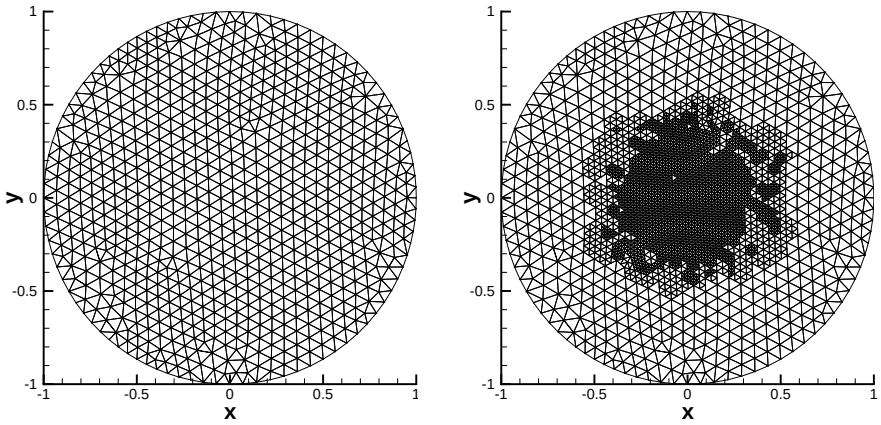

FIG. 2.2. *Initial mesh (left) and refined mesh for* 3^{rd} *cycle of Table 2.2 (right).*

3. Applications. We present the results of four problems that we solved in industrial and academic cooperations and that come from four completely different fields of application. We carry out the calculations on two distributed memory parallel computers that are installed at the Steinbuch Centre for Computing of the Karlsruhe Institute of Technology in Germany: The first one is the HP XC6000 with 1.5 GHz Intel Itanium2 processors and Quadrics interconnect, the second one is the HP XC4000 with 2.6 GHz AMD Opteron processors and InfiniBand interconnect. As we cannot present all the details for the discussed examples for the reason of space limitation, the reader can directly see the cited equations at the computer because all the references are accessible by the Internet.

3.1. Fluid-structure Interaction Problem for an Injection Pump and Heat Conduction in a Thin Annulus. In a high pressure Diesel injection pump the housing extends under the injection pressure of 2000 bar and the piston is compressed. The lubrication gap between housing and piston, which has a width of only a few micrometers, changes its form and consequently the leakage flow changes. This is a fluid-structure interaction problem. The problem is simplified by replacing the complicated shape of the housing by a tube or bush, see Fig. 3.1. The piston does not move, so we have a static configuration. A detailed presentation of this problem (industrial cooperation) is given in [8, §3.3].

The domain of solution has three subdomains with different PDEs: In the housing and piston we must solve the elasticity equations of steel, in-between we have the gap with the Navier-Stokes equations for Diesel. The coupling between these domains is the following: The fluid pressure p is the normal stress for housing and piston, so we have a direct interaction of the flow on the structure. By this normal stress, the housing expands and the piston is compressed, thus the form of the gap changes, and this changes the flow which changes p and thus the normal stress etc. So the interaction of housing and piston on the flow is indirect and more complicated and requires an iterative procedure.

We solve the problem in axisymmetrical coordinates, then x in Fig. 3.1 becomes the radius r. For the elasticity equations in housing and piston, the dependent variables are the displacements w and u in z- and r-direction, the stresses σ_z, σ_r, σ_φ and the shear stress τ_{rz} $(= \tau_{zr})$. Although we have rotational symmetry with $\partial/\partial\varphi = 0$,

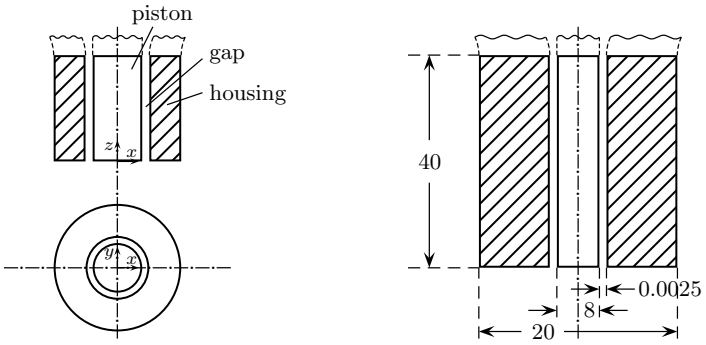

FIG. 3.1. *Symbolic configuration and dimensions in mm. In reality the gap is extremely thin.*

there is circumferential stress σ_φ. So we have six variables and need six PDEs. They are given in (3.3.4.1)–(3.3.4.6) in [8, p. 129].

In the lubrication gap we must solve the Navier-Stokes equations. The variables are the velocity components w and u in z- and r-direction and the pressure p. So we need a system of three equations that are given in (3.3.4.17)–(3.3.4.19) in [8, p. 123].

As the fluid is incompressible, we can prescribe the pressure only at one position, e.g. 2000 bar at the entry. We prescribe a parabolic velocity profile for w with w_{max} in the middle of the entry. We determine w_{max} iteratively to get an exit pressure equal to zero. Then the pressure on the dividing lines determines the displacements in the structure, and thereby also the fluid changes its form, and we have to compute a new value w_{max} again. If the grid does no longer move, we have the solution of our problem. So we have a fourfold nested iteration: The innermost iteration is the Newton iteration for the solution of the PDEs, then we must determine w_{max} for the exit pressure zero, then we must apply the displacements until the grid does no longer move. The outermost iteration gives the possibility to increase gradually the entry pressure.

TABLE 3.1

Max. value, max. and mean relat. error and volume flow through the gap for entry pressure of 2000 *bar for housing and lubrication gap.*

Housing

Var.	Unit	max. solution	max. relat. error	mean relat. error
w	cm	$0.4140 \cdot 10^{-2}$	$0.57 \cdot 10^{-4}$	$0.20 \cdot 10^{-5}$
u	cm	$0.7583 \cdot 10^{-3}$	$0.46 \cdot 10^{-3}$	$0.23 \cdot 10^{-4}$
σ_z	N/cm^2	$0.2238 \cdot 10^{+5}$	$0.22 \cdot 10^{-2}$	$0.29 \cdot 10^{-5}$
σ_r	N/cm^2	$0.2000 \cdot 10^{+5}$	$0.41 \cdot 10^{-2}$	$0.55 \cdot 10^{-5}$
σ_φ	N/cm^2	$0.2771 \cdot 10^{+5}$	$0.93 \cdot 10^{-3}$	$0.20 \cdot 10^{-4}$
τ_{rz}	N/cm^2	$0.9259 \cdot 10^{+3}$	$0.36 \cdot 10^{-1}$	$0.20 \cdot 10^{-4}$

Fluid

Var.	Unit	max. solution	max. relat. error	mean relat. error	Volume
w	cm/s	$0.3765 \cdot 10^{+4}$	$0.10 \cdot 10^{-1}$	$0.29 \cdot 10^{-2}$	2.65 cm^3/s
u	cm/s	$0.4034 \cdot 10^{0}$	$0.26 \cdot 10^{+2}$	$0.16 \cdot 10^{+1}$	
p	N/cm^2	$0.2000 \cdot 10^{+5}$	$0.83 \cdot 10^{-1}$	$0.72 \cdot 10^{-2}$	

We used a grid of 401×80 in z,r-direction for the housing, 401×641 for the fluid and 401×40 for the piston. We computed with 32 processors of the HP XC6000. The CPU time on the master processor 1 was 7.54 h, of which 7.41 h were needed

for the linear solver LINSOL [10] with full LU preconditioning. Table 3.1 gives some results for 2000 bar entry pressure for housing and fluid. In Table 3.3.5.1 in [8, p. 138], further values for entry pressures of 1500 bar to 3000 bar are given. In Fig. 3.2 one can see the form of the lubrication gap for 2000 bar entry pressure; the bold lines show the original channel. It is amazing how the high injection pressure changes the lubrication gap from the manufacturing dimension of 2.5 μm to up to 11.5 μm. The error estimates in Table 3.1 show the quality of the solution. For the fluid there is a maximal error of w of 1%, and the (arithmetic) mean error is only 0.29%.

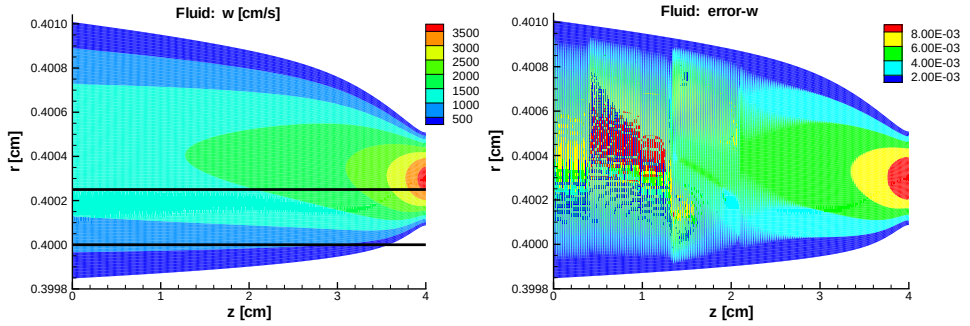

FIG. 3.2. *Contour plot for the velocity w in z-direction for* 2000 *bar and its error, and original channel (bold lines).*

Figure 3.2 shows the contour plot of the velocity w in z-direction which is responsible for the leakage flow that is in this case 2.65 cm^3/s. Figure 3.2 also shows its error plot. Here we can see that the large errors occur only locally. There a much finer grid should be used. As the mean error of w is 0.29%, we can conclude that the volume flow is also accurate to this error level. Note that this is a global error estimate that includes all the errors of all the equations in the coupled domains. Here the error estimate gives us the certainty that we can trust our solution. The reader may think how he could get this certainty by other methods for this complicated fluid-structure interaction problem.

Additionally, we are interested in the calculation of the stationary temperature field of the fluid in the lubrication gap between housing and piston. Therefore, we solve the heat equation for an incompressible Newtonian fluid using axisymmetric cylindrical coordinates in the gap. We use the same equation also in the piston and the housing, but due to vanishing flow velocities the equation becomes very simple.

For this computation, the changed form of the lubrication gap and the velocities are given, and by the solution of the PDE systems we obtain the temperature T in piston, housing and, as a matter of particular interest, in the gap. Both piston and housing consist of steel, while the lubricant in the gap is a model fluid. The material properties for the fluid, i.e. the thermal diffusivity, the kinematic viscosity and the specific heat capacity, as well as the BCs on the eight external boundaries and the CCs on the two dividing lines are given in [4]. For the three subdomains we used the same grids as for the fluid-structure interaction problem before.

We carried out the computation on 16 processors of the distributed memory supercomputer HP XC6000. The computation time for the master processor 1 is 155 sec. The results of the computation are shown in Table 3.2 where we present the maximum temperature, the maximum relative estimated error and the mean relative estimated error for the three subdomains.

TABLE 3.2
Maximum temperature, maximum and mean relative estimated error for piston, lubrication gap and housing for entry pressure of 2000 *bar.*

subdomain	T_{max} [°C]	global relat. error max.	global relat. error mean
piston	98.2	$0.29 \cdot 10^{-1}$	$0.46 \cdot 10^{-4}$
lubrication gap	98.9	$0.29 \cdot 10^{-1}$	$0.84 \cdot 10^{-3}$
housing	87.8	$0.40 \cdot 10^{-2}$	$0.24 \cdot 10^{-4}$

We see that the maximum relative errors are about 3% for the piston and the lubrication gap, but errors in the range of the maximum error appear only in a few nodes as the mean relative errors are $0.46 \cdot 10^{-4}$ in the piston and $0.84 \cdot 10^{-3}$ in the lubrication gap. In the housing the errors are even smaller.

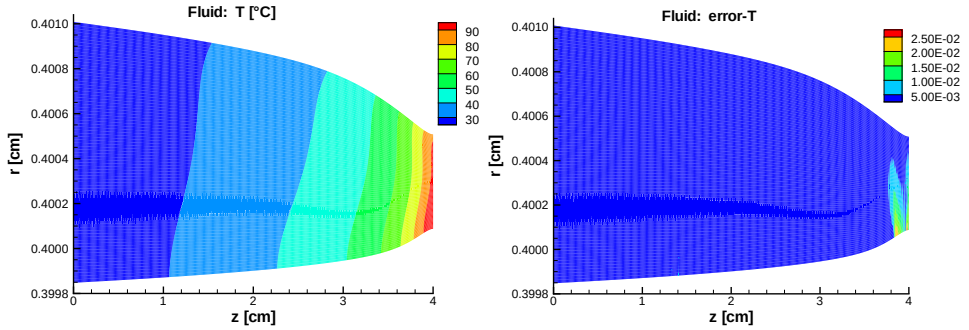

FIG. 3.3. *Contour plot for the temperature T and its error in the lubrication gap.*

Figure 3.3 shows the temperature T in the fluid and its error. The temperature increases from 20°C at $z = 0$ to 98.9°C at $z = 4\,\text{cm}$. From the error picture at the right side of the figure, we can also see that the maximum errors occur only locally. For the contour plots of the temperature in the piston and the housing, we refer to [4].

3.2. Simulation of a Power Semiconductor Module. In the following a thermal problem will be presented which is encountered in the thermal predictive simulation of power semiconductor modules (e.g. dc/ac-converters). Heat sources are MOSFET-devices (or other semiconductor devices) on the top side of the module. The cooling is applied at the bottom side of the module, either by a convective liquid or gas (air) stream. For a detailed description of the problem and the results, we refer to [3].

In power semiconductor modules, the temperature evolution $T(x,t)$ in the module and in the power dissipating devices is described by the time-dependent linear heat conduction equation. An essential nonlinearity arises due to the convective cooling at the bottom side, which is included by a corresponding boundary condition.

In order to have simple geometry and BCs, the whole module is assumed to be of rectangular structure with uniform material. The dimensions (side lengths) of the module are: 11.8 cm length; 5.8 cm width; 0.5 cm height. There are six chip heat sources in the module which do not disturb the geometry of the rectangular parallelepiped and are assumed as embedded regions with heat generation density $H(x,t)$ different from zero. Those six "chips" are quadratic and of equal dimension ($0.9 \times 0.9\,\text{cm}^2$) oriented in one row at the top side of the module, see Fig. 3.4. The

thickness of the chips is 0.02 cm (200 μm), and the top side of the chips is on the same plane as the top side of the rectangular module.

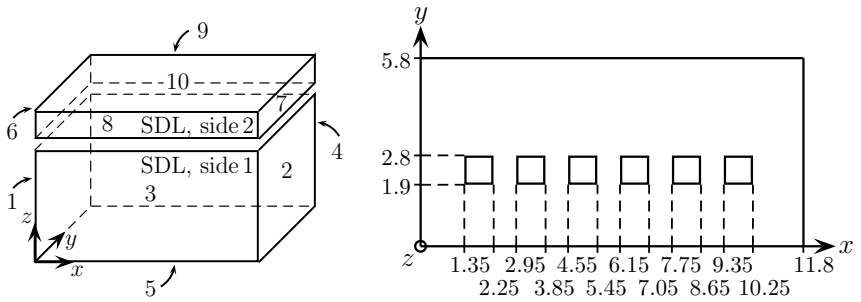

FIG. 3.4. *Illustration of the subdomains with external boundaries and SDL and of the module top surface with the position of the six chips.*

This is a 3-D problem, and as we want to compute the temperature distribution on the top surface of the module at a given time, it is also time-dependent. Furthermore, as the MOSFET-devices are very thin in comparison with the remainder of the module, we separate the module into two subdomains: The upper subdomain is as thick as the MOSFET-devices, the lower subdomain contains the remainder of the module. As we expect greater temperature gradients in the upper subdomain, and as it would be too costly to have the same fine grid in the lower subdomain, we choose different mesh sizes in x- and y-direction in the two subdomains. Thus, we introduce a sliding dividing line (SDL) between the two subdomains that allows for non-matching grid; they are coupled by CCs, see [8, §2.6].

Figure 3.4 shows the ten external boundaries and the SDL of the domain. The CCs on the SDL between the two subdomains are equal temperature and equal heat flow. The material parameters for the solid materials that we use for the computation are given in [3].

The following calculation is performed: At start time $t = 0$ the module has homogeneous temperature $T_a = 297\,\mathrm{K}$. The chips are turned on with power dissipation of 250 W/Chip. This means a heat generation density H of 250 W/chipvolume = 15,432.1 W/cm^3 for each of the six chips. What are the temperature contours on the top side of the module after 50 seconds?

We carried out the computation on 32 processors of the distributed memory supercomputer HP XC4000, and we used the consistency orders $q = 2$ and $q = 4$ to see the influence of the order. We used $237 \times 117 \times 9 = 249{,}561$ nodes in the upper subomain, and $119 \times 59 \times 9 = 63{,}189$ nodes in the lower subdomain. Therefore, the total number of grid points (and unknowns) is 312,750. The linear solver LINSOL [10] for the solution of the resulting linear system of equations consumes about 99.5% of the total CPU time.

In Table 3.3 we present the results of the computation. For the consistency orders $q = 2$ and $q = 4$, you can see the maximum temperature T_{max} and the errors of the solution for the upper and the lower subdomain. The maximum error is the maximum of the global relative estimated error, i.e. it is the maximum absolute error in the subdomain divided by the maximum of the temperature. The mean error is the arithmetic mean of all relative errors in the subdomain. The given CPU time is that of the master processor 1.

For order $q = 2$, the maximum temperature, which is actually met in the centre

TABLE 3.3
Results of the first calculation with $H = 250$ W/Chip.

Order	Subd.	T_{max} [K]	global relat. error max.	global relat. error mean	CPU time [h]
$q = 2$	upper	526.4	$0.11 \cdot 10^{-1}$	$0.81 \cdot 10^{-3}$	7.0
	lower	524.2	$0.69 \cdot 10^{-2}$	$0.17 \cdot 10^{-3}$	
$q = 4$	upper	526.2	$0.78 \cdot 10^{-2}$	$0.17 \cdot 10^{-3}$	42.0
	lower	524.1	$0.22 \cdot 10^{-2}$	$0.35 \cdot 10^{-4}$	

of the 3^{rd} chip from the left, is 526.4 K after 50 sec. In the lower subdomain, the maximum temperature is only slightly smaller. The maximum error in the upper domain is about 1%, in the lower subdomain it is only 0.7%, which means that we have a solution that is accurate to 6 K. The mean errors are much smaller which means that the maximum errors occur only in few nodes. Considering the mean error, the solution is accurate to 0.5 K.

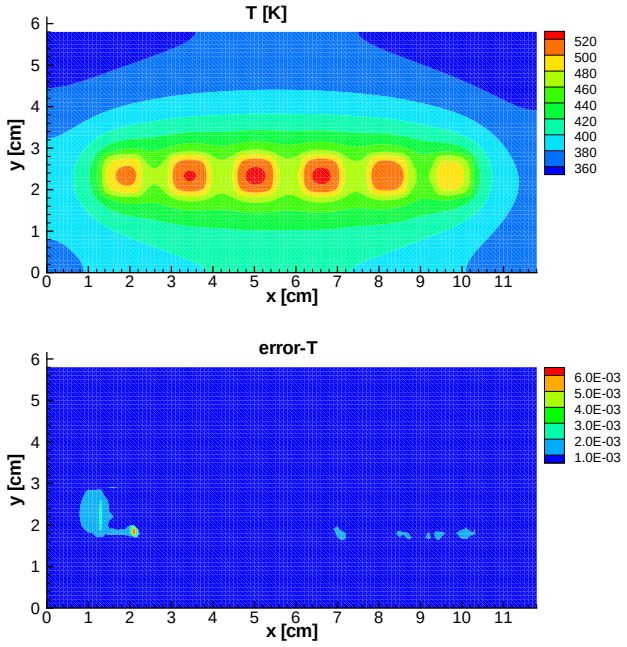

FIG. 3.5. *Contour plot for the temperature T and its error on the top side of the module for $q = 4$ after 50 sec. ($H = 250$ W/Chip).*

We see that the maximum temperatures for order $q = 2$ and $q = 4$ differ only slightly, but the maximum errors in the subdomains are reduced to 2/3 in the upper and to 1/3 in the lower subdomain if we compute with order $q = 4$ instead of $q = 2$. The mean errors are reduced to about 1/4 in both subdomains. For order $q = 4$ the solution is accurate to 4 K, and if we look at the mean error, it is accurate to 0.1 K.

Figure 3.5 shows the temperature T on the surface of the module and its error for the computation with consistency order $q = 4$. From the error picture, we can also see that the maximum errors occur only in few nodes.

For further details, we refer to [3] due to the limited space. There we also solve a second problem with a degraded array on the sixth chip with an additional power

dissipation of 50 W. Besides contour plots we present scalability tests for this problem where we computed on 32, 64, 128, 256 and 512 processors.

3.3. Numerical Solution of the PDEs for PEMFCs. The PEMFCs (Proton Exchange Membrane Fuel Cell or Polymer–Electrolyte–Membrane FC) are the "cold" FCs, operating at about 330 K. The domain of solution for the used model is the gas diffusion layer (GDL) that is half open to the oxygen channel at its lower left and half closed by a rib at its lower right, see Fig. 3.6. How this is cut out of a whole cell can be seen at Fig. 1 on p. III.4 in [9].

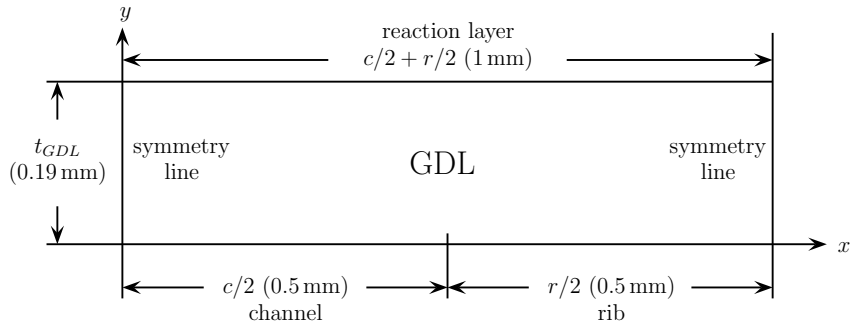

FIG. 3.6. *Domain of solution for the PEMFC.*

We use a model with a MTPM (Mean Transport Pore Model) transport mechanism. The variables for this problem are the molar flux densities of oxygen in x-direction $\dot{n}_o^x$ and y-direction $\dot{n}_o^y$, similarly for water vapor $\dot{n}_w^x$ and $\dot{n}_w^y$, and for nitrogen $\dot{n}_n^x$ and $\dot{n}_n^y$, the partial pressure for oxygen p_o, for water p_w and for nitrogen p_n, the total pressure p and as a special variable that has physical meaning only at the reaction layer the current density i. The 11 PDEs for these 11 variables are given in Part I of [9], and in Table 1 on p. III.6 is shown which equation is used for which variable position. The effective permeabilities that occur in the PDEs depend in a complicated nonlinear way from the pressure p and the partial pressures p_i. We do not discuss here the BCs, they are given on pp. III.9–III.11 in [9].

We compute with consistency order $q = 4$ on 32 processors of the HP XC6000, and we use a grid with 200×201 nodes resulting in 442,200 unknowns. The computation time was 4,123 sec on master processor 1.

TABLE 3.4
Results of the numerical simulation of a PEMFC.

Var.	max. solution	max. relat. error	mean relat. error
$\dot{n}_o^x$	$0.9850 \cdot 10^{-1}$	$0.61 \cdot 10^{-1}$	$0.55 \cdot 10^{-2}$
$\dot{n}_o^y$	$0.1062 \cdot 10^{0}$	$0.64 \cdot 10^{-1}$	$0.10 \cdot 10^{-2}$
$\dot{n}_w^x$	$0.1969 \cdot 10^{0}$	$0.61 \cdot 10^{-1}$	$0.55 \cdot 10^{-2}$
$\dot{n}_w^y$	$0.2124 \cdot 10^{0}$	$0.64 \cdot 10^{-1}$	$0.10 \cdot 10^{-2}$
$\dot{n}_n^x$	$0.3038 \cdot 10^{-4}$	$0.11 \cdot 10^{1}$	$0.11 \cdot 10^{0}$
$\dot{n}_n^y$	$0.1438 \cdot 10^{-4}$	$0.69 \cdot 10^{0}$	$0.79 \cdot 10^{-1}$
p_o	$0.1795 \cdot 10^{5}$	$0.27 \cdot 10^{-1}$	$0.69 \cdot 10^{-2}$
p_w	$0.2452 \cdot 10^{5}$	$0.27 \cdot 10^{-1}$	$0.69 \cdot 10^{-2}$
p_n	$0.6753 \cdot 10^{5}$	$0.25 \cdot 10^{-2}$	$0.65 \cdot 10^{-3}$
p	$0.1013 \cdot 10^{6}$	$0.47 \cdot 10^{-6}$	$0.12 \cdot 10^{-6}$
i	$0.3221 \cdot 10^{4}$	$0.19 \cdot 10^{-1}$	$0.50 \cdot 10^{-2}$

In Table 3.4, we present the results of the computation. Figure 3.7 shows a typical

result for $\dot{n}_w^y$ and its error. There is a quasi-singularity at the lower boundary where the BCs change from channel to rib. In [9] on pp. III.15–III.24 are figures of this type for all variables. So the engineer can see if he can trust the solution. If there was no error estimate, he had to do grid refinement tests and observe for all variables and nodes (here 442,200 values) how they change with finer grid. So the error estimate that consumes only a fraction of the total computation time is an invaluable feature in the solution process.

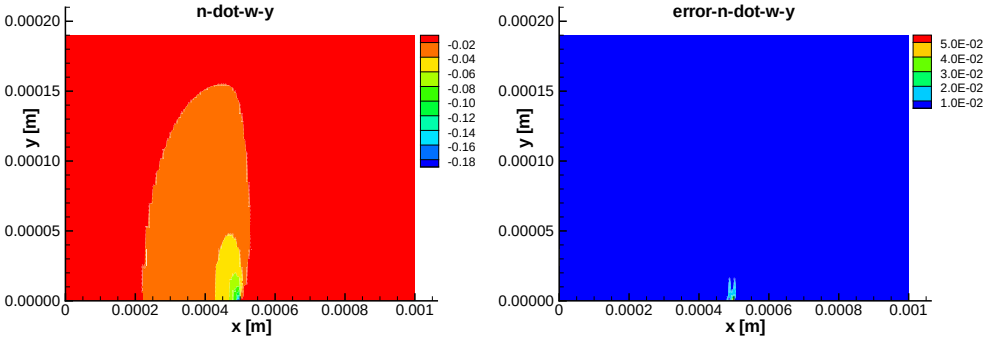

FIG. 3.7. *Contour plot of molar flux density of water vapor in y-direction* $\dot{n}_w^y$ *and its error.*

3.4. Numerical Solution of the PDEs for SOFCs. The SOFCs (Solid Oxide FCs) are the "hot" FCs, operating at about 1,200 K. The domain of solution for the used model is the anode with flow in porous media and the gas channel with Navier-Stokes equations, see Fig. 3.8.

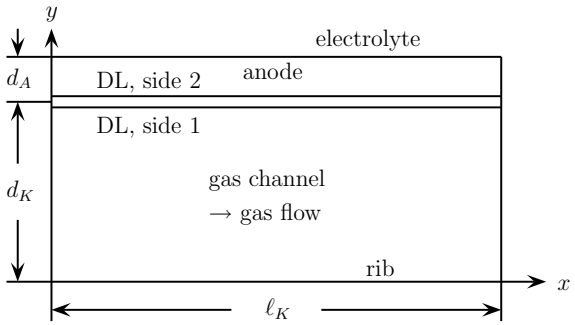

FIG. 3.8. *Domain of solution for the SOFC.*

The solution domain consists of two subdomains with different PDEs and a dividing line DL in-between which is an interior boundary where we must prescribe CCs. As the model includes methane reforming, the variables are the flow velocities u_x, u_y in x- and y-direction, the mole fractions Y_{CH_4} for methane, Y_{CO} for carbon monoxide, Y_{H_2} for hydrogen, Y_{CO_2} for carbon dioxide, Y_{H_2O} for steam, and pressure p. The eight PDEs for the eight variables are given in [9, Part II]. In the channel, we have Navier-Stokes-type equations and species transport equations. In the anode, the Navier-Stokes equations are replaced by Darcy's law. Here, the species transport equations have additional terms by the y-dependence of p and by the chemical reactions. These equations are extremely nonlinear because the viscosity of the

gas mixture and the diffusion coefficients depend nonlinearly on the $Y_{...}$, similarly the reaction rates are depending.

The BCs for the six external boundaries and the CCs for the dividing line DL are given in Tables 17–19 on pp. III.40 and III.41 in [9].

The numerical solution of these equations is much more critical than that of the PEMFCs because of the extreme nonlinearity of the coefficients. We compute with a grid with 80×41 nodes in both channel and anode, resulting in 52,480 unknowns, and we use consistency order $q = 4$. The computation time on 8 processors of the HP XC6000 is 510 sec on master processor 1.

TABLE 3.5
Results of the numerical simulation of a SOFC.

	Channel			Anode		
		global relat. error			global relat. error	
Var.	max. sol.	max.	mean	max. sol.	max.	mean
u_x	$0.6734 \cdot 10^{0}$	$0.36 \cdot 10^{-2}$	$0.35 \cdot 10^{-3}$	$0.2288 \cdot 10^{-1}$	$0.76 \cdot 10^{-1}$	$0.12 \cdot 10^{-2}$
u_y	$0.1750 \cdot 10^{-1}$	$0.39 \cdot 10^{-1}$	$0.13 \cdot 10^{-2}$	$0.1750 \cdot 10^{-1}$	$0.91 \cdot 10^{-2}$	$0.14 \cdot 10^{-2}$
Y_{CH_4}	$0.3300 \cdot 10^{0}$	$0.19 \cdot 10^{-2}$	$0.40 \cdot 10^{-4}$	$0.3300 \cdot 10^{0}$	$0.96 \cdot 10^{-2}$	$0.14 \cdot 10^{-3}$
Y_{CO}	$0.2022 \cdot 10^{0}$	$0.15 \cdot 10^{-1}$	$0.36 \cdot 10^{-2}$	$0.2022 \cdot 10^{0}$	$0.42 \cdot 10^{-1}$	$0.44 \cdot 10^{-2}$
Y_{H_2}	$0.5755 \cdot 10^{0}$	$0.64 \cdot 10^{-2}$	$0.77 \cdot 10^{-3}$	$0.5752 \cdot 10^{0}$	$0.18 \cdot 10^{-1}$	$0.99 \cdot 10^{-3}$
Y_{CO_2}	$0.8065 \cdot 10^{-1}$	$0.82 \cdot 10^{-2}$	$0.10 \cdot 10^{-2}$	$0.1270 \cdot 10^{0}$	$0.36 \cdot 10^{-1}$	$0.86 \cdot 10^{-3}$
Y_{H_2O}	$0.6700 \cdot 10^{0}$	$0.89 \cdot 10^{-2}$	$0.16 \cdot 10^{-2}$	$0.6700 \cdot 10^{0}$	$0.31 \cdot 10^{-1}$	$0.21 \cdot 10^{-2}$
p	$0.1014 \cdot 10^{6}$	$0.11 \cdot 10^{-5}$	$0.76 \cdot 10^{-6}$	$0.1082 \cdot 10^{6}$	$0.16 \cdot 10^{-2}$	$0.50 \cdot 10^{-3}$

The results of the computation are presented in Table 3.5. The colour plots of the results and error estimates for all eight variables in the channel and anode are presented in [9, pp. III.47–III.62]. Here we only present the plots of the mole fraction Y_{CO} and its error for the anode with thickness $d_A = 2\,\text{mm}$ in Fig. 3.9.

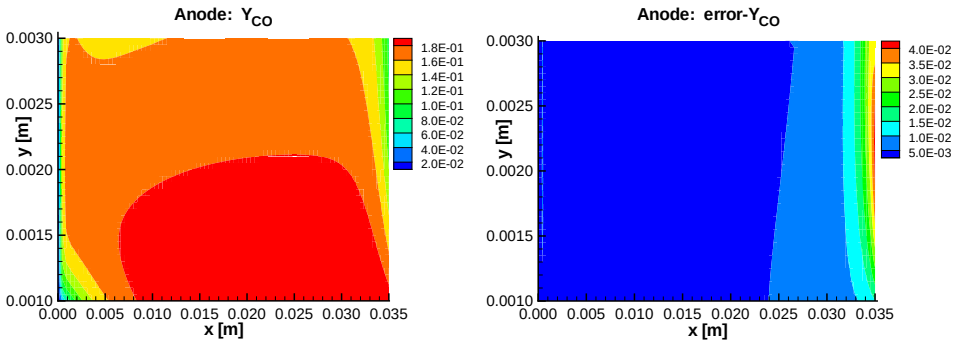

FIG. 3.9. *Contour plot of mole fraction Y_{CO} and its error for the anode ($d_A = 2\,mm$).*

The ampleness of the generated information can be estimated only if one looks at all the nice colour plots of the report [9]. There the engineer can immediately see the quality of the solution from the error plots, this is an unprecedented gain in information.

3.5. 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. The inflow conditions for the jet and the cross flow are given by two different prescribed velocity profiles. Chemical

component B enters in the main channel, by the side channel component A is entered. Fig. 3.10 shows the configuration of the investigated microreactor. We assume an incompressible fluid with Reynolds number 25 where the chemical components A and B are reacting and produce component Q.

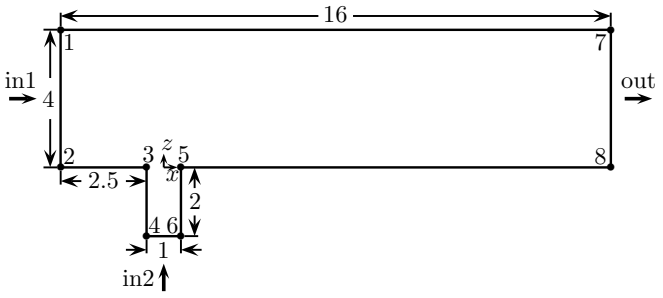

FIG. 3.10. *Configuration of the investigated microreactor.*

We use the following notations: velocity components u, w, pressure p, the mass fractions of A, B, Q are denoted by Y_A, Y_B, Y_Q.

We use nondimensional equations with reference length being the diameter D of the jet and reference velocity the velocity U_∞ of the cross flow. We need a system of six PDEs for the six variables u, w, p, Y_A, Y_B, Y_Q: For the mixture we use the continuity equation and two momentum equations, and for the chemical reaction we use two continuity equations for the components Y_A, Y_B and Dalton's law for the reaction product Y_Q. For the PDEs, the BCs and the material parameters, we refer to [2].

In order to get a global relative estimated error in the 1% region, we have to compute on a very fine grid. We use a grid with 2561×641 nodes in the main channel and 161×321 nodes in the pipe which yields 1,693,121 nodes and 10,158,726 unknowns. The computation time for consistency order $q = 4$ on 128 processors of the HP XC4000 is 11.24 h on master processor 1. The results of the computation are presented in Table 3.6.

TABLE 3.6
Results of the simulation of a microreactor.

Var.	max. solution	max. relat. error	mean relat. error
u	$0.266 \cdot 10^{1}$	$0.31 \cdot 10^{1}$	$0.12 \cdot 10^{-1}$
w	$0.301 \cdot 10^{1}$	$0.20 \cdot 10^{1}$	$0.46 \cdot 10^{-2}$
p	$0.100 \cdot 10^{6}$	$0.42 \cdot 10^{-2}$	$0.23 \cdot 10^{-2}$
Y_A	$0.100 \cdot 10^{1}$	$0.12 \cdot 10^{1}$	$0.12 \cdot 10^{-1}$
Y_B	$0.100 \cdot 10^{1}$	$0.78 \cdot 10^{0}$	$0.94 \cdot 10^{-2}$
Y_Q	$0.597 \cdot 10^{0}$	$0.74 \cdot 10^{0}$	$0.24 \cdot 10^{-1}$

We see that the maximum relative errors are quite large for five components but the mean relative errors are very good. This means that the maximum errors occur only locally. In Fig. 3.11 we illustrate the solution and the error of the chemical component Y_Q. From the error picture, we learn that the maximum errors are at the left corner where the jet enters into the main channel (in fact, there is only one node with such a large error). For the colour plots of the other variables and their errors, we refer to [2].

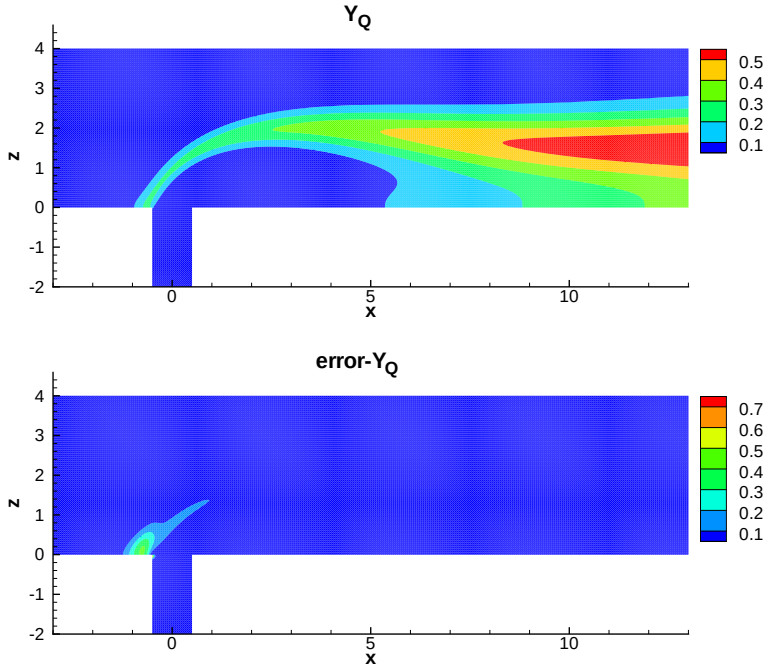

FIG. 3.11. *Contour plot of mass fraction Y_Q and its error.*

3.6. Comparison of FDEM and FLUENT. One of the referees of the paper wanted a comparison of FDEM with a commercial code. We selected an example with nonlinear PDEs, namely the Navier-Stokes equations with chemical reactions. We knew from our computations that this is a nontrivial "critical" problem. So we selected the microreactor of Subsect. 3.5. This example is fully covered by the well-known finite volume code FLUENT. Especially, we want to show the error of the solution computed with FLUENT (this error is computed with FDEM). We also want to compare the computation times, the convergence of the code with the mesh size and the scalability of the codes.

For the comparison, we use a grid that has got approximately a quarter of the nodes of the grid used in Subsect. 3.5, i.e. there are 1281×321 nodes in the main channel and 81×161 nodes in the pipe which yields 424,161 nodes and 2,544,966 unknowns.

First of all, we have to admit that the benefit of the error estimate can only be achieved by the sacrifice of memory and computation time. As we have to apply LU preconditioning to compute the solution of such a large and complex linear system of equations, we have to compute on 128 processors of the HP XC4000. The computation time of FDEM is 0.66 h.

In FLUENT we first compute the solution with the default values for the convergence check which is 10^{-3} for the flow velocities and the chemical components. After we computed the solution with the FLUENT program package which requires 20 iteration steps, we compute with the FDEM program package the error of the FLUENT solution, i.e. we store into FDEM the solution that was computed by FLUENT and then start the error computation. In Table 3.7 we present the maximum of the solution (computed by FLUENT) as well as the maximum and the mean value

TABLE 3.7
Results of the FLUENT computation with default convergence values.

Var.	max. solution (FLUENT)	max. relat. error (FLUENT)	mean relat. error (FLUENT)	mean relat. error (FDEM)
u	$0.363 \cdot 10^{1}$	$0.102 \cdot 10^{1}$	$0.109 \cdot 10^{0}$	$0.481 \cdot 10^{-1}$
w	$0.380 \cdot 10^{1}$	$0.112 \cdot 10^{1}$	$0.170 \cdot 10^{-1}$	$0.182 \cdot 10^{-1}$
p	$0.100 \cdot 10^{6}$	$0.135 \cdot 10^{-2}$	$0.260 \cdot 10^{-3}$	$0.835 \cdot 10^{-3}$
Y_A	$0.100 \cdot 10^{1}$	$0.201 \cdot 10^{1}$	$0.118 \cdot 10^{1}$	$0.266 \cdot 10^{-1}$
Y_B	$0.100 \cdot 10^{1}$	$0.192 \cdot 10^{1}$	$0.428 \cdot 10^{0}$	$0.365 \cdot 10^{-1}$
Y_Q	$0.640 \cdot 10^{0}$	$0.314 \cdot 10^{1}$	$0.223 \cdot 10^{1}$	$0.421 \cdot 10^{-1}$

of the relative estimated error (computed by FDEM) for each solution component. In column 5 we indicate the mean relative estimated error for the solution that was computed by FDEM. As we see from Table 3.7, the errors of the FLUENT solution are very large, e.g. for the most important component Y_Q the mean error is greater than 200%! The error for component Y_Q computed by FDEM is only 4.2%. The computation time of FLUENT is 0.07 h on 1 processor of the HP XC4000.

TABLE 3.8
Results of the FLUENT computation with 500 *iteration steps.*

Var.	max. solution (FLUENT)	max. relat. error (FLUENT)	mean relat. error (FLUENT)	mean relat. error (FDEM)
u	$0.231 \cdot 10^{1}$	$0.875 \cdot 10^{0}$	$0.142 \cdot 10^{0}$	$0.481 \cdot 10^{-1}$
w	$0.300 \cdot 10^{1}$	$0.404 \cdot 10^{0}$	$0.177 \cdot 10^{-1}$	$0.182 \cdot 10^{-1}$
p	$0.100 \cdot 10^{6}$	$0.295 \cdot 10^{-3}$	$0.634 \cdot 10^{-5}$	$0.835 \cdot 10^{-3}$
Y_A	$0.100 \cdot 10^{1}$	$0.201 \cdot 10^{1}$	$0.118 \cdot 10^{1}$	$0.266 \cdot 10^{-1}$
Y_B	$0.100 \cdot 10^{1}$	$0.195 \cdot 10^{1}$	$0.453 \cdot 10^{0}$	$0.365 \cdot 10^{-1}$
Y_Q	$0.628 \cdot 10^{0}$	$0.318 \cdot 10^{1}$	$0.230 \cdot 10^{1}$	$0.421 \cdot 10^{-1}$

So we try to improve these errors by allowing FLUENT to compute a better solution, i.e. we arbitrarily increase the number of iteration steps to 500, and again we computed the error of this solution by FDEM. The largest residuum after 500 iteration steps with FLUENT is $0.37 \cdot 10^{-4}$ for the flow velocity u. The results are shown in Table 3.8, and you see that only the errors of the flow velocities and the pressure decreased, but the errors of the chemical components almost did not change. By a further increase of the number of iteration steps to 1000 we could not observe any noticeable effect on the solution and the errors. The computation time for the FLUENT solution (500 iteration steps) is 2.90 h on 1 processor of the HP XC4000.

As FLUENT has no error estimate, the only way to get information about the accuracy is mesh refinement. Using the very fine grid of Subsect. 3.5 with 1,693,121 nodes, we are surprised that the errors do not become smaller at all. The mean error for the flow velocity component u is 37%, that one for the chemical component Y_Q is 211% whereas we have mean errors of 1.2% and 2.4% for u and Y_Q, respectively, for the solution computed with FDEM (see Subsect. 3.5)! The computation time for the FLUENT solution (500 iteration steps) is 15.73 h on 1 processor of the HP XC4000. This result shows the fatal drawback of the lack of an error estimate. FLUENT has two key parameters for the accuracy of the solution: the number of iterations and the mesh size. Increasing the number of iterations and reducing the mesh size does nearly not change the solution, above all the most important result component Y_Q. So the user of FLUENT may conclude that he has obtained a "good" solution, but it is completely wrong. It is not our task to find out the reasons for such a dangerous

behaviour. The price that we have to pay for the consistent and convergent solution method FDEM is increased memory and computation time, but the result is reliable.

The scalability of the FDEM code has been proven in [5]. We carried out some scalability computations for FLUENT for the grid with 424,161 nodes. Therefore, we computed on 1 to 16 processors on the HP XC4000 and respectively stopped the computations after 1000 iteration steps. From Table 3.9 we see that the overhead increases with the number of processors. From 4 to 8 processors we observe a stronger than expected reduction in computation time. This is not a scaling effect but a cache effect. Because we have more processors the data per processor are smaller and fit into the cache, thus reduce the computation time. The performance benchmark test results published on the FLUENT website [11] are similarly disillusioning: The efficiency of the parallel algorithm, which is the ratio of speedup to the number of processors, drops rapidly if one computes on more than 8 processors.

TABLE 3.9
Scalability tests for FLUENT on HP XC4000.

No. of proc. p	CPU time [h]	speedup sp	efficiency $e = sp/p$
1	5.75	1.0	100%
2	3.19	1.8	90%
4	2.15	2.7	68%
8	0.88	6.5	81%
16	0.56	10.3	64%

But one of the severest disadvantages of FLUENT for parallel computing is that there always is only a limited number of licenses at your disposal. You cannot simply increase the number of processors to compute the solution in a desired and acceptable time period, i.e. in contrast to FDEM you cannot use the maximum number of processors of a supercomputer because the FLUENT licenses are too expensive. So you will probably encounter unsurmountable difficulties especially for large 3D problems.

4. Conclusion. The purpose of this paper was to show that an error estimate is an invaluable advantage for a PDE solver. First, we solved a fluid-structure interaction problem for a high pressure Diesel injection pump. The high pressure bends up the housing that the lubrication gap widens from 2.5 micrometer up to 11.5 micrometer. The solution algorithm is a complicated nested iteration. Nevertheless, we compute a global error estimate for the coupled domains of housing, piston and fluid that tells us that we can trust our solution. Additionally, we solved the heat equation in the three subdomains. Then we solved the PDEs for the numerical simulation of the temperature in a power semiconductor module which is very challenging as it is a time-dependent problem in 3-D with two subdomains. Third, we solved the PDEs for the numerical simulation of fuel cells of the PEMFC and SOFC type and the error estimate showed the quality of the solution for all components of the systems. Finally, we simulated numerically the mixing and the chemical reaction in a microreactor. This is an extremely challenging problem as we need a very fine grid to get an error estimate in the 1% region. This is the first time that problems of this type are solved with the knowledge of the error. This knowledge forces a very fine grid for a 1% error. So we need supercomputers for seemingly simple problems. The knowledge of the error reveals the true nature of the numerical complexity of these problems. This is impressively proven by the comparison of the FDEM code and the FLUENT code.

REFERENCES

[1] T. Adolph, *The parallelization of the mesh refinement algorithm in the Finite Difference Element Method program package*, PhD thesis, Universität Karlsruhe (TH), 2005, URL http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/par_mra_fdem.pdf

[2] T. Adolph, W. Schönauer, J. Denev et al., *The snuffle problem Denev for the numerical simulation of a microreactor*, 2007, URL http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/snuffle-denev.pdf

[3] T. Adolph, W. Schönauer, Y. C. Gerstenmaier, *The snuffle problem Gerstenmaier for the heat conduction in a power module with 6 power chips*, 2007, URL http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/snuffle-gerstenmaier.pdf

[4] T. Adolph, W. Schönauer, M. Petry, *The snuffle problem Petry for the heat conduction in a thin annulus*, 2006, URL http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/snuffle-petry.pdf

[5] W. Schönauer, T. Adolph, *The FDEM (Finite Difference Element Method) program package: fully parallelized solution of elliptic and parabolic PDEs*, Computational Methods in Applied Sciences and Engineering, ECCOMAS 2000, Proceedings ECCOMAS, ISBN 84-89925-70-4.

[6] W. Schönauer, T. Adolph, *How WE solve PDEs*, J. Comput. Appl. Math., 131 (2001), pp. 473–492.

[7] W. Schönauer, T. Adolph, *FDEM: How we make the FDM more flexible than the FEM*, J. Comput. Appl. Math., 158, Issue 1 (2003), pp. 157–167.

[8] W. Schönauer, T. 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

[9] W. Schönauer, T. Adolph, M. Messerschmidt et al., *The numerical simulation of fuel cells of the PEMFC and SOFC type with the Finite Difference Element Method (FDEM)*, 2005, URL http://www.rz.uni-karlsruhe.de/rz/docs/FDEM/Literatur/fuelcells.pdf

[10] *LINSOL*, URL http://www.rz.uni-karlsruhe.de/rd/linsol.php

[11] *FLUENT performance benchmarks*, URL http://www.fluent.com/software/fluent/benchmarks.htm

Remark: Here are given only own references. The reason is that worldwide there is no other code that unifies all the properties of FDEM: arbitrary nonlinear systems of PDEs with arbitrary nonlinear BCs, subdomains with different PDEs, space and time-global error estimate, mesh refinement, optimization of consistency order, efficient parallelization with MPI.