Hardware-Oriented Multigrid Finite Element Solvers on GPU-Accelerated Clusters

Stefan Turek, Dominik Göddeke, Sven H.M. Buijssen and Hilmar Wobker

Institut für Angewandte Mathematik, TU Dortmund, Germany
stefan.turek@math.tu-dortmund.de

1 Introduction and Motivation

The accurate simulation of real-world phenomena in computational science is often based on an underlying mathematical model comprising a system of partial differential equations (PDEs). Important research fields that we pursue in this setting are computational solid mechanics and computational fluid dynamics (CSM and CFD, see Section 3). Practical applications range from material failure tests, as for instance crash tests in the automotive industry, to fluid and gas flow of any kind, for instance in chemical or medical engineering (e. g., simulation of blood flow in the human body to predict aneurysms) or flow around cars and aircrafts to minimize drag and lift forces. Moreover, the coupling of both models is essential for fluid structure interaction settings (FSI) which represent problem fields of very high technological importance. Such configurations include polymer processing or microfluidic problems exhibiting very complex multiscale behavior due to nonlinear rheological or non-isothermal constitutive laws, and also due to self-induced oscillations of the structural parts in the flow field. In all these cases, the fluid part is mostly laminar, but highly viscous.

The corresponding flow models are based on the Navier-Stokes equations which seem to have a quite simple structure at first sight; nevertheless they constitute 'grand challenge' problems for mathematicians and physicists as well as engineers and computer scientists. They are (still today) subject to very intensive research activities, especially in the following fields:

- time dependent partial differential equations in complex domains
- strongly nonlinear systems of equations
- saddle-point problems due to the incompressibility constraint
- local changes of the problem character in space and time
- temporarily stiff systems of differential equations

These characteristics impose great challenges on almost all numerical algorithms and computational approaches. Among others, the following mathematical issues have to be taken into consideration if efficient simulation tools are to be designed:

- large, ill-conditioned nonlinear systems (*millions of unknowns*)
- locally varying time steps (*implicit schemes*)
- locally anisotropic spatial meshes (*complex geometries*)
- efficient parallel solvers (*decomposition-invariant scalability*)

From a mathematical and engineering point of view, finite element methods (FEM) are considered to be the most promising approaches for the numerical treatment of such PDE problems, due to their flexibility and accuracy, particularly for general settings and complex geometries including unstructured computational meshes. Moreover, they provide a complete framework which allows rigorous a posteriori error control and corresponding adaptive grid manipulations. While classical approaches provide a sharp quantitative estimation of the error only in certain specific configurations (and should therefore be better referred to as *error indicators*), sophisticated finite element techniques overcome these deficiencies and can be formulated in a very general framework such that adaptive error control strategies can also be applied to realistic flow and FSI configurations. Summarizing the state of the art, finite element techniques form in combination with powerful and robust numerical solution schemes the underlying fabric of many modern simulation tools.

Looking at solution rather than discretization techniques next, hierarchical (geometric) multigrid methods are more or less obligatory due to their asymptotic optimality, since many of the considered (flow) problems lead to huge, ill-conditioned problems where the condition number depends on the mesh width, resp., problem size: Even for a 'simple' Poisson problem, a multigrid solver with 'standard' choices of smoother, data structure and numbering scheme for the unknowns executes faster than a (single-grid) Krylov subspace solver with very powerful elementary preconditioners for relevant problem sizes. Since the solution of Poisson-like subproblems is an essential building block of Navier-Stokes and also elasticity solvers, the use of multigrid techniques is mandatory for these problem classes.

As a conclusion, we can state that modern FEM software packages for general continuum mechanical PDE problems, especially in solid mechanics and fluid dynamics, are typically based on highly sophisticated discretization techniques, which have the potential to handle very general computational meshes. However, particularly in the case of huge realistic 3D problems, the realization of efficient parallel multigrid solvers, providing at least double precision accuracy due to typically bad condition numbers of the linear systems, is equally essential. Only the combination of all these mathematical components permits the design of flexible, robust and accurate simulation tools with high numerical efficiency.

Still, this is not enough to realize simulation software with correspondingly high computational efficiency. In numerics, hardware aspects used to be of minor importance since codes automatically ran faster with each new generation of processors. This trend has come to an end, as physical limitations (heat, leaking voltage, pin limits) have led to a paradigm change: Performance improvements are no longer driven by frequency scaling, but by parallelism and specialization. In fact, single-core performance already stagnates or even goes down. Quad-core CPUs are available off-the-shelf, and soon, CPUs will have tens of parallel cores. Future manycore chip designs will likely be heterogeneous and contain general and specialized cores with non-uniform memory access characteristics (NUMA). Commodity multimedia processors such as the Cell BE or graphics processor units (GPUs) are considered as forerunners of this trend even though they can currently only be used as co-processors (accelerators) to the general-purpose CPU.

In order to achieve a significant percentage of the available peak performance, both on conventional and novel architectures, hardware characteristics must be taken into account in all stages of the implementation and code optimization. This includes the selection of appropriate data structures (e. g., to store matrices or to communicate data over the interconnects efficiently) and parallelization techniques, in particular when combining the coarse-grained parallelism on the cluster level and the medium- and fine-grained parallelism between the CPU cores, and within accelerator devices like GPUs. Furthermore, different approaches are necessary for different architectures. The same holds true for performance tuning techniques like spatial blocking to exploit cache hierarchies or to coalesce memory transfers into large, more efficient bulk transactions. On the other hand, the meticulous tuning of each application for each new hardware generation is prohibitively expensive, and techniques are required that encapsulate the hardware-awareness inside the underlying finite element discretization *and* solver toolkit, away from the applications.

We are convinced that in the field of high performance finite element simulations, significant performance improvements can only be achieved by *hardware-oriented numerics* which is a quite young discipline in the field of computational science and engineering (CSE). The core paradigm of hardware-oriented numerics is that numerical and algorithmic foundation research must go hand in hand with (long-term) technology evolution: Prospective hardware trends enforce research into novel numerical techniques that are in turn better suited for the hardware. As an example, strong multigrid smoothers with optimal numerical properties often scale poorly in a parallel setting, due to their strong recursive coupling. Only correspondingly modified schemes that are potentially less numerically efficient on a single node (e. g., in terms of convergence rates) are able to achieve better overall performance in the user-relevant 'time-to-solution'-metric. The ultimate goal of hardware-oriented numerics is thus to balance these metrics to achieve robust and ideally predictable close-to-peak performance. Only with the combination of the 'best' numerics and 'best' computational algorithms for a given hardware architecture it is possible to satisfy the aims of hardware-oriented numerics, namely to maximize the total efficiency, i.e., to realize the following 'vision' which is the underlying key idea for our finite element software project FEAST:

Hardware-oriented Numerics: Maximize Total Efficiency

High (guaranteed) accuracy for user-specific quantities with minimal number of degrees of freedom (#DOF) via fast and robust solvers – for a wide class of parameter variations – with optimal numerical complexity (O(#DOF)) while exploiting a significant percentage of the available sequential/parallel peak performance at the same time.

2 FEAST - Finite Element Analysis & Solution Tools

FEAST is our next-generation simulation toolkit, which prototypically implements a wide range of hardware-oriented numerics concepts. FEAST is designed for large-scale distributed memory simulations and uses MPI for communication. Here, we briefly present the key concepts, and refer to previous publications for details [2, 15, 18].

2.1 Separation of Structured and Unstructured Data

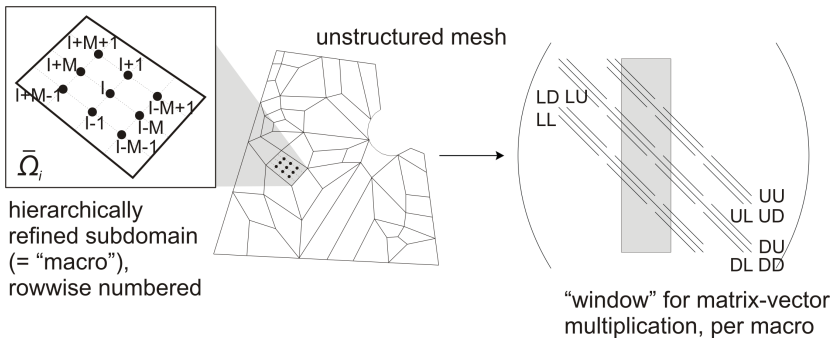

Figure 1: Locally structured, globally unstructured mesh in FEAST.

FEAST covers the computational domain with a collection of quadrilateral subdomains. The subdomains form an unstructured coarse mesh (cf. Figure 1 and also Figure 4 on page 11, bottom), and each subdomain is refined in a generalized tensor product fashion. The resulting mesh is used to discretize the set of PDEs with finite elements. This approach caters to the contradictory needs of flexibility in the discretization and high performance: The unstructured coarse mesh retains flexibility in resolving geometric and simulation details, such as boundary layers or discontinuities. The tensor

product property of the local meshes entails a linewise numbering of the unknowns which leads to a banded structure of the matrices and is exploited in optimized numerical linear algebra *and* multigrid smoothing and transfer components. Instead of keeping all data in one general, homogeneous data structure, FEAST stores only local FE matrices and vectors (corresponding to subdomains) and thus maintains a clear separation of structured and unstructured parts of the domain. Several subdomains can be grouped together and treated within one MPI process.

2.2 Parallel Multigrid Solvers

ScaRC (*Scalable Recursive Clustering*), the solver concept at the core of FEAST, generalizes techniques from *multilevel domain decomposition* and *parallel multigrid*; combining their respective advantages into a very robust, and (numerically *and* computationally) efficient parallel solution scheme for (scalar) elliptic PDEs. Matrices and vectors are stored only locally as usual for distributed memory approaches, while at the same time a minimally overlapping decomposition ensures that on the one hand, the union of all local matrices always composes the 'virtual' global matrix, and on the other hand, only the minimally necessary amount of data needs to be shared via communication, i. e., only data associated with degrees of freedom lying on subdomain boundaries has to be exchanged, ensuring good scalability by design. Between the different mesh resolutions, the coupling is multiplicative, as in classical multigrid. Within one hierarchy level, the coupling is additive, i. e. the minimally overlapping subdomains are treated simultaneously and independently of each other. Global coarse grid problems are solved with UMFPACK [6] on a master node.

Instead of blockwise application of elementary local smoothers, the ScaRC scheme employs full multigrid solvers acting locally on the individual subdomains to 'hide' local irregularities as much as possible from the outer solver, see the recursive data flow in Figure 2. The local solvers are typically configured to gain, e. g., one digit, and can fully exploit the underlying tensor product property by executing hardware-optimized code paths.

The resulting hierarchical solvers are very robust, exhibiting very good weak and strong scaling. In previously published work, we demonstrated – for the maximum available resources at that time – perfect weak scalability for the Poisson problem on up to 320 Xeon processors [10], and excellent strong scaling for applications from linearized elasticity and incompressible flow for an experiment that subsequently quadrupled the resources up to a maximum of 128 CPUs [19].

2.3 Scalar and Multivariate Problems

The guiding idea to treating multivariate problems with FEAST is to rely on the modular, highly optimized and extensively tested core routines for the scalar case in order to formulate robust schemes for a wide range of applications, rather than using the best suited numerical scheme for each application and repeatedly optimizing it for new architectures. Multivariate PDEs as they arise in CSM and CFD can be rearranged and discretized in such a way that the resulting equation systems consist of blocks that correspond to scalar subequations. We illustrate this exemplarily in Section 3.3 with help of the elasticity equation. Figure 2 summarizes the idea on a high level.

This special block-structure can be exploited in two ways: On the one hand, all standard linear algebra operations on the multivariate system (e. g., matrix-vector multiplications, defect computations, dot products) can be implemented as a series of operations for scalar systems, taking advantage of FEAST's highly tuned numerical linear algebra components. On the other hand, the process of solving multivariate linear equation systems can be brought down to the treatment of auxiliary scalar subsystems which can be efficiently solved by FEAST's optimized toolbox of parallel multigrid solvers. Once such a solution approach is set up, the hardware-awareness and all further improvements of FEAST's scalar library directly transfer to the solvers for multivariate systems. Such improvements can be the addition

of better multigrid components (e. g., more robust local smoothers) or algorithmic adaptations to dedicated HPC architectures and hardware co-processors (see Section 2.4 and Figure 2). ScaRC allows for an almost arbitrary (recursive) combination of multivariate and scalar as well as global and local solvers, thus providing great flexibility in tuning the linear solution schemes to the given problem.

2.4 Co-processor Acceleration

Hardware accelerators such as GPUs are integrated into Feast in a 'minimally invasive' way [10,11,13], encapsulating heterogeneities of the system on the compute node level, so that MPI sees a globally homogeneous system. Figure 2 illustrates how this concept fits into Feast's general solution strategy.

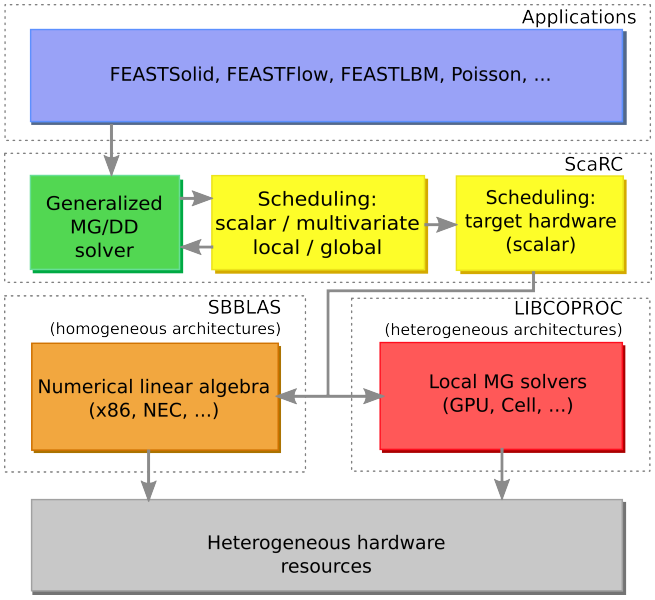

Figure 2: Illustration of the minimally invasive accelerator integration in Feast.

The currently implemented prototype [8] offloads local scalar multigrid solvers, specifically tailored to the tensor product property, onto GPUs. This concentrates sufficient fine-grained parallelism in a separate task and thus minimizes the potential overhead of repeated co-processor configuration and data transfer in case of a system integration via relatively narrow busses such as PCIe. On accelerators that do not natively provide sufficient floating point precision, a mixed precision iterative refinement technique is applied to ensure accuracy of the results [12]. The entire approach thus has one important benefit: *Application code does not need to be changed at all to benefit from co-processor acceleration.*

3 Two FEAST Applications: FEASTSOLID and FEASTFLOW

In this section, we introduce two important classes of applications that have been built on top of Feast: The solid mechanics code FeastSolid solves static and transient elasticity problems for small and finite deformations. The fluid dynamics code FeastFlow solves the transient incompressible Stokes and Navier–Stokes equations. Other applications, e. g., fluid structure interaction and Lattice Boltzmann methods, are actively being developed, but are not considered here.

3.1 Computational Solid Mechanics

In computational solid mechanics (CSM) the deformation of solid bodies under external loads is examined. We consider a two-dimensional body covering a domain $\bar{\Omega} = \Omega \cup \partial\Omega$, where $\Omega \subset \mathbb{R}^d, d = 2, 3$, is a bounded, open set with boundary $\Gamma = \partial\Omega$. The boundary is split into two parts: the Dirichlet part Γ_D where displacements are prescribed and the Neumann part Γ_N where surface forces can be applied ($\Gamma_\mathrm{D} \cap \Gamma_\mathrm{N} = \emptyset$). Furthermore the body can be exposed to volumetric forces, e. g., gravity. FEASTSOLID is able to handle nonlinear finite elasticity problems and incompressible materials [20]. However, here we do not use these advanced problem-specific features and only treat the simple yet fundamental linearized 2D model problem of elastic, compressible material under static loading, assuming small deformations. This allows us to better quantify the solver aspects we focus on in this paper. In terms of complexity, this model problem of elasticity is situated between the standard Poisson problem and the Navier–Stokes equations.

Mathematically, we formulate the linearized elasticity equation in terms of the displacements $\boldsymbol{u}(\boldsymbol{x}) = \big(u_1(\boldsymbol{x}), u_2(\boldsymbol{x})\big)^\mathsf{T}$ of a material point $\boldsymbol{x} \in \bar{\Omega}$ as the only unknowns. The strains can be defined by the linearized strain tensor $\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), i, j = 1, 2$, describing the linearized kinematic relation between displacements and strains. The material properties are reflected by the constitutive law, which determines a relation between the strains and the stresses. We use Hooke's law for isotropic elastic material, $\boldsymbol{\sigma} = 2\mu\boldsymbol{\varepsilon} + \lambda\,\mathrm{tr}(\boldsymbol{\varepsilon})\boldsymbol{I}$, where $\boldsymbol{\sigma}$ denotes the symmetric stress tensor and μ and λ are the so-called Lamé constants. The basic physical equations of elasticity are determined by equilibrium conditions. For a body in equilibrium, the inner forces (stresses) and the outer forces (external loads $\boldsymbol{f}$) are balanced:

$$-\mathbf{div}\boldsymbol{\sigma} = \boldsymbol{f}, \qquad \boldsymbol{x} \in \Omega.$$

Using Hooke's law to replace the stress tensor, the problem of linearized elasticity can be expressed in terms of the following elliptic boundary value problem, called the Lamé equation:

$$-2\mu\,\mathbf{div}\boldsymbol{\varepsilon}(\boldsymbol{u}) - \lambda\,\mathbf{grad}\,\mathrm{div}\,\boldsymbol{u} = \boldsymbol{f}, \qquad \boldsymbol{x} \in \Omega, \tag{1a}$$

$$\boldsymbol{u} = \boldsymbol{g}, \qquad \boldsymbol{x} \in \Gamma_\mathrm{D}, \tag{1b}$$

$$\boldsymbol{\sigma}(\boldsymbol{u}) \cdot \boldsymbol{n} = \boldsymbol{t}, \qquad \boldsymbol{x} \in \Gamma_\mathrm{N}. \tag{1c}$$

Here, $\boldsymbol{g}$ are prescribed displacements on Γ_D, and $\boldsymbol{t}$ are given surface forces on Γ_N with outer normal $\boldsymbol{n}$. To discretize the continuous problem, the domain $\bar{\Omega}$ is approximated by a collection of tensor product subdomains $\bar{\Omega}_i$ as described in Section 2.1. We consider the weak formulation of equation (3) and apply a finite element discretization with conforming bilinear elements Q_1. For details on the finite element technique and on the elasticity problem, see for example the textbook by Braess [3].

3.2 Computational Fluid Dynamics

To tackle problems from computational fluid dynamics (CFD) we discretize the Navier–Stokes equations. They describe the flow of Newtonian fluids like gases, water and many other liquids in a domain $\Omega \subset \mathbb{R}^d, d = 2, 3$ and are, under certain assumptions and simplifications, derived from the conservation laws for mass, momentum and energy. For the sake of simplicity we restrict ourselves to the 2D stationary case.

Confining the domain and imposing boundary conditions, i. e., in- and outflow conditions on the 'artificial' boundaries and slip or adhesion conditions at rigid walls, the following system of nonlinear equations is obtained under the assumption of constant kinematic viscosity $\nu > 0$ (independent of

pressure and specific heat capacity) and constant temperature:

$$-\nu\Delta\boldsymbol{u} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nabla p = \boldsymbol{f}, \qquad \boldsymbol{x}\in\Omega, \tag{2a}$$

$$\operatorname{div}\boldsymbol{u} = \boldsymbol{0}, \qquad \boldsymbol{x}\in\Omega, \tag{2b}$$

$$\boldsymbol{u} = \boldsymbol{g}, \qquad \boldsymbol{x}\in\Gamma_{\mathrm{D}}, \tag{2c}$$

$$\nu\partial_n\boldsymbol{u} + p\cdot\boldsymbol{n} = 0, \qquad \boldsymbol{x}\in\Gamma_{\mathrm{N}}. \tag{2d}$$

Here, $\boldsymbol{u}$ denotes the fluid velocity, p the pressure, $\boldsymbol{n}$ the outer normal vector and Γ_{D} and Γ_{N} the boundary parts with, respectively, Dirichlet and Neumann boundary conditions (i. e., inflow, outflow and adhesion conditions).

We discretize the equation with the Q_1/Q_1 bilinear element pair and use residual-based SUPG/PSPG stabilization (streamline upwind/Petrov-Galerkin and pressure-stabilization/Petrov-Galerkin) to account for both the LBB deficiency of the Q_1/Q_1 pair and to stabilize convective terms [4, 14]. To allow for anisotropic grid cells directional derivatives are incorporated in the stabilization terms [5]. For details on the CFD problem in general see for example the textbook by Ferziger and Peric [7].

3.3 Solving CSM and CFD Problems with FEAST

In order to solve CSM and CFD problems using the FEAST intrinsics described in Section 2, the degrees of freedom of the discretized systems have to be ordered corresponding to the components of the underlying multivariate equations. We exemplarily illustrate this for the case of linearized elasticity, where the technique is sometimes called separate displacement ordering [1]. In the 2D case, the unknowns $\boldsymbol{u} = (u_1, u_2)^{\mathsf{T}}$ are the displacements in x- and y-direction. A corresponding operator-splitting of the left hand side of equation (1a) yields

$$-\begin{pmatrix}(2\mu+\lambda)\partial_{xx} + \mu\partial_{yy} & (\mu+\lambda)\partial_{xy}\\ (\mu+\lambda)\partial_{yx} & \mu\partial_{xx} + (2\mu+\lambda)\partial_{yy}\end{pmatrix}\begin{pmatrix}u_1\\u_2\end{pmatrix} = \begin{pmatrix}f_1\\f_2\end{pmatrix}. \tag{3}$$

The finite element discretization of the correspondingly arranged weak form leads to a linear equation system $\mathbf{K}\mathbf{u} = \mathbf{f}$ exhibiting a corresponding block structure,

$$\begin{pmatrix}\mathbf{K}_{11} & \mathbf{K}_{12}\\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{pmatrix}\begin{pmatrix}\mathbf{u}_1\\\mathbf{u}_2\end{pmatrix} = \begin{pmatrix}\mathbf{f}_1\\\mathbf{f}_2\end{pmatrix}, \tag{4}$$

where $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)^{\mathsf{T}}$ is the vector of external loads and $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2)^{\mathsf{T}}$ the (unknown) coefficient vector of the finite element solution. Each global matrix/vector $\mathbf{K}_{ij}, \mathbf{u}_i, \mathbf{f}_i (i, j = 1, 2)$ exists only 'virtually' through the corresponding local matrices/vectors defined over the local subdomains (cf. Section 2.1). The decisive advantage of this operator-splitting is that the matrices $\mathbf{K}_{11}$ and $\mathbf{K}_{22}$ correspond to scalar elliptic operators (cf. Equation (3)) which allows using FEAST's tuned scalar solvers (cf. Section 2.2).

We now illustrate how scalar subsystems can be utilized to solve the whole multivariate system by means of a basic preconditioned defect correction method:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \omega\tilde{\mathbf{K}}_{\mathrm{BGS}}^{-1}(\mathbf{f} - \mathbf{K}\mathbf{u}^k). \tag{5}$$

This scheme acts on the global system (4) and thus couples the two sets of unknowns $\mathbf{u}_1$ and $\mathbf{u}_2$. $\tilde{\mathbf{K}}_{\mathrm{BGS}}$ is a block-Gauss-Seidel preconditioner that explicitly exploits the block structure of the matrix $\mathbf{K}$. One iteration of the global defect correction scheme consists of the following three steps:

1. Compute the global defect (cf. Section 2.3):

$$\begin{pmatrix}\mathbf{d}_1\\\mathbf{d}_2\end{pmatrix} = \begin{pmatrix}\mathbf{f}_1\\\mathbf{f}_2\end{pmatrix} - \begin{pmatrix}\mathbf{K}_{11} & \mathbf{K}_{12}\\ \mathbf{K}_{21} & \mathbf{K}_{22}\end{pmatrix}\begin{pmatrix}\mathbf{u}_1^k\\\mathbf{u}_2^k\end{pmatrix}.$$

2. Apply the block preconditioner

$$\tilde{\mathbf{K}}_{\text{BGS}} := \begin{pmatrix} \mathbf{K}_{11} & \mathbf{0} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{pmatrix}$$

by approximately solving the system $\tilde{\mathbf{K}}_{\text{BGS}}\mathbf{c} = \mathbf{d}$. This is performed by two scalar solves and one (scalar) matrix-vector multiplication:

a) Solve $\mathbf{K}_{11}\mathbf{c}_1 = \mathbf{d}_1$.

b) Update RHS: $\mathbf{d}_2 = \mathbf{d}_2 - \mathbf{K}_{21}\mathbf{c}_1$.

c) Solve $\mathbf{K}_{22}\mathbf{c}_2 = \mathbf{d}_2$.

3. Update the global solution with the (eventually damped) correction vector: $\mathbf{u}^{k+1} = \mathbf{u}^k + \omega\mathbf{c}$.

The solution of the two scalar subsystems (steps 2(a) and 2(b)) constitutes the largest amount of the total arithmetic work and fully exploits FEAST's tuned (and co-processor-accelerated) scalar solvers (cf. Section 4.3).

In the case of the Navier–Stokes equations, the whole solution process can be brought down to the solution of scalar systems as well. In a first step, the nonlinearities are resolved by a fixed point defect correction method such that repeatedly (in each nonlinear step) linear saddle point systems of the form

$$\begin{pmatrix} \mathbf{A} + \mathbf{C_1} & \mathbf{B} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{C_2} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix} \tag{6}$$

are to be solved, where the matrices $\mathbf{C_1}, \mathbf{C_2}$ stem from the SUPG/PSPG stabilization terms. In 2D, the matrix $\mathbf{A} + \mathbf{C_1}$ has a 2×2 block structure similar to that of the stiffness matrix $\mathbf{K}$ in the elasticity case (see Equation 4). Our solution algorithm is a block Schur complement (BSC) approach as described by Murphy et al. [16]. It basically consists of a global BiCGStab solver acting on the whole saddle point system which is block-preconditioned by

$$\begin{pmatrix} \tilde{\mathbf{A}} & \mathbf{0} \\ \mathbf{B}^{\mathsf{T}} & \tilde{\mathbf{S}} \end{pmatrix}. \tag{7}$$

Herein, $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{S}}$ respectively denote preconditioners of $\mathbf{A} + \mathbf{C_1}$ and of the Schur complement matrix $\mathbf{S} = \mathbf{B}^{\mathsf{T}}(\mathbf{A} + \mathbf{C_1})^{-1}\mathbf{B} - \mathbf{C_2}$. The former is realized by approximately solving subsystems of the kind $(\mathbf{A} + \mathbf{C_1})\mathbf{c} = \mathbf{d}$ for which exactly the same solution strategy is applied as for the CSM system (4). The Schur complement preconditioner $\tilde{\mathbf{S}}$ is realized by a suitably weighted linear combination of pressure mass and Laplacian matrices (see Turek [17]). Applying the preconditioner means solving scalar subsystems which can again be done with FEAST's optimized ScaRC schemes. Hence, the solution of the whole saddle point system is mainly brought down to the solution of scalar systems again.

4 Performance Assessments

We demonstrate the performance and efficiency of our approach with a number of selected benchmark tests. These results have been gathered over the past two years on a wide range of different HPC installations and test clusters using CPUs and GPUs from different hardware generations.

4.1 GPU-based Multigrid on a single Subdomain

In the literature, reported speedups obtained on GPUs vary dramatically. Our first experiment assesses the (realistic) speedup we can obtain in our setting on a single, generalized tensor product subdomain. We use a GPU-based multigrid iteration to solve the fundamental scalar Poisson problem on one of the deformed subdomains near the inner boundary in the unstructured channel flow configuration shown in Figure 4 (bottom), prescribing an analytical right hand side so that we know the exact solution of the problem and can compute the error of the approximate solution. Single precision is insufficient to accurately solve this problem while we always achieve correct results with our mixed precision iterative refinement scheme [12].

Level	**DOF**	**CPU double**	**GPU double**	**speedup**	**GPU mixed**	**speedup**
5	1 089	0.0018	0.0156	0.1	0.0140	0.1
6	4 225	0.0059	0.0187	0.3	0.0206	0.3
7	16 641	0.0272	0.0260	1.1	0.0232	1.2
8	66 049	0.1460	0.0356	4.1	0.0284	5.2
9	263 169	0.7747	0.0656	11.8	0.0435	17.8
10	1 050 625	3.3609	0.1731	19.4	0.0944	35.6

Table 1: Multigrid solvers on a single subdomain (Time to solution in seconds and speedup).

The speedup comparisons in Table 4.1 have been obtained on a typical high-end GPU workstation as of June 2008 (Core2Duo E6750, fast DDR2-800 memory, NVIDIA GeForce GTX 280 GPU). Column 3 demonstrates that the CPU gets less and less efficient as soon as the problem does not fit entirely into cache anymore, while columns 4 and 6 show that the GPU needs reasonably large problem sizes to be fully saturated and hide all latencies of accesses to off-chip memory. The speedup we observe for these two configurations executing entirely in double precision is almost $20\times$ and includes the transfer of the right hand side to the device and the transfer of the solution back from the device. We do not include the transfer of the matrix data, because this is part of the matrix assembly and thus separated from our acceleration of linear solvers, see Section 4.3. In addition, the mixed precision scheme on the device is almost twice as fast as native double precision (last two columns). It is noteworthy that we can solve a sparse linear system with one million unknowns in less than 0.1 seconds, using a fully assembled matrix, not merely a stencil.

4.2 Scalability

Our second experiment demonstrates the excellent weak scalability of our approach (we address strong scalability elsewhere [19]). The results shown in Figure 3 have been obtained on a cluster with two singlecore EM64T Xeon CPUs and a Quadro FX1400 GPU per node. We observe excellent scalability when simultaneously doubling the problem size and the number of nodes for both the Poisson problem and the FeastSolid application (using the same configuration as shown in Figure 4, top). As the GPU is one generation behind the Xeon processors and only has 128 MB of memory, it can barely hold the data associated with one subdomain , and we have to page data in and out of device memory for each local solve [10, 13]. Therefore, the obtained speedups are not spectacular but still noteworthy. In particular, we are able to accurately solve a Poisson problem with more than 1.3 billion unknowns in slightly more than 40 seconds.

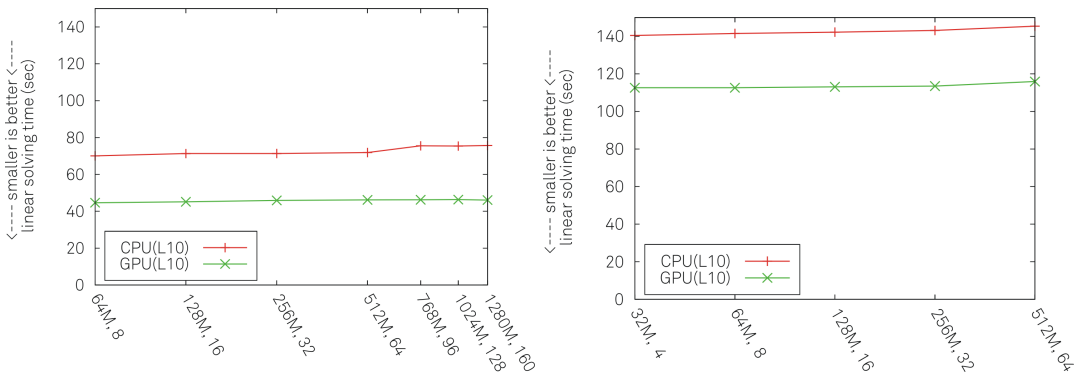

Figure 3: Weak scalability results for the Poisson (left) and the CSM (right) solvers (x-axis: #DOF and #nodes; y-axis: linear solving time in seconds). When performing the CSM tests, less nodes were available to us.

4.3 Application Speedup

In this section, we analyse how the (local) speedups of the solvers acting on a single subdomain (Section 4.1) translate to the application level. For the elasticity application FeastSolid, we solve a prototypical benchmark problem, a block subjected to an external load (see Figure 4, top). These results are obtained on a 16-node cluster (Opteron 2210 dualcore CPU, Quadro FX5600 GPU). Each of the 64 subdomains is refined up to level 10 for a total problem size of 128 million unknowns. Smoothing the local multigrid with simple Jacobi iterations is sufficient, the mesh is isotropic and the operator only mildly anisotropic. The GPU-accelerated solver achieves a speedup by a factor of 2.6 for refinement level 10, and 2.0 for level 9 respectively [13].

We benchmark the GPU-accelerated FeastFlow application on a small test cluster (four nodes, Opteron 2214 dualcore, GeForce 8800 GTX). These GPUs are slightly faster than the Quadros used in the tests of the elasticity application, but still two generations behind the GPU used in Section 4.1. For the full Navier-Stokes solver and a 'flow around a cylinder' benchmark (see Figure 4, bottom), a strong smoothing operator is required due to the nonlinearities. We resort to an alternating direction implicit variant of a tridiagonal smoother that couples each unknown with its left and right neighbor. The GPU implementation [8] of this smoother is based on cyclic reduction, while the CPU implementation can use the less expensive serial Thomas algorithm. For this configuration, only refinement level 9 fits into memory of the four nodes. The total speedup we observe in terms of 'time to solution' is 1.9× (results updated from [9]).

Discussion. We first emphasize that our reported speedups are very noteworthy, as they have been obtained without changing a single line of application code: Halving the execution time of an already carefully (numerically and implementationally) optimized code is usually impossible. Nonetheless, the question arises why the gap to the 'ideal' measurements (Section 4.1) is so large. The reason is inherent to our highly modular approach: We only accelerate the local scalar multigrid solvers, i. e., only a portion of the entire linear solver. The approach is thus prone to be limited by Amdahl's law. To quantify this effect and to assess how the achieved speedups of the local multigrid solver translate to the global application, we equip the code with timers measuring the local multigrid solves only. Looking at the elasticity solver first, we observe local speedups of 4.1 and 9.0 on level 9 and 10, respectively. The local work constitutes roughly 66 % of the entire linear solver, so the maximum achievable total speedup is 3× assuming infinite local acceleration. With a factor of 2.6× on level 10, we are reasonably close to this ideal speedup. For the Navier-Stokes solver, we measure a local speedup of 11.4×, resulting in an acceleration of the linear solver by a factor of 4.3. The accelerable fraction of the linear solver is approximately 84 %, limiting the speedup to a factor of 6.4. However,

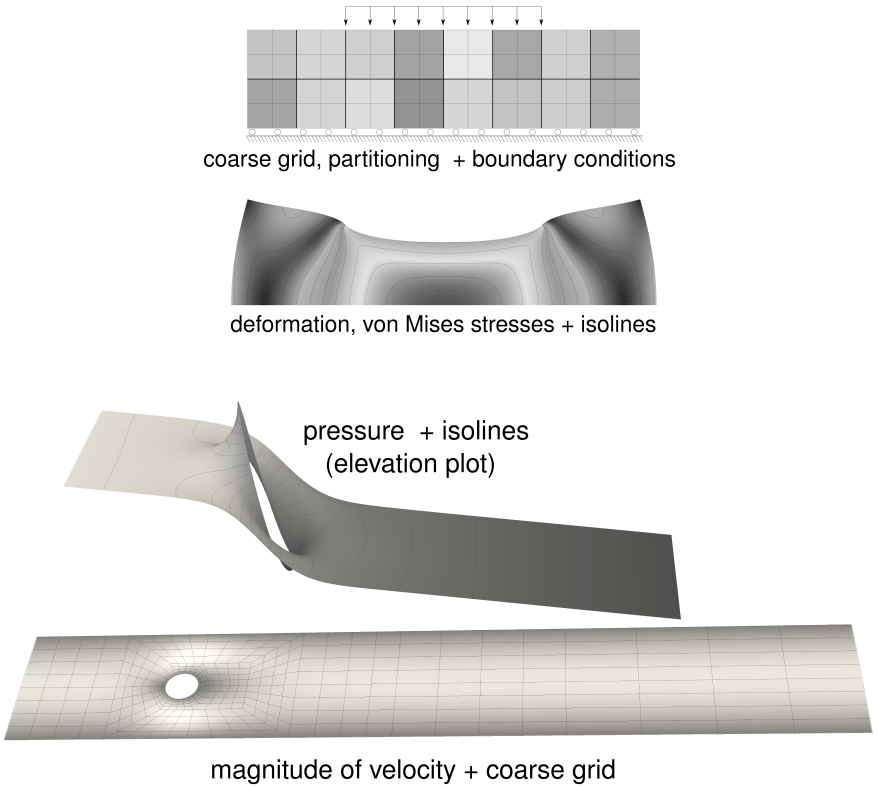

Figure 4: Computed benchmark results for the elasticity (top) and fluid dynamics (bottom) solvers.

considering the entire nonlinear problem, all matrices have to be assembled in each nonlinear step (see Section 3.3), a task that is currently not accelerated by the GPU. Furthermore, the nonlinear defect correction loop is executed on the CPU. For this particular test problem, 40 % of the entire time to solution is spent in the assembly routine, and for the accelerated solver, this fraction increases to over 70 %. Consequently, the speedups are diminishing significantly, and measurements reveal that only 50 % of the entire solution process (baseline CPU version) can be accelerated, resulting in a theoretical upper bound of a factor of two. With a measured speedup of 1.9× for the entire application, we are sufficiently close to this optimum.

Conclusion and Outlook. Even when using a strong, robust multigrid smoother, our GPU-based local multigrid solver is more than one order of magnitude faster than a corresponding CPU implementation. Since we only accelerate the scalar local portions of the entire multivariate global linear solver, the total speedup on the application level is bound by the fraction of time spent in these local solves. The only possibility to improve speedups significantly while not suffering from the implications of Amdahl's law is to re-implement large portions of each application specifically for the GPU, which can be impractical or prohibitively expensive in practice. Nonetheless, we achieve speedups by a factor of two or more for simulations in CSM and CFD. In the spirit of 'hardware-oriented numerics' however, we are pursueing ideas to design solution schemes with a higher potential for acceleration, ideally more than 90 %. The linear solver in the Navier-Stokes benchmark is already close to this goal, exhibiting an acceleration potential of 84 %. For nonlinear problems however, the assembly of the linearized systems can constitute the dominant part of the computations, at least for the GPU-accelerated version. The second challenge in future work is thus to port the assembly process to GPUs in a similarly 'minimally invasive' way, so that we maintain the most important benefit of our approach: Application code does not have to be changed at all.

5 Summary

We have motivated the concept of *hardware-oriented numerics* and described our finite element and solver toolbox FEAST, which prototypically implements many of its aspects. For two important model applications from solid mechanics and fluid dynamics we have demonstrated how we break down the solution process into sequences of scalar solves. This approach has the important advantage that all performance improvements and in particular adaptions to specific hardware architectures are automatically transferred to the application level, without having to change application code at all. In the context of this chapter, we use GPUs as accelerators to the general-purpose CPU. We implemented scalar multigrid solvers on the GPU, which executes the scalar local subdomain solves within the multivariate global parallel solver. The resulting hybrid solvers scale very well, and we observe noteworthy speedups. However, as we only accelerate portions of the entire solution scheme, our speedups are limited by the remaining fraction, and local speedups of more than one order of magnitude do not (yet) translate fully to the application level.

Acknowledgements

This work has been funded in part by German Deutsche Forschungsgemeinschaft (projects TU 102/22-2, TU 102/27-1 and TU 102/11-3), and by German Bundesministerium für Bildung und Forschung in the SKALB project (grant 01IH08003D) of call 'HPC Software für skalierbare Parallelrechner'.

References

[1] Owe Axelsson. On iterative solvers in structural mechanics; separate displacement orderings and mixed variable methods. *Mathematics and Computers in Simulations*, 50(1–4):11–30, November 1999.

[2] Christian Becker. *Strategien und Methoden zur Ausnutzung der High-Performance-Computing-Ressourcen moderner Rechnerarchitekturen für Finite Element Simulationen und ihre Realisierung in FEAST (Finite Element Analysis & Solution Tools)*. PhD thesis, Universität Dortmund, May 2007. `http://www.logos-verlag.de/cgi-bin/buch?isbn=1637`.

[3] Dietrich Braess. *Finite Elements – Theory, fast solvers and applications in solid mechanics*. Cambridge University Press, 2nd edition, April 2001.

[4] Alexander N. Brooks and Thomas J. R. Hughes. Streamline upwind/Petrov-Galerkin formulations for convection dominated flows with particular emphasis on the incompressible Navier-Stokes equations. *Computer Methods in Applied Mechanics and Engineering*, 32(1–3):199–259, September 1982.

[5] Sven H.M. Buijssen. *Efficient Multilevel Solvers and High Performance Computing Techniques for the Finite Element Simulation of the Transient, Incompressible Navier–Stokes Equations*. PhD thesis, TU Dortmund, Fakultät für Mathematik, 2010. in preparation.

[6] Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):165–195, June 2004.

[7] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 3rd edition, December 2001.

[8] Dominik Göddeke. *Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters*. PhD thesis, TU Dortmund, Fakultät für Mathematik, May 2010. `http://hdl.handle.net/2003/27243`.

[9] Dominik Göddeke, Sven H.M. Buijssen, Hilmar Wobker, and Stefan Turek. GPU acceleration of an unmodified parallel finite element Navier-Stokes solver. In Waleed W. Smari and John P. McIntire, editors, *High Performance Computing & Simulation 2009*, pages 12–21, June 2009.

[10] Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick S. McCormick, Sven H.M. Buijssen, Matthias Grajewski, and Stefan Turek. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing*, 33(10–11):685–699, September 2007.

[11] Dominik Göddeke, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick S. McCormick, Hilmar Wobker, Christian Becker, and Stefan Turek. Using GPUs to improve multigrid solver performance on a cluster. *International Journal of Computational Science and Engineering*, 4(1):36–55, November 2008.

[12] Dominik Göddeke, Robert Strzodka, and Stefan Turek. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations. *International Journal of Parallel, Emergent and Distributed Systems*, 22(4):221–256, January 2007.

[13] Dominik Göddeke, Hilmar Wobker, Robert Strzodka, Jamaludin Mohd-Yusof, Patrick S. McCormick, and Stefan Turek. Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU. *International Journal of Computational Science and Engineering*, 4(4):254–269, October 2009.

[14] Thomas J.R. Hughes, Leopoldo P. Franca, and Marc Balestra. A new finite element formulation for computational fluid dynamics: V. Circumventing the Babuška-Brezzi condition. A stable Petrov-Galerkin formulation of the Stokes problem accomodating equal-order interpolations. *Computer Methods in Applied Mechanics and Engineering*, 59(1):85–99, November 1986.

[15] Susanne Kilian. *ScaRC: Ein verallgemeinertes Gebietszerlegungs-/Mehrgitterkonzept auf Parallelrechnern*. PhD thesis, Universität Dortmund, Fachbereich Mathematik, January 2001. `http://www.logos-verlag.de/cgi-bin/buch?isbn=0092`.

[16] Malcolm F. Murphy, Gene H. Golub, and Andrew J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM Journal on Scientific Computing*, 21(6):1969–1972, May 2000.

[17] Stefan Turek. *Efficient Solvers for Incompressible Flow Problems: An Algorithmic and Computational Approach*. Springer, June 1999.

[18] Stefan Turek, Christian Becker, and Susanne Kilian. Hardware–oriented numerics and concepts for PDE software. *Future Generation Computer Systems*, 22(1-2):217–238, February 2004.

[19] Stefan Turek, Dominik Göddeke, Christian Becker, Sven H.M. Buijssen, and Hilmar Wobker. FEAST – Realisation of hardware-oriented numerics for HPC simulations with finite elements. *Concurrency and Computation: Practice and Expecience*, February 2010. Special Issue Proceedings of ISC 2008.

[20] Hilmar Wobker. *Efficient Multilevel Solvers and High Performance Computing Techniques for the Finite Element Simulation of Large-Scale Elasticity Problems*. PhD thesis, TU Dortmund, Fakultät für Mathematik, March 2010. `http://hdl.handle.net/2003/26998`.