FEAST – Realisation of hardware-oriented Numerics for HPC simulations with Finite Elements

Stefan Turek, Dominik Göddeke, Christian Becker, Sven H.M. Buijssen, Hilmar Wobker
Angewandte Mathematik und Numerik, TU Dortmund, Germany
stefan.turek@math.tu-dortmund.de

FEAST (Finite Element Analysis & Solutions Tools) is a Finite Element based solver toolkit for the simulation of PDE problems on parallel HPC systems which implements the concept of 'hardware-oriented numerics', a holistic approach aiming at optimal performance for modern numerics. In this paper, we describe this concept and the modular design which enables applications built on top of FEAST to execute efficiently, without any code modifications, on commodity based clusters, the NEC SX 8 and GPU-accelerated clusters. We demonstrate good performance and weak and strong scalability for the prototypical Poisson problem and more challenging applications from solid mechanics and fluid dynamics.

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 application domains that we pursue in this setting are computational solid mechanics and computational fluid dynamics (CSM and CFD, see Section 3). Practical applications in these fields range from material failure tests (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 aircraft to minimise drag and lift forces.

From an engineering and mathematical point of view, Finite Element (FE) approaches are well-suited for the numerical treatment of such PDE problems, due to their flexibility and accuracy. Software development for FE problems has traditionally focused on the improvement of the numerical methodology. For instance, adaptivity techniques, higher order discretisations, and a wide range of stabilisation approaches have been presented. In combination with powerful and robust numerical solution schemes (in particular hierarchical multigrid strategies), FE techniques form the underlying fabric of many modern simulation tools.

Hardware aspects used to play only a minor role, 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 specialisation. Quad-core CPUs are available off-the-shelf, and soon, CPUs will have tens of parallel cores, with hundreds to follow. Future manycore chip designs will likely be heterogeneous and contain general and specialised cores with non-uniform memory access (NUMA). Commodity multimedia processors such as the Cell BE or graphics processor units (GPUs) can be considered as forerunners of this trend even though they can currently only be used as co-processors to the general-purpose CPU, and form a viable testbed for algorithmic research. They are of particular interest, as the multi-billion dollar market of video games entails rapid evolution and superlinear performance improvements across chip generations at comparatively low acquisition costs, in contrast to more specialised solutions aiming at the HPC market alone (e. g., ClearSpeed's Advance Accelerator boards). As an example, the GeForce GTX 285, NVIDIA's flagship GPU at the time of writing, contains 30 multiprocessors with eight 'cores' and 16 kB of shared cache each, and a 512-bit interface to main memory; resulting in rather impressive peak numbers of over 1 TFLOP/s raw compute performance and 160 GByte/s of bandwidth to fast off-chip memory.

1.1 The memory wall problem

The sustained bandwidth to main memory and the latency to access data from main memory is much more important for FE codes than the raw compute performance. The discretisation process typically leads to large, but very sparse matrices, and the *arithmetic intensity* (defined as the ratio of floating point operations per memory access) of the codes is very low, a ratio ≤ 1 is common. Additionally, access to off-chip memory costs hundreds of clock cycles. We illustrate this well-known *memory wall problem* with a concrete example of a crucial component in the context of FE simulations: Sparse matrix vector multiply (`SpMV`, $\mathbf{y} = \mathbf{y} + \mathbf{A}\mathbf{x}$) for the compressed storage row (CSR) format in which all nonzero elements of the matrix $\mathbf{A}$ are stored contiguously in memory. (Throughout the paper we use bold upright letters to denote matrices and lowercase letters to denote vectors.) The array `colIndex` stores the column index of each element in the original matrix, and the array `rowStart` stores the beginning of each row. The `SpMV` kernel can thus be written in Fortran notation as:

```
do irow=1, nrows, 1
  do j=rowStart(irow), rowStart(irow+1), 1
     y(irow) = y(irow) + A(j)*x(colIndex(j))
  enddo
enddo
```

Modern commodity CPU cores can perform more than one floating point operation per cycle due to the on-chip vector units. With SSE2, a singlecore AMD Opteron 250 CPU as we use for our tests (see Section 4) is capable of performing two double precision floating point operations per cycle, resulting in a peak performance of 4.8 GFLOP/s at 2.4 GHz. With a peak memory bandwidth of 5.96 GB/s, the on-chip memory controller is capable of delivering at most 745 million double precision values per second. FE stencils couple several unknowns according to their geometric support, in the case of the conforming bilinear quadrilateral Q_1 FE we use throughout this paper, resulting in nine matrix entries for each degree of freedom. In the absence of cache memories for data reuse, the `SpMV` kernel has an arithmetic intensity of 2/5 (four loads, one store, one multiply and one add), as the values in the coefficient vector have to be accessed indirectly via the array `colIndex`. The (simplified) theoretical upper performance bound (long integer type for `colIndex`, same size as data) of `SpMV` is thus $745 \cdot 2/5 = 298$ MFLOP/s, roughly 6 % of the theoretical peak performance of the Opteron CPU. These results are in accordance with more detailed surveys of `SpMV` [26].

The traditional approach to alleviate the memory wall problem is the introduction of large on-chip cache memories that store a limited amount of data with access latency at least one order of magnitude lower (in comparison to main memory). For `SpMV`, there are still many compulsory misses, as typical problem sizes exceed the capacity of caches by far, so the bandwidth aspect of the memory wall problem is crucial. The matrix entries are only used once and then discarded, so data reuse is only possible by caching the coefficient vector $\mathbf{x}$ and the index vector `colIndex`.

1.2 Multicore and manycore trends

The parallelisation and specialisation of resources results in a major challenge for the programming model, in particular for clusters and supercomputers, where the coarse-grained parallelism (handled by message passing among distributed memories via MPI) must interact with the fine-grained on-chip parallelism and the heterogeneity on the nodes. Memory performance improves at a much slower pace than CPU performance and often scales only with the number of sockets per compute node rather than the number of cores [26]. 'Unconventional hardware' such as the above mentioned GPUs follow an entirely different approach: As an example, on NVIDIA's CUDA architectures, it still costs up to 1,000 clock cycles to access data in off-chip memory. These GPUs can 'hide' this latency via a very efficient lightweight thread scheduling mechanism implemented directly in hardware. The key to performance for kernels with low arithmetic intensity is thus their ability of keeping more than 30,000 threads 'in-flight' simultaneously, stalled threads waiting for memory transactions are suspended, and automatically resumed once the data is available. Through this approach, a significantly higher fraction

of the theoretical peak bandwidth can be achieved. In this work, we employ GPUs as a representative of future manycore chip designs.

1.3 Hardware-oriented numerics and related work

As with the coarse grained parallelism on the cluster level, compilers for standard languages are not expected to be able to parallelise existing data intensive codes efficiently in the medium term. This task requires explicit, global knowledge about the data flow, information that is hard to extract for compilers acting locally on the program instructions, or might even only be available at run-time. New languages and frameworks with data and task parallel intrinsics tailored to heterogeneous memory hierarchies (e. g., Sequoia [12]) can perform better in this situation. New languages, however, imply the reimplementation of significant portions of an application, which is prohibitive for established and actively used codes.

Our hypothesis is that in the field of high performance Finite Element simulations, significant performance improvements can only be achieved by *hardware-oriented numerics*. Numerical and algorithmic foundation research must accompany (long-term) technology trends; prospective hardware trends enforce research into novel numerical techniques that are in turn better suited for the hardware. As an example, multigrid smoothers with optimal numerical properties due to a strong recursive coupling often perform poorly in a parallel environment. Only correspondingly modified schemes that are potentially less efficient numerically (in terms of convergence rates) are able to achieve better overall performance in the user-relevant 'time-to-solution'-metric. The ultimate goal of our concept of hardware-oriented numerics is thus to balance these metrics to achieve robust and ideally predictable close-to-peak performance. In previous work, we have pursued these goals for the standard Poisson problem to focus on the general approach alone [2, 21, 25]. In this paper, we put our concept to the more general test by evaluating the performance of complex applications built on top of our solver toolkit FEAST.

Publicly available academic software packages simultaneously aiming at modern numerical methodology and efficient execution in parallel include PETSc and DUNE. Rüde et al. pursue an approach similar to ours, using patchwise multigrid smoothers and hierarchical hybrid grids [3]. A recent publication [17] reports good weak scalability for a 3D Poisson problem on a regularly refined unitcube domain for up to 9 170 cores (307 billion DOF).

Many publications discuss detailed (implementational) aspects of the memory wall problem for FE software, multigrid solvers and sparse matrices in general: Douglas and Thorne, and Douglas, Rüde et al. present cache-oriented multigrid solver components [9–11]. Butarri, Dongarra et al. discuss the impact of multicore architectures on mathematical software, in particular for (dense) BLAS and LAPACK-based applications [6]. Finally, Keyes and Colella et al. survey trends towards terascale computing for a wide range of applications including Finite Element software and conclude that only a combination of techniques from computer architecture, software engineering, numerical modeling and numerical analysis will enable a satisfactory scale-out on the application level [7, 20].

2 FEAST - FINITE ELEMENT ANALYSIS & SOLUTION TOOLS

FEAST is our next-generation simulation toolkit, implementing a wide range of hardware-oriented numerics concepts (see previous section). It is being developed as the successor of the established (and widely used in academia and industry) FE packages FEAT and FEATFLOW [24], see also `http://www.featflow.de`. Prior to describing the core features of FEAST, we should point out that even though we stated in the introduction that 're-implementations are prohibitive for large established codes,' we believe that individual aspects of our approach can be ported to other parallel FE toolkits with a reasonable effort. In particular, our approach does not imply the re-implementation of entire simulation

software packages in FEAST. What makes our approach novel is our holistic realisation of hardware-oriented numerics ranging from algorithmic and numerical foundation research via parallelisation and vectorisation, to unconventional hardware.

2.1 Separation of structured and unstructured data

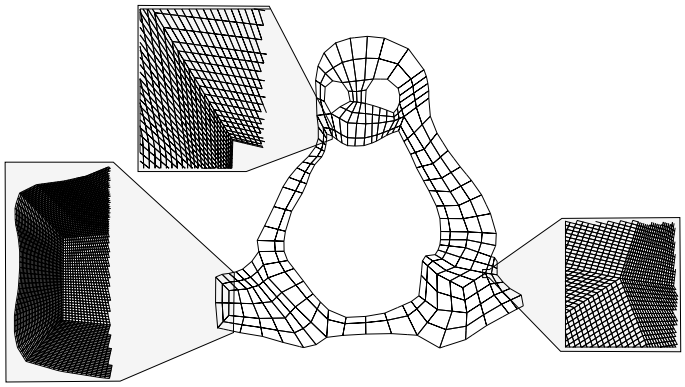

Figure 1: Exemplary `TUX` geometry. Important features of FEAST's discretisation approach are highlighted: Globally unstructured, locally structured grids (bottom left, bottom right); anisotropic refinement to, e. g., resolve boundary layers or shock fronts in CFD (microscopic anisotropies, top left); boundary adaption during refinement (bottom left); and coverage of large regions of minor interest with large, regularly refined subdomains (macroscopic feature size variations, bottom right).

FEAST covers the computational domain with a collection of quadrilateral subdomains. The subdomains form an unstructured coarse mesh (cf. Figure 1), and each subdomain is independently refined in a generalised tensor product fashion. Adaptive refinement strategies, that maintain the tensor product property of the mesh, are realised via grid deformation techniques (r-adaptivity, [18]), anisotropic refinement within each subdomain, and hanging nodes on subdomain edges. The resulting mesh is used to discretise the domain with Finite Elements. This approach caters to the contradictory needs of flexibility in the discretisation and efficient implementation: Instead of keeping all data in one general, homogeneous data structure, FEAST stores only local FE matrices and vectors (corresponding to subdomains, as usual for domain decomposition approaches) and thus maintains a clear separation of structured and unstructured parts of the domain. From an implementational point of view, global computations are performed by a series of local operations on matrices representing the restriction of the 'virtual' global matrix on each subdomain. Local information is exchanged only over boundaries of neighbouring subdomains. There is only an implicit overlap, the domain decomposition is implemented via special boundary conditions in the local matrices, a key technique to obtain good scalability. Several subdomains can be grouped together and treated within one MPI process.

The nonzero pattern of the local matrices is known a priori, which is exploited to optimise data structures and linear algebra routines. The structure of the local sub-problems allows a line-wise numbering of the unknowns, resulting in a banded matrix structure that can be stored as individual vectors per band. In view of the memory wall problem, it is worth mentioning that optimised linear algebra operations acting locally on the subdomains can thus be implemented with direct block memory transfers and without any pointer chasing. Our *Sparse Banded BLAS* library contains these optimised implementations for x86-based architectures, vector computers and recently also for GPUs (implemented in CUDA and OpenGL). Table 1 shows exemplary microbenchmark results for the `SpMV` kernel on the matrix resulting from discretising a unitsquare domain with conforming, bilinear quadrilateral Finite Elements from the Q_1 space, yielding $N = (2^{10} + 1)^2$ unknowns. We use five different numbering

schemes for a CSR format, and the band format from SbBlas. CPU timings are measured on an Opteron 250 CPU (see Section 4). The GPU results on a NVIDIA GeForce GTX 280 (much newer than the CPU) are shown for reference, as older GPUs do not provide double precision. The results indicate the superior performance of the SbBlas storage technique over the conventional CSR format, and illustrate the promising potential of GPUs for FE solvers beyond the marketing numbers quoted in the introduction. We refer to the thesis by one of the authors for updated timings on newer hardware [13].

numbering strategy	time (ms)	MFLOP/s
stochastic	348.8	54
two-level	141.2	134
Cuthill-McKee	133.6	141
hierarchical	117.9	160
XY	114.5	165
band	33.3	540
band-GPU	0.9	21313

Table 1: Microbenchmark results for the `SpMV` kernel in double precision.

2.2 Parallel multigrid solvers

In the parallelisation of Finite Element codes, numerical robustness, numerical efficiency and scalability are often contradictory properties. For the problems we are concerned with in the (wider) context of this paper, multigrid methods are obligatory from a numerical point of view. In parallel, the strong recursive character of 'optimal' serial multigrid smoothers (e. g., ILU) is usually relaxed to a block-Jacobi approach to decouple the subdomains, as strong recursion between the subdomains leads to poor scalability due to high communication requirements and an unfavourable ratio between computation and communication. On the other hand, block-Jacobi approaches are prone to degrade the global convergence in the presence of macroscopic (high aspect ratios between subdomains) and/or microscopic anisotropies (high element aspect ratios within subdomains), and anisotropies introduced by the operators. As a consequence, the numerical efficiency of the parallel solver is dramatically reduced [23, 25].

However, such blocking strategies are mandatory to achieve data locality (an amenable ratio of computation and communication) and thus good parallel efficiency. So, one of the main tasks of the numerical solution algorithm must be to minimise the negative effects of the block-Jacobi approach. Feast applies two strategies: On the one hand it employs powerful local smoothers that are able to 'hide' local irregularities from the outer solver as much as possible, and on the other hand the coarse grid solver of the global multilevel scheme provides sufficient coupling.

Our favorite local smoother for moderately complex scenarios is an alternating direction implicit linewise Gauß-Seidel smoother called `ADI-TRIGS` that uses the main diagonal, the first superdiagonal and all subdiagonals of the matrix. This smoother has much better numerical properties than 'standard' Jacobi, Gauss-Seidel or often even ILU smoothers [21]. By exploiting the fixed band pattern of the local matrices stemming from the underlying FE discretisation (see previous section), `ADI-TRIGS` has been implemented efficiently in the SbBlas library.

Using the global multilevel solver as preconditioner of a Krylov subspace method provides additional global coupling of the unknowns. Together with 'optimal step lengths' that are calculated within the Krylov-space method, this helps to greatly alleviate the negative effects of micro-/macro-anisotropies not captured by the block-Jacobi approach and thus increases the robustness of the parallel solution scheme. As a consequence the number of global iterations decreases significantly, resulting in a correspondingly reduced amount of communication.

For physically more complex scenarios that dynamically develop strongly localised effects like eddies in turbulent flow or shock fronts in compressible flow simulations, we are convinced that the two aforementioned strategies do not suffice to 'hide' these local irregularities from the global solver. We have therefore realised a more sophisticated strategy in terms of a cascaded multigrid/multilevel scheme, the so called ScaRC solvers (Scalable Recursive Clustering, [2,21,25]). The main idea is to replace the local smoothers of the global multigrid (e. g., `ADI-TRIGS`) by complete local multigrid cycles without additional communication requirements. Clustering of subdomains and adaptive adjustment of the local multigrid schemes provides great flexibility to resolve algorithmic or physical local irregularities even better.

2.3 Scalar and vector-valued problems

The guiding idea to treating vector-valued problems with Feast is to rely on the modular, highly optimised and fully 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 optimising it for new architectures. Vector-valued PDEs as they arise in CSM and CFD can be rearranged and discretised in such a way that the resulting discrete equation systems consist of blocks that correspond to scalar subequations. We illustrate this exemplarily in Section 3 with help of the elasticity equation.

This special block-structure can be exploited in two ways: On the one hand, all standard linear algebra operations on the vector-valued system (e. g., matrix-vector operations, defect computations, dot products) can be implemented as a series of operations for scalar systems, taking advantage of the highly tuned linear algebra components in Feast's SbBlas library. On the other hand, the process of solving the vector-valued linear equation system can be brought down to the treatment of auxiliary scalar subsystems which can be efficiently treated by Feast's optimised toolbox of parallel multigrid solvers. This procedure is facilitated by the use of special preconditioners that shift most of the overall work to such inner iterations. Once such a solution approach is set up, all further improvements of Feast's scalar solvers directly transfer to the solvers for vector-valued 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 next paragraph). ScaRC allows for an almost arbitrary (recursive) combination of vector-valued and scalar solvers, thus providing great flexibility in tuning the linear solution schemes to the given problem. For instance, a vector-valued Krylov-space scheme can use either a vector-valued multigrid cycle as preconditioner (which is in turn smoothed by scalar schemes) or directly scalar multigrids as block-preconditioner. Scalar solvers can be cascaded in the same way. Concrete examples of the solvers are presented in Sections 4 and 5.

2.4 Co-processor acceleration

Feast's modular solver structure has the additional benefit that it facilitates the inclusion of hardware co-processors. The abstraction layer of the suggested 'minimally invasive' integration [13, 15] encapsulates heterogeneities of the system on the node level, so that MPI sees a globally homogeneous system, while the local solver components encapsulate the heterogeneity within the node. To benefit from co-processor acceleration, application code does not need to be changed at all. The reduced precision of several co-processor architectures is addressed by a mixed precision iterative refinement scheme. The current implementation supports GPUs, and a Cell backend is being developed.

This scheme is especially beneficial for cascaded ScaRC solvers: Instead of only accelerating individual linear algebra operations, we can accelerate entire multigrid solvers for local subproblems. This concentrates sufficient fine-grained parallelism in a separate task and thus minimises the potential overhead of repeated co-processor configuration and data transfer in case of a system integration via

relatively narrow busses such as PCIe. In the current implementation, load balancing is done statically in a preprocessing step.

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-solid-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 Ω 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 [27]. However, in this paper we do not use these advanced problem-specific features and only treat the simple yet fundamental linearised 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 application.

Mathematically, we formulate the linearised 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 linearised strain tensor $\varepsilon_{ij} = \frac{1}{2}\left(\frac{\partial u_i}{\partial x_j} + \frac{\partial u_j}{\partial x_i}\right), \quad i, j = 1, 2$, describing the linearised 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, which are connected to the Young modulus E and the Poisson ratio ν as follows:

$$\mu = \frac{E}{2(1+\nu)}, \qquad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \tag{1}$$

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: $-\mathbf{div}\boldsymbol{\sigma} = \boldsymbol{f}$ for $\boldsymbol{x} \in \Omega$. Using Hooke's law to replace the stress tensor, the problem of linearised elasticity can be expressed in terms of the following elliptic boundary value problem, called the Lamé equation:

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

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

$$\boldsymbol{\sigma}(\boldsymbol{u}) \cdot \boldsymbol{n} = \boldsymbol{t}, \qquad \boldsymbol{x} \in \Gamma_\mathrm{N} \tag{2c}$$

Here, $\boldsymbol{g}$ are prescribed displacements on Γ_D, and $\boldsymbol{t}$ are given surface forces on Γ_N with outer normal $\boldsymbol{n}$. For details on the elasticity problem, see for example the textbook by Braess [4].

In order to solve vector-valued linearised elasticity problems using the FEAST intrinsics described in the previous paragraphs, the resulting degrees of freedom have to be ordered corresponding to the spatial directions, a technique called separate displacement ordering [1]. In the 2D case, the unknowns

$\boldsymbol{u} = (u_1, u_2)^\mathsf{T}$ correspond to displacements in x and y-direction. Rearranging the left hand side of equation (2a) 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 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 discretisation with conforming bilinear elements Q_1. This leads to the linear equation system $\mathbf{Ku} = \mathbf{f}$ exhibiting the following 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 FE solution. 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). Note, that 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 details of the overall solution process are now illustrated by means of a basic preconditioned defect correction method:

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

This scheme acts on the global system (4) and thus couples the two sets of unknowns $\mathbf{u}_1$ and $\mathbf{u}_2$. $\tilde{\mathbf{K}}_{\text{BGS}}$ 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):

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

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:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \omega\mathbf{c}$$

3.2 Computational fluid dynamics

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

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 equations

is obtained under the assumption of constant temperature ϑ and constant kinematic viscosity $\nu > 0$, $\nu \neq \nu(p, c_p)$:

$$-\nu\Delta\boldsymbol{u} + (\boldsymbol{u}\cdot\nabla)\boldsymbol{u} + \nabla p = \boldsymbol{f}, \qquad \boldsymbol{x} \in \Omega \tag{6a}$$

$$\operatorname{div}\boldsymbol{u} = \boldsymbol{0}, \qquad \boldsymbol{x} \in \Omega \tag{6b}$$

$$\boldsymbol{u} = \boldsymbol{g}, \qquad \boldsymbol{x} \in \Gamma_{\mathrm{D}} \tag{6c}$$

$$\nu\partial_n\boldsymbol{u} + p\cdot\boldsymbol{n} = 0, \qquad \boldsymbol{x} \in \Gamma_{\mathrm{N}} \tag{6d}$$

where p denotes 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 discretise the equation with the Q_1/Q_1 bilinear element pair and use residual-based SUPG/PSPG stabilisation (streamline upwind/Petrov-Galerkin and pressure-stabilisation/Petrov-Galerkin) to account for both the LBB deficiency of the Q_1/Q_1 pair and to stabilise convective terms [5,19]. To allow for anisotropic elements directional derivatives are incorporated in the stabilisation terms. The nonlinearities are resolved by a fixed point defect correction method such that in each nonlinear iteration a linear saddle point system is to be solved,

$$\begin{pmatrix} \mathbf{A}+\mathbf{C_1} & \mathbf{B} \\ \mathbf{B}^{\mathsf{T}} & \mathbf{C_2} \end{pmatrix} \begin{pmatrix} \mathbf{u} \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ \mathbf{g} \end{pmatrix}, \tag{7}$$

where the matrices $\mathbf{C_1}, \mathbf{C_2}$ stem from the SUPG/PSPG stabilisation terms. Our solution algorithm is a block Schur complement (BSC) approach as described by Murphy et al. [22]. It basically consists of a global BiCGStab solver acting on the whole saddle point system which is block-preconditioned by

$$\begin{pmatrix} \tilde{\mathbf{A}} & \mathbf{0} \\ \mathbf{B}^{\mathsf{T}} & \tilde{\mathbf{S}} \end{pmatrix}. \tag{8}$$

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 realised 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 realised by a suitably weighted linear combination of pressure mass (and Laplace matrices in case of nonstationary problems, see Turek [24]). The resulting scalar system can again be solved with FEAST's optimised multigrid schemes, such that the solution of the whole saddle point system is mainly brought down to the solution of scalar systems again.

4 PERFORMANCE AND SCALABILITY STUDIES

For the tests presented in the next sections, we use three different HPC installations. LiDO, though slightly outdated, is a typical representative of a commodity based cluster with fast interconnects, NEC SX-8 is a dedicated HPC system, and USC is a test installation of a heterogeneous cluster enhanced with GPUs. Table 2 lists the specifications of the machines.

All tests in this section are performed on up to 64 (out of 72) nodes of the LiDO cluster. We use the 'showcase' `TUX` geometry displayed in Figure 1 and create coarse grids comprising 32, 64, 128, 256, 512 and 1024 subdomains each. As mentioned before, the coarse grids are in parts unstructured and exhibit a moderate degree of macroscopic anisotropy. We apply anisotropic refinement towards the 'belly' of the `TUX` in such a way that in each refinement step, the innermost layer of elements is positioned with a ratio of 2:3 (1:1 denotes regular refinement), leading to a total microscopic anisotropy of 10–100 for the finest level of refinement. We only consider refinement levels $L = 9$ and $L = 10$. The resulting mesh is discretised with bilinear conforming Finite Elements from the Q_1 space, resulting in global problem sizes ranging from 8.4 M to 270 M ($L = 9$) and 34 M to 1 B ($L = 10$) mesh nodes.

	LiDO	NEC SX-8	USC
CPU type	AMD Opteron DP 250	SX-8 vector CPU	AMD Opteron 2210 NVIDIA Quadro FX 5600 GPU
CPU details	2.4 GHz, 1 MB L2	2 GHz	1.8 GHz, 2 MB L2
CPU count	2	8	1 (dualcore)
Memory	8 GB DDR 400	128 GB	8 GB DDR2 667
Memory BW	2 * 5.96 GB/s	8 * 64 GB/s	6.4 GB/s
Interconnect type	DDR Infiniband	NEC IXS	4xDDR Infiniband
Interconnect latency	approx. 3 μs	approx. 5 μs	approx. 3 μs
Interconnect BW	approx. 750 MB/s	16 GB/s	approx. 1.2 GB/s
Installation	ITMC, TU Dortmund	HLRS, Stuttgart	Los Alamos National Laboratory

Table 2: Detailed specifications (per node) of the machines used for performance and scalability tests.

On these domains, we solve the Poisson problem as a standard benchmark configuration, and exemplary simulations with the applications FeastSolid and FeastFlow. For the Poisson problem, we have verified correctness by prescribing the analytical Laplacian of some polynomial test function as right hand side to enable computation of the reduction of the discretisation error with increasing refinement. Figure 2 shows the computed von Mises stresses for the deformed `TUX` and the computed velocity field for a 'flow through the `TUX`' simulation at Reynolds number $Re = 200$.

For the moderate anisotropies present in the test domain, a BiCGStab solver on the finest level, preconditioned with a standard data-parallel multigrid cycle using the strong `ADI-TRIGS` smoother (MG-ADITRIGS), configured to perform four pre- and postsmoothing steps in an F-cycle, is the best compromise between fast convergence and time to solution. Therefore, we use this solver configuration throughout this section for the Poisson problem. FeastSolid shifts the Krylov-space iteration to the vector-valued problem, and applies the scalar MG-ADITRIGS in its block-Gauss-Seidel preconditioner, see Section 3.1 for details. FeastFlow does likewise for the velocity-related subproblems while separate calls of the scalar MG-ADITRIGS are required for the Schur complement preconditioner (see Section 3.2). The goal in all tests is to reduce initial residuals by six digits, with the exception of eight digits being used as nonlinear convergence criterion in FeastFlow. All coarse grid problems are solved very efficiently with UMFPACK [8].

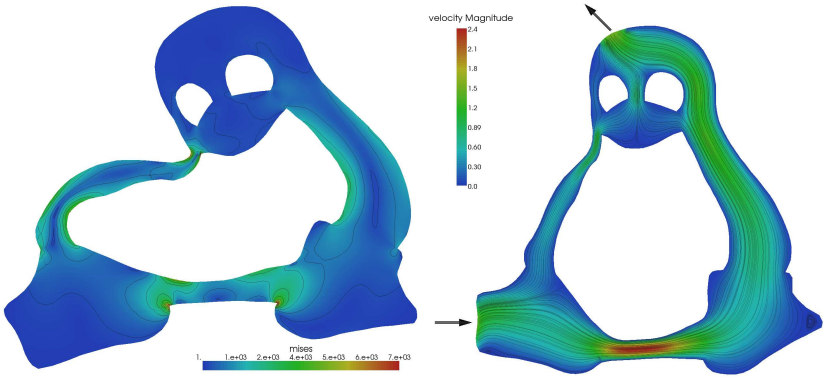

Figure 2: Computed results: Displacements and von Mises stresses for the `TUX` body under gravity with FeastSolid, flow through the `TUX` body with FeastFlow.

4.1 Weak scalability and absolute performance

Table 3 summarises the number of solver iterations required until convergence, and the timings and MFLOP/s rates (including assembly times only for the FeastFlow application, excluding pre- and postprocessing for all applications) for a weak scalability analysis where we double the problem size

and the number of CPUs simultaneously. Due to space constraints, we only present results for refinement levels $L = 9, 10$. On smaller levels of refinement, we observe a different behaviour as caching effects on the CPU start to play a dominant role, significantly changing the ratio of computation to communication. But these small problem sizes are not interesting in a weak scalability analysis (and especially in practice), which is gearing towards the largest problem size possible.

As the degrees of freedom coincide with the mesh nodes for the Q_1 FE, the largest problem sizes on 128 CPUs are thus one billion unknowns for the Poisson and FeastSolid applications (with up to 1024 and 512 subdomains fitting into memory respectively). Due to memory constraints, we are only able to execute the FeastFlow application on refinement level $L = 9$, yielding 100 M unknowns.

	Poisson			FeastSolid			FeastFlow			
CPUs	Iters	Time	MFLOP/s	Iters	Time	MFLOP/s	Nonlin	Iters	Time	MFLOP/s
8	5.0	76.0	3128	25.5	409.5	3205	8	210	3460	2720
16	5.5	84.2	6220	23.5	384.4	6294	10	303	5037	5388
32	5.5	85.0	12323	24.5	407.4	12382	8	282	4872	10469
64	5.5	91.2	22970	23.5	402.3	24054	10	391	7215	19530
128	5.0	84.6	45013	23.5	424.2	45624	9	335	9426	38197

Table 3: Weak scalability results on LiDO, refinement level $L = 10$ ($L = 9$ for FeastFlow). The fractional notation is a consequence of the outermost BiCGStab solver permitting an 'early exit' after its first preconditioning step.

The performance results are very satisfactory. The global coarse grid solver, which runs on the master process and thus constitutes the sequential part in the otherwise fully data-parallel execution, contributes less than 1 % to the overall runtime; in fact, for the Poisson test on 128 CPUs, the accumulated idle time of each compute process during coarse grid solves on the master process does not exceed 0.6 seconds. We conclude that the global coarse grid solver will not become the bottleneck in terms of scalability in the medium term. The per-CPU performance of 350–400 MFLOP/s may sound comparatively low at first, but the underlying timing includes all stalls due to communication across subdomain edges, and in our performance model, we only count mathematically necessary operations, in other words, we explicitly exclude all operations (but not the timings) due to the parallel execution. If we consider the MFLOP/s rate for the serial matrix-vector multiplication (cf. Table 1) as an upper limit of achievable performance, and assume that for a Jacobi smoother the same limit holds as for the plain matrix-vector multiplication, then detailed measurements show that the per-CPU performance only increases by approximately 30 MFLOP/s instead of the expected 150 MFLOP/s, confirming that all local operations are performed within 5-10 % of the attainable peak memory bandwidth and the difference can be fully attributed to our performance model.

The relatively high iteration numbers of the elasticity solver (compared to the Poisson solver) are a consequence of the `TUX` geometry and the used boundary conditions: Long and thin structures in conjunction with relatively small Dirichlet boundaries (only the `TUX` bottom between the feet is fixed) leads to badly conditioned stiffness matrices and thus to an increased number of Krylov-space iterations (cf. Axelsson [1]). FeastFlow exhibits consistently lower MFLOP/s rates, because computations are performed for refinement level $L = 9$, and the arithmetic operations performed during matrix assembly are not included. Due to the approximate solution of the subproblems, the nonlinear solver reacts sensitively to the different partitioning of the domain into subdomains, resulting in a different convergence behaviour and thus different runtimes. Our solvers scale very well for all three applications, Figure 3 depicts normalised timings per iteration (FeastFlow is normalised with the number of Schur complement solves).

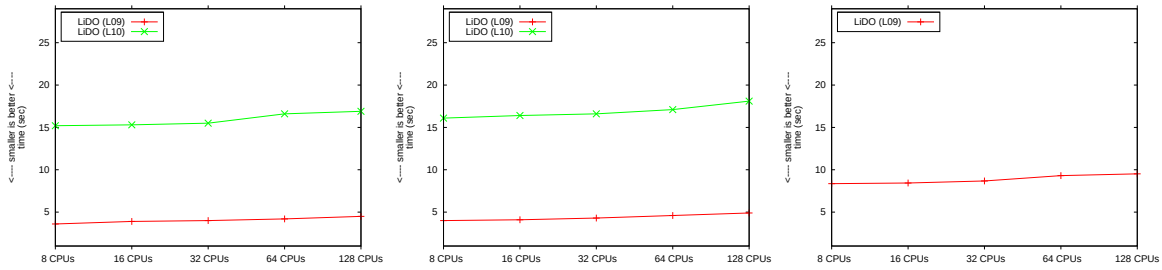

Figure 3: Weak scalability results, normalised to time per iteration.

4.2 Strong scalability

We base our evaluation of strong scalability on refinement level $L = 9$, which increases the exploration space due to the reduced memory imprint of the subdomains compared to $L = 10$. Starting from a distribution of subdomains to CPUs that uses all available memory on a given number of nodes, we reschedule subdomains to twice, four times and eight times as many nodes. Figure 4 illustrates the results. For the Poisson and FeastSolid applications, we observe near-perfect strong scalability, only after eight times the number of nodes, performance starts to degrade slightly, indicating that communication requirements start to influence performance. The CFD application FeastFlow again exhibits sensitivity to the different partitions of the domain, but strong scaling remains very good.

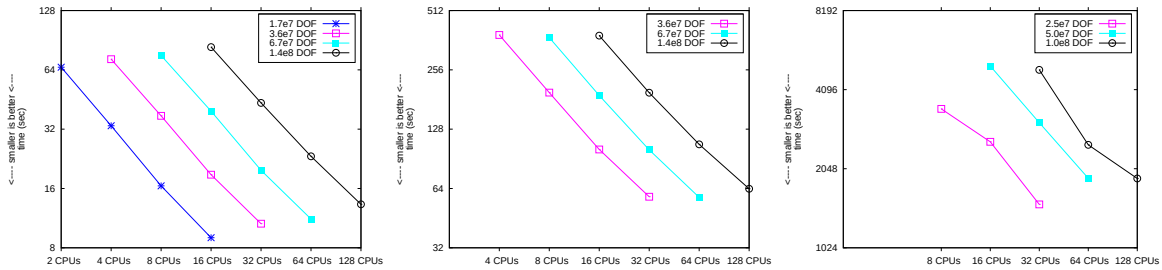

Figure 4: Strong scalability results for the Poisson, FeastSolid and FeastFlow applications.

4.3 Validation of FEAST's solver concept

As explained in Section 2.3, one aspect of Feast's approach is to reduce complex applications, vector-valued problems, and nonlinear systems to sequences of scalar solves. This modular design allows tuning of kernel features such that applications can directly benefit.

To quantify the implications of this approach, we execute the Poisson application on exactly the same input mesh and (static) partitioning into parallel jobs as FeastSolid. We use the solvers described at the beginning of this section. One iteration of the BiCGStab solver for the Poisson problem costs two scalar multigrid cycles, whereas one iteration of the vector-valued solver in FeastSolid needs four scalar multigrid cycles plus the overhead of performing one scalar matrix-vector multiplication in its block-preconditioner and the matrix-vector and vector-vector operations of the BiCGStab iterations. For the Poisson application on refinement level $L = 9$, we measure 1.93, 1.98, 2.06, 2.22 and 2.40 seconds per iteration for 8–128 CPUs respecively, and compared to the normalised numbers from Figure 3 (center), this results in a performance difference of factors 2.09, 2.08, 2.08, 2.05 and 2.05 respectively. These numbers confirm that more than 90 % of the work is done inside the scalar solvers, which is the behaviour we aimed at in our solver design. Detailed measurements for the FeastFlow

application show that 83 to 85 % of total linear solver time is spent in the scalar solver, independent of the number of processors, and that MFLOP/s rates are reobtained (cf. Table 3).

5 PERFORMANCE STUDIES ON ADVANCED ARCHITECTURES

Due to space constraints, we do not present a detailed performance study on NEC SX-8 and on USC as we did on LiDO in the previous section. The sketched results for the GPU-enhanced solver are similar to a previously published paper focussing exclusively on FEAST's GPU backend [16], note that we use a slightly different solver configuration in this paper to allow a better comparison. For the same reason, we perform the exact same experiments on NEC SX-8. Though brief, the results in this section underline important benefits of the approach we describe in this paper.

The GPU solver (cf. Section 2.4) is prototypical in the sense that it only provides a Jacobi smoother, more advanced smoothers have not been implemented at the time these tests were performed. The degree of anisotropy (both in the underlying mesh and the operators) is thus limited. Figure 5 illustrates four prototypical test problems and the deformations and von Mises stresses computed with the unmodified, yet GPU-accelerated FEASTSOLID application. We use a solver configuration with a BiCGStab scheme as outermost vector-valued solver, apply a block-Jacobi preconditioner to reduce the problem to scalar subblocks, which are 'inverted' with one iteration of a data-parallel scalar multigrid to couple across the entire domain, smoothed by local multigrids treating each subdomain individually. All 64 subdomains are refined to level $L = 10$, leading to a total problem size of 134 million DOF. USC has 16 nodes, and on NEC SX-8, we use two full nodes. Figure 6 summarises the obtained timing measurements.

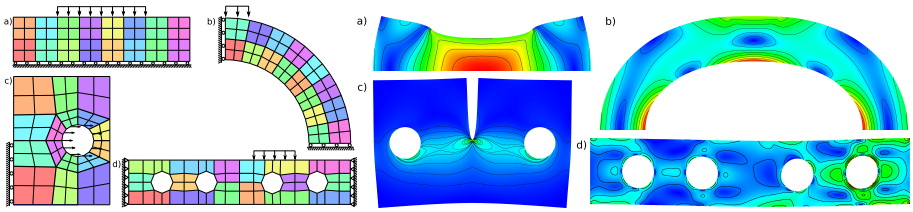

Figure 5: Test problems in linearised elasticity, computed with the GPU-accelerated FEASTSOLID application ((a) `BLOCK`, (b) `PIPE`, (c) `CRACK`, (d) `STEELFRAME`).

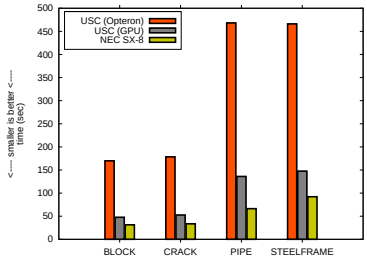

Figure 6: Timing comparison of FEASTSOLID, on USC (with and without GPUs) and NEC SX-8. The longer computation times of the `PIPE` and `STEELFRAME` configurations are a consequence of the long and thin structures in conjunction with small Dirichlet boundaries, see Section 4.1.

The results underline a very important aspect of FEAST's general approach: To benefit significantly from hardware acceleration (via GPUs in a commodity based cluster, or on a dedicated HPC system), not a single line of the application code has to be changed. The GPU-enhanced cluster executes the test problems between 3.2 and 3.5 times faster than the Opteron cluster, and the NEC SX-8 is more

than 5 times faster. For a detailed discussion of the factors limiting performance in the GPU, and a scalability analysis of the GPU-enhanced solver, we refer to previous publications [14,16]. Performance on NEC SX-8 is overall fastest, but still comparatively low compared to the peak capabilities due to the small local problem size of $L = 10$. Preliminary experiments indicate that the achievable MFLOP/s per CPU increase almost linear upon further refinement.

6 CONCLUSIONS AND FUTURE WORK

Looking at current trends in numerical simulation methods and high performance computing techniques, we believe that dedicated hardware-oriented numerics is a key tool to significantly improve FE software packages for highly challenging current and future real world problems. The previous certitude, still common among many developers focussing only on numerical aspects, that with every new processor generation existing codes automatically run faster without any modification, is not valid anymore. To benefit from this paradigm change in hardware, a change in software architecture is a must to exploit the performance of current and future multicore and manycore designs, as well as specialised hardware.

In our contribution, we have outlined FEAST, our next-generation solver toolkit that implements concepts of hardware-oriented numerics. The underlying ideas are not specific to FEAST, in fact, not all of them are novel per se, so that developers of numerical software should be able to include them in their software. We supported these concepts by detailed numerical and performance experiments, as well as scalability analyses for elliptic model problems and full applications built on top of FEAST.

Future work includes the development of a multi-level scheduling scheme between nodes, sockets and cores and the evaluation and addition of more dedicated hardware architectures such as the Cell processor. Our approach is independent of the dimensionality, so adding 3D support is rather an implementational burden than a conceptual challenge.

ACKLOWLEDGEMENTS

Parts of this work are based on a joint collaboration with Robert Strzodka (Max Planck Center), and Patrick McCormick and Jamaludin Mohd-Yusof (Los Alamos National Laboratory). Thanks to Matthias Grajewski, Michael Köster and Thomas Rohkämper for co-developing FEAST and its toolchain and many discussions. Thanks to NVIDIA for donating hardware that was used in the development of FEAST's GPU backend. Thanks to the ITMC team, TU Dortmund, for granting exclusive access to LiDO's InfiniBand subcluster; and to Heinz Pöhlmann, Uwe Küster and Michael Resch (HLRS Stuttgart) for help using the NEC SX8 machine. This work has been partially supported by Deutsche Forschungsgemeinschaft (DFG), grants TU 102/22-1, TU 102/22-2, TU 102/27-1, TU 102/11-3, and Bundesministerium für Bildung und Forschung (BMBF) in call HPC-Software für skalierbare Rechnersysteme, SKALB Project (01IH08003D / SKALB).

References

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

[2] C. 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] B. Bergen, F. Hülsemann, and U. Rüde. Is 1.7×10^{10} unknowns the largest finite element system that can be solved today? In *SC '05: Proceedings of the 2005 ACM/IEEE conference on Supercomputing*, page 5, Nov. 2005.

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

[5] A. N. Brooks and T. 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, Sept. 1982.

[6] A. Buttari, J. J. Dongarra, J. Kurzak, J. Langou, P. Luszczek, and S. Tomov. The impact of multicore on math software. In *Proceedings of PARA 2006, Applied Parallel Computing. State of the Art in Scientific Computing*, volume 4699 of *Lecture Notes In Computer Science*, pages 1–10. Springer, 2006.

[7] P. Colella, T. H. D. Jr., W. D. Gropp, and D. E. Keyes. A science–based case for large–scale simulation. Technical report, Office of Science, US Department of Energy, July 2003. `http://www.pnl.gov/scales`.

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

[9] C. C. Douglas, J. Hu, W. Karl, M. Kowarschik, U. Rüde, and C. Weiß. Fixed and adaptive cache aware algorithms for multigrid methods. In E. Dick, K. Riemslagh, and J. Vierendeels, editors, *Multigrid Methods VI*, volume 14 of *Lecture Notes in Computational Science and Engineering*, pages 87–93. Springer, 2000.

[10] C. C. Douglas, J. Hu, M. Kowarschik, U. Rüde, and C. Weiß. Cache optimization for structured and unstructured grid multigrid. *Electronic Transactions on Numerical Analysis*, 10:21–40, Feb. 2000.

[11] C. C. Douglas and D. T. Thorne. A note on cache memory methods for multigrid in three dimensions. *Contemporary Mathematics*, 306:167–177, 2002.

[12] K. Fatahalian, T. Knight, M. Houston, M. Erez, D. R. Horn, L. Leem, J. Y. Park, M. Ren, A. Aiken, W. J. Dally, and P. Hanrahan. Sequoia: Programming the memory hierarchy. In *SC '06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, Nov. 2006. Article No. 83.

[13] D. Göddeke. *Fast and Accurate Finite-Element Multigrid Solvers for PDE Simulations on GPU Clusters*. PhD thesis, TU Dortmund, Fakultät für Mathematik, 2010. to be published.

[14] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, S. H. Buijssen, M. Grajewski, and S. Turek. Exploring weak scalability for FEM calculations on a GPU-enhanced cluster. *Parallel Computing*, 33(10–11):685–699, Sept. 2007.

[15] D. Göddeke, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, H. Wobker, C. Becker, and S. Turek. Using GPUs to improve multigrid solver performance on a cluster. *International Journal of Computational Science and Engineering*, 4(1):36–55, Nov. 2008.

[16] D. Göddeke, H. Wobker, R. Strzodka, J. Mohd-Yusof, P. S. McCormick, and S. Turek. Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU. *International Journal of Computational Science and Engineering*, 4(4):254–269, Oct. 2009.

[17] T. Gradl and U. Rüde. High performance multigrid in current large scale parallel computers. In W. E. Nagel, R. Hoffmann, and A. Koch, editors, *9th Workshop on Parallel Systems and Algorithms (PASA)*, volume 124 of *GI Edition: Lecture Notes in Informatics*, pages 37–45, Feb. 2008.

[18] M. Grajewski. *A new fast and accurate grid deformation method for r–adaptivity in the context of high performance computing*. PhD thesis, TU Dortmund, Fakultät für Mathematik, Mar. 2008. `http://www.logos-verlag.de/cgi-bin/buch?isbn=1903`.

[19] T. J. Hughes, L. P. Franca, and M. 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, Nov. 1986.

[20] D. E. Keyes. Terascale implicit methods for partial differential equations. In X. Feng and T. P. Schulze, editors, *Recent Advances in Numerical Methods for Partial Differential Equations and Applications*, volume 306 of *Contemporary Mathematics*, pages 29–84. American Mathematical Society, Jan. 2002.

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

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

[23] B. F. Smith, P. E. Bjørstad, and W. D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, June 1996.

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

[25] S. Turek, C. Becker, and S. Kilian. Hardware–oriented numerics and concepts for PDE software. *Future Generation Computer Systems*, 22(1-2):217–238, Feb. 2004.

[26] S. Williams, L. Oliker, R. Vuduc, J. Shalf, K. Yelick, and J. W. Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. In *SC '07: Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, Nov. 2007. Article No. 38.

[27] H. 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, Dec. 2009.