This is a preprint of an article accepted for publication in:
High Performance Computing in Science and Engineering '08,
Transactions of the High Performance Computing Center, Stuttgart (HLRS) 2008

FEASTSolid and FEASTFlow: FEM applications exploiting FEAST's HPC technologies

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

Institute for Applied Mathematics and Numerics,
Dortmund University of Technology, 44227 Dortmund, Germany
feast@math.tu-dortmund.de

1 Introduction and Motivation

Finite Element (FE) codes typically operate on sparse matrices and feature low arithmetic intensity, resulting in their performance being limited by the available memory bandwidth rather than the peak compute performance. FEAST (Finite Element Analysis and Solution Tools) is our toolkit providing FE discretisations and corresponding optimised parallel multigrid solvers for PDE problems, addressing this *memory wall problem* with what we call "hardware-oriented numerics" [11]. These techniques allow FEAST to exploit a significant share of modern processors' peak performance for FE applications while maintaining numerical efficiency, robustness and flexibility. Last year we reported on our efforts solving Poisson problems with FEAST on NEC SX-6 and SX-8, JUMP Jülich and commodity based clusters [4]. In this paper, we address our progress in solving problems from solid mechanics and fluid dynamics with FEAST. We only briefly summarise the main ideas here (cf. Fig. 1), and refer to previous publications for related work and more details.

The two main principles underlying our approach are:

Logical tensorproduct structure: In FEAST, the discretisation is closely coupled with the domain decomposition for the parallel solution. The computational domain $\overline{\Omega}$ is covered with a collection of quadrilateral subdomains $\overline{\Omega_i}$. The subdomains form an unstructured coarse mesh and are hierarchically refined such as to preserve a logical tensorproduct structure of the mesh cells within each subdomain. Consequently, FEAST maintains a clear separation of globally unstructured and locally structured data. The resulting mesh is used for the discretisation with Finite Elements, and linewise numbering of the unknowns leads to band structured matrices.

SBBLAS: Since the underlying data structures store matrix bands as sequential vectors, there is no need for general storage formats such as CSR. Consequently, matrix-vector multiplication can be implemented bandwise, entirely without indirect addressing (*Sparse Banded BLAS*). On cache-based architectures, only slices of the complete diagonals (cf. Fig. 1) are operated on simultaneously which allows for a greater part of the result vector being held in cache. On non von Neumann

architectures such as the NEC SX-8, matrix-vector multiplication can be efficiently vectorised due to this blocking strategy.

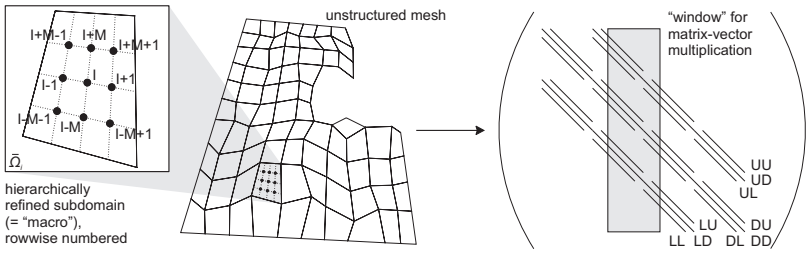

Fig. 1. Grid with logical tensorproduct structure, exemplary one coarse grid cell hierarchically refined. Matrix vector multiplication in SBBLAS operates only on slices of the corresponding FE band matrix

2 Computational Solid Mechanics

In Computational Solid Mechanics (CSM), the deformation of solid bodies under external loads is simulated. In this report, we prototypically 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. We treat the simple, but nevertheless fundamental, model problem of elastic, compressible material under static loading, assuming small deformations. We use a formulation where the displacements $u(x) = \big(u_1(x), u_2(x)\big)^{\mathsf{T}}$ of a material point $x \in \bar{\Omega}$ are the only unknowns in the equation. 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 materials, $\sigma = 2\mu\varepsilon + \lambda\,\mathrm{tr}(\varepsilon)I$, where σ denotes the symmetric stress tensor and μ and λ are the so-called Lamé constants.

The basic physical equations for problems of solid mechanics are determined by equilibrium conditions. For a body in equilibrium, the inner forces (stresses) and the outer forces (external loads f) are balanced:

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

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:

$$-2\mu\,\mathrm{div}\varepsilon(u) - \lambda\,\mathrm{grad}\,\mathrm{div}\,u = f, \qquad x \in \Omega \tag{1a}$$
$$u = g, \qquad x \in \Gamma_\mathrm{D} \tag{1b}$$
$$\sigma(u)\cdot n = t, \qquad x \in \Gamma_\mathrm{N} \tag{1c}$$

Here, g are prescribed displacements on Γ_D, and t are given surface forces on Γ_N with outer normal n. For details on the elasticity problem, see for example Braess [5].

3 Computational Fluid Dynamics

We model problems from Computational Fluid Dynamics (CFD) with the Navier–Stokes equations, which describe the flow of incompressible Newtonian fluids (e. g. water and many other liquids) in a domain Ω.

Confining the domain and imposing boundary conditions, i. e. in- and outflow conditions on the "artificial" boundaries and no-slip conditions at rigid walls, yields the following system of equations under the assumption of constant temperature ϑ and constant kinematic viscosity $\nu > 0$:

$$-\nu\Delta u + (u\cdot\mathrm{grad})u + \mathrm{grad}\,p = f, \qquad x \in \Omega \tag{2a}$$
$$\mathrm{div}\,u = 0, \qquad x \in \Omega \tag{2b}$$
$$u = g, \qquad x \in \Gamma_\mathrm{D} \tag{2c}$$
$$\nu\partial_n u + p\cdot n = 0, \qquad x \in \Gamma_\mathrm{N} \tag{2d}$$

where p denotes pressure, 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). For more details on the theoretical background of this, see for example Ferziger and Perić [7].

4 Solution Strategy

4.1 Parallel Multigrid Solvers in FEAST

For the problems we are concerned with in the (wider) context of this report, multigrid methods are obligatory from a numerical point of view. When parallelising multigrid methods, numerical robustness, numerical efficiency and (weak) scalability are often contradictory properties: A strong recursive coupling between the subdomains, for instance by the direct parallelisation of ILU-like smoothers, is advantageous for the numerical efficiency of the multigrid solver. However, such a coupling increases the communication and synchronisation requirements significantly and is therefore bound to scale badly. To alleviate this high communication overhead, the recursion is usually relaxed to the application of *local smoothers* that act on each subdomain $\overline{\Omega_i}$ independently. The contributions of the separate subdomains are combined in an additive manner only after the smoother has been applied to all

subdomains, without any data exchange during the smoothing. The disadvantage of such a (in terms of domain decomposition) *block-Jacobi* coupling is that typical local smoothers such as Gauss-Seidel are usually not powerful enough to treat, for example, local anisotropies. Consequently, the numerical efficiency of the multigrid solver is dramatically reduced [9, 11].

To address these contradictory needs, FEAST employs a generalised multigrid domain decomposition concept called SCARC (Scalable Recursive Clustering). The basic idea is to apply a *global*, data-parallel multigrid algorithm which is smoothed in an additive manner by *local* multigrids acting on each subdomain independently. In the nomenclature of the previous paragraph, this means that the application of a local smoother translates to performing few iterations – sometimes even only one iteration – of a local multigrid solver, and we can use the terms *local smoother* and *local multigrid* synonymously. This cascaded multigrid scheme is very robust as local irregularities are 'hidden' from the outer solver, the global multigrid provides strong global coupling (as it acts on all levels of refinement), and preserves the scalability of data-parallel multigrid methods by design. Obviously, this cascaded multigrid scheme is prototypical in the sense that it can only show its full strength for reasonably large local problem sizes and locally ill-conditioned systems [3].

Instead of keeping all data in one general, homogeneous data structure, FEAST stores only the local FE matrices and vectors, corresponding to the local subdomains (which is common in domain decomposition methods). Global matrix-vector operations are performed by a series of local operations on matrices representing the restriction of the 'virtual' global matrix on each subdomain. These operations are *directly* followed by exchanging information via MPI over the boundaries of neighbouring subdomains. There is only an implicit subdomain overlap; the domain decomposition is implemented via special boundary conditions in the local matrices [3]. Several subdomains are typically grouped into one MPI process, exchanging data via shared memory. All global and local coarse grid problems are solved exactly by a tuned direct LU decomposition solver taken from UMFPACK [6].

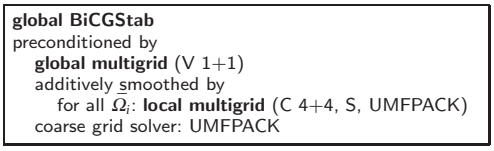

Fig. 2. Illustration of the family of cascaded multigrid solver schemes in FEAST

Figure 2 illustrates a typical solver in FEAST. The notation 'local multigrid (C 4+4, S, UMFPACK)' denotes a multigrid solver on a single subdomain, configured to perform the cycle $C \in \{\text{V,F,W}\}$ with 4 pre- and postsmoothing steps with the smoothing operator $S \in \{\text{Jacobi, Gauss-Seidel, ILU}, \ldots\}$. To improve solver robustness, the global multigrid solver is used as a preconditioner to a Krylov subspace solver such as BiCGStab which executes on the global fine grid. As a preconditioner, the global multigrid performs exactly one iteration without convergence control.

We finally emphasise that the entire concept – comprising domain decomposition, solver strategies and data structures – is independent of the spatial dimension of the underlying problem. Implementation of 3D support is tedious and time-consuming, but conceptually straightforward.

4.2 Scalar and Vector-Valued Problems

The guiding idea to treating vector-valued problems with FEAST is to rely on the modular, reliable and highly optimised scalar operations, in order to formulate robust schemes for a wide range of applications rather than using the best suited numerical scheme for each application and go through the optimisation and debugging process over and over again. Vector-valued PDEs as they arise, for instance, in the application domains in the focus on this report, can be rearranged and discretised in such a way that the resulting discrete systems of equations consist of blocks that correspond to scalar problems (for the CSM case see beginning of Sect. 4.3, for CFD see Sect. 4.4). Due to this special block-structure, all operations required to solve the systems can be implemented as a series of operations for scalar systems, taking advantage of the highly tuned linear algebra components in the SBBLAS library. To apply a scalar local multigrid solver, the set of unknowns corresponding to a global scalar equation is restricted to the subset of unknowns that correspond to the specific subdomain.

To illustrate the approach, consider a matrix-vector multiplication $\mathbf{y} = \mathbf{A}\mathbf{x}$ with the exemplary block structure:

$$\begin{pmatrix}\mathbf{y}_1\\ \mathbf{y}_2\end{pmatrix} = \begin{pmatrix}\mathbf{A}_{11} & \mathbf{A}_{12}\\ \mathbf{A}_{21} & \mathbf{A}_{22}\end{pmatrix}\begin{pmatrix}\mathbf{x}_1\\ \mathbf{x}_2\end{pmatrix}$$

As explained above, the multiplication is performed as a series of operations on the local FE matrices per subdomain $\bar{\Omega}_i$, denoted by superscript $(\cdot)^{(i)}$. The global scalar operators, corresponding to the blocks in the matrix, are treated individually:

For $j = 1,2$, do

1. For all $\bar{\Omega}_i$, compute $\mathbf{y}_j^{(i)} = \mathbf{A}_{j1}^{(i)}\mathbf{x}_1^{(i)}$.
2. For all $\bar{\Omega}_i$, compute $\mathbf{y}_j^{(i)} = \mathbf{y}_j^{(i)} + \mathbf{A}_{j2}^{(i)}\mathbf{x}_2^{(i)}$.
3. Communicate entries in $\mathbf{y}_j$ corresponding to the boundaries of neighbouring subdomains.

4.3 Solving the Elasticity Problem

In order to solve vector-valued linearised elasticity problems with the application FEASTSOLID using the FEAST intrinsics outlined in the previous paragraphs, it is essential to order the resulting degrees of freedom corresponding to the spatial directions, a technique called separate displacement ordering [2]. In the 2D case where the unknowns $u = (u_1, u_2)^\mathsf{T}$ correspond to displacements in x and y-direction, rearranging the left hand side of equation (1a) yields:

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

We approximate the domain $\bar{\Omega}$ by a collection of several subdomains $\bar{\Omega}_i$, each of which is refined to a logical tensorproduct structure as described in Sect. 4.1. We consider the weak formulation of equation (3) and apply a Finite Element discretisation with conforming bilinear elements of the Q_1 space. The vectors and matrices resulting from the discretisation process are denoted with upright bold letters, such that the resulting linear equation system can be written as $\mathbf{Ku} = \mathbf{f}$. Corresponding to representation (3) of the continuous equation, the discrete system has 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 $\mathbf{f} = (\mathbf{f}_1, \mathbf{f}_2)^T$ is the vector of external loads and $\mathbf{u} = (\mathbf{u}_1, \mathbf{u}_2)^T$ the (unknown) coefficient vector of the FE solution. The matrices $\mathbf{K}_{11}$ and $\mathbf{K}_{22}$ of this block-structured system correspond to scalar elliptic operators (cf. Equation (3)), i. e. FEAST's tuned solvers can be applied to the corresponding subsystems.

We illustrate the details of the solution process with a basic iteration scheme, a preconditioned defect correction method:

$$\mathbf{u}^{k+1} = \mathbf{u}^k + \omega \tilde{\mathbf{K}}_B^{-1}(\mathbf{f} - \mathbf{K}\mathbf{u}^k), \qquad k = 1, \ldots \tag{5}$$

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

1. Compute the global defect (cf. Sect. 4.2):

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

2. Apply the block-preconditioner

$$\tilde{\mathbf{K}}_{BGS} := \begin{pmatrix} \mathbf{K}_{11} & \mathbf{0} \\ \mathbf{K}_{21} & \mathbf{K}_{22} \end{pmatrix} \tag{6}$$

   by approximately solving the system $\tilde{\mathbf{K}}_{BGS}\mathbf{c} = \mathbf{d}$. This is performed by two global scalar solves and one global (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: $\mathbf{u}^{k+1} = \mathbf{u}^k + \omega\mathbf{c}$

Instead of the illustrative defect correction scheme outlined above, our full solver is a preconditioned BiCGStab solver. Figure 3 summarises the entire scheme. Note the similarity to the general template solver in Fig. 2, and that this specialised solution scheme is entirely constructed from FEAST intrinsics.

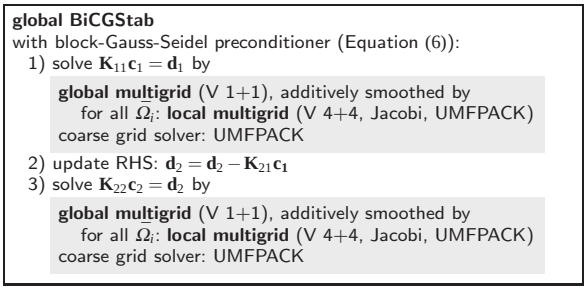

Fig. 3. Our solution scheme for the elasticity equations, scalar solvers are highlighted

4.4 Solving the Navier-Stokes Equation

The application FEASTFLOW solves the Navier-Stokes equations, a nonlinear and vector-valued problem, in a similar way as FEASTSOLID solves elasticity problems. Again, we approximate the domain Ω by a collection of quadrilateral subdomains Ω_i, each of which is refined to a logical tensorproduct structure. We consider the weak formulation of equation system (2) and apply a Finite Element discretisation with conforming bilinear elements of the Q_1 space. It is well-known that this pure Galerkin approach exhibits numeric instabilities which stem from dominating convection and from the violation of the discrete inf-sup or LBB-condition. For stability on arbitrary meshes, pressure-stabilisation (PSPG) and streamline-upwind stabilisation (SUPG) is applied, choosing the mesh-dependent parameters in accordance with Apel et al. [1].

The resulting discrete nonlinear equation system reads

$$\begin{pmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} & \mathbf{B}_1 \\ \mathbf{A}_{21} & \mathbf{A}_{22} & \mathbf{B}_2 \\ \mathbf{B}_1^T & \mathbf{B}_2^T & \mathbf{C} \end{pmatrix} \begin{pmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{p} \end{pmatrix} = \begin{pmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{g} \end{pmatrix}, \tag{7}$$

with

$$\mathbf{A}_{11} := \nu\mathbf{L}_{11} + \mathbf{N}_{11}(\mathbf{u}) + \tilde{\mathbf{C}}_{11} \qquad \mathbf{A}_{12} := \tilde{\mathbf{C}}_{12}$$
$$\mathbf{A}_{12} := \tilde{\mathbf{C}}_{21} \qquad \mathbf{A}_{22} := \nu\mathbf{L}_{22} + \mathbf{N}_{22}(\mathbf{u}) + \tilde{\mathbf{C}}_{22}$$

where the matrix $\mathbf{L}_{ii}$ corresponds to the Laplacian operator and $\mathbf{N}_{ii}(\mathbf{u})$ to the convection operator. $\mathbf{B}$ and $\mathbf{B}^T$ are discrete analogues of the gradient and divergence operator while $\mathbf{C}$ and $\tilde{\mathbf{C}}_{ij}$ stem from the discretisation of the PSPG and SUPG stabilisation terms, respectively. For the case of an isotropic mesh, we notice the following: $\mathbf{C}$ is identical to a discretisation of the pressure Poisson operator, scaled with the mesh size h^2.

The nonlinear problem is reduced to a sequence of linear problems by applying a fixed point defect correction method, which can be written in a manner similar to equation (5), where $\tilde{\mathbf{K}}_B$ can be identified with the solution of linearised subproblems.

The linearised, but still vector-valued subproblem is subsequently tackled by a pressure Schur complement approach: We illustrate it with the following basic iteration, but – as in the elasticity case – prefer a Krylov subspace solver such as BiCGStab for increased numerical efficiency in the tests in Sect. 5.2:

$$\begin{pmatrix}\mathbf{u}_{n+1}\\ \mathbf{p}_{n+1}\end{pmatrix} = \begin{pmatrix}\mathbf{u}_n\\ \mathbf{p}_n\end{pmatrix} + \mathbf{K}_\mathrm{s}^{-1}\left[\begin{pmatrix}\mathbf{f}\\ \mathbf{g}\end{pmatrix} - \begin{pmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{B}^\mathsf{T} & \mathbf{C}\end{pmatrix}\begin{pmatrix}\mathbf{u}_n\\ \mathbf{p}_n\end{pmatrix}\right] \tag{9}$$

Here, $\mathbf{A}$ is a block-structured matrix consisting of the linearised matrices $\mathbf{A}_{ij}$, $\mathbf{B}^\mathsf{T}$ is defined as $(\mathbf{B}_1^\mathsf{T}, \mathbf{B}_2^\mathsf{T})$ and the vectors $\mathbf{u}_n$ and $\mathbf{f}$ as the iterates of the solution $(\mathbf{u}_1, \mathbf{u}_2)^\mathsf{T}$ and right hand side $(\mathbf{f}_1, \mathbf{f}_2)^\mathsf{T}$, respectively. The preconditioner $\mathbf{K}_\mathrm{s}$ is defined as the block-structured matrix:

$$\mathbf{K}_\mathrm{s} := \begin{pmatrix}\tilde{\mathbf{A}} & 0\\ \mathbf{B}^\mathsf{T} & -\tilde{\mathbf{S}}\end{pmatrix} \tag{10}$$

where $\tilde{\mathbf{A}}$ denotes a preconditioner for the matrix $\mathbf{A}$ and $\tilde{\mathbf{S}}$ a preconditioner for the pressure Schur complement matrix

$$\mathbf{S} := \mathbf{B}^\mathsf{T}\mathbf{A}^{-1}\mathbf{B} - \mathbf{C}.$$

We note that solving equation (9) requires the solution of linear systems with the matrix $\mathbf{A}$ as well as some matrix-vector multiplications.

fixed point iteration
solving linearised subproblems with
global BiCGStab (reduce initial residual by 1 digit)
preconditioned by block preconditioner (Equation (10))
1) realise preconditioner $\tilde{\mathbf{A}}$ as **global BiCGstab** (1 digit)
with block-Gauss-Seidel preconditioner (Equation (6)):
a) solve $\mathbf{A}_{11}\mathbf{c}_1 = \mathbf{d}_1$ by
global multigrid (V 1+1), additively smoothed by
for all Ω_i: **local multigrid** (V 4+4, Jacobi, UMFPACK)
coarse grid solver: UMFPACK
b) update RHS: $\mathbf{d}_2 = \mathbf{d}_2 - \mathbf{A}_{21}\mathbf{c}_1$
c) solve $\mathbf{A}_{22}\mathbf{c}_2 = \mathbf{d}_2$ by
global multigrid (V 1+1), additively smoothed by
for all Ω_i: **local multigrid** (V 4+4, Jacobi, UMFPACK)
coarse grid solver: UMFPACK
2) realise preconditioner $\tilde{\mathbf{S}}$ as
$\mathbf{c}_3 = \mathbf{M}_p^{-1}(\mathbf{d}_3 + \mathbf{B}_1^\mathsf{T}\mathbf{c}_1 + \mathbf{B}_2^\mathsf{T}\mathbf{c}_2)$

Fig. 4. Excerpt of our solution scheme for the Navier–Stokes equations, scalar solvers are highlighted

Murphy et al. [8] have pointed out that the square of the iteration matrix of the preconditioned system

$$\mathbf{K} := \mathbf{I} - \mathbf{K}_\mathrm{s}^{-1}\begin{pmatrix}\mathbf{A} & \mathbf{B}\\ \mathbf{B}^\mathsf{T} & C\end{pmatrix} \tag{11}$$

vanishes. The associated Krylov space, $\mathrm{span}\{\mathbf{r}, \mathbf{K}\mathbf{r}, \mathbf{K}^2\mathbf{r}, \mathbf{K}^3\mathbf{r}, \ldots\}$, has hence dimension 2 which implies that – with exact numerics – any Krylov subspace iterative

method terminates in at most two iterations with the solution to the linear system arising in system (9) if the preconditioner $\mathbf{K}_s$ is used. A few iterations with a "good" approximation of $\mathbf{K}_s$ therefore suffice to solve (9).

The off-diagonal parts of $\mathbf{A}$ stem from the SUPG stabilisation terms only, they are of order $O(h^2)$ (and vanish for isotropic grids in the Stokes case). Taking this into account, the approximation of the upper left block of $\mathbf{K}_s$ is straightforward: The diagonal block matrices $\mathbf{A}_{11}$ and $\mathbf{A}_{22}$ correspond to scalar elliptic operators, FEAST's tuned solvers can be applied as in the elasticity case. Turek has shown that – neglecting the convective terms – the lumped pressure mass matrix $\mathbf{M}_p$ is a good preconditioner for the diffusive part of the Schur complement matrix $\mathbf{S}$ [10], i. e. $\tilde{\mathbf{S}} = \nu\mathbf{M}_p$. Solving in this case is reduced to scaling a right hand side with a diagonal matrix. Figure 4 summarises the solver scheme we use throughout this report.

5 Experimental Results

The questions we address in this report are: To what extend do FEASTSOLID and FEASTFLOW benefit from the highly tuned scalar solvers from the FEAST library? What MFLOP/s rates do we achieve on NEC SX-8? How well do these FE applications scale? The following paragraphs are dedicated to these questions.

5.1 Scalar Performance of FEAST

Table 1 lists our results of solving a Poisson problem with FEAST using 2–16 CPUs[1] on the `BLOCK` configuration grid (Fig. 5 (a)), applying Dirichlet conditions on the whole boundary, though. These results are a significant improvement compared to those we presented in last year's report. Every CPU is assigned four subdomains which are subsequently hierarchically refined. We employ the same scalar solver as used for solving scalar sub-problems in the CSM and CFD test cases later on, i. e. the global multigrid (used as a preconditioner to a global BiCGStab iteration) performs one pre- and postsmoothing step in a V cycle, and the local multigrids use a V cycle with four smoothing steps. The goal in all tests is to reduce the initial residuals by six orders of magnitude.

For FE codes, the available system bandwidth is the dominant factor for performance. On the NEC SX-8, each CPU can access main memory at 64 GB/s, while the IXS crossbar switch provides 16 GB/s per node (each node has 8 CPUs). The results obtained on 2 and 4 CPUs – on a single node – thus illustrate a monotonic increase in MFLOP/s rates for increasing level of refinement. For smaller problem sizes, the performance is inhibited by the sequential parts of the code, while for larger problem sizes the performance is mainly determined by the fully vectorised, throughput-oriented operations, and the tuned matrix-vector multiplication pays off. This is further underlined by the fact that with increasing level of refinement (four times the amount of unknowns), the time per iteration ($T_{\text{solve}}/\text{iter}$) increases by less

[1] There is always one master process which we do not list explicitly.

than a factor of four. We reach a stable performance of more than 5 GFLOP/s per CPU, which is roughly 30% of the peak performance. Detailed analysis reveals that the single-node performance is close to the peak memory bandwidth. The results obtained on 8 and 16 CPUs include communication via the interconnect, and thus exhibit a significant decrease in performance.

# CPU	level	DOF	T_{solve}/iter (s)	MFLOP/s	MFLOP/s/CPU
2	7	131,841	0.27	1,111	556
	8	525,825	0.40	2,806	1,403
	9	2,100,225	0.80	5,930	2,965
	10	8,394,753	1.94	9,098	4,549
	11	33,566,721	5.89	10,959	5,480
4	7	263,425	0.27	2,116	529
	8	1,051,137	0.43	5,336	1,334
	9	4,199,425	0.74	11,049	2,762
	10	16,787,457	2.08	17,091	4,273
	11	67,129,345	6.36	20,269	5,067
8	7	525,825	0.30	3,685	461
	8	2,100,225	0.50	9,059	1,132
	9	8,394,753	1.00	18,390	2,299
	10	33,566,721	2.57	27,645	3,456
	11	134,242,305	8.07	31,959	3,995
16	7	1,051,137	0.33	6,989	437
	8	4,199,425	0.53	17,701	1,106
	9	16,787,457	1.07	34,408	2,151
	10	67,129,345	3.03	46,888	2,931
	11	268,476,417	8.96	57,600	3,600

Table 1. Efficiency tests. Solving a scalar Poisson problem on NEC SX-8

5.2 Performance of FEASTSOLID and FEASTFLOW

With FEASTSOLID, we evaluate four configurations that are prototypical for practical applications. Figure 5 shows the coarse grids, the prescribed boundary conditions and the partitioning for the parallel execution of each configuration. The `BLOCK` configuration (Fig. 5 (a)) is a standard test case in CSM, in which a block of material is vertically compressed by a surface load. The `PIPE` configuration (Fig. 5 (b)) represents a circular cross-section of a pipe clamped in a bench vise. It is realised by loading two opposite parts of the outer boundary by surface forces. With the `CRACK` configuration (Fig. 5 (c)) we simulate an industrial test environment for assessing material properties. A workpiece with a slit is torn apart by a device attached to the two holes. In this configuration the deformation is induced by prescribed horizontal displacements at the inner boundary of the holes, while the holes are fixed in the vertical direction. For the latter two configurations we exploit symmetries and consider only sections of the real geometries. Finally, the `STEELFRAME` configuration (Fig. 5 (d)) models a section of a steel frame, which is fixed at both ends and asymmetrically loaded from above.

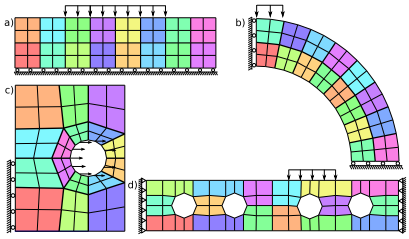

Fig. 5. Coarse grids, boundary conditions and static partition into subdomains for the configurations (a) `BLOCK`, (b) `PIPE`, (c) `CRACK` and (d) `STEELFRAME`

In all the CSM tests we configure the solver scheme (cf. Fig. 3) to reduce the initial residuals by 6 digits, the global multigrid performs one pre- and postsmoothing step in a V cycle, and the inner multigrid uses a V cycle with four smoothing steps.

For CFD, we use a standard benchmark case: a lid driven cavity at Reynolds number $Re = 100$. The solver scheme (cf. Fig. 4) is configured to reduce the residuals for velocity and pressure to below 10^{-8}.

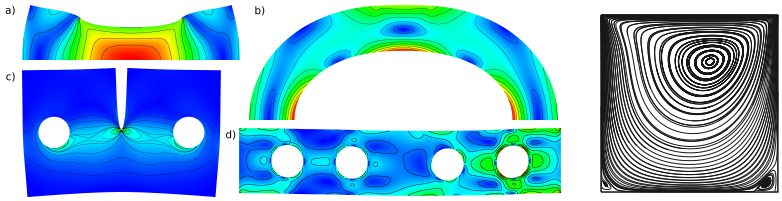

Fig. 6. Computed displacements and von Mises stresses for the CSM configurations, velocity streamline plot for Driven Cavity

Figure 6 shows the computed deformations of the four CSM geometries and the von Mises stresses, which are an important measure for predicting material failure in an object under load, as well as a streamline plot of the driven cavity velocity solution.

Absolute Performance of NEC SX-8 and a Commodity Based Cluster

To compare performance across different architectures and to be able to assess the performance of our code on NEC SX-8, we execute the elasticity solver (cf. Fig. 3) on 16 nodes of a commodity based Opteron cluster (LiDO, Dortmund) and the NEC SX-8. The (slightly outdated) cluster consists of two Opteron DP 250 CPUs with 8 GB DDR-400 memory, and is fully connected via Infiniband. For these tests, we use the four prototypical test cases illustrated in Fig. 5, and refine each subdomain 10 times for a problem size of 134,258,690 degrees of freedom.

Table 2 contains our timing measurements. We first note that the obtained results are consistent for all four configurations. The longer computation times of the PIPE

Configuration	level	NEC SX-8				LiDO			
		T_{solve} (s)	T_{scalar} (s)	MFLOP/s	MFLOP/s/CPU	T_{solve} (s)	T_{scalar} (s)	MFLOP/s	MFLOP/s/CPU
BLOCK	7	6.4	6.3	3,533	221	2.6	2.4	8,709	544
	8	9.5	9.3	9,434	590	10.9	10.1	8,156	509
	9	14.4	14.0	22,008	1,375	39.3	36.3	8,062	503
	10	31.3	29.9	40,256	2,516	157.2	145.3	8,025	502
CRACK	7	6.7	6.6	3,532	221	2.8	2.6	8,582	536
	8	9.2	9.1	10,122	633	10.8	10.0	8,634	540
	9	15.9	15.5	23,250	1,453	46.5	43.5	7,970	498
	10	33.5	32.1	39,693	2,481	166.7	155.5	7,966	498
PIPE	7	11.7	11.5	3,570	223	4.5	4.1	9,281	580
	8	16.2	15.9	10,155	635	18.6	17.2	8,827	552
	9	25.2	24.4	24,454	1,528	77.1	71.7	7,989	499
	10	66.4	63.4	41,498	2,594	345.7	321.7	7,969	498
STEEL-FRAME	7	13.8	13.6	3,544	221	5.5	5.0	8,914	557
	8	20.3	20.0	10,442	653	23.7	21.9	8,958	560
	9	35.7	34.6	24,647	1,540	103.6	96.4	8,487	530
	10	92.1	92.1	41,287	2,580	470.2	438.5	8,090	506

Table 2. Performance results for four prototypical test cases, computed with FEASTSOLID on 16+1 CPUs on LiDO and NEC SX-8

and the STEELFRAME configuration result from the fact that they exhibit a relatively long and thin geometry, while only a relatively small portion of the boundary is fixed (cf. Axelsson [2]). As expected from the experiments with the scalar Poisson problem (cf. Table 1), the Opteron cluster outperforms the NEC SX-8 on small levels of refinement. For larger problem sizes, the NEC SX-8 executes FEASTSOLID roughly 5 times faster than the Opteron cluster. The comparison between the total solving time T_{solve} and the accumulated times of the scalar solves T_{scalar} verifies that our approach of reducing vector-valued problems to sequences of scalar solves works as expected. Independent of the level of refinement, more than 95% of the total time to solution is spent inside scalar solvers. Consequently, the MFLOP/s rates for the solver scheme (cf. Fig. 3) are in the same range as in the scalar case depicted in Table 1. We expect the gap between the Opteron cluster and NEC SX-8 to widen further when refining the coarse grids more than 10 times (cf. Sect. 5.1), which we could not do on LiDO due to lack of local memory.

Weak Scalability of FEASTSOLID

We evaluate weak scalability on NEC SX-8 and LiDO with FEASTSOLID. For these tests, we employ modifications of the `BLOCK` configuration (Fig. 5 (a)) such that each CPU is assigned four subdomains. The number of CPUs (and hence, DOF) is increased from 2 to 64 (16 Mi to 537 Mi DOF, refinement level $L = 10$). Due to the different geometries, the number of solver iterations until convergence varies, so we normalise the timings with the iteration numbers to emphasise the scalability of our approach rather than presenting obscured results due to the elasticity solver's dependence on the geometry (see previous paragraph).

Figure 7 illustrates good scalability of FEASTSOLID on both architectures. On the NEC SX-8, the bump in performance as soon as more than a single node is involved is clearly visible (cf. Sect. 5.1), and for larger CPU numbers, performance

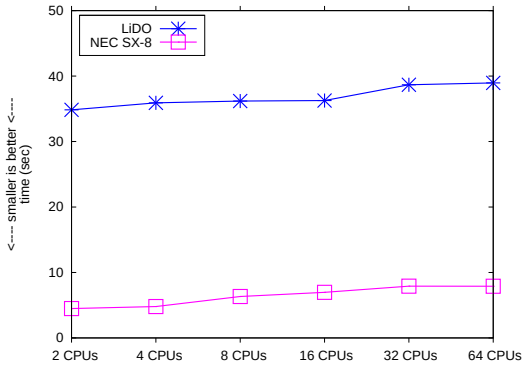

Fig. 7. Weak scalability tests for FEASTSOLID on LiDO and NEC SX-8

reaches a stable 7.9 seconds per iteration. We thus attribute the linear increase in runtime from 8 to 32 CPUs to granularity effects of the IXS crossbar switch.

When comparing performance between the two architectures, we see that the speed-up achieved by the NEC SX-8 over the Opteron cluster LiDO slowly decreases from a factor of 7.7 to a stable factor of 5 when increasing the number of CPUs, consistent with the results in the previous paragraph. We also attribute this to the interconnects.

Performance Results with FEASTFLOW

Table 3 presents results obtained by FEAST's CFD application FEASTFLOW, using the solver shown in Fig. 4 applied to the lid driven cavity benchmark problem. We only list the time spent in solving the linearised problems, as the assembly process has not been fully tuned yet and obfuscates the solver timings we are interested in. The values labeled "T_{scalar} %" demonstrate that, similar to the elasticity solver above, more than 90% of the total time is spent inside FEAST's optimised scalar solvers. This confirms the feasibility of our solution approach; and accordingly, the MFLOP/s rates obtained by the simple Poisson solver are preserved (see last column of Table 3, copied from Table 1).

# CPU	level	DOF	T_{solve} (s)	T_{scalar} (s)	T_{scalar} %	MFLOP/s	MFLOP/s/CPU	MFLOP/s/CPU poisson
2	7	395,523	35.1	33.4	95	1,155	583	556
	8	1,577,475	59.0	55.5	94	2,894	1,456	1,403
	9	6,300,675	117.7	108.2	92	5,931	2,977	2,965
	10	25,184,259	345.2	309.1	90	9,137	4,576	4,549
4	7	789,507	40.7	38.4	94	2,145	541	529
	8	3,151,875	66.9	62.8	94	5,359	1,348	1,334
	9	12,595,203	128.3	118.2	92	11,078	2,779	2,762
	10	50,356,227	404.5	363.9	90	17,197	4,306	4,273

Table 3. Performance results of FEASTFLOW

We did not perform tests on more CPUs and higher problem sizes on the NEC SX-8 yet. However, these preliminary results convince us that FEASTFLOW will scale just as well as FEASTSOLID or the simple Poisson solver.

6 Conclusions and Future Work

We have demonstrated the feasibility of our approach to reduce the solution of vector-valued problems from CSM and CFD to sequences of scalar problems to be treated with optimised and architecture-aware scalar multigrid solvers. For several prototypical applications from linearised elasticity and fluid dynamics, the approach maintains weak scalability and node performance of the prototypical Poisson problem. In particular, FEAST applications on NEC SX-8 execute significantly faster than on commodity based clusters. In future work, we will focus not only on tuning solvers, but also on the assembly process which turned out to be a bottleneck during our experiments.

Acknowledgements

We would like to thank Christian Becker for initial work with FEAST on the NEC SX-8, and his help and support. This work has been supported by DFG, under grants TU 102/22-1, TU 102/27-1, TU 102/11-3.

References

[1] Th. Apel, T. Knopp, and G. Lube. Stabilized finite element methods with anisotropic mesh refinement for the Oseen problem. In G. Lube and G. Rapin, editors, *Proceedings of the International Conference on Boundary and Interior Layers (BAIL 2006), Göttingen*, pages 1–8, 2006.

[2] Owe Axelsson. On iterative solvers in structural mechanics; separate displacement orderings and mixed variable methods. *Mathematics and Computers in Simulations*, 50:11–30, 1999. doi: 10.1016/S0378-4754(99)00058-0.

[3] 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, Fachbereich Mathematik, May 2007. `http://www.logos-verlag.de/cgi-bin/buch?isbn=1637`.

[4] Christian Becker, Sven H.M. Buijssen, and Stefan Turek. FEAST: Development of HPC technologies for FEM applications. In *High Performance Computing in Science and Engineering '07, Transactions of the High Performance Computing Center, Stuttgart (HLRS) 2007*, pages 503–516. Springer, Berlin, 2007. doi: 10.1007/978-3-540-74739-0_34.

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

[6] Timothy A. Davis. A column pre-ordering strategy for the unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):165–195, 2004. doi: 10.1145/992200.992205.

[7] Joel H. Ferziger and Milovan Perić. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 1996.

[8] Malcolm F. Murphy, Gene H. Golub, and Andrew J. Wathen. A note on preconditioning for indefinite linear systems. *SIAM J. Sci. Comput.*, 21(6):1969–1972, 1999. doi: 10.1137/S1064827599355153.

[9] Barry F. Smith, Petter E. Bjørstad, and William D. Gropp. *Domain Decomposition: Parallel Multilevel Methods for Elliptic Partial Differential Equations*. Cambridge University Press, 1996.

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

[11] Stefan Turek, Christian Becker, and Susanne Kilian. Hardware–oriented numerics and concepts for PDE software. *Future Generation Computer Systems*, 22(1):217–238, 2003. doi: 10.1016/j.future.2003.09.007.