How to gain speedups of 1000 on single processors with fast FEM solvers Benchmarking numerical and computational efficiency

Michael Köster, Dominik Göddeke, Hilmar Wobker, Stefan Turek and the FEAST group
Applied Mathematics, Technische Universität Dortmund
feast@math.tu-dortmund.de

Abstract

In Computational Science and in particular in the numerical simulation of PDE problems, optimal serial performance is essential for a successful scale-out to the tera- and petascale dimensions. In this paper, we propose a simple yet fundamental benchmark setting for a PDE problem that we believe any reasonably flexible Finite Element based software should be able to handle effortlessly. The Poisson problem used in these tests allows reliable performance estimates for more challenging simulations.

Our performance evaluation focuses on numerical methodology and data layouts rather than implementational fine-tuning. To enable a fair and realistic comparison independent of the underlying numerical methodology, we define the metric *total efficiency*. Results are presented for two different solver classes, multigrid and Krylov-subspace methods, obtained in single-core computations with our solver packages FEAT2 and FEAST. We quantitatively emphasise the effect of different storage techniques and numbering (reordering) schemes, which constitute the crucial factor in view of the *memory wall problem* that ultimately determines performance of all Finite Element codes. We demonstrate a speed-up of more than a factor 1000 by migrating from a naive implementation of a standard Krylov solver to a sophisticated implementation of an advanced multigrid solver, without applying any adaptivity.

1 Motivation

The simulation of real-world phenomena is a challenging task, and of utmost importance in Computational Science, e. g. engineering, logistics, and natural and life sciences. Typical problems are highly dynamic in both space and time, and are often modeled by systems of Partial Differential Equations (PDEs). Their accurate discretisation in space with Finite Elements leads to huge sparse linear systems of equations, which need to be solved many times when an implicit discretisation in time is applied, or when the underlying problem is of nonlinear nature. Typical numbers are in the order of 10^9 unknowns and 10^5 timesteps.

Obviously, such systems can not be solved with just one computer, and large tera- and petascale installations with tens of thousands of CPUs need to be employed. In the established TOP500 lists of supercomputers, 'petascale' is defined by the runtime of a Linpack benchmark, the computation of the LU decomposition of a large dense linear system. Despite the undoubted merits of this list, this metric is essentially meaningless for the PDE problems and application domains we are interested in. Therefore, we follow a rather 'hands-on' approach assuming ideal weak scalability: If a problem with 10^6 unknowns can be solved in less than one second on a PC, then a problem with 10^9 unknowns should be solved in the same time on a terascale system, and a problem with 10^{12} unknowns should still require less than one second on a petascale machine.

In practice, this is tremendously hard to achieve because due to Amdahl's Law, hardware limitations and many other aspects, codes do not scale perfectly in parallel. However, and this is the main motivation of our paper, excellent performance of a given code on a single CPU is

the basis of parallel High Performance Computing. As we demonstrate in this paper, excellent performance is achieved only by the combination of numerical methodology (in particular fast solvers with linear run-time in the number of unknowns, independent of the refinement level of the underlying discretisation) and implementational aspects (in particular optimal data structures, spatial and temporal blocking). We define a simple yet fundamental benchmark problem and assess these aspects quantitatively. In our experience, commercial codes are unable to achieve the performance level we expect. We cordially invite other researchers to apply their solvers to the benchmark problem we propose in this paper, and to share results.

2 Test Problem

The computational domain Ω for our benchmark tests is defined as a rectangle enclosing an inner circle (cf. Figure 2.1 (left)):

$$\Omega := [0,1] \times [0,1.25] \ \setminus \ B_r(0.5,0.5), \qquad r = \sqrt{0.02}$$

Let Γ_1 define the outer boundary component and Γ_2 define the inner boundary component (i. e. the circle). On this domain, we solve the Poisson problem with Dirichlet boundary conditions:

$$\begin{aligned} -\Delta u &= 0 \qquad \text{in } \Omega, \\ u &= 0 \qquad \text{on } \Gamma_1 \\ u &= 1 \qquad \text{on } \Gamma_2 \end{aligned}$$

Figure 2.1 (right) depicts the computed solution.

We define the following geometric points on the circle: $x_1 := (0.4, 0.4)$, $x_2 := (0.4, 0.6)$, $x_3 := (0.6, 0.6)$, $x_4 := (0.4, 0.6)$. Connecting these points with horizontal and vertical lines to the boundary defines a mesh $\mathcal{T}_1$ consisting of 8 quadrilateral cells. This mesh is used as the coarse mesh for the calculation, finer meshes $\mathcal{T}_2, \ldots, \mathcal{T}_{11}$ are created by regular refinement of each cell into four new cells via edge bisection. To prevent mesh tangling during refinement of the cells adjacent to the circle, new boundary points are projected onto the circle and element midpoints on the coarse grid forming vertices on the fine grid are corrected to the geometric mean of the surrounding vertices (stemming from the edge midpoints). An alternative coarse mesh may be created by subdividing each quadrilateral into two triangles; the triangles are again refined regularly. Table 2.1 lists the number of vertices and elements for the levels of refinement under consideration in this benchmark proposal.

The computational grids remain static during the entire computation; in this first benchmark proposal, advanced techniques such as adaptive mesh refinement or p-adaptivity are not considered.

On the resulting mesh, the Poisson equation is discretised using (bi-)linear Finite Elements. In case of the FEAT2 and FEAST results presented below, the bilinear conforming parametric quadrilateral Q_1 element is chosen. The discretisation leads to a linear system $A\mathbf{x} = \mathbf{b}$, where the unknowns coincide with the vertices in the mesh.

The test problem is chosen so that it can be computed on a typical workstation, and should fit well into 4–8 GByte of RAM. As outlined in the previous Section, we aim at solving the largest problem (refinement level 10) in less than eight seconds, although currently, we

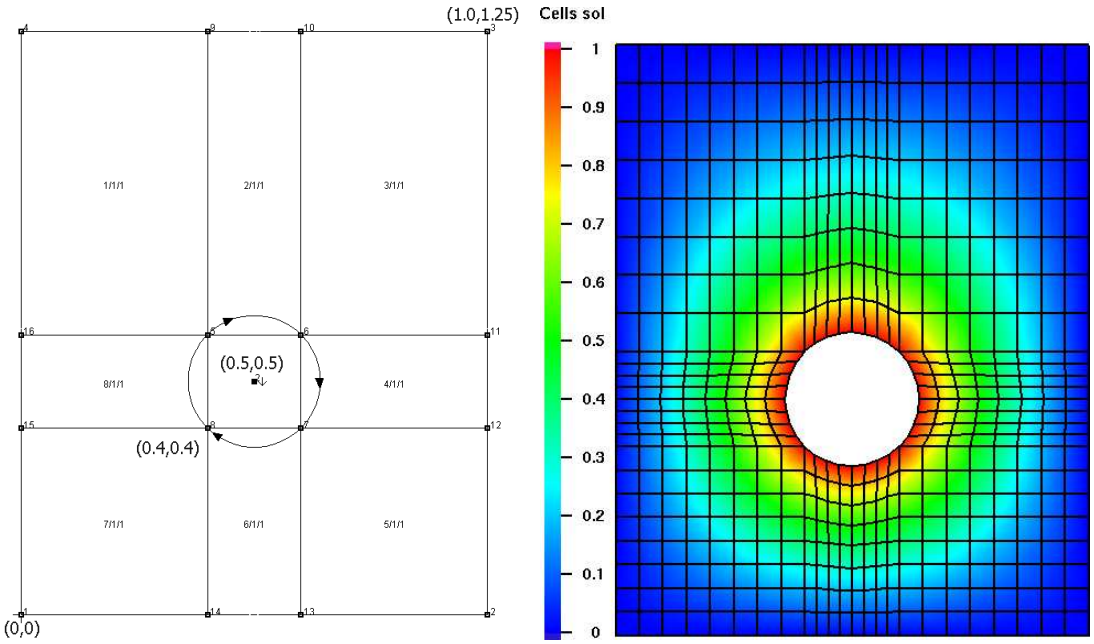

Figure 2.1: Left: Domain and coarse grid $\mathcal{T}_1$. Right: Solution on $\mathcal{T}_4$.

only achieve roughly 25% of this performance. In this first stage of the benchmark proposal, we concentrate on the time to solution and prescribe a fixed reduction in the norms of the residuals. In the long term, we will enhance this criterion by goal-oriented error estimators, e. g. to reduce the real errors to a prescribed accuracy.

Lv.	NVT	NEL
7	33 280	32 768
8	132 096	131 072
9	526 336	524 288
10	2 101 248	2 097 152
11	8 396 800	8 388 608

Table 2.1: Number of vertices (NVT) and number of elements (NEL) for different refinement levels.

3 Target Hardware

The goal of the proposed benchmarks is to evaluate the performance of different numerical techniques and data layouts, rather than implementational fine-tuning. Consequently, the benchmark proposal does not fully prescribe the compiler and the machine to use. Fine-grained performance differences in the order of few seconds are not relevant. Nonetheless, the benchmark hardware should be a commodity based compute server or workstation, and only one core is used in case of multicore CPUs. A performance comparison of parallel computations (either distributed or shared memory) will be addressed in a follow-up proposal.

4 Benchmark Scenarios & Solver Configurations

The tests in this benchmark proposal are divided in three categories:

Matrix-Vector To isolate the effects of data layouts and numbering schemes for the unknowns, performance of a defect calculation of the form $\mathbf{d} = \mathbf{b} - A\mathbf{x}$ for arbibrary vectors $\mathbf{x}$ and $\mathbf{b}$ is measured separately:

The matrix is assembled only for the restriction of the computational domain to the coarse grid cell adjacent to the point $(0, 0)$, and only the triangulations $\mathcal{T}_7, \ldots, \mathcal{T}_{11}$ are considered. Boundary conditions are completely ignored. Performance is measured in MFLOP/s: For the Q_1 Finite Element, nine entries per matrix row are nonzero except for boundary entries, hence, performance of the matrix-vector multiplication is calculated as $P = (18 \cdot N)/(T \cdot 10^6)$ MFLOP/s for $N = (2^k + 1)^2$ unknowns on $\mathcal{T}_k$ and a measured time T. The test is executed for the same matrix ordering schemes (c. f. Section 5.1) as used for the CG and MG benchmarks.

Conjugate Gradients As a representative of 'simple' Krylov subspace solvers, the Conjugate Gradient algorithm is employed. Only solving times are relevant, matrix assembly time and any pre- and postprocessing is not considered. The stopping criterion is set to reduce the initial residual by eight digits:

$$\frac{||\mathbf{b} - A\mathbf{x}_n||_2}{||\mathbf{b} - A\mathbf{x}_0||_2} \leq \varepsilon := 10^{-8}.$$

For different numbering schemes, two Conjugate Gradient variants are considered. The first one, labeled `CG-simple`, uses a Jacobi preconditioner (a simple scaling by the inverse of matrix diagonal, not damped). The second one, labeled `CG-advanced`, uses the preconditioner considered 'optimal' in the Finite Element toolkit under evaluation, balancing execution time with numerical robustness. For all tests, the number of iterations until convergence, the convergence rate and the time to solution in seconds are recorded.

Multigrid Analogously, `MG-simple` uses a multigrid solver with $4 + 4$ Jacobi pre- and postsmoothing steps (damped by $\omega = 0.7$) in an F-cycle. The multigrid scheme may use a direct solver (e. g. UMFPACK) to solve the coarse grid problems. The test `MG-advanced` uses an advanced smoother, again assumed to be 'optimal' in the Finite Element toolkit under evaluation. The stopping criterion and timing goals are the same as for the Conjugate Gradient tests. $\mathcal{T}_1$ is always the minimum level used in the grid hierarchy.

In all configurations, the initial guess $\mathbf{x}_0$ is the zero vector, with boundary conditions already applied. By running both solvers in two configurations, we try to separate implementational from numerical aspects as much as possible, as Jacobi preconditioning and smoothing are trivial and should be supported by all solver packages.

5 Results

We present some initial results based on the FEM toolkits FEAT2 and FEAST, which are actively being developed in our group [2, 4, 5].

5.1 Data layout techniques

For all benchmark problems defined above, the following numbering schemes are employed [9, 11]:

a) `2-lv`: In the two-level numbering, the numbers of the degrees of freedom at the vertices on each finer mesh coincide with the numbers on the corresponding coarse mesh.

b) `CM`: standard Cuthill-McKee renumbering strategy.

c) `XYZ`: The vertices are numbered based on their geometric coordinates of the vertices, i. e. sorted first by the X- and then for the Y-coordinate.

d) `Stoch.`: fully random ordering. This is a seemingly artificial worst-case scenario to maximise cache miss rates, which is justified as an emulation of fully adaptive approaches.

e) `Hier.`: In the hierarchical renumbering, the degrees of freedom of fine grid cells are recursively collected and numbered according to the coarse grid cells.

f) `Banded (B)`: This strategy is basically the same as `XYZ` numbering, but for each refined coarse grid cell independently. The approach stems from domain decomposition, a 'virtual' global matrix is constructed from local matrices assembled on each refined coarse grid cell [2].

g) `Banded-Const (BC)`: The matrices corresponding to square coarse grid cells are not fully assembled; instead, matrix-vector multiplication is reduced to the application of simple 9-point stencils, at least in the interior of the refined cell.

The numbering schemes `2-lv`, `CM`, `XYZ`, `Stoch.` and `Hier.` are used in the FEAT2 FEM package with a CSR matrix format, while the FEAST software provides support for the `Banded` and `Banded-Const` numbering strategy, storing matrix bands as vectors and hence uses direct memory access. The `const` format is only applied to the coarse grid cells 1, 3, 5 and 7, due to the sheared elements in the other coarse grid cells, FEAST assembles the corresponding submatrices explicitly.

5.2 Solver configuration details

The basic solver configurations `CG-simple` and `MG-simple` can be used with all solvers for the renumbering strategies described above. The `xxx-advanced` configurations however depend on the underlying software package and renumbering strategy:

`CG-advanced` For the `2-lv`, `CM`, `XYZ`, `Stoch.` and `Hier.` renumbering strategies: CG solver, undamped ILU preconditioner. No test results are available at the moment for the `Banded` and `Banded-Const` renumbering strategies.

`MG-advanced` For the `2-lv`, `CM`, `XYZ`, `Stoch.` and `Hier.` renumbering strategies: Multigrid solver, ILU smoother [3, 7] damped with $\omega = 0.9$. V-cycle, 2 pre- and 2 postsmoothing steps. We use UMFPACK to solve the coarse grid problems exactly (and compute the factorisation in a preprocessing step, not included in the timings). For the `Banded` and `Banded-Const` renumbering strategies: Multigrid solver, linewise alternating directions Gauß-Seidel smoother (ADI-TRIGS, tuned to the underlying generalised tensorproduct structure of the refined coarse grid cells [2]). V-cycle, 2 pre- and 2 postsmoothing steps, UMFPACK as coarse grid solver.

All tests have been carried out on an AMD Opteron 2214 machine, 2.2 GHz, 1 MB L2-cache, SUSE Linux 10.2 64-bit operating system. Only one core has been used, the second core remained idle, and `numactl` has been employed to avoid known issues with the Linux scheduler in kernel version 2.6.16.21-0.25-smp. The codes were compiled with the Intel Fortran compiler V9.1.040 and tuned optimisation flags.

FEAT2 is a more mathematically oriented FEM toolkit, and despite being reasonably well tuned, leaves most optimisations to the compiler. In contrast, FEAST employs special spatial and temporal blocking techniques in the core components (matrix-vector multiplication and application of smoothers) to improve data locality.

5.3 Matrix-vector multiplication

Table 5.1 presents the results of the raw performance tests of the matrix-vector multiplication for the restriction of the computational domain to the refined bottom left coarse grid cell. Our first observation is surprisingly important: Even in 2008, with non-naive evolved implementations, sophisticated compilers and powerful hardware, the application of the 'wrong' renumbering strategy still leads to extremely poor performance. Only 50 MFLOP/s (of a theoretical peak performance of more than 6 GFLOP/s per core) are achieved in case of the (admittedly artificial, but instructive) stochastic numbering scheme which also serves to emulate fully adaptive discretisations. It is interesting to note that these 50 MFLOP/s have remained constant over the last 7 years [11], despite tremendous improvements of processor performance.

	Level				
Strategy	7	8	9	10	11
2-lv	257	161	144	133	127
CM	260	166	160	147	134
XYZ	263	163	167	162	161
Stoch.	259	145	116	62	50
Hier.	260	169	159	154	154
B	1445	718	627	615	550
BC	2709	2155	2091	1714	1597

Table 5.1: Matrix-vector-product test. MFLOP/s rate for different sorting strategies. Row 1-5: Sorting strategies in FEAT2. Row 6+7: Data layout strategies in FEAST.

The other sorting strategies perform as expected: The XYZ-sorting and the hierarchical sorting are fastest in FEAT2, and the MFLOP/s rate slowly decreases with increasing level, consistently for all numbering schemes. The benefits of the data locality optimisations and block memory accesses in the `Banded` and `Banded-Const` data layouts in FEAST are clearly vis-

ible, outperforming all other matrix renumbering strategies by a factor of $5-10$. In particular, the crossover point as soon as all data does not fit into cache anymore is obvious.

5.4 Solver timings

Table 5.2 summarises detailed timing measurements and the number of iterations needed until convergence for the different solver configurations.

	Level	7		8		9		10		11	
Solver	Sort.	Time	#It.	Time	#It.	Time	#It.	Time	#It.	Time	#It.
`CG-simple`	2-lv	2.8	502	24.9	962	202.4	1882	1595.6	3675	16808.2	7110
	CM	2.8	502	23.8	962	190.9	1882	1511.9	3675	15740.3	7110
	XYZ	2.7	501	22.6	962	175.8	1882	1366.7	3675	13202.0	7110
	Stoch.	3.2	502	38.0	962	396.5	1882	3356.0	3675	43863.2	7110
	Hier.	2.8	501	23.1	962	178.6	1882	1393.5	3675	13510.2	7110
	B	1.3	506	10.4	1002	82.0	1966	635.8	3815	5066.0	7364
	BC	1.2	506	9.5	1002	71.0	1966	563.7	3815	4374.8	7364
`CG-advanced`	2-lv	1.7	221	22.5	436	197.7	866	1569.8	1700	9667.0	3322
	CM	1.6	187	17.4	339	145.4	652	1178.8	1268	8994.8	2423
	XYZ	0.9	120	15.0	233	86.9	452	708.6	903	4868.9	1768
	Stoch.	2.1	206	37.8	404	418.0	752	3583.0	1465	34984.1	2821
	Hier.	1.5	183	16.9	341	134.9	659	835.6	1284	7234.9	2498
	B										
	BC										
`MG-simple`	2-lv	1.3	17	6.2	17	27.4	17	113.8	17	583.2	17
	CM	1.3	17	6.2	17	26.6	17	110.9	17	570.9	17
	XYZ	1.3	17	5.9	17	24.6	17	99.9	17	467.9	17
	Stoch.	1.5	17	9.3	17	52.0	17	246.1	17	1670.5	17
	Hier.	1.3	17	6.0	17	25.2	17	102.1	17	477.0	17
	B	0.5	18	2.0	18	7.6	18	29.6	18	125.9	18
	BC	0.5	18	1.7	18	6.2	18	24.1	18	99.7	18
`MG-advanced`	2-lv	0.5	9	2.1	9	9.3	9	38.0	9	249.2	10
	CM	0.4	6	1.7	7	7.3	7	30.9	7	199.8	8
	XYZ	0.3	6	1.3	6	5.4	6	21.9	6	114.3	6
	Stoch.	0.6	8	3.2	8	19.8	8	105.2	9	766.5	9
	Hier.	0.4	7	1.7	7	6.8	7	30.7	8	167.8	8
	B	0.2	7	0.6	7	2.2	7	8.9	7	33.5	6
	BC	0.2	7	0.5	7	1.9	7	7.8	7	29.2	6

Table 5.2: Time in seconds and number of iterations for different solver configurations and sorting strategies. Note that the `CG-advanced` configuration is currently not supported in FEAST.

As expected, multigrid outperforms all Conjugate Gradient solvers, especially on higher levels. The timings for the CG algorithm deteriorate by a factor of ≈ 8 per refinement level (four times more unknowns, doubled iterations). `MG-simple` exhibits a factor of ≈ 4, proportional to the number of unknowns, with a constant number of iterations independent of the level. XYZ and hierarchical sorting again perform best, stochastic sorting worst by factors. As implied by the raw matrix-vector multiplication test (Table 5.1), the cache misses implied by stochastic sorting lead to an increase in time higher than the expected factors. The differences in iteration numbers are attributed to the slightly different implementation of the solvers in FEAT2 and FEAST.

The beneficial effects of the data locality optimisations in FEAST's `Banded` and `Banded-Const` data layouts are carried over to the entire solver: In all configurations, these strategies allow a speedup of a factor up to $\approx 3 - -5$ in comparison to the non-banded data layouts.

The `XYZ` renumbering roughly performs $2 - -3\times$ faster than the stochastic renumbering, while the `Banded` data layout yields a factor of more than 10. Between the simple CG and

the advanced MG we observe (depending on the numbering and the refinement level) a factor of $30 - -200$, while the worst case best case comparison (stochastically renumbered CG and advanced multigrid with `Banded` data layout) yields even a factor of more than 1300. We explain the implications of these results in the next section.

Figure 5.1 summarises these results: We use the worst timing (`CG-simple` at level 11 with stochastic renumbering strategy) as a basis and normalise it to 1. The graph clearly illustrates the speedup factors when using advanced numerical methods and hardware-oriented implemententation techniques *at the same time*.

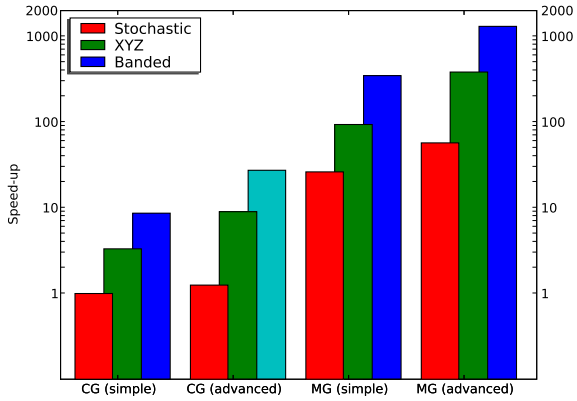

Figure 5.1: Speedup gained by data structure and algorithm. `CG-simple` is normalised to 1. Note that the `CG-advanced` configuration is currently not supported in FEAST, and the cyan plot shows expected performance only.

5.5 Solver Efficiency

The time to solution as listed above is already an appropriate measure for the (computational and numerical) efficiency of an algorithm. Unfortunately this measure is dependent on stopping criteria and the number of unknowns in a system. A more abstract measure alleviates these problems and allows a better comparison between different discretisations: Let T denote the time to solution in seconds, $N \in \mathbb{N}$ the number of unknowns, k the number of iterations until solution and $\rho = (\frac{||\mathbf{b}-A\mathbf{x}_k||}{||\mathbf{b}-A\mathbf{x}_0||})^{(1/k)}$ the convergence rate of the solver. We define the *total efficiency*, which measures the time in μs per unknown necessary to gain one digit in the residual, as follows:

$$E_{\text{total}} := -\frac{T \cdot 10^6}{\log_{10}(\rho) \cdot N \cdot k}$$

This metric reduces the efficiency of a given Finite Element solver to a single number. In terms of our motivation (c. f. Section 1), our goal is to achieve a performance of less than $0.125\,\mu s$ for our relative stopping criterion $\varepsilon = 10^{-8}$.

Corresponding to Table 5.2, Table 5.3 depicts the convergence rates and total efficiency measures for the different solver strategies. We first observe that in this metric, the worst case best case comparison (see Figure 5.1) yields even a factor of 1700, due to the reduced impact

of the relatively coarse grained convergence checks which are only performed once after each solver iteration.

For the CG algorithm, the table shows that the time per unknown doubles with every refinement level, as expected from the theory of the CG algorithm. For multigrid however, this measure stays constant for almost all configurations, which is also expected from theory, and using the best data layout allows to solve the system with less than 0.5 μs per unknown and digit. The only exception is the stochastic reordering strategy which shows a degeneration of a factor of approx. 1.5 per refinement level in case of the multigrid solver, and a significant drop in performance for refinement level 11 and the Krylov solver. As a consequence, the theoretical statement that multigrid solves a system with linear complexity in the number of unknowns is in practice not valid anymore if the ordering of the unknowns is not suitable.

	Level	7		8		9		10		11	
Solver	Sort.	ρ	E_{total}	ρ	E_{total}	ρ	E_{total}	ρ	E_{total}	ρ	E_{total}
`CG-simple`	2-lv	0.964	10.493	0.981	23.528	0.990	48.050	0.995	94.906	0.998	323.808
	CM	0.964	10.408	0.981	22.495	0.990	45.309	0.995	89.924	0.998	303.236
	XYZ	0.964	10.296	0.981	21.325	0.990	41.737	0.995	81.293	0.998	254.335
	Stoch.	0.964	11.920	0.981	35.886	0.990	94.117	0.995	199.614	0.998	845.022
	Hier.	0.964	10.482	0.981	21.833	0.990	42.404	0.995	82.886	0.998	260.273
	B	0.964	4.848	0.982	9.960	0.991	20.183	0.995	36.424	0.998	94.230
	BC	0.964	4.475	0.982	9.099	0.991	17.475	0.995	32.294	0.998	81.373
`CG-advanced`	2-lv	0.920	6.508	0.959	21.255	0.979	46.864	0.989	93.352	0.994	143.869
	CM	0.906	5.910	0.947	16.411	0.972	34.431	0.986	70.101	0.992	133.884
	XYZ	0.857	3.459	0.924	14.174	0.960	20.631	0.980	42.052	0.990	72.420
	Stoch.	0.914	7.788	0.955	35.711	0.976	99.214	0.988	213.072	0.993	520.788
	Hier.	0.904	5.759	0.947	16.006	0.972	32.002	0.986	49.700	0.993	107.674
	B										
	BC										
`MG-simple`	2-lv	0.320	4.726	0.325	5.693	0.327	6.302	0.327	6.559	0.326	8.397
	CM	0.320	4.762	0.325	5.636	0.327	6.129	0.327	6.395	0.326	8.219
	XYZ	0.320	4.708	0.325	5.395	0.327	5.659	0.327	5.759	0.326	6.736
	Stoch.	0.320	5.396	0.325	8.506	0.327	11.956	0.327	14.188	0.326	24.050
	Hier.	0.320	4.819	0.325	5.511	0.327	5.788	0.327	5.885	0.326	6.868
	B	0.342	1.791	0.344	1.815	0.344	1.731	0.343	1.684	0.342	1.788
	BC	0.342	1.791	0.344	1.543	0.344	1.412	0.343	1.371	0.342	1.416
`MG-advanced`	2-lv	0.117	1.909	0.121	1.942	0.125	2.162	0.128	2.245	0.138	3.448
	CM	0.044	1.342	0.055	1.476	0.064	1.660	0.072	1.837	0.084	2.766
	XYZ	0.036	1.199	0.041	1.202	0.044	1.256	0.045	1.290	0.045	1.680
	Stoch.	0.090	2.001	0.094	2.984	0.100	4.695	0.112	5.855	0.123	11.159
	Hier.	0.058	1.465	0.062	1.479	0.069	1.587	0.083	1.684	0.088	2.366
	B	0.057	0.690	0.056	0.518	0.053	0.468	0.050	0.465	0.044	0.490
	BC	0.057	0.690	0.056	0.432	0.053	0.404	0.050	0.408	0.044	0.427

Table 5.3: Convergence rates and total efficiency (in μs) for different solver configurations and sorting strategies. Note that the `CG-advanced` configuration is currently not supported in FEAST.

6 Conclusions and Future Work

Our proposed benchmark has a rather simple character by construction and should be computable by almost all numerical software packages designed for the simulation of PDEs. Our own benchmark results underline the following basic principles in numerical simulation:

1.) The data layout is of great importance for the efficiency of algorithms running on modern computer hardware. A simple re-sorting of the unknowns may improve the MFLOP/s rate of the matrix vector multiplication by a factor of 3, better data layouts even by a

factor of $10 - 30$. Very 'bad' data layouts (e. g. stochastic sorting) lets the performance of processors plummet to less that 1% of the peak computational performance of modern processors.

2.) The structure of the mathematical problem shall not get lost in the solvers for the linear subsystems that arise in the solution process. Utilising a hierarchical solver that exploits multigrid structures quickly pays off and may lead – in combination with the 'correct' data layout – to a speedup of a factor of 1300 and even more in computation time in comparison to a straightforward implementation of a simple numerical scheme. These numbers are a very strong argument in favour of our general approach to improve numerical and computational efficiency simultaneously: An extremely tuned implementation of a numerically poor algorithm is easily outperformed (despite the same linear complexity in the number of unknowns) by a more advanced numerical scheme, and advanced solvers implemented without awareness of the underlying machine architecture degrade asymptotically, indicating potential slowdowns on future hardware: It is not safe to assume that codes will continue to run faster automatically on newer hardware.

Our 'roadmap' in the evolution of this benchmark is as follows: On the numerical side, we want to include – in the short term – techniques such as higher order finite elements, finite differences, finite volumes and local adaptivity (e. g. with hanging nodes, mixed triangle- and quad-meshes). In the longer run, we want to accept the more challenging (but much more practically relevant) task to solve problems to a guaranteed, user-prescribed accuracy based on suitable error indicators: Ultimately, the metric *total efficiency* will have to be reformulated to reflect the time to gain this and that accuracy in the result. Global, goal-oriented adaptivity will be an important technique to pursue in future work.

On the computational side, the exploitation of multi- and many-core processors for FEM simulations is the next logical step. Current representatives of many-core architectures such as GPUs or the Cell processor are in our opinion key future topics in scientific computing. Our first results on GPUs show that we can meet our performance goals, a problem with 10^6 unknowns is solved accurately in less than 0.1 seconds [6]. We currently are not able to solve this benchmark problem with GPU acceleration, as our focus has been on the integration of GPUs into large-scale parallel solvers [10].

In the long term, our claims need to be put to the test in large-scale parallel simulations, and future work will have to investigate how scalability in parallel can be improved [10].

We kindly encourage everyone to perform these benchmark computations with other software packages. Our proposed benchmark is quite open in nature and allows many variations: While the `CG-simple` and `MG-simple` configurations are fixed to enable a direct comparison of different solver packages in terms of the total efficiency, the choice of an appropriate `xxx-advanced` setting that performes best is left unspecified. We emphasise again that fine-grained performance differences in the order of seconds are of no importance for this benchmark proposal.

Acknowledgements

This work has in parts been supported by the German Research Foundation (DFG), projects TU102/24-1 (SPP1253), TU102/22-1 and TU102/11-3.

References

[1] R. Barrett, M. Berry, T. F. Chan, J. Demmel, J. Donato, J. Dongarra, V. Eijkhout, R. Pozo, C. Romine, and H. V. der Vorst. *Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition*. SIAM, 1994.

[2] C. Becker, S. Buijssen, and S. Turek. FEAST: development of HPC technologies for FEM applications. In W. Nagel, D. Kröner, and M. Resch, editors, *High Performance Computing in Science and Engineering '07*, pages 503–516. Springer, Berlin, 2007.

[3] A. Chapman, Y. Saad, and L. Wington. High order ILU preconditioners for CFD problems. *International Journal for Numerical Methods in Fluids*, 33(6):767–788, 2000.

[4] FEAST – Finite Element Analysis and Solution Tools. `http://www.feast.tu-dortmund.de`.

[5] FEATFLOW. `http://www.featflow.de`.

[6] D. Göddeke and R. Strzodka. Performance and accuracy of hardware-oriented native, emulated- and mixed-precision solvers in FEM simulations (part 2: Double precision GPUs). Technical report, Fakultät für Mathematik, Technische Universität Dortmund, 2008. Nummer 370, invited talk at NVISION 2008 - The World of Visual Computing.

[7] D. Hysom and A. Pothen. Level-based incomplete LU factorization: Graph model and algorithms. Technical Report UCRL-JC-150789, U.S. Department of Energy, Nov. 2002. `http://www.cs.odu.edu/~pothen/papers.html`.

[8] Y. Saad and H. van der Vorst. Iterative solution of linear systems in the 20th century. *Journal of Computational and Applied Mathematics*, 123(1-2):1–33, 2000.

[9] S. Turek. On ordering strategies in a multigrid algorithm. In *Proc. 8th GAMM–Seminar*, volume 41 of *Notes on Numerical Fluid Mechanics*. Vieweg, 1992.

[10] S. Turek, D. Göddeke, C. Becker, S. Buijssen, and H. Wobker. UCHPC - Unconventional high-performance computing for finite element simulations. ISC'08, International Supercomputing Conference, Dresden, June 2008.

[11] S. Turek, A. Runge, and C. Becker. The FEAST indices – realistic evaluation of modern software components and processor technologies. *Computers and Mathematics with Applications*, 41:1431–1464, 2001.