Consequences of modern hardware design for numerical simulations and their realization in FEAST

Ch.Becker, S.Kilian, S.Turek, and the FEAST Group

Institut für Angewandte Mathematik, Universität Heidelberg, Germany
telephone: +49 6221 545446 fax: +49 6221 545634
WWW: http://www.iwr.uni-heidelberg.de/featflow
Email:featflow@gaia.iwr.uni-heidelberg.de

Abstract. This paper deals with the influence of hardware aspects of modern computer architectures to the design of software for numerical simulations. We present performance tests for various tasks arising in FEM computations and show under which circumstances performance loss can occur. Further we give key ideas for our new software package FEAST, which is especially designed for high performance computations.

1 Introduction

The situation in Scientific Computing is mainly influenced by following topics. Performing realistic calculations for instance in the field of Computational Fluid Dynamics requires still lots of computing resources like CPU–time and storage memory. On the other hand in the last years huge improvements in the hardware sector have taken place. The processors have become faster and faster and the memories bigger and bigger. This development will be continued in the future.

In about ten years one today so called "PC" will be so powerful as a nowadays complete Cray T3E. An important fact is that the memory access time will not shrink as fast as the processing speed grows. The gap between these main performance indicators will grow in the future! Therefore the performance potential is available, but are the nowadays scientific codes able to benefit from this potential?

On the other hand only to look at the implementation side is not enough. The best hardware adapted code cannot produce satisfying results, if the underlying numerical and algorithmic parts are not optimal. This has been shown by several benchmark computations like the DFG–Benchmark "Flow around a cylinder" ([6]). Improvements of the discretization and error control, which lead to less unknowns, are also necessary as efficient and robust solver strategies to achieve better convergence rates.

In the following we will examine the performance behavior of several numerical linear algebra routines for the solution of stiffness matrices arising in FEM computations. These routines are often the most time consuming part in the simulations. We will show that there are many traps to loose performance.

Special strategies are necessary to avoid these. To illustrate the strategies we will give some key notes about our software project FEAST (Finite Element Analysis & Solution Tools).

2 Typical examples for performance losses

One of the main components in iterative solution schemes, as for instance Krylov space methods or multigrid solvers, are Matrix–Vector (MV) applications. They are needed for defect calculations, smoothing, step–length control, etc., and often they consume 60 - 90% of CPU time and even more. Hereby *sparse MV* concepts are the standard techniques in Finite Element codes (and others) also well known as *compact storage* technique. Depending on the programming language the matrix entries plus index arrays or pointers are stored as long arrays containing the nonzero elements only. While this general *sparse* approach can be applied to general meshes and arbitrary numberings of the unknowns, no explicit advantage of (possible) highly structured parts can be exploited. Consequently a massive loss of performance with respect to the possible peak rates may be expected since — at least for large *sparse* problems with more than 100,000 unknowns — no "caching" and "pipelining" can be exploited such that the higher cost of memory access will dominate.

To demonstrate this failure we start with examples from our FEATFLOW code [11] which seems to be one of the most efficient simulation tools for the incompressible Navier–Stokes equations on general domains (see the results in [6]). We apply FEATFLOW to the following configuration of "2D flow around a car" and we measure the resulting MFLOP/s rates for the matrix–vector multiplication inside the multigrid solver for the momentum equation (see [8] for a more precise mathematical and algorithmic description). Here we apply the typical (for FEM approaches) "two level" (TL) numbering, a version of the bandwidth–minimizing Cuthill–McKee (CM) algorithm and an arbitrary "stochastic" numbering of the unknowns, which simulates adaptive refinement.

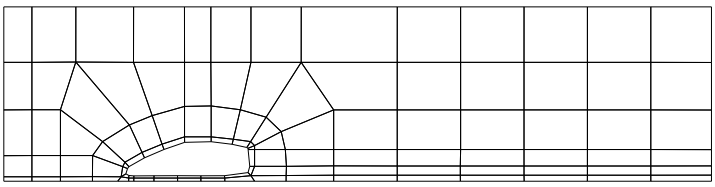

Fig. 1: Coarse mesh for "flow around a car"

All numbering strategies have common that based on the standard sparse *Compact Storage Rowwise (CSR)* technique the cost for arithmetic operations, for storage and for the number of memory accesses are identical. However the resulting timings in FORTRAN77 on a Sun Enterprise E450 (about 250 MFLOP/s peak performance!) can be very different as table 1 shows.

computer	#unknowns	**TL**	**CM**	**stochastic**
	13,688	20	22	**19**
SUN E450	54,256	15	17	**13**
($\sim$ 250 MFLOP/s)	216,032	14	16	**6**
(*CSR*)	862,144	15	16	**4**

Table 1: MFLOP/s rates of sparse MV multiplication for different numberings

These and numerous similar results can be concluded by the following statements which are quite representative for many other numerical simulation tools:

- **different** numbering strategies can lead to **identical** numerical results and work (w.r.t. arithmetic operations and memory access), but at the same time to **huge** differences in elapsed CPU time,
- sparse MV techniques are **slow** (with respect to possible peak rates) and depend massively on the problem size and the kind of memory access,
- in contrast to the mathematical theory most multigrid implementations will **not** show a realistic run–time behavior which is directly proportional to the mesh level.

Figure 2 which in contrast is based on the highly structured "blocked banded" MV techniques in FEAST shows that the same application can be performed in principle much faster: we additionally exploit **vectorization** facilities and **data locality**. Additionally we can even further differ between the case of *variable (var)* matrix entries and *constant bands (const)* as typical for Poisson–like PDE's. In comparison the same task is performed for a matrix in CSR and adaptive technique. Figure 2 is an excerpt from the Feast Indices ([9]) which is a general framework of performance measurements for many processors. It contains measurements for several vector, matrix-vector, smoothing and complete multigrid operations in 2D/3D.

Finally to show that high performance is achievable in more sophisticated algorithms, table 2 shows the performance rates for a complete multigrid cycle. As it can be seen a multigrid with about 450 MFLOP/s is possible. For more information about sparse banded BLAS, see [10].

3D case	N	MV-V	MV-C	TRIGS-V	TRIGS-C	**MG-V**	**MG-C**
DEC 21264	17^3	446	765	211	300	342	500
(500 MHz)	33^3	240	768	136	310	233	474
'**DS20**'	65^3	**249**	**713**	**109**	**319**	**196**	**447**
IBM RS6000	17^3	288	730	130	185	214	367
(200 MHz)	33^3	216	737	104	193	186	380
'**PWR3**'	65^3	**174**	**710**	**91**	**198**	**154**	**363**

Table 2: MFLOP/s rates of different sparse banded BLAS components (matrix–vector operation, TriGS smoother and multigrid cycle for variable/constant matrix entries)

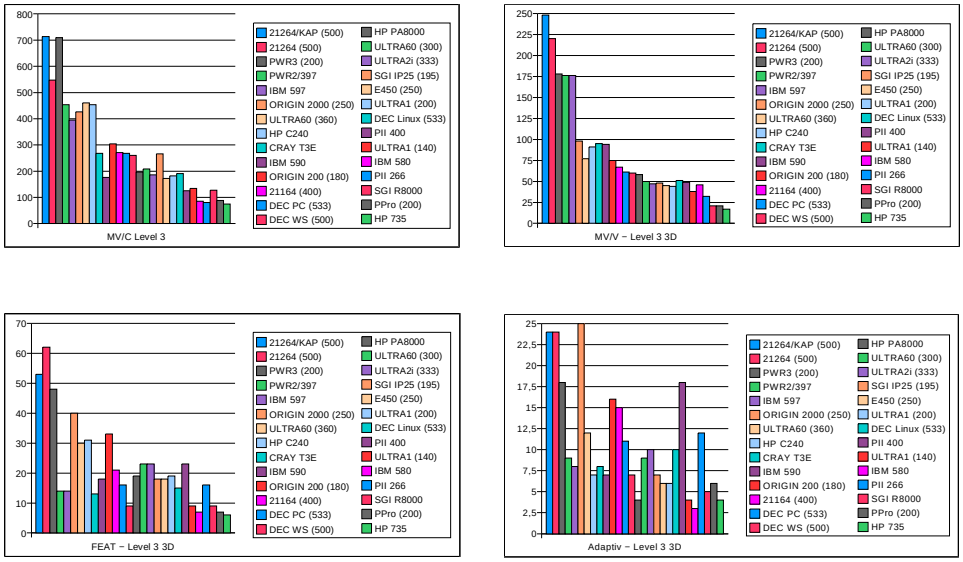

Fig. 2: The figures show the MFLOP/s for the discussed MV multiplications (MV-C/MV-V (blocked banded MV), FEAT (CSR MV), ADAP ("stochastic" MV)) which lead for certain computers as the IBM PWR2's to performance differences of almost a factor of 100! While modern workstation processors show a huge potential of supercomputing power for structured data, they loose for unstructured data in combination with sparse MV techniques.

3 FEAST: Main Principles

3.1 Hierarchical data, solver and matrix structures

One of the most important principles in FEAST is to apply consequently a *(Recursive) Divide and Conquer* strategy. The solution of the complete global problem is recursively split into smaller "independent" subproblems on "patches" as part of the complete set of unknowns. Thus the two major aims in this splitting procedure which can be performed by hand or via self–adaptive strategies are:

- *Find locally structured parts.*
- *Find locally anisotropic parts.*

Based on "small" structured subdomains on the lowest level (in fact, even one single or a small number of elements is allowed), the "higher–level" substructures are generated via clustering of "lower–level" parts such that algebraic or geometric irregularities are hidden inside the new "higher-level" patch.

Figure 3 illustrates exemplarily the employed data structure for a (coarse) triangulation of a given domain and its recursive partitioning into several kinds of substructures.

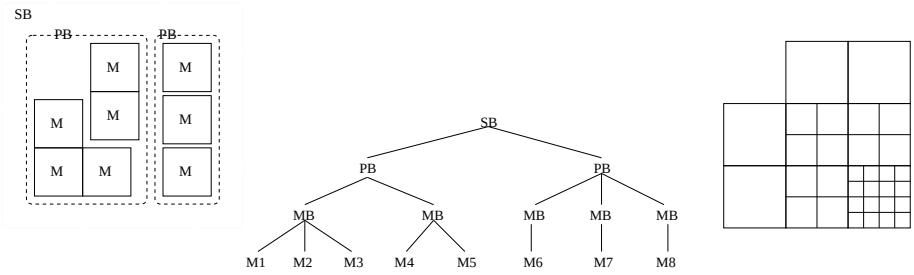

Fig. 3: FEAST domain structure

According to this decomposition, a corresponding data tree – the skeleton of the partitioning strategy – describes the hierarchical decomposition process. It consists of a specific collection of elements, macros (Mxxx), matrix blocks (MB), parallel blocks (PB), subdomain blocks (SB), etc.

The *atomic units* in our decomposition are the "macros" which may be of type **structured** (as $n \times n$ collection of quadrilaterals (in 2D) with sparse banded BLAS data structures) or **unstructured** (any sparse collection of elements). These "macros" (one or several) can be clustered to build a "matrix block" which contains the "local matrix parts": only here is the complete matrix information stored. Higher level constructs are "parallel blocks" (for the parallel distribution and the realization of the load balancing) and "subdomain blocks" (with special conformity rules with respect to grid refinement and applied discretization spaces). They all together build the complete domain, resp. the complete set of unknowns. It is important to realize that each stage in this hierarchical tree can act as independent "father" in relation to its "child" substructures while it is a "child" at the same time in another phase of the solution process (inside of the ScaRC solver).

3.2 Generalized solver strategy ScaRC

In view of their typically "excellent" convergence rates, multigrid methods seem to be most suited for the solution of PDE's. However as the examples of the previous chapter have shown, multigrid on general domains has often poor computational efficiency, at least if the implementation is based on the standard sparse techniques. As a result from our performance measurements (see [9]), the realistic MFLOP/s rates are often in the range of 1 MFLOP/s only, even on very modern high performance workstations. Moreover the linear relationship between problem size and CPU time is hardly realizable, due to the problem–size dependent performance rates of the sparse components. Additionally the robust treatment of complex mesh structures with locally varying details is hard to satisfy by typical "black box" components, even for ILU smoothing (see [1]).

Concerning the parallelization, some further serious problems occur: The parallel efficiency is often bad and far beyond peak, since the solution of the coarse

grid problem leads to a large communication overhead. Furthermore the relation between "local" arithmetic operations and "global" data transfer is poor. Besides these computational aspects, the parallelization of the global smoothers (SOR, ILU) cannot be done efficiently because of their recursive character. So smoothing can only be performed block by block, which may lead to a deterioration of the convergence rates. Further the behavior of such block by block smoothing is not clear for complicated geometries with local/global anisotropies.

Motivated by these facts, we started to develop a new strategy for solving discretized PDE's which should satisfy several conditions:

The parallel efficiency shall be high due to a non–overlapping decomposition and a low communication overhead. The convergence rates are supposed to be independent of the mesh size h, the complexity of the domain and the number of subdomains N, and they should be in the range of typical multigrid convergence rates (as $\rho_{MG} \sim 0.1$*). Further the method should be easily implementable and use only existing standard methods. The approach should guarantee the treatment of complicated geometries with local anisotropies (large aspect ratios!), without impairment of the overall convergence rates.*

The underlying idea is *hiding recursively all anisotropies in single subdomains* combined with a corresponding *block Jacobi/Gauß-Seidel smoothing* within a standard multigrid approach. This approach is based on the numerical experience that the "simple" block Jacobi/Gauß-Seidel schemes perform well as soon as all occurring anisotropies are "locally hidden", that means if the local problems on each block are solved (more or less) exactly. This procedure ensures the global robustness! On the other hand this means that the "local" solution quality in each block can significantly improve the "global" convergence behavior!

These ideas are combined with the previously explained hierarchical data and matrix structures, which prefer tensorproduct–like meshes on each *macro* and which allow to achieve very high performance rates for the necessary numerical linear algebra components in the local (multigrid) solvers. Consequently all solution processes are recursively performed via sequences of more "local" steps until the lowest level, for instance a single *macro* with the described generalized tensorproduct mesh is reached.

The complete ScaRC approach (see [2], [3], [4]) can be characterized via:

- **Sca**lable (w. r. t. quality and number of local solution steps at each stage)
- **R**ecursive ("independently" for each stage in the hierarchy of partitioning)
- **C**lustering (for building blocks via fixed or adaptive blocking strategies)

Numerous examples for this solver strategy can be found in the above mentioned papers. In this paper we shortly present a test calculation for ScaRC on a "flow around the cylinder" topology (figure 4).

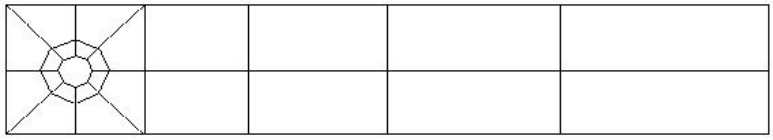

Fig. 4: 'Flow around the cylinder' topology (diameter of cylinder = 0.1)

In table 3 the quantity N_{local} denotes the local number of elements in one space direction per *macro*. We perform $l = 1$ and $l = 2$ (global) smoothing steps and solve the local problems on each *macro* "exactly" via local multigrid. The *macros* around the circle are anisotropically refined in direction towards the circle, such that a finest mesh size $h_{min} \approx 2 \cdot 10^{-7}$ is obtained. The resulting (parallel) convergence rates are the same for all kinds of local refinement, independent of the amount of local anisotropy ("AR" means "aspect ratio"): The rates are all in the range of $\rho \sim 0.1$, for $l = 1$, and still much better for $l > 1$.

N_{local}	l	isotropic		moderately anisotropic		highly anisotropic	
		$\#ITE$	ρ_{ScaRC}	$\#ITE$	ρ_{ScaRC}	$\#ITE$	ρ_{ScaRC}
16	1	**7**	0.121	**7**	0.112	**6**	0.070
	2	**4**	0.018	**4**	0.017	**3**	0.007
32	1	**7**	0.124	**6**	0.091	**6**	0.066
	2	**4**	0.023	**4**	0.018	**3**	0.005
64	1	**7**	0.121	**6**	0.081	**6**	0.065
	2	**4**	0.026	**4**	0.017	**3**	0.003
		$AR = 3$		$AR = 50$		$AR = 3,000$	
		$h_{min} \approx$ 6.1-4		$h_{min} \approx$ 1.2-5		$h_{min} \approx$ 1.9-7	

Table 3: ScaRC results for different degrees of local anisotropy

It is obvious that this parallel approach requires a modified concept of load balancing. Since in general there will be more *macros* than processors - realistic coarse meshes in 3D for complex applications require at least 1,000 up to 10,000 elements - such that each processor will provide the data for several *macros*. First of all the amount of storage on each *macro* is not identical if for instance the case of variable or constant matrix coefficients can be exploited on the tensorproduct–like meshes, or if auxiliary storage for the sparse techniques is needed. Moreover the application of ILU requires different storage cost than Jacobi– or Gauß-Seidel smoothing. However much more decisive is the actual CPU time on each *macro* for the solution of the local problems: The elapsed time depends significantly on the corresponding computational efficiency and the numerical efficiency, that means the actual convergence rates which determine the number of multigrid sweeps, and hence the total number of arithmetic operations. Additionally the problem size on each *macro* may vary.

Hence parallel load balancing is much more complex for this ScaRC approach and cannot be determined via a priori strategies, for instance by equilibrating the number of unknowns on each processor. Instead we have to perform a posteriori load balancing techniques, analogously to the adaptive approaches,

which are based on the numerical and computational run–time behavior of the last iterate to equilibrate better the total required CPU time on each processor.

4 Conclusions and Outlook

Our numerical examples show the advantages of the proposed method ScaRC with respect to efficiency and robustness. This method allows parallelization and the use of standard numerical methods as basic modules for smoothing and solving. Further it makes it possible to use very regular data structures which enables high performance facilities.

Our computational examples have shown that there is a large gap between the LINPACK rates of several hundreds of MFLOP/s as typical representatives for direct solvers and the more realistic results for iterative schemes. Moreover certain properties of cache management and architecture influence the run time behavior massively, which leads to further problems for the developer. On the other hand the examples show that appropriate algorithmic and implementation techniques lead to much higher performance rates. However massive changes in the design of numerical, algorithmic and implementation ingredients are necessary.

References

1. Altieri, M.: *Robuste und effiziente Mehrgitter-Verfahren auf verallgemeinerten Tensorprodukt-Gittern*, Diploma Thesis, to be published
2. Becker, Ch.: *The realization of Finite Element software for high-performance applications*, PhD Thesis, to appear
3. Kilian, S.: *Efficient parallel iterative solvers of* ScaRC*-type and their application to the incompressible Navier-Stokes equations*, PhD Thesis, to appear
4. Kilian, S., Turek, S.: *An example for parallel* ScaRC *and its application to the incompressible Navier-Stokes equations*, Proc. ENUMATH-97, Heidelberg, October 1997.
5. Rannacher, R., Becker, R.: *A Feed-Back Approach to Error Control in Finite Element Methods: Basic Analysis and Examples*, Preprint 96–52, University of Heidelberg, SFB 359, 1996.
6. Schäfer, M., Rannacher, R., Turek, S.: *Evaluation of a CFD Benchmark for Laminar Flows*, Proc. ENUMATH-97, Heidelberg, October 1997.
7. *The national technology roadmap for semiconductors*, 1997 edition, `http://www.sematech.org/public/roadmap/index.htm`
8. Turek, S.: *Efficient solvers for incompressible flow problems: An algorithmic approach in view of computational aspects*, LNCSE, Springer-Verlag, 1998.
9. Turek, S. et al.: *The* FEAST Indices *- Realistic evaluation of modern software components and processor technologies*, to appear
10. Turek, S. et al.: *Proposal for* `Sparse Banded BLAS` *techniques*, to appear
11. Turek, S.: *FEATFLOW : Finite element software for the incompressible Navier-Stokes equations: User Manual*, Release 1.1, 1998