The FEAST INDICES - Realistic evaluation of modern software components and processor technologies

S. Turek, Chr. Becker, A. Runge and the FEAST Group
Institut für Angewandte Mathematik, Universität Heidelberg
Im Neuenheimer Feld 294, 69120 Heidelberg, Germany
ture@gaia.iwr.uni-heidelberg.de http://gaia.iwr.uni-heidelberg.de/~ture

Summary. We examine the computational efficiency of Linear Algebra components in iterative solvers for grid-oriented simulations of PDE's. While the standard *sparse* matrix-vector (MV) techniques show significant losses of performance, especially on modern processors, out *sparse banded* components have the potential to exploit today's high computing power. We explain the major concepts of the FEAST software which contains such highly tuned Numerical Linear Algebra basic components (SPARSE BANDED BLAS) up to complete multigrid solvers, all being optimized with respect to the actual hardware platform. Based on algorithmic and computational studies, we present the FEAST INDICES which are indicators for the *true* performance of many modern processors, depending on the underlying FEM space, the problem size and the implementation style. These indices allow a new rating of the various hardware platforms with regard to different mathematical solution strategies, for academic and realistic numerical problems and ranging from 'low cost' PC's up to supercomputers.

1 Motivation

One current trend in the software development for PDE's, and here especially for Finite Element (FEM) approaches, goes clearly towards very sophisticated **object-oriented** techniques and **adaptive** methods in any sense. On the other hand, the employed data and solver structures, and particularly the 'matrix structures' (for an overview, see [15] and [16]), are mostly chosen in a somewhat old-fashioned way - as 'globally defined' types - which neglect the very specific performance facilities of modern hardware platforms. As a result, the observed computational efficiency is often far from the expected peak rates of (potentially available) 1 GFLOP/s nowadays, and the 'real life' gap is even further increasing if one extrapolates current hardware developments (see also the papers of U. Rüde, for instance [6]).

Today, high performance calculations in this field seem to be reachable only by explicitly exploiting 'caching in' and 'pipelining' in combination with sequentially stored arrays (see [3]). While the realization of such techniques tends to be easier for Finite Difference approaches, it is far from being obvious how to perform similar approaches for much more complex Finite Element codes. These discrepancies, between Numerics and software concepts and the available hardware, often lead to unreasonable calculation times for 'real world' problems, e.g. (nonstationary) CFD calculations in 3D, as can be easily seen from recent benchmark comparisons for commercial as well as research codes (for instance, see [8] and [10]). Hence, strategies for massive efficiency enhancement are necessary, not only from the mathematical (algorithms, discretizations) but also from the software side of view. To realize some of these aims our new FEM package FEAST ('Finite Element Analysis & Solution Tools') is under development (see [11] and [12]).

In this paper, we concentrate on the aspect of **testing** Linear Algebra routines for the stiffness matrices resulting from FEM discretizations. These are the most important components in the corresponding multigrid framework which is mainly responsible for the total efficiency of the complete simulation. First of all, we demonstrate the performance of typical software tools based on standard *sparse* MV techniques, before we explain our machine-dependent SPARSE BANDED BLAS approaches [14] for achieving strong performance improvements! Consequently, these techniques are the basis for our recent evaluation of software components and modern processor technologies which are collected in several types of FEAST INDICES. These explain, for instance, why many codes run faster on 'low cost' PC's than on single processors of supercomputers. However, they also demonstrate the (hidden) potential of supercomputing power on modern (workstation) processors as employed by DEC, IBM, HP, SUN and SGI.

2 Typical examples for performance losses

One of the main components in iterative solvers - Krylov-space methods or multigrid - 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. Hereby, *sparse MV* concepts are the standard techniques in FEM codes (and others), also well known as 'compact storage' technique: Depending on the programming language, the matrix entries plus index arrays/lists/pointers are stored as long arrays or heaps, containing the 'nonzero elements' only. For an overview on applied techniques, see for instance SPARSKIT [16] and the literature cited therein. While this *sparse* approach can be applied for 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 problems with more than 100,000 unknowns - no 'caching in' and 'pipelining' can be exploited such that the higher cost of memory access will dominate the resulting MFLOP/s rates.

To demonstrate this failure, we start with examples from our FeatFlow code [10] which seems to be one of the most efficient simulation tools for incompressible flow on general domains (see the results in [8]). We apply FeatFlow to the following "2D flow around a car" configuration, and we measure the resulting MFLOP/s rates for the MV multiplication inside of the multigrid solver for the momentum equation (see [10] for mathematical and algorithmic details). Here, we use the typical (for FEM approaches) 'two level' (TL) numbering (old vertices preserve their numbers if the mesh is refined!), a version of the bandwidth-minimizing Cuthill-McKee (CM) algorithm and an arbitrary 'stochastic' numbering.

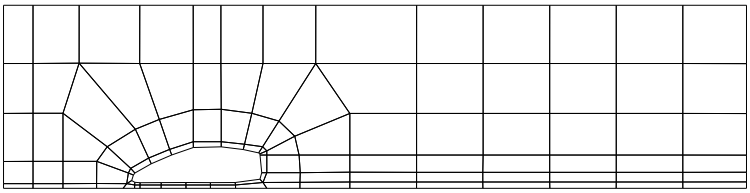


Figure 1: Coarse mesh for 'Flow around a car'

All numbering strategies have common that, based on the standard *sparse* 'Compact Storage Rowwise' (CSR) technique (see [16]), 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 (see also Section 6: 'The FeatFlow Benchmark') can be concluded by the following statements which are quite representative for many other numerical simulation tools:

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

To shed more light into this 'strange' behaviour, we examine more carefully the computer performance for 2nd order scalar PDE's on tensorproduct meshes with trilinear FEM spaces ($\approx$ matrices with bandwidth 27). Again, we apply different numbering strategies in the *sparse* CSR format, however the numerical work and the results of the MV operations are identical for all cases!

computer	#unknowns	'rowwise'	'two level'	'stochastic'
IBM RS6000/597	17^3	86	81	81
(160 MHz)	33^3	81	55	16
'**P2SC**'	65^3	**81**	**14**	**8**
IBM RS6000/590	17^3	42	42	42
(66 MHz)	33^3	41	39	27
'**POWER2**'	65^3	**41**	**17**	**7**
DEC/21164	17^3	54	31	29
(433 MHz)	33^3	51	16	10
'**CRAY T3E**'	65^3	**49**	**13**	**8**
INTEL PENTIUM II	17^3	30	28	28
(400 MHz)	33^3	30	26	24
'**ALDI PC**'	65^3	**30**	**23**	**19**

Table 2: MFLOP/s rates of *sparse* MV multiplication on tensorproduct meshes

Based on such studies [1] , we can characterize more precisely the computational run-time behaviour:

- The MFLOP/s rates are far away from the announced peak performance.
- They depend significantly on the **size** of the problem and the '**kind of memory access**'.
- 'Old' processors (IBM 590) can be even faster (!) than 'new' ones (IBM 597).
- 'Supermarket' PC's can be significantly faster than processors in 'supercomputers'!

The following Table, which now in contrast is based on the highly structured MV techniques in FEAST, shows that the same application can (!) be performed much faster: We additionally exploit **vectorization** facilities and **data locality**. Additionally, we can even further differ between the case of *variable matrix* entries ('var') and *constant bands* ('const') as typical for Poisson-like PDE's.

computer	#unknowns	'var'	'const'	'computer	'var'	'const'
IBM RS6000/597	17^3	188	480	IBM RS6000/590	102	195
(160 MHz)	33^3	172	393	(66 MHz)	94	175
'**P2SC**'	65^3	176	390	'**POWER2**'	94	176
DEC/21164	17^3	103	404	INTEL PENTIUM II	51	180
(433 MHz)	33^3	101	313	(400 MHz)	51	137
'**CRAY T3E**'	65^3	101	268	'**ALDI PC**'	48	124

Table 3: MFLOP/s rates of SPARSE BANDED BLAS [14] MV multiplication on tensorproduct meshes

Such differences in performance are caused by the fact that modern processors are already sensible supercomputers with respect to 'caching in' and 'pipelining' which has been explicitely exploited by the SPARSE BANDED BLAS-based MV techniques. However, the following examples show that also the kind of arithmetic work has to be considered ('division-free Numerical Linear Algebra') and that a precise knowledge of the cache architectures is necessary.

[1] ALDI is a supermarket in Germany which is well-known for its cheap prices but also high-quality goods. Based originally on household and food stuffs only, ALDI has offered a typical PENTIUM II/400MHz computer which has been sold out due to its cheap price and particularly ALDI's good reputation: About 250,000 pieces in 2 weeks!

Computer	#unknowns	'classic'	'var'	'const'
IBM RS6000/597	65^2	33	32	33
(160 MHz)	257^2	32	32	33
'**P2SC**'	1025^2	32	32	33

'classic'	'var'	'const'
144	130	141
80	105	150
80	106	150

Table 4: MFLOP/s rates for tridiagonal preconditioners in 'classical' formulation, compared with the SPARSE BANDED BLAS versions from FEAST for 'variable' and 'constant' matrix entries. The left table shows the results obtained from traditional implementations **with** division, while the right table gives the rates **without** division. It is obvious that divisions in an algorithm can significantly degradate the performance if compared with pure addition/multiplication-based algorithms.

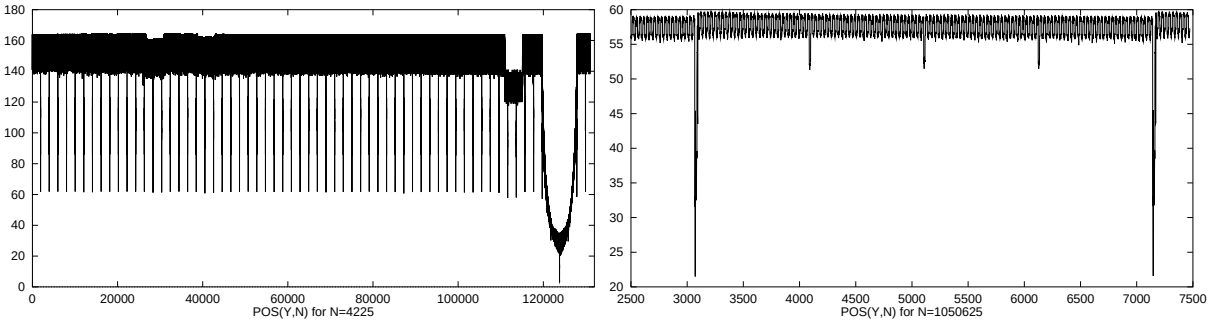


Figure 2: MFLOP/s rates of DAXPY operations based on machine-optimized performance libraries: We only vary the difference `POS` in the relative position to each other of the both vectors to be added. The results for the SUN E450 (left, 1 MByte L2-cache) are representative for processors with 1-fold associative cache architectures: The actual performance of such 'simple' Numerical Linear Algebra tasks of BLAS1-type depends massively on the relative position `POS` and varies between 160 and 3 (!!!) MFLOP/s although both small vectors fit completely into cache! The right figure shows similar losses of performance for the CRAY T3E, here for long vectors. These examples demonstrate that such failures can be only avoided by a hardware-specific and user-defined memory managment!

To make this point clear: The use of tensorproduct meshes does **not (!!!)** automatically guarantee higher performance; all proposed optimization strategies with respect to data structures, algorithmic re-design, programming language, cache architectures and memory managment must be considered!

The following Table 5 shows the expected development of the (actual) processor technology and demonstrates that such 'machine-oriented' algorithmic and implementation techniques will be absolutely necessary: The explicit handling of data locality, internal parallelism and vectorization, in contrast to minimizing the memory access at the same time, will be the key techniques for the next years.

Year of 1st shipment	**1997**	1999	2001	2003	2006	2009	**2012**
Local clock (MHz)	**750**	1250	1500	2100	3500	6000	**10K**
Transistors/chip	**11M**	21M	40M	76M	200M	520M	**1.4B**

Table 5: Excerpts from the '1997 Semiconductor Roadmap' [18]

Then, it might get really possible to exploit more adequately the performance of future processors which tend to be even more powerful - as single processors with up to 1 TFLOP/s - than the complete CRAY T3E today! On the other hand, neglecting these features may not only lead to non-significant performance acceleration, but even **performance degradation** is possible as the previous tables have shown (compare the 'old' IBM 590 with the 'new' IBM 597. At the moment, most of today's *sparse* implementations (see also Section 6: 'The FEATFLOW Benchmark') would favourize the use of 'cheap' INTEL-based computers only. This development might lead to an economic (for certain companies) and much more to a scientific desaster as the performance rates will show.

3 Description of the FEAST project

FEAST is based on the following concepts (see [11] and [12] for details) which shall enable the combination of such highly tuned Linear Algebra tools with very sophisticated FEM simulation strategies:

- consequent application of (recursive) 'Divide and Conquer' strategies
- hierarchical data and solver structures, but also hierarchical (!) 'matrix structures'
- frequent use of machine-optimized low level Numerical Linear Algebra routines

The result shall be a flexible FEM package for many 'real life' problems with special emphasis on:

- (closer to) peak performance on modern processors
- typical multigrid behaviour (with respect to efficiency and robustness)
- parallelization and vectorization directly included on 'low level'
- open for different adaptivity concepts and a posteriori error control

In contrast to many other approaches which aim to develop software for research or education topics, our approach is clearly designed for high performance applications with industrial background, especially in *CFD*. Consequently, our main emphasis lies on the aspects 'efficiency' and 'robustness' and less on topics as 'easy implementable' or 'most modern programming language'. Therefore, FORTRAN77/90 is used such that (for us absolutely necessary) the transparent access to the data structures is possible. Further, this makes it possible to adopt many reliable parts of the predecessor packages FEAT2D, FEAT3D and FEATFLOW. One of the most important principles in FEAST is the consequent application of *Divide and Conquer* strategies. The solution of a 'global' problem is recursively split into smaller independent subproblems on 'patches' as part of the complete set of unknowns. There are two major aims in this splitting procedure which can be performed by hand or via self–adaptive strategies:

- *Find and exploit locally* **structured** *parts*
- *Find and hide locally* **anisotropic** *parts*

While on such 'anisotropic' parts the usual *sparse* techniques will be applied, we try to exploit the much higher performance on the other - highly structured - patches. Consequently, the intention is to minimize the number of 'sparse areas' and to apply preferably all Numerical Linear Algebra tasks on such 'structured patches'. Then, the major three tasks for realizing such a simulation tool are:

1. The design of the 'skeleton' for the recursive splitting into local/global levels
2. The implementation of the typical FEM facilities on the 'low level' patches
3. The development of 'reference element solvers' on the 'low level' patches

The corresponding data, solver and matrix structures are described in the papers [11], [12] and particularly in [2]. The aim of this work is a (careful) description of the third task: Optimization of the 'reference element solvers' and the corresponding Numerical Linear Algebra tools on the 'low level' patches. In our context, these are quadrilaterals (2D), resp., hexaeders (3D) which are discretized with (logically equivalent) tensorproduct meshes. That means, we can (!) apply linewise or rowwise numbering, but the local mesh size may vary arbitrarily! This optimization procedure is split into two tasks:

The *manual* task of algorithmic design and corresponding implementation w.r.t. optimal MFLOP/s rates

The *numerical* task to derive 'optimal' multigrid convergence with respect to efficiency and robustness

While the *numerical* task, the optimization of multigrid convergence rates for different FEM spaces, problem sizes, (PDE) problem types and mesh topologies (but on tensorproduct meshes!) is currently examined in [1], we show recent results of the 'MFLOP/s optimization' in this paper (see also [14] for the technical details of the SPARSE BANDED BLAS. To be precise, we provide the reader with:

1. 'Optimal' MFLOP/s rates for different Numerical Linear Algebra tasks on generalized tensorproduct meshes (for bilinear and trilinear FEM!)
2. Evaluation of many modern processors with respect to the 'real' performance of such different basic tasks up to complete multigrid algorithms

4 Description of the Numerical Linear Algebra components

All following ratings and the test software (`ptest.f`) are part of the Internet and can be downloaded from our Homepage. In this paper, we publish the first version of more or less complete results for (almost) all modern hardware platforms. Since everybody has access to the data, everybody is invited to check new processors or different software environments to look for improved FEAST INDICES ! Our hope is that this (permanent) process of testing and rating of new hardware components gets into an automatic loop if sufficiently many 'test persons' participate. On the one hand, some 'pressure' on the vendors is generated, for instance to provide the users with optimal compiler options. On the other hand, there is a fair 'competition' possible between the various hardware configurations. So, the user and also 'client' of such products has the chance to compare the different test environments with regard to his own applications and with respect to the true cost/performance relation. To provide this kind of information is definitely one of the aims of the subsequent FEAST INDICES !

We will not explain the underlying tests in all details, and we will not show the results of all comparisons: These are part of the Diploma Thesis of A. Runge [7], resp., see also [14] and our Homepage! Therefore, in short terms only, the following Linear Algebra tasks are the basic components of the FEAST INDICES :

4.1 `DAXPY`-like applications

Beside the (standard) linear combination `DAXPY`, we also apply the (variable) linear combination `DAXPYV` $\boxed{y(i) = \alpha(i)x(i) + y(i)}$ which is important in banded MV multiplications on vector computers. Additionally, we check the (indexed) variant `DAXPYI` $\boxed{y(i) = \alpha(i)x(j(i)) + y(i)}$. Here, the scaling factor α is a vector acting on the components of the vector x which depend on the index i via an *index vector* $j(i)$. We apply two tests with different index vectors $j(i)$ which simulate moderate and stochastic jumps in the numberings. These tests are quite good representants for the complete *sparse* MV applications.

The arithmetic work count for all `DAXPY`-like variants is defined as $2 \times N$, with N the number of vector components, such that the corresponding MFLOP/s rates are determined via:

$$\frac{2 \times N}{CPUTIME \times 10^6}$$

4.2 Variants of `MV` multiplications

Assuming a (generalized) tensorproduct mesh with M vertices in each space dimension, the resulting number of unknowns is M^2 (2D), resp., M^3 (3D) if we consider vertex-oriented discretizations: Here conforming bilinear, resp., trilinear FEM! Assuming the typical 9–point, resp., 27–point stencil for the corresponding matrices, the resulting storage cost and hence the measure for the MFLOP/s rates are

identical for all following techniques, independent of the kind of MV multiplication! The test program `ptest.f` examines 8 different basic implementations (plus various blocking techniques, see [14] for the technical details) which all are designed to exploit potentially 'caching in' and 'vectorization' facilities. The `MV-V` results correspond to the case of arbitrary matrix entries, while `MV-C` represents the case of constant band entries as typical for Poisson-like problems on (in each 'local direction') equidistant meshes.

Additionally, we perform measurements for a corresponding *sparse* MV application in the CSR format as described above. We examine three variants of numberings: Linewise numbering but nevertheless indexed access (`SPARSE`), 'two level' numbering (`FEAT`) as typical for semi-adaptive FEM simulations without local adaptivity, and finally 'stochastic' numbering (`ADAP`) of the unknowns being representative for fully adaptive approaches. To make this point clear: All MV applications are performed for the same matrix! We only vary the storage and access techniques, hereby exploiting the tensorproduct structure or not, taking into account the case of constant entries or not. However, in all cases we define the work count to measure the MFLOP/s rates as:

$$\frac{18 \times N}{CPUTIME \times 10^6} \quad \text{(in 2D)} \quad \text{resp.,} \quad \frac{54 \times N}{CPUTIME \times 10^6} \quad \text{(in 3D)}$$

Moreover, we test the performance of matrix-vector multiplications with tridiagonal matrices which are basic tools for certain preconditioners as the 'linewise GS' schemes below. Again, we check in `ptest.f` the MFLOP/s rates for variable and constant entries, and they are determined via:

$$\frac{6 \times N}{CPUTIME \times 10^6}$$

4.3 Tridigonal-based preconditioners

Assuming tensorproduct meshes, the application of 'inverse' tridiagonal matrices as preconditioners (`TRIS`) can be easily performed and provides rather good convergence properties with respect to mesh anisotropies (see [1]). We apply the 'division-free' variants from the previous Section, with various blocking strategies, again for variable and constant matrix entries. The MFLOP/s rates are defined as:

$$\frac{5 \times N}{CPUTIME \times 10^6}$$

Additionally, this tridiagonal preconditioner can be combined with the described tridiagonal MV multiplications to work as 'linewise Gauß-Seidel' (`TRIGS`) or 'linewise ADI' preconditioners (see [14]). Taking into account the convergence studies in [1], these schemes are our actual favourites as multigrid smoothers with regard to numerical and computational efficiency. The MFLOP/s rates are defined as:

$$\frac{11 \times N}{CPUTIME \times 10^6} \quad \text{(in 2D)} \quad \text{resp.,} \quad \frac{29 \times N}{CPUTIME \times 10^6} \quad \text{(in 3D)}$$

4.4 Smoothers in multigrid

Based on the previously described `DAXPY`-like operations, the `MV` multiplications and the proposed tridiagonal preconditioners, we can determine the MFLOP/s rates of the corresponding smoothing operators which all are written and implemented in the following general notation:

$$\boxed{x^{l+1} = x^l - \omega C^{-1}(Ax^l - b)}$$

Here, A and C are matrices in $\mathbf{R}^{N\times N}$, with C being the *preconditioner*, and x^l, x^{l+1}, b are N–dimensional vectors. The parameter ω is an arbitrary relaxation factor while the indices l and $l+1$ are the usual counters in iterative procedures. Our candidates for the *preconditioner* C are:

- $C = \mathrm{diag}(A)$ corresponds to Jacobi iteration ($\approx$ `DAXPYV`)
- $C =$ `TRIS`(A) corresponds to tridiagonal preconditioners, resp., linewise variants of Jacobi
- $C =$ `TRIGS`(A) corresponds to the 'lower+tridiagonal' preconditioner, resp., linewise Gauß-Seidel

There are several reasons why we explicitly use and optimize this form of the *basic iteration* which is in contrast to many 'red-black' or other 'multi-colouring' approaches:

1. This general form allows the independent splitting into the three tasks MV multiplication, preconditioning and linear combination which all have been optimized with respect to 'caching in' and 'pipelining'.
2. The explicit use of the complete defect $Ax^l - b$ is advantageous in certain techniques for implementing complicated or 'moving' boundary conditions (see [10]).
3. All components in standard multigrid, i.e., smoothing, defect calculation, step-length control, grid transfer, are included in this *basic iteration*.

As an example, the MFLOP/s rates for 'linewise GS' smoothing `S-TRIGS(N)` which consists of `DAXPY`, `MV` and `TRIGS` (taking into account the case of variable (V) or constant (C) entries) are calculated separately on each mesh level, corresponding to problem size N. They read in 3D:

$$MFLOP_{S-TRIGS(N)} := \frac{(54+29+2)\times N}{(\frac{54\times N}{MFLOP_{MV(N)}} + \frac{29\times N}{MFLOP_{TRIGS(N)}} + \frac{2\times N}{MFLOP_{DAXPY(N)}})\times 10^6}$$

In an analogous way, the corresponding MFLOP/s rates for Jacobi or tridiagonal smoothing are estimated, based individually on the previously calculated rates in 2D and 3D with respect to the underlying Numerical Linear Algebra components.

4.5 Complete multigrid cycle

Finally, we can (recursively) determine the MFLOP/s rates for a standard multigrid cycle, consisting of m total smoothing steps (including pre- and postsmoothing steps; here: $m = 2$), defect calculation, grid transfer and coarse grid approximation. As an example, the actual rates for 'Line GS' smoothing on level l with $N(l)$ unknowns read for the corresponding multigrid variant `M-TRIGS(N(l))` in 3D:

$$MFLOP_{M-TRIGS(N(l))} :=$$

$$\frac{1.5\times(85m+54+18)\times N(l)}{(\frac{85m\times N(l)}{MFLOP_{S-TRIGS(N(l))}} + \frac{54\times N(l)}{MFLOP_{MV(N(l))}} + \frac{9\times 2\times N(l)}{MFLOP_{DAXPY(N(l))}} + 0.5\times \frac{(85m+54+18)\times N(l)}{MFLOP_{M-TRIGS(N(l-1))}})\times 10^6}$$

The factor 1.5 is achieved from the recursion through the applied W-cycle and can be analogously determined for other cycles and the 2D case (see [7] for the details).

4.6 Measurements of classical *sparse* applications

In 3D we have directly included a *sparse* MV multiplication for the same matrix as before, which results from a trilinear FEM discretization of a scalar Poisson-like problem and which leads to a (maximum) matrix stencil of 27. Then, the total number of vertices, resp., unknowns is defined as $N := M^3$. All matrix entries are sequentially stored in a long array (we perform FORTRAN77!), and as usual for the CSR format (see [16]) we employ two additional index arrays for accessing the load vector. We examine 3 variants of numberings: Linewise numbering but nevertheless indexed access (`SPARSE`), a simulated 'two level' numbering (`FEAT`) as typical for semi-adaptive FEM simulations without local adaptivity (which allows maximum differences in the numbering of $O(M^2)$), and finally a 'stochastic' numbering (`ADAP`) of the unknowns (allowing jumps in the numbering of two neighboured vertices of order $O(M^3)$) which is representative for fully adaptive approaches. Then, in all cases we define the MFLOP/s rates as before:

$$\frac{54 \times N}{CPUTIME \times 10^6}$$

Unfortunately, we have missed this direct inclusion of *sparse* MV application in 2D (now we cannot change anymore!) such that we have to simulate the analogous behaviour for the *sparse* (CSR) MV multiplication via the indexed DAXPY routines `DAXPYV` and `DAXPYI`. As in the 3D case, we allow for the index vector that jumps of order $O(M)$ (remember: $N := M^2$ in 2D!) or $O(M^2)$ can occur which compare well with the `FEAT`, resp., the `ADAP` results from the 3D case. These 2D tests correspond to our older 'Elch Tests' which have been described in [9].

5 The FEAST Indices

Before we explain more in detail all auxiliary indices and how to compute them, we present already the final result: **The actual version of the 'total'** FEAST INDEX ! They show our 'total' evaluation of most of the available processors and allow different specific rankings with respect to the explained implemenation techniques and applied data structures for standard conforming FEM. Most of the notation in the following tables and figures should be self-explanable; if not so, take a look at [7].

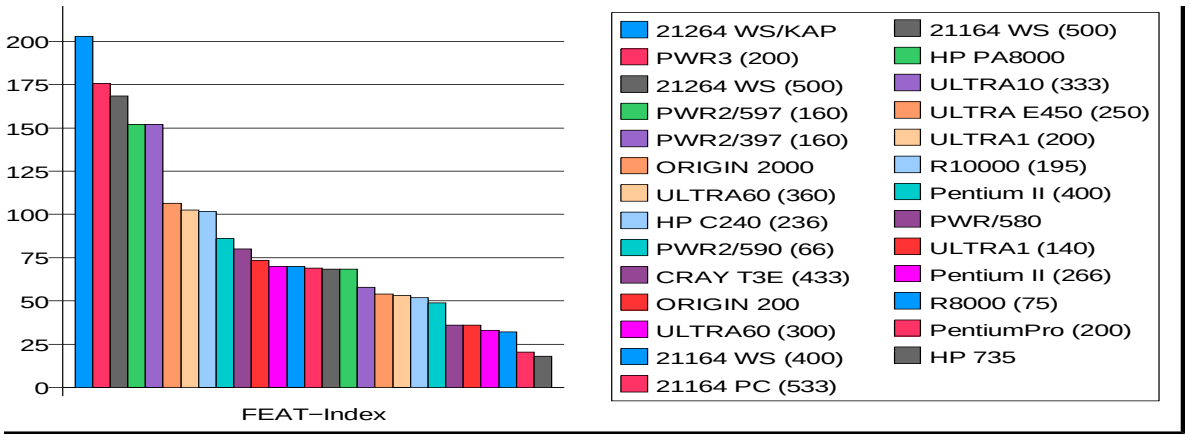


Figure 3: **The 'total'** FEAST INDEX**:** The graphical and table-based presentation of the specific 3D and 2D values is given in the Appendix: The performance in 2D is smaller than in 3D, due to the more compact matrices (see the Conclusions). Keep in mind that these are averaged MFLOP/s rates only which may significantly differ from the results for specific problem sizes!

The first term determines the kind of processor, followed by some additional denotations: PWR for IBM POWER2 or POWER3 processors, model 590 or whatever; 21264 or 21164 denotes the ALPHA chip in DEC's, with or without KAP preprocessor, being a workstation model (WS), a PC (under LINUX, but using the executable code from the workstation) or a PC under LINUX with the EGCS F77 compiler; ULTRA stands for SUN computers; ORIGIN, resp., R8000 and R10000 denote SGI's; PII and PPro denote Pentium II and Pentium Pro architecture; PPC/F50 denotes the PowerPC version F50 by IBM. In brackets, further details about the actual clock rate are given. More information about the precise definition of the tested computers can be obtained from [7], or look at our Homepage!

In the following Sections, we will discuss only some of the 'auxiliary' indices in detail, while a much more complete description of the results in 3D **and** in 2D is given in the Appendix: In most cases, the 2D results allow the same qualitative and even quantitative conclusions. Only in such cases that significant differences between 2D and 3D results occur, we explicitly state the corresponding results. Additionally, keep in mind that the indices, or better the ranking of computers, will change since new processors or different configurations (compiler, operating system, etc.) will be added. Therefore, check out the given Internet address! This Web-page contains also a full-color presentation of the results which may be helpful in better viewing the indices.

All calculations for the evaluation of the included MFLOP/s rates are performed for several levels l of (global) mesh refinement. We set for the number $N(l)$ of unknowns:

$$N(1) = 65^2\,, N(2) = 129^2\,, N(3) = 257^2\,, N(4) = 513^2\,, N(5) = 1025^2 \text{ (in 2D)}$$

$$N(1) = 17^3\,, N(2) = 33^3\,, N(3) = 65^3 \text{ (in 3D)}$$

Higher refinement levels in 2D are somewhat unrealistic. Furthermore, the storage cost are already far beyond typical cache sizes such that the effect of further increasing the problem size can be easily extrapolated. In 3D, one may state that level $l = 3$ with $N(3) = 65^3 = 274,625$ unknowns does not seem to correspond to a 'fine' mesh width, but one has to take into account that on this level 3 the storage of the stiffness matrix (bandwidth 27!) plus index arrays for the *sparse* techniques consumes already (almost) 128 MByte of RAM! This example shows that we discuss not only differences with respect to computing efficiency but also with regard to storage cost. Based on the MFLOP/s rates for the basic and composite Numerical Linear Algebra components from the previous Section, we can define the following (auxiliary) indices which are part of the shown FEAST INDICES.

5.1 The *sparse* FEAT and ADAP INDICES

These indices are measures for the computational efficiency with respect to classical *sparse* techniques, and they are based on the previously defined MFLOP/s rates for different numberings in the *sparse* (CSR) MV multiplication. As typical for all following indices, the rates from the different levels l are weighted by special factors $c(l)$. These are for the FEAT INDEX in 2D and in 3D:

$$\text{c(5)=60\%, c(4)=25\%, c(3)=10\%, c(2)=4\%, c(1)=1\%} \quad \text{resp.,} \quad \text{c(3)=80\%, c(2)=16\%, c(1)=4\%}$$

and for the ADAP INDEX:

$$\text{c(5)=40\%, c(4)=30\%, c(3)=15\%, c(2)=10\%, c(1)=5\%} \quad \text{resp.,} \quad \text{c(3)=75\%, c(2)=20\%, c(1)=5\%}$$

While the definition of both indices in 3D is straightforward, we have to do some modifications in the 2D case as explained before. To be precise, the resulting total FEAT INDEX in 2D is calculated as an average between the `DAXPYV` and `DAXPYI` values (with 'moderate' jumps of order $O(M)$), and in 3D between the `SPARSE` and `FEAT` values. Analogously, the ADAP INDEX in 2D is the average between both `DAXPYI` values

(with the two kinds of allowed jumps in numberings!) and in 3D between the `FEAT` and `ADAP` values. All these values are summed up with the special weighting factors accordingly to each mesh level.

The corresponding results of the FEAT and ADAP INDICES can be found in the Appendix: The computations demonstrate the dependence on the problem size and the kind of indexed access due the described numbering strategies: The resulting MFLOP/s rates are quite slow compared with the following results for performing the SPARSE BANDED BLAS-like MV multiplication with the **same (!!!)** matrix. Beside DEC's and IBM's top models, the SGI's and particularly the PENTIUM's lead to (relatively) good results. However, we compare 20 MFLOP/s as best values with less than 10 MFLOP/s for many other processors! In contrast, POWER2 models (IBM) and older DEC's (21164, CRAY T3E) deteriorate on the finest level 3 to about 10 MFLOP/s only! This shows impressively that some of our 'preferred' machines may have more or less problems with the described *sparse* techniques since the underlying cache architectures lead to massive cache misses and hence to performance losses! Further increasing of the number of unknowns can lead to even still slower performance results!

5.2 The DAXPY INDEX

For calculating the DAXPY INDEX, the `DAXPY` rates are averaged over all mesh levels. The corresponding weights for each mesh level are differently defined, in 2D as well as in 3D:

$$\text{c(5)=70\%, c(4)=20\%, c(3)=6\%, c(2)=3\%, c(1)=1\%} \quad \text{resp.,} \quad \text{c(3)=82\%, c(2)=15\%, c(1)=3\%}$$

5.3 The FEAST-MV-V and FEAST-MV-C INDICES

These MFLOP/s rates are based on our optimized SPARSE BANDED BLAS software [14] and have to be compared with the previous *sparse* FEAT and ADAP INDICES. FEAST-MV-V denotes the results with variable matrix entries while FEAST-MV-C measures the even higher MFLOP/s rates for the (special) case of constant band entries (see the explanations in the previous Section). Like the DAXPY INDEX, they are also part of the subsequent FEAST-V and FEAST-C INDICES (see later). The corresponding weighting factors $c(i)$ are identical as for the described DAXPY INDEX!

The following Figure 4 shows the corresponding 3D results for the different matrix-vector multiplications (MV-C, MV-V, FEAT, ADAP) on the finest mesh level 3: Performance differences of almost a factor of 50 get visible for certain computers as the IBM PWR2's! 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, particularly compared with PENTIUM's. The complete presentation of these indices is part of the Appendix.

5.4 The FEAST-MGLGS-V and FEAST-MGLGS-C INDICES

These MFLOP/s rates estimate the resulting performance of one complete multigrid sweep with 1 pre- and 1 postsmoothing step. The examined variants in `ptest.f` are FEAST-MGJAC-V and FEAST-MGJAC-C for Jacobi smoothing, FEAST-MGTRI-V and FEAST-MGTRI-C for tridiagonal smoothing and our preferred combination FEAST-MGLGS-V and FEAST-MGLGS-C with the described 'line Gauß-Seidel' (`TRIGS`) smoother: They are the most essential part of the subsequent FEAST-V and FEAST-C INDICES. The corresponding weighting factors $c(i)$ are again identical as for both previous indices.

The subsequent Figure 5 shows the corresponding results for complete SPARSE BANDED BLAS multigrid algorithms which are optimized with respect to numerical efficiency and robustness (see [1]). They are recently our 'best' multigrid work horses on such tensorproduct meshes. In the case of 'constant' bands in the matrices (see the results in the Appendix), the results further improve significantly!

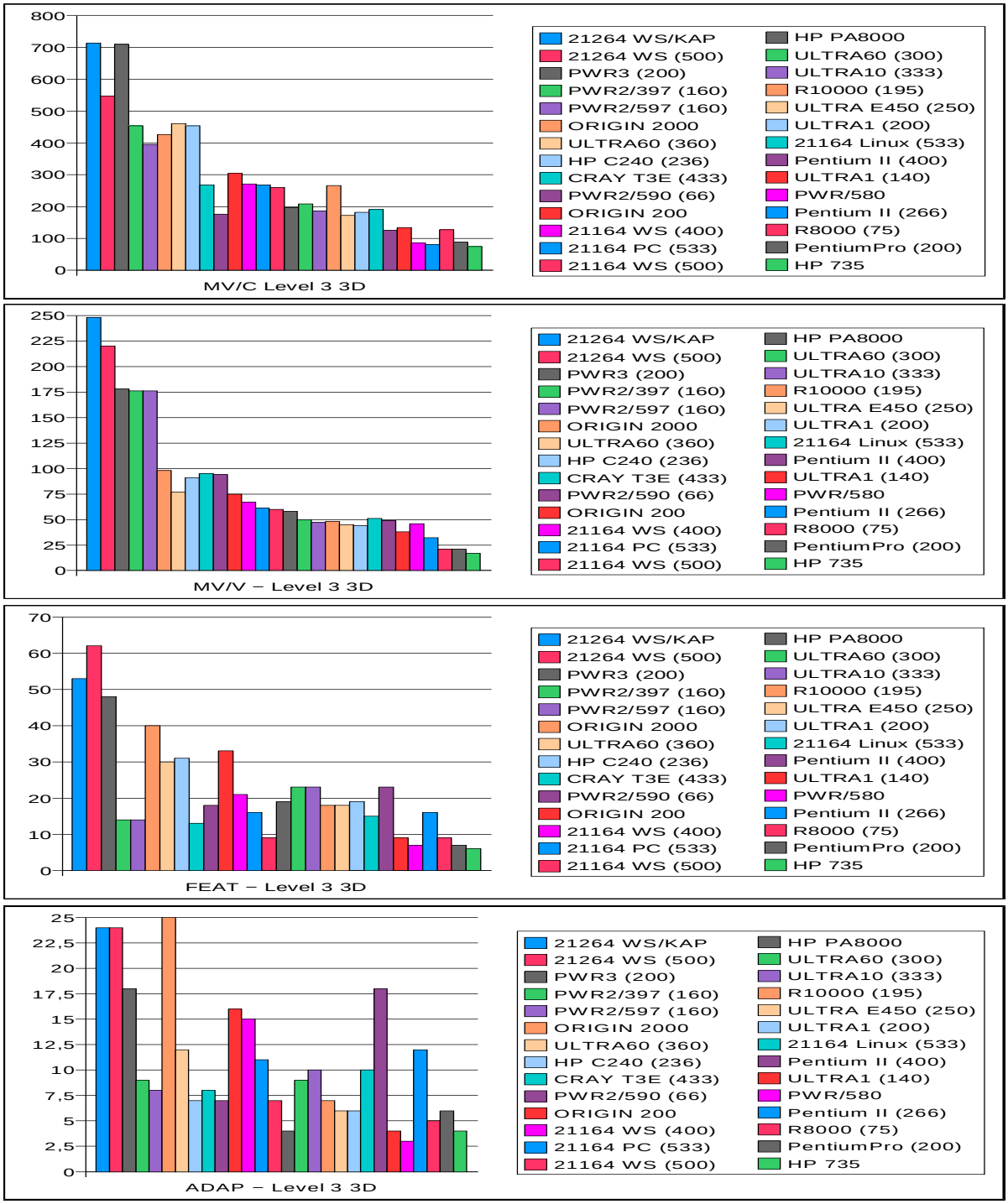


Figure 4: The figures show the MFLOP/s rates for the discussed MV multiplications (MV-C, MV-V, FEAT, ADAP) on mesh level 3 in 3D! There are huge differences possible of up to a factor of 50 if **different** MV techniques are applied to the **same** matrix!

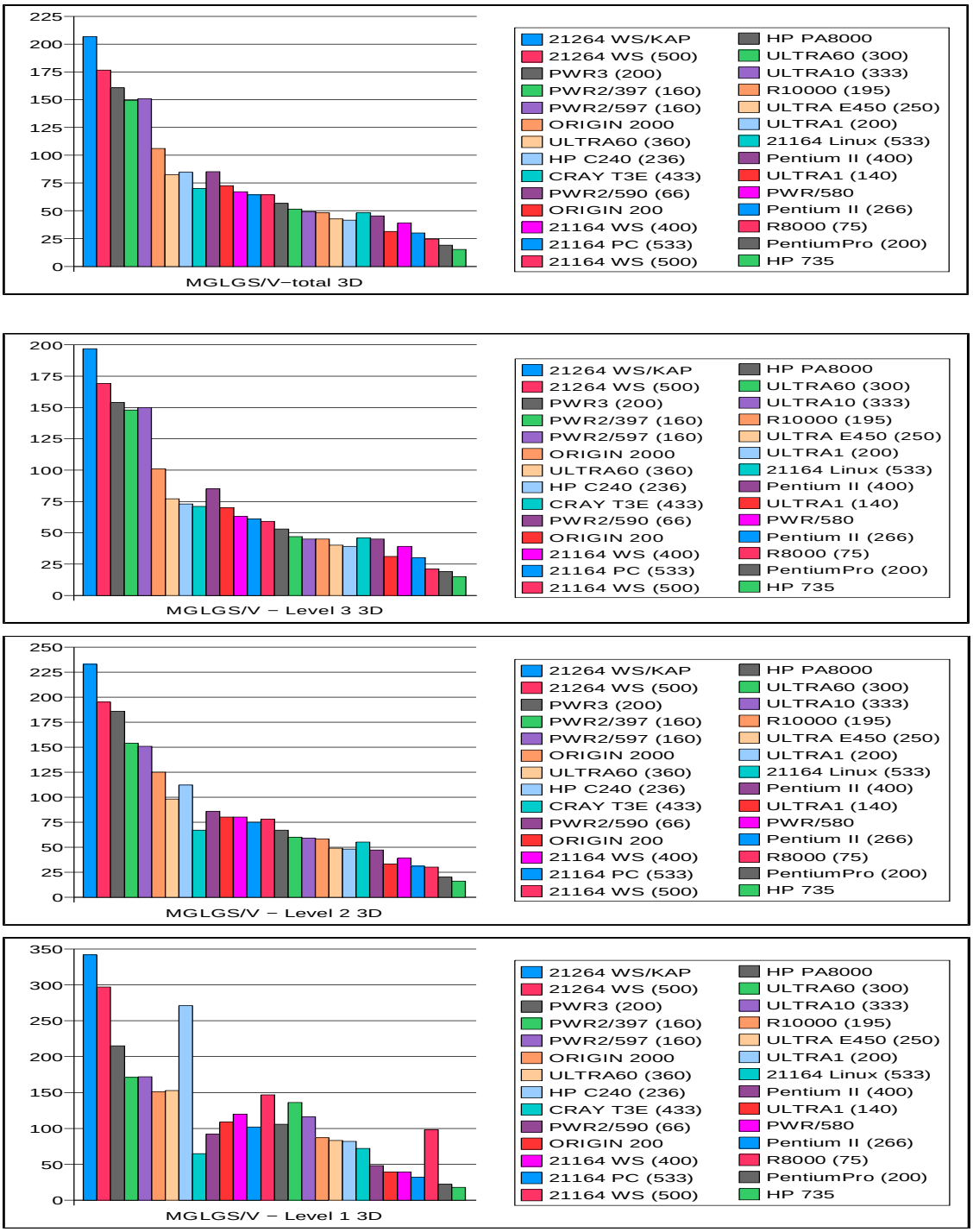


Figure 5: FEAST-MGLGS-V INDEX **in 3D:** The first row shows the FEAST-MGLGS-V INDEX, followed by the specific results for the `M-TRIGS` routines on the different mesh levels. The computations show that not only MV multiplications can be efficiently applied, but also complete multigrid algorithms with a very robust smoother inside which works very efficient for regular as well as very anisotropic meshes (see [1]). Again, the results are measured MFLOP/s rates.

5.5 The 'auxiliary' FEAST-V and FEAST-C INDICES

These indices measure the resulting performance of our SPARSE BANDED BLAS tools if they are purely applied on meshes which consist of macros (≈ quadrilaterals/hexaeders) only and which all are discretized via the described generalized tensorproduct meshes. For recent examples and actual test implementations of the FEAST package, see [2] and [4]. Both indices are averaged values, that means 50% FEAST-MGLGS-V , 25% FEAST-MGTRI-V , 10% FEAST-MGJAC-V , 10% FEAST-MV-V and 5% DAXPY , resp., the analogous values for the 'C'-versions. Again, each mesh level is differently weighted via:

$$c(5)=70\%\,, c(4)=20\%\,, c(3)=6\%\,, c(2)=3\%\,, c(1)=1\% \quad \text{resp.,} \quad c(3)=82\%\,, c(2)=15\%\,, c(1)=3\%$$

Figure 6 shows the corresponding 3D results which are 'averaged estimates' for the processors if purely applied to the highly structured patches on the low level parts of FEAST. In the Appendix, we give a direct comparison between the corresponding 2D and 3D results which demonstrates the potentially higher MFLOP/s rates in the 3D applications due to the 'wider' matrices. Since the typical bandwidth is 27 instead of 9 in 2D only, the applied cache strategies seem to work better! Moreover, they also give an impression of the difference in the resulting computing performance if such tensorproduct strategies - FEAST-C and FEAST-V - can be applied in comparison to the standard *sparse* techniques (see the FEAT and ADAP INDICES). However, these are only averaged MFLOP/s rates while the actual difference with respect to a special problem size may be even much greater as the previous examples have shown!

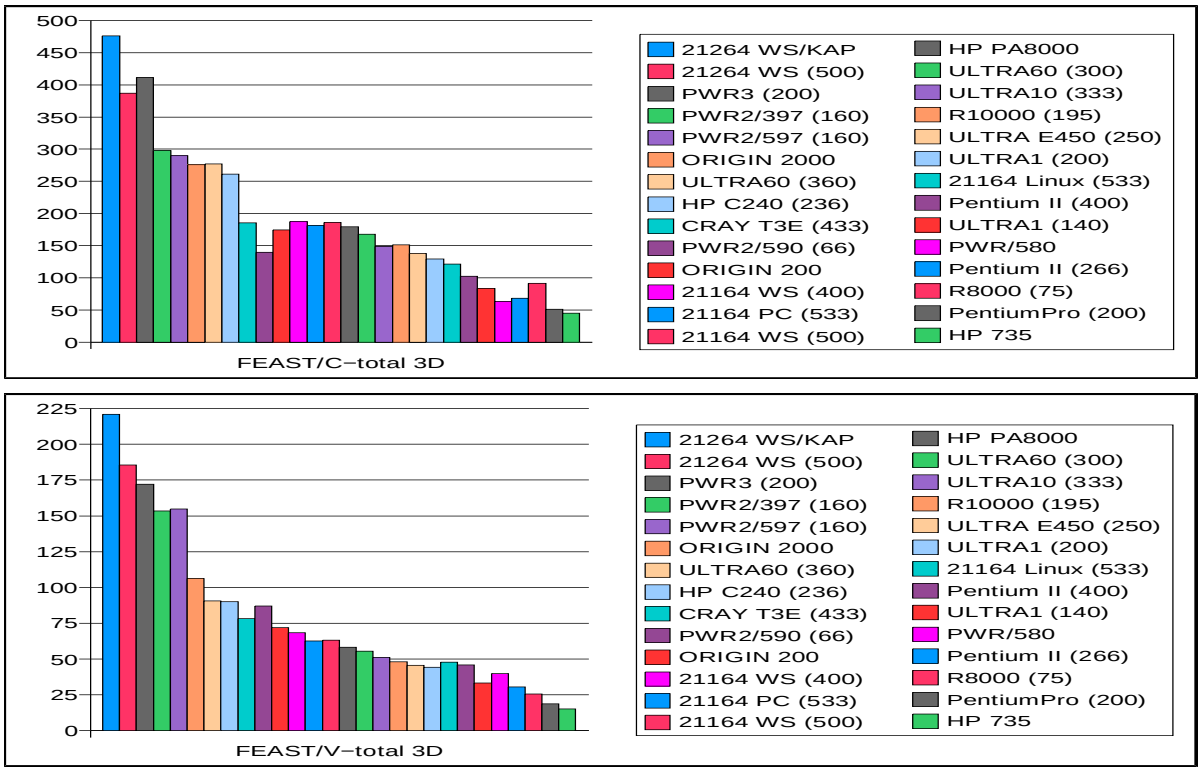


Figure 6: FEAST-V **and** FEAST-C INDICES **in 3D**

5.6 The 'total' FEAST INDICES

The final FEAST INDICES can be collected which are defined as follows. The corresponding 'total' FEAST INDEX - as average of both 2D and 3D FEAST INDICES - had been shown in the beginning of this Section.

$$\text{FEAST} := 60\%\ \text{FEAST-V} + 20\%\ \text{FEAST-C} + 15\%\ \text{FEAT} + 5\%\ \text{ADAP}$$

6 (Excerpts from) The FeatFlow Benchmark

Before we come to the conclusions from the previous Feast Indices , we still have to discuss the following major question in this context of benchmarking:

> *How realistic is our evaluation of the processors via the* Feast Indices*, if we compare with results from realistic applications, particularly computed with 'production codes'?*

The FeatFlow Benchmark is such set of test calculations which are very similar to the mentioned CFD Benchmark 'flow around a cylinder' [8]. The interesting aspects are the total CPU cost for each computer type, and how they are distributed to the separate tasks in a CFD code, i.e., mesh generation, assembling of matrices and right hand sides, solving linear subproblems or postprocessing steps.

The precise configurations are described at our Homepage; here we only provide the reader with the most important details. The complete description with all mathematical details of discretization and solver can be found in [10]: We apply a coupled multigrid approach with the so-called *Vanka* smoother in a direct steady solver (CC3D), and the *discrete projection scheme* as nonstationary scheme (PP2D). All calculations are performed with the nonconforming *rotated multilinear* finite elements. Additionally, we apply stabilization techniques of *upwind* (UPW) or *streamline diffusion* (SD) type for the convective term, and we use SOR as smoother in the multigrid solver for the scalar subproblems if applying the *projection* scheme. The following abbreviations for the elapsed time in seconds are used, with **total** denoting the complete elapsed CPU time:

PRO $\sim$ mesh generation, initialization phase and postprocessing
LC $\sim$ modifications of matrices (in *sparse* storage technique) or right hand sides
MAT $\sim$ CPU time for matrix generations
C-MG $\sim$ elapsed time for multigrid in the CC3D test with the *Vanka* smoother
U-MG $\sim$ CPU times for solving velocity, resp., pressure subproblems via multigrid
P-MG if the *discrete projection scheme* PP2D is applied

Exemplarily, we show the results of some selected computers for the direct stationary approach CC3D and for the nonstationary scheme PP2D. All tests require about 128 Mbyte of RAM. The complete results can be found at our Homepage in the Internet. All 'FeatFlow users' are invited to participate at this special computer benchmark which is part of the freely-available FeatFlow software such that the run of this set of tests can be easily performed by everybody.

Type	total	PRO	LC	MAT	C-MG
IBM RS6000/PWR3 (200 MHz)	**603**	5	17	113	468
IBM RS6000/597 (160 MHz)	**755**	7	21	182	545
HP C240 (240 MHz)	**974**	7	36	148	783
DEC WS 21164 (500 MHz)	**1029**	5	59	159	804
PC PENTIUM II (400 MHz)	**1307**	8	46	307	947
SUN ULTRA 60 (300 MHz)	**1368**	6	41	231	1089
IBM RS6000/590 (66 MHz)	**1510**	16	43	407	1043
SUN ULTRA 450 (250 MHz)	**1623**	8	50	276	1289
SGI R10000 (195 MHz)	**1936**	11	75	215	1634

Table 6: (Selected) results for test case CC3D–SD

This *Vanka* smoother inside of the coupled multigrid solver CC3D (for velocity and pressure simultanously) is a memory-access intensive process which explains why the tested IBM's, HP's and DEC's are only slightly faster than the Pentium II PC! The rates are quite in good agreement with the previous *sparse* Feat and Adap Indices and show that for such kind of 'index-oriented' techniques PENTIUM II processors lead to very satisfying results, particularly regarding the cost/performance relation!

Type	total	PRO	LC	MAT	U-MG	P-MG
IBM RS6000/PWR3 (200 Mz)	**789**	12	49	71	388	269
IBM RS6000/597 (160 MHz)	**954**	13	50	129	428	334
DEC WS 21164 (500 MHz)	**1342**	7	107	95	639	539
HP C240 (240 MHz)	**1344**	14	96	130	640	464
SUN ULTRA 60 (300 MHz)	**1597**	16	110	135	783	552
PC PENTIUM II (400 MHz)	**1861**	14	129	140	920	658
IBM RS6000/590 (66 MHz)	**1867**	29	99	298	831	609
SUN ULTRA 450 (250 MHz)	**2087**	20	134	163	1051	717
SGI R10000 (195 MHz)	**2701**	19	213	188	1276	1000

Table 7: (Selected) results for test case PP2D–UPW

In contrast, PP2D is based on operator-splitting ideas which require the numerous solution of scalar PDE problems for the velocity, resp., the pressure components. Since these kinds of tasks require much more arithmetic operations in comparison to memory accesses, the processors of type IBM, HP, DEC and SUN are significantly faster than the Pentium II processor. However, since this code is implemented without the optimized Sparse Banded Blas MV tools, we are still far away from the potentially available performance rates from the previous Feast Indices. These studies indicate how the future generation of CFD solvers has to look like, such that very sophisticated and powerful numerical methods can be combined with the available high performance rates on recent processors. For a mathematical and algorithmic discussion of such concepts, look at [10] and [13]! Having exclusively such simulation results with *sparse* techniques, there would not be any need for other processors than INTEL's!

7 Conclusions from the Feast Indices

2D vs. 3D indices: The results from the 2D tests compared with the analogous measurements in 3D are very similar: The IBM processors and the new DEC 21264 are the best, with respect to the total ranking and in most cases with regard to the specific auxiliary indices, too! However, the corresponding MFLOP/s rates are in 3D somewhat higher, at least for processors with 'large' (level2) caches. The explanation might be that the applied matrices are less 'sparse', with 27 instead of 9 bands, such that cache-blocking strategies can be better optimized. On the other hand, computers 'without' large second level (L2) cache, as for instance the IBM PWR2's and probably pure vector computers, do not improve!

As a consequence, these IBM workstations based on the PWR2 and the P2SC processors might be favourable for codes which have not yet incorporated such cache-based optimization strategies. On the other hand, also for certain nonconforming FEM approximations or cell-centered discretizations which generally lead to even more *sparse* matrices (with 5- or 7–point stencils only!), these kinds of processors may be the 'winners'. In contrast, for higher order discretizations as bi- or triquadratic (FEM) discretizations, the cache-oriented processors will even more improve!

'Optimality' of the indices: The following results have been directly provided by the vendors such that we assume that 'optimal' compiler options and hardware components have been utilized:

- DEC 21264 and DEC 21164 workstation with 400 MHz
- IBM POWER3 and P2SC, IBM F50 (PowerPC)
- SGI ORIGIN 2000 with 250 MHz
- SUN ULTRA 60 with 300 MHz and 360 MHz
- HP V2250

All other processors have been tested by us. So, everybody is invited for improving the indices of these models (and also the 'vendor-tested' versions). In particular, the use of other FORTRAN compilers (F90, EGCS, GNU-FORTRAN, etc.) or different programming languages appears to be interesting and, in fact, such tests have been already performed by us (see [7]). For instance, compare in 3D the MFLOP/s rates of DEC-PC and DEC-LINUX which are identical computers. While on the DEC-PC variant, the executable code has been compiled on a workstation environment with the original DEC compiler and then copied to the PC-target computer, in the case of the DEC-LINUX configuration the executable code has been directly generated via the EGCS F77 compiler and then executed under LINUX. The differences are very significant!

Further improvements do not have to be to restricted to examine the hardware components, it will be also interesting to modify the test software `ptest.f`. So far, we have implemented in the SPARSE BANDED BLAS several versions with different blocking strategies, and we have tested some specified blocking parameters: However, some modifications of these routines are allowed and even desired, as long as they are applicable to such special 9-point, resp., 27-point matrices which arise from conforming FEM discretizations on generalized tensorproduct meshes! Additionally, the specific adaption with respect to certain machine-dependent values as the precise cache sizes or number of registers has not been applied up to now. Hence, there is still a large potential to gain further improvements for such low level Numerical Linear Algebra routines. Look also at the 'data locality' research project of U. Rüde et al., which is well described at `http://wwwbode.informatik.tu-muenchen.de/Par/arch/cache/index.html` and which contains a lot of further literature and activities in this field.

FEAST INDICES **vs. application- and problem-specific results:** The shown FEAST INDICES - total or auxiliary - are only **averaged** values. For many applications, it might appear to be much more realistic if the corresponding results for specific tasks and particularly for different problem sizes are explicitely considered. Then, the observed differences between the processors may be much larger than the FEAST INDICES do indicate! For instance, the DEC 21164 processor shows excellent performance for 'small' problems, with dramatic losses of performance if the problem size increases. So, it might be much more favourable to work on 'many' small patches (with 500 MFLOP/s) instead of 'few' larger patches, with 50 MFLOP/s only. Therefore, watch out for your own specific problem configuration! The previous FEATFLOW Benchmark has shown that the corresponding results are significantly depending on the numerical method and solution algorithm, and that the differences in the resulting performance may be much weaker. Nevertheless, keep in mind that those FEATFLOW results have been obtained via classical *sparse* techniques only, in contrast to the architecture-optimized results of the FEAST INDICES! Additionally, if one performes similar tests with recent object-oriented languages, as for instance C++, you should compare with those optimized FEAST results which really exploit the FORTRAN facilities. So, there is a good chance for a true comparison of FORTRAN77 with C++ or similar!

IBM POWER2 and POWER3: While the tested POWER2 and especially P2SC variants show an excellent behaviour for the *sparse banded* techniques, they demonstrate dramatic losses of performance for the *sparse* applications! As explained before, they seem to be quite robust with respect to 'cache-optimized' problems, while they may 'loose' against processors with larger cache-architectures if the matrices get less *sparse*. Therefore, they belong to our favourites for applications in FEAST - and also FEATFLOW - which are mainly based on the described generalized tensorproduct meshes on the lowest level.

In contrast, the new POWER3 processor appears to be a very robust and efficient 'work horse' which can be applied for classical *sparse* as well as for our optimized highly structured Numerical Linear Algebra components. This computer is beside the DEC 21264 the clear winner at the moment!

IBM PowerPC F50: This 'low cost' model can be viewed as rival of PENTIUM II PC's, with respect to price and performance. Therefore, look at our ratings of the PENTIUM world!

DEC 21164 and 21264, CRAY T3E: The older 21164 processor shows huge performance problems

as soon as the problem size increases: Up to now, none of our cache-optimization strategies seems to work fine, in contrast to (almost) all other platforms. Similar problems are valid for the CRAY T3E (Stuttgart) which is based on a variant of the 21164 processor. It might be necessary that further optimizations are performed directly by CRAY!

On the other hand, the new 21264 processor improves in a significant way: Having the same clock rate of 500 MHz, the speed-up is almost a factor of 3! Using the KAP preprocessor, the performance can be further increased, however it is not clear to us if this additional speed-up can be achieved in more complex numerical simulations, as for instance in FEATFLOW. Nevertheless, with or without this KAP preprocessor, the DEC 21264 chip is the clear winner beside the IBM POWER3 processor!

HP, SGI and SUN: The performance of the 'top models' of these vendors - and probably the prices, too - is very similar. The processors seem to be quite robust with respect to *sparse* MV applications while they cannot achieve the same high MFLOP/s rates for highly structured data as IBM and DEC. On the other hand, it might be expected that the prices are correspondingly cheaper, too. Therefore, these models are typical 'mid-range work horses' for most numerical simulations, especially during code development and code testing. However, it is not clear if they will provide the computing power which seems to be necessary for many 'real life' problems. At the moment, we are not sure about the 'real' quality of the HP V2250; we can only hope that the corresponding 3D test will be performed to guarantee a better rating of this model!

INTEL PENTIUM II: One of our problems with this processor is that we could not get into contact with INTEL to let them perform their own optimizations, maybe with special compiler options! Additionally, we could only apply the (freely available) EGCS F77 compiler. At the moment, this processor type seems to be quite 'insensible', regarding cache optimization strategies as well as with respect to performance degradation through *sparse* MV techniques. Based on the cheapest price, this hardware selection is the clear winner if highly unstructured data and *sparse* MV techniques are applied, for instance in fully adaptive FEM simulations. However, this is more due to the huge losses of the other processors than based on the own high performance: We compare 20 MFLOP/s with less than 10 MFLOP/s's! On the other hand, we have not figured out so far 'good' optimization strategies for highly structured data such that our measured performance values are often slower by a factor of 5 and even more! Nevertheless, we claim that for most available software tools which are based on *sparse* techniques (or which do not use FORTRAN or C!), the choice of PENTIUM II processors may be preferable.

Actually, the specific choice of optimal (?) hardware can be determined via following 'thumb rules':

- For fully unstructured data and corresponding *sparse* MV techniques, all hardware platforms show slow performance only. The implementation of corresponding tools is quite easy and straightforward, especially if available software tools as for instance SPARSKIT [16], SPARSEBLAS [15], NISTBLAS [17] or similar are employed. So, the time and work for the 'code development' may be quite fast, but it seems to be impossible to obtain high performance rates since the resulting efficiency is mainly due to the cost of memory access and less due to the possible performance of the processors. Therefore, we recommend the use of PENTIUM PC's since they are by far the cheapest ones! Additionally, the choice of the programming language might appear to be quite unimportant since no special facilities can be exploited, not even by FORTRAN! Nevertheless, our experience is that even for such applications FORTRAN77 tools can be significantly faster than C++ codes!

- On the other hand, if one spends more time and work in software concepts and correspondingly in special numerical approaches which are able to exploit 'caching in' and 'pipelining', the use of modern workstation processors and corresponding optimized FORTRAN compilers is advisable! At the moment, only these hardware/software combinations seem to be able to really exploit a higher percentage of today's high performance rates. However, as the FEAST project shows, the design

and development of such simulation tools can be much harder, but the final gain in CPU time and hence the gain of validity for the simulation data will be great!

Based on the actual FEAST INDICES, we clearly prefer both IBM's and DEC's top models. The difference in single processor performance appears to be a factor of 10 in the maximum, such that one might propose to take 10 processors of a slower type to gain similar performance. However, keep in mind that parallelization is always a hard job, with regard to numerical design, implementation tasks and also stability of the hardware! If one really would be successful to apply a complete numerical simulation with several hundreds of MFLOP/s on one single processor with 4 GByte RAM, it will be very hard to beat this performance on a parallel system, especially if very 'strong' numerical components are used which are often specifically adapted for sequential runs!

- The performance of future processors will change, but looking at the 'historical' development it seems that certain processor characteristics remain preserved: The IBM processors have been and still are very efficient for highly structured data, while SUN, SGI or HP are the typical 'robust work horses' for all kind of numerical simulations. So, we believe that the specific characteristics of the different processor families will remain valid in some sense for the next future!

Therefore, we end with the following final remarks:

1. *Do not base your rating of computers exclusively on the clock rates or other technical details!*
2. *Make your own tests or look at performance ratings which are representative for your specific applications and needs!*
3. *Always keep in mind that there are numerous hidden traps to loose computing performance. In fact, it is and it will be even more and more difficult to achieve a significant percentage of the growing peak rates on modern processors!*

8 Acknowledgements

The recent state of the FEAST INDICES is due to the 'successful' work of many people. Let me mention the FEAST group and especially Alex Runge who is the major person to evaluate and particularly to 'prepare' the specific results in text-based and graphical ways.

Additionally, we were very pleased about the 'spontaneous' participation of the processor vendors and the help of many other persons: Markus Zahn from 'Rechenzentrum der Universität Augsburg', Matthias Claus from 'Rechenzentrum der TU Chemnitz', Arnd Meyer from 'Universität Chemnitz', Uli Rüde from 'Universität Augsburg', Brigitte Möllenhoff from 'Rechenzentrum der Universität Heidelberg', Rainer Wehrse from 'Universität Heidelberg', Juan Porta, Holger Holthoff and Christoph Pospiech from IBM, Werner Höhn from HP, Ralf Lange from SGI, Ulrich Gräf from SUN, Joseph Pareti from DEC/COMPAQ and many more. In advance, let us already thank all 'future test persons'!

Without their help, the more or less complete data bank with respect to today available computer systems would not have been possible. On the other hand, also the next generation of processors, computer memory and compilers has to be continuously tested. If therefore sufficiently many 'test persons' actively participate in this processor benchmarking, there is a good chance to create a quite simple but nevertheless realistic rating of modern hardware platforms, particularly in addition to the well-known LINPACK or SPEC benchmarks. So, please feel free to contribute 'new' indices to this new data bank which together with the required software can be found under the FEATFLOW Homepage:

`http://www.iwr.uni-heidelberg.de/~featflow`

References (see also http://gaia.iwr.uni-heidelberg.de/~ture)

[1] Altieri, M.: *Robuste und effiziente Mehrgitter-Verfahren auf verallgemeinerten Tensorprodukt-Gittern*, Diploma Thesis, to be published

[2] Becker, Chr.: FEAST - *The realization of Finite Element software for high-performance applications*, PhD Thesis, to be published

[3] Hellwagner, H., Rüde, U., Stals, L., Weiß, Chr.: *Data locality optimizations to improve the efficiency of multigrid methods*, Proc. 14th GAMM Seminar 'Concepts of Numerical Software', Kiel, January 1998, *NNFM*, Vieweg, 1998

[4] Kilian, S.: *Efficient parallel iterative solvers of* ScaRC*-type and their application to the incompressible Navier-Stokes equations*, PhD Thesis, to be published

[5] Kilian, S., Turek, S.: *An example for parallel* ScaRC *and its application to the incompressible Navier-Stokes equations*, Proc. ENUMATH-97, Heidelberg, October 1997, World Science Publ., 1998

[6] Rüde, U.: *Technological trends and their impact on the future of supercomputers*, to appear in: High Performance Scientific and Engineering Computing (H.-J. Bungartz, F. Durst, Chr. Zenger, eds.), LNCSE, Springer-Verlag, 1999

[7] Runge, A.: *Zur realistischen Leistung von Software-Komponenten und modernen Prozessoren zur numerischen Simulation von Partiellen Differentialgleichungen*, Diploma Thesis, to be published

[8] Schäfer, M., Turek, S.: *Benchmark computations of laminar flow around cylinder*, in E.H. Hirschel (editor), *Flow Simulation with High-Performance Computers II*, Volume 52 of *Notes on Numerical Fluid Mechanics*, pp. 547–566, Vieweg, 1996

[9] Turek, S.: *Konsequenzen eines numerischen 'Elch Tests' für Computersimulationen*, Technical Report SFB 359, University of Heidelberg, 46, 1998

[10] Turek, S.: *Efficient Solvers for Incompressible Flow Problems: An Algorithmic Approach in View of Computational Aspects*, LNCSE 6, Springer, 1999

[11] Turek, S. et al.: *Some basic concepts of* Feast, Proc. 14th GAMM Seminar 'Concepts of Numerical Software', Kiel, January 1998, *NNFM*, Vieweg, 1998

[12] Turek, S. et al.: *On the realistic performance of components in iterative solvers*, to appear in: High Performance Scientific and Engineering Computing (H.-J. Bungartz, F. Durst, Chr. Zenger, eds.), LNCSE, Springer, 1999

[13] Turek, S. et al.: *Trends in processor technology and their impact on Numerics for PDE's*, to appear (available via Internet)

[14] Turek, S. et al.: *Proposal for* `Sparse Banded BLAS` *techniques*, to appear (available via Internet)

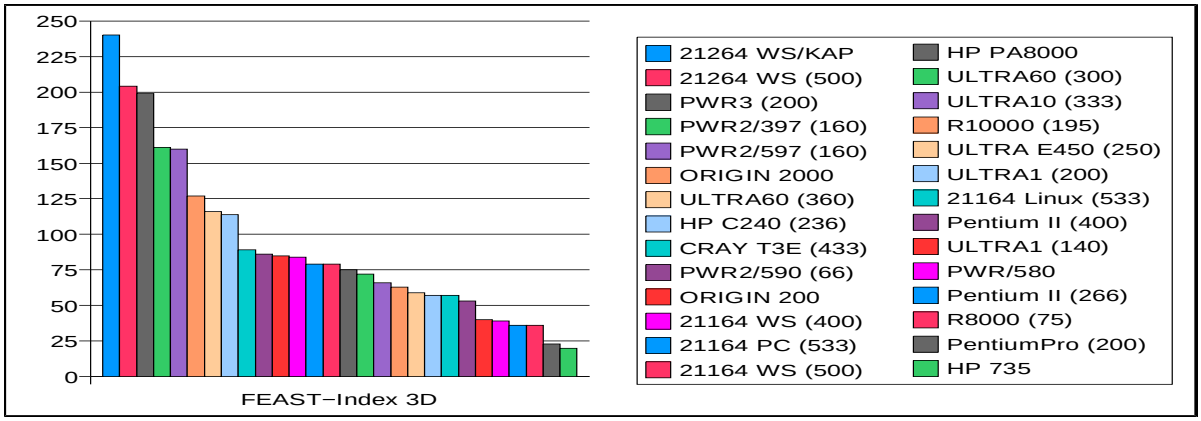
[15] BLAS Technical Forum, `http://www.netlib.org/cgi-bin/checkout/blast/blast.pl`

[16] SPARSKIT (by Y. Saad), `http://www.cs.umn.edu/Research/arpa/SPARSKIT/sparskit.html`

[17] The NIST Sparse BLAS, `http://math.nist.gov/spblas`

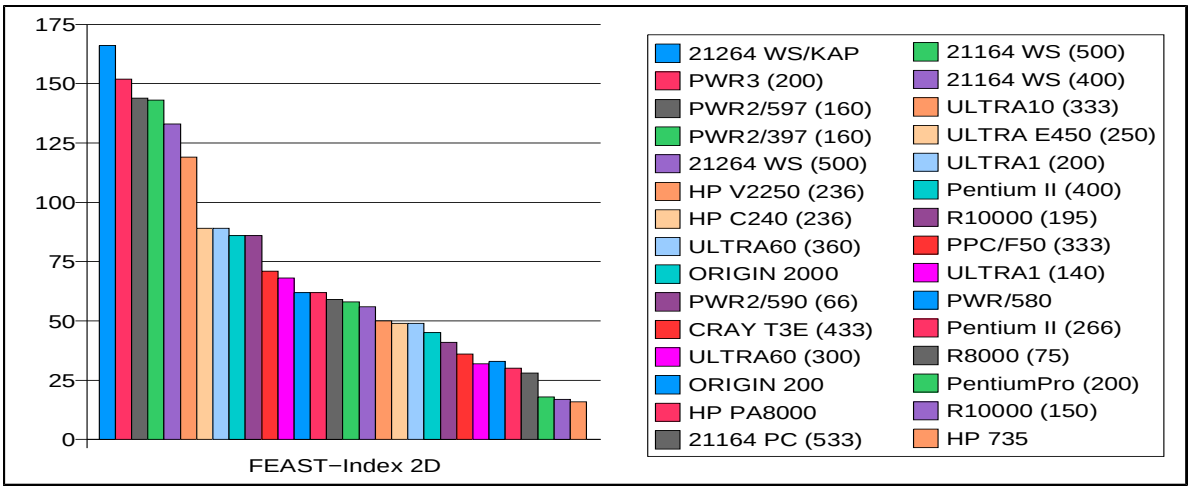
[18] The national technology roadmap for semiconductors, 1997 edition, `http://www.sematech.org/public/roadmap/index.htm`

Appendix: Some FEAST INDICES



Computer	FEAST	FEAST-V	FEAST-C	FEAT	ADAP
DEC-WS-21264-500MHz-KAP	240	220	476	66	47
DEC-WS-21264-500MHz	203	185	387	80	54
IBM-PWR3-200MHz	199	172	410	77	43
IBM-PWR2SC-397-160MHz	161	153	297	58	21
IBM-PWR2SC-597-160MHz	159	154	289	52	19
SGI-ORIGIN2000-250MHz	127	106	275	43	36
SUN-ULTRA60-360MHz	116	90	276	34	26
HP-C240-240MHz	113	90	261	39	26
CRAY-T3E(STUTTG.)	89	78	185	31	11
IBM-PWR2-590-66MHz	86	87	140	31	17
DEC-WS-21164-400MHz	84	68	187	32	22
SGI-ORIGIN200-180MHz	84	71	174	34	27
SUN-ULTRA60-300MHz	79	69	162	28	21
DEC-PC-21164-533MHz	79	62	181	31	17
DEC-WS-21164-500MHz	78	63	186	18	14
HP-PA8000	75	58	179	25	15
SUN-ULTRA2I-333MHz	65	51	148	28	20
SGI-10K-IP25-195MHz	62	47	150	18	13
SUN-ULTRA450-250MHz	59	45	138	21	14
DEC-PC-21164-533MHz-EGCS	57	48	121	21	14
SUN-ULTRA1-200MHz	56	44	129	23	15
PENTIUMII-400MHz	52	45	102	26	21
SUN-ULTRA1-140MHz	39	33	83	16	8
IBM-PWR-580	39	39	63	14	7
SGI-R8000-75MHz	35	25	92	11	8
PENTIUMII-266MHz	35	30	69	16	14
PENTIUM-PRO-200	23	18	51	10	7
HP-735	19	15	45	8	5

Table 8: **The 'total' FEAST INDEX in 3D and its (major) auxiliary indices:** The figure gives a graphical representation of the 'total' FEAST INDEX in 3D whereat the values correspond to averaged MFLOP/s rates! Additionally, the table shows the results of the (major) auxiliary indices which have been explained in Section 5; their graphical presentation and a direct comparison to the 2D values is given later. Keep in mind that these are averaged MFLOP/s rates only which may significantly differ from the results for specific problem sizes!



Computer	FEAST	FEAST-V	FEAST-C	FEAT	ADAP
DEC-WS-21264-500MHz-KAP	165	153	313	55	52
IBM-PWR3-200MHz	152	140	262	85	62
IBM-PWR2SC-597-160MHz	143	142	233	63	38
IBM-PWR2SC-397-160MHz	142	141	233	64	38
DEC-WS-21264-500MHz	132	121	248	53	49
HP-V2250	118	108	229	38	34
HP-C240-240MHz	89	82	166	32	30
SUN-ULTRA60-360MHz	89	78	171	41	38
IBM-PWR2-590-66MHz	85	88	128	36	25
SGI-ORIGIN2000-250MHz	84	70	175	33	32
CRAY-T3E(STUTTG.)	71	58	147	36	20
SUN-ULTRA60-300MHz	68	61	122	35	32
HP-PA8000	61	57	116	22	17
SGI-ORIGIN200-180MHz	61	54	123	24	22
DEC-PC-21164-533MHz	59	48	129	21	19
DEC-WS-21164-500MHz	58	49	122	20	20
DEC-WS-21164-400MHz	56	49	111	20	21
SUN-ULTRA2I-333MHz	50	42	98	23	24
SUN-ULTRA450-250MHz	49	42	98	19	20
SUN-ULTRA1-200MHz	48	43	93	19	17
PENTIUMII-400MHz	45	41	80	24	21
SGI-R10000-IP25-195MHz	41	35	87	14	14
PWRPC-F50-333MHz	36	31	74	12	10
IBM-PWR-580	32	31	53	15	9
SUN-ULTRA1-140MHz	32	29	60	13	10
PENTIUMII-266MHz	29	27	51	15	13
SGI-R8000-75MHz	28	23	56	13	15
PENTIUM-PRO-200	17	16	33	6	6
SGI-R10000-150MHz	16	12	41	5	7
HP-735	16	14	32	5	5

Table 9: **The 'total' FEAST INDEX in 2D and its (major) auxiliary indices:** The figure gives a graphical representation of the 'total' FEAST INDEX in 2D, while the table shows the results of the (major) auxiliary indices; a direct comparison with the 3D values is given later. One remarkable result is that the performance in 2D is smaller than in 3D, due to the more compact matrices (see the Conclusions).

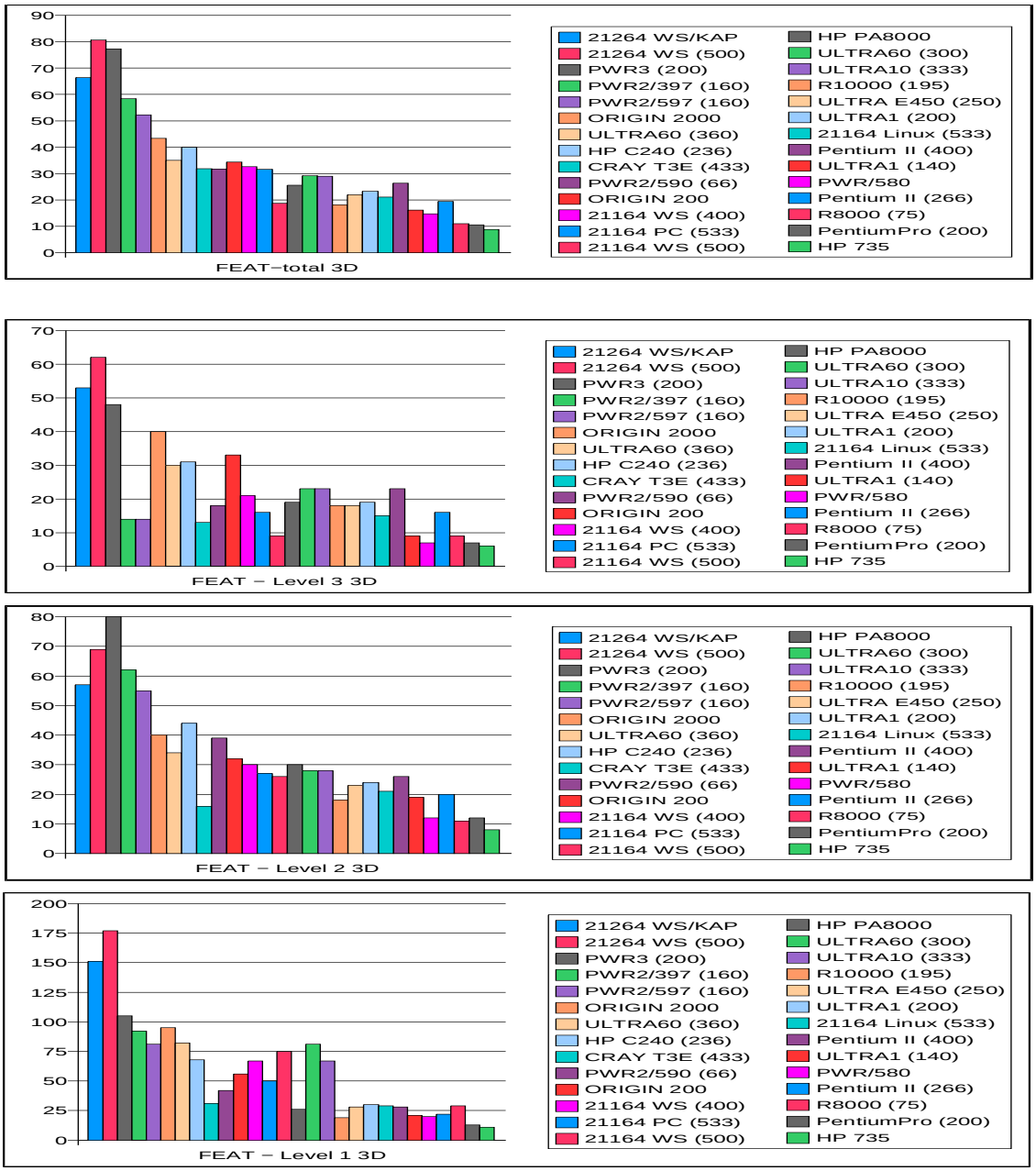


Figure 7: FEAT INDEX **in 3D:** The first row shows the FEAT INDEX, followed by the specific results for the `FEAT` MV routines (see Section 4) on different mesh levels. The computations demonstrate the dependence on the problem size and the kind of indexed access via the described numbering strategies (compare with the following ADAP INDEX!). The main results are that the resulting MFLOP/s rates are quite slow compared with the following results for performing the SPARSE BANDED BLAS-like MV multiplication with the **same (!!!)** matrix. Beside DEC's and IBM's top models, the SGI's and particularly the PENTIUM's lead to (relatively) good results. In contrast, POWER2 models (IBM) and older DEC's (21164, CRAY T3E) deteriorate on the finest level 3 to about 10 MFLOP/s only! The actual ordering of the processors is due to the total FEAST INDEX and shows impressively that some of our 'preferred' machines may have more or less problems with the described *sparse* techniques due to massive cache misses! Further increasing of the problem size can lead to even still slower performance results (compare `Level 3` vs. `Level 1`).

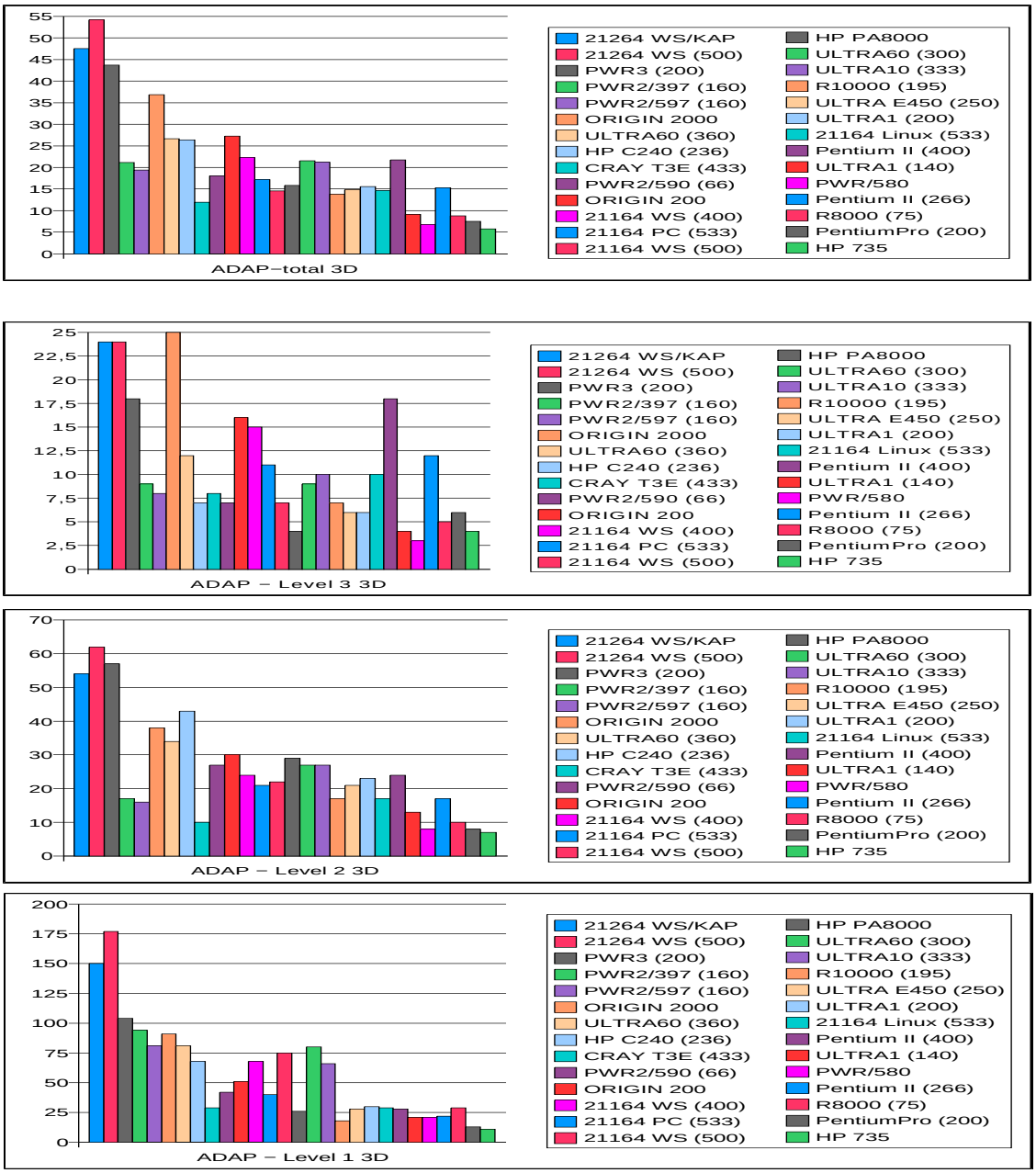


Figure 8: ADAP INDEX **in 3D:** The first row shows the ADAP INDEX, followed by the specific results for the `ADAP` MV routines (see Section 4) on different mesh levels. Due to the even higher jumps in the numbering of neighboured vertices, the MFLOP/s rates further decrease and are of maximum size 25 (!!!) on level 3! Even both DEC's and IBM's top models show about 20 MFLOP/s only, while again the SGI's and particularly the PENTIUM's lead to the (relatively) best results. However, we compare 20 MFLOP/s as best values with less than 10 MFLOP/s for many other processors! Additionally to the previous FEAT INDEX, the dependence on the mesh level, resp., the problem size, gets much more clear and varies up to a factor of 10 between `Level 1` and `Level 3`!

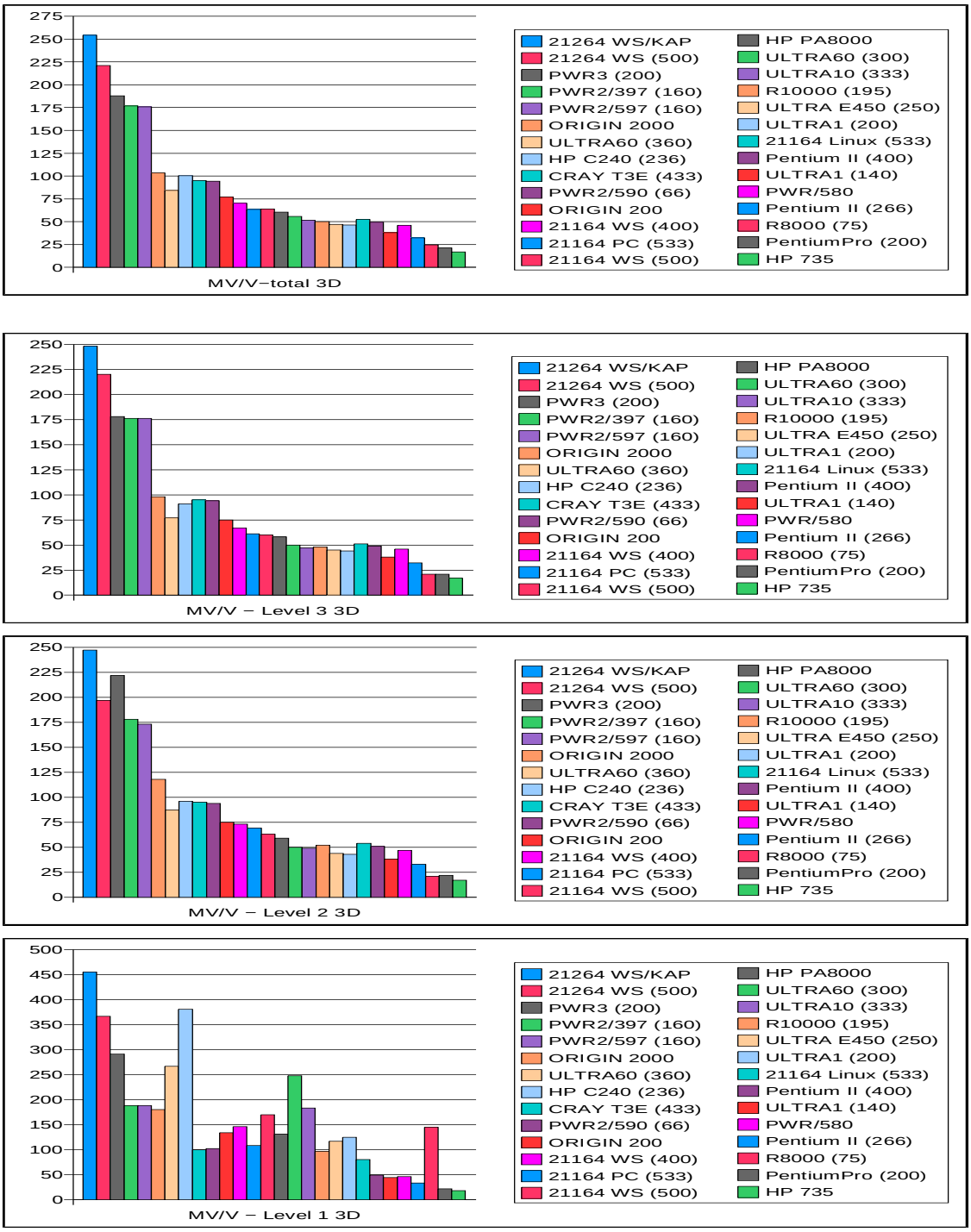


Figure 9: FEAST-MV-V INDEX **in 3D:** The first row shows the FEAST-MV-V INDEX, followed by the specific results for the `MV-V` routines on the different mesh levels. The computations demonstrate the much weaker dependence on the problem size, at least for some of the processors (it is obvious that for problems which fit completely into the cache, the rates are often better!). The main results are that the resulting MFLOP/s rates are essentially improved through the applied 'caching in' and 'pipelining' strategies, especially if compared with the previous *sparse* indices. Nevertheless, we expect in future even higher improvements if more machine-specific optimizations for this task are applied!

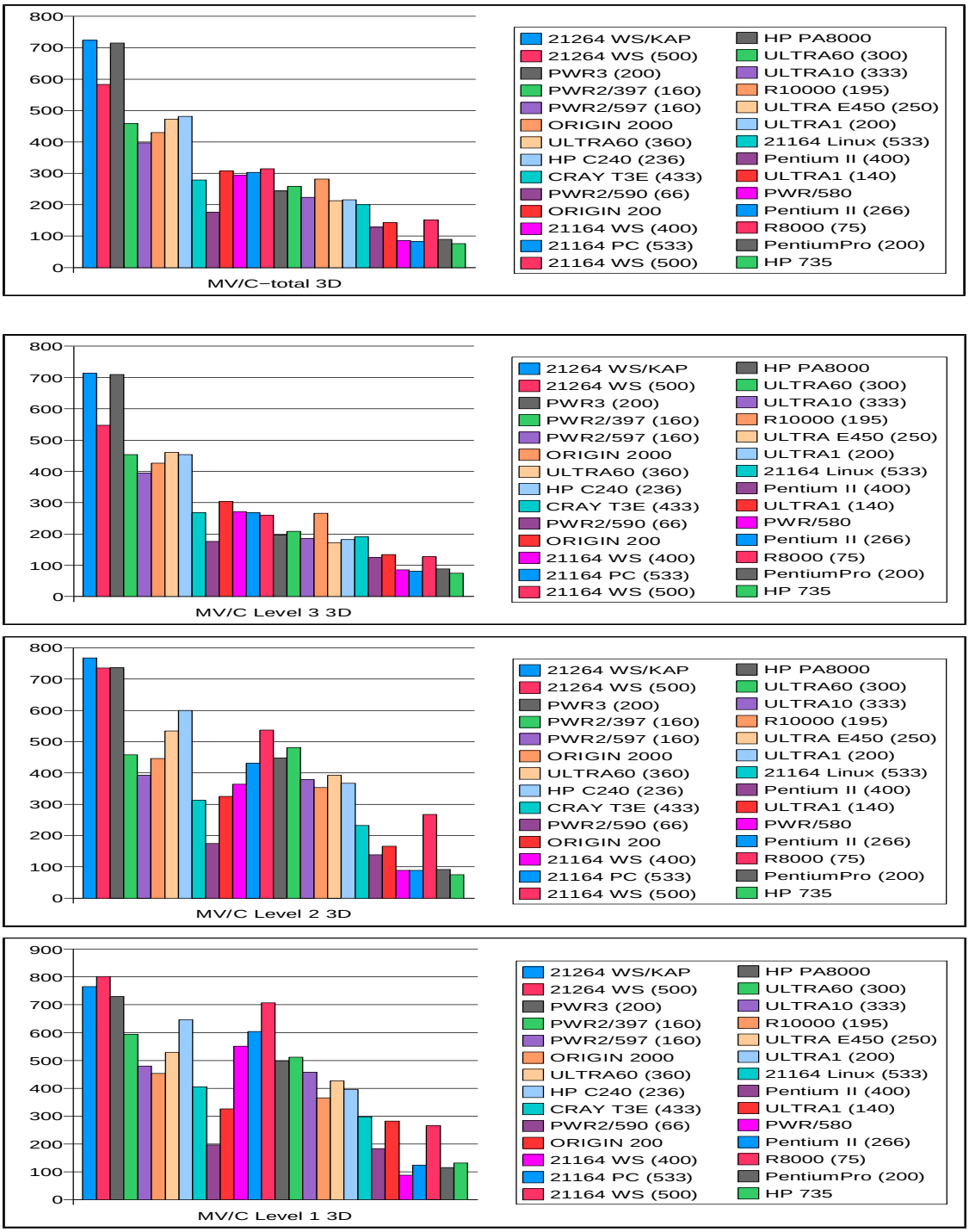


Figure 10: FEAST-MV-C INDEX **in 3D:** The first row shows the FEAST-MV-C INDEX, followed by the specific results for the `MV-C` routines on the different mesh levels. These studies show the importance of the main strategy in FEAST : 'Detect locally structured parts and exploit the correspondingly regular data'! More than 700 MFLOP/s are realistic which have to be compared with the previous *sparse* indices!

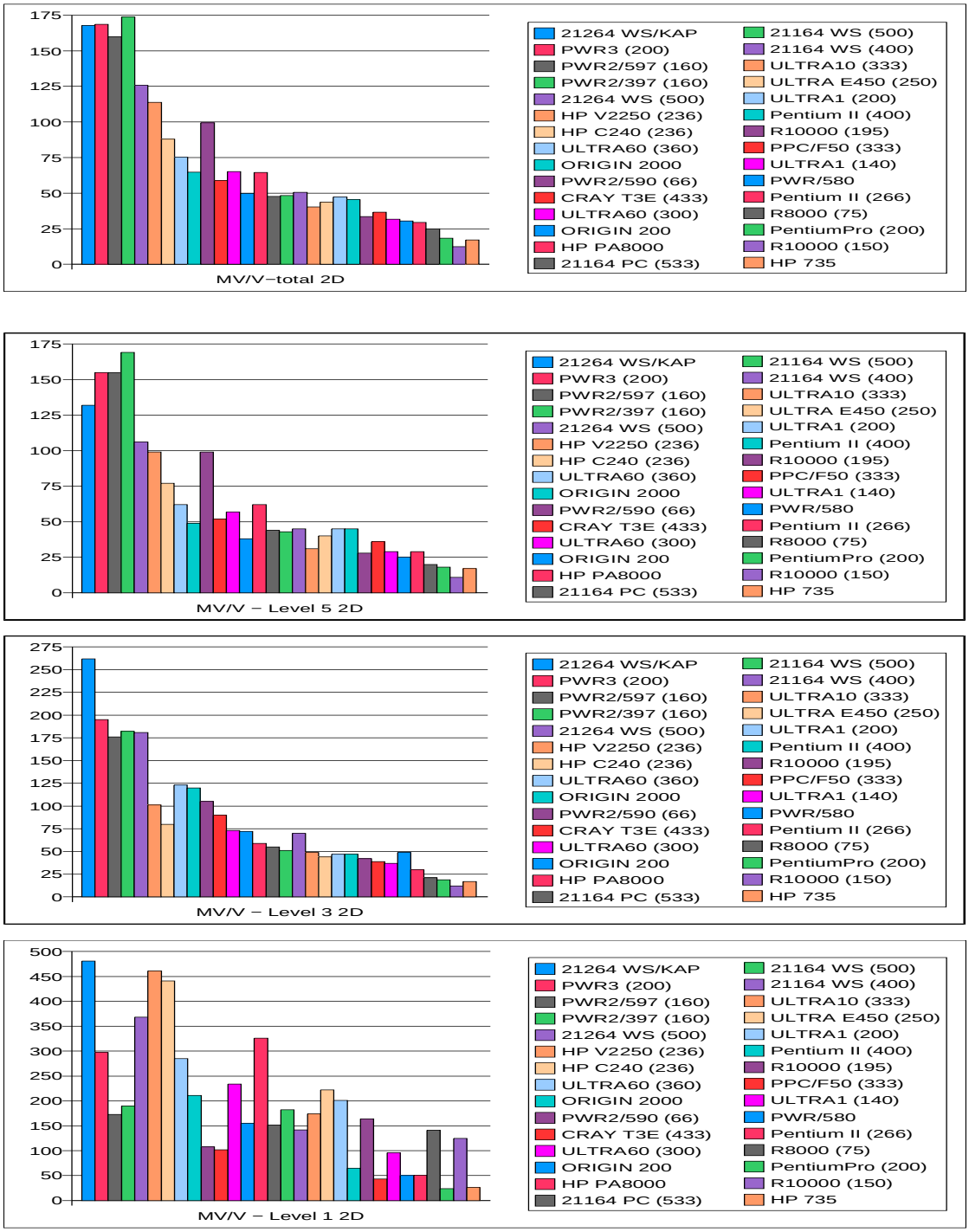


Figure 11: FEAST-MV-V INDEX **in 2D:** The first row shows the FEAST-MV-V INDEX, followed by the specific results for the `MV-V` routines on the different mesh levels. In contrast to the previous 3D case, the 'older' POWER2 architecture by IBM is superior which is equipped with much smaller L2-caches! Due to the more compact matrices (9–point instead of 27–point matrices in 3D!), the processors with larger cache sizes cannot gain the same improvements as in the 3D case. The quality of the 'really old' IBM 590 with 66 MHz clock rate only is surprisingly excellent if compared with many new processors!

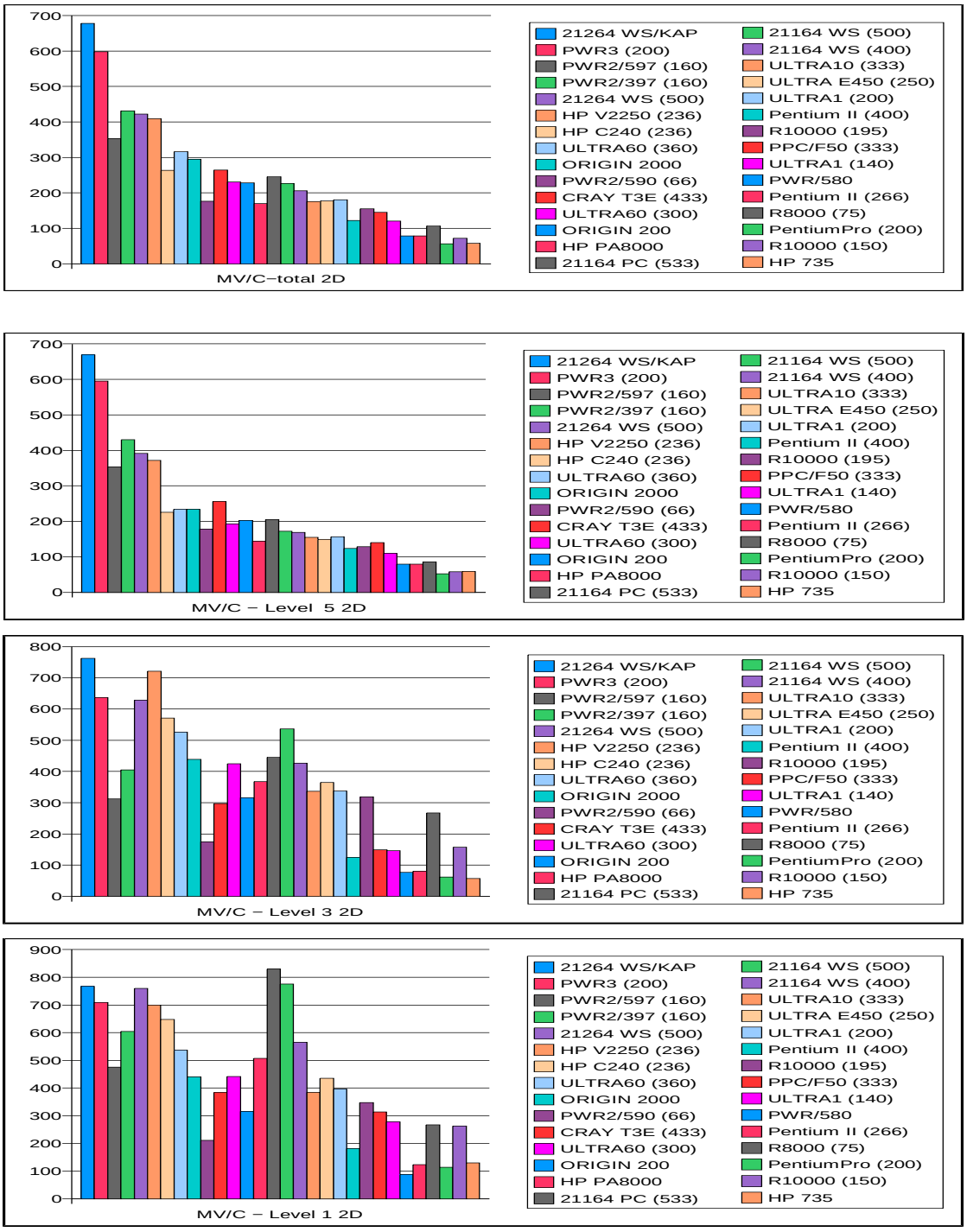


Figure 12: FEAST-MV-C INDEX **in 2D:** The first row shows the FEAST-MV-C INDEX, followed by the specific results for the `MV-C` routines on the different mesh levels. Again, these rates - up to almost 900 MFLOP/s - show the absolut importance of exploiting such structured patches if available. However, they also show how hard (or even impossible at the moment!) it is for certain computer architectures to achieve these high rates for large problem sizes.

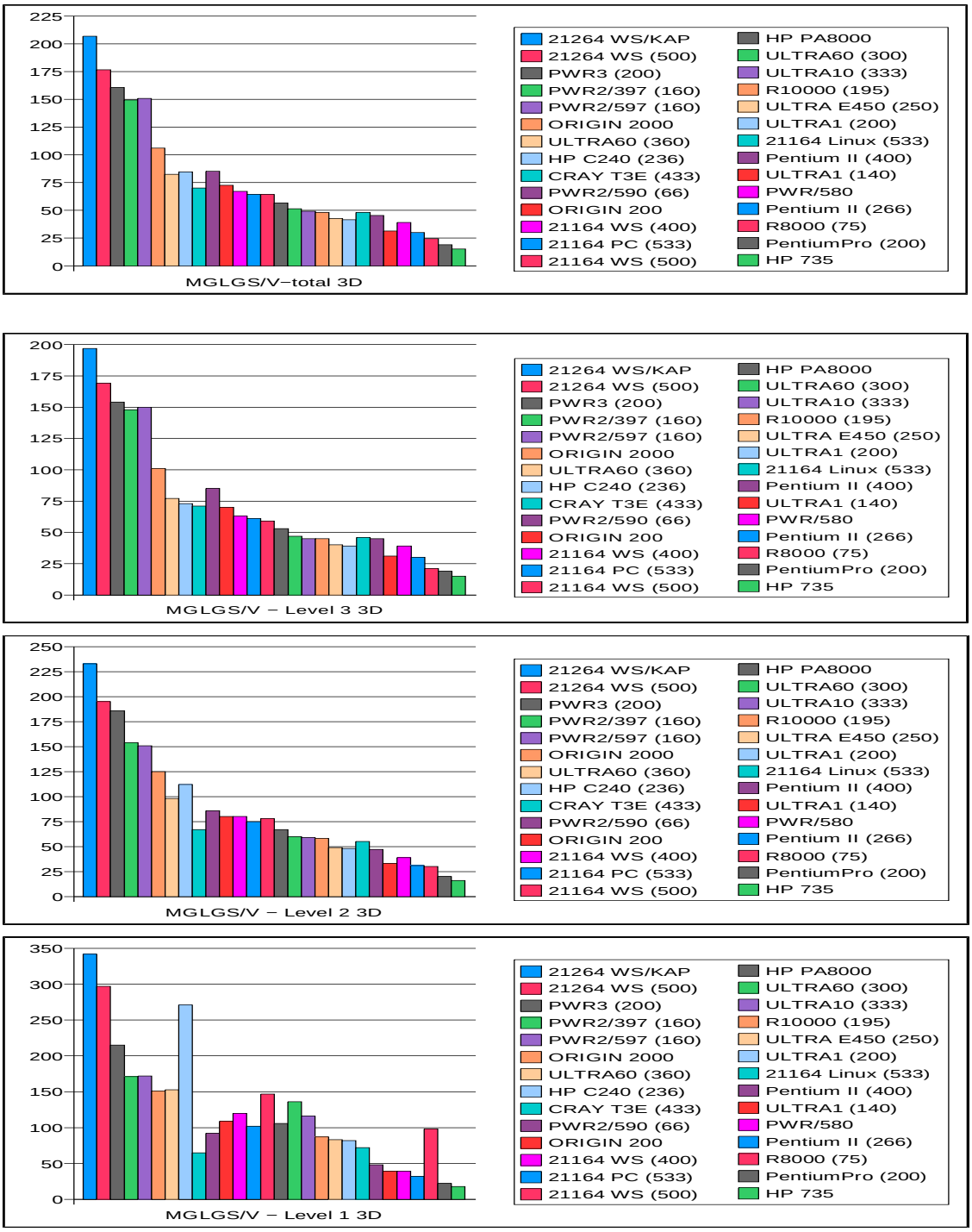


Figure 13: FEAST-MGLGS-V INDEX **in 3D:** The first row shows the FEAST-MGLGS-V INDEX, followed by the specific results for the `M-TRIGS` routines on different mesh levels. The computations show that not only MV multiplications can be efficiently applied, but also complete multigrid algorithms with a very robust smoother inside which works very efficient for regular as well as anisotropic meshes (see [1]).

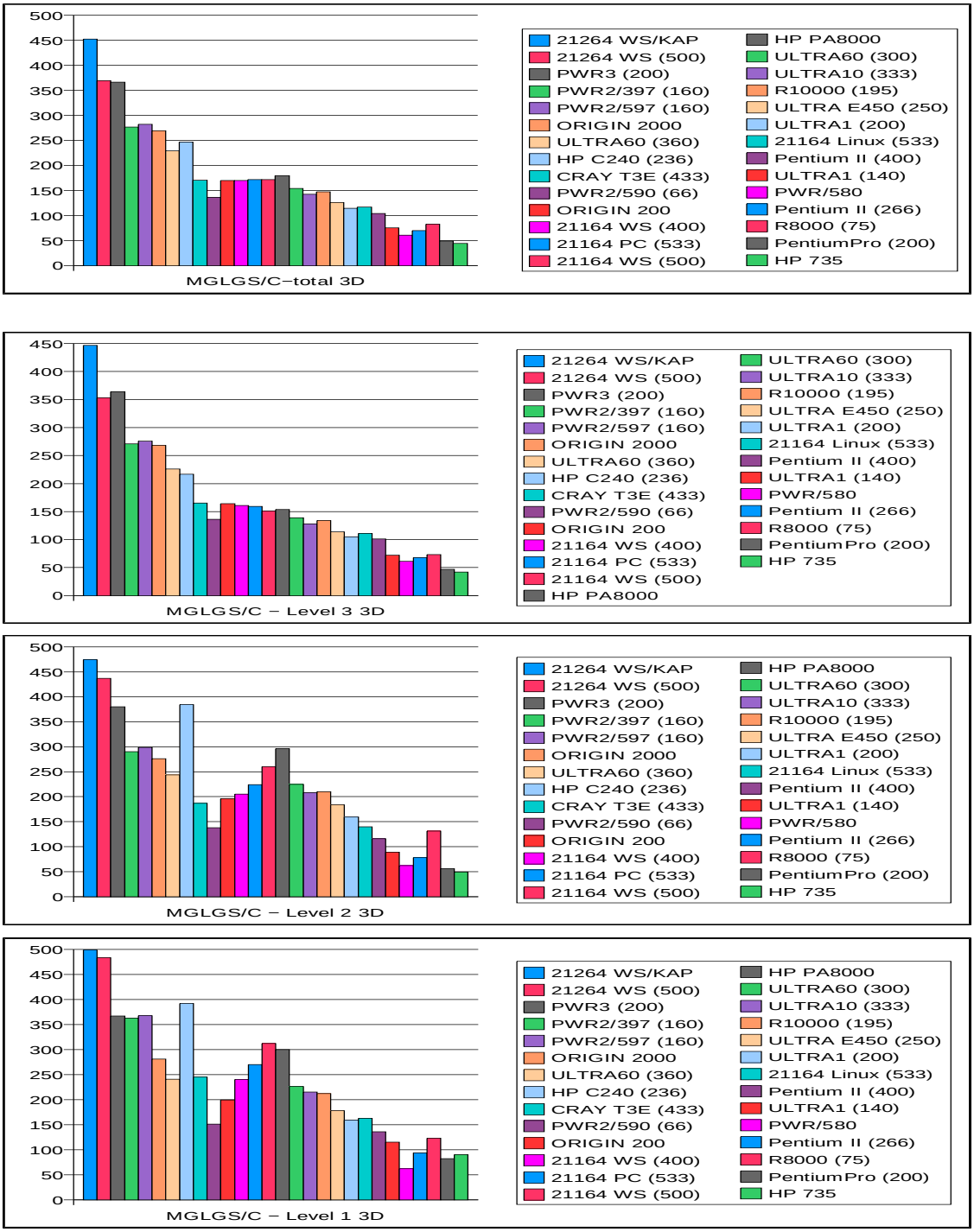


Figure 14: FEAST-MGLGS-C INDEX **in 3D:** The first row shows the FEAST-MGLGS-C INDEX, followed by the specific results for the `M-TRIGS` routines on the different mesh levels. Again, the demand for detecting locally structured patches is demonstrated since complex multigrid solvers with a very high numerical complexity can be performed with a correspondingly excellent computational efficiency, too.

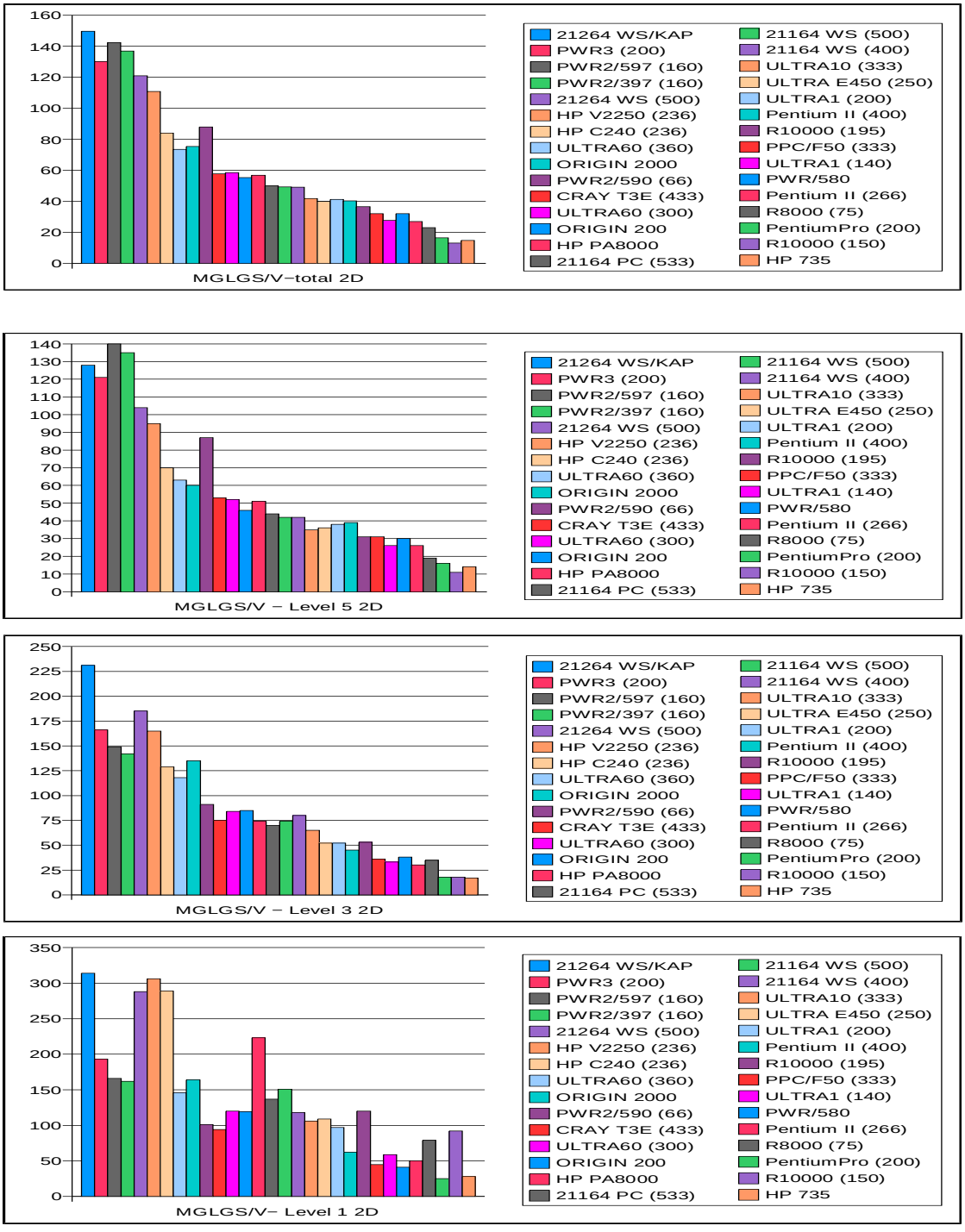


Figure 15: FEAST-MGLGS-V INDEX **in 2D:** The first row shows the FEAST-MGLGS-V INDEX, followed by the specific results for the `M-TRIGS` routines on different mesh levels. The 2D results are somewhat slower as the 3D case, as already indicated by the MFLOP/s rates for MV-V multiplication.

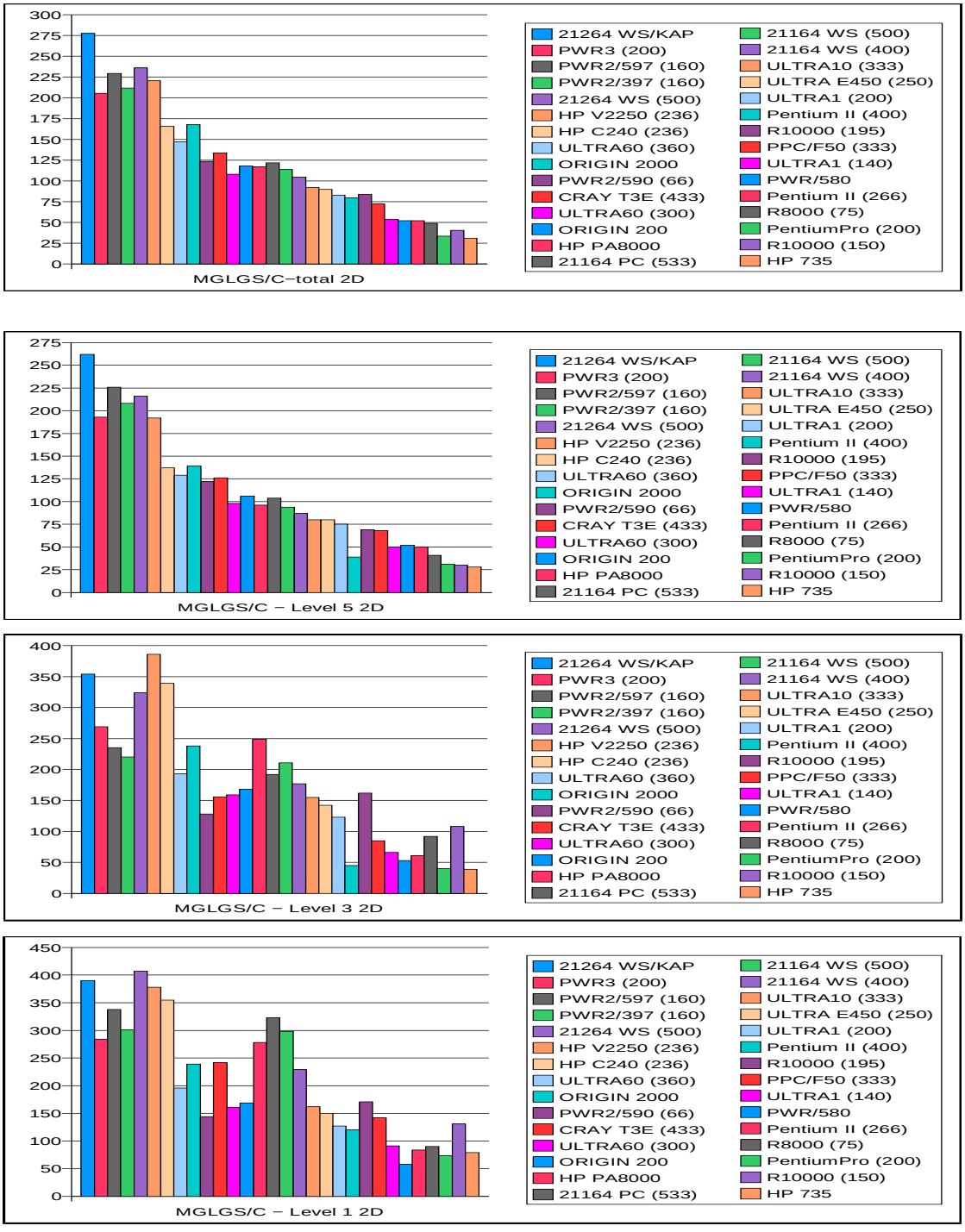


Figure 16: FEAST-MGLGS-C INDEX **in 2D:** The first row shows the FEAST-MGLGS-C INDEX, followed by the specific results for the `M-TRIGS` routines on different mesh levels. The 2D results are somewhat slower as the 3D case, as already indicated by the MFLOP/s rates for MV-C multiplication.

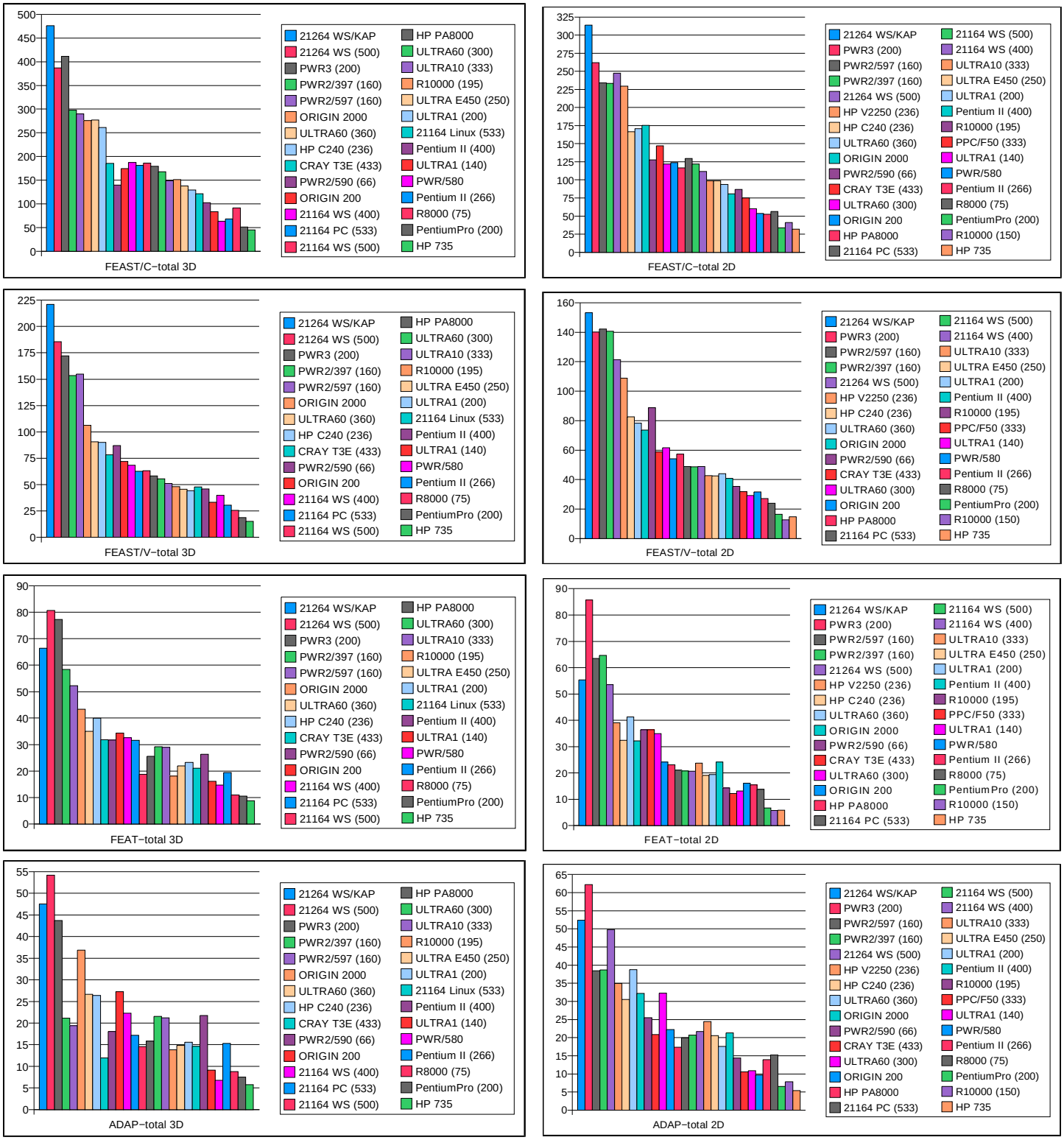


Figure 17: **Some indices (2D vs. 3D):** The reader should compare the absolute MFLOP/s results which are somewhat higher in 3D for the SPARSE BANDED BLAS applications in FEAST.