Max-Planck-Institut
für Mathematik
in den Naturwissenschaften
Leipzig

Benchmarking a Shared Memory System

by

Ronald Kriemann, Jens Burmeister
and Rainer Kleinrensing

Technical Report No.: 1 2003

Benchmarking a Shared Memory System

Ronald Kriemann, Jens Burmeister*, and Rainer Kleinrensing
Max-Planck-Institute for Mathematics in the Sciences,
Inselstr. 22-26, D-04103 Leipzig, Germany.
{rok, rainer}@mis.mpg.de

June 3, 2003

Abstract

For the evaluation of shared memory systems several programs were chosen and ported to different, today available computer systems. These applications represent important aspects of our numerical and algorithmic research. Beside the evaluation of the processor performance, the programs were chosen to test the scalability of the systems w.r.t. memory and the number of CPUs. The main characteristics of the benchmarked computer systems and the results from the individual benchmarks are presented. Furthermore, problems which occurred while porting the programs are discussed.

1 Introduction

In recent years, parallel computer systems based on cheap but powerful hardware, connected by a fast network gained more and more popularity. These *clusters* offer an almost unbeatable price/performance ratio. Although our research group also develops algorithms for such parallel computers, many of our applications only use a single processor and it was unrealistic to re-program all applications for a distributed memory machine. Instead a shared memory system was sought, which was able to address several gigabytes (GB) of main memory.

Of course, this computer should be fast, but pure processor performance was not what we looked for. More important was the scalability of the system w.r.t. the memory consumption of a program. The reason for this lies in the nature of our research, where algorithms are developed and the

*Chair of Practical Mathematics, Christian-Albrechts-University Kiel, Christian-Albrechts-Platz 4, 24118 Kiel, Germany, jb@numerik.uni-kiel.de

complexity of these algorithms is determined theoretically and confirmed by practical experiments. The same kind of theoretical and practical work is done for parallel programs, where the parallel scalability is examined.

These characteristics apply only for a shared memory system which implements *symmetric multi processing* (SMP) on a *uniform memory architecture* (UMA). A SMP-system is a single computer with two or more processors, which are managed by one operating system. Every processor has access to the same system resources, e.g. main memory or the input/output subsystem. The second term, UMA, refers to the performance of the memory subsystem. In a UMA-machine, each memory access takes the same amount of time.

In practise however, a perfect UMA-system is either too slow or too expensive to produce. Therefore, almost all computers offer a hierarchy of different memories, from a very fast, but small processor cache, to a much slower main memory. This results in a *non-UMA*- or NUMA-system and the question was, which computer showed the least influence of this architecture on the performance of the applications.

Because of this test conditions, the usual benchmarks, e.g. the SPEC-suite (see [16]), were only partially applicable. Especially the SPECint- and SPECfp-numbers, though they give a good impression of the CPU-performance, say only little about other machine parameters. Beside that, we were primarily not interested in how fast other applications are, but what the run-time of our programs is.

Therefore a suite of several benchmarks was chosen, which represent the most important types of software used in our group. These programs numerically treat integral and differential equations using multigrid methods (see [3]), the panel clustering technique (see [5]) or $\mathcal{H}$-matrices (see [6]). In section 2 these benchmark applications are introduced.

The benchmarked computer systems, along with their main properties are presented in section 3. These systems only represent some of the available machines. In particular, no *vector computers* were tested for the same reasons, as for the cluster systems, because the cost for porting the applications are too high.

The results of the benchmarks are given in section 4 and section 5 covers some problems, which occurred while porting the applications to the various operating systems and machines.

2 Benchmarks

Three parameters were identified to be important for the performance of the applications used by our research group: the absolute performance of the processor, the scaling behaviour w.r.t. the memory and the scaling w.r.t. the number of processors. The benchmark programs should be able to measure these parameters. Due to the nature of our research, this leads to the measurement of time and space complexity of the programs.

An ideal system should behave exactly like the theoretical complexity results. Especially with a large memory consumption of several GB or when using several processors simultaneously no additional effects due to the properties of the computer should be visible.

On the other hand, a computer system with exceptional scalability properties is useless, if the execution time is too high. It is therefore also desirable, that the benchmarked machine should be fast w.r.t. pure processor performance.

To test all these properties, three applications were chosen, which cover a wide range of the algorithms used in our group. The *Sensor* program solves partial differential equations using the multigrid technique. Integral equations are handled by the *LibBem* library. Our newest research area is covered by the *libHmatrix* software library, which implements $\mathcal{H}$-matrices for both kind of equations. The latter is used in two different benchmarks: matrix inversion (H-Inv) and matrix multiplication (H-Mul).

The following table gives an overview about which benchmark program was used to determine a specific property of the computer system. The benchmarks itself are described in detail in the following sections.

Benchmarks	CPU performance	Memory scaling	CPU scaling
Sensor	x	x	
LibBem	x	x	
H-Inv	x	x	
H-Mul	x	x	x

Only the matrix multiplication benchmark is used to determine the parallel scalability of the computer systems. This reflects the importance of sequential programs in our group.

For the comparison of other computer systems, the source code for all benchmark programs along with compilation and installation instructions is freely available from [17].

2.1 Sensor

This software library was created as part of a project to examine the influence of certain material parameters by computing the eigenvalues of the Maxwell equations (see [1]). It implements a multi-grid solver for the solution of the problem. In this specific benchmark a modified Poisson problem on a tensor product grid was solved using the multi-grid algorithm (see [3]) with block-Gauss-Seidel pre- and post-smoothers.

Important for the performance of an application on a computer system is the layout and the access of the data in the memory. In the Sensor benchmark the entries of the matrix are stored line-wise (w.r.t. the grid) and updated if needed. All vectors are represented by arrays. Therefore this application has a stream-like behaviour, where pre-fetch instructions could improve the overall performance.

The theoretical complexity results show a linear growth w.r.t. the problem size. For the considered test cases a factor of 4 was expected between two successive levels.

Because this library only runs on a single CPU, it was only used to check the processor performance and the memory scaling.

2.2 LibBem

The *LibBem*-library can be used to solve integral equations by the boundary element method (see [4]). In this particular case the panel clustering technique (see [7]) was applied to improve the overall complexity. Therefore the linear system is data sparse.

Due to panel clustering the system matrix is a sum of two matrices, which are both built out of small memory chunks, each in the order of 512 bytes. These sub-blocks are accessed via hash-tables or by going through lists. The memory layout and the read access to the data is therefore not necessarily contiguous and more an approximation of a random memory access.

The solution of the linear system is again done by an iterative algorithm (GMRES, see [9]) which performs matrix-vector multiplications and scalar products. The complexity of all algorithms is almost linear, up to some logarithmic terms.

Measured in the benchmark is the time to built the system matrix and to solve the linear system. The results of benchmarks using this library are used in the processor performance and memory scaling evaluation.

2.3 libHmatrix

$\mathcal{H}$-matrices, introduced by Hackbusch (see [6]), are a special format to store matrices in a data sparse way. Additionally one is able to perform the usual algebra, e.g. matrix-vector multiplication, matrix multiplication and matrix inversion in almost linear time. The exact costs are in $\mathcal{O}(n \log^2 n)$, where n is the problem size. The *libHmatrix* implements these functions for single- and for multiple-CPU machines.

The memory layout of $\mathcal{H}$-matrices is quite complex. They are hierarchically defined (therefore the "$\mathcal{H}$" in $\mathcal{H}$-matrices) and consist of single blocks. The size of these blocks varies in a large range, from some kB to several MB. Typical structures of the matrices involved in the algorithms used are shown in figure 1.

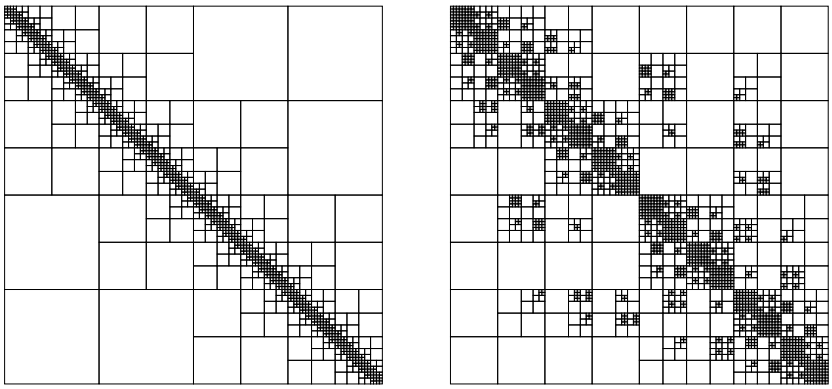

Figure 1: Block structure of the Matrices

The algorithms to multiply or to invert a $\mathcal{H}$-matrix result in a memory access pattern which exploits data locality to some degree, but also affects many blocks from different memory areas. Also temporary memory, used for intermediate results is used heavily during both computations.

The allocation pattern inside the algorithms led to some catastrophic run-times in earlier benchmarks if the memory consumption was rather high, e.g. several GBs. Therefore a special memory allocator was implemented which drastically improved the execution times. This malloc implementation is a variant of a *simple segregated storage* allocator (see [11]) and works by recycling the requested and freed memory blocks and reduces the number of calls to the system-malloc by requesting only large blocks of 5-10 MB. Due to the recycling strategy, memory is only given back to the operating system after the program terminates and therefore no work has to be done by the system-malloc to manage these freed blocks.

Many of the subproblems which occur during the $\mathcal{H}$-algorithms, can be written by using calls to BLAS- and LAPACK-functions (see [2]), e.g. matrix-vector or matrix-matrix multiplications, QR-factorisation and singular value decomposition. In particular, 80-90 % of the time is spent in these routines. Therefore an optimised version of BLAS and LAPACK for the benchmarked system is crucial for a good performance. Fortunately almost all systems supply some kind of software library providing these functions. Alternatively the freely available LAPACK implementation *CLAPACK* (see [15]) could be used which was modified to work in a multi-threaded environment. In some cases this library gave a smaller execution time than the "optimised" implementations.

Although the $\mathcal{H}$-matrix library is entirely written in C++, all matrix blocks are stored column-wise. This is a side effect of the BLAS- and LAPACK-usage where matrices are only accessed in that way.

2.3.1 Matrix Inversion Benchmark

The first benchmark based on *libHmatrix* computes the inverse of a $\mathcal{H}$-matrix. This matrix comes from a 1-dimensional problem discretized by the finite element method (FEM).

In the beginning, most of the matrix blocks are filled with zeroes which is recognised by the LAPACK routines. During the computation, the matrix entries become non-zero.

This benchmark is interesting because the asymptotic behaviour of the $\mathcal{H}$-matrix algorithms is reached at a relatively small problem size. Therefore the influence of the characteristics of the computer system can be tested.

2.3.2 Matrix Multiplication Benchmark

In the second *libHmatrix* benchmark, two $\mathcal{H}$-matrices are multiplied. The matrices itself stem from a 2-dimensional FEM-problem, but instead of the "real" matrix coefficients, they were filled using random numbers. In all benchmarks, the same sequence of these numbers was used to produce deterministic results.

This benchmark is the only program in this suite which makes use of a parallel computer. The multiplication can be done by using more than one CPU, whereby the parallelisation is based on POSIX threads.

The theoretical speedup of the multiplication is optimal, e.g. equal to the number of processors. Therefore a similar result is expected in the benchmarks.

3 Shared Memory Machines

The following sections describe the computer systems the benchmarks were performed on. For the authors access to some of these systems was possible (Sun SF6800, HP9000 and IBM p690); other systems were benchmarked by the distributor (Sun SF15k).

All of the computer systems described below try to approximate a perfect shared memory machine. This means a constant time for a memory access for all processors over the whole memory. In practise this is not possible without unreasonably high costs. Therefore usually a hierarchy of memory systems is used:

1. 1st level cache (on-chip),
2. 2nd level cache (on- or off-chip),
3. 3rd level cache (not in all systems, off-chip),
4. CPU local memory and
5. remote memory.

Some systems also have different levels of remote memory, depending on the internal network.

The enumeration above also gives the distance of the memory from the CPU, having 1st level cache the closest and remote memory the farthest. This distance usually has a big impact on two important numbers which characterise a memory system: *latency* and *bandwidth*. Latency is defined as the time between a request of a data from the memory system and the actual answer. Bandwidth is the number of bytes per second, the memory can deliver. As a rule of thumb: the farther away the memory subsystem is from the CPU, the higher the latency and the lower the bandwidth.

For a good approximation of a shared memory machine, the computer system should have a small number of memory hierarchies and especially the latency should not vary too much between all levels of the hierarchy.

All systems described below are 64-bit machines. This means that integer computations with a length of 64 bit and an address space for the main memory of 2^{64} bytes are supported . Floating point computations can be done in single (32 bit) or double (64 bit) precision following the IEEE-754 standard.

The information presented in this section was taken from the white papers [8], [12] [13] and [14], which are publicly available from the corresponding hardware vendors.

The below described computer systems do not cover all available shared memory systems. For instance the MIPS-based *SGI Origin 3000* or *Altix 3000* family of supercomputers or the *Fujitsu Primepower* series fall into the same category. But unfortunately the authors didn't have access to such systems to run the benchmark programs.

3.1 Sun SunFire 6800

The *Sunfire 6800* (SF6800) is a system based on the *UltraSparc III Cu* processor (US3). It supports up to 24 CPUs and up to 192 GB of main memory. It is built out of CPU boards with a maximum of 4 CPUs and 32 GB memory each. All CPU boards are connected by a *Sun Fireplane Interconnect*.

The US3 is a processor with a 14 stage pipeline. It has an on-chip data cache of 64 kB and a 32 kB instruction cache. An integrated controller for the 2nd level cache supports up to 8 MB. Also integrated into the CPU is the controller for the main memory, thereby reducing the latency for an access to local memory.

Inside a single CPU board, the bandwidth for all CPUs to the on-board memory is 2.4 GB/s. The Interconnect delivers a sustained bandwidth of 9.6 GB/s with a peak bandwidth between two different CPU boards of 4.8 GB/s. The latency to the local memory of a processor is 180 ns and 240 ns to memory on a different CPU board.

The SF6800 is able to be partitioned into four domains, where the minimal block for a domain is a single CPU board. All domains are completely separated from each other. Furthermore, the system allows a dynamic reconfiguration of all partitions.

In this benchmark a system was tested with 24 CPUs running at 900 MHz and 96 GB main memory. The benchmarks were done by Sun and by the authors.

The following table contains the versions of the operating system (OS), of the compilers (with optimisation flags) and of the BLAS- and LAPACK-library:

OS	Sun Solaris 9
C-Compiler	Sun Forte Developer 7 C 5.4
C-flags	-fast -xO5 -xprefetch -xprefetch_level=3 -xtarget=ultra3cu -xarch=v9b
C++-Compiler	Sun Forte Developer 7 C++ 5.4
C++-flags	same as C-flags
BLAS/LAPACK	Sun Forte Developer 7 Performance Library

3.2 Sun SunFire 15k

Very similar to the SF6800 is the *Sunfire 15k* (SF15k). The main difference is the number of CPU boards, which can be used. Instead of 6 boards, 18 boards with a maximum of 72 CPUs are possible in this system. The boards itself and the CPUs are identical between both machines.

With a bigger number of boards, a more enhanced crossbar is needed. The SF15k uses a switch with a peak bandwidth of 43.2 GB/s . The latency for a memory access over this crossbar should also be different compared to the SF6800 but unfortunately the exact numbers could not be determined. The results of the benchmarks indicate a larger value than 240 ns for remote memory accesses.

Also growing with the board number is the number of domains. Up to 18 partitions are possible with the SF15k.

The operating system and the compiler were the same as on the SF6800.

3.3 HP 9000 Superdome

Like the Sunfire 6800 also the *HP 9000 Superdome* (HP9000) is built out of single boards, each holding 4 CPUs and up to 4 GB main memory. The maximal number of CPUs is 64 resulting in a total memory of 256 GB.

Up to now, the processor available for the HP9000 is the *PA-8700+*, running at 875 MHz. It has an on-chip data cache of size 1.5 MB and an instruction cache of 0.75 MB, also on-chip. Future versions of the HP9000 are planned with the *Intel Itanium 2* processor.

On each board, the 4 processors are connected to the *cell controller* which handles memory- and IO-requests and the connection to the crossbar. The peak memory bandwidth on such a board is 4 GB/s and the bandwidth to the crossbar is 8 GB/s at maximum. The minimal latency for a request to the board local memory is 174 ns.

Depending on the total number of boards (and therefore CPUs), the HP9000 is equipped with a different number of hierarchies. In a 64 CPU system 4 boards are connected to a crossbar, resulting in 4 crossbars, which are again connected to each other. This configuration comes with a maximal crossbar bandwidth of 64 GB/s. A memory request over a single crossbar has a minimal latency of about 240 ns and of about 300 ns over two crossbars.

The HP9000 allows several modes on partitioning the system, from complete hard- and software-separation to virtual domains inside single boards. Up to 64 virtual and 16 *hard* partitions are possible, each running its own operating system. The hard partitions require at least one complete CPU board.

Used for the benchmark was an HP9000 equipped with 64 processors and 128 GB main memory. All programs were run by the authors.

The next table gives a list with the versions of the operating system, the compilers and the libraries used during the benchmark.

OS	HP-UX 11i
C-Compiler	HP AnsiC Version B.11.01.21
C-flags	-fast +DA2.0W +DD64 +O4 +Odataprefetch
C++-Compiler	HP ANSI C++ Version A.03.31
C++-flags	-mt -AA -fast +Odataprefetch +O2 +DA2.0W +DD64 +tm K8000
BLAS/LAPACK	HP MLIB Version B.07.00

When using the C++-compiler an optimisation level of more than 2 led to an error while compiling the program. Therefore only the standard optimisations were used.

3.4 Compaq AlphaServer GS320

Another system which is designed around 4 CPU blocks is the *Compaq AlphaServer GS320*. Here the CPU boards are called *Quad Building Blocks* (QBB) and support 4 *Alpha 21264 EV67* processors with up to 1.2 GHz. Putting 8 QBBs together gives the maximal configuration of 32 CPUs and 256 GB main memory.

The EV67 CPU has an on-chip instruction- and data-cache of 64 kB each. The 2nd level cache is off-chip and has a size of 4 MB. Each CPU can access the QBB-local memory with a peak bandwidth of 1.6 GB/s and a latency of about 330 ns.

Access to memory of other QBBs is much more costly. It has a latency of 960 ns. The crossbar, which connects all QBBs has a maximal bandwidth of 12.8 GB/s.

The latency difference between local and remote memory of a factor of 3 results in a system which can be better described as a NUMA-architecture. Especially serial programs consuming much main memory seem to be a problem for the GS320 as early benchmark results using the *Sensor* program indicated. Unfortunately the whole benchmark suite could not be run on this system.

Hardware partitions are also possible with the GS320. Up to 8 domains with complete hard- and software-separation can be used. The partitioning is dynamic and allows the online-add or -removal of a QBB to a domain.

3.5 IBM eServer p690

Compared to the systems so far, the *IBM eServer p690* (P690) follows a slightly different approach to implement a shared memory system. It is built around *Multi Chip Modules* (MCM) each consisting of 4 *Power4* chips running with up to 1.3 GHz.

The Power4 processor is available in two different configurations, either with one or two cores. The latter results in 8 CPUs per MCM. Beside the 32 kB data- and the 64 kB instruction-cache each Power4 chip also contains a L2 cache of 1440 kB. If two cores are present in the Power4, the L2 cache is shared, which possibly leads to contention between both cores. For each Power4 chip, the MCM has separate connections to an off-chip L3 cache of 32 MB. This is again shared in the 2-core configuration. Connected to each L3 cache is the memory controller of the Power4 chip which has access to one of two memory banks of each MCM.

Inside the MCM all Power4 chips are able to communicate directly with each other. This allows all CPU-cores to access the MCM-local memory as fast as possible, thereby using a L3 cache of one of the other Power4 chips. An optimal performance is achieved if both memory banks of the MCM are equipped with an equal amount of memory, thus balancing the load to all L3 caches.

Each MCM can be connected with up to two other MCMs, thereby eliminating the need for a special crossbar. The maximal configuration consists of 4 MCMs with 32 CPUs and up to 256 GB main memory. Such a system, only with 64 GB memory, was used during the benchmarks. All programs were run by the authors. It should be mentioned, that the benchmark machine was heavily used by other programs and therefore the execution times might not be optimal.

A list of the operating system and compilers used for the benchmark is presented in the next table. Unfortunately only an optimisation level of 2 could be used for the compiler. Any further optimisations led to errors or deadlocks of the programs.

OS	IBM AIX5
C-Compiler	IBM VisualAge C Version 6
C-flags	-q64 -O2 -qinline -qarch=pwr4 -qtune=pwr4
C++-Compiler	IBM VisualAge C++ Version 6
C++-flags	-q64 -qrtti -O2 -qinline -qarch=pwr4 -qtune=pwr4
BLAS/LAPACK	IBM ESSL Version 3.3.0.0

4 Benchmark Results

In this section, the results of the benchmarks on the different shared memory systems is presented. Beside the actual times for each run, two other numbers are important: the scaling factors w.r.t. memory and the parallel efficiency.

The size of the problem is characterised by the *level* l and the time is given w.r.t. a specific level: $t_{bench} = t_{bench}(l) = t_{bench}(l, p)$. Here subscript "bench" is the name of the corresponding benchmark and is omitted if it is clear from the context. The parameter p defines the number of processors. For serial programs ($p = 1$) this parameter is also omitted.

The quotient of the times between two successive levels is the *memory scaling coefficient*

$$S = S(l) = \frac{t(l)}{t(l-1)}.$$

It will also be printed for each run of a serial program to show the behaviour of the machine w.r.t. an increased problem size. The value of S depends on the complexity of the involved algorithms and should be 4 in the *Sensor*- and *LibBem*-benchmarks and about 2 in the matrix-inversion benchmark. Due to the slow convergence of the complexity of the matrix multiplication, S is expected to be larger than 5.

When using more than one processor, the *parallel efficiency*

$$E_p = E_p(l) = \frac{t(l)}{p \cdot t(l, p)},$$

where p is the number of CPUs, is more interesting and will be presented instead of S. In the ideal case, E_p should be 1 (or 100 %).

In addition to the absolute time values, the relative speedup to a reference platform, a Sun E6000 with 16 UltraSparc II CPUs running at 250 MHz and 16 GB main memory, is shown for each benchmark. The programs on this machine were run by the authors. Due to memory constraints the benchmarks could not be executed on the highest levels and the data was therefore extrapolated.

The last part of this section contains a summary of all results into single performance numbers for the CPU speed, the memory- and the processor-scaling. These numbers try to give an overview of the most important properties of the benchmarked computer systems for an easy comparison.

4.1 Sensor

In the *Sensor* benchmark the HP9000 shows a weak performance w.r.t. memory scaling. Although it performs very well on the smaller levels, it breaks in at the two highest runs. The reason for this could not be determined, but seems to be a large increase in the number of misses to the *translation lookaside buffer*[1]. But also the increased latency for memory accesses on remote boards may have caused this performance drop.

All other systems show a relatively stable behaviour, although an influence of remote memory accesses is visible.

level	SF6800		SF15k		HP9000		P690		
	t	S	t	S	t	S	t	S	Mem
9	14	-	14	-	8	-	5	-	76
10	60	4.36	62	4.37	31	3.67	21	4.10	221
11	269	4.50	260	4.22	115	3.76	82	3.99	799
12	1024	3.80	1061	4.08	468	4.07	372	4.55	3107
13	4194	4.10	4327	4.08	3379	7.22	1473	3.95	12332
14	17942	4.28	24149	5.58	19345	5.73	6591	4.48	49213

A value of S below 4 (HP9000 and P690) might occur due to cache effects. Especially on small levels, this effect may be larger than the increase due to accesses to remote memory.

In figure 2, which shows the relative speedup w.r.t. the reference system, the bad performance of the HP9000 on the highest levels is again visible. A clear winner is the P690 which has an excellent speedup and a constant memory scaling. A reason for this might be the very fast memory system of the P690 which seems to be superior compared to the other machines.

4.2 LibBem

A similar behaviour as in the Sensor benchmark can be seen in the LibBem program. Again the HP9000 has a very unstable memory scaling, although this time it seems to be limited to a specific problem size: the big jump from level 6 to level 7 can almost be compensated on level 8.

[1]The translation lookaside buffer (TLB) is a fast lookup table inside the CPU to convert virtual into real memory addresses. A TLB-miss leads to a delay because the real memory address has to be calculated differently.

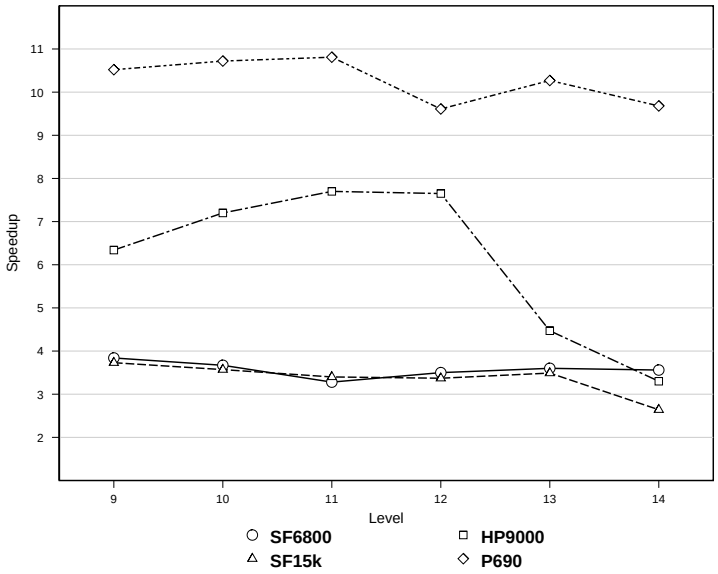

Figure 2: Relative Speedup in the Sensor Benchmark

level	SF6800		SF15k		HP9000		P690		
	t	S	t	S	t	S	t	S	Mem
5	69	-	70	-	70	-	64	-	412
6	341	4.98	356	5.06	310	4.42	281	4.41	1846
7	1377	4.03	1474	4.14	2639	8.52	1236	4.40	8185
8	6310	4.58	8130	5.52	5946	2.25	5491	4.44	36122

Another interesting point is the small gap in the execution times between all machines. This is also visible in figure 3. Compared to the large performance difference in the Sensor benchmark, especially the wide margin of the P690, this result is quite unexpected. One possible explanation might be the random memory access pattern of the LibBem program, where fast a cache subsystem is of no use.

Like in the Sensor benchmark, the best system is the P690, although only by a small margin.

4.3 libHmatrix

The first table shows the results of the matrix inversion benchmark. Because the dimension increases by a factor of 2 by each level, this factor is expected for the scaling coefficient. Unfortunately the logarithmic terms in the complexity estimations result in a factor of about 2.3 .

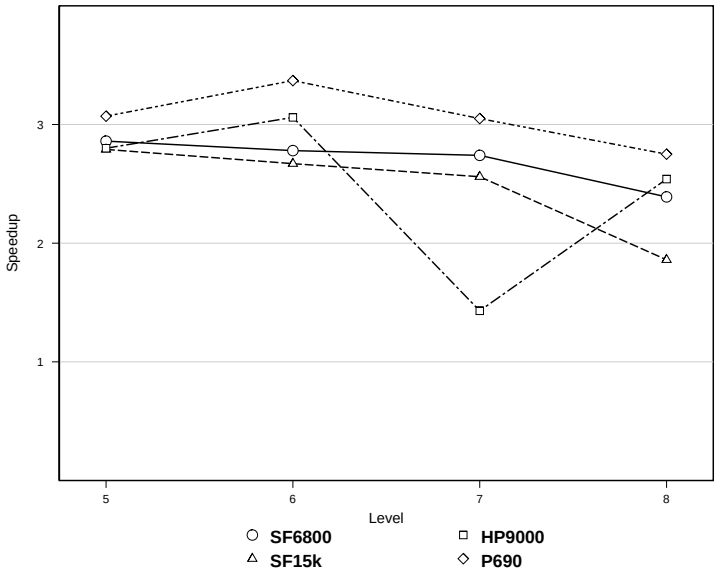

Figure 3: Relative Speedup in the LibBem Benchmark

Matrix Inversion									
level	SF6800		SF15k		HP9000		P690		
	t	S	t	S	t	S	t	S	Mem
15	53	-	56	-	45	-	33	-	314
16	129	2.42	134	2.39	108	2.42	78	2.47	667
17	300	2.33	314	2.34	261	2.42	183	2.18	1438
18	704	2.35	741	2.36	634	2.27	430	2.34	3062
19	1618	2.30	1721	2.32	1561	2.46	986	2.29	6523
20	3735	2.31	3997	2.32	3710	2.38	2364	2.40	14002
21	8665	2.32	10381	2.60	8834	2.38	5443	2.30	29602

The increase of S, observed in the *Sensor*- and *LibBem*-benchmarks is only visible on the SF15k. All other systems show a stable behaviour w.r.t. memory scaling. This stability is probably due to the massive usage of BLAS- and LAPACK-routines, which are optimised and exploit cache effects and therefore avoid too many accesses to remote memory.

A slightly different picture comes from figure 4. Here one can see a decrease in the relative speedup of the HP9000. All other systems show a stable performance compared to the reference system.

In the next table, the time to multiple two $\mathcal{H}$-matrices on one processor is given for an increasing problem size.

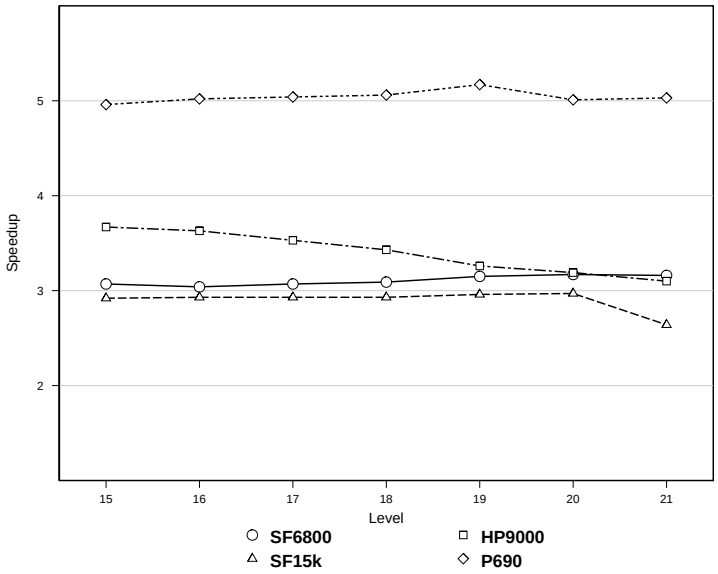

Figure 4: Relative Speedup in the Matrix Inversion Benchmark

Matrix Multiplication (serial)									
level	SF6800		SF15k		HP9000		P690		
	t	S	t	S	t	S	t	S	Mem
6	39	-	39	-	29	-	23	-	74
7	256	6.51	256	6.52	194	6.60	149	6.51	343
8	1509	5.89	1545	6.03	1148	5.93	842	5.65	1637
9	8336	5.53	8377	5.42	6519	5.68	4739	5.63	7708
10	45193	5.42	50475	6.03	38541	5.91	25587	5.40	35573

In contrast to the matrix inversion, the matrix multiplication does not seem to have reached the asymptotic behaviour with an expected value of S between 4 and 5. Only the SF6800 and the P690 seem to converge to this range. All other systems show a slight increase in S at the last level.

A little bit better are the results from the speedup comparison shown in figure 5. There all machines have a very stable performance increase throughout all levels compared to the reference system.

The last table presents the parallel efficiency of the matrix multiplication on the highest level for a different number of threads. All numbers are given in percent ($100 \cdot E(l, p)$).

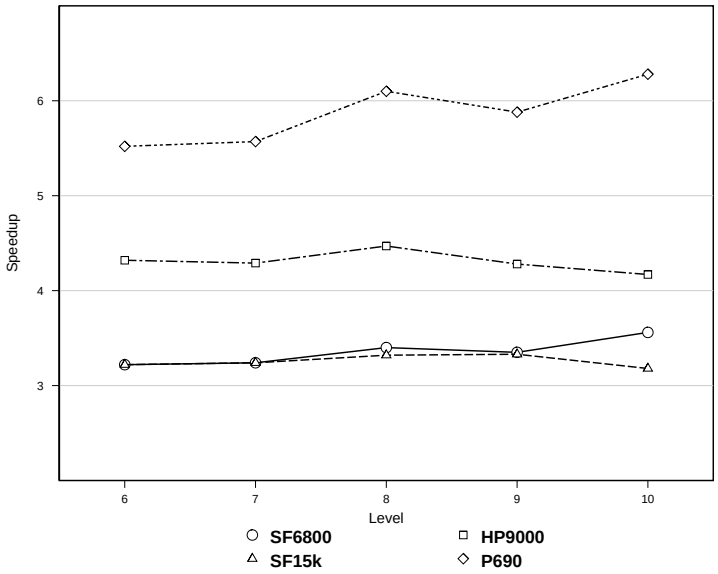

Figure 5: Relative Speedup in the Matrix Multiplication Benchmark

Matrix Multiplication (parallel)								
threads	SF6800		SF15k		HP9000		P690	
	t	E	t	E	t	E	t	E
4	11763	96.1	12188	103.5	9733	99.0	7257	88.1
8	5959	94.8	6185	102.0	5031	95.8	4245	75.3
12	4147	90.8	4762	88.3	3471	92.5	3426	62.2
16	3118	90.6	3887	81.2	2575	93.5	3490	45.8

The HP9000 shows the best efficiency with a value of about 93 %, followed by the Sun SF6800 and the Sun E6000 which seem to stabilise at about 90 %. Slightly worse was the parallel performance of the Sun SF15k, which drops to about 80 %. The worst efficiency was visible on the P690. It only seems to scale up to 8 CPUs and shows no improvement afterwards. Even tweaking some variables of the AIX POSIX thread implementation did not change this situation, which leads to the conclusion, that the P690 might have some scalability problems w.r.t. multiple CPUs.

4.4 Summary

The first coefficient sums up the results concerning the CPU performance of the benchmarked systems. For this the values of the relative speedup

$$R_{bench} = \frac{t^{ref}_{bench}(l)}{t_{bench}(l)}$$

achieved at the highest level of each benchmark are equalised. The (extrapolated) times t^{ref} for the reference system are given in the next table.

Sensor	LibBem	H-Inv	H-Mul
63833.6	15103.4	27394.4	160680.9

The CPU performance coefficient is finally defined as:

$$C_{CPU} = (R_{Sensor} \cdot R_{LibBem} \cdot R_{H\text{-}Inv} \cdot R_{H\text{-}Mul})^{1/4}$$

Due to the definition of C_{CPU} a larger value is better.

For the second number, the memory scaling is determined. But instead of dividing the times of two successive levels as for S, the following numbers were used: for the Sensor benchmark level 14 and 12, for LibBem level 8 and 6 and for the matrix inversion the levels 21 and 19. Only the matrix multiplication uses the last two levels, because here the asymptotic range has not been reached.

$$C_{MEM} = \left(\frac{16 \cdot t_{Sensor}(12)}{t_{Sensor}(14)} \cdot \frac{16 \cdot t_{LibBem}(6)}{t_{LibBem}(8)} \cdot \frac{5 \cdot t_{H\text{-}Inv}(19)}{t_{H\text{-}Inv}(21)} \cdot \frac{5 \cdot t_{H\text{-}Mul}(9)}{t_{H\text{-}Mul}(10)} \right)^{1/4}$$

The additional factors in C_{MEM} for each benchmark are related to the complexity of the algorithms and the difference in the problem sizes between the level numbers. Like S also C_{MEM} should have a value of 1 in the ideal case.

Finally the parallel efficiency of the matrix multiplication on level 10 with 16 processors is used for the third coefficient:

$$C_{PAR} = E_{16}(10)$$

The results for all machines are presented in the following table:

	SF6800	SF15k	HP9000	P690
C_{CPU}	3.13	2.53	3.23	5.39
C_{MEM}	0.91	0.76	0.70	0.89
C_{PAR}	0.91	0.81	0.94	0.46

The P690 clearly dominates the benchmark when it comes to the performance of the processor. Unfortunately this strong CPU-performance is not visible when it comes to parallel efficiency. Whether this is a general problem of the architecture or could be overcome by a special optimisation could not be determined. A better result is obtained w.r.t. memory scalability where the P690 takes the second place.

While the SF6800 and the HP9000 show a similar performance w.r.t. to the CPU-speed and the parallel efficiency, the HP9000 has an obvious problem with the memory scaling.

The last place of the competition is taken by the SF15k. It is the slowest system w.r.t. CPU-performance and has some problems with a large memory consumption and with programs using several CPUs.

5 Problems

This section describes some of the problems which occurred while porting the benchmark programs to the various computer systems.

Although all programs were developed on Sun- and Linux-machines, it was in general avoided to use specific features of these systems. Furthermore it was important to follow the definition of the programming languages as closely as possible. This strategy helped a lot when porting a program to a new system. Most of the time only a recompilation was necessary.

Only if the program worked directly with the operating system, some problems occurred. These areas are memory management and threads.

On Sun Solaris and Linux it was possible to overload the *malloc* function, which is usually supplied by the system. This led to errors on Tru64 and HP-UX. Because only C++-programs used the alternative memory manager, this problem could be circumvented by overloading the C++-operators *new* and *delete* instead.

When using more than 8 threads on the HP-UX operating system, random lockups occurred during the matrix-multiplication benchmark. This problem was solved by increasing the size of the *stack* in each thread, although an explanation for this behaviour could not be found.

A serious problem related to floating-point operations was observed on the Compaq AlphaServer. Although the *LibBem*-benchmark compiled without a warning, the program produced not the desired result, e.g. the GMRES-iteration "converged" after one step with an error of 0. The exact nature of this problem could not be determined.

5.1 Compiler Problems

Not all C++-compilers have implemented the full set of all features defined in the current definition of this programming language. Especially the correct handling of `templates` is still a problem.

The benchmarks based on *libHmatrix* are using *expression templates* (see [10]), which allow a very efficient handling of template classes by resolving some part of the computation during the compilation phase. Very often this technique is applied to classes for storing vectors. This was also the case in the benchmark programs.

On AIX and HP-UX, the C++-compilers were not able to translate this part of the code. Instead a classical, and less efficient, version of a vector class had to be used. Fortunately this affected only the grid management and not the matrix computations. Therefore it had no influence on the benchmark times.

Beside a complete implementation of the programming language standard, a compiler should also optimise the final program w.r.t. execution times. Especially for new processors, this optimisation might be difficult and probably not optimal.

On the Sun-systems (SF6800 and SF15k) the run-times of the programs were heavily reduced by using the latest compilers (Sun Forte Developer 7) which were only available as beta versions. Especially the utilisation of data pre-fetching, not supported by older compiler versions, was responsible for these improvements.

5.2 Software Libraries

On AIX, only a subset of the full LAPACK-functions is implemented in the *ESSL*-library. Additionally the names of these functions are different from the standard, although a mapping between these function sets is possible. Because lack of time, the CLAPACK-library (see section 2.3) was used instead. Fortunately the implementation of BLAS in ESSL is complete, therefore at least these functions could be used.

Another problem with the LAPACK-libraries occurred on the Sun-machines. When using many threads, often a lockup of the benchmarks was observed. A similar problem occurred if the CLAPACK-library was used instead of the SunPerf-library. Fortunately the source code for this implementation was accessible and the error was found in the initialisation of some static variables (in “dlamch” and “dlartg”). If this part of the library was executed by at least two parallel threads, the error occurred.

After calling these functions before any parallel computations, the error could be avoided. Fortunately, after doing this pre-initialisation also the error in the SunPerf-library vanished, although a bug in the library could not be found by the Sun-developers.

References

[1] S. Börm: *Mehrgitterverfahren für die Simulation zylindersymmetrischer elektromagnetischer Felder*, dissertation.de, Verlag im Internet, 2000.

[2] J. Dongarra and J. Demmel: *LAPACK: a portable high-performance numerical library for linear algebra*, Supercomputer, 8:33-38, 1991.

[3] W. Hackbusch: *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin, 1985.

[4] W. Hackbusch: *Integralgleichungen. Theorie und Numerik*, Teubner-Verlag, Stuttgart, 1989.

[5] W. Hackbusch and Z.P. Nowak: *On the Fast Matrix Multiplication in the Boundary Element Method by Panel Clustering*, Numer. Math. 54, 1989, 463-491.

[6] W. Hackbusch: *A sparse matrix arithmetic based on $\mathcal{H}$-matrices. Part I: Introduction to $\mathcal{H}$-matrices.*, Computing 62:89-108, 1999.

[7] C. Lage: *Softwareentwicklung zur Randelementmethode: Analyse und Entwurf effizienter Techniken*, Dissertation, 1995.

[8] H.M. Mathis, J.D. McCalpin, Men-Chow Chiang, P.O'Connell, P. Buckland: *IBM eServer pSeries 690 Configuring for Performance*, Jan 2002.

[9] Y. Saad and M.H. Schultz: *GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems*, SIAM J. Comput., 7(3):856-869, 1986.

[10] T. Veldhuizen: *Expression Templates*, C++ Report, Vol. 7 No. 5 June 1995, pp. 26-31.

[11] P.R. Wilson, M.S. Johnstone, M. Neely and David Boles: *Dynamic Storage Allocation: A Survey and Critical Review*, In 1995 International Workshop on Memory Management, LNCS. Springer Verlag, 1995.

[12] *New Compaq AlphaServer GS Series: Architecture White Paper*, Compaq White Paper.

[13] *Sun Fire^{TM} 3800-6800 Servers – Computing for Lower Total Cost of Ownership (TCO)*, Sun Technical White Paper, Feb 2002.

[14] *Meet the HP Superdome Servers*, Hewlett-Packard White Paper, May 2002

[15] *CLAPACK (f2c'ed version of LAPACK)*, Internet-address: http://www.netlib.org/clapack/ .

[16] *Standard Performance Evaluation Corporation (SPEC)*, Internet-address: http://www.spec.org

[17] *Max Planck Institute for Mathematics in the Sciences*, Scientific Computing, Internet-address: http://www.mis.mpg.de/scicomp/