

Performance of Scientific Applications on Modern Supercomputers

Frank Deserno, Georg Hager, Frank Brechtefeld, and Gerhard Wellein

Regionales Rechenzentrum Erlangen

Martensstr. 1

91058 Erlangen, Germany

{frank.deserno,georg.hager,gerhard.wellein}@rrze.uni-erlangen.de

Abstract. We discuss performance characteristics of scientific applications on modern computer architectures, ranging from commodity “off-the-shelf” (COTS) systems like clusters, to tailored High Performance Computing (HPC) systems, e.g. NEC SX6 or CRAY X1. The application programs are selected from important HPC projects which have been supported by the KONWIHR project cxHPC. In general we focus on the single processor performance and give some optimisation/parallelisation hints, if appropriate. For computational fluid dynamics (CFD) applications we also discuss parallel performance to compare COTS with tailored HPC systems. We find, that an HPC environment with a few tailored “central” high-end systems and “local” mid-size COTS systems supports our users’ requirements best.

1 Introduction

The rapid advances in microprocessor technology have led to fundamental changes in the HPC market over the past ten years. Commodity cache-based microprocessors arranged as systems of interconnected SMP nodes nowadays dominate the TOP500 list due to their unmatched price/peak performance ratio. For instance, in the current TOP500 list [1] 34 % of all systems are COTS clusters based on Intel processors. However, it has also been recognised that the gap between sustained and peak performance for scientific applications on such platforms is growing continuously [2].

Although it is well known that classical vector systems can bridge this performance gap especially for memory intensive codes, their fraction in the TOP500 crashed from 63 to 3.5 % (November 2003) within 10 years. One reason for this can be found in the combination of high development costs for new technologies and a limited market volume for HPC systems. Some authors also speculate that the ASCI program [3] in the USA has put a very heavy emphasis on the use of commodity-off-the-shelf components [4]. Consequently, only one manufacturer of classical vector processors has survived

who, however, have set a landmark in HPC with the installation of the Earth Simulator (based on NEC SX6 vector technology).

There is an intense discussion about future architectural directions for HPC systems. A substantial argument for increasing efforts to develop “tailored” HPC systems like vector computers is the fact that only the Earth Simulator achieves sustained performance numbers of several TFlop/s for a broad range of large-scale applications [5, 6, 7]. One potential competitor to the Earth Simulator is the CRAY X1 series which pioneers a new class of vector computers and has been very successful with 10 TOP500 installations in its first year of commercial availability.

From a user’s perspective, the above discussion is of minor interest. Their main concern is certainly application performance. The aim of our report is thus to establish a comprehensive understanding of performance characteristics for computer systems which are currently used or which we expect to be used in the near future for scientific simulation. To demonstrate the diverse requirements in these simulations we have chosen three application programs as well as a kernel benchmark. We consider a wide range of commodity processor architectures (IBM Power4, Intel Itanium 2, Intel Xeon, AMD Opteron) and system configurations (clusters vs. SMP) as well as “tailored” HPC systems (CRAY X1, NEC SX6).

In Sect. 2 we briefly introduce the architectural concepts and performance numbers of the systems used in our performance evaluations. Sects. 3 and 4 are dedicated to CFD applications as they are currently used in several KONWIHR [8, 9, 10, 11] projects on high-end systems. First we present a kernel benchmark, representing the performance of finite-volume codes and demonstrate its efficient parallelisation on shared memory systems. Second, a parallel 3D Lattice Boltzmann code is chosen as an example for a large scale CFD application. Sect. 5 deals with a Monte Carlo simulation of the 3D Ising model. Basic optimisation strategies for this type of code are discussed and the most appropriate target architecture is identified. In Sect. 6 we give a brief update of the performance evaluation presented in [12] for the quantum chemistry code Turbomole. Neither the Ising Model code nor Turbomole are in any way suited for vector machines, so they were not benchmarked on X1 and SX6.

2 Architectural Specifications

In Table 1 we briefly sketch the most important single processor specifications of the architectures examined. COTS architectures in particular are offered with a wide variety of different frequencies and cache sizes. The configurations as presented in the first two groups of Table 1 are those which are common in scientific computing centres. Concerning the memory architecture of COTS systems we find a clear tendency towards on-chip caches which run at processor speed and provide high bandwidth as well as low latencies. The tailored

Table 1. Single processor specifications. Peak performance numbers (Peak), maximum bandwidth (MemBW) of the memory interface of the processor and the sizes of the various cache levels are given. The L3 cache of the IBM Power4 processor and the L2 cache for the CRAY X1 are off-chip caches, all other caches are on-chip.

Platform	Single CPU specifications				
	Peak GFlops/s	MemBW GB/s	L1-cache kB	L2-cache MB	L3-cache MB
Intel Xeon DP, 2.66 GHz	5.3	4.3	8	0.5	—
Intel Itanium 2, 1.3 GHz	5.2	6.4	16	0.25	3.0
IBM Power4, 1.3 GHz	5.2	6.9	32	1.440	32.0
AMD Opteron, 1.6 GHz	3.2	5.4	64	1.0	—
NEC SX6 (1 CPU)	8.0	32.0	—	—	—
CRAY X1 (1 MSP)	12.8	34.1	—	2.0	—

HPC systems in the third group of Table 1 incorporate different (simpler) memory hierarchies and achieve substantially higher single processor peak performance and memory bandwidth. Note that tailored systems are usually much better balanced than COTS systems with respect to the ratio of memory bandwidth to peak performance.

2.1 Intel Xeon DP

The server variant (Xeon) of the Intel Pentium4 processor is widely used in COTS clusters and is well known for its high clock frequency. In contrast to most 64-Bit processors the 32-Bit Xeon processor can only execute two double precision floating point operations (one multiply and one add instruction) per cycle. Note that using single precision data together with SSE2 instructions the peak performance of Xeon/Pentium4 processors is doubled (4 floating point instruction per cycle). The on-chip caches of the Xeon DP (dual-processor variant) can be accessed with high bandwidth (96 GByte/s for the 3.06 GHz processor) and low latencies (7 cycles) while data transfer from memory is limited to 4.3 GByte/s by the front-side bus frequency of 533 MHz. In standard dual-processor configurations, the CPUs have to share one bus, further reducing the available memory bandwidth per processor by a factor of two.

The benchmark results reported in this paper were obtained on a cluster of 82 dual-processor nodes (using the Intel 7501 chipset) connected via a CISCO 4503 GBit Ethernet switch and running Debian GNU/Linux 3.0 at the Scientific Computing Centre Erlangen (RRZE).

2.2 Intel Itanium 2

The Intel Itanium processor is a superscalar 64-Bit CPU using the Explicitly Parallel Instruction Computing (EPIC) paradigm. In contrast to classical RISC systems, instructions are loaded in bundles of three. Only a limited number of combinations among memory, integer and floating point instructions are allowed per bundle, and the compiler has to take care of that. More importantly, the compiler also specifies groups of independent instructions which may be executed in parallel. Groups and bundles are two concepts that are, in a sense, orthogonal to each other, i. e. although Itanium can issue two bundles per cycle, a group can span any number of machine instructions. Of course this concept does not require any out-of-order execution support but demands high quality compilers to identify instruction level parallelism at compile time. While the first incarnation, the Itanium 1, has failed to become successful, the Itanium 2 is much more promising because of significant improvements in bandwidths, overall balance and compiler technology. The available clock frequencies range from 0.9 GHz to 1.5 GHz and the on-chip L3 cache sizes from 1.5 MB to 6 MB. Two Multiply-Add units are fed by a large set of 128 floating point registers, which is another important difference to standard microprocessors which comprise typically 32 floating point registers. Floating point data items bypass the L1 cache and are stored in the on-chip L2 and L3 caches, which can be accessed with high bandwidth (4 load or 2 load/2 store operations per cycle) and low latencies (5-6 cycles for L2; 10-12 cycles for L3). A large number of Itanium 2 systems from different vendors are available today, ranging from HP single processor workstations (running HP-UX) to SGI Altix shared memory systems with 64 processors in a single system image (running Linux). The basic building blocks of most systems used in scientific computing are dual-way nodes (SGI Altix, HP rx2600) or four-way nodes (NEC TX7, Bull NovaScale, HP rx5670) sharing one bus with 6.4 GByte/s memory bandwidth.

The system of choice in our report is a 28 processor SGI Altix3700 system (1.3 GHz; 3 MB L3 cache) at RRZE running RedHat Linux with SGI enhancements (“ProPack”). To emphasise the drawbacks of limited bandwidth of four-way systems we also included some benchmarks on HP rx5670 systems at HLR Stuttgart and Hewlett-Packard.

2.3 IBM Power4

The IBM Power4 processor is a 64-Bit superscalar (8-way fetch, 5-way sustained complete) out-of-order RISC processor with a maximum frequency

of 1.7 GHz and two Multiply-Add units allowing for a peak performance of 6.8 GFlop/s. The basic difference to classical RISC systems is that two processors (cores) are placed on a single chip sharing the high bandwidth (> 100 GByte/s) on-chip L2 cache, off-chip L3 cache and one path to memory. If used in the IBM pSeries 690, four chips (eight processors) are placed on a Multi-Chip-Module (MCM) and can use a large interleaved L3 cache of 128 MB aggregated size. Although large in size, the L3 cache shows several drawbacks, e. g. long cache lines (512 Bytes), large latencies (up to 340 cycles [13]) and relatively low bandwidth (11.7 GByte/s for the 1.3 GHz CPU [14]). Moreover, the L3 cache line spans all four L3 caches of one MCM. If fully equipped, a 32-way IBM p690 node (1.3 GHz Power4) can offer an aggregate theoretical memory bandwidth of 110 GByte/s for read *and* 110 GByte/s for write operations.

The Power4 measurements reported in this paper were done on a single IBM p690 node (1.3 GHz Power4) at the Computing Centre Garching (RZG). Multi-node Power4 experiments were not performed since the new “Federation” switch was not available.

2.4 AMD Opteron

The AMD Opteron Processor is a 64-Bit enabled version of the well-known AMD Athlon design. Maintaining full IA32 compatibility, Opteron has architectural enhancements that provide a seamless transition to 64-bit software and at the same time improve overall system performance. These include:

- an integrated memory controller for dual-channel PC2700 DDR RAM, effectively eliminating the need for a separate northbridge chip and reducing memory latency
- enlarged register set (compared to IA32) with eight additional 64-bit GP registers and eight additional 128-bit SSE registers
- support for Intel’s SSE2 instruction set
- three on-chip HyperTransport links (3.2 GByte/s each direction) for coupling to I/O and other Opteron processors

The larger number of GP and FP registers reduces register pressure and enables more aggressive code optimisation strategies than previously possible with IA32 designs. In SMP environments Opteron processors have one path to memory per CPU due to the integrated memory controller. Consequently, the aggregated memory bandwidth scales with CPU count. Cache-coherent shared-memory nodes with up to four processors can be easily built using the on-chip HyperTransport links.

Opteron processors are available with 64 kB of L1 and 1 MByte of L2 cache. The L1 data cache has two 64-Bit ports for a peak bandwidth of 2 loads or stores per cycle. The unified L2 cache is designed as a so-called “victim cache”, receiving only cache lines that were evicted from L1. The core can sustainedly execute one FP add and one FP multiply instruction per clock, allowing for

a peak performance of 4 GFlop/s at the maximum clock frequency of 2 GHz. The maximum memory bandwidth per CPU is 5.4 GByte/s.

The benchmark results presented here have been measured at RRZE on a dual-Opteron workstation (1.6 GHz) with PC2100 memory modules. Thus the full potential of the memory interface could not be utilised. Another problem is posed by the fact that modern, standard-adhering and stable compilers are somewhat scarce for this CPU, especially for Fortran 90. One of the benchmarks described below (TRATS) could not be run in 64-bit mode for this reason.

2.5 NEC SX6

From a programmers' view the NEC SX6 is a traditional vector processor with 8-way replicated vector pipes running at 500 MHz. One Multiply-Add instruction per cycle can be executed by each arithmetic pipe delivering a peak performance of 8 GFlop/s. The memory bandwidth of 32 GByte/s allows for one load or store per Multiply-Add instruction. The processor contains 72 vector registers, each holding 256 (64-Bit) words. For non-vectorisable instructions, the SX6 contains a 500 MHz scalar processor with a peak performance of 1 GFlop/s. Since the vector processor is significantly more powerful than the scalar unit, it is useless to run non-vectorised applications on an SX6. Each SMP node comprises eight processors and provides a total memory bandwidth of 256 GByte/s, i. e. the aggregated single processor bandwidths can be saturated!

The benchmark results presented have been measured by NEC on a 24 node system at the German High Performance Computing Centre for Climate and Earth System Research (DKRZ).

2.6 CRAY X1

The basic building block of the CRAY X1 architecture is a *multi-streaming processor* (MSP) which one usually refers to as processor or CPU. The MSP itself comprises four processor chips, which each incorporate a superscalar processor (400 MHz; 16 KByte L1 cache) and a vector section. The vector section contains 32 vector registers of 64 elements each and a two-pipe processor capable of executing four double precision (eight single precision!) floating point operations and two memory operations. Running at a clock speed of 800 MHz one MSP can thus perform up to 16 double precision floating point operations (12.8 GFlop/s) and issue 8 memory operations (51.2 GByte/s) per cycle. Note that the ratio of issued memory operations per issued Multiply-Add instruction is the same as for the NEC SX6 processor, but the memory interface of the MSP only delivers 34.1 GByte/s bandwidth and thus can not saturate the issued load instructions.

At first glance long vectorised loops are, of course, the preferred programming style because the MSP unit can operate in a way similar to classical

wide-pipe vector processors such as the NEC SX6. However, it is also possible that each vector section takes a whole (much shorter) inner loop iteration of a nested loop, avoiding the rather long start-up times for wide-pipe vector processors. Since vectorisation is the recommended programming model and the vector instruction set allows to bypass L2 cache we do not comment on the cache features. An SMP node comprises four MSPs and can in principle saturate the aggregated MSP bandwidths. Each SMP node is connected to the network with a bandwidth of 100 GByte/s.

The benchmark results presented have been provided by Cray.

3 OpenMP parallelisation of a Strongly Implicit Solver

3.1 Introduction

The Strongly Implicit Solver according to Stone [15] (SIP-solver) is a solver for sets of linear equations $A\mathbf{x} = \mathbf{b}$ and is especially suitable for systems resulting from a finite volume discretisation of partial differential equations like heat/fluid flow or diffusion. It is widely used in fluid mechanics and a relevant part of many CFD Codes.

The SIP-solver is based on an incomplete LU-factorisation of A and successive forward and backward substitution steps to minimise the residual. Unfortunately the algorithm carries data dependencies. Considering three dimensions, during the forward substitution step in order to update a point (i, j, k) the updated values of neighbour points $(i - 1, j, k)$, $(i, j - 1, k)$ and $(i, j, k - 1)$ are required. Therefore, these points have to be calculated in advance. The following code snippet shows the forward substitution:

```

1  do k=2,kMaxM
2      do j=2,jMaxM
3          do i=2,iMaxM
4              RES(i,j,k)=(RES(i,j,k)-LB(i,j,k)*RES(i,j,k-1)-
5                  *           LW(i,j,k)*RES(i-1,j,k)-LS(i,j,k)*
6                  *           RES(i,j-1,k))*LP(i,j,k)
7          enddo
8      enddo
9  enddo

```

As a result of the dependencies there is no simple parallelisation scheme possible in the first place. Currently a conventional version with 3D-indexing and a hyperplane version especially suitable for vector machines are in common use. The latter allows for vectorisation/parallelisation within a hyperplane $i + j + k = \text{const}$ (described in [16, 17]).

In this section, a different approach to parallelisation similar to [18] is proposed using *pipeline parallel processing*. This technique is adapted to cache based architectures and should offer superior performance compared to the hyperplane version on those machines.

3.2 Pipeline Parallel Processing

The middle loop (i.e. j in Fig. 1) is divided into N chunks, while N denotes the number of CPUs available. For the sake of simplicity we restrict our con-

Fig. 1. Schematic view of pipeline parallel processing in 2D with 4 processors available. As a result of the data dependencies, only certain processors may calculate during the *wind-up* and the *wind-down* phases while all others have to idle.

siderations to the 2D case depicted in Fig. 1. In a 3D system, an additional inner loop over i is required (a chunk can be thought of as a long flat bar).

As shown in Fig. 1, when the first processor (CPU 0) starts calculating its first chunk, all others have to idle since they need the updated boundary values from CPU 0. By the time CPU 0 has finished, the next processor can join in and start working on its chunk so that now both CPU 0 and CPU 1 are calculating. This procedure is continued until all available processors have started working and the so called *wind-up* phase is finished.

Proceeding through its slice, CPU 0 will finally reach the end of the system and stop calculating. Now it comes to an analogous *wind-down* phase until CPU 3 has also finished its last chunk. The negative impact on performance caused by the wind-up and wind-down-phase should be negligible for a sufficiently large lattice. Different stages within one sweep are illustrated in Fig. 1.

This algorithm can be implemented using shared memory parallelisation via OpenMP. As an example, the forward substitution for a 3D case is shown in the following listing:

```

1  !$omp parallel private(i,j,k,l,threadID)
2  do l=2,kMax+numThreads-2
3
4      threadID = OMP_GET_THREAD_NUM()
5      k = l - threadID
6
7      if ((k.ge.2).and.(k.le.kMaxM)) then
8
9          do j=jStart(threadID),jEnd(threadID)
10             do i=2,iMaxM
11                 RES(i,j,k)=(RES(i,j,k)-LB(i,j,k)*RES(i,j,k-1)-
12 *                     LW(i,j,k)*RES(i-1,j,k)-LS(i,j,k)*
13 *                     RES(i,j-1,k))*LP(i,j,k)
14             enddo
15         enddo
16
17     endif
18 !$omp barrier

```

Fig. 2. Performance in MFlop/s for all benchmarked architectures with system sizes of 91^3 and 201^3 (left and right hand side respectively). The numbers for 2 and 4 CPUs denote the parallel speed-up.

```

19  enddo
20  !$omp end parallel

```

The parallelisation must be done by hand to ensure correct synchronisation during wind-up and wind-down phase. The arrays `jStart` and `jEnd` (line 9) store the absolute indices of start and end points of particular chunks each OpenMP thread has to work on. Every thread is given a unique value k depending on its own thread ID. Please note that due to the wind-up and wind-down phase the k loop has to be extended. Within the given implementation the chunk size in direction k is equal to one.

The essential point is the barrier in line 18, which ensures correct synchronisation of all OpenMP threads involved. Without that barrier a thread would continue working, not knowing whether all necessary neighbour points for its next chunk have already been updated.

On ccNUMA-systems (e.g. SGI Altix or Origin series), initialisation of data is also an important performance issue: A common approach in 3D (index order (k, j, i)) would be to parallelise the outer most loop k . However, first touch policy necessitates parallelisation of the j -loop (middle loop) also during the initialisation step. Otherwise, not all chunks a processor has to work on would reside at the processors local memory (for the sake of simplicity we assume Round Robin memory allocation and one path to memory for each CPU). A negative impact on scalability has to be expected and can indeed be observed in this case.

3.3 Benchmarking

A benchmark kernel called “SipBench” has been implemented in Fortran77 in order to carry out performance tests on a couple of current architectures. It is based on a program written by [19]. System sizes of 91^3 and 201^3 were chosen with a memory consumption of roughly 100 and 1000 MByte respectively. The standard compiler flags for highest optimisation were used.

Figure 2 shows the performance in MFlop/s for runs with up to four processors. The numbers denote the parallel speed-up. Considering single-CPU performance for the pipeline parallel version, the SGI Altix clearly performs best and yields more than 700 MFlop/s on one of its Itanium 2 processors. One Power4 CPU attains about 400 MFlop/s whereas both Xeon and Opteron run at roughly 250 MFlop/s. However, vector machines like the Cray X1 still achieve outstanding performance compared to the other benchmarked systems. The lines in Fig. 2 denote the performance of the hyperplane version.

Increasing the number of threads to 2 leads to a reasonable speed-up in case of the SGI and the IBM machine. The latter one should benefit from the large aggregated L3-cache of 128 MByte. The Opteron also shows quite a good speed-up since each CPU has a dedicated connection to main memory. In contrast, two CPUs of the Xeon node share a bandwidth of 4.3 GByte/s to memory, which results only in a minor speed-up.

Going up to 4 processors both Altix and p690 still show reasonable scaling. Increasing the problem size by one order of magnitude does not change the qualitative behaviour. As expected, p690 performance drops quite a bit (reduced cache effects) while both Opteron and Xeon show almost the same performance. The Altix manages larger loop length very well so that the performance goes up to roughly 1700 MFlop/s on 4 CPUs. However, for the small system size (91^3) even 4 Itanium 2 CPUs are no match for one Cray X1 MSP.

To emphasise the correlation between our benchmark kernel and complex CFD applications such as LESOCC¹ from LSTM² we give relative performance numbers compared to Xeon in Fig. 3.

Fig. 3. Single CPU performance relative to Xeon for the benchmark kernel and LESOCC.

A great deal of the code's performance is contributed by the SIP-solver. Therefore it is not surprising to find a correspondence considering the fastest architecture for LESOCC and SipBench.

4 Benchmarking of a Lattice Boltzmann CFD application

4.1 Introduction

The Lattice-Boltzmann Method (LBM) has evolved into a promising alternative to conventional methods in fluid dynamics. Whereas the latter are based on a discretisation of macroscopic differential equations, the former follows a *bottom-up* approach by describing microscopic particle motion [20, 21].

The method can be considered as a cellular automaton. The computational domain is divided into orthogonal cells which contain a certain number of so-called distribution functions (see Fig. 4).

In order to calculate the next time step all lattice cells are updated by shifting and modifying the distribution functions according to given rules. For the so-called relaxation and propagation step in the standard algorithm,

¹ Large Eddy Simulation On Curvilinear Coordinates

² Lehrstuhl für Strömungsmechanik, Prof. Durst, FAU Erlangen-Nürnberg

Fig. 4. Distribution functions in a cell for a D3Q19 LBM model (3 dimensions, 19 distribution functions). The neighbouring cells are labelled according to compass notation while T refers to “top” and B to “bottom”.

a cell only needs information from its neighbouring cells (19 in case of a D3Q19 model in Fig. 4). The updating rules depend on the physical model and the kind of cell, i. e. fluid or boundary in a simple case. Figure 5 illustrates the propagation of distribution functions. In case of a boundary cell a *bounce back* scheme is used, which means that the distribution functions are reflected at the wall.

Fig. 5. Illustration of propagation of distribution functions in a lattice from time step t to $t + 1$. The rules for propagation depend on the model and the kind of cell (i. e. fluid or boundary cells). In case of a boundary cell a *bounce back* scheme is used (distribution functions are reflected at the wall).

The LBM is characterised by a computationally intensive loop body and excessive memory usage.

4.2 Benchmarking

A MPI-parallel Fortran90 code called TRATS from LSTM³ was chosen for benchmarking. It is a large scale CFD application also used intensely on the Hitachi SR8000 which implements turbulent flow (D3Q19 model) in a 3D channel. For benchmark purposes, we chose two configurations which fit into main memory of most single CPU systems. The number of cells used was $128 \times 129 \times 128$ and $256 \times 129 \times 128$, which accounts for a memory consumption of 700 and 1400 MByte respectively. The code is available in a vectorised and a cache optimised version.

An appropriate performance metric is MLUPS, which is an abbreviation for *Mega Lattice site Updates per Second*. Profiling on SGI Origin 3400 and SGI Altix systems showed that 1 GFlop/s is roughly equal to 5.5 MLUPS for the TRATS code.

Figure 6 shows single CPU performance on the benchmark architectures. The Power4, Xeon and Opteron processors achieve values around 2 while

³ Lehrstuhl für Strömungsmechanik, Prof. Durst, FAU Erlangen-Nürnberg

Fig. 6. Single CPU performance for TRATS. The system size is $256 \times 129 \times 128$.

Itanium 2 is capable of more than 5 MLUPS. This is not only a consequence of the high memory bandwidth but can also be attributed to the large register set, which is very useful with the large loop body of the LBM.

Performance numbers for shared memory systems of up to four Itanium 2 processors are shown in Fig. 7. It can be seen that neither increasing CPU

Fig. 7. Performance numbers for shared memory systems of up to four Itanium 2 processors.

frequency by 30 % ($1.0 \rightarrow 1.3$ GHz) nor doubling L3 cache size does result in a substantial performance gain for 4 CPUs. On the other hand we get reasonable speed-up with two processors even if they share a single memory bus. Going up to four CPUs reveals that two processors already saturate the memory bus so that it does not pay off to increase the number of processors without increasing the available bandwidth to memory too. In contrast, the SGI Altix has one way to memory available for each pair of CPUs, which results in good scaling from two to four CPUs.

Considering speed-up and scalability one must distinguish *strong* from *weak* scaling. The former means that the total problem size is constant while in the latter case the problem size per CPU is fixed. Considering the benchmark code, parallelisation is achieved by domain decomposition of the 3D channel. In case of strong scaling, the more CPUs are used the smaller the domains get that are assigned to a single CPU. This results in a growing negative impact on performance by communication overhead. However, if the problem size is scaled accordingly, the domain size for each CPU and the ratio of computation to communication remains the same (weak scaling). In terms of efficiency, which can be defined as

$$\text{Efficiency} = \frac{\text{Performance on } N \text{ CPUs}}{\text{Performance on } 1 \text{ CPU} \times N}, \quad (1)$$

weak scaling achieves higher values (“better scaling”) for TRATS (see Fig. 8). Figure 8 also demonstrates the drawbacks of COTS GBit clusters when latency and bandwidth of communication is an issue.

Besides scaling concerns, similar to the single CPU case the Itanium 2 based SGI Altix clearly outperforms the IBM p690 by a factor of roughly two. To achieve the performance of 1 NEC SX6 CPU or Cray X1 MSP, 8 Itanium 2 or 16 and more Power4 processors are required.

To demonstrate the advantages of tailored HPC systems for large scale applications such as TRATS, we give performance numbers for standard con-

Fig. 8. Weak and strong scaling (upper and lower picture respectively) for TRATS on a COTS GBit cluster, an SGI Altix and an IBM p690 node. In case of weak scaling a system size of $128 \times 129 \times 128$ was used while strong scaling was measured for $256 \times 129 \times 128$ per CPU. Numbers for vector machines NEC SX6 and Cray X1 are given for reference.

figurations of NEC SX6 and Cray X1 architectures in Table. 2. It is obvious

Table 2. Performance in MLUPS and efficiency (Eff.) for Trats on NEC SX6 and Cray X1 (weak scaling).

NEC SX 6			Cray X1		
CPUs	MLUPS	Eff.	CPUs	MLUPS	Eff.
1	32	1	1	25	1
2	63	0.98	28	653	0.93
4	125	0.98	64	1575	0.98
8	247	0.96	128	3124	0.98
16	480	0.94	250	5461	0.87
32	921	0.90			

that the benchmark code is well suited for these kinds of architecture and that very high levels of efficiency can be reached (0.87 for 250 CPUs of Cray X1!).

Regarding the performance equivalent for MLUPS in case of TRATS, it is worth mentioning that 250 Cray X1 CPUs achieve a TFlop/s of sustained performance:

$$5461 \text{ MLUPS} \approx 1 \text{ TFlop/s.}$$

Assuming perfect scalability, more than 3000 IBM Power4 or 1000 Intel Itanium 2 processors would be required to hit that mark. However, perfect scalability of both the problem and the architectures would be rather questionable in that case.

5 Benchmarking of a Monte-Carlo simulation tool

5.1 Description of the program

The program Isingtest is used in theoretical solid state physics⁴. It is a Monte Carlo simulation of the three-dimensional Ising model in non-equilibrium [22]. The simulation is carried out on a lattice with periodic boundary conditions.

⁴ Lehrstuhl für Theoretische Physik I, Prof. Dr. Hüller, FAU-Erlangen-Nürnberg

The update of a spin is accomplished as follows. At first a lattice site is randomly chosen. The spin at this site is flipped with a certain probability which depends only on the values of the nearest neighbour spins. This update step is repeated many times during the simulation.

The Monte Carlo program is parallelised with MPI. Thereby every process works with its own spin configuration. The trivial parallelisation just increases the statistical accuracy and is thus of minor concern here. Also the quality of the network is not important for the performance of the program. Therefore, the optimisations concern only the serial aspects of the program and performance measurements on a single CPU are sufficient.

5.2 Optimisations

Profiling has shown the hot spots of the program and serves as a basis for the following optimisation approaches:

1. The `tanh()` function has to be calculated for the determination of the spin update probability. In the program the argument of the function can take on only 7 different values. By tabulating those values, the slow function call can be replaced by a fast array access.
2. In the original version of the program the spin variables are declared as 4-byte integers. Because a spin can only take on the values +1 and -1, the declaration was changed to 1-byte integer (`integer*1`). This has reduced the memory requirements for the program's working set from 1120 kByte to 280 kByte and has led to a large performance boost, because the data now fits into the caches of modern CPUs. Please note that we expect substantial performance increase especially for the Xeon processor with an L2 cache size of 512 kByte.
3. The memory requirements of the program can be reduced even further by using a single bit for the storage of one spin. However the Fortran routines for bit manipulation have not shown the expected performance.
4. Successive updates of two spins are independent if the spins are not nearest neighbours. A pipelined version of the program was developed that can execute arithmetic instructions and load instructions simultaneously.

In practice it turned out that on most platforms only optimisations 1 and 2 increase performance (see Fig. 9 for the example of a Xeon processor). The

Fig. 9. Influence of the optimisations described in the text on the performance of Isingtest measured on a Xeon with 2.4 GHz.

effect of these two optimisations on the performance of the program is shown in Fig. 10 for different architectures.

Fig. 10. Relative performance of Isingtest on different processors. The baseline is set by the original program on a Xeon with 2.4 GHz.

Our optimisations pay off on all architectures with a performance gain between factors 2 and 5, and since the numerical core fits into on-chip caches the performance scales with processor frequency. Although the latest Itanium 2 and Power4 processors achieve the best performance numbers, they are only 5 to 10 percent ahead of comparable Xeon CPUs. If we also consider the low communication requirements of the code, a Gigabit Xeon cluster is the perfect target architecture for this application due to its unmatched price/performance ratio.

During the first 9 months after installation of the Xeon Cluster at RRZE, the optimised program has consumed about 250,000 hours of CPU time.

6 Performance of TURBOMOLE

TURBOMOLE is a widely used program for numerical research in quantum chemistry. The aim of this section is to provide a comparison of TURBOMOLE performance on different computer architectures. For this purpose the wall clock time of the “fe5f” benchmark has been measured on different processors and compilers. In fe5f, TURBOMOLE calculates properties of $\text{Fe}(\text{N}_2)'\text{N}_\text{H}\text{S}_4\text{CF}_3'$ with $'\text{N}_\text{H}\text{S}_4\text{CF}_3' - \text{H}_2 = 2,2'$ -bis(2-mercapto-1,2-bis-trifluoromethylethenyl-thio)diethylamine (see Fig. 11). This molecule structure has been studied by Reiher et al. [23]. Previous investigations [12] have shown that the MPI-parallelised TURBOMOLE program scales reasonably well for this benchmark. Thus we only measure the performance of the serial program which is shown in Fig. 12. They indicate that a Xeon processor is the optimal architecture for this program.

Fig. 11. Molecule used for the fe5f benchmark. For a reproduction of this figure in colour, see Fig. 13.

Fig. 12. Performance of TURBOMOLE (version 5.4). The values are related to the runtime on Itanium 2 with 1.3 GHz (1204 s).

7 Conclusions

The wide variety of scientific simulations requires a diverse spectrum of computational resources, but neither tailored HPC systems nor COTS systems alone can provide cost-effective and usable solutions to the whole HPC community. Program development starts at the desktop; thus “entry-level” systems should be as close as possible to the desktop environment. These systems are used by projects with relatively low processor and/or network performance requirements. Potential configurations are IA32-clusters and multi-processor systems running Linux and offering an environment, e.g. compilers and debuggers, which is known from desktop.

Increasing demands force users to continuously adapt their applications to appropriate programming models as well as the latest and most powerful computer architectures. The scientific success of “large-scale” projects usually requires high sustainable performance when using substantial parts of a supercomputer to solve a single problem. In consequence, this community must provide reasonable input and benchmarks to computing centres when new HPC systems are tailored to solve problems which can not be tackled elsewhere.

The applications discussed in this paper are a only a subset of the wide variety of scientific applications but they represent the requirements of the user projects supported by the KONWIHR project cxHPC. We have shown that Quantum Chemistry and (Quantum) Monte-Carlo applications highly benefit from cost-effective COTS solutions such as the Xeon cluster at RRZE.

For CFD codes we have demonstrated that the “tailored” architectures CRAY X1 and NEC SX6 provide comparable single processor performance. Although different in architecture, vectorisation is the optimisation strategy of choice when using the CRAY X1 system. When compared to latest cache-based microprocessors the CRAY X1 and NEC SX6 are still ahead by a factor of four to eight. The new Itanium 2 processor performs remarkably well on CFD applications and provides a performance gain of 50–100 % compared to other microprocessors such as IBM Power4, Intel Xeon or AMD Opteron. Concerning the memory subsystem we have found that it does not pay off to use more than two processors per memory path.

A shared memory parallelisation of a benchmark kernel (SIP-solver), representing the performance characteristics of finite-volume algorithms, was implemented in order achieve high performance on cache-based SMP architectures. Considering absolute performance the SGI Altix architecture is the most suitable of all tested machines for this kind of code, whereas the IBM p690 gets good marks for scalability.

As an example of a large scale CFD application we have chosen the LBM code TRATS. Only the tailored HPC systems are able to achieve a sustained performance of more than 1 TFlop/s with reasonable processor counts (e.g. 256 processors). Comparable performance numbers would require thousands

of cache-based microprocessors connected by a high-speed interconnect, assuming problem and network is scalable perfectly to that number of CPUs.

In summary, we have demonstrated that the “computer pyramid” should be built by COTS clusters at the base and tailored HPC systems at the top. The gap in between may be closed by cluster or shared memory systems which combine COTS technology (e.g. Itanium 2 processors) with tailored components (e.g. high-speed interconnects like Federation or NUMALink technology). Scientists using this environment must be supported by the computing centres at all levels to ensure efficient use of the resources. Besides support in optimisation and parallelisation, other important tasks of computing centres are to guide users to the most appropriate computer architectures and to evaluate new systems with the background of potential applications.

Acknowledgement. We would like to thank the HPC teams at LRZ Munich and HLR Stuttgart for ongoing support and fruitful discussions.

We thank W. Oed, M. Wierse (CRAY), A. Bömelburg (IBM), H. Cornelius (Intel), T. Schoenemeyer (NEC), R. Wolff and R. Vogelsang (SGI) for providing benchmark data and helpful information.

Special thanks go to P. Lammers and T. Zeiser for collaboration on the LBM work and to M. Breuer for contributions to the matter of SipBench/LESOCC.

This work was supported by the Competence Network for Scientific High Performance Computing in Bavaria (KONWIHR) through project cxHPC.

References

1. TOP500 list November 2003 available at <http://www.top500.org/>
2. L. Oliker, A. Canning, J. Carter, J. Shalf, D. Skinner, S. Ethier, R. Biswas, J. Djomehri, and R. V. d. Wijngaart, *Evaluation of Cache-based Super-scalar and Cacheless Vector Architectures for Scientific Computations*, in *Proc. SC2003*, CD-ROM, 2003.
3. The ASCI program: <http://www.llnl.gov/asci/>
4. A. J. van der Steen and J. Dongarra, *Overview of Recent Supercomputers (thirteenth edition)*, available at <http://www.phys.uu.nl/steen/web03/overview.html>
5. H. Sakagami, H. Murai, Y. Seo, and M. Yokokawa. *14.9 TFLOPS three-dimensional fluid simulation for fusion science with HPF on the Earth Simulator*, in *Proc. SC2002*, CD-ROM, 2002
6. S. Shingu, et. al., *A 26.58 Tflops global atmospheric simulation with the spectral transform method on the Earth Simulator*, in *Proc. SC2002*, CD-ROM, 2002.
7. D. Komatitsch, S. Tsuboi, C. Ji, and J. Tromp, *A 14.6 billion degrees of freedom, 5 teraflops, 2.5 terabyte earthquake simulation on the Earth Simulator*, in *Proc. SC2003*, CD-ROM, 2003.
8. The KONWIHR project: <http://konwihhr.in.tum.de/>
9. M. Glück, M. Breuer, F. Durst, A. Halfmann, and E. Rank in S. Wagner et al. (Eds.): *High Performance Computing in Science and Engineering Munich 2002*, pp. 11–20, 2003, Springer-Verlag 2003

- Numerical Prediction of Deformations and Oscillations of Wind-Exposed Structures*
10. M. Breuer, N. Jovicic, and K. Mazaev in S. Wagner et al. (Eds.): High Performance Computing in Science and Engineering Munich 2002, pp. 11–20, Springer-Verlag, 2003
Large-Eddy and Detached-Eddy Simulation of the Flow Around High-Lift Configurations
 11. P. Lammers, K. Beronov, G. Brenner, and F. Durst in S. Wagner et al. (Eds.): High Performance Computing in Science and Engineering Munich 2002, pp. 11–20, 2003, Springer-Verlag
Direct Simulation with the Lattice Boltzmann Code BEST of Developed Turbulence in Channel Flows
 12. L. Palm and F. Brechtefeld in S. Wagner et al. (Eds.): High Performance Computing in Science and Engineering Munich 2002, pp. 11–20, 2003, Springer-Verlag
A User-Oriented Set of Quantum Chemical Benchmarks
 13. Behling S., Bell R., Farrell P., Holthoff H., O’Connell F., Weir W., *The Power4 Processor Introduction and Tuning Guide*, IBM (2001), www.ibm.com/redbooks/
 14. Krämer F., IBM, private communication.
 15. H.L. Stone, *Iterative solution of implicit approximations of multidimensional partial differential equations*, SIAM J. Numerical Analysis, 5 (5), 1968
 16. G. Hager, F. Deserno and G. Wellein: *Pseudo-Vectorization and RISC Optimization Techniques for the Hitachi SR8000 Architecture*, High Performance Computing in Science and Engineering Munich 2002, Springer Verlag Berlin Heidelberg, 2003, ISBN 3-540-00474-2
 17. J. H. Ferziger, M. Perić: *Computational Methods for Fluid Dynamics*. Springer Verlag, 1999
 18. J. Reeve, A. Scurr and J. Merlin, *Parallel Versions of Stone’s Strongly Implicit Algorithm*, Concurrency Practice and Experience 13, 2001
 19. Basic code examples for the algorithms in [17] can be obtained from <ftp://ftp.springer.de/pub/technik/peric/>
 20. D. A. Wolf-Gladrow, *Lattice-Gas Cellular Automata and Lattice-Boltzmann Models*, Springer Verlag, 2000, ISBN 3-540-66973-6
 21. S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*, Numerical Mathematics and Scientific Computation, Oxford University Press, 2001
 22. M. Henkel, M. Pleimling, C. Godrèche and J.-M. Luck, *Aging, Phase Ordering, and Conformal Invariance*, in Phys. Rev. Lett. **87**, 265701 (2001).
 23. M. Reiher, O. Salomon, D. Sellmann, B.A. Hess (2001): *Dinuclear Diazene Iron and Ruthenium Complexes as Models for Studying Nitrogenase Activity*, Chem. Eur. J. **23**, 5195–5202

Fig. 13. [F. Deserno, G. Hager, F. Brechtefeld, G. Wellein] Molecule used for the fe5f benchmark