Optimizing Performance on Modern HPC Systems: Learning From Simple Kernel Benchmarks

G. Hager[1], T. Zeiser[1], J. Treibig[2], and G. Wellein[1]

[1] Regionales Rechenzentrum Erlangen (RRZE), Martensstr. 1, 91058 Erlangen, Germany
[2] Lehrstuhl für Systemsimulation (LSS), Friedrich-Alexander-Universität Erlangen-Nürnberg, Cauerstr. 6, 91058 Erlangen, Germany

Summary. We discuss basic optimization and parallelization strategies for current cache-based microprocessors (Intel Itanium2, Intel Netburst and AMD64 variants) in single-CPU and shared memory environments. Using selected kernel benchmarks representing data intensive applications we focus on the effective bandwidths attainable, which is still suboptimal using current compilers. We stress the need for a subtle OpenMP implementation even for simple benchmark programs, to exploit the high aggregate memory bandwidth available nowadays on ccNUMA systems. If the quality of main memory access is the measure, classical vector systems such as the NEC SX6+ are still a class of their own and are able to sustain the performance level of in-cache operations of modern microprocessors even with arbitrarily large data sets.

1 Introduction

Intel architectures and their relatives currently tend to stand in the focus of public attention because of their allegedly superior price/performance ratio. Many clusters and supercomputers are built from commodity components, and the majority of compute cycles is spent on Itanium2, IA32-compatible and similar processors. The availability of high-quality compilers and tools makes it easy for scientists to port application codes and make them run with acceptable performance. Although compilers get more and more intelligent, user intervention is vital in many cases in order to get optimal performance. In this context it is an important task to identify easy-to-use optimization guidelines that can be applied to a large class of codes. We use simple kernel benchmarks that serve both to evaluate the performance characteristrics of the hardware and to remedy bottlenecks in complex serial as well as shared-memory parallel applications. The focus of this report is on the bandwidth problem. Modern micorprocessors provide extremely fast on-chip caches with sizes of up to several megabytes. However, the efficient use of these caches requires complex optimization techniques and may depend on parameters which are not under

the control of most programmers, e. g. even for in-cache accesses performance can drop by a factor of up to 20 if data access is badly scheduled [1]. Although there has been some progress in technology, main memory bandwidth is still far away from being adequate to the compute power of modern CPUs. The current Intel Xeon processors, for instance, deliver less than one tenth of a double precision operand from main memory for each floating point operation available. The use of cachelines makes things even worse, if nonregular access patterns occur. Providing high-quality main memory access has always been the strength of classical vector processors such as the NEC series and therefore these "dinosaurs" of HPC still provide unique single processor performance for many scientific and technical applications [2, 3, 4, 5]. Since performance characteristics as well as optization techiques are well understood for the vector architectures, we have chosen the NEC SX6+ to serve as the performance yardstick for the microprocessors under consideration here.

This paper is organized as follows. The hardware and software characteristics of systems used in our study are summarized in Section 2. In Section 3, the popular vector triad benchmark is used to compare the ability of different architectures to use the system resources to their full extent. It is shown that, even with such a seemingly simple code, vast fluctuations in performance can be expected, depending on fine details of hardware and software environment as well as code implementation. We first discuss the gap between theoretical performance numbers of the hardware and the effective performance measured. Then we introduce an approach based on the use of assembler instructions to optimize single processor memory performance on IA32 and x86-64 processors. In Section 4, performance effects of shared memory parallelization on ccNUMA systems like SGI Altix or Opteron-based nodes are investigated. It is shown that an overly naive approach can lead to disastrous loss of performance on ccNUMA architectures. In Section 5 we finally give a brief summary and try to restate the most important lessons learned.

2 Architectural Specifications

In Table 1 the most important single processor specifications of the architectures examined are sketched. The on-chip caches of current microprocessors run at processor speed, providing high bandwidth and low latency. The NEC vector system implements a different memory hierarchy and achieves substantially higher single processor peak performance and memory bandwidth. Note that the vector processor has the best balance with respect to the ratio of memory bandwidth to peak performance.

Intel Netburst and AMD64

The Intel Pentium4, codenamed "Prescott" in its current incarnation, and the AMD Athlon64/opteron processors used in our benchmarks are the latest versions of the well-known Intel "Netburst" and AMD64 architectures, respectively. The Athlon64 and Opteron processors have the additional advantage of a fully downwards-compatible 64-bit extension (also available in Intel's recent Xeon CPUs). Both CPUs are capable of performing a maximum of two double precision floating point (FP) operations, one multiply and one add, per cycle. Additionally, AMD

Table 1. Single processor specifications. Peak performance (Peak), maximum bandwidth of the memory interface (MemBW) and sizes of the various cache levels (L1,L2,L3) are given. All caches are on-chip, providing low latencies.

Platform	**Single CPU specifications**				
	Peak GFlop/s	MemBW GB/s	L1 [kB]	L2 [MB]	L3 [MB]
Intel Pentium4/Prescott (3.2 GHz)	6.4	6.4	16	1.0	–
AMD Athlon64 (2.4 GHz)	4.4	6.4	64	1.0	–
Intel Itanium 2 (1.3 GHz)	5.2	6.4	16	0.25	3.0
NEC SX6+ (565 MHz)	9.0	36.0	–	–	–

CPUs have an on-chip memory controller which reduces memory latency by eliminating the need for a separate northbridge.

The Prescott and Athlon64 benchmark results presented in sections 3.1 and 3.2 have been measured at LSS on single-processor workstations. For the OpenMP benchmarks in section 4 a 4-way Opteron server with an aggregate memory bandwidth of 25.6 GByte/s (=4×6.4 GByte/s) was used. Both systems run Debian Linux and the benchmarks were compiled using the Intel IA32 Fortran Compiler in version 8.1-023.

Intel Itanium 2

The Intel Itanium 2 processor is a superscalar 64-bit CPU implementing the Explicitly Parallel Instruction Computing (EPIC) paradigm. The Itanium concept does not require any out-of-order execution hardware support but relies on heavy instruction level parallelism, putting high demands on compilers. Today clock frequencies of up to 1.6 GHz and on-chip L3 cache sizes from 1.5 to 6 MB are available. Please note that floating point data is transfered directly between L2 cache and registers, bypassing the L1 cache. Two multiply-add units are fed by a large set of 128 FP registers, which is another important difference to standard microprocessors with typically 32 FP registers. The basic building blocks of systems used in scientific computing are two-way nodes (e. g. SGI Altix, HP rx2600) sharing one bus with 6.4 GByte/s memory bandwidth.

The system of choice in our report is an SGI Altix 3700 with 28 Itanium2 processors (1.3 GHz/3 MB L3 cache) running RedHat Linux with SGI proprietary enhancements (ProPack). Unless otherwise noted, the Intel Itanium Fortran Compiler in version 8.1-021 was used.

NEC SX6+

From a programmer's view the NEC SX6+ is a traditional vector processor with 8-track vector pipes running at 565 MHz. One multiply and one add operation per cycle can be executed by the arithmetic pipes, delivering a peak performance of 9 GFlop/s. The memory bandwidth of 36 GByte/s allows for one load or store per multiply-add operation. 64 vector registers, each holding 256 64-bit words, are available. A SMP node comprises eight processors and provides a total memory bandwidth of 289 GByte/s, i. e. the aggregated single processor bandwidths can be saturated. Vectorization of application codes is a must on this system, because scalar CPU performance is non-competitive.

The benchmark results presented in this paper were measured on a NEC SX6+ at the Höchstleistungsrechenzentrum Stuttgart (HLRS).

3 Serial vector triad

One of the most simple benchmark codes that is widely used to fathom the theoretical capabilities of a system is the vector triad [6]. In Fortran language, the elementary operation `A(:)=B(:)+C(:)*D(:)` is carried out inside a recurrence loop that serves to exploit cache reuse for small array sizes:

```
do R=1, NITER
   do I=1,N
     A(I) = B(I) + C(I) * D(I)
   enddo
enddo
```

Appropriate time measurement is taken care of and the compiler is prevented from interchanging both loops which conflicts with the intention of the benchmark. Customarily, this benchmark is run for different loop lengths `N`, and `NITER` is chosen large enough so that startup effects are negligible. Parallelization with OpenMP directives appears straightforward, but we will first consider the serial version.

The single processor characteristics are presented in Fig. 1 for different architectures using effective bandwidth (assuming a transfer of 4×8 Byte $= 32$ Byte per iteration) as a performance measure. The performance characteristics reflect the basic memory hierarchy: Transitions between cache levels with different access speeds can be clearly identified by sharp drops in bandwidth. For all cache-based microprocessors we find a pipeline start-up effect at small loop length, two "transitions" at intermediate loop length and a maximum performance which scales roughly with peak performance. A totally different picture emerges on the vector system, where bandwidth saturates at intermediate loop lengths, when vector pipeline start-up penalties become negligible. Most notably, the vector system is able to sustain the in-cache performance of RISC/EPIC processors at arbitrarily large loop length, i. e. large data sets.

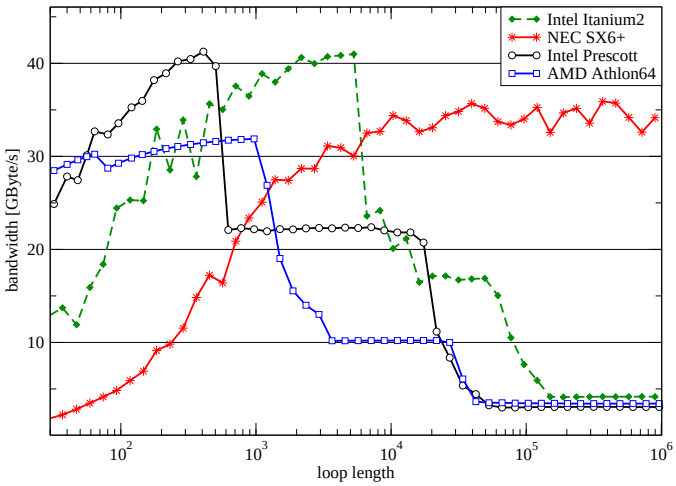

Fig. 1. Memory bandwidth vs. loop length N for the serial vector triad on several architectures.

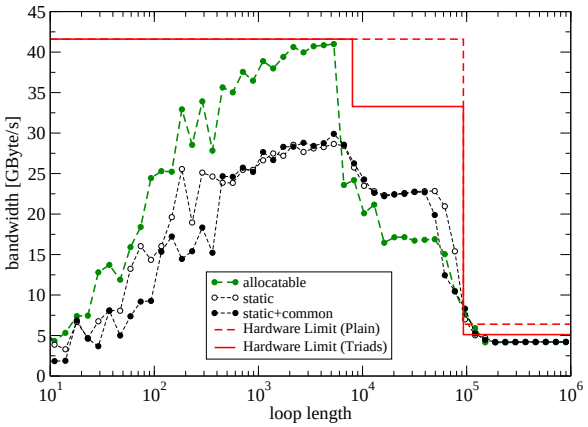

Fig. 2. Serial vector triad bandwidth on Itanium2 using different memory allocation strategies. The bandwidth limits imposed by the hardware are depicted as well, *Plain* denoting the pure hardware numbers and *Triads* taking into account the additional load operation if an L2/L3 write miss occurs.

3.1 Compiler-generated code

The cache characteristics presented in Fig. 1 seem to reflect the underlying hardware architecture very well. Nevertheless, the extremely complex logic of caches requires a closer inspection, which has been done for Itanium2 (see Fig. 2). In a first step we calculate the maximum triad bandwidth from basic hardware parameters like the number of loads/stores per cycle and theoretical bandwidth for each memory level. The Itanium2 processor has a sustained issue rate of four loads or two loads and two stores per cycle, which can in principle be saturated by the L2 cache (at a latency of roughly 7 cycles). Thus, two successive iterations of the vector triad could be executed in two clock cycles, leading to a performance of two FLOPs (half of peak performance) or 32 bytes per cycle (41.6 GByte/s). As shown in Figure 2 this limit is nearly reached for loop lengths of $N \approx$1000–7000 using dynamic allocation of the vectors at runtime. At smaller loop length pipeline start-up limits performance and

at larger data sets the next level of memory hierarchy (L3 cache) is entered. Interestingly, the allocation strategy for the vectors shows up in a significant performance variation for the L2 regime. Using static allocation of the vectors with fixed array length (10^6 for the benchmark runs shown in Fig. 2) independently of the actual loop length may reduce the available bandwidth by more than 30%. The eight-way associativity of L2 cache is not responsible for this behavior, because only four different arrays are used. Jalby et al. [12] have demonstrated that this effect can be attributed to non-precise memory disambiguation for store-load pairs (as they occur in the vector triad) and/or bank conflicts in L2 cache that lead to OZQ stalls [10]. Obviously, dynamic array allocation minimizes the potential address and bank conflicts while static allocation requires (manual) reordering of assembler instructions in order to get optimal performance.

If the aggregate data size of the four vectors exceeds the L2 cache size (at a loop length of 8000), a sharp drop in performance is observed although L2 and L3 have the same bandwidth and similar latencies (7 vs. 14 cycles). Here another effect must be taken into account. If an L2 write miss occurs (when writing to A), the corresponding cache line must first be transferred from L3 to L2 (write-allocate). Thus, four instead of three load operations are requested and the effective available bandwidth is reduced by a factor of 4/5 (see Fig. 2) if data outside the L2 cache is accessed. Nonetheless, our best measurements (for static array allocation) still fall short by at least 25 % from the theoretical maximum and achieve only 50 % of L2 bandwidth. Similar results are presented in [12] for the DAXPY operation (`A(:)=A(:)+s*B(:)`) and thus we must assume that another hardware bottleneck further reduces the available maximum bandwidth in the L3 regime. Indeed, as all FP data is transferred from L2 to registers, concurrent cache line refill and register load operations on this cache can lead to bank conflicts that induce stall cycles [11]. Further investigation of this problem is currently underway.

Memory performance, on the other hand, is limited by the ability of the frontside bus (FSB) to transfer one eight-byte double precision word at a bus frequency of 800 MHz. Following the discussion of L3 performance, nearly all RISC processors use outermost-level caches with write-allocate logic. Thus, also for main memory access every store miss results in a previous cache line load, leading to a total of four loads and one store per iteration for the vector triad, thereby wasting 20 % of the available bandwidth. In case of Itanium2, this means that the effective FSB frequency available for the memory-bound vector triad is $800 \cdot 4/5$ MHz, yielding a maximum available bandwidth 5.12 GByte/s ($= 0.8 \times 6.4$ GByte/s). Fig. 2 shows that about 80 % of this limit is achievable, which is significantly higher than for Intel Prescott (60 %) or AMD Athlon64 (67 %), at least for plain, compiler-generated code. For further comments see the next section.

3.2 Handcoded optimizations

While the achievable sustained memory performance of former x86 cpu generations was far below peak, latest generation CPUs as the AMD Athlon 64 and Intel Pentium 4 Prescott can nearly reach their theoretical peak memory bandwidth. However, this can only be done using the SSE/SSE2 instruction set extensions and special optimizations that have not found their way into standard compilers yet. We have explored the potential of these new instructions with handcoded assembly language versions of the vector triad.

SSE/SSE2

SSE and SSE2, available on Intel Pentium 4 as well as AMD Athlon64 and Opteron processors, not only provide a completely new set of SIMD registers that has significant advantages over the traditional FPU register stack, but also add special instructions for explicit memory subsystem control:

- Nontemporal stores can bypass the cache hierarchy by using the write-combine buffers (WCBs) which are available on all modern CPUs. This not only enables more efficient burst operations on main memory, but also avoids cache pollution by data which is exclusively stored, i. e. for which temporal locality is not applicable.
- Prefetch instructions, being hints to the cache controller, are provided in order to give programmers more control over when and which cachelines are brought into the cache. While the built-in hardware prefetch logic does a reasonable job identifying access patterns, it has some significant limitations [7] that might be circumvented by manual prefetch.
- Explicit cache line flush instructions also allow explicit control over cache use, making cache based optimizations more effective. These special instructions were not used in the benchmarks described below.

The following tests concentrate on achieved cache and main memory bandwidth with the vector triad. All implementations use nontemporal stores for highest write bandwidth in the memory-bound regime. Cache characteristic data was taken using a self-written benchmark suite. The results can be reproduced by other micro benchmarking tools like, e. g., the rightmark memory analyzer [8].

Cache performance

In order to get an impression of cache behavior we ran plain FPU and SSE2 versions of the vector triad (Fig. 3). An important general observation on all Pentium 4 revisions is that cache bandwidth scales with the register width used in load/store instructions, i. e. the CPU reaches its full potential only when using the 16-byte SSE registers. On the other hand, the AMD Athlon 64 bandwidth almost stays the same when going from FPU to SSE2. Moreover, this CPU shows a mediocre L2 bandwidth when compared to the Pentium 4. The AMD design clearly does not favor vector streaming applications as much as the Pentium 4 which is trimmed to SIMD performance. As a side note, the cache bandwidths on a Pentium 4 are asymmetric, meaning that read bandwidth is much higher than write bandwidth, the latter being 4 bytes/cycle on all cache levels. The Athlon64 does not show this asymmetry at all.

One must still keep in mind that the Pentium 4 CPUs derive much of their superior cache performance from a very high clock rate. Memory bandwidth, on the other hand, depends on completely different factors like the number of outstanding memory references or the effectivity of hardware prefetch.

Memory performance

Fig. 4 compares results for memory-bound, compiler-generated, vectorized (using the `-xW` compiler option) and unvectorized triad benchmarks, together with different

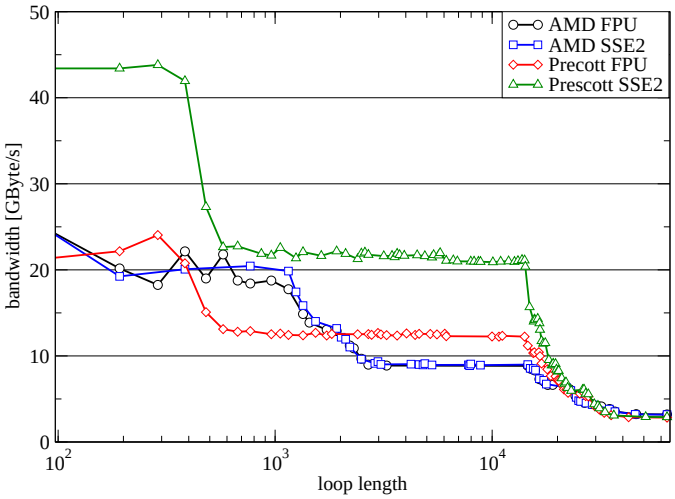

Fig. 3. Serial vector triad bandwidth on Intel Prescott and AMD Athlon64: comparison of SSE2 and FPU versions.

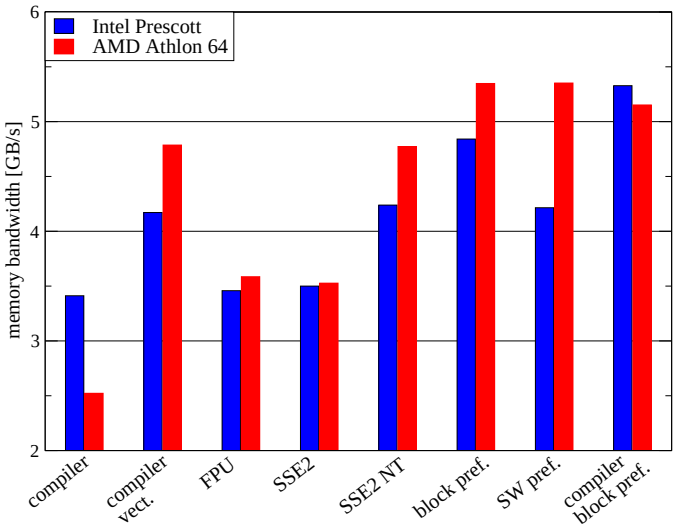

Fig. 4. Serial vector triad main memory bandwidth on Intel Prescott and AMD Athlon64

assembly language and hand-optimized compiler implementations. The x axis shows the category of the optimization. Still the code was specifically optimized for every architecture (concerning cache line lengths etc.). The version implemented with FPU instructions shows roughly the same performance as the unvectorized compiler version. The SSE2 version shows no improvement; using the wide SSE2 registers does not have any influence on memory performance, as expected. Nontemporal stores, on the other hand, yield significant speedup because of increased effective bandwidth as described above. This version shows comparable performance to the compiler version, indicating that the compiler also uses those instructions.

Categories 6 and 7 in Fig. 4 are concerned with different prefetching strategies. "Block prefetch" [9] loads a block of data for all read-only vectors into L1 cache, using

a separate prefetch loop, before any calculations are done. This can be implemented by prefetch instructions (no register spill or cache pollution) or by direct moves into registers, also referred to as "preload":

```
for SIZE
   for BLOCKSIZE
      preload B
   endfor
   for BLOCKSIZE
      preload C
   endfor
   for BLOCKSIZE
      preload D
   endfor
   for BLOCKSIZE
      A=B+C*D    ! nontemporal store
   endfor
endfor
```

Note that there is no overlap between calculation and data transfer, and the preload loops for the read-only arrays are separate. This turns out to be the best strategy for block preload. Fusing the loops would generate multiple concurrent load streams that the CPU can seemingly not handle as effectively. "Software prefetch" uses inlined SSE prefetch instructions. While this should have clear advantages (no significant overhead, no register spill and cache pollution, overlap of data transfer and arithmetic), the block preload approach still works better on Intel architectures. The Athlon64, on the other hand, shows a very slight advantage of software prefetch over block preload.

From the hand-coded assembly kernels we now come back to compiler-generated code. The question arises whether it would be possible to use the block preload in a high-level language, without the compiler messing up the careful arrangement of preload versus computation. The rightmost category in Fig. 4 shows that block preload can indeed be of advantage here, especially for the Intel Prescott processor. It must be noted though that the performance of this code is very sensitive to the tuning parameters. For instance, on an Athlon64 the best strategy is to block for L1 cache, as expected from the discussion about cache performance. On the Prescott processor, however, blocking for L2 cache is much better. This might be attributed to the smaller L1 cache size together with longer memory latencies because of a different system architecture (northbridge as opposed to built-in memory controller as on the Athlon64), and cannot be derived directly from in-cache triad performance (Fig.3) because the recurrence loop hides startup effects due to memory latency.

In summary, block preload is a very interesting alternative for Intel Prescott CPUs, but it does not really pay off for the Athlon64. On the Prescott, careful parameter tuning essentially eliminates the need for hand-coded assembly language. Using block preload techniques for CFD applications thus seems to be a viable option and is being investigated.

Contrary to the observations in section 3.1, where it became clear that naive, compiler-generated code is not able to saturate the memory bus, it is now evident

that relatively simple prefetch/preload techniques can lead to a large increase in memory bandwidth, reaching 80%-90% of peak. Although the SSE/SSE2 instruction set extensions go to great lengths trying to give programmers more control over caches, they are still of limited use, mainly because of their obviously ineffective and often undocumented implementation on different architectures. While this has improved with latest CPU generations (Intel Prescott and AMD Athlon64), it is still necessary to reduce the limitations and increase the efficiency of these instructions in order to get a real alternative to more conservative means of optimization.

4 Shared-memory parallel vector triad

As mentioned previously, OpenMP parallelization of the vector triad seems to be a straightforward task:

```
do R=1, NITER
!$OMP PARALLEL DO
   do I=1,N
     A(I) = B(I) + C(I) * D(I)
   enddo
!$OMP END PARALLEL DO
enddo
```

The point here is that all threads share the same logical address space. In an SMP system with UMA characteristics like a standard dual-Xeon node, every processor can access all available memory with the same bandwidth and latency. As a consequence, the actual physical page addresses of the four arrays do not matter performancewise, at least when N is large (see previous section). On a ccNUMA architecture like SGI Altix or Opteron nodes, memory is logically shared but physically distributed, leading to non-uniform access characteristics (Fig. 4). In that case, the mapping between physical memory pages and logical addresses is essential in terms of performance, e.g. if all data is allocated in one local memory, all threads must share a single path to the data (SHUB, NUMALink4 or NUMALink3 on SGI Altix depending on the number of threads used) and performance does not scale at all. Customarily, a first-touch page allocation strategy is implemented on such systems, i. e. when a logical adress gets mapped to a physical memory page, the page is put into the requesting CPU's local memory. While this is a sensible default, it can lead to problems because the computational kernel is not where the initial mapping takes place. Due to the first-touch policy, initialization of the four arrays `A`, `B`, `C` and `D` must be done in a way that each thread in the computational kernel can access "its" portion of data through the local bus. Two conditions must be met for this to happen: (i) initialization must be done in parallel, using the same thread-page mapping as in the computational kernel, and (ii) static OpenMP scheduling must be used:

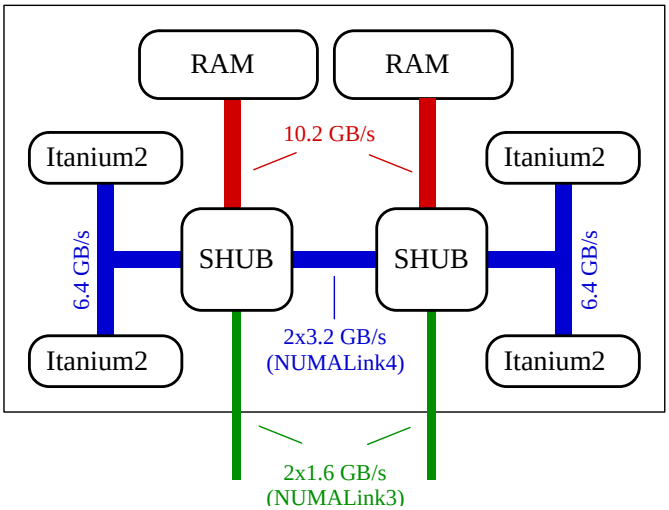

Fig. 5. SGI Altix 3700 SC-brick block diagram. One SC-brick comprises two nodes with two CPUs each. Intra-brick communication is twice as fast as brick-to-brick communicaton

Table 2. Memory performance of the parallel vector triad in GB/s for purely bandwidth-limited problem sizes. Divide by 0.016 to get performance in MFLOP/s.

Threads	IA64 (NoInit)	IA64 (ParInit)	AMD64 (ParInit)	SX6+
1	4.232	4.176	2.694	34.15
2	4.389	4.389	5.086	
4	2.773	8.678	9.957	
8	1.824	17.33		
16	1.573	34.27		

```
!$OMP PARALLEL DO
!$OMP SCHEDULE(STATIC)
do I=1,N
  A(I) = 0.0
  B(I) = BI
  C(I) = CI
  D(i) = DI
enddo
!$OMP END PARALLEL DO
```

```
do R=1, NITER
!$OMP PARALLEL DO
!$OMP SCHEDULE(STATIC)
   do I=1,N
     A(I) = B(I) + C(I) * D(I)
   enddo
!$OMP END PARALLEL DO
enddo
```

It should be obvious that especially the first condition might be hard to meet in a real user code.

Performance figures for out-of-cache data sizes (cf. Table 2) show the effects of improper initialization on ccNUMA architectures. In the column marked “NoInit”, the initialization loop was done on thread zero alone. “ParInit” denotes proper parallel initialization. The IA64 measurements were done on an SGI Altix 3700 system (Itanium2 at 1.3 GHz). While there is no difference between the NoInit and ParInit cases for two threads, which is obvious because of the UMA access characteristics inside the node, performance breaks down dramatically on four threads because two CPUs have to use the slow NUMALink4 connection to access remote memory.

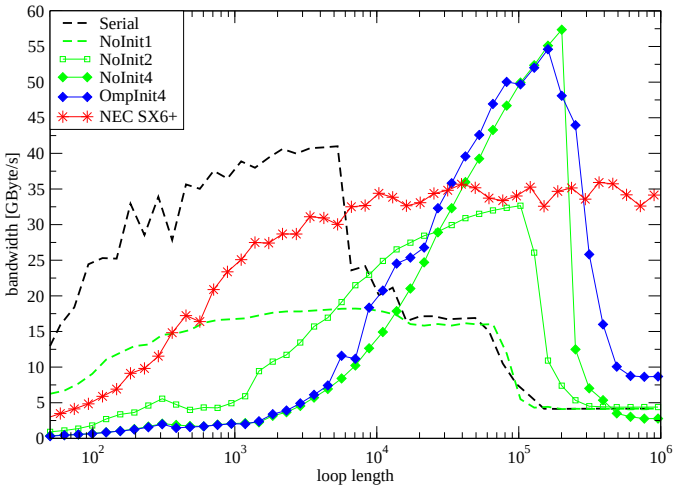

Fig. 6. Memory bandwidth vs. loop length N for the shared-memory parallel triad on SGI Altix (Itanium2) and one SX6+ processor.

Consequently, the four-CPU bandwidth is reduced to the performance of this link. With eight and more CPUs, the even slower NUMALink3 interconnect dominates available bandwidth (as there are two NL3 links per SC-brick, the breakdown is not as severe here). With parallel initialization, on the other hand, scalability is nearly perfect.

For reference, data taken on a four-way AMD Opteron system (2.2 GHz) and on a single NEC SX6+ CPU is also included in Table 2. As expected, eight Altix nodes are the rough equivalent of one single NEC processor, at least for large loop lengths. Fig. 6 gives a more detailed overview of performance data. In the "serial" version, the code was compiled with no OpenMP directives. In comparison to the OpenMP (1 thread) case, it is clear that the compiler refrains from aggressive, cache-friendly optimization when OpenMP is switched on, leading to bad L2 performance. In-cache scalability is acceptable, but OpenMP startup overhead hurts performance for small loop lengths. This has the striking effect that, even when the complete working set fits into the aggregated cache size of all processors, full cache performance is out of reach (see the 4-thread data in Fig. 6). On the other hand, SX6+ data for short loops shows that the often-cited vector pipeline fillup effects are minor when compared to OpenMP overhead. In conclusion, OpenMP parallelization of memory-intensive code on cache-based microprocessors can only be an alternative to vector processing in the very large loop length limit.

5 Conclusions

We have presented a performance evaluation of the memory hierarchies in modern parallel computers using the vector triad. On Intel Pentium4 or AMD Athlon64 processors even this simple benchmark kernel only exploits roughly half of the bandwidth available if using a standard, straightforward implementation. A handcoded block prefetching mechanism implemented in assembly language has been shown to improve the performance from main memory by nearly a factor of two. The important observation here was that flooding the memory subsystem with numerous

concurrent streams is counterproductive. This block prefetch technique can also be used in high-level languages, making it interesting for real-world applications.

Although modern shared-memory systems are widely considered as being easy to program we emphasize the importance of careful tuning in OpenMP codes, in particular on ccNUMA architectures. On these systems, data locality (or lack thereof) often dominates performance. If possible, data initialization and computational kernels must be matched with respect to data access because of the first-touch page mapping policy. To our experience, this seemingly trivial guideline is often violated in memory-bound user codes.

Even considering the significant improvements in processor performance and ccNUMA technology over the past years, classical vector processors still offer the easiest and sometimes the only path to satisfactory performance levels for vectorizable code.

Acknowledgments

This work has been financially supported by the Competence Network for Technical and Scientific High Performance Computing in Bavaria (KONWIHR). We thank H. Bast, U. Küster and S. Triebenbacher for helpful discussion. Support from Intel is gratefully acknowledged.

References

1. C. Lemuet, W. Jalby, and S. Touati: Improving Load/Store Queues Usage in Scientific Computing. The International Conference on Parallel Processing (ICPP'04). Montral, Aug. 2004, IEEE.
2. L. Oliker et al., Evaluation of cache-based superscalar and cacheless vector architectures for scientific computations. In Proc. SC2003, Phoenix, AZ, 15.-21. Nov., 2003.
3. F. Deserno et al., Performance of Scientific Applications on Modern Supercomputers. In S. Wagner et al. (Eds.): High Performance Computing in Science and Engineering Munich 2004, pp. 339-348. Springer-Verlag Berlin Heidelberg 2004 (ISBN 3-540-44326-6)
4. L. Oliker et al., Scientific Computations on Modern Parallel Vector Systems. In Proc. SC2004, Pittsburgh, PA, 6.-12. Nov., 2004.
5. T. Pohl et al., Performance Evaluation of Parallel Large-Scale Lattice Boltzmann Applications on Three Supercomputing Architectures. In Proc. SC2004, Pittsburgh, PA, 6.-12. Nov., 2004.
6. W. Schönauer: Scientific Supercomputing. Self-edition, Karlsruhe (2000)
7. Intel Corp.: IA-32 Optimization Reference Manual. Intel (2004). Available at `http://developer.intel.com/`
8. Rightmark Memory Analyzer. Available at `http://cpu.rightmark.org/products/rmma.shtml`
9. AMD: Athlon Processor, x86 Code optimization guide, pp.86-98. Available at `http://www.amd.com/us-en/assets/content_type/white_papers_and_tech_docs/22007.pdf`
10. Intel Corp.: Itanium2^{TM} Programming and Optimization Reference Manual. Intel (2004). Available at `http://developer.intel.com/`

11. H. Bast and D. Levinthal, Intel Corp.: Private communication
12. W. Jalby, C. Lemuet, and S. Touati, An Effective Memory Operations Optimization Technique for Vector Loops on Itanium2 Processors, Concurrency and Computation: Practice and Experience, USA, accepted for publication.