Performance Characteristics of Large SMP Machines

Dirk Schmidl[1,3], Dieter an Mey[1,3], and Matthias S. Müller[1,2,3]

[1] Center for Computing and Communication, RWTH Aachen University, D - 52074 Aachen
[2] Chair for High Performance Computing, RWTH Aachen University, D - 52074 Aachen
[3] JARA High-Performance Computing, Schinkelstraße 2, D 52062 Aachen
{schmidl,anmey,mueller}@rz.rwth-aachen.de

Abstract. Different types of shared memory machines with large core counts exist today. Standard x86-based servers are build with up to eight sockets per machine. To obtain larger machines, some companies, like SGI or Bull, invented special interconnects to couple a bunch of small servers into one larger SMP, Scalemp uses a special software layer on top of a standard cluster for the same purpose. There is also a trend to couple many small and simple cores into one chip, like in the Intel Xeon Phi chip. In this work we want to highlight different performance attributes of these machine types. Therefor we use some kernel benchmarks to look at basic performance characteristics and we compare the performance for real application codes. We will show different scaling behaviors for the applications which we explain with the use of the kernel benchmarks used before.

1 Introduction

OpenMP is probably the most widely used paradigm for shared memory parallelization in high performance computing. It is often said to be easy to use and in fact it is often easy to get a first loop parallel version of an application, but getting good performance is often more difficult due to the complex design of large shared memory machines. Especially for a larger number of threads good scaling can in many cases only be achieved if the underlying hardware is taken into account.

Hardware vendors have established different types of shared memory machines. Standard servers, based on x86 processors exist with up to 8 sockets in a single machine. E.g. SGI and Bull invented a special interconnect to combine smaller two or four socket machines into one larger shared memory machines. Scalemp provides a software layer, called vSMP foundation, to couple several servers with a standard Infiniband network into one machine running a single OS instance. A different approach to provide the ability to run hundreds of threads is used in Intel's new Many Integrated Core architecture. Here, a lot of small and simple cores are combined in one single chip. The first product in this architecture line is the Intel Xeon Phi coprocessor. The Xeon Phi chip resides in a PCIe extension card and can on the one hand be used as an accelerator to speedup applications and on the other hand it is able to execute standalone executables, since it runs a full operating system instance.

Given this variety of different machines, it is hard for a programmer to choose the appropriate machine for his application. It is also difficult for computing centers to decide

A.P. Rendell et al. (Eds.): IWOMP 2013, LNCS 8122, pp. 58–70, 2013.

on a machine type when a new machine is going to be purchased. Since, it is often hard to get excess to all these machine types to try out the performance on a given application, we want to present a comparison of the basic differences of these machine types. We investigate performance attributes on a standard 8 socket HP server, a SGI Altix ultraviolet, a Bull BCS machine, which both use a special network to couple smaller machines into a larger single system, a Scalemp machine which uses a software for the same purpose and a Xeon Phi extension card which can run a lot of threads on one single chip. We will look at basic characteristics of the memory subsystem, investigate the influence of memory allocation and initialization and run several applications on all platforms to highlight relevant differences between these machines.

The rest of this paper is structured as follows: First, we present related work in section 2 and describe the systems used in section 3. Then, we look at basic performance relevant attributes by means of kernel benchmarks in section 4, before we compare the performance of applications in section 5. After all, we draw our final conclusions in section 6.

2 Related Work

Comparing the performance of shared memory machines is subject to investigations for many years. Many benchmarks or benchmark suites exist to investigate attributes of standard machines. The Stream benchmark [7] for example is widely used to measure the memory bandwidth of a machine and the LMbench benchmark suite [8] offers kernels to measure certain operating system and machine properties. Other benchmarks like the EPCC benchmark [4] concentrate on the performance of the OpenMP runtime, which of course is essential for the scalability of OpenMP applications.

Besides these very basic benchmarks which can give a good indication on specific machine or software attributes, also application benchmarks exist. The SPEC OMP Benchmark Suite [1] delivers a collection of representative OpenMP applications. Other benchmark suites like the Barcelona OpenMP Task Suite [5] or the NAS parallel benchmarks [2], contain relevant application kernels. These benchmarks can be used to compare the performance of different architectures for representative application codes.

These projects focus on the benchmark techniques, also they provide reference results for several architectures. In this work we will focus more on the differences of the architecture, also we reuse some of the ideas provided in the benchmarks mentioned above.

3 Architecture Description

3.1 HP ProLiant

The HP ProLiant DL980 G7 server used for our experiments is a single server equipped with eight Intel Xeon X6550 processors. All processors are clocked at 2 GHz and connected to each other through the Intel Quick Path interconnect. Every processor contains a memory controller attached to 32 GB of main memory, making this server a ccNUMA machine with a total of 256 GB of memory.

3.2 SGI Altix UltraViolet

The SGI Altix UV system consists of several two socket boards, each equipped with two Intel Xeon E7- 4870 10-core processors clocked at 2.4 GHz. All of these boards are connected with SGIs NUMALink interconnect into a single shared memory machine. Since on one board the cache-coherence is established directly over the QPI, whereas the NUMALink network is needed for different board, this machine is a hierarchical NUMA machine, with different cache-coherency mechanisms on different hierarchical levels. The total machine used in our tests has 2080 cores and about 2 TB of main memory. All of our tests were done on up to 16 processors during batch operation of the system. For a better comparison with the 8-core processors used in the other systems, all tests were done using only eight of the ten available cores on each socket.

3.3 BCS

The BCS system consists of four bullx s6010 boards. Each board is equipped with four Intel Xeon X7550 processors and 64 GB of main memory. The Intel Quick Path Interconnect combines the four sockets to a single system and the Bull Coherence Switch (BCS) technology is used to extents the QPI to combine four of those boards into one SMP machine with 128 cores and 256 GB of main memory. So, this system is also a hierarchical NUMA system.

3.4 Scalemp

The Scalemp machine investigated here consists of 16 boards, each board is also equipped with four Intel Xeon X7550 processors clocked at 2 GHz and 256 GB of main memory. The boards are connected via a 4x QDR InfiniBand network, where every board is connected via two HCAs. Thus, from a hardware point of view this is an ordinary (small) cluster. The innovative part of the machine is the vSMP software of the company Scalemp, which runs below the operating system and creates a Single System Image on top of the described hardware. The virtualization layer of the processors and the InfiniBand network is used by the vSMP software to create cache-coherency on a per page basis and to allow remote memory access between all the boards. A partition of the main memory is reserved by the vSMP software to run different caching and prefetching mechanisms automatically in the background, as well as a page-based memory migration mechanism. These mechanisms do not only move pages on access, they can also adjust the home node of memory pages if frequently used on a remote node. This is a notable difference to standard x86-based non-uniform memory architectures (NUMA), like the Altix or BCS machine, where page migration needs to be done by the user, if possible at all. From a user point of view the machine looks like a single Linux machine with 64 eight-core processors and about 3.7 TB of main memory. About 300 GB of the available memory are used by the vSMP software internally for caching. Linux sees 64 NUMA nodes, each containing about 64 GB of main memory.

3.5 Intel Xeon Phi

The Intel Xeon Phi coprocessor is based on the concepts of the Intel Architecture (IA) and provides a shared-memory many-core CPU that is packed on a PCI Express

extension card. The version used here has 60 cores clocked at 1.053 GHz and offers full cache coherency across all cores with 8 GB of GDDR5 memory. A ring network connects all cores with each other and with memory and I/O devices. Every core supports 4-way Hyperthreading, which allows the system to run up to 240 threads in parallel. The comparably small amount of main memory is attached to the Xeon Phi Chip as one NUMA node. Thus, the system is a 240-way parallel system with a uniform memory architecture, which is another difference to the other machines with are large NUMA systems.

The Xeon Phi card used in out experiments was plugged into a host system with two Intel Xeon E5 processors. For all of our experiments we used the host system only to cross compile the executables, which were copied and executed stand-alone on the Xeon Phi. This procedure gives us insight in the performance attributes of the chip, independent from the programming model used. Of course comparing one extension card with complete systems is an uneven comparison, but for sure we will see standalone systems with hundreds of cores in the near future and the Phi might give evidence on the behavior of such systems.

4 Performance Characteristics

Before we look into the performance of application codes, it is useful to understand in detail the differences of the investigated architectures. We use several benchmarks to highlight certain properties of the machines.

4.1 Serial Memory Bandwidth

Many scientific applications are limited by the memory bottleneck in modern systems. Therefore, the memory bandwidth is the most important factor for these applications.

We measured the read and write bandwidth with one thread on these machines. Since the trends for read and write bandwidth were nearly identical with the only difference that the write bandwidth was slightly slower on all machines, we concentrate on the write bandwidth during the following discussion. The simple benchmark writes a single value to an array of a given size several times and calculates the reached bandwidth. On the NUMA machines, i.e. HP, Altix, BCS and Scalemp, we measured the bandwidth to NUMA nodes on all NUMA levels (local, on the same board and on a remote board). We used the `numactl` tool provided by Linux to influence the used core and NUMA node. On the Xeon Phi machine only one NUMA level exists, so the placement was not varied here.

Figure 1 illustrates the reached write bandwidths for increasing memory footprints. On all machines we can see a typical cache behavior. For very small data sizes the bandwidth is poor, since complete cache lines need to be written. Then the bandwidth rises for memory sizes that fit into the caches. On the Xeon Phi a peak of 4 GB/s is reach, on the other systems a bandwidth of about 14-18 GB/s is achieved. When the memory sizes exceed the capacity of the last level caches, the memory bandwidth drops down significantly. On the Xeon Phi system the `Standard` curve displays the bandwidth reached, when the code was just cross compiled for the system. The Intel Compiler allows to provide a switch to insert software prefetch instructions into the binary.

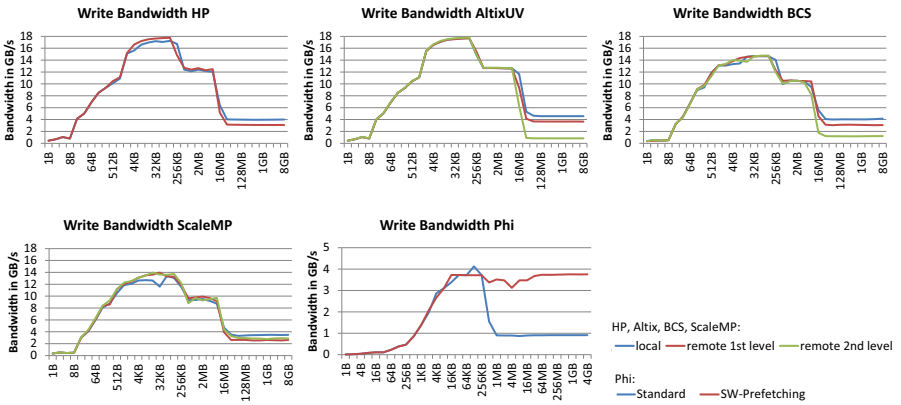

Fig. 1. Write Bandwidth in GB/s of the HP, Altix, BCS, Scalemp and Xeon Phi system for different memory footprints

The downside of this approach is, that the user needs to specify exactly the range that should be prefetched. When we instruct the Compiler to prefetch 64 elements ahead for the L2 cache and 8 for the L1 cache, we can improve the reached bandwidth for writing as depicted by the `SW-prefetching` line. These results show, that no temporal store operations are used by the compiler and that all cachelines are first read before they are written, so prefetching can have a positive effect. Here, nearly the full L2 cache bandwidth can be reached for arbitrary data sizes, as long as they fit into the 8 GB of memory.

On the other systems these switches did not work. On these systems we observe a bandwidth reduction to about 4-5 GB/s if the memory is located in the local NUMA node and to about 3 GB/s if a NUMA node on the same board is used. If the memory is located on a different board, we can observe an interesting difference between the Altix and BCS machine on the one hand and the Scalemp machine on the other hand. The bandwidth on Altix and BCS descents to 0.5 GB/s whereas it stays at 3 GB/s on the Scalemp machine. The reason therefor is, the caching inside of the vSMP software. The memory pages can be kept in a board local cache and thus only data on the local board is affected by the benchmark. This mechanism can help to achieve a good data locality, even if the data is spread across the system and no memory migration is performed by the user.

4.2 Distance Matrix

Of course the remote bandwidth to different sockets depends on the distance of the sockets and the used interconnect. In [10] we described a way to measure and present the distance between sockets in a distance matrix. Basically, we measure the bandwidth between all sockets and scale the matrix in a way that the upper left element has the value 10 and the others are scaled in a way that decreasing bandwidth between two sockets results in increasing distance. Figure 2 shows the distance matrices for the HP

and BCS machine. For the ScaleMP and Altix machine the tests were not possible, since we could not get the machine exclusively and on the Phi machine the measurements are useless, since only one socket exists.

Socket	0	1	2	3	4	5	6	7
0	10	10	17	13	18	18	18	18
1	10	10	17	13	18	18	18	18
2	17	17	10	11	18	18	18	18
3	17	17	10	11	19	19	18	18
4	18	18	18	18	10	10	17	17
5	18	18	18	18	10	10	17	17
6	18	18	18	18	17	17	10	10
7	18	19	18	18	17	17	10	9

Socket	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	10	13	13	13	57	57	57	57	59	59	59	59	57	57	57	
1	13	10	13	13	56	55	56	56	56	56	56	55	55	56	56	55
2	14	13	10	13	58	58	58	58	56	56	56	56	58	58	58	58
3	13	13	13	10	56	55	56	55	56	56	56	55	56	55	56	55
4	56	56	56	56	10	13	13	13	56	56	56	57	58	58	58	58
5	55	55	55	55	13	10	13	13	55	55	55	55	56	56	56	55
6	58	58	58	59	13	13	10	13	58	58	58	58	56	56	56	57
7	56	55	56	55	13	13	13	10	56	56	56	56	56	56	56	56
8	58	58	58	58	56	57	56	56	10	13	13	13	56	56	56	56
9	56	56	55	55	55	55	55	55	13	10	13	13	55	55	56	55
10	56	56	56	56	58	58	58	58	13	13	10	13	58	58	58	58
11	56	56	56	55	56	56	56	55	13	13	13	10	56	56	56	56
12	56	56	56	56	58	58	58	58	56	57	56	56	10	13	13	13
13	55	55	55	55	56	56	55	55	56	55	55	55	13	10	13	13
14	58	58	58	58	56	56	56	56	58	58	58	58	13	13	10	13
15	56	56	56	56	56	56	56	56	56	56	56	56	13	13	13	10

Fig. 2. Distance matrix of the HP (left) and the BCS system (right). The matrix is scaled such that the upper left value is always ten, larger numbers indicate higher distances.

Obviously, there are differences between both systems. On the HP board there are always two sockets which seem to be connected very fast, whereas the other sockets have distances between 17 - 19. On the BCS system all accesses on one board are between 10 and 13, which is faster than most of the connections on the one HP board. All connections using the BCS chip, are significantly slower, here a distance of 55 - 59 is reached. So, the BCS machine can provide cache coherence over a larger number of cores, but some performance regressions comes along with this.

4.3 Parallel Memory Bandwidth

Parallel applications of course use more than one core at a time. Thus, the total bandwidth for a parallel application is important. We modified the benchmark from section 4.1 to work with several threads on an array and measure the read and write bandwidth. We use a compact thread binding on all of the machines. So, we first fill up cores and sockets with the maximum number of threads, before we use the next core or socket.

Figure 3 shows the bandwidth for an increasing number of threads on the different platforms for a memory footprint of 16 MB per thread. On the Intel Xeon Phi machine the maximum bandwidth of about 130 GB/s for reading and 60 GB/s for writing can be achieved with about 120 threads. Beyond this, the bandwidth stagnates. On the NUMA systems the bandwidth rises with the number of sockets used and does not stagnate at all. Here, the read bandwidth is higher than the write bandwidth available for all systems. Noticeable is, that the BCS machine reaches an 5-10 % higher maximum bandwidth compared to the Scalemp machine, also the same underlying hardware is used. The reason therefor is the vSMP software layer which inserts a slight overhead for memory management and leads to a small reduction in the available bandwidth.

4.4 Memory_Go_Around

The parallel bandwidth measured before is the optimal bandwidth that is reached, when all threads work on their own data. In many algorithms, some data sharing is required

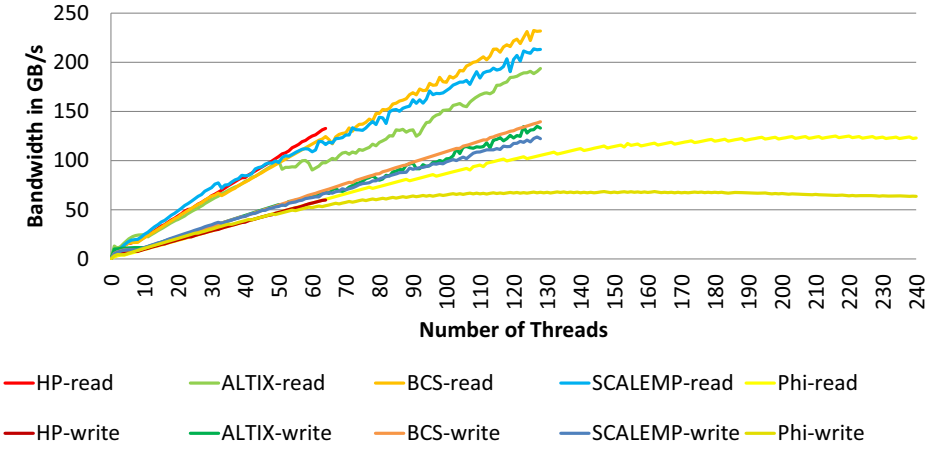

Fig. 3. Parallel read and write bandwidth on the HP, Altix, BCS, Scalemp and Xeon Phi systems for an increasing number of threads

which leads to a certain amount of necessary remote accesses. To investigate the drop down in the reached bandwidth with remote accesses, we modified the bandwidth benchmark in a way, that it no longer works only on local data.

The modified benchmark, we call it `memory_go_around`, works in $n+1$ steps for n threads. In step zero, every thread initializes its own data and measures the memory bandwidth to access it. In step one, all threads work on the data of their right neighbor, so thread t works on memory initialized by thread $(t+1)\ mod\ (n-1)$, as exemplified in figure 4. As before, we place threads in a way that neighboring threads are as close as possible to each other, meaning that there is a high chance that they run on the same NUMA node or board. Hence, the number of remote memory accesses rises for the first $n/2$ steps. Then the number shrinks again. In step $n-1$ every thread works on the memory of the left neighbor and in step n again only local accesses occur.

Figure 5 shows the result for 64 Threads on the HP, 120 threads on the Xeon Phi and for 128 threads on the Altix, BCS and Scalemp machine. Since the Xeon Phi machine has only one NUMA node, the data is always stored on this NUMA node and the reached bandwidth is the same in all steps, about 130 GB/s. On the other machine the bandwidth declines for the first half of the steps and then rises up again. Of course this is related to the increasing number of remote accesses and the increasing distance between these accesses. On the HP system, the performance drops down from about 120 to 60 GB/s. So, with maximum traffic over the QPI bus, still 50% of the peak bandwidth can be reached. On the Altix and BCS machine the drop down is from about 250 to 18 or 8 GB/s which is 6% or 3% of the available maximum bandwidth. On the Scalemp the drop down is even higher, here only 0.5 GB/s are reached, which means roughly 0.2% of the maximum bandwidth. Overall, the bandwidth goes down on all hierarchical NUMA machines (Altix, BCS and Scalemp) significantly, this is one of the biggest differences in comparison to single systems with one level of NUMAness, like the HP machine. However, if an application does not require a lot of data sharing between the threads, proper data placement can avoid these problems.

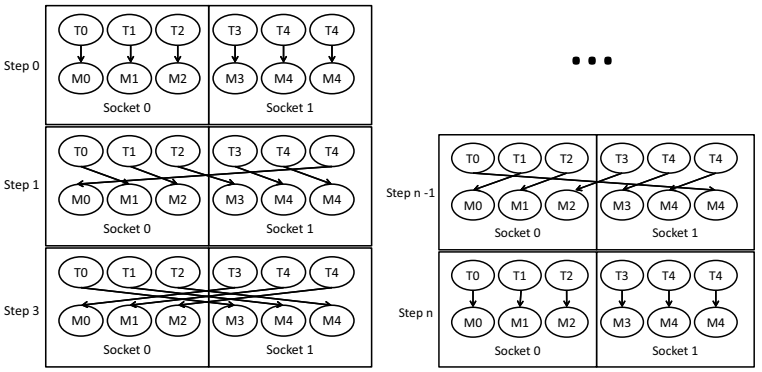

Fig. 4. The memory_go_around benchmark forks in n+1 steps. In the first step the memory of the right neighbour is used to measure the bandwidth, in the next step the memory of the next neighbour and so on. This increases the distance between thread and memory in every step, until half of the steps are done, then the distance decreases until it reaches zero in the last step.

4.5 Synchronization Overhead

Besides the memory bandwidth, locks are often critical for the performance of an application on larger shared memory systems. Locks can occur explicitly, like calls to `omp_set_lock`, or they can happen inside of other calls, e.g. a call to `malloc` requires synchronization. We measured the overhead of these two mentioned routines. The results are shown in the left part of table 1 for all investigated systems. For memory allocation the Intel OpenMP runtime provides a version of `malloc` which is optimized for multithreading, called `kmp_malloc`. We investigated this version as well, the results are shown in the bottom right part of the table.

To measure the overhead of explicit locks we used the `syncbench` out of the EPCC microbenchmarks [4]. Our `malloc` test calls `malloc` or `kmp_malloc` on all threads simultaneously 1000 times and computes the average duration of one call. The overhead of OpenMP Locks increases with the number of threads involved on all platforms. On the HP, Altix, BCS and Scalemp machine it is nearly the same, as long as only a small number of threads is involved, but for 64 and 128 threads the overhead rises more drastically on the Scalemp machine, where the maximum is about 35 microseconds, what is high compared to the 1 - 3 microseconds on the other machine. On the Xeon Phi system, the overhead for 120 threads is nearly the same on the other hardware based systems. However, the overhead with one thread is higher, due to the slower serial threads. Overall, the ratio between serial and parallel execution is advantageous on the Phi system. The overhead for a lock goes up for a factor of 5 for 120 threads (0.4 to 2 microseconds) whereas for example it goes up by a factor of about 27 on the BCS system (0.06 to 1.64). This means, that the scaling of lock based applications on the Phi is better than on the other system.

For the `malloc` calls, the overhead rises much faster with the number of threads on all systems, than for the OpenMP locks. E.g on the Altix machine it starts at 3 microseconds and goes up to about 20,000. On the other machines the behavior is nearly the same for a large number of threads.

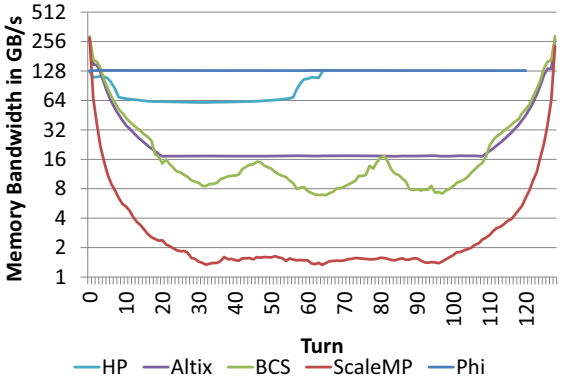

Fig. 5. Bandwidth measured with the memory_go_around benchmark for n+1 steps with n threads on the HP, Altix, BCS, Scalemp and Xeon Phi system

The `kmp_malloc` calls introduce less overhead on most of the machines. On the Scalemp machine the overhead is smaller for a medium number of threads, but higher for 128 threads compared to ordinary `malloc` calls. On the Xeon Phi machine the overhead is significantly higher for all numbers of threads, e.g. with 175,641 microseconds compared to 26,603 for 120 threads.

Another point where synchronization might be needed, which is not always obvious to the programmer is, when data is initialized. The operating system needs to establish a unique mapping between virtual and physical addresses which might require lock. Therefor we implemented a test which initializes 2 GB of data and measures the bandwidth reached during initialization. Table 1 in the upper right part shows the reached bandwidth on all systems. Noticeable is, that the bandwidth rises on all systems for a small numbers of threads. Of course this is due to the higher memory bandwidth which seems to be the bottleneck for only a few threads. On the HP, Altix and BCS machine the bandwidth goes up until the end of 64 / 128 threads is reached. However, on the Scalemp machine the bandwidth drops down as soon as more than 32 threads are involved, meaning as soon as more than one physical board is used. The overhead of the vSMP software seems to slow down the initialization of memory whereas the hardware based solution in the HP, Altix and BCS system does not. On the Phi system, the bandwidth goes slightly down for 120 threads. However, we have seen before, that the total memory bandwidth also goes down at the end, so this is not a surprise here.

Overall, the overhead for locking seems to work better on the hardware based solutions. The vSMP software seems to introduce a significant overhead, so it is much more important to avoid extensive synchronization, e.g. though many `malloc` calls, on the Scalemp machine.

5 Application Case Studies

Finally, we want to look at the performance of two applications from the RWTH Aachen University and compare the results on the different systems.

Table 1. Overhead of OpenMP locks, calls to malloc and calls to kmp_malloc on the investigated machines were measured in microseconds and initialization times to initialize 2 GB of data was measured in GB/s. 30, 60 and 120 threads were used on the Xeon Phi, 32,46 and 128 on the other machines.

	OMP Locks					**Initialization**				
#Threads	HP	ALTIX	BCS	SCMP	PHI	HP	ALTIX	BCS	SCMP	PHI
1	0.03	0.05	0.06	0.07	0.40	1.42	1.38	1.31	1.15	0.67
32/30	0.97	3.29	0.62	0.99	1.77	16.70	18.30	18.36	15.07	17.73
64/60	1.07	3.72	1.04	24.36	1.94	32.93	33.70	34.24	4.86	23.12
128/120		2.99	1.64	35.78	2.01		72.10	67.98	3.63	19.32
	malloc					**kmp_malloc**				
#Threads	HP	ALTIX	BCS	SCMP	PHI	HP	ALTIX	BCS	SCMP	PHI
1	3.61	3.46	4.30	63.64	15.27	3.04	2.60	3.63	45	981
32/30	6075	5146	4902	6887	11023	411	558	530	546	37211
64/60	12470	10473	14007	13882	19807	1476	2646	2786	12260	88717
128/120		21030	29552	28702	26603		11742	12958	54959	175641

5.1 NestedCP

NestedCP [6] is developed at the Virtual Reality Group of the RWTH Aachen University and is used to extract critical points in unsteady flow field datasets. Critical points are essential parts of the velocity field topologies and extracting them helps to interactively visualize the data in virtual environments.

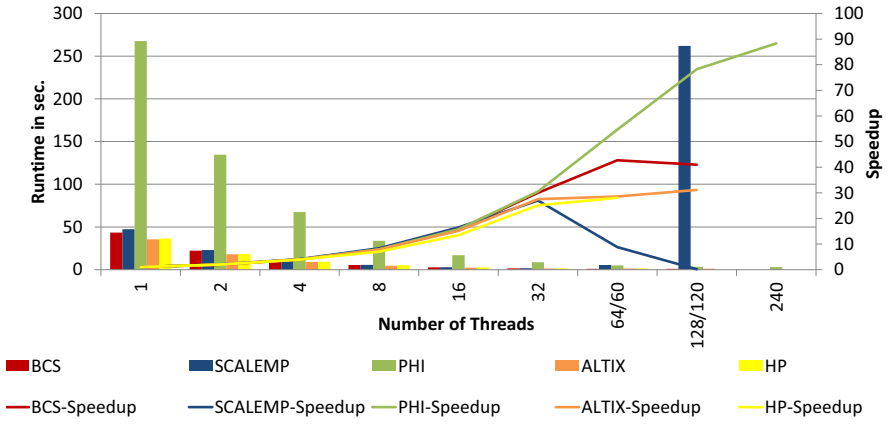

Fig. 6. Peformance and Speedup of the NestedCP code

Figure 6 shows the runtime and speedup of NestedCP on all platforms. We used a code version parallelized with OpenMP tasks for our experiments. Apparently the runtime of the Code on the Xeon Phi is significantly slower than on the other systems. This is due to the slower clockrate and the simple structure of the cores. However, the

scalability is best on the Xeon Phi system. With 240 threads a speedup of about 90 can be reached. On the BCS system, a speedup of 45 can be observed for 64 threads. On the HP system and the Altix machine, a slightly worse speedup can be observed. This is due to the slightly better single threaded runtime on these systems. But, on the HP and Altix machine, the code scales until 60 or 128 threads are used, whereas the performance slightly drops down on the BCS machine at the end. On the Scalemp machine the scalability is much worse. Here, the performance rises as long as only one board is used and drops down significantly with 64 and 128 threads.

This behavior is exactly what we have seen for the synchronization mechanisms on the mentioned machines. The Xeon Phi system has the worst serial performance but a better scaling behavior than the other machines. The HP system provides a good scaling behavior. On the Altix and BCS system the performance goes down, when more than one board is used, but not as significant as on the Scalemp machine. Our assumption is, that the NestedCP code is limited by locking routines and therefore the mentioned scaling behavior is observed.

5.2 TrajSearch

The second code investigated here is TrajSearch. TrajSearch is a code to investigate turbulences which occur during combustion. It is a post-processing code for dissipation element analysis developed by Peters and Wang [9] from the Institute for Combustion Technology at the RWTH Aachen University. It decomposes a highly resolved 3D turbulent flow field obtained by Direct Numerical Simulation (DNS) into non-arbitrary, space-filling and non-overlapping geometrical elements called 'dissipation elements'. Starting from every grid point in the direction of ascending and descending gradient of an underlaying diffusion controlled scalar field, the local maximum and minimum point are found. A dissipation element is defined as a volume from which all trajectories reach the same minimum and maximum point.

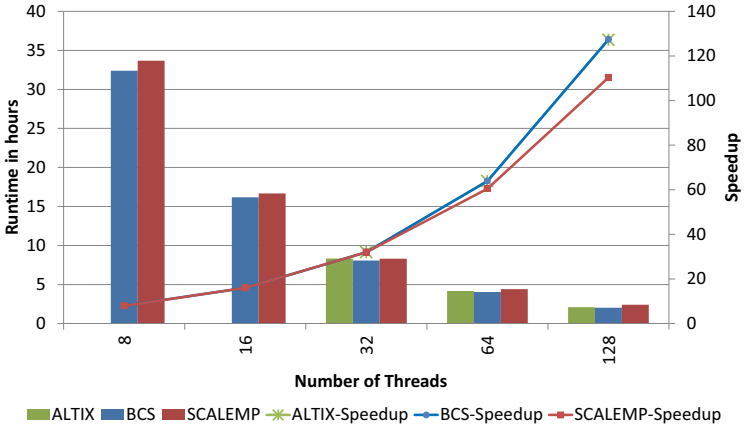

Fig. 7. Runtime and scaleability of the TrajSearch code on the Altix, BCS and Scalemp machine

We reduced locking and optimized the data placement of this code to gain good performance on the Scalemp machine, see [3] for details. Figure 7 shows the runtime and performance of TrajSearch on the Altix, BCS and Scalemp machine. The memory available for the Xeon Phi did not suffice to store the dataset, so the Phi system is not taken into account for this comparison and the HP machine was not available for a time slot large enough to do this tests. Although the code was optimized for the Scalemp system, it scales slightly better on the Altix and BCS machine. It reaches a speedup of about 127 on the Altix and BCS and about 110 on the Scalemp machine for 128 threads. This indicates, that the tuning steps done for the Scalemp machine were also useful for the Altix and BCS machine. However, the code reaches a noticeable speedup, even on the Scalemp machine, which makes all three machines suitable for execution.

6 Conclusion

We investigated performance attributes of several large SMP systems. One standard 8-socket server from HP, an SGI Altix UV, a system based on the Bull Coherence Switch, a Scalemp system and the Intel Xeon Phi chip. We showed, differences in the memory bandwidth on all systems, e.g. when the vSMP software cache has positive influence. Furthermore we investigated the performance influence of remote accesses with the help of the `memory_go_around` benchmark. The benchmark showed, that the negative influence is significantly higher on the Altix, BCS and Scalemp machine than on the HP server and that there is no influence on the Xeon Phi system. Synchronization and locks had a bad influence on all systems, when the number of threads rises. On the Xeon Phi the best ratio of locking overhead between serial and parallel execution could be observed.

This behavior was also reflected in the NestedCP code, where we observed a good scaling on the HP and Xeon Phi, a slightly worse scaling on the Altix and BCS and a bad scaling on the Scalemp machine. However, the TrajSearch code investigated at the end scaled well on all investigated systems, the Altix, the BCS and the Scalemp system.

Overall, if a code is optimized and does not need many locking routines, it can perform well on all investigated systems. If locking or remote accesses cannot be avoided, there is a high chance that a code scales best on the Phi system, followed by the one level NUMA machine and then followed by the hierarchical NUMA machines. However, non-hierarchical NUMA machines are typically limited to 8 sockets or less, whereas the hierarchical machines allow the use of 16 sockets in case of the BCS machine and several hundred sockets for the Altix UV and the ScaleMP machine. So, if the application scales well hierarchical NUMA machines offer an tremendous amount of shared memory compute resources.

Of course the scaling behavior of a system is important for the application performance, but the single core performance is equally important for the overall runtime. All investigated systems besides the Xeon Phi system use comparable Xeon Processors which deliver nearly the same single thread performance, so the differences in scaling directly reflect the overall performance. On the Xeon Phi the single thread performance is worse compared to the Xeon based systems, so the overall runtime of an application might still be worse on this system, even if it scales well.

Acknowledgement. Some of the Tests were performed with computing resources granted by JARA-HPC from RWTH Aachen University under project jara0001. Parts of this work were funded by the German Federal Ministry of Research and Education (BMBF) under Grant No. 01IH11006. The authors also like to thank the LRZ in Munich for the provided compute time on the SGI Altix system.

References

1. Aslot, V., Domeika, M., Eigenmann, R., Gaertner, G., Jones, W.B., Parady, B.: SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance. In: Eigenmann, R., Voss, M.J. (eds.) WOMPAT 2001. LNCS, vol. 2104, pp. 1–10. Springer, Heidelberg (2001)
2. Bailey, D.H., Barszcz, E., Barton, J.T., Browning, D.S., Carter, R.L., Fatoohi, R.A., Frederickson, P.O., Lasinski, T.A., Simon, H.D., Venkatakrishnan, V., Weeratunga, S.K.: The NAS parallel benchmarks. Technical report, NASA Ames Research Center (1991)
3. Berr, N., Schmidl, D., Göbbert, J.H., Lankes, S., Mey, D., Bemmerl, T., Bischof, C.: Trajectory-Search on ScaleMP's vSMP Architecture. In: Applications, Tools and Techniques on the Road to Exascale Computing: Proceedings of the 14th Biennial ParCo Conference, ParCo 2011, Advances in Parallel Computing, Ghent, Belgium, vol. 22. IOS Press, New York (2012)
4. Bull, J.M.: Measuring Synchronisation and Scheduling Overheads in OpenMP. In: Proceedings of First European Workshop on OpenMP, pp. 99–105 (1999)
5. Duran, A., Teruel, X., Ferrer, R., Martorell, X., Ayguade, E.: Barcelona OpenMP Tasks Suite: A Set of Benchmarks Targeting the Exploitation of Task Parallelism in OpenMP. In: International Conference on Parallel Processing, ICPP 2009, pp. 124–131 (2009)
6. Gerndt, A., Sarholz, S., Wolter, M., Mey, D.A., Bischof, C., Kuhlen, T.: Nested OpenMP for Efficient Computation of 3D Critical Points in Multi-Block CFD Datasets. In: SC 2006 Conference, Proceedings of the ACM/IEEE, p. 46 (November 2006)
7. McCalpin, J.: STREAM: Sustainable Memory Bandwidth in High Performance Computers (1999), `http://www.cs.virginia.edu/stream` (accessed March 29, 2012)
8. McVoy, L., Staelin, C.: lmbench: Portable Tools for Performance Analysis. In: Proceedings of the 1996 Annual Conference on USENIX Annual Technical Conference, ATEC 1996, Berkeley, CA, USA, p. 23. USENIX Association (1996)
9. Peters, N., Wang, L.: Dissipation element analysis of scalar fields in turbulence. C. R. Mechanique 334, 493–506 (2006)
10. Schmidl, D., Terboven, C., an Mey, D.: Towards NUMA Support with Distance Information. In: Chapman, B.M., Gropp, W.D., Kumaran, K., Müller, M.S. (eds.) IWOMP 2011. LNCS, vol. 6665, pp. 69–79. Springer, Heidelberg (2011)