Performance Analysis of Complex Shared Memory Systems

Abridged Version of Dissertation

Daniel Molka

September 12, 2016

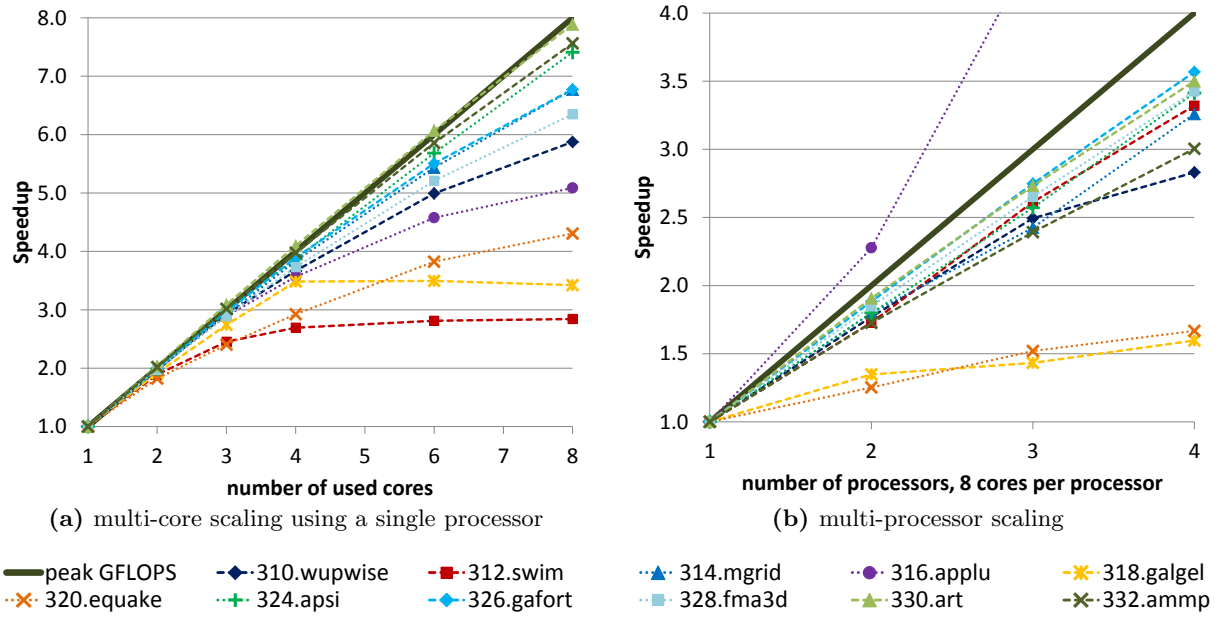
The goal of this thesis is to improve the understanding of the achieved application performance on existing hardware. It can be observed that the scaling of parallel applications on multi-core processors differs significantly from the scaling on multiple processors. Therefore, the properties of shared resources in contemporary multi-core processors as well as remote accesses in multi-processor systems are investigated and their respective impact on the application performance is analyzed. As a first step, a comprehensive suite of highly optimized micro-benchmarks is developed. These benchmarks are able to determine the performance of memory accesses depending on the location and coherence state of the data. They are used to perform an in-depth analysis of the characteristics of memory accesses in contemporary multi-processor systems, which identifies potential bottlenecks. In order to localize performance problems, it also has to be determined to which extend the application performance is limited by certain resources. Therefore, a methodology to derive metrics for the utilization of individual components in the memory hierarchy as well as waiting times caused by memory accesses is developed in the second step. The approach is based on hardware performance counters. The developed micro-benchmarks are used to selectively stress individual components, which can be used to identify the events that provide a reasonable assessment for the utilization of the respective component and the amount of time that is spent waiting for memory accesses to complete. Finally, the knowledge gained from this process is used to implement a visualization of memory related performance issues in existing performance analysis tools.

1 Introduction

High performance computing (HPC) is an indispensable tool that is required to obtain new insights in many scientific disciplines [Vet15, Chapter 1]. HPC systems are getting more and more powerful from year to year as documented in the Top500 list of the fastest supercomputers [Str+15, Figure 1]. However, scientific applications typically are not able to fully utilize this potential [FC07, Figure 1]. Even in case of the distinguished scientific applications that have been awarded the ACM Gordon Bell Prize, the achieved performance is significantly lower than the theoretical peak performance in most cases [Str+15, Table 1]. The effectiveness is even worse in several other scientific applications. Utilization levels of 10% and lower are not uncommon [Oli+05, Table 4]. The percentage of the peak performance that is achieved by an application can also vary on different systems [Oli+05, Table 2, 3, and 4].
The continuous performance improvement of HPC systems is enabled by two developments: increasing the number of processors per system [Str+15, Figure 2] and increasing the performance

per processor [Str+15, Figure 3]. In recent years, the latter is to a large extent achieved by increasing the number of cores per processor. This represents a major change in the system architecture, which poses a challenge for performance analysis and optimization efforts. For many parallel applications, it can be observed that the performance improvement that is achieved by using multiple cores of a single processor significantly differs from the attainable speedup when multiple processors are used. Figure 1 illustrates this phenomenon using the SPEComp2001 suite [Asl+01] as an example. SPEComp2001 consists of eleven individual benchmarks, which have very small sequential portions. Thus—according to Amdahl's Law [Amd67]—speedups close to the ideal linear speedup can be expected for small numbers of threads [Asl+01, Table 2]. However, as illustrated in Figure 1a, the achieved speedup on a single multi-core processor is far from optimal for some of the benchmarks. Furthermore, the scaling with the number of used processors, which is depicted in Figure 1b, significantly differs from the multi-core scaling. A Similar discrepancy between multi-core and multi-processor scaling can be observed for SPEComp2012 [Mül+12, Figure 3]. It must be assumed that these differences are caused by characteristics of the hardware.

The complexity of today's shared memory systems results in various potential bottlenecks that may lead to suboptimal performance. Modern microprocessors feature multiple levels of cache—small and fast buffers for frequently accessed data—in order to improve the performance of memory accesses. Nevertheless, memory accesses can account for a significant portion of the average cycles per instruction [BGB98] and thus constitute a significant portion of the processing time. Furthermore, caches and the memory interface can be shared between multiple cores of a processor. The contention of shared resources can limit the scalability of parallel applications [Mol+11]. In multi-processor systems the physical memory is typically distributed among the processors, which leads to different performance depending on the distance to the accessed



(a) multi-core scaling using a single processor

(b) multi-processor scaling

Figure 1: SPEC OMPM2001 scaling on a quad-socket Intel Xeon X7560 system [Mol+11]: The performance increase due to using multiple cores (left) can differ significantly from the speedup that is achieved using multiple processors (right), e.g., *312.swim* scales poorly on the selected multi-core processor, but benefits strongly from using multiple processors. *318.galgel* and *320.equake* show underwhelming performance gains if multiple processors are used. The super-linear speedup *316.applu* is a known characteristic of this benchmark [FGD07].

data. These non-uniform memory access (NUMA) characteristics influence the performance and scalability of parallel applications [MG11]. Cache coherence protocols [HP06, Section 4.2 – 4.4] also affect the performance of memory accesses [HLK97; Mol+09; MHS14; Mol+15].
In order to detect bottlenecks in the memory hierarchy, one needs to know the peak achievable performance of the individual components. Therefore, this thesis introduces highly optimized micro-benchmarks for 64 bit x86 processors (x86-64). These benchmarks measure the achievable performance of data transfers in multi-core processors as well as multi-processor systems. This includes latency and bandwidth measurements for data that is located in local and remote caches as well as the system's NUMA characteristics. Furthermore, the impact of the cache coherence protocol is considered. Based on this, a methodology for the identification of meaningful hardware performance counters—that can be used to measure the utilization of various resources and determine the impact of the memory hierarchy on the performance of parallel applications—is presented. The procedure comprises three steps:

1. a comprehensive analysis of the performance of cache and memory accesses in contemporary multi-processor systems in order to identify potential bottlenecks
2. stressing individual components in the memory hierarchy in order to identify performance counters that measure the utilization of these resources as well as the time spent waiting for the memory hierarchy
3. a proof-of-concept visualization of the component utilization within the memory hierarchy as well as memory related waiting times using existing performance analysis tools

Remote cache accesses as well as the impact of the coherence protocol are not sufficiently covered by existing benchmarks. This necessitates the development of new benchmarks in order to consider all potential bottlenecks in step 1. These highly optimized benchmarks can be configured to use individual components to their full capacity. Furthermore, the amount of data that is accessed by the benchmarks is known. This facilitates the identification of performance counters that correlate with the number of memory accesses in step 2. Step 3 shows that the identified counters can be used to analyze the influence of memory accesses on the achieved application performance. The contribution of this thesis is twofold:

- The newly developed micro-benchmarks enable an in-depth analysis of the memory performance of shared memory systems including the impact of cache coherence protocols. Their sophisticated design significantly advances the state-of-the-art in that area. The information that can be obtained using these benchmarks provides valuable input for the analytical performance modeling of shared memory systems [RH13; LHS13; PGB14; RH16].
- The methodology for the identification of meaningful performance counters greatly improves the ability of performance engineers to attribute performance problems to their cause. Due to the careful construction of the micro-benchmarks it can be verified which performance counters actually correlate with the utilization of the memory hierarchy, which is an essential prerequisite for the performance counter based performance analysis.

This document is organized as follows: Section 2 discusses related work and provides the required background knowledge. Section 3 describes the design and implementation of the micro-benchmarks. In Section 4, these benchmarks are used to analyze the characteristics of memory accesses on a contemporary NUMA system. This includes the properties of shared resources in multi-core processors and interconnects in multi-socket systems as well as the influence of the cache coherence protocol. Section 5 presents the methodology to identify meaningful performance counters as well as the visualization of memory related performance problems. The verification of the number of reported events using the micro-benchmarks shows that making assumptions based on the name of an event can easily result in wrong conclusions. Section 6 concludes the thesis with a summary.

2 Background and Related Work

Shared memory systems are commonly used in a wide range of electronic equipment from Smartphones to HPC systems. Processors are a basic component of all shared memory systems. Their structure and principle of operation as well as the construction of multi-processor systems is discussed in Section 2.1. Cache coherence protocols, which are required to maintain a consistent view on the shared memory for all connected processors, are detailed in Section 2.2. Section 2.3 introduces established performance evaluation techniques and performance analysis tools.

2.1 Processor and System Architecture

The number of transistors in integrated circuits increases by a factor of two in approximately 24 month [Moo75]. The primary challenge of processor development is to turn the steadily increasing number of transistors into more performance. The available instruction level parallelism is limited and extracting it requires excessive control logic [HP06, Chapter 3]. Therefore, further enhancements of the number of instructions per cycle are increasingly hard. Furthermore, the huge clock rate improvements that have been common since the 1990s came to an end around 2002 [DAS12, Section 1.2, Figure 1.5]. Consequently, the focus of processor development shifted towards increasing the number of processor cores in the early 2000s [HP06, Section 3.8]. The basic structure of a multi-core processor is depicted in Figure 2.
The performance of processors improves faster than the memory technology, which leads to the so-called processor-DRAM performance gap [HP06, Section 5.1]. Memory accesses can take hundreds of clock cycles [MHS14; Mol+15]. Therefore, contemporary processors implement multiple levels of cache that store frequently used data in order to bridge the processor-DRAM performance gap. The memory hierarchy is often represented as a pyramid with the level one cache being the highest and main memory being the lowest level of primary memory [DAS12, Section 4.2]. The usually very small level one cache typically supports multiple requests each cycle and delivers data within a few cycles. Each additional cache level features a higher capacity. However, the bandwidth decreases and the latency increases with each level [Mol+09].
Many workstations and servers contain multiple processors, e.g., [Hew14; Fuj14]. Multi-processor systems are also used as building blocks for larger systems, e.g., [Bul13]. Modern multi-processor systems typically have integrated memory controllers [Int09, Figure 6]. Thus, the total memory bandwidth scales with the number of processors. However, distributed memory controllers also have a downside. The characteristics of memory accesses depend on the distance between the requesting core and the memory controller that services the request. This *non-uniform memory access* (NUMA) behavior [HP06, Section 4.1] needs to be taken into account in order to achieve good performance.

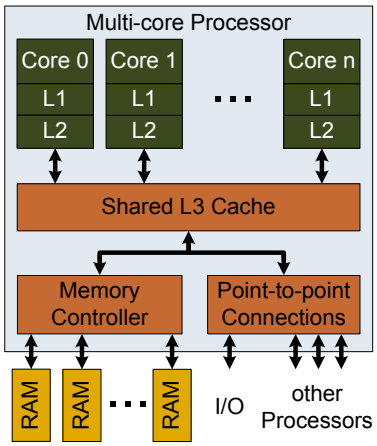


Figure 2: Composition of multi-core processors, based on [Con+10, Figure 1] (derived from [Mol+10, Figure 1]): Several processor cores are integrated on a single die. Typically, the level one caches (L1) are duplicated as well [DAS12, Section 8.4.1]. Separate level two caches (L2) for each core also are a common feature [Int14a, Section 2.1, 2.2, and 2.4]. However, certain supporting components are usually shared by all cores [DAS12, Section 8.4.1]. The last level cache (LLC), the integrated memory controller (IMC), and the processor interconnects are often implemented as shared resources.

2.2 Cache Coherence

Multiple copies of the same memory address can coexist in caches of different cores as well as in different cache levels. Cache coherence ensures that modifications made by any core eventually become visible for all cores. This is a prevalent feature in general purpose processors [BDM09]. Coherence protocols require a mechanism to perceive accesses by other processors. This can be implemented with *snooping* or using *directories* [HP06, Section 4.2 – 4.4]. *Snooping-based coherence protocols* observe all memory accesses via a shared bus or a broadcast network. This can generate a considerable amount of additional traffic, which can be reduced using snoop filters [Con+10]. *Directory-based coherence protocols* eliminate broadcasts entirely.
Coherence protocols assign a state to each cache line that changes according to a state change diagram if the memory location is accessed [DAS12, Section 7.3.2]. For instance, the *MESI* protocol [PP84] uses four states: *Modified* (M), *Exclusive* (E), *Shared* (S) and *Invalid* (I). The states *Exclusive* and *Modified* guarantee that there are no further copies. Cache lines are in the state *Shared* if multiple cores may contain a valid copy. *Invalid* cache lines do not contain useful data. They can be used to accommodate new data without replacing existing cache lines. Contemporary multi-socket systems use more sophisticated coherence protocols like MESIF [Int09] or MOESI [Amd15, Section 7.3]. Memory accesses can cause state transitions, which influences the performance of memory accesses [HLK97; Mol+09; MHS14].

2.3 Performance Evaluation

Performance evaluation includes measuring the performance of computer systems as well as testing how well applications are performing. The former typically involves benchmarks. Some commonly used benchmarks are discussed briefly in Section 2.3.1. The evaluation of applications is typically done using performance analysis tolls, e.g., *Vtune* [Int13], *Vampir* [Knü+08] or *Scalasca* [Gei+10]. This often involves hardware performance counters, which are described in Section 2.3.2.

2.3.1 Benchmarking

STREAM [McC95] measures the cache and memory bandwidths. The NUMA characteristics can be analyzed using additional tools (e.g., `numactl`). However, *STREAM* cannot be used to measure the bandwidth of remote cache accesses. The cache and memory latency can be measured with *lmbench* [MS96]. However, remote cache accesses as well as coherence protocol transactions cannot be analyzed. These restrictions also apply to the latency measurements of *X-Ray* [YPS05, Section 5]. *BlackjackBench* [Dan+13] determines cache and TLB parameters as well as the page size and the number of TLB entires. It also measures the bandwidth of core-to-core transfers. However, different coherence states and the latency of remote cache accesses are not included. *Likwid-bench* [THW12] measures the throughput of loop-kernels, which includes the bandwidth aggregated bandwidth of parallel memory accesses.

2.3.2 Hardware Performance Monitoring

Many contemporary processors include *performance monitoring units* (PMUs), e.g., [Amd13, Section 2.7]; [Int14b, Volume 3, Chapter 18]. The PMU typically contains multiple hardware performance counters, which can be programmed to count certain events, e.g., cache misses and snoop requests. Usually, each core has dedicated counters. Additional PMUs for the shared resources are common as well [Amd13, Section 2.7.2]; [Int12]. A widely-used tool to access PMUs of various processor architectures is the *Performance API* (PAPI) [Ter+09]. PAPI defines a standard interface to access performance counters from within the application in order to record performance data per thread.

3 Micro-benchmarks for Analyzing Memory Hierarchies

This section describes the design and implementation of the micro-benchmarks that are used to analyze the memory subsystem of shared memory systems [Mol+09; HMN09; Mol+10; MHS14; Mol+15]. *X86-membench* supports latency and bandwidth measurements for local and remote cache and memory accesses and complements them with a mechanism to control the coherence state of the accessed data. The benchmarks are implemented as kernels for the *BenchIT* framework [Juc+04]. *BenchIT* and *x86-membench* are available as open source[1]. The data placement and coherence state control mechanisms, which are described in Section 3.1 and Section 3.2, load data into certain cache levels and enforce a particular coherence state prior to the measurement. The measurement routines are described in Section 3.3.

3.1 Data Placement

The data placement prior to the measurement is implemented by accessing the whole data set multiple times in order to replace other data in the caches. After that the data resides in the highest level of the memory hierarchy that is large enough to accommodate all data and partially in higher levels unless all data fits into the L1 cache. Performing a latency measurement after this form of placement shows a mixture of effects from different cache levels. An optional cache flush routine can be used to place data only in a certain cache level or main memory. This enables precise performance measurements for the individual levels in the memory hierarchy. Data placement and measurement can be executed on different cores. This enables the analysis of core-to-core transfers. The measurement of core-to-core transfers is implemented as follows:

```
for(i = 0; i < num_selected_cpus; i++){
  thread on CPU[0]: if (i != 0) signal thread on CPU[i] to access its data set
  thread on CPU[i]: load data from CPU[i]'s buffer into local cache hierarchy
  thread on CPU[0]: if (i != 0) wait until other thread finishes data placement
  thread on CPU[0]: perform measurement using the buffer assigned to CPU[i]
}
```

The first iteration evaluates the local memory hierarchy. The remaining iterations determine the performance of remote cache and memory accesses.

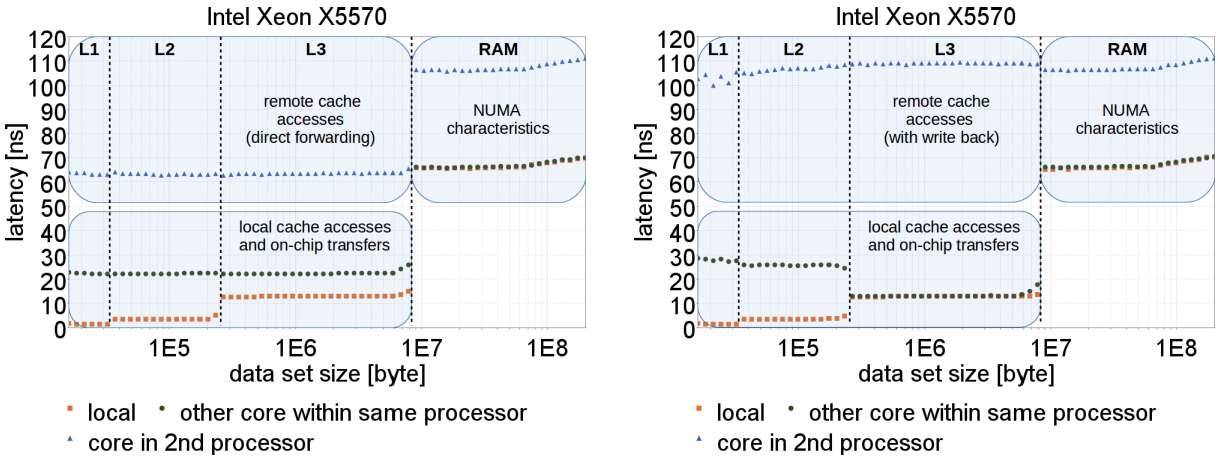
3.2 Coherence State Control

With the coherence state control mechanism [Mol+09; HMN09; MHS14] the impact of the coherence protocol can be analyzed. This is an essential capability of *x86-membench*. The example in Figure 3 shows how the coherence state influences the latency of cache accesses.
The coherence states *Modified* and *Exclusive* are generated as follows, where thread N is pinned to core N and thread M is pinned to another core [MHS14, Section 3.3]:

- *Modified* state in caches of core N is generated by:
  1) thread N: writing the data (invalidates all other copies of the cache line)
- *Exclusive* state in caches of core N is generated by:
  1) thread N: writing the data to invalidate copies in other caches,
  2) thread N: invalidating its cache using the CLFLUSH instruction,
  3) thread N: reading the data

The coherence state control mechanism supports all states of the MESI, MESIF, and MOESI coherence protocols [MHS14].

[1] https://fusionforge.zih.tu-dresden.de/frs/?group_id=885



(a) Cache lines in state *Exclusive*

(b) Cache lines in state *Modified*

Figure 3: Data is cached with a certain coherence state, which determines the required coherence state transitions as well as the source of the response (direct cache-to-cache forwarding or via main memory). These figures are based on [Mol+09, Figure 2].

3.3 Measurement Routines

The measurement routines are implemented using inline assembler. The high-resolution timestamp counter (TSC) is used to measure durations. All measurement routines can also record hardware performance counters (see Section 2.3.2) in addition to the performance metrics.

Latency Benchmark: The latency measurement uses pointer-chasing to determine the latency of memory accesses, i.e., each load operation provides the address for the next access. At least 24 loads at randomly selected addresses are performed for each measurement. The number of accesses is increased for larger data set sizes in order to reduce variation in the results. The duration of the whole access sequence is divided by the number of accesses to determine the average latency of a single access.

Single-threaded Bandwidth Benchmarks: These benchmarks measure the bandwidths that can be achieved by a single thread. They perform sequential accesses to the whole data set in order to determine the bandwidth of reads, writes, or a mixture of both. Multiple widths of load and store instructions are supported. It is important to note that one core cannot write into another core's cache. Writes to remotely cached data show a combination of two effects: first, reading the data from its original location, and second, writing to the local cache hierarchy. This has to be considered in the interpretation of the results.

Aggregated Bandwidth Benchmarks: The aggregated bandwidth benchmarks measure the achievable bandwidth for a variable number of threads that perform concurrent memory accesses. The memory access sequences performed by the threads are identical to the single-threaded bandwidth benchmarks. The memory affinity can be specified independently from the CPU binding. This can be used to measure the interconnect bandwidth by binding all threads to CPUs in one NUMA node and allocating memory from another NUMA node.

Throughput of Arithmetic Instructions: This benchmark is an adopted versions of the multi-threaded bandwidth benchmark. In its measurement routines, loads are replaced by arithmetic instructions. The benchmark is also meant to investigate the power consumption [Mol+10]. In order to perform reliable power measurements the runtime has to be extended to multiple minutes in order to reach a stable temperature. This is implemented by accessing the whole data set multiple times. Therefore, the throughput benchmarks do not consider different coherence states as the initial coherence state of the data cannot be preserved.

4 Performance Characterization of Memory Accesses

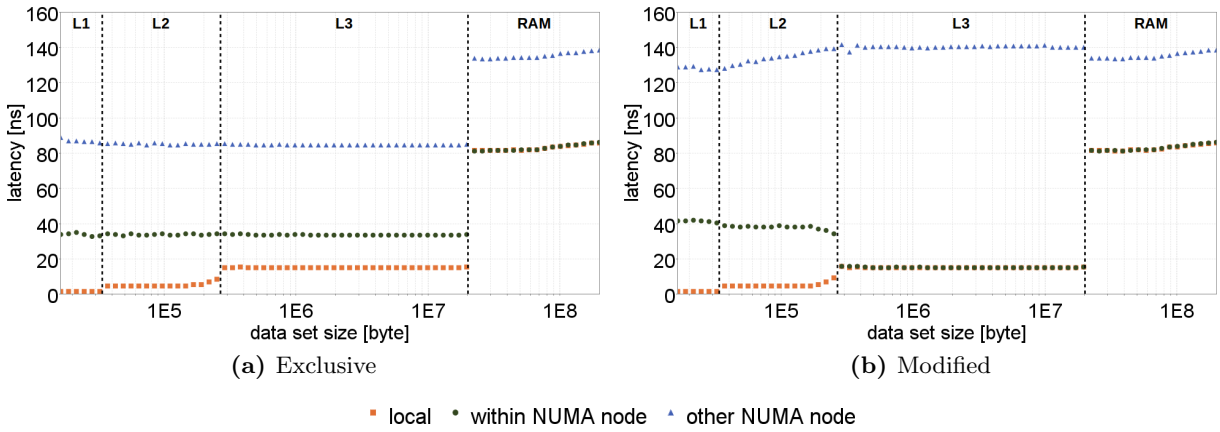
The benchmarks described in Section 3 enable an in-depth analysis of shared memory systems [Mol+09; HMN09; Mol+10; Mol+11; SHM12; MHS14; Mol+15]. This section shows this using the example of a dual-socket system with Intel Xeon E5-2670 processors [MHS14]. Each processor has eight cores, which share an inclusive 20 MiB L3 cache. The integrated memory controllers have four DDR3 channels, which are populated with PC3-12800R memory modules. This results in a theoretical peak bandwidth of 51.2 GB/s per socket. The two processors are connected with two QPI links. Together they can transfer 32 GB/s in each direction.

4.1 Latency of Cache and Main Memory Accesses

Figure 4 depicts latency measurements for different coherence states. The local L1 and L2 cache have a latency of 1.5 and 4.6 ns, respectively. L3 accesses cause an average delay of 15 ns. The additional latency for accesses to *Exclusive* cache lines that have been used by another core is caused by silent evictions, which do not clear the corresponding core valid bits [Mol+09]. The Snooping of another core increases the latency to 33.5 ns. Accesses to caches in the second processor and main memory have a higher latency. The remote L3 sends data with a delay of 84.6 ns if the cache lines are in state *Exclusive*. Cache lines in state *Modified* have a higher access latency of 127 – 141 ns as the data has to be written back to memory. The inclination shown for remote L1 and L2 accesses is caused by the decreasing likelihood of accessing already opened DRAM pages. The local main memory latency is measured with 81.5 ns while remote memory accesses take 134 ns.

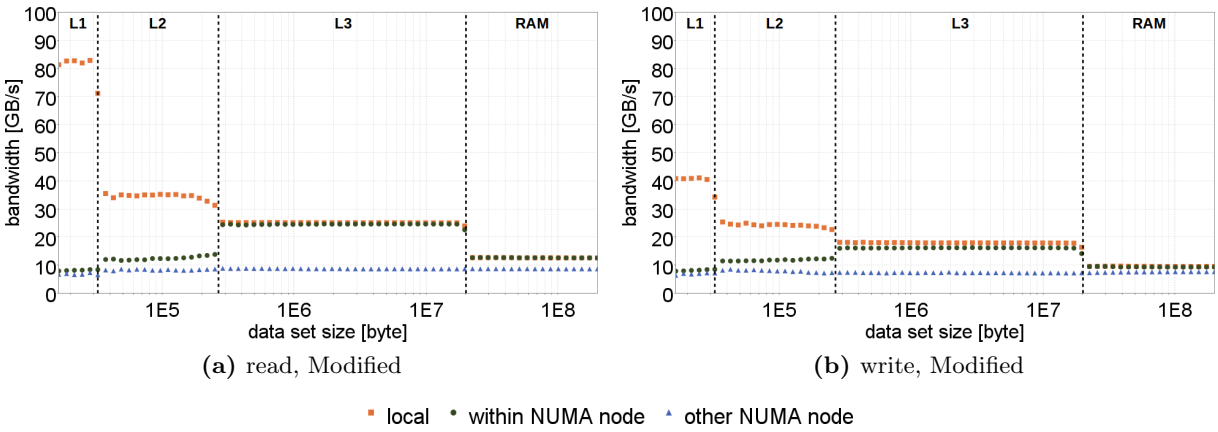
4.2 Bandwidth of Local Cache Accesses and Core-to-core Transfers

Figure 5 depicts the read and write bandwidth for accesses to *Modified* cache lines. The measured 82.8 GB/s for reads from the local L1 cache are close to the theoretical peak performance for two 128 Bit loads per cycle at 2.6 GHz. The L2 bandwidth reaches 35.2 GB/s. The inclusive L3 cache supports a read bandwidth of up to 25.1 GB/s. Data from other cores' L1 and L2 caches can be read with 8.2 and 12.0 GB/s, respectively. The write bandwidths are generally lower than the corresponding read bandwidths. The L1 and L2 cache support writes with 41.0 and 24.5 GB/s, respectively. The L3 cache can be written with up to 17.9 GB/s.



(a) Exclusive **(b)** Modified

Figure 4: Xeon E5-2670—memory read latency: Thread on core 0 accessing its local cache hierarchy (local) as well as cache lines of core 1 in the same processor (within NUMA node) and core 8 in the second processor (other NUMA node).



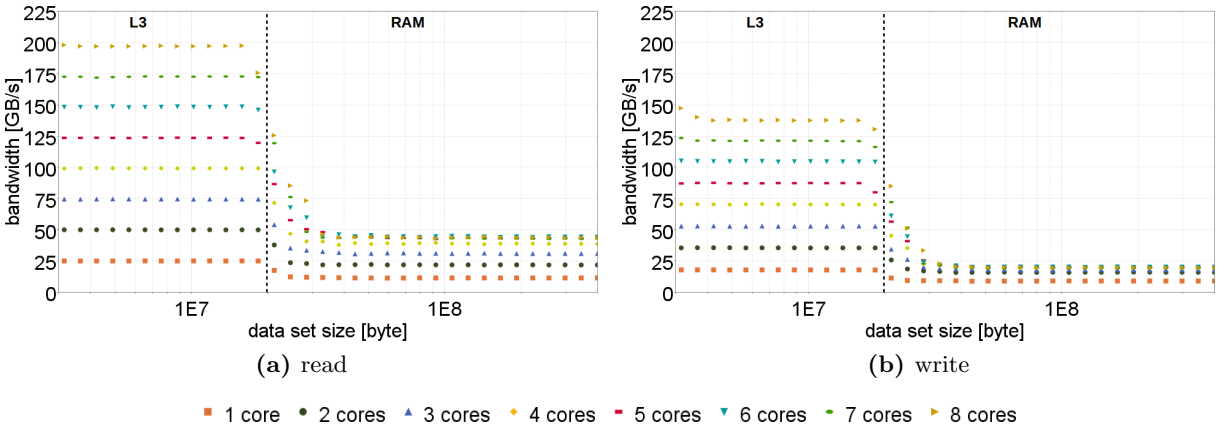
(a) read, Modified **(b)** write, Modified

Figure 5: Xeon E5-2670—single-threaded read and write bandwidths using 256 bit instructions (VMOVDQA): Thread on core 0 accesses data in its local memory hierarchy (local) as well as data that is present in caches of another core on the same chip (within NUMA node) and in the second processor (other NUMA node).

The bandwidths are significantly lower, if the second processor is involved. The read bandwidth from the remote L3 cache is limited to 8.7 GB/s. *Modified* data from L1 or L2 caches in the other NUMA node can only be read with 7.0 and 8.5 GB/s respectively. The write bandwidths are between 6.8 and 8.4 GB/s.

4.3 Bandwidth Scaling of Shared Resources

The aggregated bandwidth of one to eight concurrently reading and writing cores within one processor is depicted in Figure 6. The L3 performance scales almost linear with the number of cores. It reaches 199.6 GB/s. The memory bandwidth scales well up to four concurrently reading cores, which reach 38.9 GB/s. Using more cores slightly increases the achieved bandwidth to approximately 44 GB/s. The two QPI links have a combined bandwidth of 32 GB/s in each direction. However, only 16.8 GB/s are reached in the default configuration.

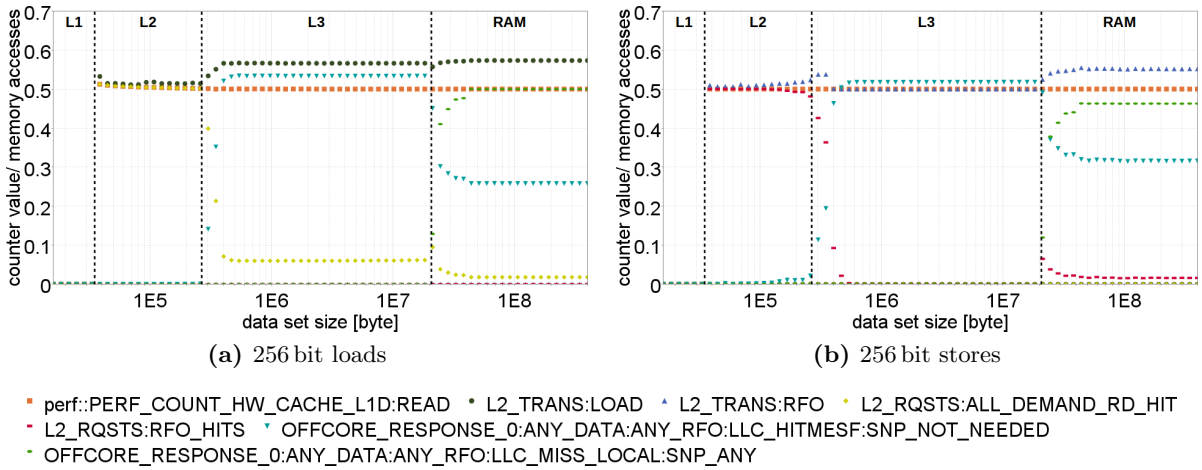


(a) read **(b)** write

Figure 6: Xeon E5-2670—bandwidth using multiple cores (one thread per core): The L3 bandwidth scales almost linear with the number of cores. The memory bandwidth can be fully utilized without using all cores. 256 bit loads and stores are used in these measurements.

5 Performance Impact of the Memory Hierarchy

The correlation of hardware performance counters with the utilization of the memory hierarchy is evaluated using the load and store variants of *x86-membench's* throughput kernel (see Section 3.3). The *perf::PERF_COUNT_HW_CACHE_L1D:READ* and *perf::PERF_COUNT_HW_CACHE_L1D:WRITE* events can be used to count the cache lines that are requested by (*:READ*) or written back (*:WRITE*) from the L1 cache. The remaining challenge is to find events that differentiate accesses to different levels in the memory hierarchy and distinguish local from remote memory accesses. Figure 7 depicts performance counter readings that dissect the data transferred to the L1 cache according to the data's prior location. The *L2_RQSTS* and *OFFCORE_RESPONSE:LLC_MISS_LOCAL* / *:LLC_MISS_REMOTE* events provide good estimates for the amount of data delivered from the L2 cache and main memory, respectively. The *OFFCORE_RESPONSE:LLC_HITMESF* event correctly represent the number of cache lines read from the L3 cache if the data is actually located in the L3 cache. However, main memory accesses (LLC misses) also cause a significant number of LLC hit events. Luckily, an estimate for the total number of requested cache lines is also available in the form of the *L2_TRANS* events. The difference between them and the number of cache lines delivered from main memory defines an upper bound for the number of cache lines delivered by the L3 cache, which can be used to compensate the overlap between the LLC hit and miss counters.
The measured peak bandwidths can be used to calculate the achievable number of transfers per second for the various locations in the memory hierarchy. Together with the observed number of events per memory access it is possible to derive the peak event ratios for the identified hardware performance counters. Unfortunately, the number of transfers per second cannot be used to derive the bandwidth utilization of the L1 cache. With 64 bit instructions it is possible to reach the maximal event rates while using only 50% of the available bandwidth. This could still be defined as 100% load as all L1 load or store ports are active each cycle. However, with 256 bit instructions it is also possible to fully utilize the available bandwidth while the event rates are at 50% of their respective maximum. For L2 accesses and beyond, performance counters provide good estimates for the used bandwidth independent of the width of the load and store instructions since events are reported on a per cache line basis. This can be used to investigate how applications utilize the memory hierarchy as depicted in Figure 8.



(a) 256 bit loads **(b)** 256 bit stores

Figure 7: Xeon E5-2670—counters that identify the source of the accessed data: The Utilization of the L2 cache can be measured via the L2_TRANS and L2_RQSTS events. L3 cache and main memory accesses can be recorded using the OFFCORE_RESPONSE counters. The correlation is not perfect but good enough to gain insight.

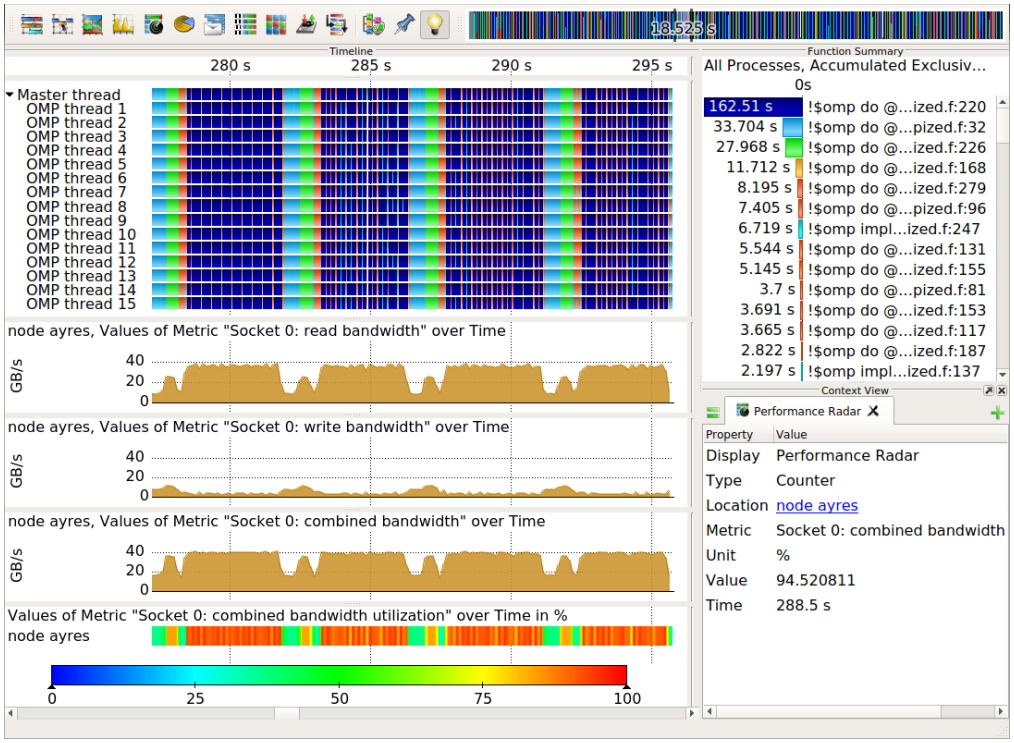


Figure 8: SPEComp2012, 351.bwaves—DRAM utilization in Vampir: Depicted are counter timelines for the read, write, and combined bandwidth of one processor. The display at the bottom shows the utilization of the memory controller using the performance radar [BW13]. In the red regions the bandwidth usage is close to the measured peak bandwidth.

6 Summary

This thesis introduces *x86-membench*—an open source micro-benchmarking suite that facilitates the performance analysis of memory accesses in cache coherent distributed shared memory systems. These benchmarks are used to perform an in-depth analysis of contemporary multiprocessor systems that identifies potential bottlenecks in the memory hierarchy. Furthermore, a methodology for the identification of meaningful hardware performance counters is presented that uses the micro-benchmarks to derive metrics for the utilization of individual components in the memory hierarchy and memory related waiting times from performance counter readings. These metrics can then be used to visualize memory related performance problems.

X86-membench is a versatile benchmark suite for the analysis of the memory hierarchy of complex shared memory systems. It extends the state-of-the-art in in various directions as shown in Table 1. While the local memory hierarchy and the impact of remote memory accesses in NUMA systems are sufficiently covered by existing benchmarks, the performance of remote cache accesses is not. The data placement mechanism described in Section 3.1 closes this gap. Furthermore, the coherence state control mechanism described in Section 3.2 can be used to measure the costs of coherence protocol transactions. The assembler implementation of the measurement routines leads to very accurate results. Moreover, the performance impact of SIMD instructions can be measured without having to rely on the compiler to properly vectorize the code.

The benchmarks expose potential bottlenecks in the memory hierarchy and the interconnection network between the processors [Mol+09; HMN09; Tho11; Mol+11; MHS14; Mol+15]. The obtained results regarding the impact of the coherence states on the characteristics of memory accesses facilitate the analytical performance modeling of cache coherent shared memory systems [RH13; LHS13; PGB14; RH16]. The benchmarks can also be used to analyze the energy

Table 1: *X86-membench* provides a wider functional range for analyzing the memory hierarchy than other established benchmarks, especially regarding the impact of the coherence protocol.

benchmark suite	latency / bandwidth			explicit SIMD support	instr. throughput with operands in register/memory	coherence protocol influence
	local cache & memory	remote memory	remote cache			
x86-membench	✓ / ✓	✓ / ✓	✓ / ✓	✓	✓ / ✓	✓
BlackjackBench	✓ / ✓	✓ / ✓	✗ / ✓	✗	✓ / ✗	✗
likwid-bench	✗ / ✓	✗ / ✓	✗ / ✗	✓	✓ / ✓	✗
X-Ray, P-Ray	✓ / ✓	✓ / ✓	✗ / ✗	✗	✓ / ✗	✗
lmbench, STREAM	✓ / ✓	✓ / ✓	✗ / ✗	✗	✗ / ✗	✗

consumption of data transfers and arithmetic operations [Mol+10] as well as to evaluate the potential for energy efficiency optimizations [SHM12]. Furthermore, the understanding of the throughput and power characteristics of data transfers and arithmetic operations has been taken into account during the development of the processor stress test FIRESTARTER [Hac+13]. Due to the extensive use of inline assembly, the implementation is tailored to the x86 architecture. However, the functional principle can be ported to other architectures [Old13].

The analysis of contemporary shared memory systems—which are the building blocks of many HPC systems—reveals several potential bottlenecks in the memory hierarchy. It is shown that the memory accesses latency can exceed the size of the out-of-order window, which stalls the execution. Furthermore, the bandwidths that are supported by the lower levels in the memory hierarchy are typically not sufficient to fully utilize the available computational performance. The bandwidth of shared caches and main memory does not necessarily scale linearly with the number of concurrently operating cores. Remote accesses are additionally limited by the point-to-point interconnections between the processors. The cache coherence protocols also have a strong influence on the characteristics of memory accesses.

Knowing the peak performance of the individual components is an essential prerequisite for the detection of memory related performance losses. However, in order to determine the impact of the memory hierarchy on the achieved application performance, the utilization of the various components while the program is running as well as the waiting times that are caused by memory accesses also have to be measured. Therefore, the presented methodology that derives meaningful metrics for the resource utilization and memory related stall cycles from hardware performance counters is another major contribution of this thesis. It is shown that resource limitations are reflected by certain hardware events in many cases. Unfortunately, the results obtained on one architecture cannot easily be transferred to other architectures as the set of available events as well as their definition and functionality can be different. Therefore, it cannot be recommended to rely on performance counter readings without validating that they are actually working as expected. *X86-membench* is ideally suited to perform such validations, which is an important improvement of the state-of-the-art in performance counter based performance analysis.

The novel approach for the selection of suitable counters and the determination of their respective possible range of values facilitates the detection of memory related performance issues. However, recording all metrics requires many runs since only a limited number of events can be counted concurrently. Nevertheless, many performance problems can be found using the presented visualization of the performance counter data. When revisiting the initially mentioned challenge of understanding the causes of limited application scaling within a node, this thesis provides both—the tools for establishing the technically possible upper limits for the performance of various components in the memory hierarchy as well as the means for measuring their impact on the achieved application performance.

References

[Amd13] *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors*. Revision: 3.14. Publication # 42301. AMD. Jan. 2013. URL: http://support.amd.com/TechDocs/42301_15h_Mod_00h-0Fh_BKDG.pdf (visited on 07/01/2016).

[Amd15] *AMD64 Architecture Programmer's Manual Volume 2: System Programming*. Revision 3.25. Publication # 24593. AMD. June 2015. URL: http://support.amd.com/TechDocs/24593.pdf (visited on 02/10/2015).

[Amd67] Gene M. Amdahl. "Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities". In: *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*. AFIPS '67 (Spring). ACM, 1967, pp. 483–485. DOI: 10.1145/1465482.1465560.

[Asl+01] Vishal Aslot et al. "SPEComp: A New Benchmark Suite for Measuring Parallel Computer Performance". In: *OpenMP Shared Memory Parallel Programming: International Workshop on OpenMP Applications and Tools, WOMPAT 2001*. Springer Berlin Heidelberg, 2001. ISBN: 978-3-540-44587-6. DOI: 10.1007/3-540-44587-0_1.

[BDM09] G. Blake, R.G. Dreslinski, and T. Mudge. "A survey of multicore processors". In: *Signal Processing Magazine, IEEE* 26.6 (Nov. 2009), pp. 26–37. ISSN: 1053-5888. DOI: 10.1109/MSP.2009.934110.

[BGB98] Luiz André Barroso, Kourosh Gharachorloo, and Edouard Bugnion. "Memory System Characterization of Commercial Workloads". In: *SIGARCH Comput. Archit. News* 26.3 (Apr. 1998), pp. 3–14. ISSN: 0163-5964. DOI: 10.1145/279361.279363.

[Bul13] *bullx DLC blade system - B700 series*. data sheet. Bull SAS, 2013. URL: http://www.bull.com/extreme-computing/download/S-bullxB700-en5.pdf (visited on 08/20/2014).

[BW13] Holger Brunst and Matthias Weber. "Custom Hot Spot Analysis of HPC Software with the Vampir Performance Tool Suite". In: *Proceedings of the 6th International Parallel Tools Workshop*. Springer Berlin Heidelberg, 2013, pp. 95–114. DOI: 10.1007/978-3-642-37349-7_7.

[Con+10] Pat Conway et al. "Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor". In: *IEEE Micro* 30.2 (Mar. 2010), pp. 16–29. ISSN: 0272-1732. DOI: 10.1109/MM.2010.31.

[Dan+13] Anthony Danalis et al. "BlackjackBench: Portable Hardware Characterization with Automated Results' Analysis". In: *The Computer Journal* (2013). DOI: 10.1093/comjnl/bxt057.

[DAS12] M. Dubois, M. Annavaram, and P. Stenström. *Parallel Computer Organization and Design*. Parallel Computer Organization and Design. Cambridge University Press, 2012. ISBN: 9780521886758.

[FC07] Wu-chun Feng and K.W. Cameron. "The Green500 List: Encouraging Sustainable Supercomputing". In: *Computer* 40.12 (Dec. 2007), pp. 50–55. ISSN: 0018-9162. DOI: 10.1109/MC.2007.445.

[FGD07] Karl Fürlinger, Michael Gerndt, and Jack Dongarra. "Scalability Analysis of the SPEC OpenMP Benchmarks on Large-Scale Shared Memory Multiprocessors". In: *Computational Science - ICCS 2007*. Vol. 4488. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2007, pp. 815–822. DOI: 10.1007/978-3-540-72586-2_115.

[Fuj14] *FUJITSU Server PRIMERGY RX4770 M1 Quad socket 4U rack server*. data sheet. Fujitsu, 2014. URL: http://globalsp.ts.fujitsu.com/dmsp/Publications/public/ds-py-rx4770-m1.pdf (visited on 08/19/2014).

[Gei+10] Markus Geimer et al. "The Scalasca performance toolset architecture". In: *Concurrency and Computation: Practice and Experience* 22.6 (Apr. 2010), pp. 702–719. ISSN: 1532-0626. DOI: 10.1002/cpe.1556.

[Hac+13] Daniel Hackenberg et al. "Introducing FIRESTARTER: A processor stress test utility". In: *Green Computing Conference (IGCC), 2013 International*. 2013. DOI: 10.1109/IGCC.2013.6604507.

[Hew14] *HP ProLiant ML350e Gen8 v2 Server data sheet*. data sheet. Hewlett-Packard, 2014. URL: http://www8.hp.com/h20195/v2/GetPDF.aspx/4AA5-0651ENW.pdf?ver=0 (visited on 08/19/2014).

[HLK97] Cristina Hristea, Daniel Lenoski, and John Keen. "Measuring Memory Hierarchy Performance of Cache-coherent Multiprocessors Using Micro Benchmarks". In: *Proceedings of the 1997 ACM/IEEE Conference on Supercomputing*. SC '97. ACM, 1997. ISBN: 0-89791-985-8. DOI: 10.1145/509593.509638.

[HMN09] D. Hackenberg, D. Molka, and W. E. Nagel. "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems". In: *MICRO 42: Proceedings of the 42nd Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2009, pp. 413–422. ISBN: 978-1-60558-798-1.

[HP06] John L. Hennessy and David A. Patterson. *Computer Architecture - A Quantitative Approach*. 4th edition. Morgan Kaufmann Publishers, 2006. ISBN: 9780123704900.

[Int09] *An Introduction to the Intel® QuickPath Interconnect*. Intel. Jan. 2009. URL: http://www.intel.com/technology/quickpath/introduction.pdf (visited on 10/23/2015).

[Int12] *Intel® Xeon® Processor E5-2600 Product Family Uncore Performance Monitoring Guide*. Intel. Mar. 2012. URL: http://www.intel.com/content/dam/www/public/us/en/documents/design-guides/xeon-e5-2600-uncore-guide.pdf (visited on 02/06/2015).

[Int13] *Intel® VTune™ Amplifier XE 2013*. 2013. URL: https://software.intel.com/sites/default/files/managed/0b/93/Intel-VTune-Amplifier-XE-overview-and-new-features.pdf (visited on 02/23/2016).

[Int14a] *Intel® 64 and IA-32 Architectures Optimization Reference Manual*. Order Number: 248966-029. Intel. Mar. 2014. URL: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf (visited on 05/01/2014).

[Int14b] *Intel® 64 and IA-32 Architectures Software Developer's Manual, Combined Volumes 1, 2A, 2B, 2C, 3A, 3B and 3C*. Order Number: 325462-050US. Intel. Feb. 2014. URL: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-manual-325462.pdf (visited on 05/01/2014).

[Juc+04] G. Juckeland et al. "BenchIT – Performance measurement and comparison for scientific applications". In: *Parallel Computing — Software Technology, Algorithms, Architectures and Applications*. Vol. 13. Advances in Parallel Computing. North Holland Publishing Co., 2004, pp. 501–508. DOI: 10.1016/S0927-5452(04)80064-9.

[Knü+08] Andreas Knüpfer et al. "The Vampir Performance Analysis Tool-Set". In: *Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing*. Springer Berlin Heidelberg, 2008, pp. 139–155. ISBN: 978-3-540-68564-7. DOI: 10.1007/978-3-540-68564-7_9.

[LHS13] Shigang Li, Torsten Hoefler, and Marc Snir. "NUMA-aware Shared-memory Collective Communication for MPI". In: *Proceedings of the 22nd International Symposium on High-performance Parallel and Distributed Computing*. HPDC'13. ACM, 2013, pp. 85–96. ISBN: 978-1-4503-1910-2. DOI: 10.1145/2462902.2462903.

[McC95] John D. McCalpin. "Memory Bandwidth and Machine Balance in Current High Performance Computers". In: *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter* (Dec. 1995), pp. 19–25. URL: http://www.cs.virginia.edu/~mccalpin/papers/bandwidth/bandwidth.html (visited on 02/08/2016).

[MG11] Zoltan Majo and Thomas R. Gross. "Memory System Performance in a NUMA Multicore Multiprocessor". In: *Proceedings of the 4th Annual International Conference on Systems and Storage*. SYSTOR'11. ACM, 2011, 12:1–12:10. ISBN: 978-1-4503-0773-4. DOI: 10.1145/1987816.1987832.

[MHS14] Daniel Molka, Daniel Hackenberg, and Robert Schöne. "Main Memory and Cache Performance of Intel Sandy Bridge and AMD Bulldozer". In: *Proceedings of the Workshop on Memory Systems Performance and Correctness*. MSPC '14. ACM, 2014, 4:1–4:10. ISBN: 978-1-4503-2917-0. DOI: 10.1145/2618128.2618129.

[Mol+09] D. Molka et al. "Memory Performance and Cache Coherency Effects on an Intel Nehalem Multiprocessor System". In: *PACT '09: Proceedings of the 2009 18th International Conference on Parallel Architectures and Compilation Techniques*. IEEE Computer Society, 2009, pp. 261–270. ISBN: 978-0-7695-3771-9. DOI: 10.1109/PACT.2009.22.

[Mol+10] Daniel Molka et al. "Characterizing the Energy Consumption of Data Transfers and Arithmetic Operations on x86-64 Processors". In: *Proceedings of the 1st International Green Computing Conference*. IEEE, 2010, pp. 123–133. DOI: 10.1109/GREENCOMP.2010.5598316.

[Mol+11] Daniel Molka et al. "Memory Performance and SPEC OpenMP Scalability on Quad-Socket x86_64 Systems". In: *Algorithms and Architectures for Parallel Processing*. Vol. 7016. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2011, pp. 170–181. ISBN: 978-3-642-24650-0. DOI: 10.1007/978-3-642-24650-0_15.

[Mol+15] Daniel Molka et al. "Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture". In: *Proceedings of the 44th International Conference on Parallel Processing (ICPP'15)*. IEEE, Sept. 2015. DOI: 10.1109/ICPP.2015.83.

[Moo75] G.E. Moore. "Progress in digital integrated electronics". In: *Electron Devices Meeting, 1975 International*. Vol. 21. 1975, pp. 11–13. DOI: 10.1109/N-SSC.2006.4804410(reprint).

[MS96] Larry McVoy and Carl Staelin. "lmbench: portable tools for performance analysis". In: *Proceedings of the 1996 annual conference on USENIX Annual Technical Conference*. ATEC '96. USENIX Association, 1996. URL: https://www.usenix.org/legacy/publications/library/proceedings/sd96/full_papers/mcvoy.pdf (visited on 02/06/2016).

[Mül+12] Matthias S. Müller et al. "SPEC OMP2012 - An Application Benchmark Suite for Parallel Systems Using OpenMP". In: *OpenMP in a Heterogeneous World: 8th International Workshop on OpenMP, IWOMP 2012, Proceedings*. Vol. 7312. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, pp. 223–236. ISBN: 978-3-642-30960-1. DOI: 10.1007/978-3-642-30961-8_17.

[Old13] Roland M. Oldenburg. "Analyse der Speicherhierarchie bei ARM Cortex A9 Prozessoren". Großer Beleg. Technische Universität Dresden, 2013.

[Oli+05] Leonid Oliker et al. “A Performance Evaluation of the Cray X1 for Scientific Applications”. In: *High Performance Computing for Computational Science - VECPAR 2004: 6th Intl. Conference, Revised Selected and Invited Papers*. Springer Berlin Heidelberg, 2005, pp. 51–65. ISBN: 978-3-540-31854-5. DOI: 10.1007/11403937_5.

[PGB14] B. Putigny, B. Goglin, and D. Barthou. “A benchmark-based performance model for memory-bound HPC applications”. In: *High Performance Computing Simulation (HPCS), 2014 International Conference on*. July 2014, pp. 943–950. DOI: 10.1109/HPCSim.2014.6903790.

[PP84] Mark S. Papamarcos and Janak H. Patel. “A Low-overhead Coherence Solution for Multiprocessors with Private Cache Memories”. In: *SIGARCH Comput. Archit. News* 12.3 (Jan. 1984), pp. 348–354. ISSN: 0163-5964. DOI: 10.1145/773453.808204.

[RH13] Sabela Ramos and Torsten Hoefler. *Modeling communication in cache-coherent SMP systems: a case-study with Xeon Phi*. technical report. Scalable Parallel Computing Laboratory, ETH Zurich, Feb. 2013. URL: http://htor.inf.ethz.ch/publications/img/ramos-hoefler-cc-modeling.pdf (visited on 03/15/2016).

[RH16] S. Ramos and T. Hoefler. “Cache Line Aware Algorithm Design for Cache-Coherent Architectures”. In: *Parallel and Distributed Systems, IEEE Transactions on* PP.99 (2016). ISSN: 1045-9219. DOI: 10.1109/TPDS.2016.2516540.

[SHM12] Robert Schöne, Daniel Hackenberg, and Daniel Molka. “Memory performance at reduced CPU clock speeds: an analysis of current x86_64 processors”. In: *Proceedings of the 2012 USENIX conference on Power-Aware Computing and Systems*. HotPower’12. USENIX Association, 2012. URL: https://www.usenix.org/system/files/conference/hotpower12/hotpower12-final5.pdf (visited on 07/02/2016).

[Str+15] E. Strohmaier et al. “The TOP500 List and Progress in High-Performance Computing”. In: *Computer* 48.11 (Nov. 2015), pp. 42–49. ISSN: 0018-9162. DOI: 10.1109/MC.2015.338.

[Ter+09] Dan Terpstra et al. “Collecting Performance Data with PAPI-C”. In: *Proceedings of the 3rd International Workshop on Parallel Tools for High Performance Computing*. Springer, 2009, pp. 157–173. ISBN: 978-3-642-11261-4. DOI: 10.1007/978-3-642-11261-4_11.

[Tho11] Michael E. Thomadakis. *The Architecture of the Nehalem Processor and Nehalem-EP SMP Platforms*. Tech. rep. Texas A&M University, 2011. URL: http://sc.tamu.edu/systems/eos/nehalem.pdf (visited on 10/15/2015).

[THW12] Jan Treibig, Georg Hager, and Gerhard Wellein. “likwid-bench: An Extensible Microbenchmarking Platform for x86 Multicore Compute Nodes”. In: *Proceedings of the 5th International Workshop on Parallel Tools for High Performance Computing*. Springer Berlin Heidelberg, 2012, pp. 27–36. ISBN: 978-3-642-31476-6. DOI: 10.1007/978-3-642-31476-6_3.

[Vet15] Jeffrey S. Vetter. *Contemporary High Performance Computing: From Petascale toward Exascale, Volume Two*. Chapman & Hall/CRC Computational Science. CRC Press, Mar. 2015. ISBN: 9781498700634.

[YPS05] Kamen Yotov, Keshav Pingali, and Paul Stodghill. “Automatic Measurement of Memory Hierarchy Parameters”. In: *SIGMETRICS Perform. Eval. Rev.* 33.1 (June 2005), pp. 181–192. ISSN: 0163-5999. DOI: 10.1145/1071690.1064233.