Comparing Cache Architectures and Coherency Protocols on x86-64 Multicore SMP Systems

Daniel Hackenberg Daniel Molka Wolfgang E. Nagel

Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden, 01062 Dresden, Germany
{daniel.hackenberg, daniel.molka, wolfgang.nagel}@tu-dresden.de

ABSTRACT

Across a broad range of applications, multicore technology is the most important factor that drives today's microprocessor performance improvements. Closely coupled is a growing complexity of the memory subsystems with several cache levels that need to be exploited efficiently to gain optimal application performance. Many important implementation details of these memory subsystems are undocumented. We therefore present a set of sophisticated benchmarks for latency and bandwidth measurements to arbitrary locations in the memory subsystem. We consider the coherency state of cache lines to analyze the cache coherency protocols and their performance impact. The potential of our approach is demonstrated with an in-depth comparison of ccNUMA multiprocessor systems with AMD (Shanghai) and Intel (Nehalem-EP) quad-core x86-64 processors that both feature integrated memory controllers and coherent point-to-point interconnects. Using our benchmarks we present fundamental memory performance data and architectural properties of both processors. Our comparison reveals in detail how the microarchitectural differences tremendously affect the performance of the memory subsystem.

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement techniques

General Terms

Performance Measurement

Keywords

Nehalem, Shanghai, Multi-core, Coherency, Benchmark

1. INTRODUCTION

Multicore technology is established in desktop and server systems as well as in high performance computing. The AMD Opteron 2300 family (Shanghai/Barcelona) and Intel's Xeon 5500 processors (Nehalem-EP) are native x86-64 quad-core designs for dual-socket systems. They integrate on-chip memory controllers that reduce the memory latency and allow the memory bandwidth to scale with the processor count in an SMP system. Serial point-to-point links provide cache-coherent inter-processor connections, thus creating a *cache-coherent non-uniform memory access* (ccNUMA) architecture. Both processor families feature a three level cache hierarchy and shared last level caches.

The evolution of processor technology towards multicore designs drives software technology as well. Parallel programming paradigms become increasingly attractive to leverage all available resources. This development goes along with a change of memory access patterns: While independent processes typically work on disjoint memory, parallel applications often use multiple threads to work on a shared data set. Such parallelism, e.g. producer-consumer problems as well as synchronization directives inherently require data movement between processor cores that is preferably kept on-chip. Common memory benchmarks such as STREAM [11] are no longer sufficient to adequately assess the memory performance of state-of-the-art processors as they do not cover important aspects of multicore architectures such as bandwidth and latency between different processor cores.

To improve this situation, we present a set of benchmarks that can be used to determine performance characteristics of memory accesses to arbitrary locations in multicore, multiprocessor ccNUMA systems. This includes on- and off-chip cache-to-cache transfers that we consider to be of growing importance. We also analyze the influence of the cache coherency protocol by controlling the coherency state of the cache lines that are accessed by our benchmarks.

Based on these benchmarks we compare the performance of the memory subsystem in two socket SMP systems with AMD Shanghai and Intel Nehalem processors. Both processors strongly differ in the design of their cache architecture, e.g. the last level cache implementation (non-inclusive vs. inclusive) and the coherency protocol (MOESI vs. MESIF). Our results demonstrate how these differences affect the memory latency and bandwidth. We reveal undocumented architectural properties and memory performance characteristics of both processors. While a high-level analysis of application performance is beyond the scope of this paper, we present fundamental data to facilitate such research.

The paper is organized as follows: Section 2 outlines characteristics of our test systems. Section 3 introduces the benchmarks that we use to generate the results in Sections 4 and 5. Section 6 presents related work. Section 7 concludes this paper and sketches future work.

MICRO'09, December 12–16, 2009, New York, NY, USA.

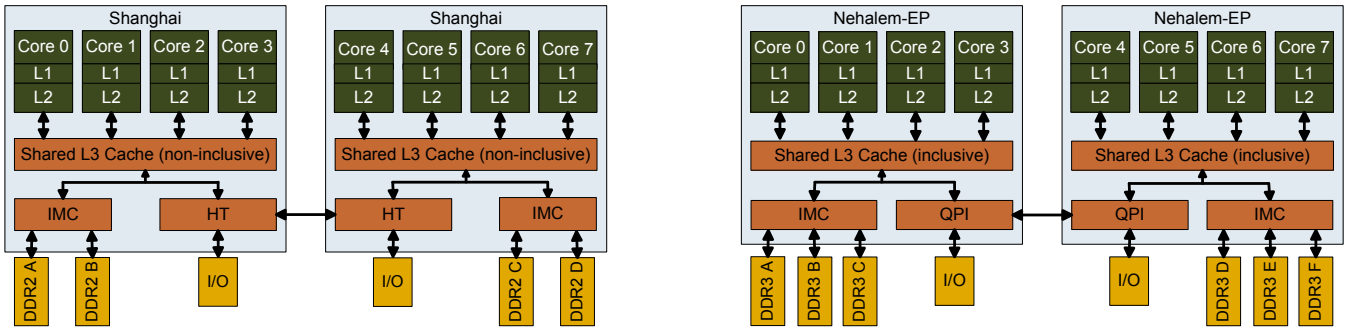


Figure 1: Block diagram of the AMD (left) and Intel (right) system architecture

2. BACKGROUND AND TEST SYSTEMS

Dual-socket SMP systems based on AMD Opteron 23** (Shanghai) and Intel Xeon 55** (Nehalem-EP) processors have a similar high level design as depicted in Figure 1. The L1 and L2 caches are implemented per core, while the L3 cache is shared among all cores of one processor. The serial point-to-point links *HyperTransport* (HT) and *Quick Path Interconnect* (QPI) are used for inter-processor and chipset communication. Moreover, each processor contains its own *integrated memory controller* (IMC).

Although the number of cores, clockrates, and cache sizes are similar, benchmark results can differ significantly. For example in SPEC's CPU2006 benchmark, the Nehalem typically outperforms AMD's Shanghai [1]. This is a result of multiple aspects such as different instruction-level parallelism, *Simultaneous Multi-Threading* (SMT), and Intel's Turbo Boost Technology. Another important factor is the architecture of the memory subsystem in conjunction with the cache coherency protocol [12].

While the basic memory hierarchy structure is similar for Nehalem and Shanghai systems, the implementation details differ significantly. Intel implements an inclusive last level cache in order to filter unnecessary snoop traffic. *Core valid bits* within the L3 cache indicate that a cache line may be present in a certain core. If a bit is not set, the associated core certainly does not hold a copy of the cache line, thus reducing snoop traffic to that core. However, unmodified cache lines may be evicted from a core's cache without notification of the L3 cache. Therefore, a set core valid bit does not guarantee a cache line's presence in a higher level cache.

AMD's last level cache is *non-inclusive* [6], i.e neither exclusive nor inclusive. If a cache line is transferred from the L3 cache into the L1 of any core the line can be removed from the L3. According to AMD this happens if it is "likely" [3] (further details are undisclosed) that the line is only used by one core, otherwise a copy can be kept in the L3. Both processors use extended versions of the well-known MESI [7] protocol to ensure cache coherency. AMD Opteron processors implement the MOESI protocol [2, 5]. The additional state *owned* (O) allows to share modified data without a write-back to main memory. Nehalem processors implement the MESIF protocol [9] and use the *forward* (F) state to ensure that shared unmodified data is forwarded only once.

The configuration of both test systems is detailed in Table 1. The listing shows a major disparity with respect to the main memory configuration. We can assume that Nehalem's three DDR3-1333 channels outperform Shanghai's two DDR2-667 channels (DDR2-800 is supported by the CPU but not by our test system). However, the main memory performance of AMD processors will improve by switching to new sockets with more memory channels and DDR3.

We disable Turbo Boost in our Intel test system as it introduces result perturbations that are often unpredictable. Our benchmarks require only one thread per core to access all caches and we therefore disable the potentially disadvantageous SMT feature. We disable the hardware prefetchers for all latency measurements as they introduce result variations that distract from the actual hardware properties. The bandwidth measurements are more robust and we enable the hardware prefetchers unless noted otherwise.

Test system	Sun Fire X4140	Intel Evaluation Platform
Processors	2x AMD Opteron 2384	2x Intel Xeon X5570
Codename	Shanghai	Nehalem-EP
Core/Uncore frequency	2.7 GHz / 2.2 GHz	2.933 GHz / 2.666 GHz
Processor Interconnect	HyperTransport, 8 GB/s	QuickPath Interconnect, 25.6 GB/s
Cache line size	64 Bytes	
L1 cache	64 KiB/64 KiB (per core)	32 KiB/32 KiB (per core)
L2 cache	512 KiB (per core), exclusive of L1	256 KiB (per core), non-inclusive
L3 cache	6 MiB (shared), non-inclusive	8 MiB (shared), inclusive of L1 and L2
Cache coherency protocol	MOESI	MESIF
Integrated memory controller	yes, 2 channel	yes, 3 channel
Main memory	8 x 4 GiB DDR2-667, registered, ECC (4 DIMMS per processor)	6 x 2 GiB DDR3-1333, registered, ECC (3 DIMMS per processor)
Operating system	Debian 5.0, Kernel 2.6.28.1	

Table 1: Configuration of the test systems

3. BENCHMARK DESIGN

Modern microprocessors feature complex cache hierarchies that are designed to hide memory latencies and improve memory bandwidths. These hardware features are transparent for applications and only limited software control is available. This makes it difficult to create memory benchmarks that can provide detailed performance properties of a given cache architecture. The benchmark design is therefore carefully optimized to circumvent perturbing hardware properties. Moreover, we use the performance analysis tool BenchIT [10] to facilitate the benchmark implementation as well as the performance studies. Both BenchIT and the benchmarks are available as Open Source[1].

Our benchmarks provide compiler independent assembler routines that allow us to use certain instruction sequences that can not be generated using high level languages (e.g. to transfer data into registers without computing on it). However, only the measurement routine itself is programmed in assembler, the rest (e.g. thread synchronization and data placement in the caches) is written in C. The *Time Stamp Counter* is a high resolution timer that is supported by both processors and therefore used to measure durations. Each thread of the benchmark program is pinned to a certain processor core. In the following description, thread N always runs on core N. If not noted otherwise, memory is allocated with the `localalloc` policy of `numactl`, allowing us to unveil effects caused by the NUMA architecture.

Prior to the measurement, data is placed in the caches in a well-defined coherency state. These states are generated as follows:

- **M**odified state in caches of core N is generated by: Thread N writing the data, which also invalidates all copies that may exist in other cores.
- **E**xclusive state in caches of core N is generated by: 1) Thread N writing to the memory to invalidate copies in other caches, 2) Thread N invalidating its cache (`clflush` instruction), 3) Thread N reading the data.
- **S**hared state in caches of core N is generated by: 1) Thread N caching data in exclusive state, 2) Reading the data from another core.

The MESIF protocol used by the Nehalem processor provides the closely related states *shared* and *forward*. However, the *forward* state apparently is only relevant for the L3 cache [8]. A cache line that is shared among multiple processors will only be in *forward* state in one L3 cache. For example, if processor 0 requests a cache line that is in *shared* state in processor 1 and in *forward* state in processor 2, only the latter will send a response. Unfortunately, our two socket system is too small to analyze this in detail. The MOESI protocol of the AMD Shanghai processor uses the additional *owned* state. For reading, results are equal to the *modified* state, as data has to be transferred from the cache that owns the cache line. For write access, cache lines in *owned* state behave like shared cache lines, as copies of other cores have to be invalidated.

Memory benchmarks and especially latency measurements often show a mixture of effects from different cache levels rather than just one. In order to separate these effects, we explicitly place data in certain cache levels. If the data set used for the measurement does not fit into a certain cache level, a special cache flush routine completely replaces the data in this (and higher) cache levels with dummy data that is not accessed during the measurement. Another effect that potentially influences our benchmark results are translation lookaside buffer (TLB) misses. *Huge pages* are used to limit that effect. Both processors have sufficient level 1 data *translation lookaside buffers* (data TLB) entries for 2 MiB pages to cover memory sizes that exceed the total cache size.

The Latency Benchmark (see Section 4) uses pointer-chasing to determine the minimal latency of cache and main memory accesses. The following implementation model describes a latency measurement of core 0 accessing memory cached by core N:

```
1) thread 0: warm-up TLB
2) if (N>0): sync of thread 0 and N
3) thread N: access data (-> E/M/S)
4) if (N>0): sync of thread 0 and N
5) all threads: flush caches (optional)
6) thread 0: measure latency
```

Step 1 ensures that all TLB entries needed for the measurement are always present in core 0. Step 3 places data in the caches of core N in a well-defined coherency state. Step 5 optionally flushes the caches. Step 6 is the final latency measurement and always runs on core 0. The routine performs a constant number of pseudo-random accesses. Every cache line is accessed only once to avoid reuse of cache lines. No consecutive cache lines are accessed to eliminate the influence of the adjacent line prefetcher.

The Single-Core Bandwidth Benchmark (see Sections 5.1 - 5.3) has a similar structure as the latency benchmark and performs steps 1 to 5 identically. We again control the cache line coherency states to unveil the effects of the coherency protocol. We determine local, inter-core, and inter-processor bandwidths.

The measurement routine running on core 0 (step 6) differs from the latency benchmark. It consecutively accesses a variable amount of data to determine the read or write bandwidth that is available for a single core. We solely use transport instructions (`MOVDQA`) to load or store data (`MOVNTDQ` in case of non-temporal stores) in order to avoid being limited by arithmetic operations.

The Multi-Core Bandwidth Benchmark (see Section 5.4) uses multiple threads concurrently to determine the aggregate bandwidth for a variable number of cores accessing their caches or main memory. This is particularly helpful to determine the characteristics of shared resources such as the L3 cache.

For this benchmark it is highly important that all parallel threads are tightly synchronized. The benchmarks use the Time Stamp Counter (TSC) for this purpose and therefore require system-wide synchronous TSCs as provided by both test systems. The benchmark structure is as follows:

```
1) all threads: access data (-> E/M)
2) all threads: flush caches (optional)
3) all threads: barrier synchronization
4) thread 0: define start_time in future
5) all threads: wait for start_time
6) all threads: measure t_begin
7) all threads: access data (read or write)
8) all threads: measure t_end
9) duration = max(t_end) - min(t_begin)
```

[1] http://www.benchit.org

4. LATENCY RESULTS

We use the latency benchmark introduced in Section 3 to compare our test systems. We place cache lines in different locations of the memory subsystem with different coherency states. Our results are depicted in Figure 2 and summarized in Table 2. We detail many of the conclusions that can be drawn from these results in the following Sections.

4.1 On-Chip Latencies

The latencies to local caches are independent of the coherency state since data can be read directly in all states. Shanghai's 3 cycle L1 latency compares to 4 cycles on Nehalem while the latter provides faster access to the L2/L3 caches. The L3 latency results are only valid for the selected processors, as they depend on the core/uncore frequency ratio. In general, our L1-L3 latency results correspond well to the numbers that have been published by the vendors [3, 8].

The latencies to other cores on the same processor strongly depend on the cache line's coherency state. On Nehalem, shared cache lines can be accessed within 13 ns as the copy in its inclusive L3 cache is guaranteed to be valid.[2] In contrast, exclusive cache lines (one core valid bit set) may have been modified in a higher level cache. This forces the L3 cache to check the coherency state in the core, thus increasing the latency to 22.2 ns. Due to the silent eviction from higher level caches, this penalty even occurs for cache lines that are solely present in the L3 cache. On the contrary, the eviction of modified cache lines can not occur silently but results in a write-back to the L3 cache that updates the core valid bits as well. Therefore, the L3 latency is again 13 ns. Modified data that is still present in higher cache levels is requested from the L2/L1 cache, thus increasing the latency to 25.5/28.3 ns.

Shanghai's cache architecture differs strongly: Cache lines from higher level caches need to be fetched from the cores or main memory if no copy exists in the non-inclusive L3 cache. The latencies indicate that requests for shared and exclusive cache lines are serviced by main memory. The former case is common to avoid multiple cache line transfers. Shanghai's non-inclusive L3 cache shows an exclusive behavior here as there is no latency improvement for accesses to shared L1/L2 cache lines that would be caused by a copy in the L3 cache. For exclusive cache lines it would clearly be possible to mark them as shared and forward them to the requesting core without main memory access. This is apparently not supported by Shanghai's MOESI implementation.

[2]Multiple *shared* cases exist on Nehalem: 1) Shared between two cores of one processor, two core valid bits set, cache line likely marked exclusive in L3. 2) Shared between two processors, one core valid bit set in each, line in shared/forwarding state in L3. The L3 provides equal latencies in all cases.

An on-chip transfer (that avoids the main memory latency) only occurs if the coherency protocol requires another core's L1/L2 cache to provide modified data. Moreover, if the data was originally allocated on the other socket, the latency increases from 44 to 81 ns (not depicted). This suggest that the (on-chip) request is forwarded to the (off-chip) memory controller of the other Shanghai processor, thus requiring an additional HT hop. This is a known behavior of the dual-core Opteron [5] and apparently remains unchanged with the shared L3 cache. On Nehalem, requests for modified cache lines are handled on-chip in this case (latency remains at 25-28 ns).

4.2 Off-Chip Latencies

The latencies to the other processor include an additional penalty for the QPI/HT data transfer. Nehalem's inclusive last level cache provides fast access to unmodified content in the other processor's caches. For exclusive cache lines, the latency of 63 ns includes a snoop of one core. For shared cache lines, the remote L3 cache can answer requests without snooping (58 ns)[3]. The latency for modified cache lines is significantly higher (> 100 ns) due to write-backs to main memory that are required by the MESIF protocol.

On Shanghai, unmodified cache lines are again not fetched from the remote L1/L2 cache but from main memory. The slope for memory sizes up to 576 KiB is possibly caused by an uncore prefetcher that fights the pseudo-random memory access pattern. The L3 cache forwards exclusive data (83 ns) while shared cache lines are fetched from main memory (118 ns). Only modified cache lines are directly forwarded to the requesting core from all cache levels. The advantage of Shanghai's MOESI protocol is that the modified cache line in the remote cache can switch to the *owned* state and therefore does not need to be written back to main memory. However, the remote L3 could theoretically be faster than the remote L1/L2 (similar to local accesses).

Interestingly, accesses to remote caches on Shanghai are slower than accesses to the local RAM (83 vs. 77 ns). In the latter case, the completion of the RAM access (that occurs concurrent to the cache request) has to be delayed until the probe miss of the remote processor returns. Unlike on Nehalem, this is apparently quicker than a probe hit.

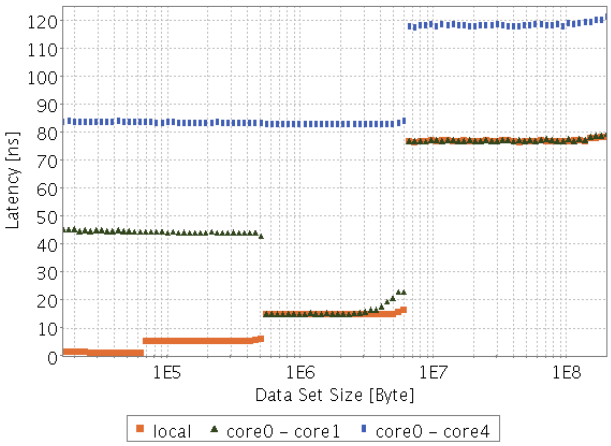
The latency to main memory is 65 ns for local memory and 106 ns for remote memory on the Nehalem system. When using more than 64 MiB of memory, the latency increases due to insufficient TLB entries (32 in L1 TLB). On Shanghai we measure a memory latency of 77 ns for local and 118 ns latency for remote accesses with stable results up to a data set size of 96 MiB (48 L1 TLB entries).

[3]The shared cache line is likely to be exclusive in the L3 cache of processor 1 with two core valid bits set.

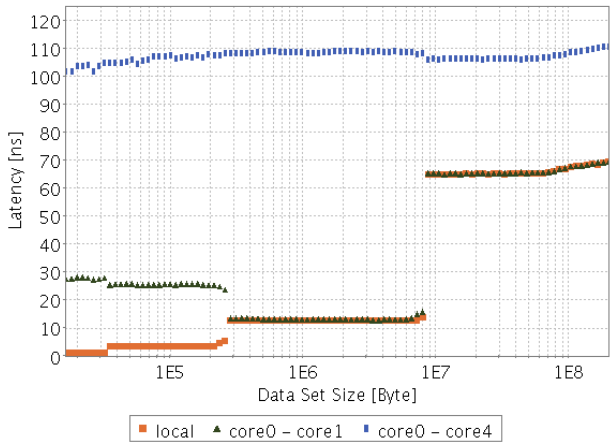
Processor		Shanghai				Nehalem			
Source	State	L1	L2	L3	RAM	L1	L2	L3	RAM
Local	M/E/S	1.1 (3)	5.6 (15)	15.2 (41)	77	1.3 (4)	3.4 (10)	13.0 (38)	65
Core1 (on die)	Modified	44 (119)				28.3 (83)	25.5 (75)	13.0 (38)	
	Exclusive	66 - 77				22.2 (65)			
	Shared					13.0 (38)			
Core4 (other die)	Modified	83 (224)		83 (224)	118	102 - 109			106
	Exclusive	99 - 116				63 (186)			
	Shared					58 (170)			

Table 2: Read latencies of core 0, all results in nanoseconds (cycles)

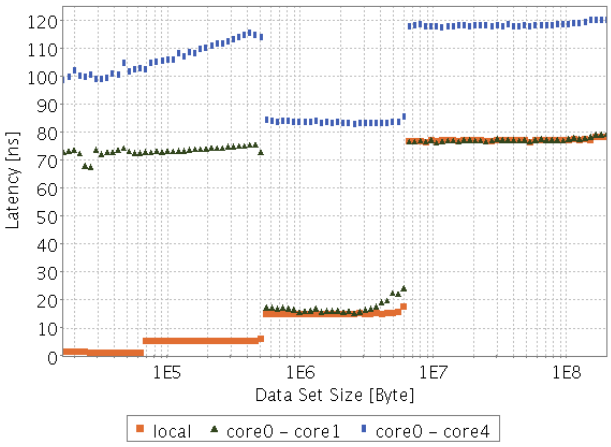
(a) Shanghai, modified



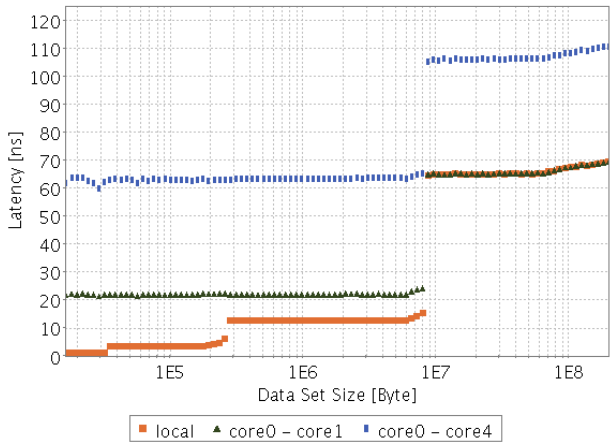
(b) Nehalem, modified



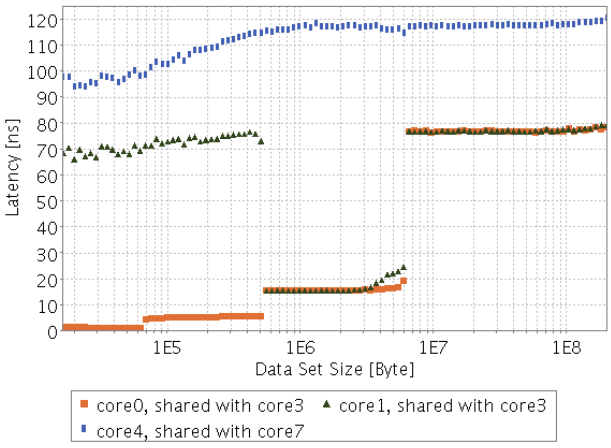
(c) Shanghai, exclusive



(d) Nehalem, exclusive



(e) Shanghai, shared



(f) Nehalem, shared

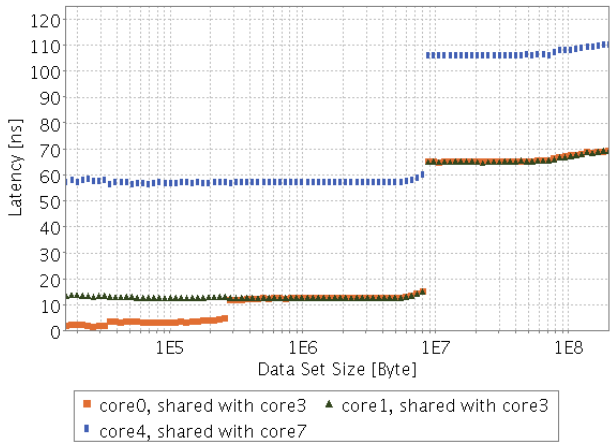


Figure 2: Read latencies of core 0 accessing cache lines of core 0 (local), core 1 (on die) or core 4 (other die)

5. BANDWIDTH RESULTS

In this section we compare the available read and write bandwidth of our test systems. For the interpretation it is important to note that all cache lines accessed by a core are placed in its L1 cache. Moreover, both microarchitectures implement a write-allocate cache policy (except for non-temporal stores). Any write access to a cache line that is shared with other cores or is not present in a local cache triggers a read for ownership that invalidates all copies in caches of other cores before actually writing the cache line. The write bandwidth results therefore combine effects of reading the data from its original location and writing it into the local caches. We use our designated cache flush routines (see Section 3) to ensure that data is fetched or written back solely to the intended cache level or main memory.

The results of our single-core bandwidth benchmark results show the available transfer rate when using a single core to access data located in certain cache levels of itself (see Section 5.1), of other cores on the same processor (see Section 5.2), and of cores on a different processor (see Section 5.3). Figure 3 plots the read bandwidth over the memory size for both exclusive, modified and shared cache lines. Table 3 summarizes the results. The results for write bandwidths are presented in Figure 4 and Table 4. The main memory bandwidth is discussed in Section 5.4 along with the bandwidth for concurrent accesses by multiple cores.

5.1 Bandwidth to Local Caches

The read bandwidth for local cache accesses is independent from the coherency state on both the Shanghai and the Nehalem system. In what is the single exception among all single core bandwidth results, Shanghai's two 128 Bit L1 read ports can achieve a bandwidth of 79.9 GB/s, exceeding the 45.6 GB/s that Nehalem can reach with only one 128 Bit port. Both processors nearly reach their theoretical peak bandwidth in this case. Shanghai's L2 read bandwidth of 21.5 GB/s (64 Bit/cycle) is likely caused by a 128 Bit interface that also has to transfer evicted data from the L1 cache due to the exclusive nature of the L2 cache. The L3 read bandwidth of 10.3 GB/s compares to 23.9 GB/s on Nehalem. Figure 3e shows that Shanghai's L3 bandwidth for shared cache lines increases marginally with higher memory sizes. This is a minor positive effect of the non-inclusive L3 cache that keeps copies of these shared cache lines and therefore reduces the write-back penalty for L2 evictions.

On Nehalem, the L1 write bandwidth equals the read bandwidth for exclusive and modified cache lines. Writing into the L2 and L3 is slightly slower than reading. Writing cache lines that are shared with another core is slower as the second copy has to be invalidated. However, write accesses generally achieve high bandwidths as the shared inclusive L3 cache allows all data transfers to occur on-chip.

On Shanghai, the low write bandwidth to shared cache lines (2.9 - 4.1 GB/s) shows that main memory accesses are performed. The L3 cache does not handle these transfers on-chip, even though the other processor is not involved. For modified data, the L1 write bandwidth of 41.3 GB/s is close to the theoretical peak (43.2 GB/s) for the two 64 Bit write ports. However, writing into the L2 and L3 cache only reaches 12.9 GB/s and 9.4 GB/s, respectively. A striking problem of Shanghai's microarchitecture are writes to exclusive cache lines. The L1 bandwidth is limited to 17.8 GB/s in this case, less than half compared to the modified case. This is likely caused by the non-inclusive L3 cache. Even though it generally behaves like an exclusive cache, there is a chance that a copy is kept in the L3 cache. This forces write access to exclusive L1 cache lines to check and eventually invalidate the L3 cache. The low L1 performance propagates into the L2 and L3 bandwidth as well.

5.2 Bandwidth to Other Cores

On Shanghai, the read bandwidth for unmodified data from other cores' L1 and L2 caches indicates that the requests are served by main memory. This is consistent with our latency measurements. For modified data that has to be transferred from another core's L1 or L2 cache, the bandwidth is only slightly higher (up to 6.8 GB/s). Reading from the L3 achieves 10.5 - 12.0 GB/s. The slope of these curves may be induced by an inclusive behavior of the L3 cache that keeps copies of the cache lines and therefore reduces write-back penalties.

Writing data cached by other cores is limited by the read for ownership. The write bandwidth reaches 3.9 GB/s for modified data that has to be fetched from another core's L1/L2 cache and written into the local cache. For unmodified data, bandwidth increases to 4.1 GB/s. The L3 cache can provide a write bandwidth of 9.4 GB/s, except for shared cache lines as they involve systemwide invalidations.

Nehalem has the advantage of an inclusive L3 cache that can provide a read bandwidth of 23.9 GB/s for any access to unmodified data of other cores. In contrast, the bandwidth for transfers of modified cache lines from another core's local L1 or L2 cache is significantly lower with 9.2 GB/s or 13.2 GB/s, respectively.

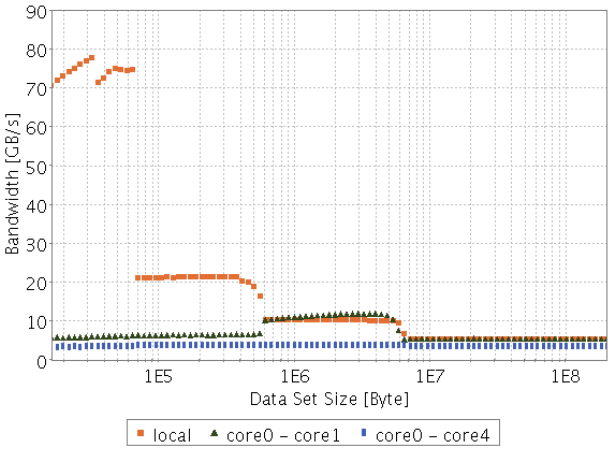
Processor		Shanghai				Nehalem			
Source	State	L1	L2	L3	RAM	L1	L2	L3	RAM
Local	all	79.9	21.5	10.3	5.5	45.6	31	23.9	13.9
Core1	M	5.5 - 6.8		10.5		9.2	13.2		
	E	4.5 - 5.4		-		23.9			
	S	5.6		12					
Core4	M	4		4	3.6	8.6			9.1
	E	3.6				10.1			
	S								

Table 3: Core 0 read bandwidth in GB/s with cache lines in Modified, Exclusive or Shared state

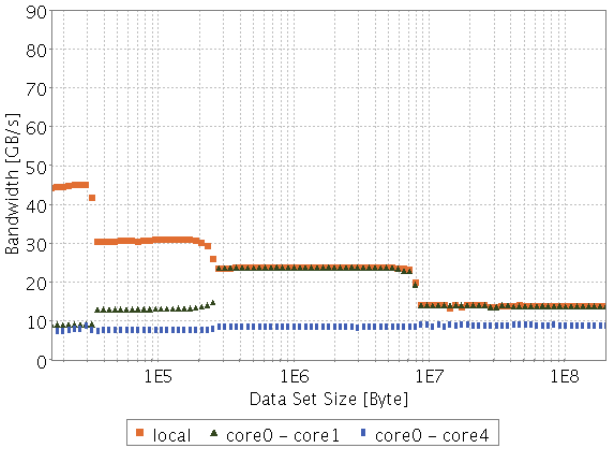
Processor		Shanghai				Nehalem			
Source	State	L1	L2	L3	RAM	L1	L2	L3	RAM
Local	M	41.3	12.9	9.4	3.6	45.6	28.8	19.3	8.9
	E	17.9	9.4						
	S	3	2.9	4.1		25.6	21.4	16.5	
Core1	M	3.9		9.4		9.4	13.4	19.3	
	E	4.1				25.6	21.4	16.5	
	S	4.1							
Core4	M	2.9		2.9	3.0	8.8	10		5.9
	E	3.4				9.9		9.3	
	S								

Table 4: Core 0 write bandwidth in GB/s with cache lines in Modified, Exclusive or Shared state

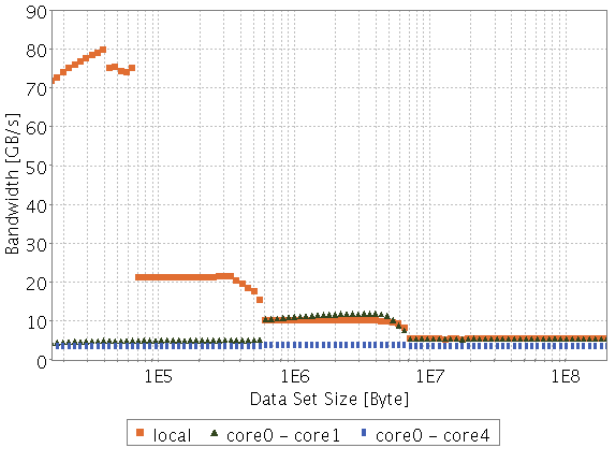
(a) Shanghai, modified



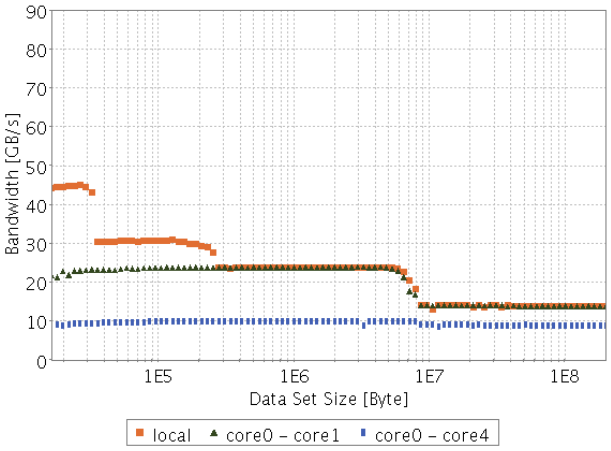
(b) Nehalem, modified



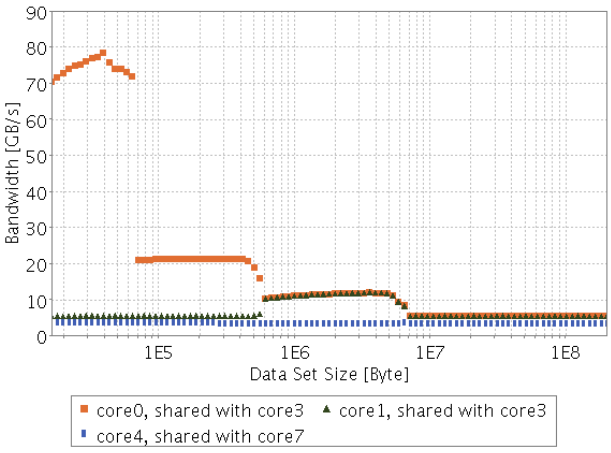
(c) Shanghai, exclusive



(d) Nehalem, exclusive



(e) Shanghai, shared



(f) Nehalem, shared

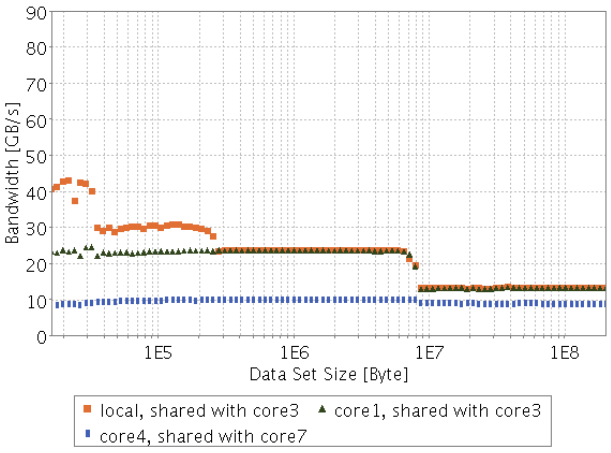
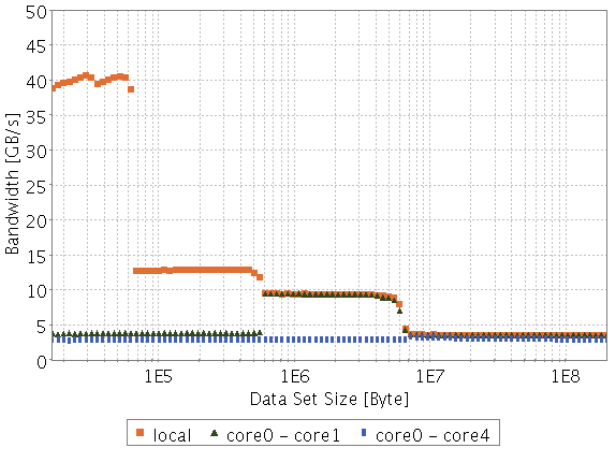
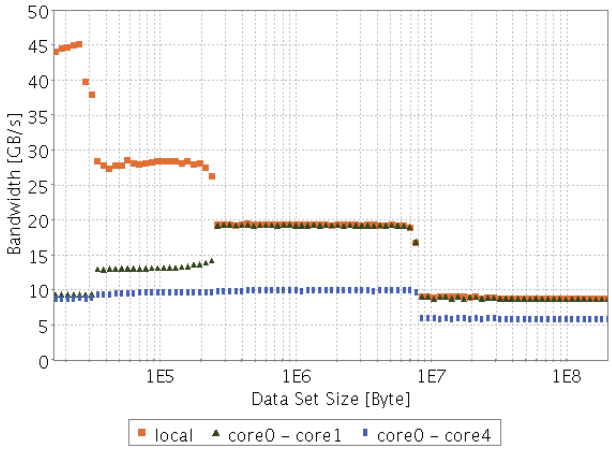


Figure 3: Read bandwidth of core0 accessing cache lines of core0 (local), core1 (on die) or core4 (other die)

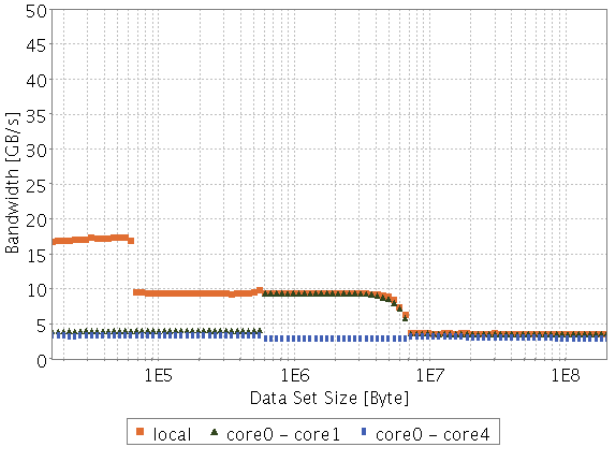
(a) Shanghai, modified



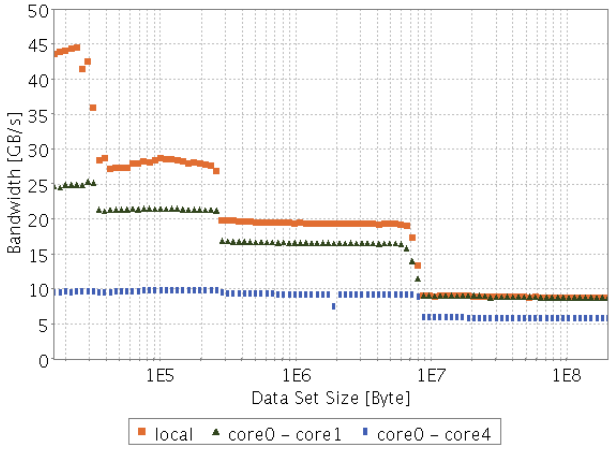
(b) Nehalem, modified



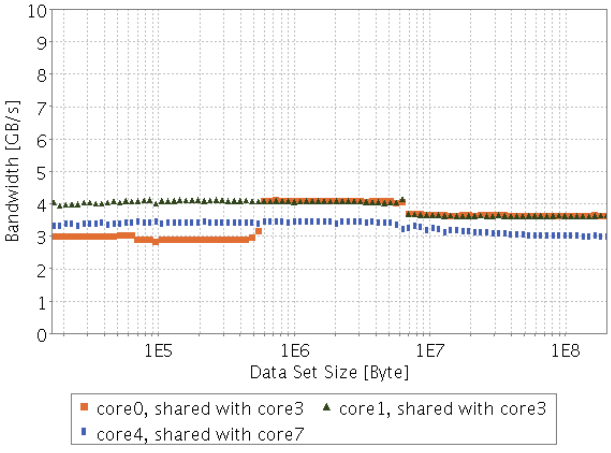
(c) Shanghai, exclusive



(d) Nehalem, exclusive



(e) Shanghai, shared (note the different y-axis scale)



(f) Nehalem, shared

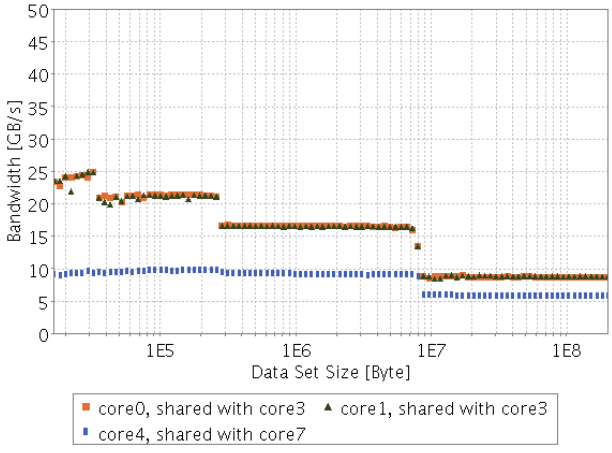


Figure 4: Write bandwidth of core0 accessing cache lines of core0 (local), core1 (on die) or core4 (other die)

Cores (Processors)	Shanghai					Nehalem				
	L3 Cache		RAM			L3 Cache		RAM		
	read	write	read	write	non-temporal	read	write	read	write	non-temporal
1 (1)	10.3	9.4	5.5	3.6	5	23.9	19.3	13.9	8.9	8.5 (8.5[4])
2 (1)	20.5	18.8	9.3	3.9	8.7	47.6	27.2	20.4	12.4	10.7 (12.3[4])
3 (1)	30.3	28	9.3	3.9	8.7	74	26.4	22.9	11.8	11.5 (13[4])
4 (1)	39.5	36.6	9.3	3.9	8.7	82	25.9	23.4	11.2	11.5 (14.1[4])
8 (2)	79.4	73.2	18	7.7	17.4	164	51.8	41.7	22.3	23.2 (28.2[4])

Table 5: L3 and main memory bandwidth in GB/s

5.3 Bandwidth to the Other Processor

The bandwidth to the second processor is mainly limited by the processor interconnect. The unidirectional transfer limit of HT in our Shanghai system is 4 GB/s and can be reached when reading modified data or exclusive cache lines from the remote L3 cache. Consistent with our latency results, shared cache lines and exclusive cache lines from the remote L1 and L2 cache are fetched from main memory and therefore slightly slower (3.6 GB/s). As for writing, we generally see slightly lower bandwidths (2.9 - 3.4 GB/s).

Nehalem's QPI bandwidth limitation is 12.8 GB/s per direction. Unmodified data can be read from the remote processor with 10.1 GB/s. The read bandwidth for modified cache lines (8.6 GB/s) is limited by the write-back to the remote main memory that is required for the state transition to shared. Writing modified cache lines is faster as no state transition and thus no write-back is required. Due to the read for ownership, writing unmodified data (9.3 - 9.9 GB/s) can not exceed the 10.1 GB/s limitation.

5.4 Multicore Bandwidth

Our single core bandwidth results are not sufficient for the evaluation of shared resources like the L3 cache and the integrated memory controller. Our multicore bandwidth benchmark allows us to stress these resources with multiple concurrent memory accesses. All threads work on different data to assure that data streams are independent from each other. Table 5 shows the results and excludes the bandwidth of the L1/L2 caches as these scale with the number of cores.

The L3 cache of Shanghai scales well up to four threads per package. Two or more threads are required to fully use the memory bandwidth of the integrated memory controller. The write bandwidth to main memory is below 4 GB/s due to the write allocate policy that requires reading before modifying the data. Non-temporal stores can be used to significantly improve this situation.

Nehalem has a higher per thread and total (82 GB/s) L3 read bandwidth, although it only scales well up to three threads. In contrast, concurrent L3 write accesses top out at 27.2 GB/s with two threads and bandwidth decreases when more threads compete for L3 write access (Shanghai performs better in this case). The main memory read bandwidth of up to 23.4 GB/s shows the significant advantage of Nehalem's three DDR3 channels. Similar to Shanghai, writing to main memory is significantly slower than reading. However, Nehalem does not benefit as much from non-temporal stores. We measure a significant improvement by disabling the hardware prefetcher, but non-temporal stores still have a relatively low bandwidth compared to the read bandwidth. Furthermore, disabling the prefetchers decreases the read bandwidth in what may outweigh the benefit of non-temporal stores.

In our single core bandwidth results, Shanghai demonstrates a higher L1 read bandwidth than Nehalem and a similar write bandwidth. Nehalem can perform one 128 Bit read and one 128 Bit write simultaneously each cycle. In contrast, Shanghai can either perform two 128 Bit loads, two 64 Bit writes, or one read and one write. This perfectly explains the L1 bandwidth results, however, it also suggests that Shanghai may not perform as well for concurrent read and write accesses, i.e. copying. Table 6 shows the results of a modified benchmark that measures the copy bandwidth. As expected, the Shanghai architecture is disadvantageous in this case. The single threaded L3 copy bandwidth is significantly higher on Nehalem while both processors perform similarly when all cores access the L3 concurrently.

Cores	Shanghai				Nehalem			
	L1	L2	L3	RAM	L1	L2	L3	RAM
1	55	16.1	9.7	3.4	79.2	34.9	24.1	10.6
8	405	127	75.3	7.9	623	278	76	27.7

Table 6: Copy bandwidth in GB/s

6. RELATED WORK

Performance measurements are common practice to analyze implementation details of the memory hierarchy. A well-known and established benchmark is STREAM [11] although it disregards many architectural details. Babka and Tůma present their work in [4], focusing mainly on translation lookaside buffers and cache associativity. Peng et al. compare the memory performance of dual-core processors including a ping-pong implementation to analyze the latency of cache-to-cache transfers [12]. However, they do not cover inter-core bandwidths. To the best of the authors knowledge, the memory benchmarks presented in this paper are the first to explicitly consider the influence of the cache coherency protocol.

With respect to our test platforms, there is a fair amount of literature that describes the AMD Opteron architecture including the MOESI protocol in much detail [2, 5]. However, details about the "non-inclusive" L3 cache of the quad-core Opteron are hardly available. As for our Intel Xeon test system, we can only refer to a very limited number of relevant publications. This is mostly due to the novelty of the Nehalem microarchitecture and the MESIF protocol. Some information can be gathered from Intel documents [8, 9].

Another important aspect is the use of a performance measurement suite that allows to develop and run benchmarks and supports the result evaluation. The Open Source tool BenchIT provides this functionality [10]. It runs microbenchmarks on POSIX compliant systems and helps to compare different algorithms, implementations of algorithms, features of the software stack and hardware details of whole systems.

[4]hardware prefetcher disabled

7. CONCLUSIONS AND FUTURE WORK

An effective use of the memory subsystem is a key factor to obtain good application performance on today's microprocessors. With the introduction of multicore technology, the cache hierarchy grows more complex: Besides the cache performance of individual cores, important design goals are aspects such as fast inter-core communication and an efficient solution to maintain the coherency of all caches. In this work we present and evaluate a set of latency and bandwidth benchmarks for multicore x86-64 architectures that expedite our understanding of the properties and performance of the memory subsystem. Our approach is characterized by the explicit control of the data's cache coherency state and the well-defined data placement in every available cache and memory location of a multicore and multiprocessor cc-NUMA system.

The potential of our approach is demonstrated with an in-depth comparison of the multi-level memory subsystem of dual-socket SMP systems based on the quad-core processors AMD Opteron 2384 (Shanghai) and Intel Xeon X5570 (Nehalem). Our results reveal significant differences in many important aspects such as local cache performance and inter-core communication capabilities. We find these differences to primarily originate from two aspects: first, the inclusiveness of the last level cache (non-inclusive/inclusive), and second, the cache coherency protocol (MOESI/MESIF).

The inclusive last level cache of Nehalem proves to be superior in most cases. The L3 cache can serve as the central and single unit for on-chip inter-core data transfers. Nehalem's *core valid bits* both reduce snoop traffic to the cores and play a key role for the performance of on-chip cache transfers.

Shanghai's non-inclusive L3 cache combines the disadvantages of an exclusive last level cache with those of an inclusive design. For example, unmodified data located in other cores' caches is fetched from main memory as the L3 cache does not hold this data (disadvantage of an exclusive design). Accesses to local caches are affected as well - the write bandwidth to unmodified L1 cache lines is severely limited due to the possibility that cache lines are duplicated in the L3 cache and require invalidation (disadvantage of an inclusive design, fixed in Nehalem with the core valid bits). We can identify few cases where Shanghai's design prevails, e.g. for multithreaded L3 write accesses. However, the negative effects are more prevalent as Nehalem's memory latency and bandwidth results often surpass those of Shanghai by a factor of four or more.

With respect to the cache coherency, AMD's MOESI protocol shows the expected performance advantage for accesses to modified cache lines in remote processors. Intel's Nehalem prevails for unmodified cache lines in remote processors due to its inclusive L3 cache. The exact effect of the MESIF state *forward* should be limited to systems with more than two sockets.

Shanghai's inferior performance is to some degree caused by its disadvantage in HT bandwidth (compared to QPI) and low two-channel DDR2 bandwidth (compared to three-channel DDR3). This situation will likely change with the introduction of new processors and sockets that will enable the use of HT 3.0 and the support for more memory channels and DDR3. However, Shanghai's cache design and coherency implementation that originates from single core SMP systems does not match the requirements of multicore processors. Its non-inclusive last level cache shows fundamental disadvantages that likely outweigh the anticipated cache size advantage. These findings should be considered in the design of future multi- and manycore microarchitectures.

In future work we plan to incorporate hardware performance counters into our benchmarks. The results should allow the correlation of performance anomalies in applications to properties of the underlying hardware. Furthermore, our benchmarks will help to analyze systems with more than two sockets (e.g. based on Nehalem-EX) as well as the ccNUMA implementation of larger shared memory systems and future many-core processors with even more complex cache architectures.

Acknowledgements

The authors would like to thank Matthias S. Müller and Robert Schöne for their valuable comments and many fruitful discussions.

References

[1] SPEC CPU2006 published results page: http://www.spec.org/cpu2006/results/.

[2] AMD. *AMD64 Architecture Programmer's Manual Volume 2: System Programming*, revision: 3.14 edition, September 2007. Publication # 24593.

[3] AMD. *Software Optimization Guide For AMD Family 10h Processors*, revision: 3.04 edition, September 2007. Publication # 40546.

[4] V. Babka and P. Tůma. Investigating cache parameters of x86 family processors. In *SPEC Benchmark Workshop*, pages 77–96, 2009.

[5] P. Conway and B. Hughes. The AMD opteron northbridge architecture. *Micro, IEEE*, 27(2):10–21, 2007.

[6] J. Dorsey, S. Searles, M. Ciraula, S. Johnson, N. Bujanos, D. Wu, M. Braganza, S. Meyers, E. Fang, and R. Kumar. An integrated quad-core opteron processor. In *IEEE International Solid-State Circuits Conference*, pages 102–103, 2007.

[7] J. L. Hennessy and D. A. Patterson. *Computer Architecture: A Quantitative Approach*. Fourth edition, 2006.

[8] Intel. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, March 2009.

[9] Intel. *An Introduction to the Intel QuickPath Interconnect*, January 2009.

[10] G. Juckeland, S. Börner, M. Kluge, S. Kölling, W. E. Nagel, S. Pflüger, and H. Röding. BenchIT - performance measurements and comparison for scientific applications. In *PARCO*, pages 501–508, 2003.

[11] J. D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture Newsletter*, pages 19–25, December 1995.

[12] L. Peng, J.-K. Peir, T. K. Prakash, C. Staelin, Y.-K. Chen, and D. Koppelman. Memory hierarchy performance measurement of commercial dual-core desktop processors. *Journal of Systems Architecture*, 54(8):816 – 828, 2008.