Main Memory and Cache Performance of Intel Sandy Bridge and AMD Bulldozer

Daniel Molka Daniel Hackenberg Robert Schöne
Center for Information Services and High Performance Computing (ZIH)
Technische Universität Dresden, 01062 Dresden, Germany
{daniel.molka, daniel.hackenberg, robert.schoene}@tu-dresden.de

Abstract

Application performance on multicore processors is seldom constrained by the speed of floating point or integer units. Much more often, limitations are caused by the memory subsystem, particularly shared resources such as last level caches or memory controllers. Measuring, predicting and modeling memory performance becomes a steeper challenge with each new processor generation due to the growing complexity and core count. We tackle the important aspect of measuring and understanding undocumented memory performance numbers in order to create valuable insight into microprocessor details. For this, we build upon a set of sophisticated benchmarks that support latency and bandwidth measurements to arbitrary locations in the memory subsystem. These benchmarks are extended to support AVX instructions for bandwidth measurements and to integrate the coherence states (O)wned and (F)orward. We then use these benchmarks to perform an in-depth analysis of current ccNUMA multiprocessor systems with Intel (Sandy Bridge-EP) and AMD (Bulldozer) processors. Using our benchmarks we present fundamental memory performance data and illustrate performance-relevant architectural properties of both designs.

1. Introduction

Multicore technology is the most important factor that drives today's microprocessor performance improvements. Providing cache coherence becomes a steeper challenge with each new processor generation and the mandatory increase in core count. Significant efforts are necessary to continue to provide the coherent caches that programmers are accustomed to. The utilized coherence mechanisms have a significant impact on memory performance. Therefore, the ever-growing complexity of the memory subsystem with several cache levels that need to be exploited efficiently is challenging to any developer striving for optimal application performance. Unfortunately, many implementation details of these memory subsystems–particularly with respect to performance characteristics–are undocumented. This complicates performance optimization, but more importantly it is harmful for many ongoing research activities in this field that predominantly rely on modeling approaches [6, 17] to evaluate and compare innovative approaches and implementations.

In this paper we present extensions to our latency and bandwidth benchmarks [8, 21] that support measurements for arbitrary locations in the memory subsystem. Using these benchmarks, we present an in-depth evaluation of the memory performance of Intel Sandy Bridge-EP and AMD Bulldozer processors. On the Sandy Bridge-EP platform, the L3 design has been changed significantly since the previous generation with the introduction of L3 slices that are connected by a ring bus. With the step from 128 Bit to 256 Bit SIMD width, the bandwidth demands to feed the cores grew considerably. The Bulldozer architecture also features a number of important architectural changes compared to the previous processor generation, most notable the module design that integrates two cores with shared resources. Furthermore, the cache coherence protocol and the probe filtering mechanism have been modified. Our analysis reveals in detail how the microarchitectural differences tremendously affect the performance of the memory subsystem.

MSPC '14, June 13, 2014, Edinburgh, Scotland.

ACM 978-1-nnnn-nnnn-n/yy/mm. . . $15.00.
http://dx.doi.org/10.1145/2618128.2618129

2. Related Work

Performance measurements are common practice to analyze implementation details of the memory hierarchy. While STREAM is a well-established memory bandwidth benchmark [19], a number of limitations are problematic for any in-depth analysis, e.g. no inherent NUMA support and the lack of latency measurements. The benchmark set lmbench [20] can be used to measure key metrics of UNIX systems but lacks support for measuring remote cache accesses considering different coherence states. Measurements that include NUMA characteristics and coherence states are available for outdated hardware [9]. Agarwal et al. state that memory latency can be hidden with keeping a number of read request in flight as long as the number of outstanding requests is sufficient [4]. Other notable work includes the extensible x86 benchmark suite likwid-bench [25], which is part of the likwid tool suite [24]. It features bandwidth benchmarks and extensive NUMA support but lacks options for in-depth cache analysis and latency measurements.

A set of sophisticated memory benchmarks that take caches and coherence effects into account has been presented in [21], along with results for an Intel Nehalem dual socket node. This work has been extended in [8] to include an in-depth comparison of Intel Nehalem and AMD Shanghai. These processor generations are now outdated, and newer CPU generations differ significantly from these systems. Moreover, AVX instructions as well as new coherence states have been introduced, effectively rendering these benchmarks unfit for state-of-the-art systems. We therefore extended the benchmarks introduced in [8, 21] in this work, and analyze the memory performance of current CPU generations.

System	Dell PowerEdge R720	SuperMicroA+ Server 1042G-LTF
Processor	2x Intel Xeon E5-2670 (Sandy Bridge-EP)	4x AMD Opteron 6274 (Bulldozer)
Cores	16 (2x 8), 32 threads with HyperThreading	64 (4x 2x 8[1]), FPU shared by 2 cores
Base (turbo) clock[2]	2.6 GHz (up to 3.3 GHz)	2.2 GHz (up to 3.1 GHz)
L1D/L2 cache	32 KiB per core / 256 KiB per core	16 KiB per core / 2 MiB per compute unit
L3 cache	20 MiB per chip	8 MiB per chip (2 MiB used as probe filter)
Interconnect	QPI 1.1, 8 GT/s	HyperTransport 3.1, 6.4 GT/s
Memory channels	4x PC3-12800R per socket	2x PC3-12800R per die (4 per socket)
Memory size	32 GiB, 8x 4 GiB (4 DIMMs per die)	64 GiB, 16x 4 GiB (2 DIMMs per die)

Table 1. Hardware configuration of test systems

3. Test Systems and Benchmarks

We examine two multi-socket x86 servers with state-of-the-art multicore processors from Intel and AMD. Both are shared memory systems with point-to-point connections between the processors. However, their microarchitectures as well as higher level processor design differ considerably. The configuration is detailed in Table 1.

3.1 Intel Xeon 2600 - Sandy Bridge

The Intel Xeon E5-2600 family [13] is based on the Sandy Bridge microarchitecture. Each core has dedicated L1 and L2 caches as well as its own FPU. The L1 data cache is a write back cache. It can handle two 128 Bit reads and one 128 Bit store per cycle. The FPU can execute two 256 Bit instructions per cycle, one addition and one multiplication. With HyperThreading enabled, two threads can run per core, and both share most resources.

Figure 1a depicts the Xeon E5 processor. The eight cores, 20 MiB L3 cache, a quad channel memory controller, and two QPI interconnects are implemented on one die. The L3 cache is inclusive of the cores' L1 and L2 caches and operates at the core frequency. It is divided into multiple slices that are accessible by all cores. Multiple rings connect the cores with the L3 slices, memory controller, and QPI links. QPI is implemented in version 1.1, operating at 8.0 GT/s. The Intel system has two sockets that are connected via the two QPI links as depicted in Figure 2a. Intel extends the *MESI* protocol with the *Forward* state [10], which enables forwarding of shared clean cache lines. A home-snoop-protocol is used [14]. However, coherence traffic is not a significant bottleneck on the two socket system.

3.2 AMD Opteron 6200 - Bulldozer

The AMD Opteron 6200 [1] series is based on the Bulldozer microarchitecture. It is based on dual-core compute units that share the instruction fetch and decode units, floating point unit, L1 instruction cache, and the L2 cache. Each core has its own out-of-order engine, integer execution units, and write-through L1 data cache. Each L1D provides two 128 Bit read ports and one 128 Bit write port. The FPU supports fused multiply-add instructions and executes up to two 128 Bit instructions per cycle. 256 Bit AVX instructions are split into two 128 Bit parts. Integer SIMD instructions are also executed in the shared FPU.

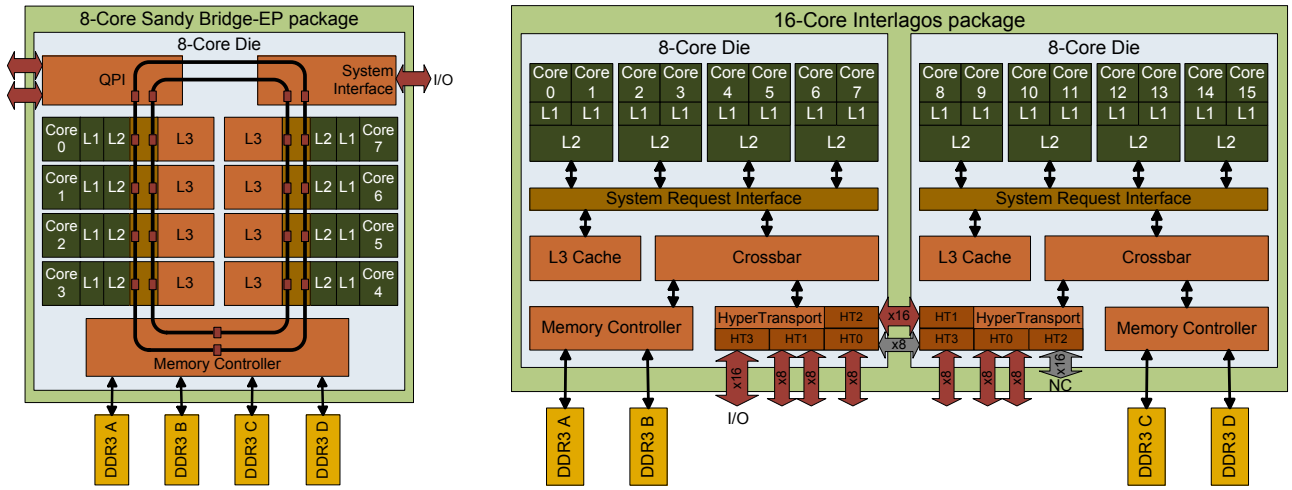
Figure 1b shows the organization of the 16-core AMD Opteron processor. It consists of two eight-core dies that are internally connected via HyperTransport links. Each die comprises four compute units, 8 MiB[3] L3 cache, a dual channel memory controller, and four HyperTransport links. The shared L3 cache is directly connected to the system request interface (SRI) and operates at the northbridge frequency of 2 GHz. It is neither inclusive nor strictly exclusive of the L2 caches. The HyperTransport links operate at 6.4 GT/s[4] (5.2 GT/s for I/O). They can be used as 16-bit (ganged) or 8-bit (unganged) links. Each socket supports four 16-Bit HT links that are connected to the different dies as depicted in Figure 1b.

[1] 4 sockets, 2 dies per socket, 8 cores per die

[2] all measurements performed at base clock

[3] 2 MiB used as directory cache by the *HT Assist* [5]

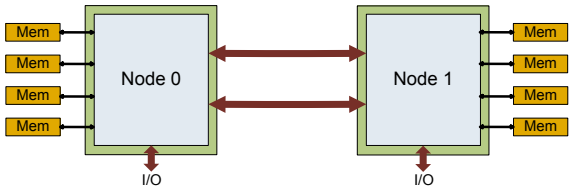
[4] according to MSR readings and definitions in [2, Chapter 3.3]



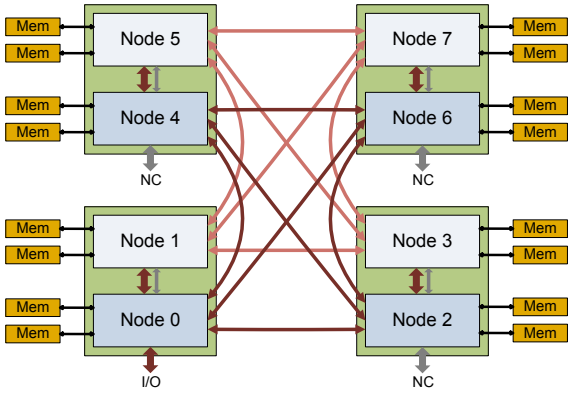
(a) 8-core Sandy Bridge-EP

(b) 16-core Bulldozer multi-chip-module

Figure 1. Composition of multicore processors: The 8-core Xeon processor (left) is implemented on a single chip. In contrast, the 16-core Opteron (right) consists of two 8-core chips in one package. Therefore, a single Opteron processor already has two NUMA nodes



(a) 2P Intel Sandy Bridge (Xeon E5-2600 family)



(b) 4P AMD Bulldozer (Opteron 6200 series)

Figure 2. Comparison of NUMA topologies: Since each Xeon processor has an integrated memory controller, the two socket Intel system also has two NUMA nodes. As each Opteron processor already has two NUMA nodes (see Figure 1b), the four socket AMD system consists of eight NUMA nodes. While the sockets are fully connected, the chips are not.

As each AMD processor already is a dual-node MCM, the four socket system contains eight NUMA nodes. Each die is directly connected to the second die in the socket via a 16-Bit link[5] and to three dies in other sockets via 8-Bit links. There are two subsets of four fully connected nodes: {0,2,4,6} and {1,3,5,7}. Data transfers between those subsets require additional HT hops, unless the nodes are in the same socket. Figure 2b illustrates this complex design.

AMD extends the *MESI* protocol by implementing the *Owned* state [3, Chapter 7.3.], which allows sharing of dirty cache lines without writing them back to memory. The Bulldozer microarchitecture apparently also implements the *ModifiedUnWritten (MuW)* state [16], which enables a faster transitions to *Modified* if multiple processors perform read-modify-write operations. The resulting protocol is depicted in Figure 3. The *MuW* state is reached when a *Modified* cache line is read by another core. Instead of changing its state to *Owned* and insert a *Shared* copy in the requesting core–as would be the case in the *MOESI* protocol–the original copy is invalidated and the requester gets the line in the *MuW* state which allows modification of the cache line without further action. If this dirty *MuW* copy is read again by another processor, it is forwarded, marked *Shared*, and inserted into the second requesters cache in the *Owned* state.

AMD implements a home-snoop-protocol, in which the node that contains the requested physical address in its memory is responsible for maintaining cache coherence. This so-called *home node* forwards requests to all nodes that could have a copy of the requested memory address. As broadcast messages are extremely expensive in a system with eight nodes, current multinode AMD processors feature a probe filter to reduce the coherence traffic. This so-called *HT Assist* uses a portion of the L3 cache which reduces

[5] the 8-Bit link connecting the dies is disabled [2, Chapter 2.12.1.5]

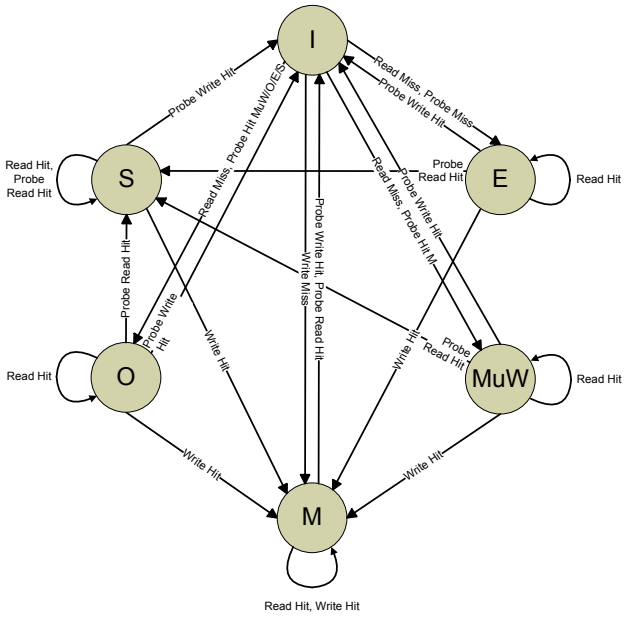


Figure 3. Extended MOESI protocol with *MuW* state [16][6][7]: The protocol is designed to always migrate the ownership of a cache line to the last requestor. Memory write requests always transition to *Modified*. Furthermore, memory read requests that involve probing other nodes always insert an *MuW* or *Owned* copy in the requestor and leave a *Invalid* or *Shared* copy in the previous owner.

the usable L3 size [2, Chapter 2.9.4.1]. Lepak et al. [16] describe an optimized version of the probe filtering mechanism, that is closely related to the *MuW* state. The exact implementation of the coherence protocol and filtering mechanism is not disclosed in the processor manuals. However, the modifications we had to implement to make our benchmarks work on the Bulldozer architecture reveal that the *MOESI* protocol has in fact been altered. The modifications were done based on the protocol described in [16] and lead to the expected results. We therefore assume that this protocol is used.

3.3 Synthetic Microbenchmarks

We use a set of microbenchmarks [8, 21] to perform a low-level analysis of each system's memory performance. They are integrated into the BenchIT [15] framework and are available as open source. Highly optimized assembler routines and time stamp counter based timers enable precise performance measurements of data accesses in shared memory systems with 64 Bit x86 processors. The benchmarks are parallelized using pthreads and individual threads are pinned to single cores using *sched_setaffinity()*. Coordinated data access sequences are performed in consideration of the coherence protocol to transfer data into a selected cache with a certain coherence state. Latencies and bandwidths of accesses to any core's caches and any socket's memory can be measured as well as the aggregated bandwidth of shared caches and memory controllers.

[6] The patent does not reveal the coherence state after read requests that do not require a probe of other nodes (Probe Hit S). Thus, there could also be a transition from *Invalid* to *Shared* if the probe filter indicates that there are only *Shared* copies.

[7] Since the probe filter does not distinguish *Modified* from *Exclusive* [5], it is not readily possible to differentiate between *Probe Hit M* and *Probe Hit E*. Thus, *MuW* and *Owned* are potentially both mapped to *Owned* in the cache and differ only in the directory state (*EM* for *MuW* and *O* for *Owned*).

The already available benchmarks support x86_64 processors up to Intel Westmere and AMD Istanbul/Magny-Cours. Bandwidth measurements are limited to 128 Bit SIMD instructions. Furthermore, the coherence state control mechanism does only support the states *Exclusive*, *Modified*, and *Shared*. In order to thoroughly analyze the more recent processors and more complex system topologies examined in this paper, the following features have been added:

- support for the coherence states *Owned* and *Forward*
- extended affinity control to allow independent CPU and memory affinity in order to better analyze NUMA characteristics
- support for 256 Bit instructions in bandwidth measurements

In order to measure the performance of accesses to a certain cache or memory from a specific NUMA node, data needs to be placed in that location prior to the measurement. Therefore, threads that perform the data placement are started on every core that should be included in the measurement (typically one core from every NUMA node). Additional threads are started on cores that are not included in the measurement in order to generate shared cache lines without involving the threads that participate in the measurement. The cache level is determined by the data set size. The NUMA node of the data is defined by the selected memory affinity of the threads. The coherence states are generated as follows:

- *Modified* in caches of core N:
  1) core N writing the data (invalidates all other copies)
- *Exclusive* in caches of core N:
  1) core N writing the data to invalidate copies in other caches,
  2) core N invalidating its cache using the *clflush* instruction,
  3) core N reading the data
- *Shared* in caches of core N:
  1) core N caching data in *Exclusive* state,
  2) another core reading the data
- *Forward* in caches of core N:
  1) another core caching data in *Exclusive* state,
  2) core N reading the data
- *Owned* in caches of core N:
  1) core N caching data in *Modified* state,
  2) another core reading the data,
  3) core N reading the data again

Forward state is only supported on Intel processors. The only difference is the order of events, resulting in the states (core N/other core): (S/F) for *Shared* and (F/S) for *Forward*. *Owned* is only available on AMD processors. The sequence that generates the *Owned* state is compatible with the original *MOESI* protocol as well as the extended version including the *MuW* state [16]. In the *MOESI* protocol the 3 steps result in the states (core N/other core):
1: (*/*) → (M/I), 2: (M/I) → (O/S), 3: no change.
In the extended protocol step 2) and 3) have different results:
1: (*/*) → (M/I), 2: (M/I) → (I/MuW), 3: (I/MuW) → (O/S).
However, the final states are identical.

The result of the *Shared* state on AMD processors is influenced by the protocol version. The first step generates the states (E/I) which the second step changes into (S/S) for the original *MOESI* protocol and (S/O) in the extended *MOESI* protocol. Thus, the coherence state on the target core is identical in both protocols. However, the always migrate approach of the extended protocol results in *Owned* copies in the caches of the helper thread[8].

[8] Benchmarks provide an option to invalidate caches of the helper thread.

For the measurement of the aggregated bandwidth [22], one thread is started on every selected core. The added option to define the memory affinity of every thread independently from its CPU affinity can for example be used to allocate all memory from a specific NUMA node in order to measure the interconnect bandwidth. Finally, new measurement routines for 256 Bit AVX instructions have been added. They are available for core-to-core transfers as well as for the aggregated bandwidth benchmark. The bandwidth is measured for sequential accesses. Thus, the 128 Bit *movdqa* instructions could simply be replaced by half as many 256 Bit *vmovdqa* instructions.

4. Latency Results

In this section we analyze latencies that are associated with accesses to local and remote caches and main memory in ccNUMA systems. We use the elaborate data placement and coherence state control mechanisms described in [21, Sec. IV] to perform a detailed comparison of the different cache coherence protocols. All measurements are performed by core 0 in node 0.

4.1 Dual socket Intel Sandy Bridge

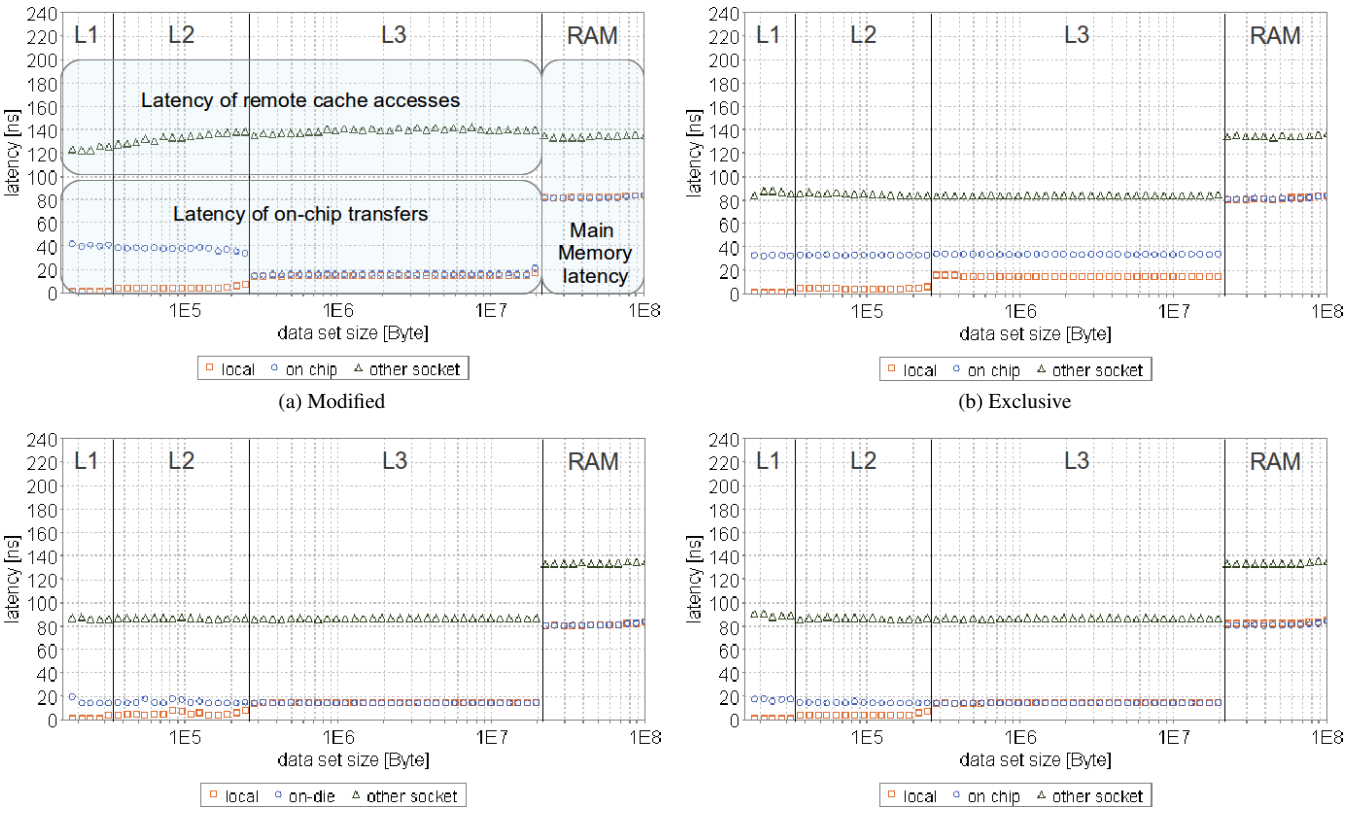
Figure 4 shows our latency measurements for cache lines in various locations with different coherence states[9] on our dual socket Intel system. All data passes through the L1 cache and gets evicted eventually. Thus, depending on the data set size, the cache lines reside in a certain cache level or–after eviction from the last level cache–in main memory. The *"local"* latencies illustrate accesses from core 0 to data that has previously been allocated and accessed by core 0. Measurements that are labeled *"on chip"* refer to data that has been allocated, accessed, and evicted by other cores on the same socket that share the L3 cache and the memory controller with core 0. However, shared caches can track from which core a cache line originates and trigger coherence state transitions upon accesses by a different core. Therefore, transfers from the same location can have different properties. The *"other socket"* latencies illustrate accesses to data that has been allocated and accessed by a core on the other socket. Table 2 summarizes the results.

On-chip transfers are handled by the inclusive L3 cache without sending a request to the home node. *Modified* cache lines from another core's L1 or L2 cache are forwarded by the respective core. If *Modified* data is evicted it is written back to the L3 that delivers *Modified* cache lines that are not present in any core's caches. The L3 latency is 15 ns. *Exclusive* cache lines are always delivered by the L3 cache. However, if a line has been placed in the L3 cache by another core the latency is higher. Since *Exclusive* lines are silently evicted, it is not known if the line is still present in the core's L1 or L2. Thus, the core that originally read the data needs to be probed. In both shared cases–*Shared* and *Forward*–two cores on the die accessed the data. They might be silently evicted as well, but the L3 data is guaranteed to be valid as the cores can not modify shared cache lines unnoticed. Thus, shared cache lines are delivered by the L3 cache without probe.

[9] hardware prefetcher disabled for *Shared* and *Forward*

coherence state	latency in ns							
	local		on-chip			other socket		
	L1	L2	L1	L2	L3	L1	L2	L3
modified	1.5	4.6	40.4	38.1	15	123 - 140		
exclusive			33.8			87.3		
forward			15					
shared								

Table 2. Latencies for accesses to various memory locations on the dual socket Intel Sandy Bridge system



(a) Modified

(b) Exclusive

(c) Forward

(d) Shared

Figure 4. Latency of memory accesses with different coherence states on a two socket Intel Sandy Bridge system

Clean data is delivered by the remote L3 cache with a latency of 87.3 ns. In Figure 4c and 4d, the L3 cache will be in state *Forward*[10] as one of the cores on the die performed the last access. Accessing the cache lines therefore triggers a transition from *Forward* to *Shared* in the second socket that includes a notification to the core that has the *Forward* copy. Thus, the latency is the same as for *Exclusive* cache lines that require a probe as well. Dirty cache lines are written back to memory when transferred to the other socket. The latency of 123 - 140 ns depends on the amount of accesses[11]. The memory latency is 81 ns for local and 133 ns for remote accesses.

4.2 Quad socket AMD Bulldozer

Figure 5 shows latency measurements for cache lines in various locations[12] with different coherence states on our AMD system. The meaning of *"local"* and *"on-chip"* latencies introduced in 4.1 also applies to Figure 5. *"2nd core"* denotes the sibling core within the same module. Results for *"node n"* show the characteristics of accesses to data allocated and accessed by cores in other NUMA nodes. Node 1 is the second die within the package of node 0. The other nodes are in other sockets with node 2/4/6 being one hop and node 3/5/7 being two hops away. The creation of the coherence states *Owned* and *Shared* involves additional accesses performed by another (third) core, meaning that multiple copies of a cache line can exist simultaneously. Therefore, probes of other nodes can be required even if a nearby copy of a cache line is available.

[10] a third socket would be required to create a *Shared* copy in a remote L3

[11] default 1024 accesses, fewer for small data set sizes

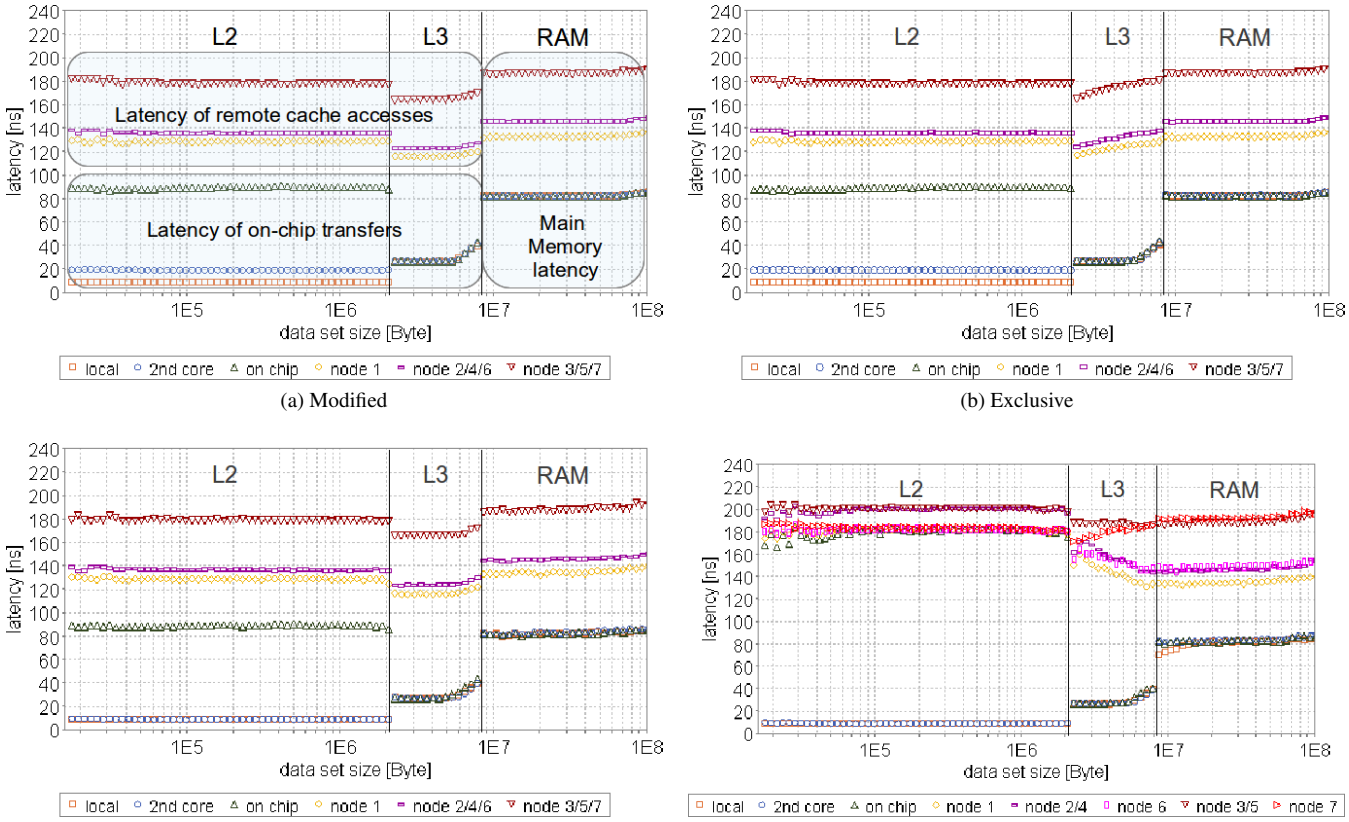
[12] L1 cache latencies are not available due to the size of only 16 KB

Latency of on-chip transfers

Coherence between the two cores in a compute unit is ensured by the shared L2 cache. The latency of accesses to *Modified* and *Exclusive* cache lines varies depending on which core has placed the data in the L2 cache. For cache lines that have been evicted by the core itself the latency is 9.1 ns (20 cycles). Accessing a cache line that has been evicted from the other core requires 19.5 ns, which indicates that the second core's L1 is probed. Thus, the L2 cache is apparently not notified when cache lines are evicted from the L1 cache. Consequently, a probe is necessary if the 2nd core could still have a copy of the line that requires a state transition to revoke the write permission (E→S, M→I). If the cache line is in *Shared* or *Owned* state, it can be read directly as no state transition would be required even if the other core still had a copy.

If no valid copy is found within the compute unit, a request is sent to the system request interface. Cache lines in the L3 are forwarded from the L3 to the requesting core within 27.3 ns. In case of an L3 miss, coherence is maintained by the memory controllers as the non-inclusive L3 does not contain complete information which cache lines are present in other compute units. *Modified*, *Exclusive*, and *Owned* cache lines are forwarded by the compute unit that holds the line upon receiving the probe request from the memory controller (data response). Although a copy is available on the chip, the latency of 89 ns for such a transfer is slightly higher than the local memory latency[13].

[13] In the depicted scenarios data in a cache of core N is always allocated from the local memory of core N. The latency would be even higher if the data would be allocated from another node as the request would be forwarded to and send back by the home node.



(a) Modified

(b) Exclusive

(c) Owned, shared copy in node 7

(d) Shared, shared copy in node 7

Figure 5. Latency of memory accesses with different coherence states on a four socket AMD Bulldozer system.

In the example shown in Figure 5d the latency of case *"on chip"* is approximately 180 ns which is much higher than for cache lines in other coherence states. The reason for this behavior is that *Shared* cache lines are not forwarded to the requesting core. Instead, the latency depends on the distance to the cache that holds an *Owned* copy or the distance to the main memory if no *Owned* copy exists. The sequence we use to generate the *Shared* state (see Section 3.3) would result in two *Shared* copies according to the original *MOESI* protocol. The *HT Assist* would indicate that the cache line is shared between multiple nodes, filter the probe request [5], and the valid data from local memory would be returned. This is obviously not the case. Instead, the data is forwarded from another cache.

The observed behavior can be explained by the protocol modifications described in [16] (see Figure 3). At first core 2 generates an *Exclusive* copy in its cache. The following read request by core 63 results in a transition to *Shared* in core 2. However, an *Owned* copy is inserted in core 63 (node 7) according to the always migrate approach in the revised coherence protocol. Consequently, node 7–that the probe filter entry identifies as owner of the cache line–is probed instead of forwarding the valid copy on the chip or using the valid data from local memory This requires four additional transfers via HyperTransport, two to forward the request to node 7 and two to send back the data response.

Latency of remote cache accesses

We now discuss the node 1 (same socket, different die), node 2/4/6 (other socket, one hop away), and node 3/5/7 (other socket, two hops away) results depicted in Figure 5. Similar to on-chip transfers, cache lines in *Modified*, *Exclusive*, or *Owned* state are forwarded to the requesting core. However, the latency increases because of the following HyperTransport transfers:

1: send request to home node

2: forward request to node that owns the line

3: send response to requesting node

In the measurements shown in Figures 5a, 5b, and 5c *HT Assist* indicates a direct probe. As the data is cached in the home node, step two is not required. Thus, nodes that are one hop away require two transfers: one to send the request (8 Byte) and one to send back the response (4 Byte header + 64 Byte data) [11]. Whether transfers occur between dies in the same socket (node 1) or between sockets (node 2/4/6) only slightly influences the measured latencies of 129.1 ns or 136.3 ns, respectively. Nodes that are two hops away require four transfers what increases the latency to 178 ns.

The different latencies can be explained based on the system design and a few fundamental latency numbers listed in Table 3. Transmission times have been calculated based on 8 Byte for the request, 68 Byte for the response and the data rates of 12.8 GB/s or 6.4 GB/s for the different link width at the HyperTransport link speed of 6.4 GT/s, e.g. $68\,Byte/12.8\,Byte/ns = 5.31\,ns$. Based on these calculated transmission times and the measured roundtrip times we derive the latency column, e.g. $(40.5\,ns - 0.62\,ns - 5.31\,ns)/2 = 17.28\,ns$. We then carry over the *total* column (one-way latency numbers) of Table 3 to Table 4 and combine these numbers with our knowledge of the NUMA topology (see Figure 2b) to calculate the resulting HyperTransport transfer times for different distributions of the involved nodes within the two fully

transfer		latency	transmission	total	roundtrip
near	request	17.28 ns	0.62 ns	17.9 ns	40.5 ns
	response		5.31 ns	22.6 ns	
far	request	18.56 ns	1.25 ns	19.8 ns	49 ns
	response		10.62 ns	29.2 ns	

Table 3. HT one-way latency and transmission times and two-way (roundtrip) total latencies for transfers between dies in one socket (near) and between dies in different sockets (far)

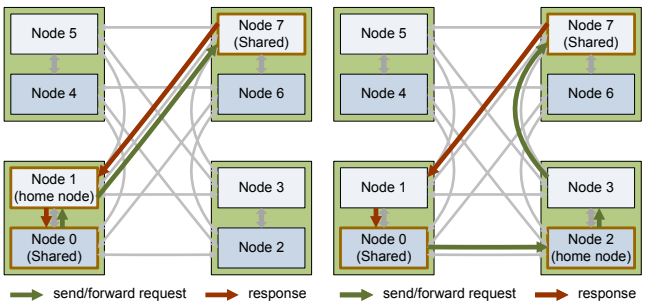
involved sockets	involved subsets	transfers [ns] 17.9	22.6	19.8	29.2	t_{trans}
1	1	0	0	0	0	0 ns
	2	1	1	0	0	40.5 ns
2	1	0	0	1	1	49 ns
	2	1	1	1	1	89.5 ns
3	1	0	0	2	1	68.8 ns
	2	2	0	2	1	104.6 ns
		1	1			109.3 ns

Table 4. HT transfer times for remote cache accesses

connected subsets. These numbers explain for example the increase of total access latency to another L2 cache from 88.6 ns for the on-chip case to 178 ns for nodes that are two hops away (see Figure 5a, node 3/5/7): the difference is 89.5 ns, resulting from one near and one far hop in each direction (see Table 3).

If the cache lines are in state *Shared* (see Figure 5d), the situation is more complex as the node that holds the second copy (*Owned* state) has to be probed as well. The latency in this case depends on the distribution of the three participating cores among the NUMA nodes. If node 0 ("on chip"), node 1, node 6, or node 7 is the home node, two sockets are involved, resulting in four HyperTransport transfers to forward the request and receive the response (see Figure 6a). If the home node is in a third socket ("node 2/4" and "node 3/5"), five transfers are necessary (see Figure 6b). The latency therefore increases to 178 ns (88.6 + 89.5) or 198 ns (88.6 + 109.3) according to Table 4 and consistent to Figure 5d. Lower latencies are possible if the *Owned* copy is closer to the requestor.

The difference between local and remote L3 accesses is higher than the HyperTransport latency, as the crossbar and the probe filter check also add latency. For *Modified* and *Owned* cache lines, the data placement works as expected. However, unmodified (*Shared* or *Exclusive*) data is evicted earlier than intended from the remote L3 caches. The latency therefore approaches the respective memory latency for increasing data set sizes. We argue that this results from a hardware detection of an inefficient L3 usage of the data placement mechanism.



(a) 3 nodes spanning 2 subsets in 2 sockets (89.5 ns case in Table 4)

(b) 3 nodes spanning 2 subsets in 3 sockets (109.3 ns case in Table 4)

Figure 6. Three-node HyperTransport transfer scenarios

location	latency in ns L2	L3	RAM
local	9.1	27.3	82.3
2nd core	19.5		
on-chip	88.6		
2nd die in MCM	129	116	133
other socket, 1 hop	136	123	146
other socket, 2 hops	178	164	187
including probe (max)	198	185	-

Table 5. Latencies for accesses to various memory locations on the quad socket AMD Bulldozer system

Main memory latency

The *HT Assist* strongly influences the memory latencies. Despite the increased HyperTransport latencies of the four socket configuration, requests to local memory are on the same level as on the two socket system with 82.3 ns (see Table 5). If a probe request had to be broadcasted, the latency of local memory accesses would be close to the 164 ns that are required for accessing the farthest L3. Instead, it can be read directly from memory as the *HT Assist* filters the request.

The difference between local and remote memory latency is higher than what can be explained according to Table 4. The gap between intra- and inter-socket transfers increases as well. We attribute this to subtle differences regarding the amount of probe responses. In case of cache hits the DRAM request is canceled [5] and a single data response is send back to the requesting core. If the data is sent from memory, there seems to be an additional probe response to indicate that data is not cached. The transfer latency of the second package increases the latency of remote accesses.

5. Bandwidth Results

In this Section we analyze the available bandwidths of local and remote caches as well as main memory on our ccNUMA test systems.

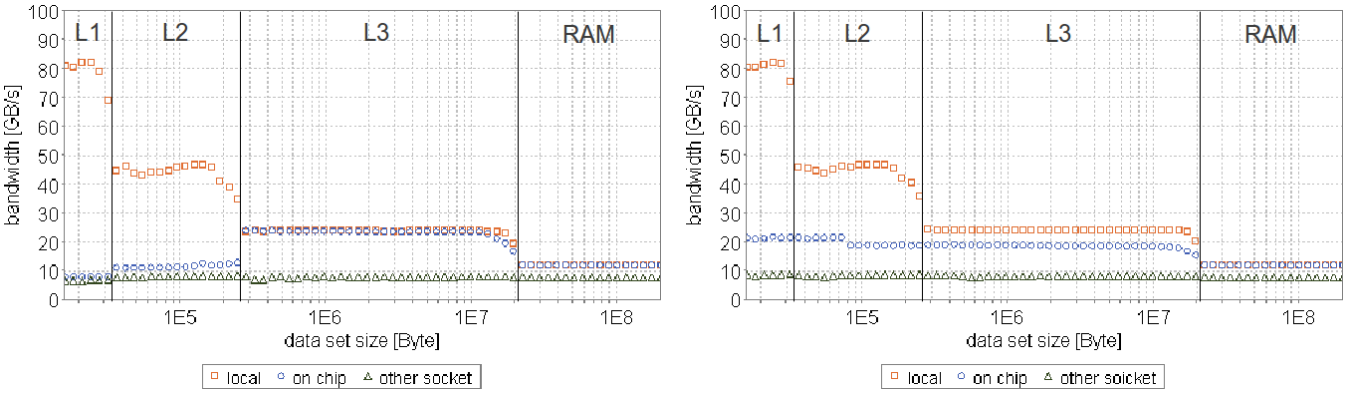
5.1 Dual socket Intel Sandy Bridge

Figure 7 shows the single threaded bandwidths that are available on the Intel system. We measure a L1 bandwidth of 82.3 GB/s, close to the theoretical peak performance for two 128 Bit loads per cycle at 2.6 GHz. The L2 cache sends data with 46.8 GB/s. The shared L3 cache provides a bandwidth of 24.4 GB/s for requests that it can service without probing other cores or 18.7 GB/s if probes are required (*Exclusive* case). *Modified* cache lines from other cores' L1 and L2 caches can be read with 8.1 and 11.2 GB/s while *Exclusive* cache lines are delivered faster by the L3 cache. The main memory read bandwidth is 11.7 for local and 7.8 GB/s for remote accesses.

Table 6 compares the memory bandwidth for local and remote accesses via QPI. Local as well as remote bandwidths are influenced by the power management on the second socket. If the other socket is idle, the local memory bandwidth is 39.7 GB/s while it is 42.2 GB/s if at least one core is active in the other socket. The im-

mem-bind	hops	node1 idle 1 thrd	node1 idle 8 thrd	node1 active 1 thrd	node1 active 8 thrd
node0	0	11.7	39,7	12.9	42.2
node1	1	7.7	18.7	8.6	23.6

Table 6. Read bandwidths in GB/s for one or eight cores running on node0 reading memory from different nodes on the two socket Intel Sandy Bridge system



(a) Modified (b) Exclusive

Figure 7. Single threaded read bandwidth on a two socket Intel Sandy Bridge system

pact on the bandwidth of remote memory accesses is even greater, as it drops from 23.6 to 18.7 GB/s (-20.7%) if the second socket is idle. The two QPI links are not wide enough to fully utilize the remote bandwidth. They provide a raw bandwidth of 32 GB/s in each direction of which only 23.6 GB/s (73.75%) are reached. This is caused by the 8 Byte packet headers [12] as well as additional 8 Byte probe request that the remote node–which is the home node of the data–will send back to check the local L3. Thus, 80 Bytes are transferred from the remote node to the requesting node for each 64 Byte cache line. This reduces the achievable bandwidth to 25.6 GB/s, which is almost fully utilized.

Table 7 lists the aggregated memory bandwidth that multiple concurrently reading cores can achieve. HyperThreading slightly increases the per-core bandwidth. Moreover, the aggregated L3 bandwidth scales linearly with the core count up to a total of 207 GB/s for read and 140 GB/s for write access. The main memory read bandwidth of 42.2 GB/s per socket has also improved significantly compared to prior microarchitectures from Intel. However, the theoretical maximum of the four DDR3-1600 channels of 51,2 GB/s is not reached.

5.2 Quad socket AMD Bulldozer

Figure 8 shows the read bandwidth that a single thread can achieve for data transfers from various locations within the memory subsystem. The shared L2 cache provides 22.5 GB/s for *Exclusive* cache lines that a core evicted itself, or 19.6 GB/s for data evicted by the second core in the MCM which–similar to the different latencies–can be attributed to the additional probes of the other core. The on-chip L3 cache delivers data with 9.8 GB/s, which is only slightly faster than the 7.8 GB/s from local memory. Transfers from other compute units only achieve a bandwidth of 6.9 GB/s.

Table 8 shows the read bandwidths that are available via the HyperTransport links for a single thread as well as for eight concurrent threads on one die. The 16 Bit link that connects the dies does only transfer data with 5.1 GB/s and 5.9 GB/s, respectively. This is much less than the 12.04 GB/s[14] that the link should provide and not nearly enough to fully utilize the remote memory bandwidth of 15.8 GB/s. For accesses to other sockets, the available bandwidth is even lower. Memory from node 2 can be read with up to 4.5 GB/s while node 4 and node 6 only provide 3.2 GB/s. The distance is one hop in both cases, and the link width is identical as well. However, node 2 is connected to a HyperTransport link with one deactivated sublink[15] while node 4 and node 6 are connected to a link that has both sublinks activated [18]. Therefore, the connection to node 2 supports more outstanding requests, as the request queue is not partitioned. As a result, even the two-hop connection to node 3 (via node 2) provides more bandwidth (4 GB/s) than the one-hop connections to node 4 or 6. The two-hop connections to node 5 or 7 provide only half of that bandwidth (2 GB/s) because of the fewer number of outstanding requests. This shows that the performance is severely limited by outstanding HyperTransport requests.

threads	cores	bandwidth in GB/s			
		L3		memory	
		read	write	read	write
1	1	24.4	17.8	11.7	9.1
2		26.5	18.3	13.4	9.3
4	2	52.8	36.4	25.7	17.3
6	3	78-7	54.2	34.3	19.0
8	4	104.5	72.0	38.1	19.2
12	6	156.1	107.0	38.8	18.8
16	8	207.1	140.5	39.7	18.4

Table 7. Bandwidth scaling on the 2P Intel Sandy Bridge system: The L3 cache bandwidth scales almost linear with the number of used cores. The main memory bandwidth can almost be fully utilized using only half of the cores.

[14] 12.8 GB/s raw bandwidth per direction, 4 Byte read response per 64 Byte

[15] The 8 Bit link between the dies is disabled (see Figure 1b)

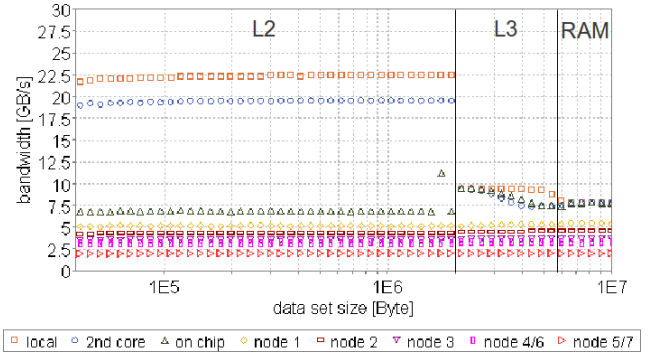


Figure 8. Single threaded read bandwidth for cached data in state *Exclusive* on the AMD Bulldozer system

memory allocated at	hops	minimal link width	bandwidth	
			1 thread	8 threads
node 0	0	-	7.5	15.8
node 1	1	16 Bit	5.1	5.9
node 2		8 Bit	4.3	4.5
node 4/6			3.2	3.2
node 3	2		3.6	4.0
node 5/7			2.0	2.0

Table 8. HyperTransport bandwidths in GB/s, one or eight threads running on node 0, reading from main memory of different nodes

threads	compute units	bandwidth in GB/s			
		L3		memory	
		read	write	read	write
1	1	9.8	5.9	7.5	4.5
2		10.6	6.8	9.0	4.8
4	2	18.3	13.7	15.4	6.2
6	3	25.0	20.5	16.5	
8	4	31.3	27.2	15.8	

Table 9. Bandwidth scaling on the AMD Opteron processor: The L3 read bandwidth for one core is only slightly higher than the main memory bandwidth. However, the L3 bandwidth scales better with increasing core counts and provides higher write bandwidths.

Figure 9 compares the local read and write bandwidths of a single compute unit. With 33.0, 11.1, 5.9, and 4.5 GB/s the single threaded write bandwidths are significantly lower than the respective read bandwidths of 59.2, 22.5, 9.8, and 7.8 GB/s. The Level 1 write bandwidth should be limited by the L2 bandwidth because of the write through policy. However, for continuous writes to a memory region smaller than 4 KiB the write bandwidth exceeds the usual L2 performance. This is caused by the so-called *Write Coalescing Cache (WCC)* [1, Chapter 2.13] that alleviates the disadvantages of a write-through cache design. Interestingly, the L1 read bandwidth does not scale with the number of cores. This is caused by the limited number of FPU pipelines that can handle the SIMD loads. In contrast, the L2 read bandwidth increases significantly if both cores are used simultaneously, as more outstanding loads are supported by the two load store units.

Table 9 shows the scaling of the last level cache and main memory bandwidth with the number of concurrent threads on one chip. Utilizing both cores of a compute unit slightly increases the bandwidths. The L3 bandwidth scales fairly well with the number of used compute units However, 31.3 GB/s per 8-core die is very low in comparison to the Intel system. Two compute units can almost fully utilize the main memory performance. Bandwidths scale linearly with the number of used dies to 62.6 GB/s of L3 and 31.6 GB/s of main memory read bandwidth per socket. However, software needs to be NUMA-aware in order to fully utilize the resources of a single processor.

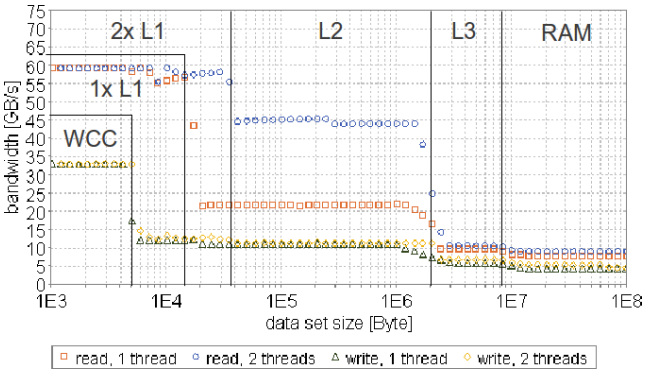


Figure 9. Read and write bandwidth on the AMD Bulldozer system. An additional buffer (the *Write Coalescing Cache*) between the two L1 caches and the L2 cache reduces the bandwidth limitation of the write-through policy

6. Conclusions

A good understanding of the hierarchical memory system of today's processors continues to be a key factor to obtain good application performance. However, exact performance data is mostly unspecified for both systems, especially regarding core-to-core transfers and inter-socket communication. This work identifies the fundamental performance properties of the latest x86_64 architectures by Intel and AMD in terms of both latency and bandwidth. Our benchmarks reveal a particularly unique set of performance numbers by considering the coherence state of cached data. We thereby document indispensable data for other scientists that strive to increase the performance of their applications or to model the behavior and performance of current CPUs [7, 17]. Moreover, our bandwidth scalability numbers are highly relevant for dynamic voltage and frequency scaling (DVFS) and dynamic concurrency throttling (DCT) based approaches that have become popular for energy efficiency optimizations [6, 22, 23].

We find the AMD Bulldozer architecture with its module concept and two independent dies per socket to be much more complex than Intel's Sandy Bridge design, creating a vast amount of different latency and bandwidth numbers. While latency figures are mostly in line with our expectations, several observed bandwidths are surprisingly low. The accumulated L3 cache bandwidth of a full Bulldozer die (8 cores) is close to the L3 bandwidth of a single Sandy Bridge core. The L3 cache bandwidth also scales better with the core count on the Intel system. Although AMD's L2 cache is very large, its performance is only on par with Intel's L3 cache in a per-core comparison. The accumulated L3 bandwidth of a Bulldozer socket exceeds the main memory bandwidth only by a factor of two, compared to more than a factor of five on the Intel system. This is even more noteworthy knowing that the Sandy Bridge system is also superior in terms of main memory bandwidth per socket. While both interconnect technologies fail to fully utilize the memory bandwidth of other NUMA nodes, the HyperTransport results are much more disappointing. The transfer rate between the sockets in the Intel system is four times higher than the transfer rate between the two dies within the AMD processor and more than ten times more effective than some of the two-hop connections in the AMD topology. Finally, on-die latencies are much better on Sandy Bridge, mostly due to the inclusive L3 cache design.

Overall, we attribute a significant portion of Intel's current advantages regarding application-level per-socket performance to the differences in the memory hierarchy. The L3 cache provides a high bandwidth per core that also scales linearly with the amount of cores. The QuickPath interconnect also provides a relatively high bandwidth for remote memory accesses. In contrast, AMD's memory subsystem severely limits the achievable processing power of the compute units in memory-intensive applications. Furthermore, parallel programs need to be exceedingly NUMA-conform to avoid being limited by the unexpectedly low HyperTransport performance for certain connections.

Acknowledgment This work has been funded by the Bundesministerium für Bildung und Forschung via the research project CoolSilicon (BMBF 16N10186).

References

[1] Advanced Micro Devices. *Software Optimization Guide for AMD Family 15h Processors, Rev 3.06*, January 2012.

[2] Advanced Micro Devices. *BIOS and Kernel Developer's Guide (BKDG) for AMD Family 15h Models 00h-0Fh Processors, Rev 3.12*, October 11, 2012.

[3] Advanced Micro Devices. *AMD64 Architecture Programmer's Manual Volume 2: System Programming, Rev. 3.22*, September, 2012.

[4] Virat Agarwal, Fabrizio Petrini, Davide Pasetto, and David A. Bader. Scalable graph exploration on multicore processors. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '10, pages 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[5] Pat Conway, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. Cache hierarchy and memory subsystem of the AMD Opteron processor. *IEEE Micro*, 30:16–29, March 2010.

[6] Matthew Curtis-Maury, Ankur Shah, Filip Blagojevic, Dimitrios S. Nikolopoulos, Bronis R. de Supinski, and Martin Schulz. Prediction models for multi-dimensional power-performance optimization on many cores. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques*, PACT '08, pages 250–259. ACM, 2008.

[7] Jianbin Fang, Henk Sips, LiLun Zhang, Chuanfu Xu, Yonggang Che, and Ana Lucia Varbanescu. Test-driving intel xeon phi. In *Proceedings of the 5th ACM/SPEC International Conference on Performance Engineering*, ICPE '14, pages 137–148, New York, NY, USA, 2014. ACM.

[8] 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*, pages 413–422, New York, NY, USA, 2009. ACM.

[9] 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*, Supercomputing '97, pages 1–12, New York, NY, USA, 1997. ACM.

[10] Herbert H. J. Hum and James R. Goodman. Forward state for use in cache coherency in a multiprocessor system, 07 2005.

[11] Hypertransport Technology Consortium. *HyperTransport I/O Link Specification*, revision 3.10c edition, June 2010.

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

[13] Intel. *Intel Xeon Processor E5-1600/E5-2600/E5-4600 Product Families Datasheet - Volume One*, May 2012. Reference Number: 326508, Revision: 002.

[14] Intel Corporation. *Intel Xeon Processor E5-2600 Product Family Uncore Performance Monitoring Guide*, March 2012.

[15] Guido Juckeland, Michael Kluge, Wolfgang E. Nagel, and Stefan Pflüger. Performance analysis with BenchIT: Portable, flexible, easy to use. In *Proceedings of the International Conference on Quantitative Evaluation of Systems*, pages 320–321, 2004.

[16] Kevin M. Lepak, Vydhyanathan Kalyanasundharam, William A. Hughes, Benjamin Tsien, and Greggory D. Donley. Method and apparatus for accelerated shared data migration, 06 2012.

[17] S. Li, T. Hoefler, and M. Snir. NUMA-Aware Shared Memory Collective Communication for MPI. In *Proceedings of the 22nd international symposium on High-performance parallel and distributed computing*, pages 85–96. ACM, Jun. 2013.

[18] Mario Ludwig. Performance-analyse von amd bulldozer-prozessoren, 7 2012. bachelor thesis, Technische Universität Dresden.

[19] John D. McCalpin. Memory bandwidth and machine balance in current high performance computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pages 19–25, 12 1995.

[20] 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, pages 23–23, Berkeley, CA, USA, 1996. USENIX Association.

[21] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller. 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*, pages 261–270, Washington, DC, USA, 2009. IEEE Computer Society.

[22] R. Schöne, D. Hackenberg, and D. 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, pages 9–9, Berkeley, CA, USA, 2012. USENIX Association.

[23] Robert Schöne and Daniel Molka. Integrating performance analysis and energy efficiency optimizations in a unified environment. *Computer Science - Research and Development*, pages 1–9, 2013.

[24] Jan Treibig, Georg Hager, and Gerhard Wellein. Likwid: A lightweight performance-oriented tool suite for x86 multicore environments. In *Proceedings of the 2010 39th International Conference on Parallel Processing Workshops*, pages 207–216. IEEE Computer Society, 2010.

[25] Jan Treibig, Georg Hager, and Gerhard Wellein. likwid-bench: An extensible microbenchmarking platform for x86 multicore compute nodes. In *Tools for High Performance Computing 2011*, pages 27–36. Springer Berlin Heidelberg, 2012.