The definitive Version of Record is published in: Proceedings of the 44th International Conference on Parallel Processing (ICPP'15), https://dx.doi.org/10.1109/ICPP.2015.83.

Cache Coherence Protocol and Memory Performance of the Intel Haswell-EP Architecture

Daniel Molka, Daniel Hackenberg, Robert Schöne, Wolfgang E. Nagel
Center for Information Services and High Performance Computing (ZIH), Technische Universität Dresden
Email: {danial.molka,daniel.hackenberg,robert.schoene,wolfgang.nagel}@tu-dresden.de

***Abstract*—A major challenge in the design of contemporary microprocessors is the increasing number of cores in conjunction with the persevering need for cache coherence. To achieve this, the memory subsystem steadily gains complexity that has evolved to levels beyond comprehension of most application performance analysts. The Intel Haswell-EP architecture is such an example. It includes considerable advancements regarding memory hierarchy, on-chip communication, and cache coherence mechanisms compared to the previous generation. We have developed sophisticated benchmarks that allow us to perform in-depth investigations with full memory location and coherence state control. Using these benchmarks we investigate performance data and architectural properties of the Haswell-EP micro-architecture, including important memory latency and bandwidth characteristics as well as the cost of core-to-core transfers. This allows us to further the understanding of such complex designs by documenting implementation details the are either not publicly available at all, or only indirectly documented through patents.**

I. Introduction and Motivation

The driving force of processor development is the increasing transistor budget which is enabled by Moore's Law. Progress on the core level is still being made. However, most of the additional space is used to increase the core count of contemporary server processors. The peak performance scales linearly with the increasing core count. However, some resources are typically shared between all cores, for example the last level cache and the integrated memory controllers. These shared resources naturally limit scalability to some extend and can create bottlenecks. A profound understanding of these effects is required in order to efficiently use the cache hierarchy and avoid limitations caused by memory accesses.

Contemporary multi-socket x86 servers use point-to-point connections between the processors [1], [2]. The available memory bandwidth scales with the number of processors as each processor has an integrated memory controller. Unfortunately, the distance between the requesting core and the memory controller influences the characteristics of memory accesses. Accesses to the local memory are generally faster than remote memory accesses which require additional transfers via the inter-processor connections. This *Non-Uniform Memory Access* (NUMA) behavior also affects the performance of parallel applications. Typically, each processor is a single NUMA node. However, processors with multiple NUMA domains already exist in the form of multi-chip-modules, e.g., the AMD Opteron 6200 series. Haswell-EP supports a *Cluster-on-Die* mode [3] that splits a single processor into two NUMA domains.

The distributed caches in Haswell-EP based systems are kept coherent using a snooping based protocol [2]. Two snooping modes—source snooping and home snooping—are available. An in-memory directory as well as directory caches are used to mitigate the performance impact of the coherence protocol. In this Paper we analyze the impact of the coherence protocol on the latencies and bandwidths of core-to-core transfers and memory accesses. We compare the three available configurations in a dual socket Haswell-EP system using micro-benchmarks. Furthermore, we investigate the influence on the performance of parallel applications.

II. Related Work

Kottapalli et al. [4] describe the *directory assisted snoop broadcast protocol* (DAS)—an extension of the MESIF protocol [2]. The DAS protocol uses an in-memory directory to accelerate memory accesses in multi-socket Intel systems. Moga et al. [5] introduce an directory cache extension to the DAS protocol which accelerates accesses to cache lines that are forwarded from caches in other NUMA nodes. Both extensions are implemented in the Haswell-EP micro-architecture [3]. Geetha et al. [6] describe a modified version of the *forward* state in the MESIF protocol which increases the ratio of requests the local caching agent (CA) can service without snooping.

Synthetic benchmarks are an important tool to analyze specific aspects of computer systems. In previous work [7], [8] we presented micro-benchmarks to measure the characteristics of distributed caches in NUMA systems with x86_64 processors. They use data placement and coherence state control mechanisms to determine the core-to-core transfers and memory accesses considering the coherence state of the data.

Application based benchmarks are often used to analyze the performance of parallel architectures. Müller et al. introduce SPEC OMP2012 [9]—a benchmark suite for shared memory systems. It contains 14 OpenMP parallel applications from different scientific fields and also covers OpenMP 3.0 features. In [10] SPEC MPI2007 is introduced. It is based on 13 scientific applications that use MPI for parallelization. The scope of SPEC MPI2007 extends to large scale distributed memory systems. However, both suites can be used to evaluate shared memory systems.

III. Haswell-EP architecture

A. ISA Extensions and Core Design

The Intel Haswell micro-architecture is the successor of Sandy Bridge. Table I compares both architectures. The front-end is similar to Sandy Bridge. Instructions are fetched in 16 byte windows and decoded into micro-ops by four decoders. The instruction set architecture (ISA) has been extended to support AVX2 and FMA3 instructions. AVX—which is available since Sandy Bridge—introduced 256 bit floating point instructions. AVX2 increases the width of integer SIMD instructions to 256 bit as well. The FMA3 instruction set introduces fused multiply-add instructions.

As shown in Table I, the out-of-order execution is enhanced significantly in the Haswell micro-architecture. There are more scheduler and reorder buffer (ROB) entries, larger register files, and more load/store buffers in order to extract more instruction level parallelism. Haswell's scheduler can issue eight micro-ops each cycle. The two new issue ports provide an additional ALU and add a third address generation unit. The support of FMA instructions doubles the theoretical peak performance. The data paths to the L1 and L2 cache are widened as well to provide the execution units with more data. However, if 256 bit instructions are detected, the base frequency is reduced to the AVX base frequency and the available turbo frequencies are restricted [11]. Furthermore, the uncore frequency is adapted to the workload dynamically by the hardware [12].

B. Uncore Design

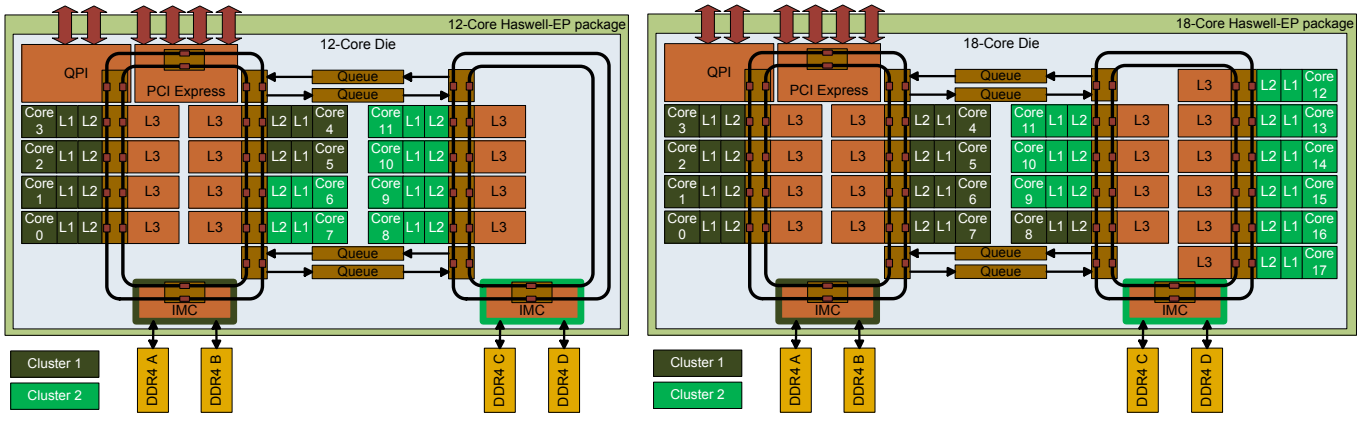
Haswell-EP is available in three variants [16, Section 1.1]—an eight-core die (4, 6, 8-core SKUs[1]), a 12-core die (10, 12-core SKUs), and an 18-core die (14, 16, 18-core SKUs) [11]. The eight-core die uses a single bi-directional ring interconnect. The 12- and 18-core dies use a partitioned design as depicted in Figure 1. In that case, eight cores, eight L3 slices, one memory controller, the QPI interface, and the PCIe controller are connected to one bi-directional ring. The remaining cores (4 or 10), L3 slices, and the second memory controller are connected to another bi-directional ring. Both rings are connected via two bi-directional queues. Some SKUs use partially deactivated dies. We are unaware of a specification which cores are deactivated in specific models. It is possible that this is determined individually during the binning process in order to improve yield. Thus, it can not be ruled out that processors of the same model exhibit different characteristics depending on how balanced the rings are populated.

The ring topology is hidden from the operating system in the default configuration, which exposes all cores and resources in a single NUMA domain. An optional Cluster-on-Die (COD) mode can be enabled in the BIOS. This mode splits each processor into two clusters as depicted in Figure 1. The clusters contain an equal number of cores and are exposed to the operating system as two NUMA nodes. However, the software view on the NUMA topology actually does not match

[1]stock keeping unit

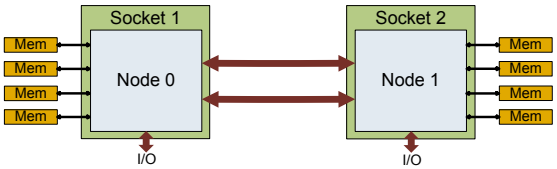
TABLE I
Comparison of Sandy Bridge and Haswell micro-architecture

Micro-architecture	Sandy Bridge	Haswell
References	[13], [14, Section 2.2]	[15], [14, Section 2.1]
Decode	4(+1) x86/cycle	
Allocation queue	28/thread	56
Execute	6 micro-ops/cycle	8 micro-ops/cycle
Retire	4 micro-ops/cycle	
Scheduler entries	54	60
ROB entries	168	192
INT/FP registers	160/144	168/168
SIMD ISA	AVX	AVX2
FPU width	2× 256 Bit (1× add, 1× mul)	2× 256 Bit FMA
FLOPS/cycle	16 single / 8 double	32 single / 16 double
Load/store buffers	64/36	72/42
L1D accesses per cycle	2× 16 byte load + 1× 16 byte store	2× 32 Byte load + 1× 32 Byte store
L2 bytes/cycle	32	64
Memory Channels	4× DDR3-1600 up to 51.2 GB/s	4× DDR4-2133 up to 68.2 GB/s
QPI speed	8 GT/s (32 GB/s)	9.6 GT/s (38.4 GB/s)

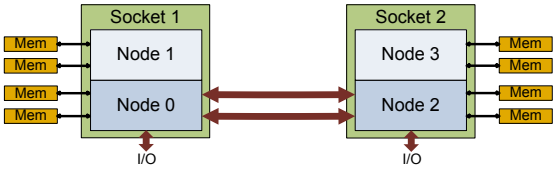


(a) 12 core Haswell-EP (b) 18 core Haswell-EP

Fig. 1. Haswell-EP block diagram: There are two rings with one memory controller (IMC) each. QPI and PCIe links are connected to the first ring. The rings are connected with bi-directional queues. In the default configuration all cores can access the whole L3 cache and memory is interleaved over all four channels. The Cluster-on-Die mode splits the chip into two NUMA nodes which share one memory controller each.



(a) default configuration



(b) Cluster-on-Die mode

Fig. 2. Dual socket Xeon E5 v3 system: In the default configuration, the partitioned chip design is not exposed to the operating system. Therefore, the system presents itself as two NUMA nodes. If COD mode is enabled each socket is divided into two nodes, thus four NUMA nodes are visible.

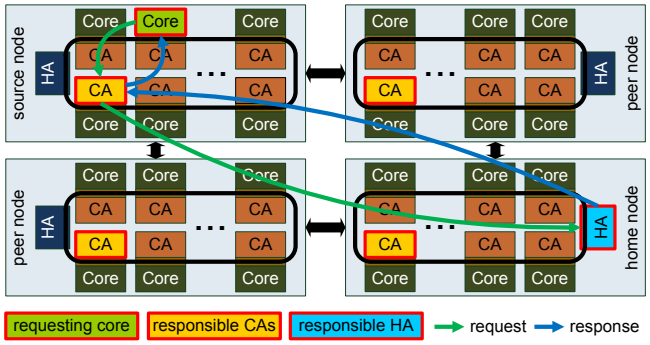


Fig. 3. The cores forward requests to the caching agents within their node. If the request cannot be serviced locally, it is forwarded to the home agent. Snoops of the CAs in the peer nodes are either handled by the CA or the HA (see Section IV-B and Section IV-C)

the hardware configuration. The 12 core chip (8-core + 4-core ring) presents itself as two 6 core nodes. The 18 core chip (8-core + 10-core ring) appears as two nodes with 9 cores each. One additional level of complexity is added in dual socket Haswell-EP systems as depicted in Figure 2. The topology consists of two NUMA nodes if COD is disabled and four NUMA nodes if the COD mode is used.

IV. Haswell-EP Cache Coherence Mechanisms

A. MESIF Implementation

Cache coherence is maintained using the MESIF protocol [2]. It inherits the *modified*, *exclusive*, *shared*, and *invalid* states from MESI and adds the state *forward* to enable cache-to-cache transfers of shared cache lines. The *forward* state designates one shared copy as being responsible to forward the cache line upon requests. The protocol ensures that at most one copy in the *forward* state exists at any one time. It does not ensure that the data is provided by the closest available copy.

The protocol is implemented by caching agents (CAs) within each L3 slice and home agents (HAs) within each memory controller as depicted in Figure 3. The cores send requests to the caching agents in their node. The responsible CA is derived from a requested physical address using a hash function [16, Section 2.3]. It provides data from its appendent L3 slice or obtains a copy from another core's L1 or L2 cache as indicated by the core valid bits [7]. In case of an L3 miss, the caching agent forwards the request to the home agent which provides the data from memory. The caching agents in other nodes need to be checked as well which can be implemented using source snooping (see Section IV-B) or home snooping (see Section IV-C).

The MESIF protocol can be augmented with directory support [4]. The "directory assisted snoop broadcast protocol" stores 2-bit of directory information for each cache line in the memory ECC bits [17, Section 2.1.2]. They encode 3 states [5]. A cache line in the *remote-invalid* state is not cached in other NUMA nodes. The *snoop-all* state indicates that a potentially modified copy exists in another node. The state *shared* indicates that multiple clean copies exist. This information can be used to limit the amount of snoops sent. This is particularly important in systems with several sockets, since broadcasts quickly become expensive for an increasing number of nodes. On the other hand, necessary snoops are delayed until the directory lookup is completed. It is also possible to send snoops to all caching agents in parallel to the directory access. In that case the home agent can forward the data from memory without waiting for the snoop responses if the directory state allows it. However, the snoop traffic is not decreased in that case. According to [16, Section 2.5], the directory should not be used in typical two-socket systems. Our test system does not expose a BIOS option to manually enable directory support, but it is automatically enabled in COD mode (see Section IV-D).

B. Source Snoop Mode

In the source snoop mode of the MESIF protocol [2], snoops are sent by the caching agents. In case of an L3 miss, the caching agent broadcasts a snoop requests to the appropriate caching agents in all other nodes as well as to the home agent in the home node. If another caching agent has a copy of the cache line in state *modified*, *exclusive*, or *forward* it will be forwarded directly to the requester. All caching agents also notify the home agent about their caching status of the requested cache line. The home agent collects all snoop responses, resolves conflicts, provides data from memory if necessary, and completes the transaction. This mode has the lowest possible latency. However, it generates a lot of traffic on the interconnect.

C. Home Snoop Mode

The MESIF protocol [2] also supports a home snoop mode in which snoops are sent by the home agents. In that case, the caching agent that requests a cache line does not broadcast snoops. Instead it forwards the request to the home agent in the home node which then sends snoops to other caching agents. The transaction then continues as in the source snoop variant. The forwarding to the home node adds latency. However, the

home agent can implement a directory to improve performance (see Section IV-A).

D. Cluster on Die Mode (COD)

In the *COD mode* each processor is partitioned as depicted in Figure 1 resulting in a system with four NUMA nodes as shown in Figure 2b. The coherence protocol then uses a home snoop mechanism with directory support as described in Section IV-A. Furthermore, Haswell-EP also includes directory caches to accelerate the directory lookup [3]. However, with only 14 KiB per home agent these caches are very small. Therefore, only highly contended cache lines are entered in the so-called "HitME" cache [5]. These are also referred to as migratory cache lines [3], i.e., cache lines that are frequently transferred between nodes. Consequently, only accesses that already require snoops result in allocations in the HitME cache (see Fig 3. in [5]). Since cache lines that are in the state *remote-invalid* do not require snoops, the first access to such a line can transfer a cache line to a remote caching agent without entering it in the directory cache in the home node. An entry in the directory cache can only be allocated if cache lines are forwarded between caching agents in different nodes and the requesting caching agent is not in the home node. If the directory cache contains an entry for a cache line, the corresponding entry in the in-memory directory is in the *snoop-all* state.

The directory cache stores 8-bit presence vectors that indicate which nodes have copies of a cache line [3]. When a request arrives at the home node, the directory cache is checked first. If it contains an entry for the requested line, snoops are sent as indicated by the directory cache. If the request misses in the directory cache, the home agent reads the status from the in-memory directory bits and sends snoops accordingly, or directly sends the data from memory if no snoops are required.

V. Benchmarking Methodology

A. Tests System

Our test system contains two 12-core Haswell-EP processors (see Table II). Each socket has four DDR4 memory channels running at 2133 MHz. Thus, 68.3 GB/s of memory bandwidth are available per socket. The sockets are connected with two QPI links that operate at 9.6 GT/s. Each link provides a bi-directional bandwidth of 38.4 GB/s. Both links combined enable 76.8 GB/s of inter-socket communication (38.4 GB/s in each direction).

The coherence protocol can be influenced by the *Early Snoop* and *COD mode* settings in the BIOS[2]. The default configuration is *Early Snoop* set to *auto* (enabled) and *COD mode* set to *disabled* which results in a source snoop behavior. If *Early Snoop* is disabled a home snoop mechanism is used. If *COD mode* is enabled the newly introduced *Cluster-on-Die* mode [3] is activated. In that case the setting for *Early Snoop* has no influence.

[2]Advanced → Chipset Configuration → North Bridge → QPI Configuration → QPI General Configuration

TABLE II
Dual socket Haswell-EP test systems

System	Bull SAS bullx R421 E4 [18]
Processors	2x Intel Xeon E5-2680 v3
cpuid	family 6, model 63 stepping 2
Cores/threads	2x12 / 2x24
Core clock	2.5 GHz[3]
Uncore/NB clock	variable, up to 3.0 GHz[4]
L1/L2 cache	2x 32 KiB / 256 KiB per core
L3 cache	30 MiB per chip
Memory	128 GiB (8x 16 GiB) PC4-2133P-R
QPI speed	9.6 GT/s (38.4 GB/s)
Operating System	Ubuntu 14.04.2 LTS, kernel 3.19.1-031901-generic
BIOS version	1.0 build date 07/30/2014

B. Microbenchmark Design

We use an extended version of the synthetic microbenchmarks presented in [7]. They include data placement and coherence state control mechanisms that place cache lines in a fully specified combination of core id, cache level, and coherence state. The data set size determines the cache level in which the data is located. If the data set does not fit into a certain cache level, optional cache flushes can be used to evict all cache lines from higher cache levels into the cache level that is large enough. The latency and bandwidth measurements can be performed on the core that performed the data placement to evaluate the local cache hierarchy. The measurement can also be performed on another core in order to estimate the performance of core-to-core transfers. The bandwidth can also be measured for concurrent accesses of multiple cores. Furthermore, the *libnuma* based memory affinity control enables the analysis of NUMA characteristics for main memory accesses.

The coherence state of the date can be controlled as well.

- State *modified* is enforced by: 1) writing to the data.
- State *exclusive* is generated by: 1) writing the data which invalidates all other copies, 2) removing the *modified* copy using *clflush*, 3) reading the data.
- State *shared* and *forward* are created by: 1) caching data in state *exclusive*, 2) another core reading the data.

The order of accesses determines which core gets the *forward* copy. It is also possible to define a list of cores that should share the data in order to span multiple nodes.

The benchmarks are designed to operate in a stable environment, i.e., at fixed frequencies. We therefore disable the Turbo Boost feature and set the core frequency to the nominal 2.5 GHz. However, the uncore frequency scaling and frequency reductions for AVX workloads can influence the results. Occasionally the bandwidth benchmarks show better performance than presented in this paper. The higher performance levels are not reliably reproducible. We selected measurements for the typical case, i.e., curves that do not show frequency related jumps.

[3]2.1 GHz base frequency for AVX workloads [11]

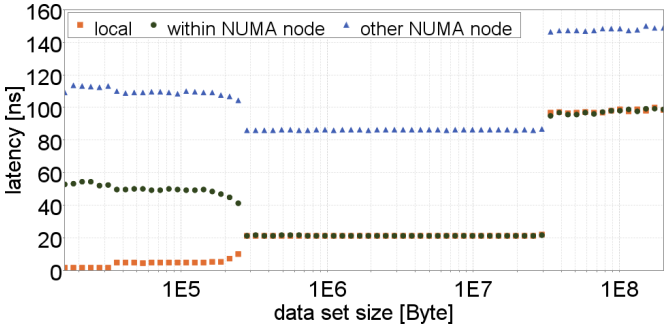
[4]Uncore frequency scaling automatically adjusts frequency [12]

VI. Latency Results

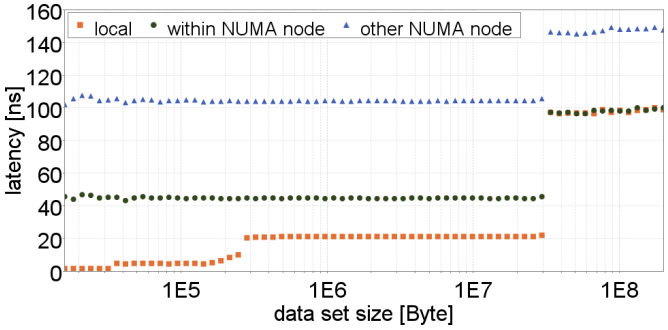
A. Source Snoop Mode

Latency measurements for the default configuration are depicted in Figure 4. The access times of 1.6 ns (4 cycles), 4.8 ns (12 cycles), and 21.2 ns (53 cycles) for reads from the local L1, L2, and L3 are independent of the coherence state. In contrast, core-to-core transfers show noticeable differences for *modified*, *exclusive*, and *shared* cache lines.

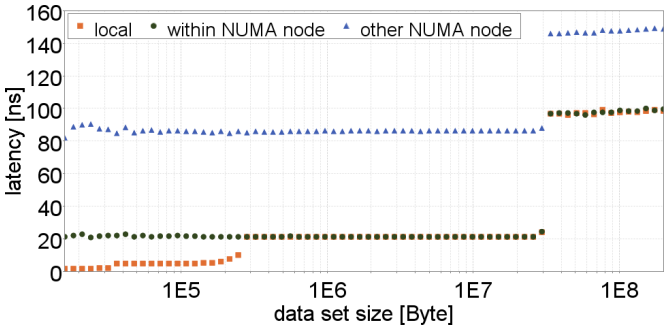
If another core contains a *modified* copy in the L1 or L2 cache, the corresponding core valid bit is set in the L3 cache. Thus, the caching agent snoops the core which then forwards the cache line to the requester. The latency is approximately 53 ns and 49 ns in that case for transfers from the L1 and L2 cache, respectively. If the other core has already evicted the *modified* cache lines, the inclusive L3 cache services the requests without delay (21.2 ns). This is possible because the write back to the L3 also clears the core valid bit.



(a) modified



(b) exclusive



(c) shared (two copies in one node)

Fig. 4. Memory read latency in default configuration (source snoop): Comparison of accesses to a core's local cache hierarchy (local) with accesses to cache lines of another core in the same NUMA node (within NUMA node) as well as accesses to the second processor (other NUMA node (1 hop QPI)).

Unmodified data is always delivered from the inclusive L3 cache. However, accesses to *exclusive* cache lines have a higher latency of 44.4 ns if they have been placed in the L3 cache by another core. This is because the caching agent has to snoop the other core as the state could have changed to *modified*. This also happens if the core does not have a copy anymore as *exclusive* cache lines are evicted silently which does not clear the core valid bit. If multiple core valid bits are set, core snoops are not necessary as the cache line can only be in the state *shared*. In that case the L3 cache forwards a copy without delay (21.2 ns).

Accesses to cache lines in the other socket follow the same pattern. *Modified* cache lines in a remote L1 or L2 cache have to be forwarded from the core that has the only valid copy in that case. The latency is 113 ns from the L1 and 109 ns from the L2. If a valid copy exists in the L3 cache it is forwarded from there with a latency of 86 ns for immediate replies or 104 ns if a core snoop is required. The memory latency is 96.4 ns for local and 146 ns for remote accesses.

B. Home Snoop Mode

Figure 5 depicts the difference between source snooping (default) and home snooping (*Early Snoop* disabled). As expected, all accesses that are handled by the L3's caching agents without any external requests are not affected by the snoop behavior. However, the latency of remote cache access is noticeably higher if home snooping is used. It increases from 104 ns to 115 ns (+10.5%). The local memory latency increases from 96.4 ns to 108 ns (+12%). Both effects can be explained by the delayed snoop broadcast in the home snoop protocol. This also shows that directory support is not activated as the local memory latency would not increase if it was enabled. The remote memory latency of 146 ns is identical to source snoop mode, as the request is sent from the requesting caching agent directly to the remote home agent in both cases.

C. Cluster-on-Die Mode

Activating the COD mode doubles the amount of possible distances. In addition to local accesses, core-to-core transfers within the node, and transfers via QPI we also have to consider

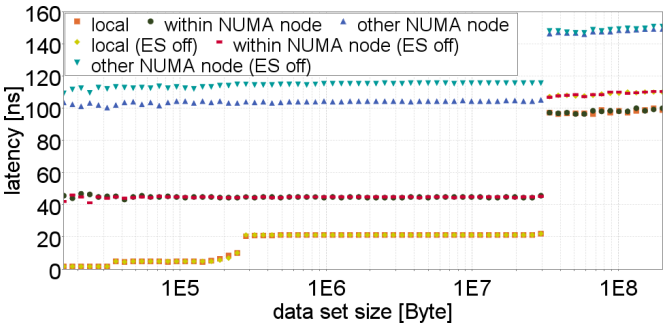
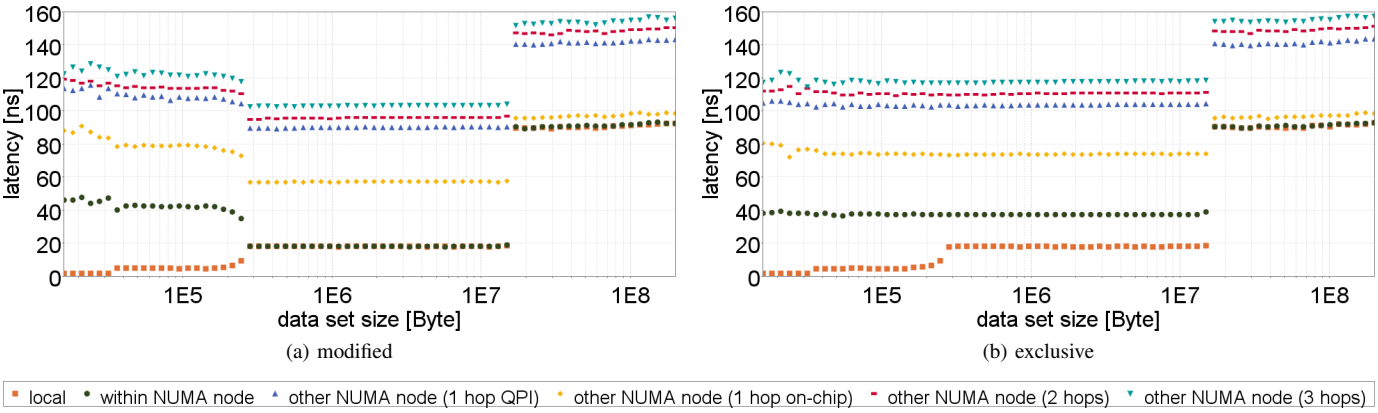


Fig. 5. Memory read latency: Comparison of source snoop and home snoop, cached data in state *exclusive*



(a) modified (b) exclusive

Fig. 6. Memory read latency: Comparison of accesses to a core's local cache hierarchy (local) with accesses to cache lines of other cores in the same NUMA node (within NUMA node) and on-chip transfers between the clusters (other NUMA node (1 hop on-chip)) as well as transfers between the two sockets. This includes connections between the two nodes that are coupled directly via QPI (other NUMA node (1 hop QPI)) as well as transfers that include additional hops between the clusters within the sockets. (other NUMA node (2 hops) / other NUMA node (3 hops)). The measurements use the first core in every node to perform the data placement.

transfers between the clusters in a socket as well as inter-socket communication between nodes that are not directly connected via QPI. Figure 6 shows the clearly distinguishable performance levels. Transfers between on-chip clusters are much faster than QPI transfers. The L3 forwards cache lines slightly faster (18.0 or 37.2 ns compared to 21.2 or 44.4 ns with COD disabled). However, the inclusive behavior of the L3 cache is limited to the cores within a NUMA node. Core-to-core transfers between the nodes involve the home node of the data, with latencies increasing from 18.0 and 37.2 ns to 57.2 and 73.6 ns if data is delivered from the L3 of the second node. Remote L3 access latency increases from 86 ns (*modified*) and 104 ns (*exclusive*) to 90 and 104 ns, 96 and 111 ns, or 103 and 118 ns depending on the number of hops. In these measurements the home agent is in the same node as the caching agents that forward the data. The latency further increases if three nodes are involved.

The memory latency with COD drops from 96.4 to 89.6 ns for local accesses. Accesses to memory attached to the second node are slower but still slightly faster than local accesses with COD disabled. If memory from the second processor is used, the latency depends on the number of hops within the nodes: 141 ns for 1-hop-connections (node0-node2), 147 ns for two hops (node0-node3 and node1-node2) and 153 ns for three hops (node1-node3).

The asymmetrical chip layout which is mapped to a balanced NUMA topology causes performance variations. The six cores in the first node (core 0-5 in Figure 1a) are connected to the same ring, thus have similar performance characteristics. The average distance to the individual L3 slices is almost identical for all cores. Consequently, the average time for forwarding the request from the core to the responsible caching agent is almost independent of the location on the ring (assuming data is distributed evenly). The distance from the caching agent to the home agents also does not depend on the requesting core's location on the ring, as the location of the agents is determined by the physical address (hash function selects the CA, NUMA node determines HA). In the second node however, the situation is different. Two cores are connected to the first ring, but allocate memory from the memory controller connected to the second ring by default (assuming NUMA aware software). On average, two thirds of the cores' memory requests are serviced by the four caching agents connected to the second ring. For the four cores connected to the second ring, one third of their requests is serviced by the two caching agents on the first ring.

Table III shows the different performance characteristics in comparison to the source snooping and home snooping modes on our test system. The latency reduction for local accesses is substantial for the cores in the first node (-15% for L3 and -7.1% for memory), slightly lower for the four cores in the second node on the second ring (-13.2% and -6.2%), and only

TABLE III
LATENCY IN NANOSECONDS; RESULTS FOR L3 CACHE LINES ARE FOR STATE *exclusive*.

source		default configuration	Early Snoop disabled	COD mode		
				first node	second node	
					first ring (core 6 and 7)	second ring (cores 8-11)
L3	local	21.2		18.0 (-15%)	20.0 (-5.6%)	18.4 (-13.2%)
	remote first node	104	115 (+12%)	104 (+/-0)	108 (+3.8%)	111 (+6.7%)
	remote 2nd node			113 (+8.7%)	118 (+13.5%)	120 (+15.4%)
memory	local	96.4	108 (+10.5%)	89.6 (-7.1%)	94.0 (-2.5%)	90.4 (-6.2%)
	remote first node	146	148 (+1.3%)	141 (-3.4%)	145 (-0.7%)	148 (+1.3%)
	remote 2nd node			147 (+0.7%)	151 (+3.4%)	153 (+4.8%)

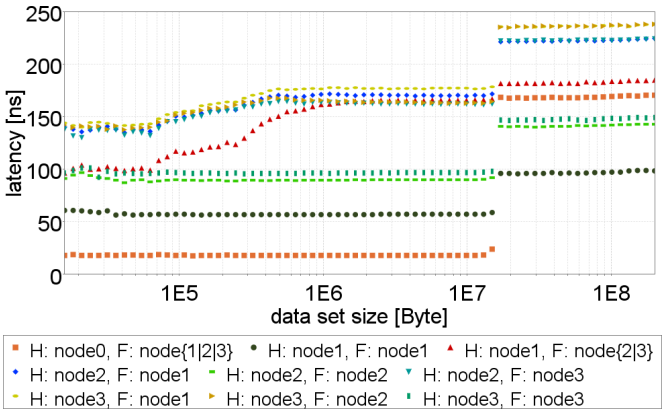


Fig. 7. Memory read latency: core in node0 accesses cache lines with *forward* copies in another node (F:) which also have different home nodes (H:). The home nodes also contain a copy which is in the *shared* state if the *forward* copy is in another node. The sharing also affects the memory latency because of stale state in the in-memory directory.

small for the two cores in the second node but on the first ring (-5.6% and -2.5%). In contrast, remote cache accesses are up to 15.4% slower. Remote memory latency varies between 3.4% faster and 4.8% slower. These cases only cover transfers that do not require broadcasts.

Figure 7 depicts measurements of accesses from node0 to data that has been used by two cores somewhere in the system which also covers situations where broadcasts are required. Small data set sizes show an unexpected behavior if the data is cached in *forward* state outside the home node. Surprisingly, the latency for L1 and L2 accesses is significantly lower than for L3 accesses[5]. Performance counter readings[6] show that data is in fact forwarded from the memory in the home node for data set sizes up to 256 KiB[7]. According to [6], this is possible if the in-memory directory indicates that cache lines are in state *shared*. This explains why data is delivered faster if node0 is the home node as no QPI transfer is involved. However, if the in-memory directory state would be responsible for this behavior, it would not depend on the data set size. This suggests that the effect is caused by the directory cache. For larger data set sizes an increasing number of cache lines is forwarded by the other node[8] as expected according to the state specifications in the MESIF protocol [2]. For 2.5 MiB and above the percentage of DRAM responses are negligible. The gradual decline matches the assumption that the directory cache is involved and the hit rates diminish. We therefore conclude that the *AllocateShared* policy [5] of the directory cache is implemented and an entry is allocated if the cache lines are transferred to another node in state *forward*. This sets the in-memory directory to state *snoop-all*. However, the presence vector in the directory cache indicates that the cache line is shared and allows forwarding the valid memory copy as long as the entry is not evicted.

Table IV details the L3 latencies for a core in node0 that accesses shared cache lines with different distributions among the nodes. The values express the performance for data set sizes above 2.5 MiB, i.e., the divergent behavior for smaller data sets is not considered here. If a valid copy is available in node0, it is forwarded to the requesting core with the same 18.0 ns latency discussed above (case COD, first node in Table III). Interestingly, this also is the case for accesses to *shared* cache lines in the local L1 or L2 cache if the *forward* copy is in another node. This indicates, that in this case the responsible caching agent is notified in order to reclaim the *forward* state. In the cases on the diagonal, a *forward* copy is found in the home node (the local snoop in the home node is carried out independent of the directory state [5]). The results are equal to the forwarding of *modified* lines from the home node (see Figure 6a). The remaining cases show the performance of forwarding cache lines from another node which is snooped by the home node. The range from 162 to 177 ns is caused by the different distances between the involved nodes. The worst case of 177 ns is more than twice as high as the 86 ns in the default configuration.

Table V shows the impact of the directory protocol on memory accesses. For data sets larger than 15 MiB data was read by multiple cores but is already evicted from the L3 caches. The diagonal represents cases where data is only shared within the home node. In that case no directory cache entries are allocated and the in-memory directory remains in state *remote-invalid*. Thus, the data is delivered by the home node without a broadcast. All other cases apparently require a broadcast. Thus, the directory state has to be *snoop all*. This is another indication that the *AllocateShared* policy [5] of the directory cache is implemented, i.e., directory cache entries are allocated if a cache line is entered in state *forward* in another node. Consequently, the in-memory state is changed to *snoop all* instead of *shared* which would be used without the directory cache [4]. Thus, the home node broadcasts a snoop request which adds between 78 and 89 ns to the latency.

[5]The behavior doesn't change if hardware prefetchers are disabled in BIOS

[6]event MEM_LOAD_UOPS_L3_MISS_RETIRED:REMOTE_DRAM

[7]the variation below 256 KiB also occurs if all caches are flushed and data is read directly from local DRAM, thus it is apparently caused by DRAM characteristics, e.g., the portion of accesses that read from already open pages

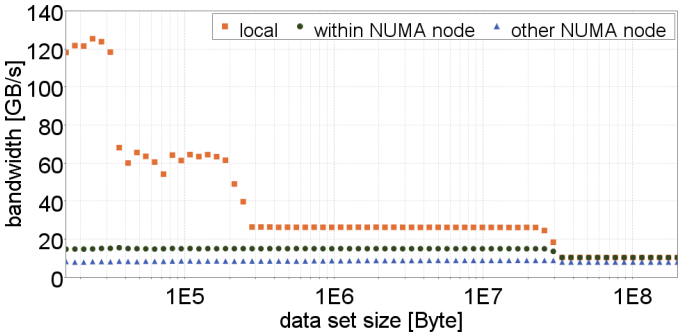
[8]indicated by MEM_LOAD_UOPS_L3_MISS_RETIRED:REMOTE_FWD

TABLE IV
LATENCY IN NANOSECONDS FOR ACCESSES FROM A CORE IN NODE0 TO L3 CACHE LINES WITH MULTIPLE SHARED COPIES.

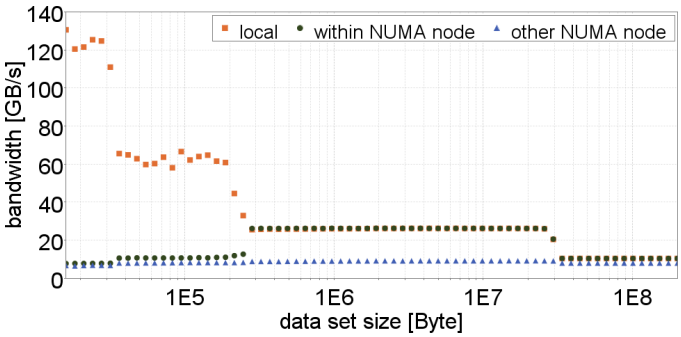
node with *forward* copy	home node (*shared* copy)			
	node0	node1	node2	node3
node0	18.0	18.0	18.0	18.0
node1	18.0	57.2	170	177
node2	18.0	166	90.0	166
node3	18.0	169	162	96.0

TABLE V
LATENCY IN NANOSECONDS FOR ACCESSES FROM A CORE IN NODE0 TO DATA IN MAIN MEMORY THAT HAS BEEN SHARED BY MULTIPLE CORES.

node that had *forward* copy	home node			
	node0	node1	node2	node3
node0	89.6	182	222	236
node1	168	96.0	222	236
node2	168	182	141	236
node3	168	182	222	147



(a) exclusive, default configuration



(b) modified, default configuration

Fig. 8. Memory read bandwidth in default configuration (source snoop): Comparison of accesses to a core's local cache hierarchy (local) with accesses to cache lines of another core in the same NUMA node (within NUMA node) as well as accesses to the second processor (other NUMA node (1 hop QPI)).

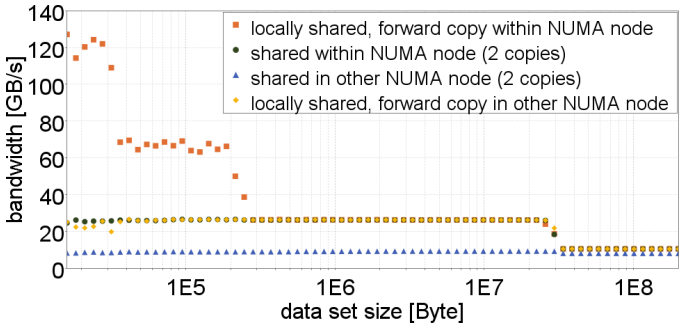


Fig. 9. Memory read bandwidth in default configuration (source snoop): Accesses to shared cache lines.

VII. Bandwidth Results

A. Single Threaded Bandwidths

Figure 8 shows the single threaded memory read bandwidth in the default configuration. Using 256 bit wide load instructions, data can be read from the L1 and L2 cache with 127.2 and 69.1 GB/s from the local L1 and L2 cache, respectively. The L1 and L2 measurements show unusually high variability which is probably caused by frequency changes between the nominal and the AVX base frequency. If 128 bit instructions are used, the bandwidth is limited to 77.1 and 48.2 GB/s as they do not fully utilize the width of the data paths. Accesses beyond the local L2 cache show little differences between AVX and SSE loads.

The core-to-core bandwidth reflect the behavior that has already been observed in the latency measurements. Only *modified* cache lines are actually forwarded from a core's L1 or L2 cache. The bandwidth is 7.8 and 10.6 GB/s for on-chip transfers and 6.7 and 8.1 GB/s for transfers between the sockets. If *modified* cache lines are evicted to the L3 cache, they can be read with 26.2 GB/s from the local and 9.1 GB/s remote L3. Accesses to *exclusive* cache lines are always serviced by the L3. However, the bandwidth only reaches 26.2 GB/s for accesses to cache lines in the local L3 that have been evicted by the requesting core. If another core needs to be snooped, the bandwidth is limited to 15.0 GB/s for the local L3 and 8.7 GB/s for the remote L3. Since *exclusive* cache lines are evicted silently, the penalty also occurs if the other core does not contain a copy anymore.

Figure 9 shows the read bandwidth of *shared* cache lines. Local L1 and L2 accesses only achieve the performance measured for *modified* or *exclusive*, if the *forward* copy is in the requesting core's node. If the *forward* copy is held by the other processor, the L1 and L2 bandwidth are limited to the L3 bandwidth of 26.2 GB/s. As already mentioned in Section VI-C, this suggests that the L3 is notified of the repeated access in order to reclaim the *forward* state. Shared data can be read with 26.2 GB/s from the local L3 and 9.1 GB/s from the remote L3 without snooping other cores, as the data is guaranteed to be valid.

Bandwidths are also affected by the coherence protocol configuration as detailed in Table VI. Deactivating *Early Snoop* decreases the local memory bandwidth from 10.3 to 9.6 GB/s. On the other hand, there is a small increase in the remote L3 and memory bandwidths. The COD mode again shows significant performance variations between cores in the different nodes within the chip. Local L3 and memory bandwidth benefit from enabling COD mode. The L3 bandwidth

TABLE VI
Single threaded read bandwidth in GB/s; results for L3 cache lines are for state *exclusive*.

source		default configuration	Early Snoop disabled	COD mode		
				first node	second node	
					first ring (core 6 and 7)	second ring (cores 8-11)
L3	local	26.2		29.0	27.2	27.6
	remote first node	8.8	8.9	8.7	8.3	8.4
	remote 2nd node			8.3	8.0	8.1
memory	local	10.3	9.5	12.6	12.4	12.6
	remote first node	8.0	8.2	8.3	7.8	8.1
	remote 2nd node			8.0	7.4	7.5

TABLE VII
MEMORY BANDWIDTH DEPENDING ON EARLY SNOOP CONFIGURATION

source	mode	number of concurrently reading cores											
		1	2	3	4	5	6	7	8	9	10	11	12
local memory	default	10.3	21.0	32.6	41.4	50.1	57.7	62.5	62.7	63.0	63.1		62.7
	early snoop off	9.6	18.5	28.7	37.4	45.6	53.9	59.5	62.7				
remote memory	default	8.0	13.1	14.1	14.7	15.2	15.6	15.9	16.3	16.5	16.6	16.8	
	early snoop off	8.2	16.1	24.0	28.3	30.2	30.4	30.6					

increases significantly in the first node (+10.6%). However, the performance in the second node does not benefit that much (+3.8%/+5.3% depending on the ring to which the core is connected). The local memory bandwidth increases by more than 20% in all cases. On the other hand, remote cache and memory bandwidth are reduced in COD mode.

B. Aggregated Bandwidths

As in earlier ring based designs, the L3 cache scales well with the number of cores. The performance is identical for source snoop and home snoop mode. The read bandwidth scales almost linearly from 26.2 GB/s for one core to 278 GB/s with 12 cores (23.2 GB/s per core). While the measurements using only a few cores are reproducible, the measured bandwidth for 7 to 12 cores strongly differs between measurements. Up to 343 GB/s have been reached in our experiments. We attribute this unpredictable behavior to the uncore frequency scaling that dynamically increases the L3 bandwidth. The write bandwidths scale from 15 GB/s to 161 GB/s—with occasional performance boosts up to 210 GB/s. In COD mode 154 GB/s of read and 94 GB/s of write bandwidth are available per node. The difference between the nodes on a chip is negligible.

Table VII shows the bandwidth scaling for the source snoop and home snoop mode. The read bandwidth from local memory is lower for up to seven concurrently reading cores if *Early Snoop* is disabled. For eight and more cores the effect is negligible. Approximately 63 GB/s are reached in both configurations. The write bandwidth is 7.7 GB/s for a single core. It increases up to 26.5 GB/s using five cores and declines afterwards. Using all cores we measure 25.8 GB/s. The read bandwidth from remote memory is much higher if *Early Snoop* is disabled. It reaches 30.6 GB/s compared to only 16.8 GB/s in the default configuration. The COD mode bandwidths are detailed in Table VIII. The local bandwidth per node is 32.5 GB/s. The bandwidth of node-to-node transfers is 18.8 GB/s within on chip and—depending on the number of hops—15.6 or 14.7 GB/s for transfers between the sockets.

TABLE VIII
MEMORY READ BANDWIDTH (GB/S) SCALING IN COD MODE

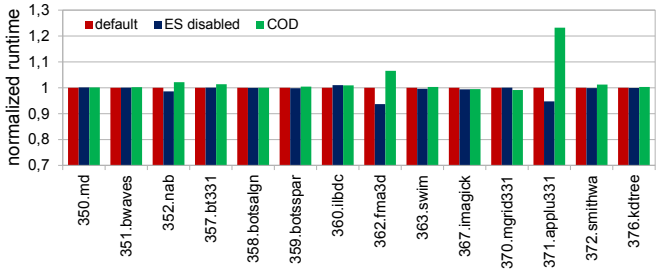
source	number of concurrently reading cores					
	1	2	3	4	5	6
local memory	12.6	24.3	30.6	32.5		
node0-node1	7.0	15.2	18.6	18.8		
node0-node2	5.9	12.8	15.4	15.6		
node0-node3	5.5	12.2	14.4	14.7		
node1-node3						

VIII. APPLICATION PERFORMANCE

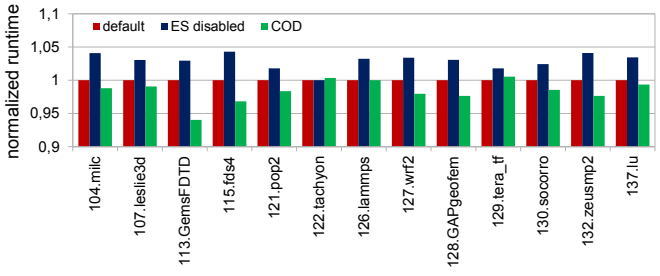
We have selected the SPEC OMP2012 and SPEC MPI2007 application benchmarks in order to evaluate the influence of the coherence protocol mode on the performance of shared memory and message passing applications. Both suites have been compiled using Intel Composer XE 2013 SP1 update 3. The results (median of 3 iterations) are depicted in Figure VIII. Threads or processes are pinned to individual cores via KMP_AFFINITY and –bind-to-core, respectively.

The SPEC OMP2012 results for the home snoop mode (*Early Snoop* disabled) are almost identical to the default configuration. 12 out of 14 benchmarks are within +/- 2% of the original runtime. 362.fma3d and 371.applu331 show a ca. 5% reduction of the runtime if *Early Snoop* is disabled. If *COD* mode is enabled, the runtime of these two benchmarks increases noticeably—up to 23% for 371.apply. They are apparently influenced by the worst case latency penalties of COD mode. No benchmark in the SPEC OMP2012 suite benefits from enabling COD mode.

The SPEC MPI2007 results are very uniform. Disabling *Early Snoop* has a tendency to slightly decrease the performance while enabling *COD* mode mostly increases the performance, i.e., the benchmarks reflect the changes in local memory latency and bandwidth. This is to be expected, since MPI programs primarily use local memory.



(a) SPEC OMP2012



(b) SPEC MPI2007

Fig. 10. Coherence protocol configuration vs. application performance

IX. Conclusion

Understanding the design and operation principles of microprocessor memory hierarchy designs is paramount for most application performance analysis and optimization tasks. In this paper we have presented sophisticated memory benchmarks along with an in-depth analysis of the memory and cache design of the Intel Haswell-EP micro-architecture. The increasing number of cores on a processor die strongly increases the complexity of such systems, e.g. due to the use of directory-based protocols that used to be applicable only to large multi-chip-systems. Our work reveals that providing cache coherence in such a system does have considerable overhead and presents scalability challenges, in particular for latency sensitive workloads. Furthermore, the performance variations that have been introduced with the Haswell-EP architecture due to complex and unpredictable frequency control mechanisms may turn out to be equally challenging for any optimization task.

The different coherence protocol modes have a significant impact on the latencies and bandwidths of core-to-core transfers as well as memory accesses on Haswell-EP. The default source snoop mode is optimized for latency, which—according to our application benchmarks—seems to be a good choice. The home snooping mode enables higher bandwidth for inter-socket transfers, creating significant advantages in our micro-benchmarks. However, it does not have a large benefit in our benchmark selection. On the other hand, the latency of the local memory is increased which reduces the performance of NUMA optimized workloads.

The optional COD mode reduces the local memory latency and improves its bandwidth which slightly increases performance in some cases. However, mapping the asymmetrical chip layout to a balanced NUMA topology creates performance variations between the cores with latency reductions between 5 and 15% depending on the location of the core on the chip. The benefit of COD mode would probably be more substantial if the hardware topology would match the software visible NUMA nodes. Moreover, the complex transactions in the coherence protocol—especially those that involve three nodes—can result in severe performance degradations. The worst case latency of core-to-core transfers is doubled. Furthermore, the memory latency increases significantly if the directory information is outdated due to silent evictions from the L3 cache. However, our measurements with shared memory parallel applications show that the performance impact is mostly negligible. Only one of the SPEC OMP2012 benchmarks shows a significant performance degradation.

Our conclusion is that processors with single-chip NUMA and directory support may not yet be mandatory, but will probably become standard in the near future. Our results–both regarding fundamental operational principles and performance impacts–are crucial to understand these systems, which represents an important pre-requisite for most performance optimization and performance modeling tasks.

References

[1] P. Conway and B. Hughes, "The amd opteron northbridge architecture," *IEEE Micro*, vol. 27, no. 2, pp. 10–21, 03 2007.

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

[3] R. Karedla, "Intel xeon e5-2600 v3 (haswell) architecture & features," 2014. [Online]: http://repnop.org/pd/slides/PD_Haswell_Architecture.pdf

[4] S. Kottapalli, H. Neefs, R. Pal, M. Arora, and D. Nagaraj, "Extending a cache coherency snoop broadcast protocol with directory information," Patent, 2, 2012, uS Patent App. 12/860,340. [Online]: https://www.google.com.tr/patents/US20120047333

[5] A. Moga, M. Mandviwalla, V. Geetha, and H. Hum, "Allocation and write policy for a glueless area-efficient directory cache for hotly contested cache lines," Patent, 1, 2014, uS Patent 8,631,210. [Online]: https://www.google.com.tr/patents/US8631210

[6] V. Geetha, J. Chamberlain, S. Kottapalli, G. Kumar, H. Neefs, N. ACHTMAN, and B. Jung, "Improving value of forward state by increasing local caching agent forwarding," Patent, 7, 2013, wO Patent App. PCT/US2012/020,408. [Online]: https://www.google.com.tr/patents/WO2013103347A1?cl=en

[7] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller, "Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system," in *International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2009. [Online]: http://dx.doi.org/10.1109/PACT.2009.22

[8] D. Hackenberg, D. Molka, and W. E. Nagel, "Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2009. [Online]: http://dx.doi.org/10.1145/1669112.1669165

[9] M. S. Müller, J. Baron, W. C. Brantley, H. Feng, D. Hackenberg, R. Henschel, G. Jost, D. Molka, C. Parrott, J. Robichaux, P. Shelepugin, M. Waveren, B. Whitney, and K. Kumaran, "Spec omp2012 - an application benchmark suite for parallel systems using openmp," in *OpenMP in a Heterogeneous World*, ser. Lecture Notes in Computer Science, B. Chapman, F. Massaioli, M. S. Müller, and M. Rorro, Eds. Springer Berlin Heidelberg, 2012, vol. 7312, pp. 223–236.

[10] M. S. Müller, M. van Waveren, R. Lieberman, B. Whitney, H. Saito, K. Kumaran, J. Baron, W. C. Brantley, C. Parrott, T. Elken, H. Feng, and C. Ponder, "SPEC MPI2007 - an application benchmark suite for parallel systems using mpi," *Concurrency and Computation: Practice and Experience*, vol. 22, no. 2, pp. 191–205, 2010.

[11] *Intel® Xeon® Processor E5 v3 Product Family - Processor Specification Update*, Intel, 1 2015. [Online]: http://www.intel.com/content/dam/www/public/us/en/documents/specification-updates/xeon-e5-v3-spec-update.pdf

[12] D. Hackenberg, R. Schöne, T. Ilsche, D. Molka, J. Schuchart, and R. Geyer, "An energy efficiency feature survey of the intel haswell processor," accepted for publication in the 11th Workshop on High-Performance, Power-Aware Computing (HPPAC'15), 2015.

[13] M. Yuffe, M. Mehalel, E. Knoll, J. Shor, T. Kurts, E. Altshuler, E. Fayneh, K. Luria, and M. Zelikson, "A fully integrated multi-cpu, processor graphics, and memory controller 32-nm processor," *Solid-State Circuits, IEEE Journal of*, vol. 47, no. 1, pp. 194–205, 1 2012.

[14] *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Intel, Sep 2014, order Number: 248966-030. [Online]: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-optimization-manual.pdf

[15] P. Hammarlund, A. Martinez, A. Bajwa, D. Hill, E. Hallnor, H. Jiang, M. Dixon, M. Derr, M. Hunsaker, R. Kumar, R. Osborne, R. Rajwar, R. Singhal, R. D'Sa, R. Chappell, S. Kaushik, S. Chennupaty, S. Jourdan, S. Gunther, T. Piazza, and T. Burton, "Haswell: The fourth-generation intel core processor," *Micro, IEEE*, vol. 34, no. 2, pp. 6–20, Mar 2014.

[16] *Intel Xeon Processor E5 v3 Family Uncore Performance Monitoring Reference Manual*, Intel, 9 2014. [Online]: http://www.intel.com/content/dam/www/public/us/en/zip/xeon-e5-v3-uncore-performance-monitoring.zip

[17] *Intel® Xeon® Processor E5-1600, E5-2400, and E5-2600 v3 Product Families - Volume 2 of 2, Registers*, Intel Corporation, Jan 2015.

[18] "bullx r421 e4 accelerated server," Bull SAS, data sheet, 2014. [Online]: http://www.bull.com/extreme-computing/download/S-bullxR421-en3.pdf