Memory System Performance in a NUMA Multicore Multiprocessor

Zoltan Majo
Department of Computer Science
ETH Zurich, Switzerland
zoltan.majo@inf.ethz.ch

Thomas R. Gross
Department of Computer Science
ETH Zurich, Switzerland
thomas.gross@inf.ethz.ch

ABSTRACT

Modern multicore processors with an on-chip memory controller form the base for NUMA (non-uniform memory architecture) multiprocessors. Each processor accesses part of the physical memory directly and has access to the other parts via the memory controller of other processors. These other processors are reached via the cross-processor interconnect. As a consequence a processor's memory controller must satisfy two kinds of requests: those that are generated by the local cores and those that arrive via the interconnect from other processors. On the other hand, a core (respectively the core's cache) can obtain data from multiple sources: data can be supplied by the local memory controller or by a remote memory controller on another processor. In this paper we experimentally analyze the behavior of the memory controllers of a commercial multicore processor, the Intel Xeon 5520 (Nehalem). We develop a simple model to characterize the sharing of local and remote memory bandwidth. The uneven treatment of local and remote accesses has implications for mapping applications onto such a NUMA multicore multiprocessor. Maximizing data locality does not always minimize execution time; it may be more advantageous to allocate data on a remote processor (and then to fetch these data via the cross-processor interconnect) than to store the data of all processes in local memory (and consequently overloading the on-chip memory controller).

Categories and Subject Descriptors

C.4 [**Performance of Systems**]: Measurement Techniques; C.4 [**Performance of Systems**]: Performance Attributes

General Terms

Experimentation, Measurement, Performance

Keywords

NUMA, multicore processors, memory system performance

1. INTRODUCTION

Current microprocessors are multicore systems, and the number of cores that are integrated onto a single processor is likely to further increase in the future. One of the challenges faced by multicore processors is to provide adequate memory access for the processor's cores. Caches can help to reduce the memory bandwidth requirements, but as the number of cores increases, processor designers must find a way to provide higher and higher memory bandwidth while avoiding further increases in memory access latency. To improve the performance of the memory interface and to supply all cores with data, newer processor designs integrate the memory controller on-chip with the processor. In comparison with previous designs that required an off-chip memory controller, this solution offers memory accesses with increased bandwidth and reduced latency.

The on-chip memory controller has another advantage as well. In a multicore multiprocessor (a multiprocessor built from several multicore processors), a local memory controller allows scaling a system as there is no single central memory controller. Instead, the physical address space is split between processors, and the cores of each processor can directly access only a part of the physical memory via the local memory interface. To support the familiar model of a shared-memory multiprocessor, each processor (and its cores) must be able to access not only the directly connected local memory, but the local memory of other processors as well. These remote memory accesses pass through a cross-chip interconnect that connects the processors. Major processor manufacturers have come up with their proprietary cross-processor interconnect technology (e.g., the Intel QuickPath Interconnect (QPI) [13], or the AMD Hypertransport [1]).

The throughput of the cross-chip interconnect, however, is lower than the throughput of the on-chip memory controllers. Remote memory accesses that pass through the cross-chip interconnect also encounter latencies larger than the latencies of local memory accesses. Because of the heterogeneity of their memory interfaces, such multiprocessors are classified as non-uniform memory architecture (NUMA) systems. The performance penalty of remote memory accesses is significant (we call this penalty *NUMA factor*); in current implementations the NUMA factor can be as high as 2 (equivalent to a 2X slowdown) for some applications.

The most important performance optimization for NUMA systems considered until now is to increase the *data locality* of the system (i.e., to allocate memory close to the computations accessing it, so that the number of remote accesses is reduced and thus the NUMA-specific performance penalty is avoided). Various researchers investigated techniques to improve application performance on NUMA systems. All approaches focus on increasing the data locality in the system using either profile-based [12, 18, 16] or on-line [21, 24, 27, 28, 25] optimizations that target the allocation of memory and the mapping of computations in the system. As the performance of many applications is ultimately limited by the performance of the memory system, it is important to understand the memory system of such NUMA multicore multiprocessors as simple and realistic models are crucial to find mappings (of data and computations) that result in good performance on these systems.

Previous research has focused on evaluating the bandwidth and latency of the on-chip memory controller and of the cross-chip interconnect of modern NUMA machines *in separation* (i.e., when there are either local or remote memory accesses in the system, but not both of them at the same time) [19, 8, 15]. However, it rarely happens in real systems that a computation's memory traffic exclusively flows through either the local memory interface or the cross-chip interconnect that connects to the memory controller of a remote processor. So it is also important to understand how these two types of memory accesses (local and remote) interact.

A recent study [2] evaluates the problems and opportunities posed by having multiple types of memory controllers in a system. The authors demonstrate that reducing data locality in the system can improve performance because excessive contention on the memory interfaces of the system is avoided. However, their study is more concerned about future architectures and less with existing ones. In this paper we analyze the bandwidth sharing properties of a commercial microprocessor and discuss the implications of these properties for optimizing programs in multicore systems. We show that in some cases when the machine is highly loaded the cross-chip interconnect outperforms the on-chip memory controller. Mapping computations so that all memory traffic flows through the local memory interface is bound to be suboptimal in many situations. Given the design times of modern processors, we expect that our evaluation techniques and observations are of interest for a while as processors based on this microarchitecture are released.

2. EXPERIMENTAL SETUP

In this section we describe the architecture of the evaluation system, the benchmark programs, and the experimental methodology we use.

2.1 Hardware

We investigate the memory system of a multicore multiprocessor machine based on the Intel Nehalem microarchitecture. The evaluation machine is equipped with two Intel Xeon E5520 quad-core CPUs and a total of 12 GB RAM (see Figure 1). Each processor has a direct connection to half of the memory space via a three-channel integrated memory controller. The on-chip integrated memory controller (IMC) provides a maximum throughput of 25.6 GB/s. Additionally, each processor has two QuickPath Interconnect (QPI) interfaces [13], one connecting to the remote processor and one to the I/O hub. The interconnect has a maximum throughput of 11.72 GB/s in one direction and 23.44 GB/s in both directions. Although the throughput of the IMC and QPI is similar, there are two IMCs in the system, while there is only one QPI link connecting the two processors. If there is good data locality in the system, the throughput of the two IMCS (2x25.6 GB/s) can be fully exploited. Otherwise the performance of the application is limited by the throughput of the single cross-chip interconnect of the system. Moreover, the latencies of local and remote memory accesses differ significantly as well.

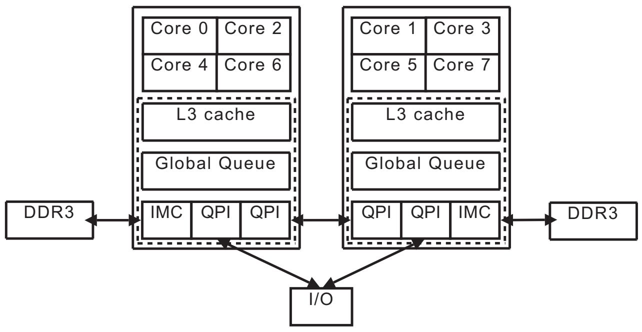


Figure 1: Intel Nehalem in 2-processor configuration.

The performance benefit of per-processor memory controllers is substantial. Figure 2 shows the maximal throughput of a memory-intensive microbenchmark executed on two consecutive generations of Intel Xeon processors: the Xeon 5345 with an off-chip memory controller and the Xeon 5520, which includes on-chip memory controllers. Both systems are equipped with two processors. The memory controller of the Xeon 5345 is already saturated by two active cores, but the throughput of the newer Xeon 5520 scales up to eight active cores.

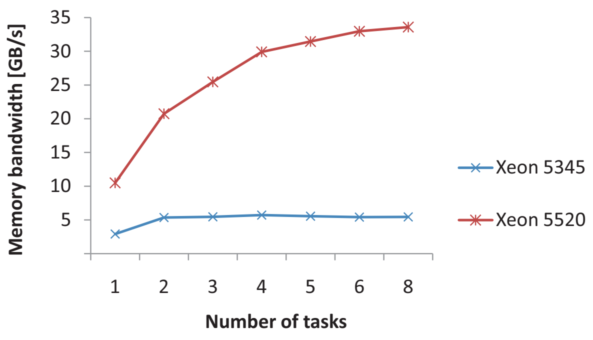


Figure 2: Memory system throughput of two consecutive generations of Intel Xeon processors.

Each core of a Nehalem processor has its own level 1 and level 2 exclusive cache, but the per-processor inclusive 8 MB last-level cache (LLC) is shared between all cores of the same processor. In this paper we refer to the subsystem incorporating the LLC, the arbitration mechanisms, and the memory controllers as the *uncore* (marked with dotted lines on Figure 1).

When a processor accesses a memory location, there are many different locations that can hold the data (e.g., local or remote caches, local or remote RAM). Similarly, there can be several outstanding memory requests, from multiple processors, in flight at any point of time, so a routing and arbitration mechanism for these requests is necessary. On the Nehalem, a part of the uncore called the Global Queue (GQ) arbitrates these requests [9]. The GQ controls and buffers data requests coming from different subsystems of the processor. For each subsystem (processor cores, L3 cache, IMC, and QPI) there is a separate port at the GQ, as shown in Figure 3. Requests to local and remote memory are tracked separately. As many different types of accesses go through the GQ, the fairness of the GQ is crucial to assure that each subsystem experiences the same service quality in terms of the share of the total system bandwidth.

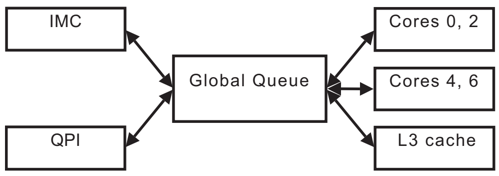


Figure 3: Global Queue.

Intel Nehalem processors feature a dynamic overclocking mechanism called Turbo Boost that allows raising the clock rate of processor cores over their nominal rate if the per-processor thermal and power limits still remain within the processor's design specifications [5]. Turbo Boost results in a performance improvement of up to 6.6% in both single- and multithreaded configurations of the benchmarks we use, but we disable it (together with dynamic frequency scaling) to improve the stability of our measurements and to allow a focus on the memory system interface. The hardware and adjacent cache line prefetchers are enabled for all our experiments.

Although our 8-core 2-processor evaluation system is small, it nevertheless allows interesting experiments, as it already offers the opportunity to study the interaction between local and remote memory accesses. It is possible to build larger systems (up to eight sockets) based on the Intel Nehalem microarchitecture. These systems use a processor with a larger number of QPIs to allow point-to-point connections between all processors. We used such a system with 4 processors/32 cores, but the uncore of these systems is more complicated [10], so a presentation of the possible interactions between the uncore components exceeds the scope of this paper. Nevertheless, to assess the performance implications of having more than four cores per processor in Section 4 we briefly evaluate the memory system performance of the 6-core die shrink of the Nehalem, the Westmere.

2.2 Benchmarks

We use the `triad` workload of the STREAM benchmark suite [17] to evaluate the sustainable memory bandwidth of individual cores, processors, and the complete system. The `triad` workload is a program with a single execution phase with high memory demands. It operates on three arrays (`a[]`, `b[]` and `c[]`) that must be sized so that they are larger than the last-level cache to cause memory controller traffic (see Figure 4).

A single instance of the `triad` workload is not capable of saturating any of the memory interfaces of our evaluation machine, thus it does not allow us to explore the limits of the machine's main memory subsystem. Besides, a single `triad` instance does not allow for evaluating the in-

```
for (i = 0; i < ARRAY_SIZE; i++)
{
   a[i] = b[i] + SCALAR * c[i];
}
```

Figure 4: `triad` program fragment.

teraction between the different types of memory controllers, because we need at least one `triad` instance for each type of memory controller to have two types of memory accesses in the system at the same time. Hence, we construct multiprogrammed workloads that consist of co-executing instances of `triad` (also referred to as `triad` clones). In this paper we refer to these workloads as xP, where x is the number of `triad` clones the workload is composed of (e.g., 3P represents a workload composed of three `triad` clones).

To keep our analysis simple, the version of the `triad` benchmark we use in this paper is multiprogrammed and uses processes (contrary to the original implementation of the STREAM benchmark suite that is implemented with OpenMP and uses threads). As a result, there is no data shared between co-executing `triad` clones. The Nehalem microarchitecture implements the MESIF cache coherency protocol, and accesses to cache lines in each different state (i.e., Modified, Exclusive, Shared, Invalid, or Forwarding) involve different access latencies [19]. By using a multiprogrammed benchmark, we restrict the types of cache lines accessed to M, E, and I, and therefore we need not account for the different latencies of accesses to cache lines in all possible states. As a result, our measurement data are easier to interpret and to understand. Nevertheless, our analysis can be easily extended to evaluate the bandwidth sharing properties of accesses to cache lines in other coherency states as well, following the methodology described by [19].

There are two useful properties of `triad` that make it well suited for the main memory system evaluation. First, `triad`'s cache miss rate per instructions executed stays constant when executed in multiprogrammed configurations, because co-executing `triad` clones that share a LLC do not cause additional inter-core misses [23] to each other. With other words, `triad` is a *cache gobbler* type of program (according to the classification proposed by Sandberg et al. in [22]. The second useful property of triad is that 94–99% of its read memory accesses are served by main memory (for details about write accesses see Section 3). As a result, in multi-programmed configurations `triad` slows down only because of bandwidth saturation, increased memory access latencies, and contention on the memory controllers of the system, but not due to contention on the shared caches of the system. Hence `triad` is very well suited for evaluating the raw performance of the main memory system, and is not influenced by caching effects at all.

We use standard Linux system calls [11] to control on which processor the memory is allocated and where the processes executing the triad clones are scheduled. We use the terms `triad` process and `triad` clone interchangeably in this paper, as there is a 1:1 mapping between a clone (an instance of the `triad` program) and the process executing it. To evaluate the interaction between the IMC and QPI, the memory used by `triad` processes is always allocated on a single processor, Processor 0, but the process-to-core mapping changes. The workloads can execute in multiple configurations, depending on the number of `triad` processes mapped onto the same processor. The terms *local* and *remote* are always relative to the processor that holds the data in memory. We denote with xL and xR the number x of local and remote processes, respectively. For example, a three-process (3P) workload executing in the (2L, 1R) configuration means that two cores access memory locally and one core accesses memory remotely, as shown in Figure 5. In this paper the instances of the workload executing locally are referred to as *L processes*, while instances executing remotely are called *R processes*. Memory accesses of L processes must pass just through the Global Queue and the IMC, while R processes have the additional overhead of passing through the processor cross-chip interconnect (QPI) and the GQ of the remote processor. The datapaths used by the (2L, 1R) workload are also illustrated in Figure 5.

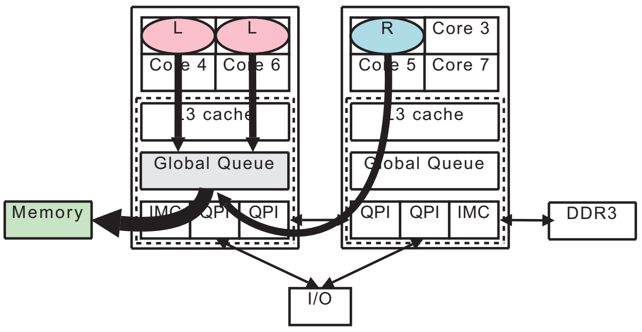


Figure 5: 3P workload in (2L, 1R) configuration.

2.3 Measurements and methodology

The evaluation machine runs Ubuntu Linux 2.6.30 patched with perfmon2 [6]. We use the processor's performance monitoring unit to obtain information about the elapsed CPU cycles and the amount of last-level cache (LLC) misses a program generates. We calculate the generated memory bandwidth using Equation 1.

$$bandwidth = \frac{64 \cdot LLC\ misses \cdot 2.27 \cdot 10^9}{CPU\ cycles \cdot 10^6} MB/s \qquad (1)$$

The cache line size of the LLC of the Intel Nehalem is 64 bytes. The processors in our system were clocked at 2.27 GHz. Due to the limitations of the performance monitoring unit, we can measure only the read bandwidth of the cores. We also use the uncore monitoring facility of the Nehalem to monitor the state of the GQ and to cross-check the readings obtained with the per-core performance counters.

As the `triad` workload is very memory bound and has a single program phase, its bandwidth and performance readings are very stable and do not depend on the factors reported by Mytkowicz et al. [20] (e.g., the size of the UNIX environment and link-order). All measurement data are the average of three measurement runs; the variation of the performance counter readings is negligible.

3. MEMORY SYSTEM PERFORMANCE

To measure the bandwidth sharing properties of the Nehalem microarchitecture we measure the bandwidth achieved by each instance of the `triad` benchmark. We configure the benchmark with a number of processes ranging from one to eight (the number of cores on our machine), and then measure all possible local-remote mapping configurations for any given number of `triad` processes. Recall that only per-process read bandwidth can be reported due to limitations of the hardware performance counter subsystem. However, the total amount of read and write bandwidth on the interfaces of the system can be measured using uncore performance counters. These measurements show that the `triad` benchmark is read-intensive, as in any configuration 75% of the total main memory bandwidth is caused by reads, and 25% is due to writes. Therefore, the measurements of the read bandwidth of the system are representative for the behavior of the memory interfaces of the Nehalem-based system.

Figure 6 shows a scenario where four local processes share the IMC bandwidth with different numbers of remote processes. If there are no remote processes, the complete bandwidth is allocated to local requests (4L, 0R). As a single remote process is added (resulting in configuration (4L, 1R)), the total bandwidth increases. Then, as the number of remote processes further increases, the total bandwidth is reduced slightly (configurations (4L, 2R) to (4L, 4R)). Two remote processes consume the maximal bandwidth that can be obtained through remote accesses; the share of the remote processes does not grow as we increase the number of remote processes. As a consequence, each remote process realizes a smaller and smaller absolute memory bandwidth.

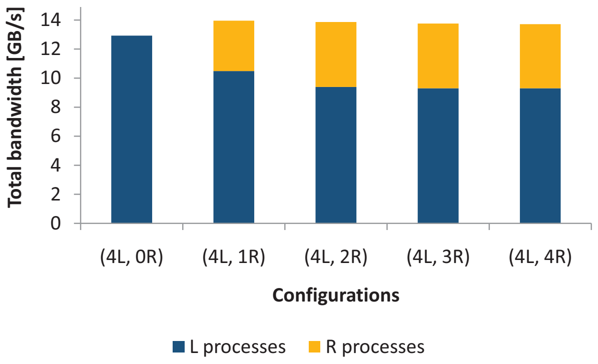


Figure 6: Bandwidth sharing (4L with variable number of R processes).

The data in Figure 6 might convince a system developer to favor mapping a process onto the processor that holds the data locally. However, the situation is more complex: Figure 6 shows the total bandwidth achieved by all processes. Figure 7 contrasts Figure 6 by showing the performance of individual R and L processes. If there is a single L and a single R process, the L process captures almost 50% more of the memory bandwidth (L: 6776 MB/s, R: 4412 MB/s). As the number of L processes increases, these L processes compete for local access, and although the R process's declines as well (to 3472 MB/s), the bandwidth obtained by each L process declines a lot more (to 2622 MB/s).

Table 1 shows the complete measurement data. Each row reports the bandwidth obtained by a each L instance in the presence of varying number of R processes. Table 2 shows the bandwidth obtained by the R processes in the same configurations as Table 1. Figure 7 contrasts column 2 of Tables 1 and 2. Tables 3 and 4 report the cumulative data (for all

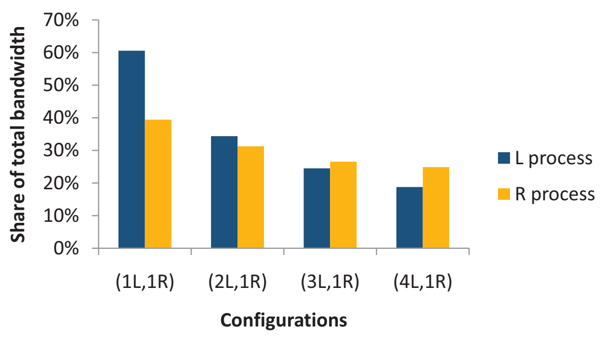


Figure 7: Percentage of L and R of the total bandwidth measured in the system.

the L respectively R processes in an experiment). Figure 6 is based on the last row of Tables 3 and 4; the sum of these rows yields the total shown in the figure.

3.1 A simple model

Our experiments show that if there are only local processes running on the system, the total bandwidth obtained by these processes can be described as:

$$bw_{L_{total}} = min(active\_cores * bw_L, bw_{L_{max}}) \qquad (2)$$

In Equation 2 bw_L is the bandwidth of a single, locally executing `triad` clone (L process) (see column "0 R" of Table 1). If the sum of the bandwidth of the individual cores $bw_{L_{total}}$ is greater than the threshold $bw_{L_{max}}$ (see column "0 R" of Table 3 for the exact values), each core obtains an equal share of the threshold value (12925 MB/s).

Similarly, the total bandwidth obtained by remote processes can be characterized as:

$$bw_{R_{total}} = min(active\_cores * bw_R, bw_{R_{max}}) \qquad (3)$$

In Equation 3 bw_R is the bandwidth achieved by a single `triad` instance executing remotely (R process) (see column "0 L" of Table 2). The maximum throughput of the R processes ($bw_{R_{max}}$) is limited by the QPI interface and is 6998 MB/s (experimentally determined). The QPI is also fair in the sense that if the threshold is to be exceeded, each R processes obtains an equal share of the total bandwidth.

The total bandwidth obtained by the system is composed of the bandwidth achieved by L and R processes and is shown in Table 5 for all configurations of the `triad` benchmark. The limit $bw_{L_{max}}$ of L processes can be observed in row "4 L" column "0 R" of Table 5. Similarly, the limit $bw_{R_{max}}$ of R processes can be observed in row "0 L" and column "2 R" of Table 5. Remote processes hit their limit $bw_{R_{max}}$ with two active cores, while four local processes are needed to hit the limit $bw_{L_{max}}$. This is because the QPI is already saturated by two `triad` clones, however all four cores need to be active to saturate the IMC. Next generations of the Nehalem have a larger number of cores connected to the same local memory controller, therefore not all cores of a processor are required to achieve the saturation limit of the IMC. In Section 4 we briefly look at a such machine.

Formally the total bandwidth in the system can be expressed as:

	0 R	1 R	2 R	3 R	4 R
0 L	0	4844	6998	6938	6807
1 L	7656	11188	12345	12205	12155
2 L	11024	12977	13708	13618	13517
3 L	12607	13842	14001	13882	13844
4 L	12925	13959	13865	13758	13719

Table 5: Total cumulative bandwidth [MB/s].

$$bw_{total} = (1 - \beta) * bw_{L_{total}} + \beta * bw_{R_{total}} \qquad (4)$$

We call the variable β the *sharing factor*. The sharing factor determines the share of the total bandwidth received by local and remote `triad` clones. β is a real value between 0 and 1. If β is 1, all bandwidth is obtained by R processes. Similarly, if β is 0, all bandwidth is obtained by L processes. As the Global Queue (GQ) arbitrates between local and remote memory accesses, the GQ determines the value of β based on the arrival rate of requests at its ports.

If the system must handle memory requests coming from a small number of cores, the bandwidth (and thus the performance) of local processes is much better than the bandwidth of remote ones. As the load on the system increases and there are more local processes, the bandwidth obtained by individual local processes (bw_L) becomes comparable to the cumulative bandwidth of the QPI ($bw_{R_{total}}$). Situations when the bandwidth of the QPI is better than the bandwidth of individual local processes are also possible (e.g., configuration (4L, 1R) and (3L, 1R)). Overloading the QPI with a large number of remotely executing memory-bound processes should be avoided, as the lower throughput of the QPI interface is divided between R processes, resulting in low performance of R processes, if their number is too large. In conclusion, if the system has a low utilization, local execution is preferred. Nevertheless, as the load on the memory system increases, remote execution becomes more favorable, but care needs to be taken not to overload the cross-chip interconnect.

To fully understand the system, the dependence of the sharing factor β of the GQ on the load coming from the local cores and remote memory interfaces needs to be characterized. However, as most implementation details of the Nehalem queuing system are not disclosed, and the performance monitoring subsystem of our Nehalem-based processor does not allow for measuring queue status directly, such a model is difficult to construct. Instead, in Sections 3.2 and 3.3 we describe two empirically observed properties of the GQ that help understanding the bandwidth sharing properties of our evaluation system: queuing fairness and aggregate throughput.

3.2 Queuing fairness

Table 4 shows that for any number of local processes there is a significant difference between the throughput of the non-saturated QPI executing a single R process (the "1 R" column), and the throughput of the QPI transferring the data for two R processes (the column labeled "2 R"). Adding more R processes (columns "3 R" and "4 R") does not modify the overall bandwidth allocation of the system, as the throughput limit of the QPI has already been reached, and the QPI is saturated. However, a large difference in the total bandwidth obtained by the L and R processes is observed by

	0 R	1 R	2 R	3 R	4 R
0 L	0	0	0	0	0
1 L	7656	6776	6325	6185	6210
2 L	5512	4460	4189	4142	4121
3 L	4202	3389	3078	3047	3048
4 L	3231	2622	2348	2325	2326

Table 1: Per-core L bandwidth [MB/s].

	0 R	1 R	2 R	3 R	4 R
0 L	0	4844	3499	2313	1702
1 L	0	4412	3010	2007	1486
2 L	0	4056	2664	1778	1319
3 L	0	3675	2383	1581	1175
4 L	0	3472	2236	1486	1104

Table 2: Per-core R bandwidth [MB/s].

	0 R	1 R	2 R	3 R	4 R
0 L	0	0	0	0	0
1 L	7656	6776	6325	6185	6210
2 L	11024	8921	8379	8283	8242
3 L	12607	10167	9235	9141	9145
4 L	12925	10487	9393	9299	9302

Table 3: Total L bandwidth [MB/s].

	0 R	1 R	2 R	3 R	4 R
0 L	0	4844	6998	6938	6807
1 L	0	4412	6020	6020	5945
2 L	0	4056	5329	5335	5275
3 L	0	3675	4765	4742	4699
4 L	0	3472	4472	4459	4416

Table 4: Total R bandwidth [MB/s].

varying the number of L processes (rows "1 L" to "4 L").

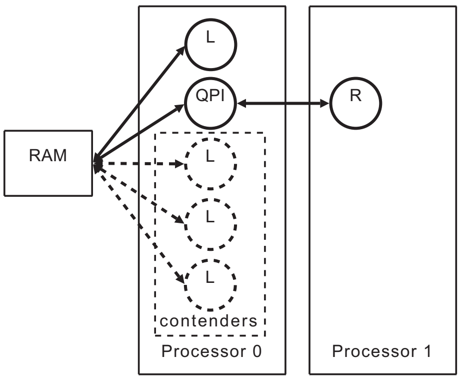
In the following, we consider the QPI as a fifth *agent* connected to the GQ (in addition to the four local cores), executing either the 1R workload, or a workload equivalent to the memory intensity generated by the 2R workload. We take as baseline the performance of two cases. In the first case the GQ is serving 1R from the QPI and 1L from the local cores, as depicted by Figure 8a. In the second case the GQ is serving the 2R in combination with 1L, as depicted by Figure 8b. Using the previously defined notation, these workloads can be denoted with (1L, 1R) and (1L, 2R). To increase the contention on the GQ, one, two, or three additional L process(es) are executed on the system. These L processes (the base L process plus the additional L processes) contend with the QPI for IMC bandwidth.

Figure 9 shows the variation of the sharing factor (parameter β of Equation 4) when contention on the local port of the GQ increases. The sharing factor depends on the load on the GQ: the more traffic L processes generate, the larger a share of the bandwidth they obtain, and the more the share of the R processes (given by β) decreases. Nonetheless, if we consider the performance degradation of the two baseline workloads (1L, 1R) and (1L, 2R) (shown in Figure 10a and in Figure 10b respectively), the performance of individual L process in each of the two workloads degrades more than the performance of the QPI does. Therefore, the more load there is on the GQ, the more attractive is to execute some processes remotely.

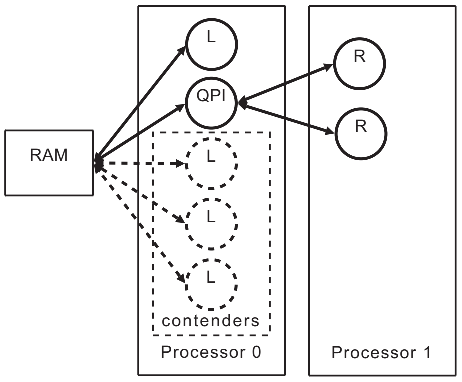
In conclusion, if the GQ is contended, the Nehalem microarchitecture is unfair towards local cores (vs. the QPI), as the performance degradation local cores experience is larger than that of the QPI. Still, this behavior is reasonable as the GQ does not allow remote cores to starve, and thus it avoids further aggravating the penalty of remote memory accesses. Nevertheless, this property of the Nehalem is undocumented and can be discovered only with experimental evaluation.

3.3 Aggregate throughput

To further motivate the benefit of having a good proportion of local and remote memory accesses in the system in Figure 11 we show the total system throughput for the 4P workload in different mapping configurations (ranging from the configuration when all processes execute locally to the configuration with all processes executing remotely). In



(a) (1L, 1R)



(b) (1L, 2R)

Figure 8: Setup to evaluate GQ fairness.

the configurations with some remote memory accesses the throughput of the memory system can be better (at a peak of 13842 MB/s) relative to the configuration when all memory accesses are local (12925 MB/s).

To take a closer look at the total system throughput, we examine two cases. First, we map the processes of the `triad` workload onto local cores. This way, all memory operations use the local ports of the Global Queue. Then, we move one process to the remote processor, thus the QPI port of the

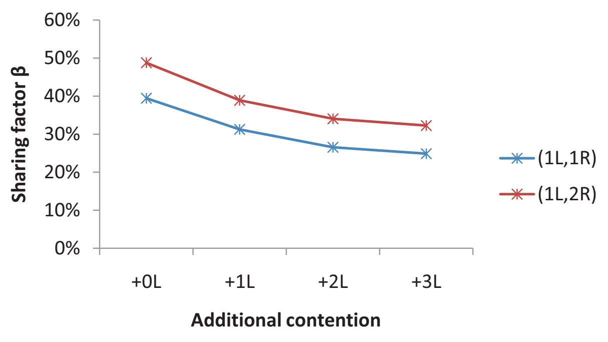
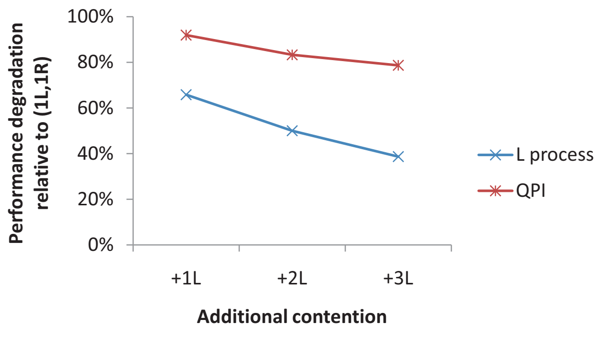
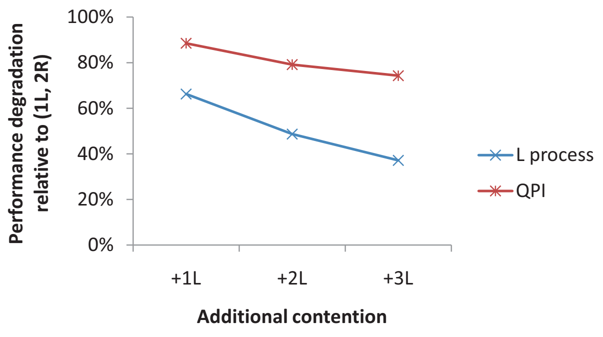


Figure 9: Dependence of β on aggregate load.



(a) Performance degradation of (1L,1R).



(b) Performance degradation of (1L,2R).

Figure 10: Performance degradation of the workloads (1L,1R) and (1L, 2R).

GQ is also used to actively handle memory requests. For both cases, we compute overall system throughput as the sum of the instructions per cycle (IPC) values obtained by the processes:

$$IPC_{total} = \sum_{p \in Processes} IPC_p \tag{5}$$

We are aware that in case of heterogeneous workloads (workloads that execute different instruction streams) using the metric defined by Equation 5 is not appropriate, as pointed out by Eyerman in [7]. However, in our case all processes execute the same tight memory-intensive loop (shown in Figure 4) operating on identical data, therefore the instructions executed by each workload are the same. The clock rate of all processor cores is also equal, so the ratio of instructions executed and cycles consumed is a precise measure for system throughput. As `triad` is very memory intensive, the aggregate memory bandwidth achieved on the system is also directly proportional to the system throughput. This metric does not characterize the fairness of the system, but it accurately reflects the throughput of the main memory system.

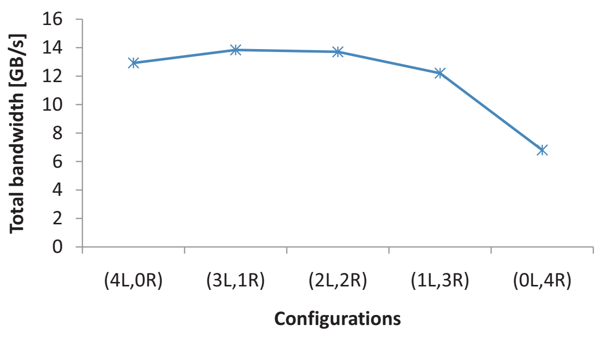


Figure 11: Total bandwidth of 4P in different configurations.

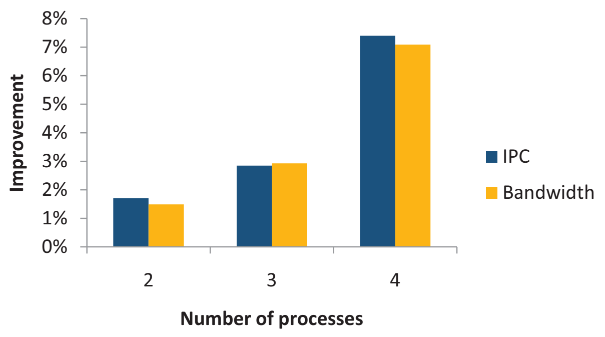


Figure 12: Improvement of aggregate IPC and total memory bandwidth when both memory controllers (IMC and QPI) are used relative to the single IMC case.

Figure 12 shows the benefit of mapping one process remotely over the all-local case (where all memory requests come from local cores). The benefit is minor (1.7%) if there are just two processes running on the system, but it gets significant (7.4 %) if there are four processes. This increase of performance in the four-process case can be explained by the distribution of contention on the GQ. When the GQ handles four locally executing `triad` clones, its local port is saturated (it is full 10% of the time). Moving one process to the remote processor transfers some of load from the local port of the GQ to its remote port. In this new configuration neither the local-, nor the remote port of the GQ is saturated, therefore system throughput increases. However, if all processes execute remotely, the remote port of the GQ gets saturated (it is full 31% of the time).

In conclusion, in a single-threaded context the bandwidth and latency of the on-chip memory interface greatly outperforms the same parameters of the QPI. However, in the case when multiple cores are competing this advantage diminishes as contention on the queuing system increases. Distributing computations such that there are both local and remote accesses in the system helps to improve aggregate throughput.

3.4 Discussion

In our analysis we did not account for the overhead of the cache coherency protocol. On every cache miss, there is a snoop request towards the cache of the adjacent processor (as measured on the read-, write-, and peer–probe-tracker of each processor's uncore). Snoop requests are transferred on the cross-chip interconnect of the system. However, while normal reads usually requests data of the size equal to a cache line, we do not know the amount of data transferred with a snoop requests, therefore we cannot calculate the amount of traffic generated by these requests, and we cannot calculate the bandwidth overhead of the cache coherency protocol.

In this work we use a single, homogeneous workload (composed of multiple `triad` clones) to evaluate the main memory system performance of a NUMA-multicore machine. Because the `triad` benchmark does not benefit of caching, its performance describes the main memory system of our evaluation system well. In case multiple, heterogeneous workloads are executing on the system (with programs that are less memory bound than `triad`), caching effects also come into play. For an analysis of these scenarios, and the applicability of the principles described in the previous section to OS scheduling, see [14].

4. THE NEXT GENERATION

In 2010 Intel has released a die shrink of the Nehalem codenamed Westmere. This new processor contains six cores instead of four (as it is in the case of the Nehalem). To see if the principles described in this paper also apply for the Westmere microarchitecture, we perform the experiments described in Section 3 with a machine based on this microarchitecture as well. The Westmere-based system (Intel Xeon X5680) we evaluate has also two processors, however it shows some differences to the Nehalem-based machine described in detail in this paper. A comparison of the two microarchitectures is shown in Table 6. Note that although the QPI bandwidth of the Westmere is higher than its IMC bandwidth, the latency of memory accesses that have to pass through the QPI is much higher than that of IMC accesses.

	Xeon E5520	Xeon X5680
Cores per processor:	4	6
L3 cache size:	8 MB	12 MB
IMC bandwidth:	25.6 GB/s	19.2 GB/s
QPI bandwidth:	23.44 GB/s	25.6 GB/s

Table 6: Parameters of the evaluation machine.

We do the same set of experiments with the Westmere as with the Nehalem, but because of space limitations we do not present the complete data. Instead, we present two projections of the Westmere data, similar to the ones presented for the Nehalem in Figure 6 and Figure 7, respectively. In Figure 13 four local processes share the bandwidth of the IMC of Processor 0. As one R process is added, the total achieved bandwidth increases. Adding more R processes increases the share of R processes until the saturation limit of the QPI is achieved (in the case of Westmere four R processes are required to saturate the QPI versus two R processes in the case of the Nehalem).

Figure 13 shows a breakdown of the total bandwidth of the two types of processes, L and R. Figure 14 shows the

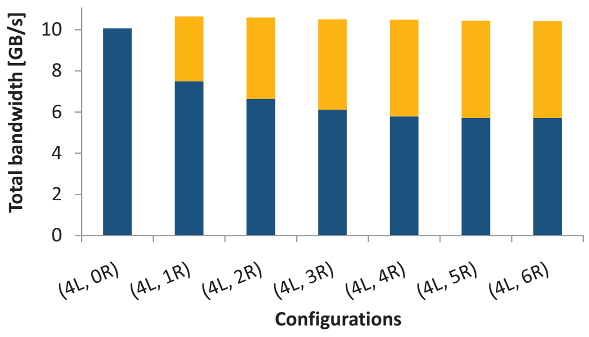


Figure 13: Bandwidth sharing on the Westmere (4L with variable number of R processes).

memory bandwidth of a single L resp. R process with increasing number of L processes. On the Westmere a single R processes is able to achieve more bandwidth than an L process already in the configuration with two locally executing processes.

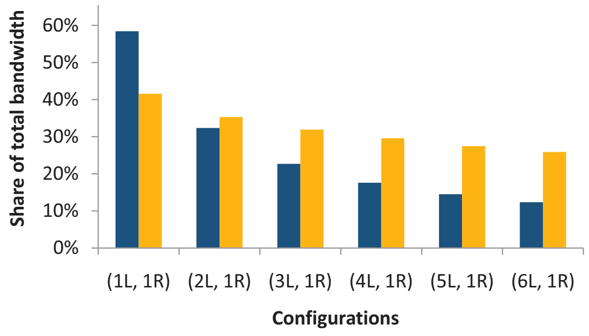


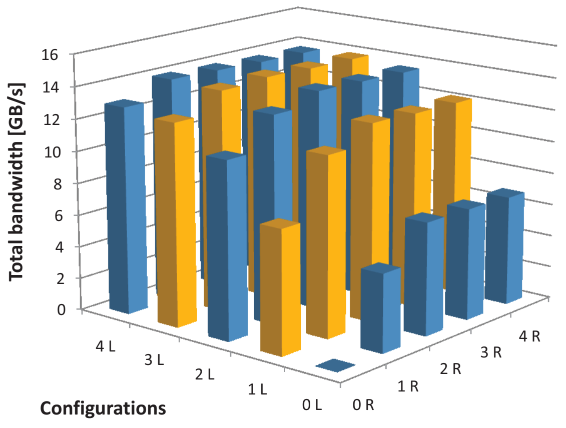
Figure 14: Percentage of L and R of total bandwidth measured in the Westmere-based system.

To compare the Nehalem to the Westmere, Figure 15a and 15b show the total read bandwidth measured on the Nehalem and the Westmere, respectively. As the Westmere includes two cores more than the Nehalem, the thresholds $bw_{L_{max}}$ and $bw_{R_{max}}$ defined in Equations 2 and 3 are more prominent than on the Nehalem: on the Westmere four `triad` clones are required to saturate the QPI (vs. two on the Nehalem), while the IMC saturates with four `triad` clones, just as on the Nehalem.

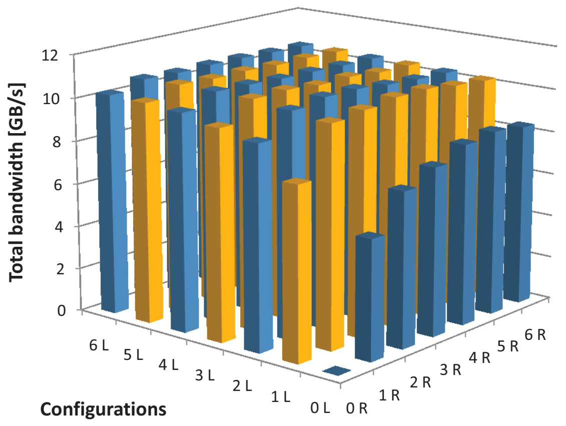
In conclusion, the principles we describe for the Nehalem also apply for the Westmere microarchitecture. Because the Westmere has more cores, a different LLC size, and memory interfaces with slightly different throughput as the Nehalem, the bandwidth sharing properties of this machine are quantitatively, but not qualitatively, different.

5. RELATED WORK

Molka et al. [19] analyze in detail the memory system performance of the Intel Nehalem. In later work, Hackenberg et al. [8] compare the performance of the Intel Nehalem with the AMD Shanghai. Their methodology measures memory bandwidth and memory access latency and also analyzes the impact of cache coherency. However, they consider the properties of the different interconnects of multicore chips



(a) Nehalem.



(b) Westmere.

Figure 15: Total bandwidth on Nehalem and Westmere.

only in isolation, and not the interaction between them. Our measurements considered only cache lines in the E, M and I states, but the measurement framework can be extended to measure the sharing of bandwidth to cache lines in other coherency states as well. Yang et al. [29] examine the dependence of application performance on memory and thread placement in an AMD Opteron-based NUMA machine. They quantify execution time, but do not measure low-level hardware issues (e.g., cache coherency traffic that is significant in some execution configuration of their benchmarks).

Mandal et al. [15] model the memory bandwidth and memory access latency of commercially available systems (among those also the Nehalem) as a function of concurrent memory references in the system. However, it is difficult to extend their model to include sharing between multiple types of memory controllers, because requests can be produced at different rates through the on-chip memory controller and the QPI. Their pointer-chasing benchmark also encounters inter-core misses, so the values reported are slightly dependent on the cache sharing behavior of the evaluated systems.

Tuduce et al. [26] describe the asymmetries of the memory system of a multicore multiprocessor with a shared off-chip memory controller. The authors argue that the hardware performance measurement unit of modern CPUs should be improved to allow system software detect and avoid the performance bottlenecks of the underlying architecture. In this paper we analyze a different architecture with multiple types of memory controllers and find that asymmetries are present in this system as well, however they are of a different nature. In the system analyzed in this paper the fairness of the arbitration mechanism (the Global Queue) is crucial for performance, therefore better monitoring of this subsystem should be made possible in successors of the Nehalem microarchitecture.

To best of our knowledge, Awasthi et al. [2] are the first to consider the problem of data placement in a system with multiple memory controllers. They identify the performance degradation caused by overloading a single memory controller in the system, and attribute the costs to increased queuing delays and decreased DRAM row-buffer hit rates. However, their evaluation focuses more on future architectures and less on present and near-future systems. Blagodurov et al. state in [4] that software management of contention for shared resources must be extended to NUMA systems as well. Because of its shortness, their paper lacks details about the implementation and the evaluation of their proposed approach. Blagodurov et al. [3] conduct a detailed evaluation of shared resource contention in multicore system. Among other factors, they quantify the contribution of contention on memory controllers to application performance degradation. Although they present NUMA-related performance issues, they do not account for issues related to the fairness of the queuing system of their evaluation machine.

6. CONCLUSIONS

Today's processors are multicores that integrate a memory controller with the cores and caches on a single chip. Such a design leads to a new generation of NUMA multicore multiprocessors that present software developers with a new set of challenges and create a different class of performance optimization problems. The cores put pressure on the memory controller to service the local memory access requests while at the same time the memory controller must deal with requests by other processors. So it is important that the software finds a balance between local and remote memory accesses if overall performance is to be optimized.

This paper presents an evaluation of the bandwidth sharing properties of a commercially available multicore system, the Intel Nehalem (Xeon 5520), and shows that if a large part or all of the cores of a processor are active, then favoring data locality may not lead to optimal performance. In addition to data locality, the bandwidth limits of the memory controllers and the fairness of the arbitration between local and remote accesses are also important. The overhead of arbitration and queuing is likely to become more important in larger systems as the complexity of this mechanism increases with a growing number of processors in the system, so it is important to develop realistic models of the memory system that can guide operating system and compiler developers.

Multicore multiprocessors will be an increasingly important class of parallel systems, as they provide (some) memory system scaling while using widely applicable off-the-shelf building blocks. Operating system and compiler (runtime systems) developers need to understand how to balance the memory system demands on such a system. The simple approach that aims only at maximizing data locality results in sub-optimal performance in important scenarios (applications limited by memory bandwidth). Instead, the software developers need to understand the memory system so that the best tradeoff between local and remote access can be found.

7. REFERENCES

[1] Advanced Micro Devices. AMD HyperTransport Technology-based system architecture. 2002.

[2] M. Awasthi, D. W. Nellans, K. Sudan, R. Balasubramonian, and A. Davis. Handling the problems and opportunities posed by multiple on-chip memory controllers. In *Proceedings of PACT'10*.

[3] S. Blagodurov, S. Zhuravlev, and A. Fedorova. Contention-aware scheduling on multicore systems. *ACM Transactions on Computer Systems*, 28:8:1–8:45, December 2010.

[4] S. Blagodurov, S. Zhuravlev, A. Fedorova, and A. Kamali. A case for NUMA-aware contention management on multicore systems. In *Proceedings of PACT'10*.

[5] J. Charles, P. Jassi, N. S. Ananth, A. Sadat, and A. Fedorova. Evaluation of the Intel Core i7 Turbo Boost feature. In *Proceedings of IISWC '09*.

[6] S. Eranian. What can performance counters do for memory subsystem analysis? In *Proceedings of MSPC'08*.

[7] S. Eyerman and L. Eeckhout. System-level performance metrics for multiprogram workloads. *IEEE Micro*, 28(3):42–53, 2008.

[8] D. Hackenberg, D. Molka, and W. E. Nagel. Comparing cache architectures and coherency protocols on x86-64 multicore SMP systems. In *Proceedings of MICRO-42*, 2009.

[9] Intel Corporation. *Intel 64 and IA-32 Architectures Optimization Reference Manual*, November 2009.

[10] Intel Corporation. *Intel Xeon Processor 7500 Series Uncore Programming Guide*, March 2010.

[11] C. Lameter. Local and remote remory: Memory in a Linux/NUMA system, 2006.

[12] H. Li, S. Tandri, M. Stumm, and K. C. Sevcik. Locality and loop scheduling on NUMA multiprocessors. In *Proceedings of ICPP'93*.

[13] R. A. Maddox, G. Singh, and R. J. Safranek. A first look at the Intel QuickPath Interconnect. 2009.

[14] Z. Majo and T. R. Gross. Memory management in NUMA multicore systems: Trapped between cache contention and interconnect overhead. In *Proceedings of ISMM'11 – to appear*.

[15] A. Mandal, R. Fowler, and A. Porterfield. Modeling memory concurrency for multi-socket multi-core systems. In *Proceedings of ISPASS'10*.

[16] J. Marathe and F. Mueller. Hardware profile-guided automatic page placement for ccNUMA systems. In *Proceedings of PPoPP'06*.

[17] J. 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, December 1995.

[18] C. McCurdy and J. S. Vetter. Memphis: Finding and fixing NUMA-related performance problems on multi-core platforms. In *Proceedings of ISPASS'10*, 2010.

[19] D. Molka, D. Hackenberg, R. Schöne, and M. S. Müller. Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system. In *Proceedings of PACT'09*.

[20] T. Mytkowicz, A. Diwan, M. Hauswirth, and P. F. Sweeney. Producing wrong data without doing anything obviously wrong! In *Proceedings of ASPLOS '09*.

[21] T. Ogasawara. NUMA-aware memory manager with dominant-thread-based copying GC. In *Proceedings of OOPSLA'09*, 2009.

[22] A. Sandberg, D. Eklöv, and E. Hagersten. Reducing cache pollution through detection and elimination of non-temporal memory accesses. In *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, SC'10.

[23] S. Srikantaiah, M. Kandemir, and M. J. Irwin. Adaptive set pinning: managing shared caches in chip multiprocessors. In *Proceedings of ASPLOS'08*, 2008.

[24] M. M. Tikir and J. K. Hollingsworth. NUMA-aware Java heaps for server applications. In *Proceedings of IPDPS'05*.

[25] M. M. Tikir and J. K. Hollingsworth. Hardware monitors for dynamic page migration. *Journal of Parallel and Distributed Computing*, 68(9):1186–1200, 2008.

[26] I. Tuduce, Z. Majo, A. Gauch, B. Chen, and T. R. Gross. Asymmetries in multi-core systems – or why we need better performance measurement units. The Exascale Evaluation and Research Techniques Workshop (EXERT) co-located with ASPLOS'10.

[27] B. Verghese, S. Devine, A. Gupta, and M. Rosenblum. Operating system support for improving data locality on CC-NUMA compute servers. In *Proceedings of ASPLOS'96*.

[28] K. M. Wilson and B. B. Aglietti. Dynamic page placement to improve locality in CC-NUMA multiprocessors for TPC-C. In *Proceedings of SC'01*, 2001.

[29] R. Yang, J. Antony, P. P. Janes, and A. P. Rendell. Memory and thread placement effects as a function of cache usage: A study of the gaussian chemistry code on the SunFire X4600 M2. In *Proceedings of ISPAN'08*.