Memory System Performance in a NUMA

Mohamed Wageh Saad
Department of Computer Engineering
Nile University
M.wageh@nu.edu.eg

Hussien Elshazly Eida
Department of Computer Science
Nile University
hu.elshazly@nu.edu.eg

Mohamed Ahmed
Department of Computer Science
Nile University
Moh.Abdelwahab@nu.edu.eg

Dariwsh Mohamed Dariwsh
Department of Computer Science
Nile University
D.Dariwsh@nu.edu.eg

Abstract**—** *The modern multiprocessor has the advantage that the memory is present with controller on the same chip of the processor. Each processor has the right to access a physical part of the memory, and it can access the other parts of the other processors purchased through the memory controller that comes with each processor. This is done through a connection between the processors at the same system. Now, as for the controller of the memory, it has two types of requests that it fulfills:1- The first type is the one that is coming from his processor, which is available with him on the same chip. 2- The second type that is coming through other processors that are exist with it in the multi core processor, and this will come through the interconnection between them. On the other hand, the processor and, accordingly, the cache memory, get the data from such sources 1- get the data via local memory controller. 2- Or a remote memory controller with another processor. In our project, we will try to do an analysis of the behavior, and this will be done through a multi-core processor Intel Xeon 5520. We will create a model to describe the Sharing of the local and the remote memory for the Bandwidth. Asymmetric or ineffective processing has an effect on limiting applications on multicore processors. It does not have to enlarge the local data to reduces the execution time. For example, it is possible that it is better that I store my data on a remote processor and get the data it through the interconnection between the processors. It is better than storing all the data in the local memory, and thus this will cause an increase in the load on the controller of the memory stick.*

Keywords— (*NUMA, memory controller, memory system performance, multicore multiprocessor****)***

INTRODUCTION

Now, the existing microprocessors are a multi core systems, and it is likely that the number of cores that will remain integrated in one processor in the future will increase. One of the challenges facing multi-core processors is to provide adequate access to the memory of the processor What could solve a problem like this. This was the solution for the designers was to use the cache memory, but with the increase in the number of courses, they had to find another, better way to reach a higher and higher bandwidth memory while avoiding additional increases in the time of access to the memory latency. In order to improve the interface with the memory and provide all the multi-cores with data, the designers of the processors worked on integrating the on-chip memory controller with the processor.

When we come to compare with the designs that existed before that, which used to be the controller for memory off chip, the solution that the controller for the memory for is on the same chip with the processor, providing access to the memory , the bandwidth is larger and the time of latency is less. The advantages of the on-chip memory controller: The local memory controller allows the expansion of the system on the basis that there is no single control unit in the central memory, instead of this, the physical address space is divided between the processors and the cores of each processor that can access a part of the actual memory through the local memory interface. And in order to port the familiar model of chip memory processors, each processor and its cores must be able to access not only the local memory that is directly connected with the processor on the same chip, but also be able to access the local memory of every other processor found in the multiprocessor system.

The processes of accessing the remote memory pass through chips that connect the processors to each other. examples of interconnection of proctors the Intel QuickPath Interconnect QPI, or the AMD Hypertransport. Also, the throughput of the connection in remote memory less than the throughput the local memory. This is because the result of the non uniform memory interface in memory. The memory in multicore processor is categorized to Non Uniform Memory Architecture. The penalty for reaching remote memory access is large, the penalty is called the NUMA factor.

In current implementations, the Neuma factor can be as high as 2X for some applications. The most important performance improvement for NUMA systems considered so far is to increase the system data (data locality), meaning allocating a memory close to the part of the computations you access, so that the number of accesses to the remote memory is reduced, and thus reducing the NUMA penalty factor. Many researchers have investigated improving the performance of applications on NUMA systems, all of which focus on increasing the data locality, which aims to allocate memory and make a map next to the parts responsible for the computations in the system. Because the performance of the applications is related to the performance of the memory system, it is important to understand the memory system, such as the NUMA, because simple, realistic models are necessary in order to know what to do with

the data next to the parts responsible for the computations, and thus reach a good result.

In this paper, we analyze the bandwidth-sharing characteristics of a commercial microprocessor and discuss the implications of this. Characteristics for software optimization in multi-core systems We show that in some cases when the hardware is heavily loaded, cross-connection outperforms the on-chip memory controller. Setting arithmetic operations so that all memory traffic flows through the local memory interface is bound to be suboptimal in many situations. Due to recent processor design times, we anticipate that our evaluation techniques and feedback will be of interest for some time as processors based on this microarchitecture are released.

Related Work

C. Xie et al. [2] make NUMA-based multi-GPU system is a promising candidate to provide sustainable and scalable performance for Virtual Reality (VR) applications. However, the entire system is viewed as a single GPU under the single programming model which greatly ignores the data locality among VR rendering tasks. To tackle these challenges, we propose object-oriented VR rendering framework (OO-VR) that conducts the software and hardware co-optimization. OO-VR provides 1.58x overall performance improvement and 76% inter-GPM memory traffic reduction over the state-of- the-art multi-PUP systems. Z. Majo et la. [3] Some programs are (unintentionally) structured so that they use the memory system of today's NUMA-multicores inefficiently. Many programs have irregular memory access patterns that are hard to predict by processor prefetcher units. A set of simple algorithmic changes coupled with commonly available OS functionality suffice to eliminate data sharing. These simple source-level changes result in performance improvements of up to 3.1X, but more importantly, lead to a fairer and more accurate performance evaluation. Molka et al. [5] examine the memory system performance of the Intel Nehalem in depth. Hackenberg et al. later compare the performance of the Intel Nehalem against that of the AMD Shanghai. Their technique assesses memory bandwidth and access latency, as well as the influence of cache coherency. However, they only analyze the features of the various interconnects of multicore devices in isolation, not their interplay. Our tests only included cache lines in the E, M, and I states, but the measurement methodology may be expanded to include bandwidth sharing to cache lines in other coherency stages as well. Yang et al. investigate the impact of memory and thread location on application performance in an AMD Opteron-based NUMA computer. They measure execution time but not low-level hardware concerns (e.g., cache coherency traffic that is significant in some execution configuration of their benchmarks). Mandal et al. [4] estimate the memory bandwidth and memory access latency of commercially available systems (including the Nehalem) as a function of the system's concurrent memory references. However, it is challenging to expand their approach to incorporate memory controller sharing across several kinds since requests might be generated at different rates via the on-chip memory controller and the QPI. Their pointer-chasing test also includes inter-core misses, therefore the reported numbers are partly reliant on the examined systems' cache sharing behavior.

The Proposed Solution

Measure the bandwidth achieved by each instance of the triad benchmark. We measure all possible local-remote mapping configurations for any given number of triad processes. Figure 6 shows a scenario where four local processes share the IMC bandwidth with different numbers of remote processes. 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.

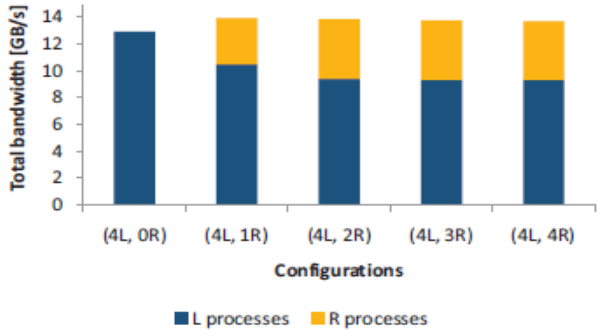


Figure 6: Bandwidth sharing (4L with variable number of R processes).

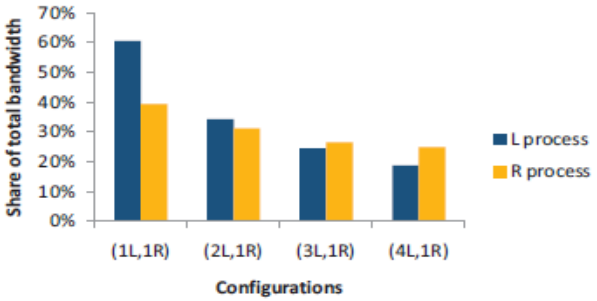


Figure 7: Percentage of L and R of the total bandwidth measured in the system.

Technical Merit

The evaluation machine is running Ubuntu Linux 2.6.30. We use Processor Performance Monitor to get information about elapsed CPU cycles and how much Last Level Cache (LLC) the program is missing. We calculate the bandwidth of the memory created with Equation Bandwidth = 64 LLC Error 2.27 109 CP U cycles 106 MB/s The LLC cache font size for Intel Nehalem is 64 bytes. The processors in our system were clocked at 2.27 GHz. Due to the limitations of the performance monitor, we can only measure the read bandwidth of the kernel. We also use Nehalem's uncertain monitoring facility to monitor GQ status and to check the readings obtained using

performance counters for each core. Since the triple workload is very memory-related and has a single program phase, its bandwidth and performance readings are very stable and not factor dependent. All measurement data are the average of three measurements; The discrepancy in the performance meter

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

readings is negligible.

We use the triad workload of the STREAM benchmark 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 .

```
Testing using Single threaded serial scheduler:
Kernel    Max (GB/s)    Min (GB/s)  Avr (GB/s)
Fill      16.34359204   4.73745602  11.45581529
Copy      22.29047955   2.62862924  12.25058444
Scale     17.61273655   5.62377623  14.41017294
Sum       17.29768948   1.68461536  10.11201218
Triad     17.56100193   4.15732138  13.89831946
```

```
Testing using Multi threaded parallel scheduler:
Kernel    Max (GB/s)    Min (GB/s)  Avr (GB/s)
Fill      24.84502979   5.58200757  19.47911895
Copy      45.51954837   7.51830350  20.86125857
Scale     23.19925329   7.18768837  14.33155116
Sum       21.94276536   8.33745105  15.24099577
Triad     24.12389746   8.18939972  14.40516221
```

A. *CONTRAST TO RELATED WORK*

The overhead of the cache coherency protocol was not taken into consideration in our investigation. A snoop request is sent to the cache of the next processor for every cache miss (as assessed by the read-, write-, and peer–probe–tracker of each CPU's uncore). Snoop requests are sent across the system's cross-chip connection. However, although normal reads often request data of the same size as a cache line, we don't know how much data is conveyed with snoop queries, so we can't estimate the amount of traffic created by these requests or the cache coherency protocol's bandwidth overhead. In this paper, we analyse the main memory system performance of a NUMA-multicore machine using a single, homogenous workload (composed of several triad clones). Because the triad benchmark does not use caching, its performance closely resembles that of our evaluation system's main memory system. Caching effects are also present when many, heterogeneous workloads are running on the system (with applications that are less memory bound than triad). For a consideration of these circumstances and the applicability of the principles presented in the previous part to OS scheduling, see the following sections.

B. *EVALUTION AND RESULTS*

	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].

simple model

Our experiments show that if there are only local processes running on the system, then the total bandwidth obtained by these processes can be described as: bwLtotal = min (active cores ∗ bwL, bwLmax) (2) in Equation 2 bwL is the width Bandwidth Single, locally implemented, triad clone (L process) (see column '0 R' of Ta ble 1). If the sum of the bandwidth of the individual bwLtotal cores is greater than the bwLmax threshold, then each core gets an equal share of the threshold value (12925 MB/s). Similarly, the total bandwidth obtained by remote operations can be described as follows: bwRtotal = min (active cores ∗ bwR, bwRmax) (3) in Equation 3 bwR is the bandwidth achieved by the sin gle instance that It is executed remotely (R process). The maximum throughput for R operations (bwRmax) limited by the QPI interface is 6998 MB/s (determined experimentally). QPI is also fair in the sense that if the threshold is exceeded, all R-processes get an equal share of the total bandwidth. The total bandwidth obtained by the system consists of the bandwidth achieved by the L and R operations.

The bwLmax limit for the L operations can be seen in row “4 L” and column “0 R”. Similarly, the bwRmax limit for R operations can be observed in row '0 L' and column '2 R'. Remote processes have reached their maximum bwRmax with two active cores, while four local processes are required to reach their maximum bwLmax. This is because the QPI is already saturated with two triplexes, but the four cores must be active to saturate the IMC. Nehalem future generations have more cores connected to the same local memory controller, so not all processor cores are required to meet the saturation limit of the IMC. In Section 4 we briefly look at such a machine. Formally, the total bandwidth in the system can be compressed as follows:

bwtotal = (1 - β) ∗ bwLtotal + β ∗ bwRtotal (4) We call the variable β the participation factor. The sharing factor determines the share of the total bandwidth that the local and remote triplets receive. β is a real value between 0 and 1. If it is 1, all bandwidth is obtained by R operations. Similarly, if it is 0, all bandwidth is obtained by L operations. Since the global queue (GQ)) controls between local memory accesses and memory accesses, GQ determines the value of β based on the rate at which requests arrive at its ports. If the system must handle incoming memory requests from a small number of cores, the bandwidth (and therefore per-form) of local

processes is much better than the bandwidth of the remote. As the load on the system increases and there are more local processes, the bandwidth obtained by individual local processes (bwL) becomes similar to the cumulative bandwidth of QPI (bwRtotal total). It is also possible for situations where the QPI bandwidth is better than the bandwidth of individual local processes (eg configuration (4L, 1R) and (3L, 1R)). Overloading QPI with a large number of memory-related processes executing remotely should be avoided, since the low throughput of the QPI interface is divided between R processes, resulting in lower performance of R processes, if their number is too large. In conclusion, if the system is of low usage, local implementation is preferred. However, as the load on the memory system increases, remote execution becomes more convenient, but care must be taken not to overload the interface across chips. To fully understand the system, the dependence of the β-sharing factor of GQ on the load coming from local cores and remote memory interfaces must be activated. However, since most of the implementation details of the Nehalem queue system are not disclosed, and the performance monitoring subsystem in our Nehalem-based processes does not allow directly measuring the state of the queue, it is difficult to build such a model. Instead, in Sections 3.2 and 3.3, we describe two experimentally observed properties of GQ that aid in understanding the bandwidth-sharing properties of our evaluation system: queuing fairness and gateway throughput.

	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	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].

C. *Queuing fairness*

For any number of local operations, there is a significant difference between the unsaturated QPI throughput performing a single R operation (the “1 R” column), and the throughput of the QPI performing the two R operations (the “2 R” column). Adding more R operations (columns “3R” and “4R”) does not modify the overall system bandwidth allocation, as the input limit in QPI has already been reached, and the QPI is saturated. However, a significant difference was observed in the total bandwidth obtained by the L and R processes with changing the number of L processes (rows '1 L' to '4 L'). In the following, we consider QPI as a fifth factor connected to GQ (in addition to the four local cores), or cut the exe as either the 1R workload, or the equivalent workload to the memory intensity generated by the 2R workload. We take the performance of two cases as a basis. In the first case, GQ serves 1R of QPI and 1L of local cores, as shown in Fig. 8a. In the second case, GQ serves 2R with 1L, as shown in Fig. 8b. Using predefined notation, these workloads can be denoted as (1L, 1R) and (1L, 2R). To increase competition for GQ, one, two or three additional L operation(s) are performed on the system. These L operations (the base process L plus additional L operations) handle the QPI of the IMC bandwidth. Figure 9 shows the variance of the participation factor (the parameter from Equation 4) when the disagreement on the local port of GQ increases. The share factor depends on the load on GQ: the more L traffic processes generate, the higher the share of bandwidth they get, and the higher the share of R processes (given by β). However, if we consider the performance degradation for the basic workload (1L, 1R) and (1L, 2R) (shown in Fig. 10a and Fig. 10b respectively), the performance of the single L process in each of the two workloads deteriorates more. of QPI performance. Therefore, the higher the load on GQ, the more attractive it is to perform some operations remotely. In conclusion, if GQ is contested, the Nehalem mi architecture is unfair towards local cores (versus QPI), as local cores experience greater performance degradation than that of QPI. However, this behavior is reasonable because GQ does not allow remote cores to be starved, thus avoiding further aggravation of the remote memory access penalty. However, this property of Nehalem is undocumented and can only be discovered by empirical evaluation.

	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

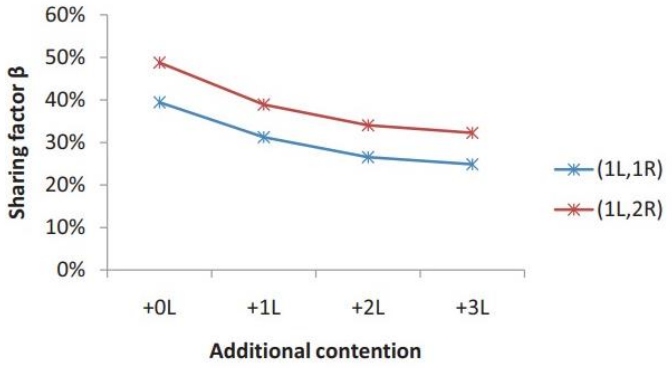
Table 4: Total R bandwidth [MB/s].

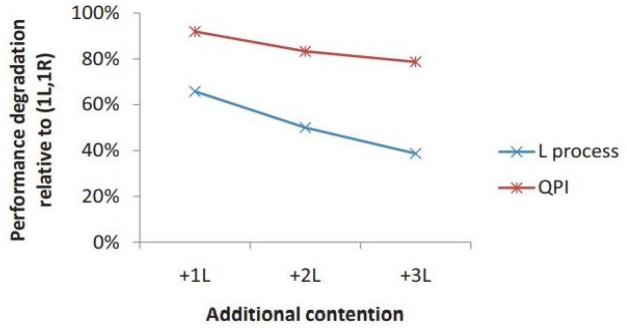
D. *Aggregate throughput*

simple model

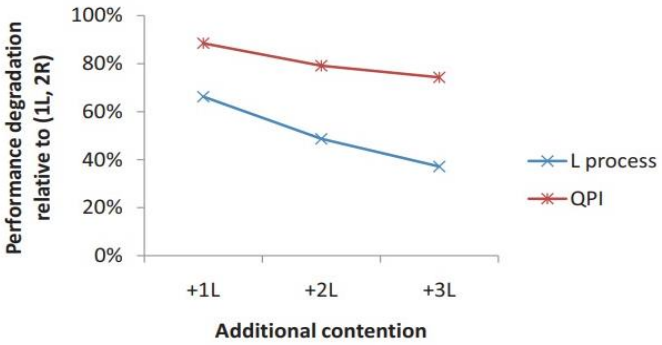
To further motivate the benefit of having a good professional portion of local and remote memory accesses in the system, we are showing total system throughput for the 4P workload in different mapping configurations (ranging from configuring when all processes are executed locally to configuring with all remotely executing processes) . In configurations with some remote memory accesses, the system memory throughput can be better (at 13842 MB/s peak) relative to a 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 assign triple workload processes to local cores. In this way, all memory processes use the local ports of the global queue. Next, we switch one process to the remote processor, and thus the QPI port: the performance of workloads (1L, 1R) and (1L, 2R) is degraded. GQ is also used to effectively handle memory requests. In both cases, we calculate the total system throughput as the sum of the instruction-per-cycle (IPC) values obtained by the processes: different instructions) using the scale specified in Equation 5 is not appropriate. However, in our case, all processes implement the same narrow memory-intensive loop that operates on identical data, so the instructions executed by each workload are the same. The clock rate of all processor cores is also equal, so monitoring the instructions being executed and the cycles consumed is an accurate measure of system throughput. Since Triad is very memory intensive, the total memory bandwidth achieved on the system is also directly proportional to the system by mode. This metric does not characterize system fairness, but rather accurately reflects the throughput of the main memory system. Shows the benefit of setting a single remote process to the entire local state (as all memory requests come from local cores). The benefit is slight (1.7%) if there are only two processes running on the system, but it becomes significant (7.4%) if there are four processes. This increase in performance in the case of the four processes can be explained by the distribution of disagreement over GQ. When GQ handles four locally executed triplets, its local port is saturated (it is full 10% of the time). Moving a single process to the remote processor shifts some of the load from the local GQ port to its remote port. In this new configuration, neither the local nor the remote port of GQ is classified, so the system throughput increases. However, if all operations are performed remotely, the remote port of GQ becomes saturated (it is full 31% of the time). In conclusion, in a single threaded context, the bandwidth and response time of the on-chip memory interface significantly outperform the same QPI parameters. However, in the case where multiple cores are competing, this advantage diminishes as the contention over the waiting system increases. Deleting discretionary calculations such as having local and remote access in the system helps improve overall productivity.

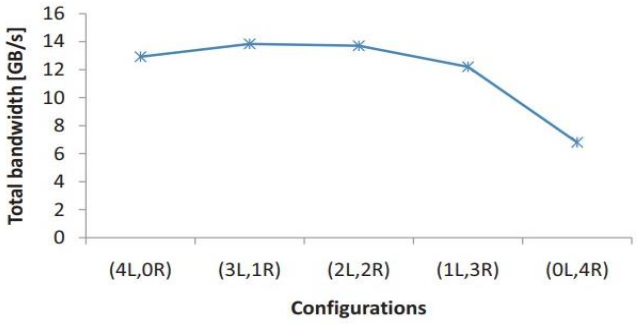


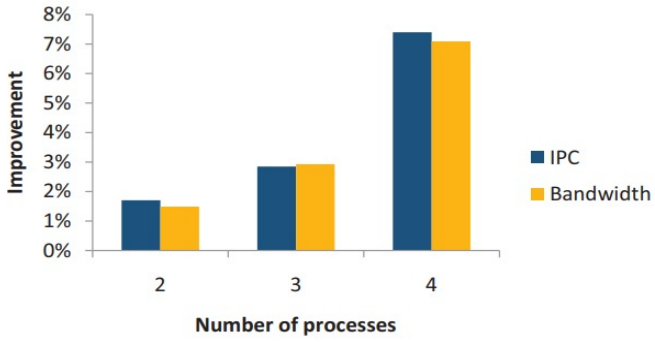


(a) Performance degradation of (1L,1R).



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





Conclusion

Today's multi-core processors that integrate a memory controller with cores and cache on a single chip. Such a design leads to a new generation of multi-core multi-core NUMA processors offered to software developers A new set of challenges and the creation of a different class of performance

improvement problems. Cores pressed Memory controller for local memory access service requests while at the same time the memory controller should Handling other processor requests. So, it is important That the program finds a balance between the local and the remote Up memory if overall performance is optimal. This paper presents an evaluation of the bandwidth-sharing characteristics of a commercially available multi core system, Intel Nehalem (Xeon 5520) processor, shows that if it's large Part or all of the processor cores are active, then preference for the data area may not lead to optimal performance. in In addition to the data location, bandwidth limits for memory controllers, and arbitrage fairness between them Local and remote access is also important. overhead Arbitration and alignment are likely to become more important in larger systems due to the complexity of this mechanism It increases with the number of processors in the system,
It is therefore important to develop realistic memory models that can guide the operating system and the assembler developers.

REFERENCES

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

[2] C. Xie, F. Xin, M. Chen and S. L. Song, "OO-VR: NUMA Friendly Object-Oriented VR Rendering Framework For Future NUMA-Based Multi-GPU Systems," 2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA), 2019, pp. 53-65.

[3] Z. Majo and T. R. Gross, "(Mis)understanding the NUMA memory system performance of multithreaded workloads," 2013 IEEE International Symposium on Workload Characterization (IISWC), 2013, pp. 11-22, doi: 10.1109/IISWC.2013.6704666.

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

[5] D. Molka, D. Hackenberg, R. Sch¨one, and M. S. M¨uller. Memory performance and cache coherency effects on an Intel Nehalem multiprocessor system. In Proceedings of PACT'09.

[6] 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.

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

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

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