# A Cluster Computer Performance Predictor for Memory Scheduling

Mónica Serrano, Julio Sahuquillo, Houcine Hassan, Salvador Petit, and José Duato

Department of Computer Engineering (DISCA)
Universidad Politécnica de Valencia
Camino de Vera s/n, 46022 Valencia, Spain
Tel.: +34-677415807
Fax: +34-963877579
monica@gap.upv.es, {jsahuqui,husein,spetit,jduato}@disca.upv.es

**Abstract.** Remote Memory Access (RMA) hardware allow a given motherboard in a cluster to directly access the memory installed in a remote motherboard of the same cluster. In recent works, this characteristic has been used to extend the addressable memory space of selected motherboards, which enable a better balance of main memory resources among cluster applications. This way is much more cost-effective than than implementing a full-fledged shared memory system.
In this context, the memory scheduler is in charge of finding a suitable distribution of local and remote memory that maximizes the performance and guarantees a minimum QoS among the applications. Note that since changing the memory distribution is a slow process involving several motherboards, the memory scheduler needs to make sure that the target distribution provides better performance than the current one.
In this paper, a performance predictor is designed in order to find the best memory distribution for a given set of applications executing in a cluster motherboard. The predictor uses simple hardware counters to estimate the expected impact on performance of the different memory distributions. The hardware counters provide the predictor with the information about the time spent in processor, memory access and network.
The performance model used by the predictor has been validated in a detailed microarchitectural simulator using real benchmarks. Results show that the prediction accuracy never deviates more than 5% compared to the real results, being less than 0.5% in most of the cases.

**Keywords:** cluster computers, memory scheduling, remote memory assignment, performance estimation

## 1 Introduction

Since their introduction, cluster computers have been improving their performance and lowering their implementation costs with respect to supercomputers. Nowadays, it is easy to find many of these type of computer organizations in the top positions of high-performance computer rankings such as TOP500 [1]. This transition has been possible as advanced microarchitectural techniques and interconnection solutions only available in supercomputers enter the consumer market (i.e., they are *commoditized*), which in turn allow new ways to improve the performance of current cluster designs while maintaining or even lowering their expenses.

However, since cluster architectures are loosely coupled by design, there is not a standard commodity framework supporting the access to memory installed on remote nodes. Therefore, to cope with applications demanding large amounts of main memory (e.g., enterprise level databases and services, large computing intensive parallel applications, etc.), cluster systems must rely on slower OS-based solutions such as swapping on remote RAM disks or implementing software-based shared memory. This, in turn, reduces the competitivity advantages of this type of systems.

So far, Remote Memory Access (RMA) hardware [2], which allows a given node to directly access remote memory, has been only available in supercomputer systems like BlueGene/L [3], BlueGene/P [4], or Cray XT [5]. Nevertheless, commodity implementations for cluster computers are already entering the market. For example, the HyperTransport consortium [6], which is composed by more than 60 members from the leading industry (AMD, HP, Dell, IBM, etc.) and universities, is extending the Hypertransport technology, enabling the development of cluster systems supporting remote memory accesses.

This work focuses on a cluster prototype that implements the aforementioned Hypertransport extensions and whose nodes are linked using a fast interconnection network. In this context, we assume that the OS running in the nodes offers inter-node memory allocation capabilities that enable the assignment of remote memory portions to local applications.

As these regions have different latencies, performance of a given application strongly depends on how its assigned memory is distributed among regions. Since each application contributes with its performance to the global performance, a memory scheduler that maximizes the global performance is required. This memory scheduler must be aware not only of the characteristics (i.e., latency, bandwidth) of the different memory regions but also of the executing applications' memory requirements. For example, allocating a 25% of the available remote memory to a memory-intensive application could lead to worse performance results than allocating the whole remote memory to an application with good cache locality.

To decide how to distribute the different memory regions among the running applications, the scheduler needs information about the expected performance of a given memory distribution. To obtain this information two solutions can be devised: i) to perform an off-line profiling of the benchmarks varying the memory distribution and ii) to dynamically predict the performance of the benchmarks by measuring during execution their utilization of the system resources. The first solution has been developed in a previous work [7], where we analyzed how the memory distribution impacts on the performance of applications with different memory requirements, and presented an ideal memory allocation algorithm (referred to as SPP) that distributed the memory space among applications to maximize global performance. The generalization of SSP to any number $n$ of applications was published in [8], where we also present an efficient heuristic algorithm that approximates the performance results provided by SPP while reducing its complexity in a factor of $(n-1)!$. Both algorithms consider a quality of service (QoS) parameter for each application in order to guarantee minimum performance requirements.

In contrast to these works, this paper proposes a performance predictor that provides the information required by the memory scheduler. The main aim of proposed predictor is to be used by the memory scheduler to maximize the system performance while guaranteeing specific QoS requirements. To perform the predictions, 3 sample executions for every benchmark are required, each one considering that the complete working set of the benchmark is stored in a different memory region (i.e., L, Lb or R). Using these samples the performance of any other memory distribution is estimated.

The proposed predictor is driven by a novel performance model fed by simple hardware counters (like those available in most current processors) that measure the distribution of execution time devoted to processor, memory, and network resources. Although the model can be implemented for any type of processor, this work considers in-order execution for simplicity reasons. The model has been validated by comparing its estimations with the performance values obtained by the execution of real benchmarks in the Multi2Sim simulation framework [9]. The results show that the dynamic predictor is very accurate, since its deviation with respect to the real results is always lower than 5%, and much lower in most of the cases.

The remaining of this paper is organized as follows. Section 2 describes the system prototype. Section 3 details our proposed performance model. Section 4 validates the model by comparing its predictions with detailed cycle-by-cycle simulation results. Section 5 discusses previous research related to this work, and finally, Section 6 presents some concluding remarks.

## 2 Cluster Prototype

A cluster machine with the required hardware/software capabilities is being prototyped in conjunction with researchers from the University of Heidelberg [2], which have designed the RMA connection cards. The machine consists of 64 motherboards each one including 4 quad-core 2.0GHz Opteron processors in a 4-node NUMA system (1 processor per node), and a 16GB RAM memory per motherboard. The connection to remote motherboards is implemented by a regular HyperTransport [10] interface to the local motherboard and a High Node Count HyperTransport [11] interface to the remote boards. This interface is attached to the motherboard by means of HTX compatible cards [12].

When a processor issues a load or store instruction, the memory operation is forwarded to the memory controller of the node handling that memory address. The RMA connection cards include their own controller, which handles the accesses to remote memory. Unlike typical memory controllers, the RMA controller has no memory banks directly connected to it. Instead, it relies on the banks installed in remote motherboards. This controller can be reconfigured so that memory accesses to a given memory address are forwarded to the selected motherboard.

Since the prototype is still under construction, in order to carry out the experiments and validate the proposed performance model, the cluster machine has been modeled using Multi2Sim. Multi2Sim is a simulation framework for superscalar, multithreaded,

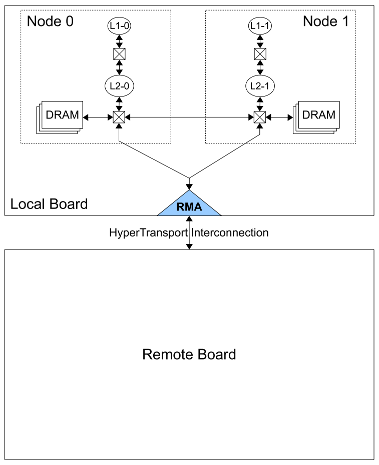

**Fig. 1.** Block diagram of the 2-node NUMA system model and RMA

**Table 1.** Memory subsystem characteristics

| **Characteristic** | **Description** |
|---|---|
| # of processors | 2 per motherboard |
| L1 cache: size, #ways, line size | 64KB, 2, 64B |
| L1 cache latency | 3 |
| L2 cache: size, #ways, line size | 1MB, 16, 64B |
| L2 cache latency | 6 |
| Memory address space | 512MB, 256MB per motherboard |
| L Latency | 100 |
| Lb Latency | 142 |
| R Latency | 410 |

and multicore processors. It is an application-only execution-driven microarchitectural simulator, which allows the execution of multiple applications to be simulated without booting a complete OS.

In addition, the whole system has been scaled down to have reasonable simulation times. The scaled system consists of two motherboards, each one composed of a 2-node NUMA system as shown in Figure 1. Each node includes a processor with private caches, its memory controller and the associated RAM memory.

Table 1 shows the memory subsystem characteristics, where memory latencies and cache organizations resemble those of the real prototype. The RMA connection cards have been assumed with no internal storage capacity. Likewise, the Multi2Sim coherence protocol has been extended to model the RMA functionality.

## 3 Performance Model

A system whose running applications can be executed using different memory distributions (L, Lb, R) needs a mechanism to determine which memory distribution should be assigned to each application. This section presents a methodology for predicting the impact on performance of the different memory distributions, and then using the predictions to guide the assignment of memory regions to applications in order to meet memory constraints and reduce performance loss.

This work assumes that the predictor evaluates seven possible memory distributions (three samples and four estimated cases) since this number of data points is enough to define sufficiently the performance of each application among the complete set of possible memory distributions [8]. To predict the performance (execution time) of a running application A when having a memory assignment $\{L = X, Lb = Y, R = Z\}$, an analytical method has been designed.

Existing processors implement performance counters for debugging purposes which are readable by software. In this paper, these counters are utilized by an application-to-memory assignment prediction mechanism. The counters are used to track the number of cycles spent for each considered event during a full scheduling quantum.

### 3.1 Analytical Model

The execution time of a given application can be estimated from two main components, as stated by equation 1.

$$T_{ex} = C_{Dispatch} + C_{mem\_stalls} \tag{1}$$

Each $C_x$ is the number of processor cycles spent on a type of activity. As the dispatch width has been assumed to be 1, the execution time can be expressed as the sum of the number of dispatched instructions plus the number of cycles stalled due to memory accesses.

In the devised system, stalls due to a full load-store queue (LSQ) are critical for performance, mainly in those benchmarks having a high rate of memory accesses. On the other hand, dispatch stage remains stalled during the execution of a load instruction. This includes both the accesses to private caches (i.e. L1 and L2) and to the main memory, with their respective access times as well as the delays related to the network or structural hazards.

To project the IPC, the performance model breaks down the memory components of the execution time into memory region-dependent and memory region-independent components:

$$C_{mem\_stalls} = C_L + C_{Lb} + C_R + C_{private\_caches} + C_{LSQ\_iwidth} \tag{2}$$

$C_L$, $C_{Lb}$, and $C_R$ refer to the cycles spent on each memory region, that is, Local, Local to Board, respectively. Each $C_{<region>}$ includes the cycles due to several activities related to this memory region. In particular, stalls due to the following reasons have been taken into account:

**Main memory access time.** This time includes both the cycles spent in the data read from the main memory and the message traffic through the memory network.

**Delayed hit.** This type of stall occurs when the memory access cannot be performed because the accessed block is already locked by another memory instruction, that is, a new block is being brought.

**Write concurrency.** This type of stall happens because concurrent accesses to the same block in a given cache are not allowed if one of them is a write.

**Full LSQ.** Dispatch stage is stalled because there is no free entry in the LSQ.

The remaining components of the equation can be considered as a constant $k$ for every memory region. The region-independent components are the following:

**Private caches access time.** Number of cycles spent in accessing the first and second level caches of the system. These accesses are region-independent since no memory module is accessed.

**LSQ issue width limitation.** Only a load or a store can be issued at a given cycle. So, if a load instruction is ready to be issued and there is an access conflict between a load and a store, they are issued in program order, and the youngest instruction will retry the next cycle.

The final equation used by the performance predictor is 3:

$$T_{ex} = C_{Dispatch} + C_L + C_{Lb} + C_R + k \tag{3}$$

## 3.2 Estimating Performance

The model assumes that the implemented target machine provides the required performance counters to obtain the values for the components of equation 3. Notice that network traffic is taken into account, so congestion is also quantified.

The predictor requires to run each benchmark three times to gather the required values to project performance. Each sample will correspond to all the memory accesses in one single region, that is, *i)* all the accesses to local memory region (i.e. $T_{ex,L=100\%}$), *ii)* all the accesses to the other node in the local motherboard memory region (i.e. $T_{ex,Lb=100\%}$), and *iii)* all the accesses to remote memory region (i.e. $T_{ex,R=100\%}$):

*Sample 1 (L = 100%, Lb = 0%, R = 0%)*: $T_{ex,L=100\%} = C_{L:L=100\%} + k$
*Sample 2 (L = 0%, Lb = 100%, R = 0%)*: $T_{ex,Lb=100\%} = C_{Lb:Lb=100\%} + k$
*Sample 3 (L = 0%, Lb = 0%, R = 100%)*: $T_{ex,R=100\%} = C_{R:R=100\%} + k$

To predict the execution time for a given memory distribution, the predictor calculates a weighted execution time, $T_{ex\_weighted}$, from the three samples. It takes each not null memory region component $C_{<region>}$ of each of the samples and multiplies it by the fraction $f_{<region>}$ of accesses of the destination memory region:

$$T_{ex\_weighted} = C_{L,L=100\%} \cdot (f_L) + C_{Lb,Lb=100\%} \cdot (f_{Lb}) + C_{R,R=100\%} \cdot (f_R) + k \tag{4}$$

**Table 2.** Performance predictor working example

| | $C_{<region>}$ | $f_{<region>}$ | $C_{<region>\_pond}$ |
|---|---|---|---|
| *Sample1* | 44687 | 0.5 | 22343.5 |
| *Sample2* | 62236 | 0.5 | 31118 |
| *Sample3* | 166757 | 0 | 0 |
| *k* | | | 2721346.3 |
| $t_{ex\_weighted}$ | | | 2774807.8 |

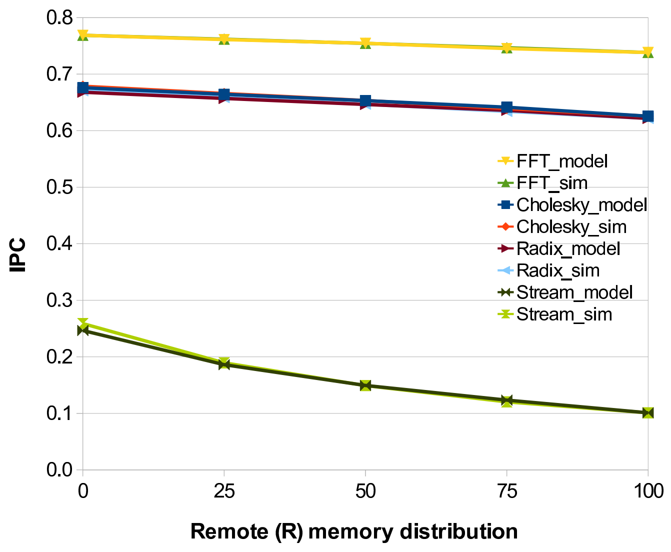

**Fig. 2.** Model Validation. Detailed cycle-by-cycle simulation vs model

For any given memory distribution, equation 4 can be used to predict its execution time given the gathered components for the three samples. This provides a mechanism to identify the optimal memory distribution at which to run a given execution phase with minimal performance loss. So this prediction will be an input for the memory scheduler.

Table 2 analyzes an example of prediction for the benchmark FFT, where the execution time of the memory distribution $(50\%, 50\%, 0)$ is obtained from the three samples. The estimated execution time is equal to 2774807.8 and the real detailed cycle-by-cycle simulation execution time is 2774931, so the model has obtained an estimation which deviates less than 0.005% with respect to the target value.

## 4 Validating the Model

This section analyzes the prediction accuracy. We have proceed by making experiments for the four benchmarks with the eight memory distributions: *i)*(100%, 0%, 0%), *ii)*(50%, 50%, 0%), *iii)*(0%, 100%, 0%), *iv)*(75%, 0%, 25%), *v)*(50%, 25%, 25%), *vi)*(50%, 0%, 50%), *vii)*(25%, 0%, 75%), *viii)*(0%, 0%, 100%). Then, we have taken the components of the three samples ($i$, $iii$, and $viii$) and have applied the model to each benchmark to obtain the execution time for each of the remaining memory distributions. Finally, the Instructions Per Cycle (IPC) has been calculated for each case.

Figure 2 shows the comparison of the simulated performance results (*sim*) against the values calculated by the performance predictor (*model*). Both model and detailed cycle-by-cycle simulation curves are overlapped, since the model provides a deviation lower than $5\%$ in the worst case, being near to $0\%$ for some of the benchmarks, for instance, FFT.

## 5 Related work

Previous research works have addressed the problem of performance prediction to characterize and classify memory behavior of applications to predict their performance.

Zhuravlev et al [13] estimated that factors like memory controller, memory bus and prefetching hardware contentions contribute more to overall performance degradation than cache space contention. To alleviate these factors they minimize the total number of misses issued from each cache. To that end they developed scheduling algorithms that distribute threads such that the miss rate is evenly distributed among the caches.

In [14] authors propose a classification algorithm for determining programs cache sharing behaviors. Their scheme can be implemented directly in hardware to provide dynamic classification of program behaviors. They propose a very simple dynamic cache partitioning scheme that performs slightly better than the Utility-based Cache Partitioning scheme while incurring a lower implementation cost.

In [15] a fast and accurate shared cache aware performance model for multi-core processors is proposed. The model estimates the performance degradation due to cache contention of processes running on CMPs. It uses reuse distance histograms, cache access frequencies, and the relationship between the throughput and cache miss rate of each process to predict its effective cache size when running concurrently and sharing cache with other processes, allowing instruction throughput estimation. The average throughput prediction error of the model was 1.57

In [16] the authors apply machine learning techniques to predict the performance on multi-core processors. The main contribution of the study is enumeration of solo-run program attributes, which can be used to predict paired-run performance. The paired run involves the contention for shared resources between co-running programs.

The previous research papers are focused on multicore or CMP processors however the work proposed in this paper is focused on cluster computers dealing with the problem of predicting the application behaviour using remote memory in order to allow a scheduler to improve system performance.

Other research papers found in the bibliography dealing with remote memory allocation are mainly focused on memory swapping. Shuang et al. design a remote paging system for remote memory utilization in InfiniBand clusters [17]. In [18], the use of remote memory for virtual memory swapping in a cluster computer is described. Midorikawa et al. propose the distributed large memory system (DLM), which is an user-level software-only solution that provides very large virtual memory by using remote memory distributed over the nodes in a cluster [19].

These papers use the remote memory for swapping over cluster nodes and present their system as an improvement of disk swapping. On the contrary, our research aims at predicting system performance depending on different assignment configurations of remote memory to applications. The predictions will be used by a memory scheduler to decide dynamically which is the best configuration to enhance system performance.

## 6 Conclusions

This paper has presented a performance predictor which is able to estimate the execution time for a given memory distribution of an application. We first carried out a study to determine the events considered by our model, and classified them as memory-region dependent and independent. The model assumes that the number of cycles spent in each considered event is obtained from some hardware counters of the target machine.

The devised predictor has been used to estimate the performance of different memory distributions for four benchmarks. The accuracy of the prediction has been validated, since the deviation of the model with respect to the real results is always lower than 5% and very close to 0% in several studied cases.

This study constitutes the first step of a deeper work in the ground of memory scheduling. The performances estimated by the predictor will feed a memory scheduler which will dynamically choose the optimum target memory distribution for each application concurrently running in the system in order to achieve the best overall performance of the system.

## Acknowledgements

This work was supported by Spanish CICYT under Grant TIN2009-14475-C04-01, and by Consolider-Ingenio under Grant CSD2006-00046.

## References

1. Hans Werner Meuer. The top500 project: Looking back over 15 years of supercomputing experience. *Informatik-Spektrum*, 31:203–222, 2008. 10.1007/s00287-008-0240-6.
2. Mondrian Nussle, Martin Scherer, and Ulrich Bruning. A Resource Optimized Remote-Memory-Access Architecture for Low-latency Communication. In *International Conference on Parallel Processing*, pages 220–227, Sept. 2009.
3. M. Blocksome, C. Archer, T. Inglett, P. McCarthy, M. Mundy, J. Ratterman, A. Sidelnik, B. Smith, G. Almási, J. Casta nos, D. Lieber, J. Moreira, S. Krishnamoorthy, V. Tipparaju, and J. Nieplocha. Design and implementation of a one-sided communication interface for the IBM eServer Blue Gene®supercomputer. In *Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 120, New York, NY, USA, 2006. ACM.
4. Sameer Kumar, Gábor Dózsa, Gheorghe Almasi, Philip Heidelberger, Dong Chen, Mark Giampapa, Michael Blocksome, Ahmad Faraj, Jeff Parker, Joe Ratterman, Brian E. Smith, and Charles Archer. The deep computing messaging framework: generalized scalable message passing on the blue gene/P supercomputer. In *ICS*, pages 94–103, 2008.

5. V. Tipparaju, A. Kot, J. Nieplocha, M.T. Bruggencate, and N. Chrisochoides. Evaluation of Remote Memory Access Communication on the Cray XT3. In *Parallel and Distributed Processing Symposium. IEEE International*, pages 1–7, March 2007.
6. HyperTransport Technology Consortium. HyperTransport I/O Link Specification Revision 3.10, 2008. .
7. M. Serrano, J. Sahuquillo, H. Hassan, S. Petit, and J. Duato. A scheduling heuristic to handle local and remote memory in cluster computers. In *High Performance Computing and Communications*, 2010. Accepted for publication.
8. M. Serrano, J. Sahuquillo, S. Petit, H. Hassan, and J. Duato. A cost-effective heuristic to schedule local and remote memory in cluster computers. *The Journal of Supercomputing*, pages 1–19, 2011. 10.1007/s11227-011-0566-8.
9. R. Ubal, J. Sahuquillo, S. Petit, and P. López. Multi2Sim: A Simulation Framework to Evaluate Multicore-Multithreaded Processors. *Proceedings of the 19th International Symposium on Computer Architecture and High Performance Computing*, 2007.
10. Chetana N. Keltcher, Kevin J. McGrath, Ardsher Ahmed, and Pat Conway. The AMD Opteron Processor for Multiprocessor Servers. *IEEE Micro*, 23(2):66–76, 2003.
11. J. Duato, F. Silla, and S. Yalamanchili. Extending HyperTransport Protocol for Improved Scalability. *First International Workshop on HyperTransport Research and Applications*, 2009.
12. H. Litz, H. Fröening, M. Nuessle, and U. Brüening. A HyperTransport Network Interface Controller for Ultra-low Latency Message Transfers. *HyperTransport Consortium White Paper*, 2007.
13. Sergey Zhuravlev, Sergey Blagodurov, and Alexandra Fedorova. Addressing shared resource contention in multicore processors via scheduling. In *Proceedings of the 15th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 129–142, 2010.
14. Y. Xie and G. H. Loh. Dynamic Classification of Program Memory Behaviors in CMPs. *2nd Workshop on Chip Multiprocessor Memory Systems and Interconnects in conjunction with the 35th International Symposium on Computer Architecture*, 2008.
15. Chi Xu, Xi Chen, Robert P. Dick, and Zhuoqing Morley Mao. Cache contention and application performance prediction for multi-core systems. In *IEEE International Symposium on Performance Analysis of Systems and Software*, pages 76–86, 2010.
16. J.K. Rai, A. Negi, R. Wankar, and K.D. Nayak. Performance prediction on multi-core processors. In *Computational Intelligence and Communication Networks (CICN), 2010 International Conference on*, pages 633 –637, nov. 2010.
17. Shuang Liang, Ranjit Noronha, and Dhabaleswar K. Panda. Swapping to Remote Memory over InfiniBand: An Approach using a High Performance Network Block Device. In *CLUSTER*, pages 1–10. IEEE, 2005.
18. Paul Werstein, Xiangfei Jia, and Zhiyi Huang. A Remote Memory Swapping System for Cluster Computers. In *Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 75–81, 2007.
19. H. Midorikawa, M. Kurokawa, R. Himeno, and M. Sato. DLM: A distributed Large Memory System using remote memory swapping over cluster nodes. In *IEEE International Conference on Cluster Computing*, pages 268–273, October 2008.