

Analyzing the Trade-off between Multiple Memory Controllers and Memory Channels on Multi-core Processor Performance

José Carlos Sancho, Mike Lang, Darren J. Kerbyson

Performance and Architecture Laboratory (PAL)

Computer Science for HPC (CCS-1)

Los Alamos National Laboratory, NM 87545, USA

{jcsancho,mlang,djk}@lanl.gov

Abstract

Increasing the core-count on current and future processors is posing critical challenges to the memory subsystem to efficiently handle concurrent memory requests. The current trend to cope with this challenge is to increase the number of memory channels available to the processor's memory controller. In this paper we investigate the effectiveness of this approach on the performance of parallel scientific applications. Specifically, we explore the trade-off between employing multiple memory channels per memory controller and the use of multiple memory controllers. Experiments conducted on two current state-of-the-art multicore processors, a 6-core AMD Istanbul and a 4-core Intel Nehalem-EP, for a wide range of production applications shows that there is a diminishing return when increasing the number of memory channels per memory controller. In addition, we show that this performance degradation can be efficiently addressed by increasing the ratio of memory controllers to channels while keeping the number of memory channels constant. Significant performance improvements can be achieved in this scheme, up to 28%, in the case of using two memory controllers with each with one channel compared with one controller with two memory channels.

1 Introduction

The increased silicon integration that is possible today and foreseen into the future has led to an unprecedented growth in the number of processor-cores. Current main-stream processors from Intel, AMD, and IBM have 6-8 cores and recent experimental designs have much more, such as the 48-cores x86 processor from Intel [8]. However, this increase in compute capability comes with a significant cost - that of tremendously stressing the memory subsystem, and making worse the well-known *memory wall* that can profoundly limit performance. Current memory subsystem designs are not able to sustain all the memory requirements from multiple cores for many memory intensive applications. A major constraint that prevents a linear performance improvement of the memory subsystem, proportional to the core-count, is the processor pin-count. The design of the memory subsystem is critical in order to achieve maximum efficiency from the available pins.

The current industry trend to cope with the memory challenge to multi-core processors is to increase the number of memory channels available to the memory controller. Examples illustrating the increase in the number memory channels include the Intel's Nehalem processors [4] that currently has three memory channels with a single memory controller (the quad-core Nehalem-EP), which will increase to four memory channels next year (the oct-core Nehalem-EX). The next generation of the IBM's power processor, the oct-core Power7, will use eight memory channels spread across two memory controllers. This trend for increasing the performance of the memory subsystem is evolutionary as it leverages previous memory controller designs. However, it does has an additional advantage of substantially boosting

the performance on sequential since a single core can potentially use all available channels simultaneously. It is not clear how this current trend will remain suitable for parallel applications in which the optimization of single threaded applications is not important but rather sustaining concurrent demand from multiple threads. Future technological innovations, such as stacking processors and memory chips using Through Silicon Vias (TSV), will also tackle this challenge but are currently in an experimental phase.

This paper aims to shed some light into memory subsystem performance for multi-core processors for large-scale scientific applications by exploring the trade-offs between multiple memory channels per memory controller and multiple memory controllers. We show that there is a alarming diminishing performance return when considering the current trend of increasing the number of memory channels per memory controller. Moreover, we show how this diminishing return could be efficiently addressed by adding more memory controllers in a processor while keeping the total number of channels per chip constant, i.e. keeping the pin-count constant. We quantify how much performance improvement is lost when increasing the number of memory channels per memory controller, and also how much performance could be achieved by increasing the number of memory controllers instead. Results from empirical experiments are included for a wide range of scientific applications, memory-bound to compute-bound, on two state-of-the-art production HPC processors - Intel’s 4-core Nehalem-EP [4] and AMD’s 6-core Istanbul [9].

The rest of this paper is organized as follows. Section 2 describes our method that enables an empirical analysis of varying the number of memory channels and memory controllers available as well as describing the testbeds and the scientific applications employed. Section 3 discusses the results obtained from the testbeds. Related work on analyzing the performance impact of the memory subsystem is summarized in Section 4. Conclusions from this work are given in Section 5.

2 Approach

Our approach to analyze the trade-off between using multiple memory controllers and multiple memory channels is described below. We employ the use of a broad set of production scientific applications and two current state-of-the-art multi-core processors from AMD and Intel. The first compute-node contains four 6-core AMD Istanbul processors and the second contains two 4-core Intel Nehalem-EP processors. As listed in Table 1, each Istanbul processor, executing at 2.6GHz, has a single memory controller with two DDR2-800MHz memory channels. Each Nehalem-EP processor, executing at 2.93GHz, also has a single memory controller but with three DDR3-1333MHz memory channels. The other processors listed in Table 1 are near-to-market processors that have also been investigated but results cannot be published at this time.

Table 1: Current state-of-the-art processing nodes

Vendor	Name	procs/node	cores/proc	Controllers/proc	Channels/controller	Memory
AMD	Istanbul	2 or 4	6	1	2	DDR2
AMD	MagnyCours	4 or 8	6	1	2	DDR3
Intel	Nehalem-EP	2	4	1	3	DDR3
Intel	Nehalem-EX	2 or 4	8	1	4	DDR3
IBM	Power7	4	8	2	8	DDR3

The default mode of operation of these processors is to use of all cores, where each core executes a separate thread of the application, and to use all of the memory channels requiring all channels to be populated by identical memory DIMMS.

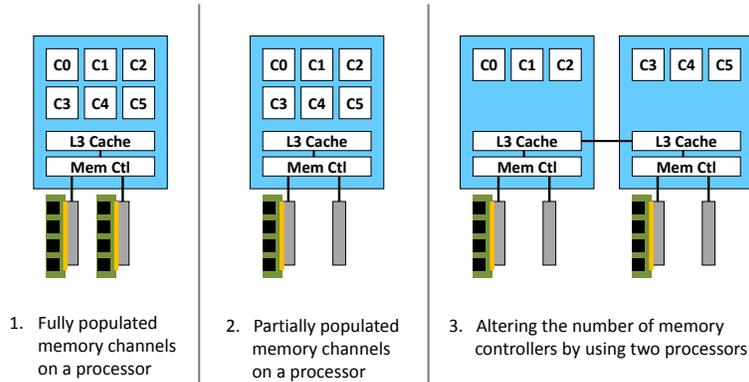


Figure 1: The three scenarios used for evaluating multiple memory channels and controllers.

In this work we explore the achievable performance using three scenarios:

1. Using the default node configuration, with fully populated memory channels, to explore the achievable performance when varying the number of processor-cores used.
2. Undertake the same analysis as in case 1 but using only a sub-set of the available memory channels.
3. Alter the ratio of memory-controllers to memory-channels by using only a sub-set of the available cores per processor, and spreading the cores-used among processors within the node.

These three cases are illustrated in Figure 1 for a six-core single processor with two memory channels. In case 1, when using a single processor, all memory channels are populated and between 1 and the maximum available number of cores are used on the single processor. Case 2 is identical except that the memory channels are underpopulated by the physical removal of memory DIMMS. The change in the ratio between memory controllers and channels, case 3, is achieved using multiple processors and by also underpopulating the memory channels. Note that this is an approximation but captures the first-order effects to enable conclusions to be drawn from this analysis.

Clearly, the advantage of this approach is that we can quickly obtain estimates on actual hardware without having to perform any simulations. However the approach provides an approximation to the performance that may be achievable, for case 3. It includes some performance penalties caused by the spread of processes across multiple processors rather than having the controllers on the same processor. Penalties include the extra communication generated to transfer data among processors. In the case of our two testbeds, we can only compare the case of using two memory controllers (two processors) with one memory channel on each and can compare it to the case of one memory controller with two memory channels. Further configurations are not possible using our testbeds.

For all cases, application performance is measured for a fixed problem size on a single processor following the *strong scaling* model. Moreover, performance results are collected for various core-counts so as to also investigate the sensitivity of core-count on the performance. Note that when spreading out the cores being used across processors, for case 3, we also increase the application’s working set in proportion to the number of processors being so as to minimize the effect of having multiple L3 caches available.

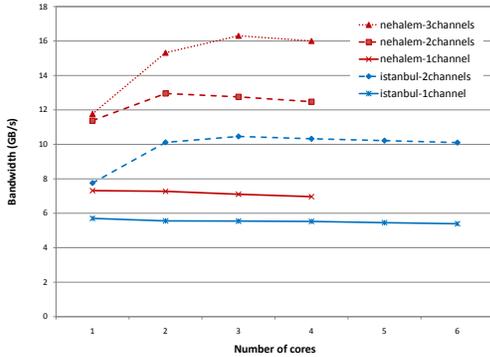


Figure 2: STREAMS performance.

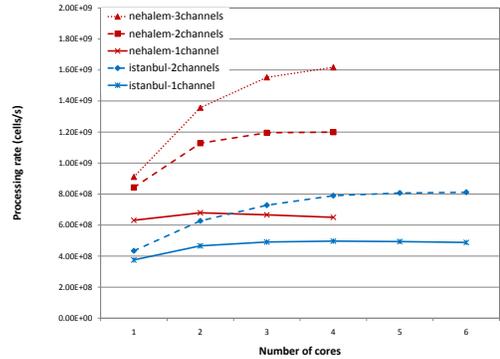


Figure 3: SAGE performance.

2.1 Scientific applications

The applications used in this analysis are SAGE [10], MILC [14], POP [11], S3D [7], XNOBEL, and SWEEP3D [12]. Many of these are taken from existing workloads within the Department of Energy, and are summarized in Table 2. These applications have differing requirements of the memory subsystem. At one extreme is SAGE - a memory-bound code, and at the other extreme is SWEEP3D - a compute-bound code. In addition we also make use the STREAMS [13] benchmark to measure the peak achievable bandwidth of the memory subsystem, and to compare against application performance.

Table 2: Description of scientific applications

Name	Description	Input deck	Problem size	Processing type
SAGE	Hydro dynamics (Hydro)	<i>timing_h</i>	140,000	Memory
MILC	Quantum chromodynamics	<i>SU3_RMD</i>	$8 \times 8 \times 40 \times 48$	Memory
POP	Ocean circulation model	<i>x1</i>	$320 \times 384 \times 40$	Memory/Compute
S3D	3D Turbulent combustion	<i>typical</i>	$100 \times 100 \times 100$	Memory/Compute
XNOBEL	Hydro with high explosives	<i>typical</i>	$10 \times 10 \times 10$	Memory
SWEEP3D	3D S_N radiation transport	<i>Pencil</i>	$20 \times 10 \times 400$	Compute

3 Evaluation

The analysis is split into two main sections - the first deals with the default processor configuration when fully (case 1) and partially populating (case 2) available memory channels, and the second details the case of using multiple memory controllers (case 3).

3.1 Multiple memory channels per memory controller

We focus initially on the performance of the STREAMS benchmark and use these results to explain the underlying principles that are also seen in the application results.

3.1.1 STREAMS performance

Figure 2 shows the performance impact on STREAMS when varying both the number of memory channels and number of cores on the Istanbul and Nehalem processors. As can be seen, a single core can achieve up to 5.7 GB/s and 7.3 GB/s from a single memory channel on Istanbul and Nehalem respectively. These numbers represent 90% and 70% of the hardware peak memory bandwidth for DDR2-800 and DDR3-1333 (6.4Gb/s and 10.6 GB/s).

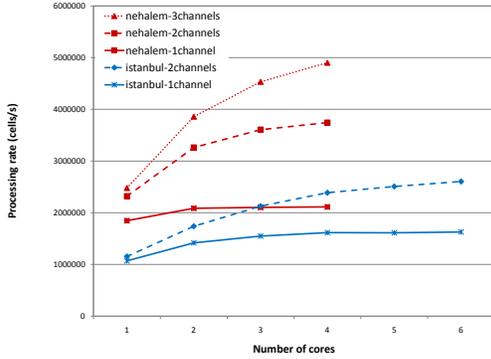


Figure 4: XNOBEL performance.

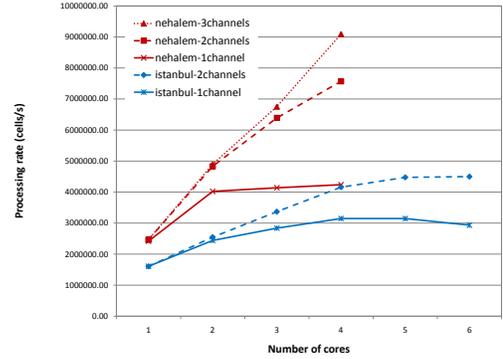


Figure 5: MILC performance.

When increasing the number of memory channels, a single core achieves improved performance by simultaneously using the additional channels. 7.7 GB/s is achieved by a single-core on Istanbul and up to 11.3 GB/s and 11.7 GB/s on two and three channels respectively on Nehalem. Therefore, performance improvements can be obtained by sequential applications when employing multiple channels.

When using multiple cores, reflecting parallel applications, the aggregate bandwidth achieved increases on both Intel and AMD processors with respect to a single-core. But, the bandwidth does not increase linearly with the number of cores used. A saturation point can be observed above which no further improvement in bandwidth occurs. The performance achieved at the saturation point on a single memory channel is denoted as $Perf_{1-channel}$. For Istanbul, the saturation point occurs at one-core and three-cores when using one or two memory channels respectively. The resulting aggregate bandwidth when using two channels is 10.5GB/s - only a 35% increase above a single-cores bandwidth. For Nehalem, the saturation point occurs at one-core, two-cores, and three-cores when using one, two and three memory channels respectively. The aggregate bandwidth increases by 14% (12.9GB/s) and 40% (16.3GB/s) for the two and three channel cases at the saturation points. Note that when using more cores beyond the saturation points the delivered performance drops, by as such as 4%, due to contention on the single memory controller from multiple cores.

Table 3: Summary of STREAMS bandwidth

Processor	Peak	1 channel		Peak	2 channel		Peak	3 channel	
		STREAMS	%		STREAMS	%		STREAMS	%
Istanbul	6.4	5.7	89%	12.8	10.5	82%			
Nehalem	10.6	7.3	68%	21.2	12.9	61%	31.8	16.3	51%

A summary of the achievable STREAMS bandwidth at the saturation points is listed in Table 3 when using different numbers of memory channels. It can be seen that only a fraction of the peak bandwidth is being achieved and that the fraction significantly decreases as the number of memory channels increases. These results clearly indicate that there is a diminishing return on the aggregate bandwidth that one or multiple cores can achieve when using multiple channels with a single memory controller.

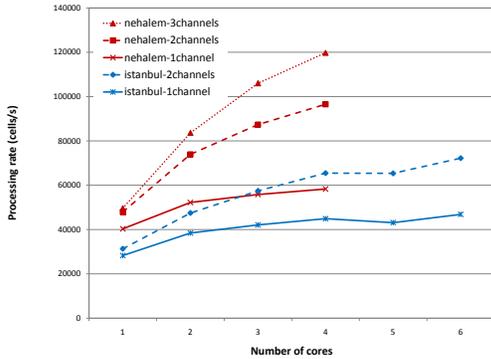


Figure 6: S3D performance.

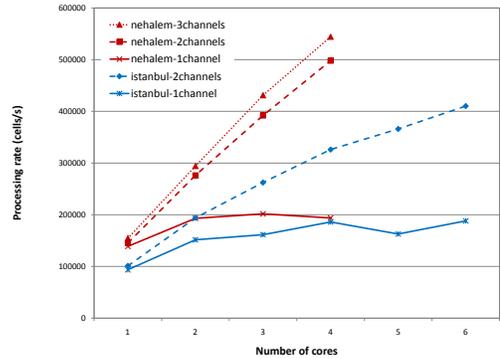


Figure 7: POP performance.

3.1.2 Scientific application performance

The application suite was run under the same conditions as the STREAMS benchmark for various memory channel configurations and core-counts. The observed performance for SAGE, XNOBEL, MILC, S3D, POP, and SWEEP3D are shown in Figures 3, 4, 5, 6, 7, and 8 respectively. The performance of these applications, when varying the number of memory channels, is similar to that observed for STREAMS apart from SWEEP3D. The performance of SWEEP3D is invariant to the number of channels as it is compute-bound and performance improvements are due solely to the increased parallelism.

Applications that are mostly memory bound, or a mixture of compute and memory bound, are impacted significantly by the number of memory channels. In particular, the performance is increased by 64%, 60%, 30%, 35%, and 54% for SAGE, XNOBEL, MILC, S3D, and POP, respectively, when using two channels compared with a single memory channel. On Nehalem, the performance is increased by 76%, 77%, 44%, 40%, and 62% using two channels, and by 35%, 30%, 17%, 20%, and 0% when using three channels for the same applications.

Note that for many of the applications, the performance increases monotonically with core-count as there is not enough cores to reach a saturation point. Only SAGE, XNOBEL, and MILC reach a saturation point. The saturation point for SAGE is at 3 cores per channel on Istanbul and 2 cores per channel on Nehalem. The saturation point for XNOBEL and MILC is at 4 cores per memory channel on Istanbul. As shown for these applications we still observe a diminishing return in common with STREAMS when using multiple channels with a single memory controller.

3.2 Multiple memory controllers

Here we mimic the case of using two memory controllers per processor by using multiple processors as described in Section 2. The principle followed is to compare the performance when using the same number of memory channels but spread across multiple memory controllers. The testbed nodes enable a direct comparison between a single processor having a single memory-controller with two channels and two processors each having a single memory-controller with a single channel. In both cases, the same number of cores is used by the applications. The observed performance improvements achieved in this case, for both Istanbul and Nehalem are shown in Figure 9. It can be seen that in all cases, a performance improvement occurs of up to 28% (XNOBEL on Istanbul) with a minimum of 5% (MILC on Nehalem).

Larger improvements are seen on Istanbul because it is the one more penalized when using multiple memory channels at the memory controller. Those performance improvements are really close, within 3%, from the ideal $Perf_{nchannels}$ on STREAMS and SAGE that are

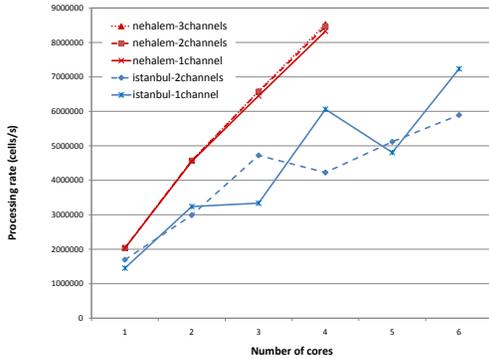


Figure 8: SWEEP3D performance.

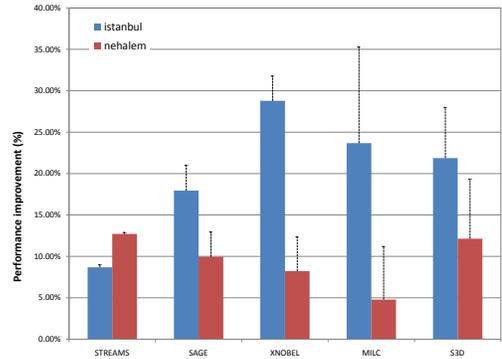


Figure 9: Application performance improvement on two memory controllers.

the applications that already have enough cores to reach saturation points on this setup.

4 Related work

The work in this paper spans the areas of application performance, memory performance analysis, and systems architecture. The original memory wall problem stems from the memory subsystem not keeping pace with the increasing processor clock speed [15]. Presently, the speed of processors have plateaued but the memory issues remain due to the increase in core-count - it is a parallel feed rather than a serial one [6] that is now poses the greatest challenge. As such, this study is similar to previous work that characterizes memory performance.

There have been many recent studies on the achievable application performance on multicore processors including [1]. Further work has focused on the optimization of the memory controller itself including [3], and others have looked at predicting future memory performance based on reducing memory bus frequency [5]. Higher density of memory controllers have been suggested and designed in the past, including Compaq’s 8-core *Piranha*, which had 8 memory controllers - one per memory channel [2] on a single die.

5 Conclusions

Currently, a trend to cope with the memory challenge posed by increasing cores in a processor is being addressed by increasing the number of memory channels available to a memory controller. In this paper, we investigate the effectiveness of this approach. Through empirical analysis using scientific applications on two state-of-the-art multicore processor nodes from Intel and AMD we have demonstrated this approach is not sufficiently effective for a wide range of parallel applications. Performance does not increase proportionally as the number of memory channels, available to a memory controller, increases. On today’s multicore processors, memory-intensive scientific applications achieve between a 30% and 76% performance increase when using two memory channel compared with one, and between a negligible and 35% performance increases when using a third channel. This trend needs to be addressed because memory channels, and thus pin count, are a scarce processor resource which should be fully exploited.

We have investigated the case of adding more memory controllers on a chip in order to overcome these diminishing returns. We have shown that by higher performances can be achieved by increasing the number of memory controllers in a chip whilst keeping the overall number of channels constant. One memory channel per controller can achieve significant improvements for parallel applications rather than having multiple channels. In particular, two memory controllers each with one channel can increase the performance by 28% in com-

parison to one controller with two channels. Larger performance improvements are expected with larger numbers of memory controllers. However, more work has to be done in order to fully deploy this approach in current processors. In particular, the affinity between cores and memory controllers as well as coherency and addressability of the entire memory from each core all been to be investigated. We feel that this work provides a unique analysis into the trade-off between memory controllers and memory channels using current production applications and state-of-the-art processing nodes.

Acknowledgments

This work was funded in part by the Advanced Simulation and Computing program of the Department of Energy, and the Office of Science. Los Alamos is operated by the Los Alamos National Security, LLC for the US Department of Energy under contract No. DE-AC52-06NA25396.

References

- [1] K. Barker, K. Davis, et al. A Performance Evaluation of the Nehalem Quad-core Processor for Scientific Computing. *Parallel Processing Letters*, 18(4):453–469, Dec. 2008.
- [2] L. A. Barroso, K. Gharachorloo, et al. Piranha: a Scalable Architecture based on Single-chip Multiprocessing. In *Proceedings of International Symposium on Computer Architecture*. Vancouver, Canada, Jun. 2000.
- [3] J. B. Carter and L. Zhang. A Study of Performance Impact of Memory Controller. In *Proceedings of Workshop on Memory Performance Issues, in conjunction with the International Symposium on Computer Architecture*. Munich, Germany, Jun. 2004.
- [4] J. Casazza. First the Tick, Now the Tock: Intel Microarchitecture (Nehalem). In *Intel White Paper*. 2009. Available at <http://www.intel.com/technology/architecture-silicon/next-gen/319724.pdf>.
- [5] D. Doerfler. Personal Communication. 2009.
- [6] J. Dongarra, G. Dennis, et al. The Impact of Multicore on Computational Science Software. *CTWatch Quarterly*, 3(1), Feb. 2007. Available at <http://www.ctwatch.org/quarterly/articles/2007/02/the-impact-of-multicore-on-computational-science-software/index.html>.
- [7] E. R. Hawkes, R. Sankaran, et al. Direct Numerical Simulation of Turbulent Combustion: Fundamental Insights towards Predictive Models. *Journal of Physics*, 16:pp. 65–79, 2005.
- [8] Futuristic Intel Chip Could Reshape How Computers are Built, Consumers Interact with Their PCs and Personal Devices. Press released at http://www.intel.com/pressroom/archive/releases/2009/20091202comp_sm.html.
- [9] AMD Istanbul Processor. Information available at <http://developer.amd.com/ZONES/ISTANBUL>.
- [10] D. J. Kerbyson, H. J. Alme, et al. Predictive Performance and Scalability Modeling of a Large-scale Application. In *Proceedings of the Supercomputing Conference*. Denver, CO, Nov. 2001.
- [11] D. J. Kerbyson and P. W. Jones. A Performance Model of the Parallel Ocean Program. *International Journal of High Performance Computing Applications*, vol. 35(no. 3):pp. 261–276, 2005.
- [12] K. R. Koch, R. S. Baker, et al. First-Order Form of the 3-D Discrete Ordinates Equation on a Massively Parallel Processor. *Trans. of the American Nuclear Soc.*, 65:pp. 198–199, 1992.
- [13] J. McCalpin. Memory Bandwidth and Machine Balance in Current High Performance Computers. *IEEE Computer Society Technical Committee on Computer Architecture (TCCA) Newsletter*, pp. 19–25, 1995. Code available at <http://www.cs.virginia.edu/stream/>.
- [14] MIMD Lattice Computation (MILC) Collaboration. Code available at <http://www.physics.indiana.edu/sg/milc.html>.
- [15] W. A. Wulf and S. A. McKee. Hitting the Memory Wall: Implications of the Obvious. *SIGARCH Computation Architecture News*, 23(1):pp. 20–24, 1995.