Can Manycores Support the Memory Requirements of Scientific Applications?

Milan Pavlovic, Yoav Etsion, Alex Ramirez

Can Manycores Support the Memory Requirements of Scientific Applications?

Milan Pavlovic[1], Yoav Etsion[1], and Alex Ramirez[12]

[1] Barcelona Supercomputing Center (BSC-CNS)
[2] Universitat Politècnica de Catalunya (UPC)
{milan.pavlovic,yoav.etsion,alex.ramirez}@bsc.es

Abstract. Manycores are very effective in scaling parallel computational performance. However, it is not clear if current memory technologies can scale to support such highly parallel processors.
In this paper, we examine the memory bandwidth and footprint required by a number of high-performance scientific applications. We find such applications require a per-core memory bandwidth of ∼300MB/s, and have a memory footprint of some 300MB per-core.
When comparing these requirements with the limitations of state-of-the-art DRAM technology, we project that in the scientific domain, current memory technologies will likely scale well to support more than ∼100 cores on a single chip, but may become a performance bottleneck for manycores consisting of more than 200 cores.

1 Introduction

The inability to efficiently scale single thread performance through frequency scaling, has left on-chip parallelism as the only viable path for scaling performance, and vendors are already producing chip multiprocessors (CMPs) consisting of dozens of processing contexts on a single die [1, 7, 11].

But placing multiple processing units on a single chip imposes greater load on the memory system. While on-chip parallelism is effective at scaling the computational performance of a single chip (i.e. the number of arithmetic operations per second), it is unclear whether the memory system can scale to supply CMPs with sufficient data.

Existing memory systems face inherent limitations that impede scaling of both memory bandwidth and size. Off-chip memory bandwidth is limited by the number of chip-to-board pins on one hand, and by the signaling frequencies of each pin on the other. On-chip caches are therefore commonly used to reduce the number of off-chip accesses, thereby reducing off-chip bandwidth (and access latencies), but are limited in size. Moreover, the amount of memory that can be put on a single board is limited by factors such as memory density and the limited distance between the memory controller and the DIMMs required to guarantee stable signaling. We focus our projections on DRAM technology, as it remains un-contended as the dominant technology for main memories.

These design issues motivate the exploration of memory requirements of real parallel applications.

In this paper we therefore characterize the memory behavior of several well-known parallel scientific applications, and try to predict the performance required from the memory system to adequately serve large-scale CMPs.

Given the lack of parallel applications explicitly targeting shared-memory CMPs, we base our predictions on the per-CPU memory requirements of distributed memory MPI applications. Although this methodology is imperfect (data may be replicated between nodes, which may result in pessimistic predictions when addressing shared-memory environments), we believe it provides a good indication of the requirements from a CMP memory system.

For the applications examined, we show the per-node memory bandwidth reaches hundreds of MB/s in most cases, and that L2 bandwidth is typically larger by an order of magnitude. In addition, we show the per-node working set size typically consists of hundreds of MBs. A simple back-of-the-envelope calculation therefore suggests that a CMP consisting of 100 cores may require dozens of GBs of memory space, accessed at rates up to 100 GB/s.

2 Analysis methodology

Detailed analysis of full-scale multiprocessor applications is infeasible due to the sheer size of the application under study. A common approach to performing detailed studies on large-scale codes is sampling. That is, most of the application is executed without analysis, and at certain intervals (random, or carefully selected), detailed analysis data is collected.

However, when it comes to parallel systems and parallel applications, the sampling methodology presents several uncertain aspects: how should the application behave with regard to timing when it is not being modeled? That is: how should the different application threads and hardware components align when the next detailed sample is reached?

In order to solve this problem, we analyze the application in order to determine its regular (iterative) behavior, and to detect the sequence of phases that compose such iterative nature. This allows us to focus our requirements analysis on each of the individual phases in a way that allows us to later on rebuild the global application behavior, as shown in Figure 1.

We perform a full execution of the application, instrumented at the higher abstraction level: CPU bursts, synchronization and communication events. It produces a full timestamped trace of events, annotated with hardware performance counters and memory usage statistics associated to each CPU burst. The full trace, representing hours of real execution, is still too large for processing.

In order to reduce it to a manageable size, we apply non-linear filtering and spectral analysis techniques to determine the internal structure of the trace and detect periodicity of applications. Based on this analysis, we can cut a sample of the original trace, between 10 and 150 times smaller than the full trace [10]. Next, we use a density-based clustering algorithm applied to the hardware counter to

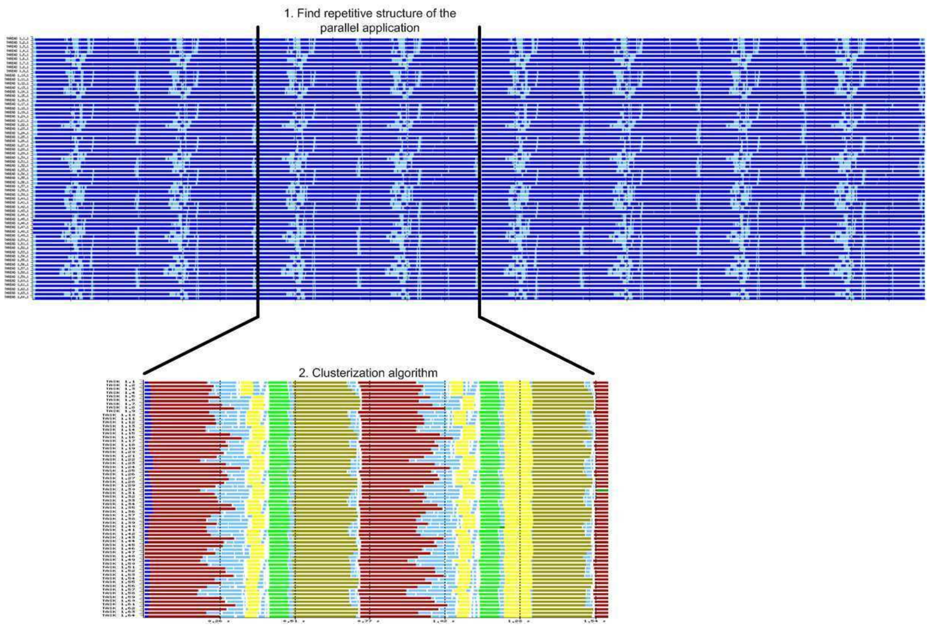

1. Find repetitive structure of the parallel application
2. Clusterization algorithm

Fig. 1. Full-scale application analysis methodology

determine the most representative computation CPU bursts inside a period of the new trace [4].

At that point, we can analyze in isolation each one of the detected CPU phases (cluster of CPU bursts). Not only that, we can also compute an average for each metric over the different representatives of the same CPU burst in each of the application threads. This allows us to represent data in an easy to understand form, yet still trace it back to the full-scale application execution.

Our evaluation platform is a cluster of JS21 blades (nodes), each hosting 4 IBM Power PC 970MP processors running at 2.3 GHz. Each node has 8 GB of RAM, shared among its 4 processors, and is connected to a high-speed Myrinet type M3S-PCIXD-2-I port, as well as two GigaBit Ethernet ports.

In order to avoid contention on the nodes' RAM, the benchmarks executed using only a single processor per node. Therefore, an application running on 64 processors actually had exclusive access 64 nodes and 256 processors, of which 192 were idle (3 per node) such that each processor used by the application had 8 GB of memory and the full bandwidth at its disposal.

3 Application analysis

3.1 Memory bandwidth

Consolidating multiple cores on a single chip imposes much higher bandwidth requirements on the shared components of the memory system — namely the

Application	Description
GADGET2	Cosmological N-body and smoothed particle hydrodynamics simulations.
GROMACS	Molecular dynamics package simulating the Newtonian motion equations of large particle systems.
LESLIE3D	Computational fluid dynamics code.
MILC	Large scale simulations of four dimensional SU(3) lattice gauge theory.
POP	Ocean circulation model.
SOCORRO	Self-consistent electronic-structure calculations.
SPECFEM3D	Southern California seismic wave propagation based upon the spectralelement method (SEM).
TERA_TF	3D Eulerian hydrodynamics application.
VAC4	Solver astrophysical hydrodynamical and magnetohydrodynamical problems.
WRF	Mesocale numerical weather prediction system.
ZEUS-MP	Computational fluid dynamics for astrophysical phenomena.

Table 1. List of examined applications

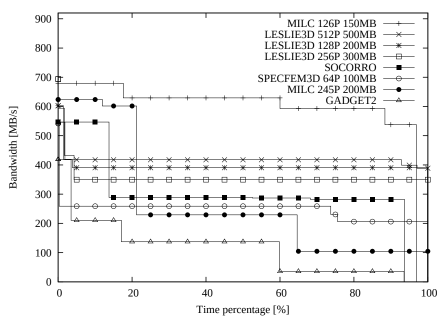
off-chip memory bandwidth and the shared caches. In order to predict the bandwidth requirements, we measured the per-processor bandwidth consumed by each benchmark at three levels: the off-chip memory bandwidth, L2 bandwidth, and L1 bandwidth. Figures 2(a)–2(b) depict the average per-processor off-chip bandwidth, and Figures 2(c)–2(d) depict the bandwidth between the L1 and L2 caches.

For the sake of readability, the benchmarks were split into two groups, based on their bandwidth consumption. As discussed Section 2, the entire execution time is divided into clusters. Phases of an application experiencing similar characteristics — specifically CPI, computation intensity, percentage of TLB misses, and bandwidth requirements — are grouped into the same cluster. For each benchmark, we focus our discussion on the four clusters dominating benchmark's runtime. Therefore, the X axis represents the percentage of the execution time spent in each cluster, and the Y axis shows the measured bandwidth in MB/s.

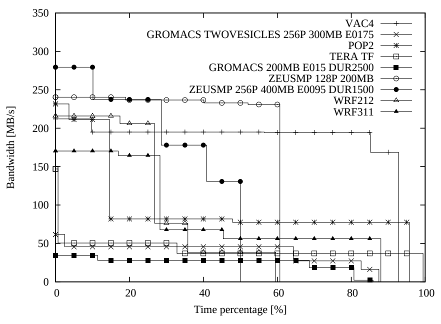
Figures 2(a) and 2(b) show the off-chip memory bandwidth measured: Figure 2(b) shows the results for benchmarks classified as having low memory bandwidth requirements, whereas Figure 2(a) shows the results for benchmarks classified as bandwidth intensive.

The figures show that the low-bandwidth benchmarks experience a typical off-chip memory bandwidth of 50–200 MB/s, whereas the high-bandwidth group typically requires between 100 and 400 MB/s, with peaks reaching as high as 700 MB/s. But as these values represent the per-processor average, they are likely to scale linearly when processors are consolidated on a single chip. Placing 100 processors on a chip is therefore likely to require sustained off-chip memory bandwidth of 10 GB/s to 40 GB/s, and may even peak to 70 GB/s.

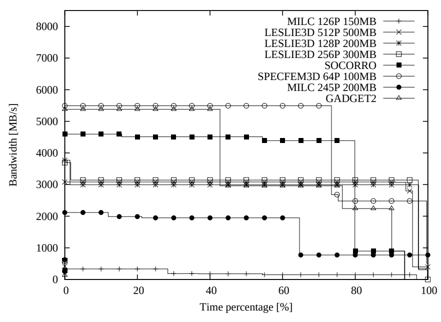
Achieving such bandwidth with existing technology is feasible, but at a cost [5]. Increasing the bandwidth of the chip-to-memory bus typically requires either increasing memory channel frequencies, or its bit width. Increasing the frequen-



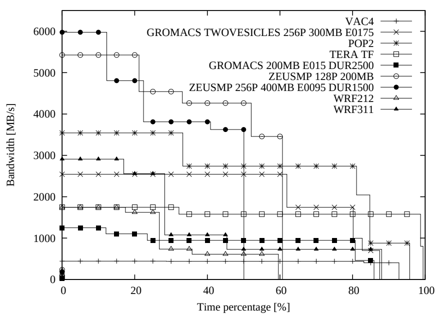
(a) Main memory: high-bw group



(b) Main memory: low-bw group



(c) L2 cache: high-bw group



(d) L2 cache: low-bw group

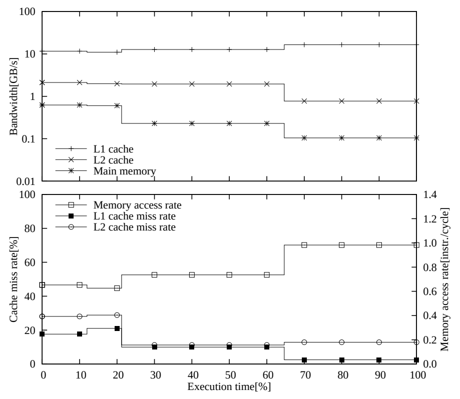
Fig. 2. Bandwidth requirements

cies hinders signal integrity, which implies shorter wires that can only support smaller memory capacities due to the smaller physical area they can cover. On the other hand, increasing the number of channels (or channel width) requires a matching increase in the number of processor pins, which results in more expensive processors: more pins imply larger processors, which are more costly (fewer processors per wafer). In addition, operating multiple channels and DIMMs in parallel greatly increases the power consumption of the memory system.

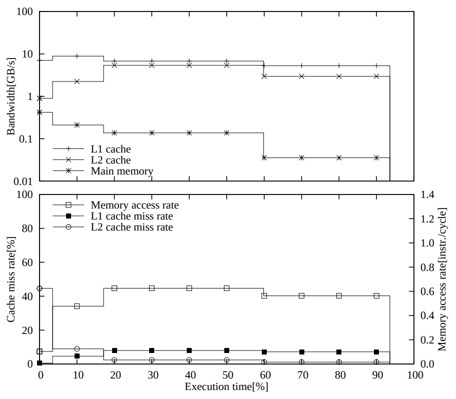
For example, the fastest configuration of the DDR3 DRAM technology achieves 1600 MT/s per 64 bit data channel, providing 12.8 GB/s per channel, with each memory interface controller (MIC) supporting up to 4 channels. Indicatively, modern architectures such as the Intel Nehalem-EX [8] employs 4 channels, whereas the IBM Power7 [6] employs twice as many DDR3 channels, peaking at 102.4 GB/s of data (the sustained bandwidth is typically 20%–25% lower because of page and bank conflicts).

Failing to provide such high memory bandwitdh would definitely have an impact on performance. For example, if an execution cluster requires 600 MB/s per processor, and only 300 MB/s is available, in the worst case scenario where a cluster is completely bandwidth bounded, we can expect its execution time to be twice as long. Clusters that require less bandwidth than provided should remain unaffected. However, this is not something that we experimentally tried to prove.

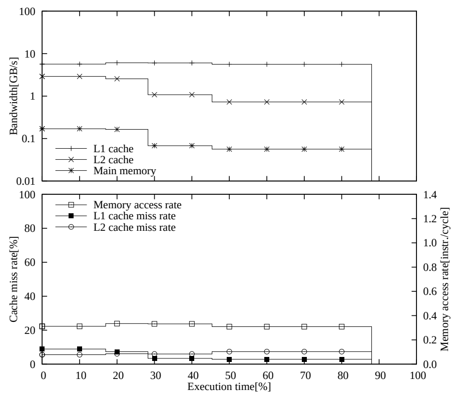
Compared with the off-chip bandwidth, the observed L2 cache bandwidth is an order of magnitude higher. This is understandable as the L2 cache hits filter bandwidth that would otherwise go off-chip (the same conclusion would apply for L1 vs. L2 bandwidth). To better understand effectiveness of the caches, as well as variations in measured bandwidths between particular clusters, we have investigated three workload related metrics: frequency of memory accesses (the number of instructions per memory access) and the L1 and L2 miss rates.



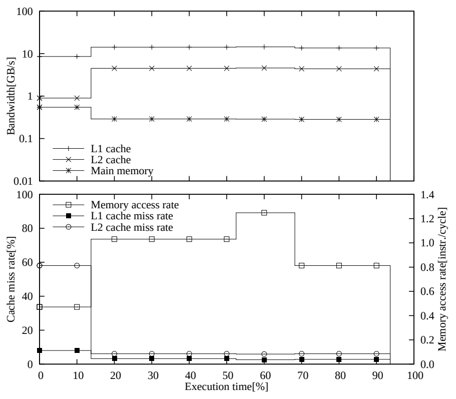
(a) MILC



(b) GADGET



(c) WRF



(d) SOCORRO

Fig. 3. Memory bandwidth requirements of selected applications, and their associated workload metrics.

Figure 3 depicts the above metrics for MILC 3(a), GADGET2 3(b), WRF 3(c) and SOCORRO 3(d), respectively. Each figure is split into two sub-plots — the top plot presents the bandwidth for the main memory, and L2 and L1 caches bandwidth, and the bottom plot shows the frequency of memory accesses, L2 and L1 miss rates of the respective clusters. Both sub-plots have the same X axis, so that bandwidth variations can be correlated with variations in cache miss rates and/or frequency of memory accesses.

For MILC, the first significant change in bandwidth is observed between second and third cluster, as the memory bandwidth decreases (note the logarithmic bandwidth scale), despite the increase in memory access frequency. This is explained by the reduction in L1 and L2 miss-rates, which indicates that more data is fetched from the caches (mainly the L1, whose bandwidth increases), suggesting the third cluster experiences better data locality. A similar trend is observed between the third and forth cluster, in which the L1 bandwidth increases even further indicating a much higher data locality.

A similar behavior can be seen in WRF, when transitioning between second and third cluster, even though in this case we do not see an increase in memory access frequency.

For GADGET2, we observe the effects of different efficiencies of L2 cache. Between the first and second cluster, we see the a large increase in L2 bandwidth accompanied by a dramatic decrease in L2 miss-rate (and a similar, yet less noticeable effect for L1), again suggesting better data locality as L2 hits filter away more of the memory bandwidth. This trend continues between the second and the third cluster. Finally, in the fourth cluster, both L1 and L2 cache miss rates remain on the same level, while frequency of the memory accesses decreases, which leads to decrease in bandwidth demands of all L1, L2 cache and main memory.

Finally, for SOCORRO we see a big increase in frequency of memory accesses between the first two clusters. The increase large enough that even though the L1 cache miss-rate decreases slightly, it cannot fully absorb the larger number of memory accesses. Still, the L2 manages to capture most of the memory access frequency, as attested by its decreased miss-rate. As a result, both L1 and L2 bandwidth increase, and the main memory bandwidth decreases.

The results suggest that despite the fact that memory bandwidth requirements are seemingly high, in many cases data locality is substantial enough such that the caches capture most of the traffic. Our evaluation suggests that for CMPs consisting of up to ∼100 processors, aggressive on-chip caching may suffice in bridging the gap between the memory bandwidth required by parallel applications and the effective off-chip bandwidth supported by current memory technologies. However, our observations also suggest that the tipping point lurks at ∼200 processor on a chip, at which point existing memory technologies will not be able to provide applications with sufficient memory bandwidth.

3.2 Memory footprint

In this section, we try to quantify the memory footprint of parallel applications, as a key factor determining the size requirements of both on-chip and off-chip memory in future CMPs.

Technological constraints not only limit off-chip memory bandwidth, but also the amount of high-bandwidth memory that can be connected to a CMP. Maintaining high-frequency signal stability poses strict limitations on the distance between the memory controller and the DRAM DIMMs, and — in the case of an off-chip controller — on the distance between the main processor and the memory controller. The limited distance in turn, restricts the board area on which DIMMs can be placed, and thereby limits the number of DIMMs connected to a single CMP. A promising venue to overcome this hurdle is 3D-stacking of DRAM layers on a single chip. But while this can greatly increase the capacity of DRAM chips, it is also limited by the number of layers that can be efficiently stacked on a single chip, the capacity of each layer, and last but not least, the power consumption of the added DRAM.

To evaluate the required memory size, we have measured the per-processor memory footprint for the four applications analyzed above, namely MILC, GADGET2, WRF and SOCORRO, running on 16, 32, and 64 processors. Results for GADGET2 are shown for runs on 32, 64, and 128 nodes, since running the application on 16 processors require more than the 8 GB per-processor, which is the amount of memory installed on each node in MareNostrum.

Application	#Procs.	Avg. mem. footprint (per-proc.)	Max. mem. footprint (per-proc.)	Footprint reduction (w/2x procs.)	Est. tot. footprint (pessimistic)
MILC	64	0.30 GB	0.31 GB	-33%	19.20 GB
	32	0.45 GB	0.48 GB	-38%	14.40 GB
	16	0.73 GB	0.80 GB	N/A	11.68 GB
GADGET2	128	0.52 GB	0.68 GB	-32%	66.56 GB
	64	0.77 GB	1.00 GB	-42%	49.28 GB
	32	1.32 GB	1.83 GB	N/A	42.24 GB
WRF311	64	0.22 GB	0.29 GB	-19%	14.08 GB
	32	0.27 GB	0.34 GB	-23%	8.64 GB
	16	0.35 GB	0.41 GB	N/A	5.60 GB
SOCORRO	64	0.23 GB	0.24 GB	-12%	14.72 GB
	32	0.26 GB	0.28 GB	-24%	8.32 GB
	16	0.34 GB	0.35 GB	N/A	5.44 GB

Table 2. Memory footprint data for selected application.

Table 2 shows the average and maximum per-processor footprints of all executions, whereas Figure 5 describe the progression of average per-processor memory for the four applications. The figures' vertical axis shows the memory footprint in GB, and the horizontal axis depicts the progression of normalized execution time.

As expected, the per-processor memory footprint decreases as the number of processors participating in the computation increases. However, doubling the number of processors does not halve the size of the per-processor memory footprint. For both WRF3 and SOCORRO, doubling the number of processors only reduces the per-processor memory footprint by $\sim$15%. Scaling is somewhat bet-

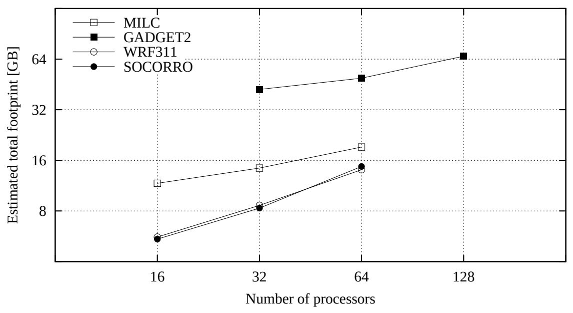


Fig. 4. Estimated total footprint for selected application.

ter for GADGET2, in which case going from 32 to 64 processors reduces the per-processor footprint by 42%, but scaling further to 128 processors only reduces the footprint by 32% more. Figures for MILC are fairly similar to those of GADGET2.

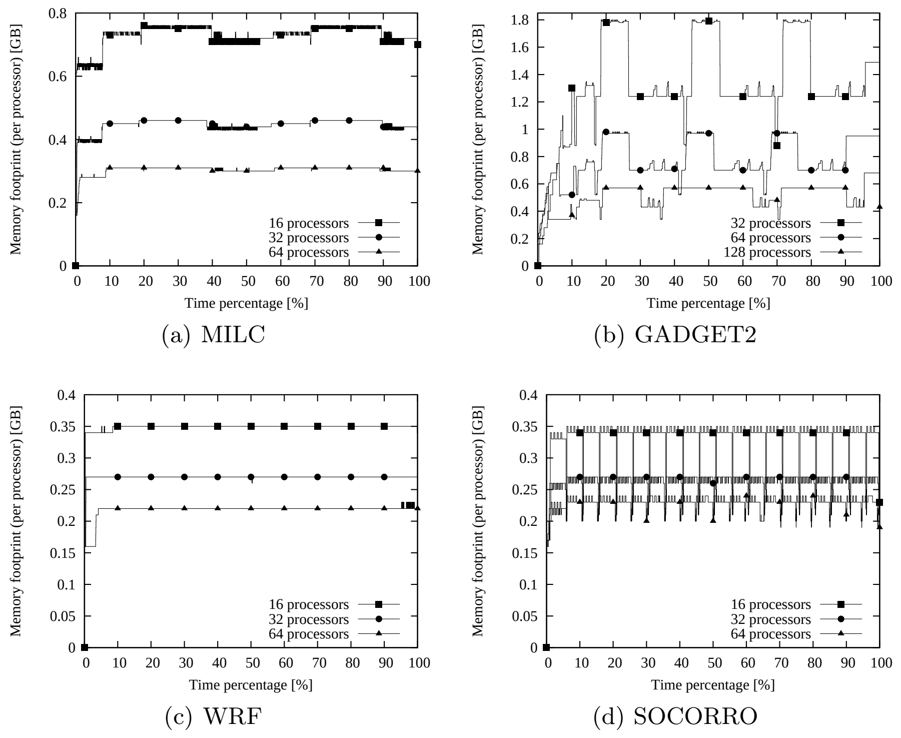
This suboptimal reduction in per-processor memory footprint is partially an artifact of using distributed memory applications, and is caused by the replication of data between processors. Many distributed algorithms that employ spatial metrics to partition the entire problem set into segments, replicate segment borders among processors assigned with neighboring segments (for example, partitioning a large matrix into small sub-matrices will typically involve replication the rows and columns on segment borders). Therefore, reducing the per-processor segment size will inevitably increase the portion of the segment that constitute as part of its border — that is replicated on the processor assigned with the neighboring segment, and thus increases the percentage of application data that is replicated among the nodes.

Our estimate of the total memory footprint of the applications, shown in Table 2, is therefore pessimistic and is based on the per-processor footprint of the least parallel execution (which has the lowest sharing ratio).

The amount of data replication identified in the discussed benchmarks also supports our projections about the usefulness of caching: even though each processor participating in a parallel computation needs to access data originally assigned to its neighbor, aggressive caching can capture such data sharing patterns and prevent them from going off-chip, thereby saving precious memory bandwidth.

Despite data replication, it seems that a per-processor memory footprint of ~300 MB is not uncommon. We therefore predict we will continue to see such per-processor footprints, particularly in multiprogrammed CMPs that consolidate multiple parallel applications on a single chip. As a result, we project that CMPs with 100 cores will require some 30 GB of off-chip memory. Such size are certainly within the capacity of today's technology.

In conclusions, we project that consolidating ~100 processors onto a single chip can be supported using today's memory technology. However, we predict



(a) MILC (b) GADGET2

(c) WRF (d) SOCORRO

Fig. 5. Evolution of the memory footprint throughout the application runtime.

the tipping point to be located at ∼200 processors (much like that of the memory bandwidth), for which denser memory technologies are required.

4 Related work

The dramatic decrease in main memory costs over the recent 15 years have reduced scientific interest in characterizing and optimizing the memory requirements of high-performance parallel applications. As a result, these requirements remained known only to parallel application developers, and has not made it into public knowledge. Published work therefore focused on characterizing established benchmarks suites, such as Woo et al.'s work on the SPLASH-2 benchmarks [12].

More importantly, little is known about the memory bandwidth requirements of such applications, and the few studies that explored this topic thus stand out.

Alam et al. studied the effects of data placement among DRAM DIMMs affects memory performance [2].

Liu et al. developed the memory intensity metric to evaluate the load imposed by parallel applications on off-chip memory bandwidth of CMPs [9]. Memory intensity was defined as the number of bytes transferred to and from the chip per executed instruction, thereby taking into account data locality that is captured by the on-chip cache.

Finally, Bhadauria et al. explored the effects of different hardware configuration on the perceived performance of the PARSEC benchmarks [3]. In order to evaluate how memory bandwidth affects the performance of PARSEC benchmarks, the authors reduced the frequency of the DRAM channels connected to a 4-way CMP, and concluded that memory bandwidth is not a limited resource in this configuration.

In contrast to the above, we focus on the potential of large-scale CMPs to serve as a supercomputing infrastructure, by characterizing common highly parallel scientific applications, and projecting how current technology can scale to meet their demands.

5 Conclusions

In this paper we have attempted to make a projection of the memory system requirements of a future integrated chip multiprocessor based on current applications running on message-passing multiprocessor clusters. That is, we have projected what would happen if we integrated a current cluster in a single chip, based on the applications that we run on that cluster today.

We must bear in mind that these are production applications that run in MareNostrum at the Barcelona Supercomputing Center, and so have been optimized to fit in the on-chip cache of the PowerPC 970MP and the 8 GB blade (2 GB per processor) that our cluster provides. It is a matter of speculation how the applications would change (and so their requirements) if the system had more memory, more on-chip cache, or more memory bandwidth. Answering that question would require an extensive programming and tuning effort beyond the scope of this paper.

When scaling up a computer system we must distinguish two different scenarios: First, by increasing the number of processors and keeping the problem size constant, we are aiming at reducing the computation time. Second, by increasing the number of processors, we can now tackle a bigger problem in the same amount of time.

An example of the first case would be computing an MRI image in minutes instead of hours to show the results to the patient in the same visit. An example of the second case would be making a weather forecast with a grid resolution of 10 meters instead of a resolution of 1 kilometer to achieve per-street predictions.

The results and conclusions we have obtained in this paper are limited to the first case: the problem size is fixed while the system performance increases. Under such scenario, it seems that current memory technologies will be stressed to handle the future CMPs, but it is still doable.

When we move to the second scenario, the increased problem size could not fit in the available DRAM memory, pushing us towards denser memory technologies. However, denser memory technologies do not offer the bandwidth that is required by such multicores. To make the problem worse, the increased working set may overflow the on-chip caches, leading to an even higher demand on memory bandwidth that could not be satisfied even with DRAM technology.

Given our results, we observe that the next generation of multicore chips will be usable as faster versions of today's processors under current memory technologies. However, they will be pushing such memories to their limit. We conclude that next generation supercomputer systems require research on new memory architectures capable of offering both capacity and bandwidth beyond what current DRAM-based designs offer.

Acknowledgments

This research is supported by an FPI research grant (BES-2008-004599) and the Consolider contract number TIN2007-60625 from the Ministry of Science and Innovation of Spain, the European Network of Excellence HIPEAC-2 (ICT-FP7-217068), the ENCORE project (ICT-FP7-248647), the TERAFLUX project (ICT-FP7-249013), and the IBM-BSC MareIncognito project. Y. Etsion is supported by a Juan de la Cierva Fellowship from the Spanish ministry of science.

References

1. Agarwal, A., Bao, L., Brown, J., Edwards, B., Mattina, M., Miao, C.C., Ramey, C., Wentzlaff, D.: Tile processor: Embedded multicore for networking and multimedia. In: Hot Chips (Aug 2007)
2. Alam, S.R., Barrett, R.F., Kuehn, J.A., Roth, P.C., Vetter, J.S.: Characterization of scientific workloads on systems with multi-core processors. In: Intl. Symp. on Workload Characterization. pp. 225–236 (Oct 2006)
3. Bhadauria, M., Weaver, V.M., McKee, S.A.: Understanding parsec performance on contemporary cmps. In: Intl. Symp. on Workload Characterization (Oct 2009)
4. Gonzalez, J., Gimenez, J., Labarta, J.: Automatic detection of parallel applications computation phases. Parallel and Distributed Processing Symposium, International 0, 1–11 (2009)
5. Jacob, B., Ng, S.W., Wang, D.T.: Memory Systems: Cache, DRAM, Disk. Morgan Kaufmann, Burlington, MA, USA (2008)
6. Kalla, R., Sinharoy, B., Starke, W.J., Floyd, M.: Power7: IBM's next-generation server processor. IEEE Micro 30, 7–15 (2010)
7. Kongetira, P., Aingaran, K., Olukotun, K.: Niagara: A 32-way multithreaded Sparc processor. IEEE Micro 25, 21–29 (2005)
8. Kottapalli, S., Baxter, J.: Nehalem-EX CPU architecture. In: Hot Chips (Aug 2009)
9. Liu, L., Li, Z., Sameh, A.H.: Analyzing memory access intensity in parallel programs on multicore. In: Intl. Conf. on Supercomputing. pp. 359–367 (2008)
10. Marc Casas, R.M.B., Labarta, J.: Automatic structure extraction from mpi applications tracefiles. In: Lecture Notes in Computer Science. pp. 3–12 (Aug 2007)
11. Seiler, L., Carmean, D., Sprangle, E., Forsyth, T., Dubey, P., Junkins, S., Lake, A., Cavin, R., Espasa, R., Grochowski, E., Juan, T., Abrash, M., Sugerman, J., Hanrahan, P.: Larrabee: A many-core x86 architecture for visual computing. IEEE Micro 29(1), 10–21 (2009)
12. Woo, S.C., Ohara, M., Torrie, E., Singh, J.P., Gupta, A.: The SPLASH-2 programs: characterization and methodological considerations. In: Intl. Symp. on Computer Architecture. pp. 24–36 (1995)