Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling *

Brian Rogers†, Anil Krishna‡, Gordon Bell‡, Ken Vu‡, Xiaowei Jiang†, Yan Solihin†

†Dept. of Electrical and Computer Engineering
North Carolina State University
{bmrogers, xjiang, solihin}@ece.ncsu.edu

‡IBM
Systems and Technology Group
{krishnaa, gbbell, kenvu}@us.ibm.com

ABSTRACT

As transistor density continues to grow at an exponential rate in accordance to Moore's law, the goal for many Chip Multi-Processor (CMP) systems is to scale the number of on-chip cores proportionally. Unfortunately, off-chip memory bandwidth capacity is projected to grow slowly compared to the desired growth in the number of cores. This creates a situation in which each core will have a decreasing amount of off-chip bandwidth that it can use to load its data from off-chip memory. The situation in which off-chip bandwidth is becoming a performance and throughput bottleneck is referred to as the *bandwidth wall* problem.

In this study, we seek to answer two questions: (1) to what extent does the bandwidth wall problem restrict future multicore scaling, and (2) to what extent are various bandwidth conservation techniques able to mitigate this problem. To address them, we develop a simple but powerful analytical model to predict the number of on-chip cores that a CMP can support given a limited growth in memory traffic capacity. We find that the bandwidth wall can severely limit core scaling. When starting with a balanced 8-core CMP, in four technology generations the number of cores can only scale to 24, as opposed to 128 cores under proportional scaling, without increasing the memory traffic requirement. We find that various individual bandwidth conservation techniques we evaluate have a wide ranging impact on core scaling, and when combined together, these techniques have the potential to enable super-proportional core scaling for up to 4 technology generations.

Categories and Subject Descriptors

C.4 [**Computer Systems Organization**]: Performance of Systems—*modeling techniques*

*Jiang and Solihin are supported in part by NSF Award CCF-0347425.

ISCA'09, June 20–24, 2009, Austin, Texas, USA.

General Terms

Performance, Measurement

1. INTRODUCTION

In recent years, a variety of concerns – power and thermal issues, instruction-level parallelism (ILP) limits, and design complexity, among others – have driven a shift in focus away from uniprocessor systems to Chip Multi-Processor (CMP) designs. CMPs help alleviate many of these uniprocessor scaling barriers by providing the potential for throughput and performance gains with an increasing number of on-chip processor cores each process technology generation. Ideally, designers would like to extract performance and throughput gains from CMPs proportional to the increase in the number of cores at each generation. However, one of the major obstacles to this goal is limited bandwidth to off-chip memory, since each additional core generates additional cache misses which must be serviced by the memory subsystem.

In general, doubling the number of cores and the amount of cache in a CMP to utilize the growing transistor counts results in a corresponding doubling of off-chip memory traffic. This implies that the rate at which memory requests must be serviced also needs to double to maintain a balanced design. If the provided off-chip memory bandwidth cannot sustain the rate at which memory requests are generated, then the extra queuing delay for memory requests will force the performance of the cores to decline until the rate of memory requests matches the available off-chip bandwidth. At that point, adding more cores to the chip no longer yields any additional throughput or performance.

Due to factors such as pin-limitations, power constraints, and packaging costs, memory bandwidth scaling typically lags significantly behind transistor density scaling described by Moore's Law. The ITRS Roadmap [14] predicts that pin counts will increase by about 10% per year whereas the number of on-chip cores is expected to double every 18 months. The net result is that the rate at which memory traffic is generated by an increasing number of cores is growing faster than the rate at which it can be serviced. Similar to the memory wall and power wall problems faced by previous generations of systems, today's designers also face a *bandwidth wall*; total system performance and throughput are increasingly limited by the amount available off-chip bandwidth.

This work tries to answer two fundamental questions: (1) *to what extent does the bandwidth wall problem restrict future multicore scaling*, and (2) *to what extent are various*

bandwidth conservation techniques able to mitigate the bandwidth wall problem. To answer these questions, we propose a simple, yet powerful analytical model that projects how the generated memory traffic is affected by the workload characteristics, number of cores on a chip, the total cache size, and the allocation of die area for cores vs. caches. Then, we extend the model to incorporate the impact of various bandwidth conservation techniques on the memory traffic generated by the cores, and we evaluate the effectiveness of each technique in allowing multicore scaling in future systems.

Using the model, we arrive at several interesting findings:

- Without bandwidth conservation techniques, the bandwidth wall problem is multiplying at an astounding rate. For example, if the starting configuration has eight cores and 50% die area allocated to cores and 50% to caches, in four technology generations the allocation for caches must grow to 90% (vs. 10% for cores) so that the caches become large enough to reduce the total memory traffic per core such that the total traffic for all cores remains unchanged. Consequently, the number of cores would increase from 8 to only 24 (3× increase), versus 128 cores (16× increase) under "proportional" scaling.

- Bandwidth conservation techniques vary greatly in their ability to reduce the memory traffic requirement. For example, using DRAM caches allows the number of cores to increase to 47 in four technology generations, whereas leveraging smaller sized cores to gain more chip area for caches has a much more limited benefit to core scaling.

- Techniques which have a direct effect on the memory traffic itself offer significantly more benefit than techniques that reduce memory traffic indirectly by reducing cache miss rates. For example, in four technology generations, link compression can enable 38 cores while cache compression can enable only 30.

- When several bandwidth conservation techniques are combined, some of them allow super-proportional scaling. For example, using the combined techniques of 3D-stacked cache, DRAM caches, cache+link compression, and ideally-sized cache lines, in four technology generations, we can increase the number of cores on a chip to 183 (71% of the die area) while still maintaining the bandwidth envelope. Considering this, it is likely that the bandwidth wall problem can be delayed by several technology generations as long as multiple bandwidth conservation techniques are combined and applied effectively.

The remainder of this paper is organized as follows. Section 2 overviews related work. Section 3 discusses the major assumptions used by our model, while Section 4 describes how we model CMP memory traffic requirements, and Section 5 describes how the model is extended to predict the memory traffic requirements of different CMP configurations. Section 6 shows evaluation results and insights into the effects of a variety of bandwidth conservation techniques. Finally, we conclude in Section 7.

2. RELATED WORK

This study is motivated in part by Emma and Puzak's work presented in [24]. Our model is built upon the study by Hartstein et al. [11], that validates the long-observed $\sqrt{2}$ rule of how cache miss rate changes with the cache size. Our work extends this general model to a CMP architecture by modeling how die area allocation to cores and caches affect the off-chip memory traffic in current as well as future technology generations. In [10], the impending memory bandwidth limitations as CMP core scaling continues is noted, and the effect of the power law of cache miss rates on the relationship cache and bandwidth is discussed. Again, our work takes this idea further by developing a formal CMP memory traffic model, as well as studying the effectiveness of a variety of bandwidth conservation techniques at reducing memory traffic. Hill and Marty [12] develop a novel analytical model to describe the performance of various CMP designs. This model focuses on the performance of chips with cores of a varying number and capability on workloads with different amounts of parallelism, while our model focuses on how memory bandwidth limitations impact CMP design.

Other prior studies have also identified the bandwidth wall as a performance bottleneck for CMP systems [13] and even for high-performance single processor systems [7]. However, these studies do not quantify its effects on performance and throughput for large-scale CMP systems as we do in this work. Additionally, some studies have looked at CMP design space search [13, 19] to determine the best chip configuration (e.g. number of cores, complexity of cores, cache organization, etc.) for a target workload mix. However, these studies focus on smaller-scale CMP systems, where the memory bandwidth bottleneck is not as critical.

A more closely related study to our work is Alameldeen's PhD thesis [1], which uses an analytical model to study how to balance cores, caches, and communication to maximize IPC in a current generation CMP that employs cache and link compression. In contrast, our work focuses on modeling how memory traffic limits the the number of on-chip CMP cores across technology generations. We evaluate the impact of many bandwidth conservation techniques, including cache and link compression, and provide a comparison on their effectiveness.

3. ASSUMPTIONS, SCOPE, AND LIMITATIONS OF STUDY

To make our analytical model tractable, we make several simplifying assumptions. First, we restrict our study to CMPs with uniform cores. A heterogeneous CMP has the potential of being more area efficient overall, and this allows caches to be larger and generates less memory traffic from cache misses and write backs. However, the design space for heterogeneous CMPs is too large for us to include in our model.

We also assume single-threaded cores. The consequence of this assumption is that our study tends to underestimate the severity of the bandwidth wall problem compared to a system with multithreaded cores. This is because multiple threads running on a multi-threaded core tend to keep the core less idle, and hence it is likely to generate more memory traffic per unit time than a simpler core.

In addition, we assume that we always have threads or

applications that can run on all cores in the CMP, regardless of the number of cores on a chip. We also assume that characteristics of workload do not change across technology generations. While it is possible that future workloads are more bandwidth efficient (i.e. they generate fewer cache misses), past trends point to the contrary, as the working set of the average workload has been increasing due to the increasing complexity of software systems. This means that the future severity of the bandwidth wall problem may be worse than what we project in our study.

In addition to these assumptions, there are configurations that we use as the base, and apply throughout the study, unless mentioned otherwise. For example, we assume a configuration in which each core has a private L2 cache and threads that run on different cores do not share data with each other, but we relax this assumption when we discuss the impact of data sharing across threads. Another example is that we assume the size of cores, in terms of number of transistors, remains unchanged across technology generations, but we relax this assumption when we discuss the impact of using smaller cores in future generations.

For the scope of this study, we do not evaluate the power implications of various CMP configurations, and only qualitatively discuss the implementation complexity of the bandwidth conservation techniques we study.

Finally, the projection obtained by our analytical model is the memory traffic generated for different configurations for a constant amount of *computation work*, rather than for a constant unit of time (i.e. bandwidth). Both metrics are equal if the rate at which memory traffic is generated does not change. However, many factors affect the latter metric (bandwidth), including the features of the core, cache configuration, as well as queuing delay at the off-chip interconnect. Predicting the actual bandwidth demand complicates our model with no additional insights in the bandwidth saturated scenario that we assume. Hence, we focus on predicting the total memory traffic generated instead.

4. MODELING CMP MEMORY TRAFFIC

At the basic level, our analytical model must relate memory traffic to the workload characteristics, the number of cores on a chip, the total cache size, and the allocation of die area for cores vs. caches. To construct such a model, we can start from a simpler model that relates the cache miss rate with the cache size in a single core system.

4.1 The Power Law of Cache Misses

In a single core system, it has long been observed that the way the cache size affects the miss rate follows the *power law* [11]. Mathematically, the power law states that if m_0 is the miss rate of a workload for a baseline cache size C_0, the miss rate (m) for a new cache size C can be expressed as:

$$m = m_0 \cdot \left(\frac{C}{C_0}\right)^{-\alpha}, \quad (1)$$

where α is a measure of how sensitive the workload is to changes in cache size. Hartstein et al. in [11] validated the power law on a range of real-world benchmarks, and found that α ranges from 0.3 to 0.7, with an average of 0.5 (hence the $\sqrt{2}$ rule).

Figure 1 plots the cache miss rate of each application we evaluate normalized to the smallest cache size as a function of cache size for a single level of cache. Note that both axes are shown in logarithmic scale; the miss rate curve will follow a straight line if it obeys the power law. The figure shows that these applications tend to conform to the power law of cache miss rate quite closely. Also, note the four bold lines in the figure correspond to the power law fit for all commercial applications, for all SPEC 2006 benchmarks, for the commercial application with the smallest α (OLTP-2), and for the commercial application with the largest α (OLTP-4). The larger the α value, the steeper the negative slope of the line. The curve-fitted average α for the commercial workloads is 0.48, close to 0.5 in Hartstein et al.'s study [11], while the minimum and maximum α for the individual commercial applications are 0.36 and 0.62, respectively. The smallest α (SPEC 2006) has a value of 0.25. Note that individual SPEC2006 applications exhibit more discrete working set sizes (i.e. once the cache is large enough for the working set, the miss rate declines to a constant value), and hence they fit less well with the power law. However, together their average fits the power law well.

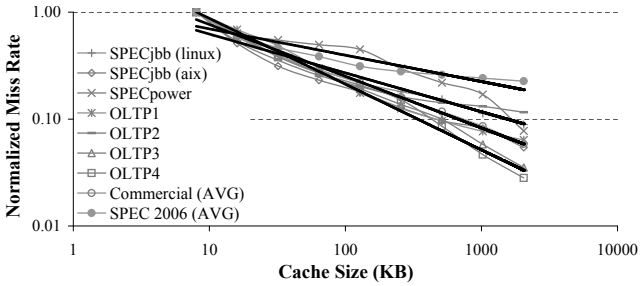

Figure 1: Normalized cache miss rate as a function of cache size.

4.2 Memory Traffic Model for CMP Systems

For our CMP memory traffic model, we extend the power law model to include the effect of additional memory traffic due to write backs, and multiple cores on a chip. We find that for a given application, the number of write backs tends to be an application-specific constant fraction of its number of cache misses, across different cache sizes. If such a fraction is denoted as r_{wb}, then the total memory traffic becomes $M = m \times (1 + r_{wb})$, where m is the miss rate. Substituting it into Equation 2, the $(1 + r_{wb})$ term appears in both the numerator and denominator, and cancel out each other, leaving us with:

$$M = M_0 \cdot \left(\frac{C}{C_0}\right)^{-\alpha} \quad (2)$$

which states that the power law holds for the memory traffic as well.

To take into account multiple cores on a chip, we first assume that the chip's die area is divided into some number of Core Equivalent Areas (CEAs). One CEA is equal to the area occupied by one processor core and its L1 caches. Additionally, we assume that on-chip components other than cores and caches occupy a constant fraction of the die area regardless of the process technology generation.

Table 1 shows the major parameters of our model which capture the fact that the die area in a CMP is allocated between cores and caches. Assuming that threads do not share data (Section 3), all cores in a CMP generate cache

Table 1: CMP system variables used in our model

CEA	Core Equivalent Area (die area for 1 core)
P	# of CEAs for cores(equivalent to # cores)
C	# of CEAs for on-chip cache
N	= P+C, total chip die area in terms of CEAs
S	= C/P, amount of on-chip cache per core

miss and write back traffic independently. Thus, the total memory traffic of the entire multicore chip is equal to the number of cores multiplied by the memory traffic generated by each core:

$$M = P \cdot M_0 \cdot \left(\frac{S}{S_0}\right)^{-\alpha} \quad (3)$$

where $S = C/P$. In order to compare the memory traffic from different CMP configurations, suppose that we have a baseline configuration with P_1 cores and S_1 amount of cache per core, and a new configuration with P_2 cores and S_2 cache per core. Dividing the expressions from Equation 3 for both configurations, we obtain:

$$\frac{M_2}{M_1} = \frac{P_2 \cdot M_0 \cdot \left(\frac{S_2}{S_0}\right)^{-\alpha}}{P_1 \cdot M_0 \cdot \left(\frac{S_1}{S_0}\right)^{-\alpha}} = \frac{P_2}{P_1} \cdot \left(\frac{S_2}{S_1}\right)^{-\alpha} \quad (4)$$

$$M_2 = \left(\frac{P_2}{P_1}\right) \cdot \left(\frac{S_2}{S_1}\right)^{-\alpha} \cdot M_1 \quad (5)$$

That is, the memory traffic of two different configurations depend on two terms: $\frac{P2}{P1}$, which accounts for the difference in the number of cores, and $\left(\frac{S2}{S1}\right)^{-\alpha}$, which depends on the change in the cache space that is allocated to each core. For example, suppose that a workload has $\alpha = 0.5$, and a baseline CMP configuration with 8 cores and 1 CEA of cache per core, for a total of 16 CEAs. If we reallocate 4 CEAs from caches to make room for four extra cores, then $P2 = 8 + 4 = 12$ and $S2 = \frac{8-4}{8+4} = \frac{1}{3}$. Using Equation 5, we find that the new configuration yields memory traffic of 2.6× more than the baseline configuration, of which 1.5× is due to the increase in the number of cores, and 1.73× is due to the increase in the memory traffic per core because of the reduced cache size per core.

5. BANDWIDTH WALL IMPACT ON CMP SCALING

The previous section described an analytical model for memory traffic as a function of die allocation to cores versus cache in the same processor generation. In this section, we will look at how the memory traffic requirement changes in future technology generations.

5.1 Bandwidth Wall Impact on CMP Scaling

To investigate the impact of the bandwidth wall on CMP scaling, we start with a baseline CMP configuration in the current process technology generation, with parameters chosen to be similar to that of Sun Niagara2 [20]. In this baseline, we assume a balanced configuration with 8 cores and 8 CEAs allocated for on-chip L2 cache (roughly corresponding to 4MB in capacity). The parameters corresponding to this baseline configuration are $N_1 = 16$, $P_1 = 8$, $C_1 = 8$, and $S_1 = 1$. We will also assume that $\alpha = 0.5$, representing the characteristic of an average commercial workload (Figure 1). We also assume that in future generations, we wish to keep the memory traffic the same as in the baseline CMP configuration.

Now, let us suppose that twice as many transistors are available in the following technology generation, and so we now have 32 CEAs of available area (assuming each core uses the same number of transistors as before). We can allocate P_2 cores, leaving $C_2 = 32 - P_2$ CEAs for cache, resulting in a cache-to-core ratio of $S_2 = \frac{32-P_2}{P_2}$. Substituting P_2 values ranging from 1 to 28 into Equation 5, we can determine how memory traffic demand changes as a function of the number of cores on a chip, normalized to the baseline system (Figure 2).

The figure shows that as we increase the number of processor cores (and reduce the area allocated for caches), the memory traffic relative to the baseline configuration grows super-linearly. If we wish to double the number of cores from the baseline to 16, then the memory traffic increases by a factor of 2. Unfortunately, off-chip pin bandwidth will not likely keep up as pin count only grows a projected 10% per year [14]. If we assume that memory traffic should remain constant in the next technology generation, then the new CMP configuration can only support 11 cores (the intersection of the two curves in Figure 2), which is equivalent to an increase of only 37.5%. Even when we assume that the bandwidth envelope can grow by an optimistic 50% in the next generation, the number of cores can only increase by 62.5% to 13. Of course, it is always possible to build more cores on a chip if one is willing to sacrifice a balanced design. Unfortunately, adding more cores beyond the bandwidth envelope will force total chip performance to decline until the rate of memory requests matches the available off-chip bandwidth. Even worse is the fact that the area taken up by excess cores could have been allocated for more productive use such as for extra cache capacity.

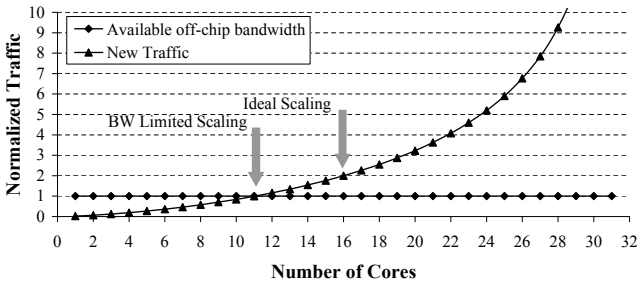

Figure 2: Memory traffic as the number of CMP cores varies in the next technology generation.

5.2 CMP Scaling and Die Area Allocation

One useful angle for chip designers to look into the bandwidth wall problem is to determine the fraction of chip area that should be allocated to cores versus cache to achieve a balanced CMP design in a future technology generation. To arrive at this answer, suppose that memory bandwidth grows by a factor of B in each technology generation, allowing the aggregate memory traffic of a next-generation CMP to grow by a factor of B relative to a current design, i.e., $M_2 = B \cdot M_1$. Incorporating this into Equation 5, we can express:

$$\left(\frac{P_2}{P_1}\right) \cdot \left(\frac{S_2}{S_1}\right)^{-\alpha} = \frac{M_2}{M_1} = B \quad (6)$$

Next, we rework Equation 6 by expressing S_2 in terms of N_2, which is known given our fixed chip area constraints, and P_2, which is what we want to find, i.e. $S_2 = \frac{N_2 - P_2}{P_2}$. Substituting this expression for S_2 in Equation 6, we obtain the following expression:

$$\left(\frac{P_2}{P_1}\right) \cdot \left(\frac{\left(\frac{N_2 - P_2}{P_2}\right)}{S_1}\right)^{-\alpha} = B \quad (7)$$

Solving for P_2 numerically, we obtain the results shown in Figure 3.

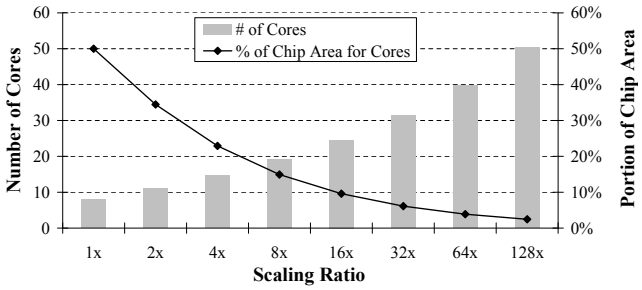

Figure 3: Die area allocation for cores and the number of supportable cores assuming constant memory traffic requirements.

Figure 3 reveals that if we keep the memory traffic constant across process generations, the fraction of the die area allocatable to cores declines rapidly. At 16× scaling (four process generations from now), only 10% of the die area can be allocated for cores, corresponding to 24 cores (versus $16 \times 8 = 128$ cores under proportional scaling). The die area allocation for cores decreases even more at future process generations.

6. ANALYZING MEMORY TRAFFIC REDUCTION TECHNIQUES

In Sections 4 and 5, our model has shown that the bandwidth wall, left unaddressed, will quickly dampen CMP scaling and force die area allocation heavily tilted toward caches. Considering the severity of the problem, we believe that future large-scale CMPs must take the bandwidth wall problem as a primary concern. The goal of this section is to evaluate the effectiveness of various bandwidth conservation techniques at addressing the bandwidth wall problem.

Recall how memory traffic is impacted by the number of cores and the amount of cache per core (Equation 5). There are two main ways to maintain a nearly-constant memory traffic requirement as we increase the number of on-chip cores proportional to the growth in transistor density. The first way is to *indirectly* reduce the memory traffic requirements by increasing the effective on-chip cache capacity, which reduces cache misses. The second way is to *directly* reduce the memory traffic requirements by either reducing the number of bytes that are brought on-chip, or by increasing the raw amount of available off-chip bandwidth. Based on this insight, we divide the discussion of various bandwidth conservation techniques into three categories: those in the *indirect* category, *direct* category, and *dual* category for techniques that produce both indirect and direct effects.

Note that the scope of our study is to evaluate existing or promising techniques that have been proposed in literature, and compare them in their effectiveness in reducing memory traffic requirements. Fully analyzing whether or when the different techniques become feasible for commercial implementation is beyond the scope of this study.

To compare the benefits of various techniques to CMP scaling, we will use the same baseline CMP as in Section 5.1, (8 cores and 8-CEA caches) and evaluate how many cores can be supported in the next generation when the number of total CEAs doubles to 32. Under proportional scaling, we would like to be able to support 16 cores, but the bandwidth wall limits it to only 11 (Figure 2).

6.1 Indirect Techniques

Indirect techniques reduce memory traffic requirements by increasing the effective cache capacity per core. In our model, the impact is captured by a multiplicative factor F, which we refer to as the *effectiveness factor*:

$$M_2 = \left(\frac{P_2}{P_1}\right) \cdot \left(\frac{\mathbf{F} \cdot S_2}{S_1}\right)^{-\alpha} \cdot M_1 \quad (8)$$

For some indirect techniques, the memory traffic equation may not conform to the simple form in Equation 8. In these cases, we develop the appropriate traffic model, as necessary.

Note that the impact of increased effective cache capacity per core (i.e. F) on memory traffic is dampened by the $-\alpha$ power. The smaller α is for a workload, the larger the dampening becomes. For example, if $\alpha = 0.9$, to reduce memory traffic by half, the cache size per core needs to be increased by a factor of 2.16×. However, if $\alpha = 0.5$, to reduce memory traffic by half, the cache size per core needs to be increased by a factor of 4×.

Cache Compression. Cache compression (e.g. [1]) exploits repeated patterns in data values to store them using fewer bits. A hardware compression engine is used to compress data before it is placed in an on-chip cache and to decompress it when the processor needs it. Prior work on cache compression has shown that a compression scheme with reasonable area and latency overheads can achieve compression ratios (i.e. effectiveness factors) of 1.4x to 2.1x for a set of commercial workloads, 1.7x to 2.4x for the SPEC2k integer benchmarks, and 1.0x to 1.3x for the SPEC2k floating point benchmarks [1, 2, 3].

Figure 4 shows the implications of this observation on the number of cores that can be supported in a next-generation CMP with 32 total CEAs, if the memory traffic requirement is kept constant relative to a baseline system with 8 cores and 8-CEA caches. Recall from Section 5 that without cache compression, our new CMP can support only 11 cores under a constant memory traffic requirement. The figure shows the number of cores possible using a cache compression scheme with various effectiveness factors (i.e. compression ratio) within the ranges observed in [1, 2, 3, 25], as well as an upper-limit case.

With cache compression support with a compression ratio of 1.3x, 1.7x, 2.0x, 2.5x, and 3.0x, the number of supportable cores grows to 11, 12, 13, 14, and 14 respectively. As CMP systems scale, this savings potentially can represent a larger raw number of additional cores that can be placed on chip. However, unless the compression ratios reach the upper end of achievable range, the ability of cache compression in helping CMP scaling is relatively modest. Note that our analysis does not factor the power, complexity, and die area overheads needed to implement the compression engine.

DRAM Cache. Another way to increase the effective

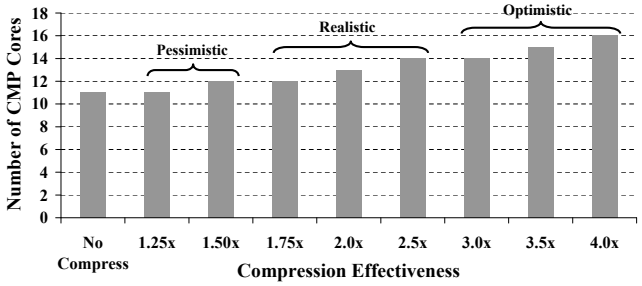

Figure 4: Increase in number of on-chip cores enabled by cache compression (32 CEAs).

cache capacity is to implement caches using denser memory such as DRAM [6, 27]. The increase in density from replacing SRAM caches with DRAM caches is estimated to be between 8× [6] and 16× [27]. Figure 5 shows the impact of changing the on-chip L2 cache from an SRAM to a DRAM technology on the number of supportable cores in the next-generation CMP under a constant memory traffic requirement.

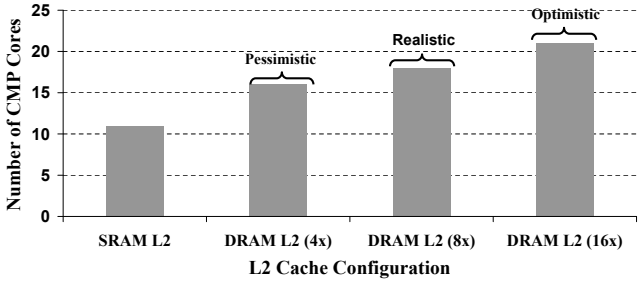

Figure 5: Increase in number of on-chip cores enabled by DRAM caches.

The bar labeled "SRAM L2" represents the baseline case in which the L2 caches are implemented in SRAM. The remaining bars show the number of cores that can be supported without increasing the memory traffic demand if DRAM technology with the above mentioned density assumptions is used to implement the L2 cache. From this figure, we can see that the desired proportional scaling of 16 cores is possible even assuming a conservative density increase of 4× in the DRAM implementation. With 8× and 16× density improvement, the situation improves further and allows super-proportional scaling to 18 and 21 cores respectively. While there are other implementation aspects to consider, such as the refresh capacity needed for DRAM or possible access latency increases, DRAM caches provide a very effective means of filtering the extra memory traffic generated by a larger number of cores, and they alleviate the bandwidth wall problem by slightly more than one technology generation.

3D-Stacked Cache. Another potential solution to achieve a large on-chip cache capacity is 3D chip stacking, where an additional die consisting of cache storage is stacked vertically on top of the traditional CMP die. We restrict our analysis to the case where there are only 2 dies stacked and the processors only occupy one die. The cache occupies the entire extra die, and, potentially, a share of the processor die. We also assume that the cache that shares the die with the processors is a traditional SRAM cache. For the extra (cache-only) die, we investigate the impact of an SRAM cache as well as DRAM cache with various density improvements over SRAM Our assumptions for this analysis are based on prior work on 3D caches and die-stacking [6, 22].

To incorporate 3D stacking into our model, we note that the extra cache-only die increases the area for cache allocation by N CEAs. The new total CEAs for cache are $N + (N - P)$. If the cache-only-layer of the stacked CMP is implemented using D×-denser DRAM cells, then the effective cache CEAs on the cache-only layer is $D \cdot N$, and the total cache CEAs for the CMP are $D \cdot N + (N - P)$. Substituting the S_2 term in the memory traffic model of Equation 5 by $\frac{D \cdot N_2 + (N_2 - P_2)}{P_2}$, we get:

$$M_2 = \left(\frac{P_2}{P_1}\right) \cdot \left(\frac{\left(\frac{D \cdot N_2 + (N_2 - P_2)}{P_2}\right)}{S_1}\right)^{-\alpha} \cdot M_1 \qquad (9)$$

Figure 6 shows the impact of adding a 3D-stacked die for cache storage on the number of supportable cores in the next generation CMP under a constant memory traffic requirement.

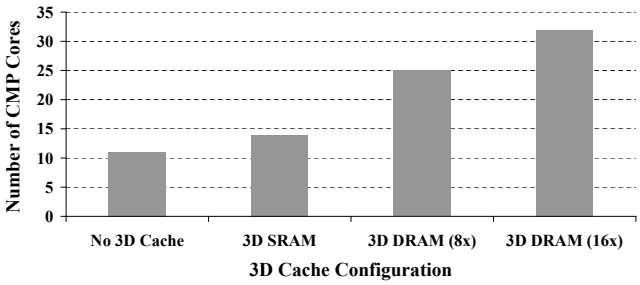

Figure 6: Increase in number of on-chip cores enabled by 3D-stacked caches.

The bar labeled "No 3D Cache" corresponds to the next-generation CMP without an additional die for caches, which can support only 11 cores. The remaining bars show how the caches in the additional die are implemented: in SRAM or DRAM with 8× or 16× density improvement over SRAM. From this figure, we can see that adding a die layer of SRAM caches allows 14 cores to be implemented, and 25 and 32 cores when DRAM caches are used with 8× or 16×, respectively. The result shows that the density improvement of DRAM caches, coupled with an entire die allocated for them, result in a highly effective way to mitigate the bandwidth wall, allowing super-proportional scaling.

Unused Data Filtering. Previous research [9, 23] has shown that a substantial amount of cached data (roughly 40%) is unused by the processor because of the mispredicted spatial locality inherent in typical cache line sizes. Unused data filtering consists of techniques that improve cache efficiency by retaining only useful words in a cache line and discarding unused words. While unused data filtering might not be designed primarily as a memory traffic reduction technique, its impact on memory traffic is significant enough to merit its inclusion in our model. There are also related studies [9, 17, 21] which reduce memory traffic directly by selectively reading in predicted-useful words of a cache line, and they will be discussed in Section 6.2 under the broad category of Sectored Caches.

Using Equation 8 and effectiveness factors for unused-data-filtering techniques from previous research [23] we can

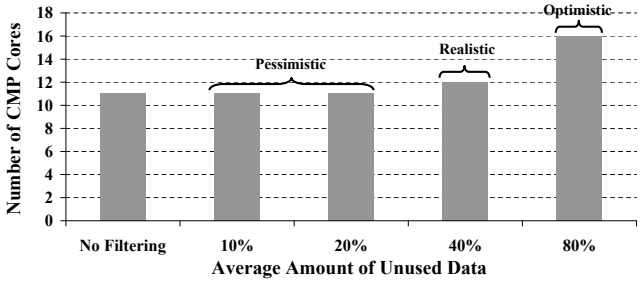

Figure 7: Increase in number of on-chip cores enabled by filtering unused data from the cache.

evaluate the sensitivity of CMP scaling to this class of techniques. Figure 7 shows that under optimistic assumptions in which 80% of cached data goes unused, corresponding to a 5× effective increase in cache capacity, a proportional scaling to 16 cores can be achieved. Under our more realistic assumptions in which 40% of cached data goes unused, the techniques provides a much more modest benefit of one additional core in the next technology generation.

Smaller Cores. One approach for allowing aggressive core scaling is to use simpler, smaller cores. One reason behind this approach is that more complex processors waste bandwidth due to speculative techniques geared towards increasing the throughput of each individual core. With less speculation, smaller cores can be more area, power, and bandwidth efficient. This efficiency allows a greater number of cores to be placed on-chip and improves the overall throughput of the chip, even as the throughput of each core is lower compared to a more complex core. However, there is a limitation to this approach. There is a certain design point which is most efficient in terms of performance per unit area and bandwidth per unit area. Any further reduction in the size of the core will yield no overall throughput benefit.

In the discussion of all the other memory traffic reduction techniques, we have assumed an area- and bandwidth-efficient base core design, such that simplifying the core further is not a good design choice. However, for sake of completeness and to draw further insights, in this section we assume that going to a smaller core leads to a more efficient design. In addition, we assume that the memory traffic requirements (not bandwidth) of the smaller core remain the same as the larger core. However, by going to smaller cores, the die area occupied by the same number of cores declines. This frees up die area that can be reallocated to increase cache capacity and reduce memory traffic. As defined in Table 1, the amount of cache per core, S, is equal to $\frac{N-P}{P}$. Let the smaller core be a fraction, f_{sm}, of the larger base core. Then the new effective amount of cache per core, S' can be expressed as:

$$S' = \frac{N - f_{sm} \cdot P}{P} \tag{10}$$

Replacing S_2 in Equation 5 by the expansion for S_2' as suggested by Equation 10 we get:

$$M_2 = \left(\frac{P_2}{P_1}\right) \cdot \left(\frac{\left(\frac{N_2 - f_{sm} \cdot P_2}{P_2}\right)}{S_1}\right)^{-\alpha} \cdot M_1 \tag{11}$$

Using Equation 11 and solving for P_2 for different values of f_{sm}, we can evaluate the sensitivity of CMP scaling to the size of the core. Figure 8 shows the number of smaller on-chip cores the next-generation chip can support. We use chip size estimates from prior work [15, 16] which shows that simpler smaller cores can be up to 80× smaller.

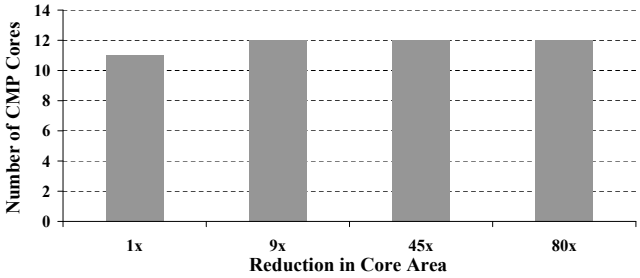

Figure 8: Increase in number of on-chip cores enabled by smaller cores.

Note that even with significantly smaller cores, the system does not scale very well even across one generation. The reason is that even when the core is infinitesimally small and the cache occupies the entire chip, the amount of cache per core only increases by 2×, whereas for proportional core scaling the cache needs to grow by 4×.

Therefore, the main reason going to a simpler core design allows a more aggressive core scaling is because simpler cores are also slower in generating memory traffic, hence they naturally fit within a lower bandwidth envelope. For example, if a simpler core is twice as slow and twice as small as a more complex one, we can fit twice as many simple cores while keeping the same bandwidth envelope. However, in such a case, the scaling is obtained directly at the cost of lower per-core performance. In addition, in practice, there is a limit to this approach, since with increasingly smaller cores, the interconnection between cores (routers, links, buses, etc.) becomes increasingly larger and more complex.

6.2 Direct Techniques

We have discussed that by increasing the effective cache capacity per core, the memory traffic demand from each core can be reduced. However, the benefit of such techniques on memory traffic reduction is dampened by the power of $-\alpha$ in Equation 5. A more promising approach is to increase the actual or effective bandwidth term, $\frac{M_2}{M_1}$, directly. To directly increase the available bandwidth, the industry has relied on (1) increasing the frequency of the off-chip memory interface, or (2) increasing the number of channels to off-chip memory. For example, Sun's Niagara2 went with Fully Buffered memory DIMMs compared to the DDR2 memory in Niagara1, increasing the peak memory bandwidth from about 25GBps to 42GBps [20]. IBM's Power6 chip doubled the number of memory controllers and memory channels on chip, and went to a faster clocked DDR2 memory compared to the Power5 chip [18] (800MHz vs. 533MHz). Unfortunately, increasing the actual bandwidth is eventually limited by the growth in pin counts and in the power and cooling requirements from a higher DRAM frequency.

Link Compression. One technique to increase the effective memory bandwidth is link compression, which reduces the amount of data communicated from/to off-chip memory by receiving/sending it in a compressed form. Previous research has shown that with relatively simple compression schemes, such as those which exploit value locality, memory bandwidth demand can be reduced by about 50% for

commercial workloads, and up to 70% for integer and media workloads [25]. In other words, link compression, can result in 2x to 3x increase in effective bandwidth. Though link compression may be applied only once, its potential impact and its orthogonality to other techniques make this promising. Figure 9 shows the number of supportable cores in a next-generation CMP assuming various compression ratios. The figure shows that proportional scaling is achievable, while a super-proportional scaling is a possibility.

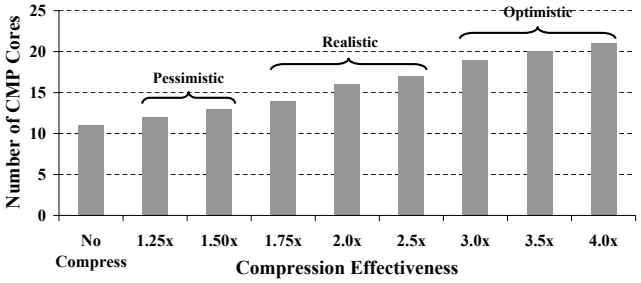

Figure 9: Increase in number of on-chip cores enabled by link compression.

Sectored Caches. As discussed in the section on Unused Data Filtering, studies have found that many words in a cache line go unused. Here, we consider techniques that exploit this finding by fetching on chip only words that are likely to be referenced [9, 17, 21]. We evaluate their potential benefit assuming that a cache line is broken into sectors, and when a cache line is requested, only sectors that will be referenced by the processor are fetched and fill the cache line. Because unfilled sectors still occupy cache space, a sectored cache only reduces memory traffic but does not increase the effective cache capacity.

Figure 10 shows the number of supportable cores in a next generation CMP, assuming various fractions of data that are unused by the processor, as in Unused Data Filtering in Section 6.1. The figure shows that Sectored Caches have more potential to improve core scaling compared to Unused Data Filtering, especially when the amount of unused data is high.

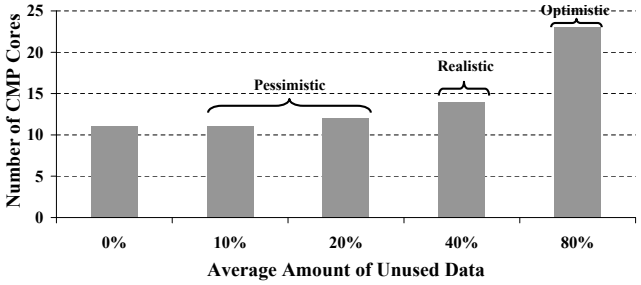

Figure 10: Increase in number of on-chip cores enabled by a sectored cache.

6.3 Dual Techniques

The previous sections explored two classes of techniques to overcome the bandwidth wall: those that directly increase the effective bandwidth at the chip interface, and those that indirectly reduce off-chip traffic through increasing the effective cache capacity. This section considers mechanisms that simultaneously achieve both of these benefits.

Small Cache Lines. Historically, larger cache lines have been used to improve application performance by banking on the existence of spatial locality for all words in a cache line. However, if not all words in a line are accessed, unused words waste bandwidth in two ways: they consume off-chip bandwidth during their transfer from the off-chip memory, and they reduce the cache capacity available to useful words while they are cached on chip. Therefore, while using smaller line sizes can increase the number of cache misses, it can reduce memory traffic both directly and indirectly, since only useful words are brought in and stored in the cache.

Prior work [9, 23] has studied the amount of useless traffic for various line sizes and concluded that, on average, 40% of the 8-byte words in a 64-byte cache line are never accessed. This represents a 40% decrease in effective cache capacity and, potentially, a corresponding 40% decrease in memory bandwidth effectiveness. Let f_w denote the average fraction of unused cache space, we can represent the new memory traffic demand from using smaller cache lines as:

$$M_2 = \left(\frac{P_2}{P_1}\right) \cdot \left(\frac{S_2 \cdot \left(\frac{1}{1-f_w}\right)}{S_1}\right)^{-\alpha} \cdot \frac{M_1}{\left(\frac{1}{1-f_w}\right)} \qquad (12)$$

where $\frac{1}{1-f_w}$ represents the factor by which the cache space increases and the memory traffic decreases. Figure 11 shows the number of supportable cores in a next-generation CMP for various fractions of unused data, with an assumption that unreferenced words consume neither bus bandwidth nor cache capacity, corresponding to the case in which word-sized cache lines are used. We point out the *realistic* case, which shows that a 40% reduction in memory traffic enables proportional scaling (16 cores in a 32-CEA). Note also that our base cache line size of 64 bytes is on the small end. Some current systems use larger L2 cache line size, in which case we would expect the effectiveness of this technique to be greater.

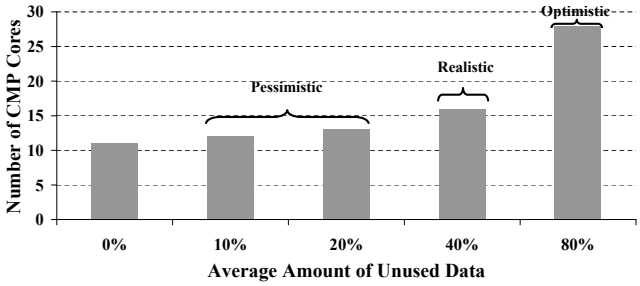

Figure 11: Increase in number of on-chip cores enabled by smaller cache lines.

Cache + Link Compression. If link and cache compression are applied together, i.e. compressed data transmitted over the off-chip link is also stored compressed in the L2 cache, memory traffic can be reduced directly and indirectly. In our model, such a combined compression is expressed by replacing the $\frac{1}{1-f_w}$ terms in Equation 12 with the achievable compression ratio. Figure 12 shows the number of supportable cores in a next generation CMP with various compression ratios. The dual benefit of direct and indirect reduction of memory traffic improves core scaling substantially. For example, even a moderate compression ratio of 2.0 is sufficient to allow a super-proportional scal-

ing to 18 cores, while higher compression ratios allow even higher scaling.

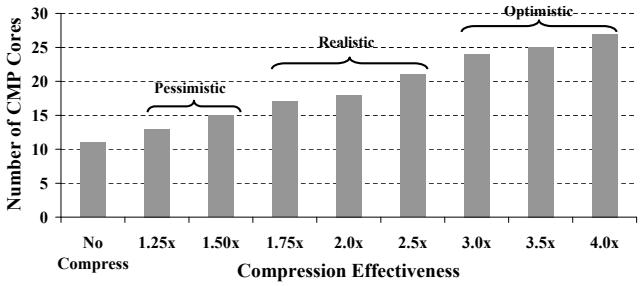

Figure 12: Increase in number of on-chip cores enabled by cache+link compression.

Data Sharing. So far we have assumed that threads that run in different cores do not share data. We recognize that parallel scientific workloads as well as multi-threaded commercial workloads may have substantial data sharing between different threads [4, 8, 26]. In this section, we relax the assumption of no data sharing, and evaluate how data sharing in multi-threaded workloads affect CMP core scaling. However, as with independent threads, we assume *problem scaling*, i.e. each additional thread brings its own working set, except for the data shared by multiple threads.

Data sharing affects our model in two ways. First, a data line that is shared by multiple threads is fetched from off-chip memory into the on-chip cache by only one thread. Secondly, how much cache capacity is occupied by a shared data block depends on the cache configuration. If we assume each core has a private L2 cache, then the block will be replicated at different caches by all cores which access it. However, if we assume the cores share the L2 cache (which may be physically distributed), the block only occupies one cache line. In order to estimate the full potential for CMP scaling due to data sharing, we assume a shared cache configuration. Thus, data sharing brings both a direct and indirect reduction in memory traffic. We also restrict our analysis to the scenario where data is either private to a thread, or is shared by all threads on the chip. This simplification, in addition to the shared cache assumption, overstates the benefit provided by data sharing in most cases, thus giving an upper bound to its effectiveness.

From the point of view of modeling, the effect of data sharing impacts the memory traffic as if there are fewer *independent* (non data sharing) cores which are generating memory requests because the first core that accesses a block suffers a cache miss, but other cores' requests result in cache hits. The effect of data sharing also appears in the cache capacity as if there are fewer independent cores which store their working sets in the shared cache, because a shared line stored in the cache by one core belongs to the working sets of several threads that run on different cores. Thus, to incorporate the effect of data sharing in our model, we define an abstract CMP configuration that has P_2' independent cores, where $P_2' < P_2$, such that our CMP power law model holds:

$$M_2 = \left(\frac{P_2'}{P_1}\right) \cdot \left(\frac{C_2/P_2'}{S_1}\right)^{-\alpha} \cdot M_1 \qquad (13)$$

where P_2' is an unknown value that must be estimated [1].

[1]If private caches are assumed, a shared block occupies multiple cache lines as it is replicated at multiple private caches. Thus, the cache capacity per core is unchanged. In the equation, $\frac{C2}{P_2'}$ should be replaced with $\frac{C2}{P2}$.

Let f_{sh} represent the average fraction of data cached on-chip that is shared by threads executing on different cores. To estimate P_2', we analyze how f_{sh} directly affects memory traffic requirement. Note that each core fetches its own private data (hence we have $(1 - f_{sh}) \cdot P_2$ fetchers), but only one core fetches the shared data (hence we have f_{sh} fetchers). Thus, there is a total of $f_{sh} + (1 - f_{sh}) \cdot P_2$ cores that fetch data from off-chip memory. However, we also know that in the abstract CMP model, we have P_2' independent cores that fetch data from off-chip memory. Since they must be equal, we can estimate P_2' as:

$$P_2' = f_{sh} + (1 - f_{sh}) \cdot P_2 \qquad (14)$$

Figure 13 shows the impact of data sharing with various fraction shared (f_{sh}) values on memory traffic requirements, normalized to our baseline CMP of 8 cores and 8-CEA caches. Each line corresponds to a future technology generation where we would like to continue proportional core scaling to 16, 32, 64, and 128 cores. An important observation from this figure is that in order to keep the memory traffic constant (at 100%) while still allowing proportional core scaling in each subsequent future technology generation, the fraction of shared data in an application must continually increase to 40%, 63%, 77%, and 86%, respectively. Even if the bandwidth envelope grows in each subsequent generation, the trend of a required increase in the fraction of data sharing in future generations still holds.

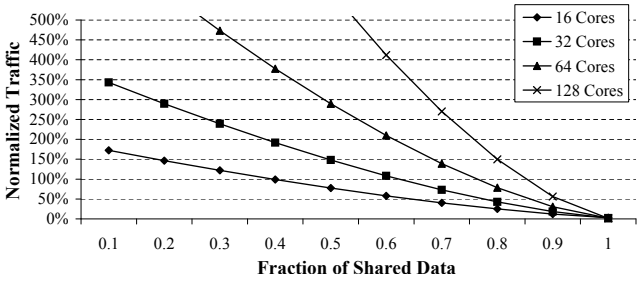

Figure 13: Impact of data sharing on traffic.

Interestingly, in practice the fraction of data sharing in an application tends to decline (rather than increase) as the application runs on more cores. To illustrate this case, we run PARSEC [5] benchmarks on a shared L2 cache multicore simulator, and measure the actual fraction of data sharing. Specifically, each time a cache line is evicted from the shared cache, we record whether the block is accessed by more than one core or not during the block's lifetime. The results are shown in Figure 14, which displays the average fraction of cache lines that are shared by two or more cores. The figure shows that the fraction of shared data in the cache tends to decrease with the number of cores, which is the opposite of what is needed to hold down the memory traffic requirement (Figure 13). A possible reason for this is that while the shared data set size remains somewhat constant, each new thread requires its own private working set [5]. Therefore, while an increase in data sharing can reduce the memory traffic requirement, it is likely that such an increase will not occur automatically and instead require a significant reworking of the algorithm or parallelization techniques.

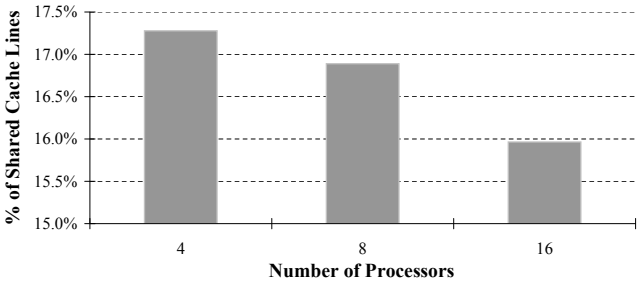

Figure 14: Data sharing behavior in PARSEC.

6.4 Discussion and Recommendations

In this section, we compare the various memory traffic reduction techniques described in Sections 6.1, 6.2, and 6.3 in their effectiveness at enabling CMP scaling in several future generations. We start with a comparison of individual techniques.

Figure 15 plots the number of supportable on-chip cores given by our model for four future technology generations (with 2, 4, 8, and 16 times the amount of on-chip transistors available), assuming a baseline of 16 available CEAs (8 for cores, 8 for cache) and memory traffic that is kept constant across generations. The *IDEAL* case shows proportional core scaling, while the *BASE* case corresponds to the number of supportable cores if no memory traffic reduction techniques are used. Each remaining configuration corresponds to one of the traffic reduction techniques. Data sharing improvement techniques are excluded because they require program, rather than architecture, changes, and they have been covered in their respective section. The labels used on the x-axis are specified in Table 2. Each data point in the figure shows a main point representing a realistic assumption, with a candle bar range representing the spread between optimistic and pessimistic assumptions (Table 2). The case of a 3D-stacked cache only shows a single point because we only consider an additional layer of SRAM die.

The figure clearly shows the growing gap between ideal versus baseline core scaling. Comparing the effect of various memory traffic reduction techniques, in general we observe that the "indirect" techniques which increase the effective amount of data that can be cached on-chip (CC, DRAM, 3D, Fltr, and SmCo) are much less beneficial than "direct" techniques which affect the actual amount of data that must be brought on-chip to execute a workload (LC and Sect), which are in turn less beneficial than "dual" techniques (SmCl and CC/LC). The reason for this is evident from our model (Equation 5): the benefit of indirect techniques is dampened by the $-\alpha$ exponent, while the benefit of direct techniques directly reduces memory traffic requirements. One exception to this general trend is DRAM storage for on-chip caches, but only because of the huge 8× density improvement over SRAM.

It is also interesting to note that the ranges of potential core scaling benefits offered by the direct techniques are quite large, depending on the effectiveness of the technique. While under realistic assumptions, no technique can outpace the ideal proportional scaling, under optimistic assumptions, small cache lines can outpace proportional core scaling for up to three future technology generations. The variability between the impact of optimistic versus realistic assumptions implies that the actual workload characteristics should be fully understood to accurately evaluate the impact of small cache lines.

Table 2 also summarizes qualitative attributes of each technique: the expected benefit to CMP core scaling (*Effectiveness*), the variability of its benefit to core scaling (*Range*), and our estimates of the cost and feasibility to implement it (*Complexity*).

As previously noted, using smaller cores as a memory traffic reduction technique has a low level of effectiveness in aiding CMP core scaling, because the die area freed up for additional cache space has a $-\alpha$ exponent dampening effect on memory traffic reduction. However, since smaller cores are slower, the memory traffic will be stretched over a longer time period, allowing them to fit the bandwidth envelope more easily, but at a direct cost to performance. We classify cache compression, 3D-stacked cache, unused data filtering, and sectored caches as techniques with a medium level of effectiveness. While these techniques also increase the effective cache capacity, compared to using smaller cores, they have the potential to increase the amount of cache per core much more drastically (up to 5×). Therefore, they provide a moderate benefit to CMP core scaling. Sectored caches directly impact memory traffic requirements by reducing the amount of data that must be fetched on-chip. However, since the fraction of memory traffic that can be eliminated is modest (40% in our realistic assumptions), it only has a medium level of effectiveness. The most effective techniques for CMP core scaling are DRAM caches, link compression, smaller cache lines, and cache+link compression. Interestingly, although DRAM caches only affect memory traffic indirectly, their huge 8× density improvement over SRAM allows very effective core scaling.

Cache compression, 3D-stacked cache, and smaller cores have a narrow range of variability, due to their small range of reasonable assumptions. This implies that the benefit of these techniques on core scaling is more predictable and does not vary much based on workload characteristics. DRAM caches, unused data filtering, and link compression have a medium variability. It is not surprising that more effective techniques have a larger variability. Finally, sectored caches, smaller lines, and cache+link compression have a large variability, and their potential benefits can vary drastically with different workload characteristics (e.g. the compressibility of data, and the degree of spatial locality). It is also interesting to note that the majority of variability for these schemes is in the positive direction, i.e. the number supportable cores with realistic assumptions is much closer to the number of cores with pessimistic assumptions. The fundamental reason is that the impact of these techniques becomes magnified as they become more effective. For example, sectored caches can reduce the memory traffic by 5× for a workload which does not use 80% of its fetched data, versus 10× for a workload which does not use 90%.

DRAM caches, smaller cores, and link compression have relatively low complexity compared to the other techniques, and in some cases they could be implemented in current-generation systems. Because of the low complexity of DRAM caches, as well as their high effectiveness and low variability, this is an extremely promising technique to enable CMP core scaling in the face of bandwidth wall. Cache compression, unused data filtering, sectored caches, cache+link compression and smaller cache lines are all likely more complex to implement. Cache compression requires a cache organi-

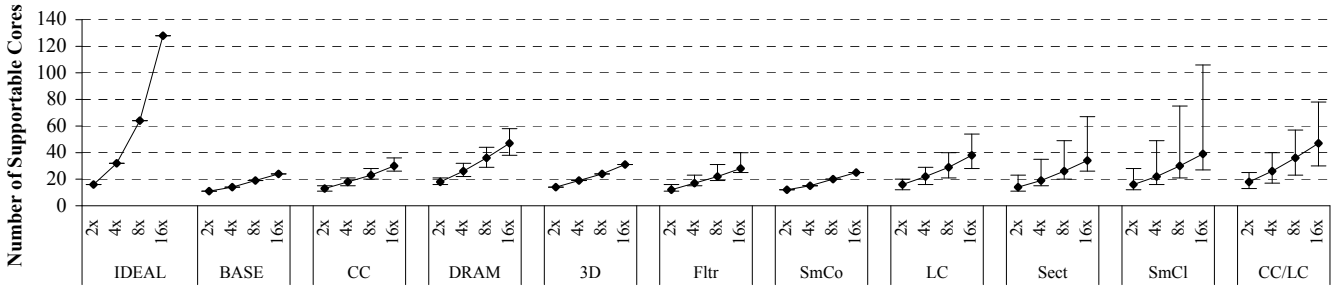

Figure 15: Core-scaling with various techniques for four future technology generations.

Table 2: Summary of memory traffic reduction techniques.

Technique	Label	Realistic	Pessimistic	Optimistic	Effectiveness	Range	Complexity
Cache Compress	CC	2x compr.	1.25x compr.	3.5x compr.	Med.	Low	Med.
DRAM Cache	DRAM	8x density	4x density	16x density	High	Med.	Low
3D-stacked Cache	3D	3D SRAM layer	-	-	Med.	Low	High
Unused Data Filter	Fltr	40% unused data	10% unused data	80% unused data	Med.	Med.	Med.
Smaller Cores	SmCo	40x less area	9x less area	80x less area	Low	Low	Low
Link Compress	LC	2x compr.	1.25x compr.	3.5x compr.	High	Med.	Low
Sectored Caches	Sect	40% unused data	10% unused data	80% unused data	Med.	High	Med.
Cache+Link Compress	CC/LC	2x compr.	1.25x compr.	3.5x compr.	High	High	Low
Smaller Cache Lines	SmCl	40% unused data	10% unused data	80% unused data	High	High	Med.

zation that handles variable-sized lines, in addition to the compression engine. Unused data filtering, sectored caches, and smaller cache lines will likely require a mechanism to accurately predict which sections of a cache line will not be referenced. Link compression seems more promising than cache compression both because of its effectiveness and its lower complexity because the cache design does not change. Lastly, we rank 3D-caches as the most complex option because it relies on a complex new technology. 3D-stacked caches on their own have a limited effectiveness and the complexity hurdle, but they become a lot more effective when combined with other techniques.

Figure 16 shows similar results as Figure 15 for various combinations of memory traffic reduction techniques, with an increasing number of techniques as we go to the right of the x-axis. This figure shows several significant insights. The rightmost case, which combines all of the highly effective techniques we evaluate, shows a very promising result: even with realistic assumptions about the effectiveness of individual techniques, super-proportional scaling is possible for all four future technology generations (183 cores vs. 128 cores with proportional scaling, at the fourth future generation). It follows that the most effective individual techniques from Figure 15 also have the most benefit when combined with other techniques. Together, the combination of link compression and small cache lines alone can directly reduce memory traffic by 70%, while the combination of 3D-stacked DRAM cache, cache compression, and small cache lines, can increase the effective cache capacity by 53×, indirectly reducing memory traffic by 84%.

Finally, Figure 17 shows the effect of different α values on CMP core scaling. Recall that α is essentially a measure of an application's sensitivity to changes in cache size. In this figure, we evaluate the minimum and maximum α values that we observed in Figure 1 on a select group of techniques starting with DRAM caches and working up through combinations of DRAM caches with several other techniques. An important observation from this figure is that α can have a large impact on core scaling in future generations. In the baseline case, a large α enables almost twice as many cores as a small α. When traffic reduction techniques are applied, the gap in number of supportable cores between a large and small α's grows even more. A small α prevents the techniques from achieving a proportional scaling, while a large α allows a super-proportional scaling.

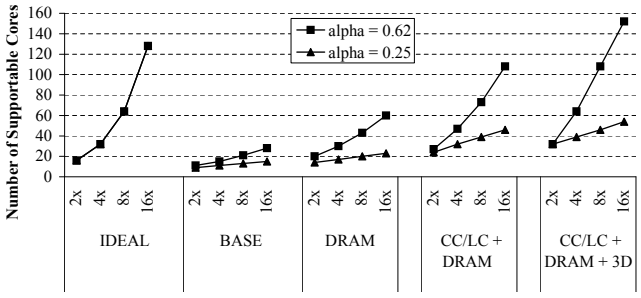

Figure 17: Core scaling with select techniques for a high and low α.

7. CONCLUSIONS

In this work we have developed a simple but powerful analytical model that allows us to study the bandwidth wall problem for CMP systems. We find that the bandwidth wall problem can severely limit core scaling. When starting with a balanced, 8 core CMP, in four technology generations the number of cores can scale to only 24, as opposed to 128 cores which is desired under proportional scaling, without increasing the memory traffic requirement. We find that the impacts of the various bandwidth conservation techniques we evaluate have wide ranging benefits to core scaling. For example, using DRAM caches can potentially enable 47 cores in four technology generations, while other techniques such

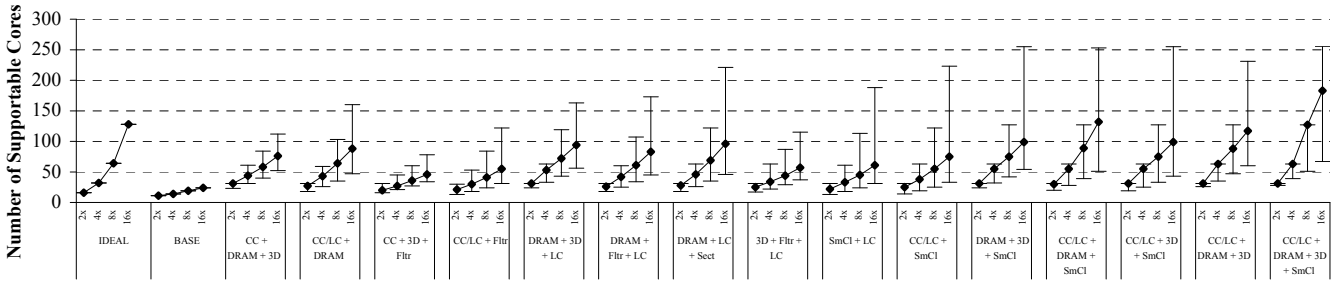

Figure 16: Core-scaling with combinations of various techniques for four future technology generations.

as using smaller cores provide a limited benefit to core scaling. Also, we find that when combined together, these techniques have the potential to enable super-proportional core scaling (183 cores for the combination of all techniques we evaluate) for up to four technology generations.

8. REFERENCES

[1] A. R. Alameldeen. *Using compression to improve chip multiprocessor performance*. PhD thesis, University of Wisconsin at Madison, Madison, WI, 2006.

[2] A. R. Alameldeen and D. A. Wood. Adaptive Cache Compression for High-Performance Processors. *SIGARCH Computer Architecture News*, 32(2):212, 2004.

[3] A. R. Alameldeen and D. A. Wood. Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches. In *Tech. Rep. 1500, Computer Sciences Department, University of Wisconsin-Madison*, 2004.

[4] L. Barroso, K. Gharachorloo, and E. Bugnion. Memory system characterization of commercial workloads. *Computer Architecture, 1998. Proc.. The 25th Intl. Symp. on*, pages 3–14, 1998.

[5] C. Bienia, S. Kumar, J. P. Singh, and K. Li. The parsec benchmark suite: Characterization and architectural implications. Tech. Rep. TR-811-08, Princeton University, 2008.

[6] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCaule, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen, and C. Webb. Die Stacking (3D) Microarchitecture. In *39th IEEE/ACM Intl. Symp. on Microarchitecture*, pages 469–479, Washington, DC, 2006. IEEE Computer Society.

[7] D. Burger, J. R. Goodman, and A. Kagi. Memory bandwidth limitations of future microprocessors. In *ISCA '96: 23rd annual international symposium on Computer architecture*, pages 78–89, New York, NY, USA, 1996. ACM.

[8] H. Cain, R. Rajwar, M. Marden, and M. Lipasti. An architectural evaluation of Java TPC-W. *High-Performance Computer Architecture, 2001. HPCA. The 7th Intl. Symp. on*, pages 229–240, 2001.

[9] C. F. Chen, S.-H. Yang, B. Falsafi, and A. Moshovos. Accurate and Complexity-Effective Spatial Pattern Prediction. In *HPCA '04: 10th Intl. Symp. on High Performance Computer Architecture*, page 276, Washington, DC, USA, 2004. IEEE Computer Society.

[10] P. G. Emma and E. Kursun. Is 3D Chip Technology the Next Growth Engine for Performance Improvement? *IBM J. Res. & Dev.*, 52(6):541–552, 2008.

[11] A. Hartstein, V. Srinivasan, T. Puzak, and P. Emma. On the Nature of Cache Miss Behavior: Is It $\sqrt{2}$? In *The Journal of Instruction-Level Parallelism*, volume 10, 2008.

[12] M. D. Hill and M. R. Marty. Amdahl's law in the multicore era. *IEEE Computer*, 41(7):33–38, 2008.

[13] J. Huh, D. Burger, and S. W. Keckler. Exploring the Design Space of Future CMPs. In *PACT '01: 2001 Intl. Conf. on Parallel Architectures and Compilation Techniques*, pages 199–210, Washington, DC, USA, 2001. IEEE Computer Society.

[14] ITRS. International Technology Roadmap for Semiconductors: 2005 Edition, Assembly and packaging. In *http://www.itrs.net/Links/2005ITRS/AP2005.pdf*, 2005.

[15] R. Kumar, D. Tullsen, N. Jouppi, and P. Ranganathan. Heterogeneous chip multiprocessors. *Computer*, 38(11):32–38, 2005.

[16] R. Kumar, D. Tullsen, P. Ranganathan, N. Jouppi, and K. Farkas. Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. *Computer Architecture, 2004. Proc.. 31st Intl. Symp. on*, pages 64–75, 2004.

[17] S. Kumar and C. Wilkerson. Exploiting spatial locality in data caches using spatial footprints . *Computer Architecture, 1998. Proc.. The 25th Intl. Symp. on*, pages 357–368, 1998.

[18] H. Q. Le, W. J. Starke, J. S. Fields, F. P. O'Connell, D. Q. Nguyen, B. J. Ronchetti, W. M. Sauer, E. M. Schwarz, and M. T. Vaden. IBM POWER6 microarchitecture. *IBM J. Res. Dev.*, 51(6):639–662, 2007.

[19] Y. Li, B. Lee, D. Brooks, Z. Hu, and K. Skadron. CMP design space exploration subject to physical constraints. In *in 12th Intl. Symp. on High Performance Computer Architecture*, 2006.

[20] H. McGhan. Niagara 2 Opens the Floodgates. *Microprocessor Report*, 2006.

[21] P. Pujara and A. Aggarwal. Increasing the cache efficiency by eliminating noise. *High-Performance Computer Architecture, 2006. The Twelfth Intl. Symp. on*, pages 145–154, 2006.

[22] K. Puttaswamy and G. H. Loh. Implementing Caches in a 3D Technology for High Performance Processors. In *ICCD '05: 2005 Intl. Conf. on Computer Design*, pages 525–532, Washington, DC, USA, 2005. IEEE Computer Society.

[23] M. K. Qureshi, M. A. Suleman, and Y. N. Patt. Line Distillation: Increasing Cache Capacity by Filtering Unused Words in Cache Lines. In *HPCA '07: 2007 IEEE 13th Intl. Symp. on High Performance Computer Architecture*, pages 250–259, Washington, DC, USA, 2007. IEEE Computer Society.

[24] Y. Solihin, F. Guo, T. R. Puzak, and P. G. Emma. Practical Cache Performance Modeling for Computer Architects. In *Tutorial with HPCA-13*, 2007.

[25] M. Thuresson, L. Spracklen, and P. Stenstrom. Memory-Link Compression Schemes: A Value Locality Perspective. *Computers, IEEE Trans. on*, 57:916–927, 2008.

[26] J. Tseng, H. Yu, S. Nagar, N. Dubey, H. Franke, and P. Pattnaik. Performance Studies of Commercial Workloads on a Multi-core System. *Workload Characterization, 2007. IISWC 2007. IEEE 10th Intl. Symp. on*, pages 57–65, 2007.

[27] T. Yamauchi, L. Hammond, and K. Olukotun. A Single Chip Multiprocessor Integrated with High Density DRAM. Tech. Rep., Stanford University, Stanford, CA, 1997.