Spring 2015

Architectural techniques to extend multi-core performance scaling

Hamza Bin Sohail
Purdue University

Graduate School Form 30
(Revised 08/14)

PURDUE UNIVERSITY
GRADUATE SCHOOL
Thesis/Dissertation Acceptance

This is to certify that the thesis/dissertation prepared

By Hamza Bin Sohail

Entitled
Architectural Techniques to Extend Multi-core Performance Scaling

For the degree of Doctor of Philosophy

Is approved by the final examining committee:

T. N. VIJAYKUMAR

MITHUNA S. THOTTETHODI

SAMUEL P. MIDKIFF

VIJAY S. PAI

To the best of my knowledge and as understood by the student in the Thesis/Dissertation Agreement, Publication Delay, and Certification/Disclaimer (Graduate School Form 32), this thesis/dissertation adheres to the provisions of Purdue University's "Policy on Integrity in Research" and the use of copyrighted material.

T. N. VIJAYKUMAR

Approved by Major Professor(s): _____

_____

Approved by: Michael R. Melloch 03/05/2015

Head of the Department Graduate Program Date

ARCHITECTURAL TECHNIQUES TO EXTEND MULTI-CORE PERFORMANCE SCALING

A Dissertation

Submitted to the Faculty

of

Purdue University

by

Hamza Bin Sohail

In Partial Fulfillment of the

Requirements for the Degree

of

Doctor of Philosophy

May 2015

Purdue University

West Lafayette, Indiana

This dissertation is dedicated to my parents.

TABLE OF CONTENTS

Page

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Sohail, Hamza Bin PhD, Purdue University, May 2015. Architectural Techniques to Extend Multi-core Performance Scaling. Major Professor: T. N. Vijaykumar.

Multi-cores have successfully delivered performance improvements over the past decade; however, they now face problems on two fronts: power and off-chip memory bandwidth. Dennard's scaling is effectively coming to an end which has lead to a gradual increase in chip power dissipation. In addition, sustaining off-chip memory bandwidth has become harder due to the limited space for pins on the die and greater current needed to drive the increasing load . My thesis focuses on techniques to address the power and off-chip memory bandwidth challenges in order to avoid the premature end of the multi-core era.

In the first part of my thesis, I focus on techniques to address the power problem. One option to cope with the power limit, as suggested by some recent papers, is to ensure that an increasing number of cores are kept powered down (i.e., dark silicon) due to lack of power; but this option imposes a low upper bound on performance. The alternative option of customizing the cores to improve power efficiency may incur increased effort for hardware design, verification and test, and degraded programmability. I propose a gentler evolutionary path for multi-cores, called successive frequency unscaling *(SFU)*, to cope with the slowing of Dennard's scaling. SFU keeps powered significantly more cores (compared to the option of keeping them 'dark') running at clock frequencies on the extended Pareto frontier that are successively lowered every generation to stay within the power budget.

In the second part of my thesis, I focus on techniques to avert the limited off-chip memory bandwidth problem. Die-stacking of DRAM on a processor die promises to continue scaling the pin bandwidth to off-chip memory. While the die-stacked DRAM

is expected to be used as a cache, storing any part of the tag in the DRAM itself erodes the bandwidth advantage of die-stacking. As such, the on-die space overhead of the large DRAM cache's tag is a concern. A well-known compromise is to employ a small on-die tag cache (*T$*) for the tag metadata while the full tag stays in the DRAM. However, tag caching fundamentally requires exploiting page-level metadata locality to ensure efficient use of the 3-D DRAM bandwidth. Plain sub-blocking exploits this locality but incurs holes in the cache (i.e., diminished DRAM cache capacity), whereas decoupled organizations avoid holes but destroy this locality. I propose *Bandwidth-Efficient Tag Access (BETA) DRAM cache* (β*$*) which avoids holes while exploiting the locality through various metadata organizational techniques. Using simulations, I conclusively show that the primary concern in DRAM caches is bandwidth and not latency, and that due to β*$*'s tag bandwidth efficiency, β*$* with a *T$* performs 15% better than the best previous scheme with a similarly-sized *T$*.

1. INTRODUCTION

For the past several decades, Moore's law has been the driving force behind the computing industry. The doubling of transistors every 2 years and increasing transistor speed at the same dollar cost delivered exponential performance improvement. In 1974, Robert Dennard presented the scaling theory for CMOS which postulated that transistors can be shrinked, and key figures of merit such as operating speed, layout density and energy efficiency can be improved as long as voltages, geometric dimensions and doping concentrations are consistently scaled to maintain constant electric field [1]. Dennard's scaling has been the major enabling factor in delivering the promise of Moore's law. During the earlier process generations, constant voltage scaling was employed but with almost a linear increase in power consumption, the industry switched to constant electric field scaling.

However, the slowing of Dennard's scaling during the last decade due to higher static power at lower threshold voltages forced the industry to move to multi-cores. Multi-cores were inevitable since uniprocessor performance improvements were only sustainable if Dennard scaling had continued. Multi-cores ushered the era in which higher performance necessitated greater parallelism in applications. Going parallel to sustain Moore's law was a major change in the landscape of computing. Multi-core performance came through exploitation of thread-level parallelism and small improvements in clock frequency. While multi-cores have successfully delivered performance improvements over the past decade, they now face problems on two fronts: power and off-chip memory bandwidth. Figure 1.1 shows chip power consumption over the years and extrapolates the effects of Dennard's scaling coming to an end. Purportedly, the imminent end of Dennard scaling will result in multi-cores hitting a utilization wall – a direct implication of the power wall. With supply voltage no longer scaling, many architects feel a significant number of transistors simply cannot be activated because

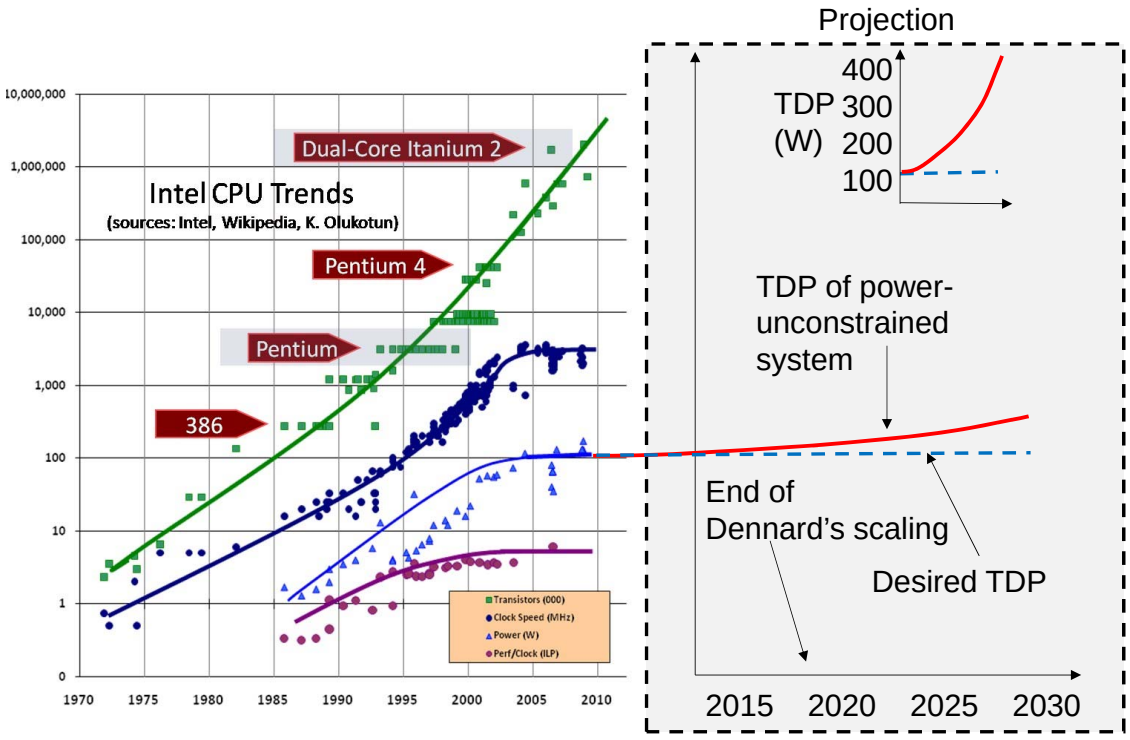

Fig. 1.1. Power projections (The trends have been extrapolated from the power trends figure courtesy of Kunle Olukotun and Lance Hammond)

of stringent power constraints.

In Chapter 2, I present an alternate evolutionary path for multi-core scaling in the absence of Dennard's scaling that can alleviate the power problem and extend the life-span of the multi-core era. I argue that architects, so far, have ignored the fact that memory plays a central role in today's applications; from databases to webservers and beyond, memory plays a crucial role in limiting the achievable performance. Because the power limits are harsh for memory-unintensive application behavior, the memory-intensive nature of a vast number of today's applications can act as a boon rather than a bane for multi-cores.

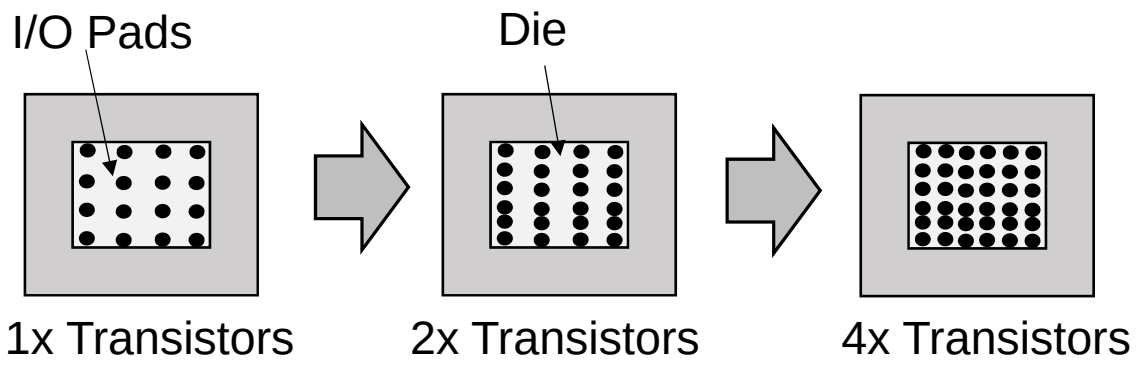

Fig. 1.2. Pin count increase with double the cores

In addition to the power conundrum, today's multi-cores have an increasing demand of memory bandwidth. Doubling the cores, even with double the cache capacity, increases the memory bandwidth demand by a factor of 2 which necessitates increasing the off-chip memory bandwidth. However, increasing off-chip memory bandwidth requires increase in pin count as well as device bandwidth. While heavy banking of DRAM will increase the device bandwidth, bus bandwidth requires an increase in pin count of the processor chip. Since die sizes do not change, it has become increasingly difficult to add more pins to increase bus bandwidth due to the limited space for pins on the die and greater current needed to drive the increasing load. 3-D Die-stacking aims to alleviate the pin-bandwidth problem. 3-D Die-stacked DRAM is stacked on top of the chip while Through-Silicon-Vias (TSVs) act as the interface between the chip and DRAM. Because the vias are on the surface of the chip and not the edges, it allows for more and wider buses. Even if DRAM is not stacked on top of the processor die, it can still offer high bandwidth by being off-die but in the same package (e.g., Intel's Haswell GT3e integrates a 128 MB DRAM in the same package). 3-D Die-stacking is meant to reduce the off-chip traffic which reduces the off-chip memory bandwidth demand. Consequently, the need to increase pin count can be avoided. However, the effectiveness of 3-D Die-stacking depends on how well its main feature (i.e., bandwidth) is used. As conventional wisdom suggests, researchers have proposed

designing 3-D Die-stacked caches that reduce the off-chip memory traffic. The primary reason for this choice is that the capacity of 3-D Die-stacked DRAM is still far less than what off-chip DDR3/DDR4 modules have to offer; it adds negligible capacity to the physical address space if it were to be added as on-package physical memory.

In Chapter 3, I explore the design challenges associated with 3-D Die-stacked caches. The main advantage of 3-D Die-stacked caches is its high bandwidth. While it may provide some latency benefits (the smaller size of arrays and shorter delays on TSVs may reduce access delay), it is really the bandwidth of 3-D-Die stacked DRAM which enables it to serve the memory demands of multi-cores. First, I conclusively show how bandwidth (and not latency) is the real feature of 3-D Die stacked caches, contrary to some recent papers which tend to argue otherwise. Second, I show the importance of metadata organization in 3-D Die-stacked DRAM in order to preserve the bandwidth advantages that 3-D Die-stacking has to offer. Chapter 3 shows that plain-subblocking, a cache design technique invented back in the 1960s, tends to exploit spatial locality in a way that helps in cutting down the bandwidth demand due to metadata accesses. However, plain sub-blocking exploits this locality but incurs holes in the cache (i.e., diminished DRAM cache capacity), whereas decoupled organizations avoid holes but destroy this locality. To satisfy these seemingly opposing contraints, Chapter 3 will describe *Bandwidth-Efficient Tag Access (BETA) DRAM cache* ($\beta\$$), a cache design which avoids holes while exploiting the locality through various metadata organizational techniques with the aim to preserve the bandwidth advantage provided by 3-D Die-stacked DRAM.

Chapter 4 wraps up the thesis with conclusions drawn from the earlier chapters, and ends with greater optimism for the future of multi-cores.

2. COPING WITH THE SLOWING OF DENNARD'S SCALING

2.1 Introduction

Historically, CMOS scaling has reduced transistor area and per-transistor dynamic power by about half and has improved switching speed by about 40% from one technology generation to the next. Specifically, Dennard's scaling of the supply voltage has allowed doubling the number of transistors without significantly worsening the dynamic power [1]. Recently, however, on one hand, Dennard's scaling has slowed down significantly due to its undesirable side-effects of higher leakage, narrower noise margins, and worse reliability (e.g., supply voltage reduces only by 2% now). On the other hand, transistor count, and hence the number of cores in a multicore, continue to double. Consequently, there is an exponential divergence between the core count and the per-core power. Unfortunately, the total chip power budget cannot be increased due to limits on cooling and power delivery (i.e., the total chip power will remain constant). These trends imply an increasing power shortage in future generations.

The imminence of *dark silicon* – silicon that must be kept deactivated because of power shortage – has been presaged[1] for some time now [2]. A recent paper [3] analyzes these trends and asserts that an increasing number of cores must be deactivated in future generations and that future multicore performance is fundamentally limited by dark silicon. I refer to the previously shown, dark-silicon induced multicore performance limit in [3] as (DSL). Alternatively, to alleviate the resulting performance loss, other papers [4–6] have suggested customizing the cores for specific functionalities to improve power efficiency and activate more cores with the same power budget. Unfor-

[1]ARM CTO Mike Muller appears to have coined the term "dark silicon". [2]

tunately, the customization option puts multicores on a potentially arduous path of (i) requiring customization to provide exponential improvements of power efficiency (i.e., every generation better than the previous), (ii) incurring increased effort for hardware design, verification, and test, and (iii) potentially degraded programmability.

In this paper, I show that DSL performance bounds can be exceeded for memory-intensive applications, and that a gentler, evolutionary path exists where customization may be optional but not essential. Previous dark silicon papers [3, 5] examine design points along the power-performance Pareto frontier covering a large space of large and small core designs and voltage-frequency-scaled operating points. The papers assert that the DSL configurations, in which a subset of the cores run at the Pareto-optimal clock speed while the rest are deactivated, achieves the best possible performance for a given technology generation and power budget. I emphasize that DSL deactivates cores and thereby bounds the *peak power* to be within the budget.

I make three key observations: First, because voltage-scaling has slowed down considerably, the Pareto frontier extends to a new region derived by frequency scaling alone. Second, because memory lags far behind processor clocks in speed, performance of most realistic workloads for future multicores will be dominated by memory latency and not processor clock speed (most future multicores with 16 or more cores are destined for servers with memory-intensive workloads). Finally, because cores wait for memory in such workloads and thereby dissipate far less power than the peak, DSL's average power is well below the budget. Our key result combines the first two observations to show that lower frequencies on the new extended Pareto frontier enable powering of more or, in many cases, all cores of a multicore which achieve more memory-latency overlap and better performance than DSL limits. I show that our results hold despite techniques for reducing, hiding, or tolerating memory latency via 3-D stacked memory, out-of-order issue, and simultaneous multithreading, respectively. While the DSL configuration bounds its peak power by deactivating cores (i.e., in space), I do so by lowering the clock frequencies (i.e., in time). However, our multicores' better performance does imply higher average power than DSL's well-below-

the-budget average. Nevertheless, as I show later in Section 2.2.2, our multicores' average power is guaranteed to be within the budget.

I arrive at our key result by exploiting two known non-linear effects, the first of which is captured by a simple analytical model for multicore performance. Our model shows that in the degenerate case of absence of memory latency, more cores running at slower clocks perform similarly to fewer running at faster clocks under the same power budget as long as the workloads are sufficiently parallel (a condition also necessary for multicores in general). In the presence of memory latency, however, more cores running at slower clocks perform *better* than fewer cores running at faster clocks. This reversal occurs because of the non-linear impact of clock speed on performance in the presence of memory latency where more active cores achieve more overlap of memory latency so that the dominant memory component of execution time reduces far more than the slight increase in the smaller non-memory component due to the slower clock. While our extra cores do incur more leakage than the DSL cores (cache capacity, and therefore cache leakage, is the same in both cases), SFU's advantage over DSL remains for memory-intensive workloads even after accounting for this extra leakage in all but extreme cases (e.g., 90% of the chip power is in leakage). However, because adjusting for the extra-leakage does modestly degrade memory-unintensive workloads, I propose to revert to the DSL configuration for such workloads. Thus, our key insight is that, for a broad range of memory-intensive commercial and scientific workloads, slow silicon is better than DSL's dark silicon as long as the slow silicon makes memory accesses. While voltage scaling has historically exploited the non-linear (cubic) relationship between power and voltage, I propose that clock-performance non-linearity be exploited in the post-Dennard era. Due to this fundamental non-linearity, DSL's performance limit can be exceeded for many realistic and important multicore workloads.

Based on our model's predictions, I propose a gentler, evolutionary path for multicores than customization, called *successive frequency unscaling (SFU)*. In SFU, more cores than DSL (and in many cases, all cores) are kept activated and run at *succes-*

sively slower clocks every generation to bridge the exponential divergence between the core count and per-core power in the post-Dennard era. While the linear relationship between power and frequency is well known, this paper is the first to propose successively slower clocks; dynamic voltage and frequency scaling (DVFS) dynamically changes the clock speed up or down for good power-performance within a technology generation but does not employ successively slower clocks from one generation to the next.

I employ SFU in two contexts with different performance metrics. In the first context of workloads where job execution time is the only metric (e.g., scientific applications), I employ full SFU wherein I unscale frequency to power all the cores. Surprisingly, despite considerably slower clocks in later generations (e.g., sub-GHz) full SFU exceeds the DSL performance limit. Not surprisingly, however, SFU does not completely close the gap between a DSL configuration and a power-unconstrained system due to the slower clock. In the other context of enterprise workloads (e.g., online transaction processing) where both throughput and response latency matter, the slower clock of full SFU would degrade single-thread performance, and hence response latency. Accordingly, I employ *controlled successive frequency unscaling (C-SFU)* which moderately slows down the clock and powers many, if not all, cores to achieve better throughput than DSL. C-SFU avoids degrading response latency despite the clock slowdown by exploiting the second non-linearity that the higher throughput of C-SFU non-linearly reduces the queuing component of response latency and thereby compensates for the slower clock. Finally, SFU's simplicity implies better performance at virtually no design effort or complexity, enabling a viable evolutionary path for multicores.

The key contributions of this paper are:

- I propose the unusual idea of successively slower cores to stay within the power budget in the post-Dennard era.

- I show that, for memory-intensive applications (which includes important commercial benchmarks) our approach can exceed the DSL's performance limits.

The key results of this paper are:

- for memory-intensive workloads, SFU performs 46% better than DSL's limits at the 11 nm technology node whereas for memory-unintensive, workloads I revert to the DSL configuration; and
- for response-time-sensitive enterprise workloads, C-SFU achieves 21% better throughput than DSL at the 11 nm technology node while maintaining the total response latency including queuing delays to be within +/- 10%.
- while out-of-order cores partially reduce opportunity for SFU by reducing the exposed memory latency, there remains ample opportunity for SFU to improve performance compared to DSL (e.g., SFU with out-of-order cores achieves 18% better performance then DSL with out-of-order cores at the 22nm technology node).

The rest of the paper is organized as follows. Section 2.2 discusses our intuition and qualitative arguments behind SFU, and then presents a simple power-performance model for multicores to provide quantitative corroboration of our intuition. Sections 2.3 and 3.5 validate our model using simulations of commercial and scientific workloads. Finally,

2.2 Multicore power and performance

I start with the intuition behind SFU followed by an analytical model.

2.2.1 Intuition

The two key claims by Esmailzadeh *et al.* [3] are that (1) the limit on performance achievable in practical multicore systems of future technology generations is

significantly lower than the performance of a power-unconstrained multicore system in the equvalent technology generation, and (2) dark-silicon is inevitable for optimal performance. Esmailzadeh *et al.*do not examine frequency scaling alone because, where both voltage and frequency scaling are possible, scaling frequency alone is not Pareto-optimal. However, in regions where further voltage scaling is infeasible, frequency-scaling alone can be used to extend the power-performance Pareto frontier. Even in this extended Pareto frontier, frequency scaling alone cannot improve upon DSL for memory-unintensive applications (i.e., applications with little exposed memory latency) because both techniques offer the same linear improvement (degradation) in performance for linear increase (reduction) in power. However, I observe that when I include the effect of exposed memory latency, the power-performance tradeoff due to frequency unscaling becomes sub-linear because the exposed memory latency does not scale. Such sublinearity is advantageous because a large reduction in frequency (which reduces power linearly) results in less-than-proportional reduction in performance. This difference in the impact of frequency scaling on dynamic power (linear) and performance (sub-linear) is central to enabling our design to achieve higher performance than the DSL limits would imply.

Recent work [7] reveals that near-threshold operation is performance-per-watt optimal for perfectly parallelizable programs. Given that the dark-silicon problem is to maximize performance under a fixed power-budget, one may think that such performance-per-watt optimality is ideal. However, near-threshold-computing's energy optimality results in very slow speed (e.g., *3-MHz* Intel Claremont). As such, even though individual cores may be performance-per-watt optimal, the system as a whole will run into other bottlenecks (e.g., area, application scalability) which can prevent the utilization of the full power budget and hence degrade performance. Further, the paper does not consider memory effects which is the main focus of our work.

Next, to support the above qualitative reasoning, I develop a simple analytical model of the combined impact of SFU's frequency unscaling and memory latency effects on overall multicore power-performance.

2.2.2 Model

Our model is derived from Amdahls' Law [8] and more recent revisits of Amdahl's Law in the context of multicores [3, 9]. However, recall from Section 3.1 that the key reason for our better performance is more cores achieving higher overlap of memory latency. Accordingly, our model specifically includes memory latency effects in addition to the usual serialization effects.

I first describe our model for a multicore that is not constrained by power. Then, I modify this model to include power constraints either via the DSL configuration or successive frequency unscaling (SFU). Let

- s be the serial portion of sequential execution time (i.e., $1 - s$ is the parallel portion);
- c be the factor by which the clock frequency improves every technology generation (e.g., if the clock speed improves by 20% then $c = 1.2$); and
- m be the fraction of sequential execution time due to memory latency (i.e., $1 - m$ is the non-memory, compute fraction).

I derive m as follows: Assuming the number of off-chip misses per kilo instructions (MKPI) is r, the per-access average exposed main memory latency is mem_{lat} in processor cycles, and the processor cycles per instruction (CPI) with 0% off-chip miss rate is $instr_{lat}$ then

$$m = r \times mem_{lat}/(1000 \times instr_{lat} + r \times mem_{lat}).$$

For example, assuming an off-chip miss rate of 2% which usually corresponds to r of 5, mem_{lat} of 400, and $instr_{lat}$ of 0.5 gives $m = 0.80$. I note that mem_{lat} denotes exposed memory latency, and hence covers both in-order- and out-of-order-issue cores though the latter's mem_{lat} and m values would be smaller than the former's.

To simplify the model, I assume that

- both the parallel and serial portions of the application incur the same fraction m of execution time due to memory latency;
- employing more parallel cores does not change the fraction m of execution time due to memory (our experimental evaluation avoids these two assumptions by using real workloads);
- the factor c is constant across generations while in reality clock speed improvements may reduce in later generations resulting in an overall average of c (our experimental evaluation uses actual, non-constant factors);
- the cores do not employ simultaneous multithreading (SMT) (I include SMT later); and
- memory bandwidth scales with the number of cores (I revisit this assumption in our results).

Defining the execution time on the generation-0 system as 1 (i.e., our normalization base), the breakdown of the total execution time may be expressed as:

$$s \times ((1-m)+m) + (1-s) \times ((1-m)+m)$$

Over n technology generations, the non-memory, compute fraction $1-m$ scales as $(1-m)/c^n$ due to clock speed improvements while the memory fraction m remains unchanged. This scaling occurs for both the serial and parallel portions so that the serial portion scales as $s \times (\frac{1-m}{c^n} + m)$ and the parallel portion scales as $(1-s) \times (\frac{1-m}{c^n} + m)$. In addition, the parallel portion gets further sped up by a factor of 2 every generation due to the doubling of the core count so that the parallel portion scales overall as

$$\frac{(1-s) \times (\frac{1-m}{c^n} + m)}{2^n}.$$

Thus, after n generations since the last uniprocessor, a power-unconstrained (PU) multicore achieves a net speedup of

$$\frac{1}{s \times (\frac{1-m}{c^n} + m) + \frac{(1-s) \times (\frac{1-m}{c^n} + m)}{2^n}} \tag{2.1}$$

I note that while the $1-m$ compute terms in both the serial and parallel portions diminish exponentially over generations due to faster clocks, the m memory term

in the parallel portion diminishes exponentially due to more cores' higher memory-level parallelism. For memory-intensive workloads, m is generally greater than $1-m$ making the number of cores more important than the clock speed for performance (from the above example, m is 0.80). As I will see shortly, this difference is the key contrast between DSL and SFU where the former keeps only a subset of the available cores powered whereas the latter keeps all the cores powered albeit at a slower clock.

For the DSL multicore which is constrained by power, let p be the factor by which the per-core dynamic power scales every technology generation due to a combination of feature size scaling, slow scaling of voltage, and transistor engineering (e.g., if power reduces by 20% then $p = 0.8$). To simplify the model, I assume that

- the dynamic power of the on-chip (non-L1) caches and network for a core's accesses are included in the core's dynamic power (an accounting simplification that does not affect the model's predictions);
- leakage is zero (I add in leakage in the next section); and
- p, like c, is constant across generations while in reality dynamic power improvements may reduce in later generations resulting in an overall average of p.

To stay within the constant power budget across generations, DSL bounds its peak power by limiting the number of cores after n generations to $(2 \times 0.5/p)^n = 1/p^n$ (i.e., bound in space). DSL differs from PU only in the number of active cores —- $1/p^n$ versus 2^n; DSL enjoys identical clock speed and last-level, shared cache size improvements, and incur similar memory latency effects. Therefore, the DSL multicore's net speedup after n generations is

$$\frac{1}{s \times (\frac{1-m}{c^n} + m) + \frac{(1-s) \times (\frac{1-m}{c^n} + m)}{1/p^n}} \tag{2.2}$$

I see that DSL exploits significantly less memory-level parallelism than PU due to fewer active cores. This limitation considerably degrades performance for realistic multicore workloads which are memory-intensive.

SFU achieves the same peak bound by successively scaling down the clock over generations so that the per-core power from one generation to the next is half allowing twice as many cores to be powered (i.e., bound in time). Let α be the factor by which the clock speed is unscaled every generation on top of the factor c provided by technology scaling, so that $\alpha \times p = 0.5$. To account for the dynamic power of the on-chip (non-L1) caches and network, this unscaling applies to those components as well. SFU differs from PU only in the scaling of the clock speeds — c^n versus $(\alpha \times c)^n$, while maintaining the same number of active cores (i.e., 2^n) and cache size. Consequently, the SFU multicore's net speedup after n generations is

$$\frac{1}{s \times (\frac{1-m}{(\alpha \times c)^n} + m) + \frac{(1-s) \times (\frac{1-m}{(\alpha \times c)^n} + m)}{2^n}} \tag{2.3}$$

Comparing DSL and SFU, I consider all four components of execution time (the denominators in the above performance expressions): serial-non-memory ($s \times (1 - m)$ terms), serial-memory ($s \times m$ terms), parallel-non-memory ($(1 - s) \times (1 - m)$ terms), and parallel-memory ($(1-s) \times m$ terms). DSL reduces the serial-non-memory component by the factor of c^n and is better than SFU which reduces by the smaller factor of $(\alpha \times c)^n$. However, this component is likely to be small for parallel, memory-intensive workloads. DSL and SFU are equal in the serial-memory and parallel-non-memory (the second and third) components. The equality in the second component is obvious. To see the equality in the third component, DSL reduces the component by a factor of p^n/c^n whereas SFU reduces by a factor of $2^n/(\alpha \times c)^n$ where $\alpha \times p = 0.5$. DSL reduces the (fourth) parallel-memory component by a factor of $(1/p)^n$ which is worse than SFU's factor of 2^n, highlighting our insight that slow silicon is better than dark silicon in the presence of memory latency (e.g., if $p = 0.8$, then DSL's and SFU's factors are 1.25^n and 2^n, respectively, giving SFU a significant advantage in memory-level parallelism). As discussed above, in memory-intensive parallel workloads, the

parallel-memory component is likely to dominate the other components, magnifying SFU's advantage. Therefore, DSL's limit can be exceeded.

Impact of leakage:

The above analysis ignores leakage power, which generally increases with lower supply voltage and higher transistor count. In the post-Dennard generations, however, leakage as a fraction of the total power budget will either remain a constant (e.g., 25-30%) or increase only slowly (e.g., under 5% per generation) due to two reasons: (1) voltage scaling has slowed down considerably and (2) the rate of growth of transistor speed over generations is reduced to compensate for the doubling of transistor count every generation (e.g., by fine-tuning the transistor threshold voltage). Recent commercial microprocessors follow this methodology to keep leakage under check [10]. Note, I assume SFU and DSL use the same fast, leaky transistors (i.e., per-transistor leakage is the same for SFU and DSL). This assumption enables us to operate a subset of processors with the same maximum frequency as DSL which is important to allow for memory-unintensive workloads.

Let l be the leakage budget as a fraction of the total power budget in the last uniprocessor generation and λ be the rate at which the total chip leakage budget increases every generation. Then, the chip leakage budget scales as $\lambda^n \times l$ after n generations (to be meaningful, $\lambda^n \times l < 1$ for any n), whereas the chip dynamic power budget scales as $(1 - \lambda^n \times l)$. To make room for leakage, this new dynamic power budget is lower than our previously-assumed full budget. Let f_l be the core leakage as the fraction of the total chip leakage, the remainder of which is the cache leakage. This scaling implies that (1) the core counts and clock frequencies for DSL and SFU, respectively, should be adjusted for this new dynamic power budget at every generation; and (2) DSL's fewer cores incur less leakage than the allotted budget (and also less than SFU), allowing more cores to be added (the DSL paper does not discuss such compensation).

Because the new, lower dynamic power budget is the same for DSL and SFU, both designs are affected similarly. Both DSL's core count for generation n, given by $1/p^n$, and SFU's clock frequency, given by $(\alpha \times c)^n$ in Equations 2.2 and 2.3, respectively, reduce by the factor $(1-\lambda^n \times l)$. Therefore, SFU maintains its advantage in memory-level parallelism, and hence performance, over DSL.

To account for DSL's leakage budget surplus, I observe that the budget for the core leakage, total dynamic power, and cache leakage scale as $f_l \times \lambda^n \times l$, $(1-\lambda^n \times l)$, and $(1-f_l) \times \lambda^n \times l$, respectively. I analyze each of these terms for DSL. Because there are 2^n cores in all, the per-core leakage in DSL is $f_l \times \lambda^n \times l/2^n$. Because $1/p^n$ DSL cores account for the full dynamic power budget, the per-core dynamic power for DSL is $(1 - \lambda^n \times l) \times p^n$. Because both DSL and SFU have the same cache capacity, the cache leakage is the same in the two designs. Therefore, assuming x DSL cores use the full power budget, the total core leakage (= per-core leakage $\times x$), total dynamic power (= per-core dynamic power $\times x$) and the cache leakage add up to 1. That is,

$$(\frac{f_l \times \lambda^n \times l}{2^n} + (1 - \lambda^n \times l) \times p^n) \times x = 1 - (1 - f_l) \times \lambda^n \times l \tag{2.4}$$

To tie leakage and performance together, the number of DSL cores in Equation 2.2 should be changed from $1/p^n$ to the value of x from Equation 2.4. To analyze Equation 2.4, I observe that the caches' large transistor counts (e.g., 75% of all on-chip transistors) are offset only partly by the fact that they can use slower, less-leaky transistors than the cores. As such, caches account for a large part of the chip leakage (e.g., the core leakage fraction f_l is 0.4). This large part forces the right hand side of Equation 2.4 not to be large. Further, compensating for DSL's leakage surplus by adding extra DSL cores adds both their small leakage and their large dynamic power (including the accompanying dynamic power for the on-chip (non-L1) caches and network). These two components make the left term of the product in the left hand side large. Consequently, x is not large (i.e., not much larger than $1/p^n$), implying that DSL's leakage surplus can accomodate only a few extra cores.

I illustrate this point with some realistic examples based on known scaling trends [4]. Assume a constant 30-70 split of the total budget for leakage and dynamic power (i.e., $l = 0.3$ and $\lambda = 1$) and a 40-60 split of the leakage budget for the cores and caches (i.e., $f_l = 0.4$). Assume generation 6 with 64 cores (i.e., $n = 6$) and DSL's $p = 0.646$ (i.e., $p^6 = 0.073$). While SFU runs all the 64 cores at a slower clock, DSL runs $1/p^6$ = 13.7 cores at the full clock speed. SFU splits a total budget of 100 as 12, 70, and 18 for the core leakage, dynamic power, and cache leakage, respectively. DSL has the same dynamic power (70) and cache leakage (18) as SFU. Because each DSL core's leakage is 12/64, DSL's total budget with 13.7 cores is $(12/64) * 13.7 + 18 + 70 = 90.57$, and not 100 as it is for SFU. Therefore, a few more DSL cores can be accomodated. Assuming x DSL cores and plugging in our values in Equation 2.4 gives $(12/64 + 70/13.7) * x + 18 = 100$, giving $x = 15.5$. DSL's leakage surplus is equivalent to only 15.5 -13.7 = 1.8 extra DSL cores. Thus, I see that SFU's advantage of memory-level parallelism remains. However, the advantage exists solely for memory intensive applications where the parallel memory component (fourth term in the denominator of Equation 2.2 and Equation 2.3) reduces with memory-level parallelism. For memory-unintensive applications, where m is negligible, DSL is better because of the additional cores due to leakage compensation.

Increasing the the total chip leakage budget from 30% to as much as 50% (i.e., $l = 0.3$ and $\lambda = 1.09$) implies that the core leakage, dynamic power, and cache leakage are 20, 50, and 30 in SFU, respectively. The DSL calculations change to $(20/64 + 50/13.7) * x + 30 = 100$, giving $x = 17.7$, or 4 extra DSL cores. Now, the number of extra DSL cores can be much larger. For example, if the total chip leakage budget goes to 90% (i.e., $l = 0.9$) and core leakage is also 90% of all leakage (i.e., f_l = 0.9), then $(81/64 + 10/13.7) * x + 9 = 100$, giving $x = 45.6$. In this extreme case, SFU has fewer than 2x cores than DSL (45.6 versus 64) but a much slower clock, so that SFU's memory-level parallelism advantage may not offset its clock disadvantage. As an aside, I note that higher leakage (from 30% to 90%) leads to more DSL cores (from 15.48 to 45.6). The dark silicon problem diminishes because dynamic power,

which is worsensed considerably by the slowing down of Dennard's scaling, is now the minor component whereas leakage, the major component, increases only slowly. In our experiments, I show the more realistic cases of 30% and 50% leakage.

Finally, I discuss a subtle point: One might think that if DSL has a 25-75 split of per-core leakage versus per-core dynamic power (this ratio is different from any of the above), then SFU having 4x more cores than DSL would imply zero dynamic power for SFU cores (Amdahl's leakage limit). While true, this limit is rarely reached. Although chip leakage is 50% in our second example above, DSL's per-core leakage versus per-core dynamic power ratio is 20/64 versus 50/13.7 or 1 versus 11.67 which is drastically different than the starting assumption of 20 in core leakage and 50 in dynamic power (or 1 versus 2.5). Because Dennard's scaling has slowed significantly while leakage stays constant or increases only slowly, DSL's per-core dynamic power far exceeds the per-core leakage in future generations, making the leakage limit irrelevant in realistic scenarios. I note that while DSL's ratio of core leakage to core dynamic power, and not SFU's ratio, is relevant for this limit, SFU's ratio in this example is 20/64 versus 50/64 or 1 versus 2.5 because SFU scales down the clock to reduce the per-core dynamic power.

SFU's peak power bound:

After adjusting for leakage, DSL and SFU have the same peak power though DSL bounds its peak power to be within the budget by deactivating cores (i.e., in space) whereas SFU does so by lowering the clock frequencies (i.e., in time). However, SFU's higher performance in memory-intensive workloads means higher average power than DSL. Nevertheless, SFU's average power is still within the budget. To validate this claim, I note that due to exposed memory latency, DSL's average power for these memory-intensive workloads falls below its peak by a factor determined by the latency. (There is no such power slack for memory-unintensive workloads.) Because SFU can at most eliminate this latency through higher memory-level parallelism,

SFU's performance, and hence average power, can increase at most by this factor. Therefore, SFU's average power is within the budget.

Effect of SMT:

The above model does not include SMT. It may seem that because SMT reduces each core's exposed memory latency via better thread overlap, SFU's opportunity would decrease with SMT. However, SMT increases the number of threads to improve thread ovelap but does not change each thread's compute-memory overlap or performance (ignoring any extra cache misses due to SMT's increased cache pressure). SMT's thread overlap is no different than that achieved by the cores of a multicore (i.e., one core's memory latency is overlapped by the other cores' computation and memory accesses) and the non-linear impact of memory latency on power-performance tradeoff holds in a multicore irrespective of the number of cores, as seen in Equation 2.1. Therefore, the non-linearity holds for SMT as well, implying that the above model stays valid for SMT. One may think that DSL, like SFU, can also increase its memory-level parallelism via SMT. However, adding SMT contexts to a core would linearly increase its activity factor, and hence dynamic power, and would force the number of DSL cores to be cut by the same factor. Therefore, DSL's performance, and hence SFU's opportunity, would remain unchanged with SMT. In reality, SMT's increased cache pressure may actually increase cache misses and thereby increase SFU's opportunity. I include SMT in all our results.

As an aside, I note that unlike SMT, out-of-order issue does increase each thread's compute-memory overlap and therefore, may decrease SFU's opportunity. I evaluate this point in our results.

Controlling single-thread latency:

SFU, as proposed, improves (1) overall execution time of parallel workloads (e.g., scientific workloads) and (2) throughput of enterprise workloads (e.g., on-line trans-

action processing). However, full SFU requires scaling the clock by $(\alpha \times c)$ every generation which may degrade single-thread performance, and hence response latency, of enterprise workloads. This degradation may be considerable despite the sub-linear impact of clock on performance particularly in later generations where the clock is slowed down significantly with full unscaling. To address this issue, I exploit the other non-linear impact of throughput on queuing delays where higher throughput super-linearly reduces the queuing delay component of response latency as dictated by queuing theory, and thereby compensates for the slower clock. Based on this non-linear relationship, I propose to reduce the unscaling factor α compared to full SFU, still allowing more cores to be powered than DSL, though not all the cores like full SFU. I choose α so that the resultant degradation of single-thread execution time is matched by the gain in the queuing delay due to higher throughput, resulting in similar total response latency and higher throughout as compared to DSL. I propose to apply such controlled SFU (C-SFU) only for response-latency-sensitive workloads and not for others.

Because C-SFU allows only a subset of the cores to be powered, I consider the option of converting the area and leakage of the remaining unpowered cores, which would otherwise be dark silicon, into additional last-level cache (unlike core customization, this option does not worsen design/programmability costs). While the same design can choose dynamically between full or controlled SFU based on the workload, this option cannot be applied dynamically and, if employed, would require different designs for full and controlled SFU. Though DSL's surplus core leakage budget is already used up for extra cores (Section 2.2.2), I still consider this conversion option for DSL as well.

2.2.3 Model's predictions

To illustrate our model's predictions, I analyze four regions of the workload-characteristics space in Figures 2.1(a) through (d), respectively: (1) mostly serial

Table 2.1.
Clock and active core scaling with technology generations ($c = 1.07$, $p = 0.73$, $\alpha = 0.68$)

Parameter	Arch.	G0	G1	G2	G3	G4
Clock (GHz)	PU,DSL	3.2	3.42	3.66	3.92	4.19
	SFU	3.2	2.33	1.69	1.23	0.90
Active cores	PU,SFU	4	8	16	32	64
	DSL	4	5.61	7.79	10.69	14.67

and memory-unintensive ($s = 0.8$ and $m = 0.02$), (2) mostly serial and memory-intensive ($s - 0.8$ and $m = 0.8$), (3) highly and memory-unintensive ($s = 0.01$ and $m = 0.02$), and (4) highly parallel and memory-intensive ($s = 0.01$ and $m = 0.8$).

The figure compares the PU (Equation 2.1), DSL, (Equation 2.2 with the adjustment for the number of cores from Equation 2.4), and SFU (Equation 2.3) multicores across technology generations. The Y axis shows the speedups for the multicores over our generation-0 4-core multicore; and the X axis shows the technology generations 1 through 4. I assume that the clock and power improvement factors across generations, c and p, are 1.07 and 0.73, respectively, in line with the conservative scaling trends in [4]. Based on this p, the frequency unscaling factor, α, is 0.68. I assume a chip leakage budget of 30% of which 40% is core leakage (i.e., $l = 0.3$, $\lambda = 1$, and $f_l = 0.4$). Using these scaling values, Table 2.1 shows the clock frequencies and the number of active cores across generations (G0 through G4) for PU, DSL, and SFU multicores.

Figure 2.1(a) shows that for serial workloads ($s = 0.8$), the active core count does not matter and DSL performs as well as PU. On the other hand, SFU performs worse due to its slower clock. However, serial workloads are unimportant for future multicores which are destined for servers where multi-threaded workload is the norm. As such, the speedups are low even for PU due to the highly-serial workload, placing SFU within normal range. Further, power is not a problem for such workloads which need only a few cores.

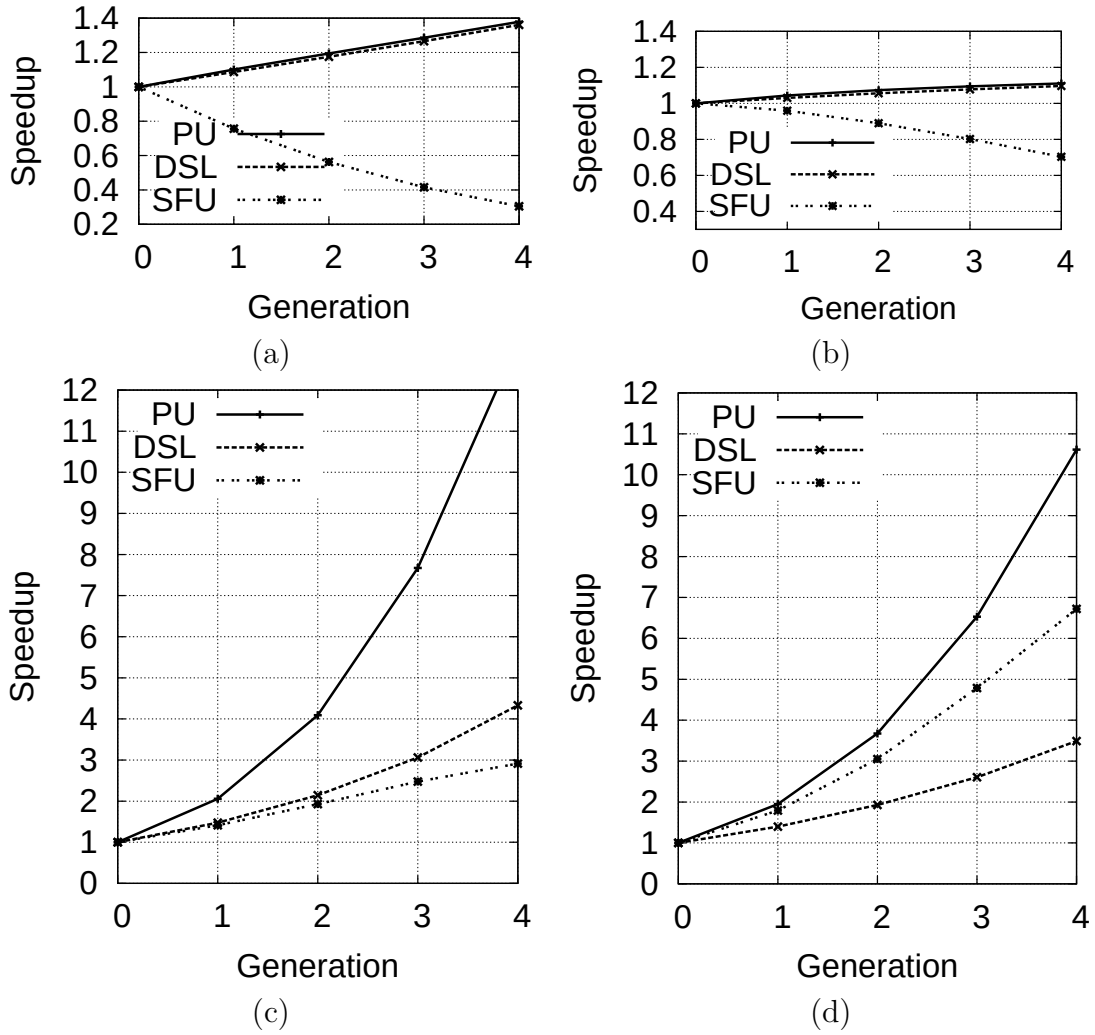

Fig. 2.1. Speedups across generations of PU, DSL, and SFU multicores over a four-core multicore (generation 0) for the following workloads: (a) mostly serial and memory-unintensive, (b) mostly serial, and memory-intensive, (c) mostly parallel and memory-unintensive, and (d) mostly parallel and memory-intensive

The same analysis holds for Figure 2.1(b) though the speedups are even lower compared to those of Figure 2.1(a) because of the higher impact of memory latency which does not scale.

In Figure 2.1(c) which shows parallel, memory-unintensive workloads, PU performs better than both DSL and SFU. However, DSL outperforms SFU because (1)

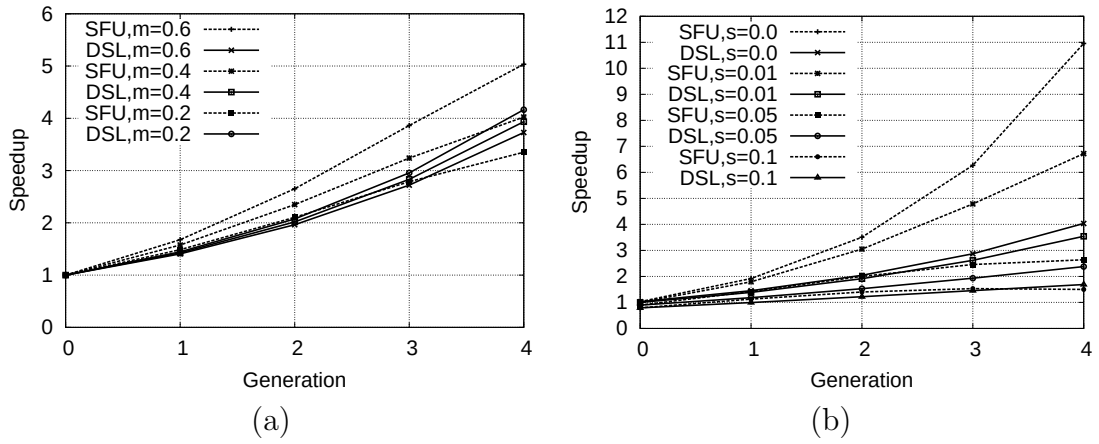

Fig. 2.2. Sensitivity to model parameters (speedup relative to generation 0, 4-core configuration) for the different values of m and s: (a) Variable memory intensity with s = 0.01, (b) Variable parallelism with m = 0.8

SFU's memory parallelism offers no advantage for memory-unintensive worklaods and (2) DSL enjoys the benefit of added cores to compensate for SFU's added leakage, as mentioned in Section 2.2.2.

Finally, Figure 2.1(d) shows that for parallel, memory-intensive workloads, PU performs better than DSL for the same reason as Figure 2.1(c). Despite the lack of power constraints, PU does not achieve linear speedups with the number of cores due to the serial portions (Amdahl's Law effect). SFU performs better than DSL due to higher memory-level parallelism achieved by SFU's more, albeit slower, cores in the major parallel-memory component of this workload. Nevertheless, SFU does not fully close the gap between PU and DSL due to SFU's slower clock (Table 2.1) which affects both the serial-non-memory and parallel-non-memory components in the workload. I show in Section 3.5 that our real-world commercial and scientific workloads closely track Figure 2.1(d).

Sensitivity to model parameters: Figure 2.2 illustrates the sensitivity of SFU's speedups to the two parameters s and m. Figure 2.2(a) confirms that SFU remains faster than DSL with m as low as 0.5 which is well below the expected exposed miss

Table 2.2.
Scaling factors relative to Gen0 (45nm)

Parameter	Gen0	Gen1	Gen2	Gen3	Gen4
Tech Node (nm)	45	32	22	16	11
Clock (c^i)	1.00	1.10	1.19	1.25	1.30
Vdd	1.00	0.93	0.88	0.86	0.84
Capacitance	1.00	0.75	0.56	0.42	0.32
Dyn. Power (p^i)	1.00	0.71	0.52	0.39	0.29
SFU factor (α^i)	1.00	0.70	0.48	0.32	0.22

latency given miss-rates (2%) and given current memory latency trends. Similarly, Figure 2.2(b) confirms that SFU performs better than DSL at $s < 0.1$. Note that with $s >= 0.1$ the Amdahl's speedup limit is 10X and thus serialization is a tighter bottleneck than the power constraint.

2.3 Experimental Methodology

I run full-system simulations using Wisconsin GEMS [11] built on top of Simics [12]. I simulate SPARC-based multicores running Solaris 10.

Technology Scaling: While our main results use technology scaling parameters from the conservative projections for planar transistors by Borkar *et al.* [4], I also show some brief results using ITRS's more aggressive projections for FinFETs [13]. Based on Borkar's projections, Table 2.2 shows the improvement factors over 45nm technology node for clock frequency (c^i for the ith generation, in Section 2.2.2), supply voltage (Vdd), capacitance, and per-core power (p^i). The table also shows SFU factors (α^i in Section 2.2.2 where $\alpha^i * p^i = 0.5^i$).

Multicore organization: I assume a tiled organization in which each multicore "tile" comprises a core, private L1 I- and D-caches using MESI coherence protocol, and a local bank of the shared, unified L2. The tiles are connected via an on-chip network with memory controllers at the network edges.

Core: While our baseline generation assumes in-order-issue, two-way SMT cores (line 1 in Table 2.3), I also show some brief results for out-of-order-issue cores. I use two-way SMT cores which are common in Intel designs to match our scaling numbers in Table 2.2 which are based on Intel designs [4]. Based on the per-core power and clock factors from Table 2.2, I list the number of active cores and the actual clock frequencies for PU, DSL and SFU in Table 2.4. While c and p are assumed to be constants in Section 2.2.2, Borkar's scaling assumes that the factors vary from one generation to the next (Table 2.2). Consequently, clock frequencies in Table 2.1 and Table 2.4 differ slightly.

Cache Capacity and Access Latency: I hold key parameters of the L1 cache and the per-core L2 bank (capacity, block size, associativity and access latency in cycles) constant across generations (lines 2 – 8 in Table 2.3). The aggregate L1 and L2 capacities double every generation as per Moore's Law. Though DSL keeps many of its cores powered down, it uses *all* of the shared L2 cache including the banks in the inactive cores' tiles (i.e., all three designs, PU, DSL, and SFU, have the same amount of L2 cache in each generation). Consequently, a deeper on-chip hierarchy (e.g., L3) will improve DSL and SFU to similar extents.

I conservatively assume that the access latencies in ns of L1 and L2 bank scale at the same rate as frequency improvement. Consequently the access latencies in cycles, stays constant across generations for PU, DSL, and SFU (line 8 in Table 2.3), though SFU's slower clock implies longer latencies in ns.

On-chip Network: I assume a dimension-order routed, 2D mesh network (line 9 in Table 2.3) which grows across generations (line 7 in Table 2.4). Our simulator models a simple 1-cycle router delay per hop where the link latency scales slower than logic due to the well-known wire-delay effects. Accordingly, I assume a modest 1-cycle increase in latency at Gen3 for PU, DSL, and SFU (line 8 in Table 2.4). Here again, SFU's slower clock implies longer latencies in ns.

DRAM Latency and Bandwidth (including impact of die-stacking): There are two sources of DRAM latency improvement: technology scaling every gen-

Table 2.3.
Invariant parameters across generations

	Parameter	Values
1	SMT contexts per core	2
2	Private L1D size (KB)	64
3	Private L1I size (KB)	64
4	Private L1 associativity	4
5	L1 access (cycles)	3
6	Shared L2 associativity	32
7	Shared L2 block size (bytes)	64
8	L2 bank (cycles)	17
9	Network Topology	2D mesh
10	Channel width (bits)	64
11	Mem. Ctrl. Queue (entries)	32
12	Page mode	closed

Table 2.4.
Parameter scaling with technology generations

	Parameter	Values					
		Gen0	Gen1	Gen2	Gen3	Gen4	Scaling Comments
1	Number of cores (PU/SFU)	4	8	16	32	64	2X per gen
2	Number of cores (DSL)	4	6	8	12	16	Power-limited
3	Clock (GHz) (PU/DSL)	3.2	3.52	3.81	4	4.16	As per Table 2.2
4	Clock (GHz) (SFU)	3.2	2.48	1.73	1.28	0.89	As per Table 2.2
5	Shared L2 size (MB)	4	8	16	32	64	2X per gen
6	Shared L2 banks	4	8	16	32	64	2X per gen
7	Network	2x2	2x4	4x4	4x8	8x8	
8	Link latency (cycles)	2	2	2	3	3	Slower than f scaling
9	Memory (DSL/PU) cycles	320	344	368	380	400	2% reduction per gen
10	Memory (SFU) cycles	320	240	180	124	86	2% reduction per gen
11	Number of DRAM banks	16	32	64	128	256	2X per gen
12	Number of memory channels	1	2	4	8	16	2X per gen
13	Number of instances (DSL)	1	1	2	4	4	Problem size scaling
14	Number of instances (SFU/PU)	1	1	2	4	8	Problem size scaling

Table 2.5.
Workloads: Description and Characteristics

Commercial Workloads	MPKI
SPECjbb: version 2005, Java-based 3-tier client/server system workload with emphasis on the middle tier. Java server VB version 1.5 with parallel garbage collection. I simulate a system with 24 warehouses (~600 MB).	7.1
Online Transaction Processing (OLTP): models database transactions of a wholesale parts supplier. I use PostgreSQL 8.3.7 database system and DBT-2 test suite which implements TPC-C benchmark. I use a database of 25,000 warehouses (~5GB). I simulate 128 concurrent database connections.	5.1
Apache: version 2.2.9, a static web server workload with repository of 20,000 files (~500 MB). SURGE is used to generate web requests by simulating 1600 clients, each with 25ms think time between requests.	17.9
Scientific Workloads (memory-intensive)	
FFT: is a Splash benchmark that computes Fourier transforms. I run the transpose computation of 4 Million complex numbers (~64 MB) for generation 0 to 2 and 16 Million complex numbers for generation 3 and 4.	10.2
canneal: is a Parsec benchmark that models cache-aware annealing to optimize routing cost of a chip design. I use the native dataset (~100 MB). Systems for generations 0 to 2, 3 and 4 optimize 1, 2 and 4 chips respectively.	4.0
Streamcluster: is a Parsec benchmark that performs online clustering of an input stream. I use 1 million 128-dimensional points, 5000 intermediate centers (~100 MB) for generation 0 to 3 and and 2 million 128-dimensional points , 20000 intermediate centers (~ 275 MB) for generation 4.	4.8

eration and one-time move to 3-D die-stacking. I assume that the former yields the usual 2% per generation (optimistic estimate as trends indicate no improvement in DRAM latency [14]) and show the resultant memory latency in cycles for PU, DSL, and SFU in lines 9 – 10 in Table 2.4. I assume that the latter achieves the expected 30% latency reduction [5, 15] which I evaluate in Section 2.4.3.

I assume that the number of DRAM banks (i.e., the internal DRAM bandwidth) doubles every generation, in line with the scaling of DRAM density (line 11 in Table 2.4). I assume closed page mode which works best for multi-threaded workloads (line 12 in Table 2.3) [16]. I show our main results assuming that processor-memory bandwidth (i.e., the number of memory channels) scales across generations (line 12 in Table 2.4). This assumption is in anticipation of the imminent deployment of 3-D stacking technology. Further, a recent paper [17] shows that compression and other

techniques can allow conventional pin bandwidth to scale beyond 100 cores, covering many future generations. Nevertheless, I include results constraining the number of memory channels in Section 2.4.3.

Leakage: As explained in Section 2.2.2, recent commercial microprocessors increase transistor threshold voltages and decrease transistor speeds to hold leakage at a constant 25-30% of the constant total power budget [10]. Consequently, the total dynamic power budget is also a constant. Borkar's conservative scaling, and therefore Table 2.2, include the effects of such adjustments. While our main results assume a chip leakage budget of 30% held constant across generations (i.e., $l = 0.3$ and $\lambda = 1$), I also show brief results for leakage growing from 30% to 50% across our four generations (i.e., $l = 0.3$ and $\lambda = 1.15$). Based on our reasons in Section 2.2.2, I assume that 40% of the chip leakage is in the cores (i.e., $f_l = 0.4$).

Workloads: I run the commercial and scientific workloads shown in Table 3.4. Note, because our focus is on memory-intensive workloads where SFU offers maximum advantage, I focus on the above workloads. However, because it is important to show that SFU does not hurt performance for memory-unintensive workloads, I show additional results in Section 2.4.5 for three other workloads that are compute-intensive. Recall from Section 2.2.2 that I revert to DSL for memory-unintensive (based on a miss-rate threshold) workloads. To account for the general trend of data increasing with cache sizes across generations, I scale up the workload size across generations so that the off-chip miss rate, shown as misses per kilo instructions (MPKI) in Table 3.4, remains about the same across generations. Because even higher miss rates would give more opportunity for SFU over DSL, this constant miss rate is a conservative choice. While the scientific workloads are amenable to easy scale-up by increasing the datasets, meaningfully scaling up the commercial workloads' datasets to 128 cores requires significant amount of domain expertise to achieve realistic settings (e.g., tune various benchmark parameters to avoid software bottlenecks). Instead, I scale up the commercial workloads by consolidating multiple instances of the same benchmark as shown in lines 13 – 14 of Table 2.4 (i.e., homogeneous consolidation as done in [18]).

This methodology is in line with the trend in software consolidation especially on large multicores [18]. I compile our workloads with full software prefetching so that the exposed memory latency is realistic. Finally, because OLTP runs are long-running, I only include OLTP in the main results (Section 2.4.1 and Section 2.4.6). I omit OLTP from the other sensitivity results.

2.4 Experimental Results

I begin with our main results (Section 2.4.1) in which I compare across technology generations the performance of PU, DSL, and SFU multicores running our commercial and scientific workloads. This comparison assumes in-order-issue processors, doubling of processor-memory bandwidth (i.e., the number of memory channels) across generations, conventional off-chip memory, Borkar's conservative scaling parameters based on planar transistors, and chip leakage of 30% constant across generations. I also evaluate the effect of (1) out-of-order issue processors in Section 2.4.2 (2) constraining the processor-memory bandwidth in Section 2.4.3; (3) faster memory (e.g., via 3-D stacking) in Section 2.4.3; (4) ITRS's aggressive scaling parameters based on FinFETs in Section 2.4.4; (5) higher chip leakage of 50% in Section 2.4.4; and (6) greater compute phase which is the case for compute-intensive workloads . Finally, in Section 2.4.6, I compare the response time (i.e., single-thread performance) and throughput achieved by controlled unscaling (C-SFU) and DSL for commercial, transaction-processing workloads.

2.4.1 Performance

In Figure Figure 2.3, I compare PU, DSL, and SFU multicores across technology generations. I evaluate full SFU here and cover C-SFU later in Section 2.4.6. The figure shows performance normalized to that of a generation-0, four-core, PU multicore on the Y axis and technology generations past our generation 0 as well as the benchmarks on the X axis. The normalized performance for our consolidated

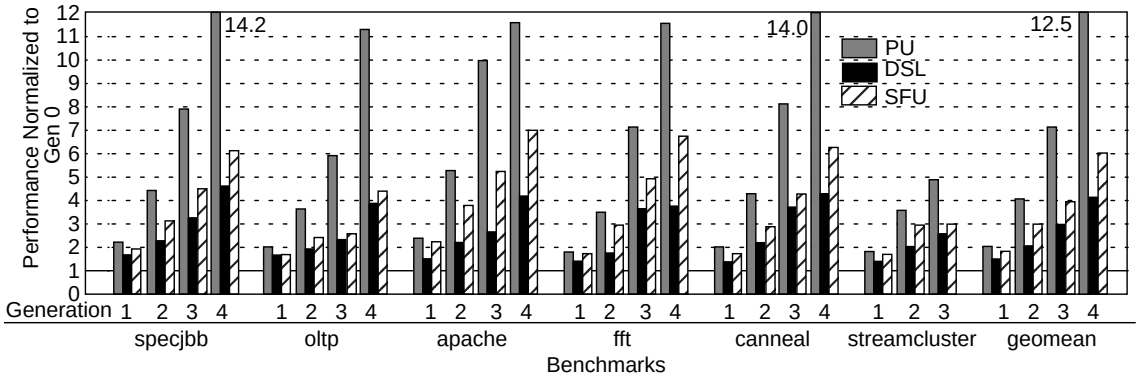

Fig. 2.3. Speedups of PU, DSL, and SFU multicores over a generation-0, four-core, PU multicore.

commercial workloads (Section 2.3) is defined as the improvement in transaction rate whereas that for our scientific workloads is defined as improvement in execution time. Recall that PU and DSL multicores use the same clock frequencies whereas the SFU multicore uses slower clock, and that while the power-unconstrained and SFU (full unscaling) multicores keep all the cores active, the DSL multicore has fewer active cores.

From Figure 2.3, I observe that for all the benchmarks DSL's relative performance improves slowly over generations with DSL lagging well behind PU, as also shown in [3]. This lag is due to the difference in the number of active cores between PU and DSL. By keeping all the cores active, albeit at slower clocks, SFU performs significantly better than DSL for our memory-intensive benchmarks (see Table 3.4), as predicted in Figure 2.1(d). Overall, SFU achieves an average 46% improvement over DSL for our workloads in generation 4.

Recall from Section 2.2.3 that our model predicts this improvement based on the reasoning that the higher memory-level parallelism enabled by the greater number of active cores in SFU than in DSL more than offsets the slower clock in SFU. To corroborate this reasoning, I show the misses per thousand instructions (MPKI) for the generation-0, PU multicore (Table 3.4), and the average number of parallel memory requests for PU, DSL, and SFU multicores (Table 2.6). Because I commensurately scale the cache and workload sizes across generations (Table 2.4), the MPKI for the

Table 2.6.
Measured average number of parallel memory accesses

Benchmarks	Gen0	Gen1			Gen2			Gen3			Gen4		
		PU	DSL	FU	PU	DSL	FU	PU	DSL	FU	PU	DSL	FU
specjbb	5.4	10.5	7.7	9.2	20.4	9.8	13.3	37.7	13.4	19.9	74.2	15.6	27.3
oltp	4.8	6.2	5.4	5.8	12.9	8.16	9.3	19.5	10.4	12.5	41.5	12.7	14.5
apache	6.0	11.2	8.6	10.2	20.7	10.7	14.3	36.9	14.9	20.8	78.2	17.8	38.2
fft	6.3	12.0	9.3	11.3	24.5	11.8	20.7	48.7	17.9	33.0	98.0	22.9	48.4
canneal	5.8	11.5	7.9	10.5	22.2	10.9	16.6	34.6	14.5	19.8	71.7	15.7	26.8
streamcluster	5.9	11.0	8.3	10.0	21.1	10.9	15.9	39.6	15.3	24.0	-	-	-

other generations are similar; and because all the three multicores have the same amount of cache, the MPKI for the other multicores (DSL and FU) are similar (not shown). From Table 3.4, I see that our memory-intensive benchmarks have usual but non-trivial MPKI (but well below the excessive point of thrashing). As expected, the greater number of parallel memory requests in SFU than in DSL (Table 2.6) indicate that SFU achieves higher memory-level parallelism, and hence higher performance. These performance results, which are the main claims of the paper, show that DSL's bounds can be exceeded for important workloads, and that slow silicon is better than DSL as long as the slow silicon makes memory accesses.

SFU, however, does not catch up to PU because while both systems have the same number of active cores and hence achieve the same amount of memory-level parallelism, the slower clock in (full) SFU results in slower compute portion. Nevertheless, the gap between SFU and PU is narrow in the earlier generations and is wide only in the fourth generation, supporting our claim in Section 3.1 that SFU provides a gentle, evolutionary path for multicores which may need to be augmented with customized cores only in later generations.

2.4.2 Impact of out-of-order issue

Our results in Section 2.4.1 use in-order-issue cores whereas out-of-order-issue cores can hide some memory latency and may diminish SFU's memory-parallelism advantage over DSL. To resolve this issue, I show in Figure 2.4 performance for

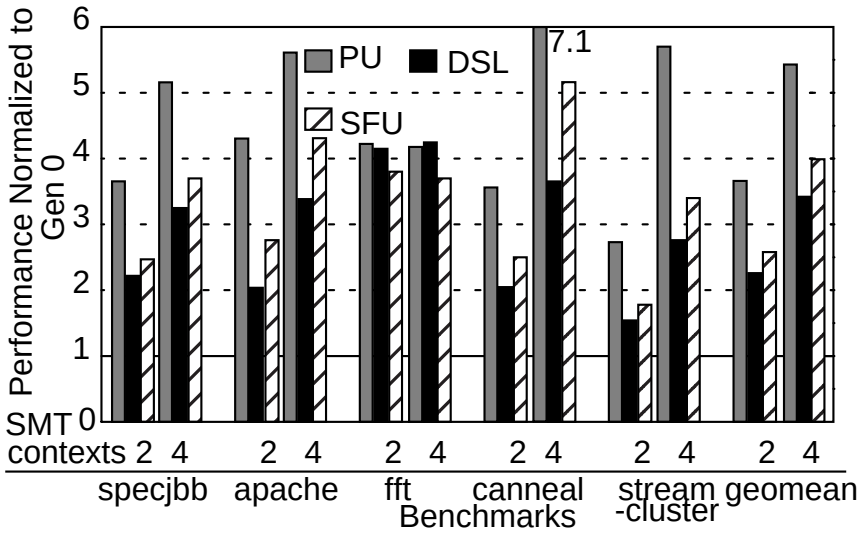

Fig. 2.4. Impact of out-of-order issue

generation-2 DSL, SFU, and PU multicores normalized to that of a generation-0, four-core, PU multicore (Y axis) running our memory-intensive workloads (X axis), where all the systems employ out-of-order-issue cores. I vary the number of SMT contexts as 2 and 4 in the X axis. Because out-of-order-issue simulations are long-running, I show results only for generation 2. I point out that while Figure 2.3 compares in-order-issue systems compared among each other, Figure 2.4 compares out-of–order-issue systems among each other. From the figures, I see that the speedups for out-of-order issue (Figure 2.4)closely match those for in-order issue generation-2 in Figure 2.3. The only exception is *FFT* which saturates the memory bandwidth in all the systems, as evidenced by excessive queuing at the memory controllers. The similarity in the speedups exists even though out-of-order issue perform around 22% better than in-order issue (not shown), implying that SFU overlaps the significant memory latency left exposed despite out-of-order issue's ability to hide memory latency. Overall, SFU retains its significant advantage over DSL, resulting in 18% improvement with 2-way SMT (for in-order issue generation 2, the improvement is 22% in Figure 2.3). Further, SFU's advantage over DSL increases from 2-way to 4-way SMT due to increased cache pressure (Section 2.2.2).

2.4.3 Impact of processor-memory bandwidth and faster, 3-D stacked memory

Because SFU exploits higher memory-level parallelism requiring more processor-memory bandwidth than DSL, I study the impact of lower processor-memory bandwidth on SFU. Separately, I study how faster memory due to 3-D stacking affects SFU (I show the two effects together to save space). Figure 2.5 shows normalized performance on the Y axis for generation-4 DSL and SFU multicores running our memory-intensive workloads (X axis). While the default configuration for our results has 16 memory channels and conventional, slow memory, the graph shows performance using fewer memory chanels (8) than the default, (labeled as 'A' on X axis), the default configuration (labeled as 'B'), and faster memory than the default (labeled as 'C'). Performance is normalized to that of the default, generation-0 PU multicore. The faster-memory case assumes a 30% lower memory latency because previously-reported latency reduction for 3-D stacking ranges from 20% to 30% [15]. I choose generation 4 because it has the highest bandwidth demand. Comparing 8 and 16 channels ('A' and 'B', respectively), SFU outperforms DSL by a significant margin (53% on average) even with 8 channels though the margin increases with 16 channels for bandwidth-hungry applications like *FFT*. As discussed in Section 2.3, these amounts of bandwidth seem achievable by the time of generation 4, given the imminence of 3-D stacking and potential of compression and other techniques [17]. Comparing conventional ('B') and faster memory ('C') cases, both DSL and SFU improve in performance due to lower memory latency. However, the reduction in latency is not enough to eliminate exposed memory latency. As such, SFU is still on average 39% better than DSL.

2.4.4 Impact of ITRS (FinFETS) and leakage

Because ITRS projections for FinFETs are more optimistic than Borkar's projections for planar transistors, I study whether ITRS projections change our results

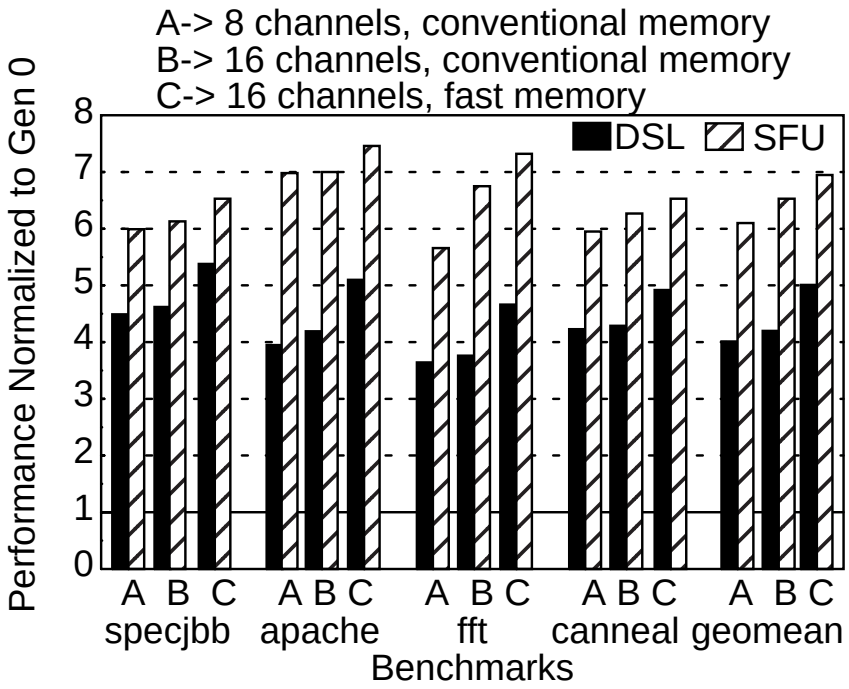

Fig. 2.5. Impact of memory bandwidth and latency

(ITRS revised its clock-speed projections from 13 GHz to 5.5 GHz which is more in line with Borkar's projections). Separately, I study the impact of higher leakage (I put these two together to save space). Figure 2.6 shows normalized performance (Y axis) for generation-4 PU, DSL, and SFU running our memory-intensive workloads (X axis). While the default configuration for our results uses Borkar's projections with conventional, slow memory and 30% chip leakage constant across generations, the graph shows performance assuming ITRS projections which includes faster memory (labeled as 'A' on X axis), the default configuration (labeled as 'B'), and higher leakage than the default (labeled as 'C'). Performance is normalized to that of the default, generation-0 PU multicore. ITRS projections predict c^i, p^i, and α^i for generation-4 as 1.73, 0.25, and 0.22, respectively, so that DSL has 16 cores running at 5.54 GHz and SFU has 64 cores running at 1.2 GHz. The higher-leakage case assumes 50% chip leakage in generation-4 which corresponds to leakage increasing from 30% at the rate of 14% per generation (more than 50% leakage would be unreasonable because then less than half of the chip power would be available for real work). Instead of giving

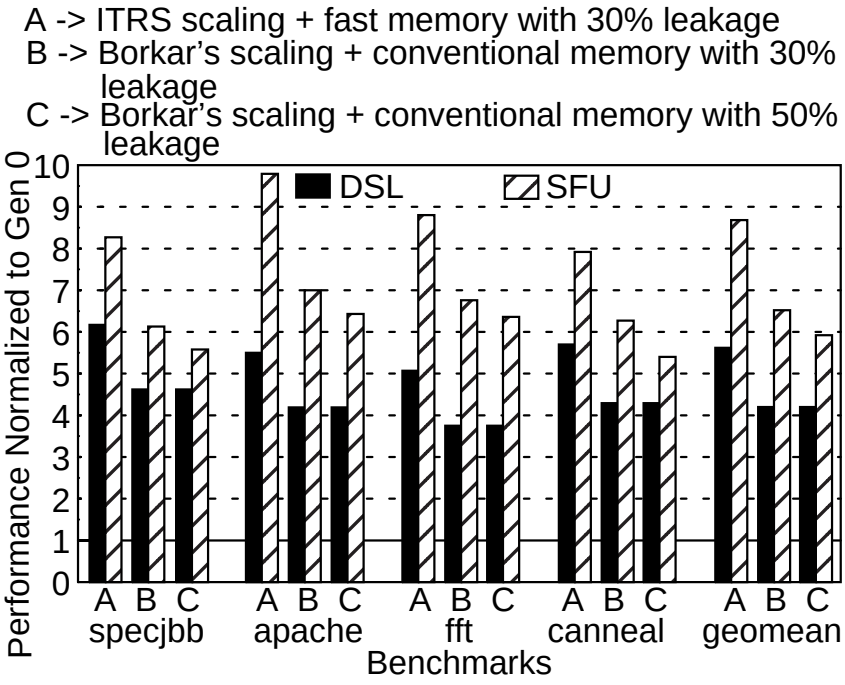

Fig. 2.6. Impact of ITRS projections and higher leakage

extra cores to DSL to account for DSL's core leakage surplus (Section 2.2.2), I reduce SFU's clock speeds from 0.89 GHz (default) to 0.78 GHz, because extra DSL cores require time-consuming simulation warm-up of a new DSL multicore. Comparing the ITRS ('A') and default ('B') cases, while both DSL and SFU perform better with ITRS projections than with the conservative default projections due to ITRS's faster clocks and memory, SFU's lead over DSL increases with ITRS because the faster clocks expose more memory latency even with the faster memory. Comparing the default ('B') and higher leakage ('C') cases, SFU's lead over DSL shrinks modestly with higher leakage due to the further lowering of SFU's clock speeds. However, because DSL's leakage surplus is not large (Section 2.2.2), SFU's lead shrinks only modestly even with leakage as high as 50%.

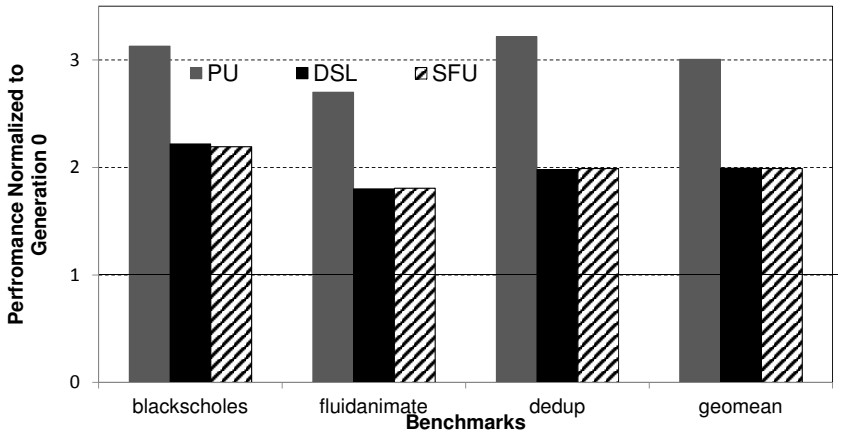

Fig. 2.7. Speedups of PU, DSL, and SFU generation-2 multicores over a generation-0, four-core, PU multicore for compute-intensive workloads

2.4.5 Memory-unintensive Workloads

Recall from Section 2.2.2 that I revert to DSL operation for memory-unintensive (compute-dominated) workloads because it yields better performance. Because SFU's key improvement is increasing memory-level parallelism, and because memory-unintensive workloads do not have any significant exposed memory latency, it is better to focus the power budget on a few cores operating at maximum frequency (similar to DSL) and deactivate other cores. On the hardware-front, the configuration is identical to DSL however, there is a key difference on the software front. Because SFU exposes more processors, the application will have likely spawned as many threads. However, when dynamically reverting to DSL operation, those larger number of threads must now run on fewer cores.

To quantify the impact of this change on memory-unintensive workloads, we focus on three Parsec benchmarks with very low MPKI (*blackscholes*, *fluidanimate*, and *dedup* which have MPKI values of 0.08, 0.91, and 0.46 respectively). Based on a simple fixed MPKI threshold of 2.0, I are trivially able to identify compute intensive workloads. The task to monitor the MPKI and decide which mode to operate SFU in can be left to the operating system. Figure 2.7 shows the performance of PU, DSL and SFU normalized to that of the default, generation-0 PU multicore (Y-axis) for

Table 2.7.
Controlled SFU Configurations

Configuration	Cores	Clock (GHz)	L2 (MB)
DSL + big cache	14	4.16	98
C-SFU + big cache	20	2.84	98
DSL	16	4.16	64
Full SFU	64	0.89	64

memory-unintensive benchmarks (X-axis). The key result is the fact that the performance of DSL and SFU for generation-2 are nearly identical (and significantly less than the impractical PU configuration); thus proving that the difference in number of threads does not have an impact on overall performance.

2.4.6 Single Thread Latency

Recall from Section 2.2.2 that to restore single-thread performance for response-time-sensitive workloads, controlled SFU (C-SFU) trades-off throughput by activating fewer additional cores than dark-silicon (but not as many as full SFU) for higher clock frequency than full SFU (but not as high as DSL). Also, recall the option of using the leakage power and spare transistors of the unactivated cores into additional last-level cache for C-SFU. (I include a DSL configuration with the big-cache as well. However, because DSL does not have the leakage budget for the additional cache, I reduce the number of cores by 2 to account for the added leakage of the cache.) I empirically determine a C-SFU configuration that corresponds to acceptable throughput-latency trade-off as listed in Table 2.7. The table shows the number of cores and frequency for C-SFU and DSL with the "big-cache" option as well as full SFU and DSL, all for generation 4 (where SFU has the biggest clock disadvantage). Due to the non-linear impact of memory latency, C-SFU's 32% slower clock amounts to 18% single-thread slowdown over DSL.

Throughput: Figure 2.8 shows throughput (transactions per second) for C-SFU + big cache, DSL + big cache, full SFU, and DSL normalized to that of a gen0 system

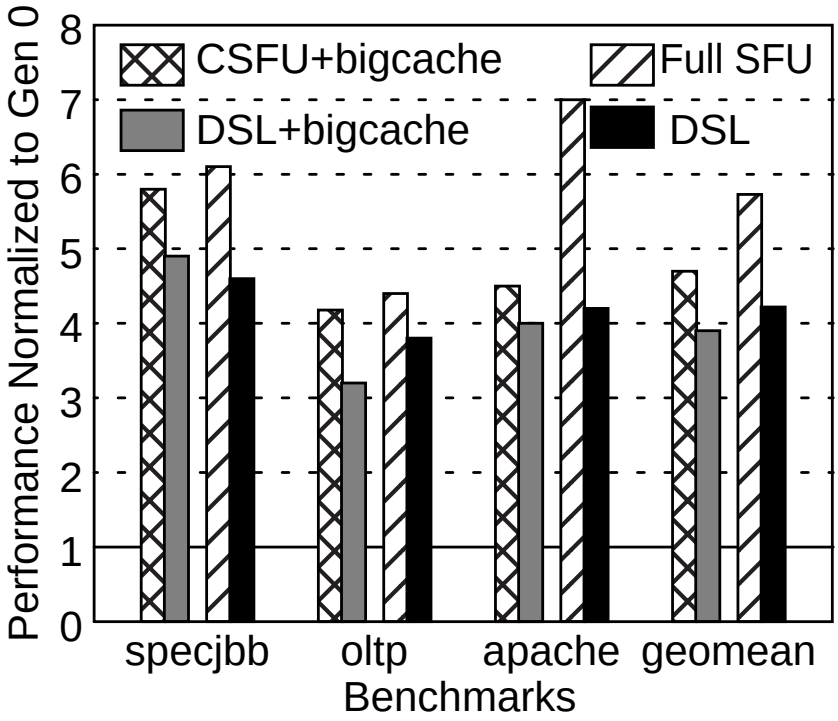

Fig. 2.8. Normalized throughput

(Y-axis) for our commercial workloads which are sensitive to response times (X axis). I see that C-SFU performs better than DSL by 21%. Further, DSL does not improve with big-cache (DSL + big cache versus DSL), because the additional cache is offset by the reduction in the number of cores. As such, neither variant of DSL matches C-SFU's performance.

Response Time: To analyze queuing delays, I use an M/M/m multiserver queuing model to relate the response time to throughput. I set the number of servers m to be 40 for C-SFU (20, 2-way SMT cores) and 28 for DSL (14, 2-way SMT cores), and the service time (i.e., single-thread execution time) to be 1.24 for C-SFU and 1.0 for DSL. Figure 2.9 plots the response time (Y-axis) vs. throughput (X-axis, normaized to DSL's saturation throughput) curves for C-SFU and DSL. The typical L-shaped curves show that at low loads, the response time is almost entirely service time (i.e., little queuing delay) and at high loads, the response time grows with the queuing delay, increasing rapidly near saturation. Typically, systems are

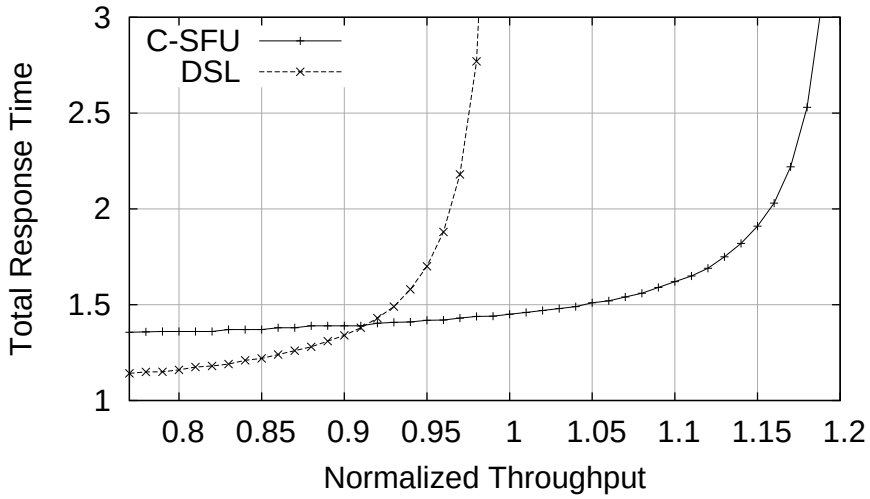

Fig. 2.9. Total response time versus throughput

operated in the flat, pre-saturation region to achieve reasonable response times and good throughput (a proxy for utilization). While C-SFU's response time is longer than DSL's at low loads, cores can be turned off at low loads without throughput loss making power constraints irrelevant. However, in the fairly typical region of load (0.87-0.93 normalized throughput), C-SFU stays within +/-10% of the response time of DSL. Thus, C-SFU exploits the non-linear impact of higher throughput on queuing delays to compensate for its slower clock.

2.5 Related Work

The multicore scaling roadmap relies on both Moore's scaling [19] and Dennard's scaling [1] to provide the area and power budgets, respectively, to support a steady, geometric increase in the number of cores. Unfortunately, the slowing down of Dennard scaling (while Moore's scaling continues) puts the multicore roadmap at risk. While in the past, a significant body of work used dynamic voltage and frequency scaling (DVFS) [20, 21] to achieve energy efficiency, the voltage knob will no longer be available.

Prevous work responds to this challenge by noting that the traditional multicore scaling will result in dark silicon [3,5]. To address the dark silicon problem, researchers

have proposed specialization [6, 22] and approximate computing [23] as two future paths.

Approximate computing address the dark silicon problem by trading off exactness with high energy efficiency. The insight is that a small degradation in output quality can provide high gains in energy efficiency [23]. While approximation is a feasible approach for client codes and specific algorithm classes where minor dilution in precision is acceptable (e.g., image/video compression/decompression, iterative convergence codes), it is not a viable approach for enterprise/commercial server class applications where inexactness is effectively incorrect. This is especially important because commercial servers are expected to be the dominant market for multicores with 32+ cores. SFU squeezes out performance gains from all the silicon without compromising on the existing (exact) computing paradigm.

Specialization aims to provide higher energy efficiency while preserving performance. Through task-specific accelerators and co-processors, specialization focuses on running the code on the right hardware at the right time [6, 22]. Our approach does not preclude the use of specialization as the two approaches are orthogonal because specialization increases the energy efficiency of compute whereas SFU leverages enhanced memory-level parallelism to achieve higher efficiency.

Prior work has demonstrated the value of simple models based on Amdahl's law [8] that are useful for understanding and demonstrating the first-order effects of various bottlenecks such as the serial portion of programs [9] and multicore power constraints [3]. Our model builds on such prior models to capture the effects of memory latency and memory parallelism for power-constrained systems. Our model directly leads to the insight that SFU achieves significantly higher performance than dark silicon for memory-bound (i.e., typical) multicore workloads.

3. METADATA ORGANIZATIONAL TECHNIQUES FOR 3-D DIE-STACKED CACHES

3.1 Introduction

Die-stacking of DRAM on top of a microprocessor is emerging as a way to continue scaling the pin bandwidth to off-chip memory [15,24]. Die-stacking will allow the processor and DRAM dies to be connected with a few thousands of fast, through-silicon vias (TSVs) as opposed to a few hundreds of slower, traditional pins, thereby providing a factor of 4-5 or higher bandwidth (perhaps over some technology generations). Though such 3-D DRAM has high capacity, the DRAM will likely be too small to play the role of main memory. As such, the DRAM will be used as a large cache with the key advantage of high-bandwidth, cache-processor connection. In Section 3.2.4, I conclusively show that the problem is memory bandwidth and not latency.

The die-stacked DRAM cache's tag poses a challenge. Placing the tags on the processor die adds significant area overhead for conventional block sizes given that the DRAM cache is off-die, large, and scales in capacity more quickly due to DRAM technology improvements than on-processor-die SRAM tag (e.g., a 256-MB DRAM cache with 64-B blocks incurs a 14-MB tag overhead while the last-level on-die cache may be 16 MB). On the other hand, placing the tag in the 3-D DRAM incurs multiple accesses for the tag and erodes 3-D DRAM's bandwidth advantage. In Section 3.2.4, I conclusively show that the tag bandwidth, and not latency, is the problem for DRAM caches.

Previous papers address the tag challenge via two approaches. The first approach places the tag in the 3-D DRAM incurring little on-die area overhead and attempts to alleviate the tag bandwidth burden. However, either the 3-D DRAM bandwidth overhead remains high (e.g., MissMap [25]) or the memory bandwidth demand is high

due to high cache miss rate (e.g., Alloy cache [26]). The second approach places the tag on the processor die incurring little 3-D DRAM bandwidth overhead but either high memory bandwidth demand due to high cache miss rate (e.g., CHOP [27]) or high on-die area overhead (e.g., plain sub-blocking (*PSB*) [28], decoupled sector cache (*DS*) [29]). The on-die overhead can be addressed by using a small on-die tag cache (*T$*, pronounced as 'T cash') [30, 31] while the full tag (and data) stays in the 3-D DRAM. Table 3.1 summarizes this multi-dimensional space.

Surprisingly, our results show that the old *PSB* with a *T$* performs better than the other above proposals. (While most previous work does not compare to *PSB*, the original MissMap paper does but its addendum with corrections to MissMap performance does not.) However, *PSB*, with or without prefetching [32,33], populates only the accessed blocks within a *page* while the rest of the page remains unused (i.e., there are "holes" in the cache); pages are large superblocks (e.g., 2 KB, not related to virtual memory). Consequently, *PSB* diminishes the effective cache capacity (only a third-fourth of a page is populated, on average). Conventional caches and *DS* avoid the holes but spread a page's blocks across multiple sets, destroying page-level locality of tag metadata (i.e., set mapping is at block granularity). Decoupled compressed cache (DCC) [34] performs set mapping at page granularity but destroys metadata locality by scattering a page's blocks among those of the other pages (ways) in the set. See Table 3.1.

This paper addresses the challenge of avoiding holes while exploiting page-level metadata locality. This locality is the dominant type of locality remaining at the DRAM cache level while the upper-level caches capture temporal and block-level spatial locality. While typical 64-B blocks sufficiently exploit spatial locality in the data, tag metadata of a single block is too small to amortize the 3-D DRAM bandwidth cost of tag access. Amortizing this cost by exploiting page-level metadata locality fundamentally depends on the DRAM cache and *T$* organizations. I emphasize that the issue here is *not* the prefetch latency effect of the page-level fetch but

Table 3.1.
DRAM cache design issues

	MissMap	Alloy	CHOP	*PSB + T$*	*DS + T$*	DCC + *T$*	*β$ + T$*
Main-memory BW (DRAM Cache miss rate)	-	×	×	×	-	-	-
DRAM Cache BW (High tag volume)	×	-	-	-	-	-	-
DRAM Cache BW (Low spatial locality)	-	-	-	-	×	×	-
DRAM Cache holes (Reduced capacity)	-	-	-	×	-	-	-

the bandwidth effect because the problem is bandwidth and not latency (prefetching improves latency but not bandwidth).

I propose *Bandwidth-Efficient Tag Access (BETA) DRAM cache* (*β$*, pronounced as 'beta cash') which decouples pages and blocks to reduce the number of holes and performs set mapping at page granularity like previous approaches, but exploits page-level metadata locality unlike previous approaches. Our novelty is *not* tag caching or page-block decoupling, but our cache organization's features to exploit this locality. While *T$* and *β$* hits do not impose any metadata bandwidth overhead, reducing the overhead of *T$* and *β$* misses is one of our two goals. To that end, *β$* employs two features so that the *T$* can fetch an entire page's metadata in as few 3-D DRAM accesses (DRAM row hits) as possible. Our second goal is to improve the *T$*'s effective size in the presence of page-block decoupling, for which *β$* and *T$* each employ a feature to reduce the amount of metadata in the *T$*. Note that the effective *T$* size affects the *T$* miss rate which in turn affects the 3-D DRAM bandwidth demand.

The first overhead is the number of 3-D DRAM accesses per *T$* miss. Page-block decoupling scatters a page's blocks among those of the other pages in the set and the correspondingly scattered metadata would incur many 3-D DRAM accesses. Instead, *β$* co-locates the metadata of a page via a set of forward pointers from a page to its (unordered) blocks. This co-location, *β$*'s first novel feature, enables *β$* both to avoid holes and to preserve metadata spatial locality, unlike *PSB*, *DS*, and DCC which achieve one but not the other. While null pointers for absent blocks impose a small space overhead (i.e., holes in the tag), this overhead is much smaller than that due to *PSB*'s holes in the data (and tag), especially considering the large 3-D DRAM.

The second overhead is the number of 3-D DRAM accesses per *β$* miss. Because multiple pages share a *β$* set (set mapping at page granularity), the set is large and accessing all of the set's metadata upon *β$* misses to find free space would incur many 3-D DRAM accesses. Instead, *β$* tracks the per-set free blocks in a free list, *β$*'s second novel feature, implemented as a compact vector of pointers and cached in the *T$* requiring no 3-D DRAM accesses in the common case.

To improve the *T$*'s effective size (our second goal), I address the amount of metadata in the *T$*. *β$*'s forward pointers for page metadata co-location helps *T$* miss bandwidth penalty but exacerbates this overhead. I make the key observation that the 3-D DRAM has plentiful capacity but insufficient spare bandwidth for tag metadata whereas the *T$* has sufficient bandwidth but limited capacity. *β$*'s null-pointer space overhead is relatively small for the large 3-D DRAM but relatively large for the small *T$*. Consequently, our idea is to switch dynamically from *β$*'s forward pointers to reverse pointers in the *T$* as the tag metadata crosses the *β$*-*T$* interface; this switching is *T$*'s novel feature.

Finally, *β$* also employs a feature to reduce the amount of metadata in the *T$* which is impacted by allocation of space in *β$*. Fine, block granularity for allocation wouid increase the amount of metadata in the *T$* (e.g., reverse pointers) whereas coarse, page granularity would default to plain-sub-blocking and incur holes. Instead, *β$* allocates at the granularity of a few blocks, called *chunks*, *β$*'s third novel feature,

which balance the metadata amount and the number of holes; each chunk is not merely larger than a block but groups *unordered* blocks of a page for maximum choice in block placement within the set. Pages, blocks, and chunks are granularities for tagging, transfer, and space allocation, respectively. Each of *PSB*, *DS*, and DCC decouple only two, albeit different two, of the three granularities: *PSB*'s tagging and allocation granularities are a page and transfer granularity is a block; and *DS*'s and DCC's tagging granularity is a page, and allocation and transfer granularities are a block. In contrast, β\$ is the first design to decouple all three granularities: tagging granularity of a page, transfer granularity of a block, and allocation granularity of a chunk. Because the allocation granularity is a chunk, β\$'s free list tracks free chunks.

In summary, for effective tag caching, (1) β\$ exploits page-level metadata locality to reduce the 3-D DRAM bandwidth cost of *T\$* misses and DRAM cache misses (our first goal) by:

- co-locating the metadata of a page, and
- tracking the per-set free chunks;

and (2) β\$ and the *T\$* reduce the amount of on-die metadata to improve the *T\$*'s effective size (our second goal) by:

- dynamically switching from β\$'s forward pointers to reverse pointers in the *T\$* at the β\$-*T\$* interface, and
- allocating space in β\$ at the chunk granularity.

Using simulations, I show that due to β\$'s tag bandwidth efficiency, β\$ with a *T\$* performs 15% better than the best previous scheme with a similarly-sized *T\$*.

The rest of the paper is organized as follows. Section 3.2 discusses the key challenges and opportunities in 3-D DRAM cache design. Section 3.3 describes the organizations of β\$ and *T\$*. Section 3.4 describes our evaluation methodology. In Section 3.5, I evaluate performance using simulations of commercial workloads.

3.2 Opportunity and Challenges

Though 3-D DRAM promises lower latency and higher bandwidth than memory, there are fundamental performance limits due to technology constraints and access characteristics.

3.2.1 Technology constraints

While expecting high bandwidth for the 3-D DRAM is reasonable (say 4-5x of main memory), assuming larger surpluses is unrealistic because of higher costs *without* corresponding performance benefits which are fundamentally limited by DRAM cache miss rates (discussed next). Increasing the DRAM cache bandwidth requires larger numbers of TSVs (or interposer pads for 2.5-D stacking) and a larger bandwidth from the 3-D DRAM. While adding TSVs may be viable (8x pin bandwidth is reasonable), the smaller size of the 3-D DRAM than main memory has two opposing effects: On one hand, smaller arrays cannot be banked as aggressively as larger arrays due to density degradation (which directly affects cost), which hurts bandwidth. On the other hand, smaller arrays are faster, which helps bandwidth. The former effect requires balancing area (density) and bandwidth. A single-banked design is optimal for area, but is choked for bandwidth. 16 banks for 32-GB main memory is a reasonable compromise between bandwidth and area with a penalty of 10% area over the unrealistic single bank, as per CACTI [35]. However, increasing the banking in a smaller 3-D DRAM well beyond that in the larger memory degrades density (e.g., increasing the banking by 4x to 64 banks for a 128-MB DRAM degrades area by more than 10% over 16 banks, as per Cacti for the 32 nm technology node). As such, I assume a banking increase of 2x. Fortunately, the second effect of faster banks yields 2-2.5x lower bank occupancy resulting in an impressive 4-5x higher bandwidth for the 3-D DRAM than memory.

On the latency front, 3-D DRAM offers net latency improvements of 1.7-2x over main memory [25], which is smaller than that seen at other levels of the hierarchy

(2-2.5x faster banks amounts to 1.7-2x net lower latency due to latency overheads involving the controller and DRAM periphery). At other levels, the improvement is 5x assuming typical latencies of 3 cycles for L1, 15 cycles for L2, 75 cycles for L3 cache, and 350 cycles for memory.

3.2.2 Access characteristics

Assuming the DRAM cache to be an L4 cache, most temporal locality and block-level spatial locality is captured by L1-L3 caches leaving mostly page-level spatial locality at the L4 level. Later, I show that for commercial workloads, the miss-ratio is typically around 35% which fundamentally limits the overall achievable bandwidth and latency improvements.

Now, consider a system without a DRAM cache that is memory-bandwidth bound. After adding a DRAM cache that yields a high miss-ratio – say 33% – the main memory must still serve every third request (i.e., the cache misses). Thus, the bandwidth headroom allows for at most a tripling of instruction throughput. Further, the true bandwidth demand (i.e., ignoring bandwidth overheads of metadata) on the DRAM cache is twice that on main memory (because the cache serves the other 67% of accesses). Even assuming non-trivial bandwidth overhead for metadata accesses, significant spare 3-D DRAM bandwidth remains (e.g., 3x of 3-D DRAM's 5x bandwidth). Tag caches spend this spare bandwidth on tag cache misses to achieve smaller on-die tag. However, both the DRAM cache and *T$* organizations should exploit metadata spatial locality to avoid overwhelming this spare bandwidth.

3.2.3 Tag Metadata Bandwidth Challenge

Recall from the Section 3.1 that the tag metadata of the three cache configurations, conventional cache, *PSB*, and *DS*) is too large to be on-die. The straightforward option of holding the metadata in the large 3-D DRAM would increase the 3-D DRAM bandwidth demand due to the metadata accesses. One may think that the tags can be looked up in one access followed by a data access so that the two 3-D DRAM

lookups per access would comfortably fit within the 3-D DRAM's 4-5x bandwidth. However, the bandwidth demand is far worse in practice.

Even on hits, multiple tags and coherence state metadata must be read which may require multiple 3-D DRAM accesses. Change to replacement information (e.g., NRU bits) and coherence state must also be written back. The bandwidth demand is higher on misses with additional 3-D DRAM accesses to account for line fills, victim reads (for dirty blocks), and tag/coherence-state updates, as summarized in Table 3.2. This large number of accesses places a high demand on the 3-D DRAM bandwidth.

I clarify that the coherence state in the DRAM cache is needed in most systems. While multi-socket systems would have coherence state at the DRAM cache, even single-socket systems would for many reasons. Single-socket multicores with per-core private L1, L2, and L3 caches would naturally place the coherence state at the shared L4 DRAM cache (in addition, the L4 may also hold directory information). Alternately, single-socket multicores with private L1 and L2 and a shared L3 (holding L1-L2 coherence state) would still have coherence state at the shared L4 DRAM cache to allow cache-coherent DMA for I/O devices; disallowing such DMA would require changes to device drivers of all the I/O devices to enforce software coherence. Further, the same microprocessor part is often used in both single- and multi-socket systems for easy extensibility and amortization of design, test, and verification costs (e.g., Intel Xeon).

3.2.4 Opportunity

Figure 3.1 isolates the impact of memory latency and bandwidth for our commercial workloads. Using configurations without 3-D DRAM caches, I compare a baseline memory system (*Baseline*, shown as line at 1.0) against memory systems (1) with double the bandwidth and same latency (*Double-BW*), (2) with half the latency (*Half-latency*), and (3) with infinite pin and device bandwidth and same latency (∞-*BW*). While halving the latency has the side-effect of doubling the bandwidth, *Double-BW* performs only modestly worse than *Half-latency*, showing that the im-

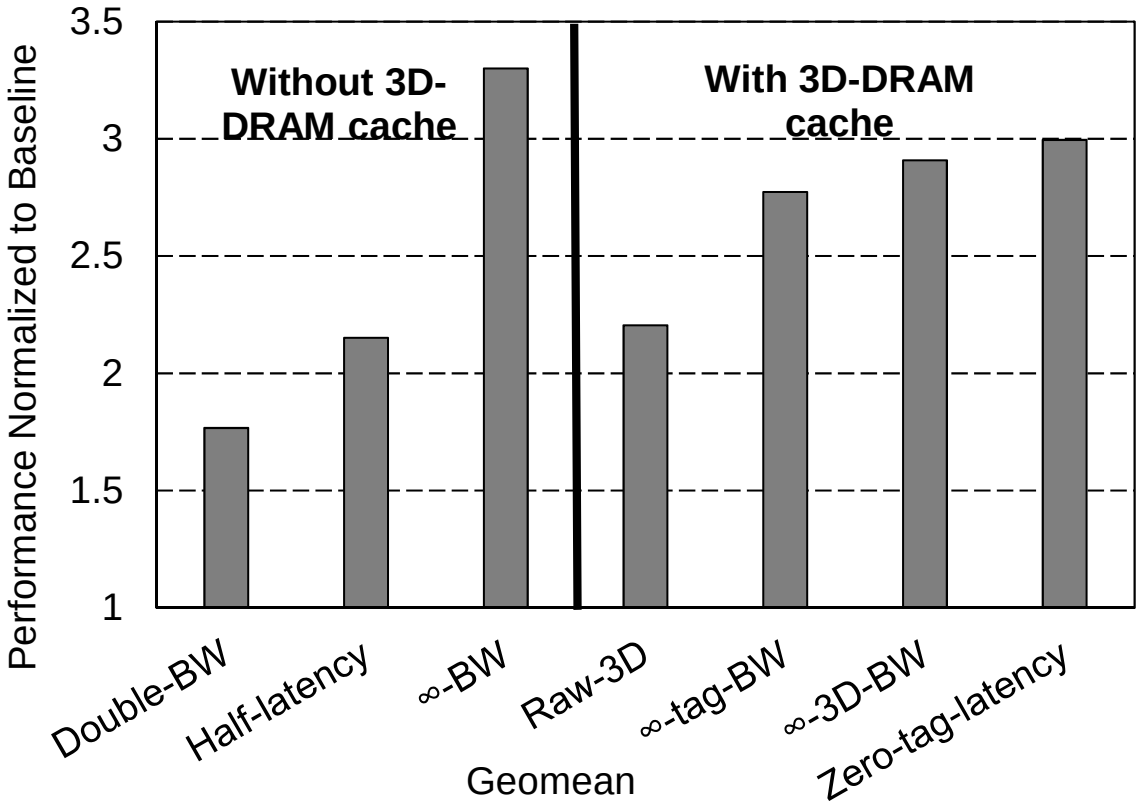

Fig. 3.1. Impact of bandwidth and latency

pact of latency is small. However, *Double-BW* performs better than *Baseline*, and ∞-*BW* performs much better than *Half-latency*, showing the high impact of bandwidth. Therefore, the problem is memory bandwidth and not latency (not surprising because die stacking addresses pin bandwidth).

Figure 3.1 also shows the impact of 3-D DRAM's latency and bandwidth on our workloads. I show conventional 3-D DRAM caches with tags in the 3-D DRAM, 64-byte blocks, and half the latency of *Baseline* (as seen in Section 3.2.1) configured (1) with 5x the bandwidth of *Baseline* (*Raw-3D*)[1], (2) with 5x the bandwidth for data and infinite pin and device bandwidth for tag (∞-*tag-BW*), (3) with infinite pin and device bandwidth for tag and data (∞-*3D-BW*), and (4) with zero latency for tag

[1]The device bandwidth is 5x because of 2x more and 2.5x faster banks than main memory while the pin bandwidth is 8x of main memory

while retaining half the latency for data, 5x the bandwidth for data and infinite pin and device bandwidth for tag (*Zero-tag-latency*). *Raw-3D* is better than *Baseline*, highlighting 3-D technology's benefits without any architectural innovations. However, ∞-*tag-BW* is better than *Raw-3D*, showing that the tag bandwidth overhead is so high that even *Raw-3D*, which has 5x higher bandwidth than main memory, cannot absorb. The difference between *Raw-3D* and ∞-*tag-BW* is the opportunity for architectural innovations. ∞-*3D-BW* and ∞-*tag-BW* are close showing that this opportunity does not increase with more 3-D DRAM bandwidth due to main memory bandwidth saturation (Section 3.2.2). Finally, *Zero-tag-latency* is only modestly better than ∞-*tag-BW*, showing that tag latency does not significantly impact performance.

3.2.5 Previous proposals

Being a direct-mapped cache, Alloy [26] reduces the metadata accesses to only one tag but incurs high miss rate for important, memory-intensive, commercial workloads. MissMap [25], which holds only presence information on-die and relegates the rest of the metadata to the DRAM cache moves in the right direction to reduce the 3-D DRAM bandwidth demand. However, MissMap is insufficient because (1) it reduces the bandwidth penalty only on misses, and (2) even on misses, many components such as line fills, victim reads (for dirty blocks) and coherence state updates are not avoided (see Table 3.2).

While tag caching would filter some metadata accesses, previous cache organizations incur many 3-D DRAM accesses upon *T$* misses due to lack of metadata spatial locality. Both conventional and *DS* caches place consecutive blocks from the same page in consecutive sets. In Figure 3.2, *page X*'s blocks, *X0, X3, X6* and *X7*, are in different sets. Such set mapping destroys row-locality for spatially-close accesses.

A recent cache compression design, DCC [34], uses page-level set mapping which ensures that blocks of a page remain on the same set (and thus the same DRAM row). However, the metadata of blocks of one page are interspersed with that of

Table 3.2.
Comparing 3-D DRAM accesses across cache designs

Cache	Cases	Number of 64-B accesses				
		Tag+coherence	Data	Tag/coher. update	Dirty vic read/fill	Total
Off-processor-die Conventional	Hit	2+	1	0	0	3+
	Miss (clean victim)	2+	0	2+	1	5+
	Miss (dirty victim)	2+	0	2+	2	6+
MissMap (assuming missmap hit)	Hit	2+	1	0	0	3+
	Miss (clean victim)	0	0	2+	1	3+
	Miss (dirty victim)	0	0	2+	2	4+
T$ $+\beta$*$*	Hit/Hit	0	1	0	0	1
	Hit/Miss (typical)	0	0	2	1	3
	Miss/Hit (typical)	2	1	2*	0	5
	Miss/Miss (typical)	2	0	4*	1+	7+

* Includes writeback of *T$* victims to β*$*

blocks of other pages (ways) in the set. In Figure 3.2, *page Y*'s blocks *Y0* through *Y2* are in the same set but dispersed across the ways and the metadata is similarly dispersed. Such dispersion necessitates multiple 3-D DRAM accesses. In contrast, *β$* (1) uses allocation granularity of chunks to balance the number of holes and metadata amount in the *T$*, (2) co-locates the metadata of a page, and (3) tracks per-set free chunks for efficient miss handling. Together, these features minimize the number of 3-D DRAM accesses needed for *T$* misses. Further, DCC employs three granularities for cache compression. Two of them are similar to pages and blocks; although they consider smaller 256-B pages in their context. The third sub-block granularity (16 B) is finer than block granularity and is used for compressing blocks. In contrast, our chunks are a capacity allocation granularity that is larger than blocks (e.g., groups of four blocks). Moreover, chunks are unordered groups of blocks as opposed to DCC's linearly-ordered sub-blocks. Not targeting 3-D DRAM bandwidth issues, DCC does not employ *β$*'s features without which performance degrades as I show in Section 3.5.3.

Prefetching of blocks within a page [32, 33] is an orthogonal optimization that can apply to any sub-blocking scheme including *β$*, *PSB*, and *DS*. More importantly, prefetching can improve only latency but not the average bandwidth, which is the main issue for 3-D DRAM caches. Later, in Section 3.5.3, I demonstrate that the latency benefit of prefetching is marginal.

3.3 Beta Cache (*β$*) and Tag Cache (*T$*)

While *β$* and *T$* have many similarities and a few differences, understanding *β$*'s organization eases understanding *T$*'s organization. Therefore, I start with *β$*.

3.3.1 Beta Cache (*β$*)

Recall from Section 3.1 that like *DS* and DCC, *β$* (1) decouples the pages and blocks to reduce the number of holes (e.g., 2-KB pages and 64-B blocks); and (2) performs set-mapping (i.e., indexing) at page granularity so that the blocks of a page

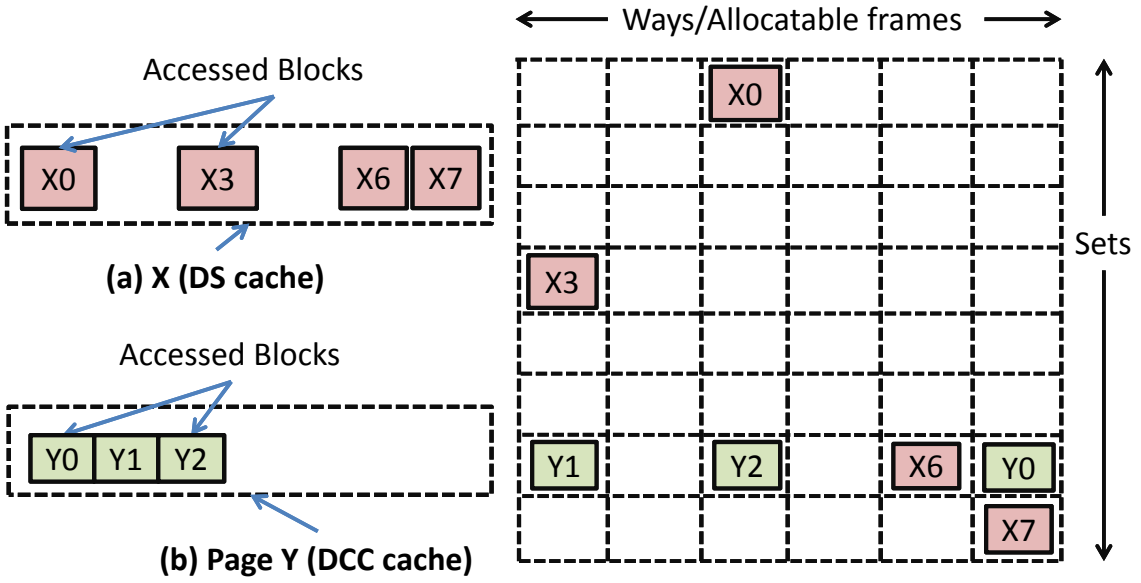

Fig. 3.2. Impact of organization on DRAM cache bandwidth (Example assumes 8 blocks per page)

are in the same set. However, unlike previous work, $\beta\$$ exploits page-level metadata locality with the goal of reducing the 3-D DRAM bandwidth overhead of $T\$$ and $\beta\$$ misses by (1) co-locating the metadata of a page and (2) tracking the per-set free space. Further $\beta\$$ also helps reduce the amount of metadata in the $T\$$ to increase the $T\$$'s effective size by allocating space at the chunk granularity.

Figure 3.3 illustrates $\beta\$$ for the same running example as in Section 3.2. For quick determination of page hit/miss, the page tags of a set are co-located, as shown by "Set Metadata" in Figure 3.3(a). To achieve page-block decoupling, the pages in a set share the set's space so that a page's blocks are scattered among those of the other pages.

Co-location: Due to this scattering, $\beta\$$ employs a vector of forward pointers from a page to its blocks to identify both which blocks are present and where they are placed within the set. Each forward pointer is associated with the block's coherence state, as shown by "Page X's Metadata" in Figure 3.3(b). As discussed in Section 3.1,

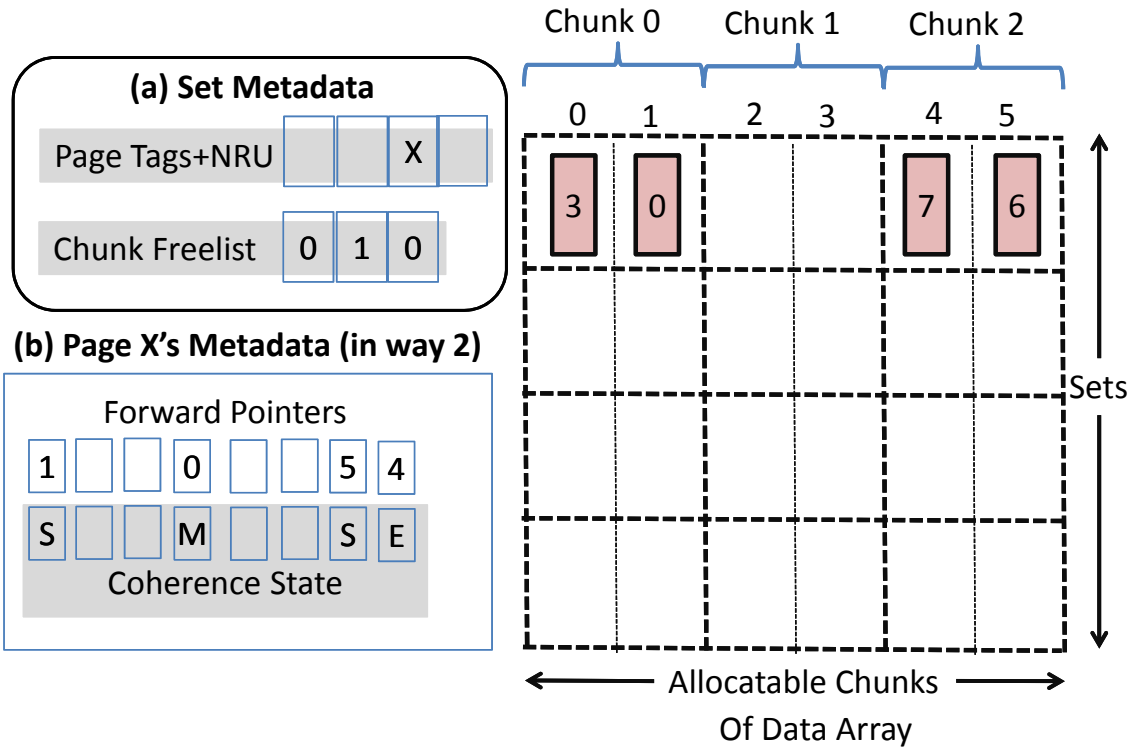

Fig. 3.3. Beta Cache ($\beta\$$) Organization

many of the forward chunk pointers are null due to absent blocks. While these null pointers impose a space overhead on the 3-D DRAM, the overhead is much smaller than that of holes in the cache (the null pointers are holes in the tag as compared to the holes in the data) and is relatively small for the large DRAM. On the positive side, the forward pointers enable co-location of the page's metadata, conserving the 3-D DRAM bandwidth (just one standard 64-B access per page for most interesting configurations).

To assess $\beta\$$'s features, I use a realistic configuration shown in Table 3.3. Assuming 32 blocks per page, I provision 256 frames per set of 16 associative ways, for an average of 16 frames per page. This provisioning works because, in typical commercial workloads, more than half the 2-KB pages are sparse with fewer than four 64-B blocks present. Figure 3.4 plots the cumulative block density distribution for our

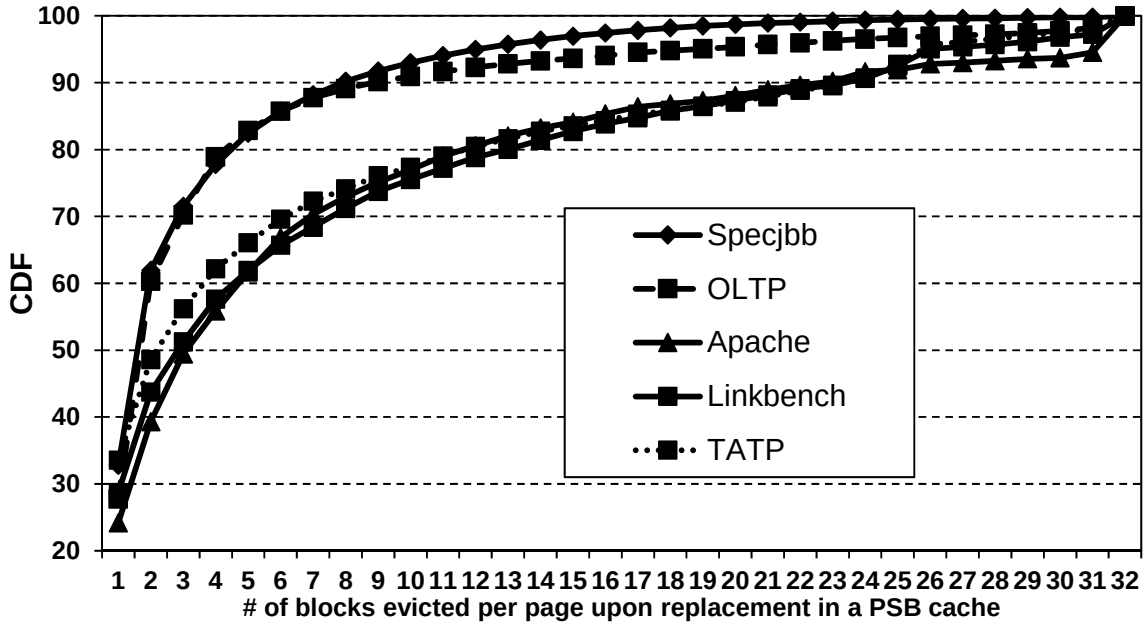

Fig. 3.4. Block density distribution

commercial workloads described in Table 3.4. Increasing the associativity would allow us to tighten the number of frames allocated per page to be closer to the statistical per-page average, around 8 blocks per page for our workloads, by sharing a set's frames among more pages and avoiding holes within each set (a law-of-large-numbers effect). However, higher associativity, past the point of diminishing returns on conflicts misses, also implies more accesses to look up the page tags of a set for page hit/miss determination.

For the example configuration. the per-page metadata amounts to 56 bytes including the null pointers (see row **A** of Table 3.3). Without the co-location via forward pointers, a page's metadata would use reverse pointers from the block to the page (one of the associative ways) and be spread over the entire set spanning 256 frame' metadata, as in DCC. In our example, this option would amount to 480 B which would have to be accessed to obtain the page's much smaller metadata (8 DRAM accesses, as shown in row **B** of Table 3.3).

Chunks: Recall from Section 3.1 that the *T$* uses reverse pointers to avoid *β$*'s null-pointer overhead which is significant for the small *T$*. However, allocating space at the granularity of only one block would imply a reverse pointer per block in the *T$* which still amounts to high space overhead. Therefore, *β$* allocates space at the chunk granularity (e.g., 4 blocks per chunk) upon a miss.

Accesses may miss in the page (i.e., the entire page is missing), or in a block of a page that is present. A page miss results in the (full) eviction of a victim page and the freeing of its chunks. Upon a block miss, *β$* allocates a chunk either by reclaiming a free chunk (from a previous eviction of a page which vacates all its chunks) or by partially evicting a chunk of a page in the set if no chunk is free. In either case, the page's metadata records a forward pointer to the first frame within the allocated chunk for the missing block, as shown by "Page X's Metadata" in Figure 3.3(b). Once a chunk is allocated, the subsequent blocks are allocated within the chunk in left-to-right order. For maximum flexibility in using the set's space, the blocks of a page may be located in any order and are discontiguous.

Chunking may cause a few holes in *β$* (data array) due to internal fragmentation but so significantly reduces the *T$*'s space overhead that the trade-off is justified. I show an example of how much the overhead reduces in Section 3.3.2.

Per-set free lists: Upon a block miss, determining which, if any, chunks are free requires examining the metadata of all the pages in the set which would incur many 3-D DRAM accesses. To alleviate this problem, I propose to maintain a free list per set as a bit vector, as shown in "Set Metadata" in Figure 3.3(a) (i.e., 64-bit vector for our example of 64 chunks per set). Freeing of chunks upon full eviction results in the corresponding free-list bits being set and allocation of chunks results in clearing of the bits. Without the chunks, the free list would grow but may still fit within a 3-D DRAM access. However, this growth imposes significant space overhead on the small *T$*, as I will see in Section 3.3.2.

Table 3.3.
$\beta\$$ Configuration

Common Configuration Assumptions 2KB page; 32 blocks/page with 64B per block; Metastate overhead per block = 6b (stable and transient coherence states)		
$\beta\$$: 16-way associative, 256 frames per set (64 chunks of 4 frames each), 16K sets		
Row	Configuration	Description
A	Forward pointers	Pointers per page = *32*; 8b forward pointers to pool of 256 frames; Total per-page overhead = *32*×(8+6) bits = 56 bytes (fits within one 64-B DRAM access).
B	No colocation	4-b pointers to 16 associative ways (reverse pointers) and 5-b block offsets within page; Total metadata transfer = 256 × (4+5 + 6) bits = 480 B (requires eight 64-B transfers)
T\$: 32-way associative, 360 block metadata frames (90 chunks of 4 frames each) per set, 1K sets		
Row	Configuration	Description
C	Reverse-pointers + Chunks	5b reverse pointer (to one of 32 ways) for each of 90 chunks; 5 bit block-offset for each of 350 blocks; 90b freelist for 90 chunks; 64b freelists per $\beta\$$ set, at most 16 $\beta\$$ sets per *T\$* set; Total = $((5 \times 90) + (5 \times 360) + 90 + (64 \times 16) + (6 \times 360))/8 = 690.5$B
D	No reverse-pointers	9b Forward pointers (to point to one of 360 frames) for each of 32 blocks in each of 32 ways of set; 90b freelist for 90 chunks; 64b freelists per $\beta\$$ set, at most 16 $\beta\$$ sets per *T\$* set; Total = $((32\times32\times9)+90+(64\times16)+(6\times360))/8 =$ 1562.25B
E	No chunks	5b reverse pointer (to one of 32 ways) plus 5b block offset for each frame; 360b freelist for 360-frame *T\$* set; 256b freelists per $\beta\$$ set, at most 16 $\beta\$$ sets per *T\$* set; Total = $((10 \times 360) + 360 + (256 \times 16) + (6 \times 360))/8 = 1277$B
F	Page tags	4B page tag, 32 page tags per *T\$* set; Total = $(4 \times 32) =$ 128B

The remaining issues are replacement and row locality. β\$ tracks the replacement state (e.g., NRU) at the page granularity for full evictions and at the chunk granularity for partial evictions (i.e., evict any NRU chunk in the set). The per-set replacement state is co-located with the set's tags to reduce the bandwidth demand (see "Set Metadata" in Figure 3.3(a)).

Because a page stays in one set, β\$ enjoys row hits for (1) the metadata of not only all the page's blocks and but also the entire set (e.g., tags, replacement state, and free list) as well as (2) the data of the page (though the tag metadata and data would be in different rows). In contrast, conventional and *DS* caches spread the page over multiple sets and incur row misses (Section 3.2). β\$'s tag and data share the same 3-D DRAM with all of the tag preceding all of the data, both laid out linearly.

Finally, I saw above that β\$ incurs some holes in the cache (data array) to reduce the *T\$* size though page-block decoupling can eliminate most of the holes as shown in *DS* [29]. However, a detailed design trade-off study is left to future work while this work focuses on the first step of design and illustrates one good design point in the evaluation.

3.3.2 Tag Cache (*T\$*)

Because *T\$* is a cache for the tag metadata in β\$, *T\$*'s organization largely mirrors that of β\$ metadata (i.e., with page-based set mapping, and block metadata grouped into chunks). The *T\$* cache is significantly smaller than the β\$ metadata which implies that there is a many-to-one mapping of β\$'s sets to the *T\$*'s sets.

Organization: Due to the many-to-one mapping, the following items may be different in β\$ and *T\$*: (1) a page's associative way, (2) a page's replacement rank in the respective sets, (3) the chunks occupied by a page, and (4) the per-set chunk free lists. Items (1), (2), and (3) are needed for correctly updating a page's metadata in the β\$ upon eviction from the *T\$*. Accordingly, the *T\$* holds a page's way and rank in the β\$ (see "Page Tags" in Figure 3.5(a) for *page X*) and the chunk numbers in the β\$

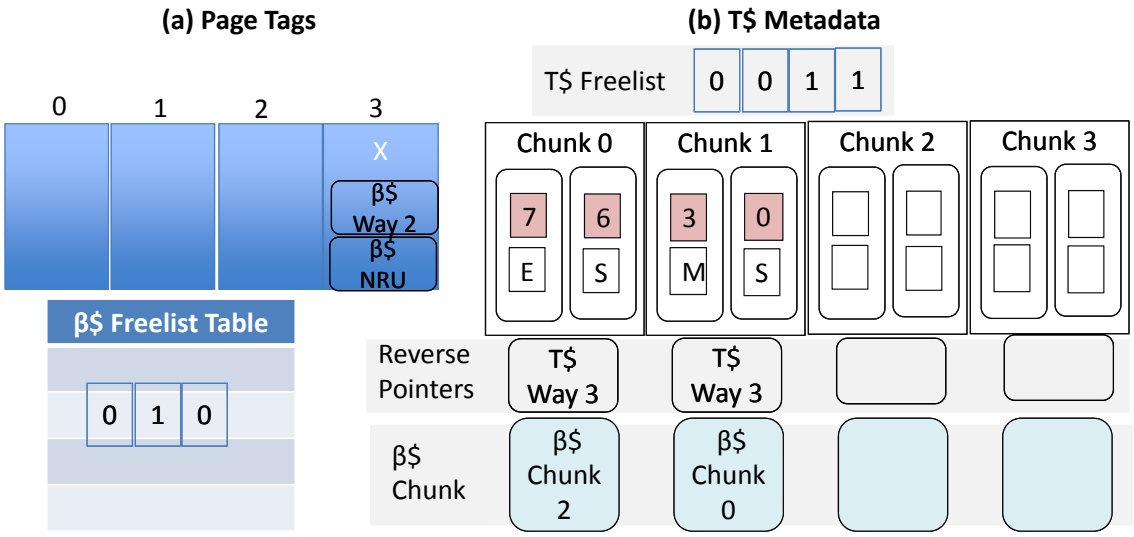

Fig. 3.5. Tag Cache ($T\$$) Organization

(see "$\beta\$$ Chunk" under "$T\$$ Metadata" in Figure 3.5(b)). The fourth item is needed for locating a free chunk upon a block miss. To that end, the $T\$$'s set holds the free lists of all the $\beta\$$ sets whose pages may map to an $T\$$ set (see "$\beta\$$ Freelist Table" under "Page Tags" in Figure 3.5(a)). Fortunately, the $T\$$ need not be provisioned for the worst case of as many $\beta\$$ sets as the $T\$$'s associativity. Because the $T\$$ has fewer sets than $\beta\$$ and the same associativity, set-mapping rules imply that only a few $\beta\$$ sets can map to the same $T\$$ set (e.g., though the $T\$$'s associativity is 32, if the $T\$$ is 16 times smaller than $\beta\$$ then the $T\$$ needs to hold at most 16 $\beta\$$ free lists).

Reverse-pointers: Another issue is that $\beta\$$'s co-location feature (wherein co-location of page metadata via forward pointers from the page to the blocks incurs many null pointers which impose significant space overhead on the $T\$$ (e.g., each page's forward pointer metadata in row **A** in Table 3.3 is 56 bytes of which nearly 65-75% is null based on average page occupancy of between a third and a fourth in Figure 3.4). To avoid the null pointers, the $T\$$ uses reverse pointers from the blocks to the page, as shown by "Reverse Pointers" under "$T\$$ Metadata" in Figure 3.5(b). In addition,

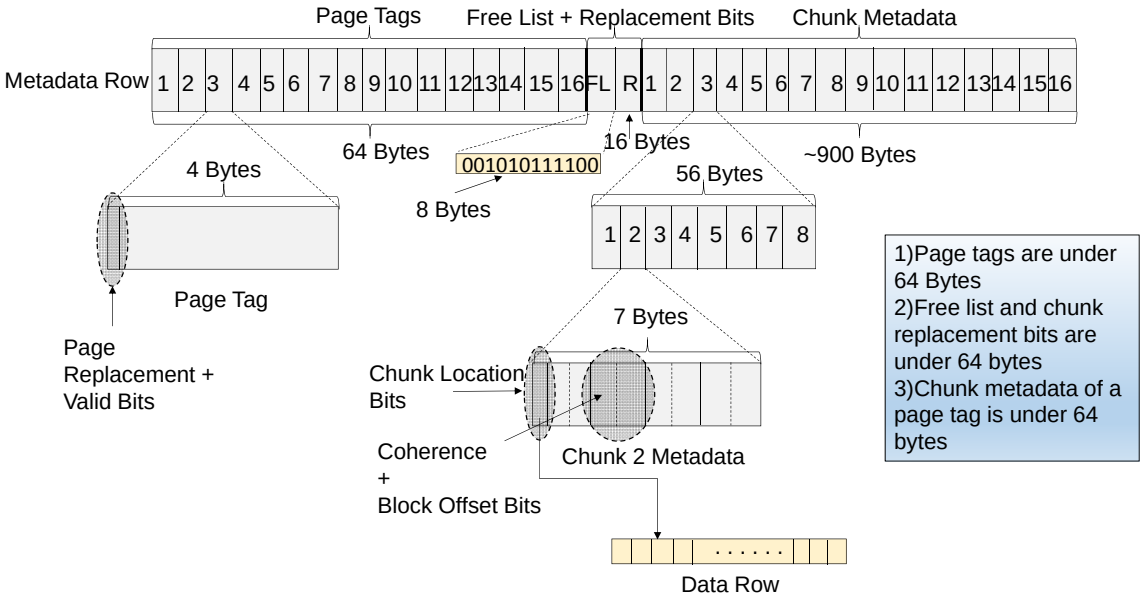

Fig. 3.6. Metadata layout in the 3-D DRAM row

the use of reverse pointers also necessitates saving block-offsets with the metastate to identify which block the metastate belongs to (e.g., the block positions, 3, 0, 7 and 6 in Figure 3.5(b)). The use of reverse pointers reduces the metastate overhead by 56% from 1562 Bytes to 690 Bytes as shown in Rows **D** and **C** of Table 3.3. Because the *T$* is on-die, access bandwidth to the reverse pointers is not an issue, unlike the off-die β*$*. As the tag metadata transits between the *T$* and β*$*, the forward pointers in β*$* are changed to reverse pointers in the *T$*, and vice versa.

Recall from Section 3.3.1 that the key motivation for chunking is to reduce the *T$* metastate overhead. Indeed, the per set overhead of *T$* bloats by 1.85X in the absence of chunking as shown in rows **E** and **C** (1277B versus 690B). The overall reduction is a combination of (1) a linear reduction (proportional to degree of chunking) in the freelist overhead and reverse-pointer overheads and (2) no reduction in block-offset and coherence overheads which are unaffected by chunking.

Figure 3.6 describes the metadata layout in the DRAM row. The page tags are stored together followed by free-list and chunk replacement information. The chunk

metadata associated to each page tag is laid out at the end. It is important to note that all page tags, the free-list and chunk replacement bits, and chunk metadata for the matching page tag (or victim page tag) can be transferred in 3 memory transactions. Great care is taken to ensure that the metadata layout is done in a way that minimizes the number of memory transactions needed to fetch the required metadata.

Operation: The *T$* and the metadata in the *β$* operates like a traditional 2-level hierarchy wherein addresses are first looked up in the *T$* before accessing the *β$* (in case of a miss in the *T$*). Because of the two-level organization, each access can face one of four possible outcomes (hit/miss in the *T$*, hit/miss in the *β$*). The number of DRAM accesses for each of these four cases is summarized in Table 3.2. Note, not all cases are equally likely; hits are common and misses are uncommon in each of the two levels. Later, in Section 3.5.1, I show that *β$* + *T$* reduces the total number of transfers from 3D-DRAM.

A key invariant I enforce in the *T$*'s operation is that for the pages present in the *T$*, the block information in the *T$* is the same as that in the *β$*. That is, the *T$* maintains full and current information about a page. This invariant guarantees that for any given access, a page hit in the *T$* ensures that the block hit (miss) in the *T$* is a hit (miss) in the *β$* as well. The rationale for this invariant is that (1) it prevents unnecessary accesses to the *β$*, and (2) it matches the *T$*'s goal of exploiting page-level spatial locality by fetching (and maintaining) full page information.

As with any tag-caching scheme, the *T$* requires page metadata invalidations whenever a page is evicted from the underlying cache (*β$*, in our case). Further, to enforce the invariant, I require one additional safeguard – in the case of partial replacement of a chunk of a page, the update must be actively propagated to the corresponding page entry in the *T$* to ensure that the metadata of the victim chunk is also evicted from the *T$*.

For example, eviction of a page results in the tag-cache entry being similarly evicted from the *T$*.

Accordingly, accesses to the *T$* and *β$* fall into **four** cases (in increasing order of difficulty): page hit and block hit (in both the *T$* and *β$*), page hit (in the *T$*) and block miss (in the *β$*), page miss in the *T$* and block hit in the *β$*, page miss in the *T$* and block miss in the *β$*.

First case: In the first case, the metadata is served completely from the *T$* without requiring any access to the 3-D DRAM. In the *T$*, the way pointers of all the chunks in the indexed set are matched to identify the relevant chunks (e.g., 4-bit way pointers of 64 chunks). Then the offsets of the matching chunks are searched to identify the relevant block. This matching can be simplified to a few sequential steps involving fewer matches per step (e.g., 4 groups of 16 chunks each). Such grouping costs only a few cycles compared to the 3-D DRAM data access of more than 50 cycles.

Second case: In the second case, if space is already available in a chunk that is not full (sub-case 1), the pre-allocated space is assigned to the block. If there is no free space in existing chunks (sub-case 2), a new chunk, preferably a free chunk, is allocated in both the *T$* and *β$*. If there were no free lists, locating the free chunks would incur many 3-D DRAM accesses to search through the forward pointers of all the pages in the set (e.g., 8 64-B accesses each of which covers 2 pages' forward pointers shown in row A of Table 3.3). If no free chunks are available in the *T$* and *β$* (sub-case 3), then a partial eviction occurs from the *β$* (i.e., only one chunk is evicted) and a full eviction from the *T$* (i.e., a full page is evicted); recall that there are no partial evictions , based on the replacement state for the respective sets in the *T$* and *β$*; from the *T$*. The partial *β$* eviction incurs a 3-D DRAM read of the coherence state in the *β$* possibly followed by a writeback of the partially evicted data to main memory and an update of the victim page's block metadata (typically two 3-D DRAM accesses). The *T$* full eviction (in sub-case 3) incurs a reverse-to-forward-converted metadata writeback to the *β$* (typically one DRAM access for the

page's tag and replacement state, another access for the page's co-located metadata, and one more for updating the free list vector). If the page's metadata were not co-located, the writeback would incur many 3-D DRAM accesses (e.g., 6 64-B accesses for the page's metadata scattered across the set as shown in row C of Table 3.3). While the writeback can be reduced with metadata dirty bits, replacement state and block occupancy often change and make the dirty bits ineffective (our implementation does not use metadata dirty bits).

Irrespective of the sub-case (i.e., presence or absence of free chunks), the second case is a block miss in the $\beta\$$ which requires a fill of the data from main memory into the DRAM cache. Assuming the $\beta\$$ (and $T\$$) is provisioned appropriately with enough chunks, free chunks would be available in the common case. As such, the first two cases (both page hits) correspond to the prevalent page-level spatial locality and therefore capture a majority of accesses (e.g., 80%) most of which require only a few or no 3-D DRAM accesses. Table 3.2 briefly summarizes the number of 3-D DRAM accesses for the two cases (rows "Hit/Hit" and "Hit/Miss"). While full/partial evictions from the $T\$$ and $\beta\$$ are uncommon in the second case, they do occur in the fourth case below where $\beta\$$'s mechanisms of chunking, co-location and free lists are more important.

Third case: In the third case of page miss in the $T\$$ and block hit in the $\beta\$$, there are typically two 3-D DRAM accesses for the block hit: one to read the page tags of the set in the $\beta\$$ and another to read the co-located per-page metadata. Recall that in a set in the $\beta\$$, the page tags are co-located for quick page hit/miss determination. In addition, the page miss in the $T\$$ requires a full eviction from the $T\$$ incurring a metadata writeback to the $\beta\$$ (like the second case above) and a page metadata fill via a forward-to–reverse conversion (see "Miss/Hit" row in Table 3.2; the transfers may increase in case of dirty evictions). This case is uncommon as it corresponds to the page being absent in the $T\$$ but being present in the $\beta\$$ which would not occur often in the presence of page-level locality and a well-provisioned $T\$$.

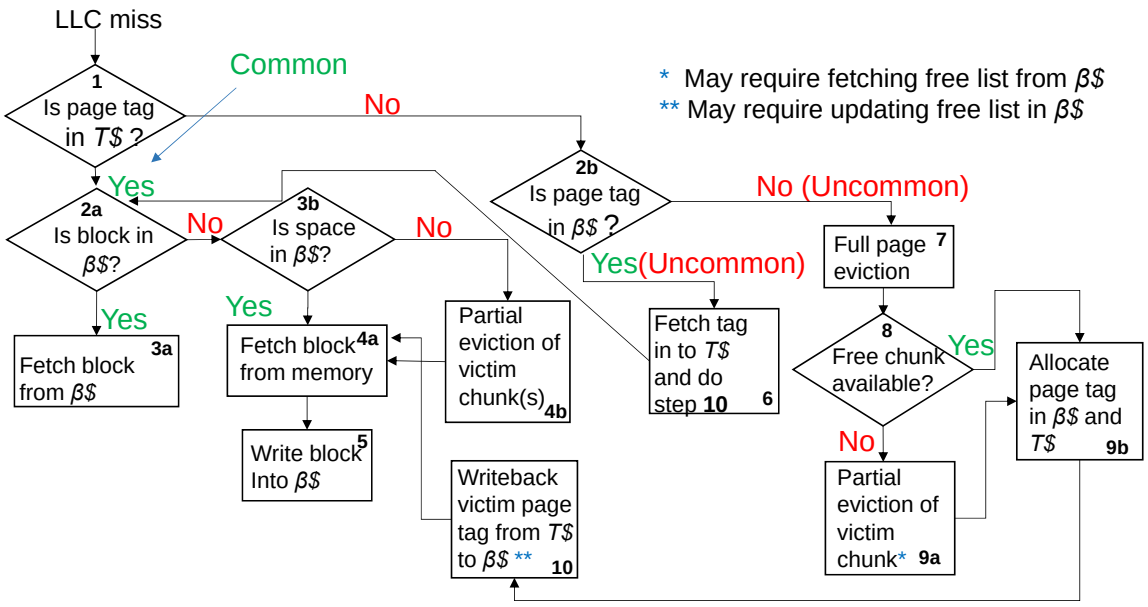

Fig. 3.7. Sequence of Operations on a $T\$$ Access

Fourth case: The fourth and most difficult case of page miss in the $T\$$ and block miss in the $\beta\$$ requires a full eviction in both the $\beta\$$ and the $T\$$. Determining the block miss in the $\beta\$$ typically requires a 3-D DRAM access to the co-located page tags and is followed by a full page eviction from the $\beta\$$. This full eviction typically requires a 3-D DRAM read for the victim page's co-located metadata followed by several reads of dirty data to be written back to main memory. The full eviction from the $T\$$ is similar to the third case above but without any metadata fill from the $\beta\$$ because the page is not in $\beta\$$. Instead, the $T\$$ starts the page with just the missing block. Like the second case above, this case is a block miss in $\beta\$$ which requires a fill of the data from main memory into the DRAM cache. Table 3.2 shows the fewest accesses in this "Miss/Miss" case; dirty block evictions would require more accesses.

Figure 3.7 summarizes the operations that take place upon an access to the $T\$$ in the form of a flowchart.

Table 3.4.
Benchmarks

Benchmark Description	Memory Footprint (GB)
Specjbb: version 2005, Java-based 3-tier client/server system workload with emphasis on the middle tier. Java server VB version 1.5 with parallel garbage collection. I simulate 2 JVMs each hosting 24 warehouses. I warmup the benchmark for 180,000 transactions and measure performance for 2,000 transactions	1.2
Online Transaction Processing (OLTP): models database transactions of a wholesale parts supplier. I use PostgreSQL 8.3.7 database system and DBT-2 test suite which implements TPC-C benchmark. I reduced number of items and districts per warehouse and customers per district to allow a larger number of warehouses. I use 8 databases each with 25,000 warehouses. I simulate 128 concurrent database connections per database. I warmup the databases for 80,000 transactions each before taking measurements for 200 transactions	64
Apache: version 2.2.9, a static web server workload. I use 8 webservers hosting a 160,000 files. SURGE is used to generate web requests by simulating 12800 clients, each with 25ms think time between requests. I warmup the webservers for 1,000,000 transactions each before taking measurements for 1,000 transactions	4
Linkbench: models database queries to a social graph e.g Facebook. I use MySQL 5.6.14 database system and the Linkbench social graph and query generator. I use 8 databases each with 5 million keys. I configure MySQL with a 4 GB buffer pool and InnoDB storage engine. I generate the social graph, load the keys, warmup the databases for 300,000 transactions, and perform our measurements for 500 transactions	48
Telecom Application Transaction Processing (TATP): models database transactions of a Home Location Register (HLR) database used by a mobile carrier. I use MySQL 5.6.14 database system and OLTP-Bench which implements the request generator in java. I use 8 databases each with a scale factor of 40. I use 40 client terminals and default weights. Each MySQL database is configured with an 4 GB buffer pool and InnoDB storage engine. I warmup the database for 300,000 transactions and take measurements for 500 transactions	40
Spec2006: I evaluate 4 spec programs (astar,omnetpp,mcf,libquantum) in rated mode (8 copies each). I use simpoints to identify a simulation interval. I warmup the caches for 250 million instructions and do measuremeants for 100 million instructions	2-6

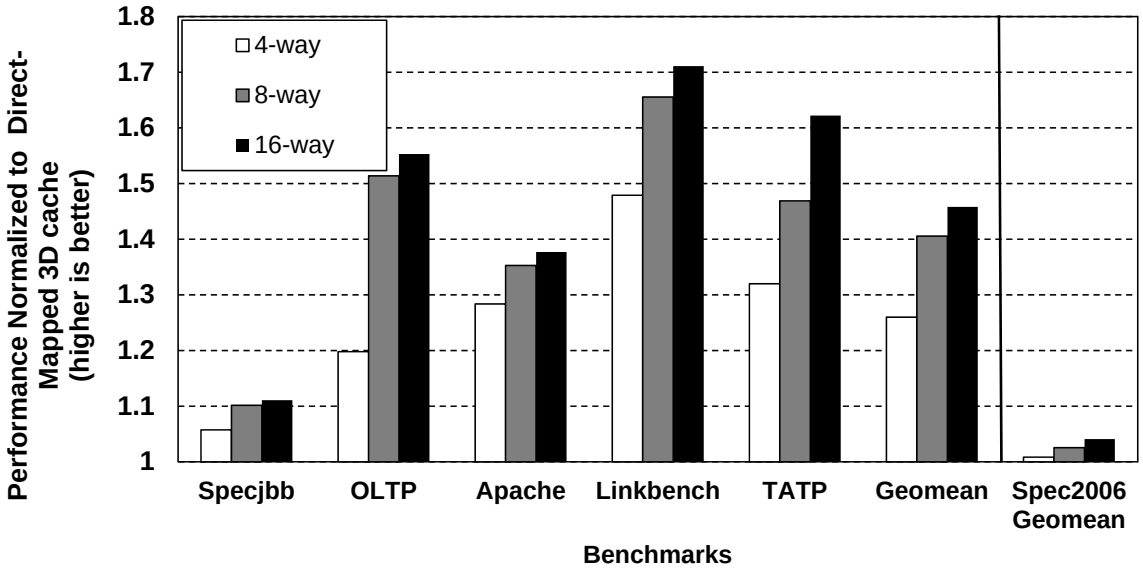

Fig. 3.8. Impact of Associativity on Performance

3.4 Experimental Methodology

I run full-system simulations using Wisconsin GEMS [11] built on top of Simics [12]. I simulate SPARC-based, single-socket multicores running Solaris 10.

Workloads For our main results, I simulate the commercial benchmarks shown in Table 3.4 which also shows their memory footprints. Following GEMS methodology, I fully warm up the caches before measuring performance to avoid cold-start effects. More than 90% of the simulated cache frames see several misses during each measurement confirming steady-state behavior. In addition, I also simulate multiprogrammed SPECPU 2006 benchmarks [36] to validate the behavior of previous proposals that have demonstrated performance benefits with scientific/engineering workloads.

System Configuration Table 3.5 lists the key parameters of the simulated system. The system uses 16 4-way SMT multicore organized as a 4x4 tiled architecture where each tile includes private L1/L2 caches and a slice of the shared L3 cache. To simulate the effects of TSVs, the 3-D DRAM uses twice as many channels, twice

Table 3.5.
Common System Configuration Parameters

Processors, On-chip Caches and Network	
Cores	16 (4-way SMT)
Core Frequency	3.2 GHz
Private L1I	64KB, 4-way, 3 cycles
Private L1D	64KB, 4-way, 3 cycles
Private L2	128KB, 4-way, 4 cycles
Shared L3	8MB, 16 banks, 16-way
Shared L3 latency	6 cycles (local bank)
Link latency	2 cycles
Interconnection network	4x4 Mesh
Die-stacked DRAM (256 MB)	
Bus frequency	1.6 GHz
Channels	4
Ranks	1 per channel
Banks	8 per rank
Bus width	128 bits per channel
tCAS-tRP-tRCD-tRAS	9-9-9-27
Main Memory (64 GB)	
Bus frequency	800 MHz
Channels	2
Ranks	1 per channel
Banks	8 per rank
Row buffer size	2KB
Bus width	64 Bits per channel
tCAS-tRP-tRCD-tRAS	11-11-11-33

the bus speed, and twice the bus-width as main memory. Other timing parameters for the 3-D DRAM and main memory are shown in Table 3.5; the DRAM has 5x more bandwidth than memory (2x more banks and 2.5x shorter bank occupancy, as discussed in Section 3.2). Recall, the $\beta\$$ and $T\$$ parameters are shown in Table 3.3. I determine the associativity of the conventional DRAM cache empirically. As shown in Figure 3.8, commercial workloads see significant performance improvements as associativity (X-axis) increases up to 16. However, the impact of associativity in spec2006 programs is insignificant. For the commercial workloads, increasing the associativity beyond 16 resulted in insignificant performance gains. As such, all our experiments use a 16-way associative DRAM cache.

Table 3.6.
Tag Overhead (MB)

Components	Conventional	DS	Large Blk	Ideal CHOP	Ideal Alloy	MissMap	PSB	β\$
Total for 256-MB	17.5	10.2	0.5	0.5	16	17.5	3.5	15
Total for 1-GB	70	40	2.1	2.1	64	70	14	60
Total for 4-GB	280	160	8.5	8.5	256	280	56	240

Comparison with other cache organizations I compare the following schemes: (1) a conventional cache using large, 2-KB blocks; (2) an ideal version of CHOP [27] using 2-KB blocks with an experimentally determined migration threshold of 8 accesses, and an unbounded filter-cache to filter sparse pages without any loss of history; (3) an ideal Alloy cache using 64-B blocks with the tag metadata in the 3-D DRAM and perfect hit-miss prediction [26]; (4) MissMap with compound accesses to the tag metadata and data which are stored in the 3-D DRAM [25]; and (5) a *PSB* cache using 2-KB pages and 64-B blocks with tag metadata in DRAM cache and a *T\$* for caching the tag metadata; and (6) a β\$ cache and a *T\$* using 2-KB pages, 64-B blocks, and 4-block chunks. I model the bus and 3-D DRAM bank occupancy of non-critical-path events such as writeback of coherence and replacement state for all the caches, as these events impact the bandwidth demand on the 3-D DRAM, as discussed in Section 3.2.

3.4.1 Tag Overhead

Varying the DRAM cache size as 256 MB, 1 GB, and 4 GB, Table 3.6 shows the total tag overhead conventional 64-B-block cache, *DS*, conventional 2-KB-block

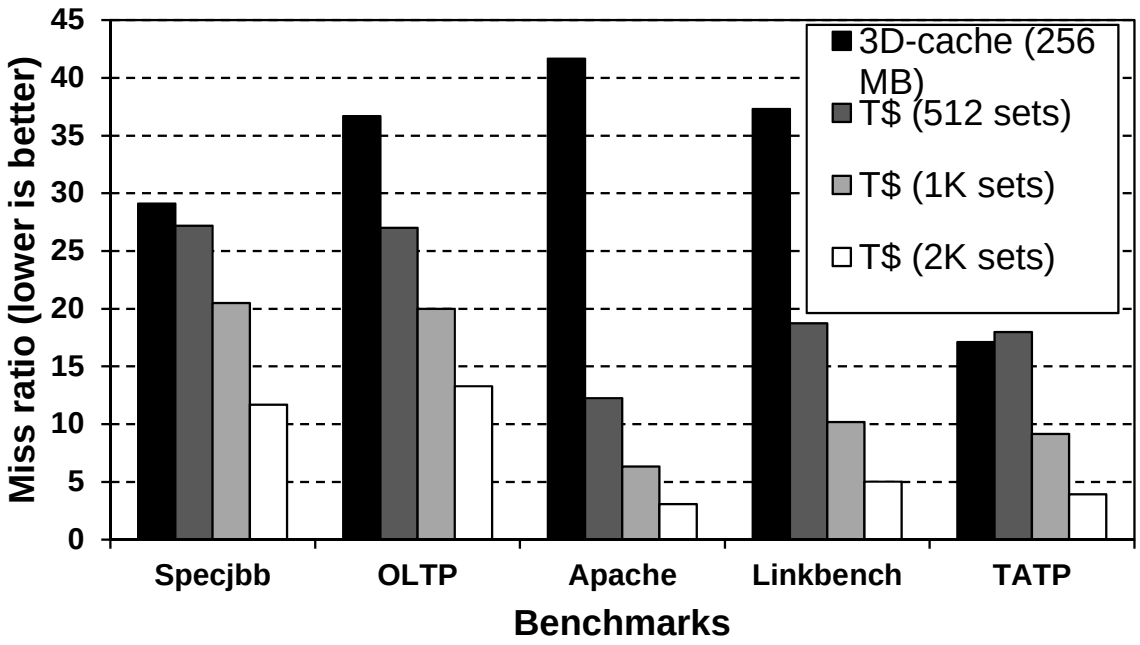

Fig. 3.9. *T$* miss rate

caches (Large Blk in table), ideal 2-KB-block CHOP [27], MissMap [25], ideal Alloy cache [26], *PSB*, and *β$*. The ideal variants are described in Section 3.4. The overhead includes tags, replacement state (NRU bits at the appropriate granularity), sub-block pointers or presence bit vectors (for *β$* and *PSB*), per-block coherence state, reverse (block-to-page) pointers (for *DS*) and free-lists (for *β$*). Excluding the large-block variants, the metastate for all other techniques is too large to include on the processor die. As such, our performance comparisons configure these schemes to use a tag cache to hold a subset of the total metadata on the processor die.

In Figure 3.9, I show the *T$*'s page miss rate for our benchmarks as I vary its size as 512, 1K (default),and 2K sets (410, 820, and 1640 KB in total size), which are 30x, 15x, and 7.5x fewer than the DRAM cache's sets. I also show the 256-MB DRAM cache's block miss rate for reference. While the *T$*'s page miss rate lowers as expected with its size, the more important point is that the *T$*'s page miss rate is

far better than the DRAM cache's block miss rate highlighting the key role of spatial locality in DRAM cache design.

3.5 Results

I compare the performance of $\beta\$ + T\$$ against several previous schemes. I first show that $\beta\$ + T\$$ outperforms the competition because of improved bandwidth efficiency. I then show that omitting $\beta\$$'s features results in increased bandwidth demand and hence lower performance. Finally, I show some results for sensitivity to 3-D DRAM bandwidth.

3.5.1 Performance

In Figure 3.10(a), I compare the performance of a conventional 2-KB-block cache (labeled 'Large Blk'), ideal CHOP, ideal Alloy cache, MissMap, ATCache, $PSB + T\$$, and $\beta\$ + T\$$, all configured as a 256-MB DRAM cache. The Y axis shows performance normalized to that of a conventional 64-B-block DRAM cache with all of its tag metadata in the 3-D DRAM. The X axis shows our benchmarks. To help understand the performance graph, I also include the total access latency incurred at the 3-D DRAM (Figure 3.10(b)) and at main memory (Figure 3.10(c)). The schemes compared are the same as those in Figure 3.10(a); with the following key differences. The Y-axis shows total access latency normalized to that of a conventional DRAM cache with 64 byte blocks and tags in DRAM. Each bar is subdivided into two components: the raw access latency at zero load and the queuing delay.

Conventional 2-KB-block cache and ideal CHOP both perform poorly due to main memory bandwidth wasted on transfering entire 2-KB blocks of which the useful data is around a quarter to a third (Figure 3.4). Compared to the conventional 2-KB-block cache, ideal CHOP does filter the bandwidth demand modestly. However, asFigure 3.4 shows, that the page occupancy is not bi-modal but rather has a long tail so that CHOP's bandwidth filtering is insufficient. For both these techniques, their high memory bandwidth demand manifests as large main memory queuing delays in Fig-

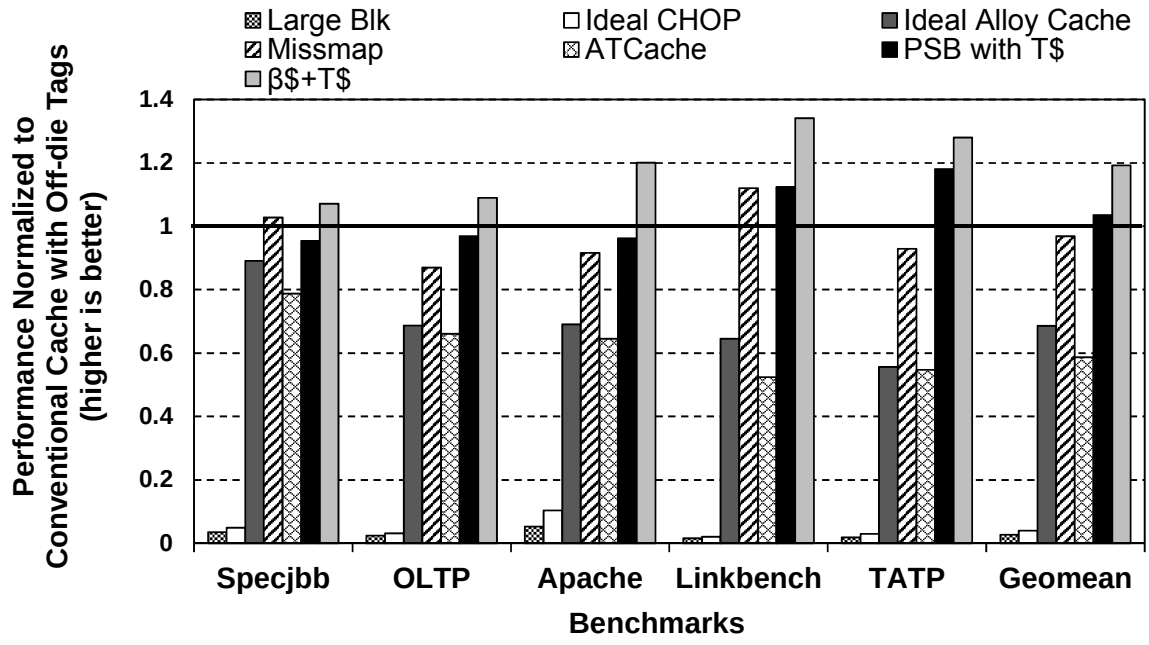

Fig. 3.10. Performance

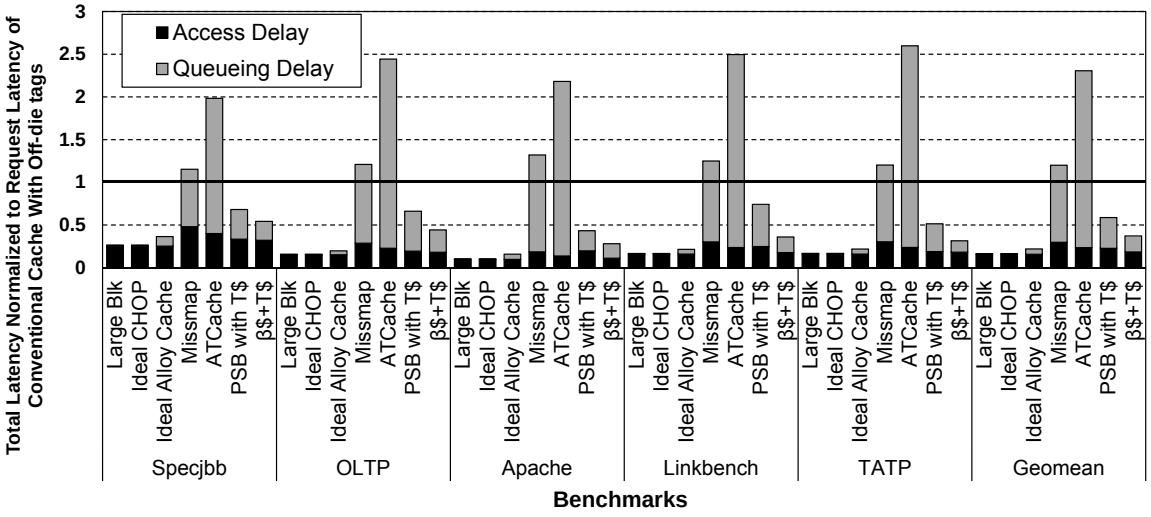

Fig. 3.11. Queuing delay for DRAM cache

ure 3.10(c). Ideal Alloy cache performs worse than the baseline due to higher DRAM cache miss rate, which translates to larger queuing delays at main memory. (Recall from Section 3.4 that Alloy cache's direct-mapped design incurs higher missrates for

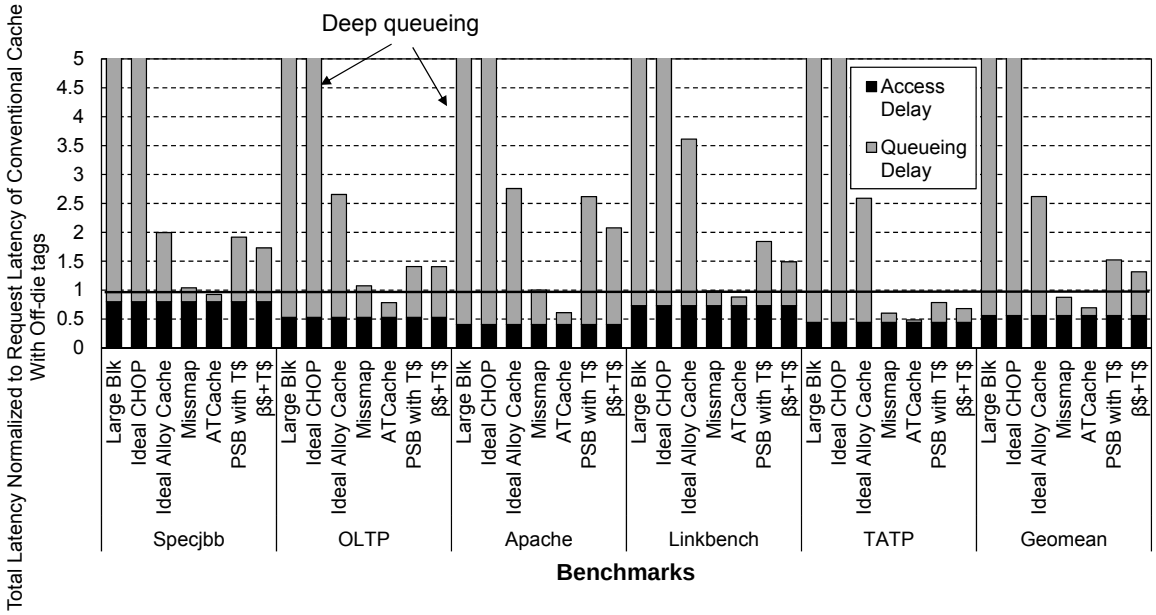

Fig. 3.12. Queuing delay for main memory

commercial benchmarks than for SPEC2006 benchmarks.) Because MissMap exerts similar 3-D DRAM bandwidth pressure for tag metadata as the conventional baseline, as seen in Figure 3.10(b), MissMap performs close to the baseline. Despite using a *T$*, the ATCache fares worse than the baseline because the ATCache employs set-based tag prefetches. Consequently, ATCache fetches a lot of useless tags (i.e., tags for blocks that will not be accessed) in to the *T$* thereby worsening the bandwidth pressure on 3-D DRAM. Also, the layout of ATCache is no different than a conventional cache. By spending only a little of the 3-D DRAM bandwidth on tag metadata because of their *T$* s, both *PSB* and β*$* perform better than the conventional baseline. However, *PSB* incurs holes in the DRAM cache and higher DRAM cache miss rate (shown next), which has two effects. (1) main memory bandwidth pressure goes up; and (2) the tag metadata activity increases driving up 3-D DRAM bandwidth pressure (more misses means more tag accesses). In contrast, β*$* incurs fewer holes due to page-block decoupling. As such, β*$* + *T$* enjoys lower 3-D DRAM and main

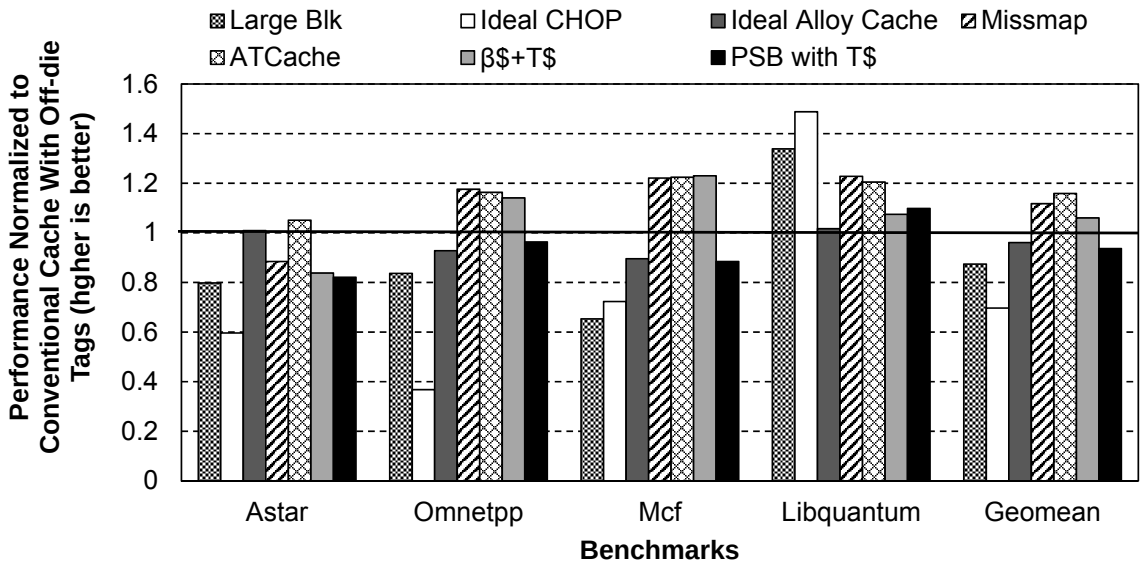

Fig. 3.13. Spec Performance

memory bandwidth pressure, which translates to 15% better performance than *PSB + T$*.

3.5.2 Comparison with SPEC2006 workloads

The above results are different than those in the previous papers primarily because of our commercial workloads which are significantly more memory-intensive than SPEC workloads. Figure 3.13 shows the performance of the various schemes on SPEC workloads. As with commercial workloads, performance is normalized to that of a conventional cache with 64-byte blocks (Y-axis). While SPEC workloads show some bi-modality in page occupancy which helps CHOP's filtering, our commercial workloads do not and hence our results are different from the CHOP paper's. With some SPEC benchmarks (see libquantum in Figure 3.13) there are many dense pages, which results in performance improvements for ideal CHOP and conventional 2KB-block cache. The MissMap's authors published a correction to the performance results in their paper but the addendum does not compare to our baseline though the original MissMap paper does. The Alloy cache paper uses multiprogrammed SPEC

Table 3.7.
DRAM Cache Misses per thousand instructions (MPKI)

Benchmarks	Conventional	Large Blk	Ideal CHOP	Ideal Alloy Cache	MissMap	ATCache	PSB+T\$	β\$ +T\$
Specjbb	2.2	2.5	3.1	5.8	2.1	2.2	3.2	2.5
OLTP	3.7	1.1	2.0	8.5	4.1	4.2	5.0	4.3
Apache	11	2.8	3.6	21	12	12	13.5	13
Linkbench	2.2	1	1.1	6.6	1.9	2.0	3.0	2.7
TATP	1.5	0.5	0.7	13	2.1	2.0	1.2	1.4

benchmarks whereas our commercial workloads are more memory-intensive. Our own SPEC comparisons do show that associativity is not as important for SPEC benchmarks. Finally, the results show that $\beta\$ + T\$$ is within 9% of ATCache, which is the best technique for SPEC benchmarks. However, for commercial workloads, $\beta\$ + T\$$ is more than 2x better than ATCache (Figure 3.10(a)).

The queuing delays shown in Figure 3.10(b) and Figure 3.10(c) are indirect measures of bandwidth demand. To provide more direct measures, I show both the DRAM cache miss rates (in MPKI) in Table 3.7, and the bandwidth demand (in normalized number of transfers) at both the DRAM cache and main memory in Table 3.8. Due to space constraints, Table 3.8 shows only the (geometric) mean bandwidth demand across all benchmarks.

Table 3.7 shows the miss rates in MPKI for the above schemes starting with the baseline conventional 64-B-block cache. While the miss rates for the large-block cache and ideal CHOP are significantly lower than the conventional design's, their main memory bandwidth demand is much higher due to the unwanted data in their large blocks. The ideal Alloy Cache has higher miss rates due to more conflict misses which translates to 190% increase in bandwidth demand from main memory (Table 3.8). By avoiding holes via page-block decoupling (Section 3.4.1) MissMap has low miss rates close to the conventional design's but performs slightly worse due to the slightly

Table 3.8.
Bandwidth demand at 3-D DRAM and Main Memory

	Normalized Bandwidth Demand (Conventional Cache with Off-die tags = 1.0)						
DRAM	Large Blk	Ideal CHOP	Ideal Alloy Cache	MissMap	ATCache	PSB+T\$	β\$ +T\$
3D-DRAM	$\gg$5	$\gg$5	0.4	1	2.2	0.7	0.6
Main Memory	$\gg$5	$\gg$5	2.9	1.1	1.0	1.4	1.2

eroded 3-D DRAM bandwidth advantage (Table 3.8). ATCache is a conventional cache and hence has miss rates similar to ocnventional but lack of page-level metadata locality implies 2.2x more 3-D DRAM bandwidth (Table 3.8). *PSB* +*T*\$'s miss rates are higher than the conventional design's due to holes, yet *PSB* performs better by avoiding excessive pressure on the 3-D DRAM bandwidth for tag metadata. In *TATP*, *PSB* incurs lower miss rate than the conventional design because the former's coarse-grain page replacements preserve the pages with high and frequent reuse where the block reuse is high but spreaad out in time. whereas the latter's fine-grain block replacement evicts such blocks. β\$'s miss rates are better than *PSB*'s due to fewer holes but worse than conventional's because though page-block decoupling in β\$ can eliminate *PSB*'s holes and achieve miss rates similar to the conventional design's, β\$ incurs some holes in return for a smaller *T*\$ and lower 3-D DRAM bandwidth demand (Section 3.3.1).

3.5.3 Other Comparisons

Impact of Perfect Footprint Prefetch: Figure 3.14 compares the performance (Y-axis, normalized to conventional cache with 64B blocks) of *PSB* and β\$ to that of *PSB* with an oracular perfect prefetch mechanism — all three configurations use the *T*\$. Experimentally, perfect prefetch is simulated by (1) including the bandwidth

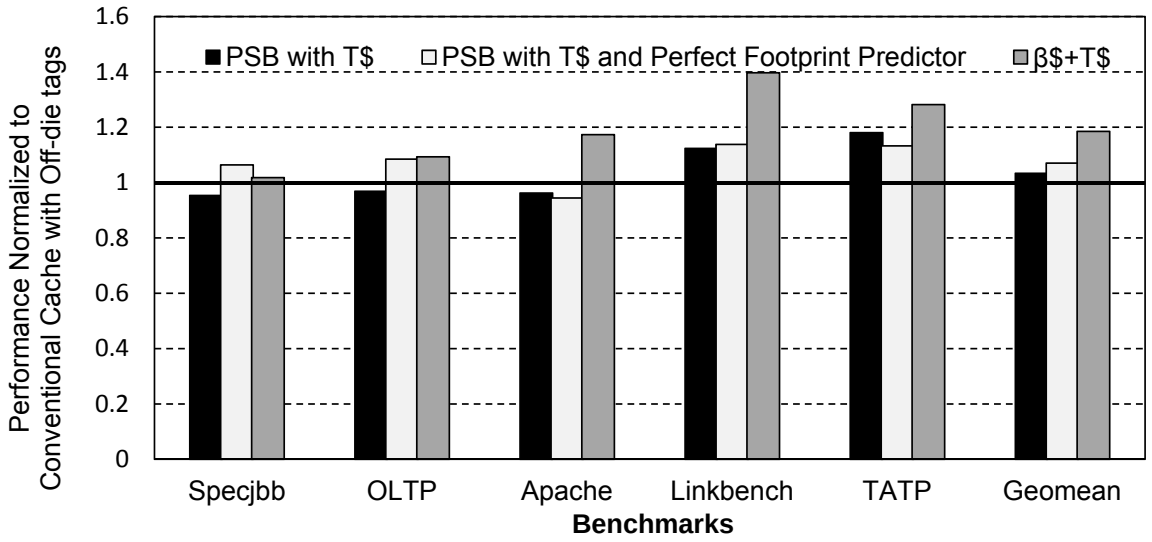

Fig. 3.14. Perfect Footprint Prefetch

cost (bank occupancy) of the subset of blocks of a page that are accessed before page replacement, and (2) considering any page hit to be a block hit. The addition of perfect footprint prefetching offers a modest improvement over *PSB + T$*; *β$ +T$* remains significantly better *without any prefetching at all*. This result is not surprising because the problem is one of bandwidth and not latency and prefetching improves latency but not bandwidth (Section 3.2.4).

Impact isolation and comparison with DCC: Recall that two of our features (colocation, free-lists) serve to minimize transfers from the DRAM cache and two of them (chunking and reverse pointers) to reduce the size of the *T$*. The impact of omitting the latter two features was previously isolated in Section 3.3.2 and Table 3.3; the per-set state is reduced by more than 1.85X as a result. As such, I focus on the impact of the former two features in this section.

To isolate the impact of *β$*'s features, Figure 3.15 shows the performance of *β$* with and without the features (both variants use the *T$*). Without the features, *β$* defaults to a decoupled cache with set mapping at the page granularity (i.e., few holes), similar to DCC [34]. In such an organization, the metadata is scattered

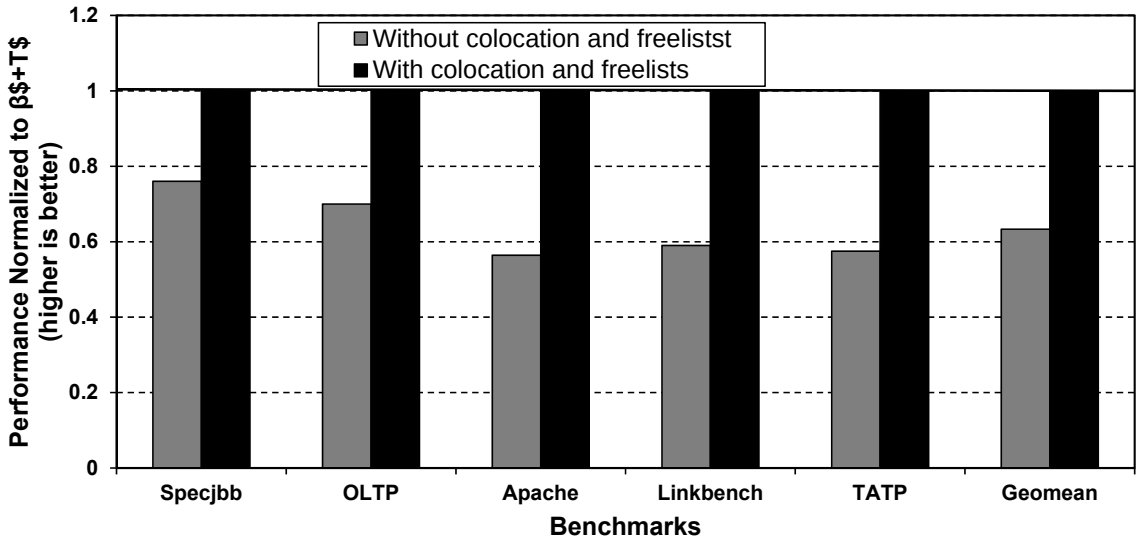

Fig. 3.15. Impact of $\beta\$$'s features and comparison with DCC

across the set, which destroys metadata locality at the page-level. Consequently, $\beta\$$ loses significant performance despite using the $T\$$ because the lack of page-level metadata spatial locality results in many more 3-D DRAM accessses than the full $\beta\$$ (Section 3.3.1). Thus, both $T\$$ and $\beta\$$'s features are needed for good performance. Note, the above comparison is equivalent to a comparison of $\beta\$ + T\$$ with DCC+$T\$$.

3.5.4 Sensitivity to 3-D DRAM bandwidth

Figure 3.16 shows the normalized performance (mean across all benchmarks, Y-axis) of three systems (conventional with 64B blocks and off-die tags, $PSB + T\$$, and $\beta\$ + T\$$) while varying 3-D DRAM bandwidth between 4X and 6X of main memory bandwidth. (Recall that our main results assume 3-D DRAM has 5X the bandwidth of main memory. I use the conventional cache with 64B blocks and off-die tags and 5X bandwidth as our normalization baseline.)

When available bandwidth is reduced (see 4X bars in Figure 3.16, 3-D bandwidth becomes a more precious resource. This results in $\beta\$ + T\$$ widening its performance gap relative to both $PSB + T\$$ and conventional cache. In contrast, when I increase

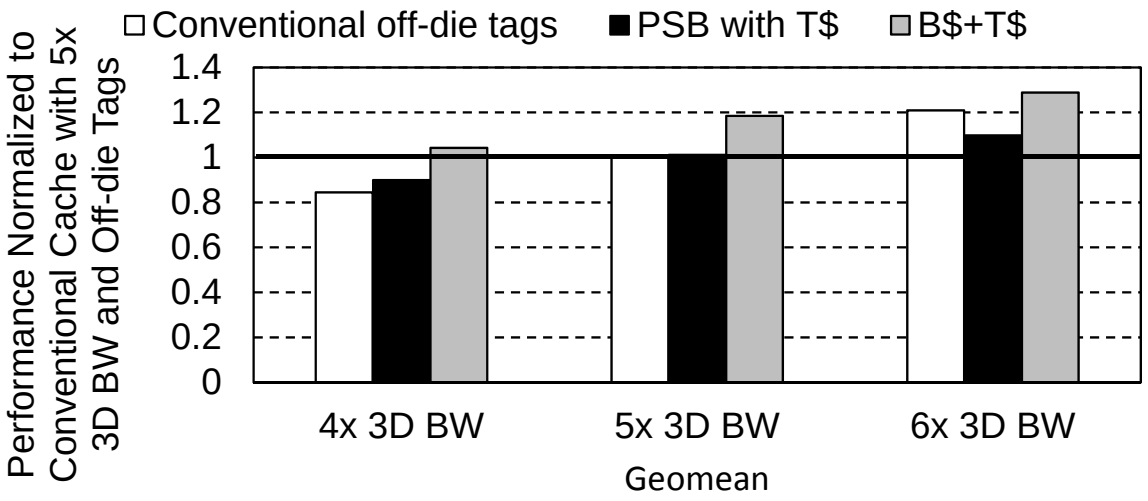

Fig. 3.16. Sensitivity to 3-D DRAM bandwidth

the bandwidth to 6X, the conventional cache, which is starved for 3-D bandwidth, benefits significantly relative to $\beta\$ + T\$$. This is not surpising as abundant bandwidth reduces the benefit of bandwidth efficiency that $\beta\$$ enjoys. Interestingly, $PSB + T\$$, which is not limited by 3-D DRAM bandwidth, becomes worse than both conventional and $\beta\$ + T\$$ configurations. The abundant bandwidth neutralizes any bandwidth efficiency advantage $PSB + T\$$ may have with respect to the conventional cache; and *PSB*'s holes ensures that it has a disadvantage relative to both $\beta\$$ and the conventional cache.

3.6 Related work

Off-chip memory bandwidth has been predicted to be a limiting factor even before the multicore era began [37]. While memory latency has always been a roadblock to maximizing processor performance, the advent of multicores meant that the memory subsystem will face another roadblock in the form of higher bandwidth demand. Not as harsh as the *memory latency wall* – a name given to the ever-increasing diverging gap between the speed of operation of the processor and memory – designing multicores with memory subsystem capable of sustaining the memory bandwidth de-

mand has its own challenges. Because die areas are not supposed to increase a lot with smaller process nodes, the available physical space along the chip edges doesn't increase. Consequently, this makes it harder to increase the pin count as memory traffic increases due to increase in core count. It is precisely for this reason pin-bandwidth has been identified to be a major bottleneck for future multicores [38]. 3-D die-stacking is the industry's response to the pin-bandwidth problem [15, 24]. 3-D die-stacking packages the on-chip logic and DRAM together albeit on different dies. Through the use of TSVs, the 3-D DRAM is stacked on top of processors. Another form of packaging, 2.5-D packaging, instead of TSVs makes use of high density interconnects between heterogeneous dies on a single package. As of now, it isn't clear which of the two packaging techniques will become more prevalent in the coming years. However, it is fair to say that on-chip logic and DRAM will be packaged together to serve the bandwidth demands of multicore. The idea of a tag cache has been proposed before for off-chip DRAM sector cache [31]. While sector caches capture spatial locality very well without increasing metadata storage overhead, they suffer from capacity loss due to holes incurred since page-level spatial locality varies. I compare with this design point in my evaluation. [39] proposes a sector cache with tags in 3-D, which on top of footprint prediction [32,33], uses way prediction to avoid the tag lookup latency overhead if the tags are stored in 3-D cache. [39] operates under the assumption that (1) 3-D bandwidth is plentiful (2) the loss of capacity due to holes is a non-issue. First, while it is true that 3-D bandwidth is more than main memory bandwidth, it is overly far-fetched to assume that such bandwidth would justify wasting it in metadata lookup and updates. Architects will try to utilize the copious bandwidth by increasing the core count, multithreading, or through other means. Second, while the spatial locality exhibited by scaleout workloads may be regular and plentiful, it is more varying for commercial workloads (Figure 3.4) where the loss of capacity in the form of holes hurts MPKI. In contrast, $\beta\$$ tries to minimize loss of capacity due to holes by decoupling tagging granularity and allocation granu-

larity, and ensuring that bandwidth is not worsened due to the additional metadata accesses through compact metadata layout techniques.

4. CONCLUSIONS

In this thesis, I have addressed the two fundamental problems that act as hindrance for multi-core performance scaling: power and off-chip memory bandwidth.

Multi-cores are becoming increasingly power-contrained. The slowing of Dennard's scaling, despite transistor innovations, is making it harder to get the desired power savings transistor scaling has traditionally offered. Researchers have responded to the slowing of Dennard's scaling by arguing that dark silicon inevitably imposes a performance limit and by advocating for customization to harness more performance with the same power budget. While the DSL suggests an undesirable bound on performance because of dark silicon, customization places multi-cores on a potentially arduous path of considerable design/programmability cost. In chapter 2, I showed that previously shown dark-silicon induced bounds on multi-core performance can be surpassed, and that a gentler, evolutionary path for multi-cores exists. This path, called successive frequency unscaling (SFU), involves successively scaling down the clock frequency with each technology generation. SFU is based on the insights that (1) frequency unscaling lowers per-core power where voltage-frequency scaling is infeasible, enabling more cores to be activated than the DSL approach; and (2) typical multi-core workloads are memory-bound and benefit from the increased memory-level parallelism achieved by the higher active core count. Guided by these insights and a simple analytical model, SFU exploits two non-linearities: (1) the sub-linear impact of clock speed on performance for memory-bound workloads and (2) the super-linear impact of throughput on queuing delays. The first non-linearity implies that SFU's increased memory-level parallelism more than offsets the slower clock so that for memory-intensive workloads, full SFU, where all the cores are powered up, performs 46% better than the DSL limit at the 11 nm technology node (18% better with out-of-

order cores). The second non-linearity comes into play for enterprise workloads where both throughput and response times are important and full SFU's aggressive clock slowdown may penalize response times. To address this issue, I proposed controlled SFU (C-SFU) which moderately slows down the clock and powers many, but not all, cores to achieve 21% better throughput than the DSL limit at the 11 nm technology node. The higher throughput non-linearly reduces queuing delays and thereby compensates for the slower clock, resulting in C-SFU's total response latency to be within +/- 10% of that of DSL.

Finally, SFU's simplicity enables a viable, evolutionary path of higher performance for multi-cores at virtually no design effort or complexity.

On the memory bandwidth front, increasing the pin-bandwidth has become harder due to the physical space limitations along the chip edges. While die sizes can be increased to allow greater number of pins, doing so will increase the cost per chip due to lower yield. Fortunately, the problem of memory bandwidth can be alleviated by die-stacking of DRAM on the processor die which promises to continue scaling the pin bandwidth to off-chip memory. The 3-D DRAM is expected to be used as a large cache, which poses a choice between placing its large tag on the processor die versus in the 3-D DRAM. Tag caching can strike a compromise between these choices but fundamentally requires exploiting page-level metadata locality to ensure efficient use of 3-D DRAM bandwidth. This locality crucially depends on the DRAM cache and tag cache (*T$*) organizations. While plain sub-blocking exploits this locality but incurs holes in the cache due to absent blocks, decoupled organizations avoid holes but destroy this locality. In chapter 3, I proposed *Bandwidth-Efficient Tag Access (BETA) DRAM cache* (β*$*) which both avoids holes and exploits the locality via its four features. β*$* targets two goals: (1) reducing the 3-D DRAM bandwidth cost of *T$* misses and DRAM cache misses, and (2) improving the *T$*'s effective size. For the first goal, β*$* exploits page-level metadata locality by (a) co-locating the metadata of a page, and (b) tracking the per-set free space. For the second goal, both β*$* and *T$* reduce the amount of on-die metadata by (a) dynamically switching from

$\beta\$$'s forward pointers to reverse pointers in the $T\$$ at the $\beta\$$-$T\$$ interface, and (b) allocating space in $\beta\$$ at the chunk granularity. Using simulations, I conclusively showed that (1) the primary concern in DRAM caches is bandwidth and not latency; (2) previous proposals incur bandwidth bottlenecks at either main memory or 3-D DRAM; (3) $\beta\$$ with a $T\$$ performs 15% better than the best previous scheme with a similarly-sized $T\$$; and (4) $\beta\$$'s improvements are due to its tag bandwidth efficiency. As die stacking is increasingly adopted, $\beta\$$'s tag bandwidth efficiency will be a key advantage over the other cache organizations.

While multi-cores do face challenges in delivering the promise of Moore's law, this thesis makes an attempt to show that multi-core performance can scale (for atleast a decade) without a drastic paradigm shift in computing.

REFERENCES

REFERENCES

[1] R. Dennard, F. Gaensslen, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *Solid-State Circuits, IEEE Journal of*, vol. 9, no. 5, pp. 256–268, 1974.

[2] "ARM-CTO: Power Surge could create 'Dark Silicon' http://www.eetimes.com/electronics-news/4085396/ARM-CTO-power-surge-could-create-dark-silicon-."

[3] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *Proceeding of the ISCA-38*, 2011, pp. 365–376.

[4] S. Borkar and A. A. Chien, "The future of microprocessors," *Communications of the ACM*, vol. 54, pp. 67–77, May 2011.

[5] N. Hardevellas, M. Ferdman, B. Falsafi, and A. Ailamaki, "Toward dark silicon in servers," *IEEE Micro*, vol. 31, no. 4, pp. 6–15, 2011.

[6] G. e. a. Venkatesh, "QsCores: trading dark silicon for scalable energy efficiency with quasi-specific cores," in *Proceedings of IEEE/ACM MICRO-44*, 2011, pp. 163–174.

[7] N. Pinckney et al., "Assessing the performance limits of parallelized near-threshold computing," in *Proceedings of DAC-49*, ser. DAC '12, 2012, pp. 1147–1152.

[8] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 1967 spring joint computer conference*, 1967, pp. 483–485.

[9] M. D. Hill and M. R. Marty, "Amdahl's law in the multicore era," *IEEE Computer*, vol. 41, pp. 33–38, 2008.

[10] S. Borkar, Personal communication, 2012.

[11] M. M. K. e. a. Martin, "Multifacet's general execution-driven multiprocessor simulator (gems) toolset," *SIGARCH Comput. Archit. News*, vol. 33, pp. 92–99, November 2005.

[12] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hållberg, J. Högberg, F. Larsson, A. Moestedt, and B. Werner, "Simics: A full system simulation platform," *Computer*, vol. 35, no. 2, pp. 50–58, Feb. 2002.

[13] "ITRS. international technology roadmap for semiconductors, 2011 winter update, 2011." [Online]. Available: http://www.itrs.net

[14] "DDR4: Double the speed, double the latency?" [Online]. Available: http://www.chipestimate.com/techtalk.php?d=2011-11-22

[15] B. Black et al., "Die stacking (3d) microarchitecture," in *Proceedings of IEEE/ACM MICRO-39*, 2006, pp. 469–479.

[16] D. e. a. Wang, "DRAMsim: a memory system simulator," *SIGARCH Comput. Archit. News*, pp. 100–107, 2005.

[17] B. M. e. a. Rogers, "Scaling the bandwidth wall: challenges in and avenues for cmp scaling," in *Proceedings of ISCA-36*, 2009, pp. 371–382.

[18] M. R. Marty and M. D. Hill, "Virtual hierarchies to support server consolidation," in *Proceedings of ISCA-34*, 2007, pp. 46–56.

[19] G. e. a. Moore, "Cramming more components onto integrated circuits," *Proceedings of the IEEE*, vol. 86, no. 1, pp. 82–85, 1998.

[20] G. e. a. Semeraro, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings of the 8th International Symposium on High-Performance Computer Architecture*, ser. HPCA '02, 2002.

[21] T. e. a. Simunic, "Dynamic voltage scaling and power management for portable systems," in *DAC '01: Proceedings of the 38th conference on Design automation*. ACM, 2001, pp. 524–529.

[22] G. e. a. Venkatesh, "Conservation cores: reducing the energy of mature computations," *SIGARCH Comput. Archit. News*, vol. 38, no. 1, pp. 205–218, Mar. 2010.

[23] H. e. a. Esmaeilzadeh, "Architecture support for disciplined approximate programming," in *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems*, ser. ASPLOS XVII. ACM, 2012, pp. 301–312.

[24] G. H. Loh, "3d-stacked memory architectures for multi-core processors," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 453–464.

[25] G. H. Loh and M. D. Hill, "Efficiently enabling conventional block sizes for very large die-stacked dram caches," in *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-44 '11. New York, NY, USA: ACM, 2011, pp. 454–464.

[26] M. K. Qureshi and G. H. Loh, "Fundamental latency trade-off in architecting dram caches: Outperforming impractical sram-tags with a simple and practical design," in *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 235–246.

[27] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, "Chop: Integrating dram caches for cmp server platforms," *IEEE Micro*, vol. 31, no. 1, pp. 99–108, 2011.

[28] J. S. Liptay, "Structural aspects of the system/360 model 85: Ii the cache," *IBM Syst. J.*, vol. 7, no. 1, pp. 15–21, Mar. 1968.

[29] A. Seznec, "Decoupled sectored caches: conciliating low tag implementation cost," in *Proceedings of the 21st annual international symposium on Computer architecture*, ser. ISCA '94. Los Alamitos, CA, USA: IEEE Computer Society Press, 1994, pp. 384–393.

[30] C.-C. Huang and V. Nagarajan, "Atcache: Reducing dram cache latency via a small sram tag cache," in *Proceedings of the 23rd International Conference on Parallel Architectures and Compilation*, ser. PACT '14, 2014, pp. 51–60.

[31] Z. Zhang, Z. Zhu, and X. Zhang, "Design and optimization of large size and low overhead off-chip caches," *IEEE Trans. Comput.*, vol. 53, no. 7, pp. 843–855, Jul. 2004.

[32] S. Kumar and C. Wilkerson, "Exploiting spatial locality in data caches using spatial footprints," in *Proceedings of the 25th annual international symposium on Computer architecture*, ser. ISCA '98. Washington, DC, USA: IEEE Computer Society, 1998, pp. 357–368.

[33] D. Jevdjic, S. Volos, and B. Falsafi, "Die-stacked dram caches for servers: hit ratio, latency, or bandwidth? have it all with footprint cache," in *Proceedings of the 40th Annual International Symposium on Computer Architecture*, ser. ISCA '13. New York, NY, USA: ACM, 2013, pp. 404–415.

[34] S. Sardashti and D. A. Wood, "Decoupled compressed cache: Exploiting spatial locality for energy-optimized compressed caching," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture*, ser. MICRO-46, 2013, pp. 62–73.

[35] S. Thoziyoor, J. H. Ahn, M. Monchiero, J. B. Brockman, and N. P. Jouppi, "A comprehensive memory modeling tool and its application to the design and analysis of future memory hierarchies," in *Proceedings of the 35th Annual International Symposium on Computer Architecture*, ser. ISCA '08. Washington, DC, USA: IEEE Computer Society, 2008, pp. 51–62.

[36] J. L. Henning, "Spec cpu2006 benchmark descriptions," *SIGARCH Comput. Archit. News*, vol. 34, no. 4, pp. 1–17, Sep. 2006.

[37] D. Burger, J. R. Goodman, and A. Kägi, "Memory bandwidth limitations of future microprocessors," in *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, ser. ISCA '96. New York, NY, USA: ACM, 1996, pp. 78–89.

[38] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, "Scaling the bandwidth wall: Challenges in and avenues for cmp scaling," in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ser. ISCA '09. New York, NY, USA: ACM, 2009, pp. 371–382.

[39] D. Jevdjic, G. H. Loh, C. Kaynak, and B. Falsafi, "Unison Cache: A Scalable and Effective Die-Stacked DRAM Cache," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2014.

VITA

VITA

Hamza Bin Sohail was born on October 29, 1984 in Islamabad, Pakistan. In March 2007, he obtained his BSc in Elctrical Engineering from the University of Engieering and Technology, Lahore, Pakistan.

In August 2007, he joined the PhD program in the School of Electrical and Computer Engineering at Purdue University. He began research work with Prof. T. N. Vijaykumar during spring 2009, and continued to work with him up to the completion of his doctoral degree. Hamza Bin Sohail interned at Intel Corporation at Hillsboro, Oregon, from May 2013 to August 2013, and at Apple Inc. at Cupertino, California, from May 2014 to August 2014.

The focus of his PhD research was on investigating ways to extend the performance scaling of multicores in the face of growing power constraints and increasing memory bandwidth demands. The problems he investigated include providing a gentler evolutionary path for multicores which ensures that there is no *dark silicon* while remaining within the desired Thermal Design Power budget. Since multicores require the memory bandwidth to scale proportionately, he investigated ways to design Die-stacked caches with bandwidth-efficient metadata accesses in order to preserve the bandwidth advantage the technology has to offer. In addition, he has also looked in to techniques to design memory systems with Phase Change Memory.