

# Wire delay is not a problem for SMT (in the near future)

Zeshan Chishti and T. N. Vijaykumar

School of Electrical and Computer Engineering, Purdue University

{zchishti, vijay}@purdue.edu

## Abstract

*Previous papers have shown that the slow scaling of wire delays compared to logic delays will prevent superscalar performance from scaling with technology. In this paper we show that the optimal pipeline for superscalar becomes shallower with technology, when wire delays are considered, tightening previous results that deeper pipelines perform only as well as shallower pipelines. The key reason for the lack of performance scaling is that superscalar does not have sufficient parallelism to hide the relatively-increased wire delays. However, Simultaneous Multithreading (SMT) provides the much-needed parallelism. We show that an SMT running a multiprogrammed workload with just 4-way issue not only retains the optimal pipeline depth over technology generations, enabling at least 43% increase in clock speed every generation, but also achieves the remainder of the expected speedup of two per generation through IPC. As wire delays become more dominant in future technologies, the number of programs needs to be scaled modestly to maintain the scaling trends, at least till the near-future 50nm technology. While this result ignores bandwidth constraints, using SMT to tolerate latency due to wire delays is not that simple because SMT causes bandwidth problems. Most of the stages of a modern out-of-order-issue pipeline employ RAM and CAM structures. Wire delays in conventional, latency-optimized RAM/CAM structures prevent them from being pipelined in a scaled manner. We show that this limitation prevents scaling of SMT throughput. We use bitline scaling to allow RAM/CAM bandwidth to scale with technology. Bitline scaling enables SMT throughput to scale at the rate of two per technology generation in the near future.*

## 1 Introduction

CMOS scaling trends are leading to a greater number of smaller and faster transistors in a single chip, but a relative increase in wire delays. Clock speeds have increased consistently, owing not only to faster transistors but also to deeper pipelines with a minimal degrading impact on instruction throughput (i.e., instructions per cycle or IPC). However, because global wire delays (e.g., register bypass wires) and RAM/CAM delays scale much slower than transistor delays, deeper superscalar pipelines experience increased latencies and a significant degradation in instruction throughput.

Optimizing the pipeline depth is based on balancing the clock rate and IPC to maximize the number of instructions executed per second. The tradeoff between clock rate and IPC has been a topic

of several recent studies. [14] suggests that the optimal logic depth per pipeline stage is 8 FO4 inverter delays. Because [14] implicitly assumes that wire delays and RAM/CAM delays scale at the same rate as logic gate delays, the optimal pipeline depth of 8 FO4 is valid only for the 100-nm technology assumed by the study; the paper does not provide any quantitative data of optimal depths taking into account the slow scaling of wire delays. This gap is partly filled by [2] which points out that wire delays dictate that deeper pipelines will not perform better than shallower pipelines in future technologies. [2] concludes that superscalars do not have sufficient parallelism to tolerate the relatively-increased wire delays. [12] also makes similar observations about the impact of wire delays.

In this paper, we extend the analyses of the previous papers: First, we show that the optimal pipeline becomes *shallower* with technology when wire delays are considered, tightening [2]’s results that deeper pipelines perform only as well as shallower pipelines. Second, while [14,2] analyze superscalars, which do not have sufficient parallelism, we analyze Simultaneous Multithreading (SMT) [27], which has sufficient parallelism. SMT overlaps multiple (either multiprogrammed or explicitly parallel) threads on a wide-issue pipeline and provides the much-needed parallelism to tolerate the slowly-scaling wire delays. We show that an SMT running a multiprogrammed workload with just 4-way issue not only retains the optimal pipeline depth over technology generations, enabling at least 43% increase in clock speed every generation, but also achieves the remainder of the expected speedup of two per generation through IPC. As wire delays become more dominant in future technologies, the number of programs needs to be scaled modestly to maintain the scaling trends, at least till the near-future 50nm technology.

We put this result in perspective. While single-program performance may continue to be important, our results are interesting because future desktop- and server-class computers are likely to run multiprogrammed workloads on SMT. Our results show that wire delays can be tolerated in this important class of machines.

While this result ignores bandwidth constraints, using SMT to tolerate *latency* due to wire delays is not that simple, because SMT causes *bandwidth* problems. *Most* of the stages of a modern out-of-order-issue pipeline employ RAM and CAM structures. While decode and execute stages are combinational logic, fetch, rename, register read, memory, and register write involve RAMs (caches and register file), and issue and memory involve CAMs (issue queue and load/store queue). As such, SMT’s throughput is bound by the bandwidth of these structures. Wire delays of RAM/CAM bitlines and wordlines limit the structures’ bandwidth which in turn limits not only single-program but also SMT perfor-

mance. While it is generally believed that wire delays increase latency, we show that they also fundamentally limit bandwidth. [14] ignores this constraint and implicitly assumes that the bandwidth will scale at the same rate as logic gate delays. This bandwidth issue is the third point analyzed in this paper.

SMT’s bandwidth demand may not be met easily by traditional multiporting, banking [22], line buffers [28], or hierarchical bitlines [17]. As we explain in Section 3, these approaches are also limited by wire delays. Another technique to improve RAM bandwidth is pipelining. Some previous papers [16, 8] propose a shallow, two-stage pipeline of the wordline decode in one stage followed by the rest of the access in another.

As pointed out in [8], the key impediment to deeper pipelining of RAM is that the bitline delay cannot be pipelined because the signals on the bitlines are weak, and not digital; latching can be done only *after* the sense amplifiers convert the bitline signals to digital. Conventional designs partition RAM structures into sub-arrays and the bitline delay depends on the subarray size [21]. Because the bandwidth demand of superscalar processors can be satisfied without resorting to deep pipelining, and because superscalar processors are sensitive to RAM (especially L1 cache) latency, traditional designs choose subarray sizes that optimize latency. However, we show that SMT throughput does not scale with shallow pipelining of such latency-optimized RAM structures; deeper pipelining is needed.

To achieve deep pipelining, a simple strategy is to reduce the subarray size such that the bitline + sense amplifier delay fits within one clock cycle (e.g., [1,20]). Because the subarray size has to scale to counter wire delays in future technologies, we call such designs as *bitline-scaled* designs. Reducing the subarray size, however, both increases the amount of address decoding needed to determine the subarray, and requires a larger subarray multiplexer. Bitline scaling counters the resulting delay increases by pipelining the subarray decode and subarray multiplexer steps themselves. We similarly pipeline CAM bitlines into multiple subarrays. Bitline-scaled CAM has some similarities to segmented issue queues of [14,19].

For the final result in the paper, we show that even though a bitline-scaled RAM (or CAM) has worse latency than a latency-optimized RAM (or CAM), the higher bandwidth offsets the latency penalties allowing SMT throughput to scale. While previous papers have explored bitline scaling and segmented issue queue, this is the first paper to show that SMT can use the techniques to tackle the wire delay problem.

Using simulations with SPEC2000 benchmarks, we find the following:

- **Latency effect on superscalar:** The optimal pipelines for superscalar become shallower due to slow scaling of wire delays in future technologies.
- **Latency effect on SMT:** With unlimited bandwidth, a 4-way issue SMT retains the optimal pipeline depth over technology generations, enabling at least 43% increase in clock speed every generation, and achieves the remainder of the expected speedup of two per generation through IPC, at least till the near-future 50nm technology.
- **Bandwidth limitation of wire delays:** Wire delays in conventional, latency-optimized RAM/CAM structures pre-

vent them from being pipelined in a scaled manner. This bandwidth limitation prevents scaling of SMT throughput.

- **Overcoming bandwidth limitation:** Bitline scaling enables SMT throughput to scale at the rate of two per technology generation in the near future.

The rest of the paper is organized as follows. We discuss the latency effect of wire delays and deeper pipelines next. In Section 3, we discuss the bandwidth effect. In Section 4, we describe the experimental methodology. We present our results in Section 5 and conclude in Section 6.

## 2 Latency effect of wire delays and deeper pipelines

In this section, we analyze the impact of wire delays and deeper pipelines on the performance of both superscalar and SMT processors. While some of this analysis is presented in [23,2,14], we present more comprehensive analysis by combining the notion of pipeline loops (from [5]) and wire delays. While [5] showed the detrimental effect of lengthening the loops on performance, we derive new properties exhibited by the loops and explain the impact of wire delays in the context of these properties.

### 2.1 Pipeline loops

Figure 1 shows a conventional out-of-order processor pipeline. The out-of-order pipeline can be thought of as being composed of two in-order half-pipelines connected by the issue queue: the front-end and the back-end. Figure 1 also shows the loops in the pipeline. The branch misprediction loop and bypass loops are well-known. The fetch loop is due to the dependence between the current PC being used to predict the next PC. The rename loop is due to the dependence between a previous instruction assigning a rename tag and a later instruction reading the tag. The issue loop is due to the dependence between the select of a producer and wakeup of a consumer. The load misspeculation loop is due to load-miss replay. The load/store queue loop is due the dependence between a previous store and a later load to the same address.

The loops affect performance differently from each other, depending on: 1) frequency of loop usage, 2) loop length, and 3) the interaction among loops.

The more frequent a loop usage is, the higher the impact on performance. While the fetch, rename, issue, and bypass loops are all fairly frequent, load misspeculation and branch misprediction

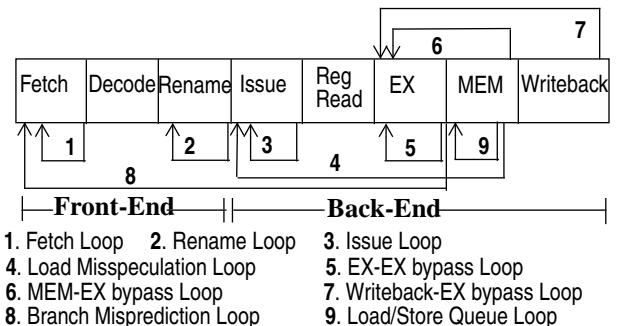


FIGURE 1: Pipeline Loops

loops are used only upon load misses and branch mispredictions, respectively, and are less frequent.

As pointed out by [5], lengthening any of the loops in terms of number of clocks negatively impacts IPC. A loop gets longer if 1) the latency of any pipeline stage encompassed by the loop increases, or 2) wire delay between the loop endpoints increases. Increasing the clock speed by deepening the pipeline results in increasing the latency of pipeline stages. As shown in [14], the increase in the clock speed beyond a limit is offset by the decrease in IPC due to longer loops. Furthermore, because wire delays constitute a significant portion of RAM/CAM delays, and because wire delays scale slowly as compared to gate delays, the latency of several pipeline stages increases with technology. The slow scaling of various tables (prediction, rename), caches, issue queue, and load/store queue lengthens fetch, rename, issue, MEM-EX, writeback-EX, load misspeculation, and load/store queue loops, many of which are used frequently. Apart from RAM/CAM structures, the slow scaling of global wires [12,4] lengthens all the bypass loops (also observed by [18]), which are used frequently.

The effect of one loop on IPC is not entirely independent of the other loops. The interplay among loops exhibits the following two properties: 1) *Loop Inclusion property*: An increase in the latency of a pipeline stage affects all the loops that encompass the stage. 2) *Dominance property*: For the set of all mutually-disjoint loops in a half-pipeline that are used by a producer-consumer dependence, the loop with the maximum length impacts IPC and the shorter loops do *not* further degrade IPC. The longest loop stalls the consumer enough from the producer that the shorter loops do not cause any more stalls. Thus the longest loop dominates the rest of the loops. For example, a back-to-back-issued producer-consumer register dependence uses both the issue loop and the EX-EX bypass loop. If the issue loop is 3 cycles, increasing the bypass loop from 1 to 2 or 3 cycles does not have any impact on performance because the issue loop already stalls the consumer such that by the time the consumer reaches EX, the producer exits EX allowing the bypass loop to provide the bypassed value without any more stalls. However, lengthening the rename loop affects performance, because the rename loop and the issue loop belong to different half-pipelines.

While the dominance property has the positive implication that lengthening non-dominant loops do not cause IPC loss, there is a negative implication that *all* the dominant loops (if multiple loops are at maximum length then all such loops are dominant) need to be shortened to improve the IPC.

The effect of loops can be reduced by exploiting independence. If there are enough independent instructions to fill all the pipeline stages in the dominant loop(s), then the impact of the loops can be eliminated. While superscalars do not have sufficient parallelism, SMT does.

## 2.2 Latency effect on superscalar

There are several architectural techniques to enhance independence and reduce the impact of the loops in superscalars. We list a few examples: Slow scaling of CAM has made it difficult to retain single-cycle issue loop (i.e., back-to-back issue of dependent instructions using single-cycle issue queue). Several techniques

Fetch	Decode	Rename	Issue	Reg Read	EX	MEM	Writeback
RAM	Logic	RAM	CAM	RAM	Logic	RAM / CAM	RAM

FIGURE 2: Circuits used in pipeline stages

[24,19,10] have been proposed to obtain back-to-back issue, even with a multiple-cycle issue queue. Similarly, lengthening of the EX-EX bypass loop due to multiple-cycle ALUs can be alleviated by partial bypasses which allow partial ALU outputs to be bypassed back to the next instruction [6]. Also, load misspeculation and branch misprediction loops result in squashes whose penalty may be reduced by selective squashing [11,5]. Finally, lengthening of the rename loop due to multiple-cycle rename tables can be alleviated by using bypasses from the later rename stages to the front of the rename tables [23].

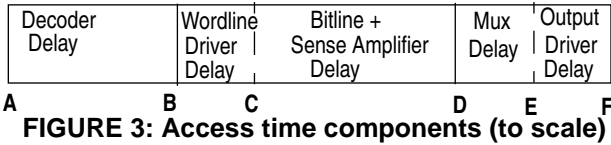
Unfortunately, the dominance property implies that *all* the loops have to be shortened to improve IPC, requiring *most, if not all*, of the above schemes. This requirement adds substantial complexity to the pipeline, not to mention extra levels of logic in many of the stages putting a limit on pipeline depth scaling. Even if all the above schemes are used to enable back-to-back issue and perfect pipelining in rename, issue, and EX-EX bypass loops, there are still some difficult loops which are lengthened by wire delays and prevent back-to-back issue. The difficult loops are the MEM-EX bypass loop, which is lengthened by cache latency and is used frequently by load-use dependencies, and the less-frequently-used branch misprediction and load misspeculation loops. Finally, even if there were as much parallelism available as needed by a deep pipeline, pipelines cannot be arbitrarily deepened due to latch, clock skew and jitter overheads [14]. Due to the above problems, the optimal pipeline for superscalar at 50nm technology is actually shallower than that at 100nm technology, as we show in Section 5.2.

## 2.3 Latency effect on SMT

SMT overlaps multiple threads on a wide-issue pipeline and provides the much-needed parallelism to tolerate slowly-scaling wire delays and deeper pipelines. *All* the loops in the back-end are alleviated by SMT's parallelism even without using the aforementioned aggressive schemes. However, the loops in the front-end need a slight change in SMT's fetch policy. The ICOUNT fetch policy [26] favors fetching from the thread that has the fewest in-flight instructions motivated by the fact that such a thread is the fastest progressing thread. Multiple-cycle loops in the front-end imply that instructions from the same thread cannot be fetched back to back. Accordingly, we modify ICOUNT so that a thread fetched in cycle  $i$  is not considered for fetching until cycle  $i + \text{length of the dominant loop in the front-end}$ . The best thread, as per ICOUNT applied to the rest of the threads, is chosen in cycle  $i+1$ , and so on. As we show in Section 5.2, SMT throughput can be scaled by 2x every technology generation even as wire delays become more dominant by modestly increasing the number of programs.

## 3 Bandwidth scaling

Using SMT to tolerate latency due to wire delays is not that



simple because SMT causes bandwidth problems. Figure 2 shows that *most* of the stages of a modern out-of-order-issue pipeline employ RAM and CAM structures. Wire delays of RAM/CAM bitlines and wordlines limit the bandwidth of these structures, which in turn limits not only single-program but even SMT performance. While it is generally believed that wire delays increase latency, we show that they also fundamentally limit bandwidth.

SMT’s bandwidth demand may not be met easily by traditional multiporting, banking [22], line buffer [28], or hierarchical bitlines [17,3]. Adding true ports causes considerable increase in area and wire delays. Bank conflicts and line buffer misses, which are inevitable, reduce the effectiveness of banking and line buffer. Hierarchical bitline approach breaks up the bitlines into a two-level hierarchy of many “local” and one “global” bitlines. The hierarchy allows two accesses, one in each level, to proceed simultaneously. Because global bitlines are as long as all local bitlines put together, global bitline wire delay fundamentally limits the bandwidth through the hierarchy.

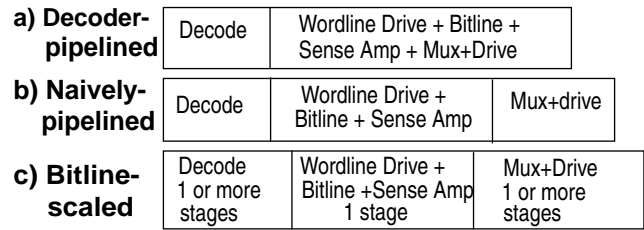
While pipelining is another approach to achieve high bandwidth, we show how naive pipelining is also limited by wire delays. Then we describe bitline scaling. We discuss RAM using cache as example and then CAM.

### 3.1 Bitline-scaled RAM

Figure 3 shows the components of cache access time: i) decoder delay (A to B) ii) wordline driver + bitline + senseamp delay (B to D) iii) mux + output driver delay (D to F). The cache access time is usually reduced by partitioning the cache into multiple smaller subarrays. The parameters  $N_{dwl}$  and  $N_{dbl}$  represent the extent to which partitioning is done.  $N_{dwl}$  indicates vertical partitions (creating more, but shorter wordlines), and  $N_{dbl}$  indicates horizontal partitions (creating more, but shorter bitlines). The total number of subarrays is  $N_{dwl} * N_{dbl}$ . Conventional cache designs are latency-optimized and adjust the subarray size such that the overall access time is minimized [21].

One solution to improve bandwidth is to pipeline the cache access. Some designs have a separate stage for wordline decode [16], shown in Figure 4(a). The decoder delay contributes only 25%-30% of the overall cache latency and the bandwidth of a pipeline is determined by the slowest pipeline stage. Therefore, using a pipelined decoder alone results in an ineffective pipeline. This imbalance can be removed by deeper pipelining. However, because the signals on bitlines are weak, and not digital, latching can be done *only* after the sense amplifiers convert the analog bitline signals to digital [8]. Therefore, in Figure 3, the delay from B to D cannot be pipelined, and a latch can be placed only at D. Figure 4(b) shows such a naively-pipelined cache. We observe that pipelining a latency-optimized cache naively results in a substantially uneven split, because the wordline + bitline + sense amplifier delay dominates the mux + output driver delay.

A bitline-scaled cache (e.g., [1,20]) eliminates the uneven split in pipeline stages by reducing the wordline + bitline + sense



**FIGURE 4: Cache pipelining configurations**

amplifier delay so that it no longer dominates the mux + data driver delay. In a bitline-scaled cache, the size of subarrays is chosen such that the wordline + bitline + sense amplifier delay fits in one clock cycle. As a result, the wordline + bitline + sense amplifier stage no longer limits the bandwidth. However, for same cache size, a reduction in subarray size means an increase in the total number of subarrays, thus increasing both the subarray decode delay and the subarray multiplexer delay, which may become multiple clock cycles. Fortunately, unlike the wordline + bitline + sense amplifier stage, the decode and mux + drive stages have digital data and can be pipelined further. The delay increases in decode and mux + drive stage are countered by pipelining these stages, so that the bandwidth remains unaffected, despite an increase in overall latency. Pipeline latch overhead also increases the bitline-scaled cache’s latency. Typically, we see a one- or two-cycle difference between the latency of a latency-optimized cache and that of a bitline-scaled cache. However, the bitline-scaled cache’s higher bandwidth offsets the increase in latency for SMT.

Some implementation issues with bitline-scaled caches are:

(1) The extra latches needed for deeper pipelining increase both the area and power. (2) Precharging creates a resource conflict: While an access in the wordline + bitline + sense amplifier stage is discharging the bitlines, *at the same time* another access in the decode stage would try to precharge. Therefore, precharging and discharging must occur atomically in the wordline + bitline + sense amplifier stage. (3) Stores cause structural hazards: After reading the bitlines and matching the tags, stores also write to the bitlines. Thus, stores reuse some of the stages causing structural hazards. Such hazards can be resolved by appropriate waiting.

Wave pipelining the cache overlaps wordline delays of the next access with the bitline + sense-amplifier delays of the previous access (without using latches) [8]. Therefore, in a wave-pipelined cache, only the delay from C to D in Figure 3 cannot be pipelined. However, this delay worsens in future technologies due to slow-scaling wire delays of the bitlines, becoming more dominant, and limiting the cache bandwidth. Thus, bitline scaling is needed even if wave pipelining is employed. To add bitline scaling to a wave-pipelined cache, the subarray size is chosen such that the bitline + sense amplifier delay fits in one clock cycle. We note that wave pipelining requires controlling signal propagation delays to a high precision which may be hard to do in future technologies due to increasing process variations. In contrast, bitline scaling is simpler and more robust.

Banked caches [22] and line buffers [28] have uncertainty in even their load hit latencies. This uncertainty disrupts the scheduling of wakeups, and degrades performance significantly. In comparison, the bitline-scaled cache has a fixed load hit latency, and does not disrupt wakeups.

### 3.2 Bitline-scaled CAM

A conventional CAM structure is implemented as a fully-associative tag array, in conjunction with a direct-mapped data array. A CAM access first searches all tag entries in parallel, and then reads/writes the matching entry in the data array. The main components of the CAM access time are (1) tag bitline drive delay (2) tag comparison and match signal generation delay (3) match line or-ing delay (4) direct-mapped array’s access time. Because CAMs typically have small data arrays, the access time is dominated by components (1), (2), and (3). As shown in [18], the tag bitline drive delay scales slowly, and becomes more and more significant at feature sizes of 180nm and below. The overall CAM access time scales slower relative to logic delays, preventing the CAM access from fitting within a cycle.

To explain bitline-scaled CAMs, we consider the example of the issue queue. The main components of issue queue are wakeup and select [18]. Wakeup performs a CAM access for waking up instructions when their source operands become available, and select chooses instructions for execution from the pool of ready instructions. Because wakeup is a CAM access and select is a tree of logic, wakeup scales slowly and select scales as well as gate delays [18]. We use bitline scaling to pipeline the CAM access by partitioning the CAM into multiple segments such that each segment fits within a cycle. Consequently, wakeup proceeds in a pipelined fashion from one segment to the next. While select scales with technology, it may still take multiple cycles for deeper processor pipelines within one technology. Fortunately, select is logic and can be pipelined easily. Unlike a single-cycle issue queue where wakeup and select operate in close synchrony, a bitline-scaled issue queue’s wakeup and select pipelines proceed independent of each other. Wakeup proceeds from one segment to another and select chooses from whichever instructions are ready at any given cycle. The pipelined wakeup of bitline scaling is similar to the segmented issue queue in [14].

Bitline-scaling of CAM structures such as the issue queue and load/store queue raises some architectural issues. While pipelining the structures allows multiple instructions to proceed simultaneously, dependencies will still prevent back-to-back issue. One option, called *back-to-back (b2b)*, is to use the architectural schemes mentioned in Section 2.2 to enhance independence using just one thread. Another option, called *no-back-to-back (nb2b)*, is to use more programs to provide independent instructions without incurring the complexities of the schemes. In this paper we are interested in scaling bounds. While the schemes mentioned in Section 2.2 incur misspeculations and stalls, we assume idealized versions of these schemes without such inefficiencies. We optimistically assume that in *b2b*, even dependent instructions can be issued in consecutive cycles. And in *nb2b*, we conservatively assume that only independent instructions can be issued in consecutive cycles. These two options provide the best-case and worst-case bounds for scaling. Using a subset of the schemes will result in performance between these bounds. To simplify the analysis, we show only these two options in our results.

## 4 Methodology

Table 1 shows the baseline configuration for the simulated pro-

**Table 1: Parameters**

Issue Width	4
Active List	64 entries per thread
LSQ	32 entries per thread
Physical Registers	256 int and 256 FP
L1 I-cache	64KB, 2-way, 2-ports
L1 D-cache	64KB, 2-way 2-ports
L2-cache	1MB, 8-way, unified, 1-port
Issue Queue	32 entries
Branch predictor	2-level hybrid, 8K per thread

**Table 2: Benchmarks**

Number Benchmark	Type
<b>Low IPC</b>	
1 mcf, 2 twolf, 3 vpr	int
4 ammp, 5 parser, 6 lucas	FP
<b>Medium IPC</b>	
7 bzip, 8 eon, 9 gap, 10 gzip, 11 perlbnk	int
12 applu, 13 apsi, 14 art, 15 galgel	FP
<b>High IPC</b>	
16 gcc, 17 crafty, 18 vor-tex,	int
19 facerec, 20 fma3d, 21 mesa, 22 mgrid, 23 swim, 24 wupwise	FP

cessor which is similar to Alpha 21264 [15]. We modify SimpleScalar 3.0 [7] to simulate a high-performance, out-of-order SMT. We model separate active lists, load store queues, first-level branch predictors and return address stacks for each context. We carefully model both the latency and bandwidth effects of pipeline loops (Section 2). The SMT in this study has two fetch ports, and fills up fetch bandwidth from up to two programs. We use the ICOUNT fetch policy [26]. Our simulator also incorporates thread squashing upon L2 misses [25].

Table 2 shows the benchmarks from the SPEC2K suite, categorized into three classes—*Low*, *Medium* and *High*—based on their single-program IPCs. For each application, we use ref inputs, and fast-forward 2 billion instructions. For single-program workloads, we run 200 million instructions, and for multiprogrammed workloads, we execute till one of the benchmarks completes 200 million instructions.

Table 3 lists the SMT workloads used in our simulations. We obtain multiprogrammed workloads by using different combinations of benchmarks from the three classes shown in Table 2. For example, MIX1.1.2 represents a workload with one benchmark each from *low* and *medium* IPC classes, and two benchmarks from *high* IPC class. Our SMT workloads contain combinations both within (e.g., MIX3.0.0) and across classes (e.g., MIX1.1.1). We ensure that every benchmark appears in exactly one 2-program, one 3-program, and one 4-program workload, so that the average IPS (instructions per second) for workloads with any number of programs can be compared directly to the average IPS for single-program workloads. For the 5-program workload MIX 0.3.2, we use an additional medium-IPC benchmark, *equake*, because the number of benchmarks is not divisible by 5.

We modify Cacti 3.2 [21] to model the latency and bandwidth of RAM and CAM structures. Our modifications to Cacti are as follows: 1) We extend Cacti’s wire delay model by using the resistance and capacitance parameters for mid-level metal wires projected by the 2001 ITRS roadmap [4] and its 2002 update. 2) Cacti assumes that the voltage swing on bitlines decreases as fast as supply voltage. As pointed out by [3,9], the input offset voltage of sense amplifier does not scale, causing a slower reduction in

**Table 3: Multiprogrammed workloads**

MIX	Benchmark numbers	MIX	Benchmark numbers
<b>2-Program</b>			
		1.2.0	3, 8, 15
2.0.0	1, 4	1.0.2	5, 16, 21
0.2.0	9, 12	0.2.1	9, 10, 22
0.2.0	10, 14	0.1.2	11, 18, 24
0.0.2	22, 24	<b>4-Program</b>	
0.0.2	17, 21	2.1.1	1, 4, 14, 20
1.1.0	6, 11	1.1.2	6, 12, 17, 19
1.1.0	3, 15	1.2.1	5, 8, 10, 23
1.0.1	5, 16	1.3.0	2, 7, 11, 13
1.0.1	2, 23	1.0.3	3, 16, 21, 22
0.1.1	13, 19	0.2.2	9, 15, 18, 24
0.1.1	10, 18	<b>5-Program</b>	
0.1.1	8, 20	3.1.1	1, 4, 6, 14, 20
<b>3-Program</b>			
		1.3.1	5, 7, 11, 13, 17
3.0.0	1, 4, 6	1.1.3	3, 8, 16, 21, 22
0.3.0	12, 13, 14	1.2.2	2, 10, 12, 19, 23
0.0.3	17, 19, 23	0.3.2	9, 15, quake, 18, 24
1.1.1	2, 7, 20		

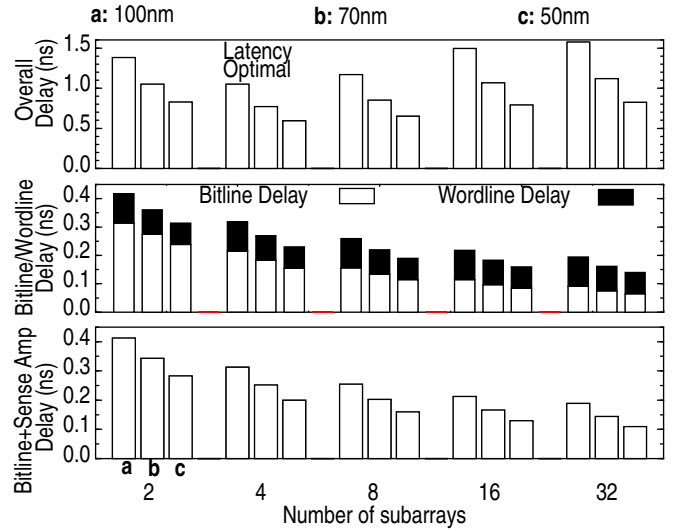
bitline swing relative to supply voltage. As recommended in [3], we modify Cacti to keep input offset voltage constant at 50mV for all technologies. 3) The cycle time calculation in Cacti does not consider the bitline precharge delay during the calculation of cycle time. As mentioned in Section 3.1, the bitline precharge and discharge must complete in the same cycle. We extend Cacti to account for bitline precharge delay in cycle time calculation.

We use Cacti to model CAMs as fully-associative caches with appropriate number of tag and data bits. We obtain issue queue wakeup latency from Cacti. To obtain select latency, we use Alpha 21264 at 180nm as reference point. Based on results in [18], we scale the select latency with technology. We obtain the latencies for integer and floating-point operations by scaling the Alpha 21264 functional-unit latencies using a methodology similar to that of [14]. The functional-unit latencies in terms of FO4 remain unchanged at different technologies.

We show results in Section 5 only for 6, 8, and 10 FO4 clock periods. FO4 measures the delay of an inverter driving a fan-out-of-four. 1 FO4 is roughly equal to 360 picoseconds times the minimum gate length for a technology, measured in microns [12]. The FO4 metric abstracts away transistor speed variations due to technology. We experimented with other clock periods, but found them to be non-optimal at the technologies being considered. We believe that the results for 6, 8, and 10 FO4 are comprehensive enough to show the scaling trends.

## 5 Results

We present circuit results in Section 5.1 We quantify the impact of latency on superscalar and SMT, and that of bandwidth on SMT in Section 5.2 and Section 5.3, respectively. We analyze the effect of issue width on scaling trends in Section 5.4

**FIGURE 5: Effect of varying partitioning**

### 5.1 Circuit Results

In this section, we quantify the latencies and bandwidths of the structures in the pipeline. We show that the RAM and CAM delays scale slower with technology, relative to gate delays. We also show that bitline scaling in RAMs and CAMs results in higher bandwidth as compared to a latency-optimized design.

#### 5.1.1 RAM

We show results for L1 cache and then other RAM components. Because wave pipelining provides some bandwidth advantage to the latency-optimized design, we assume wave pipelining for both latency-optimized and bitline-scaled RAMs.

Figure 5 shows the effect of changing the partitioning schemes on the bitline delay, wordline delay, bitline + sense-amplifier delay, and overall delay for the 64KB, dual-ported L1 cache at different technologies. The X-axis shows the different partitioning schemes in terms of  $N_{dwl} \times N_{dbl}$  (Section 3.1). We fix  $N_{dwl}$  at 2, because other values of  $N_{dwl}$  result in sub-optimal overall delays. For each scheme, the three bars from left to right represent the delays for 100nm, 70nm, and 50nm technology, respectively.

As the subarray size decreases (i.e., number of subarrays increases), the overall access time (top graph) first decreases, reaches a minimum value, and then increases. The latency-optimized approach chooses the partitioning that minimizes the overall delay. In all the three technologies, the latency-optimized cache has 4 (2x2) partitions. We discuss optimal partitioning for bitline-scaled cache later. The wordline and bitline delays (middle graph) scale slowly with technology. As gate delays decrease by 50% from 100nm to 50nm technology, the wordline and bitline delays for the 2x2 partitioning scheme decrease only by 28% and 30%, respectively. Although decoder delay and mux + output driver delay scale as fast as logic, the slow scaling of bitline and wordline delays significantly impacts scaling of overall delay. Going from 100nm to 50nm, the overall delay for latency-optimized cache decreases by 42%, though gate delays decrease by 50%. The bitline + sense-amplifier delay (bottom graph) becomes

**Table 4: Optimum partitioning for latency-optimized (l-o) and bitline scaled (b-s) caches**

Tech nol- ogy (nm)	Clock Period (FO4/ ns)	Optimum number of subarrays		bitline +sense amp delay (ns)		latency(cycles)/ bandwidth (access per cycle)	
		l-o	b-s	l-o	b-s	l-o	b-s
		100	6/0.22	4	16	0.32	0.21
	8/0.29	4	8	0.32	0.25	4/0.5	5/1
	10/0.36	4	4	0.32	0.32	3/1	3/1
70	6/0.16	4	16	0.25	0.16	5/0.5	7/1
	8/0.21	4	8	0.25	0.2	4/0.5	5/1
	10/0.26	4	4	0.25	0.25	3/1	3/1
50	6/0.11	4	32	0.2	0.11	6/0.5	8/1
	8/0.14	4	16	0.2	0.13	5/0.5	6/1
	10/0.18	4	8	0.2	0.16	4/0.5	4/1

more dominant in future technology generations. For the 2x2 partitioning scheme, the bitline + sense-amplifier delay increases from 29% of overall cache delay at 100nm, to 35% of overall cache delay at 50nm. Because the optimum for bitline scaling is the partitioning where the bitline + sense-amplifier delay fits in one clock cycle, this dominance implies that with deeper pipelines (i.e., higher clock speeds or smaller FO4) finer partitioning is needed. This trend is clear in Table 4 which shows the optimum partitioning schemes for the latency-optimized (“l-o”) and bitline-scaled (“b-s”) caches (assuming wave pipelining in both cases), and compares their latency and (hazard-free) bandwidth for different technologies and clock periods. The bandwidth numbers are based on bitline + sense-amplifier delay and clock period, also shown in Table 4.

We observe that the optimum subarray size in the bitline-scaled cache is always smaller (i.e., more subarrays) or equal to that in the latency-optimized cache. The bandwidth of the latency-optimized cache is limited by the bitline + sense-amplifier delay. At low clock speeds, latency-optimized subarrays allow access every cycle. However, as clock speed increases, the latency-optimized cache cannot be accessed every cycle despite wave pipelining. In comparison, the subarray size scaling in bitline-scaled cache enables access every cycle. However, we see that the latency of the bitline-scaled cache is up to two cycles more than that of the latency-optimized cache, as expected from

**Table 5: RAM results for other components**

Tech- nol- ogy (nm)	Clock Period (FO4)	Register File		Rename Table		L2 cache latency (cycles)
		latency(cycles)/ bandwidth (access per cycle)		latency(cycles)/ bandwidth (access per cycle)		
		l-o	b-s	l-o	b-s	
100	6	4/1	4/1	3/1	3/1	27
	8	3/1	3/1	2/1	3/1	21
	10	3/1	3/1	2/1	3/1	16
70	6	4/0.5	4/1	3/1	3/1	30
	8	3/1	3/1	3/1	3/1	23
	10	3/1	3/1	2/1	3/1	18
50	6	5/0.5	5/1	4/0.5	4/1	39
	8	4/0.5	4/1	3/1	3/1	30
	10	3/1	3/1	3/1	3/1	23

**Table 6: Issue Queue Results**

Tech nol- ogy (nm)	Clock Period (FO4)	latency-optimized			bitline-scaled			
		# of cycles		Access per cycle	# of seg- ments	# of cycles		Access per cycle
		Wake up	Sel ect			Wake up	Sel ect	
100	6	2	2	0.5	2	2	2	1
	8	1	1	1	1	1	1	1
	10	1	1	1	1	1	1	1
70	6	2	2	0.5	3	3	2	1
	8	2	1	0.5	2	2	1	1
	10	1	1	1	1	1	1	1
50	6	2	2	0.5	4	4	2	1
	8	2	1	0.5	3	3	1	1
	10	2	1	0.5	2	2	1	1

Section 3.1. Table 5 shows similar results for register file, rename table, and L2 cache. These results were obtained by using the same methodology as that for the L1 cache. Like L1 cache, both register file and rename tables need bitline scaling at high clock speeds to allow them to be accessed every cycle. We also show L2 latency. For the same FO4, L2 latency increases with technology because of the slow scaling of global wires in L2. Because we assume a simple L2 which allows only one access at a time, its bandwidth is 1/latency and is not shown.

[2] uses ITRS 1999 projections for wire parameters, and assumes that the sense-amplifier input offset voltage scales linearly with technology. We ran simulations using these assumptions and obtained results which match [2] within 5%. However, because we use the more recent ITRS 2001 roadmap and consider the lack of scaling of sense-amplifier input offset voltage, our latencies are longer.

### 5.1.2 CAM

Table 6 shows the latency and bandwidth results for the 32-entry issue queue at different technologies. We scale select latency linearly with technology, as argued in [18]. The latency-optimized issue queue chooses a single segment for optimum latency. At higher clock speeds, the wakeup latency for single segment does not fit in one clock cycle, preventing the issue queue port from being accessed every cycle. This limitation implies that for *nb2b* (Section 3.2), even independent instructions cannot perform wakeups in consecutive cycles using the same port. This restriction does not apply to *b2b* because *b2b* uses idealized versions of schemes mentioned in Section 2.2 to circumvent such issue queue limitations. Bitline-scaled CAM uses multiple segments to allow wakeup to proceed in a pipelined fashion every cycle. For example, the bitline-scaled issue queue at 50nm technology requires 2, 3, and 4 segments for 10, 8, and 6 FO4 clock periods, respectively. Similar to bitline-scaled RAM, the bitline-scaled CAM provides more bandwidth at the expense of extra latency (at most 2 cycles), as compared to latency-optimized CAM.

We performed latency and bandwidth analysis for the load/store queue (LSQ) as well. We found that the LSQ latency is always less than L1 cache latency. Because the LSQ access and cache access proceed in parallel, the LSQ latency can be hidden behind L1 cache access. We also found the LSQ bandwidth to be

**Table 7: Functional unit and bypass wire latencies**

Technology (nm)	Clock Period (FO4)	Bypass wires	# of cycles					
			Integer		Floating Point			
			Add	Mult	Add	Div	Sqrt	Mult
100	6	2	3	28	15	50	76	17
	8	2	2	19	10	33	51	12
	10	1	2	14	8	25	38	9
70	6	3	3	28	15	50	76	17
	8	2	2	19	10	33	51	12
	10	2	2	14	8	25	38	9
50	6	4	3	28	15	50	76	17
	8	3	2	19	10	33	51	12
	10	2	2	14	8	25	38	9
Alpha	19	0.5	0.5	7	4	12	18	4

equal or better than that of the L1 cache. Therefore, we do not show LSQ results.

### 5.1.3 Bypass wires and functional units

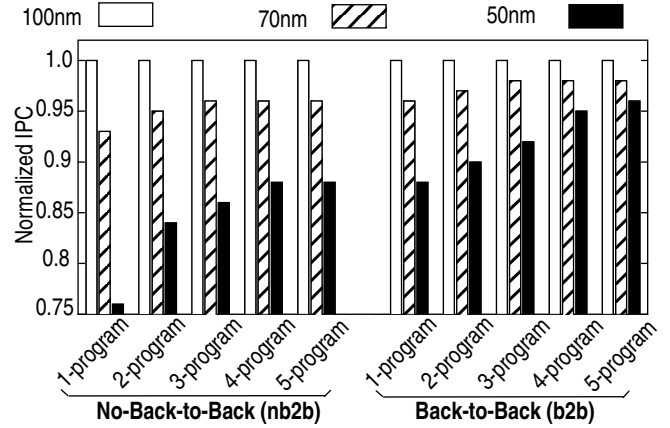
Table 7 shows the number of cycles for bypass wires and functional units. We list Alpha 21264 (800MHz at 180nm) functional-unit and bypass-wire latencies as well. Alpha 21264 has 19FO4 clock period. For bypass wires, we use the half-cycle delay in Alpha 21264 at 19FO4 clock period and scale to other clock periods. To obtain bypass delays at other technologies, we use the ITRS 2001 projections [4], [12], and [13] for wire delay scaling. For functional units, we use a methodology similar to that of [14]. Assuming 10% of clock period (1.8FO4) for latch overhead and clock skew, we scale the functional unit latencies from  $19 - 2 = 17$  FO4 to  $10 - 2 = 8$  FO4,  $8 - 2 = 6$  FO4, and  $6 - 2 = 4$  FO4 for clock periods of 10, 8, and 6 FO4, respectively. We see that while the number of cycles for functional units remain unchanged across technology, the number for bypass wire delays increase.

## 5.2 Latency effect of wire delays

In this section, we analyze the latency effect of slow scaling of wire delays on superscalar and SMT processors. As mentioned in Section 3.2, we consider the two options of *no-back-to-back* (*nb2b*) and *back-to-back* (*b2b*). Because we want to isolate the latency effect of wire delays from bandwidth effect, we assume that all the RAM/CAM structures are perfectly pipelined. In Section 5.3, we will discuss the bandwidth effects.

### 5.2.1 Relative impact of wire delays on superscalar & SMT

Figure 6 show the normalized IPCs for 1-, 2-, 3-, 4- and 5-program workloads for 100, 70, and 50nm technologies at 8 FO4 clock period for *nb2b* and *b2b*. We obtain the latency numbers for the various structures from Table 4 through Table 7. The IPC values for a certain workload and technology are normalized with respect to the IPC for that workload in 100nm technology. For example, the left-most black bar in Figure 6 represents the average single-program IPC at 50nm technology and 8 FO4 clock period normalized with respect to the average single-program IPC for 100nm technology and 8 FO4 clock period. This normaliza-


**FIGURE 6: Effect of wire delays on IPC for 8FO4 pipeline**

tion clearly shows the relative effect of slow scaling of wire delays with technology on different numbers of programs. In the interest of space, we show only IPC averages, and not individual IPCs, of the benchmarks in the single-program and multiprogrammed workloads. While we show the results for 8 FO4, 6 FO4 and 10 FO4 (not shown) follow similar trends as 8 FO4. We will show 6 and 10 FO4 to find the optimal clock period in the next subsection. As mentioned in Section 2.2, the single-program IPC substantially degrades due to relatively-increased wire delays with technology. As we go from 100nm to 50nm, the average single-program IPC decreases by 24% and 12% for *nb2b* and *b2b*, respectively. In comparison, the IPC degradation in multiprogrammed workloads due to relatively-increased wire delays is less severe. Going from 100nm to 50nm, the average IPC degradation for 2-, 3-, 4-, and 5-program workloads is 16%, 14%, 12%, and 12%, respectively, for *nb2b*, and 10%, 8%, 5%, and 4%, respectively, for *b2b*.

Two other trends are clear: First, in both single-program and multiprogrammed workloads, the IPC degradation for *nb2b* is higher than that for *b2b*. By using architectural techniques to circumvent pipeline loops (Section 2.1), *b2b* is more tolerant to wire delays and incurs less IPC degradation. However, as mentioned in Section 2.2, some loops are still lengthened, even if these techniques are used. Consequently, even *b2b* incurs IPC degradation. Using multiple programs compensates for this degradation. Second, as the number of programs increase from 2 to 5, the IPC degradation becomes progressively less substantial. As mentioned in Section 2.3, more programs provide more latency tolerance, making the relatively-increased wire delays less significant.

### 5.2.2 Optimal pipeline depths for superscalar and SMT

We discuss the optimal pipeline depths for superscalar and SMT processors, based on maximizing the number of instructions executed per second. Figure 7 shows throughput in billions of instructions per second (BIPS) for 1-, 2-, 3-, 4- and 5-program workloads at different technologies and clock periods. We obtain  $n$ -program BIPS by multiplying the clock frequency (in GHz) and the average IPC of all  $n$ -program workloads. The absolute values of BIPS are shown so that the optimal pipeline depths at each technology are clearly visible. The horizontal lines represent 2x and 4x improvement in throughput over the optimal single-pro-



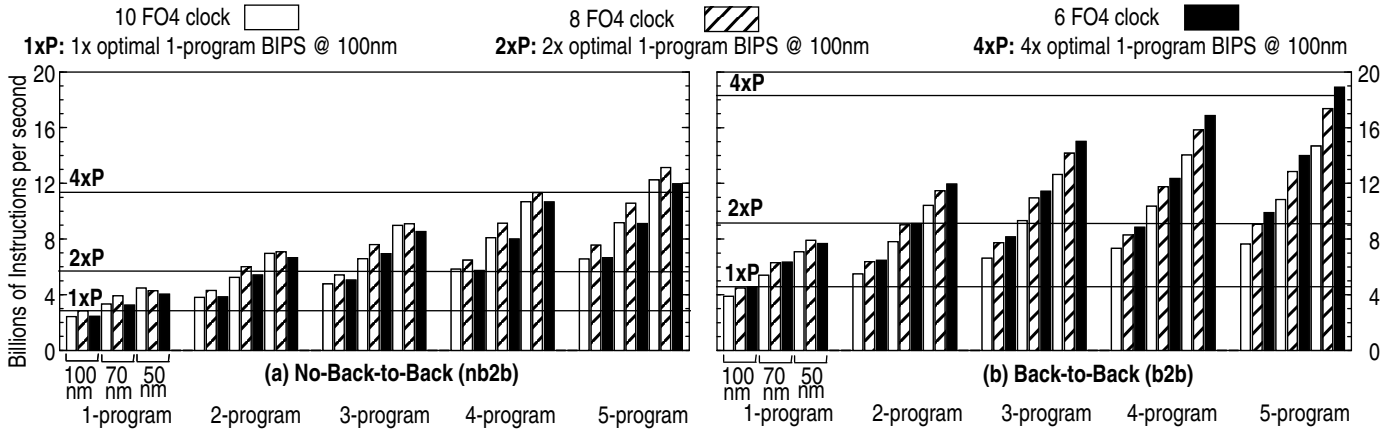


FIGURE 7: BIPS for single-threaded and multi-threaded workloads

gram throughput at 100nm technology.

For single program, the optimal pipeline depth decreases with technology. Going from 100nm to 50nm technology, the optimal clock period for single-program increases from 8 FO4 to 10 FO4 in *nb2b*, and from 6 FO4 to 8 FO4 in *b2b*. Our results for 100nm technology match with those in [14], which shows that the optimal pipeline depth for 100nm technology is 6 to 8 FO4. However, the optimal pipeline depth decreases as wire delays become more dominant, as mentioned in Section 2.2. Even in the wire-delay-tolerant *b2b*, there are some pipeline loops whose effect cannot be compensated (Section 2.2), resulting in shallower optimal pipelines.

In contrast, multiprogrammed workloads maintain a constant optimal pipeline depth across technology generations. All the multiprogrammed workloads achieve optimal clock period at 8 FO4 in *nb2b*, and 6 FO4 in *b2b*. Because the *b2b* numbers seem like throughput may not peak at 6 FO4, we simulated 4FO4 (not shown) for *b2b* and found 4 FO4 to be worse than 6 FO4, as mentioned in Section 4. This constant pipeline depth enables the clock period to be increased by 43% every technology generation without having sub-optimal throughput.

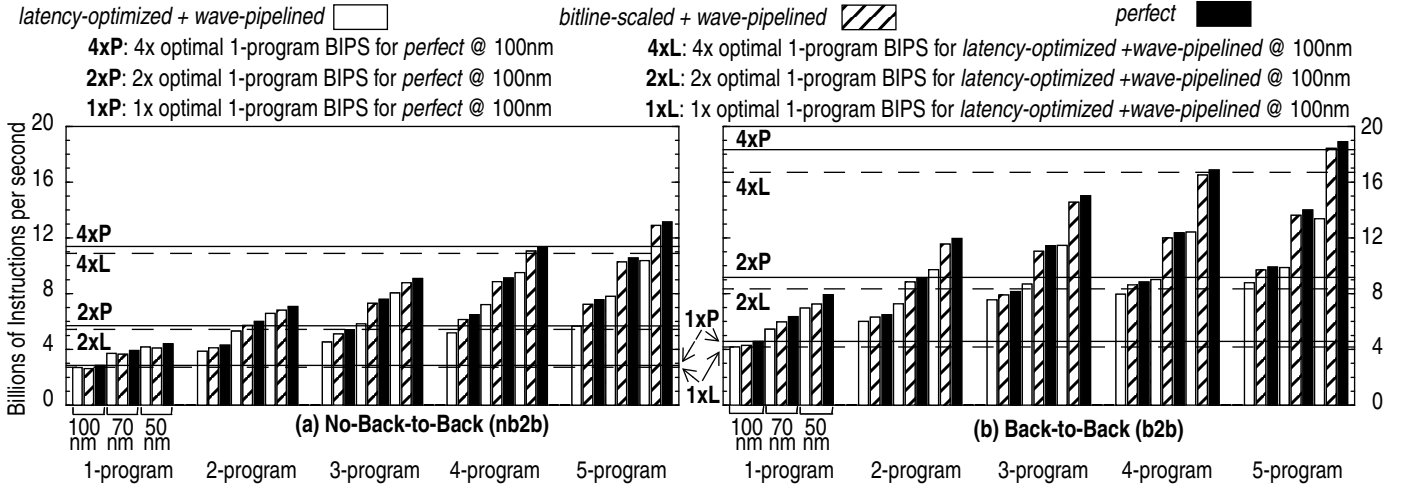
We see three other trends. First, *b2b* has a deeper optimal pipeline than *nb2b* in all technologies. Deeper pipelines cause relatively less IPC degradation in *b2b*, allowing a higher optimal clock period. Second, single-program optimal throughput scales slowly with technology. Going from 100nm to 50nm, the optimal throughput improves 1.54x (from 2.9 to 4.5BIPS) in *nb2b*, and 1.73x (from 4.5 to 7.9BIPS) in *b2b*. These improvements are significantly less than the 4x improvement in two technology generations expected by Moore’s Law. Using multiple programs, SMT achieves the 2x improvement every generation with just a modest 4-way issue width. As compared to the optimal single-program throughput at 100nm, *nb2b* requires 2 and 4 programs for 2x improvement from 100nm to 70nm and from 70nm to 50nm, respectively. *b2b* requires 3 and 5 programs for the same improvement. Third, *b2b* requires more programs than *nb2b* to provide 2x improvement every generation. Because *b2b* uses wire-delay-tolerance mechanisms, the gap between the single-program and multiple-program throughput in *b2b* is relatively less compared to that in *nb2b*. Therefore, *b2b* needs more programs. With the wire delay effect more dominant in *nb2b*, SMT requires fewer programs to scale throughput because of its ability to tolerate the

wire delays effectively. Recall from Section 3.2 that *b2b* and *nb2b* provide bounds. Therefore, a pipeline that implements some techniques to alleviate pipeline loops (such as those discussed in Section 2.2) will do better than *nb2b* and worse than *b2b* for single-program. Consequently, such a pipeline will require between 2 to 3 programs and between 4 to 5 programs for 2x improvement from 100nm to 70nm and from 70nm to 50nm, respectively.

### 5.3 Bandwidth effect of wire delays

In this section, we analyze the effect of bandwidth scaling of RAM/CAM structures on SMT throughput. As mentioned in Section 4, we assume wave pipelining in both the latency-optimized and bitline-scaled RAM/CAM. This assumption favors *latency-optimized* by reducing the bandwidth gap between *latency-optimized* and *bitline-scaled*. We consider bandwidth constraints of RAM and CAM structures in both *b2b* and *nb2b*, with the exception that the issue queue in *b2b* is perfectly pipelined, as explained in Section 5.1.2. For comparison purposes, we also show a *perfect* configuration with the latency of *latency-optimized* and the bandwidth of perfect pipelining. Because Section 5.2 assumes perfect pipelining, the results for *perfect* are the same as those using the optimal clock period for each technology in Section 5.2.

Figure 8 shows the BIPS for *perfect*, *bitline-scaled*, and *latency-optimized*, both for *nb2b* and *b2b* at different technologies. We compared other bandwidth schemes, 8-way banked cache, and line buffer having 8-full-cache-block entries (as recommended by [28]), with latency-optimized (2-ported, wave-pipelined) cache, and found latency-optimized cache to perform better (we do not show those numbers in the interest of space). While [28] reports the schemes to be better than two ports, [28] does not consider wave-pipelining, and ignores scheduling disruptions due to line buffer misses and bank conflicts (Section 3.1). Therefore, we compare to a latency-optimized cache. Though we evaluated the schemes only for caches, other RAM structures using banking or line buffer would incur similar disruptions in scheduling. Therefore, we compare to latency-optimized designs for all structures. We obtain the latency and bandwidth numbers for various structures from Table 4 through Table 7. As in Figure 7, we show average BIPS for different number of programs. All the results are shown for the clock periods found to be



**FIGURE 8: Effect of bandwidth scaling on performance**

optimal for each technology in Section 5.2. The solid horizontal lines show the 1x, 2x and 4x BIPS values as compared to the optimal single-program BIPS in *perfect* at 100nm, while the dotted horizontal lines show the 1x, 2x, and 4x BIPS values as compared to the optimal single-program BIPS in *latency-optimized* at 100nm.

The throughput for *latency-optimized* is reasonably close to that of *perfect* for single-program, but the gap widens significantly with increasing number of programs. For example, at 50nm technology, *latency-optimized* lags behind *perfect* by 5%, 7%, 11%, 16%, and 21% in *nb2b*, and by 11%, 19%, 24%, 26%, and 29% in *b2b* for 1, 2, 3, 4, and 5 programs respectively. With more programs, the bandwidth demand on RAM/CAM structures also increases. The inability of *latency-optimized* to provide the much-needed bandwidth limits the throughput of multiprogrammed workloads. *b2b* suffers more degradation due to limited bandwidth than *nb2b*. Because *b2b* uses techniques to extract more parallelism and hide latency, the bandwidth requirement in *b2b* is higher than that in *nb2b*.

The scaling trends for *latency-optimized* are substantially worse than those for *perfect*. In comparison with *latency-optimized* single-program throughput at 100nm, *latency-optimized* requires 3 programs in both *nb2b* and *b2b*, for 2x improvement at 70nm technology. Going from 70nm to 50nm technology, even 5 programs are not enough to get further 2x improvement in both *nb2b* and *b2b*. We simulated 6 programs (not shown), and found 2.02x and 1.71x improvement in *nb2b* and *b2b*, respectively. Increasing the number of programs does not provide enough improvements for *latency-optimized* due to limited bandwidth.

The throughput for *bitline-scaled* is reasonably close to *perfect* in all the workloads. For example, at 50nm, *bitline-scaled* lags behind *perfect* by 7%, 4%, 3%, 2%, and 2% in *nb2b*, and by 8%, 3%, 3%, 2%, and 3% in *b2b* for 1, 2, 3, 4, and 5 programs respectively. The increased latency of RAM/CAM structures in *bitline-scaled*, as compared to *perfect*, does not have a significantly worse impact on throughput of multiprogrammed workloads due to SMT’s ability to hide latency.

*Bitline-scaled* shows almost similar scaling trends as *perfect*. In comparison with the single-program throughput for *perfect* at 100nm, *bitline-scaled* requires 2 programs in *nb2b*, and 3 pro-

grams in *b2b* to achieve 2x improvement at 70nm, and 5 programs in both *nb2b* and *b2b* to achieve further 2x improvement at 50nm. Note that in the *latency-optimized* case we see if throughput scales to 2x and 4x of that of itself and not *perfect*, whereas in the *bitline-scaled* case, we see if throughput scales to 2x and 4x of that of *perfect*. These results confirm that bitline scaling is key to scaling SMT throughput in future technologies.

#### 5.4 Effect of issue width

In this section, we analyze the effect of increasing issue width on the scaling trends for superscalar and SMT. We use a *wide-issue* configuration for all results in this section. *wide-issue* uses 8-way issue with a 48-entry issue queue and 400 integer and FP physical registers, both for superscalar and SMT processors. Except the register file, and issue queue, the other architectural parameters for *wide-issue* are similar to those for the base configuration shown in Table 1. We compare the scaling trends in *wide-issue* due to the latency and bandwidth effects of wire delays with the scaling trends in base configuration.

Table 8 shows the latency and bandwidth results for the issue

**Table 8: Issue queue and register file results for *wide-issue***

Technology (nm)	Clock Period (FO4)	Register File		Issue Queue					
		latency(cycles)/bandwidth (access/cycle)		l-o			b-s		
		l-o	b-s	Wakeup	Select	Access/cycle	Wakeup	Select	Access/cycle
100	6	4/1	4/1	2	2	0.5	3	2	1
	8	3/1	4/1	2	1	0.5	2	1	1
	10	3/1	3/1	1	1	1	1	1	1
70	6	5/0.5	5/1	3	2	0.3	5	2	1
	8	4/0.5	4/1	2	1	0.5	3	1	1
	10	3/1	3/1	2	1	0.5	2	1	1
50	6	5/0.5	5/1	3	2	0.3	6	2	1
	8	4/0.5	4/1	2	1	0.5	4	1	1
	10	3/1	4/1	2	1	0.5	3	1	1

**Table 9: Optimal clock periods for *wide-issue***

Technology (nm)	Optimal Clock Period (FO4)									
	No-back-to-back(nb2b)					Back-to-Back(b2b)				
	Number of Programs					Number of Programs				
	1	2	3	4	5	1	2	3	4	5
100	8	8	8	8	8	6	6	6	6	6
70	10	8	8	8	8	8	6	6	6	6
50	10	8	8	8	8	8	6	6	6	6

queue and register file in *wide-issue*. For all other RAM/CAM structures, the results are the same as the ones already shown for the base case in Table 4 through Table 7. We obtain latency and bandwidth results for register file and issue queue wakeup from Cacti. We use the results in [18] to scale the select latency for wider issue.

In both the latency-optimized and bitline-scaled cases, the register file in *wide-issue* has higher latency as compared to the register file in the base case. While the latency-optimized register file in *wide-issue* has worse bandwidth, the bitline-scaled register files in both *wide-issue* and the base case have the same bandwidth.

The latency and bandwidth results for latency-optimized and bitline-scaled issue queues in both *wide-issue* and the base case show similar trends as the latency and bandwidth results for the register files in the two configurations. Also, note that the larger and wider bitline-scaled issue queue in *wide-issue* requires more segments than in the base configuration.

Table 9 shows the optimal clock period for *wide issue* at different technologies. We obtain the optimal clock periods by maximizing the IPS, using a methodology similar to the one used for the base case in Section 5.2.2. In the interest of space, we do not show complete IPS results, and show only the optimal periods.

For single program, the optimal pipeline depth decreases with technology. Going from 100nm to 50nm technology, the optimal clock period for single-program increases from 8 FO4 to 10 FO4 in *nb2b*, and from 6 FO4 to 8 FO4 in *b2b*. In contrast, multiprogrammed workloads maintain a constant optimal pipeline depth across technology generations. These results are similar to the results obtained for the base case in Section 5.2.2.

Figure 9 shows the BIPS in *wide-issue* for *perfect*, *bitline-scaled*, and *latency-optimized*, both for *nb2b* and *b2b* at different technologies. All the results are shown for the optimal clock peri-

ods shown in Table 9.

We discuss latency effect first. Recall from Section 5.3 that *perfect* has the latency of *latency-optimized* and the bandwidth of perfect pipelining. To isolate latency effect of wire delays from bandwidth effect, we analyze the results for *perfect*.

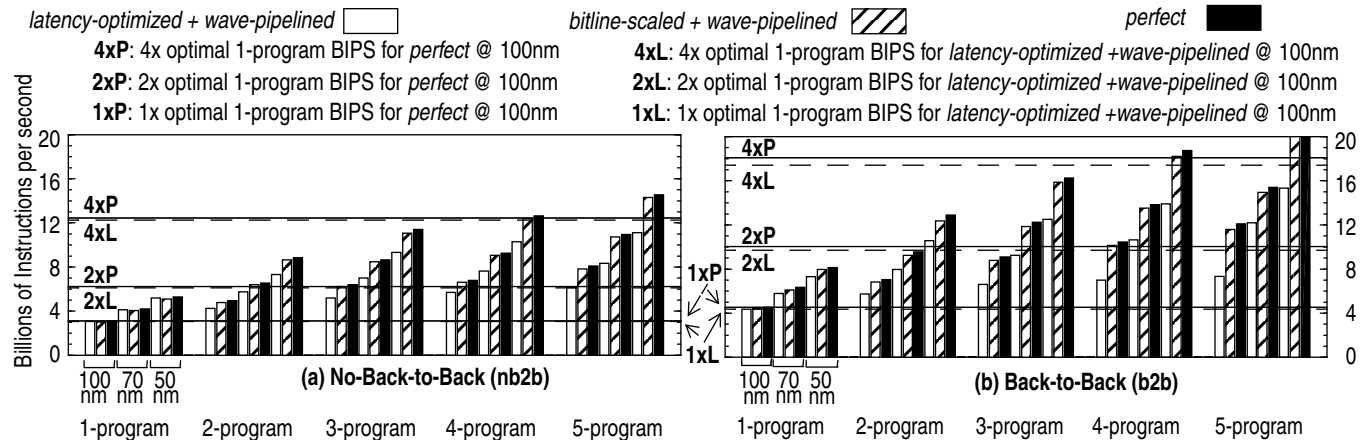
Single-program optimal throughput scales slowly with technology. Going from 100nm to 50nm, the optimal throughput improves 1.71x (from 3.1 to 5.3BIPS) in *nb2b*, and 1.76x (from 4.6 to 8.1BIPS) in *b2b*. As compared to the optimal single-program throughput at 100nm, both *nb2b* and *b2b* require 2 and 4 programs for 2x improvement from 100nm to 70nm and from 70nm to 50nm, respectively. Comparing these results with the base case results in Figure 7, we see that *wide-issue* requires fewer programs to achieve 2x improvement every generation. Because of more parallelism, SMT utilizes the increased issue width available in *wide-issue* more efficiently, and is able to achieve the 2x improvement with fewer programs.

Next, we discuss bandwidth effect. The throughput for *latency-optimized* is reasonably close to that of *perfect* in single-program, but lags behind *perfect* progressively with increasing number of programs. In comparison with *latency-optimized* optimal single-program throughput at 100nm, *latency-optimized* requires 3 programs both in *nb2b*, and *b2b*, for 2x improvement at 70nm. From results not shown here, we found that even 6 programs are not enough to get further 2x improvement from 70nm to 50nm in both *nb2b* and *b2b*. The multiprogrammed workloads, because of their higher bandwidth demand, show worse scaling trends for *latency-optimized* in *wide-issue* than in the base case.

The throughput for *bitline-scaled* is reasonably close to that of *perfect* in all the workloads. In comparison with the optimal single-program throughput for *perfect* at 100nm, *bitline-scaled* requires 2 programs to achieve 2x improvement at 70nm, and 4 programs to achieve further 2x improvement at 50nm, both in *nb2b* and *b2b*. Comparing with the base case *bitline-scaled* results in Figure 8, the *wide-issue bitline-scaled* SMT requires fewer programs to maintain the scaling trends.

## 6 Conclusions

Using simulations with SPEC2000 benchmarks, we found the following: (1) The optimal pipelines for superscalar become shallower due to slow scaling of wire delays in future technologies.



**FIGURE 9: BIPS results for *wide-issue***

(2) An SMT running a multiprogrammed workload with just 4-way issue not only retains the optimal pipeline depth over technology generations, enabling at least 43% increase in clock speed every generation but also achieves the remainder of the expected speedup of two per generation through IPC. As wire delays become more dominant in future technologies, the number of programs needs to be scaled modestly to maintain the scaling trends, at least till the near-future 50nm technology. (3) While this result ignores bandwidth constraints, using SMT to tolerate latency due to wire delays is not that simple because SMT causes bandwidth problems. We show that using conventional, latency-optimized RAM/CAM structures prevents scaling of SMT throughput. (4) We use bitline scaling to allow RAM/CAM bandwidth to scale with technology. Bitline scaling enables SMT throughput to scale at the rate of two per technology generation in the near future.

Previous papers have concluded that revolutionary changes are needed to achieve single-program performance scaling in wire-delay-dominated future technologies. We have shown that for multiprogrammed workloads, SMT suffices for the near future. While single-program performance may continue to be important, many commercial microprocessors have adopted SMT and future high-performance machines are likely to run multiprogrammed/multithreaded workloads on SMT. However, more programs will be needed to continue the scaling trends beyond the technologies we have considered.

## Acknowledgments

We would like to thank the anonymous reviewers for their comments on an earlier draft of this paper. This research is supported in part by NSF under CAREER award 9875960-CCR and NSF Instrumentation grant CCR-9986020.

## References

- [1] A. Agarwal, K. Roy, and T. N. Vijaykumar. Exploring high bandwidth cache architecture for scaled technology. In *Design Automation and Test in Europe Conference 2003*, pages 778–783, 2003.
- [2] V. Agarwal, M. Hrishikesh, S. Keckler, and D. Burger. Clock rate versus IPC: The end of the road for conventional microarchitectures. In *Proceedings of the 27th Annual International Symposium on Computer Architecture*, pages 248–259, June 2000.
- [3] B. Amrutur and M. Horowitz. Speed and power scaling of SRAMs. *IEEE Transactions on Solid-State Circuits*, 35(2):175–185, 2000.
- [4] S. I. Association. International technology roadmap for semiconductors, 2002.
- [5] E. Borch, E. Tune, S. Manne, and J. Emer. Loose loops sink chips. In *Eighth International Symposium on High Performance Computer Architecture (HPCA)*, pages 299–310, Feb. 2002.
- [6] M. Brown and Y. Patt. Using internal redundant representations and limited bypass to support pipelined adders and register files. In *Eighth International Symposium on High Performance Computer Architecture (HPCA)*, pages 289–298, Feb. 2002.
- [7] D. Burger and T. Austin. The SimpleScalar Tool set, version 2.0. Technical report, University of Wisconsin-Madison, 1997.
- [8] T. Chappell, B. Chappell, S. Schuster, J. Allan, S. Klepner, R. Joshi, and R. Franch. A 2-ns cycle, 3.8-ns access 512-kB CMOS ECL SRAM with a fully pipelined architecture. *IEEE Journal of Solid State Circuits*, 26(11):1577–1585, 1991.
- [9] V. De and S. Borkar. Technology and design challenges for low power and high performance. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 163–168, 1999.
- [10] D. Ernst and T. Austin. Efficient dynamic scheduling through tag elimination. In *Proceedings of the 28th Annual International Symposium on Computer Architecture*, pages 253–264, June 2002.
- [11] G. Hinton, D. Sager, M. Upton, D. Boggs, D. Carmean, A. Kyker, and P. Roussel. The microarchitecture of the Pentium 4 processor. *Intel Technology Journal*, Feb. 2001.
- [12] R. Ho, K. Mai, and M. Horowitz. The future of wires. *Proceedings of the IEEE*, 89(4):490–504, 2001.
- [13] R. Ho, K. Mai, and M. Horowitz. Managing wire scaling: A circuit perspective. In *Digest of Technical Papers International Interconnect Technology Conference*, 2003.
- [14] M. Hrishikesh, D. Burger, N. Jouppi, S. Keckler, K. Farkas, and P. Shivakumar. The optimal logic depth per pipeline stage is 6 to 8 FO4 inverter delays. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 14–24, June 2002.
- [15] R. Kessler. The Alpha 21264 microprocessor. *IEEE Micro*, 19(2):24–36, 1999.
- [16] K. Nogami and T. Sakurai. A 9-ns hit delay, 32-kbyte cache macro for high speed RISC. *IEEE Journal of Solid State Circuits*, 25(1):100–108, 1990.
- [17] K. Osada, H. Higuchi, K. Ishibashi, N. Hashimoto, and K. Shiozawa. A 2-ns access, 285 Mhz, two-port cache macro using double global bit-line pairs. In *ISSCC Dig. Tech. Papers*, pages 402–403, 1997.
- [18] S. Palacharla, N. P. Jouppi, and J. E. Smith. Complexity-effective superscalar processors. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, June 1997.
- [19] S. Raasch, N. Binkeri, and S. Reinhardt. A scalable instruction queue design using dependence chains. In *Proceedings of the 29th Annual International Symposium on Computer Architecture*, pages 318–329, June 2002.
- [20] T. Sherwood, G. Varghese, and B. Calder. A pipelined memory architecture for high throughput network processors. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 288–299, 2003.
- [21] P. Shivakumar and N. P. Jouppi. Cacti 3.0: An integrated cache timing, power and area model. Technical report, Compaq Computer Corporation, Aug. 2001.
- [22] G. S. Sohi and M. Franklin. High-bandwidth data memory systems for superscalar processors. In *Proceedings of the 4th International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 53–62, 1991.
- [23] E. Sprangle and D. Carmean. Increasing processor performance by implementing deeper pipelines. In *Proceedings of 29th Annual International Symposium on Computer Architecture*, pages 25–36, 2002.
- [24] J. Stark, M. Brown, and Y. Patt. On pipelining dynamic instruction scheduling logic. In *Proceedings of the 33rd International Symposium on Microarchitecture*, pages 57–66, Dec. 2000.
- [25] D. Tullsen and J. Brown. Handling long-latency loads in a simultaneous multithreading processor. In *Proceedings of the 34th International Symposium on Microarchitecture*, pages 318–327, 2001.
- [26] D. Tullsen, S. Eggers, J. Emer, H. Levy, J. Lo, and R. Stamm. Exploiting choice: instruction fetch and issue on an implementable simultaneous multithreading processor. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 191–202, 1996.
- [27] D. Tullsen, S. Eggers, and H. Levy. Simultaneous multithreading: Maximizing on-chip parallelism. In *Proceedings of the 22nd Annual International Symposium on Computer Architecture*, pages 392–403, 1995.
- [28] K. Wilson, K. Olukotun, and M. Rosenblum. Increasing cache port efficiency for dynamic superscalar microprocessors. In *Proceedings of the 23rd Annual International Symposium on Computer Architecture*, pages 147–157, 1996.