Cache Pipelining with Partial Operand Knowledge

Erika Gunadi and Mikko H. Lipasti
Department of Electrical and Computer Engineering
University of Wisconsin - Madison
{egunadi,mikko}@ece.wisc.edu

Abstract

Caches consume a significant amount of power in modern microprocessors while also constraining clock frequency due to their access time. In this paper, we propose a bit-sliced cache, which reduces dynamic power consumption and achieves higher clock frequency as well as increased cache throughput while adding little complexity. Our bit-sliced cache reduces 20-40% of dynamic power for a variety of cache organizations by activating only the necessary row decoders and subarrays. To reduce cycle time, the cache access is pipelined, which results in higher bandwidth without suffering from the complexity and power and area penalty caused by an additional cache port. We report cycle time improvements nearly proportional to the degree of bit-slice pipelining, as well as performance improvements averaging 9% and 11% for an out-of-order processor with a 2-sliced and 4-sliced cache and ALU.

1. Introduction and Motivation

The evolution of microprocessor technology, both in terms of process technology and microarchitectural innovation, has driven rapid increases in processor performance over the last several decades. As clock frequency increases, dynamic power consumption also increases, which creates challenges for integrating more transistors in a single chip due to cooling, packaging, and reliability problems. The size of on-chip caches is also increasing rapidly, but there is pressure to keep access times low to provide high performance. The device count for on-chip caches often becomes a significant fraction of the total transistor count for the entire chip. Hence, the power consumed by the cache becomes a significant part of the total power consumption, e.g. 25% of the total chip power for the DEC 21164 [1] and 43% of the total power for the SA-110 [2].

This paper proposes a relatively straightforward scheme that simultaneously alleviates both dynamic power and effective access latency for primary data caches. We propose bit-slice-pipelined cache access and show that this scheme both saves power and improves performance over a nonpipelined baseline case. This scheme also does not add much hardware complexity or verification complexity compared to the previous work on cache subbanking [3] and cache pipelining [12]. The proposed optimization is enabled by the early availability of partial operand values; such partial values are available when ALUs are structured in a staggered fashion, as in the Pentium 4 processor [11]. Early availability allows overlapping of the pipelined cache stages with address generation and mitigates the expected negative effects of deep pipelining. Further, the increased cache bandwidth realized by pipelining of the cache results in measurable speedup. We show that this new organization saves 20-40% of dynamic L1 data cache power while improving performance by 9% and 11% for 2-slice and 4-sliced caches in a deeply-pipelined out-of-order processor.

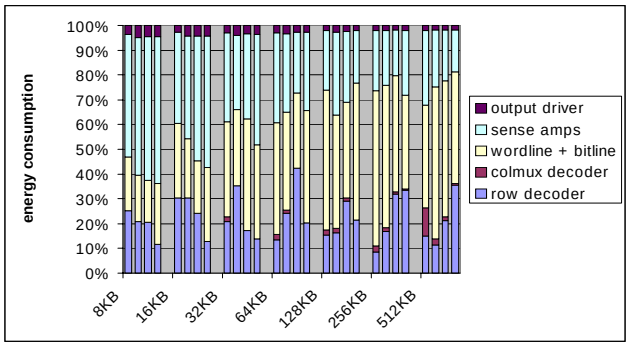

FIGURE 1. Power Consumption in Caches. Figure above shows the division of power consumption in caches with various sizes (8KB-512KB), direct mapped to 8-way (left to right). Power consumption by row-decoder range from 9% to 42%**.**

2. Background

2.1. Power Savings in Caches

Several techniques have been proposed for reducing the power dissipated by caches. These techniques include cache subbanking [3,4], bitline segmentation [4], cache decomposition [5], and block buffering [3]. Other techniques focus on leakage power (e.g. drowsy caches [6,7], cache decay [8], gated-Vdd [9], scheduling techniques [18] and alternative realizations [19].

2.2. Cache Subbanking

Cache subbanking was proposed by Su [3] to reduce power consumption in caches by fetching only the requested subline, rather than the entire logical cache line. In order to achieve this goal, the data array in the cache is partitioned into several subbanks, and only the subbank containing the

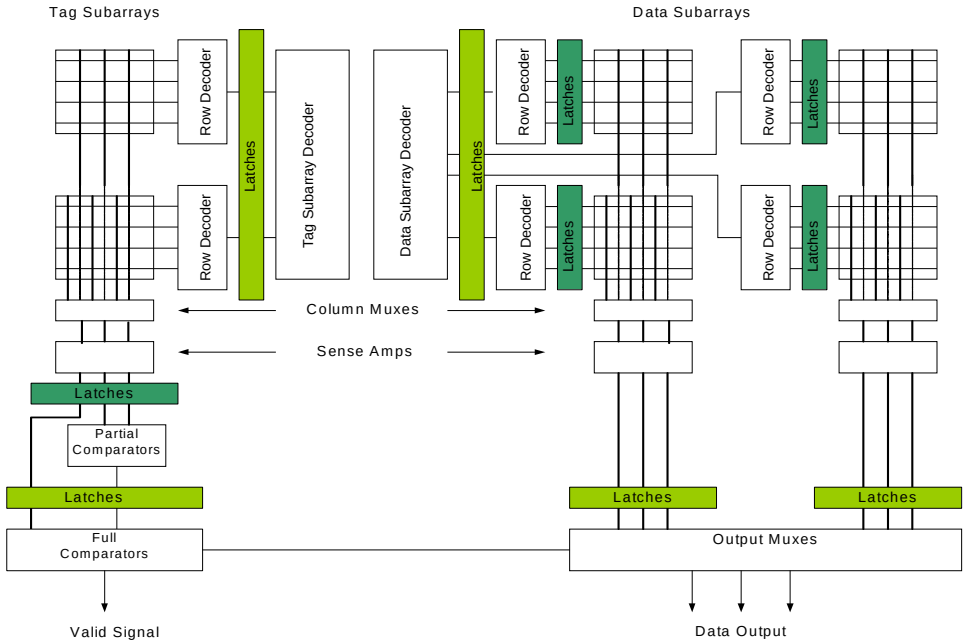

FIGURE 2. Bit-sliced Cache. Shown above are all pipeline latches needed for a four-sliced cache. A two-sliced cache can be formed from the picture above by removing the lighter latches and partial comparators. Note: input latches and output latches are not shown in the picture.

requested data is activated. Therefore, a n-subbanked data array only consumes approximately 1/n power compared to a conventional cache data array.

Further study by Ghose and Kamble [4] not only divides the data array vertically as proposed by Su, but also partitions the data array horizontally into several segments of bit-cells to get more power savings. In this technique, greater power reductions are achieved with smaller precharge drivers and sense amplifiers.

2.3. Bit-sliced ALUs

Bit-slicing the ALU was originally proposed by Hsu [10] to obtain a high-performance, cost-effective pipeline when the workload contains a large number of integer additions. Addition is often a cycle-time limiter due to propagation of the carry bit through every bit position; it would be helpful if the operation was sliced, for example, to be four pipelined eight-bit additions instead of one 32-bit addition. In this case, if there are two dependent addition operations, it is not necessary to wait until the first addition is finished. Instead, the partial operand can be bypassed to a dependent instruction that computes its low-order slice in parallel with the high-order slice of the first instruction.

A bit-sliced ALU has actually been successfully implemented in the Pentium 4 processor's *staggered adders* [11]. In this design, most integer arithmetic operations are executed across two half-cycle pipeline stages, where each stage computes 16 bits of the result, followed by internal bypassing of the partial results to dependent instructions.

2.4. Cache Pipelining

In order to achieve a high bandwidth cache, Agarwal [12] proposed pipelining the cache access by inserting latches between modules in the cache. Cache access time is divided into decoding delay, wordline to sense amplifier delay, and mux to data out delay. Using this technique, cache accesses can start before the previous access is completely done, resulting in high bandwidth and a high frequency cache.

3. Bit-sliced Cache

We propose an extension of cache subbanking that saves additional power by enabling the row decoder only for the subbank that is being accessed. A subarray decoder is added in series with the row decoders in order to determine which row decoder and subarray will be activated. Low order index bits are fed to the subarray decoder to do this selection while the rest of the index bits are fed to the row decoders. Since address decoding consumes up to one-third of active power (see Figure 1), selective row decoding can reduce a significant amount of cache power consumption.This scheme added minimal changes to cache subbanking since the column mux decoder already exists to do bank selection in parallel with row decoding operation.

However, since this proposed logic needs to be accessed in series with the row decoder, it increases the cache access time. One way to hide this delay is by pipelining the cache access, as shown in Figure 2. According to the degree of bit-slicing (two or four), cache operations are pipelined as follows:

The access steps for four-sliced <two-sliced> pipelined cache are:

Cycle 1 <Cycle 1>:
- Start subarray decoding for data and tag

Cycle 2:
- Activate necessary row decoders using the result of the subarray decoding and do row decode operation
- Read tag array while waiting for data row decoding

Cycle 3 <Cycle 2>:
- Read data array
- Concurrently, do partial tag comparison

Cycle 4:
- Compare the rest of the tag bits to determine if it is hit or miss
- Use tag comparison result to select data if cache associativity is greater than one

Operations done for each pipeline stages are carefully chosen to balance the latency of each stage. Since the critical path for the data array is different than the one for the tag array, cache operations for tag and data part are arranged differently. Due to the fact that tag comparison takes the longest time in the tag access path while reading the data array is the critical path for the data access path, tag array access is shifted to the earlier cycle to even out the latency. Figure 2 shows the bit-sliced cache with latches to perform pipeline operations as described in the previous paragraph.

Besides saving power, a bit-sliced cache can also provide performance benefit. Most researchers agree that cache access and addition are two major limiting factors for increasing cycle time. By breaking down the cache access into several cycles, cache cycle time can be eliminated from the list of limiting factors. Moreover, pipelining the cache also increases the cache bandwidth. An *n*-sliced cache can produce the same throughput as a conventional cache with *n* ports. Since adding ports means increased area and power consumption, pipelining the cache access offers the same throughput with less cost. The fact that modules in bit-sliced cache are separated by latches also makes timing verification less complex since the timing of cache modules can be verified separately.

Measurable performance benefit can be achieved when a bit-sliced ALU is used. In this case, the cache access can start as soon as the first slice (low order bits) of the address is available. However, since subarray decoding has to be done using the first slice, a minor drawback exists when the data array size is large while the address slice is small. When the address slice is too small, it is hard to achieve optimal power savings since the cache data array can only be partitioned into a limited number of subarrays, since only a few low-order bits are available in time for the subarray decode.

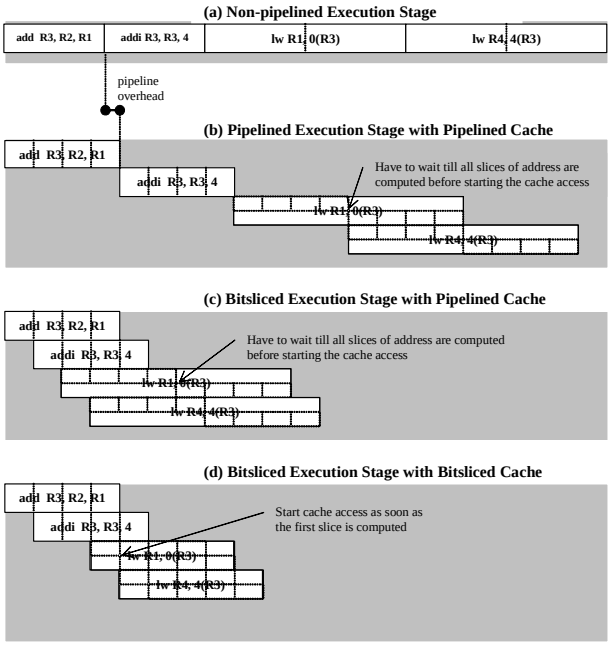

FIGURE 3. Pipelining with bit-sliced cache. (a) Non-pipelined execution stage. (b) Dependent add and load instruction has to wait until all bits are computed. (c) Dependent load instruction has to wait until all bits are computed. (d) Dependent load instruction can start accessing the cache as soon as the first slice is available. Note that load instruction is decomposed into address computation and cache access**.**

Figure 3 presents a high-level overview of pipelined execution using a four-sliced cache. Without a bit-sliced cache, cache operations in a pipelined execution stage have to wait for all address bits to be available before starting the access. When a four-sliced cache is used, an access can start as soon as the first slice of the address is available.

It is important to note that the power saving benefit of this

TABLE 1. Machine Configurations.

Out-of-order Execution	4-wide fetch/issue/commit, 128 ROB, LSQ, 32-entry scheduler, speculative scheduling for loads: replay load and dependent instructions on load mis-schedule, 20-stage pipeline
Branch Predictions	64K-entry gshare, 8-entry RAS, 4-way 512-entry BTB
Memory System (latency)	L1 I-Cache: 32KB, 2-way, 64B line size (1-cycle) L1 D-Cache: 8KB, 4-way, 64B line size (1-cycle), virtually tagged and indexed L2 Unified: 512KB, 8-way, 128Bline size (6-cycle) Off-chip memory: 100-cycle latency
Functional Units	4 integer ALU (1-cycle), 1 integer mult/div (3/20-cycle), 4 floating-point ALU (2-cycle), 1 floating-point mult/div/sqrt(4/12/24-cycle)

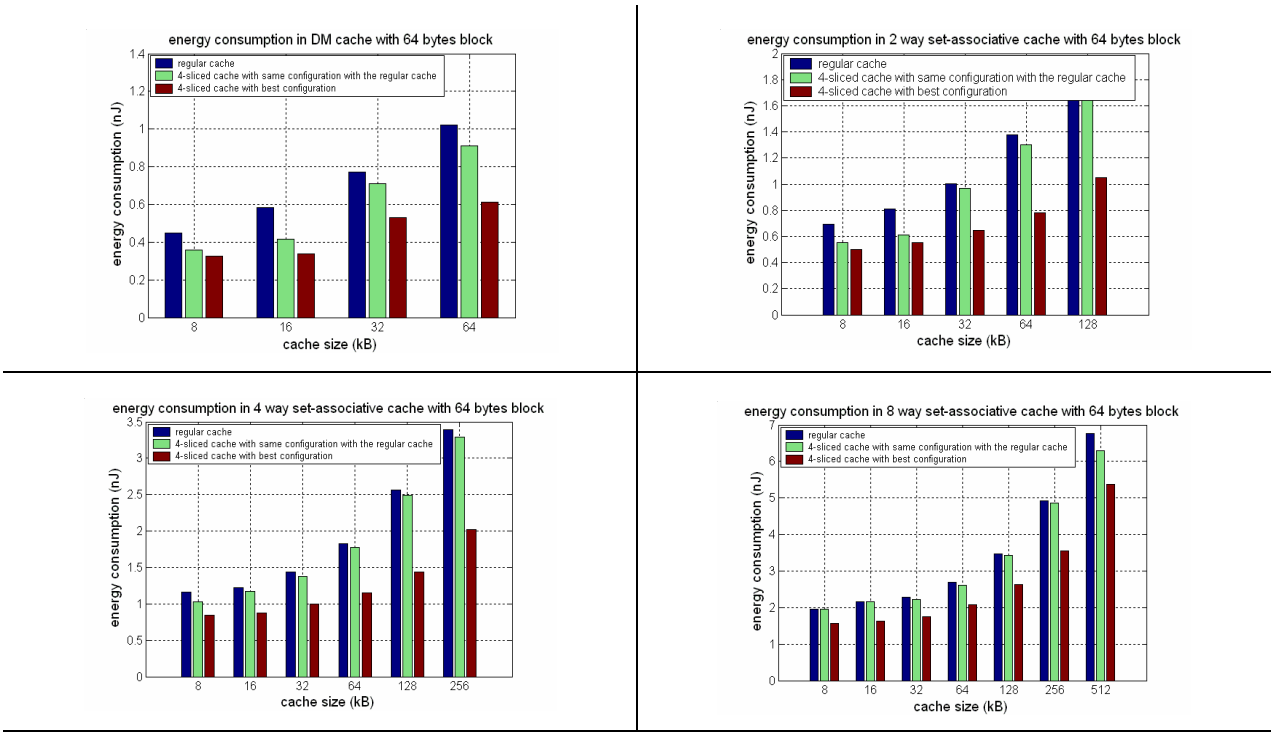

FIGURE 4. Energy Consumption per Access among Various Caches.

technique is not limited to L1 cache. Lower level on-chip caches can also save some power by adding subarray decoders and only activating the necessary row decoders. Since lower level cache does not directly affect a processor's cycle time, it may not be necessary to pipeline the cache access in order to hide the additional access latency caused by the subarray decoder. However, when faster cycle time is needed, latches can be added accordingly.

4. Experimental Framework

The evaluation methodology combines a detailed cache model to estimate power consumption and cache latency and a detailed processor simulation for performance analysis.

A modified version of the CACTI 3.0 simulator [13,14,15] is used to do power and latency characterization of each cache operation and to model the bit-sliced cache. Given a cache organization, CACTI will enumerate every possible internal configuration and choose the one with the best weighted value. Components use for the weight are cycle time and energy consumption. For our study, we use 0.18um technology and one read-write port. We choose several different cache sizes: 8KB, 16KB, 32KB, 64KB, 128KB, 256KB, and 512KB, block size 64B, and four different associativities: 1, 2, 4, 8. Since we use a 32-bit address in our model, based on the pipeline stages in the previous section, the maximum cache size we used is 64KB for direct mapped cache, 128KB for 2-way set associative cache, 256KB for 4-way set associative cache, and 512KB for 8-way set associative cache.

We use a heavily modified version of the SimpleScalar 3.0 simulator [16] to model a processor with bit-sliced execution stages and speculative slice execution [17]. A benchmark suite consisting of eight programs chosen from SPECint2000 are used in this study. These benchmarks are compiled for the SimpleScalar PISA instruction set with optimization level -O3. The benchmarks are run with the full reference input sets, fast-forwarding 400M instructions and simulating 100M instructions. Our model supports speculative scheduling with selective recovery; instructions that are data dependent on loads are scheduled as if the load instruction hits in the cache, and then replayed if the load misses. Machine configurations used for the simulation are shown in the Table 1.

5. Experimental Results

5.1. Power Saving and Cycle Time Reduction

Our cache model is based on the CACTI cache model. Given cache size, associativity, block size, number of ports, and technology, CACTI will calculate access time, power consumption, and area of the cache. CACTI uses six organization parameters to determine the most desirable cache (lower power consumption and lower cycle time): Ndbl, Ndwl, Nspd, Ntbl, Ntwl, and Ntspd. Ndbl and Ntbl determine how many times data and tag array is cut horizontally. Ndwl and Ntwl determine the number of times data and tag array is cut vertically to produce smaller subarrays. Nspd and Ntspd indicate how many sets are mapped to a single wordline. Using these parameters, data and tag array are subdivided into smaller subarrays. CACTI will go through every possible valid configuration to get the best cache structure based on the weighted value of time, power, and

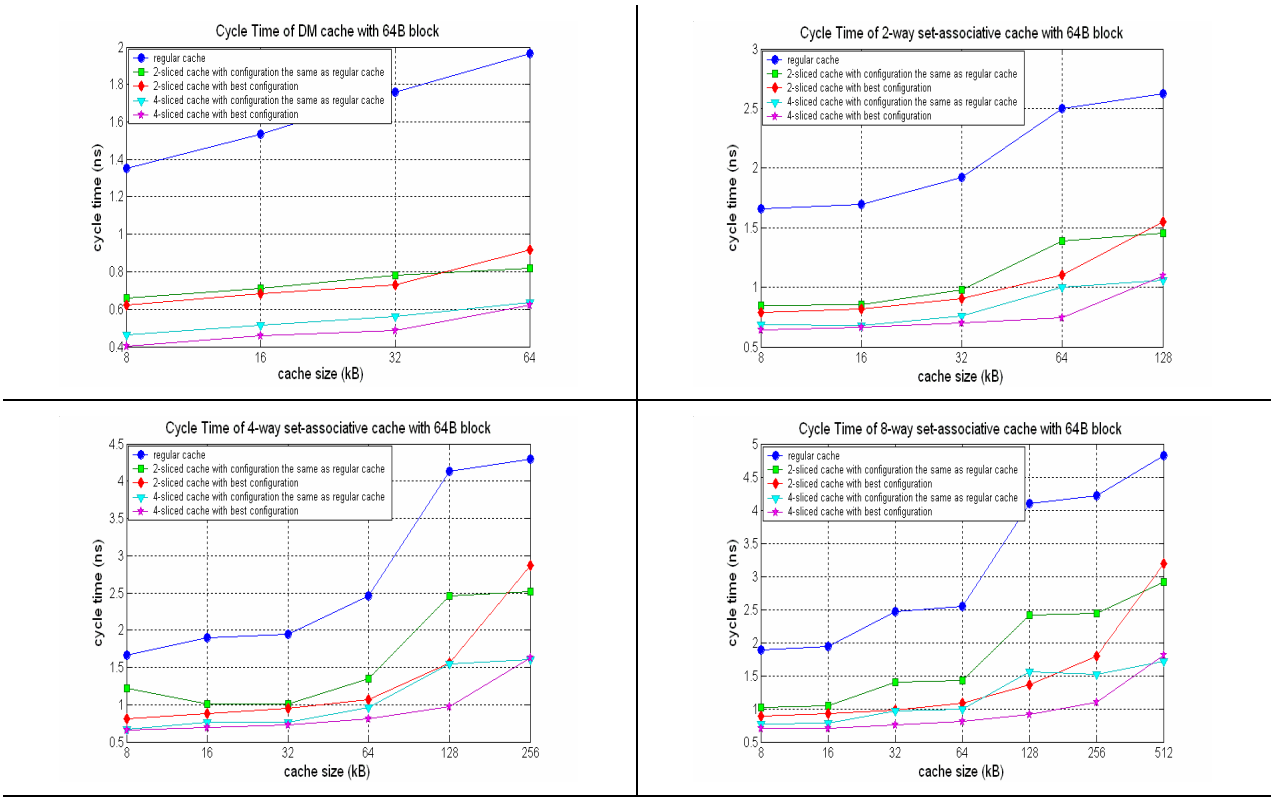

FIGURE 5. Cycle Time Comparison among Various Caches. The cycle time of a 2-sliced cache ranges from 50% to 70% of the cycle time of a regular cache. While the cycle time of a 4-sliced cache ranges from 25% to 40% of the cycle time of a regular cache**.**

area.

CACTI models a technique similar to cache subbanking and bitline segmentation for saving power in the data and tag arrays. A column mux decoder is accessed in parallel with the row decoders to determine which column muxes are activated and which subarray to access. Besides reducing power consumption, smaller subarrays also enable the cache array to be as square as possible to minimize wire capacitance, which results in faster access time and lower power dissipation. However, since each subarray has its own row decoder, more subarrays mean more power is consumed by row decoders. This last factor becomes a limiting factor for CACTI to partition the data and tag array further into more and smaller subarrays.

We enhanced CACTI to accommodate the subarray decoders described in Section 3. By modifying the colmux decoder and putting it in series with the row decoder, we decrease power consumption, since only the required row decoder will be activated. Since adding more subarrays costs less power in our model, the data and tag array can be partitioned further into smaller subarrays. This results in more power saving as less power will be consumed by these smaller subarrays. Of course, the series configuration increases access latency, which we mitigate with pipelining.

Simulation results show that a bit-sliced cache saves approximately 20% to 40% of power per access compared to a regular cache, which is quite significant. Figure 4 compares power consumption between regular cache, four-sliced cache with the same internal organization as the regular cache, and a four-sliced cache with the best organizational parameters. We have observed similar behavior for a two-sliced cache.

The percentage of energy reduction increases as the cache size grows larger. This is consistent with the fact that the percentage of power allocated for decoding increases as cache size increases. Also, a larger cache can be divided into more subarrays, which results in greater savings. However, the energy savings in a bit-sliced cache with the same organizational parameters as the regular cache is not as much as the one with best configuration. This is due to the fact that bit-sliced cache with the same organizational parameters as regular cache cannot save power from more aggressive subarray partitioning.

In a bit-sliced cache, cycle time is calculated as the maximum latency of the cache operation steps listed in section 3. Figure 5 shows the comparison of cycle time between regular caches, two-sliced caches (best configuration and the same configuration used by regular cache), and four-sliced caches (best configuration and the same configuration used by regular cache).

Since the cycle time is calculated from the maximum latency of cache sub-operations, the pipelined cache does not always achieve the optimal 50% (2-sliced) or 75% (4-sliced) improvement in cycle time. Figure 5 shows that the

TABLE 2. Benchmark Programs Simulated.

Bench-mark	Number of Instructions Simulated (FastForward)	IPC on Base Model
bzip2	100 M (400 M)	1.414
gcc	100 M (400 M)	0.946
gzip	100 M (400 M)	1.630
mcf	100 M (400 M)	1.337
parser	100 M (400 M)	0.784
twolf	100 M (400 M)	1.017
vortex	100 M (400 M)	1.053
vpr	100 M (400 M)	1.478

cycle time of the two-sliced cache ranges from 50% to 70% of the cycle time of the regular cache, while the cycle time of a four-sliced cache ranges from 25% to 40% of the cycle time of the regular cache. In some cases, bit-sliced caches with the best configuration result in a longer cycle time than caches with the same configuration as the regular cache. This occurs since CACTI chooses the best configuration using a weighted factor of cycle time and power consumption so that the cache with the shortest cycle time will not always be chosen to be the best one. The cost function could be modified to favor cycle time or power consumption depending on how balanced the processor's remaining pipe stages are.

5.2. IPC Evaluation

In this section, we compare the performance of the base case, with a conventional single-cycle execution stage, with a pipelined execution stage and a pipelined cache (PA+PC), a bit-sliced/staggered ALU integer unit with a pipelined cache (BA+PC), and a bit-sliced/staggered ALU integer unit and a bit-sliced cache (BA+BC). We study two different configurations: sliced by two, in which 32-bit register operands are divided into two 16-bit slices, and sliced by four, in which 32-bit register operands are divided into four 8-bit slices. We assume a fixed cycle time for each case, with double-clocked (2-slice) and quad-clocked (4-slice) execution and memory access stages for the pipelined and bit-sliced ALU and cache. The results in Figure 5 suggest that double- and quad-clocking is achievable for our 8K4W data cache, depending on how balanced the other pipestages are and what fraction of cycle time is dedicated to clock skew and latch overhead. We note that our 2-slice BA+PC case is most similar to the Pentium 4 configuration [11].

The IPC results for our base case is shown in Table 2. The speed up comparison over the base model is shown in Figure 6. We see that IPC increases as pipelining is applied due to increased execution bandwidth (fewer structural hazards) and increases more as bitslicing is applied due to reduced effective latency. On average, the model with pipelined ALU and pipelined cache (PA+PC) gains 2.8% (two-slice) and 3.0% (four-slice) speedup over the non-pipelined machine.When a bit-sliced ALU is added, the speedup

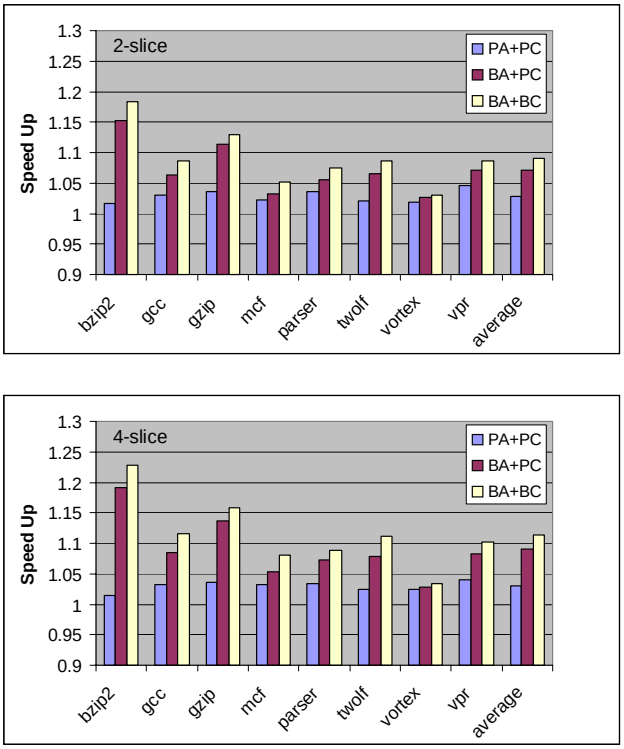

FIGURE 6. Speed-up comparison. On the average, BA+BC achieve 9.0% (2-slice) and 11.3% (4-slice) speed up over non-pipelined design**.**

become 7.2% and 8.9% for two-slice and four-slice respectively. As both ALU and cache are bit-sliced, speedup of 9.0% (two-slice) and 11.3%(four-slice) is achieved.

6. Conclusions

In a bit-sliced cache, through the addition of a subarray decoder in series with row decoders and the activation of only necessary subarrays, significant power reduction is achieved without adding much hardware complexity. The delay caused by the subarray decoder is overcome by pipelining cache access which results in considerable potential cycle time reduction.

Also, since the bit-sliced cache provides more bandwidth through pipelining and can be accessed as soon as there are enough bits available, it provides measurable speedup when the bit-sliced cache is used together with a bit-sliced ALU. This additional bandwidth can be realized in a much simpler and complexity-effective manner than conventional approaches that add banks or additional cache ports.

References

[1] J.F. Edmondson, et al. Internal Organization of the Alpha 21164, a 300-MHz 64-bit Quad-issue CMOS RISC Microprocessor, In *Digital Technical Journal*, vol. 7, no. 1, 1995.

[2] J. Montanaro, et al. A 160 MHz, 32b 0.5 W CMOS RISC Microprocessor, In *IEEE ISSCC 1996 Digest of Papers*, 1996.

[3] C.L. Su and A.M. Despain. Cache Designs for Energy Efficiency, In *Proceedings of the 28th Annual Hawaii International Conference of System Sciences*, 1995.

[4] K. Ghose and M.B. Kamble. Reducing Power in Superscalar Processor

Caches Using Subbanking, Multiple Line Buffers and Bit-Line Segmentation, In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2001.

[5] M. Huang, J. Renau, S. Yoo, and J. Torellas. L1 data Cache Decomposition for Energy Efficiency, In *Proceedings of the International Symposium on Low Power Electronics and Design*, 2001.

[6] K. Flautner, N.S. Kim, S. Martin, D. Blaauw, T. Mudge. Drowsy Caches: Simple Techniques for Reducing Leakage power. In *ACM SIGARCH Computer Architecture News*, vol. 30, May 2002.

[7] N.S. Kim, K. Flautner, D. Blaauw, T. Mudge. Energy Efficient Memory Systems: Drowsy Instruction Caches: Leakage Power Reduction using Dynamic Voltage Scaling and Cache Sub-bank Prediction, In *Proceedings of the 35th annual International Symposium on Microarchitecture*, November 2002.

[8] S. Kaxiras, Z. Hu, and M. Martonosi. Cache Decay: Exploiting Generational Behavior to Reduce cache Leakage Power, In *Proceedings of the 28th annual International Symposium on Computer Architecture*, May 2001.

[9] M. Powell, et.al. Gated-Vdd: A Circuit technique to reduce leakage in deep-submicron cache memories, In *Proceesings of International Symposium of Low power Electronics and Design*, 2000.

[10] P.Y.T Hsu, J.T. Rahmeh, E.S. Davidson, and J.A. Abraham. TIDBITS: Speedup via Time-Delay Bit-Slicing in ALU Design for VLSI Technology, In *Proceesings of the 12th Annual International Symposium on Computer Architecture*, June 1985.

[11] G. Hinton, D. Sager, N.mUpton, D. Boggs, D. Carmean, A. Kyker, P. Roussel. The Microrchitecture of the Pentium 4 Processor, *Intel technology Journal Q1*, 2001.

[12] A. Agarwal, K. Roy, T.N. Vijaykumar. Exploring High Bandwidth Pipelined cache Architecture for Scaled Technology, In *Design, Automation, and Test in Europe Conference and Exhibition*, 2003.

[13] S.J.E. Wilton and N.P. Jouppi. CACTI: An Enhanced cache Access and Cycle Time Model, In *IEEE Journal of Solid-State Circuits*, May 1996

[14] G. Reinman and N.P. Jouppi. CACTI 2.0: An Integrated Cache Timing and Power Model.

[15] P. Shivakumar and N.P. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power, and Area Model.

[16] D.C. Burger and T.M. Austin. The SimpleScalar Tool Set, Version 2.0, Technical Report CS-1342, Computer Sciences Dept. , University of Wisconsin-Madison, 1997.

[17] B. Mestan and M.H. Lipasti. Exploiting Partial Operand Knowledge, In *Proceedings of International Conference on Parallel Processing*, 2003.

[18] C.L. Su and A.M. Despain. Cache Design Tradeoffs for Power and Performance Optimization, In *Proceedings of the International Symposium on Low Power Design*, 1995.

[19] R. Fromm, et. al. The Energy Efficiency of IRAM Architecture, In *Proceedings of the 24th International Symposium on Computer Architecture*, 1997.