I-Cache Multi-Banking and Vertical Interleaving

Sangyeun Cho

Department of Computer Science
University of Pittsburgh
Pittsburgh, PA 15260, USA
cho@cs.pitt.edu

ABSTRACT

This research investigates the impact of a microarchitectural technique called *vertical interleaving* in multi-banked caches. Unlike previous multi-banking and interleaving techniques to increase cache bandwidth, the proposed vertical interleaving further divides memory banks in a cache into vertically arranged sub-banks, which are selectively accessed based on the memory address. Under this setting, we are particularly interested in how accesses to instruction cache are dispersed toward different cache banks. We quantitatively analyze the memory access pattern seen by each cache bank and establish the relationship between important cache parameters and the access patterns. Our study shows that the vertical interleaving technique distributes accesses among different banks with tightly bounded run lengths. We then discuss possible applications that utilize the presented concept, including power density reduction. Very simple interleaving configurations can lead to as much as 67% reduction of maximum power density under a realistic machine configuration. Our study suggests that the idea of vertically interleaving cache lines has potential for optimizing memory accesses in a number of interesting ways.

Categories and Subject Descriptors: B.3.1 [**Design Styles**]: Cache memories; B.7.1 [**Types & Design Styles**]: VLSI (Very Large Scale Integration)

General Terms: Design

Keywords: Cache memory, memory sub-banking, power density

1. INTRODUCTION

Cache memories have long been used in microprocessors as an indispensable means of achieving lower memory access latency and external bus traffic [17]. A well-designed memory hierarchy based on caches provides a virtual view of fast and large main memory. A great deal of research has been devoted to novel cache organization and management, as well as uncovering program behaviors that allow clever handling of memory accesses for even more performance improvement [4, 10, 12].

The basic cache design parameters are *cache size*, *block size*, and *associativity*. From the viewpoint of the internal organization of a cache, there are again a number of design choices. For example, SRAM array can be partitioned into several banks to save power [20] or to tweak the cache dimensions to fit smoothly in the given silicon real estate. Such internal cache organizations, often not explicitly disclosed in literature, affect cache footprint on silicon, power consumption, and access latency [21].

This paper investigates a less explored internal structure of a cache, namely *vertical line interleaving* over multiple banks. Cache line interleaving in a multi-bank cache has been used in recent high-ILP processors to increase data cache bandwidth [18, 22]. We call this type of interleaving *horizontal*, since all the available cache banks are accessed and made visible to the processor simultaneously so that cache lines from these banks become available for multiple read operations in the same clock cycle. On the other hand, cache banks can be arranged *vertically*, allowing only one bank to be visible to the processor at a time, as in Figure 1(b), (c), and (d). It is possible that cache banks in a horizontal multi-bank cache can be vertically sub-divided into a number of banks.

In this work, we study the impact of vertical interleaving in the context of a multi-banked instruction cache. We are particularly interested in how the vertical interleaving technique changes the access pattern seen by each cache bank. Furthermore, how basic cache parameters interact with the access patterns is studied. Our quantitative evaluation indicates that vertical line interleaving finely distributes the observed memory accesses among multiple banks, effectively limiting per-bank spatial locality. Applying this concept maintains the outside view of the *architected* instruction cache, while uncovering an interesting and important opportunity for previously unavailable optimizations.

As an application of the proposed concept, we studied how vertical interleaving can lead to power density reduction in an instruction cache. We observed up to 48% reduction in maximum power density with two banks and up to 67% with four banks. Microarchitectural structures with a lower power density can lead to a cooler design by generating less heat and allowing more lateral thermal migration [16]. The practical example given in this work and the simplicity of the proposed vertical interleaving technique promise to bring easy, quick, and transparent improvement of a desired

GLSVLSI'07, March 11–13, 2007, Stresa-Lago Maggiore, Italy.

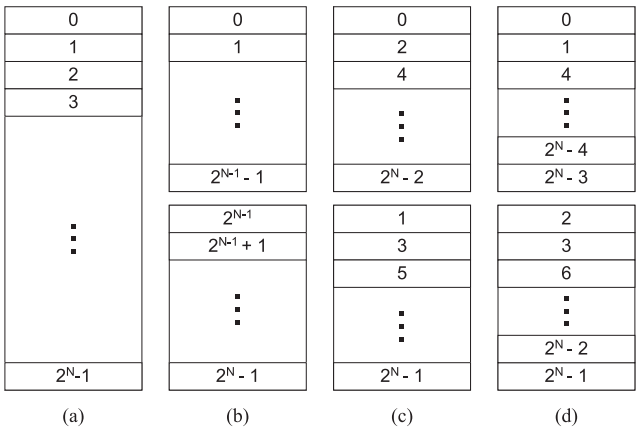

Figure 1: (a) A single-bank cache with cache lines labeled with their index. The structure can be also thought of as a conventional cache bank in a multi-banked cache; (b) Two banks with consecutive line placement; (c) Two banks with interleaving; (d) Two banks with two-line group interleaving.

quality, such as power density, to existing microprocessors.

The rest of this paper is organized as follows. We briefly summarize the cache line interleaving techniques and introduce vertical interleaving in the next section. A qualitative, yet insightful analysis of the impact of vertical interleaving will be presented. Section 3 then provides a quantitative evaluation of the proposed technique. Section 4 discusses other previous works and compares them with the present work. Finally, concluding remarks and future works will be summarized in Chapter 5.

2. INTERLEAVING LINES AMONG CACHE BANKS

2.1 Basic concepts

Cache line interleaving has to do with *which memory bank each line is placed onto*, given multiple cache banks. One may reorder cache lines within a single cache bank, but this case poses little insight and consequences. In this subsection, we will use a multi-banked direct-mapped cache to present the concept in a clear manner. The concept introduced here is not limited to direct-mapped caches, however.

A conventional cache structure organizes its L cache lines in an increasing order of their index. If we have an N-bit index, L equals 2^N and the index ranges from 0 to 2^N-1, as can be seen in Figure 1(a). Let's denote the N index bits to be {I[$N-1$], I[$N-2$], I[$N-3$], ..., I[1], I[0]}.

Directly partitioning this monolithic structure will result in two banks shown in Figure 1(b), where I[$N-1$] is used to select between the two banks, and the remaining ($N-1$) bits {I[$N-2$], I[$N-3$], ..., I[1], I[0]} are used for indexing. We call I[$N-1$], with which we can determine the accessed bank, the *bank selection bit*. In this setting, each cache line with the index "X" is placed next to the cache lines labeled "X-1" and "X$+1$" except the cache lines on the four bank boundaries. If we have more than two banks, say four or eight, we will need multiple bank selection bits.

Now, Figure 1(c) shows a cache with two banks, each containing the lines with even labels and the lines with odd labels, respectively. In this case, I[0] is used to decide the bank, and the bits {I[$N-1$], I[$N-2$], ..., I[2], I[1]} are used for indexing. In general, if high-order bits are used to select a bank as in Figure 1(b), the interleaving scheme is called *high-order interleaving*, and if low-order bits are used, the corresponding scheme is called *low-order interleaving*, as shown in Figure 1(c).

We can choose other bits as bank selection bit(s). For example, if we choose to use I[1], we are interleaving two cache lines as a group, as can be seen in Figure 1(d). In this paper, we focus on 2^M*-line group low-order interleaving*, by choosing B low-order bits starting from I[M] as the bank selection bits, when there are 2^B banks. In Figure 1(c), M is 0 and in Figure 1(d) M is 1. B is 1 in both the cases, since there are only two banks in the examples. With this setting, a high-order interleaved structure in Figure 1(b) can be thought of as a low-order interleaved one with a $2^{(N-1)}$-line group (M is ($N-1$)).

2.2 Impact of vertical line interleaving

Unlike previous multi-banking and interleaving techniques to increase cache bandwidth [18, 22], the proposed vertical interleaving further divides memory banks in a cache into vertically arranged sub-banks, which can be selectively accessed based on the memory address. Cache lines, sequentially arranged in an equivalent single bank, are interleaved among the sub-banks as in Figure 1(c). The net impact of vertical line interleaving is a modified sequence of memory accesses toward cache banks. In fact, it is not particularly difficult to portray how this interleaving will change the memory access behavior.

Let's go back to Figure 1(a) and assume that a straight-line code is cached, occupying three cache lines. As the program is executed, the three cache lines will be visited in the order of their index. If there is no idle cycle (*e.g.*, due to a branch misprediction), the cache is "active" at least during the period of accessing the cache lines. When cache lines are interleaved as in Figure 1(c), however, not all the three cache lines will be in the same bank and the active period will be split. When the high-order interleaving is used as in Figure 1(b), on the other hand, it is unlikely that accesses will be split unless the lines are on the boundaries of a cache bank. Moreover, memory references will likely be directed to a single cache bank for a long period while the other bank is idle, since small straight-line codes and heavily executed loops are often allocated to consecutive cache lines within a single cache bank.

To quantitatively analyze how cache banks are accessed, we define *active intervals* and *idle intervals*. An active interval (or A-Interval) is defined to be a period of consecutive cache/bank accesses, surrounded by idle intervals. Similarly, an idle interval (or I-Interval) is defined to be a period of cache/bank idle cycles, bordering on two A-Intervals. Within a given period of time, the number of A-Intervals and the number of I-Intervals in a single bank will differ by at most one. Figure 2 clearly shows the notions.

With multiple vertical banks in a cache, the total number of A-Intervals and I-Intervals in all the available banks can be close to that of a single-bank cache if all the activities are directed to a single bank, or can be much larger if accesses are dispersed over the banks. Given a T-cycle period, there can be up to T A-Intervals. To be more accurate, the num-

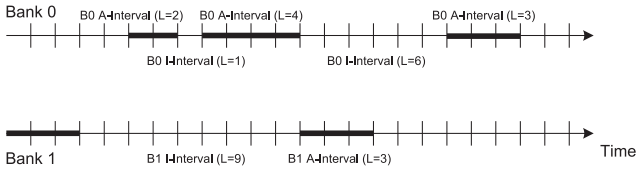

Figure 2: Active Intervals (A-Intervals) and Idle Intervals (I-Intervals). Intervals whose lengths cannot be computed are not labeled. B0 has three A-Intervals and four I-Intervals and B1 has two A-Intervals and two I-Intervals.

ber of A-Intervals can be as many as the sum of active cycles found in the architecturally equivalent single-bank cache, if each A-Interval length is just one.

Within each bank, the following holds:

$$T = n_A \cdot l_A + n_I \cdot l_I$$

where n_A is the number of A-Intervals in the given bank, l_A is the average length of the A-Intervals, n_I is the number of I-Intervals, and l_I the average length of the I-Intervals. If n_A and n_I are sufficiently large, the following holds:

$$T \simeq n_A \cdot (l_A + l_I)$$

since n_A and n_I can differ by at most one.

As T is given "fixed" in the above equation, there is a seesawing interaction between the two terms n_A and $(l_A + l_I)$, which depends critically on the interleaving method used and the program behavior such as cache misses. As we will see in Section 3, the low-order interleaving can finely split and distribute the A-Intervals, thereby increasing n_A significantly.

With N banks, the following holds:

$$L_A = \sum_{i=0}^{N-1} n_A(i) \cdot l_A(i)$$

$$L_I = \sum_{i=0}^{N-1} n_I(i) \cdot l_I(i) = N \cdot T - L_A$$

where L_A and L_I are the sum of the A-Intervals and I-Intervals in all banks, respectively, and the terms with a subscript i are related with the i-th bank. Since L_A is fixed with given architectural parameters as well as T, L_I increases with N. With finely distributed A-Intervals and increased I-Intervals, cache banks are visited more sparsely.

2.3 Applications

There are several interesting applications based on the presented vertical interleaving concept. First, one can reduce power density in cache-like structures. With sufficient I-Intervals injected into each memory bank, its power density can be greatly reduced since circuit activities are distributed both spatially and temporally. We applied the idea to instruction caches in this work and report our results using detailed simulation in Section 3. Second, one can utilize the available I-Intervals to perform reliability-enhancing actions without adversely affecting program execution. Reliability-enhancing actions, such as cache scrubbing [13] and SRAM cell flipping against NBTI [9], may incur redundant memory read and write operations, which, if not hidden effectively, can degrade the memory performance. Third, one can reduce the bank leakage current during its I-Intervals, with proper mechanisms to predict when they begin and end. If the temporal usage patterns of each bank can be accurately predicted at run time, very fine-grained and effective leakage control schemes can be developed. All these approaches build on the populated I-Intervals and the mixed distribution of A-Intervals and I-Intervals. Exploring all possible applications in detail is beyond the scope of this paper; we only study and report on the first application, reducing power density.

3. QUANTITATIVE ANALYSIS

3.1 Experimental setup

We use a derived version of sim-outorder, running Alpha binaries, in the SimpleScalar tool set (version 4) [3]. The modeled processor can fetch, issue, and commit up to four instructions at a time. It has a 4K-entry combined branch predictor, 128-entry ROB, and a 2MB L2 cache. The baseline L1 caches are 32KB, 2-way set-associative, and have 64B line size. L1 caches are pipelined and allow selective bank access as in [20]. Memory access latencies are 2 cycles, 6 cycles, and 100 cycles for the L1 cache, L2 cache, and main memory, respectively.

Eight integer and six floating-point benchmarks from the SPEC 2000 suite [19] are used with the reference input sets. Programs were compiled using Compaq Alpha C compiler (V5.9) with the `-O3` optimization flag. We run each benchmark with detailed timing, and start collecting statistics after the first 400M cycles for the period of 500M cycles. No sampling-based time-saving techniques were used in simulation since it is our prime interest to capture the continuous behavior of cache. For cache/bank power calculation and area estimation, we used eCACTI [11] with its 100nm technology parameters.

3.2 Results

3.2.1 A first look at the interleaving impact

Figure 3 (upper graph) shows the average A-Interval lengths with their standard deviation. It is clearly shown that the low-order interleaving greatly reduces the average. In all cases, they were less than five, regardless of banks. It is also shown that the differences between two banks are small, showing the efficacy of low-order interleaving as opposed to the high-order interleaving, denoted (b). The high-order interleaving can lead to a large unbalance between banks, as can be seen in *gzip* and *eon*. It is noted that the low-order group interleaving such as (c) is also putting a tight bound on the A-Interval lengths, but with a larger value than the single-line interleaving as in (d).

The lower graph in Figure 3 shows the average I-Interval lengths with their standard deviation. The single-bank configuration shows small I-Interval averages often, *e.g.*, *gzip*, *gcc*, *eon*, *mesa*, *art*, and *facerec*. This is because N, the number of banks, significantly increases the total idle cycles L_I (in Section 2) in a multi-bank cache, likely leading to a large average value unless the idle cycles are split into many I-Intervals as in (d). Again, the reduced average and variance of I-Intervals in a low-order interleaved configuration, such as (c) and (d), strongly suggest that the vertical interleaving finely distributes accesses among cache banks.

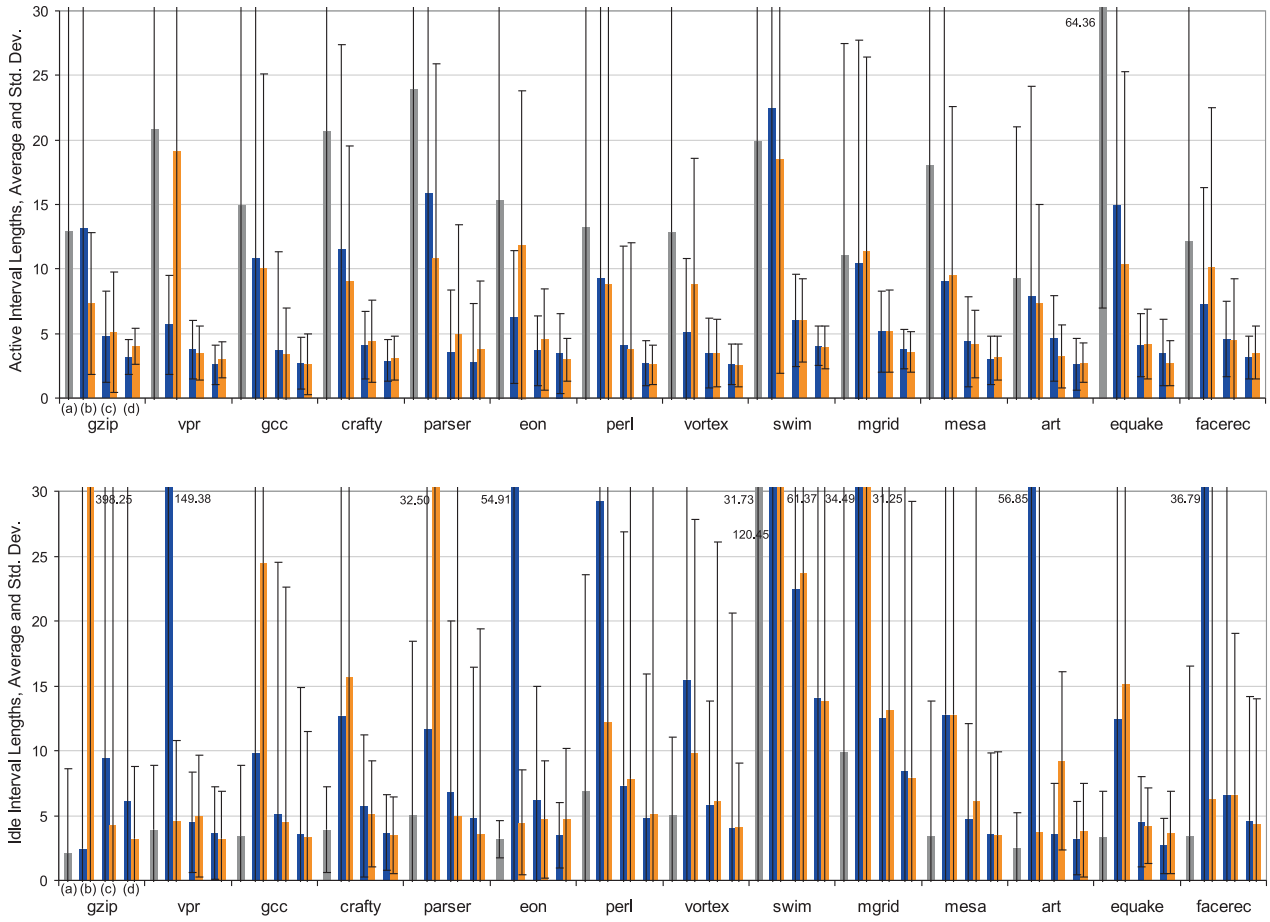

Figure 3: Active intervals (upper graph) and idle intervals (lower graph). Each bar shows the average and the standard deviation. (a) Single-bank cache; (b) Two-bank cache, high-order interleaved; (c) Two-bank cache, 2-line group low-order interleaved; (d) Two-bank cache, low-order interleaved; In two-bank cases, we showed results for each bank.

3.2.2 *Sensitivity to cache size*

Figure 4 shows how cache size affects A-Intervals. Single-bank A-Interval lengths are increased with the cache size. This is because there are less stalls due to cache misses. With multiple banks, however, A-Interval lengths become much immune to the cache size, since the low-order interleaving finely splits both long and short A-Intervals. This trend is consistently found in all the benchmark programs studied.

3.2.3 *Sensitivity to cache associativity*

Figure 5 shows how cache associativity affects A-Intervals. In the single-bank case, a higher associativity leads to longer A-Interval lengths in general. With multiple banks, however, A-Interval lengths are insensitive to the associativity. Figure 3, 4, and 5 suggest that A-Interval lengths are insensitive to cache hit rates and other program behaviors. In fact, I-Interval lengths are more indicative of the program behavior and performance. For example, programs showing a high (L1 and L2) data cache miss rate, such as *swim* and *mgrid*, tend to have longer I-Interval lengths, since during the time a long-latency data cache miss is handled, the instruction cache becomes idle soon after the pipeline buffers become full.

3.2.4 *Sensitivity to cache line size*

Figure 6 shows how different line sizes affect A-Intervals. The A-Interval of the (2, 1) configuration is more directly related with the cache hit rate. There was an increase in the average A-Interval length from 16B to 32B to 64B, then a decrease from 64B to 128B. With multiple banks, however, we observed a continuous increase from 16B to 128B. This is because with a larger cache line it is more likely that the bank that stores the line is active for a longer period of time, as a straight-line code will fit in a larger cache line which would be split into different banks otherwise. In certain cases, a small loop can be completely contained within a single large cache line, resulting in significantly longer A-Intervals.

Our results show that A-Interval lengths are relatively insensitive to cache size and cache associativity. They are affected noticeably by the degree of group interleaving and line size, however. The I-Interval length can be heavily affected by pipeline stall conditions such as a long memory latency caused by L1/L2 data cache misses.

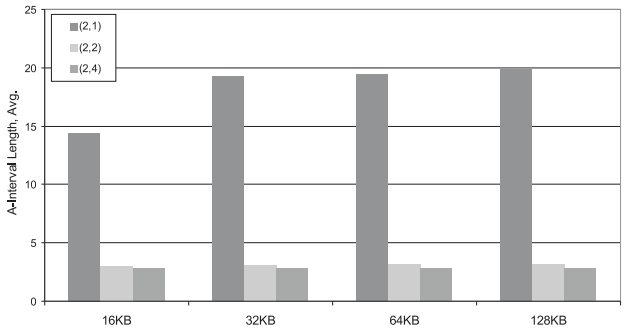

Figure 4: Cache size and A-Interval lengths. In (N, M) N is cache associativity and M is the number of (vertical) cache banks. Each bar is the average A-Interval length of all the benchmarks.

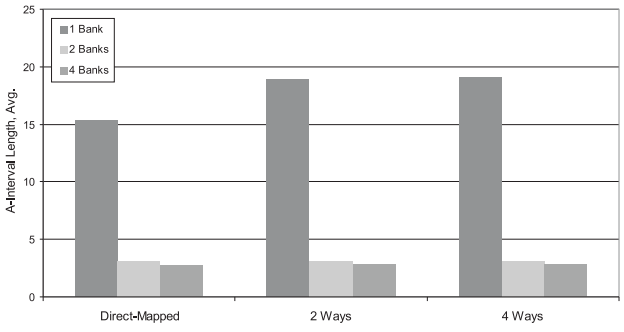

Figure 5: Cache associativity and A-Interval lengths. 32KB cache.

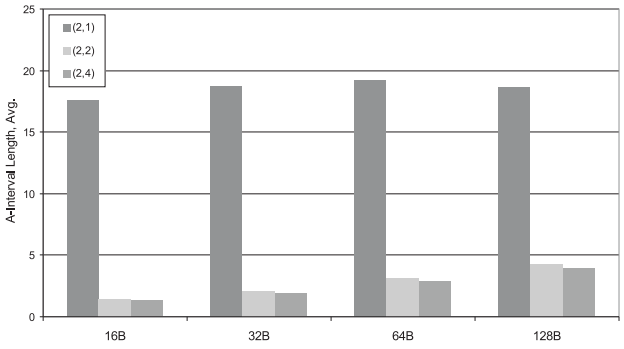

Figure 6: Cache line size and A-Interval lengths. 32KB cache.

3.2.5 *Power density reduction*

In this experiment, we study how the proposed vertical interleaving will change the power density of an instruction cache. We used a sampling period of 8,192 processor cycles to compute power density (W/cm^2) as in [16].

In the following discussions, the notation (N, M) denotes a cache configuration where the cache has N ways and M vertically interleaved cache banks per way. Relative to (2, 1), (2, 2) reduces the average power density by as much as 52% with an average of 49%. The (2, 4) configuration reduces by up to 67% with an average of 65%. Looking at the maximum power density during the whole simulation period, (2, 2) achieved a maximum 48% reduction (in the case of *vpr*) with an average of 23%, and (2, 4) resulted in a maximum 67% reduction (*vpr*) with an average of 31%. The reduction rates were consistent and robust across benchmarks, except a few programs containing very small loops, notably *perl*, *parser*, *gcc*, and *facerec*. When the cache line size is changed to 32 bytes instead of 64 bytes that we used in our experiment, fortunately, these programs show similar power density reductions. Vertically multi-banked configurations also reduce the power density consistently over time, as shown in Figure 7.

4. RELATED WORKS

Memory multi-banking and bank interleaving has been used extensively in the high-bandwidth memory subsystem of parallel and vector machines [1, 6]. To allow simultaneous memory access by a number of processors, such memory subsystems consist of multiple memory banks that can be independently addressed. Memory blocks are typically interleaved with low-order address bits over the available memory banks in order to minimize bank conflicts. More recently, multi-banking and interleaving concepts were applied to high-bandwidth, multi-ported data cache designs [14, 18, 22]. Some instruction caches for a wide-issue processor exploiting instruction-level parallelism employ the same interleaving technique also [5]. All these works use *horizontal* banking and interleaving to expose opportunities to access multiple memory blocks in parallel. This paper proposed and studied *vertical* banking and interleaving, which can be combined and used together with existing horizontal techniques. The primary goal of the vertical interleaving is to localize and distribute circuit activities to a small portion of memory, whereas the goal of horizontal interleaving is to facilitate parallel access to multiple banks.

In addition to implementing multiple ports for higher bandwidth, cache multi-banking has been used to reduce access time, to reduce energy consumption, or to facilitate aspect ratio fitting [20, 21]. Our work in this paper suggests that vertical interleaving has potential of leading to lower power density. The power density issue in a high-performance processor design has been widely recognized [15] and techniques to control power density and the resulting temperature are being actively sought [2, 16]. Ku *et al.* [8], for example, selectively puts memory rows in low power modes to reduce power density or statically rearranges cache blocks in a cache way to scatter thermal hot spots. John *et al.* [7] studied two subarray placement techniques to improve the thermal behavior of a data cache. While they focused on spatially separating subarrays that are accessed simultaneously, we focused in this work on analyzing how accesses are distributed temporally to different banks, providing more insights into cache subarray access behaviors. We believe that our work is complementary to these previous works and vertical interleaving can be combined with such techniques.

5. CONCLUSIONS

This paper studied the vertical cache line interleaving technique. There are three contributions this paper makes. First, we presented the vertical cache banking and inter-

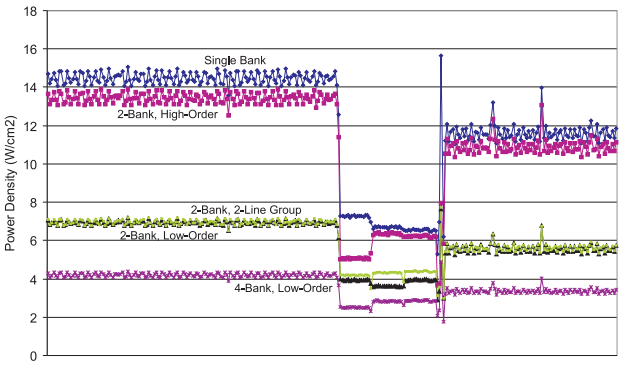

Figure 7: Power density measurement. Each point is a summary of 8,192 processor cycles. A snapshot of 300 points from the mgrid benchmark.

leaving idea. Unlike previous banking strategies to expose opportunities for parallel memory accesses, we proposed to use banking and interleaving to selectively access a small segment of memory. Second, we quantified the impact of the proposed vertical interleaving scheme in the context of instruction caches. Our results show that the proposed scheme finely distributes memory accesses over multiple banks and effectively limits the number of consecutive accesses to a single bank. Third, we discussed the possible applications of the proposed vertical interleaving idea. As a concrete example of such applications, we further showed that the concept can lead to an iso-performance, thermally efficient instruction cache design.

The ideas presented in this paper are certainly not limited to the instruction cache design, and can be extended to other on-chip structures, such as data cache, translation look-aside buffer (TLB), register file, and various branch prediction tables. It will be naturally our future work to study these on-chip structures in terms of their temporal usage. Furthermore, we note that the greatly decreased variability of bank access patterns has potential of leading to highly accurate temporal usage prediction mechanisms. We plan to study such mechanisms and their applications to improve important design qualities like as reliability and leakage.

6. REFERENCES

[1] F. A. Briggs and E. S. Davidson. "Organization of Semiconductor Memories for Parallel-Pipelined Processors," *IEEE Trans. Computers*, C(26):152–169, Feb. 1977.

[2] D. Brooks and M. Martonosi. "Dynamic Thermal Management for High-Performance Microprocessors," *Proc. Int'l Symp. High-Performance Computer Architecture*, pp. 171–182, Jan. 2001.

[3] D. Burger and T. M. Austin. "The SimpleScalar Tool Set, Version 2.0," *Computer Sciences Dept.*, TR 1342, Univ. of Wisconsin, June 1997.

[4] S. Cho, P.-C. Yew, and G. Lee. "Decoupling Local Variable Accesses in a Wide-Issue Superscalar Processor," *Proc. Int'l Symp. Computer Architecture*, pp. 100–110, May 1999.

[5] G. F. Grohoski. "Machine organization of the IBM RISC System/6000 processor," *IBM J. R & D*, 34(1):37–58, Jan. 1990.

[6] K. Hwang. *Advanced Computer Architecture: Parallelism, Scalability, Programmability*, McGraw-Hill, 1993.

[7] J. K. John, J. S. Hu, and S. G. Ziavras. "Optimizing the Thermal Behavior of Subarrayed Data Caches," *Proc. Int'l Conf. Computer Design*, pp. 625–630, Oct. 2005.

[8] J. C. Ku, S. Ozdemir, G. Memik, and Y. Ismail. "Thermal Management of On-Chip Caches Through Power Density Minimization," *Proc. Int'l Symp. Microarch.*, pp. 283–293, Dec. 2005.

[9] S. Kumar, C. Kim, and S. Sapatnekar. "Impact of NBTI on SRAM Read Stability and Design for Reliability," *Proc. Int'l Symp. Quality Electronics Design*, pp. 210–218, Mar. 2006.

[10] M. H. Lipasti, C. B. Wilkerson, and J. P. Shen. "Value Locality and Load Value Prediction," *Proc. Int'l Conf. Architectural Support for Prog. Languages and Operating Systems*, pp. 138–147, Oct. 1996.

[11] M. Mamidipaka and N. Dutt. "eCACTI: An Enhanced Power Estimation Model for On-chip Caches," *CECS TR 04-28*, UC Irvine, Sep. 2004.

[12] A. Moshovos and G. S. Sohi. "Streamlining Inter-Operation Memory Communication via Data Dependence Prediction," *Proc. Int'l Symp. Microarchitecture*, pp. 235–245, Dec. 1997.

[13] S. S. Mukherjee, J. Emer, T. Fossum, and S. K. Reinhardt. "Cache Scrubbing in Microprocessors: Myth or Necessity?" *Proc. Pacific Rim Int'l Symp. Dependable Computing*, pp. 37–42, Mar. 2003.

[14] J. A. Rivers, G. S. Tyson, E. S. Davidson, and T. M. Austin. "On High-Bandwidth Data Cache Design for Multi-Issue Processors," *Proc. Int'l Symp. Microarchitecture*, pp. 46–56, Dec. 1997.

[15] R. Ronen *et al.* "Coming Challenges in Microarchitecture and Architecture," *Proc. IEEE*, 89(3):325–340, Mar. 2001.

[16] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan, and D. Tarjan. "Temperature-Aware Microarchitecture," *Proc. Int'l Symp. Computer Architecture*, pp. 2–13, June 2003.

[17] A. J. Smith. "Cache Memories," *ACM Computing Surveys*, 14(3):473–530, Sep. 1982.

[18] G. S. Sohi and M. Franklin. "High-Bandwidth Data Memory Systems for Superscalar Processors," *Proc. Int'l Conf. Architectural Support for Programming Language and Operating Systems*, pp. 53–62, Apr. 1991.

[19] Standard Performance Evaluation Corporation. `http://www.specbench.org`.

[20] C.-L. Su and A. M. Despain. "Cache Designs for Energy Efficiency," *Proc. Hawaii Int'l Conf. System Sciences*, pp. 306–315, Jan. 1995.

[21] T. Wada, S. Rajan, and S. A. Przybylski. "An Analytical Access Time Model for On-Chip Cache Memories," *IEEE J. Solid-State Circuits*, 27(8):1147–1156, Aug. 1992.

[22] K. C. Yeager. "The MIPS R10000 Superscalar Microprocessor," *IEEE Micro*, 16(2):28–40, Apr. 1996.