VL-CDRAM: Variable Line Sized Cached DRAMs

Ananth Hegde, N. Vijaykrishnan, Mahmut Kandemir and Mary Jane Irwin
Micro Systems Design Lab
The Pennsylvania State University, University Park
{hegdeank,vijay,kandemir,mji}@cse.psu.edu

ABSTRACT

Many of the current memory architectures embed a SRAM cache within the DRAM memory. These architectures exploit a wide internal data bus to transfer an entire DRAM row to the on-memory cache. However, applications exhibit a varying spatial locality across the different DRAM rows that are accessed and buffering the entire row may be wasteful. In order to adapt to the changing spatial locality, we propose a Variable Line size Cached DRAM (VL-CDRAM) that can buffer portions of an accessed DRAM row. Our evaluation shows that the proposed approach is effective in not only reducing the energy consumption but also in improving the performance across various memory configurations.

Categories and Subject Descriptors: B.3.1 [Memory Structures]: Dynamic Memory (DRAM)
General Terms: Measurement, Performance, Design
Keywords: Variable Line, VL-CDRAM, CDRAM, Energy

1. INTRODUCTION

The growing disparity in the performance of memory and the processor is an important concern for many data intensive embedded applications. The memory performance is influenced by both the limited bandwidth of data transfer between the processor and off-chip DRAMs [1] and the large DRAM access latency. While advances in bus technology have significantly mitigated the bandwidth problem [2], the DRAM access latency has not improved much. Many of the current memory architectures address the DRAM latency problem by embedding a SRAM cache within the memory [8]. These architectures rely on the fact that SRAM accesses are faster than DRAM accesses and also exploit the use of a wide internal data bus for data transfers from the DRAM to the on-memory cache. A key difference between on-memory caches and traditional on-processor L1 caches (that can also benefit from the faster SRAM accesses) is the width of the transfer.

There are different variants of the DRAM architectures that employ on-memory caches such as cached DRAMs [6, 7], enhanced DRAMs and VC-DRAMs. A main distinction between the different approaches is in the number of cache lines that they support. Case a in Figure 1 shows the DRAM organization where only a single cache line is used to buffer an entire row of DRAM. Subsequent memory accesses to the same row are then serviced directly by the SRAM without having to access the DRAM. However this configuration is inefficient if successive accesses are not confined to the same DRAM row and can cause thrashing of the cache line.

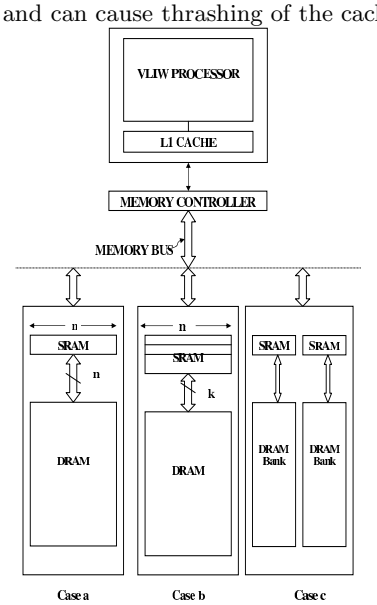

Figure 1: DRAMs with on-memory SRAM caches. (a) Single bank configuration with a single cache line SRAM, (b) Single bank configuration with multiple cache lines SRAM and (c) Multi-bank configuration with single cache line SRAM.

There are two approaches to handling this problem, one is the use of multiple banks of DRAM and associating a SRAM buffer with each bank(see Case c in Figure 1). The other approach is to use multiple cache lines with the DRAM memory (see Case b in Figure 1(b)). Prior approaches that have used multiple on-memory cache lines either use a small cache line as in traditional L1 caches [3] or use wide cache lines that are of the same size as a DRAM row [4, 5]. Further, the cache lines can either be fully-associative, direct-mapped or set-associative. Prior investigation shows that fully-associative configuration is preferable from a performance perspective [12].

The on-memory caching approaches are based on the premise that there is a significant spatial locality between the accesses in a DRAM row. Further, using fixed size cache lines assumes uniform spatial locality across different DRAM row accesses. However, applications may exhibit a varying ex-

CODES+ISSS'03, October 1–3, 2003, Newport Beach, California, USA.

tent of spatial locality across the different rows that are accessed.

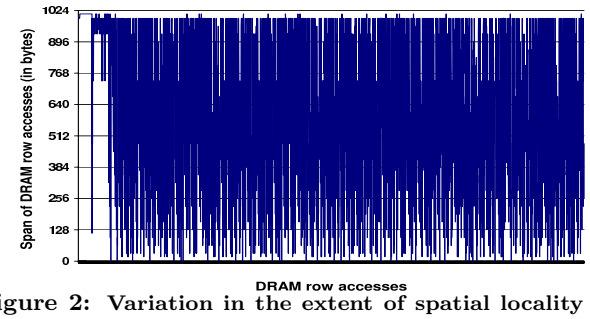

Figure 2: Variation in the extent of spatial locality in the different DRAM rows accessed during the execution of tomcatv benchmark. Here we use 1KB DRAM rows.

Figure 2 shows the variation in spatial locality in different DRAM rows that are accessed for the `tomcatv` application in our suite using a metric defined as the `span`. The `span` shown in this figure is measured by using an 8KB fully associative on-memory cache that caches the DRAM rows that are accessed. All DRAM rows are buffered in this on-memory cache and the difference between the maximum and minimum addresses accessed from this buffered row before it is evicted is defined as the `span`.

There are various reasons for the variation in spatial locality observed from this figure. Scalar variables and arrays have different spatial locality. Further, each array has a different size and this can influence the extent of locality. Another reason is that data reuse might occur in different loop levels. Let us consider the following code fragment as an example.

```
for j= 1 to 32
      for i = 1 to 1024
            a[i]
            b[1024*i]
            c[j]
      endfor
endfor
```

Here, we observe that arrays `a` and `b` have different strides in their accesses. While array `a` would benefit from caching the entire row of 1024 bytes, caching array `b` in contrast does not provide any performance benefits. In fact, since there is no spatial reuse of the cached line, caching the entire row of `b` wastes energy in the on-memory cache. Further, it can affect performance as the storage of array `b` will conflict with currently cached rows that exhibit more spatial locality. The conflicts are of a concern because of the limited number of cache lines typically available in on-memory caches. In this example, we also note that elements of array `c` have little spatial locality in the inner loop. Hence, caching an entire row of `c` can be wasteful as well. Further, we can see that only 32 elements of `c` are required and caching more elements from the DRAM row in which it is contained may not be useful. Since the amount of energy expended in the integrated DRAM ranged between 30-90% of the energy expended in the data memory hierarchy for the applications in our suite, optimizing the DRAM energy by reducing the wasted energy is significant.

In this paper, we propose a *Variable Line size Cached DRAM* (VL-CDRAM) that uses a variable line size buffering for the on-memory cache in order to adapt to the varying degrees of spatial locality. Dynamically varying cache line sizes has been used previously for L1 caches in [13] [14] to exploit varying spatial locality in applications. Our technique is different from [14], in that while [14] focuses on data-caches in an multiprocessor environment, our work focus is on the cache in the DRAM. [14] uses variable line sizes to reduce false sharing between different processors. For example, if a large line is cached in processor 1, there may be portions that it never uses actually that may required for processor 2. Thus, variable line sizing helps reduce coherence activity in this case. Due to this inherent difference, the scheme for choosing the line sizes is entirely different. Variable line caching in DRAMs have been explored before in [9] [10]. Our technique is different from them in that our approach relies on static profiling to associate the line size information with the load/store instructions while it uses dynamic information about the presence of adjacent lines to adjust the fetch/store width and associates line size information with the physical cache line (instead of the load/store instructions in our case). Due to the dynamic nature, their work incurs a learning time overhead to adapt to the appropriate cache size. Further, when the same load instruction accesses different cache lines (as in the case of array codes that access non adjacent entries), our technique is able to capture patterns of correlation in line sizes across cache lines. In contrast, the approach in [9] requires training across each cache line. While the underlying goal is the same in both the approaches, the solution for determining the line sizes are very different. Using a set of eight array-dominated applications that exhibit good spatial locality and a VLIW processor based system, we show that the VL-CDRAM can reduce the energy consumed in the DRAM by 31% as compared to using a scheme that always buffers the entire DRAM row (our default parameters explained later). We also show that our technique is effective across different memory configurations.

The rest of this paper is organized as follows. The next section describes the operation of the on-memory SRAM caches in our system. Section 3 shows the necessary architectural support for the VL-CDRAM. Section 4 explains our experimental framework and simulator. The evaluation of the VL-CDRAM with respect to energy is performed in Section 5. Section 6 provides the concluding remarks.

2. CACHED DRAM OPERATION

The target system is based on a VLIW processor that contains an on-chip cache and an off-chip cached DRAM (CDRAM). The CDRAM is an integrated memory unit consisting of an on-memory cache and a DRAM core and is accessed using a memory controller whenever there is a cache miss in the on-chip cache.

The memory controller maintains the tags corresponding to the cache lines of the on-memory cache. Whenever there is a read access to the memory unit, the tags in the memory controller are checked for a hit in the on-memory cache. If one of the tags matches, the cache line index of the matching cache line is transferred to the memory unit along with the column address. Then, the on-memory cache is accessed and the column address is used to address the appropriate portion of the indexed cache line. However, when none of the tags matches, it indicates a read miss for the on-memory cache. In addition,the index of the cache line selected for replacement is also sent to the CDRAM. The least recently used cache line is selected for replacement in our implementation. Next, the DRAM row is activated and the data is

transfered to the on-memory cache. Then, the column address is used to find the appropriate offset of data from the replaced cache line to read. Note that a write hit is similar to a read hit except the additional operation of setting the dirty bit corresponding to the written cache line in the memory controller. This dirty bit is used to identify whether a write back is required when replacing a cache line or to perform write backs of the cache lines whenever DRAM core is idle. Figure 3 shows the variation in memory access latencies when a memory request hits in the on-memory cache and misses in the on-memory cache.

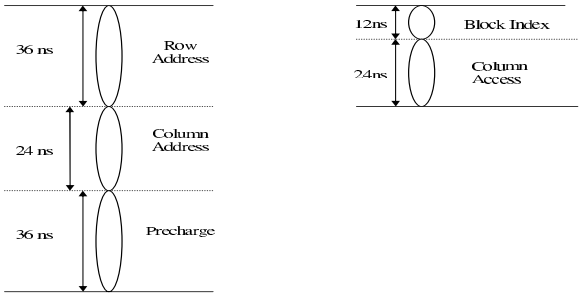

Figure 3: Access Sequences for CDRAM Accesses. On-memory cache miss (left) and on-memory cache hit (right).

3. VL-CDRAM ARCHITECTURE AND ITS OPERATION

The proposed VL-CDRAM operates similar to the CDRAM except that it does not buffer a fixed portion of the DRAM row. In the VL-CDRAM the size of the cache line is smaller than the DRAM row size. Based on the extent of spatial locality in a given DRAM row access, multiple adjacent cache lines are used to buffer the selected width of the DRAM row. The replacement policy identifies the least recently used cache line and also uses the lines adjacent to it if the buffering size is larger than a cache line. We consider the LRU policy at the cache line size level and not the replacement block size level since considering it at the replacement block size level requires more tag comparisons and additional hardware. This would not only complicate the design, but would also lead to an increase in energy consmption. In order to enable transfer of multiple cache lines in the same cycle, we enforce an alignment for the variable size blocks as shown in Figure 4 to select the adjacent cache lines.

The tag matching as usual takes place in the memory controller. Cache hit works similar to that of the CDRAM except that the row address and the column address are sent to the control unit of the memory irrespective of a hit or a miss. Since the address bus is multiplexed, the row address and column address are sent in consecutive cycles

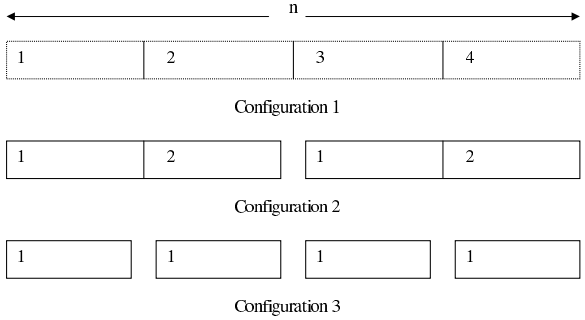

Figure 4: Valid alignments of DRAM row buffering in multiple cache lines. Here four cache lines equal the size of the DRAM row.

and get latched in the row address latch and column address latch. On a hit, the request is serviced directly by the on-memory cache. On the other hand a cache miss would signal a row address strobe which would decode the row address. The miss signal would also enable the sizing control logic which would read the `log n` input bits (Block size bits) being fed by the memory controller along with the `log n` most significant bits of the column address, where $n = size\ of\ DRAM\ row/size\ of\ cache\ line$. The column address bits are needed to decide on the offset of the block in the DRAM row. The sizing control logic is a combinational logic which generates `log n` signals which enable/disable the sense amplifiers that are divided into `log n` groups. When the row address gets decoded and a particular row gets activated, the sizing control logic would disable the appropriate sense amplifiers to support variable length DRAM row buffering. Figure 5 shows the timing diagram for a cache miss. The additional hardware to support VL-CDRAM shown in the Figure 6 was designed in verilog and synthesized using 0.13 micron technology and the energy consumed by it was calculated. We found that the energy consumed by this additional hardware is less then 0.01% of the total energy consumed by a DRAM access. Also the area impact of this additional hardware is negligible.

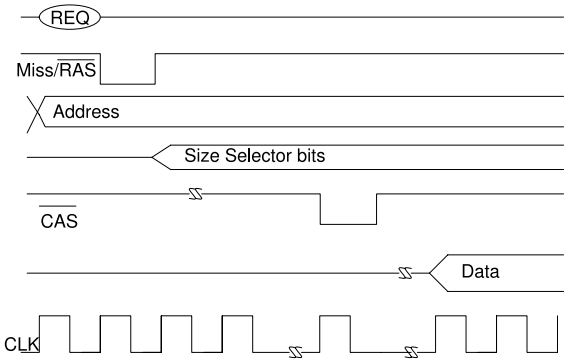

Figure 5: Timing diagram on a cache miss.

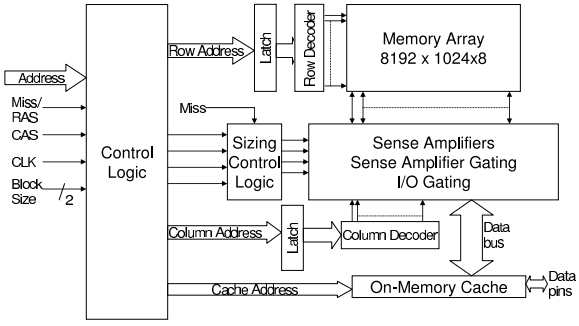

Figure 6: Modifications to the underlying hardware.

Variable block sizes impose two other constraints that need to be handled. First, if multiple cache lines chosen for replacement are dirty, the delay incurred is proportional to the number of cache lines that must be written back. However, the probability of encountering multiple dirty lines for replacement is small as write backs of the dirty blocks are performed in the background whenever the DRAM core is not busy. Second, due to the caching of multiple cache lines of variable length, we need to ensure that there is no duplication of the data in the on-memory cache. For example, consider the sequence shown in Figure 7. First, the on-memory cache buffers the entire DRAM row containing

A and B in two cache lines. Then, A is evicted later by a block C. The next load that accesses A observes that there is an on-memory cache miss and indicates the entire row consisting of two cache lines should be buffered. While the tag miss for A is the one that initiated an access to the DRAM row, the tag match for B is required to avoid duplicate buffering. Thus, in the variable line size buffering, we perform the tag check for each of the multiple cache lines that are buffered. After buffering, the valid bits of only the cache lines whose tag does not match with that of an existing cache line are marked as valid. Instead of increasing the number of tag ports to simultaneously check the tags of multiple cache lines, we perform this operation of marking tag bits valid sequentially for the adjacent cache lines in the memory controller. This operation can be overlapped with the column access in the CDRAM to mask any performance penalty. In our simulation tool we have incorporated this technique to avoid duplicates.

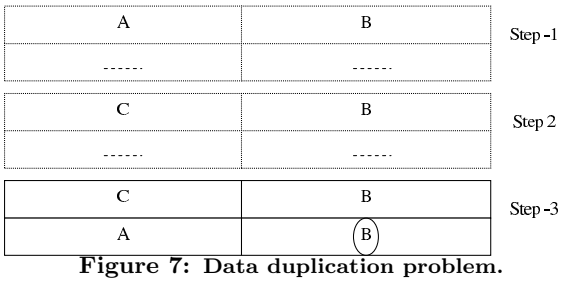

Figure 7: Data duplication problem.

Next, it is necessary to generate the log n bits which provide information on the extent of the DRAM row to buffer. Our approach is based on *profiling* which is widely used for optimizing embedded applications. We profiled the applications to identify the extent of spatial locality of load/stores. All the load/stores that incur an on-memory cache miss were tracked as they initiate the DRAM row buffering. To associate a single value with the static load/store instructions in the program , we average the extents observed at all dynamic instances of that static instruction. For example, Figure 8 shows average values and standard deviations of the span of buffered rows accessed by dynamic instances for a subset of static load/store instructions for the mxm application in our suite. We make use of log n bits to associate this value with each static instruction. This associated information indicates the number of cache lines that need to buffered when the DRAM row access was initiated by that static instruction. The additional log n bits with load/store instruction is accommodated using the unused bits in instruction format of the VLIW instruction set architecture ([11] provides more details of such opportunities in the load/store instruction format in the case of IA64). We call this technique, the Average Block Size(ABS) technique.

4. SIMULATION FRAMEWORK

In order to evaluate the VL-CDRAM approach, we used a set of applications executing on a VLIW processor architecture to generate the memory access behavior. The VLIW architecture was simulated using the Trimaran framework [15] while the VL-CDRAM simulator was custom designed. Only data cache trace was considered for the simulations. The default parameters used in our simulations are shown in Table 1. Table 2 shows the applications used in our evaluation and the last column shows the energy consumed in the CDRAM when using the base configuration with a fixed on-memory cache line size of 1024 bytes.

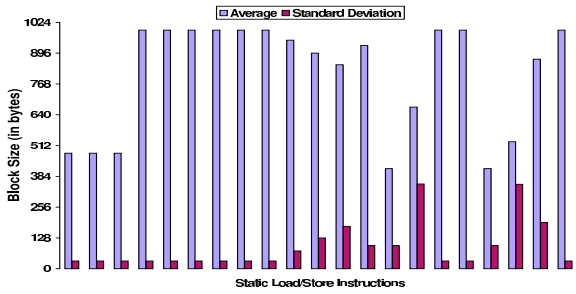

Figure 8: Average span for the blocks for each static load/store instance and their standard deviation for the mxm benchmark

Parameter	Value
Technology	0.13micron
VLIW Issue Width	9
Number of LD/ST units	2
L1 data cache size	32KB
L1 data cache associativity	2-way associative
L1 data cache line size	32 bytes
CDRAM Internal Bus Width	1024 bytes
On-Memory Cache Size	4KB
On-Memory Cache Line Size	256 bytes
DRAM core size	16MB
DRAM Banks	1
CPU Cycle Time	2ns
On-Memory Cache Hit Latency	12 cycles
DRAM Row Access Latency	18 cycles
DRAM Column Access Latency	12 cycles
DRAM Row Activation Energy	17 nJ
DRAM Read/Write Energy per byte	2.1 nJ
DRAM internal bus energy per byte	0.48 nJ
No. of address bits used by the size selection logic	2
No. of tag bits generated by the compiler with each load/store	2

Table 1: Default simulation parameters

We primarily focus on the reduction in the memory energy consumption because of the use of VL-CDRAM. Energy consumption in the VL-CDRAM is modeled as follows. A hit in the on-memory cache incurs energy consumed in accessing the associative cache. This energy was modeled for a 0.13 micron technology using CACTI [16]. A miss on the other hand requires a DRAM access. The energy consumed on a miss involves the DRAM access energy in addition to the on-memory cache access energy. A DRAM access activates the selected row and transfers that row's data to the sense amps. The energy consumed by this row activation is represented as $E(ACT)$. This is followed by a read (write) energy during which the column address selects certain number of cells for amplifying and latching into (from) the output buffers. Finally, energy is consumed in the internal buses based on the length of the DRAM row that is buffered and driven. Thus the DRAM access involves, a row activation energy ($E(ACT)$), read/write energy $E(RW)$, and the bus energy $E(DQ)$. These energy numbers were evaluated using average activation current (IDD0), active standby current (IDD3N), read/write current (IDD4(R/W)), the row activation cycles (Trc), supply voltage (VDD), output current (Iout) and the DRAM cycle time obtained from data sheets [17]. These energies are calculated as follows

E(ACT)=(IDD0-IDD3N)*Trc*VDD*clock_period
E(RW)=(IDD4(R/W)-IDD3N)*VDD*clock_period*buffering_size
E(DQ)=VDD*Iout*clock_period*8*buffering_size

Note that using the optimal sizes for the buffering the DRAM row will reduce the read/write energy and the bus energy as only the portions that need to be buffered in the on-memory cache are read and transferred. Further, we also

Program	Source	Memory energy (nJ)
tsf	Perfect Club	1257189667
eflux	Perfect Club	1028302543
btrix	Specfp92	14746347130
tomcatv	Specfp95	1551753573
mxm	Specfp92	3051618425
bmcm	Perfect Club	365088561
vpenta	Specfp92	65165438853
adi	Livermore	107773748

Table 2: Benchmark characteristics.

save energy in the on-memory cache because the amount of data written into it is smaller. Thus, reading only the required data for buffering saves energy even if there is no improvement in the hit rate to the on-memory caches.

We already discussed the latency reduction that can be achieved when a read or write hits in the on-memory cache. Hence, any improvement in hit rates to the on-memory caches using the VL-CDRAM approach can reduce the memory access latency. Further, hits in the on-memory cache limits energy consumption to only the on-memory cache and no energy is consumed in the DRAM core. Finally, performance improvements due to reduced memory access latencies can also reduce energy consumed due to refresh and also in the rest of the system (these savings will improve the presented results but are not accounted for in this evaluation).

5. RESULTS

We used two techniques to simulate the VL-CDRAM.

1. Oracle Technique : This technique would know the exact size of the block that needs to be buffered for each dynamic instance of load/store instruction. We profiled the applications to identify the extent of spatial locality of loads/stores. All the load/store instructions that incur an on-memory cache miss were tracked as they initiate the DRAM row buffering. We measured the extent of the buffered DRAM row that was accessed before it was replaced from the on-memory cache for each load/store instructions.
2. Average Block Size (ABS) technique : In contrast to the Oracle technique, we associate the buffering size with each static load/store instruction in the ABS technique as discussed above in Section 4.

We show the results for the ABS technique while comparing it with the Oracle technique. While the Oracle technique would tell us the maximum savings that could be achieved, the ABS technique shows how well this could be practically achieved.

Figure 9 shows the variation of the spatial locality in the different applications. These results were obtained by monitoring the access behavior of the buffered DRAM row that was accessed before it was replaced using an 8K on-memory cache and other default parameters. We find buffering the entire DRAM row would have been unnecessary in at least 30% of the cases across all applications. We also find that using a finer granularity of buffering can potentially save a significant amount of energy expended in reading and buffering unused portion of the DRAM row.

Next, we investigate how well this characteristic translates into energy savings. Figure 10 shows the amount of energy savings that can be obtained when different sizes are chosen for the on-memory cache lines as compared to the currently used scheme of buffering the entire row. For example, an 128 byte cache line can support buffering of either 128, 256, 512 or 1024 bytes of a DRAM row that is accessed. An average energy savings when using 128, 256 and 512 bytes

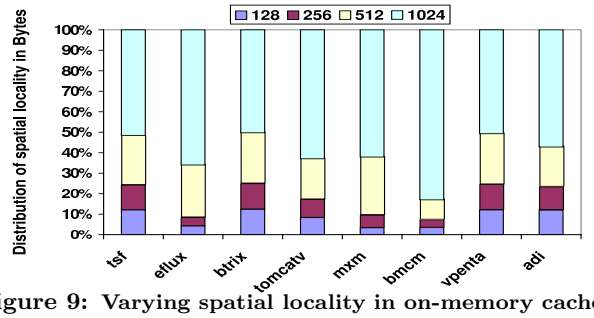

Figure 9: Varying spatial locality in on-memory cache.

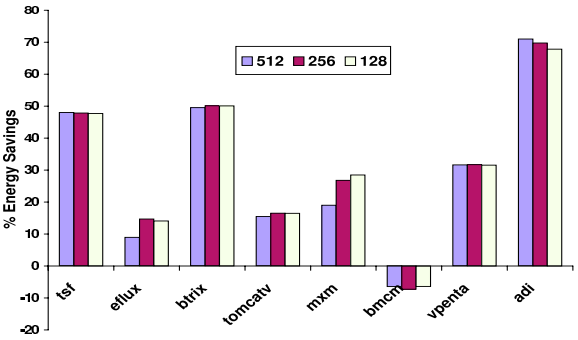

Figure 10: Percentage of DRAM energy reduction using VL-CDRAM with base configuration and varying cache line sizes over using fixed cache line size of 1024 bytes.

for cache line size provided on the average 31%, 31% and 30% energy savings for the ABS technique when compared to 38%, 36% and 29% of energy savings for the Oracle technique. While one might expect the energy savings to be higher when the block sizes are 128 and 256 when compared to using a block size of 512, we see a much more uniform behavior across different block sizes when using ABS from Figure 10. This is due to the fact that different instances of a static load/store instruction might require different block sizes and when we average the extents, we end up using one average block size value for each instruction. We also see that the energy consumption increases for one of the benchmarks `bmcm`, this occurs due to a very high irregularity in the block sizes for different dynamic instances of a static load/store instruction.

Next, we analyzed the sensitivity to the capacity of the on-memory cache sizes. While larger cache sizes can potentially capture more number of distinct cache lines, the complexity and area overheads need to be limited. Specifically, the complexity is of concern in the on-memory cache architecture as it is fully associative and the area overhead is an issue because we are embedding the SRAM within the DRAM which is preferred for its compactness. With these limitations in consideration, we varied the on-memory cache size with 2KB, 4KB and 8KB and analyzed the energy savings of the VL-CDRAM approach using a 256 bytes cache line. Figure 11 shows the results for the different cache sizes. We observe that the energy savings for the 2KB, 4KB and 8KB caches were 32%, 31% and 30% on the average respectively, across all the benchmarks using the ABS technique when compared to 36%, 36% and 33% of maximum energy savings that we could have achieved. This shows that our technique is quite effective across different configurations. We also observe that the energy savings achieved increases for smaller sized on-chip caches as there is a lesser place for contentions for replacements.

We also tried to study the impact of these energy opti-

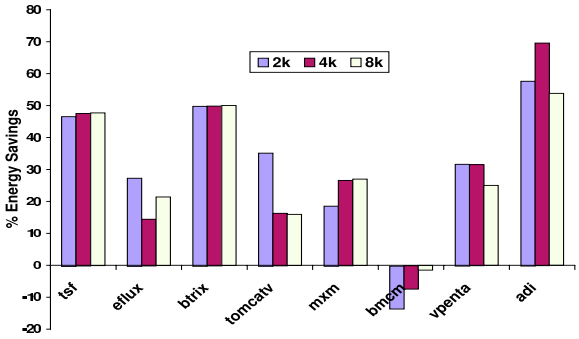

Figure 11: Percentage of DRAM energy reduction using VL-CDRAM with 256 bytes on-memory cache lines over fixed 1024 bytes on-memory cache lines for different on-memory cache sizes.

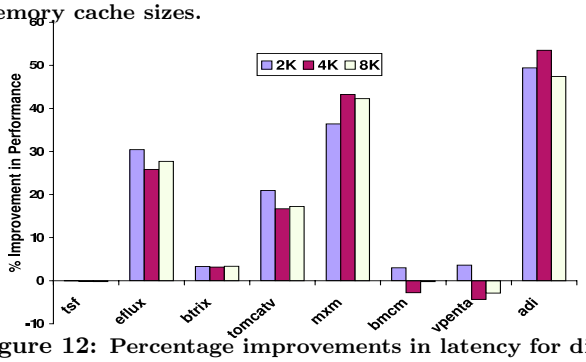

Figure 12: Percentage improvements in latency for different sizes of on-memory cache with cache line size of 256.

mizations over the latency. We observed that if the size of the cache is small, the latency will almost always improve. Figure 12 shows the percentage improvements in latency for different sizes of on-memory cache. We observed a percentage improvement in latency by 23%, 18% and 17% for 2KB, 4KB and 8KB using the ABS technique as compared to maximum achievable values of 21%, 15% and 13%. We see that certain benchmarks do not show any improvement in latency due to increase in miss rates since we consider the LRU policy at the cache line size level and not the replacement block size level.

The L1 cache sizes in the processor can influence the memory access pattern. Thus, we experimented by varying the L1 data cache size from 8KB to 32KB. Our results show that while larger L1 cache reduces the number of memory accesses and the overall energy expended in the CDRAM, it does not significantly impact the extent of spatial reuse in the on-memory cache. On the contrary, the average energy savings increase. The average energy savings when employing the VL-CDRAM with cache line size of 256 bytes and default parameters were 23%, 27% and 31% for L1 cache sizes of 8KB, 16KB and 32KB, respectively using the ABS technique, as compared to 33%, 34% and 36% for the Oracle technique.

6. CONCLUSIONS AND FUTURE WORK

On-memory caches are being employed in most commercial DRAM chips in order to reduce the effective memory access latency. However, most of the current approaches buffer a fixed size of an accessed DRAM row into the on-memory cache and waste a significant portion of the energy due to this inflexibility. In this work, we presented a VL-CDRAM that provides adaptivity to the size of DRAM row that can be buffered. Our analysis using various programs showed that there is a noticeable variation in spatial locality within different DRAM rows that are accessed. Specifically, we find that for at least 30% of the accessed DRAM rows, in all the eight applications that we used, caching the entire row wastes energy. Our approach focuses on eliminating this wastage by capturing the variability in the DRAM row spatial locality and associating this information with the load/store instructions in the application. This information is used to direct the amount of buffering to be used when the DRAM rows are accessed. We observe that our technique is effective in reducing energy across different configurations of the on-memory chip and the L1 data cache in the processor. Using 256 bytes cache line for the VL-CDRAM and other default parameters, on the average, 31% of the memory energy was saved as compared to using a fixed cache line size.

Our future effort will focus more on the impact of specific code optimizations on the effectiveness of our approach. Specifically, we will try to design better techniques which could achieve results much closer to the Oracle technique. We will also investigate techniques that transform the data layout to improve the locality in DRAM row accesses. Finally, our on-going work aims at analyzing in more detail the impact of the different bank interleaving strategies on the spatial locality of the DRAM row accesses.

7. ACKNOWLEDGEMENTS

This work was supported in part by NSF Grant 0103583, NSF CAREER Awards 0093082 & 0093085 and MARCO 98-DF-600 GSRC grant.

8. REFERENCES

[1] D. Burger, J. R. Goodman, and A. Kagi. Memory bandwidth limitations of future microprocessors. In *Proc. International Symposium on Computer Architecture*, 1996.

[2] V. Cuppu et al. A Performance comparison of contemporary DRAM architectures. In *Proc. International Symposium on Computer Architecture*, 1999.

[3] W. Hsu and J. E smith. Performance of cached DRAM organizations in vector supercomputers. In *Proc. International Symposium on Computer Architecture*, 1993.

[4] R. P. Koganti and G. Kedem. WCDRAM : a fully associative integrated cached-DRAM with wide cache lines. Technical Report CS-1997-03, Dept. of Computer Science, Duke Univ, Durham, N. C., 1997

[5] W. Wong and J. L. Baer. DRAM on-chip caching. Technical Report UW CSE 97-03-04, Dept. of Computer Science and Engineering, Univ. of Washington, 1997.

[6] H. Hidaka et al. The cache DRAM architecture with on-chip cache memory. *IEEE MICRO*, vol. 10, no. 2, Mar. 1990.

[7] C. A Hart. CDRAM in a unified memory architecture. In *Proc. International Computer conference (COMPCON 94)*, 1994.

[8] F. Jones. A new era of fast dynamic RAMs. *IEEE Spectrum*, Oct. 1992.

[9] K.Inoue, K. Kai, and K. Murakami. Dynamically Variable Line-Size Cache Architecture for Merged DRAM/Logic LSIs. *IEICE Transactions on Information and Systems*, May 2000.

[10] K.Inoue, K. Kai, and K. Murakami. Performance/Energy Efficiency of Variable Line-Size Caches on Intelligent Memory Systems A new era of fast dynamic RAMs. *Proc.2nd Workshop on Intelligent Memory SystemsIEEE Spectrum*, Nov. 2000.

[11] IA-64 Application developer's architecture guide, Intel corporation, 1999.

[12] Z. Zhang, Z. Zhu, and X. Zhang. Cached DRAM for ILP processor memory access latency reduction. *IEEE MICRO*, vol. 21, no. 4, July/Aug.. 2001.

[13] Weiyu Tang, Alexander V. Veidenbaum, Alexandru Nicolau, Rajesh Gupta. Adaptive Line Size Cache. *UC, Irvine, Technical Report ICS-TR-99-56*, Nov.1999.

[14] C.Dubnicki and T.J. LeBlanc. Adjustable Block Size Coherent Caches. In *Proceedings of the 19th International Symposium on Computer Architecture*, May 1992.

[15] Trimaran home page. http://www.trimaran.org.

[16] P. Shivkumar and N. Jouppi. CACTI 3.0: An Integrated Cache Timing, Power and Area Model. WRL Research Report, August 2001.

[17] Data Sheet for 16Mb SDRAM. http: download.micron.com/pdf/datasheets/dram/