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-
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 a 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
transferred 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.

![Diagram showing access sequences for CDRAM accesses](image)

**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 consumption. 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 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 \( n = \text{size of DRAM row/size of cache line} \). 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.

![Timing diagram on a cache miss](image)

**Figure 5:** Timing diagram on a cache miss.

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.

---

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 averages and standard deviations of the span of buffered rows accessed by dynamic instances for a subset of static load/store instructions for the mmx benchmark 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.

---

### Table 1: Default simulation parameters

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

---

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

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
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 instruction.

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

<table>
<thead>
<tr>
<th>Program</th>
<th>Source</th>
<th>Memory energy (nJ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsf</td>
<td>Perfect Club</td>
<td>1257189667</td>
</tr>
<tr>
<td>eflux</td>
<td>Perfect Club</td>
<td>1028302543</td>
</tr>
<tr>
<td>btrix</td>
<td>Specfp92</td>
<td>14746347130</td>
</tr>
<tr>
<td>tomatuv</td>
<td>Specfp95</td>
<td>365088561</td>
</tr>
<tr>
<td>nxm</td>
<td>Specfp92</td>
<td>1257189667</td>
</tr>
<tr>
<td>bmcm</td>
<td>Perfect Club</td>
<td>3051618425</td>
</tr>
<tr>
<td>77773748</td>
<td>Livermore</td>
<td>107773748</td>
</tr>
</tbody>
</table>

**Table 2: Benchmark characteristics.**

...
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 cache 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 0003082 & 0093085 and MARCO 98-DF-600 GSRC grant.

8. REFERENCES

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