Cache-In-Memory

Jason T. Zawodny[1] and Peter M. Kogge
Dept. of Computer Science & Engineering, Univ. of Notre Dame
{jzawodny, kogge}@cse.nd.edu

Abstract

The new technology of Processing-In-Memory now allows relatively large DRAM memory macros to be positioned on the same die with processing logic. Despite the high bandwidth and low latency possible with such macros, more of both is always better. Classical techniques such as caching are typically used for such performance gains, but at the cost of high power. This paper summarizes some recent work into the potential of utilizing structures within such memory macros as cache substitutes, and under what conditions power savings may result.

1.0 Introduction

Processing-In-Memory (PIM), also known as Intelligent RAM (IRAM [11]), merged logic and memory, embedded RAM, or Systems-on-a-Chip, involves integrating dense memory and logic onto the same CMOS die. On the surface, this seems like a straight-forward twist on technology, but in reality it has proved somewhat difficult to bring to fruition as a viable technology. However, now that it is available, PIM technology is having a profound effect on how we design computing systems. In fact, its major characteristics, low latency and high on-chip bandwidth, directly attack the key obstacles facing modern high performance computer designs. Projects such as PIM Fast [7], [8], HTMT [6], [1], [8], and DIVA [4] have investigated a spectrum of new architectural techniques to leverage PIM into higher performance systems.

Performance, however, is not the only characteristic of value to designers today. Perhaps a more pressing concern, especially in embedded or mobile markets, is energy consumption. Any reduction in the energy required to keep a device active enables longer usage by the user, or reduces battery size which improves the weight and size of the unit as well.

The energy savings available by simply replicating the general designs from previous multi-chip implementations to a new single chip solution has the potential to reduce energy consumption by up to 40% [3]. These replicated solutions, however, still utilize large and power hungry SRAM caches and tag arrays between the memory macros and the processing logic.

This work specifically tries to make those structures inside the DRAM macros operate as replacements or enhancements for the first level cache in embedded systems where sizes average 4 to 16 KB. We term the resulting architectures *Cache-In-Memory* (*CIM*). Section 2 summarizes such macros. Section 3 summarizes CIM architecture. Section 4 summarizes models, and Section 5 describes some early results. Section 6 then briefly discusses some continuing work.

More details on the models and initial simulation results can be found in [13] and [14].

2.0 DRAM Memory Macro Internals

Fig. 1 diagrams the layout of a typical modern memory macro. The macro contains a single base unit and multiple subarrays, which split the memory into manageable units. The base unit manages overall timing, redundancy, self-test, and interface with external processing logic. The subarrays contain the storage cells and associated access circuitry. This includes precharge logic, a row decoder, a rectangular array of DRAM bit cells, a row of primary sense amps, and column multiplexing. Typical total storage for a subarray is on the order of 0.5 to 2 Mbits, arranged in a rectangular array of several hundred rows by several thousand columns. We term a single row of such an array a *full word*, with typical values of between 1K and 4K. The memory macro's data outputs is an even fraction of a full word, called a *wide word*. Typically there are 4 to 16 wide words (of 64 to 512 bits) per full word. Storage is maintained by a capacitor in each bit cell. Two different charge levels, charged and discharged, represent a logical 1 or 0.

We use the term *aspect ratio*, the ratio of the number of bits in a full row to the number of rows, as a measure of the "rectangularness" of the subarray of cells.

In a read operation, an address as presented to a memory macro has three subfields: *subarray_select*, *row_select*, and *column_select*. The subarray_select activates a particular subarray within the macro. The row_select is decoded within that subarray and drives a unique *word line* across a subarray row. This turns on each *array transistor* in each bit cell of that row. This activation

[1] The work described here was performed as part of Mr. Zawodny's M.S. thesis at Notre Dame. He is currently at Goodrich Avionics.

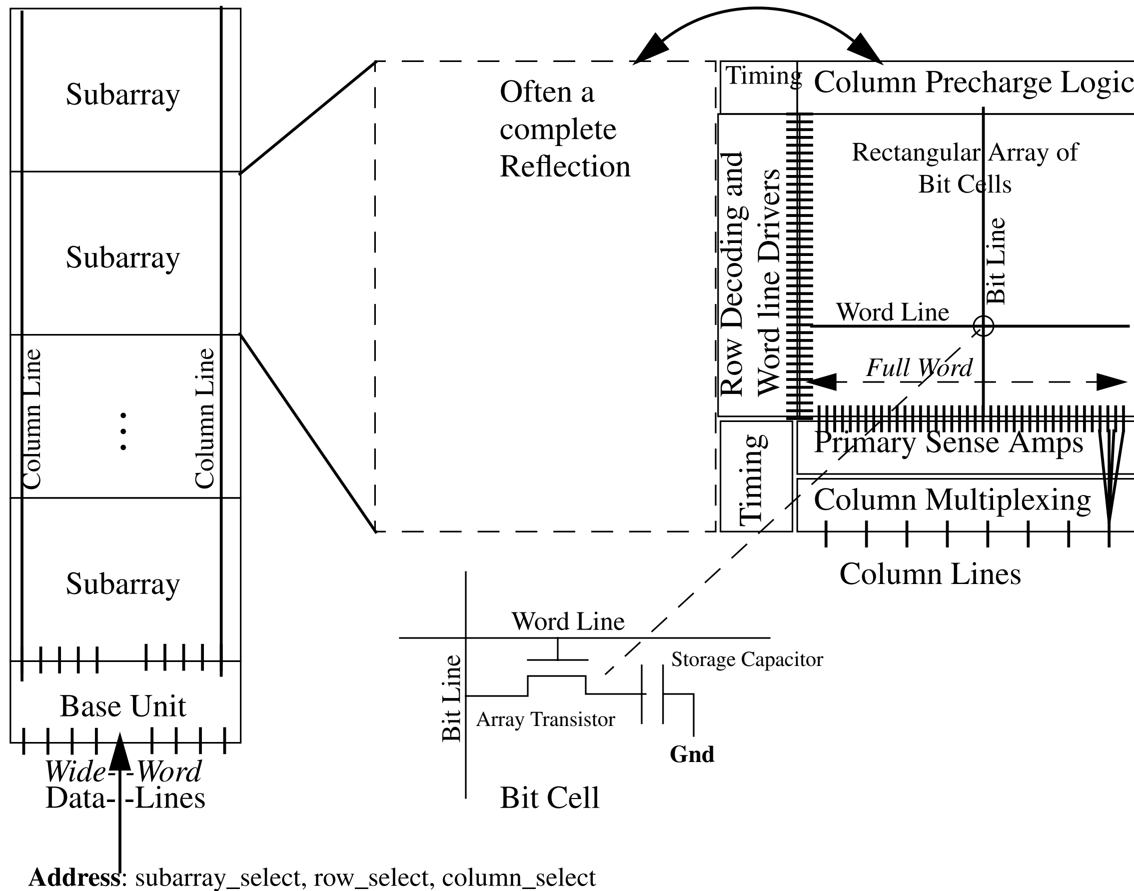


Figure 1: Basic Memory Macro Structures

couples the cell's *capacitor* to a *column bit line* which is shared by all cells in the same column of the subarray, and which has been pre-charged to a voltage approximately half way between a 0 and a 1. Combining these two charges causes a very slight voltage shift on the bit line. This is sensed by, and latched in, the *primary sense amplifiers* at the edge of the subarray. There is one sense amp for each bit line, meaning that a full row of several thousand bits is saved at each access. We call the row whose contents have been so latched the *currently open row.*

Because the charges on the row's bit cells have been changed by the read process, a *rewrite cycle* is needed to restore the information. The output of the sense amps is redirected back onto the bit lines to reset the charge on the capacitors. After this, the word line is deactivated, isolating the storage capacitors.

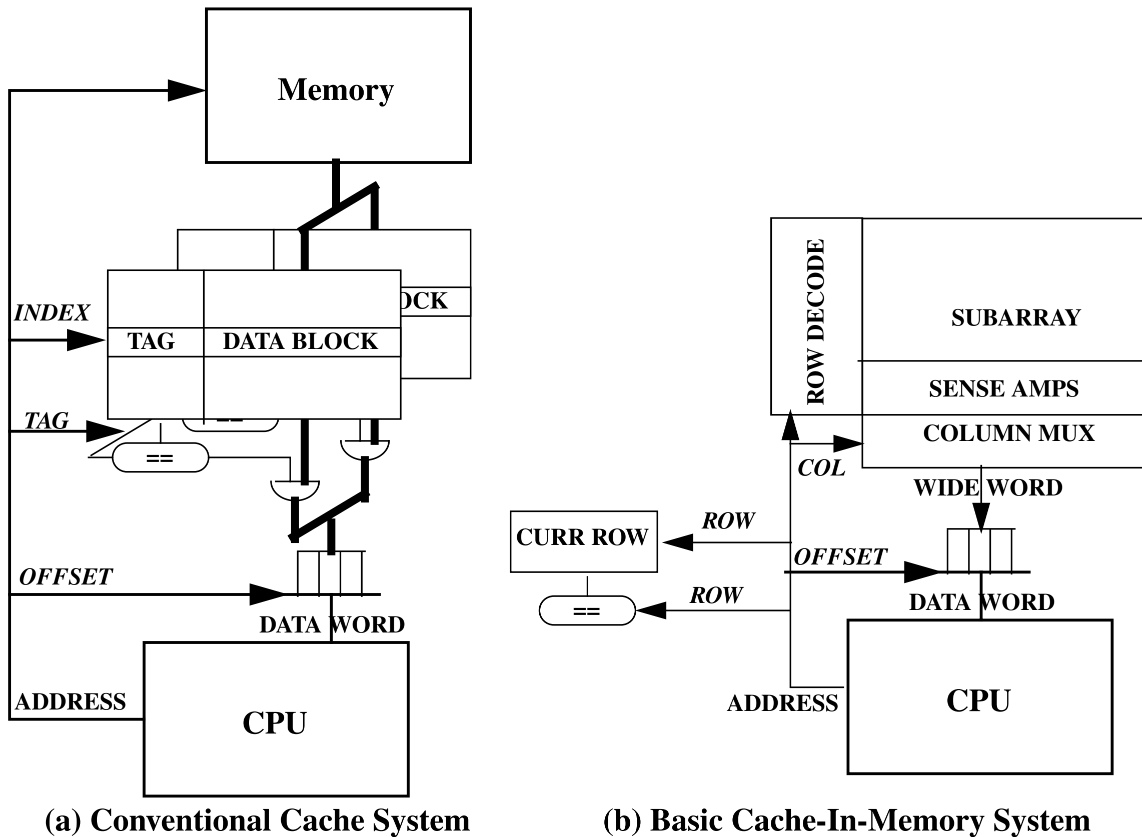
For a variety of reasons, not all of these bits can be presented to the output of the macro. Instead, the third field, the *column_select*, of the address drives a multiplexer to select a particular *wide word* out of the open row and pass it to the base unit over the *wide word column lines*. Inside the base unit a variety of logic functions may be performed, including accounting for defective columns, optionally performing parity or ECC checks, and redriving the signals to be compatible with processing logic.

A write operation starts with a read of the full word containing the bits to be changed, and latching this full word into the sense amps. Then, before the rewrite cycle, the data to be written is placed on the appropriate column lines and latched into just those positions in the sense amps.

Finally, because the storage medium is capacitive, even very slight resistive paths to ground eventually remove any charge, causing the cell to "forget." To prevent this, periodically each row in each subarray must be read, latched, and rewritten back into the memory. This is called a *refresh cycle*, and must be repeated on each row every few milliseconds.

3.0 Constructing CIM Models

A normal cache stores a copy of the data from a memory address in an SRAM array to reduce the amount of time required to retrieve it. CIM proposes that, with mini-



(a) Conventional Cache System **(b) Basic Cache-In-Memory System**

Figure 2: Conventional and Cache-In-Memory Architectures

mal additional intelligence to the on-chip DRAM, similar reductions in time can be achieved by using the latches in the sense amps. Further, by eliminating or reducing external cache structures, such gains may be achieved at lower energy expenditures.

3.1 Basic Cache/CIM Principles

Fig. 2 diagrams both a classical cache and a basic CIM system. The classical system takes an address from the CPU and divides it into three subfields: a *tag*, an *index*, and an *offset*. The index field accesses the tag and data arrays of the cache. The output of the tag array are compared to the tag field from the address. A match selects the corresponding data array output, and the offset field selects the desired word from the data block. In an N-way set associative cache there are N tag and data array pairs, all of which are accessed concurrently, with all tag outputs compared to the tag field simultaneously. On a miss, the block returned from memory is stored in the data array entry for the address's index, with the address's tag stored in the tag array.

The most basic incantation of a CIM system is shown in Fig. 2(b). This system consists of a single subarray. It also divides the address from the CPU into several fields discussed above for the basic memory macro: *row_select* and *column_select*, plus an *offset*. The *Current Row* latch contains the row number of the most recently referenced row of the subarray. The sense amps thus correspond to a single data block cache whose tag is in the Current Row latch.

Note that unlike a conventional memory macro, once an access is made, the data need not be rewritten immediately back into the memory. This can save a significant amount of energy if any updates are then made to the open row.

When an address is presented, the row_select field is compared to the Current Row latch. If there is no match, then the current access has "missed" the currently open row. When such a miss occurs, the current row written back into the subarray, the Current Row latch is then updated from the current address, and the requested row is read into the sense amps.

If the current row bits do match those from the previous access, then a "hit" has occurred. No read or write operations are performed to the actual cells in the subarray. The sense amps act as the target or source for the transaction. The column_select address field will then select the wide word out of the open row, and the offset

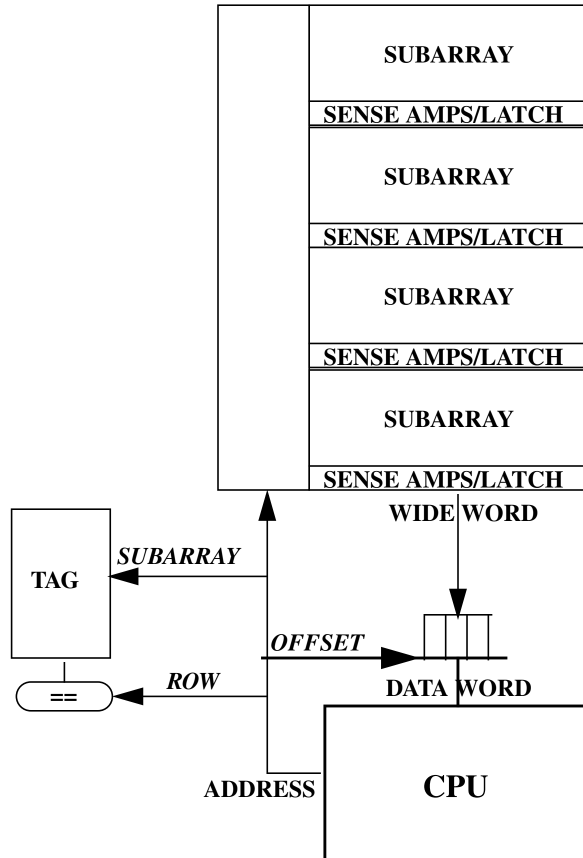


Figure 3: Expanded Cache-In-Memory Architectures

field will select the desired word out of the wide word.

An important observation about this "hit" is that there are actually two variants. The first is where the column select is to the same wide word as the last access. The second is where a different wide word is selected. The former requires no extra energy since the wide word outputs do not change. The latter is termed an *expensive hit* which requires changing the multiplexer between the sense amps, and driving the column lines to the outputs.

3.2 Extended CIM Architecture

Fig. 3 diagrams an extended version of the CIM concept. Here there are multiple subarrays (up to 16 are not atypical for modern technology), each of which has a separate row decoder and sense amp. Each of these sense amps is thus capable of saving the last full word accessed from its subarray. Total equivalent cache capacity is thus the sum of all the bits in all the sense amps.

The address from the CPU in this case is divided into the same set of fields as discussed in the basic model, plus a *subarray_select* which selects which subarray in the memory macro contains the desired data. In operation, when an address is presented, the circuitry must determine whether or not one of the sense amps have the desired data, and if so whether or not the appropriate wide word is switched to the wide word outputs. In Fig. 3, this sequence is handled by using the subarray select to access a small register file that contains one entry for each subarray, much like a conventional tag array. Comparing this output to the row subfield determines if we have a hit in the subarray. If not, then we have to perform an access cycle and update the Tag Array (with perhaps a write back if the sense amps are "dirty").

In a very real sense, what we have is equivalent to one of the very earliest of cache designs, a *sector cache*, where a *sector* (a sense amp row) holds a number of data blocks (wide words in our case) from consecutive locations in memory. A tag is needed only for the sector as a whole.

As an example, the baseline technology assumed here is one where a memory macro can hold up to 16 subarrays, each of which has 512 rows, 2,048 columns (a full row), with a wide word output of 256 bits. Thus the capacity of each subarray is 1 Mbit, the capacity of the whole macro is 16*1 Mbit = 2 MB, and the equivalent cache capacity is 16 sense amps of 2,048 bits each, for 4 KB.

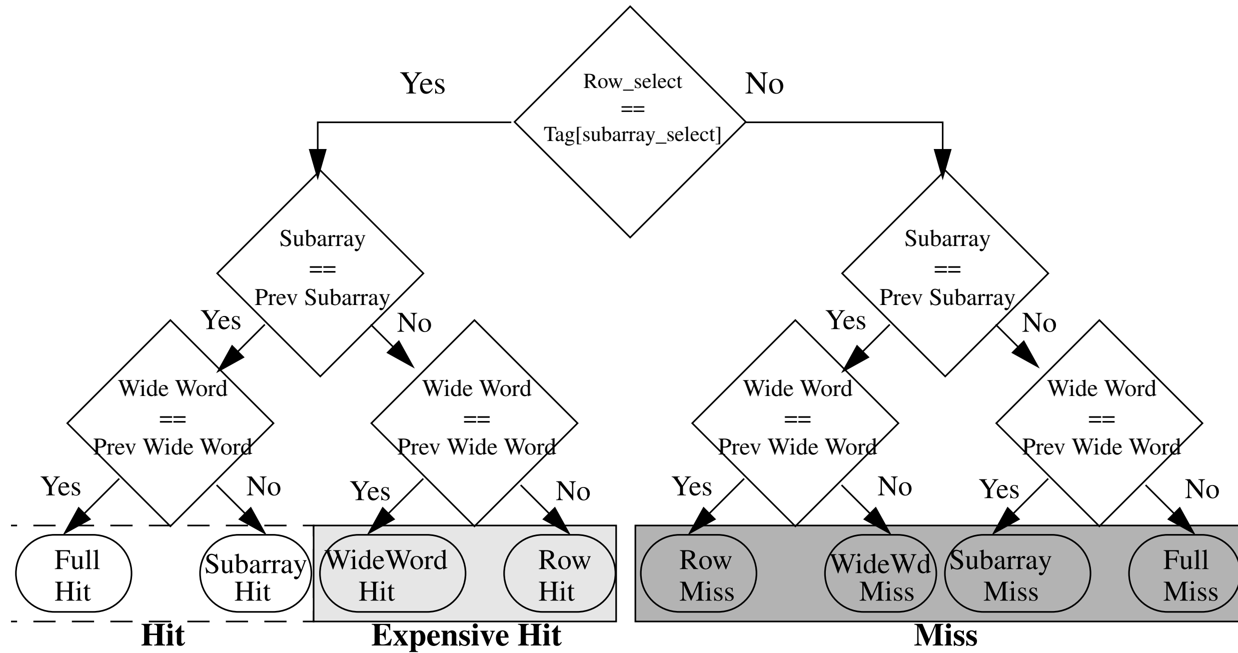


Figure 4: CIM Access Taxonomy

3.3 Hit Models

Unlike a conventional cache, this hit or miss from the tag array is not the end of the story. Fig. 4 diagrams a taxonomy of cases, each of which have a potential effect on either latency of access, or energy required, or both. The root decision point determines whether the requested memory location is currently held open in the sense amp row for the applicable subarray. This is done explicitly by the tag array check described above. If the access cannot be satisfied by the sense amp row, then a write-back and read of the proper row must be performed. These two operations are the most time and energy expensive operations in the CIM system.

The second decision deals with whether the requested subarray matches the subarray most recently accessed. Due to the energy and time costs with shutting down one set of tri-state buffers that connect the sense amp output to the wide word column bus, activating another, and driving or receiving the appropriate data, this subarray switch represents a substantial investment.

The third level of decision deals with whether the wide word requested is the same wide word as was accessed last time the appropriate subarray was accessed. In terms of energy and time investment, this will be minimal (changing the multiplexer between the sense amp outputs and the wide word bus), but is included in the taxonomy for completeness sake.

The result of these three tiers of decision are three main groups of accesses: hits, expensive hits, and misses, with variations in each.

3.4 Architectural Variations and Extensions

While the model of Fig. 3 looks straightforward, there are actually multiple variations. First, the natural assumption is that of the subfields on an address the subarray_select represents the most significant bit positions, the subarray_select the next highest, the wide word select next, and the offset the least significant bits. This, however, need not be the case. [14] discusses the variations.

Next, there is nothing that requires exactly one sense amp/latch per subarray. Additional latches can be kept below each sense amp row to save prior full words when a new one must be opened. This creates “associativity” in the subarrays.

Finally, if both instructions and data are stored in the same memory macro, and there is no other caching mechanism in the CPU accessing the memory, then the alternating sequence of instruction reads and data accesses from the instructions may very well cause an almost continuous sequence of expensive hits, even if by themselves the instruction and data streams could have had a high hit ratio. Thus, it may be advantageous to have some simple cache-like structures, at least wide word sized instruction buffers, outside the memory.

4.0 Models

Latencies of a complex system such as a CIM memory macro depend to a large extent on how much overlap

there is between different sub circuits. For example, if we were interested in minimum latency, we would always start the subarray decode at the same time as the tag array access. This work, however, is focused on minimum energy, so that we will look at models where nothing is done unless it absolutely has to, and is performed only after the chain of events that drive the decision process for its activation have completed.

Further, we assume that for many circuits, such as a multiplexer, if the control inputs do not change, and the data inputs do not change, then there is no energy dissipated. Depending on the logic family and how the control inputs in particular are clocked, this may or may not be a valid assumption. It is, however, a minimum energy assumption.

4.1 Latency Equations

To build an equation for latency, we assume the following parameters:

- $T_{tag+compare}$ = time to access the tag array and make a comparison.
- $T_{subarray_decode}$ = time to decode the subarray_select field from an address and activate that subarray.
- $T_{access_rewrite}$ = time to write a dirty open row in a sense amp latch back into the subarray.
- T_{access_read} = time to access a subarray and latch the new open row into the sense amps.
- T_{column_mux} = time to change the column mux between a sense amp and the column bus.

With this, the categories from Fig. 4 match up with latencies as follows:

$$T_{Full_hit} = T_{tag+compare}. \quad \text{(EQ 1)}$$

$$T_{subarray_hit} = T_{tag+compare} + T_{column_mux}. \quad \text{(EQ 2)}$$

$$T_{wide_word_hit} = T_{tag+compare} + T_{subarray_decode}. \quad \text{(EQ 3)}$$

$$T_{row_hit} = T_{tag+compare} + T_{subarray_decode} + T_{column_mux}. \quad \text{(EQ 4)}$$

$$T_{row_miss} = T_{tag+compare} + P_{dirty_row} * T_{access_rewrite} + T_{access_read} \quad \text{(EQ 5)}$$

$$T_{wide_word_miss} = T_{tag+compare} + P_{dirty_row} * T_{access_rewrite} + T_{access_read} + T_{column_mux}. \quad \text{(EQ 6)}$$

$$T_{subarray_miss} = T_{tag+compare} + T_{subarray_decode} + P_{dirty_row} * T_{access_rewrite} + T_{access_read}. \quad \text{(EQ 7)}$$

$$T_{full_miss} = T_{tag+compare} + T_{subarray_decode} + P_{dirty_row} * T_{access_rewrite} + T_{access_read} + T_{column_mux}. \quad \text{(EQ 8)}$$

In many real technologies, T_{column_mux} is small relative to a tag or subarray access, as is $T_{subarray_decode}$. Thus, to a first approximation, the four misses all take equivalent time, as do the wide word hit and row hit, and the full hit and subarray hit.

4.2 Energy Equations

Changing each of the “T” parameters in the above latency equations to an equivalent “E” parameter gives an estimate of the energy associated with each type of access. As with latency, this is a lower bound because in real life it is impossible to prevent some of these events from not happening concurrently with earlier events unless there is sophisticated latching or clocking. Such circuitry may be difficult to analyze.

4.3 Parameterization

Many of these parameters, both on the energy and latency side, are themselves functions of memory macro design parameters such as:

- the number of subarrays
- the number of rows in a subarray
- the number of columns in a subarray - the number of bits latched into primary sense amps
- the width of a wide word and the degree of multiplexing between a full word and a wide word

In most cases, these parameters affect a physical parameter in a linear way, which in turn affects the amount of capacitance, and thus both energy and latency, associated with the structure. In some cases, such as the number of subarrays or number of rows per subarray, the area and capacitance include terms which grow more like Nlog(N) to reflect the decoders needed.

5.0 Some Early Simulation Results

To perform some initial simulations of the potential of CIM, it is necessary to develop some real values for the design parameters which lead to those described above, develop a simulator that can estimates the probabilities of ending up in each of the access configurations of Fig. 4, and then perform a comparative analysis to look at the combined results.

Since the expected major application for CIM at the time of the study was in embedded systems, a benchmark suite was chosen consisting of a subset of 18 programs from the SPECMark 95 suite, plus a matrix multiplication routine adapted for computer architecture analysis and a JPEG image conversion program.

5.1 A Performance Model and Configurations

To compare CIM against traditional cache architectures, we need to test them head-to-head. In order to gain as much acceleration as possible from a software simulation, a C language library package called Shade [2] was utilized to run some benchmark code natively on SUN Ultrasparc (tm) workstations and develop appropriate trace runs. These traces were then processed by a set of cache simulation programs that simulated the general characteristics of Fig. 3, with a separate simulation of a system like Fig. 2(a) as a traditional reference.

Inputs to this program for the CIM consisted of the general parameters of the CIM such as:

- number of subarrays - ranging from 8 to 64
- number of rows per subarray - ranging from 64 to 4096
- number of columns per subarray - ranging from 2,048 to 16,384
- address mapping between addresses as they exit the CPU into the various subfields
- the degree of local subarray level associativity - 1, 2, 3, or 4

In all, 40 different configurations were selected, in four series. The first and second series both explored different aspect ratios for the CIM DRAM subarrays, where the total subarray capacity was fixed at 1 Mbit (corresponding to current practice). These series also included variations in local associativity and address mapping. The third series represented a departure from the 1 Mbit subarray point by increasing the number of rows per subarray, and decreasing the number of subarrays so that the total memory macro had a constant capacity of 2 MB. The fourth series was similar, but increased the number of columns.

Additionally, each of these 40 configurations was tested with both an instruction line buffer and with a separate theoretically perfect instruction cache that totally removed instruction traffic from the stream seen by the memory.

The traditional model was of a classical computer with a split I and D L1 cache of capacity approximating the number of bits of equivalent cache held within the CIM model. Nine separate configurations of 4KB, 8KB, or 16 KB, at associativities of 1, 2, or 4, were run.

Besides computing hit and miss counts, the trace processor also estimated execution time assuming a simple single issue pipelined CPU such as a PowerPC 401 was driving the memory. For purpose of this timing, we assumed that a hit (full or subarray) could be handled within the CPU machine cycle making the request (i.e. equivalent to a single cycle cache), that an expensive hit took a total of two cycles (a one cycle penalty), and a miss was 5 cycles.

5.2 Energy Model

To develop energy parameter to accompany the performance counts, a detailed SRAM energy simulator, called "Memory Optimization for Energy (MOE)", was adapted from previous work [10]. MOE is based on a combination of detailed equations that describe the behavior of the digital portions of a memory system, coupled with experimentally derived equations to model sections of the design which exhibit more analog behavior, such as the sense amps. The energy is computed for each component based on various capacitance inputs, as well as the geometry of various parts of the memory system. MOE accepts two sets of inputs: architectural parameters and device and cell-level electrical characteristics.

The technology verified for the original MOE simulations utilized a 0.8 micron process. These numbers were updated to reflect a more realistic 0.25 micron merged logic and DRAM process characteristic of those in use by IBM[5], NEC, etc for merged logic and DRAM parts. They were then combined with the 40 configurations discussed above to produce energy estimates for the DRAM array under the various operation modes.

MOE was also to produce estimates for the tag arrays and for the conventional cache arrays in the traditional configuration.

5.3 Hit/Miss Results

We found that the miss rates of the basic CIM architecture simulations ranged from 14% to less than 0.2%, and were generally below 8% for reasonably sized CIM configurations. However, the percentages of accesses that result in expensive hits is greater than 50% for all but a few of these cases. This means that for an 8% miss case, 92% of all accesses were hits, but over half of these were expensive hits. However, out of these expensive hits about 80% are two expensive hits generated by successive accesses to memory.

In comparison, miss rates for the traditional configurations were stable at about 2%. If we use the 1/2/5 cycle numbers for hit/expensive hit/miss times then we get:

$$\text{Traditional_Latency} = 5*0.02 + 1*0.98 = 1.08 \text{ cycles} \quad \text{(EQ 9)}$$

$$\text{CIM_Latency} = 5*0.08 + 2*(0.5) + 0.42 = 1.82 \text{ cycles} \quad \text{(EQ 10)}$$

When we filter out instruction fetches from the access stream to memory (i.e. assuming a separate perfect instruction cache), we see miss rates drop to between one

half and one fourth of their original values. Further, the expensive hit rates also drop to as little as one fourth their original values, to less than 12%. In terms of equivalent average latency, such configurations yield numbers in the 1.2 cycle range.

Using a simple instruction line buffer as an intermediate design point, the average rate of expensive hits decreased between 45% and 55%. While not as good as the perfect I-cache, this is still a considerable improvement for a relatively simple implementation addition.

5.4 Configuration Energy Trends

We can reduce the 40 CIM configurations outlined in this work down to 10 unique aspect ratios, arranged in three series. These series deviate from “normal” aspect ratio memory array to test the energy for cache trade-off for each.

The highest energy expenditure occurs of course in a full miss, where everything is exercised. Starting with a “normal ratio” of 512 rows by 2,048 columns, keeping a constant 64 subarrays and making the arrays shorted but fatter increased the energy considerably, up to 42% for a 64 by 16,384 subarray - while at the same time increasing the effective cache storage by a factor 0f 8 (16,384/2,048). Whether this gain in cache storage is worth the energy increase is discussed below.

We also see an increase in energy if we increase the number of rows per subarray, while keeping the columns constant and decreasing the number of subarrays. The effective cache storage decreases and the energy per access almost doubles. Similar numbers occur when we increase the column size, but decrease the number of subarrays so that cache storage is constant. The energy increase approaches 93% over the baseline.

If we look at expensive hits, increasing the number of columns while decreasing the number of rows increases the effective cache size as discussed above, and increase the energy per expensive hit. Going from 512x2,048 to 256x1,024 doubles the data storage at a cost of 63% in energy per expensive hit. This energy flattens out at about 98% for larger column sizes.

Reducing the number of subarrays and increasing the rows per subarray has the interesting effect of decreasing expensive hit energy for a while - -32% for 32 sectors of 1,024x2,048. This, however, turns around and begins to grow as the subarrays get too big.

Growing the number of columns while keeping the rows constant has almost no effect on energy per expensive hit.

The energy trends for hits follow similar patterns. Keeping the number of subarrays constant while increasing the columns (and decreasing the rows), increases the effective cache storage and energy, with a peak of 111% at 128x8,192. Keeping the columns constant and decreasing the number of subarrays decreases the energy for a while, and then it starts climbing again. Keeping the rows per subarray constant exhibited an energy increase as we increased the number of columns.

5.5 Putting it Together

To perform a complete analysis, the results of the architectural analysis were combined with the average energy costs for each type of access for the same configuration, to compute a total energy expended per benchmark suite. To perform realistic comparisons, we matched up conventional caches with CIM caches of comparable data capacity. The net effect is that the best of the CIM configurations take on a “U” shape with a minimum of energy around 32-64 KB. The conventional cache curve is also a U, but much flatter. At the minimum of the CIM curve, the CIM tends to be about 20% better in energy usage.

An additional set of analysis looked at performance (in terms of “cycles per instruction” CPI) of our simple CPU as we change these configurations. Here the larger latencies of CIM (due to the assumed 2 cycle expensive hit) impact performance for the CIM configurations. The CPI results indicated a strong preference towards CIM configurations with larger cache sizes. However, the significantly higher per-access memory requirements of the larger cache configurations made these cases less efficient in energy performance. The “winning” CIM configurations suffered a CPI penalty of 42.5% at an 8.4% energy savings.

6.0 Conclusions and Future Work

In the configurations studied to date, CIM does offer energy savings, at the cost of latency penalties. Having an Instruction Line Buffer for instructions seems to be a requirement to avoid many of the “back-to-back” expensive hits.

If we are in a conventional performance-oriented design environment, the increased latencies seems largely to offset the energy savings. However, if we are interested in minimum energy systems, or systems where in times of lower performance needs, we may be willing to back off in performance to gain even more energy savings (such as the on-going DARPA-sponsored Morph project [9]). That is in fact the direction of our future work. In particular, some of the directions we are pursuing include:

- using a complete embedded system benchmark as a reference rather than a suite of unrelated programs.

- running the simulator to get detailed counts on all 8 classes of accesses from Fig. 4.
- starting with the optimal configurations identified here, perform detailed SPICE-level timing and energy analyses of memory macro components.
- alternative implementations for the tag array, which is accessed for almost every memory access, and which dissipates energy not found in a conventional memory macro.
- integration of the TLB into the CIM tag array so that the tag array energy access do not show up as a separate penalty.
- timing the memory accesses so that for low energy situations we perform nothing in the memory until we are sure of the type of access. This may impact latency, but will gain in energy.
- look at better designs in terms of expensive hit designs.
- use CIM as a cheap L2 cache in back of a simple and small L1 cache.
- look at compiler data placement algorithms that can increase the kinds of hits that CIM is capable of.
- dual-porting the memory macro to separate the memory traffic so that many of the expensive hits are converted into lower energy hits.

7.0 Acknowledgements

This work was sponsored in part by the Jet Propulsion Lab and the California Institute of Technology as part of the JPL Remote Exploration and Exploitation (REE) program, in part by support from the Hybrid Technology Multi-Threaded (HTMT) project through DARPA and the NSA through an agreement with NASA, and in part by DARPA and Rome Laboratory, Air Force Material Command, USAF, under cooperative agreement number F30602-98-2-0180 as part of the Data IntensiVe Architecture (DIVA) Project under the Data Intensive Systems (DIS) program. Continuing work is being supported in part by DARPA and Rome Laboratory, USAF as part of the Morph project under the Power Aware Communication and Computation program (PACC) under contract FC 306020020525.

References

[1] Jay B. Brockman, Peter, M. Kogge, Vincent Freeh, and Thomas Sterling, “Microservers: A New Memory Semantics for Massively Parallel Computing,” Int. Conf. on Supercomputing, Rhodes, Greece, June 20-25, 1999, pp. 454-463.

[2] B. Cmelik. *The SHADE Simulator*, Technical Report, SUN Corp., 1993.

[3] Richard Fromm, et al, “The Energy Efficiency of IRAM Architectures,” Int. Symp.on Computer Architecture (ISCA), 1997, pp. 327-337.

[4] Mary Hall, Peter Kogge, Jeff Koller, Pedro Diniz, Jacqueline Chame, Jeff Draper, Jeff LaCoss, John Granacki, Apoorv Srivastava, William Athas, Jay Brockman, Vincent Freeh, Joonseok Park, Jaewook Shin, “Mapping Irregular Applications to DIVA, A PIM-based Data-Intensive Architecture,” Supercomputing, Portland, OR, Nov. 1999.

[5] IBM Microelectronics, “IBM chip advances spur “system-on-a-chip” products,” www.chips.ibm.com/news/1999/sa27e/, Feb. 22, 1999.

[6] Peter M. Kogge, S. C. Bass, J. B. Brockman, D. Z. Chen, E, H. Sha, “Pursuing a Petaflop: Point designs for 100TF Computers Using PIM Technologies,” 6th Symp. on Frontiers of Massively Parallel Computation, Annapolis, MD, Oct. 25-31, 1996

[7] Peter M. Kogge, Jay B. Brockman, Thomas Sterling, and Guang Gao, “Processing-In-Memory: Chips to Petaflops,” IRAM Workshop, Int. Symp. on Computer Arch., Denver, CO, June 1, 1997.

[8] Peter M. Kogge, Jay B. Brockman, Vincent Freeh “Processing-In-Memory Based Systems: Performance Evaluation Considerations”, Workshop on Performance Analysis and its Impact on Design, held in conjunction with Int. Symp.on Computer Architecture (ISCA) ‘98, June 27-28, 1998.

[9] Peter M. Kogge, Vincent W. Freeh, Kanad Ghose, Nikzad Toomarian, Nazeeh Aranki, “Morph: Adding an Energy Gear to a High Performance Microarchitecture for Embedded Applications,” Kool Chips Workshop, MICRO-33, Monterey, CA, Dec. 10, 2000

[10] Kartik Nanda, *Analysis and Optimization of PIM Memory Organization*, M. S. Thesis, Dept. of Computer Science and Engineering, Univ. of Notre Dame, Notre Dame, IN, July 1998.

[11] D. Patterson et al., “A Case for Intelligent DRAM: IRAM,” IEEE Micro, April 1997.

[12] Semiconductor Industry Association, *International Technology Roadmap for Semiconductors - 1998 Update*, http://notes.sematech.org/ntrs/Rdmpmem.nsf.

[13] Jason T. Zawodny and Peter M. Kogge, “Cache-In-Memory: A Lower Power Alternative,” Workshop on Power-Driven Microarchitecture, Int. Symp.on Computer Architecture (ISCA), Barcelona, Spain, June 27-28, 1998.

[14] Jason T. Zawodny, *Cache-In-Memory: Searching for Lower Energy Embedded DRAM*, M. S. Thesis, Dept. of Computer Science and Engineering, Univ. of Notre Dame, Notre Dame, IN, October, 2000.