Computational RAM: The case for SIMD computing in memory

Duncan Elliott, Michael Stumm and Martin Snelgrove
University of Alberta, University of Toronto and Carleton University respectively[1]

Abstract

DRAMs contain a very wide datapath internal to the chip which can deliver orders of magnitude greater memory bandwidth inside the chip than through the pins. The dilemma lies in deciding what to connect to the other end of this wide pipe to memory. We compare two alternatives: microprocessors (including MIMD multiprocessors) and SIMD processors. Microprocessors are better accepted and are suited to more applications. SIMD processors can better utilize the memory bandwidth available internal to the DRAM chips. These approaches can coexist.

Keywords

SIMD, smart memory, C•RAM, DRAM, PIM, IRAM, logic in memory, architecture.

1. Introduction

An immense, but under-used aggregate memory bandwidth exists within semiconductor memories, including DRAMs. In many projects, processors are being brought into memory in attempts to better use that bandwidth. We can best exploit this memory bandwidth by placing the "consumers" of this memory bandwidth, the processors, adjacent to the sense amplifiers. Depending on the particular memory architecture, the processors may be connected to secondary sense amplifiers, primary sense amplifiers, or the bitlines themselves. Our preferred target IC process is DRAM. DRAM processes, architectures and packaging have been tuned for high density and low price per bit. Our goal is to add processing to memory with minimal additional area or cost. To do this, we must avoid making changes to the architecture of the basic memory arrays, the IC process; and constrain die sizes, signal counts and power.

Our work on Computational RAM (C•RAM) has been in placing simple and medium complexity SIMD PEs in memory, from 8 Kb of ASIC SRAM to 16 Mb commodity DRAM [1-3]. These chips have run applications and have been shown to be useful in the fields of signal and image processing, computer graphics, CAD, database, and scientific computing. So far as memory bandwidth and processor hardware utilization are concerned, a strong argument can be made for incorporating SIMD PEs into memory. Indeed, a range of SIMD PEs have been incorporated into nearly every form of semiconductor memory [4-10].

The SIMD computing model, however, is not widely accepted. Rounding to the nearest power of a thousand, there are millions of programmers in the world, but only thousands of SIMD programmers. If C•RAMs were to only run user-written applications, there would be a small market indeed. Some programs, however, are distributed with the hardware (e.g. drivers for graphics accelerators) so the user does not require skills in mapping tasks onto SIMD processors. A conventional microprocessor incorporated into memory, which is supported with compilers for popular programming languages, will attract more applications, although some potential memory bandwidth and performance is lost.

2. Bandwidth

Memory speed has become an increasingly significant performance bottleneck [11,12], but memory bandwidth does not directly translate into computing performance. Many have observed that the bandwidth internal to DRAM chips is plentiful [13,2,14]. The memory bandwidth at various locations in a modest computer architecture are expressed in the graph in figure 1. Currently, 16Mb DRAMs are the cheapest per bit.

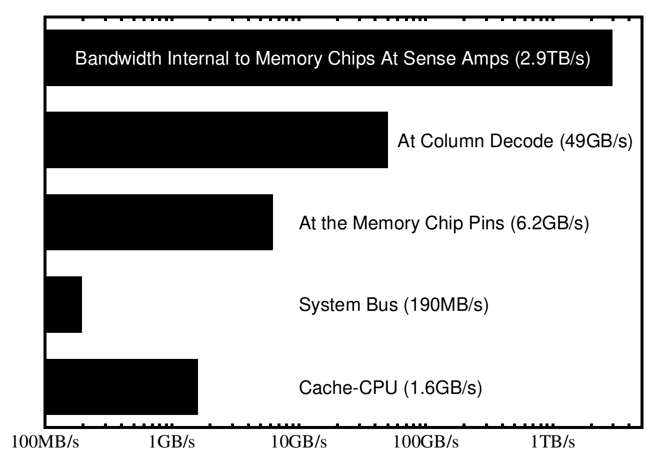

Fig. 1: Bandwidths at various points in a workstation

1.Further information can be found at:
http://www.ee.ualberta.ca/~elliott/cram/
email: elliott@ee.ualberta.ca;

This figure considers a "current" 200 MHz 64-bit workstation with 128 16 Mb DRAM chips (256 Mbytes total) with a row access time of 50ns, a cycle time of 90ns, and a fast page mode cycle time of 35ns. The DRAM chip requires 1 K cycles to refresh each memory cell, which implies there is one active sense amplifier per 1024 memory cells. Thus, the width of the internal datapath in one chip is 16 Kb per chip. In this 256 Mbyte system with 128 memory chips, the datapath is 2 Mb wide with an aggregate memory bandwidth of 2.9 Tbytes/s for each of read and write as shown in the top bar of figure 1. This wide internal datapath is best used by distributed logic (such as SIMD PEs), unless wiring area is to vastly exceed memory area.

As the 2 million wire datapath is multiplexed in multiple stages onto a 64 bit bus, memory bandwidth is lost. The first such constriction of bandwidth occurs inside the memory chips where the memory column decoders multiplex the datapath down to 128 bits per chip resulting in an aggregate system bandwidth of 23 Gbytes/s using the standard DRAM cycle. If 4 fast-page-mode accesses are made in 166ns, the bandwidth limit increases to 49 Gbytes/s. This bus can reasonably be routed across a chip to centralized logic, such as a microprocessor.

If, however, memory is only accessed from off-chip, the 16 data pins per chip (2 K pins total) limit the bandwidth to 2.8 Gbytes/s aggregate over all chips using the standard DRAM cycle, or 6.2 Gbytes/s in fast page mode transferring 4-word bursts between row accesses.

When the memory is accessed over a 64-bit bus then the bandwidth is further constricted to 190 Mbytes/s when accessed at the fast-page-mode rate described above. The 4-word burst transfers are reasonable for a cache line fill or write-back. Faster DRAM interfaces, such as Rambus [16] have peek transfer rates of 500 Mbytes/s and beyond, but the sustained bandwidth for cache line fills is still limited by the cache line size and the internal memory row access latency when accessing random data.

If a cache hierarchy could provide a 100% hit rate then the effective peak memory-data bandwidth is 1.6 Gbytes/s at a processor memory port, which is three orders of magnitude less than the untapped bandwidth available at the sense amps. For applications or routines with poor temporal locality for data references, such as the inner product of long vectors, the system bus dictates performance, with four orders of magnitude less bandwidth than the memory.

Future Bandwidths

We use a 1Gb DRAM [17] and a 500MHz 64-bit microprocessor as model memory and processor architectures for

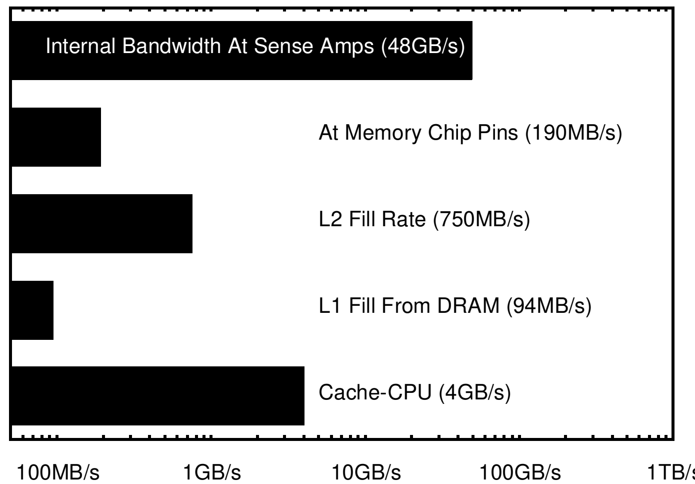

Fig. 2: Bandwidths in a 1Gb DRAM + microprocessor

future processor in memory systems. This chip has 32K active sense amps in a given cycle and a cycle time of 85ns in which a burst of 8 words of data can be transmitted with a period of 4.5ns. The memory bandwidth at the sense amps is 48 Gbytes/s. As designed, the memory bandwidth at the memory chip pins is 190 Mbytes/s for bursts of 8 16-bit words given a new row address each burst.

We examine a single and double level cache memory systems for the integration of a microprocessor with memory. Because cache performance is application dependent, we examine only the bandwidth at the DRAM memory interface for cache line fills or writebacks. We assume that the L2 cache has a wide line of 8 words by 64 bits which can be filled in parallel, exploiting some of the width of the wide internal memory datapath. For the single level cache, on the other hand, the 500MHz microprocessor access rate would keep this cache small. In particular, short wordlines would be employed, limiting the width of a parallel cache load to one 64-bit word in our model. Longer line sizes on small high-speed caches would increase conflict misses in any event. The cache fill bandwidth for L2 cache and the single level cache are 750 Mbytes/s and 94 Mbytes/s respectively. Note that the fill bandwidth at the single level cache is less than the bandwidth available at the memory chip pins.

The microprocessor's demand for data bandwidth, assuming a single data port to the memory system at the processor's full clock rate is 4 Gbytes/s. This is more than 5 times the fill bandwidth for the two level cache and 40 times that of the single level cache. Sparing the cost of a multilevel cache hierarchy could lead to a significant performance penalty when incorporating high-performance microprocessors into memory. The memory gap can be lessened by integrating a microprocessor into memory vs. having separate chips, but the peak data bandwidth of the microprocessor is less than a tenth of the internal memory bandwidth.

Bandwidth Utilization

The bandwidth internal to the memory can be utilized by moving the consumers of the bandwidth, namely the processors, on to the memory chip. How well the bandwidth is utilized depends on the configuration of the processors and where they are placed. Processors can be integrated into memories at the sense amplifiers, where the most memory bandwidth is available, or after the datapath is constricted by column decoders.

This approach of placing processors in memory is done with C•RAM. If the PEs are pitch-matched to a small number (e.g. 4-8) of sense amplifiers (or secondary sense amplifiers depending on the memory architecture), a significant fraction of the memory bandwidth can be harvested (top line in figure 1). The lengths of wires between processors and memory are uniform and are kept to a minimum since the PEs are abutted with memory. Memory latencies are kept low and consistent.

Despite the lower bandwidth at the column decoders vs. at the sense amps, some designers attach processors to DRAM after the column decoders [7]. They use more conventional processor designs that are too large to be directly connected to the wide internal databus at the sense amps. By placing memory column decoders between the memory and the microprocessor, a large fraction of the memory bandwidth is lost (see the second line in figure 1). Redesigning DRAM for lower latencies (such as reducing row access time by approximately a factor of two to 20 ns [15]) is possible, but radical changes to speed will have a radical impact on chip area and/or power consumption. Three possible ways to bridge the remaining memory gap are to use a conventional cache hierarchy, constrain applications to memory access patterns which are suited to the memory architecture or redesign the memory architecture to support multiple outstanding memory accesses. The wide internal datapath to memory can be used for wide parallel cache line loads and transparent write-backs [16]. Redesigning applications to make the best use of data already loaded into sense amps can improve memory bandwidth (equivalent to the DRAM fast page-mode access), by reducing the number of memory row accesses which are expensive in both energy and time. A vector processor in a memory chip [12] would have the bandwidth advantage of accessing memory along a row of DRAM (for unit strides or small strides) and memory access patterns which allow effective use of memory interleaving. Large strides (e.g. if a large array must be traversed in both row and column order) may require a memory row access for each word, increasing latency and sacrificing memory bandwidth. Some of the applications which use only unit stride memory accesses may also perform well on a SIMD architecture. It would be feasible to redesign the memory architecture by replicating the control circuitry to support a large number of independently addressable banks. Standard memories support up to two banks. While additional banks will not decrease memory latency, a processor architecture which supports a large number of non-blocking outstanding memory requests could use such a highly interleaved DRAM to increase memory bandwidth for *random* accesses, not just sequential.

Energy Efficiency

When the computing is done in the memory, less energy is consumed in sending data between chips. Energy can also be saved if each memory row access is better utilized [3]. C•RAM PEs consume 10-25% of the chip power. The sensing energy per memory row access dominates. A memory array can have 64 to 2048 times as many sense amps powered compared with the number of bits which are actually available after a column decode and used in a memory access. If more bits of these memory rows are used before the next memory row cycle, then the sensing energy is amortized over more processing.

3. Long Wordlines

An interesting question is whether conventional microprocessor architectures can be integrated efficiently into memory at the sense amps? We believe not. Architectures which place processors at the memory sense amps are constrained by the long wordlines found in DRAM designs. The wordlines are designed to be long to minimize the area taken by row decoders and other drivers. If a significant fraction of the DRAM internal memory bandwidth is to be harvested, then some form of multiprocessing is necessary, and the DRAM architecture clearly favours SIMD processing. In standard DRAM designs, all columns (or bitline pairs) in a basic memory array are addressed by a single row address. If the row decode and long wordlines found in commodity DRAM memories are not to be modified, then the processors are presented with data from typically 2048 to 4096 columns of memory, all accessed with the same row address. This presented memory word is too wide to use effectively as a processor word (in any conventional sense). To allow smaller blocks of memory to be separately addressed, more row decoders would be required with an additional overhead in area. Since, row decoders, not including pre-decoders, can be greater than 300 times the width of a DRAM memory cell, adding extra row decoders is not an inexpensive option. The DRAM's wide memory word can be divided among multiple processors with smaller datapaths, so that each processor sees a private memory, as in a distributed memory multiprocessor. The processors would make ac-

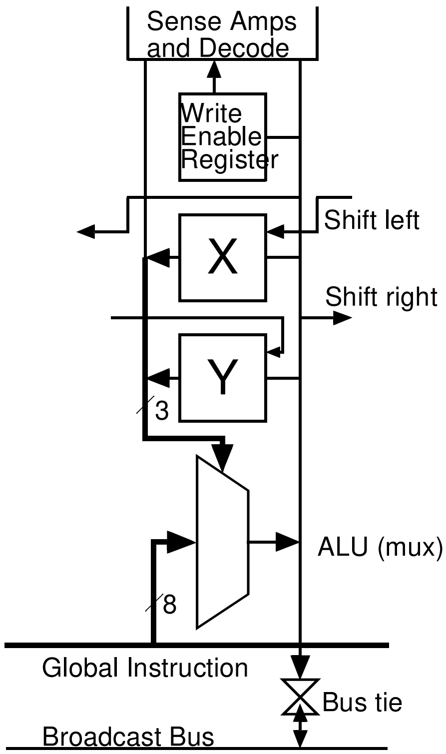

Fig. 3: The simple C•RAM PE that can be implemented in dynamic logic with only 75 transistors

cesses to a common memory address (uniform memory addressing). It would be unworkable for a MIMD multiprocessor to benefit from having processors autonomously executing their own instruction streams but performing loads, stores and instruction fetches to and from the same addresses in local memory at the same time. The shared memory address stream strongly suggests the use of a shared instruction stream as well.

In summary, we wish to maintain the DRAM architecture as it has already been optimized for low area and high yield. In maintaining the DRAM architecture, we are constrained to use a uniform memory addressing SIMD architecture.

4. Processor Resource Utilization

When should hardware resources be spent on building one complex processor or multiple simple processors? If the applications have little parallelism available (e.g. only instruction level parallelism) then a single processor with fast functional units with as wide a datapath as is useful may be called for. Not all bits of the datapaths of all functional unit would necessarily be fully utilized, but any utilization could bolster performance. If applications have a massive degree of parallelism and are suited to SIMD computation there are cost/performance advantages to using more copies of a simpler processor whereby those processors are more thoroughly utilized.

The C•RAM PE takes simplicity and high hardware utilization to extremes. The PE, shown in figure 3, requires only 75 transistors. The multiplexor used as an ALU to compute an arbitrary function of 3 inputs is the only functional unit and is used every cycle. There can be no loss of performance from bubbles in the pipeline or cache misses since it has but a single stage of logic and the PE runs at memory speeds without a cache. Since it runs a memory speed, it only needs a minimal register set. These registers are reused for interprocessor communication between nearest neighbours. One disadvantage of such a simple PE is that some operations, such as multiplication require more memory accesses to store intermediate results

Table 1: ALU algorithm and complexity

hardware algorithm	datapath width	trans-istors	area	time	area-time product
	Addition				
bit serial	1	$O(1)$	$O(1)$	$O(n)$	$O(n)$
ripple carry	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
carry save	n	$O(n)$	$O(n)$	$O(1)$	$O(n)$
carry save, binary out	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
carry look-ahead	n	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(n^2 \log n)$
	Multiplication				
bit serial	1	$O(1)$	$O(1)$	$O(n^2)$	$O(n^2)$
ripple carry	n	$O(n)$	$O(n)$	$O(n^2)$	$O(n^3)$
carry save	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
Wallace tree	n	$O(n^2)$	$O(n^2)$	$O(\log n)$	$O(n^2 \log n)$
	Bit permutation				
bit serial	1	$O(1)$	$O(1)$	$O(n)$	$O(n)$
bit-wise	n	$O(n)$	$O(n)$	$O(n)$	$O(n^2)$
memory look-up	n	$O(n2^n)$	$O(n2^n)$	$O(1)$	$O(n2^n)$
Batcher banyan	n	$O(n \log n)$	$O(n^2)$	$O(\log n)$	$O(n^2 \log n)$

Assuming an abundance of parallelism in applications, simple ALU structures can have greater performance per silicon area compared to the conventional complex ALU structures found in microprocessors. Simple ALU structures can be more efficient in the use of transistors and circuit area. In the C•RAM bit serial ALU, for instance, every bit of every output of every stage of logic is active during every ALU cycle — there is only one stage with one output bit. The architects of modern high performance microprocessors, on the other hand, make a trade-off and choose complex ALUs with low latency, over simpler ALUs with reduced area and transistor count. These microprocessors are better suited for applications with not much parallelism.

To compare several ALU algorithms without looking at specific silicon implementations, we look at their asymptotic complexity as functions of operand size in bits (n) with $O()$ notation in table 1. The areas of the carry look-ahead adder and Wallace tree multiplier are assumed to be $O(n^2)$ based on the number of transistors. We assume that any memory required is sufficiently large and that memory size and latency are independent of n. This asymptotic analysis of algorithms ignores constant coefficients and lower order terms which are not necessarily insignificant for common 8-bit to 64-bit data-types. While table 1 examines latency, we should note that pipelining can be used to increase the throughput of multi-bit multipliers (and occasionally adders) by at most the number of pipeline stages.

For addition, the bit-serial ALU with a one-bit datapath requires a constant number of transistors and area regardless of the size of the operands, but is tied for the longest latency, $O(n)$, that is, latency is linear in the length of the operands. We use the area·time product here as a rough estimate of the cost/performance ratio, which we wish to minimize. Bit-serial addition and carry save addition both have the lowest order area·time product of $O(n)$. Ripple-carry addition uses an n-bit datapath, but is not faster asymptotically than the bit-serial addition with $O(n)$ latency. Carry-save addition postpones propagation of the carries to reduce latency. In order to postpone the propagation of carries further than one binary digit position the encoding uses multiple bits to represent one binary digit position. For example, the carry-save result of an addition may have for each binary digit position i, a bit representing the sum of the i^{th} bits and a bit representing the carry from the sum of the $i-1$ bits. Without the carry propagation, the latency for addition is $O(1)$ and the area·time product is $O(n)$. The result is provided in the redundant carry-save notation, which is suitable as an operand for another addition. When a *binary* representation result is required, any carry which does not increase the order of the hardware cost (e.g. ripple carry), makes carry-save addition just as slow as the ripple-carry adder. A carry look-ahead ALU is designed for low latency at a greater hardware cost. It can produce a binary result in $O(n)$ time. The carry-propagation path requires $O(n^2)$ transistors to achieve the fastest carry propagation of these adders. The low latency of the carry-look-ahead adder comes at the cost of a greater area, resulting in the worst area·time product. Other low-latency adders, such as logarithmic carry select [18] have similar lower asymptotic bounds on latency and transistors.

Considering multiplication with n-bit operands, the bit-serial and carry-save ALUs again have the best area·time product. The ripple-carry ALU is less competitive. For all three of these ALUs, multiplication is performed as iterative addition. A Wallace-tree [19] combinational multiplier includes a tree of carry-save adders. In contrast to the previous three ALUs, each gate in the Wallace-tree of combinational logic is used once per new operand. The Wallace tree has the lowest latency, but at a high cost in area.

For some applications, such as graphics and logic simulation, manipulating bits on a processor with a word wider than a bit (e.g. 32 bits or 64 bits) is inefficient (processor logic is used to manipulate data but the result is not used). We use permutation here as an example of bit manipulation. Both a bit-serial and an n-bit word processor can perform an arbitrary reordering of the bits in an n-bit word, one bit at a time but the bit-serial processor can do it with significantly less hardware. The n-bit processor can perform the permutation faster using a memory lookup table, but only at great hardware expense. A dedicated permutation switch (such as a Batcher banyan network) is less expensive and has an area·time product of $O(n^2 \log n)$. For performing an arbitrary bit permutation of an n-bit word, the bit-serial processor is the most effective.

Bit-serial arithmetic has other advantages and disadvantages. With bit-serial arithmetic, any sized operand can be handled efficiently on the same hardware, including different-sized operands to the same operator. Results from addition and multiplication can be sized to maintain full accuracy. In contrast to the low latency ALU designs, the same bit-serial hardware is used for addition, multiplication, and any other operation. One downside to bit-serial arithmetic, however, is that an $O(1)$ PE cannot store an n-bit partial sum during a multiplication operation, so more memory operations are required when compared to an n wide multiplier. Bit-serial arithmetic is justified when more PEs with simpler ALUs can be fully utilized. If the applications do not have sufficient parallelism, then the silicon resources would be better spend on PEs which have the word parallelism that we take for granted in microprocessors.

In summary, for addition, multiplication, and bit permutation, the bit-serial ALU has the lowest order area·time product which corresponds to the best performance/cost ratio. The carry-save n-bit wide ALU has the same order area·time product as the bit serial ALU for addition and multiplication, *provided* that a binary result is not required. The ALU structures used in modern microprocessors have lower latencies, which is important for sequential computation. In our opinion, if the parallelism is available, silicon area and circuit resources are better spent on having more copies of simpler ALU structures such as ALUs which perform bit-serial or carry save arithmetic.

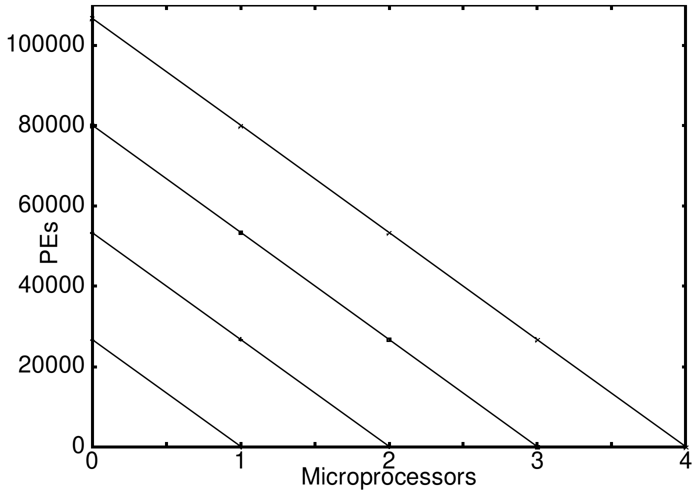

Fig. 4: Trade-off between number of SIMD PEs and Microprocessors on a chip containing PEs, Microprocessors and Memory

5. Hybrid MIMD-SIMD

As die sizes grow and feature sizes shrink, it will be possible to integrate multiple "medium complexity" microprocessors with memory. Multiple microprocessors can be used to reduce the latency of a parallel application or simply increase the processor throughput by running multiple processes without need to find parallelism in applications[20].

Although 2 billion transistor DRAMs have been fabricated [22], the transistors in logic are less densely packed. In IC manufacture, the largest DRAMs and microprocessors utilize a significant portion of the area of the reticule used in exposing a pattern from a mask onto photoresist. Efforts to place microprocessors in memory have resulted in compromises in the density of both. For instance Toru Shimizu et al. incorporate a RISC microprocessor into 16Mb of DRAM in a 64Mb DRAM process [21].

When chip resources become available to build larger systems, we advocate a hybrid approach to computing in memory involving one or more microprocessors plus SIMD PEs. Modern microprocessors range in complexity from 1 million transistors for video game processors to 9 million transistors for high-end microprocessors. In figure 4, a simple trade-off is shown between devoting transistors to PEs which use 75 transistors and medium complexity processors assumed to require 2 million transistors. MIMD-SIMD hybrids have been previously proposed as a way to reduce the latency of portions of an application which are not suitable for SIMD parallelism.

6. Conclusion

There are bandwidth advantages to integrating processors into memory chips. We indicate that SIMD processors can most effectively use this wide internal memory datapath. Assuming that applications have the necessary parallelism to exploit, the asymptotic analysis shows that simple bit-serial ALUs can provide higher performance per area than the conventional wider ALUs found in microprocessors. For non-parallel applications or the sequential portion of partially-parallel applications, fast sequential processing is required. For the sequential and lightly parallel tasks, one of more microprocessors, on-chip or off-chip, would complement the SIMD PEs.

7. Acknowledgments

The authors wish to thank Robert McKenzie, Christian Cojocaru, Peter Nyasulu, Dickson Cheung, Albert Kwong, Bruce Cockburn, Sethuraman Panchanathan, Tet Yeap, Wayne Loucks, Thinh Le, Dick Foss, Peter Gillingham, Graham Allan, Iain Scott, Randy Torrance, David Frank, David Somppi, Randy Gibson, Howard Kalter and John Barth for technical exchange and feedback. This work has received support from MOSAID Technologies, IBM, Nortel, NSERC, CMC and Micronet.

8. References

[1] D. G. Elliott and W. M. Snelgrove. "C•RAM: Memory with a Fast SIMD Processor." In *Proceedings of the Canadian Conference on VLSI*, pages 3.3.1--3.3.6, Ottawa, October 1990.

[2] Duncan G. Elliott, W. Martin Snelgrove, and Michael Stumm. "Computational RAM: A Memory-SIMD Hybrid and its Application to DSP." In *Custom Integrated Circuits Conference*, pages 30.6.1--30.6.4, Boston, MA, May 1992.

[3] Duncan Elliott, Martin Snelgrove, Christian Cojocaru, and Michael Stumm. "A PetaOp/s is Currently Feasible by Computing in RAM." In *PetaFLOPS Frontier Workshop*, Washington DC, February 1995.

[4] Jim Childers, Peter Reinecke, and Hiroshi Miyaguchi. "SVP: A Serial Video Processor". *IEEE 1990 Custom Integrated Circuits Conference*, pages 17.3.1--17.3.4, May 1990.

[5] Per-Erik Danielsson, Par Emanuelsson, Keping Chen, and Per Ingelhag. "Single-Chip High-Speed Computation of Optical Flow." In *IAPR International Workshop on Machine Vision Applications*, pages 331--335, November 1990.

[6] Nobuyuki Yamashita, Tohru Kimura, Yoshihiro Fujita, Yoshiharu Aimoto, Takashi Manaba, Shin'ichiro Okazaki, Kazuyuki Nakamura, and Masakazu Yamashina. "A 3.84GIPS Integrated Memory Array Processor with 64 Processing Elements and 2Mb SRAM." *IEEE Journal of Solid-State Circuits*, 29(11):1336--1343, November 1994.

[7] Peter M. Kogge, Toshio Sunaga, Hisatada Miyataka, Koji Kitamura, and Eric Retter. "Combined DRAM and Logic for Massively Parallel Systems." In *Conference*

on Advanced Research in VLSI*, pages 4--16, Chapel Hill, NC, March 1995.

[8] Maya Gokhale, Bill Holmes, and Ken Iobst. “Processing in Memory: the Terasys Massively Parallel PIM Array.” *Computer*, 28(3):23--31, April 1995.

[9] Yoshiharu Aimoto, Tohru Kimura, Yoshikazu Yabe, Hideki Heiuchi, et al. “A 7.68GIPS 3.84GB/s 1W Parallel Image-Processing RAM Integrating a 16Mb DRAM and 128 Processors.” In *International Solid-State Circuits Conference*, pages 372--373, San Francisco, February 1996. IEEE.

[10] Masuyoshi Kurokawa, Akihiko Hashiguchi, Ken’ishiro Nakamura, Hiroshi Okuda, et al. “5.4GOPS Linear Array Architecture DSP for Video-Format Conversion.” In *International Solid-State Circuits Conference*, pages 254--255, San Francisco, February 1996. IEEE.

[11] Wm. A. Wulf and Sally A. McKee. Hitting the Memory Wall: Implications of the Obvious. *Computer Architecture News*, 23(1):20--24, March 1995.

[12] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. A Case for Intelligent RAM. *IEEE Micro*, 17(2):34--44, March 1997.

[13] Harold S. Stone. “A Logic-in-Memory Computer.” *IEEE Transactions on Computers*, C-19(1):73--78, January 1970.

[14] D. Patterson, T. Anderson, N. Cardwell, R. Fromm, K. Keeton, C. Kozyrakis, R. Thomas, and K. Yelick. Intelligent RAMs: Chips That Remember and Computer. In *International Solid-State Circuits Conference*, pages 224--225, San Francisco, February 1997. IEEE.

[15] Glen Giaclone, Robert Busch, Frank Creed, Alex Davidovich, Sri Divakaruni, Charles Drake, et al. A 1MB, 100MHz Integrated L2 Cache Memory with 128b interface and ECC Protection. In *International Solid-State Circuits Conference*, pages 370--371, San Francisco, February 1996. IEEE.

[16] Rambus. Rambus: Architectural Overview. Technical report, Mountain View, California, 1992.

[17] M. Horigucho, T. Sakata, T. Sekiguchi, S. Ueda, H. Tanaka, and all. An experimental 220MHz 1Gb DRAM. In *International Solid-State Circuits Conference*, San Francicso, February 1995.

[18] Daniel Booberpuhl, Richard Witek, Randy Allmon, Robert Anglin, and Sharon Britton. A 200MHz 64b Dual-Issue CMOS Microprocessor. In *International Solid-State Circuits Conference*, pages 106--107, San Francisco, February 1992.

[19] C. S. Wallace. A suggestion for a Fast Multiplier. In Jr. Earl E. Swartzlander, editor, *Computer Arithmetic*, pages 114--117. IEEE Computer Society Press, 1990.

[20] John L. Hennessy and David A. Patterson. *Computer Architecture: a Quantitative Approach*. Morgan Kaufman, San Mateo CA, fourth printing edition, 1990.

[21] Toru Shimizu, Jiro Korematu, Mitsugu Satou, Hiroyuki Kondo, et al. A Multimedia 32b RISC Microprocessor with 16Mb DRAM. In *International Solid-State Circuits Conference*, pages 216--217, San Francisco, February 1996. IEEE.

[22] T. Murotani, I. Naritake, T. Matano, T. Ohtsuki, N. Kasai, H. Koga, K. Koyama, K. Nakajima, H. Yamaguchi, H. Watanabe, and T. Okuda. A 4-level Storage 4Gb DRAM. In *International Solid-State Circuits Conference*, pages 74--75, San Francisco, February 1997. IEEE.