GP-SIMD Processing-in-Memory

AMIR MORAD, Technion
LEONID YAVITS, Technion
RAN GINOSAR, Technion

We present GP-SIMD, a novel hybrid general purpose SIMD computer architecture that resolves the issue of synchronization by in-memory computing, through combining data storage and massively parallel processing. The GP-SIMD's arithmetic, logic and associative characteristics and its intercommunication network are discussed. GP-SIMD employs modified SRAM storage cells. An analytic performance model of the GP-SIMD architecture is presented, comparing it to Associative Processor and to conventional SIMD architectures. Cycle-accurate simulation supports the analytical comparison. Assuming a moderate die area, GP-SIMD architecture outperforms both the Associative Processor and the SIMD co-processor architectures by almost an order of magnitude while consuming less power.

Categories and Subject Descriptors: **C.1.2 Multiple Data Stream Architectures (Multiprocessors), C.1.3 Other Architecture Styles**

General Terms: Design, Algorithms, Performance, Energy

Additional Key Words and Phrases: Multicore, SIMD, Associative Processor, Processing In Memory, PIM

1. INTRODUCTION

Machine learning algorithms performed on High Performance Computers (HPC) [56] address complex challenges such as mapping the human genome, investigating medical therapies, pinpointing tumors, predicting climate trends and executing high frequency derivative trading. These problems are plagued by exponentially growing datasets and pose a major challenge to HPC architects as they cannot be adequately addressed by simply adding more parallel processing. These problems are sequential in the sense that each parallelizable step depends on the outcome of the preceding step, and typically, large amount of data is exchanged (synchronized) between sequential and parallel processing cores in each step [47] (Figure 1). While microprocessor performance has been doubling every 18-24 months, this improvement has not been matched by external memory (DRAM) latencies, which have only improved by 7% per year [27]. HPC architectures are thus challenged by the difficulty of synchronizing data among the processing units, having material impact on speedup and power dissipation.

The preferred solution no longer features a single processing unit augmented by memory and disk drives, but a different class of processors capable of exploiting data-level parallelism. Vector machines and SIMD architectures are a class of parallel computers with multiple processing units performing the same operation on multiple data points simultaneously [1][23][6]. Such machines exploit data level parallelism, and are thus well suited for machine learning over Big Data [56]. High utilization of SIMD processor requires very high computation-to-bandwidth ratio and large data sets [37]. Excess of coarse SIMD computing elements operating at high rates results in irregular thermal density and hotspots [55], further limiting SIMD scalability.

Power dissipation and on-chip communication are the primary factors limiting the scalability of on-chip parallel architectures [12]. These factors may become more pronounced as the size and complexity of data sets continue to grow faster than computing resources.

This research was funded in part by the Intel Collaborative Research Institute for Computational Intelligence (ICRI-CI) and by Hasso-Plattner Institute (HPI).

Author's addresses: Amir Morad (*), E-mail: amirm@tx.technion.ac.il; Leonid Yavits (*), E-mail: yavits@tx.technion.ac.il; Ran Ginosar (*), E-mail: ran@ee.technion.ac.il; (*) Authors are with the Department of Electrical Engineering, Technion-Israel Institute of Technology, Haifa 3200003, Israel.

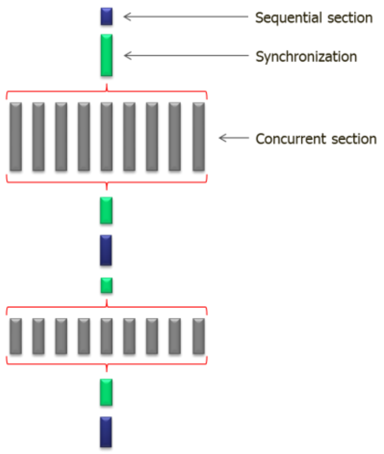


Figure 1. Typical sequential / parallel processing flow

It has been shown [42][29] that the maximal acceleration of a fine grain workload containing sequential and concurrent parts, occurs when the sequential section is assigned to a large, high ILP sequential processor (containing, for example, accelerators such as double precision floating point unit, function generator and branch prediction unit), depicted on the right side of Figure 2, while the concurrent part is assigned to a massive number of fine grain low power processing array, shown on the left side of Figure 2. That array is organized as a conventional SIMD (CSIMD). An immediate limiting factor of such architectures is synchronization requiring data exchange [63] between the sequential processor and the processing array, depicted as a red bus in Figure 2.

In this paper we propose a novel, hybrid general purpose SIMD computer architecture that resolves the issue of synchronization by in-memory computing, through combining data storage and massively parallel processing. Figure 3 details the architecture of the GP-SIMD processor, comprising a sequential CPU, a shared memory array, instruction and data caches, a SIMD coprocessor, and a SIMD sequencer. The SIMD coprocessor contains a large number of fine-grain processing units, each comprising a single bit ALU, single bit function generator and a 4-bit register file. The GP-SIMD processor is thus a large memory with massively parallel processing capability. No data synchronization between the sequential and parallel segments is required since both the general purpose sequential processor and SIMD co-processor access the very same memory array. Thus, no time and power penalties are incurred for synchronization.

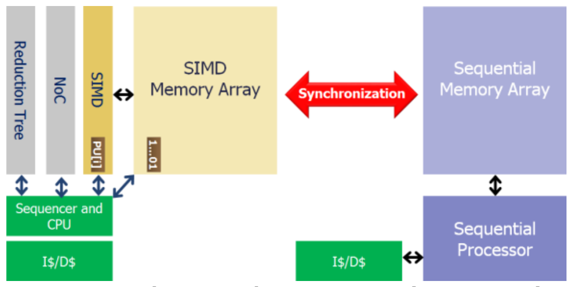


Figure 2. Synchronization between sequential processor and conventional SIMD co-processor

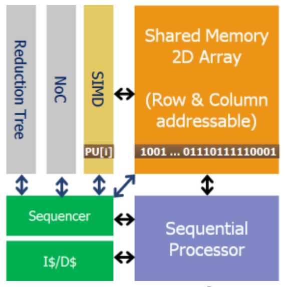


Figure 3. GP-SIMD architecture

The GP-SIMD delivers a number of advantages over CSIMD architecture:

- Data processing and data storage are unified. There is no need for data transfer between sequential memory and SIMD PUs;
- GP-SIMD allows concurrent operation of the sequential processor and SIMD co-processors on the shared memory, allowing the sequential processor to offload a task to the SIMD while continuing to process some other sequential functions.
- The number of GP-SIMD fine grain processing units matches the number of memory rows, striving to match the entire dataset. This enables massive parallelism and mitigates the PU-to-external-memory bottleneck of CSIMD architectures [43].
- The GP-SIMD architecture enables the sequential processor to associatively address the memory array. It may thus allow reduction of software complexity for certain sequential algorithms.
- GP-SIMD power dissipation is distributed uniformly over the entire processing array rather than being concentrated around a smaller number of large, power hungry processing cores. Thus, there are fewer hotspots leading to further reduction of temperature dependent leakage power [8].

The first contribution of this paper is setting a taxonomy categorizing previous works in the processing-in-memory (PiM) and SIMD fields. The second contribution is the novel integration of a fine grain massively parallel SIMD co-processor with a standard general purpose sequential processor and a shared 2D memory array, leading to improvement in performance as well as reduction in power dissipation. The third contribution of this paper is the comparative performance and power analysis of GP-SIMD, a Conventional SIMD processor (CSIMD) and an Associative Processor (AP) [62], supported by analytical modeling and cycle-accurate simulations. Our research indicates an inflection point where GP-SIMD outperforms both CSIMD and AP in both performance and power.

The rest of this paper is organized as follows. Section 2 discusses the related work. Section 3 provides a detailed description of the GP-SIMD architecture and its operation. Section 4 presents analytical modeling and cycle accurate simulation of GP-SIMD performance and power consumption and compares it to CSIMD processor and to an AP models. Section 5 concludes this paper.

2. RELATED WORK

A substantial portion of this work is reserved for development and evaluation of analytical models for the GP-SIMD, CSIMD and AP models. Analytical models for most building blocks of modern ICs (e.g., processing units, interconnection networks) have been extensively researched. These models enable the exploration of the chip design space in a reasonable timeframe, and are thus becoming an increasingly important technique used in the design of chip multiprocessors [15][26][29][36][61]. Polack [45] modeled the performance of modern CPUs as a square root function of the resource assigned to them. Morad *et al.* [42] and Hill *et al.* [42][29] augmented Amdahl's law with a corollary to multicore architecture by constructing a model for multicore performance and speedup.

The interactions between multiple parallel processors incur performance overheads. These overheads are a result of synchronization, communication and coherence costs. Analytical models for these overheads have been studied as well. Morad *et al.* [42] modeled the synchronization, communication and coherence as a time penalty on Amdahl law, concluding that asymmetric multiprocessors can reduce power consumption by more than two thirds with similar performance compared to symmetric multiprocessors. Yavits *et al.* [63] studied the overheads and concluded that in applications with high inter-core communication requirements, the workload should be executed on a small number of cores, and applications of high sequential-to-parallel synchronization requirements may better be executed by the sequential core. Loh *et al.* [36] introduced an extension to Hill and Marty's multi-core cost/ performance model to account for the uncore components, concluding that to sustain the scalability of future many-core systems, the uncore components must be designed to scale sub-linearly with respect to the overall core count. Morad *et al.* [39][41][40] presented several frameworks that, given (a) a multicore architecture consisting of last level cache (LLC), processing cores and a NoC interconnecting the cores and the LLC; (b) workloads consisting of sequential and concurrent tasks; and (c) physical resource constraints (area, power, execution time, off-chip bandwidth), find the optimal selection of a subset of the available processing cores and the optimal resource allocation among all blocks.

Modeling of SIMD processing performance has been thoroughly discussed. Hong *et al.* [30] propose a simple analytical model that estimates the execution time of massively parallel programs. Zhang *et al.* [66] develop a microbenchmark-based performance model for NVIDIA GeForce 200-series GPUs.

The concept of mixing memory and logic has been around since the 1960s [47]. Similar to DAP [51], STARAN [53][9], CM-2 [59], and GAPP [16] computer architectures, GP-SIMD belongs to a Processing-In-Memory (PiM) class of architectures that use a large number of Processing Units (PUs) positioned in proximity to memory arrays to implement a massively parallel SIMD computer. To differentiate between GP-SIMD and other works, and since keywords like PiM and SIMD are often used with different meanings in mind, we first classify previous works as follows:

— In-Memory, SIMD: A very large number of small (typically single bit) SIMD processing unit are implemented on memory periphery, matching the number of memory rows or columns.
— In-Memory, Associative: A very large number of small (typically single bit) associative processing unit are implemented on memory periphery, matching the number of memory rows, or columns, or even the number of memory bits.
— Near-Memory, SIMD: Many processing unit (usually clustered) are integrated with large blocks of memory. Typically, the processing units incorporate floating point engines.
— Near-Memory, Non-SIMD: Large sequential processors integrated with large memory blocks.
— Off-Memory, SIMD Accelerators: Several processing units, typically with integer and floating point engines, operated by main processors.

The taxonomy of related works is presented in Figure 4.

In-Memory, SIMD category has been a popular subject of research: Gokale *et al*. [22] designed and fabricated Terasys, a processor-in-memory (PIM) chip, a standard 4-bit memory augmented with a single-bit ALU controlling each column of memory. Lipovsky *et al.* [35] introduced a Dynamic Associative Access Memory (DAAM) architecture where a large number of single bit processing units are put in a DRAM's sense amps. Such DRAM with single-bit ALU delivering arithmetic as well as associative processing is shown to offer nearly three orders of magnitude better cost performance than a conventional microprocessor. Dlugosch *et al.* [16][17] introduced the Automata Processor, a massively parallel processor integrated on a DRAM chip, specifically designed to implement complex regular expression automata for pattern matching.

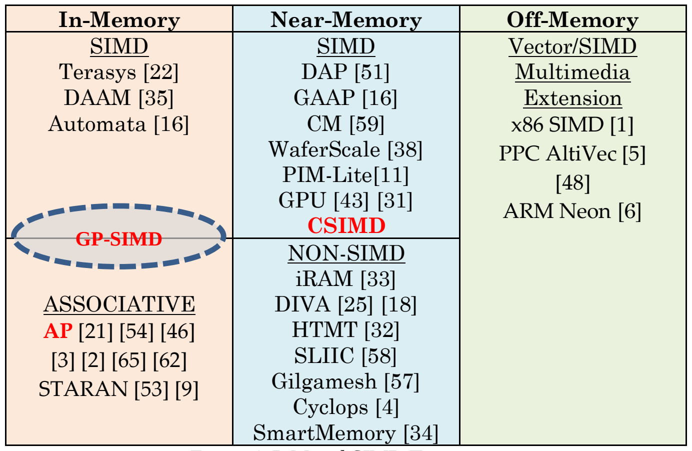
In-Memory, Associative category has been thoroughly researched: Foster [21] detailed an Associative Processor that combines data storage and data processing, and functions as a massively parallel SIMD processor and a memory at the same time. Sayre *et al.* [53] and Batcher [9] discussed STARAN, Goodyear's single-bit SIMD array processor, where the PEs communicate with a multi-dimensional access (MDA) memory. In bit-slice access mode, associative operations are performed on single bit-slice of all words in parallel, while the word access mode is used in the I/O operations to access a single word, so as to allow faster word access to the array from an external data source. Scherson *et al.* [54] distributed logic among slices of storage cells such that a number of bit-planes share a simple logic unit enabling bit-parallel arithmetic. Potter *et al.* [46] presented a parallel programming paradigm called ASC (Associative Computing), offering an efficient associative-based, programming model combining numerical computation (such as convolution, matrix multiplication, and graphics) with non-numerical computing (such as compilation, graph algorithms, rule-based systems, and language interpreters). Akerib *et al.* [3][2] demonstrated an efficient implementation of several computer vision algorithms on Associative Processor. Yavits *et al.* [65] designed and implemented stand-alone Associative Processor chip and studied its performance. Further, Yavits *et al.* [62] presented a computer architecture where an Associative Processor replaces the last level cache and the SIMD accelerator, showing performance and power benefits.

Near-Memory, SIMD category incorporates several notable works. Reddaway [51] presented a Distributed Array Processor (DAP), a 64×64 single bit SIMD processing units (PEs) with 4096 bits of storage per PE. Programs for the DAP were written in DAP FORTRAN with 64x64 matrix and 64 units vector primitives. Cloud [16] presented the Geometric Arithmetic Parallel Processor (GAPP), a two dimensional array of single bit SIMD processors, where each processor is allocated with and addresses a distinct memory bank. Tucker *et al.* [59] presented the Connection Machine’s hypercube arrangement of thousands of single bit SIMD processors, each with its own 4 kbits of RAM. Each chip contained a communication channel, 16 processors and 16 RAMs. The CM-1 employed a hypercube routing network, a main RAM, and an input/output processor. The CM-2 added floating-point numeric co-processors and more RAM. Midwinter *et al.* [38] presented a wafer-scale processor containing 128*128 processing SIMD units each consisting of ALU, 128 bits of local RAM, an Input/Output register and a control register. Brockman *et al.* [13] evaluated the die cost *vs.* performance tradeoffs of PIM-Lite system consisting of a multithreaded core with SIMD accelerator integrated with DRAM, that could serve as the memory system of a host processor, realize a performance speedup of nearly a factor of 4 on N-Body force calculation. Note however that the integration of logic into DRAM cells is neither simple nor efficient due to the different manufacturing process utilized for DRAM.

Near-Memory, Non-SIMD category has been a popular subject of research. Kozyrakis et al. [33] studied IRAM (intelligent RAM). IRAM utilizes the on-chip real-estate for dynamic RAM (DRAM) memory instead of SRAM caches, based on the fact that DRAM can accommodate 30 to 50 times more data than SRAM. Having the entire memory on the chip, coupled to the processor through an on-chip high bandwidth and low-latency interface, benefits architectures that demand fast memory I/O. This is clearly limited to applications that require no more data than can fit in that DRAM. Hall *et al.* [25] developed DIVA, the Data-Intensive Architecture, combining PIM memories with one or more external host processors and a PIM-to-PIM interconnect. DIVA increases memory bandwidth through performing selected computation in memory, reducing the quantity of data transferred across the processor-memory interface; and provides communication mechanisms for moving both data and computation throughout memory, further bypassing the processor-memory bus. Kogge *et al.* [32] proposed to overcome latencies between the main memory and the high performance CPUs with HTMT, a multi-level memory system using Processing-In-Memory architectures which actively manage the flow of data without centralized CPU control. Suh *et al.* [58] presented a PIM-based multiprocessor system, the System Level Intelligent Intensive Computing (SLIIC) Quick look (QL) board. This system includes eight DRAM PIM array chips and two FPGA chips implementing an interconnect network. Sterling *et al.* [57] studied Gilgamesh, an architecture that extends existing PIM capabilities by incorporating advanced mechanisms for virtualizing tasks and data and providing adaptive resource management for load balancing and latency tolerance. The Gilgamesh execution model is based on a middleware layer allowing explicit and dynamic control of locality and load balancing. Almási *et al.* [4] introduced Cyclops, an architecture that integrates large number of processing cores, main memory and communications hardware on a single chip, and studied the performance of several scientific kernels running on different configurations of this architecture. Kumar [34] introduced "Smart Memory", 2D array of packet processing units, each with local memory array. The PEs are interconnected together via a 2D mesh NoC. The smart memory chips are aimed at speeding packet processing jobs on large dataset.

Off-Memory Vector/SIMD Multimedia Extension architectures include Intel's x86 SIMD (from MMX to later generations [1]), IBM/Motorola/Apple AltiVec engine [5] [48], and ARM Neon [6].

Our proposed GP-SIMD architecture is different from the cited works, as it combines a sequential processor, in memory hybrid SIMD and Associative co-processor and a two-dimensional-access memory such that: (a) the data are shared between the sequential processor and the SIMD co-processor, that is, the data remain in the same place rather than moved back-and-forth between the two; (b) the sequential processors and the SIMD co-processor do not incur penalties due to the shared memory (relative to a single port memory); and (c) the shared memory array is similar to a standard on-chip single port memory in terms of area, power and performance. The GP-SIMD is somewhat similar to the Associative Processor [21]. We compare these two architectures, as well as an abstract GPU like SIMD processor in terms of area, energy and performance, concluding that the GP-SIMD prevails in all criteria.



In-Memory	Near-Memory	Off-Memory
SIMD Terasys [22] DAAM [35] Automata [16]	SIMD DAP [51] GAAP [16] CM [59] WaferScale [38] PIM-Lite[11] GPU [43] [31] CSIMD	Vector/SIMD Multimedia Extension x86 SIMD [1] PPC AltiVec [5] [48] ARM Neon [6]
GP-SIMD		
ASSOCIATIVE AP [21] [54] [46] [3] [2] [65] [62] STARAN [53] [9]	NON-SIMD iRAM [33] DIVA [25] [18] HTMT [32] SLIIC [58] Gilgamesh [57] Cyclops [4] SmartMemory [34]	

Figure 4. PiM and SIMD Taxonomy

3. THE GP-SIMD PROCESSOR

In this section we present the GP-SIMD, detail its internal architecture and its arithmetic, logic and associative processing capabilities, and establish analytical performance and power models. Further, we discuss SRAM-like implementation of the GP-SIMD memory array.

3.1 Top Level Architecture

The GP-SIMD is a hybrid general purpose and SIMD computer architecture that resolves the issue of synchronization by in-memory computing, through combining data storage and massively parallel processing. As illustrated in Figure 7, references to on-chip memory 'row' (r) and 'column' (c) are physical. Each row may contain many words of software programmable width (w) (if w is constant for all words, the number of words is thus $r \cdot c/w$). The number of rows typically matches the dataset elements, N.

— Sequential processor accesses either one word at a time, or multiple words. Typically, such a transaction accesses one physical row at a time.
— The SIMD reads/writes a bit-slice (having r bits) comprising the same bit-number from all words in some partition of the memory. Physically, it may access multiple bits in a physical row and all rows per access, namely accesses multiple columns of the physical array.

Figure 3 details the architecture of a GP-SIMD processor, comprising the sequential CPU, shared memory array, instruction and data cache, SIMD coprocessor, SIMD sequencer, interconnection network and a reduction tree. The sequential processor schedules and operates the SIMD processor via the sequencer. In a sense, the sequential processor is the Master controlling a slave SIMD co-processor. The SIMD coprocessor contains a number of fine-grain processing units (PUs), as depicted in Figure 5, each containing a single bit Full Adder (*FA*), single bit Function Generator (*FG*) and a 4-bit register file, *RA*, *RB*, *RC* and *RD*. A single PU is allocated per row of the shared memory array, and physically resides close to that row. The PUs are interconnected using an interconnection network (discussed in the context of Figure 8 below). The set of all r registers of the same name constitute a *register slice*. Note that the length of the memory row (e.g., 256 bits) may be longer than the word length of the sequential processor (e.g., 32 bits), so that each memory row may contain several words.

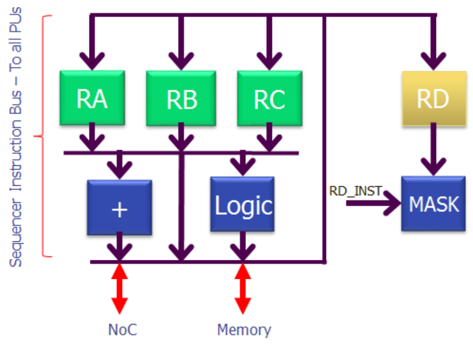


Figure 5. GP-SIMD Processing unit

When the SIMD reads data from the shared memory, the contents of a bit slice of the memory are transferred into the register slice (*RAs*, *RBs* or *RCs*). Upon writing to the shared memory, the contents of one of the register slices are transferred into the GP-SIMD memory array. A conditional register (*RD*) is utilized to enable special/masked operations as depicted in TABLE 1.

The *RD_INST* is a part of the SIMD co-processor instruction bus driven by the sequencer, and each RD_INST value specifies an operation (depicted as a bus going from the sequencer to the SIMD co-processor in Figure 3). While the first four operations are self-explanatory, the last two operations allow the sequential processor to perform associative commands on the memory array, as detailed in section 3.4 below.

TABLE 1
Conditional/Masked Operations

RD_INST	RD Value	Operation
00	0	Memory access (read/write) by the sequential processor or the SIMD co-processor
00	1	Memory access; If memory-read by SIMD co-processor, reset RB
01	0	SIMD co-processor memory-write of RA
01	1	SIMD co-processor memory-write of RB
10	0	Disable row for memory access by sequential processor
10	1	Enable row for memory access by sequential processor

3.2 Operating Modes

The GP-SIMD may be operated in three modes:

— *Sequential Processing mode*, in which the general purpose sequential processor accesses the data in memory during the execution of the sequential segments of a workload; Sequential processor addresses a row of the memory, and reads/writes to a group of columns of the memory, corresponding to its data word-length (e.g., 32 bits). In practice, the sequential processor utilizes a cache and accessing the memory is based on cache lines, as discussed in Section 3.7 below.

— *SIMD Processing mode*, in which the parallelizable segments of a workload are executed. SIMD addresses a column of the memory, and reads or writes to all or a portion of the rows of the memory. Note that each PU corresponds to a single memory row. Thus in a single SIMD memory access, all PUs (or portion thereof) simultaneously read from or write into a single column of the memory array.

— *Concurrent Mode*, in which both the sequential processor and the SIMD co-processor can access the shared memory.

No data synchronization between sequential and parallel segments is required since both the sequential processor and SIMD co-processor access the very same memory array. Thus, time and power are no longer incurred for data synchronization.

3.3 Arithmetic / Logic Operations

GP-SIMD can implement a wide range of arithmetic and logic processing tasks. Consider a workload using two datasets, A and B, each containing N elements, where each element is m bits wide. These vectors are mapped into the GP-SIMD memory array such that two m bit adjacent column-groups hold vectors A and B. Assume that we need to add the two vectors and place the results into $m+1$ bit column-group S, as illustrated in Figure 6 (where $m=4$). The addition is performed in m single-bit addition steps:

$$c[*] \mid s[*]_i = a[*]_i + b[*]_i + c[*] \quad \forall\, i = 0, \dots, m-1 \tag{1}$$

where i is the bit index and '$*$' is the vector index (corresponding to a PU and memory row). A bitwise vector addition of datasets having four elements ($N=4$) of four bits word-length ($m=4$) is shown in Figure 7. Six cycles are demonstrated in six sub-figures. Each sub-figure illustrates the PU registers RA, RB, RC and the memory array with the operands A, B and the output S. In cycle 1, column $A[0]$ is copied into RA and RC is reset to all zeroes. In cycle 2, column $B[0]$ is copied into RB. In cycle 3, RA and RB are added; the sum is written into RB and the carry replaces RC. In addition, column $A[1]$ is read into RA. In cycle 4, the sum in RB is copied to $S[0]$. In cycle 5, column $B[1]$ is read into RB. This process repeats in subsequent cycles. Since addition is carried out simultaneously for all vector elements, fixed point m bit addition consumes $3m \in O(m)$ cycles, independent of the size of the vectors N.

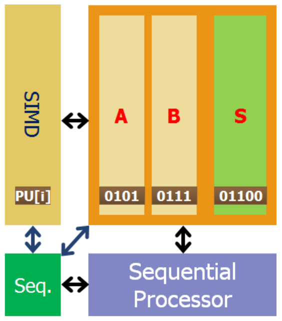


Figure 6. Memory array containing three operands (*m=4*)

Using the same logic, subtracting or performing logic AND, OR, XOR via the function generator on the two operand sets entails $O(m)$ cycles as well. Note the contrast with CSIMD architectures having k PUs where $k<<N$, such as 2-16 PUs SIMD accelerators in CPUs [6]; they require $O(N/k) \in O(N)$ cycles to add N data elements. *Compare* operation between the two sets A and B entails $2m \in O(m)$ cycles. *Compare immediate* operation between set A and a fixed word sourced from the sequential processor requires only $1m \in O(m)$ cycles since the second operand is sourced from the sequencer, not from the memory array.

Fixed point multiplication and division in GP-SIMD are also implemented bit-serially but word-parallel, consisting of a series of add-shift and subtract-shift vector operations. Shift is implemented by appropriate column addressing and therefore requires no extra cycles. Thus, fixed point $m \times m$ *bit* vector multiplication requires $3m * m \in O(m^2)$ cycles, regardless of the vector size, N. Floating point arithmetic for GP-SIMD is somewhat more complex to implement. Different exponents require shifting mantissas by different lengths, resulting in a sequence of bit-serial vector operations. IEEE single precision floating point vector multiplication takes close to 2500 cycles, regardless of the length of the dataset, N.

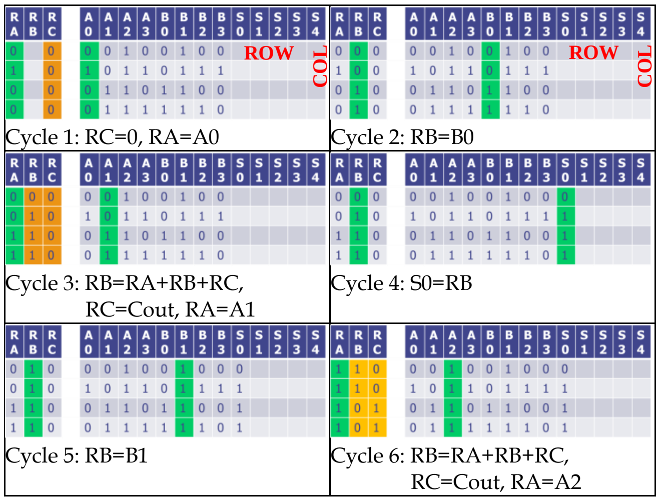

ROW
COL
Cycle 1: RC=0, RA=A0
Cycle 2: RB=B0
Cycle 3: RB=RA+RB+RC, RC=Cout, RA=A1
Cycle 4: S0=RB
Cycle 5: RB=B1
Cycle 6: RB=RA+RB+RC, RC=Cout, RA=A2

Figure 7. Addition, cycle by cycle.

3.4 Associative Operations

GP-SIMD, besides being a massively parallel SIMD accelerator, can implement classical CAM operations such as associative search, sorting and ordering. The CAM allows comparing all data words to a key, tagging the matching words, and possibly reading some or all tagged words one by one. Consider a large vector, where each element is m bits wide, illustrated by column A in Figure 6. The Sequential processor wishes to find all elements in vector A matching a certain *Key* of m bits, and reset the matched values of A (that is, $A[i \mid A[i]{=}{=}Key]=0$). The sequential CPU issues a *compare immediate* of *Key* on column A, storing the single bit-slice compare results output in register *RD*. At this point, register *RD* has logic *one* in all rows where A matches the *Key* and *zero* elsewhere. Next, a masked write is performed by the sequential processor *only* to flagged rows of the memory array. To that end, the output of *RD* enables writing of each memory row (*RD_INST* bus is set to *'10'*), and the sequential processor writes '0' to the A column of the memory array. Only the matching rows are enabled for writing, and the A values of only these rows are reset. Elsewhere, in non-matching rows, the A values are left unaffected.

Content-addressable access is achieved as follows. Assume that the memory array contains a vector of unique indices (A), adjacent to a vector of data (B). Comparing vector A with a key, followed by setting the *RD_INST* Bus to '10' while issuing *read* to the memory array, allows the sequential processor to fetch a single value of vector B, corresponding to the row in which vector A matched the *Key* (that is, Output=$B[i \mid A[i]{=}{=}Key]$). When multiple rows match the key, the values must be read one by one.

A portion of GP-SIMD memory grid may be programed to mimic bit-serial TCAM. Wildcard functionality can be achieved by storing a bit and its complement in adjacent columns of a row, while programing the SIMD to *OR* the output of the results of two separate compare operation performed on each bits against a key. Searching with an *immediate* wildcard is accomplished by simply skipping the relevant columns.

3.5 Interconnection Network

Arithmetic, logic and associative operations presented in the previous sub-sections require the relevant operands to be available at the memory row of the PU. However, common workloads require inter-PU data communications. Depending on the workload, communication requirements may vary from no communication (for "embarrassingly parallel" tasks such as Black-Scholes option pricing) to relatively intense communications (e.g., for FFT). In some cases, support for special pre-defined communication patterns or permutations can be useful (e.g., the permutations required for FFT). A dedicated interconnect is employed to allow all PUs to communicate in parallel.

Since GP-SIMD processing operation is mainly bitwise, the interconnection can be a relatively simple circuit-switched network. An example of an efficient network is a logarithmic ±k nearest neighbor, forming N-bit shift register. Assuming each PU has a single bit direct access to its $\pm Y$ neighbors, where $Y \in \{1,2,4,\dots,log_2 N\}$, transferring in parallel an entire vector of N rows (a slice of the shared array) by H rows up/down entails a maximum of $O(m + m\log_2(H))$ cycles, independent of the vector size, N. Note that if $H \subset Y$, the transfer time entails $O(2m) \in O(m)$ cycles. For instance, Figure 8 shows a ± 8 interconnect. Shifting an entire vector of any length (up to the entire memory) by 32 rows upwards requires $4 \cdot 2m$ cycles.

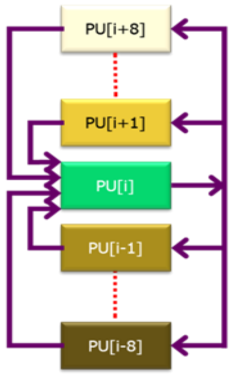


Figure 8. GP-SIMD ±k Nearest Neighbor Interconnection Network: Each PU is interconnected with nearest neighbors $\pm1, \pm2, \dots, \pm8$

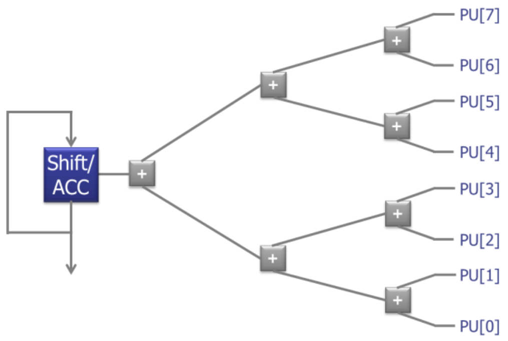


Figure 9. Hardware reduction tree, illustrated for eight PUs

3.6 Reduction Tree

A common reduction operation sums up a large array of values. Other common reduction operations are minimum or maximum. Reduction Tree ([49], earlier introduced as a 'response counter' in [65]) is an adder tree, enabling bit-serial parallel summation of PU values.

Consider a vector A of N fixed point m-bit elements, as illustrated in Figure 6. Further, consider a hardware reduction tree implemented using a pipelined binary adder tree. The first level of the tree sums two single bits from two adjacent PUs. Following $\log_2 N$ levels, the scalar sum of the entire array becomes available, as illustrated in Figure 9. The fixed precision summation of vector A entails reading a single column slice of vector A, LSB first, and summing this column via the reduction tree. The addition is carried out simultaneously for all vector elements, column-slice at a time until all m columns have been processed. The m outputs of the adder tree are summed together using an accumulator, where in each summation loop, the output of the adder tree is shifted left corresponding to the bit location being processed. Therefore, fixed point m bit reduction entails $O(m + \log_2(N) + 1) \in O(m + \log_2(N))$ cycles.

The reduction tree can also be implemented in software using the PUs and the interconnection network. Consider for example a vector A of eight fixed point 7-bit elements, as in Figure 10. Vectors T, S contain partial sums. Similarly to the hardware implementation, the parallel binary tree is implemented as follows: In the first (initialization) step, vector A is copied onto S. In the second step, vector S is shifted one row up and stored in vector T, followed by adding vectors T and S. The oddly addressed elements of vector S contain the results of the first step of the binary tree addition. In the third step, vector S is shifted two rows up and stored in vector T, followed by adding vectors T and S. Vector S now contains the results of the second level of the addition tree. In the fourth step, vector S is shifted four rows up and stored in vector T, followed by adding vectors T and S. Vector S now contains the results of the third level of the tree, and the sum of all elements of A appears in the first element of S.

A	T	S
1		1
2		2
4		4
8		8
16		16
32		32
64		64

Step-1:
S=A

A	T	S
1	2	3
2	4	6
4	8	12
8	16	24
16	32	48
32	64	96
64	0	64

Step-2:
T=S↑1; S=S+T

A	T	S
1	12	15
2	24	30
4	48	60
8	96	120
16	64	112
32	0	96
64	0	64

Step-3:
T=S↑2; S=S+T

A	T	S
1	112	127
2	96	126
4	64	124
8	0	120
16	0	112
32	0	96
64	0	64

Step-4:
T=S↑4; S=S+T

Figure 10. Software reduction tree, illustrated for eight PUs

In general, given a $\pm\log_2 N$ interconnection network, fixed point m-bit reduction entails $O(2\mathrm{m} + (2m + 3m)\log_2(N)) \in O(m\log_2(N))$ cycles. In case of large datasets and limited interconnection networks, the last few shifting and summation steps can be executed serially by the sequential processor, so as to save processing time and power.

3.7 Circuit Implementation

Algorithmic computation necessitates storage of temporary variables. Each GP-SIMD's PU utilizes a memory row as its register file for storage of memory variables and temporaries. The width of the memory array is likely to be much higher than the word width of the sequential processor (for example the sequential processor may have 64-bit word-length, while the SIMD may need to address 256 distinct columns). A typical GP-SIMD memory array is depicted in Figure 11. Two implementations are proposed: a single memory array, which contains the shared and SIMD-only sections, and a dual memory array, in which the SIMD-Only and the shared memory sections are separate.

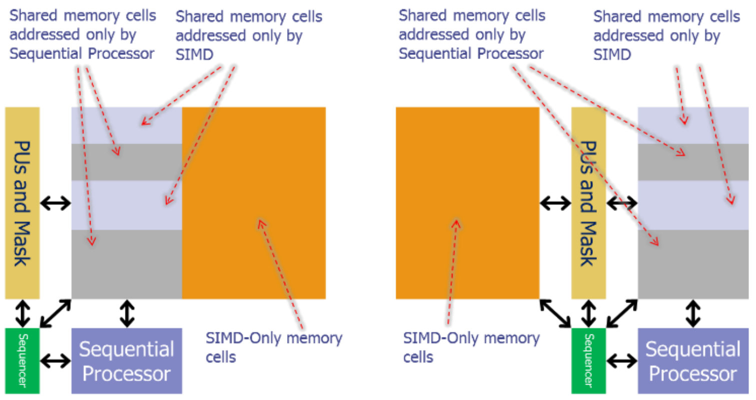

Shared memory cells addressed only by Sequential Processor
Shared memory cells addressed only by SIMD
PUs and Mask
Sequencer
Sequential Processor
SIMD-Only memory cells
Shared memory cells addressed only by Sequential Processor
Shared memory cells addressed only by SIMD
SIMD-Only memory cells
PUs and Mask
Sequencer
Sequential Processor

Figure 11. GP-SIMD memory segmentation. Single memory array (a) and dual memory array (b).

Consider the single memory array of Figure 11(a). To enable 2D access (the sequential processor accessing by words and the SIMD co-processor accessing by columns), two types of cells are proposed. A shared-memory cell consisting of a 7-transistor SRAM bit is used in the shared columns (Figure 12), and a SIMD-Only cell using a 5-transistor cell is used in the SIMD-only columns (Figure 13).

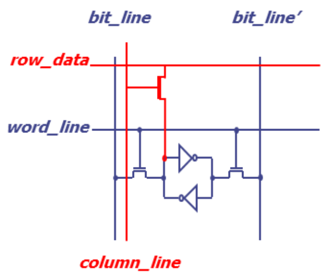


Figure 12. Shared SRAM cell

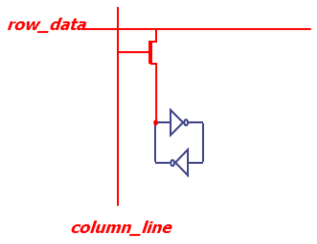


Figure 13. SIMD Only SRAM cell

The standard 6T SRAM bit cell (blue in Figure 12) is amended by a pass-gate to enable column read/write access (red). In the SIMD-only static bit cell (Figure 13), a 4T latch (blue) is connected by a pass-gate (red) to enable column read/write access. The *bit_line, bit_line_not*, *word_line* and associated pass-gates of Figure 12 are eliminated in Figure 13 as the sequential processor does not address these cells. Reading and writing to the two dimensional memory array is performed as follows:

— By sequential processor: to read data from memory, the *bit_line* and *bit_line-not* lines are pre-charged, the *word_line* is asserted, and the bit lines are sensed. To write data to the memory, the *bit_line*, *bit_line-not* and the *word_line* are asserted.
— By SIMD co-processor: to read data from memory, the *column_line* is asserted, and the *row_data* is pre-charged. To write data to the memory, the *column_line* and the *row_data* are asserted.

When the width of the memory is sufficiently small (e.g., 256 columns), a single pass-gate transistor is tied to the row data. In wider arrays, a differential pair, namely, *row_data* and *row_data-not* and an additional pass-gate transistor may be required. GP-SIMD's 6T and 5T SRAM cell outweigh the AP 12T cell of [62] both area and power-wise. For the same die area, GP-SIMD array thus packs nearly twice the number of memory rows, and offers twice the AP performance for the same die area.

The memory architecture depicted in Figure 11(a) allows the sequential processor and the SIMD co-processor to concurrently read from and write to the memory array. Further, the architecture inherently allows segmentation of the memory array along both dimensions, so as to avoid conflict and allow processing on parts of the data, as follows:

— Segmentation in the horizontal dimension: since the memory width is larger than sequential processor data-bus (for example, 64 bit data-bus and 256 bit wide array), the SIMD co-processor may be programmed to read from the entire memory array, but write only to the SIMD-Only section, while the sequential processor continues to process the shared memory space.
— Segmentation along the vertical dimension: *RD* can be used to mask the *word_line* of certain rows.
— Full segmentation: Access by the sequential processor and the SIMD co-processor may be interleaved, to avoid all interference.

Since the number of memory rows in a typical GP-SIMD implementation could be quite high (1M+), the GP-SIMD array can be partitioned into n smaller blocks each having N/n processors (and N/n complete memory rows). Each block could have, for example, its own sequencer. In such a case, all blocks share a piped version of the bus to the sequential processor. A potential layout of such an implementation is depicted in Figure 14, in which a GP-SIMD array is partitioned into nine smaller blocks. Note the connectivity simplicity: the interconnection network is chained (red), and the sequential processor memory bus (blue) is connected to all blocks in parallel. Note that since the sequential processor accesses no more than a single memory row at a time, such access involves only a single GP-SIMD block and the path to it; only that path is activated to conserve power. Reading data from a GP-SIMD block could entail a few cycles due the long path the signal may have to transverse. To mitigate such access delays, the sequential processor may include a data cache.

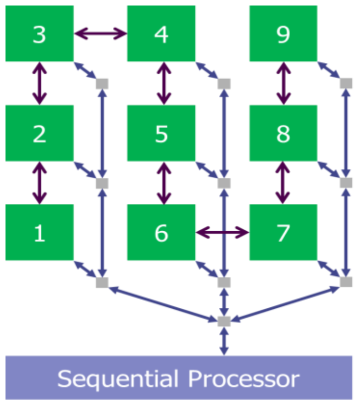


Figure 14. GP-SIMD array partitioning.

3.8 GP-SIMD Performance Summary

Consider a data set having two m-bit N-element vectors A and B. TABLE 2 summarizes the arithmetic/logic performance of the GP-SIMD processor, as analyzed in the previous sections. Note that the AP's arithmetic/logic performance [62] is 2.5-5× lower than GP-SIMD's arithmetic performance, regardless of the number of processing elements, N.

TABLE 2
Arithmetic/Logic performance

Command	Performance (cycles)
Add/Sub(A, B)	$O(3m)$
Add/Sub(A, Immediate)	$O(2m)$
Mult/Div(A, B)	$O(3m^2)$
Mult/Div(A, Immediate)	$O(2m^2)$
Cmp(A,B)	$O(2m)$
Cmp(A, Immediate)	$O(m)$
AND/OR/XOR	$O(2m)$
AND/OR/XOR(A, Immediate)	$O(m)$
Invert(A)	$O(2m)$
FP Add/Sub/Mult(A, B)	2500
HW Reduction tree	$O(m + log_2(N))$
SW Reduction tree	$O(5m\,log_2(N))$

4. ANALYTIC MODEL AND COMPARATIVE ANALYSIS

In this section, we provide an analytical performance and power consumption model of the CSIMD, AP and GP-SIMD and compare their relative performance, area and power consumption under constrained area and power resources. For the purpose of modeling, we assume that the sequential processor is identical in all architectures, thus we focus on concurrent workloads. We further assume that the CSIMD executes a single fused multiply operation in 1.5GHz, while the GP-SIMD and AP employ faster clocks of 2.5GHz, as they execute bitwise rather than word-parallel instructions and thus have shorter critical paths.

Consider a workload having WL single cycle instructions (i.e., arithmetic, control, register file access and alike). The workload consists of N fine grain concurrent execution segments, requiring $T_{Ref} = WL * 1$ cycles to execute on a baseline *reference* floating point engine capable of performing 1 FLOP/Cycle (single precision floating point operation per cycle). Such floating point engine consumes an area of 0.01 mm^2 in 45nm [44] and dissipates 10mW [31]. With 22nm process technology, the same floating point engine would consume an area of 0.003 mm^2 and dissipate roughly 5mW [15][31]. These figures pertain to a baseline floating point engine, without the accompanying connectivity logic, register file and the like.

A portion of the workload instructions, C_{Sync}, is dedicated to data synchronization before and after each segment starts and ends respectively, while a further portion of the workload, C_{Inter}, is dedicated to data exchange among the execution units. The remaining portion of the workload, C_{Proc}, is actual processing. We thus have:

$$C_{Proc} + C_{Sync} + C_{Inter} = 1 \tag{2}$$

In a large design comprising thousands of PUs, it is impractical to consider a complex network topology that would consume super-linear area. We thus assume a linear complexity ring topology interconnection network in all three architectures, where in CSIMD the network is 32bit wide, and in GP-SIMD and AP, the network is 1bit wide. In a ring topology, within *O(N)* serial steps, *N* PUs can exchange data according to any permutation.

4.1 Conventional SIMD Processor

A Conventional SIMD (CSIMD) co-processor is depicted on the left side of Figure 2. The CSIMD coprocessor contains n_{CSIMD} baseline PUs, each containing a floating point unit and a register file. The PUs are interconnected using an interconnection network. The PUs are also connected to the sequential processor shown on the right, through a bandwidth-limited interface (denoted by the red bus). *Synchronization* is moving data from the sequential processor to the CSIMD coprocessor before the parallel segment begins, and back from CSIMD after the parallel segment completes. Since it involves access to a shared resource, $T_{Sync-CSIMD}$ might depend on the number of PUs [20][51]. The synchronization time [63] can be expressed as follows:

$$T_{Sync-CSIMD} = \frac{C_{Sync}WL}{BW} \tag{3}$$

where C_{Sync} denotes the fraction of the workload that corresponding to synchronization overhead, and *BW* denotes the bandwidth (words per second) of the interface. The execution time of the workload on the CSIMD co-processor is thus:

$$T_{CSIMD} = \frac{C_{Proc}WL}{n_{CSIMD}} + C_{Inter}WL + T_{Sync-CSIMD} \tag{4}$$

where n_{CSIMD} is the number of CSIMD's Pus. The speedup of the CSIMD processor over the sequential CPU can be written as follows:

$$S_{CSIMD} = \frac{T_{Ref}}{T_{CSIMD}} = \frac{n_{CSIMD}}{C_{Proc} + n_{CSIMD}\left[C_{Inter} + \frac{C_{Sync}}{BW}\right]} \tag{5}$$

The area of the CSIMD processor can be presented as follows:

$$A_{CSIMD} = n_{CSIMD}(A_{ALU} + A_{RF}) \tag{6}$$

where A_{ALU} is the ALU area and A_{RF} is the register file area. The inter-PU connection network is omitted for simplicity. For efficient comparison between PU and memory areas, we represent all area values (ALU, registers, memory) in terms of baseline SRAM cell area. Let the baseline SRAM cell area be 1. In 22nm CMOS technology, the actual figure is in the range of $0.1\mu m^2$ [7]. Then we can write:

$$\begin{aligned} A_{ALU} &= A_{CSIMDALUo} m^2 \\ A_{RF} &= A_{CSIMDRFo} km \end{aligned} \tag{7}$$

where $A_{CSIMDALUo}$ is the area of a single bit of a high speed parallel ALU and $A_{CSIMDRFo}$ is the area of a register bit (a flip-flop), both measured in baseline SRAM cell area units; m is data wordlength and k is the size of the register file. This model is quite simplistic and does not take into account numerous aspects of CSIMD design (instruction cache, communication and control, etc.). Its purpose is providing the best case reference figures for the comparative analysis of the CSIMD processor's speedup, area and power. The average power of the CSIMD processor can be written as follows:

$$\begin{aligned} P_{CSIMD} = \frac{1}{T_{CSIMD}} \Big[& \frac{C_{Proc} WL}{n_{CSIMD}} \cdot P_{Proc-CSIMD} + C_{Inter} WL \cdot P_{Inter-CSIMD} \\ & + \frac{C_{Sync} WL}{BW} P_{Sync-CSIMD} \Big] + P_{Leak-CSIMD} \end{aligned} \tag{8}$$

where $P_{Proc-CSIMD}$ is the average power consumed during a single processing operation; P_{Inter} is the average power consumed during inter-PU communication; $P_{Sync-CSIMD}$ is the average power consumed during synchronization, and $P_{Leak-CSIMD}$ is the leakage power. Just as in the case of area comparison, we represent all power values (ALU, registers, memory) through the write power consumption of a baseline SRAM memory cell. Let the power consumption of the baseline SRAM cell during write from '0' to '1' or from '1' to '0' be 1. In 22nm CMOS technology, the actual figure is in the range of $1\mu W$ at 4GHz [28]. Then we can further write the CSIMD power consumption as follows:

$$\begin{aligned} P_{Proc-CSIMD} &= n_{CSIMD} (P_{CSIMDALUo} m^2 + P_{CSIMDRFo} km) \\ P_{Inter-CSIMD} &= n_{CSIMD} P_{INTERo} m \\ P_{Sync-CSIMD} &= P_{SYNCo} m \end{aligned} \tag{9}$$

where $P_{CSIMDALUo}$ and $P_{CSIMDRFo}$ are the average per-bit power consumptions of the ALU and RF respectively during computation. P_{INTERo} is the per-bit power consumption during the inter-PU communication. P_{SYNCo} is the per-bit power consumed during synchronization. $P_{Proc-CSIMD}$, $P_{Inter-CSIMD}$ and $P_{Sync-CSIMD}$ are measured in SRAM cell write power consumption units. Leakage power can be expressed as follows:

$$P_{Leam-CSIMD} = \beta A V^{\alpha} = \gamma A_{CSIMD} \tag{10}$$

where A is the area, V is the supply voltage, α and β are constants, and γ is the leakage area coefficient that depends on silicon process and operating conditions.

4.2 Associative Processor

In this section we detail the analytical model for the speedup, area and power consumption of the AP. In a similar manner to GP-SIMD, the AP [62] does not entail data synchronization unless the entire data set does not fit in the AP's memory array. Thus:

$$T_{Sync-AP} = \begin{cases} 0, & N \le n_{AP} \\ \dfrac{C_{Sync} WL}{BW} \dfrac{(N - n_{AP})}{N}, & N > n_{AP} \end{cases} \tag{11}$$

The execution time of the concurrent portion of the workload can be written as follows:

$$T_{AP} = \frac{C_{Proc} WL}{S_{APe} n_{AP}} + C_{Inter} WLm + T_{Sync-AP} \tag{12}$$

where n_{AP} is the number of PUs in the AP, S_{APe} is the speedup of associative PU relative to the CSIMD's PU. Lacking *a-priori* knowledge of the workloads to be executed on the AP, we assume the worst case scenario comprising a continuous series of single precision floating point multiplications, which in one direct implementation takes 8800 cycles *vs.* a single cycle on the baseline CSIMD's PU. In this case $S_{APE} = 1/8800$. The speedup of the AP can then be written as follows:

$$S_{AP} = \frac{T_{Ref}}{T_{AP}} = \frac{n_{AP}}{\frac{C_{Proc}}{S_{APe}} + n_{AP}\left[C_{Inter}m + \frac{T_{Sync-AP}}{WL}\right]} \tag{13}$$

The area of the AP can be written as follows:

$$A_{AP} = n_{AP}(A_{APTAGo} + A_{APo}km + 2A_{ALUo}) \tag{14}$$

where k is the width of the associative processor memory array (in m-bit data words) reserved for temporary storage, A_{APTAGo} is the AP TAG cell area, A_{APo} is the AP memory cell area, both measured in SRAM cell area units, and $2A_{ALUo}$ is the per-PU hardware reduction tree size, assuming the adders of each tree level are of growing wordlength [62]. Similarly to the CSIMD coprocessor, we ignore the area of the interconnection network for simplicity.
The average power of the AP can be written as follows:

$$P_{AP} = \frac{1}{T_{AP}}\left[\frac{C_{Proc}WL}{S_{APe}n_{AP}} \cdot P_{Proc-AP} + C_{Inter}WL \cdot P_{Inter-AP} + \frac{C_{Sync}WL}{BW}P_{Sync-AP}\right] + P_{Leak-AP} \tag{15}$$

where $P_{Proc-AP}$ is the average power consumed during a single processing operation; $P_{Inter-AP}$ is the average power consumed during inter-PU communication; $P_{Sync-AP}$ is the average power consumed during synchronization when the entire data set does not fit in the AP, and $P_{Leak-AP}$ is the leakage power. In a similar manner to the previous section, we can further write the AP power consumption as follows:

$$\begin{gathered} P_{Proc-AP} = n_{AP}P_{APo} \\ P_{Inter-AP} = n_{AP}P_{INTERo}m \\ P_{Sync-AP} = \begin{cases} 0, & N \le n_{AP} \\ P_{SYNCo}m\frac{(N-n_{AP})}{N}, & N > n_{AP} \end{cases} \end{gathered} \tag{16}$$

where P_{APo} is the average per-bit power consumptions of the AP during computation. $P_{INTER-o}$ is the per-bit power consumption during the inter-PU communication. P_{SYNCo} is the per-bit power consumed during synchronization. $P_{Proc-AP}$, $P_{Inter-AP}$ and $P_{Sync-AP}$ are measured in SRAM cell write power consumption units. Leakage power can be expressed as follows:

$$P_{Leak-AP} = \gamma A_{AP} \tag{17}$$

Note that for comparison purposes we use the same leakage power formulation (represented as a function of area only as in (10)) for both the AP and the CSIMD processor. This may overestimate AP power somewhat: First, the leakage power per area could be lower for memory than for logic [45]. Second, APs have fewer hotspots [64]. Since the leakage power is highly temperature dependent, hotspots may lead to higher leakage in the CSIMD processor [8]. A detailed analysis of P_{APo} can be found in [62].

4.3 GP-SIMD Processor

In this section we detail the analytical model for the speedup, area and power consumption of the GP-SIMD. As discussed in section 3, the GP-SIMD does not entail data synchronization unless the entire data set does not fit in the GP-SIMD's memory array. Thus:

$$T_{Sync-GPSIMD} = \begin{cases} 0, & N \leq n_{GPSIMD} \\ \dfrac{C_{Sync}WL}{BW}\dfrac{(N-n_{AP})}{N}, & N > n_{GPSIMD} \end{cases} \tag{18}$$

The execution time of the concurrent portion of the workload can be written as follows:

$$T_{GPSIMD} = \frac{C_{Proc}WL}{S_{GPSIMDe}n_{GPSIMD}} + C_{Inter}WLm + T_{Sync-GPSIMD} \tag{19}$$

where n_{AP} is the number of PUs in the GP-SIMD, $S_{GP-SIMDe}$ is the speedup of associative PU relative to the CSIMD's PU. Similar to the AP analysis, we assume the worst case scenario comprising a continuous series of single precision floating point multiplications, which in one direct implementation takes 2500 cycles *vs.* 1 cycle on the baseline CSIMD's PU. In this case $S_{GP-SIMDe} = 1/2500$. The speedup of the GP-SIMD can then be written as follows:

$$S_{GPSIMD} = \frac{T_{Ref}}{T_{GPSIMD}} = \frac{n_{GPSIMD}}{C_{Proc}/S_{GPSIMDe} + n_{GPSIMD}\left[C_{Inter}m + \dfrac{T_{Sync-GPSIMD}}{WL}\right]} \tag{20}$$

Following section 3.7, we assume the GP-SIMD's sequential processor accesses only the first L bits of the memory array. Thus, the average size of the memory bit cell is as follows:

$$A_{GPSIMD-CELLo} = \begin{cases} A_{SHAREDo}, & km \leq L \\ \dfrac{A_{SHAREDo}L + A_{SIMDONLYo}(km-L)}{km}, & km > L \end{cases} \tag{21}$$

where k is the width of the GP-SIMD memory array (in m-bit data words) reserved for temporary storage, $A_{SHAREDo}$ is the shared memory cell area and $A_{SIMDONLYo}$ is the SIMD-only memory cell area, both measured in SRAM cell area units. The total area of the GP-SIMD can be written as follows:

$$A_{GPSIMD} = n_{GPSIMD}(A_{GPSIMDPUo} + A_{GPSIMD-CELL}km + 2A_{ALUo}) \tag{22}$$

where $A_{GPSIMDPUo}$ is the GP-SIMD processing unit cell area measured in SRAM cell area units, and $2A_{ALUo}$ is the per-PU hardware reduction tree size. Similarly to the CSIMD coprocessor and the AP, we ignore the area of the interconnection network for simplicity.

The average power of the GP-SIMD can be written as follows:

$$P_{GPSIMD} = \frac{1}{T_{GPSIMD}}\left[\frac{C_{Proc}WL}{S_{GPSIMDe}n_{GPSIMD}} \cdot P_{Proc-GPSIMD} + C_{Inter}WL \cdot P_{Inter-GPSIMD} + \frac{C_{Sync}WL}{BW}P_{Sync-GPSIMD}\right] + P_{Leak-GPSIMD} \tag{23}$$

where $P_{Proc-GPSIMD}$ is the average power consumed during a single processing operation; $P_{Inter-GPSIMD}$ is the average power consumed during inter-PU communication; $P_{Sync-GPSIMD}$ is the average power consumed during synchronization when the entire data set does not fit in the AP, and $P_{Leak-GPSIMD}$ is the leakage power. In a similar manner to the previous section, we can further write the GP-SIMD power consumption as follows:

$$\begin{gathered} P_{Proc-GPSIMD} = n_{GPSIMD}P_{GPSIMDo} \\ P_{Inter-GPSIMD} = n_{GPSIMD}P_{INTERo}m \\ P_{Sync-GPSIMD} = \begin{cases} 0, & N \leq n_{GPSIMD} \\ P_{SYNCo}m, & N > n_{GPSIMD} \end{cases} \end{gathered} \tag{24}$$

where $P_{GPSIMDo}$ is the average per-bit power consumptions of the AP during computation. P_{INTERo} is the per-bit power consumption during the inter-PU communication. P_{SYNCo} is the per-bit power consumed during synchronization. $P_{Proc-GPSIMD}$, $P_{Inter-GPSIMD}$ and $P_{Sync-GPSIMD}$ are measured in SRAM cell write power consumption units. Leakage power can be expressed as follows:

$$P_{Leak-GPSIMD} = \gamma A_{GPSIMD} \tag{25}$$

Note that for comparison purposes we use the same leakage power formulation (represented as a function of area only as in (10)) for the CSIMD, AP and GP-SIMD architectures. This might be somewhat unfair to the GP-SIMD: First, the leakage power per area could be lower for memory than for logic [45]. Second, GP-SIMD, in a similar manner to AP, should have fewer hotspots [64]. Since the leakage power is highly temperature dependent, hotspots may lead to higher leakage in the CSIMD processor [8].

4.4 Performance under constrained area

Consider a given die total area, A_{Total}. The number of the CSIMD, AP and the GP-SIMD processing units can be found as follows:

$$\begin{aligned} n_{CSIMD} &= \frac{A_{Total}}{A_{ALUo}m^2 + A_{RFo}km} \\ n_{AP} &= \frac{A_{Total}}{A_{APTAGo} + A_{APo}km + 2A_{ALUo}} \\ n_{GPSIMD} &= \frac{A_{Total}}{A_{GPSIMDPUo} + A_{GPSIMD-CELL}km + 2A_{ALUo}} \end{aligned} \tag{26}$$

Substituting n_{SIMD}, n_{AP} and n_{GPSIMD} into the equations listed in the previous sub-sections, we determine the speedup and the power equations of each architecture. The area and power parameters we use for modeling purposes are listed in TABLE 3.

TABLE 3
Model Parameters

Parameter	Description	Attributed to	Value
$A_{CSIMDALUo}$	FP ALU bit cell area	CSIMD	40 (1)
$P_{CSIMDALUo}$	FP ALU bit cell power	CSIMD	40 (1)
A_{ALUo}	ALU bit cell area	All	10 (1)
P_{ALUo}	ALU bit cell power	All	10 (1)
$A_{CSIMDRFo}$	Register bit (FF) area	CSIMD	3 (1)
$P_{CSIMDRFo}$	Register bit (FF) power	CSIMD	3 (1)
S_{APe}	AP speedup relative to sequential CPU	AP	1/8800
A_{APo}	AP bit cell area	AP	2 (1)
P_{APo}	AP bit cell power	AP	4 (1)
A_{APTAGo}	AP TAG cell area	AP	1 (1)
$S_{GPSIMDe}$	GP-SIMD speedup relative to sequential CPU	GP-SIMD	1/2500
$A_{SHAREDo}$	GP-SIMD Shard bit cell area	GP-SIMD	7/6 (1)
$A_{SIMDONLYo}$	GP-SIMD SIMD-Only cell area	GP-SIMD	5/6 (1)
$A_{GPSIMDPUo}$	GP-SIMD PU area	GP-SIMD	10 (1)
$P_{GPSIMDPUo}$	GP-SIMD PU power	GP-SIMD	10 (1)
P_{INTERo}	GP-SIMD Intercommunication power	All	200 (1)
P_{SYNCo}	GP-SIMD Synchronization power	All	200 (1)
m	Data wordlength	All	32
k	Register file size (in 32-bit words)	All	8
L	Sequential processor wordlength	GP-SIMD	64
γ	Static power coefficient	All	50mW/mm^2
C_{Sync}	Workload portion dedicated for Synchronization	All	3%
C_{Inter}	Workload portion dedicated for inter-core communication	All	0%

(1) *Area and power parameters are relative to the area and power of SRAM bit cell respectively; the values are based on typical standard cell libraries.*

Speedup *vs.* area for the CSIMD coprocessor, the AP and the GP-SIMD is shown in Figure 15(a). For example, at $2mm^2$ about 500 reference floating point engines are enabled (including registers and other logic besides the ALUs) in the CSIMD processor, and dissipate about 8W. The portion of the workload dedicated for synchronization C_{Sync}, is assumed to be 0.03 (namely, synchronization takes 3% of the workload). Note that 3% is quite conservative, as for certain workloads such as DMM (Dense Matrix Multiplication), the CPU-GPU synchronization may consume 20% of the processing time [60].

As the area budget increases, the speedup of the CSIMD coprocessor exhibits diminishing returns caused by synchronization, because the size of the data set grows while the CPU⇔SIMD bandwidth remains fixed. After sufficient area becomes available the speedup saturates. For smaller total area, the speedups of the AP and GP-SIMD are lower than the speedup of the CSIMD coprocessor. The breakeven point lies around $14mm^2$ for the GP-SIMD. Diminishing returns affect the GP-SIMD and AP speedups to a lesser extent than in the CSIMD, since they only occur when the data set does not fit into the internal memory array. Following [62], to emphasize this effect, and for illustrative purpose only, we assume that the data set size grows with the GP-SIMD and AP size respectively, and up until $A_{Total} = 25mm^2$ (at which point, $n_{GPSIMD} = 1.0 \times 10^6$ and $n_{AP} = 5.3 \times 10^5$) the GP-SIMD and AP internal memory array have just enough space to hold all necessary program variables. However, from that point on, both processors will have to exchange additional data with, for example, an external DRAM. We assumed that for each additional processing unit beyond $A_{Total} = 25mm^2$, a single 32b data-word per 300 additional processing units will incur synchronization delay. These assumptions cause the GP-SIMD and AP speedup to saturate as well. Note that without this assumption, both GP-SIMD and AP Speedup would have continued to climb linearly with area.

The results show that at $A_{Total} = 25mm^2$ (just prior to saturation), the GP-SIMD provides speedup of 5.5× over the CSIMD, and 7× over the AP. The speedup factor over CSIMD continues to grow till the GP-SIMD saturates at $A_{Total} = 40mm^2$.

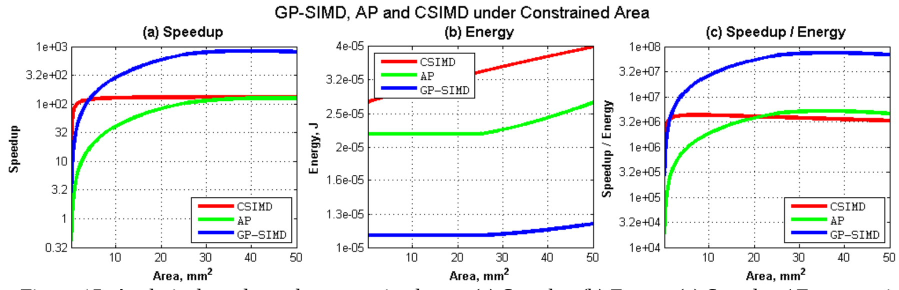


Figure 15. Analytical results under constrained area: (a) Speedup (b) Energy (c) Speedup / Energy ratio

Energy *vs.* area is shown in Figure 15 (b). Energy of both GP-SIMD and AP is lower than that of the CSIMD processor. Note that even when the speedups saturate, energy continues to grow with area, due to addition of processing units and leakage. The speedup/energy ratio *vs.* area is shown in Figure 15(c). For the CSIMD processor, the speedup/energy ratio eventually drops because the speedup saturates while energy continues to grow with increasing area. As CSIMD's speedup saturates, both GP-SIMD and AP yield better speedup/energy ratios. With further area growth, the speedup/energy ratio of both the GP-SIMD and AP drop due to assumed saturation.

In general, GP-SIMD achieves higher performance than CSIMD due to CSIMD stalling on synchronization. GP-SIMD achieves higher performance-per-area than AP due to its lower memory cell area (~2x), allowing the integration of twice the amount of PUs and memory rows per a given area, and its faster floating point implementation (~3.5x) due to GP-SIMD arithmetic PU *vs.* associative-only PU of AP.

4.5 GP-SIMD Cycle-Accurate Simulation

We verify our analytical model findings using a home-grown cycle-accurate simulator of GP-SIMD. Following [63] and [62], four workloads have been selected for performance and energy consumption simulations:

— N-point Fast Fourier Transform (FFT)
— N-option pairs Black-Scholes option pricing (BSC)
— N-point Vector Reduction (VR)
— Dense Matrix Multiplication of two $\sqrt{N}\times\sqrt{N}$ matrices (DMM)

where N is the data set size, scaled for simplicity to the processor size (following the methodology suggested in [24] and [62]), i.e. $N = n_{GPSIMD}$. Note that simulations do not cover the cases where the data size exceeds the size of the processor (requiring data synchronization).

The workloads are hand-coded for the simulations. For FFT, we use optimized parallel implementation outlined in [50]. For N point FFT, the sequential computing time is $O(N \log N)$. For N PU parallel implementation, computing time is reduced to $\mathrm{O}(\log N)$. Following each computing step, N intermediate results need to be exchanged among the PUs. We thus assign each multiply-accumulate operation to a single PU. For Black-Scholes (BSC), we used a direct implementation, based on formulation in [11]. With sequential computing time of $O(N)$, Black and Scholes option pricing requires no interaction between separate option calculations (Black and Scholes option pricing is an example of an 'embarrassingly parallel' task). We thus assign a single PU to handle a single call option of a single security at a single strike price and a single expiration time. Vector reduction (VR) is implemented using the software reduction tree where a single PU retains a single vector element. For $\sqrt{N} \times \sqrt{N}$ dense matrix multiplication (DMM), the sequential execution time is $O(N^{3/2})$. One possible implementation utilizes N PUs, yielding parallel execution time of $O(\sqrt{N})$, with N data elements being shifted every step [50]. The DMM uses the hardware reduction tree to accelerate vector summation.

The simulator is cycle based, keeping records of the state of each register of each PU, and of the memory row assigned to it. Each command (for example, floating point multiply) is broken down to a series of fine-grain single bit PU operations. In a similar manner to SimpleScalar [14], the simulator also keeps track of the registers, buses and memory cells that switch during execution. With the switching activity and area power models of each baseline operation detailed in TABLE 3, the simulator tracks the total energy consumed during workload execution. We follow the area assumptions detailed in TABLE 3, and simulate speedup and power for growing datasets corresponding to growing number of PUs and hence growing total area.

Cycle-based simulated speedup is presented in Figure 16(a); speedup is defined in (20). Energy consumption is shown in Figure 16(b), with DMM being the most energy hungry benchmark. Figure 16(c) depicts the speedup/energy ratio. Note the small discontinuities in FFT and VR charts, which are due to the fact that these algorithms are designed for specific data sizes (N integral power of 2). DMM shows the worst speedup/energy over all workloads. As BSC is an embarrassingly parallel workload, its speedup/energy ratio remains practically constant with growing data set size and area.

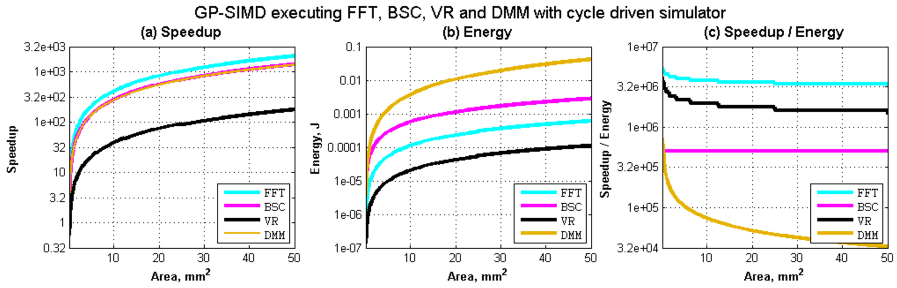


Figure 16. Simulation results: (a) Speedup (b) Energy (c) Speedup / Energy ratio, for FFT, Black-Scholes, vector reduction and dense matrix multiply algorithms

We should emphasize that these results were obtained through analytical modeling and cycle-accurate simulation; they have yet to be validated through real measurements on fabricated circuits.

5. CONCLUSIONS

GP-SIMD is a novel processor architecture that combines a general purpose sequential processor, a massively parallel SIMD array, and a large memory shared by both processors. The shared memory may be accessed simultaneously by the sequential processor (providing a row address and accessing the data columns) and by the SIMD processor (providing a column address while each processing unit, PU, accesses a memory row). Thus, data do not need to be transferred (synchronized) back and forth between separate memories of the two processors.

GP-SIMD is compared with two other massively parallel, processing-in-memory (PIM) architectures: Conventional SIMD (CSIMD) employing separate memories for the SIMD and sequential processors and requiring data synchronization (the SIMD array comprising a moderate number of FPUs, *a-la* GPU), and an Associative Processor (AP) combining storage and processing in each cell. Comparison is made by means of analytical study, and a cycle-accurate simulator is employed to validate performance and energy of four benchmarks on GP-SIMD.

GP-SIMD speedup grows faster with area than that of CSIMD due to finer granularity of its processing unit, resulting in more processing units per equivalent area. GP-SIMD speedup also grows faster with area than that of the Associative Processor due to the finer granularity of its storage elements, resulting in larger memory per equivalent area. Unlike CSIMD, the performance of GP-SIMD depends on the data word-length rather than data set size. As area budget grows beyond the speedup breakeven point, the GP-SIMD's energy remains below that of the CSIMD coprocessor, and it thus outperforms CSIMD in terms of speedup/energy ratio over a broad spectrum of area and power budget for workloads with high data-level parallelism. In general, GP-SIMD achieves higher performance than CSIMD due to CSIMD stalling on synchronization. GP-SIMD achieves higher performance-per-area than AP due to its lower memory cell area, allowing the integration of twice the amount of PUs and memory rows per given area, and its more than twice faster floating point implementation due to GP-SIMD arithmetic PU *vs*. associative-only PU of AP. GP-SIMD however is not universally efficient. While yielding high speedup when implementing fine-grain massively data-parallel workloads (such as sparse linear algebra and machine learning algorithms), its efficiency is much lower under workloads with low data-level parallelism.

To conclude, GP-SIMD architecture achieves significantly higher performance on large sets at lower power consumption over other parallel on-chip architectures. GP-SIMD is thus an efficient and noteworthy solution for the implementation of machine learning algorithms.

ACKNOWLEDGMENT

This research was funded in part by the *Intel Collaborative Research Institute for Computational Intelligence* (ICRI-CI) and by Hasso-Plattner Institute (HPI).

REFERENCES

[1] "The Intel® Xeon Phi™ Coprocessor". Available at: http://www.intel.com/content/www/us/en/high-performance-computing/high-performance-xeon-phi-coprocessor-brief.html
[2] Akerib, A. and Adar R. "Associative approach to real time color, motion and stereo vision." *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on. Vol. 5.* IEEE, 1995.
[3] Akerib, A. J., and Ruhman S. "Associative Array and Tree Algorithms in Stereo Vision." *Proc. 8th Israel Conf. on Artif. Intel. Vision & Pattern Recog., Elsevier.* 1991.
[4] Almási G. *et al.*, "Dissecting Cyclops: A detailed analysis of a multithreaded architecture", ACM SIGARCH Computer Architecture News 31.1 (2003): 26-38.
[5] AltiVec Engine http://www.freescale.com/webapp/sps/site/overview.jsp?code=DRPPCALTVC
[6] ARM® NEON™ general-purpose SIMD engine, http://www.arm.com/products/processors/technologies/neon.php
[7] Auth, C., et al. "A 22nm high performance and low-power CMOS technology featuring fully-depleted tri-gate transistors, self-aligned contacts and high density MIM capacitors." VLSI Technology (VLSIT), 2012 Symposium on. IEEE, 2012.
[8] Banerjee K. *et al.*, "A self-consistent junction temperature estimation methodology for nanometer scale ICs with implications for performance and thermal management," IEEE IEDM, 2003, pp. 887-890.
[9] Batcher, K. E., "STARAN parallel processor system hardware", *National Computer Conference*, pp. 405–410, (1974)
[10] Binkert N., *et al.* "The gem5 simulator." ACM SIGARCH Computer Architecture News 39.2 (2011): 1-7.
[11] Black F. and M. Scholes, "The pricing of options and corporate liabilities," Journal of Political Economy, 81 (1973), pp. 637–654, 1973.
[12] Borkar S.. "Thousand Core Chips: A Technology Perspective," *Proc. ACM/IEEE 44th Design Automation Conf. (DAC)*, 2007, pp. 746-749.
[13] Brockman J., *et al.* "A low cost, multithreaded processing-in-memory system", 31st international symposium on computer architecture, 2004.
[14] Burger D., T. Austin. "The SimpleScalar tool set, version 2.0", ACM SIGARCH Computer Architecture News 25.3 (1997): 13-25.
[15] Cassidy A. and A. Andreou, "Beyond Amdahl Law - An objective function that links performance gains to delay and energy", IEEE Transactions on Computers, vol. 61, no. 8, pp. 1110-1126, Aug 2012.
[16] Cloud E. L., "The geometric arithmetic parallel processor." *Frontiers of Massively Parallel Computation, 1988.* Proceedings., 2nd Symposium on the Frontiers of. IEEE, 1988.
[17] Dlugosch P., Brown D., Glendenning P., Leventhal M., Noyes H., "An Efficient and Scalable Semiconductor Architecture for Parallel Automata Processing." to appear in IEEE Transactions on Parallel and Distributed Systems, 2014.
[18] Draper, Jeff, et al. "The architecture of the DIVA processing-in-memory chip." Proceedings of the 16th international conference on Supercomputing. ACM, 2002.
[19] Esmaeilzadeh H. *et al.*, "Power challenges may end the multicore era." *Communications of the ACM* 56.2 (2013): 93-102.
[20] Flatt H. *et al.*, "Performance of Parallel Processors," Parallel Computing, Vol. 12, No. 1, 1989, pp. 1-20.
[21] Foster C., "Content Addressable Parallel Processors", Van Nostrand Reinhold Company, NY, 1976
[22] Gokhale M. *et al.*, "Processing In Memory: the Terasys Massively Parallel PIM Array," IEEE Computer, 1995, pp. 23-31
[23] Gschwind M. *et. al.*, "Synergistic processing in Cell's multicore architecture", IEEE Micro 26 (2), 2006, pp. 10–24
[24] Gunther N., S. Subramanyam, S. Parvu, "A Methodology for Optimizing Multithreaded System Scalability on Multi - Cores", http://arxiv.org/abs/1105.4301 (2011)
[25] Hall M. *et al.*, "Mapping irregular applications to DIVA, a PIM-based data-intensive architecture", ACM/IEEE conference on Supercomputing, 1999.
[26] Hardavellas N. *et al.,* "Toward dark silicon in servers." IEEE Micro 31.4 (2011): 6-15
[27] Hennessy J. and Patterson D. A., "Computer Architecture: A Quantitative Approach," 2nd Edition, Morgan Kaufmann Publishers, Inc., 1996.
[28] Hentrich D. *et al.*, "Performance evaluation of SRAM cells in 22nm predictive CMOS technology," IEEE International Conference on Electro/Information Technology, 2009.
[29] Hill M. *et al.*, "Amdahl's law in the multicore era", *IEEE Computer* 41 (7) (July 2008) 33–38.
[30] Hong, Sunpyo, and Hyesoon Kim. "An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness." *ACM SIGARCH Computer Architecture News. Vol. 37. No. 3. ACM,* 2009.
[31] Keckler S. W. *et al.*, "GPUs and the future of parallel computing." Micro, IEEE 31.5 (2011): 7-17.

[32] Kogge P. *et al.*, "PIM architectures to support petaflops level computation in the HTMT machine", *International Workshop on Innovative Architecture for Future Generation Processors and Systems*, 2000.
[33] Kozyrakis, C. E., *et al.* "Scalable processors in the billion-transistor era: IRAM." *Computer 30.9* (1997): 75-78.
[34] Kumar, Sailesh. "Smart Memory." (2012). http://www.hotchips.org/wp-content/uploads/hc_archives/hc22/HC22.23.325-1-Kumar-Smart-Memory.pdf
[35] Lipovski G., C. Yu, "The dynamic associative access memory chip and its application to SIMD processing and full-text database retrieval.", *IEEE International Workshop on Memory Technology, Design and Testing*, 1999.
[36] Loh G., "The Cost of Uncore in Throughput-Oriented Many-Core Processors", the Workshop on Architectures and Languages for Throughput Applications (ALTA), June 2008.
[37] Luebke D., "General-purpose computation on graphics hardware", Workshop, SIGGRAPH, 2004.
[38] Midwinter T., Huch M., Ivey P.A., Saucier G., "Architectural considerations of a wafer scale processor," *VLSI for Parallel Processing*, IEE Colloquium on , vol., no., pp.4/1,4/4, 17 Feb 1988
[39] Morad A. *et. al.*, "Generalized MultiAmdahl: Optimization of Heterogeneous Multi-Accelerator SoC," *Computer Architecture Letters*, 2013.
[40] Morad A. *et. al.*, "Convex Optimization of Resource Allocation in Asymmetric and Heterogeneous MultiCores," 2013, (under review).
[41] Morad A. *et. al.*, "Optimization of Asymmetric and Heterogeneous MultiCore," 2013, (under review).
[42] Morad T. *et. al.*, "Performance, power efficiency and scalability of asymmetric cluster chip multiprocessors", IEEE Computer Architecture Letters, Jan.-June 2006, Volume 5, Issue 1, pages 14 – 17.
[43] Owens J. *et al.*, "GPU Computing," Proceedings of the IEEE, Vol. 96, No. 5, pp. 879-899, May 2008
[44] Pedram A. "Algorithm/architecture codesign of low power and high performance linear algebra compute fabrics." PhD dissertation, *University of Texas* (2013) http://repositories.lib.utexas.edu/bitstream/handle/2152/21364/PEDRAM-DISSERTATION-2013.pdf?sequence=1
[45] Pollack F., "New microarchitecture challenges in the coming generations of CMOS process technologies", MICRO 32, 1999
[46] Potter J., *et al.* "ASC: an associative-computing paradigm", Computer 27.11 (1994): 19-25.
[47] Potter, J. and W. Meilander. "Array processor supercomputers", Proceedings of the IEEE 77, no. 12 (1989): 1896-1914.
[48] PowerPC Vector/SIMD Multimedia Extension http://math-atlas.sourceforge.net/devel/assembly/vector_simd_pem.ppc.2005AUG23.pdf
[49] Qing G., X. Guo, R. Patel, E. Ipek, and E. Friedman. "AP-DIMM: Associative Computing with STT-MRAM," ISCA 2013
[50] Quinn M., "Designing Efficient Algorithms for Parallel Computers", McGraw-Hill, 1987, page 125.
[51] Reddaway S. F. "DAP—a distributed array processor." *ACM SIGARCH* Computer Architecture News 2.4 (1973): 61-65.
[52] Rogers B. *et. al.*, "Scaling the Bandwidth Wall: Challenges in and Avenues for CMP Scaling". In ISCA '09: Proceedings of the 36th annual international symposium on Computer architecture, pages 371–382, New York, NY, USA, 2009. ACM
[53] Sayre, G. E. "STARAN: An associative approach to multiprocessor architecture." *Computer Architecture*. Springer Berlin Heidelberg, 1976.
[54] Scherson I. *et al.*, "Bit-Parallel Arithmetic in a Massively-Parallel Associative Processor", IEEE Transactions on Computers, Vol. 41, No. 10, October 1992
[55] Sheaffer J. *et al.* "Studying thermal management for graphics-processor architectures," ISPASS 2005
[56] Steinkraus D., L. Buck, P. Simard, "Using GPUs for machine learning algorithms," IEEE ICDAR 2005.
[57] Sterling T., H. Zima. "Gilgamesh: a multithreaded processor-in-memory architecture for petaflops computing," ACM/IEEE Conference on Supercomputing, 2002.
[58] Suh J. *et al.* "A PIM-based multiprocessor system", 15th International Symposium on Parallel and Distributed Processing, 2001.
[59] Tucker, L. W., Robertson G. G., "Architecture and applications of the connection machine." *Computer* 21.8 (1988): 26-38.
[60] Volkov, Vasily, and James W. Demmel. "Benchmarking GPUs to tune dense linear algebra." *Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE Press*, 2008.
[61] Wentzlaff D., *et al.*, "Core Count vs. Cache Size for Manycore Architectures in the Cloud." *Tech. Rep. MIT-CSAIL-TR-2010-008, MIT,* 2010.
[62] Yavits L. *et al.*, "Computer Architecture with Associative Processor Replacing Last Level Cache and SIMD Accelerator," accepted for publication, *IEEE Trans. On Computers*, 2013
[63] Yavits L. *et al.*, "The effect of communication and synchronization on Amdahl's law in multicore systems", accepted for publication, *Parallel Computing journal*, 2013
[64] Yavits L. *et al.*, "Thermal analysis of 3D associative processor", http://arxiv.org/abs/1307.3853v1

[65] Yavits L., “Architecture and design of Associative Processor for image processing and computer vision”, MSc Thesis, Technion – Israel Institute of technology, 1994, available at http://webee.technion.ac.il/~ran/papers/LeonidYavitsMasterThesis1994.pdf

[66] Zhang, Yao, and John D. Owens. "A quantitative performance analysis model for GPU architectures." *High Performance Computer Architecture (HPCA), 2011 IEEE 17th International Symposium on. IEEE*, 2011.