COMPUTER ARCHITECTURE WITH ASSOCIATIVE PROCESSOR REPLACING LAST LEVEL CACHE AND SIMD ACCELERATOR

L. Yavits, A. Morad, R. Ginosar

Abstract—This study presents a computer architecture where a last level cache and a SIMD accelerator are replaced by an Associative Processor. Associative Processor combines data storage and processing, and functions as a parallel SIMD processor and a memory at the same time. An analytic performance model of this computer architecture is introduced. Comparative analysis supported by cycle-accurate simulation and emulation shows that this architecture may outperform a conventional computer architecture comprising a SIMD coprocessor and a shared last level cache while consuming less power.

Index Terms— Multicore, SIMD, Associative Processor, Processing In Memory, PIM.

1 Introduction

Machine learning, data mining, network routing, search engines and other big data applications can be significantly sped up by massively parallel SIMD machines [42]. Many of today's computing architectures include vector, or SIMD coprocessors [1][16][24]. However data transfer between processing units (PUs) and memory significantly limits the performance of SIMD architectures [32]. High utilization of SIMD processor requires very high computation-to-bandwidth ratio and large data sets [30].

Power dissipation and on-chip communication are among the main factors limiting the scalability of parallel architectures [8]. Data synchronization and communication between PUs of SIMD processor and their private and shared memories lead to wasting energy on non-processing tasks and limit the speedup of parallel SIMD architectures [46].

When operating at high rates, arrays of computing elements in SIMD processors are very active, resulting in irregular thermal density and hotspots [41] and further limiting the scalability of conventional SIMD architectures.

The *Associative Processor* (AP) is a viable alternative to conventional SIMD processors [13][40][47]. The AP comprises a modified Content Addressable Memory (CAM) and facilitates processing in addition to storage. AP can be used as an efficient accelerator of massively-parallel fine-grain SIMD workloads.

In this study we propose to replace the last level cache (LLC) of a baseline CPU architecture (Fig. 1(a)), or the combination of the LLC and a dedicated SIMD coprocessor (Fig. 1(b)), by an AP (Fig. 1(c)). The goals we set to achieve are as follows:

- Convert the data cache into a massively-parallel processor capable of performing a variety of data-parallel fine-grain tasks.
- Eliminate a power- and bandwidth-limited SIMD coprocessor.
- Combine data storage and data processing and eliminate performance degradation and energy dissipation due to massive PU-to-memory data synchronization.

The AP may be operated in two modes:

- *Conventional Cache mode*, in which the AP serves as data cache during the execution of the sequential segments of a workload;
- *Associative Processing mode*, in which the parallelizable segments of a workload are executed on the AP. No data synchronization between sequential and parallel segments is required since the data is stored in the AP prior to the parallel execution and remains there after the parallel segment completes.

The AP delivers a number of advantages over a conventional SIMD architecture:

- Data processing and data storage are unified. There is no need for data transfer between memory and PUs;
- Two basic operations of AP are essentially standard memory operations: write and read. The third basic operation, *compare,* is implemented similarly to read, and is performed along memory rows rather than columns. Therefore the per-bit power consumption of the AP is almost identical to that of RAM, which may consume an order of magnitude lower active power (and lower leakage power) per area than logic [34];
- In conventional cache mode, use of CAM instead of RAM enables full associativity. Consequently, it may allow reduction of hardware and software complexity of the cache (for example, the elimination of costly tag array circuitry), as well as elimination of conflict

- *Leonid Yavits (*), E-mail: yavits@tx.technion.ac.il.*
- *Amir Morad (*), E-mail: amirm@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 32000, Israel.

(interference) misses.

- There are fewer hotspots. AP power is distributed uniformly over the entire processing array rather than being concentrated around PUs as in the case of conventional SIMD. Since leakage power may super-linearly depend on temperature [5], this could provide a significant advantage.

The drawback in replacing the LLC by the AP is that the effective cache size in the conventional cache mode is nearly halved, since the AP bit cell is about twice the size of a RAM cell (Fig. 3). This may lead to certain performance degradation during the execution of the sequential portion of a workload, but the speedup achieved during the execution of the parallel portion of the workload may yield a significant improvement in the overall system performance.

The first contribution of this paper is the integration of an AP on-chip of a standard CPU. The memory of the AP replaces the LLC, while the processing of the AP replaces the on-chip SIMD accelerator. This contribution leads to improvement in performance, reduction in power dissipation, and lower temperature, enabling 3D integration.

Another contribution of our work is the comparative performance and power analysis of AP *vs*. a conventional SIMD processor, supported by analytical modeling, cycle-accurate simulation and emulation. Thanks to modern feature scaling and aggressive memory integration on one side, and the rise of big data on the other, we believe we are at an inflection point where AP may outperform conventional SIMD 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 AP and its operation. Section 4 presents simulation and analytical modeling of AP and compares it to a conventional SIMD processor. Section 5 combines analytical, simulation and emulation methods to compare performance and power consumption of the three architectures of Fig. 1, and Section 6 offers conclusions.

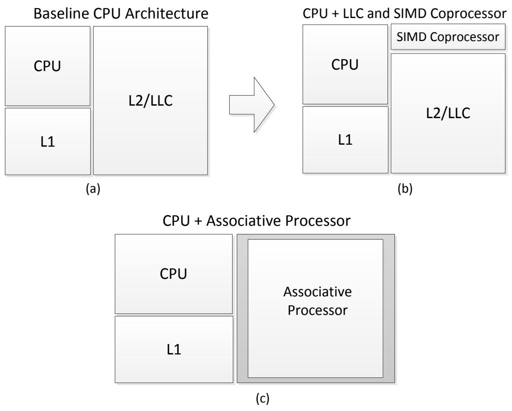
Baseline CPU Architecture
CPU
L1
L2/LLC
(a)
CPU + LLC and SIMD Coprocessor
SIMD Coprocessor
CPU
L1
L2/LLC
(b)
CPU + Associative Processor
CPU
L1
Associative Processor
(c)

Fig. 1. (a) Baseline CPU with the LLC, (b) CPU with the LLC and the SIMD coprocessor, (c) CPU with the AP replacing the LLC.

2 Related Work

A major notion of our work is using AP to unify processing and storage, i.e. achieve 'processing in memory' (PIM). Hence we place our research in the context of prior work on PIM. The concept of mixing memory and logic has been around since 1960s. The DAPP, STARAN, CM-2, and GAPP computer architectures [36] used large number of PUs positioned in proximity of memory arrays to implement massively parallel SIMD computer.

M. Gokale *et al.* [15] designed TeraSys, a computer architecture comprising a conventional host processor, with at least part of its memory replaced by PIM array, integrating memory and ALUs in close proximity. M. Hall *et al.* [19] developed DIVA, the Data-Intensive Architecture, combining PIM memories with external host processors. One of their main focuses was performing selected computation in processing elements near memory and reducing the quantity of data transferred across the long and slow processor-memory interface. G. Almási *et al.* [3] developed Cyclops, an architecture combining memory and a large number of simple PUs. According to their findings, standard benchmarks are not efficient when evaluating the performance of PIM architecture. Instead, they focused on scientific kernels including FFT, matrix-matrix and matrix-vector multiplication, etc. T. Sterling *et al.* [43] developed Gilgamesh, a PIM based massively parallel architecture, with the focus on advanced mechanisms for virtualizing tasks and data. P. Kogge *et al.* [25] developed HTMT, a parallel multilevel memory architecture, where each RAM level is a PIM memory (memory blocks interconnected to ALUs). J. Suh *et al.* [44] introduced a SLIIC QL computer featuring a processor integrated on the same die with DRAM. J. Brockman *et al.* [9] developed PIM lite, a PIM architecture featuring a multithreaded core with SIMD accelerator integrated with DRAM on the same chip. Last, G. Lipovsky *et al.* [28] developed a dynamic associative access memory architecture that combined DRAM and a single-bit processing element, capable of associative and conventional arithmetic processing, placed in DRAM's sense amplifier area. More recently, it became impractical to embed processing on DRAM chips, as the IC technology of DRAM does not support logic circuits. All these PIM architectures placed processing *in proximity* of memory. In contrast, this work considers AP, in which processing is carried out within each bit cell.

Prior work on the AP concept was conducted over the years. Foster [13] laid the foundations for associative processing. J. Potter *et al.* [35] developed an associative programing model and language and applied it to a wide variety of applications including image processing, graph algorithms, data base management, graphics, etc. I. Scherson *et al.* [40] developed high-speed AP architectures [40]. The present authors have implemented a complete stand-alone AP as a VLSI chip [47].

This work progresses from processing *in proximity* of memory to processing combined inside memory PIM. Its key contribution is to integrate an AP on-chip of a standard CPU. The memory of the AP replaces the LLC, while the processing of the AP replaces the on-chip SIMD accelerator.

3 The Associative Processor

In this section we present the architecture of the AP and explain the principles of associative computing.

3.1 Associative Processor Architecture

AP is based on modified CAM. The CAM allows comparing all data words to a key, tagging the matching words, and possibly reading some or all of the tagged words one by one. In addition, standard memory read and write operations of a single word at a time can also take place.

Unlike CAM, typical operations in AP are consecutive compare and write, usually involving just a few bit columns. The AP enhances the CAM by allowing parallel writing into selected bits of all tagged words. The architecture of AP is presented in Fig. 2. The Associative Processing Array comprises bit cells (further described below) organized in bit-columns and word-rows. Typically, a word-row makes a PU (although parts of a row, or alternatively multiple rows, may also be configured as a PU). Since we operate the AP in dual mode (conventional cache and associative processing), single PU may be aligned with a cache line (for example 64 bytes) for higher efficiency. Several special registers are appended to the associative processing array. The KEY register contains a key data word to be written or compared against. The MASK register defines the active fields for write and read operations, enabling bit selectivity. The TAG register marks the rows that are matched by the compare operation and may be affected by consecutive parallel write. The AP may require a microcontroller and an instruction cache. An optional Interconnect allows PUs of the AP to communicate in parallel. Since associative processing operation is mainly bitwise, the Interconnect can be a relatively simple circuit-switched network. The Interconnect is further discussed in Section 3.2. Reduction Tree ([37], earlier introduced as 'response counter' in [47]) is an adder tree, enabling quick parallel summation of TAG bits. This operation is useful whenever a vector needs to be reduced into a scalar.

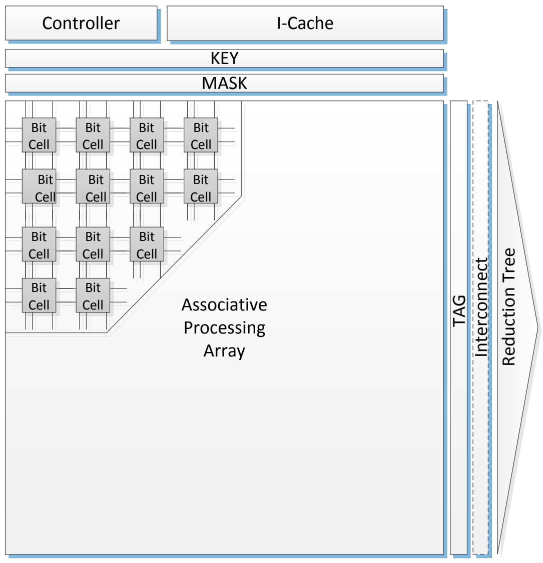

Fig. 2. Associative processor architecture

A static memory based associative bit cell is shown in Fig. 3. Its two main components are the 6-Transistors (6T) SRAM bit cell and the 4T N-type XOR. Two additional transistors (gated by the Mask wire) are used to mask the write operation at the bit (column) level. Alternative designs have also been proposed, to reduce power dissipation [27], to save area [33] or to exploit non-transistor technology [37].

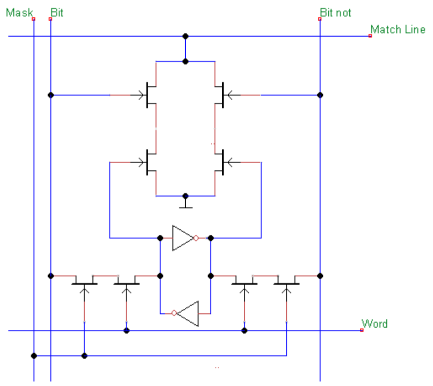

Fig. 3. NOR-type Associative Bit Cell

To compare the key data word to the data stored in the associative memory (the entire row, a number of bits or a single bit), the Match line is precharged and the inverted key is set on Bit and Bit-not lines. In the columns that should be ignored during comparison, Bit and Bit-not lines are set to 0. If all unmasked bits in a row match the key (i.e. every unmasked bit in a row is different from the corresponding inverted key bit), the Match line remains high and a 1 is written into the corresponding TAG bit. If the key differs from the row data (even in one bit), the Match line discharges and a 0 is written into the TAG bit.

In AP, compare is typically followed by a parallel write into the unmasked bits of all tagged words. To write data (from the KEY register) into the associative memory, each TAG bit (set earlier by the compare) is connected to the corresponding Word line. If a row matched during the compare, the key data is written into it in accordance with the MASK pattern. Otherwise (in the case of mismatch), the write does not affect the row. Typically, 12.5-25% of the rows are written during a write in arithmetic operations as further shown in Section 3.2.

A compare-write sequence is illustrated in Fig. 4. A KEY value '001' with MASK value '011' is compared against the associative memory content. Afterwards a KEY value '111' with MASK value '110' is written into all associative memory rows that matched during Compare. In Compare, an inverted KEY bit is compared with each associative memory bit of its bit-column in parallel. The results of bit-compares are AND-ed in each row to generate a match or mismatch. The AND output is stored in a TAG bit. The masked-out bit columns do not affect the Compare result. In consequent Write, the KEY value is broadcast to the entire associative memory array. The logic AND of a MASK bit and the TAG bit is used to enable / disable the write operation: only the rows that matched during the Compare and only the bits which MASK is 1 are written.

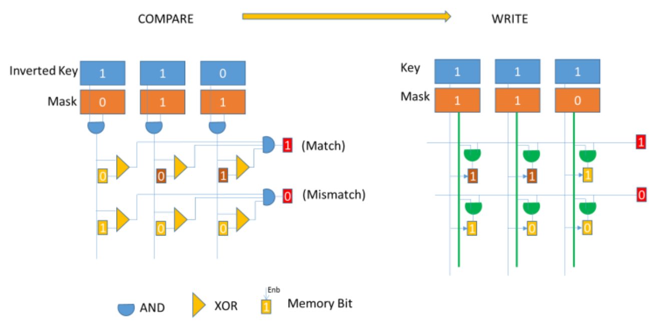

Fig. 4. Compare-Write Logic Sequence

To read data from memory, the Bit and Bit-not lines are precharged and the Word line is asserted. Parallel write and sequential read operations are enabled only for the columns whose mask bits are set in the MASK register.

A complete design of an AP is presented in [47].

3.2 Associative Computing

AP is a massively parallel SIMD accelerator. It can implement a wide range of processing tasks, as well as classical CAM operations such as associative search, sorting and ordering. In addition it supports standard memory operations (word and block read and write). AP is efficient for computational tasks that require fine-grain massive data parallelism, such as high-resolution image processing or large data set sparse linear algebra algorithms as may be required in machine learning.

Arithmetic operations in the AP can be performed in parallel on all PUs in a word-parallel, bit-serial manner. For instance, vector addition may be performed as follows [13]. Two m bit columns hold vectors A and B (Fig. 5). Their sum A+B is written over B. A one-bit column C holds the carry bit. The addition is carried out in m single-bit addition parallel steps (1):

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

where i is the bit index and '$*$' is the word index in the vector. The single-bit addition (TABLE 1) is carried out in a series of compare-write steps (as illustrated by Fig. 4). In each such step, one input entry of the truth table (a three bit input pattern) is matched against the contents of the $a[*]_i, b[*]_i, c[*]$ bit columns in the associative array, and the matching rows (PUs) are tagged; then the logic result (two-bit output of the truth table in TABLE 1) is written into b_i and c bits of all tagged rows. During each compare and write step, all but three input bit columns and two output bit columns respectively are masked out, so that 2.5 bit columns are active on average. Some input combinations do not change the output and therefore can be skipped ("No action" in the table). Since the operation overwrites one of the inputs, computation must be carried out according to the order indicated in TABLE 1 [13].

Overall, four compare - write steps are required to complete the single-bit addition. Therefore, fixed point m bit addition takes $8m \in O(m)$ cycles. Subtraction and comparison operations are performed similarly and also require $O(m)$ cycles. Note the stark contrast with SIMD architectures of low PU count that require $O(N)$ cycles to add N data elements (without taking into account the load / store / move time).

Fixed precision multiplication and division in AP are implemented by long multiplication and division respectively, consisting of a series of add-shift and subtract-shift operations, executed bit-serially but in parallel for all data words. The addition or subtraction are done as described above (multiplication is usually done "MSB first"), while shift is implemented by activating different bit columns and therefore requires no cycles. Thus, fixed point $m-bit \times m-bit$ vector multiplication requires $O(m^2)$ cycles [13], regardless of the length of the vectors.

Floating point arithmetic for APs is somewhat more complex to implement. Different exponents require shifting mantissas by different lengths, resulting in a sequence of bit-serial operations. Still, a direct implementation of IEEE single precision floating point element-by-element vector multiplication $(B =)A \times B$ requires only 4400 cycles, regardless of the length of the vector.

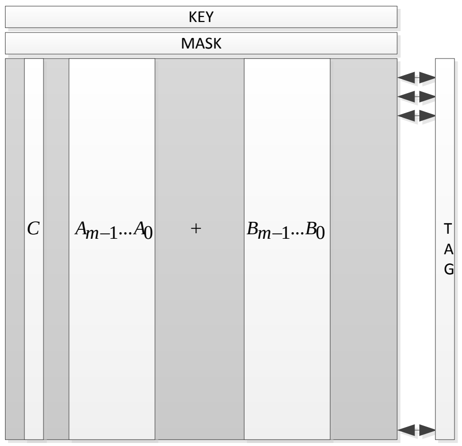

Fig. 5. Addition Example

TABLE 1
IMPLEMENTING FULL ADDER IN ASSOCIATIVE PROCESSOR

Entry	Input C	Input B	Input A	Output C	Output B	Comments
0	0	0	0	0	0	No action
1	0	0	1	0	1	2nd pass
2	0	1	0	0	1	No action
3	0	1	1	1	0	1st pass
4	1	0	0	0	1	3rd pass
5	1	0	1	1	0	No action
6	1	1	0	1	0	4th pass
7	1	1	1	1	1	No action

Pass = COMPARE cycle followed by WRITE cycle

Arithmetic operations are presented in this Section under the assumption that the relevant operands are located in the same PU. However, many workloads require inter-PU data communications. Depending on the workload, communication requirements may vary from no communications (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 of advantage (e.g., for FFT). The inter-PU communication can be implemented serially, through a series of associative memory reads and writes. Alternatively, the dedicated Interconnect introduced in Section 3.1 can be employed to provide parallel communication capabilities, i.e. to allow all PUs to communicate in parallel.

4 Analytic model and comparative analysis

Analytical modeling is becoming an increasingly important technique used in the design of chip multiprocessors [11][20][23][29][45]. In this section we develop an analytical performance and power consumption model of the conventional SIMD and the AP and compare their relative performance, area and power consumption under constrained area and power resources. Here we study only the parallelizable portion of a workload. For simplicity, we assume that the parallelizable portion contains single-cycle instructions (i.e. arithmetic, control, register file access and alike). We also assume the performance of the baseline sequential CPU to be 1 for the sake of estimating the relative speedup delivered by the reference SIMD coprocessor and the AP.

We verify our analytical modeling findings using cycle-accurate simulation of the AP. The simulator, our simulation methodology and simulation results are described in details in Section 4.4.

4.1 Reference SIMD Processor

Fig. 6 presents the computer architecture comprising the sequential CPU, the shared LLC and the SIMD coprocessor, as depicted in Fig. 1(b). The reference SIMD coprocessor contains a number of baseline PUs (BPUs), each containing a floating point ALU and a register file. The BPUs are connected to the shared LLC through a bandwidth-limited interface, and are interconnected using an interconnection network (not shown).

Let the serial execution time of the parallelizable portion f of the program on the baseline sequential CPU be T_1. The execution time $T_{f,SIMD}$ of that parallelizable portion on the SIMD coprocessor can then be written as follows:

$$T_{f,SIMD} = \frac{T_1}{n_{SIMD}} + \frac{T_C}{n_{SIMD}} + T_S \qquad (2)$$

where n_{SIMD} is the number of BPUs, T_C is the time spent exclusively on inter-BPU communication, and T_S is the time spent exclusively on synchronization of the LLC to the private SIMD memory [46]. The synchronization consists of the time to move data from LLC to SIMD before the parallel segment begins, and from SIMD to LLC after the parallel segment completes. Since it involves access to a shared resource, T_S might depend on the number of BPUs in the SIMD coprocessor [12][39]. This is especially the case when the data set size is scaled down to the processor size.

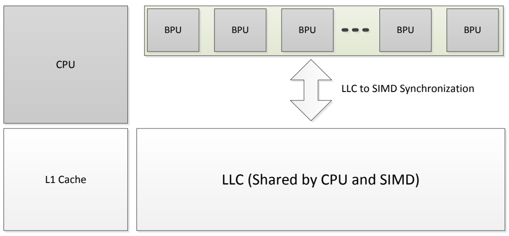

Fig. 6. CPU with SIMD coprocessor and shared LLC

While a number of BPUs (or all BPUs) can communicate with each other in parallel (although using a potentially congested interconnection network that affects T_C), the LLC-to-SIMD synchronization is done essentially serially for each BPU. Therefore the inter-BPU communication time scales by the number of BPUs while T_S does not scale. The speedup of the SIMD processor over the sequential CPU can be written as follows:

$$S_{SIMD} = \frac{T_1}{T_{f,SIMD}} = \frac{1}{\frac{1}{n_{SIMD}} + \frac{I_c}{n_{SIMD}} + I_s} \qquad (3)$$

where $I_c = T_C/T_1$ is the *connectivity intensity*, or ratio of the time spent on inter-BPU communication to the serial execution time, and $I_s = T_S/T_1$ is the *synchronization intensity*, or the ratio of time spent on LLC-to-SIMD synchronization to the serial execution time.

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

$$A_{SIMD} = n_{SIMD}(A_{ALU} + A_{RF}) \qquad (4)$$

where A_{ALU} is the ALU area and A_{RF} is the register file area. As noted above, the inter-PU connection network is omitted.

For easy 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 $0.1\mu m^2$ [4]. Then we can write:

$$\begin{aligned} A_{ALU} &= A_{ALUo} m^2 \\ A_{RF} &= A_{RFo} km \end{aligned} \qquad (5)$$

where A_{ALUo} is the area of a single bit of the ALU and A_{RFo} 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 basic and does not take into account numerous aspects of SIMD design (instruction cache, communication and control, etc.). Its purpose is providing the best case reference figures for the comparative analysis of the conventional SIMD processor's speedup, area and power.

The average power of the SIMD processor (over the execution span $T_{f,SIMD}$) can be written as follows:

$$P_{SIMD} = \frac{E_{COMP} + E_C + E_S + E_{LEAK}}{T_{f,SIMD}} = \\ = \frac{\frac{P_{COMP}}{n_{SIMD}} + \frac{I_c P_C}{n_{SIMD}} + I_s P_S}{(\frac{1}{n_{SIMD}} + \frac{I_c}{n_{SIMD}} + I_s)} + P_{LEAK} \tag{6}$$

where $T_{f,SIMD}$ is the execution time of the parallelizable portion of the program on the SIMD processor (2); E_{COMP} and P_{COMP} are the energy and the average power consumption during computation; E_C and P_C are the energy and the average power consumed during inter-BPU communication; E_S and P_S are the energy and the average power consumed during LLC-to-SIMD synchronization; E_{LEAK} and P_{LEAK} are the leakage energy and power; I_c and I_s are the connectivity and synchronization intensities as defined above.

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 $1\mu W$ [22]. Then we can further write the SIMD power consumption as follows:

$$\begin{gathered} P_{COMP} = n_{SIMD}(P_{ALUo} m^2 + P_{RFo} km) \\ P_C = n_{SIMD} P_{Co} m \\ P_S = P_{So} m \end{gathered} \tag{7}$$

where P_{ALUo} and P_{RFo} are the average per-bit power consumptions of the ALU and RF respectively during computation. P_{Co} is the per-bit power consumption during the inter-BPU communication. P_{So} is the per-bit power consumed performing LLC-to-SIMD synchronization. We assume the amount of data that needs to be synchronized with LLC, as well as transferred during inter-BPU communication, is limited to a single data word per BPU. P_{COMP}, P_C and P_S are measured in SRAM cell write power consumption units.

Leakage power can be expressed as follows:

$$P_{LEAK} = \beta A V^{\alpha} = \gamma A \tag{8}$$

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. Therefore the total power can be written as follows:

$$P_{SIMD} = \frac{P_{ALUo} m^2 + P_{RFo} km + I_C P_{Co} m + I_s P_{So} m}{\frac{1}{n_{SIMD}} + \frac{I_c}{n_{SIMD}} + I_s} + \\ + \gamma n_{SIMD}(A_{ALUo} m^2 + A_{RFo} km) \tag{9}$$

4.2 Associative Processor

In this section we construct the analytical model for the speedup, area and power consumption of the AP. The execution time of the parallelizable portion f of the program on the AP can be written as follows:

$$T_{f,AP} = \frac{T_1}{s_{APE} n_{AP}} + \frac{T_C}{n_{AP}} + T_S(N - n_{AP}) \tag{10}$$

where n_{AP} is the number of PUs in the AP, N is the data set size, s_{APE} is the speedup of associative PU relative to the BPU, T_C and T_S are as defined in (2). Since AP in our research replaces the LLC, there is no need for data synchronization unless the entire data set does not fit in the AP. In other words, $T_S(N - n_{AP}) = 0$ for $N \leq n_{AP}$.

Assuming single precision floating point arithmetic, the longest among frequently used arithmetic operations is multiplication, which in one direct implementation takes 4400 cycles vs. 1 cycle on the baseline sequential CPU or the BPU. Lacking *a-priori* knowledge of the workloads to be executed on the AP, we assume the worst case scenario comprising a continuous series of floating point multiplications. In this case $s_{APE} = 1/4400$. The speedup of the AP can then be written as follows:

$$S_{AP} = \frac{1}{\frac{1}{s_{APE} n_{AP}} + \frac{I_c}{n_{AP}} + I'_s} \tag{11}$$

where $I'_s = I_s(N - n_{AP})$. The area of the AP can be written as follows:

$$A_{AP} = n_{AP}(A_{APo} km + 2A_{ALUo}) \tag{12}$$

where k is the size of the associative PU (in data words), including temporary storage, A_{APo} is the AP cell area, measured in SRAM cell area units, and $2A_{ALUo}$ is the per-PU reduction tree size. Similarly to the reference SIMD coprocessor, we ignore the area of the interconnection network.

The average power of the AP can be written as follows:

$$\begin{gathered} P_{AP} = \frac{E_{COMP} + E_C + E_S(N - n_{AP}) + E_{LEAK}}{T_{f,AP}} = \\ = \frac{\frac{P_{COMP}}{n_{AP}} + \frac{I_c P_C}{n_{AP}} + I'_s P_S}{\frac{1}{n_{AP}} + \frac{I_c}{n_{AP}} + I'_s} + \gamma n_{AP} A_{APo} km \\ P_C = n_{AP} P_{Co} \\ P_S = P_{So} m \end{gathered} \tag{13}$$

where E_{COMP} and P_{COMP} are the AP computation energy and power consumption; E_C and P_C are the AP energy and power consumption during inter-PU communication; E_S and P_S are the energy and the average power consumed during synchronization, if the entire data set does not fit in the AP; E_{LEAK} and P_{LEAK} are the AP leakage energy and power. P_{Co} is the per-bit power consumption during the inter-PU communication; P_{So} is the per-bit power during synchronization. Note that for comparison purposes we use the same leakage power (represented as a function of area only as in (8)) for both the AP and the SIMD processor. This might be somewhat unfair to the

AP: First, the leakage power per area could be lower for memory than for logic [34]. Second, the AP has fewer hotspots [48]. Since the leakage power is highly temperature dependent, hotspots may lead to higher leakage in the SIMD processor [5].

In order to further detail P_{COMP}, recall the implementation of single-bit addition (on which other arithmetic operations are based) described in Section 3.2. In each pass of the single-bit addition, a three bit input combination $a[*]_i, b[*]_i, c[*]$ is compared in parallel in all PUs and afterwards a two bit result $b[*]_i, c[*]$ is written into the tagged PUs; that sequence is repeated m times for m-bit words. Since there are eight independent logic combinations (TABLE 1), each PU has $1/8$ probability of match and $7/8$ of mismatch (in which case the Match line discharges). Similarly, each PU has $1/8$ probability of write and $7/8$ probability of a *miswrite* (when Bit and Bit-not lines are charged without Word line being asserted). Since we define the power consumption of a single SRAM cell during write operation as 1, P_{COMP} can be presented as:

$$P_{COMP} = \frac{2\cdot\left({}^{1}/_{8} + {}^{7}/_{8}\cdot p_{mw}\right) + 3\cdot\left({}^{1}/_{8}\cdot p_m + {}^{7}/_{8}\cdot p_{mm}\right)}{2} n_{AP} \qquad (14)$$

for 2-bit write and 3-bit compare operations, where p_{mw} is the normalized per-bit power consumption of a miswrite, p_{mm} is the normalized per-bit power consumption of a mismatch, and p_m is the normalized per-bit power consumption of a match (TABLE 3).

Model (13) is fairly basic and does not account for certain statistics that work in favor of the AP. For example, a certain percentage of associative memory cells that are written a new value in fact do not change (consuming considerably less power); similarly, a certain percentage of asserted bit lines do not recharge (or discharge) since the same value is asserted. Our goal is to create a simple power model that reflects the worst case power consumption of the AP.

4.3 Modeling under constrained area

The number of AP PUs may be derived as the function of the constrained area budget A using (12) as follows:

$$n_{AP} = \frac{A}{A_{APo}km + 2A_{ALUo}} \qquad (15)$$

We can further substitute n_{AP} in (11) and (13) by (15) and obtain the speedup and the power consumption of the AP as function of the area budget. The area parameters we use for modeling purposes are presented in TABLE 2.

Speedup vs. area for the reference SIMD coprocessor and the AP is shown in Fig. 7. For mathematical simplicity, synchronization intensity I_s is assumed to be constant 0.01 (namely, synchronization takes 1% of the serial execution time).

TABLE 2
AREA MODEL PARAMETERS

Parameter	Description	Attributed to	Value
A_{ALUo}	ALU bit cell area	SIMD	20 (1)
A_{RFo}	Register bit (FF) area	SIMD	3 (1)
S_{APE}	AP speedup relative to sequential CPU	AP	1/4400
A_{APo}	AP bit area	AP	2 (1)
m	Data wordlength	Both	32
k	Register file size in SIMD, AP PU size (in 32-bit words)	Both	8

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

As the area budget increases, the speedup of the reference SIMD coprocessor exhibits diminishing returns caused by the LLC-to-SIMD synchronization. Eventually the speedup saturates:

$$\lim_{n_{SIMD}\to\infty} S_{SIMD} = \lim_{n_{SIMD}\to\infty} \frac{1}{\frac{1}{n_{SIMD}} + \frac{I_C}{n_{SIMD}} + I_s} = \frac{1}{I_s} \qquad (16)$$

As evident from Fig. 7, the speedup of the AP is lower than the speedup of the reference SIMD coprocessor at low area, but it increases to reach the breakeven point at around $30mm^2$. Diminishing returns affect the AP speedup to a lesser extent, since they only occur when the data set does not fit into the AP. To demonstrate this effect, we assume that the data set size grows with the AP size (same as for SIMD) until $n_{AP} = N = 10^4$, after which the data set size N grows twice as fast as the n_{AP}. This is what causes the AP speedup to eventually saturate as well.

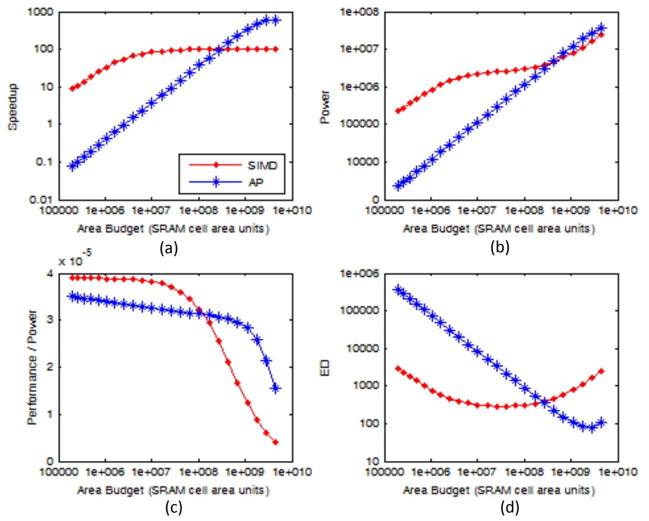

Fig. 7. Analytical results under constrained area: (a) Speedup (b) Power (c) Performance / Power ratio (d) $Energy \times Delay$

The power consumption vs. area budget for the SIMD and the APs is shown in Fig. 7(b). The power consumption of the AP is lower than that of the SIMD processor when area is under $30mm^2$. For larger area, the AP consumes more power than the SIMD processor. Note that even when the speedups saturate, power consumption continues to grow with area, due to the leakage.

The performance/power ratio vs. area for the SIMD and the APs is shown in Fig. 7(c). For the SIMD processor, the performance/power ratio drops because speedup saturates while power dissipation continues to grow with increasing area. For lower area, AP underperforms SIMD in terms of performance/power ratio. But as SIMD's speedup saturates, AP yields better performance/power ratio. Eventually, the latter also drops, as the AP speedup saturates.

The energy-delay product (ED) vs. area for SIMD and AP is shown in Fig. 7(d). Since the task size is not constant, we use normalized delay, i.e. the ratio of the delay to the serial execution time. The SIMD processor's ED reaches a minimum around $5mm^2$ (where the speedup saturates) and begins to grow again due to growing power consumption. The AP's ED follows a similar pattern but reaches its minimum at much larger area budget. Both performance/power and ED product breakeven points (at which AP begins to outperform SIMD) occur at $20mm^2$.

4.4 AP Simulation

The purpose of the simulation is to validate the analytic results obtained in Section 4.3. The workloads are defined, followed by description of the cycle-accurate AP simulator, our simulation methodology and simulation results.

4.4.1 Workloads

The following workloads have been selected for performance and power consumption simulations:

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

where N is the data set size, for simplicity scaled to the processor size (following the methodology suggested in [17]), i.e. $N = n_{AP}$. Note that simulations do not cover the cases where the data size exceeds the size of the processor (requiring data synchronization).

4.4.2 Simulator

We simulate the AP using an in-house cycle-accurate simulator. The workloads are hand-coded. For FFT, we use optimized parallel implementation outlined in [38]. For Black-Scholes, we used a direct implementation optimized for associative processing, based on formulation in [7]. Vector reduction is implemented using the reduction tree. Matrix multiplication uses AP's compare and arithmetic capabilities to match the input matrix element pairs and multiply them. The singleton products are summed by the reduction tree.

The first step of AP programing is identification of the finest data parallelization level and mapping of the workloads on the associative processing array. For matrix multiplication, each pair of elements to be multiplied is processed by a single PU. For FFT, each multiply-accumulate operation is carried out by a single PU. For Black-Scholes option pricing, a single PU handles a single call option of a single security at a single strike price and a single expiration time. For vector reduction, a single PU retains a single vector element. At the next step, we break each fine-grain data thread into a series of arithmetic and data communication operations, and manually allocate temporary storage. At the last step, each arithmetic and communication operation is converted into a series of compares, writes and data moves. Simulation times are presented in TABLE 4.

For power simulation, we follow the methodology of SimpleScalar [10], which allows keeping track of what units are active during execution and records the total energy consumed for a workload. During the AP execution, we record and count all baseline operations (match, mismatch, write, miswrite, data move, reduction). Using power models of each baseline operation, detailed in TABLE 3, we are able to estimate the total energy consumed during execution of each case.

TABLE 3
POWER MODEL PARAMETERS

Parameter	Description	Value
p_{nw}	per-bit power consumption during a miswrite	0.1 (1)
p_m	per-bit power consumption during a match	0.1 (1)
p_{mm}	per-bit power consumption during a mismatch	0.75 (1)
γ	static power coefficient	$5 \cdot 10^{-2}\ \mathrm{W/mm^2}$ (2)

(1) Based on [22], relative to the power consumption of SRAM bit cell during write from 0 to 1 or from 1 to 0 operation

(2) Based on typical industry data at typical conditions for advanced technologies

TABLE 4
DATA SET SIZES AND SIMULATION TIMES

Workload	Date Set Size	Simulation Time
BSC	$2^8 \div 2^{20}$	4 sec ÷ 1hr 50min
FFT	$2^8 \div 2^{20}$	3 sec ÷ 2hr 35min
DMM	$2^8 \div 2^{20}$	2 sec ÷ 12hr 55min
VR	$2^8 \div 2^{20}$	2 sec ÷ 6 sec

Simulations performed on Intel® Core™2 Quad CPU Q8400 with 8GB RAM

4.4.3 Results

We simulate speedup and power per workload for 16 different values of area. In all cases, the PU size is 256 bits (TABLE 2).

Simulated speedup results are presented in Fig. 8(a). DMM uses the reduction tree as an accelerator. BSC is an embarrassingly parallel workload. Hence DMM and BSC obtain higher speedup than FFT. VR is an outlier, since it is implemented using the word- and bit-parallel reduction tree rather than bit-serial associative arithmetic, thus achieving considerably higher speedup.

Power consumption results are presented in Fig. 8(b). All workloads consume power of the same order of magnitude (hence we use linear rather than log-log scale).

This happens because all workloads are implemented using mostly identical associative primitives (compare and write). Although VR and to a lesser extent DMM use the relatively power-hungry reduction tree, reduction time is almost negligible compared to the time of associative operations. Performance/power ratio and ED product are shown in Fig. 8(c) and (d) respectively. Among DMM, FFT and BSC workloads, DMM shows the best performance/power and ED, thanks to the accelerated reduction operation. Since BSC is an embarrassingly parallel workload, its performance/power ratio remains almost constant with data set size / area. The power consumption of VR is significantly higher than that of the rest of the workloads. However since its speedup is also at least an order of magnitude higher, VR exhibits considerably better performance/power ratio and ED product.

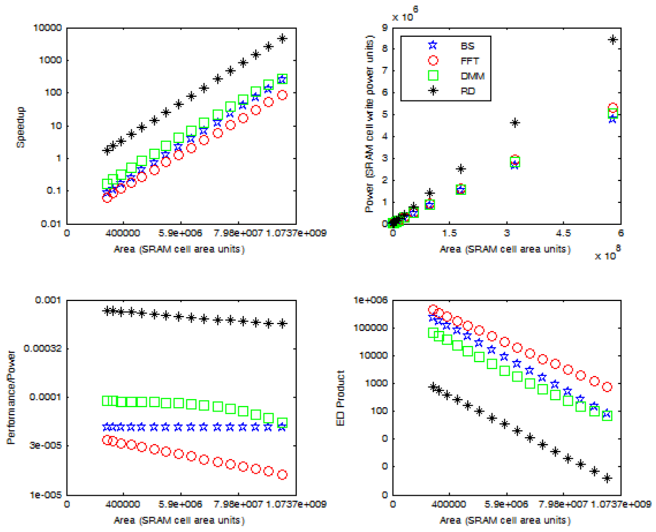

Fig. 8. Simulation results: (a) Speedup (b) Power (c) Performance / Power ratio (d) $Energy \times Delay$

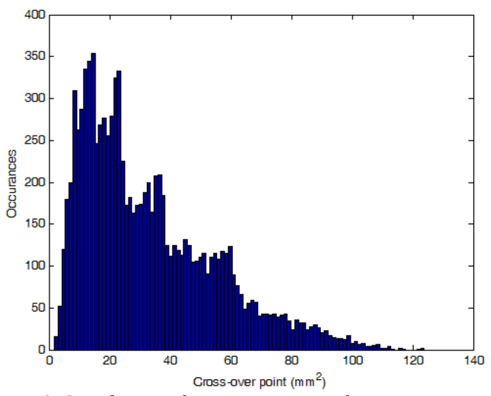

Fig. 9. Speedup Breakeven Points Distribution

4.5 Sensitivity to parameter variation

The parameters used in our modeling are technology and design dependent. In order to determine how the changes in these parameters affect the results, we randomize the parameters in TABLE 2 using uniform distribution of ∓50%.

Fig. 9 shows the distribution of the speedup breakeven point (i.e. the area at which the speedup of the SIMD processor is the same as that of the associative processor, $\sim 30mm^2$ in Fig. 7). As expected, the distribution of speedup breakeven point is close to lognormal (because at least some of the independent random parameters are positive and multiplicative), with a mean value of $\sim 32mm^2$.

5 CPU WITH AP VS. CPU WITH LLC AND SIMD COPROCESSOR

While in the previous section the AP has been compared with a standalone SIMD processor, in this section they are considered in the context of a CPU architecture.

Complexity and runtime requirements make it challenging to rely on cycle-accurate simulation for the large-scale design space exploration that we undertake. We use analytical modeling to compare the performance and the power consumption of a CPU with an AP (Fig. 1(c)) vs. CPU with a LLC and a SIMD coprocessor (Fig. 1(b)), under constrained area resource. In this analysis we assume that the areas of the CPU and the L1 cache are constant. The variable area budget is therefore assigned entirely to the LLC in the baseline architecture (Fig. 1(a)), or divided among the LLC and the SIMD coprocessor (Fig. 1(b)), or assigned entirely to the AP (Fig. 1(c)). We begin our comparative analysis with performance and follow with power consumption.

We confirm our analytic results by emulation, whereby the workload is executed and performance is measured on a state-of-art computer system with SIMD accelerator, as explained in Section 5.3. Emulation results are also combined with the results of cycle-accurate simulations of AP, to derive the performance of CPU with the AP.

5.1 Performance Modeling

Following [11] and [31], we can present the execution time of a workload on the baseline CPU architecture (Fig. 1(a)) as a function of its LLC size $A_{LLC} = A$ as follows:

$$T_1(A_{LLC}) = M[g \cdot CPI_{MEM} + (1-g) \cdot CPI_{CPU}] \tag{17}$$

where A is the area budget, M is the number of instructions in the workload, g is the fraction of memory access instructions, CPI_{CPU} is the average number of cycles per instruction for instructions that require no memory access (assumed to be constant, as defined in TABLE 5), and CPI_{MEM} is the average number of cycles per memory access. CPI_{MEM} can in turn be presented as follows [49]:

$$CPI_{MEM} = (1-m_1)d_{L1} + m_1(1-m_2)d_{LLC} + m_1 m_2 d_D \tag{18}$$

where m_1 and m_2 are miss rates of L1 and LLC respectively, d_{L1} and d_{LLC} are access times of L1 and LLC respectively, and d_D is the off-chip DRAM access time.

The miss rate of the LLC can be written as follows [21]:

$$m_2 = m_1\sqrt{A_{L1}/A_{LLC}} \tag{19}$$

where A_{L1} and $A_{LLC} = A$ are the areas of the L1 and the

LLC respectively.

The execution time of the same workload on the CPU with the LLC and the SIMD coprocessor can be written as follows:

$$T_2 = (1-f) \cdot T_1(A_{LLCo}) + \frac{f \cdot M}{s_{SIMD}(A_{SIMDo})} \tag{20}$$

where f is the parallelizable portion of the program and s_{SIMD} is the speedup of the SIMD coprocessor as defined in (3); A_{LLCo} and A_{SIMDo} are the areas of the LLC and the SIMD coprocessor, respectively, so that $A_{LLCo} + A_{SIMDo} = A$. The parallelizable portion of the workload is assumed to contain single-cycle instructions, similarly to Section 4.

The execution time of the same workload on the CPU with the AP can be written as:

$$T_3 = (1-f) \cdot T_1(A/A_{APo}) + \frac{f \cdot M}{S_{AP}} \tag{21}$$

where S_{AP} is the speedup of the AP as defined in (11); A/A_{APo} is the effective area of the LLC implemented by the AP (operated in the conventional cache mode during the execution of the serial fraction of the workload), where A_{APo} (the area of the AP cell in SRAM cell units) is 2, as defined in TABLE 2.

Following (20) and (21), the effective number of cycles per instruction for the CPU with the LLC and the SIMD coprocessor (Fig. 1(b)) and for the CPU with the AP (Fig. 1(c)) can be written as follows:

$$CPI_{SIMD} = \frac{T_2}{M}; \qquad CPI_{AP} = \frac{T_3}{M} \tag{22}$$

We further define the overall speedup of these two architectures as follows:

$$SU_{SIMD} = \frac{T_1(A)}{T_2}; \qquad SU_{AP} = \frac{T_1(A)}{T_3} \tag{23}$$

The timing and area parameters used for modeling purposes are specified in TABLE 2 and TABLE 5.

Fig. 10 shows the effective number of cycles per instruction vs. area budget for both architectures for $f = 0.75, 0.9, 0.99$ and 0.999. Fig. 11 shows the overall speedup of these architectures for the same f.

In Section 4 we established that if area budget and data set size are sufficiently large, the AP may outperform the SIMD coprocessor. This outcome is supported by our findings here. For high f (e.g., 0.9 and above), the effective CPI and overall speedup breakeven points occur at relatively low area budget, and the overall speedup is relatively high. For lower f (e.g., 0.75 and below), the data set size and the area budget required for the AP to outperform the SIMD coprocessor are considerably more significant.

TABLE 5
TIMING AND AREA MODEL PARAMETERS

Parameter	Description	Value
g	fraction of memory access instructions	$0.2^{(3)}$
CPI_{CPU}	average number of cycles per instruction for instructions with no memory access	$1^{(3)}$
A_{CPU}	CPU area	10^8 (1) (3)
A_{L1}	L1 cache area	10^8 (1) (3)
d_{L1}	L1 cache access time	1 (2) (3)
d_{LLC}	LLC access time	5 (2) (3)
d_D	DRAM access time	100 (2) (3)
m_1	L1 cache miss rate	0.05(3)

(1) *Area parameters are relative to the area of SRAM bit cell*
(2) *Timing parameters are in cycles*
(3) *Values based on typical industry data in advanced technologies*

5.2 Power Modeling

The power consumption of the baseline architecture in Fig. 1(a) can be presented as a function of its LLC size $A_{LLC} = A$ based on [18]:

$$P_1(A_{LLC}) = g \cdot P_{MEM} + (1-g) \cdot P_{CPU} + P_{LEAK} \tag{24}$$

where P_{MEM} can be written as follows:

$$P_{MEM} = (1-m_1)P_{L1} + m_1(1-m_2)P_{LLC} + m_1 m_2 P_D \tag{25}$$

where P_{L1}, P_{LLC} and P_D are the power consumption of L1 cache, LLC, and off-chip DRAM access, respectively. P_{L1} is assumed to be constant and defined in TABLE 6 below; P_{LLC} to P_{L1} ratio equals the square root of the LLC to L1 areas ratio, while P_{CPU} is proportional to the CPU area [11]:

$$P_{LLC} = P_{L1} \cdot \sqrt{A_{LLC}/A_{L1}}; \qquad P_{CPU} = P_{CPUo} \cdot A_{CPU} \tag{26}$$

where P_{CPUo} is the power consumption of the baseline CPU, assumed to be constant and defined in TABLE 6 below.

The power consumption of the CPU with the LLC and the SIMD coprocessor can be written as:

$$P_2 = (1-f) \cdot P_1(A_{LLCo}) + f \cdot P_{SIMD}(A_{SIMDo}) + P_{LEAK} \tag{27}$$

where P_{SIMD} is the power dissipation of the SIMD coprocessor, which is the dynamic component of (6) above. The leakage power P_{LEAK} is defined in (8) above (with A being the sum of $A_{CPU}, A_{L1}, A_{SIMD}$ and A_{LLC}).

The power consumption of the CPU with the AP can similarly be written as follows:

$$P_3 = (1-f) \cdot P_1(A/A_{APo}) + f \cdot P_{AP} + P_{LEAK} \tag{28}$$

where P_{AP} is the power dissipation of the AP, as defined in (13) above. The leakage power P_{LEAK} is defined in (8) above (with A being the sum of A_{CPU}, A_{L1} and A_{AP}). The power parameters used for modeling are presented in TABLE 3 and TABLE 6.

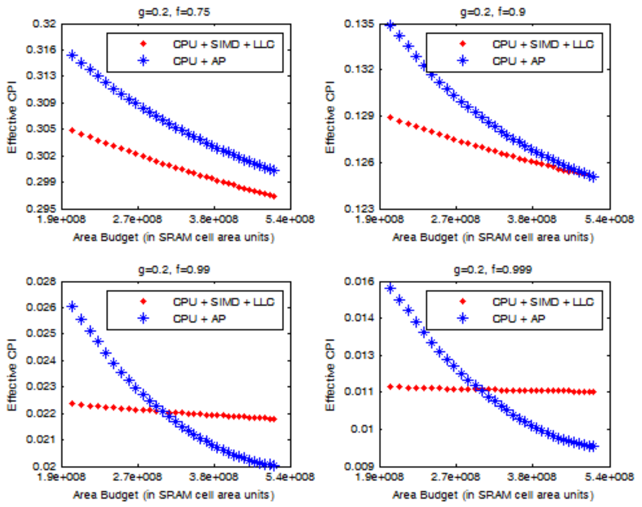
Fig. 10. Effective CPI_{SIMD} and CPI_{AP} vs. Area

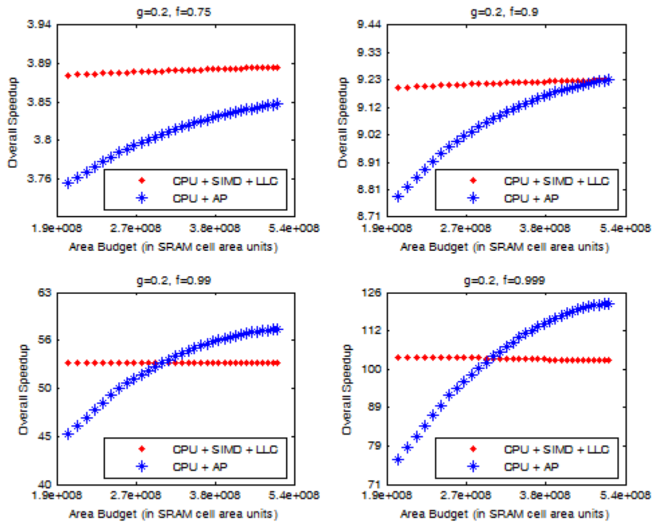
Fig. 11. SU_{SIMD} and SU_{AP} vs. Area

TABLE 6
POWER MODEL PARAMETERS

Parameter	Description	Value
P_{CPUo}	Baseline CPU power	$5 \cdot 10^{-3}$ (1) (3)
P_{L1}	L1 power	$5 \cdot 10^{4}$ (2) (3)
P_D	Power of off-chip DRAM access	10^{3} (2) (3)

(1) Power of SRAM bit cell unit over area of SRAM bit cell unit

(2) Relative to the power consumption of SRAM bit cell during write

(3) Values based on typical industry data in advanced technologies

The power consumption, the performance/power ratio and the ED product of the CPU with the LLC and the SIMD coprocessor vs. the CPU with the AP for $f = 0.75, 0.9, 0.99$ and 0.999 are shown in Fig. 12, Fig. 13 and Fig. 14, respectively. Similarly to speedup, the results for higher f are in line with the findings of Section 4.

Note the significant difference in behavior of the performance/power ratio and the ED product for higher values of f. While at lower f the AP consumes a large portion of the overall power while making small contribution to the overall speedup, for higher f the AP adds to the overall speedup quite significantly. Consequently, for f=0.999, the CPU with the AP charts exhibit a different trend *vs.* the CPU with the LLC and the SIMD curves.

Fig. 15 shows the effective CPI_{AP} vs. CPI_{SIMD} breakeven point, namely the area above which the CPU with the AP outperforms the CPU with the LLC and the SIMD, as a function of f and g. The plateau marks the region of f and g values for which a breakeven point cannot be achieved under the maximum area budget used in our analysis ($8A_{L1}$). In other words, the CPU with the AP does not have enough area to outperform the CPU with the LLC and the SIMD coprocessor. However, the SIMD processor is also less than useful in the plateau region: it is well established that conventional SIMD accelerators are inefficient in implementing low parallelizable / low arithmetic intensity (the ratio of computations to memory traffic [25] (workloads, characteristic of low f and high g (the plateau region) [30] [32].

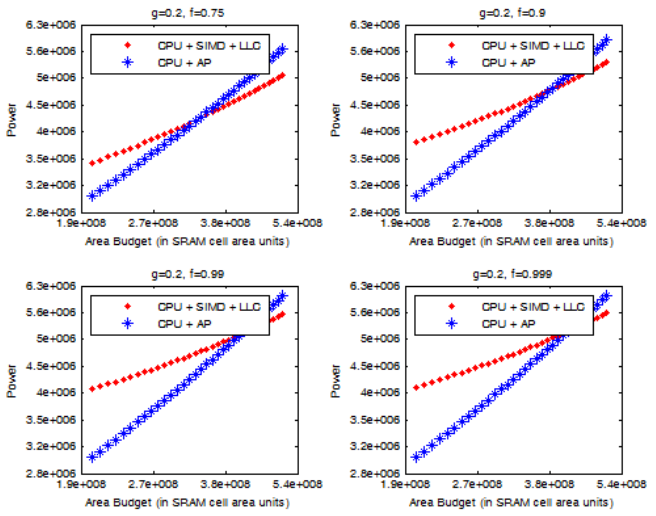
Fig. 12. Power vs. Area

On the other hand, the combination of f close to 1 and low g, which is typical for workloads with high level of fine-grain data parallelism, is advantageous for the CPU with the AP architecture, allowing it to outperform the SIMD accelerated architecture over a wide span of the area budget.

5.3 Emulation Methodology

We validate our analytic modeling findings using emulation. We follow the methodology established in [31]. To emulate the CPU with the LLC and the SIMD coprocessor of Fig. 1(b), we use a stand-alone server featuring Intel ® Pentium ® 4 processor with the SSE2 SIMD accelerator, operated under Fedora Linux. We evaluate the performance using a dense matrix multiplication kernel since it has the highest arithmetic intensity among the workloads considered in Section 4.4.1. With arithmetic intensity of $O(\sqrt{N})$ (where N is the data set size and $\sqrt{N}$ is the matrix dimension), matrix multiplication is a better fit for SIMD implementation than FFT (with arithmetic intensity of $O(\log_2 N)$ and Black-Scholes (with arithmetic intensity of $O(1)$).

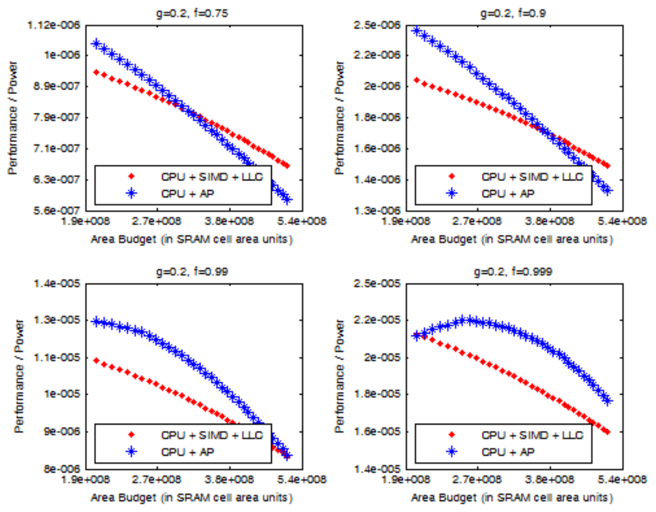

Fig. 13. Performance / Power vs. Area

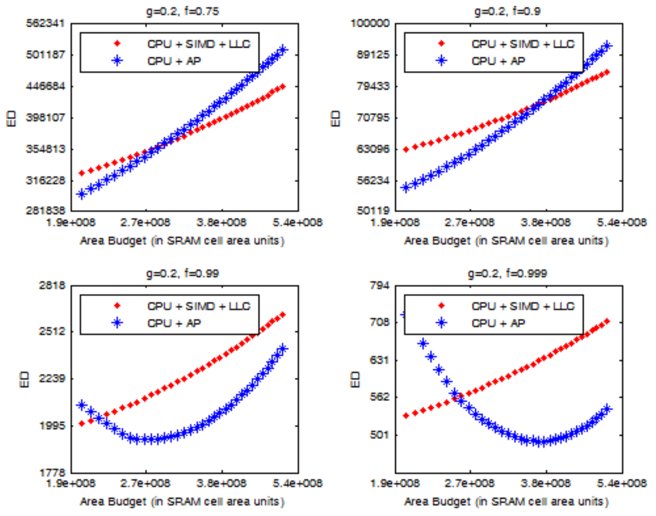

Fig. 14. ED vs. Area

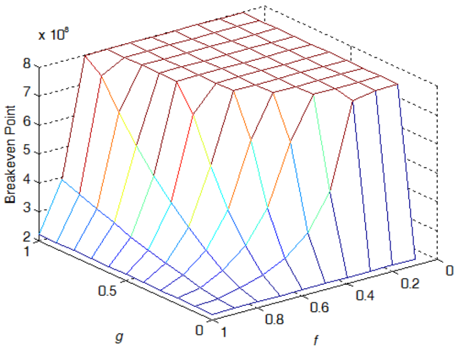

Fig. 15. Effective CPI breakeven point vs. f and g

Synchronization in SSE2 SIMD accelerator has two main components: MOV instructions that synchronize SSE2 registers with L1 data cache, and L1 to L2 synchronization when the entire data set does not fit into L1. To unwind the dependency of execution time on the cache size and hierarchy (which are constantly improving in newer CPU revisions), we deduct the data cache miss penalty time from the execution time.

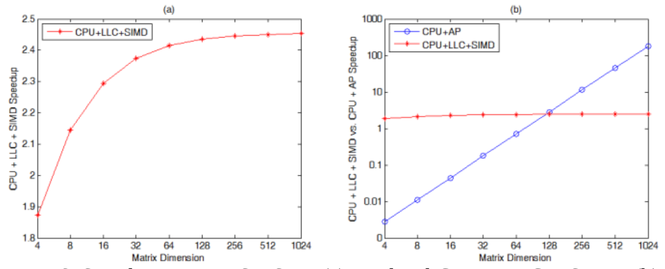

Fig. 16. Speedup vs. Data Set Size: (a) emulated CPU + LLC + SIMD, (b) emulated CPU + LLC + SIMD vs. emulated CPU + AP

The results of the CPU with the LLC and the SIMD accelerator' speedup (over naïve serial execution that takes $O(N^{1.5})$ cycles) vs. matrix dimension are presented in Fig. 16(a). They are consistent with findings of D. Aberdeen *et al.* [2] who researched matrix multiplication using Intel's SSE instruction set.

The analytical model of the CPU with the AP is verified using a combination of the cycle accurate simulation of standalone AP (Section 4.4) and the baseline CPU emulation. In this approach, the sequential fraction of the workload is emulated on the CPU, followed by the cycle-accurate simulation of the parallel fraction of the workload on the AP simulator. The sum of emulated and simulated run times gives us total execution time. The results of the CPU with AP speedup (over naïve serial execution) vs. matrix dimension are presented in Fig. 16(b), along with the speedup of the CPU with the LLC and the SIMD accelerator taken from Fig. 16(a). These results are quite expected, since the latter is limited by the SSE2 size, while the AP scales up with the data set size.

To estimate how the comparison would change if we had the ability to enlarge the vector accelerator, we separate the MOV instructions (analogous to T_S of (2)) from the rest of the code (which is analogous to $T_1 + T_C$ of (2)), and scale the latter by n_{SIMD}/n_{SSE2}, where n_{SIMD} is the size of the hypothetical SIMD accelerator and $n_{SSE2} = 4$ is the size of SSE2. We then sum up the T_S and the ideally-scaled $T_1 + T_C$, to estimate SIMD execution time. The speedup vs. area results are presented in Fig. 17. Area is received by substituting n_{SIMD} into (4). The hypothetical speedup of the CPU with the LLC and the SIMD accelerator grows with area (with n_{SIMD}), but it is still affected by synchronization (MOV part of the code) and therefore eventually underperforms the CPU with AP, in line with our analytical modeling findings.

6 Discussion and Conclusions

An associate processor is essentially a large memory with massively-parallel processing capability. It offers dual use: either the CPU accesses the data in that memory, or the data is being processed associatively within the same memory. This paper investigates the merit of using AP instead of on-chip last level cache (LLC) combined with a SIMD accelerator.

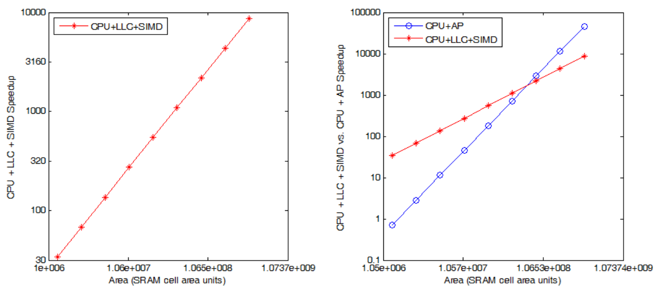

Fig. 17. Speedup vs. Area: (a) hypothetical CPU + LLC + SIMD, (b) hypothetical CPU + LLC + SIMD vs. emulated CPU + AP

Converting LLC into AP adds parallel processing capabilities to otherwise sequential architecture. The price of this conversion is the reduction (approximately halving) of the effective LLC size in its conventional sequential processing mode. However, our study shows that even for workloads with relatively low parallelism ($f \leq 0.75$), replacing the LLC by AP may lead to an overall speedup over the baseline CPU architecture. For f close to 1, such speedup can be quite significant.

An alternative way of improving the performance of parallelizable workloads is to allocate some of the constrained area budget (originally assigned to the LLC) to a conventional SIMD coprocessor. This study shows that the speedup of SIMD coprocessor is ultimately limited by data synchronization between its private memory and the shared LLC. This effect becomes more significant as the data set size and the SIMD coprocessor size grow.

The principle advantage of the AP is the unification of data storage and processing, which in most cases eliminates the need for data synchronization with a higher level cache. The AP speedup grows faster with area than the speedup of the conventional SIMD processor. Consequently, when the area budget (and the corresponding data set size) is sufficiently large, the AP may outperform the conventional SIMD coprocessor. The speedup breakeven point is in the area range of a few square millimeters to low tens of square millimeters depending on the workload, the feature size, the design specifics, etc. AP 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.

As area budget grows beyond the speedup breakeven point, AP's power is similar to that of SIMD coprocessor. However, AP seems to outperform the conventional SIMD in terms of performance/power ratio and energy-delay product over a broad spectrum of area and power budget for workloads with high data-level parallelism.

Associative processing has been known and extensively studied since the 1960s. Commercial associative processing never quite took off, because only limited amounts of memory could be placed on a single die [33]. Due to data sets and tasks of limited size, a standalone bit- and word-parallel SIMD significantly outperformed APs. However, the progress in IT industry and semiconductor technology in recent years opens the door for reconsidering APs:

- The rise of big data pushes the computational requirements to levels never seen before. The amounts of data to be processed simultaneously require a new parallel computing paradigm. This paper shows that AP's performance and efficiency improves with data set size.
- Power consumption, which used to be a secondary factor in the past, has become a principal limitation to integration and performance today. The AP is shown to achieve higher performance/power ratio and energy-delay product.
- Off-chip memory bandwidth remains to be one of the main factors limiting performance and scalability of parallel architectures. Associative processing mitigates this limitation by intertwining computing with data storage.
- In high performance dies, thermal density is becoming the limit on total computation capabilities; associative processing leads to uniform power and thermal distribution over the chip area, avoiding hot spots and enabling higher power dissipation.

Thanks to the memory integration and the feature scaling enabled by current silicon technology on one side, and the rise of big data on the other, we are at the inflection point where AP may considerably outperform conventional SIMD in both performance and power.

Acknowledgment

This research was partially funded by the Intel Collaborative Research Institute for Computational Intelligence and by Hasso-Plattner-Institut.

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] Aberdeen, D., J. Baxter. "Emmerald: a fast matrix–matrix multiply using Intel's SSE instructions." *Concurrency and Computation: Practice and Experience* 13.2 (2001): 103-119.

[3] Almási G. *et al.*, "Dissecting Cyclops: A detailed analysis of a multithreaded architecture", ACM SIGARCH Computer Architecture News 31.1 (2003): 26-38.

[4] 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.

[5] 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.

[6] Binkert N., *et al.* "The gem5 simulator." ACM SIGARCH Computer Architecture News 39.2 (2011): 1-7.

[7] Black F. and M. Scholes, "The pricing of options and corporate liabilities," Journal of Political Economy, 81 (1973), pp. 637–654, 1973.

[8] Borkar S.. "Thousand Core Chips: A Technology Perspective," *Proc. ACM/IEEE 44th Design Automation Conf. (DAC)*, 2007, pp. 746-749.

[9] Brockman J., *et al.* "A low cost, multithreaded processing-in-memory system", 31st international symposium on computer architecture, 2004.

[10] Burger D., T. Austin. "The SimpleScalar tool set, version 2.0", ACM SIGARCH Computer Architecture News 25.3 (1997): 13-25.

[11] 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.

[12] Flatt H.., K. Kennedy "Performance of Parallel Processors," Parallel Computing, Vol. 12, No. 1, 1989, pp. 1-20.
[13] Foster C., "Content Addressable Parallel Processors", Van Nostrand Reinhold Company, NY, 1976
[14] Fung Y., "Associative Processor Architecture - a Survey", ACM Computing Surveys Journal (CSUR), Volume 9, Issue 1, March 1977, Pages 3 – 27
[15] Gokhale M. *et al*., "Processing In Memory: the Terasys Massively Parallel PIM Array," IEEE Computer, 1995, pp. 23-31
[16] Gschwind M. *et. al*., "Synergistic processing in Cell's multicore architecture", IEEE Micro 26 (2), 2006, pp. 10–24
[17] Gunther N., S. Subramanyam, S. Parvu, "A Methodology for Optimizing Multithreaded System Scalability on Multi - Cores", http://arxiv.org/abs/1105.4301
[18] Guz Z. *et. al.* "Threads vs. Caches: modeling the behavior of parallel workloads", 2010 IEEE International Conference on Computer Design (ICCD), Oct. 2010, Pages: 274-281.
[19] Hall M. *et al*., "Mapping irregular applications to DIVA, a PIM-based data-intensive architecture", ACM/IEEE conference on Supercomputing, 1999.
[20] Hardavellas N. *et al*., "Toward dark silicon in servers." IEEE Micro 31.4 (2011): 6-15
[21] Hartstein A. *et. al*., "On the nature of cache miss behavior: is it square root of 2?", Journal of Instruction-Level Parallelism, 2008
[22] Hentrich D. *et al*., "Performance evaluation of SRAM cells in 22nm predictive CMOS technology," IEEE International Conference on Electro/Information Technology, 2009.
[23] Hill M., M.. Marty, "Amdahl's law in the multicore era", IEEE Computer 41 (7) (July 2008) 33–38.
[24] http://www.arm.com/products/processors/technologies/neon.php
[25] S. Kamil, C. Chan, L. Oliker,, J. Shalf, S. Williams, "An Auto-Tuning Framework for Parallel Multicore Stencil Computations", IEEE International Symposium on Parallel & Distributed Processing (IPDPS) 2010, pages 1-12.
[26] Kogge P. *et al*., "PIM architectures to support petaflops level computation in the HTMT machine", International Workshop on Innovative Architecture for Future Generation High-Performance Processors and Systems, 2000.
[27] Li H. *et al*. "An AND-type match line scheme for high-performance energy-efficient content addressable memories," IEEE Journal of Solid-State Circuits , vol. 41, no. 5, pp. 1108 – 1119, May 2006.
[28] 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.
[29] Loh G., "The Cost of Uncore in Throughput-Oriented Many-Core Processors", the Workshop on Architectures and Languages for Throughput Applications (ALTA), June 2008
[30] Luebke D., "General-purpose computation on graphics hardware", Workshop, SIGGRAPH, 2004
[31] 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.
[32] Owens J. *et al*., "GPU Computing," Proceedings of the IEEE, Vol. 96, No. 5, pp. 879-899, May 2008
[33] Pagiamtzis K. and A. Sheikholeslami, "Content-addressable memory (CAM) circuits and architectures: a tutorial and survey," IEEE Journal of Solid-State Circuits, vol. 41, no. 3, pp. 712 – 727, March 2006
[34] Pollack F., "New microarchitecture challenges in the coming generations of CMOS process technologies (keynote address)", MICRO 32, 1999
[35] Potter J., *et al.* "ASC: an associative-computing paradigm", Computer 27.11 (1994): 19-25.
[36] Potter, J. and W. Meilander. "Array processor supercomputers", Proceedings of the IEEE 77, no. 12 (1989): 1896-1914.
[37] Qing G., X. Guo, R. Patel, E. Ipek, and E. Friedman. "AP-DIMM: Associative Computing with STT-MRAM," ISCA 2013
[38] Quinn M., "Designing Efficient Algorithms for Parallel Computers", McGraw-Hill, 1987, page 125.
[39] 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
[40] Scherson I. *et al*., "Bit-Parallel Arithmetic in a Massively-Parallel Associative Processor", IEEE Transactions on Computers, Vol. 41, No. 10, October 1992
[41] Sheaffer J. *et al*. "Studying thermal management for graphics-processor architectures," ISPASS 2005
[42] Steinkraus D., L. Buck, P. Simard, "Using GPUs for machine learning algorithms," IEEE ICDAR 2005.
[43] Sterling T., H. Zima. "Gilgamesh: a multithreaded processor-in-memory architecture for petaflops computing." , ACM/IEEE Conference on Supercomputing, 2002.
[44] Suh J. *et al*. "A PIM-based multiprocessor system", 15th International Symposium on Parallel and Distributed Processing, 2001.
[45] Wentzlaff D., *et al*., "Core Count vs. Cache Size for Manycore Architectures in the Cloud. Tech. Rep. MIT-CSAIL-TR-2010-008, MIT, 2010.
[46] Yavits L. *et al*., "The effect of communication and synchronization on Amdahl's law in multicore systems", http://arxiv.org/abs/1306.3302
[47] 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/publication-link/index/id/633
[48] Yavits L. *et al*., "Thermal analysis of 3D associative processor", http://arxiv.org/abs/1307.3853v1
[49] Yavits L. *et al*., "Cache Hierarchy Optimization", IEEE Computer Architecture Letters, July 2013

Leonid Yavits received his MSc in Electrical Engineering from the Technion. After graduating, he co-founded VisionTech where he co-designed a single chip MPEG2 codec. Following VisionTech's acquisition by Broadcom, he co-founded Horizon Semiconductors where he co-designed a Set Top Box on chip for cable and satellite TV.

Leonid is a PhD student in Electrical Engineering in the Technion. He co-authored a number of patents and research papers on SoC and ASIC. His research interests include Processing in Memory and 3D IC design.

Amir Morad received his BSc and MSc in Electrical Engineering from the Technion. Amir co-founded VisionTech, a major provider of ICs for set top boxes market. Following VisionTech's acquisition by Broadcom, Amir co-founded Horizon Semiconductors, where he co-designed SoCs for HD cable and satellite set top boxes.

Amir is a PhD student in Electrical Engineering in the Technion. He co-authored a number of patents and research papers on SoC and ASICs. His research interests include analytical modeling and optimization of many-core architectures.

Ran Ginosar received his BSc from the Technion and his PhD from Princeton University. After conducting research at AT&T Bell Laboratories, he joined the Technion where he is now professor at the Electrical Engineering department and a head of the VLSI Research Center.

Professor Ginosar has been a visiting Associate Professor with the University of Utah and co-initiated the Asynchronous Architecture Research Project at Intel (Oregon). He has co-founded a number of VLSI companies. Professor Ginosar has published numerous papers and patents on VLSI. His research interests include VLSI architecture, asynchronous logic and synchronization.