(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
19 September 2013 (19.09.2013)

WIPO | PCT

(10) International Publication Number
WO 2013/136259 A2

(51) International Patent Classification:
G06F 15/167 (2006.01)

(21) International Application Number:
PCT/IB2013/051929

(22) International Filing Date:
12 March 2013 (12.03.2013)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
PCT/IB2012/051150 12 March 2012 (12.03.2012) IB

(71) Applicant: **ECOLE POLYTECHNIQUE FEDERALE DE LAUSANNE (EPFL)** [—/CH]; EPFL-TTO, Quartier de l'innovation - J, CH-1015 Lausanne (CH).

(72) Inventors: **DOGAN, Ahmed Yasir**; Ch. de Bonne Espérance 28, CH-1006 Lausanne (CH). **CONSTANTIN, Jeremy**; Av. de Mont-Goulin 11, CH-1008 Prilly (CH). **BURG, Andreas**; Rte de Vallaire 112, CH-1024 Ecublens (CH). **ATIENZA ALONSO, David**; ch. de Montaney 44, CH-1024 Ecublens (CH).

(74) Agent: **WEIHS, Bruno**; Andre Roland S.A., P.O. Box 5107, CH-1002 Lausanne (CH).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

[Continued on next page]

(54) Title: Ultra-Low Power Multicore Architecture For Parallel Biomedical Signal Processing

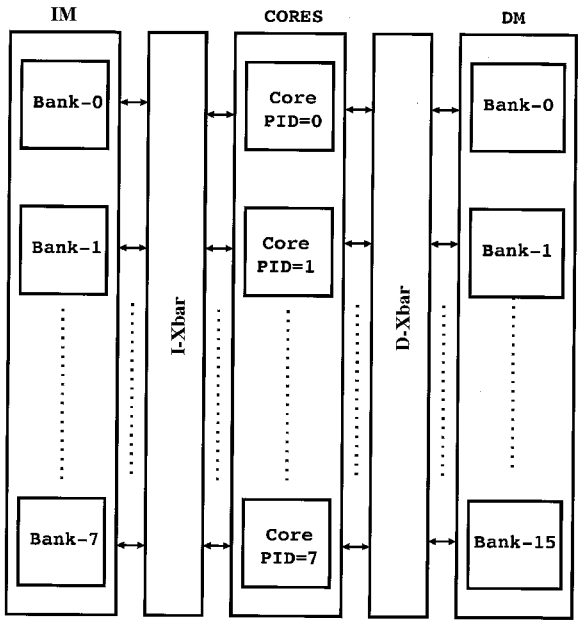

FIG. 1

(57) Abstract: A multi-core architecture with ultra-low power consumption is needed for a wide variety of applications, especially in the bio-medical domain. In this patent, an ultra-low power multi-core architecture is presented: it is composed of one or more cores, several (one or more] shared multi-banked instruction and data memories, and flexible crossbar interconnects. The interconnect includes a selective broadcasting mechanism, enabling coordinated multiple accesses to the shared memories, thus energy savings in the memory hierarchy and in the interconnects. The memory hierarchy enables power gating of the unused banks to lower leakage power. The core instruction set of the novel architecture has been customized to exploit the specific features of bio-signal events, as well as the highly parallel computation opportunities of bio-signal processing characteristics. In addition to near threshold computing, the proposed architecture also exploits other advanced low-power features, which lead to further energy savings. The architecture has a synchronization method and unit that allows power- efficient handling of data dependencies when code is parallelized across the multiple processing cores.

WO 2013/136259 A2

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

Ultra-Low Power Multicore Architecture For Parallel Biomedical Signal Processing

0 Technical field

The invention is in the field of multi-core processors.

1 Background

According to the World Health Organization, cardiovascular and modern human behavior-related diseases are the major cause of mortality worldwide. Close and potentially continuous medical supervision is strongly needed to control these types of diseases. They are thus expected to soon require healthcare costs and medical management needs that are unsustainable for traditional healthcare delivery systems. Personal health monitoring systems are poised to offer large-scale and cost-effective solutions to this problem.

Wireless body sensor networks (WBSNs) are the enabling technology for such personal health systems. A WBSN for health monitoring consists of a number of lightweight sensor nodes attached to the human body, where each node is responsible for processing a specific low rate physiological signal. For instance, one of the most important physiological signals is the electrocardiogram (ECG), which is typically acquired at sampling rates between 125 Hz and 1 kHz to capture the often important details of the waveform. In order to monitor the heart rate for extended periods of time (up to multiple days or weeks), an ultra-low-power (ULP) design with embedded biomedical signal processing for feature extraction on the sensor node is necessary to reduce the costly signal storage or transmission to the essence.

1.1 Prior Art

An effective technique to achieve energy efficiency is supply voltage scaling. In the literature, voltage scaling has been extensively analyzed, including its limita-

tions and disadvantages [1]. One of the main issues with low-voltage operation is performance degradation, which can limit the degree of use of voltage scaling for a given processing requirement. Parallel computing using multiple cores can alleviate this issue, provided that the algorithms to be executed can be parallelized. To this end, the work presented in [2] explored the power/performance tradeoffs between sequential and parallel near-threshold computations for various biomedical signal-processing requirements. The comparison shows that multi-core systems do not only solve the performance degradation problem, but also achieve good energy efficiency. In [3], the authors proposed a near threshold computing (NTC), cluster-based multi-processor architecture with a shared cache that operates at a higher supply voltage to be able to serve multiple cores at the same time. Also, Yu et al [4] introduced a sub/near threshold processor specialized for low-energy mobile image processing using architecture-level parallelism to compensate the performance loss.

The multi-core architectures in [3] and [4] are built for a more general use, however they achieve limited energy efficiency, notably at low workloads.

1.2 BRIEF DESCRIPTION OF THE INVENTION

In a first aspect, the invention provides a multi-core processor for processing a plurality of input data, comprising one or more processing cores; one or more shared multi-banked instruction memories; one or more shared multi-banked data memories; one or more instruction crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more shared multi-banked instruction memories; one or more data crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more multi-banked data memories; selective broadcast means configured to implement a selective broadcast of instructions from the one or more instruction memories to the one or more processing cores using the one or more interconnecting instruction crossbar interconnects; and analyzing means for determining whether a same

instruction is to be performed on each of the plurality of input data, or whether for each of the plurality of input data a different instruction is to be performed. The analyzing means and the selective broadcast means are functionally connected, whereby when the analyzing means determines that the same instruction is to be performed on each different input data, the selective broadcast means reads the same instruction only once from a single one of the one or more instruction memories and broadcasts the instruction to each of the one or more interconnected processing cores to be performed; and when the analyzing means determines that different instructions are to be performed implements no broadcast of instructions.

In a first preferred embodiment, the multi-core processor further comprises at least a memory management unit configured to enable each processing core to access its individual working data with a single instance of a compiled application executed by all the cores, the memory management unit comprising a processor identity number attributing means that attributes a unique processor identity number for each core; data placing means configured to place the working data in the one or more shared multi-banked data memories by using the processor identity number; and, translating means configured to translate a same decoded address from different ones of the one or more processing cores to different physical memory addresses according respectively to the processor identity number.

In a second preferred embodiment, the multi-core processor further comprises interleaving means configured for supporting interleaving of instructions across the banks of the one or more shared multi-banked instruction memories or of the one or more shared multi-banked data memories to minimize instruction memory conflicts in the case of synchronization lost between the processing cores.

In a third preferred embodiment, the multi-core processor further comprises mapping means configured for supporting mapping of instructions to the minimum number of instruction memory banks of the one or more shared multi-

banked instructions memories; and power gating means configured to identify memory banks from the one or more shared multi-banked instruction memories or the one or more shared multi-banked data memories which are not accessed, and enable gating of the memory banks that are not accessed.

In a fourth preferred embodiment, the multi-core processor, further comprises mapping means configured for supporting mapping of instructions to the minimum number of instruction memory banks of the one or more shared multi-banked instructions memories; power gating means configured to identify memory banks from the one or more shared multi-banked instruction memories or the one or more shared multi-banked data memories which are not accessed, and enable gating of the memory banks that are not accessed; and an instruction addressing schemes means configured to analyze the significant bits of addresses to map onto the one or more shared multi-banked instruction memories or of the one or more shared multi-banked data memories and to in case it is found that the least significant bits of the address to map are chosen, then the interleaving means is to be used, and in the case it is found that the most significant bits are chosen then the mapping means and the power gating means are to be used.

In a fifth preferred embodiment, the multi-core processor further comprises setting means for setting one or more of the processing cores in a status of waiting processing cores; memory access conflict detecting means configured to detect memory access conflicts for memory requests, whereby in case a memory access conflicts is detected, the memory requests are served alternately, while the waiting processing cores are stalled using clock gating to avoid active power consumption.

In a sixth preferred embodiment, the instruction crossbars and data crossbars are a Mesh-of-Trees (MoT) interconnection network to support high-performance communication between processors and memories.

In a seventh preferred embodiment, each of the one or more processing core comprises a 3-stage pipeline: fetch, decode and execute stages, and 3 external memory ports: one for instruction read, one for data read, and one for data write, all accessible in the same cycle, with each instruction being executed in one cycle, by having complete data bypassing inside the processing core for registers as well as memory write-back data.

In a eighth preferred embodiment the one or more shared multi-banked data memories is split into at least a shared section and a private section; each processing core's private data is stored in the private section, while data shared across the different processing cores is stored in the shared section; with the decoded address from an instruction directly used as physical memory address for the shared section and an address translation mechanism used to obtain the physical memory address for the private section.

In a ninth preferred embodiment the private sections of each processor core are located into different memory banks, thus the private section is accessed without conflicts.

In a second aspect, the invention provides a method for processing a plurality of input data with a multi-core processor, wherein the multi-core processor comprises one or more processing cores; one or more shared multi-banked instruction memories; one or more shared multi-banked data memories; one or more instruction crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more shared multi-banked instruction memories; one or more data crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more multi-banked data memories; the method comprising the steps of selectively broadcasting instructions from the one or more instruction memories to the one or more processing cores using the one or more interconnecting instruction crossbar interconnects; and analyzing whether a same instruction is to be performed on each of the plurality of input data, or whether for each of the plurality of input data a different instruction is to

be performed. When the analyzing determines that the same instruction is to be performed on each different input data, the step of selectively broadcasting comprises reading the same instruction only once from a single one of the one or more instruction memories and broadcasting the instruction to each of the one or more interconnected processing cores to be performed; and when the analyzing determines that different instructions are to be performed the step of selectively broadcasting implements no broadcast of instructions.

In a tenth preferred embodiment the method further comprises steps of putting to sleep processing cores that finish operations and wait for the completion of the execution of one or more cores due to data-dependencies such as to save power consumption, and waking up sleeping processing cores when the corresponding cores complete their operations.

In an eleventh preferred embodiment, the following synchronization method is added to support synchronization of the different one or more processing cores:

a. determining for a given application, data-dependent code sections, marking the beginning of the data-dependent code sections as check-in points, and marking the corresponding ending of the data-dependent code sections as check-out points;
b. assigning for each data-dependent code section check-in and check-out pair, a data memory position to trace the synchronization process, wherein, both the processing core identities and total number of cores (the core counter), currently running the corresponding data-dependent program section are stored;
c. once a processing core arrives to a check-in point, modifying by the processing core of the corresponding data memory position by adding a signature of the processing core as well as incrementing the core counter;
d. once a processing core arrives to the corresponding checkout point for the check-in point identified in the preceding step, decrementing the core counter at the data memory, and waiting at the processing core for the other expected cores to reach to the same checkout point,

whereby the waiting processing cores go to sleep immediately after decrementing the core counter; and

e. once all the expected cores running the same data-dependent program section reach to the check-out point (the core counter becomes zero) then the processing cores that are executing the waiting step are woken up to continue processing.

In a twelfth preferred embodiment, the method further comprises steps of implementing an instruction set architecture comprising of one or more arithmetic logic unit instructions, whereby the said arithmetic logic unit instructions work on 3 operands, using the exact same addressing mode options for each instruction, thereby decoupling operand fetch logic and the arithmetic operation; and the instruction word encoding is designed to be regular with fixed bit positions allowing for very efficient decoding of the operands and the different instruction words.

In a thirteenth preferred embodiment, the synchronization method further comprises appending the core ids and incrementing the core counter according to a number of parallel requests, when multiple processing cores reach a check-in point at the same time; for multiple check-out requests, decrementing the core counter accordingly; and writing the calculated value for check-in/check-out requests to the data memory through one of the data write ports used by the processing cores, whereby the data write port is assigned for the core which has the minimum identity number among the other synchronous cores.

In a fourteenth preferred embodiment, each processing core locks the assigned memory position for a check-in/check-out process until the content has been modified to avoid possible hazards among the cores, by generating an external signal (lock output signal), indicating a lock request; and the signal locks the memory position until it has been modified with the new value for a safe message passing among the cores in sequential check-in/check-out processes.

In a fifteenth preferred embodiment, the method further comprises the steps for implementing an address translation mechanism, wherein any address targeting the private section of the data memory is shifted by $\log_2 N_{AC}$ and the processing core identity is appended, where N_{AC} stands for the number of active cores; then the address is shifted by $\log_2(N_{TMB}/N_{AMB})$, where N_{TMB} and N_{AMB} are the total number of data memory banks and the number of active data memory banks.

In a sixteenth preferred embodiment of the multi-core processor, each processing core performs the same operation, such as signal filtering, on different channels of the same biological signal, such as ECG or EEG, with each processing core sharing the same interactions that are broadcasted to them from the instruction memory, and different data is obtained from the data memory.

In a seventeenth preferred embodiment, each core operates at near-threshold voltage, achieving ultra-low power operation.

The invention, described in the present description relates to a multi-core architecture design and a method for processing a plurality of input data with a multi-core processor that achieve ultra-low-power consumption for bio-medical applications. The proposed multi-core architecture comprises one or more cores, several (one or more) shared multi-banked instruction and data memories, and flexible crossbar interconnects. The core instruction set of the novel architecture is customized to exploit the specific features of bio-signal events, as well as the highly parallel computation opportunities of bio-signal processing characteristics. In addition to NTC, the proposed architecture also exploits other advanced low-power features, which lead to further energy savings. In particular, the interconnect includes a broadcasting mechanism, enabling coordinated multiple accesses to the shared memories, thus energy savings in the memory hierarchy and in the interconnects. Moreover, the memory hierarchy enables power gating of the unused banks to lower leakage power.

Power/performance trade-offs of the proposed architecture are explored for different target workloads. The results show that the proposed multi-core solu-

tion achieves between 38.8% and 45.7% power savings with respect to the state-of-the-art for low (i.e., 5 kOps/s) to high workloads (637 MOps/s) due to the combination of multiple power management options.

The benefit of instruction and data broadcasting (up to 45.7% active power savings [5]) relies on lockstep execution of algorithm parts that can be executed using the single instruction multiple data (SIMD) processing scheme. Therefore, these substantial power savings can only be achieved by synchronous instruction execution, which for many parallel applications is not automatically given, due to data dependent program flow as well as data memory access conflicts, which bring the processing cores out of lockstep. Barrier insertion techniques are widely used in parallel computing architectures to achieve synchronization [5]. Cores are synchronized at certain barrier points of execution, i.e. a core is stalled until all other cores reach the same point. These techniques however are generally employed to avoid data dependency conflicts between processing cores, e.g. at the end of a parallelized loop construct. We propose to apply the technique of barrier synchronization to achieve reduced power consumption on the ULP multi-core architecture. In the literature, many software-only [6] and software-hardware hybrid implementations of barriers are proposed [7]. However, these proposed techniques require rather complicated mechanisms for an embedded ULP platform, where both energy efficiency and low complexity due to real-time applications, are critical.

We introduce the usage of barrier synchronization for ULP computing architectures, reducing power consumption on multi-channel data processing platforms, through lockstep SIMD code execution, maximizing the effects of instruction and data broadcasting, where possible. A hybrid lightweight hardware-software barrier technique is proposed to synchronize code execution among cores. The proposed hardware-software solution involves a hardware synchronizer in conjunction with an instruction set extension (ISE) for the processing cores. Moreover, we offer configurability for the number of active processing cores and memory sizes, to reduce leakage power consumption.

1.3 BRIEF DESCRIPTION OF THE FIGURES

The invention will be better understood in light of the description of preferred embodiments and the attached figures in which:

FIG 1 illustrates an example of the proposed multi-core architecture according to the invention;

FIG 2 show a custom designed core (Tamarisc) architecture;

FIG 3 contains a graphs showing power distribution in the **mc-ref** architecture;

FIG 4 shows an example of Data-Dependent Code Section;

FIG 5 illustrates an improved multi-core architecture with hardware synchronizer;

FIG 6 is a graph containing normalized power consumption at various workloads;

FIG 7 is a bar graph illustratubf dynamic vs leakage power consumption at various workloads; and

FIG 8 is a graph showing total power consumption without and with synchronizer.

1.4 DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

1.4.1 Bio-potentials Processing Features

Signal processing on wearable personal health monitoring systems consists mostly of arithmetic computations with relative complexity on single- or multi-input biological signals. Hence, it has been recently shown that they can be optimized to run in real-time on typical embedded low-power microcontrollers. For instance, multi-lead ECG signals, using a complex multi-scale wavelet transform algorithm, can be realized on a commercially available personal health monitoring system node with limited computation capability [8]. In fact, multi-lead biological signals analysis is often needed to obtain an accurate view of biological events. However, the analysis of these multi-lead signals entails considerably parallel computation opportunities, which can be exploited on multi-core processing platforms.

To illustrate our design, we use a reference: a real-time multi-lead ECG processing application, which comprises two components, compressed sensing (CS) and Huffmann coding. CS performs a 50% compression on a block of 512 samples of ECG data (sampled at 250 Hz) per lead whereas the Huffmann coding part encodes the compressed data further for wireless transmission. The benchmark operates on 8 leads in parallel (one core per lead) to make the system more accurate and resilient to noise artifacts. The CS part follows always the same program flow independent of the input data, however the Huffmann coding adds a short section of data-dependent program flow.

For a single lead, the benchmark uses a total of 552 bytes for instructions and 16922 bytes for data. This data consist of two parts, namely working data (2586 bytes) and read-only data (14336 bytes). More specifically, the read-only data is comprised of 3 lookup tables (LUTs), i.e, a random vector for CS (12288 bytes)

with a linear access pattern and two data dependent LUTs (1024 bytes each) for the Huffmann coding.

1.4.2 Processing Architecture

The single-core and multi-core configurations include the same processing unit (PU) and a data memory (DM). However, the multi-core processing platform also involves a central data crossbar interconnect (D-Xbar), connecting the PUs with the shared DM.

The multi-core architecture proposed in [2], herein referred to as **mc-ref** architecture, is a starting point of our investigation to explore an energy-efficient multi-core architecture for biomedical applications. Similar to the **mc-ref** architecture, our proposed architecture, shown in FIG. 1, involves multiple cores sharing a data memory (DM), divided into multiple memory banks via a central data crossbar (D-Xbar) interconnect. However, as opposed to the **mc-ref** architecture, our proposed architecture offers instruction memory (IM) sharing via a central instruction crossbar interconnect (I-Xbar) similar to the D-Xbar. To this end, the proposed architecture includes memory management units (MMUs) enabling each core to access its individual working data with a single instance of a compiled application executed by all the cores. A unique processor identity (PID) for each core is used to place the working data in the DM. In case of memory access conflicts, the requests are served alternately, while the waiting cores are stalled using clock gating to avoid unnecessary active power consumption. The following subsections explain in detail the features of the proposed architecture.

1.4.3 Low-Power Core: TamaRISC

The core we have developed for the presented system architecture is a custom-designed reduced instruction set computing (RISC) architecture for bio-signal analysis. It is shown in FIG. 2. The core architecture focuses on minimizing the instruction set complexity, while still providing enough hardware support, especially regarding addressing modes, for efficient execution of the target biomedi-

cal applications. The processor has a 3-stage pipeline (fetch, decode and execute stages). The core operates on a data word length of 16-bit, comprises 16 working registers and 3 external memory ports (one for instruction read, one for data read, and one for data write, all accessible in the same cycle. The instruction word length is 24-bit, and every instruction has a single-word size. All instructions are executed in one cycle, guaranteed by the complete data bypassing inside the core for registers as well as memory write-back data.

The instruction set architecture (ISA) comprises a total of 11 unique instructions, with 8 arithmetic logic unit (ALU), 2 program flow and 1 general data-move instructions. The ALU supports addition, subtraction, shift, logical AND, OR and XOR, as well as full 16-bit by 16-bit multiplications. All ALU instructions work on 3 operands, using the exact same addressing mode options for each instruction, which reduces the complexity of the architecture, since the operand fetch logic and the arithmetic operation are completely decoupled. Additionally, the instruction word encoding is designed as regular (fixed bit positions) and as simple as possible to allow for very efficient decoding of the operands and the different instruction words in general. The supported addressing modes are register direct, register indirect (with pre- or post-increment and decrement), as well as register indirect with offset.

Branching is possible in direct and register indirect mode, as well as by an offset with 15 different condition modes (dependent on the processor status flags: carry, zero, negative and overflow).

1.4.4 Crossbar Interconnects

The crossbar interconnects, both the D-Xbar and I-Xbar, are a Mesh-of-Trees (MoT) interconnection network to support high-performance communication between processors and memories. The interconnects are intended to connect a number of processing cores to a multi-banked memory on data and instruction sides. The total memory access latency is one clock cycle, however in case of mul-

tiple conflicting requests, for fair access to memory banks, a round-robin scheduler arbitrates access and a higher number of cycles is needed depending on the number of conflicting requests, with no latency in between.

To reduce memory access time and increase shared memory throughput, a read broadcast can be used and no extra cycles are needed when such a broadcast occurs.

1.4.5 Instruction Memory Organization

As shown in FIG. 3, a significant amount of power (54% of the total power consumption) is consumed by the IM in the **mc-ref** architecture while executing the benchmark.

This is due to dedicated IM banks for each core. Biomedical signal processing platforms often execute the same operations on different input data on multiple cores, thus the same instructions are read from the IM banks if the cores are in synchronization. Nevertheless, all the IM banks are accessed, thus power would be wasted. However, this power can be reduced by minimizing the number of accesses to the IM banks by reading the identical instructions only once and broadcasting them to all the cores. The IM banks have mostly identical content, only differ in few instructions due to different memory locations for the working data.

However, a single instance of the compiled application can be used for all the cores, provided that the cores can access different working data with the same instruction words. To this end, MMUs translate the same decoded address to different physical memory addresses according to the PID numbers. Using the same compiled application for all the cores facilities IM sharing via the I-Xbar. As opposed to the **mc-ref** architecture, each core can access the entire IM, 96 kBytes in total.

However, the instruction broadcasting is beneficial if the cores remain in synchronization. This can be limited due to possible DM conflicts. Hence to exploit instruction broadcasting, we reorganize the DM and data broadcasting is applied to minimize the conflicts.

Our proposed multi-core architecture enables two different IM organizations: interleaved and banked instructions. The first one, **ulpmc-int**, interleaves instructions across the banks to minimize IM conflicts in the case of synchronization lost between the cores. The second one, **ulpmc-bank**, maps instructions into the minimum number of IM banks. The **ulpmc-bank** is intended to reduce the memory leakage power consumption, by applying power gating to the unused IM banks, which has a significant impact on the overall power consumption at low workloads.

The only architectural difference between the **ulpmc-int** and **ulpmc-bank** is the selection bits assignment for the IM banks. The **ulpmc-int** selects the IM banks based on the least significant bits while the **ulpmc-bank** chooses the IM banks according to the most significant bits.

1.4.6 Data Memory Organization

The working data sets are individual for each core whereas read-only data can be shared between all cores. Application profiling of the benchmark for DM accesses shows a distribution of 76% private versus 24% shared accesses. Out of the shared accesses, 92% are on the CS random vector while 8% are on the Huffmann coding LUTs. To minimize data access conflicts, the proposed architecture offers two different sections in the DM: shared and private sections. The size of the private and shared sections is configurable and determined during compilation of applications. The working data is separate for each core, thus it is placed in the private section whereas the shared LUTs are linked into the shared section. The decoded address is used as the physical memory address for a shared

section access, whereas the address translation is applied to generate the physical memory addresses for a private section access. The private sections of each core are located into different memory banks, thus the private section is accessed without conflicts.

Moreover, shared data (read-only data) is interleaved across the memory banks to minimize conflicts when shared data is accessed. More specifically, the CS random vector accesses are with a linear pattern thus can be performed conflict free with the data broadcasting, provided that the cores are in synchronization. However, the data dependent Huffmann coding LUTs can produce data conflicts, due to all 8 cores processing different sample data.

1.4.7 Proposed Synchronization Technique

The benefit of instruction and data broadcasting relies on synchronous code execution which enables coordinated multiple accesses to shared memories. A synchronization loss between the cores limits this substantial benefit. In addition, the degree of voltage scaling can be limited for a given workload since asynchronous code execution leads to increased IM conflicts, and thus a higher clock frequency requirement for a given workload. Loss of synchronization between the cores is mainly due to two reasons: data access conflicts and data-dependent program flow. A data access conflict occurs when a DM bank is accessed by more than one core (i.e. a shared data access). In this case, the data-served cores continue code execution while the rest of the cores wait for data to be served. As a result, synchronization is lost. On the other hand, many applications involve data-dependent code sections in their nature, which lead to conditional execution of different parts of the code, and, consequently, synchronization is lost among the cores.

The synchronization loss issue due to data access conflicts is simply solved by enhancing the data serving policy in the data interconnect. This enhancement only applies to data accesses for synchronous cores. All synchronous cores are

stalled until all of them have been served successfully. To detect whether the cores are synchronous, their program counters are compared.

On the other hand, synchronization loss due to data-dependent program flow is a rather complicated issue and needs to be addressed carefully. To this end, we propose to resynchronize the cores at the end of data-dependent code sections with a synchronization framework, detailed as follows:

- Firstly, the data-dependent code sections in applications are determined. For instance, for a given application with a program flow as depicted in FIG. 4, A-A', B-B' and C-C' are the pairs which show the data-dependent code sections. A, B and C (check-in points) are the beginning of the data-dependent code sections, whereas A', B' and C' (checkout points) are the points where corresponding data-dependent code sections end. Indeed, A', B' and C' are the program points where the code execution is resynchronized.

- Secondly, for each data-dependent code section pair (i.e. A-A', B-B' and C-C') a DM position is assigned to trace the synchronization process. In these memory positions, both the core identities and total number of cores (the core counter), currently running the corresponding data-dependent program section are stored. More specifically, the first byte stores the core identities, whereas the second byte is the core counter, which traces the number of cores executing the corresponding section. To this end, once a core arrives to a check-in point (i.e A, B or C) it modifies the corresponding DM position by adding its signature as well as incrementing the core counter.

- Lastly, cores arriving to the checkout points (i.e. A' or B' or C') decrement the core counter and wait for the other expected cores to reach to the same checkout point. The waiting cores go to sleep immediately after decrementing the core counter. Once all the expected cores running the same data-dependent program section reach to the checkout point (the core

counter becomes zero) then the cores are woken up to continue processing.

In order to apply the proposed framework, a hardware synchronizer and ISE in the cores are required. The following sub-sections will explain these enhancements in detail.

1.4.8 Synchronization Unit

In the proposed synchronization framework, the cores modify the assigned memory word for a check-in/check-out process. Usually, several cores reach to a check-in point together and then the cores may branch different conditional code sections. When the conditional code sections end, the cores reach to the corresponding checkout point, which can happen at same time or with some delay between the cores depending on the taken conditional code sections.

To achieve a check-in/check-out a core needs two clock cycles since a memory read and then a memory write is needed. When several cores apply a check-in/check-out for the same synchronization point at the same time, these check-in/check-out processes are achieved sequentially. Therefore, it causes $2*(N_{synch}-1)$ times of stall cost in the architecture, where N_{synch} stands for the number of synchronous cores accessing to the same check-in/check-out points. However, this is solved with the hardware synchronizer, shown in FIG. 5, which handles multiple check-in/check-out requests to the same synchronization point. To indicate a check-in/check=out request for a synchronization point, the cores rise an external output.

The synchronizer merges multiple check-in/check-out requests for a synchronization point and modifies the assigned memory position, accordingly. More specifically, for multiple check-in requests, the synchronizer appends the core ids and increments the core counter according to the number of parallel requests. On the other hand, for multiple check-out requests the synchronizer decrements

the core counter accordingly. The calculated value for check-in/check-out requests is written to the DM through one of the data write ports used by the cores (i.e. the data write port assigned for the core which has the minimum identity number among the other synchronous cores). Thanks to the parallel access handling in the synchronizer, independent of the number of synchronous cores only two clock cycles are required for multiple check-in/check-out processes. However, for the sequential accesses to a synchronization point (occurring when the cores reach to the same point at different times) each core still needs two clock cycles to achieve a check-in/check-out process. In this case, each core locks the assigned memory position for a check-in/check-out process until the content has been modified to avoid possible hazards among the cores. To this end, the cores generate an external signal (lock output signal), indicating a lock request (FIG. 6)

When all the expected cores reach to a check-out point, the core counter becomes zero. In this case, the synchronizer wakes up all the cores indicated by the core id flags, and the corresponding memory word is initialized by zero.

1.4.9 ISE in the Processing Cores

We have customized the instruction set architecture of the core by adding two new instructions **SINC** and **SDEC** and a core output (lock signal) to support the check-in and check-out processes. More specifically, **SINC** and **SDEC** are dedicated to the check-in and check-out processes, respectively, whereas the lock output signal is used for a safe sequential check-in/check-out process. In addition, a specific core register R_{sync} is used to store the base address of the assigned DM array used for synchronization purposes. The details of the extensions are as follows:

SINC: The assembler semantic is: **SINC** *#literal*. The literal stands for the synchronization point index, which addresses the position of the synchronization point in the assigned DM array. The instruction reads data from the memory address, calculated by adding the literal value to the R_{sync} base address. This read

data is forwarded (without any manipulation) to the write port, and the output indicating a check-in request is activated.

SDEC: The assembler semantic is: **SDEC** *#literal*. It is similar to **SINC**, but the output indicates a check-out request. In addition, after requesting the check-out the core is going into sleep mode until a wake up occurs (i.e. when all expected cores reach to the checkout point).

Lock Output Signal: This output is activated when **SINC** and **SDEC** instructions are executed. This signal locks the memory position until it has been modified with the new value for a safe message passing among the cores in sequential check-in/check-out processes.

1.4.10 Synchronization Points Insertion

The proposed synchronization framework requires the resynchronization points to be determined and the code to be instrumented accordingly. While these steps are manually implemented in this study by inserting assembly instructions, they can be in principle automated during the compilation process. For a given code the check-in and checkout instructions can be inserted as shown below.

```
1 ....
2 SINC #(Synchronization Point X)
3 if (condition)
4 {//if operations... }
5 else
6 {//else operations... }
7 SDEC #(Synchronization Point X)
8 ....
```

These check-in and check-out instructions can be inserted in any code segments where the cores are intended to resynchronize, including but not limited to while loops, for loops, case and if-elsif-else statements, etc..

1.4.11 Proposed Leakage Power Saving Technique

The baseline multi-core architecture allows configuring the number of active IM banks to reduce the leakage power consumption, crucial especially at low workloads. To further improve the power consumption at low workloads, the number of active cores and IM banks are configured depending on the requirements of the application. We apply an address translation to generate the physical memory addresses, enabling configuration of the number of active DM banks and cores. The address translation is performed as follows: Firstly, any address targeting the private section of the DM is shifted by $\log_2 N_{AC}$ and the core identity is appended where N_{AC} stands for the number of active cores. Then the address is shifted by $\log_2(N_{TMB}/N_{AMB})$, where N_{TMB} and N_{AMB} refer to the total number of DM banks and the number of active DM banks, regardless of shared or private data access. This feature allows activating any number of cores and DM banks, which are a power of two. The remaining of the cores and DM banks are power gated, similar to unused IM banks, which dramatically reduces power consumption at low workloads.

2 Experimental Setup and Results

To explore the power/performance trade-offs between the architectures, we have built the reference **mc-ref** and the proposed designs. The designs are implemented in a 90 nm low leakage process technology trading peak performance for significant leakage power reduction, especially in the memories. The reference benchmark is executed on the designs for various workloads while exploiting voltage scaling to accomplish minimum power solutions.

The scaling of the operating voltages is limited to the transistor threshold voltage level to avoid performance variability and functional failure issues occurring mainly at sub-threshold voltages. The power values at scaled voltages are calculated regarding the fact that the power decreases with the square of the supply voltage.

The processing core is described in LISA (Language for Instruction Set Architectures), which enables rapid design space exploration for the software as well as hardware aspects of the system. Synopsys Processor Designer (PD) is used to generate the RTL description of the core, a cycle accurate instruction set simulator as well as the necessary software tools (assembler, linker) for creating program binaries from the LISA specification. Additionally, the tool chain is extended by a custom C compiler, which is based on the PD built-in CoSy compiler development system. The C compiler allows for easier benchmark development. The design flow contains a custom regression test for cycle accurate verification of the LISA model simulation against the behavioral simulation of the generated HDL code. The HDL code is integrated into the multi-core architectures written in VHDL, providing the crossbar interconnects and memory banks. The complete system architecture is then synthesized, placed, routed and optimized to have a full layout design. This design is then post-layout simulated using the memory contents extracted from the compiled benchmark program binary. The resulting trace file is finally used to perform an accurate power analysis of the complete system.

2.1 Energy efficiency of the Core

TamaRISC is more energy efficient than other state-of-the-art cores for biomedical signal processing. It consumes only 15.6 pJ/Ops at 1.0 V. For the same supply voltage level (1.0 V), yet 130 nm process Kwong et al. [15] report 47 pJ/cycle energy consumption for their 16-bit core where the number of clock cycle per instruction is higher than one. In another work, Ickes et al. [16] introduce a 32-bit core implemented in 65 nm, and the energy consumption of the core [16] is estimated for 1.0 V between 19.7 pJ/Ops and 27.0 pJ/Ops. Compared to these state-of-the-art processing cores, our optimized core consumes less energy per operations notably due to its simplified architecture as well as reduced instruction set.

2.2 Multi-Core Architectures Comparison

To execute the considered benchmark the **mc-ref** architecture requires 90.20k clock cycles whereas the **ulpmc-int** and the **ulpmc-bank** versions of the proposed architecture require 90.40k and 101.8k clock cycles, respectively. These differences are mainly due to the shared data access conflicts. The CS random vector with linear pattern is accessed conflict free, because the cores are in synchronization, and thus benefit from the data broadcasting. However, the data dependent Huffman coding LUTs cause memory conflicts since all the cores process different input data. To reduce the conflicts these LUTs are placed into the private section of the DM. In that case, the proposed architecture with **ulpmc-int** version requires almost 90.20k cycles, as the **mc-ref** architecture. However, the **ulpmc-bank** version needs 94.00k clock cycles, only 4% increase with respect to the **mc-ref** architecture. This is due to the data dependent program flow in the Huffman coding which leads the cores to lose the synchronization. Thus, the **ulpmc-bank** version suffer from the IM conflicts due to the banked instruction organization.

With only the broadcasting mechanism implemented in the I-Xbar, the IM is accessed totally 428740 times in both **ulpmc-int** and **ulpmc-bank** versions while the number of access in the **mc-ref** architecture is 90100 per core, adding up to 720800. Therefore, the number of accesses in the proposed architecture is reduced by 40% with respect to the **mc-ref** architecture. However, as a result of the DM organization together with the broadcasting mechanisms, the cores remain mostly in synchronization, and thus the number of accesses is reduced to 90220 (87% reduction with respect to the **mc-ref** architecture) for both **ulpmc-int** and **ulpmc-bank** versions. This leads to significant power savings on the IM of both versions. Additional power costs of the I-Xbar are only 0.03 mW and 0.01 mW in the **ulpmc-int** and the **ulpmc-bank** versions, respectively. Moreover, the broadcasting in the I-Xbar and D-Xbar do not lead to any significant additional power consumption in the proposed architecture. However, the cores in both versions of the proposed architecture consume more power than the ones in the **mc-ref** architecture. This is due to the signal activity increase caused by the I-

Xbars. In the **mc-ref** architecture, the cores are directly connected to the IM banks whereas in the proposed architecture, there exist the I-Xbar between them, leading to more signal activities on the instruction paths. However, as a consequence of reduced memory powers the **ulpmc-int** and the **ulpmc-bank** versions accomplish 42.7% and 45.7% active power savings, respectively compared to the **mc-ref** architecture for the same workloads. The **ulpmc-bank** version achieves higher active power saving due to less signal activity on the instruction paths than the **ulpmc-int** version. The cores and I-Xbar consume less power in the **ulpmc-bank** version than the ones in the **ulpmc-int** version, because the instructions are read only from one IM bank instead of multiple banks. This leads to less signal activities at the output nets of the I-Xbar and, thus less power consumptions.

At nominal voltage (1.2 V), the **mc-ref** architecture achieves 797.6 MOps/s while the **ulpmc-int** and **ulpmc-bank** operate up to 662.3 MOps/s and 636.9 MOps/s, respectively. When the supply voltages reach the threshold level, the **mc-ref**, **ulpmc-int** and **ulpmc-bank** architectures still accomplish around 10 MOps/s.

FIG. 7 shows power consumptions of the architectures normalized to **mc-ref** design power consumption for various workloads. During this experiment, both voltage and frequency scaling are applied for workloads higher than 10 MOps/s, however for workloads lower than this, only frequency scaling is used and the supply voltages are kept at the minimum level. As seen from this figure, the ulpmc-bank design is more energy efficient than the **ulpmc-int** and **mc-ref** designs for a given workload requirement. More specifically, at the highest workload (636.9 MOps/s) that all the designs can achieve, the **mc-ref** architecture consumes around 46.8 mW, whereas the **ulpmc-int** and the **ulpmc-bank** designs consume 31.2 mW and 30.1 mW, respectively. Thus the **ulpmc-int** achieves 33% while the **ulpmc-bank** accomplishes 36% power savings with respect to the **mc-ref** architecture. As the workload requirement becomes low, around 5 MOps/s, the **mc-ref** architecture consumes 80.8 μW whereas the **ulpmc-int** and the **ulpmc-bank** consume 48.4 μW and 43.9 μW, respectively. Therefore, the

ulpmc-bank accomplishes 45.7% power saving with respect to the **mc-ref** architecture.

Finally, FIG. 7 shows the dynamic and the leakage power consumptions of the circuits logics and memories (both instruction and data memories) for the workloads lighter than 600 kOps/s. As shown in the figures, the **mc-ref** and the **ulpmc-int** designs leak almost the same amount of power, whereas the **ulpmc-bank** design has 38.8% less leakage power consumption than the **mc-ref** design. This is due to the power gating on the unused IM banks. The leakage power consumption of the **ulpmc-bank** and **ulpmc-int** become comparable with their dynamic power consumption for a workload of 400 kOps/s and 500 kOps/s, respectively.

Even though the **ulpmc-int** design is more efficient than the **mc-ref** design in terms of dynamic power dissipation, the **ulpmc-int** architecture falters for the low workloads regarding the total power consumption. Notably, as seen in FIG. 5, the power consumption of the **ulpmc-int** becomes almost equal with the **mc-ref** around 5 kOps/s. However, the **ulpmc-bank** architecture maintains its efficiency, 38.8% power saving for the same workload whereby the architectures almost only leak.

2.3 Synchronizer Use

FIG. 8 compares the total power consumption of the base-line and improved architecture (using the synchronizer proposed in this work) for the benchmark, while exploiting voltage scaling. All logic and memory banks in both architectures are powered for these comparisons. As can be seen from the figure, the improved architecture achieves 211 MOps/s, whereas the baseline architecture accomplishes only 89 MOps/s. This is due to higher Ops / cycle performance in the improved architecture compared to the baseline. This is why the improved architecture achieves a higher benefit from voltage scaling, therefore it accomplishes a considerable better power efficiency for a given workload, as can be

seen in the figure. More specifically, for a given high workload of the benchmark (89 MOps/s), the baseline architecture operates at a supply voltage of 1.2 V, whereas the improved architecture can satisfy the workload at a voltage less than 0.9 V, thus the improved architecture achieves 64% power savings compared to the baseline architecture.

REFERENCES

[1] Hanson, S.; et al., "Exploring Variability and Performance in a Sub-200-mV Processor," IEEE Journal of Solid-State Circuits, 43, 881--891, 2008.

[2] Dogan, A.; et al., "Power/Performance Exploration of Single-core and Multi-core Processor Approaches for Biomedical Signal Processing," Proc. of PATMOS , 2011.

[3] Dreslinkski, R. G.; et al., "An Energy Efficient Parallel Architecture Using Near Threshold Operation," 16th International Conference on Parallel Architecture and Compilation Techniques, 2007.

[4] Yu P.; et al., "An Ultra-Low-Energy Multi-Standard JPEG Co-Processor in 65 nm CMOS With Sub/Near Threshold Supply Voltage," IEEE J. Solid-State Circuits, 45, 668--680, 2010.

[5] D. E. Culler et al., Parallel Computer Architecture: A Hardware/Software Approach. San Francisco, USA: Morgan Kaufmann Publishers, Inc, 1999.

[6] C. Ferri et al., "Energy-optimal synchronization primitives for single-chip multi-processors," in Proceedings of the 19th ACM Great Lakes symposium on VLSI, ser. GLSVLSI '09. New York: ACM, 2009, pp. 141–144.

[7] C. Stoif et al., "Hardware synchronization for embedded multi-core processors," in Circuits and Systems (ISCAS), 2011 IEEE International Symposium on,

may 2011, pp. 2557 –2560.

[8] F. Rincon et al., "Development and evaluation of multi-lead wavelet-based ECG delineation algorithms for embedded wireless sensor nodes," IEEE Trans. On Information Technology in Biomedicine, vol. 15, no. 6, pp. 854–863, Nov. 2011.

Claims

1. A multi-core processor for processing a plurality of input data, comprising
one or more processing cores;
one or more shared multi-banked instruction memories;
one or more shared multi-banked data memories;
one or more instruction crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more shared multi-banked instruction memories;
one or more data crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more multi-banked data memories;
selective broadcast means configured to implement a selective broadcast of instructions from the one or more instruction memories to the one or more processing cores using the one or more interconnecting instruction crossbar interconnects; and
analyzing means for determining whether a same instruction is to be performed on each of the plurality of input data, or whether for each of the plurality of input data a different instruction is to be performed;
the analyzing means and the selective broadcast means being functionally connected, whereby
when the analyzing means determines that the same instruction is to be performed on each different input data, the selective broadcast means reads the same instruction only once from a single one of the one or more instruction memories and broadcasts the instruction to each of the one or more interconnected processing cores to be performed; and
when the analyzing means determines that different instructions are to be performed implements no broadcast of instructions.

2. The multi-core processor of claim 1, further comprising
at least a memory management unit configured to enable each processing

core to access its individual working data with a single instance of a compiled application executed by all the cores, the memory management unit comprising

a processor identity number attributing means that attributes a unique processor identity number for each core;

data placing means configured to place the working data in the one or more shared multi-banked data memories by using the processor identity number;

translating means configured to translate a same decoded address from different ones of the one or more processing cores to different physical memory addresses according respectively to the processor identity number.

3. The multi-core processor of claim 1, further comprising

interleaving means configured for supporting interleaving of instructions across the banks of the one or more shared multi-banked instruction memories or of the one or more shared multi-banked data memories to minimize instruction memory conflicts in the case of synchronisation lost between the processing cores.

4. The multi-core processor of claim 1, further comprising

mapping means configured for supporting mapping of instructions to the minimum number of instruction memory banks of the one or more shared multi-banked instructions memories;

power gating means configured to identify memory banks from the one or more shared multi-banked instruction memories or the one or more shared multi-banked data memories which are not accessed, and enable gating of the memory banks that are not accessed.

5. The multi-core processor of claims 3, further comprising

mapping means configured for supporting mapping of instructions to the minimum number of instruction memory banks of the one or more

shared multi-banked instructions memories;

power gating means configured to identify memory banks from the one or more shared multi-banked instruction memories or the one or more shared multi-banked data memories which are not accessed, and enable gating of the memory banks that are not accessed, and

an instruction addressing schemes means configured to analyze the significant bits of addresses to map onto the one or more shared multi-banked instruction memories or of the one or more shared multi-banked data memories and to in case it is found that the least significant bits of the address to map are chosen, then the interleaving means is to be used, and in the case it is found that the most significant bits are chosen then the mapping means and the power gating means are to be used.

6. The multi-core processor in claim 1, further comprising

setting means for setting one or more of the processing cores in a status of waiting processing cores;

memory access conflict detecting means configured to detect memory access conflicts for memory requests, whereby in case a memory access conflicts is detected, the memory requests are served alternately, while the waiting processing cores are stalled using clock gating to avoid active power consumption.

7. The multi-core processor in claim 1, where the instruction crossbars and data crossbars are a Mesh-of-Trees (MoT) interconnection network to support high-performance communication between processors and memories.

8. The multi-core processor of claim 1, where each of the one or more processing core comprises

a 3-stage pipeline: fetch, decode and execute stages, and

3 external memory ports: one for instruction read, one for data read, and

one for data write, all accessible in the same cycle, with each instruction being executed in one cycle, by having complete data bypassing inside the processing core for registers as well as memory write-back data.

9. The multi-core processor of claim 1, wherein the one or more shared multi-banked data memories is split into at least a shared section and a private section;

each processing core's private data is stored in the private section, while data shared across the different processing cores is stored in the shared section;

with the decoded address from an instruction directly used as physical memory address for the shared section and an address translation mechanism used to obtain the physical memory address for the private section.

10. The multi-core processor of claim 9, where in the private sections of each processor core are located into different memory banks, thus the private section is accessed without conflicts.

11. A method for processing a plurality of input data with a multi-core processor, wherein the multi-core processor comprises

one or more processing cores;

one or more shared multi-banked instruction memories;

one or more shared multi-banked data memories;

one or more instruction crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more shared multi-banked instruction memories;

one or more data crossbar interconnects that respectively connect a corresponding of the one or more processing cores with a corresponding of the one or more multi-banked data memories;

the method comprising the steps of

selectively broadcasting instructions from the one or more instruction memories to the one or more processing cores using the one or more interconnecting instruction crossbar interconnects; and

analyzing whether a same instruction is to be performed on each of the plurality of input data, or whether for each of the plurality of input data a different instruction is to be performed;

whereby when the analyzing determines that the same instruction is to be performed on each different input data, the step of selectively broadcasting comprises reading the same instruction only once from a single one of the one or more instruction memories and broadcasting the instruction to each of the one or more interconnected processing cores to be performed; and when the analyzing determines that different instructions are to be performed the step of selectively broadcasting implements no broadcast of instructions.

12. The method of claim 11, further comprising steps of

putting to sleep processing cores that finish operations and wait for the completion of the execution of one or more cores due to data-dependencies such as to save power consumption, and

waking up sleeping processing cores when the corresponding cores complete their operations.

13. The method of claim 11, wherein the following synchronization method is added to support synchronization of the different one or more processing cores:

a. determining for a given application, data-dependent code sections, marking the beginning of the data-dependent code sections as check-in points, and marking the corresponding ending of the data-dependent code sections as check-out points;

b. assigning for each data-dependent code section check-in and check-out pair, a data memory position to trace the synchronization process, wherein, both the processing core identities and total number of

cores (the core counter), currently running the corresponding data-dependent program section are stored;

c. once a processing core arrives to a check-in point, modifying by the processing core of the corresponding data memory position by adding a signature of the processing core as well as incrementing the core counter;

d. once a processing core arrives to the corresponding checkout point for the check-in point identified in the preceding step, decrementing the core counter at the data memory, and waiting at the processing core for the other expected cores to reach to the same checkout point, whereby the waiting processing cores go to sleep immediately after decrementing the core counter; and

e. once all the expected cores running the same data-dependent program section reach to the check-out point (the core counter becomes zero) then the processing cores that are executing the waiting step are woken up to continue processing.

14. The method of claim 13, further comprising steps of

implementing an instruction set architecture comprising of one or more arithmetic logic unit instructions, whereby

the said arithmetic logic unit instructions work on 3 operands, using the exact same addressing mode options for each instruction, thereby decoupling operand fetch logic and the arithmetic operation; and

the instruction word encoding is designed to be regular with fixed bit positions allowing for very efficient decoding of the operands and the different instruction words.

15. The method of claim 13, wherein the synchronization method further comprises

appending the core ids and incrementing the core counter according to a number of parallel requests, when multiple processing cores reach a check-in point at the same time;

for multiple check-out requests, decrementing the core counter accordingly;

writing the calculated value for check-in/check-out requests to the data memory through one of the data write ports used by the processing cores, whereby the data write port is assigned for the core which has the minimum identity number among the other synchronous cores.

16. The method of claim 13, wherein

each processing core locks the assigned memory position for a check-in/check-out process until the content has been modified to avoid possible hazards among the cores, by generating an external signal (lock output signal), indicating a lock request;

the signal locks the memory position until it has been modified with the new value for a safe message passing among the cores in sequential check-in/check-out processes.

17. The method of claim 11, further comprising the steps for implementing an address translation mechanism, wherein any address targeting the private section of the data memory is shifted by $\log_2 N_{AC}$ and the processing core identity is appended, where N_{AC} stands for the number of active cores; then the address is shifted by $\log_2(N_{TMB}/N_{AMB})$, where N_{TMB} and N_{AMB} are the total number of data memory banks and the number of active data memory banks.

18. The multi-core processor of claim 1, wherein each processing core performs the same operation, such as signal filtering, on different channels of the same biological signal, such as ECG or EEG, with each processing core sharing the same interactions that are broadcasted to them from the instruction memory, and different data is obtained from the data memory.

19. The multi-core processor of claim 1, wherein each core operates at near-threshold voltage, achieving ultra-low power operation.

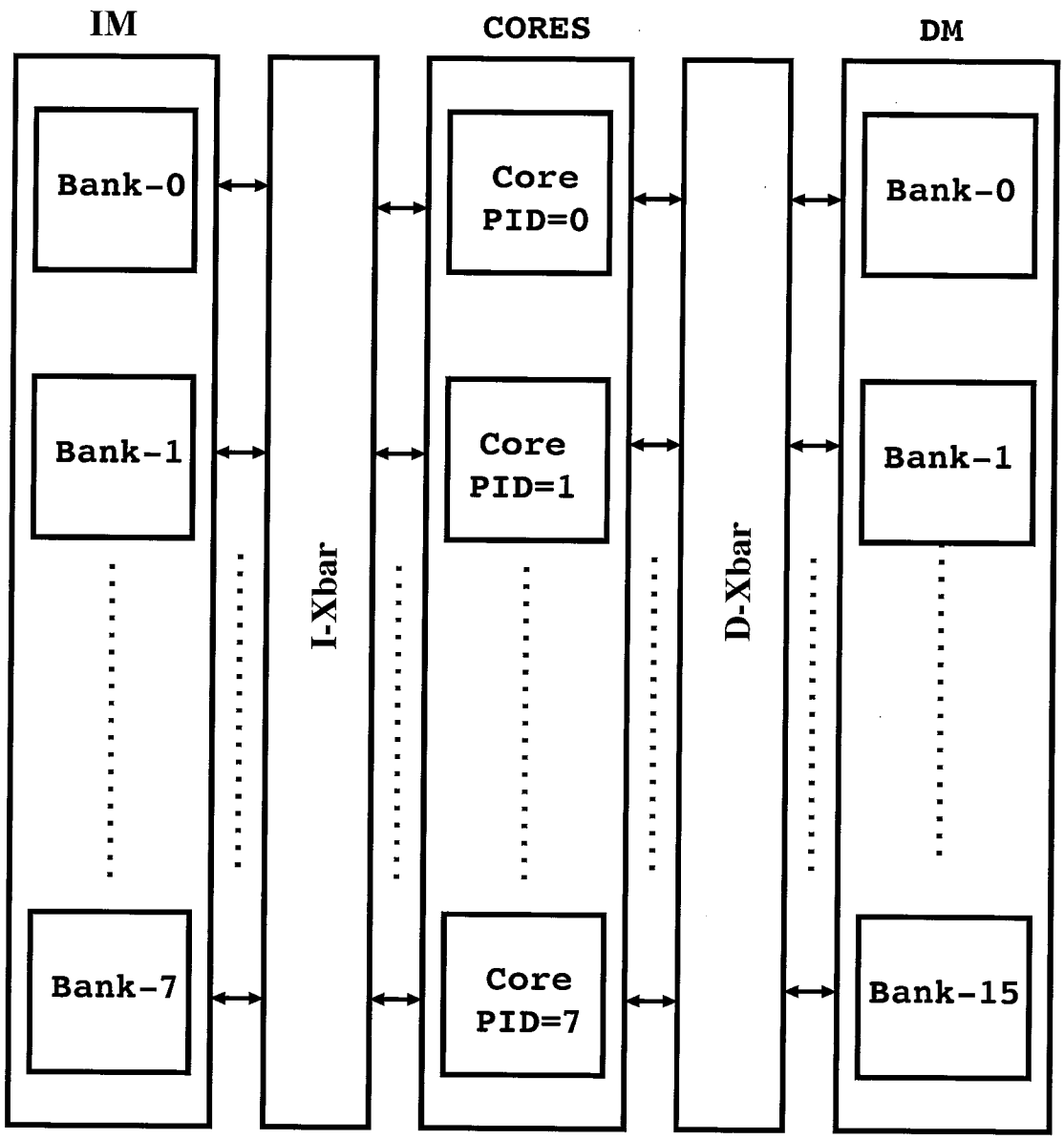
IM
CORES
DM
Bank-0
Bank-1
Bank-7
I-Xbar
Core PID=0
Core PID=1
Core PID=7
D-Xbar
Bank-0
Bank-1
Bank-15

FIG. 1

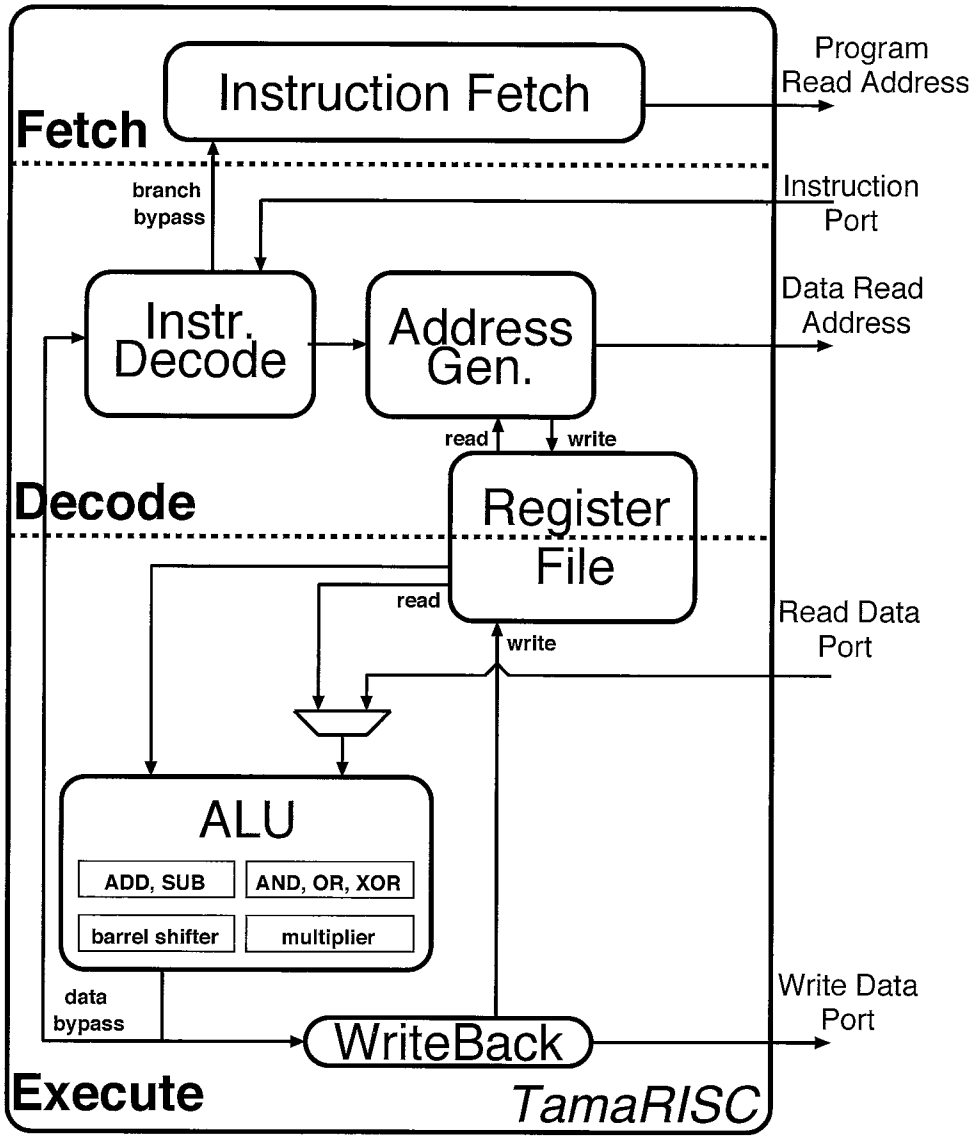
Fetch
Instruction Fetch
Program Read Address
branch bypass
Instruction Port
Instr. Decode
Address Gen.
Data Read Address
read
write
Decode
Register File
read
write
Read Data Port
ALU
ADD, SUB
AND, OR, XOR
barrel shifter
multiplier
data bypass
WriteBack
Write Data Port
Execute
TamaRISC

FIG 2.

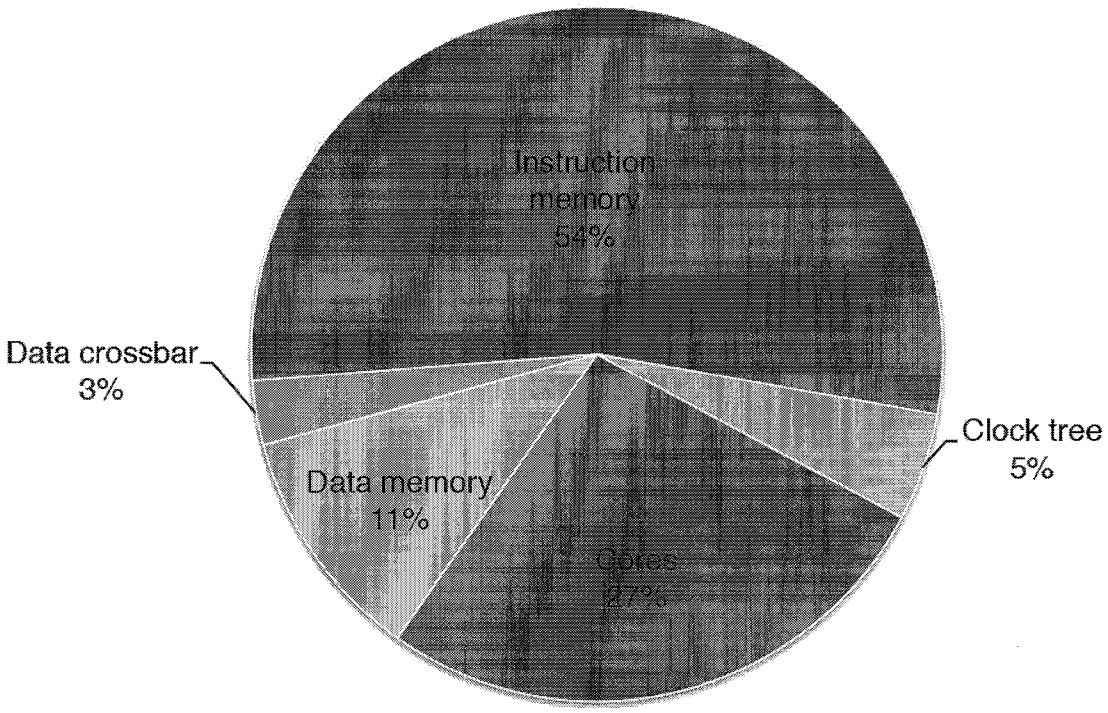
Instruction memory 54%
Data crossbar 3%
Clock tree 5%
Data memory 11%
Gates 27%

FIG 3.

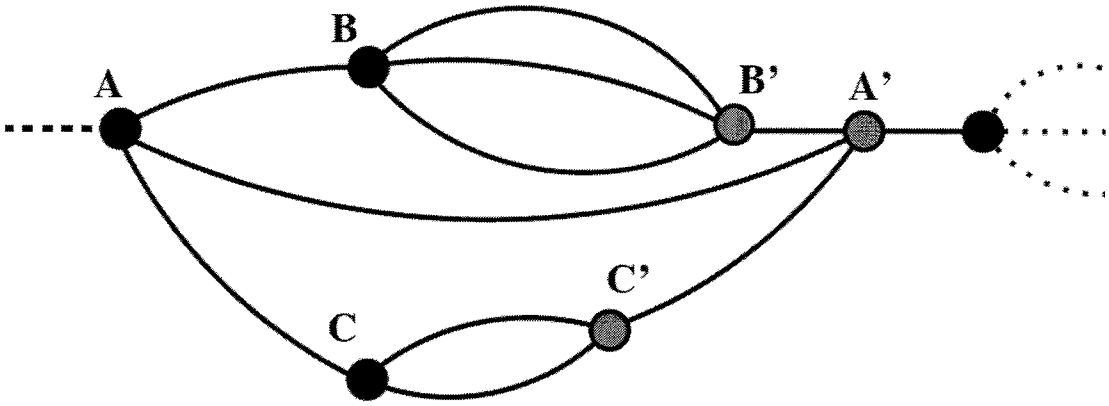

FIG. 4

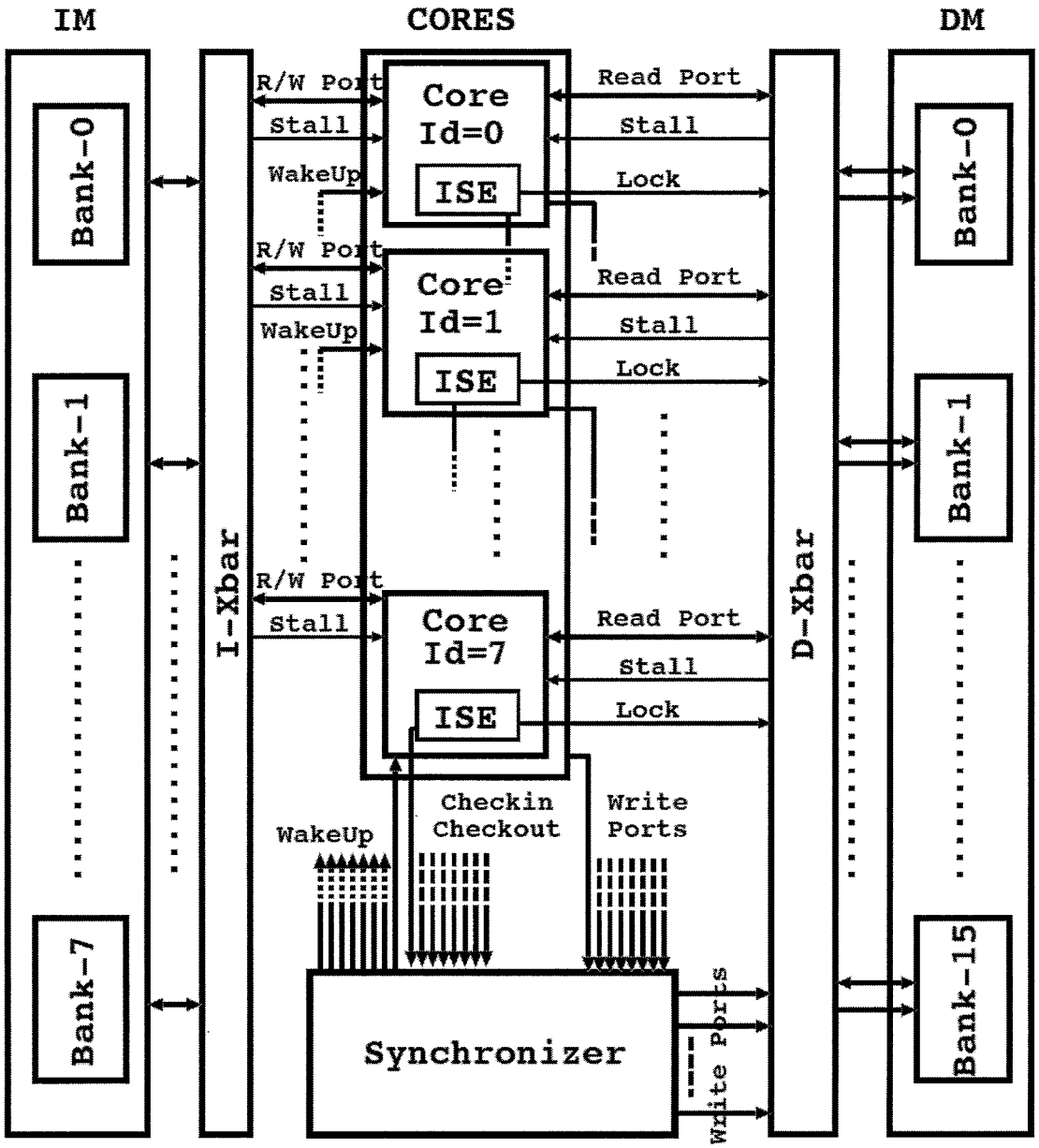
IM
CORES
DM
Bank-0
Bank-1
Bank-7
I-Xbar
R/W Port
Stall
WakeUp
Core Id=0
ISE
Core Id=1
ISE
Core Id=7
ISE
Read Port
Stall
Lock
D-Xbar
Bank-0
Bank-1
Bank-15
Checkin
Checkout
Write Ports
WakeUp
Synchronizer
Write Ports

FIG 5.

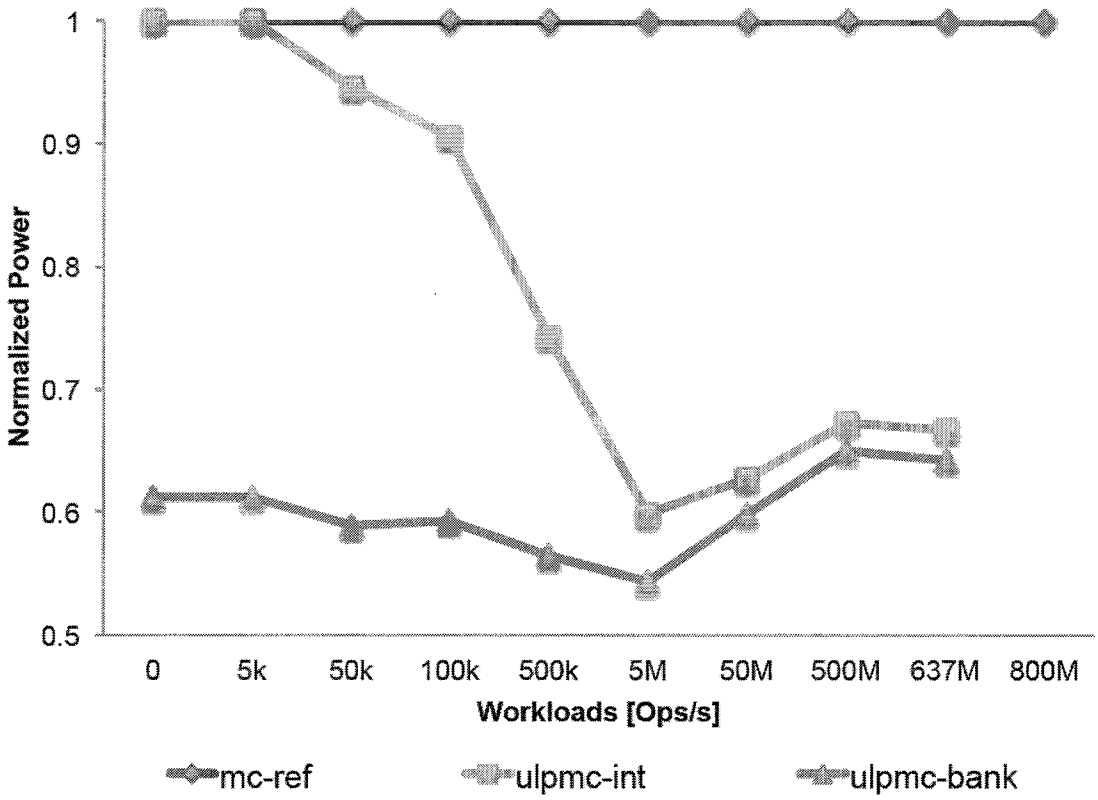
1
0.9
0.8
0.7
0.6
0.5
Normalized Power
0
5k
50k
100k
500k
5M
50M
500M
637M
800M
Workloads [Ops/s]
mc-ref
ulpmc-int
ulpmc-bank

FIG. 6

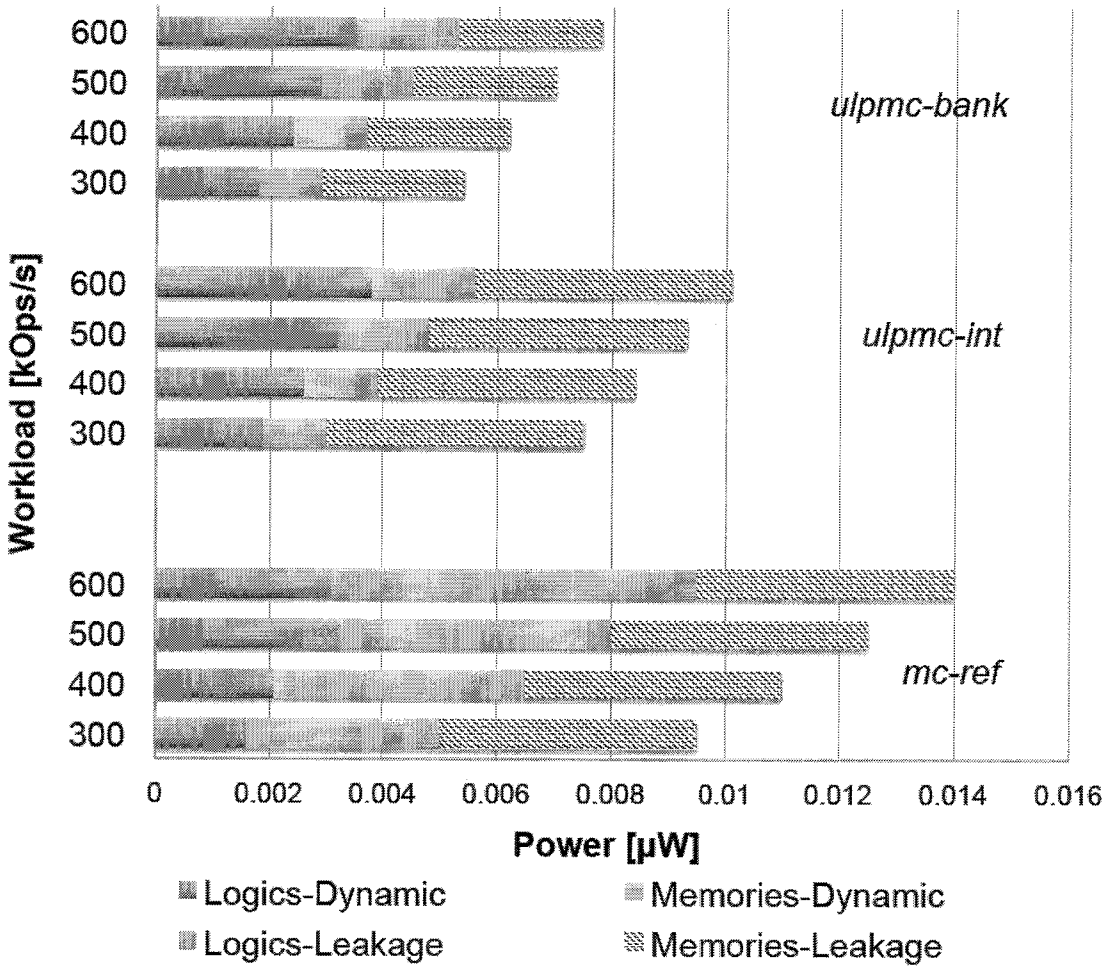
600
500
400
300
ulpmc-bank
600
500
400
300
ulpmc-int
600
500
400
300
mc-ref
Workload [kOps/s]
0
0.002
0.004
0.006
0.008
0.01
0.012
0.014
0.016
Power [µW]
Logics-Dynamic
Memories-Dynamic
Logics-Leakage
Memories-Leakage

FIG. 7

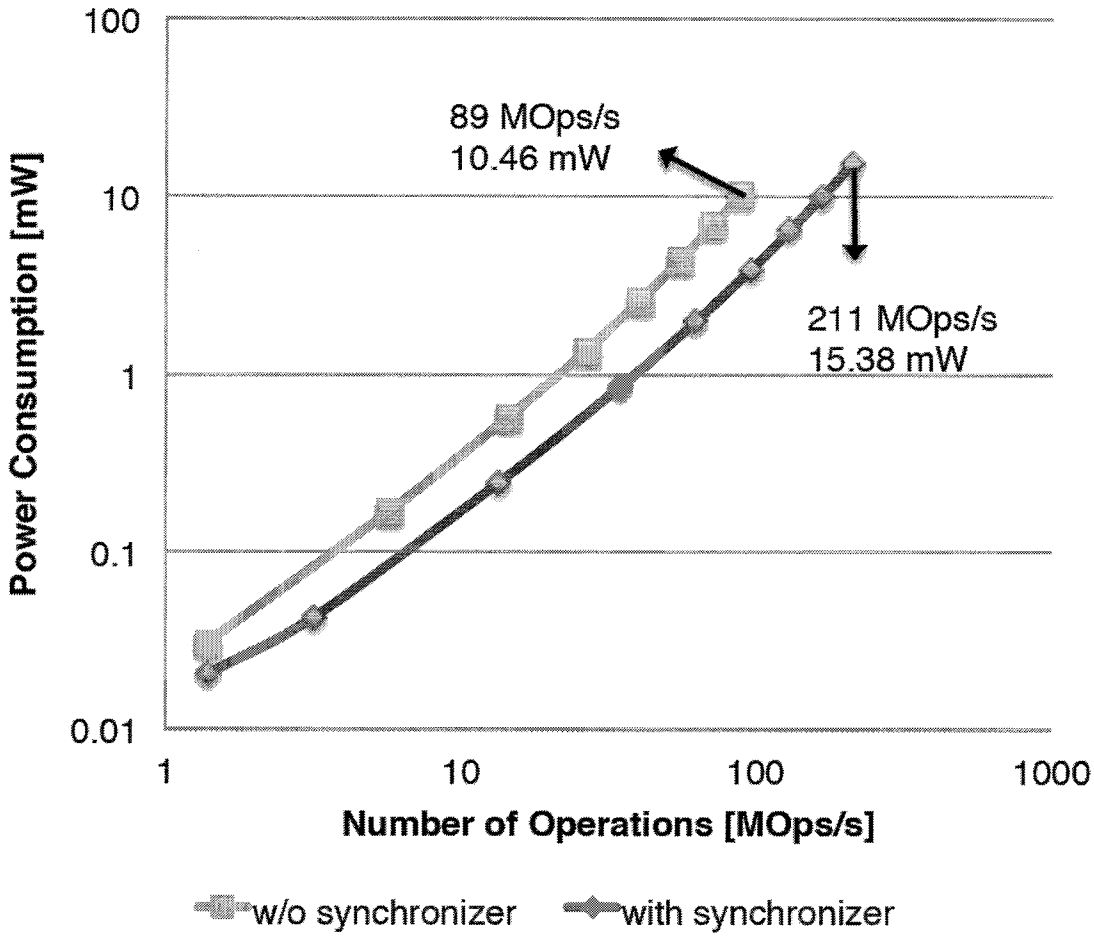
100
10
1
0.1
0.01
Power Consumption [mW]
89 MOps/s
10.46 mW
211 MOps/s
15.38 mW
1
10
100
1000
Number of Operations [MOps/s]
w/o synchronizer
with synchronizer

FIG. 8