Synergistic Architecture and Programming Model Support for Approximate Micropower Computing

Giuseppe Tagliavini
and Davide Rossi
DEI, University of Bologna, Italy
Email: {giuseppe.tagliavini, davide.rossi}@unibo.it

Andrea Marongiu
and Luca Benini
DEI, University of Bologna, Italy
ETH Zurich, Switzerland
Email: {a.marongiu, luca.benini}@iis.ee.ethz.ch

***Abstract*—Energy consumption is a major constraining factor for embedded multi-core systems. Using aggressive voltage scaling can reduce power consumption, but memory operations become unreliable. Several embedded applications exhibit inherent tolerance to computation approximation, for which indicating parts that can tolerate errors has proven a viable way to reduce energy consumption. In this work we propose an extension to OpenMP to specify what regions of code and data are tolerant to approximation. A compiler pass places data into memory regions with different reliability guarantees according to their tolerance to errors. The voltage supply level is dynamically adjusted according to tolerance policies, with the overall goal of minimizing energy in full compliance with precision constraints.**

***Keywords*—OpenMP, approximate computing, micropower computing, many-core accelerators.**

I. Introduction

Traditionally, ultra-low power (ULP) systems are mostly based on single-core microcontrollers, where low-power operation correlates with low computational requirements. This assumption is less and less true for today's deeply embedded sensing systems [12], which execute computation-demanding tasks that are inherently parallel. To match conflicting requests for energy consumption and performance, several (ultra) low-power multi-core systems have been recently proposed [17] [14] [23] in application fields such as industrial automation, mission critical systems, pervasive video infrastructures and bio-medical computing.

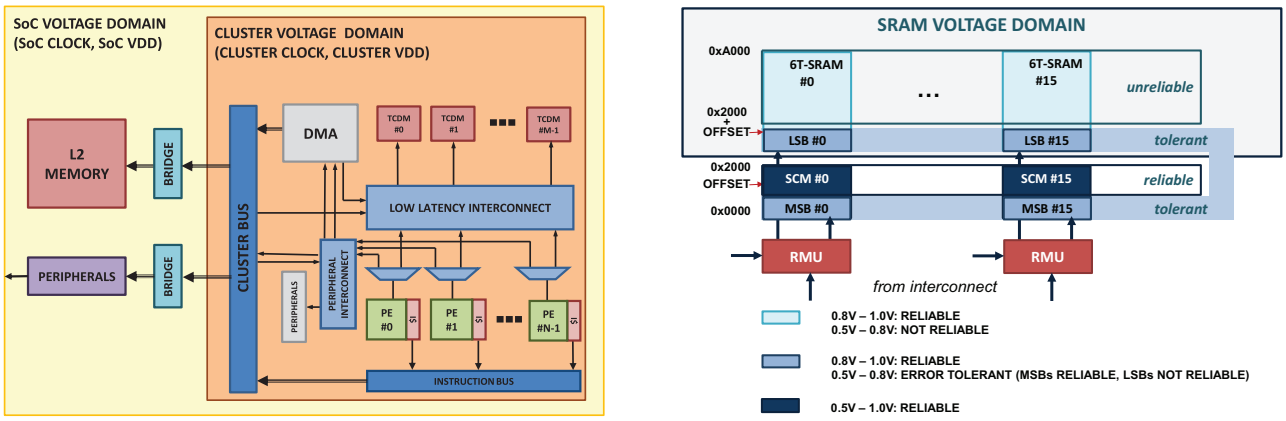
For such systems, the memory system typically consumes more than 50% of the total chip power, and are thus the key component to be optimized. Using aggressive voltage scaling can reduce energy, but memory operations become unreliable due to the lack of sufficient static noise margin [7]. Approximate computing has recently emerged as a promising approach to design energy-efficient digital systems working at unreliable voltage levels. The notion of approximate computing [13] refers to a set of techniques ranging from programming language- to transistor-level, with a common aim to allow computing systems to save energy to the detriment of the *quality* of the computed results. Approximate computing is a promising approach in the ULP domain when applications exhibit inherent tolerance to errors, i.e. the property to produce *acceptable* outputs despite some computations execute in an inaccurate manner within controlled thresholds [22], [19] [16]. The error tolerance may derive from several factors [8]:

- some algorithms process large amounts of redundant input data, that in turn may contain inherent noise;
- errors can easily get averaged down or averaged out in statistical and aggregating algorithms;
- some algorithms use iterative refinement techniques, and subsequent iterations may correct errors introduced in previous iterations;
- some algorithms do not have a single "golden" result, they may return an element of a set of multiple solutions that are equally acceptable;
- less-than-perfect results are perceived as correct by the users.

In this work we propose a set of HW/SW approximation techniques for PULP, a ultra-low-power multi-core platform implemented in 28 nm UTBB FD-SOI technology [23]. PULP is a scalable, clustered many-core computing platform that features a parametric number of Processing Elements (PE) per cluster, sharing a multi-banked tightly coupled L1 data memory (TCDM, a scratchpad).

At the HW level, our proposal focuses on energy saving techniques for the TCDM, while the PEs work at the most energy efficient operating point (0.5V @ 20MHz). We consider two different memory technologies, Standard Cell Memory (SCM) and six-transistor Static RAM (6T-SRAM). Although 6T-SRAM provides a much better storage density than SCM, its minimum operating voltage is much higher (0.8V as opposed to 0.5V). Accessing 6T-SRAM below 0.8V results in a flip-bit error with a certain probability. In our solution, the TCDM is physically partitioned into two different regions, implemented with the two technologies. With this layout, the operating voltage of 6T-SRAM can be safely lowered whenever data located in this region is not accessed. In addition, to leverage approximation tolerance in applications in a controlled manner we provide a transparent mechanism to split multi-byte data into multiple banks at the architecture level. We call the associated address range *tolerant memory*. The most significant bits (MSB) of a word are stored in the reliable, low-voltage SCM, while the least significant bits (LSB) are stored in the 6T-SRAM. This allows to access 6T-SRAM cells at low voltage for error-tolerant computations, bounding the error to the LSB. Our results demonstrate that this approach provides much better precision than just dropping the LSB.

At the SW (programming model) level, we provide constructs for specifying regions of code and data that are tolerant to approximation in a C program. A compiler optimization pass (based on Clang+LLVM [1] [18]) places data into different memory types according to its tolerance to errors, activating the tolerant memory region and inserting voltage switch points for 6T-SRAM when this does not violate precision constraints. A greedy allocation algorithm uses a simple heuristic to minimize the power consumption due to data accesses when other policies are not possible.



(a) The baseline PULP architecture

(b) Hybrid TCDM organization

Fig. 1: The PULP architecture and the extended L1 memory system

Using cycle accurate simulation models of the target platform annotated with power numbers extracted from a silicon implementation, we demonstrate that our hybrid memory architecture can reduce by 25% on average the energy consumption, very close to the "ideal" savings achievable with SCM cells only.

II. Hardware Architecture

ULP systems are largely based on microcontrollers featuring simple, cache-less cores (e.g., Cortex M0 or M4), coupled to simple support for power management and a standard set of peripherals. The Parallel processing Ultra-Low-Power platform (PULP) [23] aims at providing a significant boost to the peak performance that ULP systems can achieve by coupling the multi-core paradigm to the most advanced FDSOI design technology and associated techniques for energy efficiency (near-threshold computing, body biasing, etc.). We describe here the PULP platform (Sec. II-A) and the extensions to the memory system for computation approximation (Sec. II-B).

A. The PULP Architecture

PULP is a scalable, many-core computing fabric, organized as a set of *clusters*. Figure 1a shows the main building blocks of single-cluster PULP SoC. Multiple clusters can be interconnected at the top level to share L2 memory and peripherals for off-chip communication. Within a cluster, a parametric number of Processing Elements (PEs) based on an optimized OpenRISC microarchitecture [2] share a L1 multi-banked tightly coupled data memory (TCDM). The TCDM is configured as a shared scratchpad memory, featuring as many R/W ports as the number of memory banks. This allows concurrent access to memory locations mapped on different banks, via a one-cycle-latency logarithmic interconnect implementing word-level interleaving to reduce contention. In addition, each PE is equipped with a private instruction cache. A lightweight, multi-channel DMA enables fast and flexible communication with other clusters, the L2 memory and external peripherals [24] (the latter leverages a dedicated peripheral interconnect).

The specific platform instance considered in this work consists of 8 PEs and 16 TCDM banks of 2.5KB each (40KB total), plus 64KB of L2 memory. Each core features 1 Kbyte of I$. Two voltage domains are considered: i) the SoC domain includes the L2 memory and peripherals; ii) the cluster domain includes the PEs the TCDM, the DMA and the cluster interconnect. A 28nm STMicroelectronics UTBB FD-SOI implementation of the platform in this configuration can operate at 20MHz @ 0.5V, and at 100MHz @ 0.6V.

For improved energy efficiency, we extend this baseline platform with a hybrid L1 memory system that supports computation approximation in strict cooperation with the programming model.

B. L1 Memory System Extensions

On-chip memory is traditionally implemented with 6T-SRAM in super-threshold designs. When operating close to the threshold voltage, standard cell memory (SCM) has demonstrated a better tradeoff between reliability, energy efficiency, area and portability among technology nodes [26]. In particular, although 6T-SRAM cells provide a much better storage density than SCM ($\sim$ 3x), their minimum operating voltage is much higher (0.8V-1.0V) [9], while SCMs are reliable over all the operating voltage range of the architecture (0.5V – 1.0V). Accessing 6T-SRAM at a low voltage may results in a flip-bit error, as shown in Table I (values estimated for our chip).

Voltage [V]	0.50	0.55	0.6	0.65	0.7	0.75
P(bit-flip)	0.0037	0.0012	0.0003	5.24e-5	4.35e-6	6.16e-8

TABLE I: Probability of bit-flip errors in 6T-SRAM.

Both memory types have similar retention voltages (0.25V and 0.29V), above which data persistence is guaranteed. SCM can thus operate at the same low voltage of the logic in a reliable way, with the key benefit of providing much smaller energy/access ($\sim$ 4x) [21]. Based on these observations and on the evidence that we cannot afford to build the entire TCDM with SCM, we propose a hybrid L1 memory design.

We organize the TCDM in two different regions: 16 128x32 SCM cuts plus 16 standard 512x32 6T SRAM cuts. Reliability Management Units (RMU) are introduced in the path between the low-latency interconnect and the TCDM. RMUs are simple combinational logic blocks that allow to remap the address space of the TCDM into three distinct logical memory regions:

- *reliable* – mapped in the SCM, reliable at any operating point.

```
int main()
{
  #pragma omp tolerant var
  int sparse_A[];
  int i = 0, index;

  while ( func(i) )
  {
    …
    index = compute_index();
    #pragma omp tolerant computation
    sparse_A[index] = compute_element();
    …
    update(i);
  }
}
```

Fig. 2: A sparse matrix computation with tolerant directives.

- *unreliable* – mapped into the 6T-SRAM, reliable above 0.8V.
- *tolerant* – mapped in the SCM (most significant bits - MSB) and in the 6T-SRAM (least significant bits - LSB), errors can show up on the LSB below 0.8V.

The *tolerant* region guarantees a correctness threshold for data when executing approximate code regions.

Figure 1b shows the new TCDM design and the logical regions. A third voltage domain is considered for the *unreliable* region (6T-SRAM cuts). Level shifters are introduced at the boundaries between voltage domains, i.e., when the SRAM is operating at 0.8V and the logic at a lower voltage[1]. The RMUs provide access to the *tolerant* region only at word or half-word level. In both cases, MSB (upper 16 or 8 bits) are placed into SCM cells, while LSB (lower 16 or 8 bits) are placed into 6T-SRAM cells.

The offset that defines the boundaries between the three logical regions can be configured by writing into a memory-mapped peripheral, accessible by every PE. Thus, the memory map can be re-configured on-the-fly by the software, enabling application-specific optimizations.

III. Programming Model and Compiler

In the target platform the program code is stored in L2 and accessed via private I$. The latter is implemented with SCM cuts, and thus is always reliable. The focus of our techniques in thus only on data, which resides in the TCDM[2]. To conveniently control energy-efficient data mapping, we propose an extension to an OpenMP implementation for embedded multicore systems [20], in the form of two new directives:

- `#pragma tolerant var` – coupled to a variable declaration to specify it tolerates approximation.
- `#pragma tolerant computation` – coupled to a program statement to specify the memory addressed therein can be accessed in tolerant (low-voltage) mode.

Figure 2 shows an example of use of these directives. In sparse matrix computation, matrix indexes cannot be approximated (a single error implies wrong element choice), while matrix elements may tolerate approximation. Thus, we declare the `sparse_A` array and the element computation as tolerant, while `index` and its computation are not tolerant. Declaring pointers with the `tolerant var` directives is not supported,

[1]The power overhead of the level shifters is $< 1\%$ of a memory bank.

[2]The DMA can be used to move data between the L2 and the TCDM

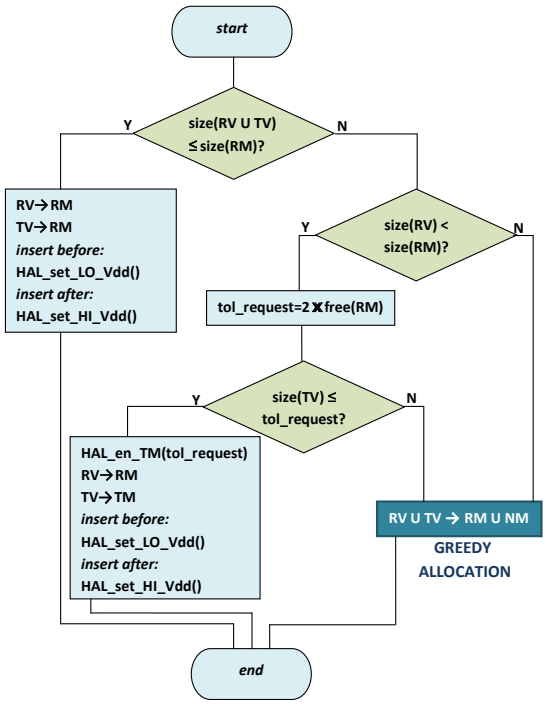


Fig. 3: Flowchart of the reliable allocation algorithm.

as an error on pointer dereferencing would lead to a random memory area access.

Our compiler (based on *Clang + LLVM*) transforms the directives into annotated tokens in the intermediate representation (IR). These annotations are used by an allocation optimization pass which operates as follows. Ideally, if all the data could be allocated in the reliable SCM we could keep the voltage low for the entire program life. However this is typically not possible, as the SCM is very small, so we need to identify what can go in SRAM and in which conditions it is safe to lower the voltage. The focus of the optimization is on the `tolerant computation` regions, since out of these regions the voltage must be kept high anyhow (again, unless everything fits in the SCM). Within `tolerant computation` regions, under low voltage operating mode:

1) tolerant variables can safely have their LSB in SRAM (we call these variables *TV*)
2) non-tolerant variables can only go in SCM (we call these variables *RV*)

The remaining variables are those whose live ranges do not overlap with the `tolerant computation` regions, so it is safe to lower the voltage within such regions even if they reside in SRAM (we call these variables *SV*).

First, standard *use-defs* analysis is applied to each node of the call graph, to extract usage estimates of global and local variables[3]. For each variable the following attributes are determined:

- *tolerant* – a flag that specifies whether the variable is tagged as tolerant or not;
- *scope* – a flag that specifies whether the variable is local or global;

[3]Static use-defs analysis limits the applicability of the approach to programs written as a single translation unit. Inter-Procedural Analysis and Link-Time Optimization approaches can be used to avoid this limitation.

Fig. 4: Example of greedy allocation.

- *class* – the type of the variable (scalar/array/structure);
- *use intervals* – live range within the current scope;
- *tolerant use interval* – live range within tolerant code regions;
- *alloca* – for local variables, a reference to the declaration statement;
- *usedefs* – use-defs chains for the variable within the current scope.

Live ranges are computed with respect to call-graph nodes in case of global variables, and to basic blocks in case of local variables.

The flowchart in Figure 3 describes our **reliable allocation algorithm**. *RM* is the reliable memory (SCM), *NM* is the unreliable memory (6T-SRAM), and finally *TM* is the tolerant memory that can be optionally activated. For each `tolerant computation` region, the algorithm computes the memory size of RV∪TV. If this value is less than or equal to the size of the reliable memory region, (i) all data are allocated in RM, (ii) TM is not activated, and (iii) the voltage of NM is switched down without introducing any level of approximation. If the previous check fails, it considers the memory size of RV. If this value is less than the size of RM, it verifies whether the memory size of TV is less than or equal to the maximum size that we can set for TM to avoid segment overlaps (i.e., the amount of free reliable memory multiplied by two). If this last check is successful, the compiler forces the allocation of RV variables in RM, and TM is properly set to allocate TV variables. In this case, (i) the voltage of NM can be switched down just before executing the tolerant computation, and switched on to previous value after finishing, using proper API calls provided by the hardware abstraction layer (HAL); at the same time, (ii) TM is activated by another HAL call, and its offset is then propagated to the linker.

In the fall-through case of reliable allocation algorithm, when RV does not fit RM or TV does not fit TM, we execute a **greedy allocation algorithm**. In this case we do not take into account the tolerant flag, first of all we allocate the stack in RM and then we start allocating all the other structures, again prioritizing RM allocation, because accesses to this memory area imply a minor power consumption. Figure 4 shows an example of a typical allocation obtained by this algorithm. In any case, the core stacks are always allocated in the reliable memory, and also local variables that are tagged as tolerant are allocated onto the stack when their size is less than a limit value (the default value in current implementation is 16 bytes). This reduces the payload of the allocation algorithm with a minimum impact on the final memory layout. At link time, tolerant data are mapped at the proper memory address using a specific ELF segment. This implies that local variables allocated in tolerant memory are no longer allocated on the stack, and the stack reference in the function code is replaced with a unique global instance.

IV. Experimental Evaluation

A. Simulation Setup and Methodology

The proposed architecture has been modeled in Virtual SoC [6], a SystemC-based cycle-accurate virtual platform for the acceleration of massively parallel heterogeneous System-On-Chips. The energy consumptions of all the key components of the platform (cores, I$, SRAM banks, SCM banks, interconnect) are extracted through back-annotated simulations performed on the post place&route netlist, and fed into the power models of the VSOC SystemC simulator (see Table III). These numbers are derived by the first silicon implementation of PULP, realized using STMicroelectronics 28nm UTBB FD-SOI technology. This approach couples the advantages of very accurate power models with the simulation speed of the SystemC model, that allows to perform a wide exploration utilizing real-life benchmarks. In average, the virtual platform shows a maximum error in timing accuracy below 6% with respect to a complete RTL simulation of the same benchmark.

	SCM@0.5V	6T@0.5V	6T@0.8V
Read	0.609	2.766	5.347
Write	0.813	2.719	5.316
Leakage	0.002	0.009	0.018

TABLE III: Comparison of dynamic and leakage energy for SCM and 6T-SRAM. All values are reported in pJ.

B. Benchmark Suite

To assess the results of our solution, we consider the benchmarks of Table II, which have been implemented in C using OpenMP directives to split the workload over the 8 available cores. Considering the intrinsic data parallel nature of the selected benchmarks, in all cases we use a `omp parallel for` directive with a static chunking pattern. The algorithm execution is always marked as tolerant, while the tagging of data structures is detailed for each benchmark. The first two columns of table IV report a detailed analysis of the memory footprint of the benchmarks and the related number of read/write operations on the TCDM. In the experiments we take into account four reference platforms:

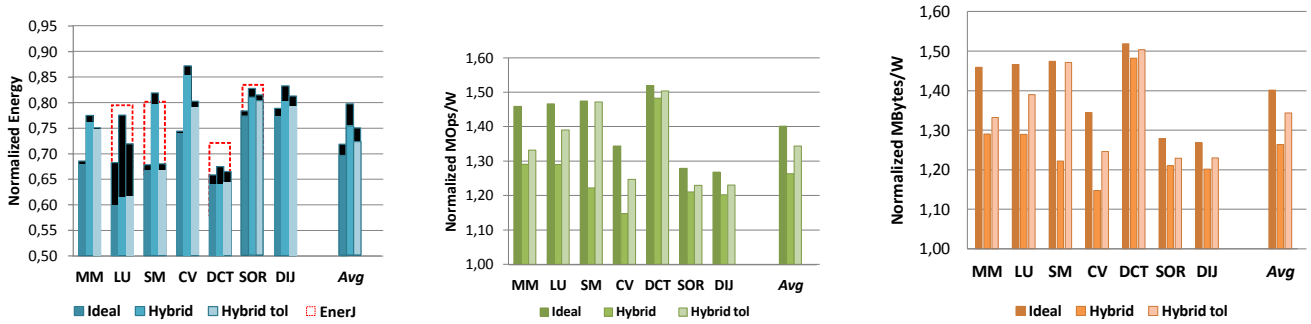
- *Base* – A PULP architecture with uniform memory cells (6T-RAM)
- *Ideal* – A PULP architecture with uniform memory cells (SCM)
- *Hybrid* – Our architecture with hybrid memory, with support to greedy allocation.
- *Hybrid tolerant* – Like the previous one, with support for voltage switching and tolerant region activation.

	Memory (KB)	R/W (words)	Zero-ing error	Flip-bit error
MM	RM: 4.0 (50%) TM: 8.0 (33%)	67728	8.37e-3	5.50e-7
LU	RM: 2.9 (36%) TM: 9.8 (41%)	428	2.39e-1	2.26e-5
SM	RM: 2.0 (25%) TM: 7.1 (30%)	2752	2.61e-2	1.15e-4
CV	RM: 4.0 (50%) TM: 4.0 (16%)	43088	5.35e-6	4.54e-10
DCT	RM: 1.0 (13%) TM: 8.0 (33%)	65536	3.55	1.29e-2
SOR	RM: 1.5 (19%) TM: 12.5 (52%)	99404	6.83e-3	3.65e-5
DIJ	RM: 1.0 (13%) TM: 14.0 (58%)	24207	8.92e-5	4.85e-11

TABLE IV: Benchmark results: memory footprint, number of TCDM accesses, MSE with zero-ing, MSE for flip-bit

	Description	Annotations
MM	Matrix multiplication	Matrix operands are tolerant
LU	LU decomposition	Matrix is tolerant, row pointers/pivots are reliable
SM	Sparse matrix multiplication with a vector	Matrix is tolerant, indexes/vectors are reliable
CV	Convolution of a matrix with a 5x5 kernel	Matrix operands are tolerant
DCT	DCT transform	Coefficients are tolerant, intermediate/output values are reliable
SOR	Successive over-relaxation	Matrix is tolerant, row pointers/pivots are reliable
DIJ	Minimum path using Dijkstra algorithm	Connectivity data are reliable, weights are tolerant

TABLE II: Benchmarks.



(a) Normalized energy consumption. (b) Normalized operation efficiency. (c) Normalized data transfer efficiency.

Fig. 5: Experimental results

C. Accuracy

Approximate algorithms can "absorb" errors which occur with a probability lower than a given threshold throughout the entire computation. To assess the accuracy of our approach, we compared the use of unreliable LSB with a more drastic alternative, that is not computing them at all. To make the two solutions fully comparable, we used the same algorithms and the same data type, but in the second case we forced the LSBs to zero.

The last two columns in Table IV reports mean squared errors (MSE) for both approaches (referred to an implementation where data is kept on a reliable memory area). For all applications our approach achieves at least two orders of magnitude smaller MSE.

D. Energy Consumption

Figure 5a shows the energy consumption of the benchmarks on each platform. All values are normalized to the result of the *Base* architecture, and the rightmost set of bars shows the average. Bars are broken down in leakage (upper part) and dynamic energy (lower part).

The *Ideal* platform allows on average a 28% energy reduction w.r.t. the *Base* platform. This platform is not a real alternative due to the area cost of SCMs, but the related value is an upper bound to the energy reduction that we can obtain with any other method, whereas the *Base* platform is a lower bound.

The energy reduction of the *Hybrid* architecture is around 20% in the average case, to which the *Hybrid tolerant* platform sums an additional 5% saving. Overall our approach gets very close to the *Ideal*. Figure 5a also reports the energy consumption estimated by *EnerJ* [25] for a subset of common benchmarks (LU, SM, DCT, SOR)[4]. EnerJ consists mainly of a software approach that considers different methods of approximation (DRAM, SRAM, ALU, FPU) gleaned from the literature, while in our platform we do not have a DRAM and we do not support any form of approximation of hardware computation units. However, since SRAM is the main contributor to energy consumption for ULP systems, in general our solution performs better in terms of energy efficiency.

Figure 5b shows the computation efficiency, that is the number of operations executed per power unit (MOps/W) (numbers normalized to the *Base* platform). On average, the *Ideal* platform achieves a 40% improvement for this metric. The average improvement for the *Hybrid* platform is 27%, and 35% for the *Hybrid tolerant*. Again, our approach gets very close to the *Ideal*.

Figure 5c shows the number of read/write accesses performed in TCDM memory per power unit (MBytes/W), normalized to the *Base* platform. The values of this chart are really close to the ones reported for computation efficiency. This is an important outcome for our work, because it confirms that memory operations are the main limiting factor to reduce power consumption, and our approach leads to an equivalent increase of the computation power efficiency.

V. Related work

Many studies have shown that approximate computing is an amenable solution for applications in the ULP domain [22] [19] [16], and this facilitates the translation of such algorithms into energy-efficient hardware implementations. Architecture-level approaches include inserting control knobs in the design [10] and dual-voltage supply for SRAM and logic [13]. Focusing on the characteristics of ULP systems (where most of the energy is spent on on-chip memory), our approach is limited to 6T-SRAM memories, while other voltage domains are designed to work safety at low voltage. In the context of ULP multi-core systems this assumption simplifies the overall

[4]normalized energy values as reported in [25]

model with a minimum loss of generality.

Other HW approaches include approximate arithmetic blocks [15], hybrid memory architecture for MPEG-4 video processors [9] and dynamically reconfigurable SRAM array for low-power mobile multimedia application [11]. Such approaches share some commonalities with our techniques, but are clearly optimized for specific applications and lack the flexibility of our techniques.

Raising power concerns to the software enables a range of energy savings opportunities. Some approaches explore a set of abstractions for managing system resources under energy constraints [27] [5], targeting complete frameworks that put together architecture-, OS- and application-design. Approximate code transformations at the compiler level have been addressed in several works, targeting automatic identification/auto-tuning of tolerant code [3] [4] or code annotations [25]. The latter approach, EnerJ, is the closest to ours and probably the most known approach for approximate programming. For this reason we compare with this approach in our results (see Figure 5a). The comparison, limited to the results reported in [25], shows that our solution performs $\approx$6% better in terms of energy efficiency.

VI. Conclusion

In this work we propose a HW/SW approach to enable approximate computing on a multi-core ULP architecture. On the HW side we propose a hybrid L1 data memory design, composed of SCM and 6T-SRAM cells. We provide an additional mechanism to split multi-byte data into multiple banks, introducing a memory region to host variables that can tolerate computation approximation.

At the software level, we support OpenMP extensions for specifying what regions of code and data are tolerant to approximation. A compiler pass allocates data in various memory regions to allow for voltage reduction and, ultimately, energy savings. The experimental results show that our hybrid memory architecture can reduce by 25% on average the energy consumption, which is quite close to the 28% "ideal" reduction that we could achieve if the TCDM was entirely implemented with SCM cells.

VII. Acknowledgements

This work has been supported by the EU-funded research projects PHIDIAS (g.a. 318013) and P-SOCRATES (g.a. 611016).

References

[1] "clang: a C language family frontend for LLVM ," http://clang.llvm.org/, accessed: 2015-04-15.

[2] "OpenRISC project," http://www.opencores.org/or1k/, accessed: 2015-04-15.

[3] A. Agarwal, M. Rinard, S. Sidiroglou, S. Misailovic, and H. Hoffmann, "Using code perforation to improve performance, reduce energy consumption, and respond to failures," in *MIT CSAIL Technical Reports*, 2009.

[4] J. Ansel *et al.*, "Language and compiler support for auto-tuning variable-accuracy algorithms," in *Proceedings of the 9th Annual IEEE/ACM International Symposium on Code Generation and Optimization*, 2011.

[5] W. Baek and T. Chilimbi, "Green: A system for supporting energy-conscious programming using principled approximation," TR-2009-089, Microsoft Research, Tech. Rep., 2009.

[6] D. Bortolotti, C. Pinto, A. Marongiu, M. Ruggiero, and L. Benini, "VirtualSoC: A Full-System Simulation Environment for Massively Parallel Heterogeneous System-on-Chip." in *IPDPS Workshops*, 2013.

[7] B. H. Calhoun and A. Chandrakasan, "Analyzing static noise margin for sub-threshold SRAM in 65nm CMOS," in *Proceedings of the 31st European Conference on Solid-State Circuits*, 2005.

[8] S. T. Chakradhar and A. Raghunathan, "Best-effort computing: re-thinking parallel software and hardware," in *Proceedings of the 47th Design Automation Conference*, 2010.

[9] I. J. Chang, D. Mohapatra, and K. Roy, "A priority-based 6T/8T hybrid SRAM architecture for aggressive voltage scaling in video applications," *IEEE Transactions on Circuits and Systems for Video Technology*, 2011.

[10] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: exploiting algorithmic resilience for energy efficiency," in *Proceedings of the 47th Design Automation Conference*, 2010.

[11] M. Cho, J. Schlessman, W. Wolf, and S. Mukhopadhyay, "Reconfigurable SRAM architecture with spatial voltage scaling for low power mobile multimedia applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2011.

[12] A. Y. Dogan, J. Constantin, D. Atienza, A. Burg, and L. Benini, "Low-power processor architecture exploration for online biomedical signal analysis," *Circuits, Devices & Systems, IET*, 2012.

[13] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2012.

[14] D. Fick *et al.*, "Centip3De: A 3930DMIPS/W configurable near-threshold 3D stacked system with 64 ARM Cortex-M3 cores," in *IEEE International Conference on Solid-State Circuits*, 2012.

[15] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *18th IEEE European Test Symposium (ETS)*, 2013.

[16] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *Proceedings of the 1999 International Symposium on Low power Electronics and Design*, 1999.

[17] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang, "Synctium: a near-threshold stream processor for energy-constrained parallel applications," *IEEE Computer Architecture Letters*, 2010.

[18] C. Lattner and V. Adve, "LLVM: A Compilation Framework for Lifelong Program Analysis and Transformation," in *International Symposium on Code Generation and Optimization*, 2004.

[19] J. T. Ludwig, S. H. Nawab, and A. P. Chandrakasan, "Low-power digital filtering using approximate processing," *IEEE Journal of Solid-State Circuits*, 1996.

[20] A. Marongiu and L. Benini, "An OpenMP compiler for efficient use of distributed scratchpad memory in MPSoCs," *IEEE Transactions on Computers*, 2012.

[21] P. Meinerzhagen, S. Y. Sherazi, A. Burg, and J. N. Rodrigues, "Benchmarking of Standard-Cell Based Memories in the Sub-Domain in 65-nm CMOS Technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2011.

[22] S. H. Nawab, A. V. Oppenheim, A. P. Chandrakasan, J. M. Winograd, and J. T. Ludwig, "Approximate signal processing," *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, 1997.

[23] D. Rossi *et al.*, "Energy efficient parallel computing on the PULP platform with support for OpenMP," in *Electrical & Electronics Engineers in Israel (IEEEI), 2014 IEEE 28th Convention of*, 2014.

[24] D. Rossi, I. Loi, G. Haugou, and L. Benini, "Ultra low latency lightweight DMA for tightly coupled multi core clusters," *Proceedings of the 11th ACM Conference on Computing Frontiers*, 2014.

[25] A. Sampson *et al.*, "EnerJ: Approximate data types for safe and general low-power computation," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2011.

[26] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Controlled placement of standard cell memory arrays for high density and low power in 28nm FD-SOI," in *Design Automation Conference (ASP-DAC), 2015 20th Asia and South Pacific*, 2015.

[27] H. Zeng, C. S. Ellis, A. R. Lebeck, and A. Vahdat, "ECOSystem: Managing energy as a first class operating system resource," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, 2002.