Synergistic HW/SW Approximation Techniques for Ultralow-Power Parallel Computing

Synergistic HW/SW Approximation Techniques for Ultra-Low-Power Parallel Computing

Giuseppe Tagliavini, *Student Member, IEEE,* Davide Rossi, Andrea Marongiu, *Member, IEEE,* and Luca Benini, *Fellow, IEEE*

***Abstract*—Ultra-low-power embedded systems have recently started the move to multi-core designs. Aggressive voltage scaling techniques have the potential to reduce the power consumption within the admitted envelope, but memory operations on standard six-transistor static RAM (6T-SRAM) cells become unreliable at low voltages. While standard cell memory (SCM) overcomes this limitation, it has much lower area density than SRAM, and thus it is too costly. On the other hand, several applications have inherent tolerance to computation errors, and executing such workloads with approximation has already proven a viable way to reduce energy consumption. In this work we propose a novel HW/SW approach to design energy-efficient ultra-low-power systems which combine the key ideas of approximate computing and hybrid memory systems featuring both SCM and 6T-SRAM. We introduce a novel hardware support to split error-tolerant data so to host most significant bits (MSB) in the SCM and least significant bits (LSB) in the 6T-SRAM. This allows to power the memory system at a low voltage while ensuring correct operation by binding possible (flip-bit) errors to the LSBs only. In addition, by organizing 6T-SRAM banks into multiple and independent voltage domains we enable fine-grained, software-controlled voltage switching policies. At the software level, we propose language constructs to specify what regions of code and what variables are tolerant to approximation, plus compiler support to optimize data placement. Experimental results show that our proposal can reduce the energy consumption of the memory system by 47% on average, always complying with the result accuracy required by practical applications constraints.**

***Index Terms*—Approximation algorithms, Low-power electronics, Memory architecture, Multicore processing, Parallel programming**

I. Introduction

Embedded systems targeting ultra-low power (ULP) markets are mostly implemented with single-core microcontrollers [1][2][3]. Usually, the computational power delivered by such systems within the admitted power envelope is sufficient for satisfying the very low demand of the target applications. This assumption is less and less valid for today's deeply embedded sensing applications [4], whose computation requirements grow to match the complexity of increasingly sophisticated workloads. To match conflicting requirements for energy consumption and performance, near-threshold multi-core systems have been recently proposed [5][6][7] in application fields such as industrial automation, wearable consumer electronics, human-computer interfaces and pervasive video infrastructures. By joining parallelism with near threshold computing, these systems are able to provide more than one order of magnitude increase in energy efficiency [8] preserving the performance target.

In the context of energy efficient computing platforms operating in near-threshold, the memory system emerges as one of the most critical components, burning more than 50% of the total chip power [9][10]. While standard voltage scaling techniques can be to some extent applied to reduce energy, their aggressive use is not possible for memory energy reduction as operations on standard six-transistor static RAM (6T-SRAM) cells become unreliable at low voltages due to the lack of sufficient static noise margin and read/write stability [11]. On these premises, the design must take into account a specific voltage domain to be kept at higher voltage that includes the memory system, and energy requirements become even more memory-dominated. Advanced design techniques have been proposed to improve the performance of SRAM banks at low voltage [12][13][14], but overall their adoption is difficult due to area and cost considerations [15]. A promising solution consists of adopting a *heterogeneous memory architecture* [16][17], which includes both 6T-SRAM and standard-cell memory (SCM) banks.

The key idea of this approach is that SCM banks and 6T-SRAM banks can be powered at different voltage levels. At high voltage both memories operate reliably, but the SCM consumes significantly less energy. At low voltage, the 6T-SRAM has an associated probability of error (flip-bit) if it is read or written, while the SCM remains reliable and more energy efficient. Effectively managing a hybrid memory system requires techniques to partition data in a way that minimizes energy consumption. For example, greedy-allocation heuristics can be applied at the compiler level to place most frequently accessed data into the memory that maximizes the metric of interest, e.g., energy efficiency [18], predictability [19] or performance [20].

In recent years approximate computing has emerged as a promising approach to design energy-efficient digital systems working at unreliable voltage levels, ranging from functional units [21] to interconnect [22] and memory systems. The notion of approximate computing [23] refers to a set of techniques ranging from programming language- to transistor-level, with a common aim to allow computing systems to save

G. Tagliavini, D. Rossi, A. Marongiu, and L. Benini are with the Department of Electrical, Electronic and Information Engineering "Guglielmo Marconi" (DEI), University of Bologna. e-mail: {giuseppe.tagliavini, davide.rossi, a.marongiu, luca.benini}@unibo.it.

A. Marongiu and L. Benini are also members of the Department of Information Technology and Electrical Engineering of the Swiss Federal Institute of Technology Zurich (ETH Zurich). e-mail: {a.marongiu, luca.benini}@iis.ee.ethz.ch

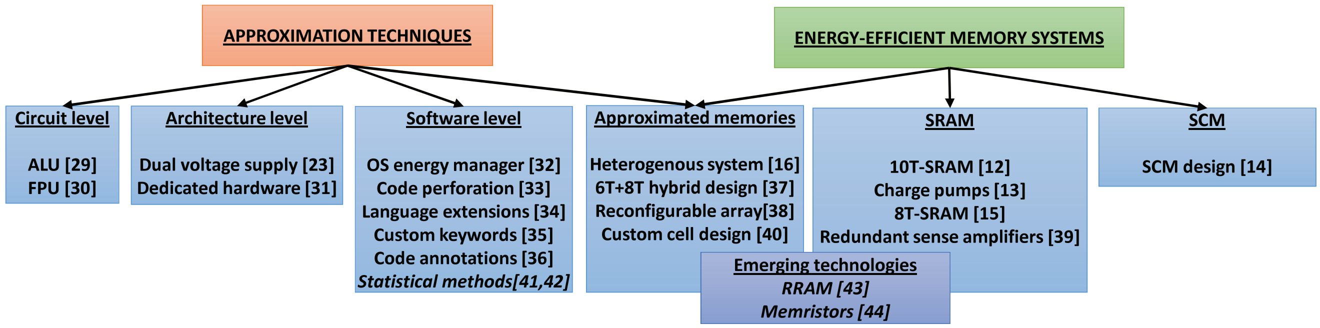


Fig. 1. Overview of the related work.

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 [24][25][26].

In this work we propose a novel HW/SW approach to design energy-efficient ULP systems which combine the key ideas of hybrid memory designs and approximate computing. At the architecture level, we design a mechanism to split multi-byte data that is “tolerant” to approximation into multiple memory banks. The most significant bits (MSBs) of a word are stored in the SCM, while the least significant bits (LSBs) are stored in the 6T-SRAM. Both memories can be safely powered at the lower voltage level: here the SCM operates reliably, while the probability of error on the 6T-SRAM is guaranteed to be tolerated by the computation (i.e., the error is bound to the LSBs). Our experiments demonstrate that this approach provides much better precision than just dropping the LSBs. More specifically, we provide an upper bound to the accepted error for each application included in the benchmark suite, and we show that our approach complies with these bounds with a safety margin.

To expose the hardware extensions at the software level, we introduce appropriate source code annotations used to specify what regions of code and what variables are tolerant to approximation. We choose to extend the OpenMP programming model, since this approach enables programmers to simply handle both parallelism and approximation efforts by annotating a C program with pragma directives. The annotations are processed by a compiler pass that implements an allocation heuristic which places data into one of the available logical memory areas (SCM, 6T-SRAM and split) according to their tolerance to errors. In addition, since different variables might be accessed in different program regions, we extend this heuristic to also take into account live ranges of tolerant variables. At the hardware level the 6T-SRAM memory is partitioned into multiple, independent voltage domains (1, 2 or 4), and the heuristic allocates variables with non-overlapping live ranges into distinct domains to allow for lowering the voltage when a variable is not accessed in the program.

The proposed techniques have been implemented in the parallel ultra-low-power platform (PULP) [7]. PULP is a scalable, clustered parallel computing platform that features a parametric number of processing elements (PE) per cluster, sharing a multi-banked tightly coupled L1 data memory (TCDM), acting as 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. To implement and validate the HW extensions required by our approach we used cycle-accurate simulation models, back-annotated with energy and performance numbers taken from a silicon implementation of the baseline system-on-chip (SoC) with 28 nm ultra-thin body and buried oxide fully depleted silicon-on-insulator (UTBB FDSOI) technology. Experimental results demonstrate that our approach can reduce the energy consumption of the memory system by 47% on average (for a set of real-world benchmarks) compared to the baseline SoC. Focusing on the whole-system energy, our technique allows on average 27% savings and outperforms other solutions. We finally show that our approach is fully compliant with a set of realistic accuracy levels deriving from practical constraints, which assesses the effective usability of the described techniques in real-life applications.

The rest of the paper is organized as follows. Section III introduces the baseline architecture, and then describes in further detail the proposed hardware extensions. Section II discusses the related work. Section IV describes the software stack of our framework, including the programming model, the runtime and the compiler. Section V introduces the framework setup and reports the experimental results. Finally Section VI concludes and points out future work directions

II. Related work

In the past few years approximate computing has been considered a promising approach in different research areas [27] [28]. Many studies have also pointed out that approximate computing is an amenable solution for applications in the ULP domain [24][25][26], and this facilitates the translation of such algorithms into energy-efficient hardware implementations. Figure 1 depicts a comprehensive overview of the related work. The figure highlights two main categories, that are *approximation techniques* and *energy-efficient memory systems*, from which we derive a set of related topics categories.

Circuit level [29][30] and *architecture level* [31][23] techniques have been used to to reduce overall energy consumption to the detriment of numerical accuracy. However, different research works observed that most of the energy consumed by ULP systems is spent on on-chip memory [9][10]. In addition, memory is the primary source of faults, while logic components are typically more robust. Starting from these considerations, our approach considers the use of two voltage levels for a set of 6T-SRAM domains, with the aim to achieve a significant power reduction with a disciplined relaxation to approximate results. The rest of our architecture is designed to work safely at low voltage, and this assumption simplifies the overall model with a minimum loss of generality.

Raising power concerns to the *software level* enables a range of energy savings opportunities at OS [32] and application level [33] [34] [35] [36]. These approaches require approximate code transformations with the aim to modify the application code to support the management of performance/accuracy tradeoffs. The extensions to the compiler toolchain required by our framework can be implemented at a higher level of abstraction w.r.t. the cited approaches. A specific support by the OS is not required, conversely the requested features can be provided by a lightweight runtime layer. Moreover, we introduce a minimum set of preprocessor directives to write error-tolerant parallel programs using an annotation mechanism that is of immediate use to embedded C/C++ programmers. Note that our approach could be considered orthogonal to other approximation techniques, since the programming model frontend can be decoupled by the approximation support, which includes the hardware design, the runtime layer and the compiler support. In this perspective the cited tools could be extended to leverage our approximation support by providing additional directives/keywords.

Approaches based on hardware design of *approximated memories* can be classified into two main categories, that are custom [37] and domain-specific [38] [16]. These solutions are tailored for specific applications or algorithms and achieve significant energy savings, but overall they lack the hardware and software support required to implement a general-purpose application. Using our platform we provide dedicated hardware, runtime features and compiler support to fulfill this goal.

Different approaches have been proposed to improve the performance of *SRAM* at low voltage [12] [13] [39]. A different approach to the problem is to implement low-voltage memory structures relying on standard-cell memory (*SCM*) [14]. All the described approaches cause serious overheads in terms of area, leakage and dynamic power consumption (at same supply voltage) with respect to standard 6T-SRAM [15]. Hence, their adoption for the implementation of the whole memory system is impractical, due to area and cost considerations. Frustaci et al. [40] explore the use of approximate SRAM banks in the context of error-tolerant applications, at the cost of the occurrence of read/write errors in the least significant bits of data. Although this technique is effective, it requires the design of custom SRAM banks featuring deep circuit-level optimizations, which leads to a low technology portability. Our approach leverages standard 6T-SRAM cells that can be realized with any memory generators provided by silicon vendors, and SCM that can be implemented with standard semi-custom design flows relying on industrially qualified standard-cells for implementation.

Recent works [41] [42] propose statistical techniques to measure the program response to injected approximation and derive the behavior of code and data. Compared to a programmer-annotated version, these techniques can lead to significant errors in some use cases, by marking as approximable a variable that is not tolerant for specific execution paths. Other works propose the adoption of *emerging technologies* to realize approximate memory cells, such as RRAM [43] and memristors [44]. These are promising approaches for the future, but today their application is limited to specific domains (e.g., neural networks) and requires a custom design. The solution proposed in this paper is totally general and can be directly applied using standard design flows provided by tool vendors.

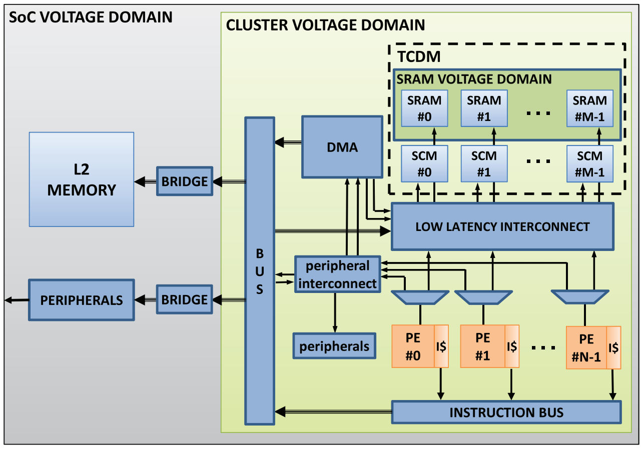


Fig. 2. The PULP architecture.

III. 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) [10][17] 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.) [45]. The following subsections describe the baseline PULP platform (Section III-A) and the extensions to the memory system introduced by our work to support computation approximation (Section III-B).

A. The PULP Architecture

PULP is a scalable, many-core computing fabric, organized as a set of *clusters*. Figure 2 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. A cluster includes a parametric number of processing elements (PEs) consisting of an optimized microarchitecture based on the OpenRISC ISA [46], each equipped with a private instruction cache. To avoid memory coherency overhead and increase energy efficiency the PEs do not have private data caches, but they share a L1 multi-banked tightly coupled data memory (TCDM). The TCDM is configured as a shared scratchpad memory, featuring as many

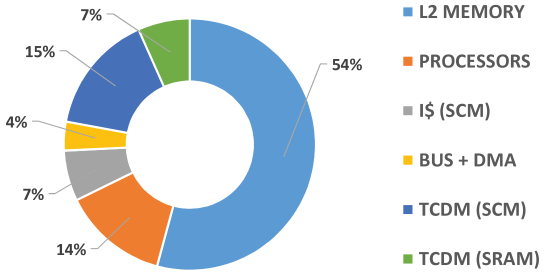


Fig. 3. Breakdown analysis of the PULP SoC area.

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. A lightweight, multi-channel DMA enables fast and flexible communication with other clusters, the L2 memory and external peripherals [47]. The arbitration and protocol adaptation necessary for the processors to communicate to the TCDM and peripheral interconnect is implemented by the DEMUX block connected to the data interface of each PE.

The PULP instance considered in this work consists of a single cluster featuring 8 PEs and a TCDM composed by 16 6T-SRAM banks of 4KB each (64KB total) and 16 SCM banks of 1KB each (16KB total), plus 256KB of L2 memory. Each core features 1 Kbyte of I$ implemented with SCM, hence reliable down to 0.5V. Three voltage domains are considered: i) the SoC domain includes the L2 memory and peripherals; ii) the cluster domain includes the PEs the SCM, the DMA and the cluster interconnect; and iii) the 6T-SRAM banks. A 28nm UTBB FDSOI (STMicroelectronics technology) implementation of the platform in this configuration can operate at 20MHz @ 0.5V. We extend this baseline platform to support computation approximation in strict cooperation with the programming model. Figure 3 shows the contributions of the hardware components to the total area of a PULP SoC, that we consider a baseline configuration for this work.

B. TCDM Reliability Extensions

On-chip memory is traditionally implemented with 6T-SRAM banks working in super-threshold operating region. When operating close to the threshold voltage, SCM has demonstrated a better tradeoff between reliability, energy efficiency, area and portability among technology nodes [14]. In particular, although 6T-SRAM cells provide a much better storage density than SCM ($\sim 3\times$), SCMs are reliable over all the operating voltage range of the architecture (0.5V – 0.8V) [37]. Accessing 6T-SRAM at a voltage lower than 0.8V may results in a flip-bit error, as shown in Table I. These values are derived executing the test patterns on PULP's silicon prototypes at different voltages using an Advantest SoCV93000 tester system. The program performs 10^{10} memory accesses on the 6T-SRAM memory area of PULP chips [17], and probability is computed from the error ratio.

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 ($\sim 4\times$) [48]. 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 **physical memory areas**, including 16KB of SCM and 64KB of 6T-SRAM. In this work we consider alternative scenarios which provide a different number of voltage domains for the 6T-SRAM area, corresponding to the following memory layouts described in Table II. Considering this size of the 6T-SRAM region (64KB), a further partitioning would produce a significant overhead, since the total area would be heavily dominated by the periphery and embedded power switches in small memory cuts.

TABLE I
PROBABILITY OF BIT-FLIP ERRORS IN 6T-SRAM.

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

A set of *reliability management units* (RMUs) are introduced in the path between the interconnect and the TCDM. These are simple combinational logic blocks that allow to remap the physical address range of the TCDM into three distinct **logical memory areas**:

- *SCM* – mapped in the SCM physical memory area (reliable at any operating point);
- *6T-SRAM* – mapped into the 6T-SRAM physical memory area (reliable at 0.8V), include multiple regions corresponding to distinct voltage domains;
- *split* – MSBs are mapped in the SCM physical memory area and LSBs in the 6T-SRAM physical memory area.

Figure 4 shows the new TCDM design and the defined memory areas for the memory layout including four 6T-SRAM voltage domains. Level shifters are required at the boundaries between voltage domains, i.e., when the 6T-SRAM is operating at 0.8V and the logic at a lower voltage (0.5V). Considering the power breakdown of each memory bank, the overhead of the level shifters is $< 1\%$. The impact on critical path is also negligible, mitigated by the fact that delay is dominated by SRAM banks when they operate at low-voltage (0.5V), while the paths toward TCDM is not critical when the 6T-SRAM area operates at high voltage (0.8V).

The RMUs provide access to the *split area* only at word or half-word level. In both cases, MSBs (upper 16 or 8 bits) are placed into SCM cells, while LSBs (lower 16 or 8 bits) are placed into 6T-SRAM cells. Figure 5 depicts the remapping of physical addresses when accessing the split area at word level. This implies that errors can show up only on LSBs at voltage levels below 0.8V, guaranteeing a correctness threshold for data when executing approximate code regions. In this design

TABLE II
MEMORY LAYOUT CHANGING THE 6T-SRAM VOLTAGE DOMAINS.

6T-SRAM domains	SCM cuts	6t-SRAM cuts
1	64 128 × 16	16 2048 × 16
2	64 128 × 16	32 1024 × 16
4	64 128 × 16	64 512 × 16

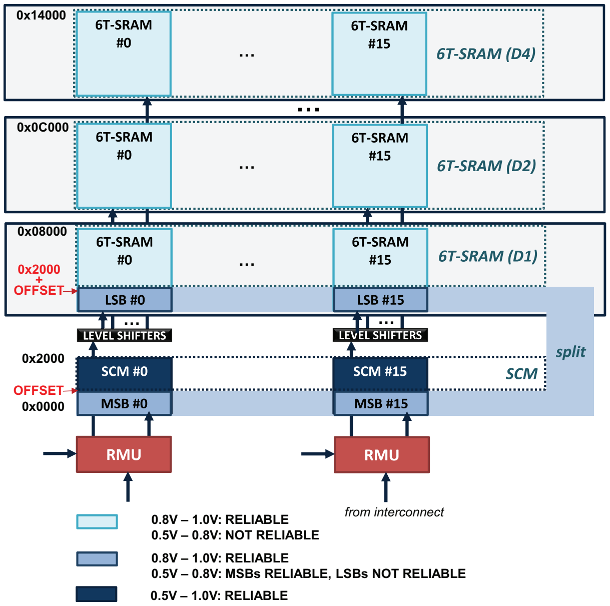


Fig. 4. Hybrid TCDM organization.

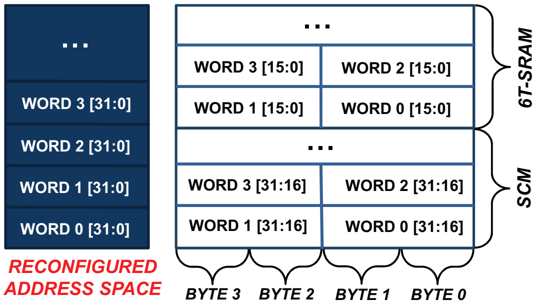


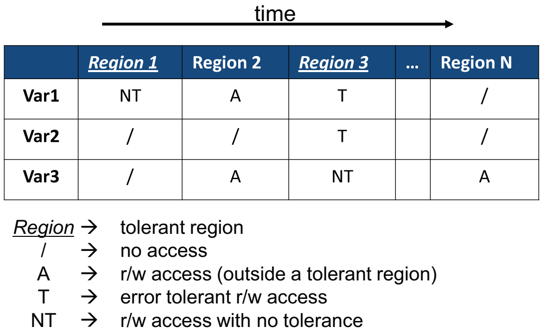
Fig. 5. Reconfigured address space for a word level access in the split memory area.

memory cuts are 16-bit wide. A word operation (32-bit) on SCM and 6T-RAM logical areas implies an access to two 16 bits banks in the corresponding physical area, while a word operation on the split logical area produces an access to a 16-bit bank in the SCM physical area and to a 16-bit bank in the 6T-SRAM physical area. The offset that defines the boundaries between the three logical memory areas 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 the optimization of application-specific policies. This design choice is due to the fact that 32-bit data types are common when programming ULP microcontrollers. A 64-bit operation in the split area is not supported, the compiler discards the tolerant flags related to 64-bits variables and reports a warning message.

IV. Software Stack

A. Programming Model

Many works in literature have shown that a variety of applications are tolerant to errors [49][50][51]. In most cases, some code regions of the application are inherently tolerant, while others must be protected from errors. To exploit the hardware support to reliability management, we propose a programming model which is based on a small set of annotations involving program statements and variables, that is a common approach in the field of approximate computing [36][23]. To fully exploit the parallel capabilities of our multi-core platform with a limited programming effort, we adhere to the OpenMP specification [52], which provides a model for parallel programming that is portable across different shared memory architectures. In the PULP 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 reside in the TCDM. The DMA unit introduced in Section III-A can be used to move data between the L2 and the TCDM. Based on these premises, we propose an extension to OpenMP including two constructs:



	Region 1	Region 2	*Region 3*	...	Region N
Var1	NT	A	T		/
Var2	/	/	T		/
Var3	/	A	NT		A

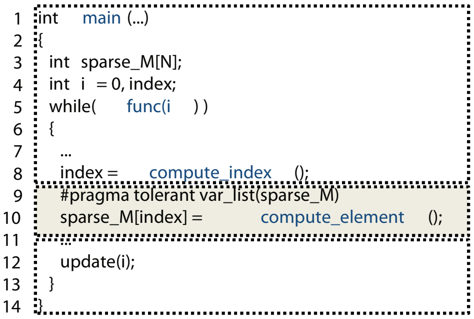
Region → tolerant region
/ → no access
A → r/w access (outside a tolerant region)
T → error tolerant r/w access
NT → r/w access with no tolerance

Fig. 6. Evolution of variables in different program regions.

- `#pragma tolerant` – a directive applied to a program statement to assert that the related code is tolerant to approximation;
- `var_list(var1, var2, ...)` – a clause coupled to the tolerant directive to qualify what variables can be actually approximated.

This model takes into account the full orthogonality between code and data in terms of approximation behavior. For analytical purposes the application code can be divided into multiple regions, each one including one or more statements. Each `#pragma tolerant` directive defines a *tolerant region*, while the rest of the code is grouped in *non-tolerant regions* (function starting/ending points and tolerant regions define their limits). At execution time, the program counter evolves through statements belonging to different regions, and each program variable can be accessed or not in a specific region (i.e., it is out of scope or not required). When a variable is accessed inside a tolerant region, it could be not tolerant to errors due to specific program constraints. Moreover, a variable can be error tolerant or not at different points of the execution flow, depending on the specific region it is accessed in. Figure 6 depicts the status of a set of variables when executing a use case that includes an alternation of tolerant and non-tolerant regions. In most practical cases programs start and end with non-tolerant regions with the precise aim to provide a consistent view to their execution environment.

Figure 7 shows an example of C code including a tolerant directive. In sparse matrix computation, matrix indexes can-



```
int main (...)
{
  int sparse_M[N];
  int i = 0, index;
  while( func(i ) )
  {
    ...
    index = compute_index ();
    #pragma tolerant var_list(sparse_M)
    sparse_M[index] = compute_element ();
    ...
    update(i);
  }
}
```

Fig. 7. A sparse matrix computation with a tolerant directive.

not be approximated (a single error implies wrong element choice), while matrix elements may tolerate approximation. Thus, we declare the sparse_M array and the element computation as tolerant, while index and its computation are not tolerant. Three code regions are highlighted in the code, a tolerant region and two non-tolerant. Note that error tolerance is not a property of sparse_M, but of the code region. When a tolerant variable is copied to a non-tolerant variable, its value is automatically promoted to non-tolerant state after the read. Also the opposite copy is permitted, for instance the automatic variable containing the result of compute_element() (non-tolerant) is copied to sparse_M (tolerant).

In principle, approximate algorithms can "absorb" errors which occur with a probability lower than a given threshold. To lower this idea in the context of our approach, we need a rigorous methodology to identify the tolerant regions and evaluate the impact of flip-bit errors on data accuracy. Even if an extensive discussion of automatic techniques is out of scope for this paper, we focus on describing the empirical process that we followed to derive suitable approximation thresholds for the various benchmarks. To annotate the applications described in Section V, we first performed a set of accuracy tests on a x86 workstation. We simulated the effects of an unreliable memory by instrumenting variables with a macro (APPROX) which injects flip-bit errors on the LSBs with a configurable probability. Different code regions can be assigned to a specific accuracy level by modifying the flip-bit probability with runtime calls, which correspond to voltage switch operations in the final platform. Statistics on the result accuracy are collected for different configurations, each one corresponding to a bijective association among program variables and accuracy levels. The acceptable error threshold is application specific, and we provide details in Section V-E.

B. Runtime Extensions

The framework described in this paper requires a lightweight runtime support. The extensions that must be added to the OpenMP runtime can be summarized as follows:

- an interface to activate the split memory area and set its actual size;
- an interface to allocate data structures in a specific logical memory area;
- an interface to modify the voltage supply of 6T-SRAM domains.

TABLE III
ENERGY CONSUMPTION (PJ) OF A 32-BIT READ/WRITE ACCESS.

6T-SRAM voltage	SCM	6T-SRAM	SPLIT
0.5V	0.6 / 0.6	-	3.2 / 2.9
0.8V	0.6 / 0.6	13.6 / 12.2	7.1 / 6.4

In addition, the main modification to the OpenMP runtime involves the allocation of internal data structures. Whenever an OpenMP construct is present inside a tolerant region, the data structures describing its behavior are not tolerant to errors, and they are allocated partly on the stack (e.g., the bound variables generated to distribute the workload among processing elements in loop constructs) and partly on the heap (e.g., the work-share descriptors). The heap area for these data structures is typically reserved using a dynamic allocator (a call to a malloc in POSIX API). To handle this aspect of the runtime environment, we introduced the concept of *allocation control variable*. This is an internal control variable, akin to others defined by the OpenMP standard, that specifies the logical memory region the runtime must allocate its dynamic data structures in. The positioning of the stack and the value of this variable is a further control knob for our architecture, as described in the next section.

C. Compile-time Optimizations

Our compile time optimizations are based on the analysis of the application *call graph*, that is the directed graph representing calling relationships between functions in the execution flow. After applying the *outlining* technique typically used for OpenMP constructs, each statement annotated by a #pragma tolerant directive is translated into an equivalent function, and so it follows that the statement in the program flow is replaced by a function call. After this source code transformation, each tolerant region corresponds to a single node in the call graph, while original nodes represent sets of non-tolerant regions. As a preliminary step, an extended *use-defs* analysis is applied over the call graph to extract statistics about global and local variables[1]. For each variable the following attributes are collected:

- *scope* – a flag that specifies whether the variable is local or global;
- *dynamic* – a flag that specifies whether the variable is allocated statically (stack or global variable) or dynamically (call to malloc);
- *class* – the type of the variable (scalar/array/reference);
- *use-intervals* – live range of the variable within its entire scope;
- *tolerant-use-intervals* – live range of the variable restricted to tolerant code regions;
- *usedefs* – use-defs chains for the variable.

Table III reports the memory consumption (in pJ) of a single 32-bit read/write access to different logical memory regions. It considers the case of a single 6T-SRAM voltage domain, but

[1] 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 considered to avoid this limitation.

the trend is similar when the number of domains increases. Taking into account the high voltage level (0.8V), an access to SCM costs less than an access to 6T-SRAM, while an access to the split memory area costs more than an access to SCM but less than an access to 6T-SRAM; at the low voltage level (0.5V), the cost of accessing the split memory area decreases, while 6T-SRAM cannot be accessed any more due to precision constraints. Accesses to the 6T-SRAM logical memory area are not allowed below 0.8V. Considering the reported values, an approach that uses data tiling on the SCM could be considered the most viable solution to minimize the energy consumption. By leveraging the knowledge of the memory access pattern, data tiling exploits spatial locality in a program by partitioning large data structures into smaller chunks that are brought in and out of the target memory via DMA transfers. By reducing the granularity of data tiling it is theoretically possible to efficiently manage even very small memories. However, creating smaller tiles implies increasing the number of transfers required to process a given data structures, which is subject to increasing impact of DMA latency and programming overheads. Due to these considerations, the energy consumption of the whole system is indeed greater w.r.t. other solutions due to overhead effects, as explained in further detail in Section V.

Taking into account the usage of both SCM and 6T-SRAM areas, the problem of allocating variables into logical memory areas minimizing the energy consumption can be solved using two general approaches, a formal integer linear programming (ILP) model or a heuristic algorithm. ILP models of the memory allocation problem are known in literature [53]. However in this context they cannot be fed with a complete set of parameters, since experimental evidence shows that access frequency and data size cannot be exactly determined in all practical cases by means of a static analysis.

On the basis of the previous considerations, variables are allocated into different logical memory regions using a *heuristic algorithm* based on the following policies:

- tolerant variables are allocated in the split area, since SCM is a limited resource while using the 6T-SRAM logical area could miss a voltage switch opportunity;
- non-tolerant variables are allocated in SCM, with the aim to switch down the voltage of unused 6T-SRAM domains;
- variables accessed inside a non-tolerant region (if not considered by previous rules) are allocated in a single 6T-SRAM voltage domain;
- variables devoted to SCM and split memory areas are organized into separate lists, ordered on the basis of the most-frequent-accessed-first heuristic. The variables that eventually do no fit the related memory area are allocated in 6T-SRAM areas, applying (when feasible) the single-domain-per-area heuristic;
- the allocation control variable in the OpenMP runtime is set to use SCM (if available), or a previously allocated 6T-SRAM area with free space (in the worst case it could reference a new 6T-SRAM domain, which cannot be switched down in the region containing the current OpenMP directive);
- the voltage levels of unused domains (i.e., domains which are not used for allocation or contain variables that are not accessed) are switched down at the beginning of a region, with the aim to reduce the leakage power of the related memory banks.

All these transformations involving the source code are performed at compile time on the current application, but in any case they do no affect the binary size since program data are moved but not increased. This algorithm takes into account all global variables and a subset of local variables, including: arrays, variables allocated with a malloc request, and local variables used in `var_list` clauses. Ultimately, what is not included are automatic variables that are left on the program stack. In the most common cases, the stack is accessed when executing tolerant regions and it typically contains non-tolerant variables (e.g., indexes and pointers), so it must be allocated in SCM or in a 6T-SRAM region kept at high voltage. The previous steps guarantee that the stack is reduced to a minimal set of scalar variables; since the experimental evidence shows that stack accesses are still frequent in tolerant regions after this process, in our solution the stack is preferably allocated in the SCM area. The size of the stack can be determined through static analysis in some cases (e.g., consider the restrictions to the standard C language imposed by OpenCL for kernels [54]), but this is not true in the most general case. For most applications included in our benchmark suite, the aforementioned heuristics enable the allocation of a limited stack area (512 bytes per core). A check for stack overflow is provided at hardware level to guarantee a controlled termination in all the cases that are not properly managed by the static analysis, and therefore require a manual stack sizing.

V. Experimental Evaluation

This section introduces the methodology adopted to validate our architecture, and then focuses on the implementation of the software stack and the reference benchmarks. The results for most relevant metrics are reported and commented (energy consumption in Section V-C, area compared to energy in Section V-D and accuracy in Section V-E).

A. Simulation Setup and Methodology

The power consumption of the baseline architecture at different voltage levels has been measured on the first silicon implementation of PULP, realized with STMicroelectronics 28nm UTBB FDSOI technology [10]. Figure 8 depicts the layout of this reference chip, highlighting its key subsystems. The power figures measured from real silicon have been then partitioned among the key components of the platform (cores, I$, SRAM banks, SCM banks, interconnect) taking into account back-annotated simulations performed on the post place&route netlist of the SoC. Finally, the energy numbers have been fed into the models of Virtual SoC [55], a SystemC-based cycle-accurate virtual platform for the simulation of massively parallel heterogeneous architectures, that is used to perform a design exploration of the extensions proposed in this work. Table III reports the dynamic energy required for each read/write operation. Table V reports the leakage energy

TABLE IV
CHARACTERIZATION OF THE BENCHMARK SUITE.

Application	Description	Regions	Tol. regions	Tolerant data %
Color Tracking	A sequence of image processing filters, with the aim to find the center of mass for objects of a specific color in a video	5	3	100% / 50% / 95%
HOG	Histogram of oriented gradients, that is a feature descriptor used to perform object detection	5	3	50% / 80% / 50%
CNN	A convolutional neural network including 6 layers	16	6	50% / 100% / 50% / 100% / 50% / 100%
Health	A signal processing algorithm to process ECG data series with the aim to predict seizures	5	2	50% / 50%
Navi	A navigation support for unmanned vehicles, including the Djikstra algorithm (to find the shortest path between known locations) and a heuristic to plan recharging stops (based on distance and power consumption)	6	2	50% / 25%

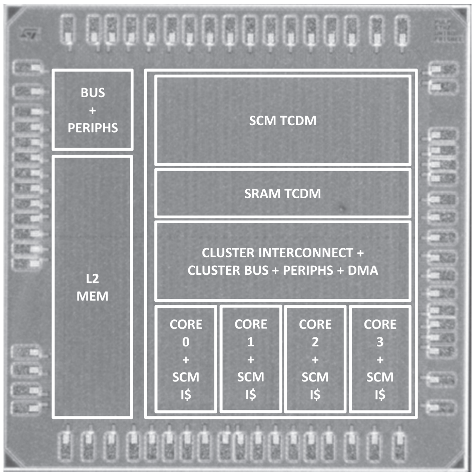


Fig. 8. Layout of the PULP chip used for memory reliability and power characterization.

consumed by the memory cuts for a single clock cycle at 20MHz, that is the operational frequency of our platform. The size of the memory cuts is the one described in Table II for alternative setups. 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. Compared to a complete RTL simulation, the virtual platform guarantees a maximum error on the number of reported cycles that is between 5% and 6%. This error has a minimum impact on the leakage component of the total energy.

B. Software Stack and Benchmark Suite

To experimentally validate our framework, we set up a toolchain based on *Clang* [56] and *LLVM* [57]. The compiler frontend transforms the directives into annotated tokens in its intermediate representation (IR) format. These annotations are collected and interpreted by the heuristic algorithm described in Section IV-C, which is implemented as an IR optimization pass in the LLVM compiler infrastructure. To provide programming model and runtime support, we extended an OpenMP implementation tailored for embedded multicore systems [58], adding the features discussed in Sections IV-A and IV-B. The benchmarks are implemented in C using standard OpenMP directives to split the workload over the 8 available cores. Considering the intrinsic data parallelism of the computational kernels, for the selected benchmarks we use a `omp parallel for` directive with a static chunking pattern. The data footprint of considered applications does not entirely fit into the TCDM, so we use the DMA engine to implement a *data tiling* technique; in addition, we adopt *double-buffering* with the aim to hide the DMA transfer latency. The applications included in the benchmark suite are described in Table IV, that also provides more information on single applications. *Regions* is the total number of code regions, while *Tol. regions* includes only the tolerant regions (Section IV-C). *Tolerant data %* reports the percentage of accessed data which are error tolerant in the corresponding region. For instance, in the first tolerant region of `Color Tracking` all the data are tolerant, in the second one half data are tolerant and the other half are not.

TABLE V
LEAKAGE ENERGY (PJ PER CYCLE) OF SCM AND 6T-SRAM CUTS.

	SCM@0.5V	6T-SRAM@0.5V	6T-SRAM@0.8V
1 region	0.002	0.0056	0.021
2 regions	0.002	0.0043	0.013
4 regions	0.002	0.0022	0.0083

C. Energy Consumption

In the experiments that follow we take into account five alternative configurations:

- *Default* – No optimization is applied, program data are allocated into TCDM using standard compilation and linking policies;
- *Heuristic* – Program data are allocated in TCDM using a most-accessed-first heuristic;
- *Heur+VS* – Like *Heuristic*, but enabling low voltage operation for SRAM memory areas that are not used within a target code region;
- *Split+VS* – Tolerant data is allocated in the split memory area and the allocation heuristic maximizes low voltage SRAM memory operation;
- *Tiling* – All program data is accessed in small chunks from the SCM only, using DMA transfers to update

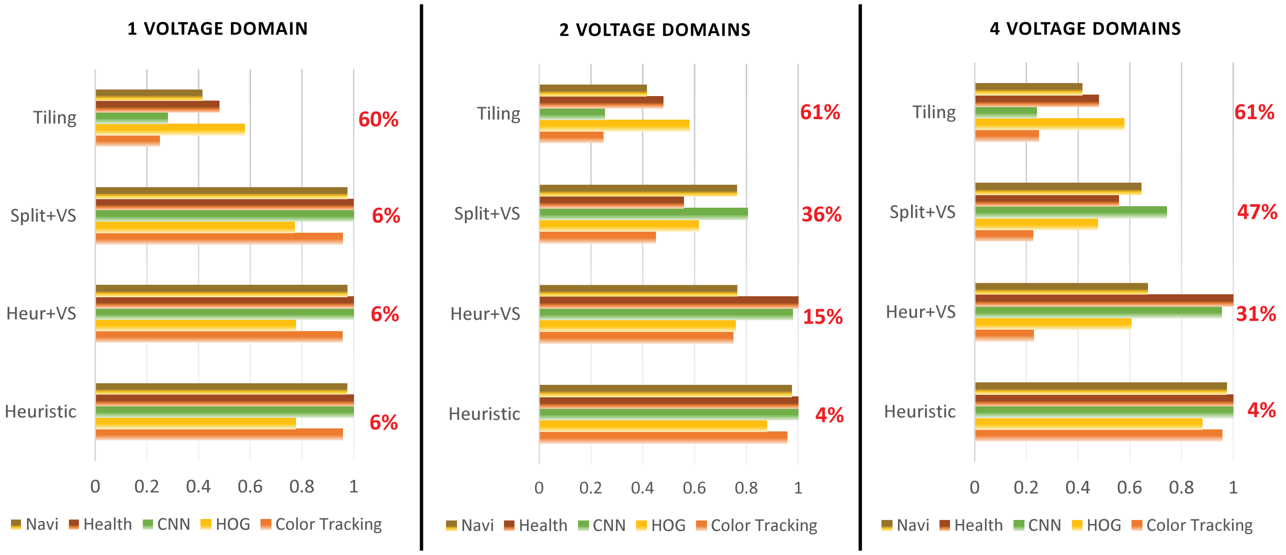


Fig. 9. Normalized energy consumption of the TCDM (average energy reduction is reported in percentage).

continuously its content. SRAM is constantly powered at low voltage.

We first analyze the behavior of the memory system in isolation. Figure 9 depicts the energy consumption of the TCDM system, including the memory banks and the RMU logic block. The figure shows three plots, one for each explored configuration in terms of SRAM voltage domains: 1 (left), 2 (middle) and 4 (right). Each bar in a group depicts the energy consumption of a given application (normalized to the *Default* configuration), and different groups of bars represent a different configuration from the list above. The average energy reduction (% of the *Default* configuration) is also reported for each group. Considering a single SRAM voltage domain, *Heuristic*, *Heur+VS* and *Split+VS* are identical. This is expected, because unless all data fits in the SCM (which happens only for *Tiling*), it is impossible to lower the voltage of the SRAM without corrupting the correctness of the results. Increasing the number of voltage domains, the benefits of *Split* are more and more evident. With 2 voltage domains, the energy efficiency is increased by 15% on average by just enabling voltage switching on idle SRAM banks, and by 36% by allocating tolerant data on the split memory area. With 4 voltage domains, these values are respectively 31% and 47%. *Tiling* is ≈60% more efficient than the *Default* configuration, independent of the number of voltage domains (as the SRAM is constantly powered at a low voltage). This is not surprising, as SCM accesses are much more energy efficient than SRAM accesses, as reported in Table III.

However, while every configuration requires DMA transfers to accommodate larger data structures than the whole TCDM in chunks, *Tiling* requires more DMA transfers that any other configuration. This is because the SCM is only a fraction of the total TCDM size (one fifth in our architecture), and part of it is devoted to data that always need to be reliably accessed, such as the stacks of the threads and the OpenMP runtime metadata. As a consequence, the main drawback of this configuration is that it is more sensitive to DMA management overhead (the finer and more numerous the DMA transfers, the higher the overhead for their management). Focusing on the whole system energy consumption in Figure 11 this effect becomes evident. In this plot, energy numbers are normalized to the *Default* configuration and broken down into the contributions of the TCDM system (what we already showed) and other SoC components (cores, I$, DMA engine, interconnect and L2). Again, the figure shows three plots, one for each explored configuration in terms of SRAM voltage domains: 1 (left), 2 (middle) and 4 (right). The higher energy values reported for `HOG` and `Navi` in the *Tiling* configuration are due to the fact that these applications require a bigger stack (768 bytes per core) and a larger footprint of the OpenMP metadata (due to the use of a higher number of dynamic parallelization constructs). Compared to other applications, this reduces the available SCM space to host data tiles, which will eventually be very small and numerous and ultimately imply a larger DMA overhead. Table VI reports execution cycles for *Tiling* compared to other configurations. The effect of DMA management is clearly visible also in terms of execution time, with the obvious effect on overall energy consumption[2]. On average, the *Split+VS* approach allows a 13% reduction of the total

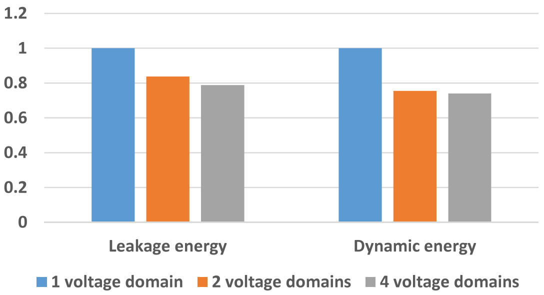


Fig. 10. Normalized components of the energy consumption.

[2] The execution time is not affected by the number of voltage domains, as the performance overhead of a voltage switch is just 2 additional cycles.

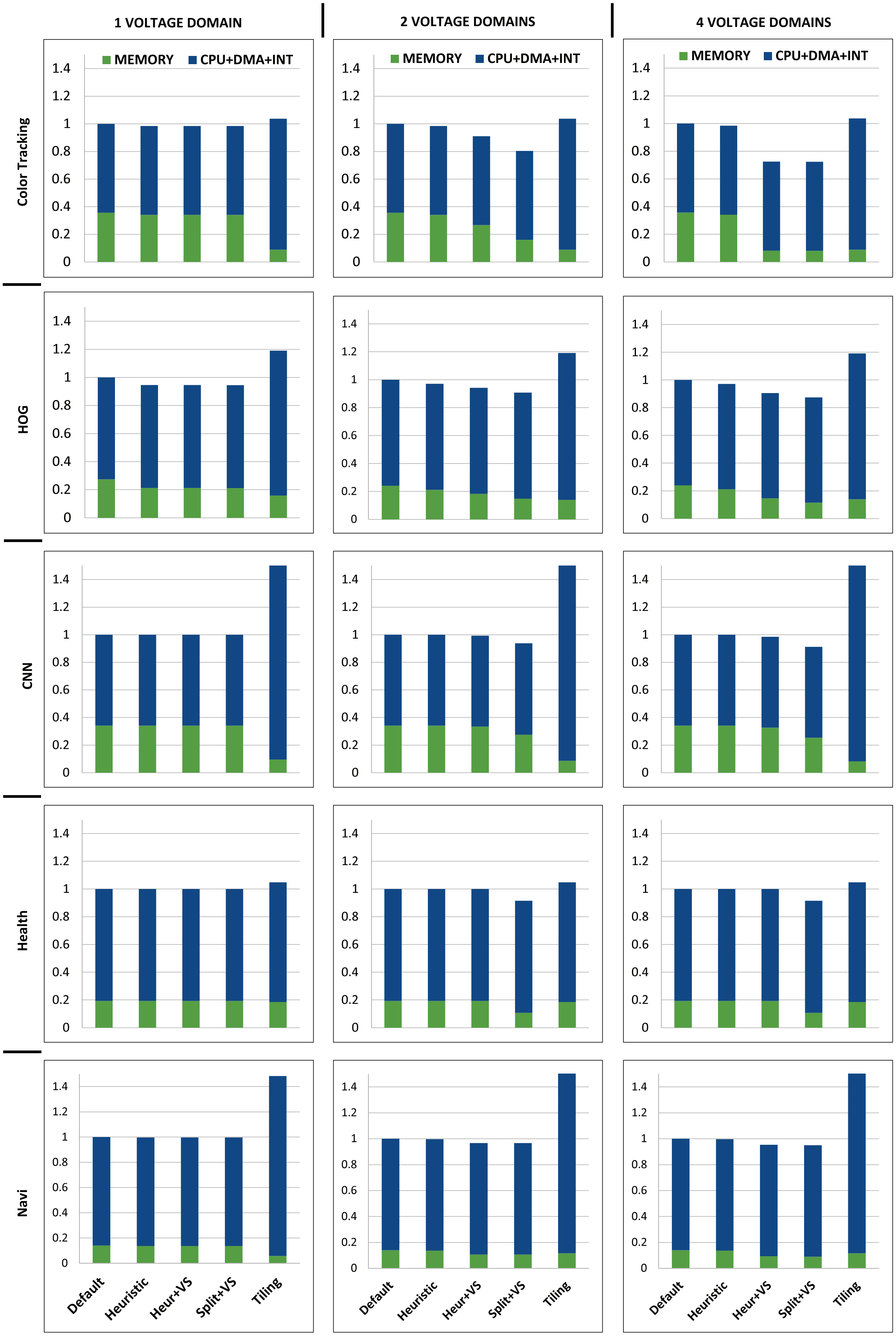


Fig. 11. Normalized energy consumption of the full SoC.

TABLE VI
TOTAL NUMBER OF CORE CYCLES (IN MILLIONS).

	Others	Tiling
Color Tracking	29.48	32.15
HOG	70487.50	134540.00
CNN	77.78	141.66
Health	1460.16	1563.77
Navi	54.12	99.49

SoC energy (up to 28% for the considered applications). The benefits of *Tiling* only become visible when the granularity of the technique is such that DMA management overhead becomes negligible (see Section V-D). This approach never allows tangible benefits for the SCM size considered in this work. It increases the average energy consumption by 27% (by 54% in the worst case).

Figure 10 shows dynamic and leakage components of the energy consumption for the *Split+VS* approach. The reported values are the averages of all the applications and are normalized to the baseline case of a single voltage domain. Both leakage and dynamics components of the energy consumption are reduced by our approach. The leakage energy is reduced by switching down the voltage of the 6T-SRAM banks that are not accessed or are part of the split area. The dynamic energy is reduced accessing the 6T-SRAM banks at low voltage in the split area and maximizing the access to the SCM area.

Note that the *Split+VS* approach can work in synergy with other approximation techniques, for instance arithmetic units [29] or interconnects [22], with the aim to increase the energy saving of the full system. In this perspective, our programming model can be also extended to support additional knobs.

D. SoC Area

Table VII reports the area increase to implement our *Split+VS* technique when considering 2 and 4 SRAM voltage domains. The table also reports average and max energy savings enabled by the technique. *Normalized energy* and *Normalized area* are referred to the baseline SoC (with a single SRAM voltage domain). Looking at energy alone, 4 SRAM voltage domains are better than 2 only. If we consider a combined energy×area metric the difference is no longer visible (for both the average and the best case).

Another approach to use additional design area would obviously be that of increasing the size of the SCM. As we discussed earlier, this would have a beneficial effect on the *Tiling* approach, as it would allow to amortize the overhead of DMA programming. Figure 12 compares the effectiveness of *Split* and *Tiling* approaches at using additional chip area.

TABLE VII
AREA AND ENERGY/AREA VALUES.

	1 domain	2 domains		4 domains	
		Average	Best	Average	Best
Area (μm^2)	2943648	2968768		3062400	
Normalized energy	1.00	0.91	0.80	0.88	0.72
Normalized area	1.00	1.01		1.04	
Norm en. × area	1.00	0.92	0.81	0.92	0.81

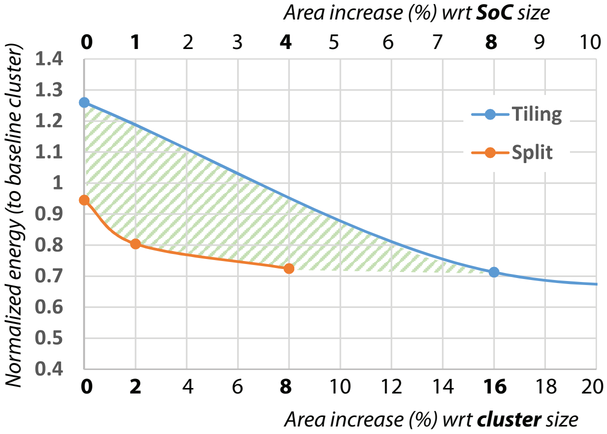


Fig. 12. Normalized energy consumption for two solutions (SCM with tiling VS hybrid memory with our approach).

Specifically, the X-axis reports the area increase (% of the original design; for the whole SoC on top and for the cluster only on bottom) and the Y-axis reports normalized energy (compared to the *Default* allocation). The plot shows that *Tiling* is capable of amortizing DMA overheads when the SCM size is increased to 24KB, which corresponds to 16% of the original cluster area. *Split* makes better use of additional chip area when the number of voltage domains is increased to up to 4. Beyond this number we reach a plateau in the energy saving curve. Further partitioning the 6T-SRAM into additional voltage domains, the total area would be dominated by the periphery and embedded power switches. The area between the two curves represent the design space where *Split* is more energy/area-efficient than *Tiling*.

E. Application accuracy

To assess the benefits of our approach, we compared the use of unreliable LSBs 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 data types, but in the second case we forced the LSBs to zero. Practically, the zeroing case represents an upper bound to the LSB error, since it totally discards the LSB part. All computations are performed on the integer range (32 bit words), and the values reported in the flip-bit column are the worst cases over 1000 executions. To be conservative, we used the highest flip-bit probability considered by our platform (0.0037, as reported in Table I). Table VIII reports the value of the mean squared error (MSE) for both approaches, compared to a maximum accepted value. The maximum accepted MSE is not an inherent property of the algorithms, nevertheless it is a requirement of the application context. To derive meaningful values for our benchmarks, we made some hypotheses based on practical use cases. Most works on approximate computing use a similar approach, first checking a wide but yet limited subset of the output and then assuming that it is representative. These approaches have been shown to produce average errors that are acceptable for the considered use cases, but they cannot take corrective actions when large errors occur. Recent works [59][60] tackle the result quality problem applying

TABLE VIII
MEAN SQUARED ERROR FOR ZEROING AND FLIP-BIT ERROR.

Benchmark	Zeroing	Flip-bit	Max MSE
Color Tracking	676	64	225
HOG	2.12E+13	58564	2814663
CNN	17114769	26244	36864
Health	6867734	378	25000
Navi	1681	36	100

runtime checks; a lightweight application-specific metric could be used to derive a quality check of the result, with the aim to adapt the application behavior to an unacceptable quality loss. While this research topic is really promising, its discussion is beyond the scope of this paper. However, runtime techniques to support advanced quality management can be straightforwardly implemented on top of our framework.

`Color Tracking` computes a point centered on the object of a specified color in the input image, and the error metric is based on the Chebyshev distance. Considering an industrial application with a 640×480 image source and a minimum object size of 60 pixels enforced by fixed camera positioning, a maximum error of 15 pixels guarantees that the midline between the center of mass and the object border is not exceeded by the approximated value. From these premises, the maximum MSE is 225. `HOG` computes a feature descriptor that counts discretized occurrences of gradient orientations in different regions of an input image. For human detection applications, the miss rate of HOG-based solutions is around 0.5. We verified that a 0.01% error rate on the computed descriptor does not affect this recognition rate. Using the difference between the binary checksums of feature descriptors as an error metric, this corresponds to a maximum MSE of 2814663 in our experimental setup. `CNN` computes a set of binary features filtered by multiple neural layers. Applied to face recognition, a CNN can achieve a 98% recognition rate. Using the same error metric of the previous benchmark, also in this case we verified that a 0.01% error rate does not affect the expected results. This error rate corresponds to a maximum MSE of 36864 in our experimental setup. `Health` computes the energy levels associated to its final wavelet transform. Considering the typical dynamics of energy levels in our experimental setup, an absolute error of 25 on a single energy level does not compromise the results. Using the average of the differences of energy levels to measure the error, this corresponds to a maximum MSE of 2500. `Navi` computes the travel plan and the required recharging stops of a unmanned vehicle. We consider the total travel time as a key metric to evaluate errors, and then we consider 10 minutes as an upper bound for the maximum delay acceptable by an impatient human being. Finally the maximum MSE is 100.

F. Comparison with other approaches

In this section we compare energy and area of our solution to several alternative approaches: (i) *6T* uses 6T-SRAM without applying voltage scaling; (ii) *6T+VS* is the solution adopted by EnerJ [36], using 6T-SRAM and voltage scaling; (iii) *8T/6T+VS* and *10T/6T+VS* implement a hybrid memory system, using respectively 8T-SRAM [15] and 10T-SRAM [12]; (iv) *SCM* uses SCM cuts to implement the full SRAM. For each of the proposed approaches we configure our simulation infrastructure (introduced in Section V-A) based on energy numbers reported in the literature and we collect the results running the benchmarks previously described.

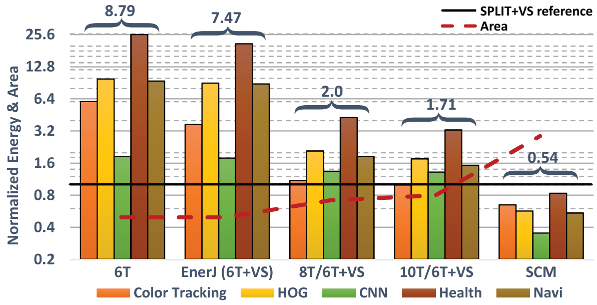


Fig. 13. Energy consumption and area compared to Split-VS.

TABLE IX
NORMALIZED ENERGY-AREA PRODUCT (NEAP).

	Our	6T	6T+VS	8T/6T+VS	10T/6T+VS	SCM
Memory	1.00	4.50	3.83	1.48	1.41	1.58
Cluster	1.00	1.74	1.58	1.03	1.03	1.80

Since our technique is explicitly aimed at reducing energy spent in the memory, we first focus on energy and area numbers for the memory subsystem only. Results are shown in Figure 13 (bars represent memory energy and the dashed line represents memory subsystem area). Energy/area numbers for each approach are normalized to energy/area numbers for our technique (numbers below one are better than our solution, numbers above one are worse than our solution). The numbers on top of each group of bars show average normalized energy for each benchmark. On average, EnerJ consumes 8.9× higher energy than our solution. From the comparison with 8T/6T+VS and 10T/6T+VS architectures it is also evident that hybrid memory enables major energy savings.

Since from these results there is an obvious trade-off between energy-consumption and area, we also show results for a combined metric – normalized energy-area product (NEAP). We show NEAP for the memory part only (like in Figure 13) and for the whole cluster (the benefits of our approach are less evident). NEAP for these two configurations is shown in Table IX. Focusing on the memory part only, our solution always provides the best NEAP. When considering also the rest of the cluster components, the benefits are reduced, as expected. However, although the techniques *8T/6T+VS* and *10T/6T+VS* reach very close NEAP to ours, they never perform better.

VI. CONCLUSION

In this work we propose a novel HW/SW approach to design energy-efficient ULP architectures which combine approximate computing and hybrid memory systems featuring both SCM and 6T-SRAM. At the hardware level, we introduce a support to split error-tolerant data so to host most significant bits (MSB) in the SCM and least significant bits (LSB) in the 6T-SRAM. This allows to power the TCDM system at a low voltage while ensuring correct operation by binding potential flip-bit errors to the LSBs only. In addition, by organizing 6T-SRAM banks into multiple and independent voltage domains we enable fine-grained, software-controlled voltage switching policies. At the software level, we propose language constructs to specify what regions of code and what variables are tolerant to approximation. A compiler pass implements a heuristic algorithm which allocates data into available memory regions and leverages hardware knobs to maximize energy savings. Experimental results show that our hybrid memory architecture can reduce by 47% the energy consumption of the TCDM memory. Focusing on the whole-system, our technique allows on average 27% savings and outperforms other solutions. At the same time we can guarantee the exact level of accuracy required by real-life applications, since the MSE of our approach is always below a maximum reference value when a proper approximation policy is applied. Overall, these results encourage further research activities in the field of hybrid memory architectures, as these solutions represent a promising opportunity for the design of future ULP systems.

Acknowledgment

This work is supported by the European FP7 ERC Advanced project MULTITHERMAN (g.a. 291125) and by IcySoC and YINS RTD projects, evaluated by the Swiss NSF and funded by Nano-Tera.ch with Swiss Confederation financing.

References

[1] "STM32L4 ultra-low-power MCUs," http://www.st.com/ stm32l4.

[2] "Texas Instruments MSP Microcontrollers," http://www.ti.com/lsds/ti/microcontrollers_16-bit_32-bit/msp/overview.page.

[3] "Ambiq Apollo," http://ambiqmicro.com/low-power-microcontroller.

[4] A. Y. Dogan, J. Constantin, D. Atienza, A. Burg, and L. Benini, "Low-power processor architecture exploration for online biomedical signal analysis," *IET Circuits, Devices Systems*, vol. 6, no. 5, pp. 279–286, Sept 2012.

[5] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang, "Synctium: a Near-Threshold Stream Processor for Energy-Constrained Parallel Applications," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 21–24, Jan 2010.

[6] D. Fick, R. G. Dreslinski, B. Giridhar, G. Kim, S. Seo, M. Fojtik, S. Satpathy, Y. Lee, D. Kim, N. Liu, M. Wieckowski, G. Chen, T. Mudge, D. Sylvester, and D. Blaauw, "Centip3De: A 3930DMIPS/W configurable near-threshold 3D stacked system with 64 ARM Cortex-M3 cores," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*, Feb 2012, pp. 190–192.

[7] D. Rossi, I. Loi, F. Conti, G. Tagliavini, A. Pullini, and A. Marongiu, "Energy efficient parallel computing on the PULP platform with support for OpenMP," in *Electrical Electronics Engineers in Israel (IEEEI), 2014 IEEE 28th Convention of*, Dec 2014, pp. 1–5.

[8] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge, "Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 253–266, Feb 2010.

[9] D. Bol, J. De Vos, C. Hocquet, F. Botman, F. Durvaux, S. Boyd, D. Flandre, and J.-D. Legat, "A 25MHz 7μW/MHz ultra-low-voltage microcontroller SoC in 65nm LP/GP CMOS for low-carbon wireless sensor nodes," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2012 IEEE International*. IEEE, 2012, pp. 490–492.

[10] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gürkaynak, A. Bartolini, P. Flatresse, and L. Benini, "A 60 GOPS/W,- 1.8 V to 0.9 V body bias ULP cluster in 28nm UTBB FD-SOI technology," *Solid-State Electronics*, vol. 117, pp. 170–184, 2016.

[11] B. H. Calhoun and A. Chandrakasan, "Analyzing static noise margin for sub-threshold SRAM in 65nm CMOS," in *Solid-State Circuits Conference, 2005. ESSCIRC 2005. Proceedings of the 31st European*, Sept 2005, pp. 363–366.

[12] M. E. Sinangil, N. Verma, and A. P. Chandrakasan, "A Reconfigurable 8T Ultra-Dynamic Voltage Scalable (U-DVS) SRAM in 65 nm CMOS," *IEEE Journal of Solid-State Circuits*, vol. 44, no. 11, pp. 3163–3173, Nov 2009.

[13] B. H. Calhoun and A. P. Chandrakasan, "A 256-kb 65-nm Sub-threshold SRAM Design for Ultra-Low-Voltage Operation," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 3, pp. 680–688, March 2007.

[14] A. Teman, D. Rossi, P. Meinerzhagen, L. Benini, and A. Burg, "Power, Area, and Performance Optimization of Standard Cell Memory Arrays through Controlled Placement," in *ACM Transactions on Design Automation of Electronic Systems*, 2016.

[15] P. Meinerzhagen, S. M. Y. Sherazi, A. Burg, and J. N. Rodrigues, "Benchmarking of Standard-Cell Based Memories in the Sub-V_T Domain in 65-nm CMOS Technology," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 1, no. 2, pp. 173–182, June 2011.

[16] D. Bortolotti, A. Bartolini, C. Weis, D. Rossi, and L. Benini, "Hybrid memory architecture for voltage scaling in ultra-low power multi-core biomedical processors," in *Design, Automation and Test in Europe Conference and Exhibition (DATE), 2014*, March 2014, pp. 1–6.

[17] D. Rossi, A. Pullini, I. Loi, M. Gautschi, F. K. Gurkaynak, A. Teman, J. Constantin, A. Burg, I. M. Panades, E. Beign, F. Clermidy, F. Abouzeid, P. Flatresse, and L. Benini, "193 MOPS/mW 162 MOPS, 0.32V to 1.15V Voltage Range Multi-Core Accelerator for Energy-Efficient Parallel and Sequential Digital Processing," in *Cool Chips XIX*, 2016.

[18] G. Tagliavini, D. Rossi, A. Marongiu, and L. Benini, "Synergistic Architecture and Programming Model Support for Approximate Micropower Computing," in *VLSI (ISVLSI), 2015 IEEE Computer Society Annual Symposium on*, July 2015, pp. 280–285.

[19] V. Suhendra, T. Mitra, A. Roychoudhury, and T. Chen, "WCET centric data allocation to scratchpad memory," in *26th IEEE International Real-Time Systems Symposium*, Dec 2005, pp. 10 pp.–232.

[20] J. Hu, C. J. Xue, Q. Zhuge, W. C. Tseng, and E. H. M. Sha, "Towards energy efficient hybrid on-chip Scratch Pad Memory with non-volatile memory," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, March 2011, pp. 1–6.

[21] M. Gautschi, M. Schaffner, F. K. Grkaynak, and L. Benini, "A 65nm CMOS 6.4-to-29.2pJ/FLOP@0.8V shared logarithmic floating point unit for acceleration of nonlinear function kernels in a tightly coupled processor cluster," in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, Jan 2016, pp. 82–83.

[22] A. Mineo, M. Palesi, G. Ascia, P. Pande, and V. Catania, "On-Chip Communication Energy Reduction through Reliability Aware Adaptive Voltage Swing Scaling," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. PP, no. 99, pp. 1–1, 2016.

[23] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture support for disciplined approximate programming," in *ACM SIGPLAN Notices*, vol. 47, no. 4. ACM, 2012, pp. 301–312.

[24] 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*, vol. 15, no. 1-2, pp. 177–200, 1997.

[25] J. T. Ludwig, S. H. Nawab, and A. P. Chandrakasan, "Low-power digital filtering using approximate processing," *IEEE Journal of Solid-State Circuits*, vol. 31, no. 3, pp. 395–400, Mar 1996.

[26] R. Hegde and N. R. Shanbhag, "Energy-efficient signal processing via algorithmic noise-tolerance," in *International Symposium on Low Power Electronics and Design*, Aug 1999, pp. 30–35.

[27] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate Computing: A Survey," *IEEE Design Test*, vol. 33, no. 1, pp. 8–22, Feb 2016.

[28] S. Mittal, "A Survey of Techniques for Approximate Computing," *ACM Comput. Surv.*, vol. 48, no. 4, pp. 62:1–62:33, Mar. 2016.

[29] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-Power Digital Signal Processing Using Approximate Adders," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, no. 1, pp. 124–137, Jan 2013.

[30] A. Rahimi, A. Marongiu, R. K. Gupta, and L. Benini, "A variability-aware OpenMP environment for efficient execution of accuracy-configurable computation on shared-FPU processor clusters," in *Hardware/Software Codesign and System Synthesis (CODES+ISSS), 2013 International Conference on*, Sept 2013, pp. 1–10.

[31] V. K. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. T. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 555–560.

[32] 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*, ser. ASPLOS X. ACM, 2002, pp. 123–132.

[33] 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 Tech. Reports*, 2009.

[34] J. Ansel, Y. L. Wong, C. Chan, M. Olszewski, A. Edelman, and S. Amarasinghe, “Language and compiler support for auto-tuning variable-accuracy algorithms,” in *9th IEEE/ACM International Symposium on Code Generation and Optimization*, April 2011, pp. 85–96.

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

[36] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, “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*, ser. PLDI ’11. ACM, 2011, pp. 164–174.

[37] 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*, vol. 21, no. 2, pp. 101–112, Feb 2011.

[38] 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*, vol. 19, no. 1, pp. 161–165, Jan 2011.

[39] N. Verma and A. P. Chandrakasan, “A 256 kb 65 nm 8T Subthreshold SRAM Employing Sense-Amplifier Redundancy,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp. 141–149, Jan 2008.

[40] F. Frustaci, M. Khayatzadeh, D. Blaauw, D. Sylvester, and M. Alioto, “SRAM for Error-Tolerant Applications With Dynamic Energy-Quality Management in 28 nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 50, no. 5, pp. 1310–1323, May 2015.

[41] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, “Analysis and characterization of inherent application resilience for approximate computing,” in *Design Automation Conference (DAC), 2013 50th ACM/EDAC/IEEE*, May 2013, pp. 1–9.

[42] P. Roy, R. Ray, C. Wang, and W. F. Wong, “ASAC: Automatic Sensitivity Analysis for Approximate Computing,” in *Proceedings of the 2014 SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems*, ser. LCTES ’14. ACM, 2014, pp. 95–104.

[43] B. Li, P. Gu, Y. Shan, Y. Wang, Y. Chen, and H. Yang, “RRAM-Based Analog Approximate Computing,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 12, pp. 1905–1917, Dec 2015.

[44] B. Li, Y. Shan, M. Hu, Y. Wang, Y. Chen, and H. Yang, “Memristor-based approximated computation,” in *Proceedings of the 2013 International Symposium on Low Power Electronics and Design*, ser. ISLPED ’13. IEEE Press, 2013, pp. 242–247.

[45] N. Planes, O. Weber, V. Barral, S. Haendler, D. Noblet, D. Croain, M. Bocat, P. O. Sassoulas, X. Federspiel, A. Cros, A. Bajolet, E. Richard, B. Dumont, P. Perreau, D. Petit, D. Golanski, C. Fenouillet-Branger, N. Guillot, M. Rafik, V. Huard, S. Puget, X. Montagner, M. A. Jaud, O. Rozeau, O. Saxod, F. Wacquant, F. Monsieur, D. Barge, L. Pinzelli, M. Mellier, F. Boeuf, F. Arnaud, and M. Haond, “28nm FDSOI technology platform for high-speed low-voltage digital applications,” in *VLSI Technology (VLSIT), 2012 Symposium on*, June 2012, pp. 133–134.

[46] “OpenRISC project,” http://www.opencores.org/or1k/.

[47] D. Rossi, I. Loi, G. Haugou, and L. Benini, “Ultra-low-latency light-weight DMA for tightly coupled multi-core clusters,” in *Proceedings of the 11th ACM Conference on Computing Frontiers*, 2014, p. 15.

[48] P. Meinerzhagen, S. M. 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*, vol. 1, no. 2, pp. 173–182, June 2011.

[49] L. Leem, H. Cho, J. Bau, Q. A. Jacobson, and S. Mitra, “ERSA: Error Resilient System Architecture for probabilistic applications,” in *Design, Automation Test in Europe Conference Exhibition (DATE), 2010*, March 2010, pp. 1560–1565.

[50] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, “Flikker: saving DRAM refresh-power through critical data partitioning,” *SIGPLAN Not.*, vol. 46, no. 3, pp. 213–224, Mar. 2011.

[51] V. Wong and M. Horowitz, “Soft error resilience of probabilistic inference applications,” in *In Proceedings of the Workshop on System Effects of Logic Soft Errors*. SELSE, 2006.

[52] “OpenMP 4.0 Specification,” http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf.

[53] S. Steinke, L. Wehmeyer, B.-S. Lee, and P. Marwedel, “Assigning program and data objects to scratchpad for energy reduction,” in *Design, Automation and Test in Europe Conference and Exhibition, 2002. Proceedings*, 2002, pp. 409–415.

[54] “OpenCL 2.1 Specification,” https://www.khronos.org/registry/cl/specs/opencl-2.1.pdf.

[55] 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 *Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2013 IEEE 27th International*, May 2013, pp. 2182–2187.

[56] “clang: a C language family frontend for LLVM ,” http://clang.llvm.org/.

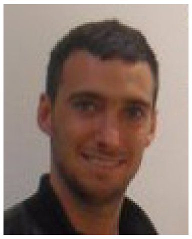
[57] C. Lattner and V. Adve, “LLVM: A Compilation Framework for Lifelong Program Analysis & Transformation,” in *Proceedings of the International Symposium on Code Generation and Optimization*. IEEE Computer Society, 2004, pp. 75–86.

[58] A. Marongiu and L. Benini, “An OpenMP Compiler for Efficient Use of Distributed Scratchpad Memory in MPSoCs,” *IEEE Transactions on Computers*, vol. 61, no. 2, pp. 222–236, Feb 2012.

[59] B. Grigorian and G. Reinman, “Dynamically adaptive and reliable approximate computing using light-weight error analysis,” in *Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on*, July 2014, pp. 248–255.

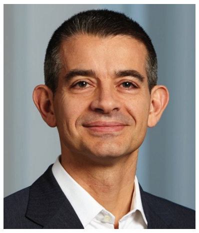
[60] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, “Rumba: An Online Quality Management System for Approximate Computing,” in *Proceedings of the 42Nd Annual International Symposium on Computer Architecture*, ser. ISCA ’15. ACM, 2015, pp. 554–566.

Giuseppe Tagliavini received his MS degree in Computer Engineering from the University of Bologna, Italy, in 2010. He is currently a PhD student at the Department of Electrical, Electronic and Information Engineering (DEI) at the University of Bologna. His research interests include programming models and run-time optimization for many-core embedded accelerators, software design for high-performance embedded systems and compiler support for emerging computing architectures.



Davide Rossi received the PhD degree from the University of Bologna, Italy, in 2012. He currently holds an Assistant Professor position at the Department of Electrical, Electronic and Information Engineering (DEI) at the University of Bologna. His research interests focus on energy efficient digital architectures in the domain of heterogeneous and reconfigurable multi and many-core systems on a chip. He has published more than 30 paper in peer-reviewed international journals and conferences.

Andrea Marongiu received the PhD degree in Electronic Engineering from the University of Bologna, Italy, in 2010. He currently is a postdoc researcher at the Swiss Federal Institute of Technology, Zurich (ETHZ). He also holds a postdoc position at the University of Bologna. His research interests concern parallel programming model and architecture design in the single-chip multiprocessors domain, with special emphasis on compilation for heterogeneous architectures, efficient usage of on-chip memory hierarchies and SoC virtualization. He has published more than 50 papers in peer-reviewed international journals and conferences.



Luca Benini is the chair of Digital Circuits and Systems at ETHZ and a Full Professor at the University of Bologna.He has served as Chief Architect for the Platform2012/STHORM project in STMicroelectronics, Grenoble. He has held visiting and consulting researcher positions at EPFL, IMEC, HewlettPackard Laboratories, Stanford University. Dr. Beninis research interests are in energy-efficient system design and Multi-core SoC design. He is also active in the area of energy-efficient smart sensors and sensor networks for biomedical and ambient intelligence applications. He has published more than 700 papers in peer-reviewed international journals and conferences, four books and several book chapters. He is a fellow of the IEEE and a member of the Academia Europaea.