Hybrid CMOS/Magnetic Process Design Kit and application to the design of reliable and low-power non-volatile logic circuits

G. Prenat · B. Dieny · G. Di Pendina · K. Torki

Abstract Spintronics is a continuously expending area of research and development at the merge between magnetism and electronics. It aims at taking advantage of the quantum characteristic of the electrons, i.e. its spin, to create new functionalities and new devices. Spintronic devices typically rely on the Magneto Resistive (MR) effects, which consists in a dependence of the electrical resistance upon the magnetic configuration. These devices can be used to conceive innovative non-volatile memories, high-performance logic circuits, RF oscillators or field/current sensors. We present here how these hybrid CMOS/Magnetic logic circuits can help circumventing some of the limits of CMOS-only microelectronics in terms of reliability or power consumption as well as the tools required to design such circuits.

Keywords Low power · Reliable · Spintronics · Magnetic Tunnel Junctions

1 Introduction

Magnetic Tunnel Junctions (MTJs) are well known as the basic elements of Magnetic Random Access Memories (MRAM, [1]), a new kind of non-volatile memory which combines high writing/reading speed, low-power consumption, density, hardness to radiations and endurance. Although their intrinsic non-volatility naturally encourage to use them in memories, these unique set of performance allows to use them to intrinsically mix memory and logic parts of the circuits to push forward the limits of CMOS microelectronics and allow new functionalities. To evaluate the benefits of such hybrid circuits in terms of reliability or power consumption, it is necessary to be able to integrate these new devices in classical full custom or digital design flows. We will first present the hybrid Magnetic/CMOS technology. Then, we will describe how mixing CMOS and magnetic devices can result in innovative and high-performance non-volatile logic circuits. At the end, we will present a full Magnetic Process Design Kit (MPDK) to design such circuits.

2 Device and technology

MTJs (Fig.1) are nanostructures composed of two FerroMagnetic (FM) layers separated by an insulating layer. The magnetization of one layer, called Reference Layer (RL) is pinned in a fixed direction and acts as a reference, while the magnetization of the second one, called Storage Layer (SL) can typically be switched between two stable directions, Parallel (P) or AntiParallel (AP) to the magnetization of the RL. According to the relative orientations of the magnetizations, the electric resistance of the MTJ changes, with an hysteresis behavior. Today, most MRAM technologies are based on 1 MTJ-1T cell arrays (1 MTJ to store information and 1 selection transistor, [2],[3]). Reading the stored data consists in measuring the resistance of the device. Writing the MTJ can be performed in different manners, using either an external magnetic field or a spin-polarized current. The first MRAM generation is called FIMS for Field Induced Magnetic Switching. In this approach, the MTJ is written by an external magnetic field, generated by two current lines close to the MTJ to select the row and column in a memory array (Fig.2(a)): only the addressed MTJ sees a field strong enough to switch its magnetization. This technique suffers from selectivity issues: indeed, when the density increases,
the field generated to write a MTJ can accidentally write the neighbour cells resulting in writing errors. This issue was partially solved by the so-called “toggle” writing scheme from Freescale, but it remains hardly scalable below 90nm because the value of the magnetic field required does not efficiently scale with the size of the device. Another improvement was proposed in [4] and is currently used by Crocus-technology. This approach is called TAS for Thermally Assisted Switching (Fig.2(b)). It relies on the strong dependance of the magnetic and transport parameters upon the temperature. The principle consists in heating the junction by Joule effect using a pulse of current through the stack before applying the magnetic field. Above a blocking temperature, the magnetization of the soft layer is released and can be switched at lower field. The selectivity issue is solved since only the heated junction can be written and the scalability is improved since the heating current scales with the surface of the MTJ. The last generation (Fig.2(c)) relies on the Spin Transfer Torque effect (STT) [5]: when a current flows through a FM layer called polarizer, the electrons get polarized. On the other hand, a spin-polarized current exerxes a torque on the magnetization of a FM material so that, if the current is strong enough, the magnetization can be switched. It is then possible to write the SL of a MTJ using this effect, with the RL acting as a polarizer (Fig.3). This technique offers a high density (no current lines are required to generate a magnetic field anymore) and is totally scalable.

The hybrid technology presented here has been developed in the framework of French national projects.

The magnetic devices are TAS MTJs and are fabricated at CEA-LETI and CROCUS-Technology, in post-process above the STMicroelectronics 130nm CMOS process. Fig.4 shows a cross-section of the technology: it integrates the whole standard CMOS process, the magnetic layers with the MTJ itself, top and bottom electrodes plus vias to connect the MTJ to the CMOS layers. An interesting feature of this post process is its full compatibility with any standard CMOS process.

3 Hybrid CMOS/Magnetic logic circuits

Due to its performance in terms of speed and density, MRAM in its 1MTJ-1T implementation could advantageously replace parts of the memory hierarchy. As a result, the architecture of processors could be redesigned to take advantage of MTJs to improve performance, in particular in terms of power consumption and related heating issues. However, the operating frequency of Flip-Flops or low-level cache is larger than 1GHz. Classical MRAM cannot reach such speeds. Hopefully, beside classical 1T-1MTJ structure, non-volatile SRAM has been proposed, called Black and Das cell ([6]). This structure looks like a classical SRAM latch and operates at the same speed in conventional use, but with an additional pair of MTJs in differential mode. The active value of the latch can be stored at any moment in the MTJs and restored in a few hundreds of ps (in the restoration phase, the latch operates as a sense amplifier to read the resistance of the MTJs). Combined with
a classical SRAM latch, this cell can be used to create a Non-Volatile Flip-Flop (NVFF, [7], [8]). Non-volatile SRAM latches could be used to combine the advantages of the SRAM and Flash-based FPGAs to obtain a really fast non-volatile FPGA. In such a FPGA, the configuration can be stored once in a non-volatile manner so that it can be loaded (or reloaded) in less than 1ns at startup or at any time in case of failure ([8]). This use of the non-volatility also allows new functionalities, like shadowed reconfiguration and improve reliability. Using non-volatile registers allows easing the power gating techniques: indeed, the data can be very quickly stored in the non-volatile parts and the power supply totally switched off. On demand, the data can be immediately reloaded and the circuit restarts with full performance and without additional operations. This has given birth to the concept of normally-off electronics ([9]). Beyond the use of MTJ as non-volatile memory elements in electronic circuits, the advantages of MRAM technology especially in terms of writing speed, cyclability and their ability to be manufactured in back-end magnetic technology above CMOS elements also allow to introduce memorization capabilities in the logic part of the circuit itself. This concept is often referred as logic-in-memory ([10]). In this framework, it is possible to develop components composed of a logic function in CMOS technology right on the Si substrate and a memory capability built above the logic function in back-end magnetic technology, the two functions being interconnected by vertical vias. In addition to reduce the standby power consumption, this concept allows multiplying and shortening interconnections between logic and memory, reducing footprint, improving speed, simplifying interconnections and reducing dynamic power consumption.

4 Hybrid CMOS/magnetic Process Design Kit

Designing hybrid architectures embedding CMOS and magnetic devices requires integrating the magnetic elements in standard design suites of microelectronics. It is therefore necessary to provide a PDK compatible with standard design suites to be able to conceive hybrid circuits containing magnetic devices. A compact model was developed at SPINTEC ([11]), valid for all MTJ writing schemes, written in C language, compiled for SPECTRE simulator of Cadence. It takes into account the static and dynamic behaviors of the device, including the temperature dependance of the parameters and the stochastic effects due to thermal fluctuations. Fig.5 shows simulation results for a Spin Transfer Torque (STT, [5]) writing scheme in which writing the MTJ is performed by applying a current directly through the MTJ. To draw the layout of hybrid circuits, a pCell of the MTJ has been proposed. The technology parameters corresponding to the magnetic back-end have been integrated to the technology files to allow physical verifications: Design Rules Checking (DRC), extraction and Layout Versus Schematic (LVS).

To allow digital design of complex circuits, a non-volatile latch has been designed [12], based on the the loadless 4 transistors SRAM cell [13] and Black and Das cell [6]. This cell is represented in Fig.6: it is based on a classical loadless 4T SRAM with two additional MTJs operating in differential mode: one is in P state while the other one is in AP state. Two informations are stored in this cell: an electrical information in the latch and a magnetic information in the magnetic state of the MTJs. In standard use, the cell operates as a standard latch. In retention mode, the bit lines $bl$ and $\bar{bl}$ are connected to the supply voltage and the PMOS transistors in off mode act as charge transistors to maintain the logic levels in $q$ and $\bar{q}$ by their leakage currents. To ensure good voltage levels for $q$ and $\bar{q}$, the leakage currents of PMOS should be bigger than those of NMOS. PMOS are typically high-speed transistors and NMOS low-leakage transistors. In writing mode, the bit lines are connected to the value to be written and the PMOS transistors are turned on to be used as access transistors. The presence of additional resistances in the branches due to the MTJs does not affect these operations. In order to transfer the data stored in
the MTJs into the latch, the bit lines are precharged to the supply voltage by turning on the PMOS transistors. In this configuration, due to the difference of resistances of the MTJs, the voltages at \( q \) and \( \bar{q} \) are slightly different. Then, these transistors are turned off and the latch reaches a stable state corresponding to the value stored in the MTJs. To write the MTJs in TAS writing scheme, it is necessary to send a current through them, then apply a magnetic field. The differential pair is written here in two steps. To write MTJ1, transistor P2 is first turned on by the word line WL2 to precharge \( \bar{q} \) to \( V_{dd} \). This turns on N1. P1 is then turned on by WL1 to generate the heating current through MTJ1. When the blocking temperature is reached, the magnetic field is applied using a circuit not represented in the figure. This operations is repeated for MTJ2, with an opposite magnetic field to write the complementary value. This cell can operate as a standard latch, at CMOS speed. At any moment, it is possible to back-up the data in the magnetic part and restore it in a few hundreds of ps. This cell was designed and fully characterized at device level, using our compact model of the MTJ. A behavioral model of the latch was proposed with the libraries describing the timing of the cell. This cell can operate as a standard latch, at CMOS speed.

5 Conclusions

Hybrid CMOS/Magnetic circuits appear to be good candidates to circumvent the limits of CMOS electronics. First realizations have been proposed with very encouraging results. Moreover, the writing scheme based on STT is very scalable, since the switching of the magnetic state is driven by the current density. The advantages of mixing MTJs with CMOS devices should become more and more obvious with advanced technologies. Of course, this new technology can be used together with other design techniques to improve the performance of electronic circuits, especially in terms of reliability. Some research has to be carried out to see how these different techniques can be combined. In particular, some of these techniques are efficient at the full system level. It is then necessary to evaluate the opportunities provided by this technology for complete systems and in comparison or combination with existing techniques and for advanced technologies.

References

Towards the Design of Tunable Dependable Systems

C. Bolchini · M. Carminati · A. Miele

Abstract In this paper we introduce a novel methodology for the design of systems with a "tunable" level of reliability, balanced with respect to performance penalties and costs, adaptable at run-time on explicit demand of the system or, implicitly, according to the execution context. A dynamic scheduling system for many/multi-core architectures is presented as a case study.

Keywords Tunable Fault Tolerance · Reliability-driven Design · Many/multi-core Architectures

1 Introduction

The trend of building new complex systems by integrating low-cost, inherently unreliable Commercial Off-The-Shelf (COTS) components is one of today’s challenges in the design, analysis and development of systems exhibiting a certain degree of dependability. In fact, the last decade has seen the complexity of electronic systems growing faster and faster, thanks to the decrease of the components size and cost. However, the use of low-cost execution resources to achieve high performance makes modern electronic devices more and more unreliable.

At the same time, the increasing number of faults is mainly due to the shrinking of components itself, to the variations in the manufacturing process and to the expositions of devices to radiations and noise fluctuations. Radiations, in particular, are the cause of transient faults and they also accelerate the aging process electronic devices are commonly subject to. Radiations are particularly frequent in space, but are becoming an issue also at ground level [8]. Therefore, when adopting the COTS-based design approach for the realization of modern electronic systems, also considering their pervasiveness, reliability has become one of the main goals, together with performance. A third aspect that emerged to be crucial in the design phase is power consumption, due to the general awareness of the need for electronic devices to be less energy-hungry. The performance × reliability × power solution space is to be explored while designing new circuits, to find an acceptable balance between the three aspects or to favor one rather than another, according to the designer’s needs.

The work described in this paper aims at introducing a new methodology which takes into consideration both performance and reliability in designing new systems, while leaving the power issue as a future development. The goal is to introduce our vision and support its validity, by applying the approach to a case study at the methodological level. The adopted case study is the one discussed in [3], where the authors proposed a reliable dynamic scheduler, describing a unique hardening technique, but focusing their attention on a wider sphere of issues, including diagnosis, components aging and more. The present work’s goal is, instead, to introduce a new methodology for varying the reliable technique implemented by the scheduler, evaluating the performance/reliability trade-off at run-time and not considering, at the moment, other issues.

The remainder of this paper is organized as follows: next Section 2 will introduce and describe the envisioned methodology from an high abstraction level, pointing out its principal and innovative aspects. In order to investigate the application of such a methodology to a real system such as a many/multi-core system, Section 3 will describe the details of the architecture, programming, and fault model and the design of a tunable fault mitigation layer to enhance an oper-
ating system with reliability-aware scheduling capabilities. A brief description of the related work is given in Section 4, while Section 5 wraps up the authors’ conclusions, presenting future developments.

2 Tunable reliability

The approach we propose aims at enhancing a given system with varying or, better, tuning reliability features at run-time, i.e. dynamically. In this scenario, the main challenge is the design and evaluation of policies for delivering a tunable reliability, triggered either on-demand or autonomously, in a self-adaptive fashion. This “tuning” action is to be achieved by envisioning several hardening techniques, characterized by different benefits/costs aspects and working at different abstraction levels. More precisely, by selecting the proper technique at a given time instant, it is possible to adjust the performance/reliability trade-off, according to the user’s modified requirements or to environment changes. To enable the selection of the right technique, suitable metrics, able to comprehensively describe the characteristics of each strategy in terms of both performance and reliability, must be defined. The value of these metrics has to be periodically checked and compared with the desired goals, supporting the techniques selection process.

Evaluation Metrics. One of the main issues the methodology needs to tackle is the identification or definition of metrics allowing to evaluate how much a solution is suitable in terms of performance and reliability. When referring to performance, it is possible to find in literature metrics that proved to be effective in quantifying the computation capability of a system:

- **throughput**: defined as the average quantity of data processed by the system in a time unit;
- **turnaround time**: refers to the average time elapsed between the instant in which the input task is provided to the system and the instant in which its output is ready.

When evaluating reliability, it is quite difficult to find widely-adopted indexes; at present we take into account:

- **coverage**: standard reliable techniques allow to guarantee a 100% coverage of the system at high resource or time overhead. It would be interesting to define new techniques able to provide an incomplete, though known a-priori, fault coverage, trading resources or time for reliability. Indeed, the quantitative evaluation of the achieved fault coverage poses several challenges, especially when working at system/application level. Usually such metrics is evaluated a-posteriori, once the system is implemented. At the best of authors’ knowledge, no assessed and universally recognized methods are available at such an high abstraction level;
- **fault detection latency**: it quantifies the time elapsed between the instant the error (the observable effect of a fault) occurs and the instant it is detected;
- **mean time to failure (MTTF)**: it is the average time before a component fails in a permanent way; it represents an estimation of a component’s life time. In case of reparable systems, MTTF is replaced by MTBF (**mean time between failures**).

**Fields of application.** A methodology such as the one we envision is suited for application scenarios where the dependability requirements need to be enforced only in specific working conditions (e.g., when an emergency situation arises) and/or for limited time windows. In these cases, the system can adapt its behavior and expose robust properties with possibly limited performance, or the other way around. To enable this behavior, the methodology can act both on the hardware-level (e.g., FPGA-based systems, on which dynamic partial reconfiguration can be exploited to vary the required reliability of an hardware component) and on the software-level (as in a multi-many core scenario proposed as a case-study in Section 3), based on i) how hardening techniques can be applied and ii) how the tuning of the hardening is managed. In this paper, the application scenario refers to many-core architectures, given the availability of several processing resources and the opportunity to balance such computational power to achieve high performance and/or high reliability.

3 A many-core architecture with tunable reliability/performance characteristics

We briefly introduce the adopted models, more details are available in [3].

**Reference model.** The adopted scenario refers to many-core computing fabric, highly modular and configurable. The fabric contains multiple processing units and a hardware fault tolerant controller, devoted to the communications with the rest of the environment and to the booting of the applications. Moreover, there is a module featuring basic mechanisms for managing application parallelism (thread creation and synchronization) exploited by a minimal operating system running on the architecture and responsible for coordinating applications’ execution. This architectural model is implemented by some real examples, such as each cluster in the ST/CEA P2012 [10] or in the Teraflux [11]. Nevertheless, the presented approach is designed to be as general as possible, not limited by architectural requirements.

The **programming model** commonly adopted for implementing parallel multi-threaded applications for multi-core
Towards the Design of Tunable Dependable Systems

F
E E
J
Thread 0
Thread 1 Thread 2
Thread 0

Fig. 1 A simple application task-graph.

architectures is the so-called fork-join model, and a widely-used paradigm that implements such a model is OpenMP [1]. The smallest execution unit is a task and applications are usually represented by means of a direct acyclic graph (DAG) consisting of tasks and, thus, called task-graphs. Figure 1 shows an example of a simple application represented through a fork-join task-graph. Four different types of nodes exist, classified according to the topology of the graph and the interaction with the underlying operating system, without considering the specific functionality implemented by the node itself.

- **Elaboration** node. It begins with the process/thread execution start and finishes with the process/thread execution end. Intuitively, it is represented as a node with a unique input arrow and a unique output arrow (E nodes in Figure 1).
- **Fork** node. This node is delimited by the process/thread execution start and a system call requesting the operating system to create a given number of new threads (e.g., fork() or thread_create()). F in Figure 1 is an example fork node, having a single incoming arrow and multiple output arrows.
- **Join** node. Symmetrically to the previous node, the join node starts with an instruction that causes the execution to pause, waiting for termination of other threads (typically a wait() or thread_join()); node J is an example. Multiple arrows are expected to enter this node, while only one exits from it.
- **Join_Fork** node. It is delimited by join() and fork() instructions. In this case, multiple arrows both enter and exit the node, commonly referred to as FJ node; it corresponds to a J-E-F chain of nodes.

It is worth noting, for the sake of completeness, that a fork node always generates children tasks belonging to different threads, while the join node is part of the same thread the corresponding fork node belongs to. Hence, in our example, node F and J belong both to the same master thread, which generates the two children threads containing E nodes.

The fault model adopted for this project is known as single failure model. A fault can affect one processing unit at a time: the faulty unit can exhibit an incorrect behavior transiently or permanently. Subsequent transient faults in different units are managed, assuming that between two consecutive fault events there is enough time to allow the first erroneous condition to be detected. Faults can have different effects: they can lead the computation to an erroneous result, force the task to end in an unexpected way or even be the cause of a non-termination. In the first case, the fault is propagated to the children tasks as wrong inputs and can be discovered by using checker or voter tasks to compare the tasks’ output. In the last two cases, a silent fault occurs and children tasks are not even created and scheduled: to detect this kind of faults the use of watchdogs and time-outs is needed.

**Fault Management Layer.** To define a reliable system for the envisioned scenario two steps are needed: i) identify/define suitable mitigation strategies and their characteristics according to the defined metrics; ii) investigate how the OS is to be modified to support the introduction of the layer in charge of adopting the hardening techniques (Fault Management layer), which will also support the autonomous tuning. This new layer is located between the applications and the OS, straddling the user mode and the super-user mode. In the present scenario, the FM layer is in charge of providing reliability at a software level, i.e., by implementing a reliability-aware dynamic task scheduler. For this kind of action, a well known practice is to create replicas of the tasks and then compare their execution results to detect possible malfunctioning (fault detection – FD) and mask them (fault tolerance – FT). Triple Modular Redundancy (TMR) is a common mitigation technique that provides fault tolerance capabilities, by creating three replicas of the same task and voting their results. With respect to the adopted fault model, the correctness of the result is guaranteed and it also possible, in some situations, to detect which unit is faulty, allowing for diagnosis to take place.

A simple example of “tunability” of the mitigation policies is the granularity at which the TMR technique is applied. Each level (application, thread, or task, as intuitively shown in Figure 3) implies different values of performance...
and reliability metrics: from different overheads in terms of execution time or needed resources, to the possibility or not to execute diagnosis and to the varying fault detection delay (Table 1 reports the values for the adopted metrics). As already sketched, three are the granularities at which the FM layer can decide to apply the TMR technique:

- **Application.** At this level only the output of the JOIN nodes is voted. The application of this technique adds the lowest time overhead, while implying the highest latency in case of fault detection and not allowing to identify which is the faulty task, thus excluding the opportunity to perform diagnosis.
- **Thread.** This is the intermediate granularity at which TMR can be applied. It represents a trade-off between the other ones, having an intermediate time overhead (both JOIN and ELABORATION nodes are voted) but also having an intermediate fault detection latency. Also in this case, not enough information is available for diagnosis to be performed.
- **Task.** Task is the finest available granularity, since a voter task is added for each node. It guarantees the minimum latency in case a fault is detected and allows diagnosis to be performed. On the other hand, it adds to the execution the maximum time overhead.

With reference to the execution times listed in Table 1, $t_{app}$ represents the overall execution time for the application if no faults occur, considering as best case the one where all the replicas (and all the their tasks), if possible, are executed perfectly in parallel. Moreover, $n_X$ is the number of nodes of type $X$, where $X \in \{F, E, J, FJ\}$, and $t_V$ is the execution time for the voter tasks (assuming legitimately that all the voters will take the same time to compare the three replicas’ results). By switching the level TMR is performed at, the dynamic scheduler can favor performance or reliability. The **Fault Detection Latency** column refers to the amount of execution (in terms of tasks) that must be carried out for the error to become detectable: namely the nearest voting node. All the quantities in Table 1 are computed, as an example, on the simple task-graph of Figure 1.

The FM layer is designed to be completely transparent, with respect to both applications (executed in user mode) and the operating system (running in super-user mode). This transparency is achieved by wrapping (some of) the system calls to the operating system; in particular:

- **init:** before allowing the OS to actually execute the process, the FM layer, according to the reliability technique selected for the current process, must create replica(s) of the code and of the relevant data (input data, shared memory locations, . . . ), and set up the environment for the replica(s) execution (refer to [9] for further details).
- **fork:** this system call corresponds to the end of a task, thus the chosen reliability policy could request a check to be executed. If this is the case, the FM layer verifies that all the replicas of a process have reached the same *fork* instruction and instantiates a voter task to verify the correctness of the computation up to that point. It then acts according to the result of the voter task and eventually allows the system calls of the process replicas to reach the OS and be actually executed.
- **join:** when a thread ends its execution, again, if replicas exist, a comparison of the output could be necessary, similarly to the *fork* case.
- **exit:** this instruction represents the beginning of a new node. The FM layer has to guarantee that this instruction, before being issued, will wait not only for the children to be executed, but also for possible subsequent voter tasks to be completed.

The FM layer is in charge of selecting the most appropriate hardening technique (in the example, the granularity for the application the TMR) to meet the fault detection latency vs. performance goals, which may change over time, during the operational life of the system. While in the past other approaches focused on the exploration with respect to the use of different hardening techniques considering performance penalties and costs, the aim of our proposal is to enable this adaptability at run-time, thus being able to adopt different hardening solutions based on the current state of the system and the environment (the overall context).

4 Related Work

The definition of a methodology for adapting the reliability × performance trade-off is a quite original topic in literature and no relevant work at present exists.
On the other hand, when considering the implementation of a dynamic scheduler for a multi-many core architecture, a wider literature is to be considered. A clear overview of the problem and of the developed solutions is given in [5], in which approaches based on redundant execution to achieve fault detection/tolerance are discussed. Code replication and checkpointing are other widely applied techniques ([7, 6, 2]), but they require architectural support to manage the replicated threads and the comparison/voting activity. Moreover, they usually consider only single threaded applications.

Our proposal allows a system to execute applications in a fault tolerant fashion, or not; to dynamically vary the scheduling algorithm to favor performance instead of reliability or vice-versa, based on the user’s requirements. Similar features are offered by the works proposed in [12] or in [4]. The former approach is mainly based on an architectural solution consisting of loosely-coupled processors, used to execute applications with reliability requirements. The rest of the applications are dispatched to the remaining cores, to limit performance penalty. The latter project, TriThread, is an application-level approach where OpenMP programs are hardened at design time, and the replicated code is dynamically scheduled on the different cores, with no specific heuristics. In both the cited cases the hardening techniques are applied at a fixed granularity, which may result too fine or too coarse according to the considered scenario.

Table 1  TMR mitigation strategies description and classification.

<table>
<thead>
<tr>
<th>Granularity</th>
<th>Execution time – Best Case</th>
<th>Overhead</th>
<th>Fault Detection Latency</th>
<th>Mitigation Strategy</th>
<th>Diagnosis</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application</td>
<td>$t_{APP} + n_f \times t_v$</td>
<td>$t_v$</td>
<td></td>
<td>Fault Tolerance</td>
<td>No</td>
</tr>
<tr>
<td>Thread</td>
<td>$t_{APP} + (n_I + n_E) \times t_v$</td>
<td>$3 \times t_v$</td>
<td></td>
<td>Fault Tolerance</td>
<td>No</td>
</tr>
<tr>
<td>Task</td>
<td>$t_{APP} + (n_I + n_E + n_F + n_F) \times t_v$</td>
<td>$4 \times t_v$</td>
<td></td>
<td>0</td>
<td>Fault Tolerance</td>
</tr>
</tbody>
</table>

5 Conclusions

In this paper we introduce a new methodological approach to the design of systems with tunable reliability. The idea is to add a fault management layer to the system, able to dynamically adapt the mitigation strategy to achieve desired performance/reliability goals. This is possible only if various hardening techniques are available, with different costs/benefits. In this paper we explore this novel approach with respect to a many-core architecture, enhanced with a dynamic reliable scheduler able to select among different hardening/performance levels.

These ideas are still at an early stage, and present effort is devoted to the refinement of the selected metrics and the strategies to dynamically select the goals to be optimized with respect to the active context.

References