Resilient CMPs with Mixed-grain Reconfigurability

I. Sourdis, D.A. Khan, A. Malek, S. Tzilis
Computer Science and Engineering Department
Chalmers University of Technology, Sweden
Email: {sourdis, adanish, aliradm, tzilis}@chalmers.se
G. Smaragdos, C. Strydis
Neuroscience Dept.
Erasmus Medical Center, The Netherlands
Email: {g.smaragdos, c.strydis}@erasmusmc.nl

Abstract

This Chip Multi-Processor (CMP) design mixes coarse- and fine-grain reconfigurability to increase core availability of safety-critical embedded systems in the presence of hard-errors. We conduct a comprehensive design-space exploration to identify the granularity mixes that maximize CMP fault-tolerance and minimize performance and energy overheads. By adding fine-grain reconfigurable logic to a coarse-grain sparing approach, we can tolerate 3× more hard errors than core redundancy and 1.5× more than any other purely coarse-grain solution.

I. Introduction

Recent semiconductor-technology trends have resulted in the emergence of new reliability challenges in CMP design. Smaller transistors are more sensitive to manufacturing defects, wear-out phenomena and dielectric breakdown, all of which result in permanent faults [1]. Even a small portion of faulty circuitry could render an entire chip unusable, lower production yields or cause failures during the chip lifetime.

The wide use of embedded systems in various domains, such as medical, space and automotive, makes reliability and availability even more pressing issues. In particular, safety-critical systems with low or no accessibility for replacing damaged parts require efficient techniques for online self-repair and fault tolerance to deal with the increasing number of faults in shrinking technology nodes.

A general fault-tolerance strategy suggested in literature is *component sparing*. Future chips can potentially be abundant in spare resources due to dark silicon. Sparing advocates dividing a chip into smaller, redundant substitutable blocks (SBs) that can be statically or dynamically isolated and replaced if damaged, by identical spare parts. Redundancy is then facilitated by reconfiguration, ranging from coarse (e.g. core-level) to finer (e.g. pipeline stages or functional units) and even very-fine (e.g. gate-level) granularity. The granularity chosen introduces a clear tradeoff between availability and efficiency: Finer-grain reconfigurability renders a design more resilient to faults but also makes it slower, more power-hungry and less area-efficient.

II. Background and motivation

High-availability commercial systems such as the IBM z-series [2] and the HP NonStop systems [3] cover permanent faults through double- or triple-modular redundancy (DMR or TMR, respectively) applied statically at the core level. Dynamically coupling cores to apply DMR can improve efficiency [4]. Moreover, disabling [5] and isolating [6] permanently faulty cores can sustain degraded availability.

Finer than core-sized SBs have also been proposed, such as functional units [7] or pipeline stages [8]–[10]. In the latter case, processors – mostly simple RISCs – are modified to support dynamic sparing of stages. Some researchers have opted for simpler designs at the cost of flexibility [8], others offered a higher degree of sparing introducing more radical micro-architectural changes [9], [10].

At even finer granularities, a processor data-path can be protected by adding spare bits [11] or a control-path can use field-programmable control logic to tolerate faulty states [12]. Finally, Field-Programmable Gate Array (FPGA) and Programmable Logic Array (PLA) technologies can potentially avoid hard errors by changing the configuration and/or the placement of a design [13], [14].

Generally speaking, the SB size affects the overheads of a design as well as its flexibility to tolerate faults. Then, how can designers select the granularity that maximizes availability and minimizes performance and power overheads for a given technology and fault density?

Instead of selecting a particular granularity, in this study we propose tackling this problem by exploiting a mixed coarse- and fine-grain reconfigurable fabric. Damaged parts can be replaced offline or at runtime by identical spare parts or by their fine-grain reconfigurable, functional equivalent. The delay incurred due to this reconfigurability is largely hidden through inserting

Fig. 1. Mixed-grain reconfigurable fabric for tolerating hard-errors. Each component is partitioned in SBs that use pipelined reconfigurable interconnects to allow the blocks to be spared. Fine-grain logic can be used to instantiate any SB.

on demand additional registers in the design. We propose an *adaptive processor* in which extra pipeline stages can be created dynamically. In addition, we perform a comprehensive *design-space exploration* to determine the availability, performance and energy efficiency of different granularity mixes at various fault densities. To the best of our knowledge, this is the first study to combine multigranular reconfigurability and to systematically analyze how availability and its overheads scale for different granularities and fault densities.

III. Mixed-grain reconfigurable fabric

Components — in our case CMP cores — can be implemented on a mixed-grain reconfigurable substrate as follows. Each component is partitioned into SBs with reconfigurable interconnection. In addition, components share fine-grain, FPGA-like, reconfigurable logic that can replace any damaged block lacking identical spares. A faulty SB can, thus, either be replaced by an identical spare block (presumably taken from a neighboring unused/faulty component) or instantiated in the fine-grain reconfigurable fabric. This can be performed offline after manufacture-time testing or online during system lifetime, based on online detection mechanisms.

Figure 1a illustrates the SB partitioning as an array. Each column comprises identical, interchangeable SBs and each row represents a component in its default (fault-free) configuration. The block at the top represents the shared, fine-grain logic. A functioning component can be assembled by connecting an undamaged SB from each column with the fine-grain block acting as a wild card able to replicate any array SB.

The largest SB dictates the minimum size of the fine-grain block, hence it is more efficient to partition a component into blocks of similar size. Using a larger number of smaller blocks requires more wiring but also reduces the area needed for the fine-grain block. This creates an interesting tradeoff for the mixed-grain reconfigurable fabric. As illustrated in Figure 1a, partitioning can be achieved by dividing a component into pipeline stages (vertical partitioning) and/or by splitting (part of) it into concurrent parts (horizontal partitioning).

The wires and multiplexers needed for sparing introduce delays proportional to the distance between the spares. To accommodate these delays we add registers to the wires spanning across components to form extra pipeline stages, as illustrated in Figure 1c. When needed, we also increase the pipelining of the logic in the fine-grain reconfigurable version of a SB.

The advantage of pipelining is that scaling the number of components in the array does *not* affect the operating frequency. In contrast, previous solutions like StageNet [9] and CCA [8] increase the cycle time proportionally to the number of components. Consequently, our approach is more scalable and the fault-free configuration of the components can be significantly faster than the worst-case delay suffered in previous designs that use all-to-all switches. The drawback, however, of pipelining the wires is that variable-depth pipelines are formed in different component configurations. A variable pipeline depth is simpler to achieve in a feed-forward dataflow pipeline than in a control-flow (i.e., von Neumann) architecture. As explained in the next Section, the latter case requires significant changes, such as decoupling the pipeline stages and providing distributed control to allow dynamic insertion of extra stages.

An additional issue is that a SB instantiated in the fine-grain reconfigurable logic operates slower than its ASIC counterpart which – in turn – slows down the entire component[1]. However, since the fine-grain block is probably used by a single component at any point in time, the remaining components can likely be clocked at a higher frequency. We, thus, provide two dynamically selected clocks per SB: the faster clock is used for components exclusively composed of coarse-grain parts and the slower one for components that use fine-grain logic.

Although pipelining and multiple clock domains reduce the impact of reconfigurability on the operating frequency, adding extra stages to a pipeline affects processing throughput. Figure 1c shows that the number of extra stages needed depends on the vertical distance of the connected blocks. Consequently, a good configuration of the component array should maximize the number of working components while, at the same time, minimizing the distances between connected blocks. We use a fast, deterministic greedy algorithm to generate such configurations, achieving 90% or more of the performance of an (exhaustively sought) optimal configuration [15]. Due to its short execution time (a few tens of μsecs), our algorithm allows to apply the proposed reconfiguration approach at runtime.

The reconfigurable interconnects of the adaptive processors are based on existing reconfigurable architectures. Most often, SBs of the same component will be connected with each other; thus, vertical wires will be used infrequently. We, therefore, opt for bidirectional wires and use tri-stated switches for across-components connections. Compared to unidirectional interconnects, this approach halves the wiring but adds some extra delay to the switches.

[1] A component — in our CMP design an entire core — is considered to have a single clock defined by the slowest SB.

Fig. 2. The proposed adaptive processors using the mixed-grain reconfigurable substrate. Each processor is partitioned in decoupled pipeline stages (vertical partitioning), while the execution stage is further split in three concurrent parts (horizontal partitioning) due to its size. When a part of EX is implemented in the fine-grain logic, the entire stage is further pipelined in two stages to improve speed.

IV. A MICRO-ARCHITECTURE WITH ADAPTIVE PIPELINE

For deploying a CMP on the mixed-grain reconfigurable substrate, our design must be able to handle a variable number of decoupled pipeline stages and, subsequently, distributed control. Our processor architecture must be adapted to remain oblivious to the number of (extra) stages in its pipeline; that is, reconfigurability needs to be transparent to the application level. This design requirement is imperative for guaranteeing binary compatibility across different processor configurations, making recompilation of the code unnecessary.

We consider a typical 32-bit, in-order RISC, baseline architecture with five pipeline stages: Instruction Fetch (IF), Decode (DC), Execute (EX), Memory (ME) and Write-Back (WB). It has local instruction- and data-memories (32-KByte each) and a 16-entry register file (RF).

The first design consideration is the partitioning of the processor to SBs. As shown in previous works [8]–[10], pipeline-stage granularity is more convenient and keeps the complexity of the micro-architectural changes tractable. Figure 2 illustrates the partitioning of our cores. The processor is divided at the stage boundaries (vertical partitioning). Horizontal partitioning is also needed when fine-grain logic is used to balance block sizes.

The above design requirements call for several micro-architectural changes in the data flow and control flow of the baseline core. Allowing a variable number of stages per processor and eliminating global control directly influences the hazard resolution in a typical RISC architecture. There are three issues to be addressed:

1) Data-hazard resolution;
2) Control-hazard resolution and pipeline flushing; and
3) Global-stall support.

Next, we discuss each one in more detail and describe the distributed strategies devised for tackling them.

A. Data-hazard resolution

Since we allow a variable pipeline depth, the commit time of instructions in the pipeline is not fixed. Whenever extra stages are inserted after the DEC stage, the number of uncommitted instructions in flight effectively increases. Thus, the instruction window within which a data conflict may occur widens, too.

As a first step towards alleviating this problem, we reduce the number of instructions in the pipeline by removing the WB stage. We incorporate the WB in the reconfigurable interconnects and the bulk of its logic (a few multiplexers) in the DC stage (RF). The functionality of the processor remains intact but, without loss of generality, we can now approach the architecture as a four-stage pipeline.

Generally speaking, detecting and servicing data conflicts with *value bypassing* and *stalling* would be extremely complex. A bypass path would be needed from each new stage to all stages that might require a value. Moreover, for cases where simple bypassing is insufficient, the pipeline needs to be stalled (e.g., due to a memory load). Our bypass mechanism explained next and stalling discussed in Section IV-C are both designed to be agnostic to the number of stages.

We address the value-bypassing problem by storing copies of produced (but uncommitted) results locally at the two stages in which they are produced and consumed (EX and ME) using two FIFO *bypass buffers*. Each buffer stores up to a maximum number of uncommitted results, as dictated by the deepest allowed pipeline[2]. Each buffer entry stores (i) two bits indicating whether the instruction produces a value and where (EX, ME, nowhere), (ii) its destination register, and (iii) the actual value.

For every instruction entering EX or ME, the respective bypass buffer is checked for data conflicts. On a match to any of the buffer entries, the locally buffered value is used. A mismatch means that the value is not available due to being produced elsewhere or being the result of a load instruction that has not yet arrived, and the pipeline must be stalled. To reduce the overheads of such cases, a bypass path is used to forward values from the ME to the EX buffer; this path is also pipelined if extra stages are inserted between EX and ME, so the processor SBs can still be decoupled.

Our bypass mechanism is somewhat similar to StageNet [9] as both store uncommitted results at the stages they are produced, however, in StageNet EX and ME are merged into a single stage. Such partitioning is expected to be unbalanced and, therefore, affect StageNet's fault tolerance, but compared to our approach, it requires only a single bypass buffer and avoids forwarding between ME and EX.

[2]A bypass buffer requires 2N-1 entries to allow sparing of any SB between a cluster of N processors.

B. Control-hazard resolution and pipeline flushing

On branch misprediction, all invalid instructions fetched before branch resolution need to be flushed. This should be supported without global signals in order to avoid long wires. We adopt a scheme similar to that in StageNet [9], yet somewhat simpler due to the different architectures.

We introduce an *Instruction-Flow (InF)* status bit at the IF and EX stages to identify whether an instruction in flight is valid or part of an invalid instruction stream due to a misprediction. More precisely, each instruction travels in the pipeline carrying one additional bit, a *Stream-Identification Bit (SIB)*. Its value is assigned in the IF stage by the InF bit stored there. When an instruction enters the EX stage, its SIB is compared to the InF bit of the stage. If the values match, the instruction is allowed to continue, otherwise it is flushed and makes no changes in the bypass buffers of the EX stage.

To flush the pipeline, the value of the InF bit in the EX stage is inverted. When the correct branch target is loaded, the InF bit of the IF stage is also toggled. As a result, the invalid instructions that were fetched in between are marked with a different SIB than the current value of the EX InF bit and are, therefore, flushed. On the other hand, the subsequent instructions will have the same SIB value as the InF bit of the EX stage and are executed normally. As such, the flush mechanism for a branch misprediction is implemented locally in the EX stage eliminating the need for global flush signals while requiring very little extra circuitry. As opposed to StageNet, which needs to store and check the InF bit at every stage, our processor needs to store it only at the IF and EX stages and check it only in the latter.

C. Global-stall support

As explained above, requiring a distributed control calls, lastly, for a new mechanism to facilitate pipeline stalling. In order to avoid global signals there are two alternatives: (i) per-cycle propagation of the stall signal across stages, as performed in StageNet, or (ii) flush and reload instead of an actual pipeline stall. The first option would drastically increase complexity and require double buffering of all instructions in the pipeline. We, therefore, opted for the second option which is already supported as described above. Flushing has a higher impact on performance than stalling, but it is relatively tractable for short pipelines. Using flush and reload to cope with data hazards entails the danger of not having the data written back at the time of reloading (e.g., while waiting for a load to arrive from memory). In such a case, the conflict is detected again and the pipeline is flushed once more.

D. Further design considerations

When implemented in the fine-grain logic, the EX stage requires substantially more area and has a longer delay than any other SB of our processor. To reduce this imbalance, we further partition it horizontally and vertically, as illustrated at the bottom of Figure 2.

The EX stage is split (horizontally) into three concurrent sub-blocks, each roughly containing a chunk of the ALU. The fine-grain logic can, then, replace a single EX sub-block rather than the entire stage, reducing its area requirements. Still, in coarse-grain sparing, the entire EX stage is considered as a single block and substituted by an identical spare EX stage from another core. Moreover, in order to improve the speed of the EX sub-blocks, we add the option to further pipeline the EX stage (vertically breaking the ALU logic in two) when one of its sub-blocks is implemented in fine-grain logic.

E. Applicability to other architectures

Most of the proposed techniques for mixed-grain reconfigurability are tailored to the above in-order RISC microarchitecture. In the embedded domain, where low power consumption and predictability are important, in-order RISCs are quite commonly used. Then, the proposed microarchitecture could be fully applicable, even to processors with complex functional units and deeper pipelines.

On the other hand, Out-of-Order (OoO) superscalar processors would require different techniques to ensure decoupled pipeline stages. Even so, such architectures decouple fetch and execution of instructions through using instruction queues. Dynamic scheduling of instructions allows OoO processors to tolerate unpredictable delays in the execution of instructions which suits our reconfigurable-design approach well.

V. Evaluation of the adaptive microarchitecture

In this Section we evaluate the performance, power and area costs of our adaptive processor to access the effect of the microarchitectural modifications. The coarse-grain and fine-grain parts of our design (henceforth denoted as CG and FG, respectively) are implemented in 65-nm technology considering STM-65nm and a Xilinx Virtex-5 65-nm FPGA substrate, respectively.

When using only CG parts, our adaptive processor maintains 90% of the baseline frequency, dropping from 500 MHz to 450 MHz. Pipelines also using FG logic (denoted as CG+FG) operate at 40% of the baseline speed (200 MHz), limited by the slowest FG implementation of a SB.

(a) IPS of the adaptive processor vs. the baseline.

(b) Power consumption of the adaptive processor vs. the baseline.

(c) Energy efficiency of the adaptive processor vs. the baseline.

Fig. 3. Performance (IPS), power and energy overheads, compared to the following configurations of a single, adaptive, processor baseline: CG sparing with zero, 2, 5 and 15 extra stages (denoted as CG 0/2/5/15 extra stages), FG sparing of an EX part or a DC stage with no extra stages (denoted as CG+FG(EX) and CG+FG(DC)).

(a) Average number of functioning cores.

(b) Guaranteed availability (Probability of having at least 4 functioning cores).

(c) Average number of Instructions per Second (IPS).

(d) Energy efficiency (IPS/Watt).

Fig. 4. Design-space exploration of coarse-grain and mixed-grain reconfigurable CMPs with various granularities, evaluating availability, performance and energy-efficiency at different fault densities. All designs use area smaller or equal to 9 baseline cores. Fault density is measured as the number of permanent faults per area of a baseline. Granularity is the fraction of a component that constitutes a SB.

The reconfigurable interconnects and the microarchitectural changes contribute evenly to the area overheads of the adaptive processor (CG part, only), which occupies 25% more area than the baseline. The area cost of the FG part is roughly 6× the (ASIC) size of the largest SB it instantiates.

Figure 3 depicts the performance, power consumption and energy-efficiency results of our adaptive processor. Using the EEMBC CoreMark and Telebench 1.1 embedded benchmarks, we evaluate various CG and CG+FG configurations of the processor compared to the baseline.

The fault-free CG configuration maintains 86% of the baseline performance (measured in instructions per second – IPS) and gradually drops to 73%, 63% and 43% when adding 2, 5 and 15 extra pipeline stages, respectively. Implementing a spare DC or EX in the FG logic maintains 39% and 26% of the original performance, respectively, due to the lower frequency. In comparison, CCA maintains 95% of its original performance using clock borrowing [8] but provides limited flexibility in repairing faults. StageNet [9] and Viper [10] optimistically assume that, despite their more complex design modifications, their operating frequency remains intact. Still, StageNet preserves 48% and 83% of its performance (with and without binary compatibility, respectively) and Viper 62.5%; both approaches sustain the above costs even when fault-free.

Our fault-free CG configuration consumes about 1.4× more power than the baseline. The power consumption increases to 1.5×, 1.7× and 2× when 2, 5 and 15 extra stages are inserted, respectively. Surprisingly, when using FG logic for the DC stage, power drops to 83% of the baseline, due to the lower clock rate. In contrast, replacing part of the EX has a power overhead of 50%.

CG configurations maintain 62%, 49%, 37%, and 22% of the baseline energy efficiency (measured in IPS/Watt), for zero, 2, 5 and 15 extra stages, respectively. For a processor that uses FG logic for its DC or part of its EX, energy efficiency drops to 48% and 18%, respectively.

To exemplify the occurrence rate of various configurations, which in turn determines the efficiency of processors: at a fault density of one fault per core, 38%, 60%, and 2%, of the configurations in a CG design have zero, 1-4 and 5-15 extra stages, respectively. For designs with CG+FG substrate these percentages are 30%, 50% and 2%, respectively, and the remaining 18% of processor configurations use the FG logic.

In summary, when FG logic or distant CG spares are used, performance drops to $\frac{1}{2}$ or even to $\frac{1}{4}$, power consumption increases up to 2×, and energy efficiency reduces to $\frac{1}{5}$. However, more efficient configurations maintain over $\frac{4}{5}$ of their performance, have 0.8-1.4× the baseline power consumption, and retain 60% of the baseline energy efficiency. The above constitute a significant cost for providing increased tolerance to permanent faults, however this is a fair price to pay for online self-repairing in safety-critical embedded systems with low accessibility. Still, when single-core performance is more important than the number of available cores, e.g., so as to guarantee a real-time constraint, one may opt for fewer cores with more efficient configurations.

VI. Design-space exploration of Mixed-grain Reconfigurability

Considering the above area, power and performance overheads for a specific design point of our adaptive multiprocessors, we explore various granularity mixes. As shown in Figure 4, we evaluate the fault tolerance, performance and energy efficiency of different CG and CG+FG CMP designs at different fault densities and granularities, all occupying the same area[3]. The number

[3]The silicon-area constraint in our design-space exploration is equal to that of 9 baseline cores.

of wires needed to interconnect a block is estimated using Rent's rule with Rent's exponent $\beta = 0.5$. We consider a fault density that causes up to 20 faults in the total area[4], which is about 10^7 transistors for our small CMP design.

We quantify fault tolerance as the *average number of available cores* by analytically calculating the probability of constructing correctly functioning cores at each design point, as described in [16]. Figure 4a shows that for a low number of faults ($\leq$2), component-level redundancy provides a higher number of fault-free components, as the reconfigurability benefits have not yet been capitalized on. For a higher number of faults ($\leq$ 8), CG reconfigurability of granularity $\frac{1}{8}$ (CG($\frac{1}{8}$)) maximizes the availability of components. Adding FG logic at smaller SB sizes (CG+FG($\frac{1}{16}$)) is up to 13.5% better for fault densities beyond that point. Even for a high number of faults, CG+FG($\frac{1}{16}$) provides almost five available components, tolerating over 3$\times$ and 1.5$\times$ more faults than component-level redundancy (CG(1)) and other CG points, respectively.

Another measure of reliability, important for safety-critical systems requiring guaranteed rather than best-effort performance, is the *probability of maintaining at least 4 working components* (roughly half the baseline components that fit in the area). As shown in Figure 4b, this probability for the CG+FG($\frac{1}{16}$) design does not drop below 99% for up to 20 faults. On the contrary, for component redundancy (CG(1)) it is below 90% and 50% beyond 6 and 10 faults, respectively. Finally, the best CG granularity crosses the threshold of 90% for more than 17 faults.

We measure performance and power by analytically calculating the probability of each particular processor configuration using or not the FG block or having a particular number of extra stages. Each of these configurations, then, has a specific IPS and power consumption[5] per core, extracted from our previous evaluation. Performance, shown in Figure 4c, follows the same trend as the average number of available cores, however it drops faster as the fault density increases. This is because the repaired cores have lower performance due to deeper pipelines or due to using the FG block more often. CG(1) scores a better performance up to 4 faults. For higher fault densities CG($\frac{1}{8}$) is better, while CG+FG($\frac{1}{16}$) provides up to 7% higher performance above 13 faults. Besides, as illustrated in Figure 4d, CG(1) is up to 1.7$\times$ more energy-efficient than any other design alternative, although, for high fault densities the number of working cores is dramatically reduced. For medium and high fault densities CG is up to 12% more energy-efficient than CG+FG.

VII. Conclusions

Adding fine-grain logic in a CMP design to bypass hard errors can increase availability tolerating 3$\times$ more faults than core-level redundancy. Different fault densities require different granularities in the substitutable core-parts to maximize availability and minimize performance and energy overheads. Core-level redundancy is the most cost-effective solution for low fault densities. As the number of faults increases, coarse-grain and mixed-grain reconfigurability (of granularity $\frac{1}{8}$ and $\frac{1}{16}$, respectively) provide the best availability-cost tradeoff.

Acknowledgements

This work has been supported by the European Commission Seventh Framework Programme under the DeSyRe Project (www.desyre.eu), grant agreement #287611. We are grateful to Sally McKee for her help and valuable comments.

References

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation," *Micro, IEEE*, vol. 25, no. 6, pp. 10 – 16, 2005.

[2] M. Fair et al., "Reliability, availability, and serviceability (ras) of the ibm eserver z990." *IBM Journal of Research and Development*, vol. 48, no. 3-4, pp. 519–534, 2004.

[3] D. Bernick et al., "Nonstop advanced architecture." in *DSN*, 2005, pp. 12–21.

[4] C. LaFrieda et al., "Utilizing dynamically coupled cores to form a resilient chip multiprocessor," in *DSN*, 2007, pp. 317–326.

[5] S. Shamshiri and K.-T. Cheng, "Modeling yield, cost, and quality of a spare-enhanced multicore chip," *IEEE Trans. on Computers*, vol. 60, no. 9, pp. 1246–1259, 2011.

[6] N. Aggarwal et al., "Configurable isolation: building high availability systems with commodity multi-core processors." in *ISCA*, 2007.

[7] S. Shyam et al., "Ultra low-cost defect protection for microprocessor pipelines," in *ASPLOS XII*, 2006, pp. 73–82.

[8] B. F. Romanescu and D. J. Sorin, "Core cannibalization architecture: improving lifetime chip performance for multicore processors in the presence of hard faults," in *PACT*, 2008, pp. 43–51.

[9] S. Gupta et al., "Stagenet: A reconfigurable fabric for constructing dependable cmps," *IEEE Trans. on Computers*, vol. 60, no. 1, pp. 5–19, 2011.

[10] A. Pellegrini, J. L. Greathouse, and V. Bertacco, "Viper: virtual pipelines for enhanced reliability," in *ISCA '12*, 2012, pp. 344–355.

[11] D. J. Palframan, N. S. Kim, and M. H. Lipasti, "Resilient high-performance processors with spare ribs." *IEEE Micro*, vol. 33, no. 4, pp. 26–34, 2013.

[12] I. Wagner, V. Bertacco, and T. Austin, "Shielding against design flaws with field repairable control logic," in *DAC*, 2006, pp. 344–347.

[4] We consider that faults are uniformly distributed in the silicon, including both the CG and FG part. For the FG part we consider that the faults can be tolerated in the above fault densities via reconfiguration, even without using any detection (testing) mechanism, simply by relocating the bitstream as proposed in [14].

[5] Unused processor parts are considered switched off and, thus, not consuming power.

[13] A. DeHon and H. Naeimi, "Seven strategies for tolerating highly defective fabrication," *Design Test of Computers, IEEE*, vol. 22, no. 4, pp. 306–315, 2005.

[14] W.-J. Huang and E. McCluskey, "Column-based precompiled configuration techniques for fpga," in *FCCM*, 2001, pp. 137–146.

[15] V. Vasilikos et al., "Heuristic search for adaptive, defect-tolerant multiprocessor arrays," *ACM TECS.*, vol. 12, no. 1s, pp. 44:1–44:23, 2013.

[16] A. Malek et al., "A Probabilistic Analysis of Resilient Reconfigurable Designs," in *DFT*, 2014.