Reliability Improvement in Multicore Architectures Through Computing in Embedded Memory

Hadi Hajimiri, Somnath Paul, *Student Member, IEEE*, Anandaroop Ghosh, *Student Member, IEEE*, Swarup Bhunia, *Senior Member, IEEE*, and Prabhat Mishra, *Senior Member, IEEE*

Abstract*—Nanoscale devices provide the capability of gigascale integration in modern electronic systems. However, such systems suffer from high defect rates and large parametric variations that can adversely affect system reliability. Hardware duplication is an obvious direction but it incurs significant area overhead that is unacceptable in many scenarios. Memory-based computing (MBC) is a promising alternative to improve overall system reliability when few functional units are defective or unreliable under process-induced or thermal variations. Existing works demonstrated the utility of MBC in single-core based designs. In this paper, we explore the effectiveness of MBC in multicore architectures where each core uses a small private cache while a set of cores share a large second-level cache. The private as well as shared caches are used to perform computation on demand using a lookup table. When a functional unit fails, temporarily due to temperature induced variations or permanently, the associated computation is transfered to caches. Experimental results demonstrate that on-demand memory based computing can significantly improve reliability with minor loss in performance.

I. Introduction

Design and fabrication technologies are successful in scaling down the transistor dimensions to integrate more and more transistors in a single System-on-Chip (SoC). Technology scaling also introduces major challenges such as high defect rate and device parameter variations [1]. Increasing process-induced variations and high defect rate in nanometer regime leads to reduced yield [2]. These variations change the propagation delay in CMOS circuits, which may lead to delay failures. In Static Random Access Memory (SRAM) these variations may cause data retention or read/write failures [12]. Higher power density in modern high-performance microprocessors ($100W/cm^2$ for 50-nm technology [3]) leads to an overall increase of the processor temperature due to the limited cooling capacity of the package. Moreover, the power density varies across the chip surface resulting in localized hotspots [6]. Existing approaches address reliability concerns, caused by device parameter variations, during design time or by post-silicon correction and compensation solutions. Reliable computation with unreliable components has been actively studied for a long time. A wide variety of solutions have been proposed over the years with the goal of dynamic detection and correction of defects and variation-induced failures [2-4, 8]. These techniques typically incur large performance overhead [4] or do not address manufacturing defects [8].

In this paper, we propose an architecture-level solution for improving reliability for multicore architectures in the presence of both manufacturing defects and parameter variation induced reliability issues. Each core in a typical multicore architecture uses a small private (e.g., L1) cache whereas a set of cores share a large next level cache (e.g., L2). Our proposed scheme allows on-demand transfer of computation from functional units such as integer Arithmetic Logic Unit (ALU) to the private and shared caches. Here we assume that under a fixed delay target, one or more logic units are non-functional while any hard defect or variation induced parametric failure in memory has been addressed using suitable redundancy [13] or re-mapping [2] techniques. In such a scenario, the memory can compensate for hard manufacturing defects as well as parametric failures (such as delay failures) in logic units. Moreover, the method can be used to address reliability problems to thermal variations, by dynamically transferring activities of a functional unit (FU) to memory when the FU experiences high temperature. The basic idea is that part of a cache (or separate embedded memory) can be used to implement the functionality of different execution units, such as adder or multiplier by storing the results of Boolean functions in lookup table (LUT) format. As a result, reconfigured caches can be used as a private or shared reconfigurable computing resource for on-demand computing. Note that the proposed computing model does not adversely affect the memory integration density. Moreover, it also remains transparent to the software execution flow.

The rest of the paper is organized as follows. Section II provides an overview of memory based computation. Section III describes the proposed methodology for on-demand transfer of computation to the on-chip cache. Section IV presents the experimental results for a set of benchmark applications. Finally, Section V concludes the paper.

II. Memory Based Computation: An Overview

This section presents an overview of memory-based computation. Lookup table based implementation is common in case of Field Programmable Gate Array (FPGA), where small logic functions can be implemented using LUTs [9]. The memory based computing framework developed in [19] has made several important contributions: a) it focuses on realizing activities of common execution units in a processor (such as addition, multiplication etc.) and not specific iterative tasks, thus extending the applicability of the approach to general-purpose computations; b) it addresses improvement in yield and reliability under manufacturing defects and parameter variations; and c) it preserves the advantage of high device integration density of conventional SRAM based embedded memory design. The remainder of this section answers two important questions. First, it describes the circumstances when MBC is beneficial. Next, it describes how to perform computation in memory using illustrative examples.

A. Applicability and Limitations

MBC is suitable for arithmetic (such as add, subtract, compare) as well as logical (such as "and", "xor") operations which can be represented in terms of LUT. [19] uses a time-multiplexed reconfigurable computing model [10], where large arithmetic/logical functions are bit-sliced and the slices are represented as multi-input multi-output LUTs. These slices are evaluated in topological order over multiple cycles. The on-chip cache provides easy dynamic reconfigurability, by loading the lookup table for different operations on demand. The LUTs for different operations can be stored in the physical memory and fetched on-demand to the on-chip cache. A memory array provides a regular structure which is very scalable. For example, to achieve higher memory bandwidth the memory can be organized into multiple banks. For bit-sliced operation, these banks can be accessed in parallel to improve performance.

The functions which have relatively small number of inputs and outputs or the ones which can be bit-sliced are suitable for memory based computations. Arithmetic operations, such as additions and

978-1-61284-857-0/11/$26.00

multiplications, often involve operands of large sizes (e.g., two 32 or 64-bit operands). However, we note that such operations can be easily bit-sliced and hence efficiently represented in terms of LUTs. Complex functions including trigonometric, logarithmic, exponent, square root or other transcendental functions are also shown to be very amenable to memory based implementation. These functions can be either bit-sliced [17] or spatially decomposed for evaluation using multiple LUTs [18]. Implementing such functions would however require small adaptation in the glue logic [19] (an additional hardware unit used to interface the processor pipeline with the MBC framework).

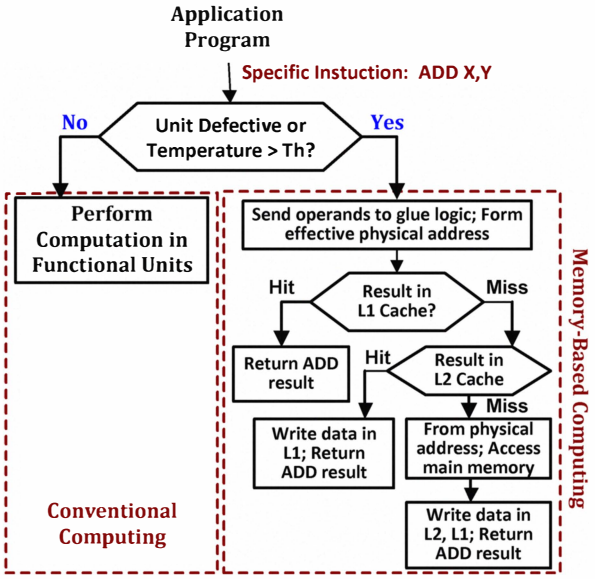

Fig. 1. An overview of memory-based computation

Dynamic transfer of computation to on-chip memory can be effective in salvaging chips under various failure scenarios. Here, we outline three common failure cases where MBC can be beneficial. First, when one or more of the functional units, such as integer or floating point adder/multiplier, are not-functional due to manufacturing defects, MBC can ensure that chips with failing functional units are not discarded. Next, MBC can address parameter variations. In scaled technologies, process induced device parameter variations can affect circuit parameters such as critical path delay. Such variations can lead to localized within-die delay variations among the functional units, which increases with technology scaling. Similar to logic, memory is also subject to within-die variations. However, memory cell failures due to within-die variations can be tolerated using circuit/architecture level techniques such as dynamic resizing of the cache to avoid the faulty cells [2]. Finally, MBC can also address temperature variations. Highly active portions of the die such as the issue stage, or the execution unit and the integer register file can have twenty times the power density of less active blocks such as the on-chip cache block [5]. The most commonly used technique to maintain on-die thermal integrity include Dynamic Voltage and Frequency Scaling (DVFS) technique [4]. The scheme is useful for handling thermal emergency and prevents the processor from overheating. However, DVFS limits the performance of the processor by causing a global slowdown of all the units until the transistors in the hotspots have an acceptable junction temperature. The on-chip memory of a processor typically possesses a cooler temperature profile [6]. Hence, it can be used to share a part of the workload of the execution units when necessary. This reduces the switching activity and hence decreases the power density of the execution units, thereby reducing its temperature.

Fig. 1 shows an overview of the memory based computing scheme as proposed in [19]. If one or more functional units are defective, the operands for the faulty functional unit is used to form the effective physical address for accessing the LUTs corresponding to the mapped function. If these LUTs already reside in on-chip cache, outputs for the mapped function are easily retrieved. In case of a cache miss, these values are retrieved from the main memory.

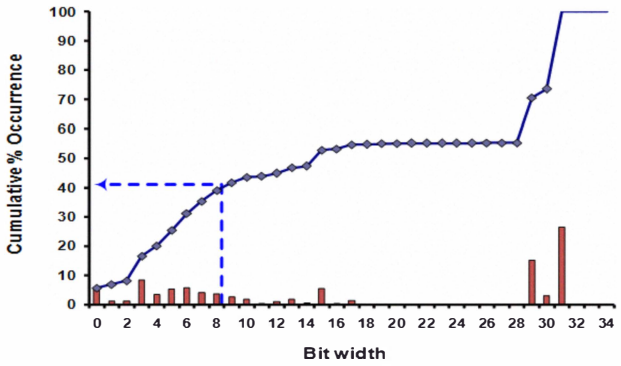

Fig. 2. In SPECInt2000 benchmarks, 40% of the integer instructions have bitwidth less than 8-bits.

It is possible that MBC can introduce performance degradation. However, the performance loss due to the memory based computation remains within tolerable limits primarily for two reasons. The operands for a specific instruction (e.g., add/multiply) experience high locality of reference across different clock cycles thus requiring loading only part of the LUTs in the private L1 cache whereas the rest can be in shared L2 cache or in main memory. Also, most of the operations in the functional units do not use the full-width operands [14] as shown in Fig. 2. In such cases, the operation latency of computing in caches can be comparable to the latency of computing directly in the cores. Fig. 2 shows that for integer operations in SPECInt2000 benchmark applications over 40% operations have bitwidth of 8 or less. Memory based computation can be effective for these operations since smaller operand width translates to fewer memory accesses and improved performance.

B. Memory-based Addition: An Example

Carry-select addition of two 32-bit operands using memory based computation is shown in the Fig. 3. If one of the operands is zero, the addition is completed in one cycle. If not, the 32-bit operands are bit-sliced into 8-bit operands. For each set of 8-bit operands, the addition result for both input carry zero and one is looked up from the cache. The input carry is then used to select one of the two results. The same operation is repeated for all the 8-bit operands. Thus the entire addition procedure is completed in two steps, a memory lookup and subsequent carry-select addition using the 8-bit operand addition results. Note that due to the commutative property of "add" (a + b = b + a), total memory required to store all the "add" results is halved and comes to 64KB. Considering the result for all the sub-operands (X_i, Y_i) are present in the on-chip memory, the worst-case evaluation time for two 32-bit operands is 4 cycles. Although this evaluation time is more than that of respective functional units, due to the fact that most of the operations (almost half of the integer operations) are narrow width [14], the average penalty in performance is not significant. The exact latency of operation depends on number of memory accesses as well as the number of cycles required to access the memory. The later is determined by the location of relevant LUT in the memory hierarchy.

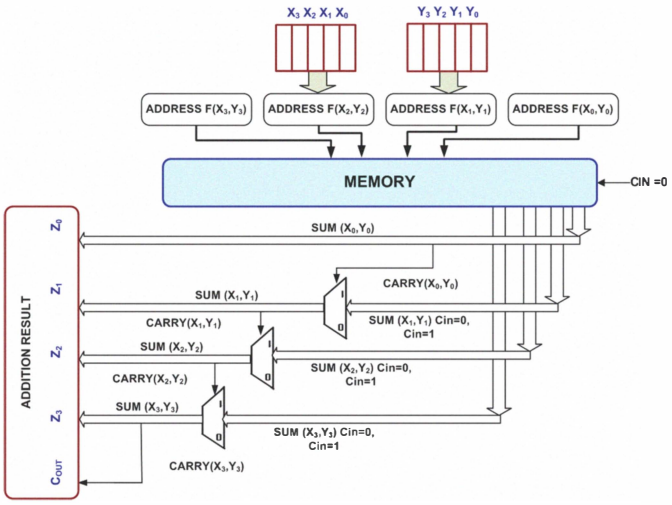

Fig. 3. Implementation of memory based addition using carry-select addition.

III. Dynamic Transfer of Computation to Caches

In this section, we explore the effectiveness of memory based computing (MBC) in multicore architectures. We have applied MBC to realize the functionality of the integer execution unit (adder and multiplier) in each core. Fig. 4 shows a broad overview of MBC in a multicore framework. This architecture has **n** cores each having it's own private L1 data and instruction caches. All the cores share a L2 combined (instruction+data) cache which is connected to main memory. To support MBC, each core also has a L1-level MBC cache that store most frequently accessed entries of the LUTs. Unless additional embedded memory can be used for L1 MBC cache, the existing private L1 data cache can be partitioned into two parts: one part dedicated for MBC cache to store mostly frequently used LUTs, and the other part will be used for conventional data accesses. Similarly, to implement L2-level MBC cache either additional embedded memory can be used or existing shared L2 cache can be partitioned to make space for MBC LUTs.

Under normal circumstances, issue logic sends the instruction to the respective functional units. However, if the functional unit is not available (due to defect or temperature stress), issue logic bypasses the original functional unit for memory based computation, as shown by dashed lines in Fig. 4. In both normal and defective (MBC) scenarios, the L1 (instruction as well as data) and L2 caches performs in conventional manner i.e., stores frequently used data (instruction) for performance improvement. Once a need for bypassing the normal execution unit has been detected the processor will issue an indication for the Operating System (OS) to load the result tables for that particular operation in a section of the main memory. Note that the OS is responsible for mapping the virtual memory address space into the physical memory address space. The formation of virtual and physical addresses are same as described in [19]. After calculation of the effective address, this address is used to access a location of the MBC cache memory where the result of the operation is stored. Depending on the operand width, multiple memory accesses may be required to complete an operation. The result bits obtained from evaluation of multiple bit-slices are accumulated inside the glue logic.

Note that loading of the ADD/MULT results table would follow the same steps as loading of instruction/data into the main memory and from the main memory into the processor cache. When MBC is invoked for the first time, virtual to physical address mapping will encounter a page fault. The exception handling routine of the OS would load the corresponding pages into the main memory and load the virtual to physical translation into the page table. Subsequent access to the same virtual address would bring the mapping and the data into the MBC mapping table and the on-chip cache, respectively. The proposed activity transfer scheme requires additional hardware that generates the virtual and the physical addresses for accessing the cache and the main memory, respectively. In order to exploit the commutativity property of the addition and multiplication operation, we use a 32-bit comparator for comparing the operands and aligning the larger operand to form the most significant bits. The same comparator can also be used for the multiplication procedure. Additional hardware required for multiplication includes i) a 32-bit shifter to obtain partial products of appropriate weight before they are added and ii) 32-bit priority encoder in order to minimize the number of memory accesses for the generation and addition of partial products in case of narrow width operands.

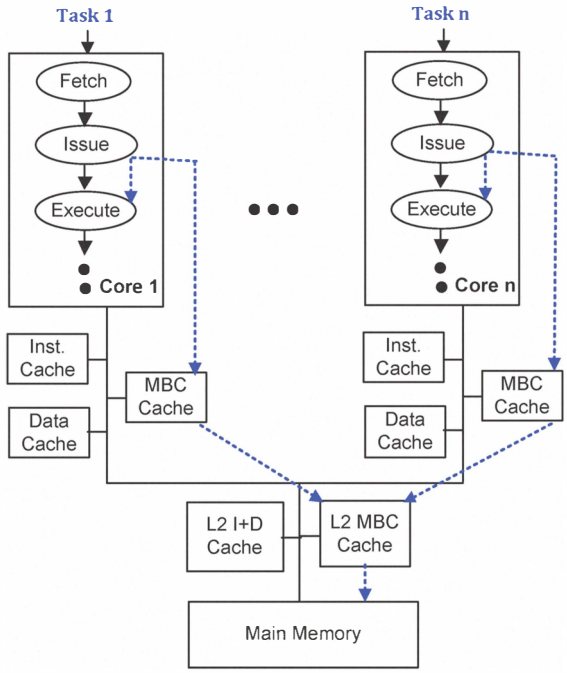

Fig. 4. Memory-based Computation in Multicore Architectures

IV. Experiments

A. Experimental Setup

To check the effectiveness of the proposed scheme, we implemented the computation transfer mechanism in a widely used multicore simulator, M5 [15]. We enhanced M5 to make the required modifications in processor cores as well as in memory hierarchy. We modified memory hierarchy to support cache partitioning, to introduce L1 private MBC caches and shared L2 MBC cache. We configured the simulated system with a two-core and four-core processor each of which runs at 500MHz. The DerivO3CPU model [15] in M5 is used which represents a detailed model of an out-of-order SMT-capable CPU which stalls during cache accesses and memory response handling. The effectiveness of the proposed framework was validated for two different scenarios: 1) improving the reliability of operation under temperature variation and 2) improving manufacturing yield of a processor when some functional units become inoperative due to manufacturing defects. For the first case, the temperature threshold for the integer execution unit was set at $100^{\circ}C$. We have used reference inputs for the benchmarks in the SPEC2000 benchmark suite [7]. "Hotspot 2.0" tool [16] was used for estimating the temperature profile of the integer ALU units. In order to estimate the die thermal profile from Hotspot, power dissipation values of the individual functional units were obtained from Wattch 1.0 [11] at regular time intervals.

B. Results

Fig. 5 shows the performance loss in case of different Spec2000 benchmarks due to MBC. We formed different task sets to be executed together in multi-core environment (one task per core). For example, in the 2-core framework, the tasks are organized as: (applu, art), (apsi, bzip), (eon, gcc), (gzip, lucas), and (mgrid, perlbmk). Similarly, for 4-core scenario they are organized as: (applu, art, apsi, bzip), (eon, gcc, gzip, lucas), and (mgrid, perlbmk, bzip2, gcc). Average increase in CPI of 1.32%, 1.20%, and 0.80% was observed using MBC for temperature management in the integer ALU for single-core, 2-core, and 4-core framework respectively. The impact on performance due to MBC is mainly two-fold: 1) increased operational latency compared to execution in the functional units, and 2) higher capacity misses in the cache due to reduction in cache size.

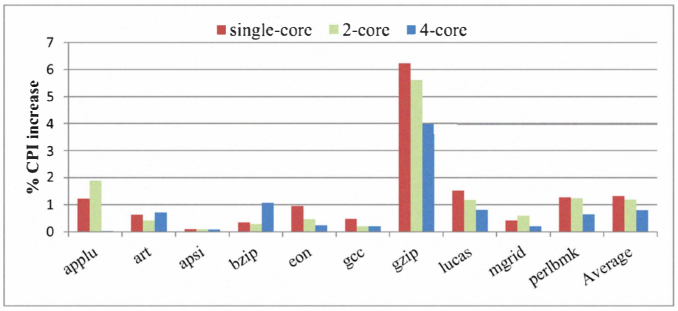

Fig. 5. Impact on processor performance (CPI) for adaptation to thermal stress using memory based computation

From Fig. 5 it can be observed that the performance penalty decreases as the number of cores increases. This is due to the fact that tasks on different cores compete for the shared resources (memory access or shared L2 cache) and as the number of cores increases, the overall performance decreases. This performance degradation hides some portion of the performance overhead introduced by MBC. Moreover, our study suggests that increasing cores can be beneficial in certain MBC scenarios when multiple cores share the same LUT in L2 cache. Thus in a system with large number of cores and/or tasks, performance degradation caused by using MBC is less noticeable.

In order to investigate the effect of using MBC for tolerating manufacturing defects we have considered a two-core processor[1] with the following task sets: Set 1 (qsort, vpr), Set 2 (basicmath, stringsearch), Set 3 (swim, untoast), Set 4 (parser, toast) and Set 5 (applu, sixtrack). We considered the scenario of 4 adders and 1 multiplier being defective in each core. Fig. 6 compares the performance of a processor core with and without MBC when multiple FUs are considered inoperative and an extra on-chip memory hierarchy is present to support MBC operations. In the proposed scheme, activities of these defective functional units are dynamically transferred to the memory. As observed from Fig. 6, if we do not employ memory based computation, the performance overhead for a faulty processor core is 5.5%. However, when the proposed activity transfer scheme is incorporated in the processor architecture, the loss in performance is considerably less (1.6%).

V. Conclusion

We presented a novel memory based computing framework for multicore architectures that enables dynamic transfer of computation to embedded memory for improving yield and reliability. The basic idea is to use both private and shared caches as reconfigurable computing resources. Various functions are implemented in caches

[1]The baseline configuration has 6 adders and 2 multipliers in each core

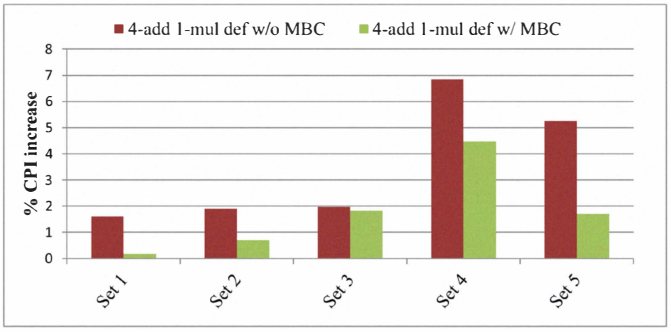

Fig. 6. Performance results for 4-adder/1-multiplier defective scenario.

using a lookup table. Our approach can be effectively used to tolerate permanent manufacturing defects in a processor core or in a functional unit of a core, thus improving functional yield of multicore architectures. It can improve parametric yield under within-die variations when specific functional unit becomes unusable while memory remains functional. It can also be applied to temporarily bypass the activity in functional units under time-dependent local variations, thus providing dynamic thermal management by activity migration. Our experimental results using a set of applications on M5 multiprocessor simulator demonstrate that our proposed activity transfer method provides considerable benefit in yield and reliability at the expense of small loss in performance and low hardware overhead. Future work will investigate the use of memory-based computing for hardware acceleration of compute-intensive tasks.

References

[1] S. Borkar, "Designing reliable systems from unreliable components: the challenges of transistor variability and degradation", *IEEE Micro*, 2005.
[2] A. Agarwal et al, "A Process-Tolerant Cache Architecture for Improved Yield in Nanoscale Technologies", *IEEE Trans. on VLSI*, 13,27-38, 2005.
[3] C. Minsik, "TACO: temperature aware clock-tree optimization", *International Conference on Computer Aided Design*, 2005.
[4] R. Mcgowen et al, "Power and temperature control on a 90-nm Itanium family processor", *IEEE Journal of Solid State Circuits*, 2005.
[5] K. Asanovic et al, "Reducing power density through activity migration", *International Symp. on Low Power Electronics and Design*, 2003.
[6] K. Skadron et al, "Temperature-aware microarchitecture", *International Symposium on Computer Architecture*, 2003.
[7] Spec 2000 benchmarks [Online], *http://www.spec.org/cpu/*.
[8] D. Ernst et al, "Razor: A Low-power Pipeline Based on Circuit-Level Timing Speculation", *IEEE Micro*, 2003.
[9] P. Chow, "The Design of an SRAM-Based Field Programmable Gate Array, Part II: Circuit Design and Layout", *TVLSI*, 1999.
[10] D. Jones and D.M. Lewis, "A time-multiplexed FPGA architecture for logic evaluation", *Custom Integrated Circuits Conference*, 1995.
[11] D. Brooks et al, "Wattch: A framework for architectural-level power analysis and optimizations", *ISCA*, 2000.
[12] S. Mukhopadhyay et al., "Modeling of Failure Probability and Statistical Design of SRAM Array for Yield Enhancement in Nanoscaled CMOS", *TCAD*, 2005.
[13] C.T. Huang et al, "Built-in redundancy analysis for memory yield improvement", *IEEE Transactions on Reliability*,Vol.52, pp.386-399, 2003.
[14] D. Brooks and M. Martonosi, "Dynamically Exploiting Narrow Width Operands to Improve Processor Power and Performance", *International Symposium on High Performance Computer Architecture*, 1999.
[15] N. Binkert et al., The M5 simulator: Modeling networked systems, *IEEE Micro*, vol. 26, no. 4, pp. 52 60, 2006.
[16] Hotspot 2.0: http://lava.cs.virginia.edu/HotSpot/documentation.htm
[17] C. Rebeiro, D. Selvakumar and A.S.L. Devi, "Bitslice Implementation of AES", *Lecture Notes in Computer Science*, Vol. 4301, 2006.
[18] T. Sasao, S. Nagayama, J.T. Butler, "Numerical Function Generators Using LUT Cascades", *IEEE Trans. on Computers*, Vol. 56, No. 6, 2007.
[19] S. Paul and S. Bhunia, "Dynamic Transfer of Computation to Processor Cache for Yield and Reliability Improvement", *IEEE TVLSI*, 2011.