HW-SW Integration for Energy-Efficient/Variability-Aware Computing

Gasser Ayad, Andrea Acquaviva, Enrico Macii
DAUIN, Politecnico di Torino
Corso Duca Degli Abruzzi 24
Torino 10129, Italy
Email: gasser.ayad, andrea.acquaviva, enrico.macii@polito.it

Brahim Sahbi, Romain Lemaire
CEA-Leti, 17 rue des Martyrs
38054 Grenoble cedex 9, France
Email: brahim.sahbi, romain.lemaire@cea.fr

***Abstract*—Recent trends in embedded system architectures brought a rapid shift towards multicore, heterogeneous and reconfigurable platforms. This imposes a large effort for programmers to develop their applications to efficiently exploit the underlying architecture. In addition, process variability issues lead to performance and power uncertainties, impacting expected quality of service and energy efficiency of the running software. In particular, variability may lead to sub-optimal runtime task allocation.**

In this paper we present a holistic approach to tackle these issues exploiting high level HW/SW modeling to customize the runtime library. The customization introduces variability awareness in task allocation decisions, with the final purpose of optimizing a given objective: Execution time, power consumption, or overall energy consumption.

We present a complete walkthrough, from top-level modeling down to variability-aware execution using a parallelized computational kernel running on a next generation, NoC based, heterogeneous multicore simulation platform.
[1] [2]

I. Introduction

MULTICORE architectures are becoming indispensable to high-end embedded computing as application energy-efficiency requirements exceed 10 GOPS/Watt. Unfortunately, sub-65 nm CMOS technology nodes will be increasingly affected by the variation phenomena, and multicore architectures will be impacted in many ways by the variability of the underlying silicon fabrics. In particular, intradie process variations result in significant core-to-core frequency variations [2]. This problem is being addressed at multiple levels of abstraction, from the circuit to the architectural level. Variation-tolerant multicore platforms require circuits to monitor variations and to compensate them, as well as software policies to decide when and how to apply compensation in response to static and dynamic perturbations of the nominal operating characteristics [2].

In that sense, chip design has turned enormously complex and imposing a large effort for the programmers to develop their applications. For this reason, new and more efficient tools for software development are needed to ensure software productivity and time to market of new applications. Specifically, the automation of the software design process starting from high level models all-the-way down to a customized implementation on specific architectures has become a key factor to increase programmer productivity.

In this paper, an innovative approach for variability compensation from a high level model of the underlying hardware platform is being presented. The approach starts from a SysML description of the target HW platform under Artisan Studio™ tool suite. The abstraction level of the description is thought to be detailed enough to capture information relevant for the runtime manager to make task allocation decisions: Types and organization of cores, memory hierarchy, topology of the interconnect. Variability-relevant information is annotated in the SysML components as properties: Clock frequency, static power, dynamic energy. These quantities results from the post-manufacturing characterization, so that process variability is taken into account. This information is passed through a customization language to the runtime library to make variability aware decisions. Artisan Studio is used to generate runtime customization information in XML format. To characterize the type of information required for variability aware runtime allocation, we considered state-of-art policies and we implemented a simple test case to verify the whole customization flow.

This paper is organized as follows. First, state of the art works on high level modeling as well as variability awareness and tolerance will be presented in Section II. A methodology for modeling the target platform using the SysML modeling language will then be illustrated in Section III. Runtime library customization is reported in Section IV. A software application is used as a benchmark to test the runtime customization feasibility and efficiency. A description of experimental set-up is reported in Section V. The paper closes by delivering the overall results and conclusion as well as the prospect for further development in Section VI

II. Related Work

A. High-Level Modeling of Target Hardware Platforms

The increasing amount of hardware resources in next generation MultiProcessor Systems-on-Chip (MPSoC) calls for

[1]This work is supported by the research project TouchHMore FP7-ICT-2011-7-288166.

efficient design methodologies and tools to reduce their development complexity. In [4] presented is a candidate MPSoC design environment Gaspard2, which uses the MARTE (Modeling and Analysis of Real-Time and Embedded systems) standard profile for high-level system specification. Gaspard2 adopts a methodology based on Model-Driven Engineering. It promotes separation of concerns, reusability and automatic model refinement from higher abstraction levels to executable descriptions.

In addition, [5] presents a novel methodology for modeling partially dynamic reconfigurable hardware at transaction level. The paper covers the lack of tools and mechanisms for the design of reconfigurable logic at system level and for the exploration of the different configurations of such architectures. The presented mechanisms have been implemented in ReSP, a transaction-level simulation platform especially targeted to multi-processor embedded architectures. The adopted methodology allows the definition of the reconfigurable functionalities through scripted languages, therefore the switching of any software function for a hardware one, the modeling of configuration delays, area use, etc., can be easily performed. Overall, the methodology enables a powerful exploration of the system functionalities by switching between hardware and software components.

B. Awareness of And Compensating The Variability Factor

Recently, much attention has been given to task allocation and scheduling strategies for MPSoCs affected by variability and aging. [1] gives an overview of the concept of variability. Concerning the allocation countermeasures proposed in literature, a process variation-aware thread mapping has been recently proposed in [6]. In that work, the main purpose is to maximize performance and it targets loop-intensive applications. However, this approach does not provide an optimal solution and does not take energy consumption into account. Moreover, [7] proposes a statistic scheduling approach to mitigate the impact of parameter variations in a multiprocessor platform. The proposed policy is based on a static estimation of task execution times and variability information but it does not consider power consumption [3].

In paper [3], the concept of time-constrained variability-aware task allocation methodology with the objective of minimizing the energy consumption is proposed. The allocation problem was formulated in two sequential steps where the solution computed by a Linear-Programming (LP) approach was fed into a Bin-Packing (BP) algorithm for final task allocation. That paper targets realtime streaming multimedia applications [2]. Also in scope of streaming applications, [2] focuses on software counter-measures which reshape application workload to account for variability in the underlying multiprocessor fabric. Proposed is workload allocation policy to compensate for core-level performance and power variations. The focus is multimedia processing, which is typically characterized by application-level frame-rate constraints. In that context, the top-priority goal of variability compensation policies is to meet the real-time constraints imposed by the frame rate of the multimedia stream, while minimizing energy as a secondary objective [2].

Most closely related to our approach, variability-aware workload allocation policies for independent task sets are presented in [8]. Two policies are considered, aiming at maximizing performance or minimizing power, with the assumption that voltage scaling is available on a per-core basis (this is not supported in our platform). Moreover [8] assumes that the number of tasks is not larger than the number of cores (in our paper, it is larger). Our results are obtained with similar versions of the policies described in [8], with suitable modifications to suit our system setup [3].

C. From High-Level Model to Runtime - A Top-down Approach

According to the aforementioned relevant work, there is an obvious gap between top-level modeling of target hardware MPSoCs and variability awareness at the runtime level. In more detail, the center of the methodology is the high level modeling language (UML/SysML) that will be used to describe the target platform and application. High level modeling allows an architectural independent description of the application and for this reason it is prone to customization for different architectural templates. Customization will be performed in an automated way through generation of parallel code for multicore tiles and the required mechanisms to manage reconfigurable DSPs/accelerators. In addition, the customization environment will focus on energy efficiency and robustness of the generated code, where the uncertainties due to fabrications of transistors in nanometer technologies will be hidden, thus mitigating their impact in terms of energy and performance. From a research perspective, this work is taking a lead in bringing variability issues into the software design flow and thus closes the gap mentioned right before. That is, coupling together the automatic toolchain customization strategy with high level platform modeling.

The paper in hand presents a primary version of that innovating approach for automatic runtime customization for variability compensation and energy efficiency, by provisioning the customization information through a high-level model of the target platform. The model is developed using one of leading tools in the system modeling industry, Atrisan Studio. Our platform is a software environment that simulates a conceptual cut-down version of Genepy [9] hardware platform.

III. Description of Target Platform and Modeling Methodology

The target platform is a simulation environment for a Genepy-like architecture. This simulator is a simplified version of the Genepy platform, since it models only the MIPS subsystem of each SMEP cluster, without the DSPs (Fig. 1). Nevertheless, it has the same NoC topology and though all embedded applications could be tested on this first basic version.

This platform performs simulation in HCE, or Host Code Execution, mode. In this mode, compiling the application takes

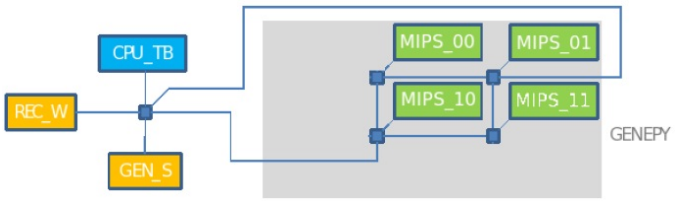

Fig. 1. Simplified Genepy Platform

place for the user's machine platform. For instance if the underlying physical hardware is an x86 architecture, then the application code is compiled for x86. A library is created from the application source code, and is linked dynamically with the platform. This mode is fast and allows the use of GDB, the GNU Project debugger.

The platform is connected to an external unit called CPU_TB. This unit is used to boot the clusters (illustrated in Fig. 2), load the application in the internal memory of the mips cores, and is able to interact with any of them via the NoC routers (illustrated as blue squares in Fig. 1).

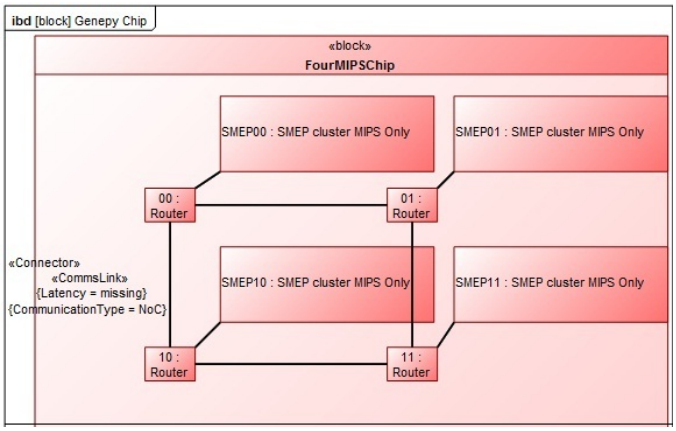

Fig. 2. Platform Structural Modeling in Artisan Studio

From modeling perspective, a methodology is proposed for abstracting existing RTL IPs into SysML[3] components. During the abstraction flow, it is possible to set the level of detail to be maintained in SysML, such as hierarchical structure and data types of the IPs, in order to allows designers to choose the level of detail to be preserved in the SysML model. The methodology aims at producing SysML models with both structural and behavioral information.

In that sense, the target platform consists of hardware structure and also hardware capability information. Structural information is all about hardware units (the building blocks) and interconnections, while capability information describes what mechanisms are supported for energy and variability management of the target hardware.

[3]SysML is specified as a profile (dialect) of the Unified Modeling Language (UML™), the industry standard for modeling software-intensive systems, so SysML is frequently implemented as a plugin for popular UML modeling tools.

In Artisan Studio, the hardware model, as illustrated in Fig. 3, has been called "LimitedPlatform" and it consists of a number of packages and subpackages (depicted as yellow folders in Fig. 3). Inside the FourMIPSChip package we got a component block named after the same name of the parent package and depicted as a red cube (Fig. 3). This block is composed of eight parts (four routers and four MIPS cores) as well as an internal block diagram called "Genepy Chip" (Fig. 3). This diagram shows how the Genepy platform is structurally built, as appears in Fig. 2.

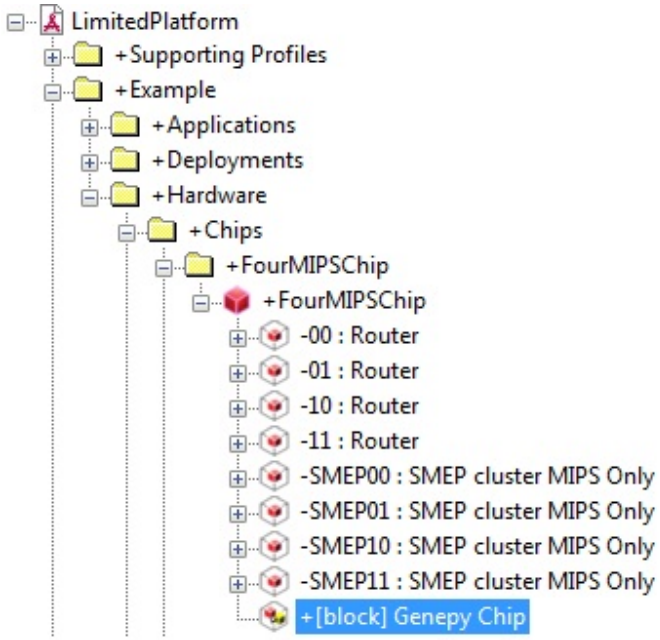

Fig. 3. Model Composition

The capability information, however, are located under the Cores package (Fig. 4). In the figure, the word "manufacture" means that, for instance, clock frequency is a design-time feature and its value is to be set by the model developer.

IV. Runtime Library Customization

The objective of this part of the work is the development of a methodology for the automatic customization of the runtime library devoted to the mapping of tasks to processing cores and the allocation of communication resources for the interfacing with those cores. The customization includes energy-efficiency and variability-awareness features. The runtime library will be customized by automatically generating or deriving a hardware description language from the hardware model developed in section III. This language is in XML format and contains the structural and capability information of the target hardware platform. We also refer to it as a "customization language".

The parameters used for customization of the runtime library are specified here. These are the parameters that mainly concern the dynamic decisions that are not taken at the compiler level because they depend on runtime conditions and will be related to both performance and power consumption. Parameters are mainly clock frequency and power

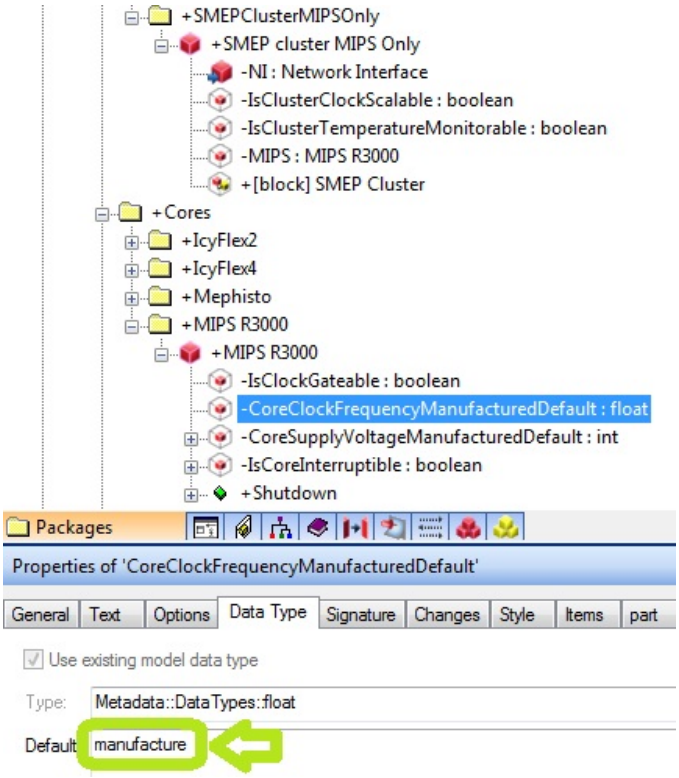

Fig. 4. Capability Modeling

```
<Core name="MIPS" type="MIPS R3000" id="31"
      ClockScalingDelay="missing"
      CoreClockFrequencyMAX="manufacture"
      CoreClockFrequencyMIN="manufacture"
      CoreClockFrequencyManufacturedDefault="0.87"
      CoreSupplyVoltageMAX="manufacture"
      CoreSupplyVoltageMIN="manufacture"
      CoreSupplyVoltageManufacturedDefault="manufacture"
      IsClockGateable="true"
      IsClockScalable="false"
      CoreAveragePowerManufacturedDefault="1.2"
      IsCoreDynamicPowerMonitorable="false"
      IsCoreFrequencyMonitorable="false"
      IsCoreInterruptible="true"
      IsCoreLeakagePowerMonitorable="false"
      IsVoltageGatable="false"
      IsVoltageScalable="false"
      VoltageScalingDelay="na"
      ></Core>
```

Fig. 5. An Excerpt from The XML File - Parameters per One Core

```
<Platform name="FourMIPSPlatform" id="1">
  <!-- inter chip connectors go here (but our platform has one
chip only) -->
  <Chip name="FourMIPSChip" type="FourMIPSChip" id="2">
    <!-- intra chip connectors external to clusters go here (8
connections) -->
    <Connector end1="3" end2="18" id="5" latency="missing"
CommunicationType ="NoC"></Connector>
    <Connector end1="4" end2="19" id="6" latency="missing"
CommunicationType ="NoC"></Connector>
    <Connector end1="17" end2="18" id="7" latency="missing"
```

Fig. 6. Another Excerpt from The XML File - Communication Delays

consumption settings per each single core. The customization methodology supports a class of policies for task allocation and scheduling of tasks on the available cores. The policies have the objective of either overall performance maximization, power consumption minimization, or energy saving. A number of policies have been discussed in [3] and [11]. The policies are ranging in terms of approach (i.e. heuristic or probabilistic etc.), complexity, and effectiveness. For our work we have employed two of such policies, namely: frequency rank, and power rank.

The XML customization language file is generated from Artisan Studio, which we used in section III to model the Genepy platform. The file contains the per-core frequency and power values. Figure 5 displays part of the generated XML file. It shows that the model designer has set the frequency value to 0.87 and the power value to 1.2 for one of the cores of the hardware platform. The other values have not been set (so their values have been left unchanged to "manufacture"). The rest of the XML file is pretty similar to Fig. 5 but for the remaining cores. Also some other information is generated in the XML file such as connection delays as in Fig. 6 (they have been left here because they are believed to be rather small in our platform).

Tailoring the runtime behavior relies on a selected ranking policy. In scope of this work we target probabilistic frequency ranking (for performance optimization) and probabilistic power ranking (for power consumption cut). The XML file is parsed and the frequency and power values for each core are obtained. Core#00 is in charge of the runtime and is responsible for calculating the allocation decisions (that is, it's considered as the master core). So no tasks are assumed to be executed on core#00 i.e. locally. Instead, all tasks are distributed over the other cores, which we consider as slaves.

The application is loaded to the master core only. Per each task, a decision is calculated to determine which core of the slaves is the one for the next task allocation. For the probabilistic frequency ranking, to each core a probability of allocation is associated which is proportional to the speed difference among the cores, to achieve overall execution time equalization. A reward/penalty is given depending on the distance (as number of hops) from the master core. For power, a similar approach applies, but without considering a reward or penatly for distance since power estimation tends to be more complex when taking the NoC elements into account.

The following pseudo code describes the execution and allocation mechanism in general terms:

```
1  Load application to master core
2  Parse XML file
3  Calculate allocation probabilities based on frequency or power-saving values
4  Do for each task until end of input stream:
5      Call a method to get a core (slave) number for processing the task
6      Allocate task to the assigned core
7      Anticipate and read processing results from the core
8      Go back to loop beginning for a new task
9  End
```

Homogeneous			
	Norm. Freq.	Weighted Values	Allocation Percentage
Core#01	0.95	1	34.3%
Core#10	0.95	1	34.2%
Core#11	0.95	0.9	31.5%
Quasi-Homogeneous			
	Norm. Freq.	Weighted Values	Allocation Percentage
Core#01	1.00	1.05	35.6%
Core#10	0.95	1.00	33.9%
Core#11	0.95	0.90	30.5%
Fully Heterogeneous			
	Norm. Freq.	Weighted Values	Allocation Percentage
Core#01	0.95	1.00	34.1%
Core#10	0.90	0.95	33.0%
Core#11	1.00	0.95	32.9%

TABLE I
RESULTS ABOUT ALLOCATION OF MATRIX MULTIPLICATION THREADS ON THE MIPS CORES IN GENEPY PLATFORM SIMULATOR USING THE PROBABILISTIC RANK FREQUENCY POLICY.

Homogeneous		
	Power Value (mW)	Allocation Percentage
Core#01	15	33.7%
Core#10	15	32.9%
Core#11	15	33.4%
Quasi-Homogeneous		
	Power Value (mW)	Allocation Percentage
Core#01	13	34.0%
Core#10	15	31.9%
Core#11	13	34.1%
Fully heterogeneous		
	Power Value (mW)	Allocation Percentage
Core#01	15	31.8%
Core#10	13	35.2%
Core#11	14	33.0%

TABLE II
RESULTS ABOUT ALLOCATION OF MATRIX MULTIPLICATION THREADS ON THE MIPS CORES IN GENEPY PLATFORM SIMULATOR USING THE PROBABILISTIC RANK POWER POLICY.

V. EXPERIMENTAL SETUP

Matrix multiplication was chosen as a convenient benchmark because it is representative of many multimedia kernels and easily scales for a wide range of performance testing because the work grows like N3 for matrices of order N. There are three nested loops in the code; the inner loop is short, consisting, in the simplest implementation, of a single multiply and add.

Our application is aimed to be multiplication of two matrices A and B. Each row of A is considered as a single thread (i.e a task). So the total number of tasks to be mapped to slaves is equal to the total number of rows of matrix A. Our target hardware platform is restricted to just three slaves (referred to as #01, 10, and 11). The reward given for the distance from the master core is 0.05 on the normalized frequency values.

VI. RESULTS AND CONCLUSION

We ran the simulation, as described in Section IV, using three different configurations: i) Homogeneous, where all cores are equal; ii) Two equal frequencies; iii) Fully heterogeneous, where all cores are different. These cases are representative of variability scenarios. In the first we assume the platform is homogeneously degraded. In the second the degradation is localized while in the third the degradation is randomly distributed across the cores. Note that frequencies values are normalized with respect to the maximum frequency. Results reported in Table I for probabilistic rank frequency and in Table II for rank power.

Results show that the percentages of allocation depend on the speed and power differences. In case of probabilistic frequency policy we report also in the third column the weighted frequency values accounting for the distance from the master core (Core#00).

VII. ONGOING WORK

Future work will be devoted to test more complex allocation policies considering also offloading to DSPs and accelerators. Moreover, the HW/SW integration approach will be applied not only to customize the runtime library, but also to generate variability and energy-aware annotations into the application code. To achieve this, we will extend the code generation capabilities of the Artisan Studio tool within the ToucHMore project.

REFERENCES

[1] D. Marculescu and E. Talpes, "Variability and Energy Awareness: A Microarchitecture-Level Perspective", Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA.

[2] Paterna, F.; Acquaviva, A.; Caprara, A.; Papariello, F.; Desoli, G.; Benini, L.; , "Variability-Aware Task Allocation for Energy-Efficient Quality of Service Provisioning in Embedded Streaming Multimedia Applications," *Computers, IEEE Transactions on*, vol.61, no.7, pp.939-953, July 2012.

[3] Paterna, F.; Benini, L.; Acquaviva, A.; Papariello, F.; Desoli, G.; , "Variability-tolerant workload allocation for MPSoC energy minimization under real-time constraints," *Embedded Systems for Real-Time Multimedia, 2009. ESTIMedia 2009. IEEE/ACM/IFIP 7th Workshop on*, vol., no., pp.134-142, 15-16 Oct. 2009.

[4] Jean-luc Dekeyser , Rabie Ben Atitallah , Abdoulaye Gamati , Pierre Boulet , Anne Etien, "Using the UML Profile for MARTE to MPSoC Co-Design," INRIA Lille Nord Europe and LIFL, France.

[5] Beltrame, G.; Fossati, L.; Sciuto, D.; , "High-Level Modeling and Exploration of Reconfigurable MPSoCs," *Adaptive Hardware and Systems, 2008. AHS '08. NASA/ESA Conference on*, vol., no., pp.330-337, 22-25 June 2008.

[6] S. Hong and et al., "Process variation aware thread mapping for chip multiprocessors," in *Design, Automation and Test in Europe 09, IEEE*, 2009, pp. 821826.

[7] F. Wang and et al, "Variation-aware task allocation and scheduling for mpsoc," in *International Conference on Computer-Aided Design 07, IEEE/ACM*. Piscataway, NJ, USA: IEEE Press, 2007, pp. 598603.

[8] R. Teodorescu and J. Torrellas, "Variation-aware application scheduling and power management for chip multiprocessors," *SIGARCH Comput. Archit. News*, vol. 36, no. 3, pp. 363374, 2008.

[9] C. JALIER and D. LATTARD, Architecture and IC Design, Embedded Software Annual Reasearch Report 2010 (page 24), CEA LETI DACLE, France.

[10] Nelson H. F. Beebe, "High-Performance Matrix Multiplication," Center for Scientific Computing, Department of Mathematics, University of Utah, USA, 1990.

[11] A. Tiwari and J. Torrellas, "Facelift: Hiding and Slowing Down Aging in Multicores," *Proc. IEEE/ACM Intl Symp. Microarchitecture*, pp. 129-140, 2008.