Parallelizing Online Error Detection in Many-core Microprocessor Architectures

Manolis Kaliorakis[1], Mihalis Psarakis[2], Nikos Foutris[1], Dimitris Gizopoulos[1]

[1] Dept. of Informatics & Telecomm., University of Athens, Greece
{manoliskal, nfoutris, dgizop}@di.uoa.gr

[2] Dept. of Informatics, University of Piraeus, Greece
mpsarak@unipi.gr

***Abstract*—Many-core processors encounter significant dependability issues due to their design complexity and the inherent unreliability of the underlying deep nanometer technologies. Online functional (software-based) testing, well-established in single processor cores, can provide a low-cost error detection solution for many-core architectures as well. We study the parallelization of online test programs to reduce test execution time. We evaluate the proposed test program parallelization method on a popular research many-core architecture, Intel's Single-chip Cloud Computer (SCC), showing up to 47.6X speedup.**

I. Introduction

Functional or software-based testing has gained increasing acceptance for off-line and on-line error detection in uniprocessor architectures [1], [2], [3], while recent approaches [4], [5], [6] have studied its feasibility in multicore processor architectures. An online error detection strategy for many-core architectures must exploit the massive execution parallelism and reduce the duration of test program execution. Otherwise, the overall execution time will excessively scale with the number of cores. In this paper, we study the *parallelization of online test programs* in many-core microprocessor architectures.

Online error detection in many-core architectures *radically differs* from conventional parallel programming problems, since the entire test program must be simply applied to *every* core. Therefore, parallelizing the test programs seems to be effortless: all processor cores must run the test program in parallel and there is no need for communication. However, the experimental results we performed on a popular many-core microprocessor chip (Intel's SCC) show that the straightforward parallel application of typical memory-intensive test programs achieves *very low speedup* (i.e. less than 11X in a 48-core architecture), which is far below the theoretical maximum speedup in such an architecture. This is mainly due to fact that the memory-intensive test programs running in parallel in all processor cores push the memory system and the interconnection network to their limits exceeding the maximum memory bandwidth they can deliver.

In this paper, we propose an effective parallelization method which aims to *accelerate the test preparation phase* of the test programs. The key idea of the proposed method is that every processor core produces a subset of the test patterns set and then distributes it to the other processors through the high-bandwidth on-chip message-passing interconnection network. This parallelization method significantly speeds up the memory-intensive test programs. Furthermore, we propose a simple scheduling method that runs in parallel the memory-intensive and CPU-intensive test programs to reduce the traffic congestion in the interconnection network. The proposed methods are demonstrated on Intel's Single-chip Cloud Computer (SCC) platform.

II. Online Error Detection in a Many-core Architecture

There are two approaches for the online execution of a test program in a many-core architecture: (a) *non-intrusive approach*: each core runs the test programs individually during its idle periods and (b) *intrusive approach*: groups of cores or all cores together are periodically set out-of-service for normal workload execution to run simultaneously the test programs. In this paper, we consider the application of an intrusive online error detection approach in a many-core architecture, Intel's *Single-chip Cloud Computer* (*SCC*) [7]. SCC contains 48 in-order Pentium cores (organized in 24 tiles, with two cores per tile) connected in a 6x4 2D mesh network. It integrates four DDR3 memory controllers and hardware-support for message passing.

For our experimentation, we developed two functional test programs for the SCC chip with different characteristics which represent typical test program formats used in functional online testing: (a) *Load-Apply-Accumulate* (LAA) test program: this memory-intensive test program applies ATPG-generated test patterns (or pre-computed by other means) stored in the off-chip main memory (DRAM). It first reads two test vectors from memory, applies the target instruction and accumulates the responses. (b) *Linear-Feedback-Shift-Register* (LFSR) test program: this CPU-intensive test program applies pseudorandom patterns generated by an LFSR. Similarly to LAA program, the LFSR program first generates two pseudorandom test patterns, applies the target instruction and accumulates the responses.

III. Test Program Parallelization

Among the three basic phases of a functional test program, i.e. test *preparation*, test *application* and test response *compaction*, the last two cannot be parallelized because each test pattern must be applied to every core and the core's response must be compacted separately. The test preparation phase (the on-chip production of all patterns that must be eventually applied to each core) is the only that can be performed *only once* for the entire chip. Thus, the test preparation task can be divided and parallelized in a many-core processor architecture balancing

the test preparation workload between the cores to achieve the maximum speedup.

Before focusing on the parallelization of the test preparation phase, we carried out a set of experiments to identify the most efficient way to load the test vectors of a memory intensive test program (i.e. LAA) in the processing cores. Based on the experimental results and a theoretical performance analysis of the memory subsystem we concluded that an efficient parallelization method should use *message passing* instead of the off-chip DRAM to share test data between cores. Based on this observation, we propose an effective method for the parallelization of the LAA test program. The test patterns are divided into 48 segments each one assigned to the private memory region of a core. The LAA test program is divided into two phases. First, all cores load in parallel the test patterns from their *private* memories, apply the tests on themselves and accumulate the responses. Subsequently, each core copies the corresponding test patterns from the local Message Passing Buffers (MPBs) of the other 47 cores and applies/accumulates the tests. It is essential that in each cycle of the second phase each MPB serves the memory requests of only one core in order to limit the traffic congestion in the mesh and the routers and thus reduce execution time.

Regarding the LFSR test program, its test preparation phase cannot be parallelized in a more efficient way since the time each core requires to run the LFSR code to generate a certain number of test patterns is shorter than the time to copy these test patterns from the local MPB of an adjacent core. Thus, a second improvement in the parallelization of the entire online test program (consisting of LAA and LFSR programs) could be the parallel execution of memory-intensive test programs (LAA tests) and CPU-intensive test programs (LFSR tests). The rationale of the proposed method is to balance the memory-intensive and the CPU-intensive test programs to avoid high traffic congestion in the mesh and the routers.

Table I presents the execution times of the LAA and LFSR test programs running in all 48 cores for the Serial, Naïve Parallel and Proposed Parallel approaches. LAA test program applies 384KB test data, while LFSR program applies 10 times more pseudorandom test data, i.e. 3840KB. In the case of the memory-intensive LAA test program the speedup is less than 11X compared to the serial approach, while in the case of the CPU-intensive LFSR test program the speedup is close to the theoretical maximum. In the naïve method, all processor cores execute in parallel the same test program without communicating each other. After the normal workload has been paused in all processor cores, the cores are synchronized to execute the test program in parallel. Upon completion of the online test phase, normal workload is resumed in all cores. In the serial method, the processor cores execute the test program one after the other, while the remaining cores remain idle. In both serial and naïve parallel methods, test data are stored in the shared memory of the SCC in order to be directly accessible by all cores. Note that Intel supports the configuration of shared memory either as cacheable or non-cacheable memory. The experimental results shown in Table I for the serial and naïve parallel approaches are for cacheable memory configuration. The proposed parallel method achieves 38.4X speedup for the LAA test program compared to the serial approach. Applying the proposed parallel approach for the LAA and the naïve parallel approach for the LFSR test program results in 44.4X speedup. Furthermore, the proposed hybrid parallel LAA/LFSR method achieves a further 7% improvement and up to 47.6X speedup over the serial approach.

TABLE I. Execution times of the test programs for the Serial, Naïve Parallel and Proposed Parallel approaches (Execution times are in 10^6 cycles. Numbers in parentheses denote the speedup against the serial execution)

Test Program	Serial	Naïve Parallel	Proposed Parallel
LAA	145.8	14.0 (10.4X)	3.8 (38.4X)
LFSR	1047.8	23.1 (45.4X)	-
LAA+LFSR	1193.6	37.1 (32.2X)	26.9 (44.4X) 25.1 (47.6X)

IV. Future Work

We currently investigate the parallelization of the non-intrusive on-line error detection approach. In this case, the number of processor cores running in parallel the test program is not predetermined and consequently, the partitioning of the test patterns into cores and their distribution through the message passing buffers are resolved dynamically. Every core has access to all test patterns and a test scheduler monitors the cores, assigns the task of loading test patterns into the idle cores and determines the optimal routes for the distribution of test patterns through the MPBs.

Acknowledgment

This research is co-financed by the European Union and Greek national funds through the project “Hardware and Software Techniques for Multi/Many-core Processor Architectures Reliability Enhancement (Thalis/HOLISTIC)”. The work is also funded by a research gift from Cisco Research on online error detection for multicore microprocessor architectures. It is also supported by Intel’s MARC Program providing access to the Single-chip Cloud Computer (SCC) chip.

References

[1] L.Chen, S.Ravi, A.Raghunathan, S.Dey, “A Scalable Software-Based Self-Test Methodology for Programmable Processors”, IEEE/ACM Design Automation Conference (DAC), pp. 548-553, 2003.

[2] S.Gurumurthy, S.Vasudevan and J.Abraham, “Automatic generation of instruction sequences targeting hard-to-detect struc-tural faults in a processor”, in Proc. IEEE Intl. Test Conf., paper 27.3, 2006.

[3] M.Psarakis, D.Gizopoulos, E.Sanchez, M.S.Reorda, "Microprocessor Software-Based Self-Testing," IEEE Design & Test of Computers, vol.27, no.3, pp.4,19, May-June 2010.

[4] A.Apostolakis, D.Gizopoulos, M.Psarakis, A.Paschalis, "Software-Based Self-Testing of Symmetric Shared-Memory Multiprocessors," IEEE Transactions on Computers, vol. 58, no. 12, pp. 1682-1694, 2009.

[5] N.Foutris, et al., "Mt-sbst: Self-test optimization in multithreaded multicore architectures, Proc. IEEE Intl. Test Conf., pp. 1-10, 2010.

[6] M.A.Skitsas, C.A.Nicopoulos and M.K.Michael, "Toward Selective Software-Based Self-Testing in Multi-Core Microprocessors," in Proc. 1st MEDIAN Workshop, pp. 71-75, 2012.

[7] SCC Programmer's Guide, rev. 1.0, Jan. 2012.