

MO901R/MC039A - Seminário de Computação - Seminários em Arquitetura de Computadores e Compiladores

INFORMAÇÕES GERAIS

Professor: [Rodolfo Jardim de Azevedo](#) (email)

Aula: Seg: 10h-12h

Atendimento: Procurar diretamente na minha sala.

Sala: 316

AValiação

- A avaliação será baseada nas apresentações, presença e participação em sala
- **Qualquer tentativa de fraude implicará em Nmo901/Nmc039 = 0 para todos os envolvidos**
- $NotaFinal = 0,5 * NotaApresentação + 0,5 * NotaParticipação$
- Conceitos
 - A: $8,5 \leq Nmo901 \leq 10,0$
 - B: $7,0 \leq Nmo901 < 8,5$
 - C: $5,0 \leq Nmo901 < 7,0$
 - D: $Nmo901 < 5,0$

VEJA TAMBÉM

- Alunos matriculados
- Acompanhamento e notas
- Wiki

Avisos

08/08 Artigos alocados aos apresentadores. Se seu nome não está na listagem abaixo, entre em contato urgentemente comigo.

08/08 Verifiquem as datas importantes no [calendário de pós-graduação](#) e no [calendário de graduação](#).

Objetivos

Ler, apresentar e discutir artigos recentes da área de Arquitetura de Computadores e Compiladores

Calendário

As referências para os artigos estão no final desta página.

Data	Artigo	Apresentadores
08/08	Apresentação, seleção dos artigos	Rodolfo
15/08	Power management of online data-intensive services	Leonardo Piga
22/08	Specifying and Checking Semantic Atomicity for Multithreaded Programs	Felipe Silva
29/08	Dynamic Filtering: Multi-Purpose Architecture Support for Language Runtime Systems	Raphael
05/09	PetaBricks: a language and compiler for algorithmic choice.	Pedro
12/09	Scalable Speculative Parallelization on Commodity Clusters	Paulo
19/09	Improving the Performance of Trace-based Systems by False Loop Filtering	Julio e Leonardo
26/09	Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU	Alexandre e Daniel
03/10	DRFx: A Simple and Efficient Memory Model for Concurrent Programming Languages	Gabriel e Sérgio
10/10	ASF: AMD64 Extension for Lock-Free Data Structures and Transactional Memory	Alexandro

17/10	Dark silicon and the end of multicore scaling	Caio Hoffman e Tiago
24/10	Conservation cores: reducing the energy of mature computations	Valdiney
31/10	RCDC: A Relaxed-Consistency Deterministic Computer	Raoni
07/11	Looking back on the language and hardware revolutions: measured power, performance, and scaling	Rubens e Eduardo
14/11	Feriado	
21/11	Toward a Practical Methodology to Collect and Use Multiple Profiles	Rafael Auler
28/11	Task Superscalar: An Out-of-Order Task Pipeline	Cesar e Mario
Artigos		

Todos os alunos devem ler todos os artigos selecionados abaixo. Além disto, para cada artigo, uma dupla de alunos fará uma apresentação em sala seguida de tópicos para discussão.

1. Dynamic Filtering: Multi-Purpose Architecture Support for Language Runtime Systems. Tim Harris, Saša Tomić, Adrián Cristal, Osman Unsal, ASPLOS 2010

This paper introduces a new abstraction to accelerate the read-barriers and write-barriers used by language runtime systems. We exploit the fact that, dynamically, many barrier executions perform checks but no real work -- e.g., in generational garbage collection (GC), frequent checks are needed to detect the creation of inter-generational references, even though such references occur rarely in many workloads. We introduce a form of dynamic filtering that identifies redundant checks by (i) recording checks that have recently been executed, and (ii) detecting when a barrier is repeating one of these checks. We show how this technique can be applied to a variety of algorithms for GC, transactional memory, and language-based security. By supporting dynamic filtering in the instruction set, we show that the fast-paths of these barriers can be streamlined, reducing the impact on the quality of surrounding code. We show how we accelerate the barriers used for generational GC and transactional memory in the Bartok research compiler. With a 2048-entry filter, dynamic filtering eliminates almost all the overhead of the GC write-barriers. Dynamic filtering eliminates around half the overhead of STM over a non-synchronized baseline -- even when used with an STM that is already designed for low overhead, and which employs static analyses to avoid redundant operations.

2. Improving Software Diagnosability via Log Enhancement. Ding Yuan, Jing Zheng, Soyeon Park, Yuanyuan Zhou, Stefan Savage University of Illinois, Urbana-Champaign and University of California, San Diego, 2University of California, San Diego - ASPLOS 2011

Diagnosing software failures in the field is notoriously difficult, in part due to the fundamental complexity of trouble-shooting any complex software system, but further exacerbated by the paucity of information that is typically available in the production setting. Indeed, for reasons of both overhead and privacy, it is common that only the run-time log generated by a system (e.g., syslog) can be shared with the developers. Unfortunately, the ad-hoc nature of such reports are frequently insufficient for detailed failure diagnosis. This paper seeks to improve this situation within the rubric of existing practice. We describe a tool, LogEnhancer that automatically "enhances" existing logging code to aid in future post-failure debugging. We evaluate LogEnhancer on eight large, real-world applications and demonstrate that it can dramatically reduce the set of potential root failure causes that must be considered during diagnosis while imposing negligible overheads.

3. RCDC: A Relaxed-Consistency Deterministic Computer, Joseph Devietti, Jacob Nelson, Tom Bergan, Luis Ceze, Dan Grossman, University of Washington - ASPLOS 2011

Providing deterministic execution significantly simplifies the debugging, testing, replication, and deployment of multithreaded programs. Recent work has developed deterministic multiprocessor architectures as well as compiler and runtime systems that enforce determinism in current hardware. Such work has incidentally imposed strong memory-ordering properties. Historically, memory ordering has been relaxed in favor of higher performance in shared memory multiprocessors and, interestingly, determinism exacerbates the cost of strong memory ordering. Consequently, we argue that relaxed memory ordering is vital to achieving faster deterministic execution. This paper introduces RCDC, a deterministic multiprocessor architecture that takes advantage of relaxed memory orderings to provide high-performance deterministic execution with low hardware complexity. RCDC has two key innovations: a hybrid HW/SW approach to enforcing determinism; and a new deterministic execution strategy that leverages data-race-free-based memory models (e.g., the models for Java and C++) to improve performance and scalability without sacrificing determinism, even in the presence of races. In our hybrid HW/SW approach, the only hardware mechanisms required are software-controlled store buffering and support for precise instruction counting;

we do not require speculation. A runtime system uses these mechanisms to enforce determinism for arbitrary programs. We evaluate RCDC using PARSEC benchmarks and show that relaxing memory ordering leads to performance and scalability close to nondeterministic execution without requiring any form of speculation. We also compare our new execution strategy to one based on TSO (total-store-ordering) and show that some applications benefit significantly from the extra relaxation. We also evaluate a software-only implementation of our new deterministic execution strategy.

4. Efficient Processor Support for DRFx, a Memory Model with Exceptions Abhayendra Singh, Daniel Marino, Satish Narayanasamy, Todd Millstein, Madanlal Musuvathi, ASPLOS 2011

A longstanding challenge of shared-memory concurrency is to provide a memory model that allows for efficient implementation while providing strong and simple guarantees to programmers. The C++0x and Java memory models admit a wide variety of compiler and hardware optimizations and provide sequentially consistent (SC) semantics for data-race-free programs. However, they either do not provide any semantics (C++0x) or provide a hard-to-understand semantics (Java) for racy programs, compromising the safety and debuggability of such programs. In earlier work we proposed the DRFx memory model, which addresses this problem by dynamically detecting potential violations of SC due to the interaction of compiler or hardware optimizations with data races and halting execution upon detection. In this paper, we present a detailed micro-architecture design for supporting the DRFx memory model, formalize the design and prove its correctness, and evaluate the design using a hardware simulator. We describe a set of DRFx-compliant complexity-effective optimizations which allow us to attain performance close to that of TSO (Total Store Model) and DRF0 while providing strong guarantees for all programs.

5. Specifying and Checking Semantic Atomicity for Multithreaded Programs. Jacob Burnim, George Necula, Koushik Sen, ASPLOS 2011

In practice, it is quite difficult to write correct multithreaded programs due to the potential for unintended and nondeterministic interference between parallel threads. A fundamental correctness property for such programs is atomicity---a block of code in a program is atomic if, for any parallel execution of the program, there is an execution with the same overall program behavior in which the block is executed serially. We propose semantic atomicity, a generalization of atomicity with respect to a programmer-defined notion of equivalent behavior. We propose an assertion framework in which a programmer can use bridge predicates to specify noninterference properties at the level of abstraction of their application. Further, we propose a novel algorithm for systematically testing atomicity specifications on parallel executions with a bounded number of interruptions---i.e. atomic blocks whose execution is interleaved with that of other threads. We further propose a set of sound heuristics and optional user annotations that increase the efficiency of checking atomicity specifications in the common case where the specifications hold. We have implemented our assertion framework for specifying and checking semantic atomicity for parallel Java programs, and we have written semantic atomicity specifications for a number of benchmarks. We found that using bridge predicates allowed us to specify the natural and intended atomic behavior of a wider range of programs than did previous approaches. Further, in checking our specifications, we found several previously unknown bugs, including in the widely-used `java.util.concurrent` library.

6. Improving the Performance of Trace-based Systems by False Loop Filtering, Hiroshige Hayashizaki, Peng Wu, Hiroshi Inoue, Mauricio Serrano, Toshio Nakatani, ASPLOS 2011

Trace-based compilation is a promising technique for language compilers and binary translators. It offers the potential to expand the compilation scopes that have traditionally been limited by method boundaries. Detecting repeating cyclic execution paths and capturing the detected repetitions into traces is a key requirement for trace selection algorithms to achieve good optimization and performance with small amounts of code. One important class of repetition detection is cyclic-path-based repetition detection, where a cyclic execution path (a path that starts and ends at the same instruction address) is detected as a repeating cyclic execution path. However, we found many cyclic paths that are not repeating cyclic execution paths, which we call false loops. A common class of false loops occurs when a method is invoked from multiple call-sites. A cycle is formed between two invocations of the method from different call-sites, but which does not represent loops or recursion. False loops can result in shorter traces and smaller compilation scopes, and degrade the performance. We propose false loop filtering, an approach to reject false loops in the repetition detection step of trace selection, and a technique called false loop filtering by call-stack-comparison, which rejects a cyclic path as a false loop if the call stacks at the beginning and the end of the cycle are different. We applied false loop filtering to our trace-based Java™ JIT compiler that is based on IBM's J9 JVM. We found that false loop filtering achieved an average improvement of 16% and 10% for the DaCapo benchmark when applied to two baseline trace selection algorithms, respectively, with up to 37% improvement for individual benchmarks. In the end, with false loop filtering, our trace-based JIT achieves a performance comparable to that of the method-based J9 JVM/JIT using the corresponding optimization level.

7. DRFx: A Simple and Efficient Memory Model for Concurrent Programming Languages. Daniel Marino (University of California, Los Angeles), Abhayendra Singh (University of Michigan, Ann Arbor), Todd Millstein (University of California, Los Angeles), Madanlal Musuvathi (Microsoft Research, Redmond), Satish Narayanasamy (University of Michigan, Ann Arbor). PLDI 2010.

The most intuitive memory model for shared-memory multithreaded programming is sequential consistency(SC), but it disallows the use of many compiler and hardware optimizations thereby impacting performance. Data-race-free (DRF) models, such as the proposed C++0x memory model, guarantee SC execution for data-race-free programs. But these models provide no guarantee at all for racy programs, compromising the safety and debuggability of such programs. To address the safety issue, the Java memory model, which is also based on the DRF model, provides a weak semantics for racy executions. However, this semantics is subtle and complex, making it difficult for programmers to reason about their programs and for compiler writers to ensure the correctness of compiler optimizations. We present the DRFx memory model, which is simple for programmers to understand and use while still supporting many common optimizations. We introduce a memory model (MM) exception which can be signaled to halt execution. If a program executes without throwing this exception, then DRFx guarantees that the execution is SC. If a program throws an MM exception during an execution, then DRFx guarantees that the program has a data race. We observe that SC violations can be detected in hardware through a lightweight form of conflict detection. Furthermore, our model safely allows aggressive compiler and hardware optimizations within compiler-designated program regions. We formalize our memory model, prove several properties about this model, describe a compiler and hardware design suitable for DRFx, and evaluate the performance overhead due to our compiler and hardware requirements.

8. PetaBricks: a language and compiler for algorithmic choice. Jason Ansel, Cy Chan, Yee Lok Wong, Marek Olszewski, Qin Zhao, Alan Edelman, and Saman Amarasinghe. 2009. In Proceedings of the 2009 ACM SIGPLAN conference on Programming language design and implementation (PLDI '09). ACM, New York, NY, USA, 38-49.

It is often impossible to obtain a one-size-fits-all solution for high performance algorithms when considering different choices for data distributions, parallelism, transformations, and blocking. The best solution to these choices is often tightly coupled to different architectures, problem sizes, data, and available system resources. In some cases, completely different algorithms may provide the best performance. Current compiler and programming language techniques are able to change some of these parameters, but today there is no simple way for the programmer to express or the compiler to choose different algorithms to handle different parts of the data. Existing solutions normally can handle only coarse-grained, library level selections or hand coded cutoffs between base cases and recursive cases. We present PetaBricks, a new implicitly parallel language and compiler where having multiple implementations of multiple algorithms to solve a problem is the natural way of programming. We make algorithmic choice a first class construct of the language. Choices are provided in a way that also allows our compiler to tune at a finer granularity. The PetaBricks compiler autotunes programs by making both fine-grained as well as algorithmic choices. Choices also include different automatic parallelization techniques, data distributions, algorithmic parameters, transformations, and blocking. Additionally, we introduce novel techniques to autotune algorithms for different convergence criteria. When choosing between various direct and iterative methods, the PetaBricks compiler is able to tune a program in such a way that delivers near-optimal efficiency for any desired level of accuracy. The compiler has the flexibility of utilizing different convergence criteria for the various components within a single algorithm, providing the user with accuracy choice alongside algorithmic choice.

9. Scalable Speculative Parallelization on Commodity Clusters. Hanjun Kim, Arun Raman, Feng Liu, Jae W. Lee, and David I. August. 2010. In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '10). ACM, New York, NY, USA, 38-49.

While clusters of commodity servers and switches are the most popular form of large-scale parallel computers, many programs are not easily parallelized for execution upon them. In particular, high inter-node communication cost and lack of globally shared memory appear to make clusters suitable only for server applications with abundant task-level parallelism and scientific applications with regular and independent units of work. Clever use of pipeline parallelism (DSWP), thread-level speculation (TLS), and speculative pipeline parallelism (Spec-DSWP) can mitigate the costs of inter-thread communication on shared memory multicore machines. This paper presents Distributed Software Multi-threaded Transactional memory (DSMTX), a runtime system which makes these techniques applicable to non-shared memory clusters, allowing them to efficiently address inter-node communication costs. Initial results suggest that DSMTX enables efficient cluster execution of a wider set of application types. For 11 sequential C programs parallelized for a 4-core 32-node (128 total core) cluster without shared memory, DSMTX achieves a geometric speedup of 49x. This compares favorably to the 15x speedup achieved by our implementation of TLS-only support for clusters.

10. ASF: AMD64 Extension for Lock-Free Data Structures and Transactional Memory. Jaewoong Chung, Luke Yen, Stephan Diestelhorst, Martin Pohlack, Michael Hohmuth, David Christie, and Dan Grossman. 2010. In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '43).

Advanced Synchronization Facility (ASF) is an AMD64 hardware extension for lock-free data structures and transactional memory. It provides a speculative region that atomically executes speculative accesses in the region. Five new instructions are added to demarcate the region, use speculative accesses selectively, and control the speculative hardware context. Programmers can use speculative regions to build flexible multi-word atomic primitives with no additional software support by relying on the minimum guarantee of available ASF hardware resources for lock-free programming. Transactional programs with high-level TM language constructs can either be compiled directly to the ASF code or be linked to software TM systems that use ASF to accelerate transactional execution. In this paper we develop an out-of-order hardware design to implement ASF on a future AMD processor and evaluate it with an in-house simulator. The experimental results show that the combined use of the L1 cache and the LS unit is very helpful for the performance robustness of ASF-based lock free data structures, and that the selective use of speculative accesses enables transactional programs to scale with limited ASF hardware resources.

11. Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. Victor W. Lee, Changkyu Kim, Jatin Chhugani, Michael Deisher, Daehyun Kim, Anthony D. Nguyen, Nadathur Satish, Mikhail Smelyanskiy, Srinivas Chennupaty, Per Hammarlund, Ronak Singhal, and Pradeep Dubey. 2010. SIGARCH Comput. Archit. News 38, 3 (June 2010), 451-460.

Recent advances in computing have led to an explosion in the amount of data being generated. Processing the ever-growing data in a timely manner has made throughput computing an important aspect for emerging applications. Our analysis of a set of important throughput computing kernels shows that there is an ample amount of parallelism in these kernels which makes them suitable for today's multi-core CPUs and GPUs. In the past few years there have been many studies claiming GPUs deliver substantial speedups (between 10X and 1000X) over multi-core CPUs on these kernels. To understand where such large performance difference comes from, we perform a rigorous performance analysis and find that after applying optimizations appropriate for both CPUs and GPUs the performance gap between an Nvidia GTX280 processor and the Intel Core i7-960 processor narrows to only 2.5x on average. In this paper, we discuss optimization techniques for both CPU and GPU, analyze what architecture features contributed to performance differences between the two architectures, and recommend a set of architectural features which provide significant improvement in architectural efficiency for throughput kernels.

12. Task Superscalar: An Out-of-Order Task Pipeline. Yoav Etsion, Felipe Cabarcas, Alejandro Rico, Alex Ramirez, Rosa M. Badia, Eduard Ayguade, Jesus Labarta, and Mateo Valero. 2010. In Proceedings of the 2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO '43). IEEE Computer Society, Washington, DC, USA, 89-100.

We present $\text{\textit{Task Super scalar}}$, an abstraction of instruction-level out-of-order pipeline that operates at the task-level. Like ILP pipelines, which uncover parallelism in a sequential instruction stream, task super scalar uncovers task-level parallelism among tasks generated by a sequential thread. Utilizing intuitive programmer annotations of task inputs and outputs, the task super scalar pipeline dynamically detects inter-task data dependencies, identifies task-level parallelism, and executes tasks out-of-order. Furthermore, we propose a design for a distributed task super scalar pipeline front end, that can be embedded into any many core fabric, and manages cores as functional units. We show that our proposed mechanism is capable of driving hundreds of cores simultaneously with non-speculative tasks, which allows our pipeline to sustain work windows consisting of tens of thousands of tasks. We further show that our pipeline can maintain a decode rate faster than 60ns per task and dynamically uncover data dependencies among as many as $\sim 50,000$ in-flight tasks, using 7MB of on-chip eDRAM storage. This configuration achieves speedups of 95–255x (average 183x) over sequential execution for nine scientific benchmarks, running on a simulated CMP with 256 cores. Task super scalar thus enables programmers to exploit many core systems effectively, while simultaneously simplifying their programming model.

13. Power management of online data-intensive services. David Meisner, Christopher M. Sadler, Luiz André Barroso, Wolf-Dietrich Weber, and Thomas F. Wenisch. 2011. In Proceeding of the 38th annual international symposium on Computer architecture (ISCA '11). ACM, New York, NY, USA, 319-330.

Much of the success of the Internet services model can be attributed to the popularity of a class of workloads that we call Online Data-Intensive (OLDI) services. These workloads perform significant computing over massive data sets per user request but, unlike their offline counterparts (such as MapReduce computations), they require responsiveness in the sub-second time scale at high request rates. Large search products, online advertising, and machine translation are examples of workloads in this class. Although the load in OLDI services can vary widely

during the day, their energy consumption sees little variance due to the lack of energy proportionality of the underlying machinery. The scale and latency sensitivity of OLDI workloads also make them a challenging target for power management techniques. We investigate what, if anything, can be done to make OLDI systems more energy-proportional. Specifically, we evaluate the applicability of active and idle low-power modes to reduce the power consumed by the primary server components (processor, memory, and disk), while maintaining tight response time constraints, particularly on 95th-percentile latency. Using Web search as a representative example of this workload class, we first characterize a production Web search workload at cluster-wide scale. We provide a fine-grain characterization and expose the opportunity for power savings using low-power modes of each primary server component. Second, we develop and validate a performance model to evaluate the impact of processor- and memory-based low-power modes on the search latency distribution and consider the benefit of current and foreseeable low-power modes. Our results highlight the challenges of power management for this class of workloads. In contrast to other server workloads, for which idle low-power modes have shown great promise, for OLDI workloads we find that energy-proportionality with acceptable query latency can only be achieved using coordinated, full-system active low-power modes.

14. Dark silicon and the end of multicore scaling. Hadi Esmaeilzadeh, Emily Blem, Renee St. Amant, Karthikeyan Sankaralingam, and Doug Burger. 2011. In Proceeding of the 38th annual international symposium on Computer architecture (ISCA '11). ACM, New York, NY, USA, 365-376.

Since 2005, processor designers have increased core counts to exploit Moore's Law scaling, rather than focusing on single-core performance. The failure of Dennard scaling, to which the shift to multicore parts is partially a response, may soon limit multicore scaling just as single-core scaling has been curtailed. This paper models multicore scaling limits by combining device scaling, single-core scaling, and multicore scaling to measure the speedup potential for a set of parallel workloads for the next five technology generations. For device scaling, we use both the ITRS projections and a set of more conservative device scaling parameters. To model single-core scaling, we combine measurements from over 150 processors to derive Pareto-optimal frontiers for area/performance and power/performance. Finally, to model multicore scaling, we build a detailed performance model of upper-bound performance and lower-bound core power. The multicore designs we study include single-threaded CPU-like and massively threaded GPU-like multicore chip organizations with symmetric, asymmetric, dynamic, and composed topologies. The study shows that regardless of chip organization and topology, multicore scaling is power limited to a degree not widely appreciated by the computing community. Even at 22 nm (just one year from now), 21% of a fixed-size chip must be powered off, and at 8 nm, this number grows to more than 50%. Through 2024, only 7.9x average speedup is possible across commonly used parallel workloads, leaving a nearly 24-fold gap from a target of doubled performance per generation.

15. Looking back on the language and hardware revolutions: measured power, performance, and scaling. Hadi Esmaeilzadeh, Ting Cao, Yang Xi, Stephen M. Blackburn, and Kathryn S. McKinley. 2011. In Proceedings of the sixteenth international conference on Architectural support for programming languages and operating systems (ASPLOS '11). ACM, New York, NY, USA, 319-332.

This paper reports and analyzes measured chip power and performance on five process technology generations executing 61 diverse benchmarks with a rigorous methodology. We measure representative Intel IA32 processors with technologies ranging from 130nm to 32nm while they execute sequential and parallel benchmarks written in native and managed languages. During this period, hardware and software changed substantially: (1) hardware vendors delivered chip multiprocessors instead of uniprocessors, and independently (2) software developers increasingly chose managed languages instead of native languages. This quantitative data reveals the extent of some known and previously unobserved hardware and software trends. Two themes emerge. (I) Workload: The power, performance, and energy trends of native workloads do not approximate managed workloads. For example, (a) the SPEC CPU2006 native benchmarks on the i7 (45) and i5 (32) draw significantly less power than managed or scalable native benchmarks; and (b) managed runtimes exploit parallelism even when running single-threaded applications. The results recommend architects always include native and managed workloads when designing and evaluating energy efficient hardware. (II) Architecture: Clock scaling, microarchitecture, simultaneous multithreading, and chip multiprocessors each elicit a huge variety of power, performance, and energy responses. This variety and the difficulty of obtaining power measurements recommends exposing on-chip power meters and when possible structure specific power meters for cores, caches, and other structures. Just as hardware event counters provide a quantitative grounding for performance innovations, power meters are necessary for optimizing energy.

16. Conservation cores: reducing the energy of mature computations. Ganesh Venkatesh, Jack Sampson, Nathan Goulding, Saturnino Garcia, Vladyslav Bryksin, Jose Lugo-Martinez, Steven Swanson, and Michael Bedford Taylor. 2010. SIGPLAN Not. 45, 3 (March 2010), 205-218.

Growing transistor counts, limited power budgets, and the breakdown of voltage scaling are currently conspiring to create a utilization wall that limits the fraction of a chip that can run at full speed at one time. In this regime, specialized, energy-efficient processors can increase parallelism by reducing the per-computation power requirements and allowing more computations to execute under the same power budget. To pursue this goal, this paper introduces conservation cores. Conservation cores, or c-cores, are specialized processors that focus on reducing energy and energy-delay instead of increasing performance. This focus on energy makes c-cores an excellent match for many applications that would be poor candidates for hardware acceleration (e.g., irregular integer codes). We present a toolchain for automatically synthesizing c-cores from application source code and demonstrate that they can significantly reduce energy and energy-delay for a wide range of applications. The c-cores support patching, a form of targeted reconfigurability, that allows them to adapt to new versions of the software they target. Our results show that conservation cores can reduce energy consumption by up to 16.0x for functions and by up to 2.1x for whole applications, while patching can extend the useful lifetime of individual c-cores to match that of conventional processors.

17. Toward a Practical Methodology to Collect and Use Multiple Profiles, Paul Berube, Adam Preuss, José Nelson Amaral, International Conference on Performance Engineering, Karlsruhe, Germany, March, 2011.

Security testing is an important step in the lifetime of both newly-designed and existing systems. Different methodologies exist to guide testers to the selection, design, and implementation of the most appropriate testing procedures for various contexts. Typically, each methodology stems from the specific needs of a particular category of actors, and consequently is biased towards some aspect of peculiar interest to them. This work compares the most commonly adopted methodologies to point out their strengths and weaknesses, and, building on the results of the performed analysis, proposes a path towards the definition of an integrated approach, by defining the characteristics that a new methodology should exhibit in order to combine the best aspects of the existing ones.