

# Automatic Application-Specific Customization of Multi-Processor Microarchitecture \*

Extended Abstract

Shobana Padmanabhan, Ron K. Cytron, and John W. Lockwood

shobana@arl.wustl.edu, cytron@acm.org, lockwood@arl.wustl.edu

Department of Computer Science and Engineering

Washington University in St. Louis

## Abstract

Reconfigurable logic devices offer new and interesting opportunities for *application-specific customization and processor configuration*. However, the most compelling optimization problems in this area are NP-hard.

In our preliminary work, we customized the *microarchitecture* of a single processor core for a target application by viewing it as a multi-objective constrained integer nonlinear optimization problem. Our results demonstrated the validity, feasibility, and scalability of that approach [9]. Here, we propose extensions to customize the microarchitecture of multiple processor cores.

## 1 Introduction

With the number of cores on a chip expected to double with every silicon generation [2], systems with thousands of cores per chip have become a research target. With **Field Programmable Gate Arrays** (FPGAs) achieving sufficient density, embedded applications with tight runtime and resource restrictions are increasingly relying on **Multi-Processor system-on-chip** (MPSoC) approaches [12], for emulation or deployment.

Multicores of vendor-specific architectures (such as MicroBlaze [14]) are readily available. However, with the inclusion of open-source processor implementations such as LEON [5] in high-end projects

such as RAMP [11], commodity architectures (such as SPARC) are candidates for multicore deployment [4]. Standard architectures enable developers to leverage existing operating systems [13] and tool chains, such as the Linux and the GNU compiler and debugger.

**Opportunities** Because FPGAs are less dense than their **Application-Specific Integrated Circuit** (ASIC) cousins, naive instantiation of a stock, multicore architecture would not be competitive. However, FPGAs offer new and interesting opportunities for *application-specific customization and configuration of the processors*, in particular processor *microarchitecture customization*, which is the subject of our research. There are hundreds of microarchitecture features in each processor that can be customized to favor a given application. Examples of such features include cache size, cache associativity, and multiplier implementation.

With MPSoCs, in addition to customizing each processor's microarchitecture for a task, we can also consider customization of microarchitecture across processors for the entire application, in a system-wide manner.

**Challenges** Unfortunately, optimizing the use of reconfigurable logic for microarchitecture features across multiple processors is an *NP-hard* problem. Exhaustive approaches are infeasible: in LEON2, there are 190 customizable feature values, yielding some  $5 \times 10^{24}$  configurations.

The second challenge is with respect to measuring

---

\*Sponsored by National Science Foundation under grant ITR-0313203.

costs. Estimation can be inaccurate, causing inaccurate optimization. Therefore, we measure *actual* costs, for all the dimensions that are being optimized and constrained (currently, application runtime and FPGA resources).

With multi-processor cores and long running applications, *profiling* the application on every possible configuration may not be feasible at times. Instead, we propose correlating application runtime to microarchitecture features from a single application execution. FPGA *resource usages* are measured by actually building processor configurations from the source VHDL. Each build is very time-consuming, on the order of 30 minutes, even on modern computers.

These two challenging makes the customization *harder* than a traditional optimization problem because they make it infeasible to build an exact model and search for the best solution exhaustively.

**Other considerations** On MPSoC, logic can also be devoted to realization of application-specific custom circuitry to outperform high-end general-purpose processors on compelling applications [3, 8]. Some vendors also offer fixed realization of a standard **Instruction Set Architecture** (ISA) on MPSoC (such as Virtex 4-FX with its embedded PPC processor [14]). While some of our proposed research applies to the reconfigurable logic around the fixed ISA, a soft-core processor offers more opportunities for optimization, as we discuss below.

**Related work** There is no existing research that considers customization of all microarchitecture features as we do. The relevance of similar approaches is summarized elsewhere [9] due to space constraints.

## 2 Problem Formulation

In a preliminary work [9], we developed an automatic technique that can be used by application developers to trade resources (FPGA resources, power consumption, energy dissipation, and so on) for performance of the target application. In this section, we first summarize that technique and then propose

extensions for multi-processor microarchitecture optimization.

**Research platform** For our preliminary work, we used our liquid architecture platform [10] and its hardware-based, cycle-accurate, non-intrusive profiler [7]. For multi-processor microarchitecture customization, we plan to use RAMP [11] as our platform and extend our profiler as needed.

### 2.1 Single-core Microarchitecture Customization

We solved the problem of application-specific customization of single core microarchitecture as a *constrained integer nonlinear optimization* problem.

From the stock processor configuration, we change microarchitecture features one at a time and generate the corresponding processor configurations. We then construct the search space for exploration by approximating the remaining configurations based on the generated ones [9].

**Problem formulation** The objective of the optimization is to *minimize* application runtime or resource as desired by application developers. By using appropriate *weights* in the following function, developers can choose the tradeoff between application performance and resources that they wish to obtain. The costs below are normalized [9].

$$\text{Minimize } \sum_{i=1}^n [w_1(\text{timecost}_i x_i) + w_2(\text{FPGAcost}_i x_i) + \dots]$$

$x_i$  is a binary variable indicating the presence or absence of  $i^{\text{th}}$  configuration (corresponding to change in one microarchitecture feature);  $w_i$ 's are independent.

Because we use binary variables to represent the different possible values of a microarchitecture feature, we use constraints to select only one variable (value) per feature. Constraints are also used to ensure that the hardware resource limits are not exceeded. As an example, the FPGA resource constraint is presented below.

$$\sum_{i=1}^n \text{FPGAcost}_i x_i \leq \text{AvailableFPGA}$$

There are also constraints to ensure that the resulting processor configuration is valid. For instance, for the LEON processor, the cache replacement policy of **Least Recently Replaced** (LRR) is possible only when the number of cache sets is 2.

## 2.2 Extensions for Multi-Processor Microarchitecture Customization

Generally, there is a conflict between using chip area for more cores and for “bigger” microarchitecture features. For the purposes of this paper, the number of processor cores, along with the mapping of tasks onto them, is provided by the application developer; the number of cores remains fixed during optimization; each core supports the execution of exactly one task of the application; and each core hosts the same processor architecture.

We then vary the microarchitecture of each processor, aiming for high application performance but constrained by available chip resources.

We further assume that task execution time dominates wait time for data (if any), communicating processor-cores are interconnected by point-to-point buses of fixed width, and FPGA resources are available for use in optimizing the cores’ microarchitectures. We focus on streaming applications.

Data gathering for optimization begins with generating a sequence of processor configurations. Each configuration is designed to evaluate the benefits of a given microarchitecture feature. The processor configurations are dependent only on the microarchitecture features and not the application itself. Since all the processor cores are homogeneous to begin with, the set of configurations that we generate for each processor core is the same. During processor configuration generation, we track FPGA resource usage cost of each configuration.

We then execute each application task on every processor configuration that we generated and measure the corresponding application runtime cost, power consumption cost, energy dissipation cost, and other costs that we are interested in optimizing or constraining. For our work here, we focus on minimizing application runtime under resource constraints and hence we only measure task runtime costs here.

**Our heuristic** Performance of streaming applications is determined by the *bottleneck* task, which is the longest running task in the pipeline. One way to improve performance of such applications would be to speedup the bottleneck task through microarchitecture customization. This would be done iteratively as described below. First, find the longest running task across processor cores. Next, customize microarchitecture of its processor core with the objective of minimizing the task’s runtime, using additional resources as needed. This customization is similar to our heuristic to optimize single-core microarchitecture summarized in Section 2.1 with the difference of using only the current task runtime cost in the objective function:

$$\text{Minimize } \sum_{i=1}^n \text{timecost}_i x_i$$

$x_i$  is a binary variable indicating the presence or absence of  $i^{\text{th}}$  configuration.

Once we get the optimal processor microarchitecture for this task, we recompute the task with the longest runtime and the still available FPGA resources. We repeat the single-core optimization for this task as explained above.

We iterate the optimization until we gain no further improvement in task execution time, run out of resources, reach the target performance or the timeout value specified by the application developer. The multi-processor microarchitecture configuration we have at that point is the recommended configuration. This process is depicted in Figure 1.

## 2.3 Addressing Simplifications

Once we solve the simplified version of the problem of multi-processor microarchitecture optimization as described in Section 2.2, we will address the simplifications we made.

In addition to letting application developers identify and input application parallelism and task mapping to our heuristic, we will attempt to automate it using dependence analysis [1]. Alternatively, we consider timestamping message calls and inferring, not only parallelism, but also inter-processor communication bandwidth. The latter can then be used to optimize processor interconnects besides processor microarchitecture. Instead, we could also use

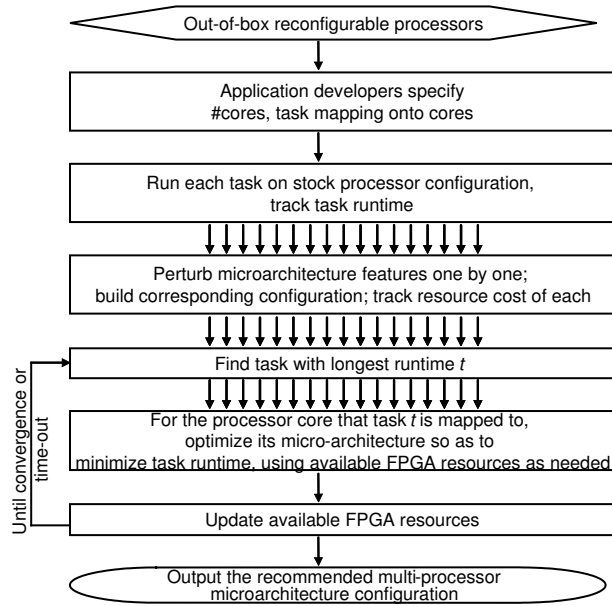


Figure 1: Process for multi-processor microarchitecture customization.

RAMP Description Language (RDL) [6] to describe the processor interconnects and include the interconnect configurations in the optimization.

While profiling an application’s runtime, in our preliminary work, we measured the effect of a given configuration only in terms of the program’s overall execution time. For multi-processor optimization however, we will attribute and correlate application runtime to microarchitecture features. For example, where we formerly measured the benefits of features such as cache organization in terms of the application’s overall execution time, we will instead measure the number of cache misses. Separate accounting of microprocessor artifacts such as cache misses and branch mispredictions allows us to infer more from a single execution of a program than if we only see the application’s execution time. The result will be fewer program executions, since a single run can provide information about the effectiveness of multiple microarchitecture features.

## References

[1] A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison Wesley, 2007.

[2] K. Asanovic, R. Bodik, B. Catanzaro, J. Gebis, P. Husbands, K. Keutzer, D. Patterson, W. Plishker, J. Shalf, S. Williams, and K. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. In *EECS Technical Report UCB/EECS-2006-183*, December 2006.

[3] B. C. Brodie, R. K. Cytron, and D. E. Taylor. A scalable architecture for high-throughput regular-expression pattern matching. In *ISCA '06: Proceedings of the 33rd International Symposium on Computer Architecture*, pages 191–202, Washington, DC, USA, 2006. IEEE Computer Society.

[4] J. Gaisler and E. Catovic. Multi-core processor based on LEON3-FT IP core. In *Proceedings of the DASIA Conference: Special Session on The LEON Processor*, May 2006.

[5] Gaisler Research. LEON Specification. <http://www.gaisler.com/doc/leon2-1.0.21-xst.pdf>, 2003.

[6] G. Gibeling. RAMP Description Language (RDL): Applications and Debugging, January 2006. RAMP Retreat.

[7] R. Hough, P. Jones, S. Friedman, R. Chamberlain, J. Fritts, J. Lockwood, and R. Cytron. Cycle-accurate microarchitecture performance evaluation. In *Proc. of Workshop on Introspective Architecture*, Feb. 2006.

[8] P. Krishnamurthy, J. Buhler, R. D. Chamberlain, M. A. Franklin, K. Gyang, and J. Lancaster. Biosequence similarity search on the Mercury system. In *Proceedings of the 15th IEEE International Conference on Application-Specific Systems, Architectures, and Processors*, pages 365–75, 2004.

[9] S. Padmanabhan, R. K. Cytron, R. D. Chamberlain, , and J. W. Lockwood. Automatic Application-Specific Microarchitecture Reconfiguration. In *Proc. of IEEE International Parallel & Distributed Processing Symposium (IPDPS)*. 13th Reconfigurable Architectures Workshop (RAW), April 2006.

[10] S. Padmanabhan, P. Jones, D. V. Schuehler, S. J. Friedman, P. Krishnamurthy, H. Zhang, R. Chamberlain, R. K. Cytron, J. Fritts, and J. W. Lockwood. Extracting and improving microarchitecture performance on reconfigurable architectures. *International Journal of Parallel Programming*, 33(2–3):115–136, June 2005.

[11] D. Patterson, Arvind, K. Asanovic, D. Chiou, J. C. Hoe, C. Kozyrakis, S.-L. Lu, M. Oskin, J. Rabaey, and J. Wawrzynek. RAMP: Research Accelerator for Multiple Processors. In *Hot Chips*, August 2006.

[12] C. Rowen. The reinvention of the microprocessor. <http://www.tensilica.com/pdf/Reinvention>

[13] SnapGear. *SnapGear Embedded Linux Distribution*, 2006.

[14] Xilinx, Inc. <http://www.xilinx.com>.