Synthesis of Quality Configurable Systems

Seogoo Lee, Behzad Boroujerdian, Lizy K. John, and Andreas Gerstlauer
The University of Texas at Austin
sglee@utexas.edu, behzadboro@gmail.com, {ljohn,gerstl}@ece.utexas.edu

1. Introduction

Approximate computing is the idea of welcoming a system with lower quality in favor of higher performance or lower energy. Dependence of quality on various factors such as inputs or phases of the program makes finding a single solution that fits all scenario difficult. In addition, the desired quality of the system itself can vary at runtime depending on user constraints or environmental factors, such as the amount of battery left. These reasons motivate design of quality-adaptive systems that perform dynamic quality management akin to traditional power management. In such systems, quality trade-offs need to be carefully and systematically controlled. Such systems should be capable of navigating through various *quality states* (Q-states) depending on the environment and user needs. In general, operating points for a system are determined by settings of various hardware or software quality knobs. Q-states are those system's operating points that are located on a quality versus energy (and possibly performance) Pareto-front. An ideal system is capable of: 1) identifying such Q-states, and 2) swiftly reacting to the environment and the user needs by moving from one Q-state to another (agile quality configurability).

The Q-state of a system will be determined by the Q-states of tasks running on different cores. Today's heterogeneous systems are composed of various programmable processors and accelerators, where applications are partitioned into tasks and mapped onto processing elements (Fig.1). To enable quality adaptivity, hardware and software processing elements (PEs) need to be quality configurable. However, the large number of possible knobs poses a challenge in finding and configuring optimal operating points. Q-states associated with the possible mappings of tasks to PEs should be identified at design or compile time. At runtime, the operating system then picks the best mapping and Q-state for each task and core considering the quality goal and the overall state of the system.

In the realm of programmable processors, several approximate processors that expose quality knobs in the hardware through the ISA have been proposed [2, 8]. Similarly, purely software-based quality optimization techniques have been developed [7]. Identifying Q-states requires compilers that can generate different optimal binaries of a task for each desired quality level and input condition. By contrast, in the realm of accelerators, quality-configurable accelerators with different Q-states need to be synthesized. This ultimately requires high-level synthesis (HLS) tools that can automatically explore, determine, and generate quality-energy-performance optimal hardware configurations.

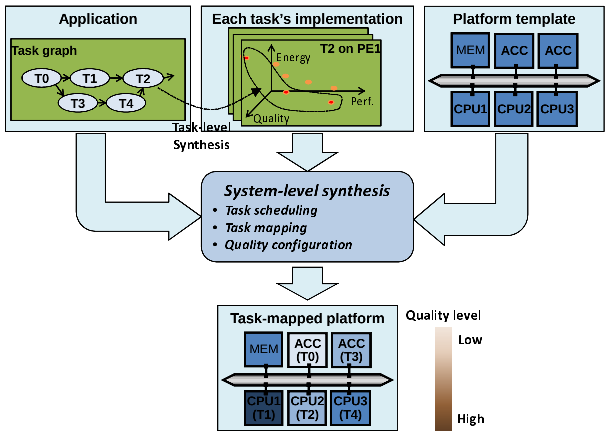


Figure 1: Quality-configurable system synthesis.

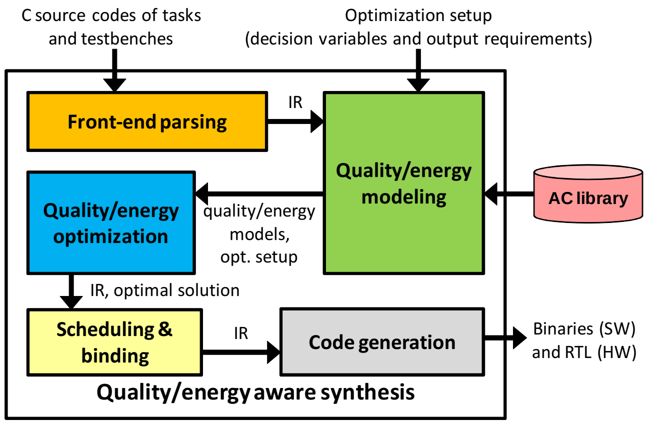
In the remainder of this paper, we present our initial strategies for synthesizing such quality-configurable, Pareto-optimal hardware and software building blocks out of which quality configurable systems can be realized.

2. Hardware and Software Synthesis

Fig. 2 shows our envisioned framework for synthesis of quality-configurable hardware and software implementations of a given task. We first parse a task's source code and generate an intermediate representation (IR). A heuristic using the IR, an optimization setup, and quality/energy models finds the Pareto-frontiers associated with a specific mapping of tasks to PEs. Code generation for both software and hardware follows. For software, the final outputs are multiple binaries corresponding to the different operating points on different processors. For the tasks mapped to dedicated accelerators, the outputs are register-transfer level (RTL) descriptions of the accelerators with various quality configurations.

2.1 Quality and Energy Modeling

Fast and accurate energy and quality estimation is a fundamental requirement for effective synthesis. Two common modeling approaches are simulation-based [1, 6] and analytical methods [3]. Simulation-based methods are generally applicable, but time-consuming, limiting their feasibility for


C source codes of tasks and testbenches
Optimization setup (decision variables and output requirements)
Front-end parsing
IR
Quality/energy modeling
AC library
Quality/energy optimization
quality/energy models, opt. setup
IR, optimal solution
Scheduling & binding
IR
Code generation
Binaries (SW) and RTL (HW)
Quality/energy aware synthesis
Figure 2: Overview of hardware and software synthesis.

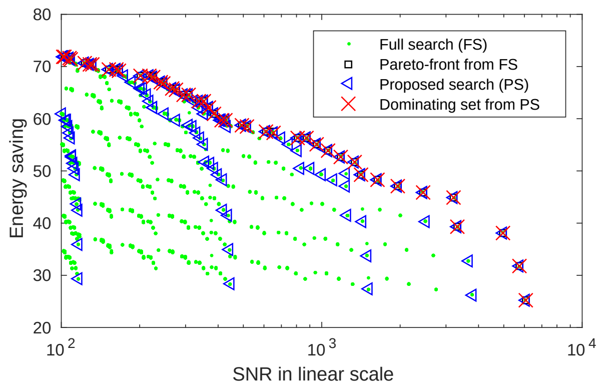


Figure 3: Optimization for Gaussian blur example.

synthesis. By contrast, analytical methods are limited in accuracy, but suited more for synthesis where fast execution is expected. We instead propose to follow a semi-analytical approach that is both fast and accurate [4].

For quality estimation, the main concerns are (1) modeling of individual operation-level error behavior and (2) propagation of operation-level errors through a program to estimate quality at primary outputs, including feedback loops. Our current approach relies on one-time simulation-based profiling to capture dependencies of errors on input statistics. We then analytically characterize the error distributions of various operations without any further simulation overhead. The use of general error distributions allows for translation into a variety of quality metrics, such as SNR or min/max error. Quality metrics at primary outputs of a task are then estimated from individual operation-level quality estimates and an appropriate error propagation model.

For energy estimation, basic hardware and software approximations generally result in dynamic power savings due to (1) reductions in computational complexity and hence switching activity, and (2) scaling of supply voltages to exploit reduced critical path delays and/or operate the design in an overscaling regime. These effects can be captured and modeled through appropriate pre-characterization of the hardware library, coupled with a proportional energy savings model.

2.2 Optimization Heuristic

Existing general optimization techniques for finding the Pareto-frontier associated with mapping a task into a hardware or software implementation, such as simulated annealing [4], greedy approach [6], and integer linear programming with sensitivity measurement [5], are either often suboptimal or high in complexity. Instead, we use a breadth-first search algorithm to successively explore the design space and find feasible and dominating implementation options. Our current optimization targets hardware accelerators with bit rounding as the approximation method. However, the heuristic is also applicable to software with different approximation methods. At every stage in our search algorithm, we evaluate all possible bit roundings, and only select feasible and dominating rounding values to be further considered in the next stage.

Fig. 3 shows our optimization process for the example of a Gaussian blur kernel in image processing. We compare our proposed search (PS) with an exhaustive full search (FS). The figure indicates both the optimality and the complexity of our heuristic. Each point in the figure maps to a combination of rounding values evaluated in the optimization process. The number of evaluated points of FS and PS shows differences in the complexity of our approach. The gain in complexity is 60× in this example. Furthermore, the dominating set found from our approach is very close to the Pareto-frontiers resulting from an exhaustive search.

2.3 Hardware and Software Synthesis

At design time, we find the Pareto frontier associated with various mappings of tasks to various PEs. On the programmable processor side, these Pareto frontiers are used to generate multiple binaries corresponding to different Q-states. On the accelerator side, the Pareto frontiers are fed into our high-level synthesis (HLS) framework to generate RTL description of quality-configurable accelerators.

3. Conclusion

In this paper, we presented our vision for quality-configurable synthesis of hardware and software in heterogeneous systems. We envision an approach that finds the optimal set of configurations for each PE when mapped with a task. The optimal sets enable the system-level optimization of task scheduling, mapping, and quality configuration, and provide us with Q-states, which can be dynamically adapted during runtime. We have developed an initial optimization framework that is suitable for finding quality-optimal hardware implementations in a fast and accurate manner. In the future, we plan to expand this approach into a full system-level optimization setup.

References

[1] V. Chippa et al. Analysis and characterization of inherent application resilience for approximate computing. In *DAC*, May 2013.

[2] H. Esmaeilzadeh et al. Architecture support for disciplined approximate programming. In *ASPLOS*, Apr 2012.

[3] Z. Kedem et al. Optimizing energy to minimize errors in dataflow graphs using approximate adders. In *CASES*, Oct 2010.

[4] S. Lee et al. Statistical modeling of approximate hardware. In *ISQED*, Mar 2016.

[5] C. Li et al. Joint precision optimization and high level synthesis for approximate computing. In *DAC*, Jun 2015.

[6] K. Nepal et al. ABACUS: A technique for automated behavioral synthesis of approximate computing circuits. In *DATE*, March 2014.

[7] Sidiroglou-Douskos et al. Managing performance vs. accuracy trade-offs with loop perforation. In *ESEC/FSE*, Sep 2011.

[8] S. Venkataramani et al. Quality programmable vector processors for approximate computing. In *Micro*, Dec 2013.