Addressing Dark Silicon Challenges with Disciplined Approximate Computing

Hadi Esmaeilzadeh Adrian Sampson
Michael Ringenburg
Luis Ceze Dan Grossman
University of Washington
Department of Computer Science & Engineering
http://sampa.cs.washington.edu/

Doug Burger
Microsoft Research
dburger@microsoft.com

Abstract

With the power constraints of the dark silicon era looming, architectural changes to improve energy efficiency are critical. Disciplined approximate computing, in which unreliable hardware components are safely used in error-tolerant software, is a promising approach. We discuss programming models, architectures, and accelerators for approximate computing.

1. Introduction

Moore's Law [9], coupled with Dennard scaling [3], through device, circuit, microarchitecture, architecture, and compiler advances, has resulted in commensurate exponential performance increases for the past three decades. With the end of Dennard scaling—and thus slowed supply voltage scaling—future technology generations can sustain the doubling of devices every generation (Moore's Law) but with significantly less improvement in energy efficiency at the device level. This device scaling trend heralds a divergence between energy efficiency gains and transistor integration capacity. Based on the projection results in [1, 4, 8, 12], this divergence will result in transistor integration capacity underutilization or *Dark Silicon*. Even the optimistic upper bound projections [4] show that conventional approaches including multicore scaling fall significantly short of the historical performance scaling to which the microprocessor research and industry is accustomed. We refer to this shortfall as the *Dark Silicon performance gap*.

Esmaeilzadeh et al. show that fundamental performance limitations stems from the processor core [4]. Architectures that move well past the power/performance Pareto-optimal frontier of today's designs are necessary to bridge the dark silicon performance gap and utilize transistor integration capacity. Improvements to the core's efficiency will have impact on long performance improvement and will enable technology scaling even though the core consumes only 20% of the power budget for an entire laptop, smartphone, or tablet. The model introduced in [4] provides places to focus on for innovation and shows that radical depurates are needed. To bridge the dark silicon performance gap, designers must develop systems that use significantly more energy-efficient techniques than conventional approaches. In this paper, we make a case for general-purpose approximate computing that allows abstractions beyond precise digital logic (error-prone circuitry) and relies on program semantics permitting probabilistic and approximate results.

Large and important classes of applications tolerate inaccuracies in their computation, e.g., vision, machine learning, image processing, search, etc. [2, 10, 11]. While past work has explored that property for performance and energy efficiency, they focused on either hardware (e.g., [2]) or software (e.g., [11]) in isolation and missed tremendous opportunities in co-design. We believe that hardware/software co-design for approximate computing can lead to orders of magnitude improvement in energy efficiency. In a nutshell, we advocate the following approach: (1) provide the programmer with safe ways to write programs that can tolerate small inaccuracies while guaranteeing strong program semantics; (2) build architectures that can save energy by running these disciplined approximate programs; (3) build accelerators that can leverage approximation to execute hot code in general-purpose programs more efficiently.

2. Disciplined Approximate Programming

In order to make approximate computing viable, we need an easy and *safe* way to express approximate programs. Namely, it must be straightforward to write reliable software that uses unreliable, error-prone hardware. To accomplish this, the programmer must be able to distinguish important pieces of data and computation that retain precise semantics in the presence of approximation. This is extremely important for programmability and argues for programming model involvement. We have been exploring the use of type systems to declare data that may be subject to approximate computation in a language extension called EnerJ [10]. Using programmer-provided type annotations, the system automatically maps approximate variables to low-power storage, uses low-power operations, and even applies more energy-efficient algorithms provided by the programmer. In addition, EnerJ can statically guarantee isolation of error-sensitive program components from approximated components. This allows a programmer to control explicitly how the errors from approximate computation can affect the fundamental operation of the application. Crucially, employing static analysis eliminates the need for dynamic checks, further improving energy savings. We call this language-based approach "disciplined" approximate programming. In this model, there is no formal guarantee on the quality of the output; however, the runtime system or the architecture can be configured such that the approximation does not produce catastrophically altered results.

An approximation-aware programming model also helps programmers write generalizable approximate programs. Using an *abstract* semantics for approximation, programs written in approximation-aware languages like EnerJ can take advantage of many different hardware mechanisms—including both traditional architectures and accelerators, both of which are described below.

3. Architecture Design for Approximation

In a recent paper [5], we proposed an efficient mapping of disciplined approximate programming onto hardware. We introduced an ISA extension that provides approximate operations and storage, which give the hardware freedom to save energy at the cost of accuracy. We then proposed *Truffle*, a microarchitecture design that efficiently supports the ISA extensions. The basis of our design is dual-voltage operation, with a high voltage for precise operations and a low voltage for approximate operations. The key aspect of the microarchitecture is its dependence on the instruction stream to determine when to use the low voltage. We introduce a transistor-level

design of a dual-voltage SRAM array and discuss the implementation of the core structures using dual-voltage primitives as well as dynamic, instruction-based control of the voltage supply. We evaluate the power savings potential of in-order and out-of-order Truffle configurations and explore the resulting quality of service degradation. We evaluate several applications and demonstrate energy savings up to 43%. Even though our microarchitecture implementation of Truffle is based on dual-voltage functional units and storage, the ISA extensions and the architecture design can leverage other approximate technologies such as custom-designed approximate functional units as well as analog operation or storage units. Utilizing these technologies, we can further improve the efficiency of our architecture design which is enabled by breaking the accuracy abstraction while preserving safety guarantees.

The Truffle architecture enables fine-grained approximate computing, where the granularity of approximate operations is a single instruction. This approach also builds on fine-grained approximation at the storage level, where a single register or a single data cache line can be approximate or precise state. Thus, the Truffle microarchitecture is able to run a mixture of approximate and precise instructions that dynamically choose the level of precision they should execute. In order to execute this fine-grained mixture of precise and approximate instructions, the Truffle microarchitecture relies on the compiler to provide static guarantees on safety of the execution. Our ISA extensions allow the compiler to statically encode the precision level of all the operations and storage for the microarchitecture. This compiler–architecture co-design approach simplifies the microarchitecture and alleviates the overheads of fine-grained approximate execution.

Since the lower-power units in Truffle are restricted to execution and data storage, the overhead of processor frontend is a limiting factor in the efficiency of this class of microarchitectures. Hence, we are introducing *approximate accelerators*.

4. Approximate Accelerators

Recent research on addressing the dark silicon problem has proposed configurable accelerators that adapt to efficiently execute hot code in general-purpose programs without incurring the instruction control overhead of traditional execution. Recent configurable accelerators include BERET [7], Conservation Cores [12], DySER [6], and QsCores [13]. Approximate accelerators have the potential to go beyond the savings available to correctness-preserving accelerators when imprecision is acceptable.

We are exploring approximate accelerators based on a novel *data-driven* approach to mimicking code written in a traditional, imperative programming language. Our approach transforms error-tolerant code to use a learning mechanism—namely, a neural network—to perform some computation that mimics the behavior of some hot code in an approximation-aware application. This program transformation transparently trains a neural network based on empirical inputs and outputs for the "hot" target code. Then, a hardware neural network implementation can be used at run time to replace the original imperative code. In this sense, this program transformation uses hardware neural network implementations as a new class of approximate accelerators for general-purpose code. We call these accelerators *Neural Processing Units (NPUs)*.

Hardware neural network implementations, in both analog and digital logic, have been thoroughly explored and can be extremely efficient, so there is great efficiency potential in replacing expensive computations with neural network recalls. To effectively use them as accelerators, however, requires low latency and thus tight coupling with the main core. We are investigating digital and analog NPU implementations that integrate with the processor pipeline.

This learning-based approach to approximate acceleration also opens the door to acceleration using existing hardware substrates: GPUs, FPGAs, or FPAAs. By mapping the trained neural network to high-performance implementations on these substrates, the program transformation has the potential to provide speedup even without special hardware for some applications.

To realize learning-based approximate acceleration with NPUs, we must address a number of important research questions:

- What should the programming model look like?
- How can a neural network be automatically configured and trained with limited intervention from the programmer?
- How should the compilation workflow transform programs to invoke NPUs?
- What ISA extensions are necessary to support NPUs?
- How can a high-performance, low-power, configurable neural network implementation tightly integrate with the pipeline?

Space prohibits us from treating each of these issues in detail here, but we have explored solutions to each of these important questions and will discuss them in the presentation.

5. Conclusions

We strongly believe that approximate computing offers many opportunities to improve computer systems and bridge the dark silicon performance gap imposed by energy inefficiency of the device scaling. While we are actively developing new technologies at programming level, architecture level, and circuit level, we need language and runtime techniques to express and enforce reliability constraints to seize this opportunity.

References

[1] S. Borkar and A. A. Chen. The future of microprocessors. *Communications of the ACM*, 54(5), 2011.

[2] M. de Kruijf, S. Nomura, and K. Sankaralingam. Relax: An architectural framework for software recovery of hardware faults. In *ISCA*, 2010.

[3] R. H. Dennard, F. H. Gaensslen, V. L. Rideout, E. Bassous, and A. R. LeBlanc. Design of ion-implanted mosfet's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9, October 1974.

[4] H. Esmaeilzadeh, E. Blem, R. St. Amant, K. Sankaralingam, and D. Burger. Dark silicon and the end of multicore scaling. In *ISCA*, 2011.

[5] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger. Architecture support for disciplined approximate programming. In *ASPLOS*, 2012.

[6] V. Govindaraju, C.-H. Ho, and K. Sankaralingam. Dynamically specialized datapaths for energy efficient computing. In *HPCA*, 2011.

[7] S. Gupta, S. Feng, A. Ansari, S. Mahlke, and D. August. Bundled execution of recurring traces for energy-efcient general purpose processing. In *MICRO*, 2011.

[8] N. Hardavellas, M. Ferdman, B. Falsafi, and A. Ailamaki. Toward dark silicon in servers. *IEEE Micro*, 31(4):6–15, July–Aug. 2011.

[9] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[10] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman. EnerJ: Approximate data types for safe and general low-power computation. In *PLDI*, 2011.

[11] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard. Managing performance vs. accuracy trade-offs with loop perforation. In *FSE*, 2011.

[12] G. Venkatesh, J. Sampson, N. Goulding, S. Garcia, V. Bryksin, J. Lugo-Martinez, S. Swanson, and M. B. Taylor. Conservation cores: Reducing the energy of mature computations. In *ASPLOS*, 2010.

[13] G. Venkatesh, J. Sampson, N. Goulding, S. K. Venkata, S. Swanson, and M. Taylor. QsCores: Trading dark silicon for scalable energy efficiency with quasi-specific cores. In *MICRO*, 2011.