Approximating to the Last Bit

Thierry Moreau, Adrian Sampson
Luis Ceze, Mark Oskin

***Abstract*—Existing architectures provide little incentive to minimize the precision of arithmetic and memory operations. This paper looks at approximation opportunities from reducing precision on compute and memory operations in compute-intensive kernels. We present Axe, an approximation-aware precision tuning framework for C/C++ programs built on top of ACCEPT, that maximizes bit-savings while satisfying application-level quality constraints. We use Axe on 14 PERFECT benchmark kernels, and argue that the significant savings achievable at varying quality levels motivates the need for architectures that incorporate mechanisms to translate lower precision requirements into energy savings.**

I. Introduction

Reducing bit-width has been a popular approach in the design of accelerators to maximize efficiency. By narrowing the datapath to just the right width, system designers can minimize area and power, and benefit from improved performance and effective memory bandwidth at the expense of precision that they don't need. Unfortunately, general purpose-processors and programmable accelerators are not designed with precision flexibility in mind. This usually means that a programmer is faced with choosing between single-precision or double-precision, or between 32- or 64-bit fixed-point integers. As if that wasn't limiting enough, large control and instruction-fetch overheads can limit the energy savings resulting from precision reduction in general-purpose processors, thus providing little incentive for programmers to minimize precision.

If hardware offered the ability to gracefully trade precision for energy savings, it would push application programmers to think more seriously about quality. Surely adding a quality knob to instructions or data would add some complexity and overheads, but we need to understand the potential for savings in a hardware-agnostic way. This paper addresses the question of how many of the precision bits in a program are really that important. And the answer is, if you are willing to sacrifice some quality, it's surprisingly few. This paper makes the following contributions:

- We present Axe, an approximation-aware precision tuning framework for C/C++ programs that fine-tunes the precision of safe-to-approximate instruction.
- We introduce *bit-savings*, a hardware-agnostic metric that quantifies precision reduction over the execution of an approximate program.
- We use Axe to explore instruction-level precision-reduction opportunities on the PERFECT benchmark suite.

We use the results of this study to motivate the need for precision-scalable architectures that can graciously translate bit-savings into energy savings.

II. Axe: A Precision Autotuner.

We present Axe, an instruction-level precision tuning framework built on top of ACCEPT [1], the C/C++ approximate compiler. In a nutshell, Axe takes an annotated approximate program and greedily finds a configuration that maximizes *bit-savings* while maintaining application-level quality bounds. Bit-savings is a hardware-agnostic metric that quantifies how aggressively precision can be reduced over the execution of an approximate program. Bit-savings can be obtained with Equation 1, where r_i and a_i is the precision in bits of reference and approximate instruction i, and e_i is the number of times instruction i is executed. For floating-point instructions the precision r_i and a_i get computed over the mantissa bits.

$$BitSavings = \sum_{i=1}^{N} \frac{(r_i - a_i)}{r_i} * \frac{e_i}{\sum_{j=1}^{N} e_j} \quad (1)$$

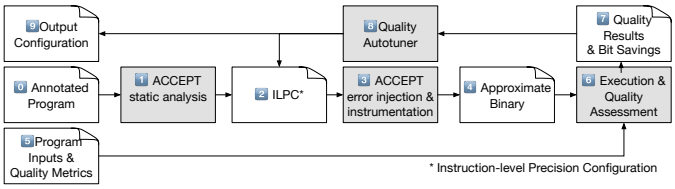

Fig. 1: Overview of the Axe framework.

A high-level overview of Axe is provided in Figure 1. Axe is built on top of ACCEPT [1]. The idea is to provide a program with EnerJ-style [2] data annotations ⓪ which can be used by ACCEPT's analysis libraries ① to statically identify safe-to-approximate instructions in an Instruction-Level Precision Configuration (ILPC) ②. Each safe-to-approximate instruction is a knob that the precision autotuner can tweak in order maximize bit-savings. For each relaxed instruction in the ILPC, Axe emulates the effect of reduced-precision execution using ACCEPT's customizable error-injection library ③. On low-precision arithmetic and load instructions, ACCEPT truncates the destination register value after the operation executed, while on low-precision store instructions, ACCEPT truncates the data register value before it is written to memory. We implement an instrumentation pass ③ in ACCEPT in order to collect dynamic statistics. ACCEPT produces an approximated and instrumented binary ④ that can be executed on programmer-provided program inputs ⑤. The output quality of the approximate program is then assessed ⑥ using programmer-provided quality metrics ⑤. Along with an output quality score, a bit-savings score ⑦ is computed by Axe by combining the ILPC parameters with execution statistics.

The goal of the autotuner ⑧ is to find an ILPC that satisfies application-level quality requirements and maximizes bit-savings. The algorithm used by the autotuner is an iterative greedy search inspired by previous work [3]. It finds a local-minimal configuration in $O(m^2 * n)$ worst-case time where m is the number of static safe-to-approximate instructions (knobs) and n is the number of precision levels (knob settings). The autotuner returns a locally-optimal configuration ⑨ that satisfies the user-provided quality requirements, and reports the resulting bit-savings.

App.	Kernel	Approx. Static Count	Approx. Dyn. Ratio	Savings at 20dB	40dB	60dB
PA1	2D Convolution	9	37%	47%	29%	22%
	DWT	54	47%	86%	77%	74%
	Histogram Eq.	16	55%	59%	51%	46%
STAP	Outer Product	142	86%	77%	59%	45%
	System Solver	77	83%	79%	64%	50%
	Inner Product	83	88%	82%	67%	53%
SAR	Interpolation 1	40	72%	85%	75%	66%
	Interpolation 2	40	57%	87%	76%	67%
	Backprojection	43	86%	68%	59%	51%
WAMI	Debayer	168	41%	50%	41%	40%
	Lucas-Kanade	93	56%	89%	73%	57%
	Gaussian MMs	140	66%	86%	11%	4%
Req.	FFT-1D	46	61%	77%	65%	53%
	FFT-2D	100	64%	65%	52%	42%
	Average	75	64%	74%	57%	48%

TABLE I: PERFECT overview and results summary.

III. Evaluation

Applications and Quality. We use Axe to evaluate instruction-level precision requirements of the PERFECT benchmark suite [4]. PERFECT is composed of compute-intensive benchmarks that span image processing, signal processing, compression and machine learning. Table I provides an overview of the PERFECT benchmark suite. For quality assessment, we follow the PERFECT manual guidelines [5], and use a uniform Signal-to-Noise Ratio (SNR) quality metric across all benchmarks to measure quality degradation.

Program Characteristics. Table I summarizes relevant characteristics of the PERFECT kernels that were derived using Axe. The *approximate static count* column lists the number of static instructions that are safe-to-approximate according to ACCEPT. The more safe-to-approximate static instructions the more work the autotuner will have to perform. The *approximate dynamic ratio* is the ratio of safe-to-approximate dynamic instructions to total dynamic instructions. The higher the ratio, the larger the opportunity for approximation in a program. The approximate dynamic ratio is on average 64% which indicates that the PERFECT benchmark suite is a compelling target for approximate computing. The bulk of the not-safe-to-approximate instructions are composed of branching instructions and integer arithmetic used for address computation, neither of which could be approximated without compromising the safety of the program. The good news is that for costly instruction categories, such as floating point arithmetic, loads and stores to memory and standard C math functions, all instructions are safe-to-approximate.

Quality vs. Bit-Savings. For each PERFECT application kernel, we use Axe to derive the bit-savings on safe-to-approximate instructions over a range of SNR targets from 120dB down to 10dB (0.0001% up to 31.6% MSE). Figure 2 shows the average bit-savings obtained across the PERFECT benchmark for safe-to-approximate integer arithmetic, floating point arithmetic, memory ops and standard C math functions at different SNR targets. In addition, we compute the aggregate bit-savings for each benchmark, averaged over all benchmarks.

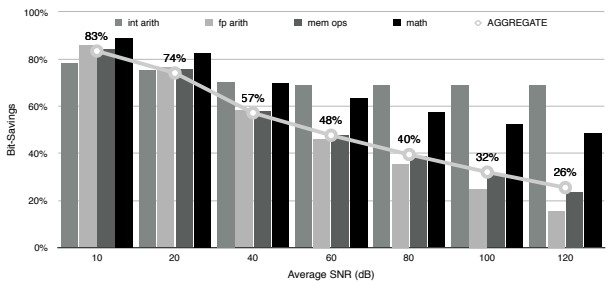

Fig. 2: Bit-savings vs. SNR averaged over PERFECT kernels, for integer arithmetic, FP arithmetic, memory ops and math functions.

In general, the lower the quality target, the higher the bit-savings. All instruction categories exhibit a smooth quality vs. bit-savings trade-off curve. We provide a breakdown of the aggregate bit-savings over all safe-to-approximate instructions in Table I at 20dB, 40dB and 60dB (10%, 1% and 0.1% respectively) for each PERFECT kernel. At 20dB, 40dB and 60dB SNR, bit-savings reach a substantial average of 74%, 57% and 48% for the safe-to-approximate instructions in the PERFECT benchmarks. Interestingly, bit-savings for integer arithmetic remains large at high SNR. This is because the most-significant bits of integer values can be trimmed off without affecting the final output.

Towards Bit-Serial Architectures. Future challenges will consist in designing *precision-scalable* architectures that can effectively translate bit-savings into energy savings. Precision-scalable architectures exhibit (1) mechanisms to dynamically scale precision in the compute and memory pipeline, and (2) minimal control overheads that won't impede the gains obtained from a leaner memory and compute pipeline. While the Quora [6] work proposed a compelling quality-scalable vector processor architecture, the mechanisms it used resulted in limited power reduction in the compute units. We believe that by using the old trick of bit-slicing arithmetic operations, we can translate bit-savings into *performance* improvements. *Bit-serial* computation for instance can execute simple operations like addition and bit-wise arithmetic in $O(n)$ time, and multiplication in $O(n^2)$ time, where n is the precision requirement, thus providing a compelling incentive to minimize precision in compute-intensive programs. Preliminary results on one of the PERFECT integer benchmarks using a bit-serial datapath shows an order-of-magnitude performance improvement at 10dB compared to its precise execution. Much of the challenge in designing such a quality-scalable architecture will be to ensure that precise evaluation is not worse on a bit-serial architecture than it is on a bit-parallel architecture. We believe that such a mechanism is key in generating significant energy savings, thus motivating programmers to think more about quality trade-offs.

Conclusions. We present Axe, a framework that fine-tunes precision requirements of approximate C/C++ programs. Using Axe, we find that at reasonable quality targets, a surprisingly large number of precision bits are not required in compute-intensive programs. This observation motivates the need for architectures that can translate lower-precision requirements into large energy savings.

REFERENCES

[1] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, "ACCEPT: A programmer-guided compiler framework for practical approximate computing," Tech. Rep. UW-CSE-15-01-01, U. Washington, 2015.

[2] 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 *ACM Conf. Programming Language Design and Implementation (PLDI)*, 2011.

[3] C. Rubio-Gonzalez, C. Nguyen, H. D. Nguyen, J. Demmel, W. Kahan, K. Sen, D. H. Bailey, C. Iancu, and D. Hough, "Precimonious: Tuning assistant for floating-point precision," in *Int. Conf. High Performance Computing, Networking, Storage and Analysis*, 2013.

[4] "DARPA PERFECT program." http://hpc.pnl.gov/PERFECT/.

[5] K. Barker, T. Benson, D. Campbell, D. Ediger, R. Gioiosa, A. Hoisie, D. Kerbyson, J. Manzano, A. Marquez, L. Song, N. Tallent, and A. Tumeo, *PERFECT (Power Efficiency Revolution For Embedded Computing Technologies) Benchmark Suite Manual*. Pacific Northwest National Laboratory and Georgia Tech Research Institute, December 2013. http://hpc.pnnl.gov/projects/PERFECT/.

[6] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality programmable vector processors for approximate computing," in *IEEE/ACM Int. Symp. Microarchitecture (MICRO)*, 2013.