Just-In-Time Approximate Computing

15-745 Project Proposal

Joseph Lee and Naman Jain

{josephle, namanj}@andrew.cmu.edu

Thursday 20th March, 2014

1. Project Description

Project Web Page: `http://www.contrib.andrew.cmu.edu/~namanj/15-745/`

Modern computer systems have to be correct at all costs. Though error-free execution is beneficial, the tradeoff is high execution time. For many applications, perfect correctness is unnecessary and todays over-provisioned systems waste time and energy providing precision they dont need. Many image, audio, and video processing algorithms use approximation techniques to compress and encode multimedia data to various degrees that provide tradeoffs between size and correctness such as lossy compression techniques [1]. For example, while trying to smooth an image, the exact output value of a pixel can vary. If the output quality is acceptable for the user or the quality degradation is not perceivable, approximation can be employed to improve the performance. Approximate computing systems exploit error-resilient software to reduce resource usage. Even with all this abundant error tolerance, it would be a mistake to completely abandon correctness guarantees.
Most approximate computing systems are currently static in nature and some generate multiple versions of a kernel with varying degrees of accuracy, and during runtime they select the kernel version that satisfies *target output quality* (TOQ) requirements [1]. Such systems do not fully exploit the amount of approximation possible. We propose to implement dynamic approximate computing, where a Just-In-Time compiler provides optimal code with desired precision or TOQ during run-time.

Minimum Goal (75%): Our 75% goal is to make a JIT framework give profiling information useful for approximate computing.

Primary Goal (100%): Our 100% goal is to complete the JIT framework and implement at least one approximate computing algorithm.

Secondary Goal (125%): If we complete our 100% goal we will try to implement one more optimal algorithm and test performance gains.

2. Logistics

Plan of Attack and Schedule:

21st March - 27th March:
Naman: Determine approximate computing optimization to implement
Joseph: Get familiar with JikesRVM

28th March - 3rd April:
Naman: Write one motivating example/benchmark
Joseph: Begin writing profiling code

4th April - 10th April:
Naman and Joseph: Begin implementation of optimization

11th April - 17th April (Milestone):
Naman: Continue writing of optimization
Joseph: Write more benchmarks

18th April - 25th April:
Naman: Run benchmarks and analyze benchmark data
Joseph: Test and Debug optimization

26th April - 30th April (Final Report):
Naman and Joseph: Write final report and poster

Milestone

Till our milestone, we expect to complete most of our framework, approximate computing pass and benchmark, so that we can start off with test and debug the following week.

Literature Search

We are collecting several papers related to approximate computing and JITs. We plan to do extensive literature search and find optimal algorithms that we could implement.

Required Resources

We plan to use JikesRVM. It is free and open source software.

Getting Started

We went through a few papers on approximate computing and are currently doing thorough study of optimal algorithms for implementation in dynamic compilers. Currently, there is nothing that prevents us to start.

References

[1] Samadi, Mehrzad, et al. "SAGE: self-tuning approximation for graphics engines." Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture. ACM, 2013.