Loop Perforation and Automatic Memoization for Approximate Computing

Joseph Lee and Naman Jain

{josephle, namanj}@andrew.cmu.edu

Thursday 17th April, 2014

Major Changes

After reading many papers on the topics of approximate computing and dynamic compilation, we have come to the conclusion that many of the approximate computing algorithms are unsuited for dynamic compilation. However, loop perforation is in fact well-suited for dynamic compilation. Instead of working on dynamic compilation, however, we believe that loop perforation could potentially benefit from automatic memoization, which would potentially reduce the number of failure cases that loop perforation can create and even increase performance on other cases.

While this has been done before, it was done with the help of a Z3 theorem prover, which seems to be rather overkill. As we are not doing the mathematical theory for robustness of programs, a more pragmatic implementation is preferred.

What You Have Accomplished So Far

We have a completed benchmark that we wished to use to explore one possible approximate computing optimization. We are also settled on doing loop perforation as our approximation computing optimization.

Meeting Your Milestone

We currently have a simple loop perforation compiler pass that can be expanded to do more extensive optimizations.

Surprises

The major surprise is that we severely underestimated the difficulty of combining approximate computing with dynamic compilation. While we resolved to do one approximate computing optimization, we have come to realize that many of the optimizations are not well-suited to dynamic compilation.

Revised Schedule

April 21
Joseph: Continue working on the loop perforation implementation.
Naman: Write more benchmarks and test cases.

April 28
Joseph: Debug and finalize loop perforation.
Naman: Begin and finish work on final report and poster.

Revised Goals

75%: Working loop perforation
100%: Working loop perforation with automatic memoization
125%: Working loop perforation framework with automatic memoization (automatic loop perforation analysis)

Resources Needed

We are using LLVM for our implementation. Nothing is required as of now.

References

[1] Agarwal, Rinard, Sidiroglou, Misailovic, Hoffman. Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures. `http://dspace.mit.edu/handle/1721.1/46709`