Using Code Perforation to Improve Performance, Reduce Energy Consumption, and Respond to Failures

By: Henry Hoffmann, Sasa Misailovic, Stelios Sidiroglou, Anant Agarwal, and Martin Rinard

Presented by: Aditya Chitta, Andrew Lu, and Junwon Shin (Group 4)

The Problem

Standard approaches to trade accuracy for performance, robustness, energy savings, etc. require specific knowledge

- e.g. Optimization heuristics for MP3 audio compression can’t be applied to other types of compression e.g. image and video

Code Perforation and SpeedPress

- Automatically enhances applications to support the management of performance/accuracy tradeoff
- Given a user-specified distortion bound, automatically identifies parts of the computation that can be skipped without violating this bound
- Transformed applications ran two to three times faster while distorting the output by < 10%

Loop Perforation

Transforming loops to execute a subset of iterations, trading accuracy for performance.

Applications:

- Lossy video/audio encoders
- Machine learning algorithms
- Monte-Carlo simulations
- Information retrieval
- Scientific/economics computations

Loop Perforation Overview

Step 1: Induction variable transformation

- Transforms induction variable to following form:

```
for (i = 0; i < maxVal; i++) {/* … */}
```

Step 2: Loop perforation

- Given perforation rate (PR) and perforation strategy, transform loops to following form:

```
for (i = 0; i < maxVal; i++) {
        if (doPerforate(i, PR)) continue
        // …
}
```

Perforation Strategies

1) **Modulo Perforation:** skip every n-th iteration

```
doPerforate(i, PR) {
        n = 1 / PR
        return i % n == 0
}
```

Acceptability Model

To measure the effect of loop perforation, SpeedPress requires a user-provided acceptability model for the program output with 2 components:

Output Abstraction

Mapping from program's specific output to numeric value(s)

- e.g. bitrate for a video encoder
- Requires a basic understanding of application

Distortion Metric

Assume that abstraction model produces outputs in form $o_1 ... o_m$

$$d = \frac{1}{m} \sum_{i=1}^{m} \left| \frac{o_i - \hat{o}_i}{o_i} \right|$$

Step 1: Identify Candidate Loops

```
for i in inputs
   candidateLoops[i] = performProfiling(program, i)
   for each l in candidateLoops[i]
      scores[i][l] = assignInitialLoopScore(l)
   filterProfiledLoops(candidateLoops[i])
```

Captures:

- # of loop iterations
- # of loop invocations
- # of basic block accesses within loop

Based on profile data

Prioritize loops that execute the most instructions

Step 2: Individual Loop Performance and Distortion

```
for l in candidateLoops
    spdup, dist = perforateLoopSet(program, {l}, i)
    scores[i][l] = updateScore(spdup, dist, scores[i][l])
filterSingleExampleLoops(candidateLoops[i], scores[i], maxDist)
```

```
perforateLoopSet(program, loopSet, input)
   program' = instrumentLoops(loopSets)

   time, output = execute(program, input)
   time', output' = execute(program', input)

   abstrOut = abstractOutput(output)
   abstrOut = abstractOutput(output')

   speedup = calcluateSpeedup(time, time')
   distortion = calculateDistortion(abstrOut, abstrOut')

   return speedup, distortion
```

Filter out loops that cause program to:

- Terminate unexpectedly after perforation
- Become unresponsive

Prioritize loops that maximize performance or maximize accuracy

Step 3: Discover Loop Sets

```
candidateLoopSets = {}
for i in inputs
   candidateLoopSets[i] =
      selectLoopSet(program, candidateLoops, scores, i, maxDist)
```

Observe joint perforation effects on program accuracy and performance for each input

```
selectLoopSet(program, candidateLoops, scores, input, maxDist)

   loopQueue = sortLoopsByScore(candidateLoops, scores)

   LoopSet = {}
   cummulativeSpeedup = 1
   while loopQueue is not empty
      tryLoop = loopQueue.remove()
      trySet = LoopSet U {tryLoop}
      speedup, distortion = runPerforation(trySet, input)

      if speedup > cummulativeSpeedup and distortion < maxDist
         loopSet = trySet
         cummulativeSpeedup = speedup

   return LoopSet
```

Step 4: Select Best Performing Loop Set

```
loopsToPerforate =
    findBestLoopSet(program, candidateLoopSets, inputs)
return loopsToPerforate
```

Score for each loop set is derived as a statistic of the scores for all inputs (i.e. mean or minimum)

```
findBestLoopSet(program, loopSets, inputs)

   for each ls in loopSets
      for each i in inputs
         speedup, distortion =
                perforateLoopSet(program, ls, i)
         score[ls][i] = assignScore(speedup, distortion)

      score[loopSet] = scoreFinal(score[ls][*])

   return argmax(score)
```

Evaluation

Benchmarks

x264

H.264 Encoding on Video Stream

streamcluster

Data Mining Application for Online Clustering

swaptions

Pricing a Portfolio of Swaptions

canneal

Simulated Annealing of Routing Cost of Microchip Design

blackscholes

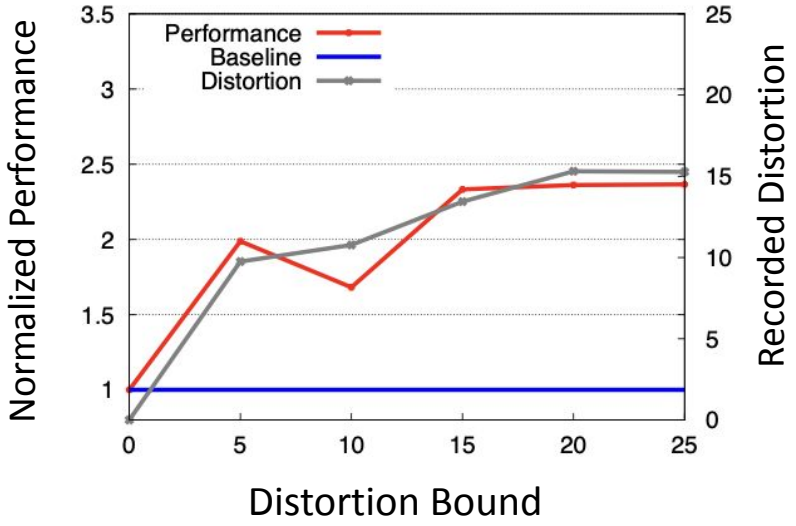
Pricing a Portfolio of European Options

bodytrack

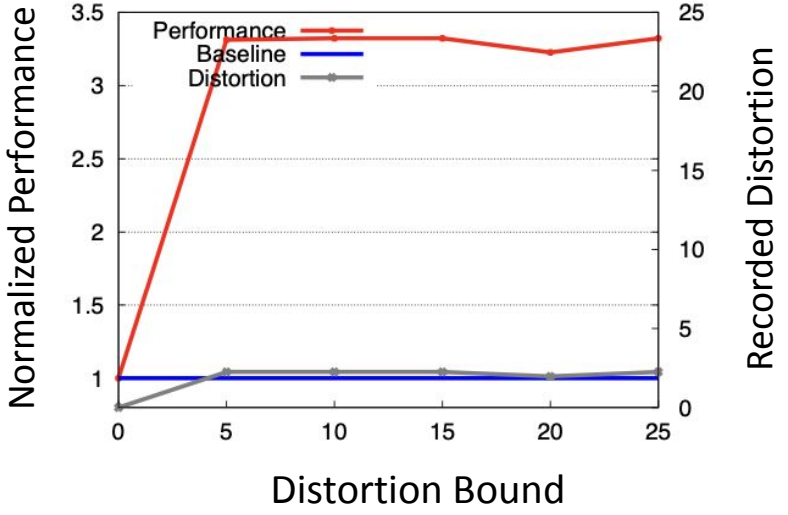
CV Application to Track Human Movement

Results

x264



bodytrack



Results - General Trends

- Both performance and distortion increase as the acceptable distortion bound increases
- For all benchmarks, SpeedPress provided at least 2x speedup for a maximum distortion bound of 15%
- Several benchmarks can achieve 3x speedup or more

Strengths vs Weaknesses

Strengths

- Works well on a wide range of application types
- Achieves high speedup benefits for a relatively low distortion
- SpeedPress allow applications to automatically scale speedup while keeping distortion within acceptable bounds

Weaknesses

- Not appropriate for applications that require accuracy
- Speedup sometimes does not scale linearly - which requires various bounds to be tried to find optimal balance between distortion and speedup
- Algorithm doesn’t explore optimizing perforation rates

Summary

- SpeedPress is a LLVM compiler that exploits code perforation to trade accuracy for performance
- Not applicable for applications that have hard logical correctness requirements
- Performs extremely well across a wide variety of applications that have challenging performance, failure, and accuracy requirements