Experimenting with Neuroevolution for Graal Inlining Heuristics
Makarand Parigi

Twitter VM Team Intern Summer 2019
University of Michigan Computer Science Senior

@MakarandParigi
parigim@umich.edu
Overview

- Inlining is an important optimization
- Machine Learning could help
- Evolutionary techniques may be applicable
Inlining & Heuristics
public int add(int a, int b) {
    return a + b;
}

global void foo() {
    int x = add(2, 3);
    int y = add(4, 9);
}

public void foo() {
    int x = 2 + 3;
    int y = 4 + 9;
}
Inlining

- Replacing callsites with bodies of callees
- Enables other optimizations
- Many interdependent decisions
- “How” is easy. “When” is hard.
  - Too much is bad, too little is not optimal.
How do we decide when to inline?
Current Inlining Policy

- Manually constructed and tuned
- Often tuned for a specific benchmark
- Same across environments and workloads
- Graal’s current policy
  - Simple if-statements
  - Gather some data about the decision point, run it through simple flow
Incremental Inlining Algorithm

- Online inlining algorithm
- Identifies and inlines clusters of callsites
- Alternates between inlining and optimizations

Machine Learning

ML helps us make decisions. Can it tell us when to inline? Doug Simon et al. say yes. (about 10% speedup)

Specifically - Evolutionary Algorithms.

The Evolution Process

1. Evaluation
   Evaluate all the organisms in the current population.

2. Selection
   Select the fittest organisms.

3. Crossover
   Combine parent organisms to produce children.

4. Mutation
   With some probability, modify the children.

5. Repeat!
   As the new generation is created, repeat the process until convergence or a maximum limit.
Neuroevolution of Augmenting Topologies (NEAT)
NEAT

- Applying evolution to Neural Networks
- Neuroevolution algorithm developed by Kenneth O. Stanley in 2002
- Can evolve structure and weights

Applying NEAT
NEAT for the Inlining Problem

- Inlining requires making decisions using a heuristic
- Neural Networks can be used as a heuristic
  - Input is the same data the current policy receives
  - Output is YES/NO
- They cannot be trained for inlining using traditional Gradient Descent
  - No way of knowing the correct decisions
- A heuristic can be evolved
- Evaluating an organism means getting running time on benchmarks
Thank you.
Result of NEAT

- When NEAT ends, it gives you the best organism of the last generation
- “Bake” this into the compiler, totally offline
- Can make different network/policies for different environments
Benefits of NEAT for Inlining

- Automatic construction of heuristic without expert work
- Easy to customize for environments, workloads
- Done offline
  - Result is a single heuristic
- Doug Simon et al. showed improvements over default algorithm in MaxineVM (C1X)
In More Detail

- NEAT
  - Produces
    - Neural Network to Evaluate
      - Write to
    - Calls
      - NN File
        - Read by
          - Write to
            - Benchmark
              - Read by
                - Runtime Data File

Libraries

Neuroph is used for Neural Network support. There is a NEAT for Neuroph implementation.

Why Neuroph? Easy to transfer.
Running

- Aurora/Mesos
- Running on dedicated JVM cluster machine
- No interference
Evolution Hyperparameters

- Population Size
- Number of Generations (or Termination Condition)
- Mutation rates
- Selection
- Speciation
Other Knobs & Dials

- What benchmarks to run
  - How to evaluate a neural network
- Warmup run count
- How to turn neural net output into YES/NO
  - Threshold
  - Probability
Output

- Fitness of every organism evaluated
- Generation Statistics
  - Best Organism
  - No. of Species
- Save each generation
  - Can resume from any point later
Results
The Experiment

- Population Size = 12
- Warmups = 12
- Benchmarks
  - "dacapo:lusearch"
  - "dacapo:jython"
  - "dacapo:pmd"
  - "dacapo:avrora"
  - "dacapo:luindex"
  - "dacapo:sunflow"
  - "dacapo:xalan"
  - "dacapo:fop"
The Outcome

Ran for ~271 Generations before kill
The Secondary Experiment

Something Faster.

Optimize for only a single benchmark, reduce warm up runs.

Only Benchmark = dacapo:pmd
Why? Demonstrated variance between policies
Warmups = 6
The Secondary Outcome

Ran for 1000 Generations
What Happened?

- No significant improvements over the generations
- **Theory: Maybe inlining policies don’t matter?**
  - Unlikely, running these benchmarks with varying policies led to large differences in running time
  - E.g. for dacapo:pmd –
    - never inline - 910ms
    - always inline - 1515ms
    - default graal - 301ms
- **Theory: Search is not wide enough**
  - Mutation rate was manually increased over default
  - Could use further experimentation
What Happened?

- **Theory: Network Sensitivity**
  - Generally, near the beginning the networks do not seem to be sensitive to inputs, thus essentially always being always-inline or never-inline
  - Investigate input normalization, mutation rates, binarization

- **Theory: NEAT isn’t suited for this**
  - Doug Simon et al. says otherwise
  - The other theories are credible enough to merit further investigation before this

- **Theory: Implementation Bugs**
  - NEAT program is able to solve other problems
  - Further testing
Future Work & Potential Improvements

- **Workload Testing**
  - Create/Use benchmark that models different Twitter workloads

- **Experimental**
  - Mutation
  - Binarization
  - Input normalization
  - Testing

- **Parallelism**
  - Evaluate organisms simultaneously

- **Other Applications**
  - Autovectorization
Applicability

- Heuristics/Policies are everywhere
- In a compiler alone,
  - Instruction selection
  - Register allocation
  - Instruction scheduling
  - Software pipelining
- NEAT could help
- Investing in this work could mean returns in many different areas
  - Anywhere decisions need to be made using a policy
Addendum: Reinforcement Learning

- RL used for Go, Chess, Robotics, Chemistry
- Agent, Environment, Reward
- Network is Agent, Benchmark is Environment, Running Time is Reward
- RL Literature/Techniques may be applicable to Inlining & other optimizations
  - Q Learning
  - Actor-Critic Model
  - Deep Deterministic Policy Gradient