End-to-end Deep Learning of Optimization Heuristics

http://chriscummins.cc/pact17
Chris Cummins  University of Edinburgh
Pavlos Petoumenos  University of Edinburgh
Zheng Wang  Lancaster University
Hugh Leather  University of Edinburgh
compilers are very complex

```c
int main(int argc, char** argv) {
  ...
}
```

hand-coded heuristics
(out of date by time of release)
Machine learning in compilers

\[ y = f(x) \]
Machine learning in compilers

- Training Programs
- Feature Extractor
- Driver
- Best Decisions
- Feature Vectors
- Training Data
- Optimization Heuristic
Machine learning in compilers

- Training Programs
- Feature Extractor
- Driver
- Best Decisions
- Feature Vectors
- Training Data
- Optimization Heuristic

The human bit!

1. Hard to get right
2. Time consuming
3. Repetitious
Use a GPU

Use a CPU

Learned Heuristic

Feature space

Feature “Y”

Feature “X”
need good features!
Ways to fail

- **irrelevant**
  - e.g. not capturing the right information

- **incomplete**
  - e.g. missing critical information

- **unsuitable**
  - e.g. wrong combination of features / model
What we have

Training Programs

Driver

Best Decisions

Feature Extractor

Feature Vectors

Training Data

Predictive Model
What we need

- Training Programs
- Driver
- Best Decisions
- Training Data
- Predictive Model
Contributions

Heuristics without features

Beats expert approach

Learning across heuristics
Our approach

Program Code

Deep Learning

Optimization Decision

int main(int argc, char **argv) {
    ...

    Our approach involves using deep learning to optimize decision-making processes based on the input code.
Our approach

Code in

Rewriter

Encoder

preprocessing

normalize identifiers & code style
1. var/fun names: 'foo', 'bar', ... to 'a', 'b', ...
2. sanitize whitespace
3. consistent use of optional braces

encode as sequence of vocabulary indices

Vocabulary table for characters + lang keywords

Deep Learning

Optimization Decision

Program Code
Our approach

- **Embedding**: map vocab indices into real space
- **Language Model**: summarize sequence as vector (2 layer LSTM network)
- **Heuristic Model**: predict optimization on vector (2 layer DNN)

**Deep Learning**

**Optimization Decision**
Our approach

Program Code → Embedding → Language Model → Heuristic Model → Optimization Decision
How does it work?
```
void memset_kernel()
{
  global char* mem_d, short val...
}

void A()
{
  global char* a, short b...
}
```
How does it work?
Portable Mapping of Data Parallel Programs to OpenCL for Heterogeneous Systems

Grewe et al. (CGO'13) and Magni et al. (PACT'14) have made significant contributions to the field of heterogeneous mapping and thread coarsening. These works address the challenges of efficiently mapping parallel programs to heterogeneous systems, which are composed of multiple types of processors (e.g., CPUs, GPUs, and FPGAs). The authors propose techniques that optimize the performance of these programs on such heterogeneous architectures.

**CGO'13: Grewe et al.**

- **Title:** Portable Mapping of Data Parallel Programs to OpenCL for Heterogeneous Systems
- **Parallel System:** Portable mapping techniques
- **Coarse-Grained OpenCL (CGO)**
- **Heterogeneous Systems:** Utilizes OpenCL for efficient mapping

**PACT'14: Magni et al.**

- **Title:** Automatic Optimization of Thread- Coarsening for Graphics Processors
- **Parallel System:** Optimization techniques for thread coarsening
- **GPU Acceleration:** Focuses on graphics processors
- **Heterogeneous Systems:** Techniques for heterogeneous systems

These works are crucial for developers and researchers working on parallel and heterogeneous computing, as they provide insights into optimizing performance and efficiency in complex, multi-processor environments.
Heterogeneous Mapping
Thread Coarsening

Prior Art

Decision Space

Binary classification
One-of-six classification

Model

Decision Tree
Cascading Neural Networks

{CPU, GPU}
{1, 2, 4, 8, 16, 32}
Heterogeneous Mapping

Prior Art

Thread Coarsening

Features

4 features

Combined from 7 raw values.

Instruction counts / ratios.

7 features

Principal Components of 34 raw values.

Instruction counts / ratios / relative deltas.

2 papers!
Our Approach

Heterogeneous Mapping

1. Use the same model design for both
2. No tweaking of parameters
3. Minimum change - 3 line diff

Thread Coarsening
Prior Art

Heterogeneous Mapping

Thread Coarsening

Hardware

Training Programs

2x CPU-GPU architectures

4x GPU architectures

7 Benchmark Suites

3 Benchmark Suites

CGO’13

PACT’14
results
14% and 5% improvements over state-of-the-art

- **Heterogeneous Mapping**
  - State-of-the-art: 2.09x
  - DeepTune: 2.38x

- **Thread Coarsening**
  - State-of-the-art: 1.01x
  - DeepTune: 1.06x
14% and 5% improvements over state-of-the-art

Heterogeneous Mapping

State-of-the-art: 2.09x
DeepTune: 2.38x

Thread Coarsening

State-of-the-art: 1.01x
DeepTune: 1.06x

256 benchmarks
17 benchmarks
Transfer Learning

Heterogeneous Mapping

Thread Coarsening

general → specialized

Embedding → Language Model → Heuristic Model

Embedding → Language Model → Heuristic Model
Transfer Learning

Heterogeneous Mapping

Thread Coarsening

general $\rightarrow$ specialized

initialize with values
14% and 5% improvements over state-of-the-art

- **Heterogeneous Mapping**
  - State-of-the-art: 2.09x
  - DeepTune: 2.38x

- **Thread Coarsening**
  - State-of-the-art: 1.01x
  - DeepTune: 1.06x
14% and 11% improvements over state-of-the-art

- **Heterogeneous Mapping**
  - State-of-the-art: 2.09x
  - DeepTune: 2.38x
  - w. Transfer Learning: 2.09x

- **Thread Coarsening**
  - State-of-the-art: 1.01x
  - DeepTune: 1.06x
  - w. Transfer Learning: 1.12x
Try it for yourself!

[Image of a computer screen showing a webpage with code and data on GitHub, and text saying "runs in the browser" and "code and data on GitHub" at the bottom.]

http://chriscummins.cc/pact17
Problem: feature design is hard

Featureless heuristics

First cross-domain learning

11-14% speedups

http://chriscummins.cc/pact17