Using Machine Learning to Automate GPU Program Performance
Why GPU Auto-Tuning?

• Optimization plays a critical role in better utilizing GPU compute power

• Optimization effect heavily depends on GPU architecture and other optimizations
  – Difficult to select the best-performing optimization(s)

• Fast-paced architectural enhancements demand frequent re-tuning
Why Machine Learning?

• Traditional approaches
  – Analytical modeling (heuristic)
  – Empirical search

<table>
<thead>
<tr>
<th>Performance</th>
<th>Speed</th>
<th>Amount of Effort</th>
</tr>
</thead>
</table>

### Why Machine Learning?

- **Traditional approaches**
  - Analytical modeling (heuristic)
  - Empirical search

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Speed</th>
<th>Amount of Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analytical Modeling</strong></td>
<td><img src="image" alt="Performance" /></td>
<td><img src="image" alt="Speed" /></td>
<td><img src="image" alt="Amount of Effort" /></td>
</tr>
</tbody>
</table>
Why Machine Learning?

- Traditional approaches
  - Analytical modeling (heuristic)
  - Empirical search

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Speed</th>
<th>Amount of Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td>Analytical Modeling</td>
<td><img src="image1" alt="Performance" /></td>
<td><img src="image2" alt="Speed" /></td>
<td><img src="image3" alt="Amount of Effort" /></td>
</tr>
<tr>
<td>Empirical Search</td>
<td><img src="image1" alt="Performance" /></td>
<td><img src="image2" alt="Speed" /></td>
<td><img src="image3" alt="Amount of Effort" /></td>
</tr>
</tbody>
</table>
## Why Machine Learning?

- **Traditional approaches**
  - Analytical modeling (heuristic)
  - Empirical search

<table>
<thead>
<tr>
<th></th>
<th>Performance</th>
<th>Speed</th>
<th>Amount of Effort</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Analytical Modeling</strong></td>
<td>![Circle]</td>
<td>![Circle]</td>
<td>![Circle]</td>
</tr>
<tr>
<td><strong>Empirical Search</strong></td>
<td>![Circle]</td>
<td>![Circle]</td>
<td>![Circle]</td>
</tr>
<tr>
<td><strong>Machine Learning</strong></td>
<td>![Circle]</td>
<td>![Circle]</td>
<td>![Circle]</td>
</tr>
</tbody>
</table>
• Motivation

• Feasibility study
  – Should loops be interchanged in image-processing kernels?

• Conclusions and future work
Feasibility Study: Application Domain

• Auto-tuning for mobile GPUs

• Computational photography on smartphones

• Start with image processing applications
Feasibility Study

• A typical image processing application

```c
read image;
:

for (row = 0; row < img_rows; ++row)
  for (col = 0; col < img_cols; ++col)
  {
    read image pixels at and around (row,col);
    process image pixels;
    write image pixel at (row,col);
  }

write image;
```
Feasibility Study

• A typical image processing application

```c
read image;
:
#pragma kernel main tblock(BY,BX) thread(TY,TX)
#pragma loop_partition over_tblock over_thread
for (row = 0; row < img_rows; ++row)
#pragma loop_partition over_tblock over_thread
for (col = 0; col < img_cols; ++col)
{
    read image pixels at and around (row,col);
    process image pixels;
    write image pixel at (row,col);
}
:
write image;
```

What should the loop order be given a launch configuration?
Performance Impact of Loop Order

Dilate

Erode

Simple Blur

Sobel

Laplace

Scharr

Downsample

Canny
Why Loop Order Matters?

Loop Order + Launch Configuration

Distribution of loop iterations among GPU threads

Image pixels accessed by concurrent GPU threads

Degree of memory coalescing

Kernel Performance
Learning Experiment Overview

Training Programs

- Program Features
- Launch Configuration
- Loop Order
- Performance

Train

Model

Predict

New Program

- Program Features
- Launch Configuration
- Preferred Loop Order
Program Features

- What influences the preference of loop order?
  - Degree of coalescing of each memory access
  - Interleaved computation that hides access latencies

Raw features:
\[
\{ L_i, C_{outer,i}, C_{inner,i}, OFST_i \} \times N
\]
First Experiment: Raw

- **Model inputs:**
  - \( \{ L_i, C_{\text{outer},i}, C_{\text{inner},i}, \text{OFST}_i \} \times N \)
  - Launch configuration: BX, BY, TX, TY
  - When loops interchanged, swap \( C_{\text{outer}} \) and \( C_{\text{inner}} \)

- **Model output:** kernel execution time

- **Given a new kernel + a launch configuration**
  - Use model to predict execution time with both loop orders
  - Choose the loop order that gives lower execution time
Experiment Setup (Raw)

• Synthetically generated kernels
  – Each has two perfectly nested loops
  – … but differs in computation length and memory accesses in inner loop body

• Two kernel sets
  – K1: 4000 single-epoch kernels
  – K10: 4000 kernels with at most 10 epochs

• Collect execution time of each kernel with 3 launch configurations and both loop orders
  – On NVIDIA Tesla M2070
  – \((TX, TY) = (32, 8), (8, 32), (2, 128); (BX, BY) = (360, 1)\)
Experiment Setup (Raw)

• Two ML algorithms (regression)
  – SVMLight (default over-fitting parameter, Gaussian kernels)
  – Waffles RandomForest (160 trees, 4 attributes per tree)

• For each kernel set, train on a random 1000 kernels and test on the remaining 3000 kernels

• How to evaluate prediction accuracy on the test set?
Evaluation Metric

- **Count-based Prediction Accuracy**
  - % of test samples where the predicted loop order does give better kernel performance

- **Penalty-weighted Prediction Accuracy**
  - % of best performance achieved by predicted loop order
Experiment Result (Raw)

<table>
<thead>
<tr>
<th>Count-based Prediction Accuracy</th>
<th>SVM</th>
<th>RandomForest</th>
<th>SVM</th>
<th>RandomForest</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>80%</td>
<td>90%</td>
<td>55%</td>
<td>65%</td>
</tr>
<tr>
<td>K10</td>
<td>60%</td>
<td>70%</td>
<td>70%</td>
<td>80%</td>
</tr>
</tbody>
</table>
Experiment Result (Raw)

<table>
<thead>
<tr>
<th>Penalty-weighted Prediction Accuracy</th>
<th>SVM</th>
<th>RandomForest</th>
<th>SVM</th>
<th>RandomForest</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>55%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>65%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>70%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>75%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>80%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>85%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>90%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>95%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100%</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>K10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dealing with Large Program Space

• Program space is inherently large

• The number of input features in RAW grows with program length (# of epochs)

• Train one or more smaller models, each focusing on a short program segment

• Use program structure to link these models
  – e.g., execution time of a series of code segments is roughly the sum of per-segment execution times
Second Experiment: Raw-S

Train on single-epoch kernels

• Model inputs:
  – \{ L_i, C_{outer,i}, C_{inner,i}, OFST_i \}
  – Launch configuration: BX, BY, TX, TY
  – When loops interchanged, swap \( C_{outer} \) and \( C_{inner} \)

• Model output: kernel execution time

Given a new \( K \)-epoch kernel + a launch configuration

• Predict execution time with each loop order, by
  – Using the model (\( K \) times) to predict execution time of each epoch
  – Summing the per-epoch predicted time

• Choose the loop order that gives lower execution time
Experiment Setup / Result (Raw-S)

- Same two kernel sets K1 and K10
- Train on a random 1000 kernels in K1, and test on all kernels in K10
Applying GPU Expertise

• Loop order affects the degree of memory coalescing, thus kernel performance

• We can estimate # of DRAM transactions for each memory access in the inner loop body
  – Given the launch configuration

• A heuristic for kernel performance: total memory transactions from the inner loop body
### Experiment Result (All)

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>RandomForest</th>
</tr>
</thead>
<tbody>
<tr>
<td>K1</td>
<td>80%</td>
<td>80%</td>
</tr>
<tr>
<td>K10</td>
<td>70%</td>
<td>70%</td>
</tr>
</tbody>
</table>

**Model Comparison:**
- **SVM**
- **RandomForest**

**Accuracy Types:**
- **Raw**
- **Raw-S**
- **Heuristic**
What We Learnt

Machine learning can be a fast, accurate solution to auto-tuning,

If it is intelligently applied and integrated with our generic knowledge about programs
Beyond the “Toy” Problem

• Expand the optimization space to consider
  • Currently working on a 7-D optimization space, with about ~50K valid configurations

• Two challenges:
  – Large program space
  – Large optimization space

• Collect kernel performance data for all configurations?
• Train a performance predictor?
• Compare ML performance against Oracle?
Thank You!

Questions?