Predicting GPU Performance from CPU Runs Using Machine Learning

Ioana Baldini
Stephen Fink
Erik Altman

IBM T. J. Watson Research Center
Yorktown Heights, NY USA
To exploit GPGPU acceleration …

… need to port code to GPGPU programming model

**Pitfalls:**
Porting costs time/money

Sometimes GPU speedup is not worth the effort

**Problem:** Can a non-expert predict whether a data-parallel application is likely to perform well on a GPU, **without** incurring the costs of porting?
Maybe this is easy?

Suppose we already have data parallel CPU Version, e.g. OpenMP.

**Hypothesis**: Since data parallelism is data parallelism,
- SMP speedup predicts GPU speedup

![Graph showing the relationship between OpenMP speedup and GPU speedup.](image)
What is correlation between OpenMP and GPU performance?

\[ \rho = 0.146 \]
Selected results, OpenMP vs. GPU

- Similar scaling on CPU/OpenMP
- GPU results vary based on device/code
GPU performance prediction: traditional approaches

**Detailed analytic model of algorithm** + **Detailed model of GPU architecture** → **Accurate Performance Models**

- Relatively simple code structures
- Reasoning about non-trivial transformations
- Expert knowledge
- Complex mappings
- Device – Specific Models
Desired Solution: Fully automatic, for non-experts
- No static analysis
- No detailed architectural models
- Automatically tune for new device

Our Approach:
Use machine learning to learn from past experiences porting sequential code to GPUs

Input:
- corpus of past ports from sequential (C) to GPU/parallel code
- features from sequential code

Output:
- model of GPU speedup

Notes:
- Does not explicitly model specific transformations or architectural detail
  - Unlikely to produce highly accurate model, but rough estimate may be useful
- Input to model includes human expert knowledge embodied in previous ports
System architecture
## Application Features – Dynamic Instrumentation (Pin)

<table>
<thead>
<tr>
<th>Category</th>
<th>Feature</th>
<th>Mnemonic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Computation</strong></td>
<td>Arithmetic and logic instructions</td>
<td>ALU</td>
</tr>
<tr>
<td></td>
<td>SIMD-based instructions</td>
<td>SIMD</td>
</tr>
<tr>
<td>Memory</td>
<td>Memory loads</td>
<td>LD</td>
</tr>
<tr>
<td></td>
<td>Memory stores</td>
<td>ST</td>
</tr>
<tr>
<td></td>
<td>Memory fences</td>
<td>FENCE</td>
</tr>
<tr>
<td><strong>Control Flow</strong></td>
<td>Conditional and unconditional branches</td>
<td>BR</td>
</tr>
<tr>
<td><strong>OpenMP</strong></td>
<td>Speedup of 12 threads over sequential execution</td>
<td>OMP</td>
</tr>
<tr>
<td><strong>Aggregates</strong></td>
<td>Total number of instructions</td>
<td>TOTAL</td>
</tr>
<tr>
<td></td>
<td>Ratio of computation over memory</td>
<td>ALU-MEM</td>
</tr>
<tr>
<td></td>
<td>Ratio of computation over GPU communication</td>
<td>ALU-COMM</td>
</tr>
</tbody>
</table>
Supervised Learning: [WEKA 3] Binary/Ternary classifiers

Nearest Neighbors with Generalized Exemplars (NNGE)

Support Vector Machines (SVM)

Source: [Martin 95]
Methodology

System
- Dual-processor Intel Xeon X5690
- 12 cores total @ 3.47 GHz
- ATI FirePro v9800
- Nvidia Tesla C2050

Benchmarks
- Parboil 2.0
- Rodinia 2.2

Runs
- 48 runs total
- Cross validation:
  - Leave one out
  - Leave two out
Learning whether GPU acceleration is beneficial

Binary classifier: “GPU Speedup > 1”

### Features Selected

<table>
<thead>
<tr>
<th></th>
<th>SVM</th>
<th>NNGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tesla</td>
<td>ALU, LD, BR, TOTAL</td>
<td>ALU, LD, ST, ALU-MEM, BR, TOTAL</td>
</tr>
<tr>
<td>FirePro</td>
<td>ALU, LD, BR, TOTAL</td>
<td>ALU, LD, BR, TOTAL, OMP</td>
</tr>
</tbody>
</table>
Learning the speedup factor of GPU execution (SVM)

5 Binary classifiers:
“GPU Speedup > 1”
“GPU Speedup > 2”
“GPU Speedup > 3”
“GPU Speedup > 4”
“GPU Speedup > 5”

Classifier Accuracy
Predicting the Best Device for OpenCL runs

Which of 3 devices is best for a particular program?

• 3-way classifier using NNGE: “CPU”, “Tesla”, “FirePro”

Correct 39 of 46 runs. Effective accuracy = 91%
Summary of Results

• Classifiers are ~80% effective at deciding GPU speedup class for k = 1, …, 5
• Small set of features (ALU, LD, BR) are sufficient
• Predictions are effective for heterogeneous scheduling problem

Bottom Line

• Is 80% classifier accuracy good enough? Useful?
• Machine Learning doesn’t illuminate cause-and-effect
• Early days – results are promising, but much more exploration needed
Backup
Related Work

- Jia et al. [14] used regression techniques to predict the results of simulations of GPU architectures.
- Grewe et al. [7], [8] use machine learning to help split code between cores of a CPU and a GPU.
  - Uses a detailed analytic model with pattern-matching static analysis.
- Kremlin [6] provides a methodology to discover opportunities for parallelization in sequential code
  - Primarily by deriving upper bounds on potential parallelism from a dynamic critical path measurement.
- Parallel Prophet [16] estimates parallel performance in the presence of memory contention.
  - Based on instrumented, annotated serial programs.
- Hong et al. [11] model GPU performance from GPU memory access patterns and computational density per memory request
  - It requires a detailed model of the code to run on the device.