CLTune: A Generic Auto-Tuner for OpenCL Kernels

IEEE MCSoC
September 24, 2015

Cedric Nugteren (presenter), Valeriu Codreanu
Example: blur filter

Targets:
- GPUs
- Multi-core CPUs
- Other OpenCL-capable devices

Example: convolution

\[ B_{x,y} = w \cdot \sum_{i=-1}^{1} \sum_{j=-1}^{1} F_{i,j} A_{x+i,y+j} \]

example: 3 by 3 filter
OpenCL 2D convolution

$$B_{x,y} = w \cdot \sum_{i=1}^{i} \sum_{j=1}^{j} F_{i,j} A_{x+i,y+j}$$

Each thread: one output pixel

Thread coarsening (2D)?

Double for-loop

Unroll loops?

OpenCL work-group size?

Vector data-types?

Caching in local memory?
Search-space explosion

Large search-space:
- Not feasible to explore manually
- Perhaps not even feasible automatically?

3424 configurations

filter illegal configurations
Why do we need an auto-tuner?

**Large search-space:**
- Not feasible to explore manually
- Perhaps not even feasible automatically?

**Wide variety of devices:**
- Different optimal kernels
- Even from the same vendor

**User-parameter dependent:**
- Examples: matrix sizes, image size, filter sizes, etc.
Option 0: Full search

😊 Finds optimal solution
🚫 Explores all options

3424 configurations on Tesla K40m GPU

rotated histogram
mean

search space

performance [% of best-known]
Search strategies

Option 0: Full search
- Explores arbitrary fraction
- Performance varies

Option 1: Random search
- ☺ Explores arbitrary fraction
- ☹ Performance varies

Example: 107 out of 3424 configurations (1/32\textsuperscript{th})
Search strategies

Option 0: Full search
Option 1: Random search
Option 2: Simulated annealing

😊 Explores arbitrary fraction
😥 Performance varies
😢 Meta-parameter
😢 Local optima

Example: 107 out of 3424 configurations (1/32th)

Colours: 3 example runs
### Search strategies

**Option 0: Full search**
  - Explores arbitrary fraction

**Option 1: Random search**
  - Performance varies

**Option 2: Simulated annealing**
  - Meta-parameter

**Option 3: Particle swarm optimisation**
  - Local optima

---

**Example:** 107 out of 3424 configurations (1/32<sup>th</sup>)

**Colours:** 3 example runs

**Line-types:** 3 swarms
Search strategies evaluation

- **Performance (% of best-known)**: The performance of different search strategies is evaluated as a percentage of the best-known performance.

- **Random Search**: Performance is shown with black circles.
- **SA (Simulated Annealing)**: Performance is shown with red circles for different temperatures (T=2, T=4, T=6).
- **PSO (Particle Swarm Optimization)**: Performance is shown with blue circles for different swarm sizes (S=3, S=6).

- **Device**: The device used for the search is K40m.

- **Search Space**: Each search considers 107 out of 3424 configurations (1/32\textsuperscript{th}).

- **Average Best Result**: The average best result of 128 searches is highlighted.

- **Meta-Parameters**: Meta-parameters for SA and PSO are shown for comparison.

Each search: 107 out of 3424 configurations (1/32\textsuperscript{th})
Search strategies evaluation

Conclusions:
• Different per device
• PSO performs poorly
• Random search and SA perform well
## Conclusions:

- Different best parameters for different:
  - devices (see paper)
  - filter-sizes
- Performance equal or better than the state-of-the-art [1]

### Table: Parameters and Best Parameters

<table>
<thead>
<tr>
<th>Parameter(s)</th>
<th>Allowed Values</th>
<th>GeForce GTX480</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_{wg}$, $Y_{wg}$</td>
<td>{8,16,32,64}</td>
<td>64,8 32,8 32,8</td>
</tr>
<tr>
<td>$X_{wpt}$, $Y_{wpt}$</td>
<td>{1,2,4,8}</td>
<td>1,4 2,8 2,4</td>
</tr>
<tr>
<td>$L$</td>
<td>{0,1,2}</td>
<td>0 2 1</td>
</tr>
<tr>
<td>$VW$</td>
<td>{1,2,4,8}</td>
<td>1 2 2</td>
</tr>
<tr>
<td>$PAD$</td>
<td>{0,1}</td>
<td>0 0 0</td>
</tr>
<tr>
<td>$UNR$</td>
<td>{yes,no}</td>
<td>yes yes yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Filter Size</th>
<th>Best Parameters for</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>3x3</td>
</tr>
<tr>
<td>3x3</td>
<td>100%</td>
</tr>
<tr>
<td>7x7</td>
<td>65%</td>
</tr>
<tr>
<td>11x11</td>
<td>66%</td>
</tr>
</tbody>
</table>

GEMM case-study

Conclusions:
- Different best parameters for different devices
- Performance better than clBLAS, but not as good as assembly-tuned cuBLAS

<table>
<thead>
<tr>
<th>parameter(s)</th>
<th>allowed values</th>
<th>K40m</th>
<th>GTX480</th>
<th>HD7970</th>
<th>Iris</th>
</tr>
</thead>
<tbody>
<tr>
<td>$M_{wg}$, $N_{wg}$, $K_{wg}$</td>
<td>{16,32,64,128}</td>
<td>{16,16}</td>
<td>{8,16,32}</td>
<td>{yes,no}</td>
<td>{8,16,32}</td>
</tr>
<tr>
<td>$M_{dimC}$, $N_{dimC}$</td>
<td>{yes,no}</td>
<td>{yes}</td>
<td>{yes}</td>
<td>{yes}</td>
<td>{yes}</td>
</tr>
<tr>
<td>$L_{SA}$, $L_{SB}$</td>
<td>{16,32,64,128}</td>
<td>{32,16}</td>
<td>{32,32}</td>
<td>{32,32}</td>
<td>{32,32}</td>
</tr>
<tr>
<td>$M_{stride}$, $N_{stride}$</td>
<td>{yes,no}</td>
<td>{yes}</td>
<td>{yes}</td>
<td>{yes}</td>
<td>{yes}</td>
</tr>
<tr>
<td>$M_{vec}$, $N_{vec}$</td>
<td>{1,2,4,8}</td>
<td>{2,1}</td>
<td>{2,2}</td>
<td>{4,4}</td>
<td>{4,4}</td>
</tr>
<tr>
<td>$K_{wi}$</td>
<td>{1,2,4,8}</td>
<td>{8}</td>
<td>{8}</td>
<td>{2}</td>
<td>{8}</td>
</tr>
</tbody>
</table>

GEMM performance comparison
CLTune: A Generic Auto-Tuner for OpenCL Kernels

Auto-tuning OpenCL kernels:
• Large search-space
• Wide variety of devices
• User-parameter dependent

Advanced search strategies:
• Simulated annealing
• Particle swarm optimisation

Case-studies:
• Fastest 2D convolution
• Fast matrix-multiplication

Future: machine-learning [2]
• Train a model on a small subset
• Use the model to predict the remainder

Source-code on GitHub: https://github.com/CNugteren/CLTune