CLTune: A Generic Auto-Tuner for OpenCL Kernels

IEEE MCSoC
September 24, 2015

Cedric Nugteren (presenter), Valeriu Codreanu
Example: convolution

Example: blur filter

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

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

example: 3 by 3 filter
### OpenCL 2D Convolution

Each thread: one output pixel

**Thread coarsening (2D)?**

Double for-loop

**Unroll loops?**

Caching in local memory?

**OpenCL work-group size?**

**Vector data-types?**

---

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

```c
#define HFS (3)       // Half filter size
#define FS (HFS+HFS+1) // Filter size

__kernel void conv_reference(const int size_x, const int size_y,
                            const __global float* src,
                            __constant float* coeff,
                            __global float* dest) {

    const int tid_x = get_global_id(0);
    const int tid_y = get_global_id(1);

    float acc = 0.0f;

    // Loops over the neighbourhood
    for (int fx=-HFS; fx<=HFS; ++fx) {
        for (int fy=-HFS; fy<=HFS; ++fy) {
            const int index_x = tid_x + HFS + fx;
            const int index_y = tid_y + HFS + fy;

            // Performs the accumulation
            float coefficient = coeff[fy+HFS]*FS + (fx+HFS)];
            acc += coefficient * src[index_y*size_x + index_x];
        }
    }

    // Stores the result
    dest[tid_y*size_x + tid_x] = acc;
}
```
Large search-space:
- Not feasible to explore manually
- Perhaps not even feasible automatically?

Search-space explosion

16 \times 2 \times 16 \times 4 \times 5 = 10240

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.

---

### Vendor and Device Performance Table

| Vendor and Device Name | Architecture | Compiler and SDK | Peak GFLOPS | Peak GB/s | GFLOPS per GB/s |
|------------------------|--------------|------------------|-------------|-----------|-----------------|
| NVIDIA Tesla K40m      | Kepler       | CUDA 7.0         | 4291        | 288       | 14.9            |
| NVIDIA GeForce GTX480  | Fermi        | CUDA 5.5         | 1345        | 177       | 7.6             |
| AMD Radeon HD7970      | Tahiti       | APP 2.9          | 4368        | 288       | 15.1            |
| Intel Iris 5100        | Iris         | Apple 2.4.2      | 832         | 26        | 32.5            |

---

### Example Diagram

- Caching in local memory?
- OpenCL group size?
- Thread coarsening (2D)?
- Unroll loops?
- Vector data-types?
Option 0: Full search

😊 Finds optimal solution
😊 Explores all options

3424 configurations on Tesla K40m GPU

rotated histogram

mean

performance [% of best-known]

search space
Search strategies

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

Option 1: Random search

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

Colours: 3 example runs
Search strategies

Option 0: Full search
- ☺ Explores arbitrary fraction
- ☹ Performance varies
- ☹ Meta-parameter
- ☹ Local optima

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
Option 1: Random search
Option 2: Simulated annealing
Option 3: Particle swarm optimisation

- Explores arbitrary fraction
- Performance varies
- Meta-parameter
- Local optima

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

Colours: 3 example runs
Line-types: 3 swarms
Search strategies evaluation

- **Average best result** of 128 searches
- **Meta-parameters** for SA and PSO

Each search: 107 out of 3424 configurations (1/32th)
Conclusions:
• Different per device
• PSO performs poorly
• Random search and SA perform well
Convolution case-study

| parameter(s) | allowed values | GeForce GTX480 |
|--------------|----------------|----------------|
| $X_{wg}$, $Y_{wg}$ | \{8,16,32,64\} | 64,8 32,8 32,8 |
| $X_{wpt}$, $Y_{wpt}$ | \{1,2,4,8\} | 1,4 2,8 2,4 |
| $L$ | \{0,1,2\} | 0 2 1 |
| $V$ | \{1,2,4,8\} | 1 2 2 |
| $W$ | \{0,1\} | 0 0 0 |
| $P$ | \{yes,no\} | yes yes yes |

Conclusions:

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

[1]: B. Van Werkhoven, J. Maassen, H.E. Bal, and F.J. Seinstra. Optimizing Convolution Operations on GPUs Using Adaptive Tiling.
GEMM case-study

Conclusions:

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

[2]: T.L. Falch and A.C. Elster. Machine Learning Based Auto-tuning for Enhanced OpenCL Performance Portability.