Support Tools for Porting Legacy Applications to Multicore

Natsuki Kawai, Yuri Ardila, Takashi Nakamura, Yosuke Tamura
Agenda

→ Introduction

→ PEMAP: Performance Estimator for MAny core Processors
  ✓ The overview of PEMAP
  ✓ Estimation Methods
  ✓ Demonstration of PEMAP

→ BEMAP: BEnchMarks for Automatic Parallelization
  ✓ The overview of BEMAP
  ✓ Optimization Methods
  ✓ BEMAP as a benchmark for PEMAP
  ✓ BEMAP as a standalone benchmark
In the first step of software parallelization, we need to estimate performance benefit of parallelization. Also, this performance estimator needs a benchmark to examine its performance liability. We propose two development tools for them:

- **PEMAP**: Performance Estimator for MAny core Processors
- **BEMAP**: BEnchMark for Automatic Parallelization
PEMAP:
Performance Estimator for MAny-core Processors
PEMAP estimates performance benefits of parallelization of existing sequential programs

- Without any parallel programming

Users have to insert two annotations to target sequential programs

```c
void abs(float *d)
{
    int i;
    PEMAP_LOOP_START;
    for (i = 0; i < 1000000; i++) {
        d[i] = d[i] >= 0.0 ? d[i] : -d[i];
    }
    PEMAP_LOOP_END;
}
```
Performance of GPU programs is strongly depends on their calculation costs and memory accesses.

We propose to reconstruct them on GPU by existing methods of loop analysis.

- **PEMAP** does **NOT** analyze inter-loop data dependency because it is not important for performance estimation.
The first annotation calls PEMAP

PEMAP statically analyzes the target

Runs the target

- To collect actual branch directions and memory access patterns

Generates a dummy program

- It has virtually same performance as a hand-parallelized program
The analyzing method

1. Create graphs from the target programs by existing loop-analyzing methods
2. Pick up memory accesses and condition branches to collect
3. Traverse the graphs to generate dummy CUDA programs
The analyzing method

1. Create graphs from the target programs by existing loop-analyzing methods
2. Pick up memory accesses and condition branches to collect
3. Traverse the graphs to generate dummy CUDA programs
We selected Black-Scholes and Grayscale benchmarks from BEMAP

- Estimated the performance of sequential samples in BEMAP
- Ported OpenCL samples in BEMAP to CUDA
Cloud PEMAP Front Page

Cloud PEMAP is a web service version of a performance estimation tool PEMAP.

Language:
- C
- C++

Device:
- Quadro FX 3700

```c
void grayscale_geno(float r_out, const float *r_in, const int w)
{
    float rr, gg, bb;
    float v;
    int i, j;
    int idx;

    /* Pemap_loop_start */
    for (i = 0; i < h / 2; ++i) {
        for (j = 0; j < w / 2; ++j) {
            idx = i * w + j;
            rr = (float) r_in[idx];
            gg = (float) r_in[idx + w];
            bb = (float) r_in[idx + 2 * w];
            v = R_RATE * rr + G_RATE * gg + B_RATE * bb;
            t_out.data_r[idx] = (pixel_u8) v;
        }
    }
    /* Pemap_loop_end */
}
```

Estimate!!

Copyright 2013 Fixstars Corporation. Fixstars and the Fixstars logo are registered trademarks of Fixstars.
Cloud PEMAP Result

PEMAP succeeded to estimate the performance of your program. Please confirm the result table and PEMAP-result below.

You always can access this result from http://localhost:4567/68/.

Return to the top-page of Cloud PEMAP to estimate your next program.

The result of estimation

<table>
<thead>
<tr>
<th>GPU</th>
<th># threads</th>
<th>exec time</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>1</td>
<td>4.958791318000001ms</td>
</tr>
<tr>
<td>Quadro FX 3700</td>
<td>32</td>
<td>2.024175ms</td>
</tr>
</tbody>
</table>

The output of PEMAP (including output of your program before PEMAP was called)

```
num_all_reg_arg: 15
load0: 0
load1: 1
load2: 2
store0: 0
load0 is NOT a random access.
load1 is NOT a random access.
load2 is NOT a random access.
store0 is NOT a random access.
tmpxf1_00001dc0_00000000-14_dummy.i
Target GPU: Quadro FX 3700
nitr=1 nthread=32 : 2.024175[msec]
```
BEMAP: BEnchMark for Auto Parallelizer
Background

Aims to:

- Measure an auto-parallelizer tool’s performance and liability
- Provide a simple interface to conduct a comparison between reference code and parallelized code
- Provide a multi-platform benchmark using the OpenCL framework
A benchmark for parallelizer consisting of reference (single thread) codes and hand-tuned (parallel) OpenCL codes

An open-source project, can be downloaded from:

http://sourceforge.net/projects/bemap/

Currently consists of 8 algorithms:

- Black-Scholes for European Option
- Gaussian Blur
- Grayscale
- Linear Search
- Monte-Carlo for European Option
- Runlength Encoding
- Backprojection
- Scale Invariant Feature Transform (SIFT)
Has well-optimized OpenCL kernel codes

Both platform independent and platform dependent OpenCL kernels are provided

Optimization methods and benchmark results are well-documented

http://sourceforge.net/projects/bemap/files/Documentations/
Overview (cont.)

- BEMAP Hand-tuned Program
- OpenCL Compiler
- Parallel Program

- BEMAP Sequential Program
- Auto Parallelizer
- Compiler
- Parallel Program

- Speed Comparison
- Output Comparison
Optimization Methods

- SIMD (explicit vectorization)
  - Force the compiler to use wide registers to do a single operation for multiple data (e.g. Intel’s [X|Y]MM registers)

- Simple loop-unrolling

- Memory Access
  - Coalesced, Back-conflicted, Random

- Caching with shared memory
  - Programmable cache may boost up the memory transfer performance (DMA architecture)

- Memory Mapping
  - Mapped (Pinned) memory gives asynchronous memory access for the host-device communication

- Native Math Functions
  - Built-in math functions are provided in some platforms
BEMAP as a benchmark for PEMAP
# BEMAP as a standalone benchmark

Time unit is **milliseconds (ms)**, using BEMAP’s default parameters

<table>
<thead>
<tr>
<th>Workload</th>
<th>Ref Plat1</th>
<th>CPUOCL Plat1</th>
<th>GPUOCL Plat1</th>
<th>Ref Plat2</th>
<th>CPUOCL Plat2</th>
<th>GPUOCL Plat2</th>
</tr>
</thead>
<tbody>
<tr>
<td>BlackScholes</td>
<td>6109.07</td>
<td>32.05</td>
<td>1.70</td>
<td>1587.80</td>
<td>21.41</td>
<td>1.40</td>
</tr>
<tr>
<td>Gaussian</td>
<td>227.42</td>
<td>1.24</td>
<td>0.39</td>
<td>179.30</td>
<td>1.40</td>
<td>0.27</td>
</tr>
<tr>
<td>Grayscale</td>
<td>4.32</td>
<td>0.53</td>
<td>0.06</td>
<td>5.47</td>
<td>0.52</td>
<td>0.07</td>
</tr>
<tr>
<td>LinearSearch</td>
<td>26.25</td>
<td>8.42</td>
<td>47.14</td>
<td>33.33</td>
<td>14.24</td>
<td>24.08</td>
</tr>
<tr>
<td>MonteCarlo</td>
<td>193817.4</td>
<td>696.53</td>
<td>49.68</td>
<td>27423.70</td>
<td>264.49</td>
<td>41.50</td>
</tr>
<tr>
<td>Runlength</td>
<td>518.28</td>
<td>17.07</td>
<td>90.29</td>
<td>370.50</td>
<td>16.77</td>
<td>43.36</td>
</tr>
<tr>
<td>Backprojection</td>
<td>11297.43</td>
<td>195.03</td>
<td>75.83</td>
<td>11872.04</td>
<td>116.08</td>
<td>69.70</td>
</tr>
<tr>
<td>SIFT</td>
<td>1860.00</td>
<td>194.11</td>
<td>55.54</td>
<td>1452.00</td>
<td>170.51</td>
<td>48.02</td>
</tr>
</tbody>
</table>

**RefPlat1 & CPUOCLPlat1:**
Intel i7-X990 @ 3.47GHz (Nehalem)
8GB memory, 6 cores, 12 threads (HT)
**GPUOCLPlat1:**
NVIDIA GTX 570 @ 1.46GHz (Fermi GF-110)
480 CUDA cores, 1.2GB GDDR5 memory

**RefPlat2 & CPUOCLPlat2:**
Intel Core i7-3770K @ 3.50GHz (Ivy Bridge)
8GB memory, 4 cores, 8 threads (HT)
**GPUOCLPlat2:**
NVIDIA GTX 680 @ 1.06 GHz (Kepler GK-104)
1536 CUDA cores, 2GB GDDR5 memory
We proposed two development tools for parallelization.

PEMAP is an estimation tool of performance increase through parallelization.

Users of PEMAP need to do nothing other than inserting two annotations.

BEMAP is a benchmark suite to assist the development of an auto-parallelizer.