ANNOUNCEMENT

Paper number [5] and [7]
  will be presented by one person rather than a group of two

Send e-mail to jsmin@dcslab.snu.ac.kr
  about your preference to the papers
IF you have sent email about your preference and group, just ignore this announcement.

  16 15 8 2 ~ all 16 of them ~ 1.
CUDA Programming Assignment

Sunggon Kim
<skim@dcslab.snu.ac.kr>
Table Of Contents

CUDA Programming Introduction

Assignment

Machine For Assignment
## Multi-Core Rules

<table>
<thead>
<tr>
<th>Rank</th>
<th>Site</th>
<th>System</th>
<th>Cores</th>
<th>Rmax (TFlop/s)</th>
<th>Rpeak (TFlop/s)</th>
<th>Power (kW)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>National University of Defense Technology China</td>
<td>Tianhe-2 (MilkyWay-2) - TH-IVB-FEP Cluster, Intel Xeon E5-2692 12C 2.200GHz, TH Express-2, Intel Xeon Phi 31S1P NUDT</td>
<td>3,120,000</td>
<td>33,862.7</td>
<td>54,902.4</td>
<td>17,808</td>
</tr>
<tr>
<td>2</td>
<td>DOE/SC/Oak Ridge National Laboratory United States</td>
<td>Titan - Cray XK7, Opteron 6274 16C 2.200GHz, Cray Gemini interconnect, NVIDIA K20x Cray Inc.</td>
<td>560,640</td>
<td>17,590.0</td>
<td>27,112.5</td>
<td>8,209</td>
</tr>
<tr>
<td>3</td>
<td>DOE/NNSA/LLNL United States</td>
<td>Sequoia - BlueGene/Q, Power BQC 16C 1.60 GHz, Custom IBM</td>
<td>1,572,864</td>
<td>17,173.2</td>
<td>20,132.7</td>
<td>7,890</td>
</tr>
<tr>
<td>4</td>
<td>RIKEN Advanced Institute for Computational Science (AICS) Japan</td>
<td>K computer, SPARC64 VIIIfx 2.0GHz, Tofu interconnect Fujitsu</td>
<td>705,024</td>
<td>10,510.0</td>
<td>11,280.4</td>
<td>12,660</td>
</tr>
<tr>
<td>5</td>
<td>DOE/SC/Argonne National Laboratory United States</td>
<td>Mira - BlueGene/Q, Power BQC 16C 1.60GHz, Custom IBM</td>
<td>786,432</td>
<td>8,586.6</td>
<td>10,066.3</td>
<td>3,945</td>
</tr>
<tr>
<td>6</td>
<td>Texas Advanced Computing Center/Univ. of Texas United States</td>
<td>Stampede - PowerEdge C8220, Xeon E5-2680 8C 2.700GHz, Infiniband FDR, Intel Xeon Phi SE10P Dell</td>
<td>462,462</td>
<td>5,168.1</td>
<td>8,520.1</td>
<td>4,510</td>
</tr>
</tbody>
</table>
Graphic Processing

Extremely Computing Intensive Job

Also, Parallel Friendly
GPU: Graphic Processing Unit

- Specialized for
  - Compute Intensive, Highly Parallel Computation
GPU: Graphic Processing Unit

What If We Can Use GPU For General Purpose?
CUDA™

A General-Purpose Parallel Computing Platform and Programming Model

Introduced By NVIDIA, Nov 2006

Can Program GPU with standard programming languages such as C
Scalable Programming Model

- GPU is built around an array of SMs
  - Streaming multiprocessors
- Program is partitioned into blocks of threads
CUDA: The basic

• Terminology
  • Host – The CPU and its memory
  • Device – The GPU and its memory
Hello, World

• __global__ void kernel( void ) {

• } (__global__ = runs on the device)

• int main( void ) {
  • kernel<<1,1>>(); ← go run kernel function up there
  • printf( "Hello, World!\n" );
  • return 0;

• }
CUDA Program: Memory Hierarchy

Diagram showing the memory hierarchy with:
- Thread
- Per-thread local memory
- Thread Block
- Per-block shared memory
CUDA Program: Memory Hierarchy

Grid 0
- Block (0, 0)
- Block (1, 0)
- Block (2, 0)
- Block (0, 1)
- Block (1, 1)
- Block (2, 1)

Grid 1
- Block (0, 0)
- Block (1, 0)
- Block (0, 1)
- Block (1, 1)
- Block (0, 2)
- Block (1, 2)

Global memory
CUDA Program: Memory Hierarchy

- `cudaMalloc`
  - Allocate Device Memory

- `cudaMemcpy`
  - Transfer Data Between Device And Host
    - `cudaMemcpyDeviceToHost`
    - `cudaMemcpyHostToDevice`
    - ...and MORE
      - AsyncMemcpy, using streams

- `cudaFree`
  - Fairly simple. Just frees the memory
CUDA Program: Kernel Function

// Kernel definition
__global__ void add(int* A, int* B, int* C)
{
    *c = *a + *b;
}

REMEMBER to use pointer as the arguments.
A, B, C must point to device memory
int main()
{
    ...
    int hostA, hostB, hostC;
    int *a,*b,*c;
    cudaMalloc((void**)&a, sizeof(int); //allocate device mem
    cudaMemcpy(a, &hostA, sizeof(int), cudaMemcpyHostToDevice);
    // Kernel invocation with N threads
    VecAdd<<<1, 1>>>(a, b, c);
    cudaMemcpy(&hostC, c, sizeof(int), cudaMemcpyDeviceToHost);
    cudaFree(a,b,c);
}
Cuda Program: Kernel Function

● Normal Functions
  ○ Runs on CPU
  ○ Same With Standard Program

● Kernel Functions
  ○ Runs on GPU
  ○ Similar With Standard Program
CUDA Program: Kernel Function

Initialize → Do Job1 → Do Job2 → Do Job3 → Show Results
CUDA Program: Kernel Function

- Initialize
- Do Job1
- Do Job2
- Do Job3
- Show Results

Normal function

Kernel function
Parallel Programming

- Add\(<numBlocks>, \text{threadsPerBlock}\)

![Diagram showing parallel programming flow with numBlocks and threadsPerBlock connections.]
CUDA Program: Thread Hierarchy

All the threads of a block are expected to reside on the same processor core.

# of threads per block:
~ 2048
Block hierarchy

• __global__ void add( int *a, int *b, int *c ) {
  • c[blockIdx.x] = a[blockIdx.x] + b[blockIdx.x];
• }

```
Block 0
  c[0] = a[0] + b[0];

Block 2

Block 1
  c[1] = a[1] + b[1];

Block 3
```
CUDA Program: Thread Hierarchy

__global__ void add( int *a, int *b, int *c ) {

c[threadIdx.x] = a[threadIdx.x] + b[threadIdx.x];
}

CUDA Program: Build And Run

$ nvcc --output-file output sourcecode.cu
$ ./output

$ nvcc --help
Reference


Machine For Assignment

Server Address: 147.46.242.21
ID: cudateam<1-10>
PW: cuda

Email-me for your team name
Time Sharing

● Do Aggressive, Dynamic Time Sharing
  ○ Use Task Queueing Program (tasq)
  ○ Just Execute Program Using tasq

● Policy Can Be Changed Later
tasq: Task Queueing Program

Basic Usage:
$ tasq <enq | list> [command] [output]

Example:
$ tasq enq nvcc --output-file matmul matmul.cu output
$ tasq enq ./matmul output2
$ tasq list
Assignment

Implement DES Algorithm Using C and CUDA

Due Date: ??

Submit: skim@dcslab.snu.ac.kr

Mail Subject Should Contain:
[DIP2016_CUDA_TeamNo_Submit]
Question?