18-640 Foundations of Computer Architecture

Project 5 Overview

Mridula

November 19, 2014

Contents

- Overview
- GPGPU-Sim - Brief overview
- CUDA Programming model - Recap
- Tasks
- Deadline
- Resources

Overview

1. Optional Project - Grading will be equally distributed amongst all team members.
2. Students interested in doing the project can form new teams, if required.
3. Create your teams in https://docs.google.com/a/west.cmu.edu/spreadsheets/d/1dKhihPNXMmXcOzCOUIwo5eKZ9jLTbyC7oXla11tVxh8/edit#gid=0 before 11:59 pm on 11/21/2014.

Tasks

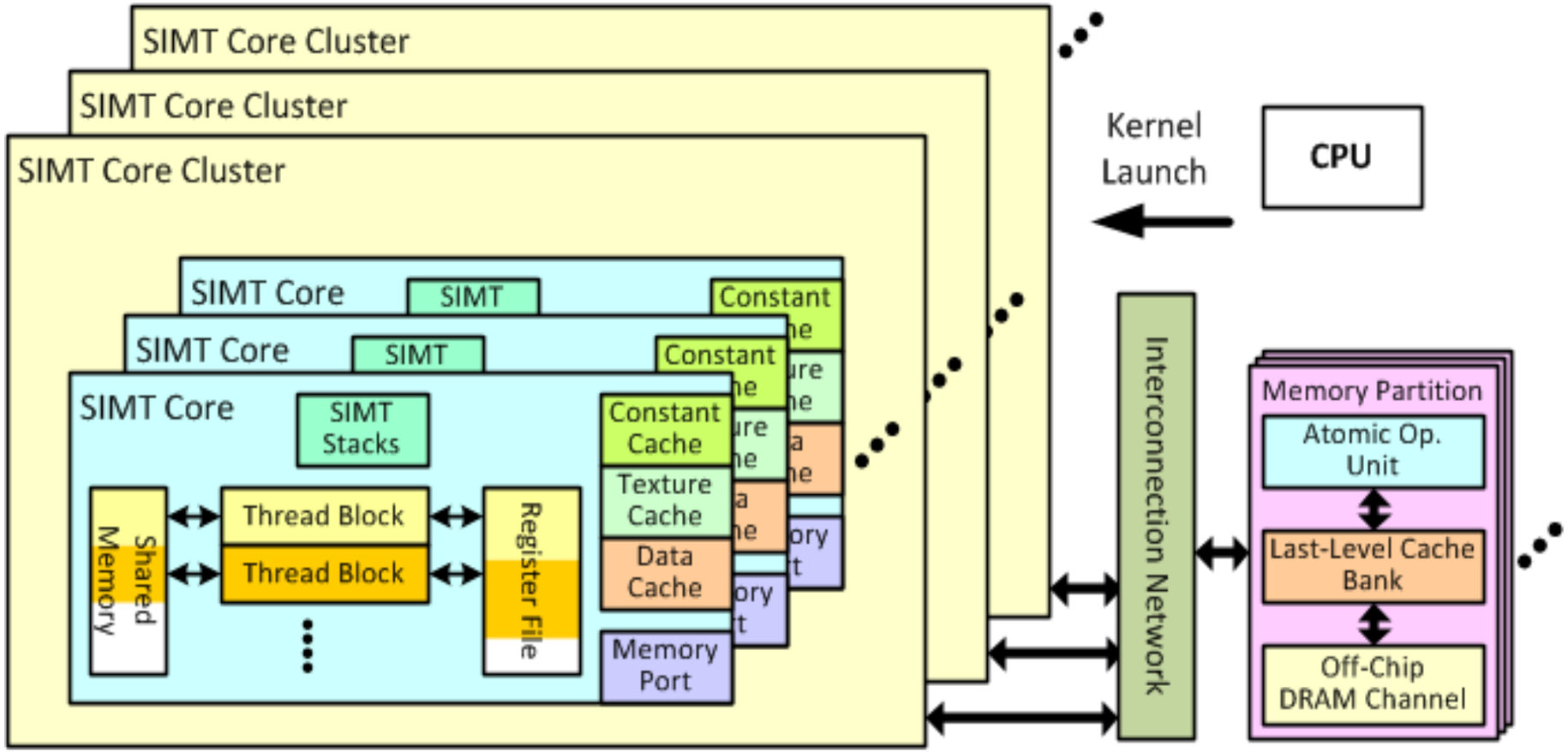
1. Setup GPGPU-Sim
2. Analyze given matrix multiplication program
3. Optimize the matrix multiplication program

GPGPU-Sim - A brief overview

1. GPGPU-Sim - A cycle-level GPU performance simulator that focuses on "GPU computing"
   Runs unmodified CUDA applications

2. Simulates:
   a.Timing model - relevant to gpu kernel
   b.Functional model for virtual ISA (PTX - Parallel thread execution) and Native ISA (SASS)
   c. Power model for the computation part of the GPU kernel which estimates power according to timing behavior

GPU Microarchitecture modeled by GPGPU-Sim



GPUWattch Power modeling

1. GPUWattch - An architectural-level power model for GPGPUs integrated with GPGPU-Sim
2. Models static power and dynamic power

GPGPU-Sim determines

- Which units are accessed
- How many times they are accessed

GPUWattch provides

- Per access energy of each unit

CUDA Programming model - Recap

Kernels - Functions that are executed in parallel by huge number of threads

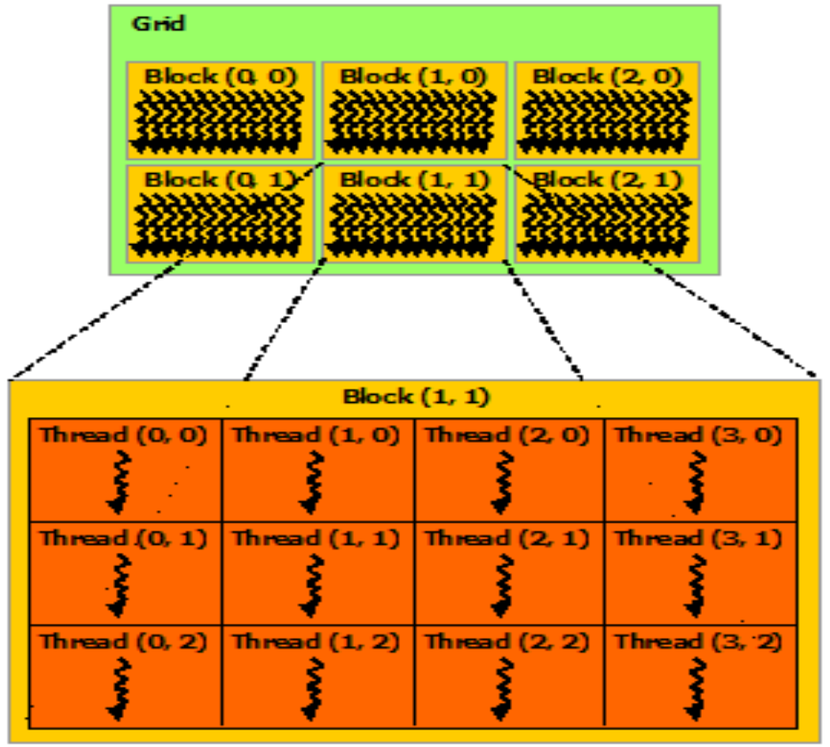
Eg:

```
// Kernel definition
__global__ void Add(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
int main()
{
    ...
    // Kernel invocation with N threads
    Add<<<1, N>>>(A, B, C);
    ...
}
```

From:: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#ixzz3JXXu0cBD

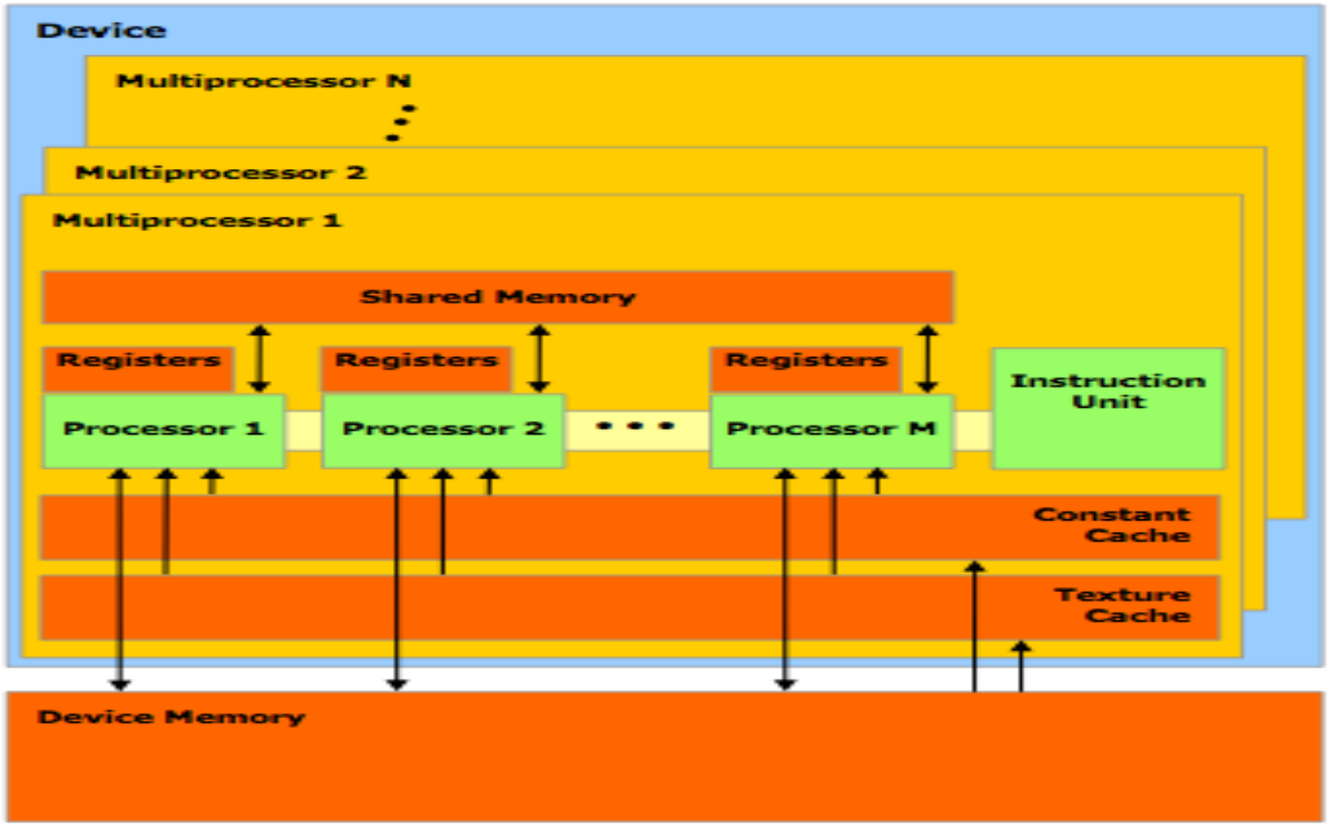
CUDA Programming model - Recap

Organization of Threads



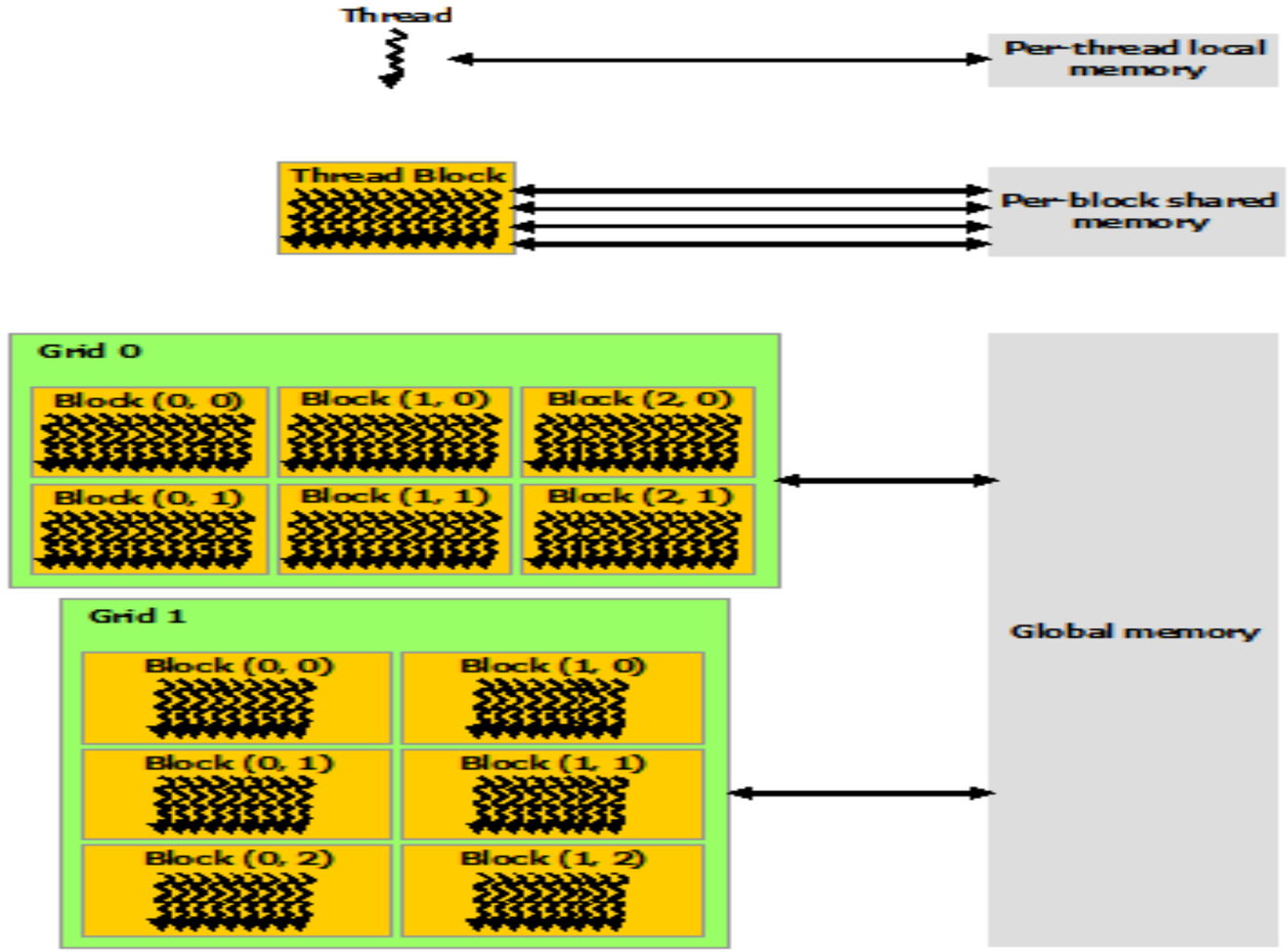
CUDA Programming model - Recap

Memory model



CUDA Programming model - Recap

CUDA Memory hierarchy



CUDA Program - Example

```
 // Device code
__global__ void VecAdd(float* A, float* B, float* C, int N){
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    if (i < N)
        C[i] = A[i] + B[i];
}
```

CUDA Program - Example

```
int main()
{
    int N = 32;
    size_t size = N * sizeof(float);

    // Allocate input vectors h_A and h_B in host memory
    float* h_A = (float*)malloc(size);
    float* h_B = (float*)malloc(size);

  // Initialize the vectors
  ……..
```

CUDA Program - Example

```
// Allocate vectors in device memory
float* d_A;
cudaMalloc(&d_A, size);
float* d_B;
cudaMalloc(&d_B, size);
float* d_C;
cudaMalloc(&d_C, size);

// Copy vectors from host memory to device memory
cudaMemcpy(d_A, h_A, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_B, h_B, size, cudaMemcpyHostToDevice);
```

CUDA Program - Example

```
    // Invoke kernel
    int threadsPerBlock = 256;
    int blocksPerGrid = (N + threadsPerBlock - 1) / threadsPerBlock;
    VecAdd<<<blocksPerGrid, threadsPerBlock>>>(d_A, d_B, d_C, N);
   // Copy result from device memory to host memory
    cudaMemcpy(h_C, d_C, size, cudaMemcpyDeviceToHost);
   // Free device memory
    cudaFree(d_A);
    cudaFree(d_B);
    cudaFree(d_C);
  //Free host memory
}
```

Read more at: http://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#ixzz3JXcx5H1u

Tasks

Task 1: Install GPGPU-Sim

- Detailed instructions are provided in the handout.
- Contact TA for any issues during installation.
- Currently, GPGPU-Sim can only be installed on a linux machine (preferably Ubuntu).

Tasks

Task 2: Analyze CUDA Matrix-multiplication application

- We have provided you a basic unoptimized version of matrix multiplication application.
- Copy config files from GPGPU-Sim (as described in handout)
- Compile and run the program
- Vary the parameters in the config file(number of registers , size of shared memory etc) and source code(block size, grid size etc) and observe the effect on performance and power

Tasks

Task 3: Optimize the matrix multiplication program

- Use atleast 2 optimization techniques to make the program run faster (Difference in speed is usually observed for large input matrices)
- Hint: Some techniques used in Project 4 could be helpful
- Optimize memory usage and instruction usage for better performance.
- Use your knowledge about thread hierarchy organization to maximize throughput.

Deadline

12/8/2014

11:59 pm EST

Resources

- http://gpuwattch.ece.utexas.edu/
- http://gpuwattch.ece.utexas.edu/resources/workshop/ispass-2013/slides/ISPASS_Tutorial_GPGPUSIM.pdf
- http://gpgpu-sim.org/manual/index.php/GPGPU-Sim_3.x_Manual
- http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz3JXWE
- http://docs.nvidia.com/cuda/cuda-c-best-practices-guide/#axzz3JXWEdhKg