CUDA Basics
Murphy Stein
New York University
Overview

- Device Architecture
- CUDA Programming Model
- Matrix Transpose in CUDA
- Further Reading
What is CUDA?

CUDA stands for:

"Compute Unified Device Architecture"

It is 2 things:

1. Device Architecture Specification
2. A small extension to C

= New Syntax + Built-in Variables – Restrictions + Libraries
Device Architecture: Streaming Multiprocessor (SM)

1 SM contains 8 scalar cores

- Up to 8 cores can run simultaneously
- Each core executes identical instruction set, or sleeps
- SM schedules instructions across cores with 0 overhead
- Up to 32 threads may be scheduled at a time, called a warp, but max 24 warps active in 1 SM
- Thread-level memory-sharing supported via Shared Memory
- Register memory is local to thread, and divided amongst all blocks on SM
Transparent Scalability

- Hardware is free to assign blocks to any processor at any time
  - A kernel scales across any number of parallel processors

Each block can execute in any order relative to other blocks.

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2009
SM Warp Scheduling

- SM hardware implements zero-overhead Warp scheduling
  - Warps whose next instruction has its operands ready for consumption are eligible for execution
  - Eligible Warps are selected for execution on a prioritized scheduling policy
  - All threads in a Warp execute the same instruction when selected

4 clock cycles needed to dispatch the same instruction for all threads in a Warp in G80

- If one global memory access is needed for every 4 instructions
- A minimal of 13 Warps are needed to fully tolerate 200-cycle memory latency

© David Kirk/NVIDIA and Wen-mei W. Hwu, 2007-2009
Device Architecture

- Host
- Block Execution Manager
- Streaming Multiprocessor (SM)
- Constant Memory 64KB
- Texture Memory
- Global Memory 768MB -- 4GB

1 GPU

Streaming Multiprocessor (SM) 1
Streaming Multiprocessor (SM) 2
Streaming Multiprocessor (SM) N-1
Streaming Multiprocessor (SM) N
C Extension

Consists of:

- New Syntax and Built-in Variables
- Restrictions to ANSI C
- API/Libraries
New Syntax:

- `<<<< ... >>>>
- `__host__, __global__, __device__`
- `__constant__, __shared__, __device`
- `__syncthreads()`
C Extension: Built-in Variables

Built-in Variables:

- **dim3 gridDim;**
  - Dimensions of the grid in blocks (*gridDim.z* unused)
- **dim3 blockDim;**
  - Dimensions of the block in threads
- **dim3 blockIdx;**
  - Block index within the grid
- **dim3 threadIdx;**
  - Thread index within the block
C Extension: Restrictions

New Restrictions:

- No recursion in device code
- No function pointers in device code
CUDA API

- CUDA Runtime (Host and Device)
- Device Memory Handling (cudaMalloc,...)
- Built-in Math Functions (sin, sqrt, mod, ...)
- Atomic operations (for concurrency)
- Data-types (2D textures, dim2, dim3, ...)

Compiling a CUDA Program

C/C++ Code → NVCC → Virtual PTX Code

C/C++ Code

gcc

CPU Instructions

PTX to Target Compiler → GPU Instructions

C/C++ CUDA Application

GPU Instructions
Matrix Transpose

$M_{i,j} \rightarrow M_{j,i}$
Matrix Transpose

\[
\begin{array}{cc}
A & B \\
C & D \\
\end{array}
\rightarrow
\begin{array}{cc}
A^T & C^T \\
B^T & D^T \\
\end{array}
\]
Matrix Transpose: First idea

- Each thread block transposes an equal-sized block of matrix M
- Assume M is square (n x n)
- What is a good block-size?
- CUDA places limitations on number of threads per block
- 512 threads per block is the maximum allowed by CUDA
Matrix Transpose: First idea

#include <stdio.h>
#include <stdlib.h>

__global__
void transpose(float* in, float* out, uint width) {
    uint tx = blockIdx.x * blockDim.x + threadIdx.x;
    uint ty = blockIdx.y * blockDim.y + threadIdx.y;
    out[tx * width + ty] = in[ty * width + tx];
}

int main(int args, char** vargs) {
    const int HEIGHT = 1024;
    const int WIDTH = 1024;
    const int SIZE = WIDTH * HEIGHT * sizeof(float);
    dim3 bDim(16, 16);
    dim3 gDim(WIDTH / bDim.x, HEIGHT / bDim.y);
    float* M = (float*)malloc(SIZE);
    for (int i = 0; i < HEIGHT * WIDTH; i++)
    { M[i] = i; }
    float* Md = NULL;
    cudaMemcpy((void**)&Md, M, SIZE, cudaMemcpyHostToDevice);
    float* Bd = NULL;
    cudaMemcpy((void**)&Bd, Md, SIZE, cudaMemcpyDeviceToHost);
    transpose<<<gDim, bDim>>>(Md, Bd, WIDTH);
    cudaMemcpy(M, Bd, SIZE, cudaMemcpyDeviceToHost);
    return 0;
}
Optimization Example: Matrix Transpose
Further Reading

- On-line Course:
  - UIUC NVIDIA Programming Course by David Kirk and Wen Mei W. Hwu
  - [http://courses.ece.illinois.edu/ece498/al/Syllabus.html](http://courses.ece.illinois.edu/ece498/al/Syllabus.html)
- CUDA@MIT '09
- Great Memory Latency Study:
  - "LU, QR and Cholesky Factorizations using Vector Capabilities of GPUs" by Vasily & Demmel
- Book of advanced examples:
  - "GPU Gems 3" Edited by Hubert Nguyen
- CUDA SDK
  - Tons of source code examples available for download from NVIDIA's website