University of Manchester

Valeriy Titarenko

School of Materials, University of Manchester, UK

Valeriy.Titarenko@manchester.ac.uk

Filtered backprojection, CPU and GPU

Input data: 4k $\times$ 6k; output data: 4k $\times$ 4k

Filtered backprojection, CPU and GPU

Input data: 4k × 6k; output data: 4k × 4k

CPU: Intel Q6600; time: 114 s (1 core) or 43 s (4 cores).

Filtered backprojection, CPU and GPU

Input data: 4k × 6k; output data: 4k × 4k

CPU: Intel Q6600; time: 114 s (1 core) or 43 s (4 cores).

GPU: Tesla C870

	2k × 2k	4k × 6k
1k × 1k	0.70	2.97
2k × 2k	1.48	5.09
3k × 3k	2.75	8.57
4k × 4k	4.50	13.69
5k × 5k	6.87	19.94

GPU (graphics processing unit)

GPU (graphics processing unit)

GPU (graphics processing unit)

CUDA (Compute Unified Device Architecture)

Tesla project

1 Multiprocessor = 8 Processors

1 Multiprocessor = 8 Processors

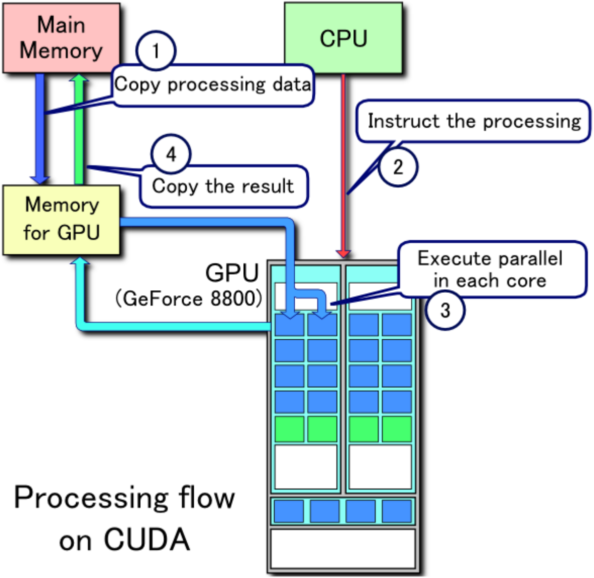
- Tesla C870 = 16 multiprocessors
- Tesla C1060, GeForce GTX 285, GTX 280 = 30 multiprocessors
- GeForce GTX 295 = 2 × 30 multiprocessors

1 Multiprocessor = 8 Processors

- Tesla C870 = 16 multiprocessors
- Tesla C1060, GeForce GTX 285, GTX 280 = 30 multiprocessors
- GeForce GTX 295 = 2 × 30 multiprocessors

Personal supercomputer

- 4 GeForce GTX 295 (240 multiprocessors), 4 × $560
- motherboard Asus P6T7 WS SuperComputer, $400
- memory 12 GB, $600
- Intel Xeon processor, $250
- case (for dual power supplies), $280
- power supplies, 2 × $230
- . . . Total ≈ $4600

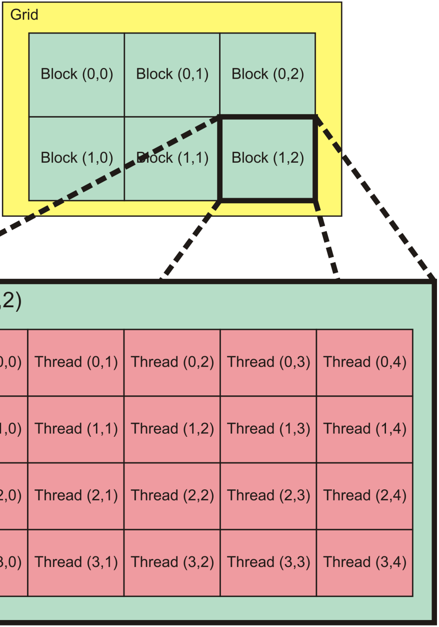


Processing flow on CUDA

Matrix addition, 1

```
// Kernel definition

__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
    int i = threadIdx.x;
    int j = threadIdx.y;
    C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(N, N);
    MatAdd<<<1, dimBlock>>>(A, B, C);
}
```

Threads, blocks, grid



Grid

Block (0,0)	Block (0,1)	Block (0,2)
Block (1,0)	Block (1,1)	Block (1,2)

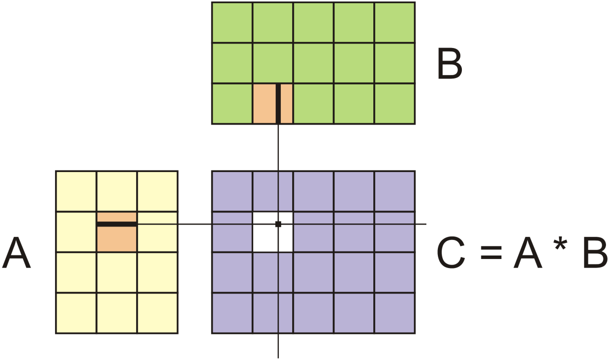
Block (1,2)

Thread (0,0)	Thread (0,1)	Thread (0,2)	Thread (0,3)	Thread (0,4)
Thread (1,0)	Thread (1,1)	Thread (1,2)	Thread (1,3)	Thread (1,4)
Thread (2,0)	Thread (2,1)	Thread (2,2)	Thread (2,3)	Thread (2,4)
Thread (3,0)	Thread (3,1)	Thread (3,2)	Thread (3,3)	Thread (3,4)

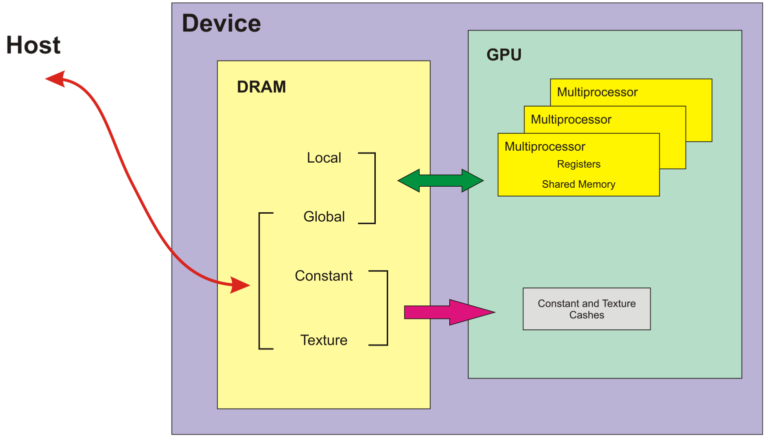
Matrix addition, 2

```
// Kernel definition

__global__ void MatAdd(float A[N][N], float B[N][N], float C[N][N])
{
    int i = blockIdx.x * blockDim.x + threadIdx.x;
    int j = blockIdx.y * blockDim.y + threadIdx.y;
    if (i < N && j < N)
        C[i][j] = A[i][j] + B[i][j];
}

int main()
{
    // Kernel invocation
    dim3 dimBlock(16, 16);
    dim3 dimGrid((N + dimBlock.x – 1) / dimBlock.x, (N + dimBlock.y – 1) / dimBlock.y);
    MatAdd<<<dimGrid, dimBlock>>>(A, B, C);
}
```

Matrix multiplication



GPU memory

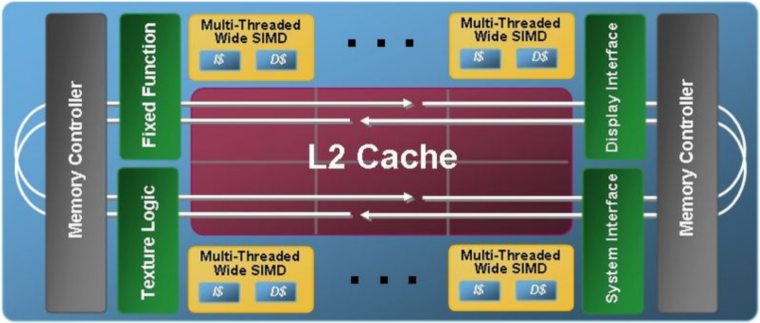


Host
Device
DRAM
Local
Global
Constant
Texture
GPU
Multiprocessor
Multiprocessor
Multiprocessor
Registers
Shared Memory
Constant and Texture Cashes

- Visual Fortran and C++ Compilers
  OS: Windows, Linux, and Mac OS X
- Threading Building Blocks (TBB)
  C++ template library that abstracts threads to tasks to create reliable, portable, and scalable parallel applications
- Math Kernel Library (MKL)
  a library of highly optimized, extensively threaded math routines for science, engineering, and financial applications that require maximum performance
- Integrated Performance Primitives (IPP)

nVidia vs Intel (future)

- Intel’s Larrabee



- nVidia’s Fermi