CUDA Introduction

Piotr Luszczek
Per-Core CPU Performance

Individual core speed no longer increases
GPU vs. CPU Performance over Years

- **Gaming-oriented GPUs**
- **Compute-oriented GPUs**

Core speed * core count
GPU and GPGPU: The Origin Story

- Programmable graphics pipeline
  - GLSL (shader language)

- Interpolation vs. dynamic range
  - Colors in graphics look better in floating-point

- Early attempts at programming
  - Cg, Brook, ...

- Modern standards or de facto standards
  - CUDA (currently 11)
    - Compute Unified Device Architecture
  - OpenCL (currently 2), SYCL, DPC++

- High-level languages
  - OpenMP 4.5 and offload directives
  - OpenACC (like OpenMP but GPU oriented)
**Hardware: CPU vs. GPU**

**Main Memory RAM:** DDR3 or DDR4  
- Size: ~500 GiB  
- Bus width: 64 bits  
- Speed: ~100 GB/s

**PCIexpress:**  
- Bandwidth: 15 GB/s

**GPU Memory RAM:** HSM2  
- Size: ~10s GiB  
- Bus width: 4096 bits  
- Speed: ~1000 GB/s

**SLI**  
**NVLink**

Long latency
// parallel function (GPU)
__global__ void sum(double x, double y, double *z) { *z = x + y; }

// sequential function (CPU)
void sum_cpu(double x, double y, double *z) { *z = x + y; }

// sequential function (CPU)
int main(void) {
  double *dev_z, hst_z;

  cudaMalloc( &dev_z, sizeof(double) );

  // launch parallel code (CPU → GPU)
  sum<<<1,1>>>(2.0, 3.0, dev_z);

  cudaMemcpy( &hst_z, dev_z, sizeof(double), cudaMemcpyDeviceToHost );
  printf("%g\n", hst_z[i]);

  cudaFree(dev_z);

  return 0;
}
Software: CPU + GPU

CPU code

GPU code

Ideal: CPU and GPU codes overlap
CUDA Basics: Blocks, Grids, and Threads

Piotr Luszczek
Minimal CUDA Code Example

```c
__global__ void sum(double x, double y, double *z) {
  *z = x + y;
}

int main(void) {
  double *device_z, host_z;

  cudaMalloc( &device_z, sizeof(double) );

  sum<<<1,1>>>(2.0, 3.0, device_z);

  cudaMemcpy( &host_z, device_z, sizeof(double), cudaMemcpyDeviceToHost );

  printf("%g\n", host_z);

  cudaFree(device_z);

  return 0; }
```

```
$ nvcc sum.cu -o sum
$ ./sum
5
```
Introducing Parallelism to CUDA Code

- Two points where parallelism enters the code
  - Kernel invocation
    - `sum<<< 1,1>>>( a, b, c )`
    - `sum<<<10,1>>>( a, b, c )`
  - Kernel execution
    - `__global__ void sum(double *a, double *b, double*c)`
    - `c[blockIdx.x] = a[blockIdx.x] + b[blockIdx.x]`

- CUDA makes the connection between:
  - invocation “sum<<<10,1>>>” and
  - execution and its index “blockIdx.x”

- Recall GPU massive parallelism
  - Many CUDA cores
  - Many CUDA threads
  - Many GPU SM (or SMX) units
CUDA Parallelism with Blocks

```c
int N = 100, SN = N * sizeof(double);
__global__ void sum(double *a, double *b, double *c) {
   c[blockIdx.x] = a[blockIdx.x] + b[blockIdx.x]; // no loop!
}

int main(void) {
   double *dev_a, *dev_b, *dev_c, *hst_a, *hst_b, *hst_c;

   cudaMalloc(&dev_a, SN); hst_a = calloc(N, sizeof(double));
   cudaMalloc(&dev_b, SN); hst_b = calloc(N, sizeof(double));
   cudaMalloc(&dev_c, SN); hst_c = malloc(N, sizeof(double));

   cudaMemcpy(dev_a, hst_a, SN, cudaMemcpyHostToDevice);
   cudaMemcpy(dev_b, hst_b, SN, cudaMemcpyHostToDevice);
   sum<<<10,1>>>(dev_a, dev_b, dev_c); // only 10 elements will be used
   cudaMemcpy(&hst_c, dev_c, SN, cudaMemcpyDeviceToHost);

   for (int i=0; i<10; ++i) printf("%g\n", hst_c[i]);

   cudaFree(dev_a); free(hst_a);
   cudaFree(dev_b); free(hst_b);
   cudaFree(dev_c); free(hst_c);
   return 0; }
```
## Details on Execution of Blocks on GPU

- Blocks is a level of parallelism
- There are other levels
- Blocks execute in parallel
- Synchronization is
  - Explicit (special function calls, etc.)
  - Implicit (memory access, etc.)
  - Mixed (atomics, etc.)
- Total number of available blocks is hardware specific
- CUDA offers inquiry functions to get the maximum block count

<table>
<thead>
<tr>
<th>// BLOCK 0</th>
<th>// BLOCK 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>c[0]=a[0]+b[0];</td>
<td>c[1]=a[1]+b[1];</td>
</tr>
<tr>
<td>// BLOCK 2</td>
<td>// BLOCK 3</td>
</tr>
<tr>
<td>// BLOCK 4</td>
<td>// BLOCK 5</td>
</tr>
<tr>
<td>// BLOCK 6</td>
<td>// BLOCK 7</td>
</tr>
<tr>
<td>// BLOCK 8</td>
<td>// BLOCK 9</td>
</tr>
</tbody>
</table>
Adding Thread Parallelism to CUDA Code

- **Kernel invocation**
  - `sum<<<10, 1>>>( x, y, z ) // block-parallel`
  - `sum<<< 1,10>>>( x, y, z ) // thread-parallel`

- **Kernel execution**
  - `z[threadIdx.x] = x[threadIdx.x] + y[threadIdx.x]`

- **Consistency of syntax**
  - Minimum changes to switch from blocks to threads
  - Similar naming for blocks and threads
CUDA Parallelism with Threads

```c
int N = 100, SN = N * sizeof(double);

__global__ void sum(double *a, double *b, double *c) {
    c[threadIdx.x] = a[threadIdx.x] + b[threadIdx.x]; // no loop!
}

int main(void) {
    // sequential function (CPU)
    double *dev_a, *dev_b, *dev_c, *hst_a, *hst_b, *hst_c;

    cudaMalloc( &dev_a, SN ); hst_a = calloc(SN);
    cudaMalloc( &dev_b, SN ); hst_b = calloc(SN);
    cudaMalloc( &dev_c, SN ); hst_c = malloc(SN);

    cudaMemcpy( dev_a, hst_a, SN, cudaMemcpyHostToDevice);
    cudaMemcpy( dev_b, hst_b, SN, cudaMemcpyHostToDevice);

    sum<<<1,10>>>(dev_a, dev_b, dev_c);

    cudaMemcpy( &hst_c, dev_c, SN, cudaMemcpyDeviceToHost );

    for (int i=0; i<10; ++i) printf("%g\n", hst_c[i]);

    cudaFree(dev_a); free(hst_a);
    cudaFree(dev_b); free(hst_b);
    cudaFree(dev_c); free(hst_c);
    return 0;
}
```
More on Block and Thread Parallelism

- When to use blocks and when to use threads?
  - Synchronization between threads is cheaper
  - Blocks have higher scheduling overhead

- Block and thread parallelism can be combined
  - Often it is hard to get good balance between both
  - Exact combination depends on
    - GPU generation
      - Tesla, Fermi, Kepler, Maxwell, Pascal, Volta, Turing, Ampere, ...
    - SM/SMX configuration
    - Memory size
Thread Identification Across APIs

- **POSIX threads**
  - `pthread_t tid = pthread_self();`

- **MPI**
  - `MPI_Comm_rank(comm, &rank);`
  - `MPI_Comm_size(comm, &size);`

- **OpenMP**
  - `int tid = omp_get_thread_num();`
  - `int all = omp_get_num_threads();`

- **CUDA**
  - `int blkid = blockIdx.x + (blockIdx.y + blockIdx.z * gridDim.y) * gridDim.x`
  - `int inside_blk_tid = threadIdx.x + (threadIdx.y + threadIdx.z * blockDim.y) * blockDim.x`