Parallel Programming (English)

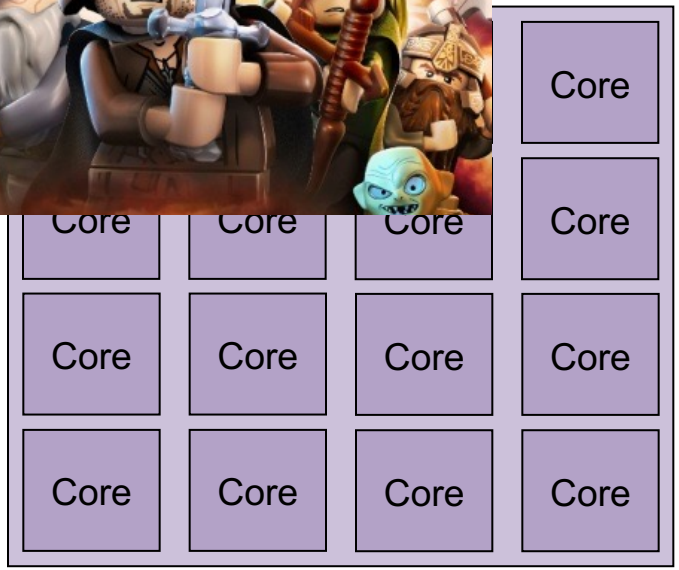
(Week 2)

Weifeng Liu

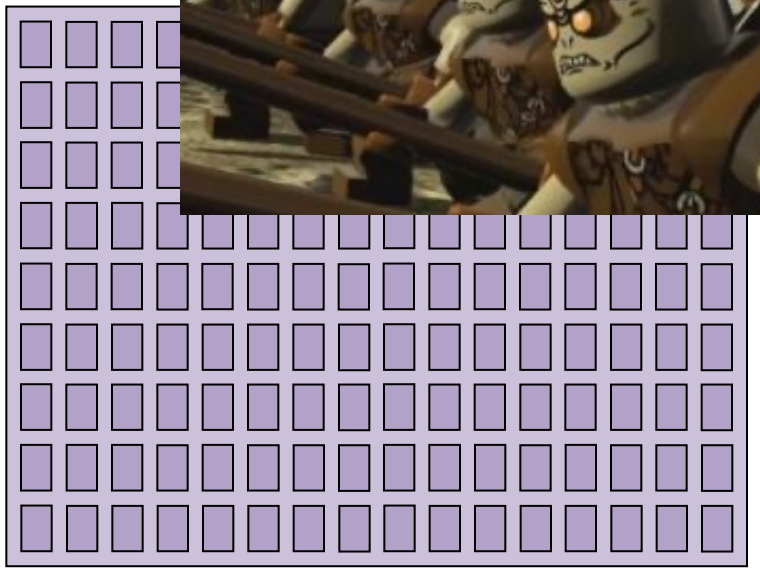
Department of Computer Science and Technology

China University of Petroleum - Beijing

CPUs and GPUs – Which is faster?

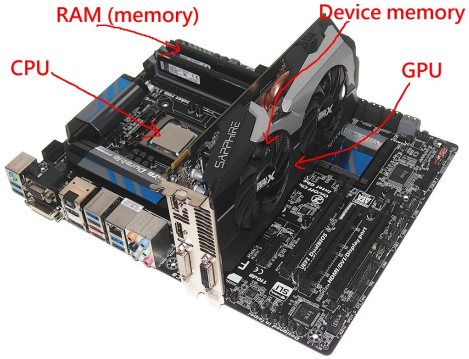


CPU: 16 cores @ 2.2 GHz



GPU: 6912 cores @ 1.8 GHz

CPU-GPU heterogeneous system and GPU Computing



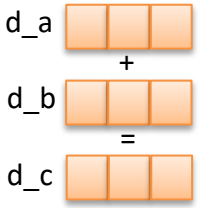
PC

Laptop

Bitcoin farm

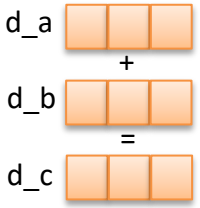
CUDA Vector Addition Code (host code)

```
line 01: #include <stdio.h>
line 02: #include <stdlib.h>
line 03: #include <string.h>
line 04: #include <sys/time.h>
line 05:
line 06: int main(int argc, char ** argv)
line 07:{
line 08:    struct timeval t1, t2;
line 09:    int repeat = 100;
line 10:    int n = atoi(argv[1]);
line 11:
line 12:    // create vectors
line 13:    int *a = (int *)malloc(sizeof(int) * n);
line 14:    int *b = (int *)malloc(sizeof(int) * n);
line 15:    int *c = (int *)malloc(sizeof(int) * n);
line 16:
line 17:    double data = 3 * n * sizeof(int) / (double)1e9;
line 18:    printf("vector length = %i, data volumn = %f GB\n", n, data);
```



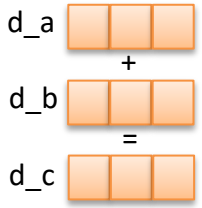
CUDA Vector Addition Code (host code) cont.

```
line 19:  // cuda code for vector addition
line 20:  // step 1. malloc GPU memory for the three vectors
line 21:  int *d_a;    int *d_b;    int *d_c;
line 22:  cudaMalloc((void **)&d_a, n * sizeof(int));
line 23:  cudaMalloc((void **)&d_b, n * sizeof(int));
line 24:  cudaMalloc((void **)&d_c, n * sizeof(int));
line 25:
line 26:   // step 2. copy the input data from CPU memory to GPU memory
line 27:   cudaMemcpy(d_a, a, n * sizeof(int),   cudaMemcpyHostToDevice);
line 28:   cudaMemcpy(d_b, b, n * sizeof(int),   cudaMemcpyHostToDevice);
```



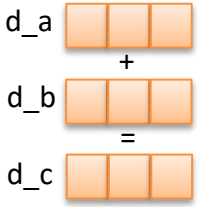
CUDA Vector Addition Code (host code) cont.

```
line 29:    // step 3. call CUDA kernel
line 30:    gettimeofday(&t1, NULL);
line 31:    for (int ri = 0; ri < repeat; ri++)
line 32:    {
line 33:        int num_threads = 128;
line 34:        int num_blocks = ceil ((double)n / (double)num_threads);
line 35:        vecadd_cuda<<< num_blocks, num_threads >>>(d_c, d_a, d_b, n);
line 36:    }
line 37:    cudaDeviceSynchronize();
line 38:    gettimeofday(&t2, NULL);
line 39:    double time_cuda = (t2.tv_sec - t1.tv_sec) * 1000.0 + (t2.tv_usec - t1.tv_usec) / 1000.
line 40:    time_cuda /= repeat;
line 41:    double bw_cuda = data/(time_cuda/1000.0);
line 42:    printf("CUDA   CODE takes %4.2f ms, bandwidth is %4.2f GB/s\n", time_cuda, bw_cuda)
line 43:
line 44:    // step 4. copy the output data from GPU memory to CPU memory
line 45:    cudaMemcpy(c, d_c, n * sizeof(int),   cudaMemcpyDeviceToHost);
```



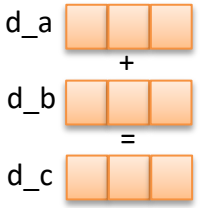
CUDA Vector Addition Code (host code) cont.

```
line 46:    // step 5. free GPU memory
line 47:    cudaFree(d_a);
line 48:    cudaFree(d_b);
line 49:    cudaFree(d_c);
line 50:
line 51:    free(a);
line 52:    free(b);
line 53:    free(c);
line 54:
line 55:    return 0;
line 56:}
```



CUDA Vector Addition Code (device code)

```
// CUDA parallel code
__global__
void vecadd_cuda(int *d_c, int *d_a, int *d_b, int n)
{
    const int tid = blockIdx.x * blockDim.x + threadIdx.x;
    if (tid < n)
    {
        d_c[tid] = d_a[tid] + d_b[tid];
    }
}
```

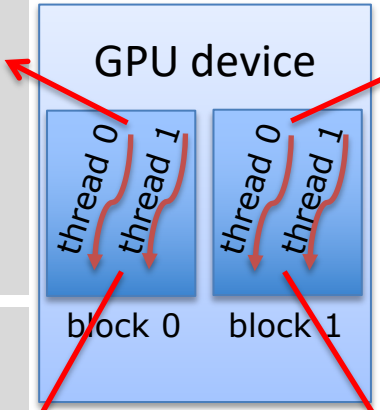



```
// C serial code
void vecadd_c_serial(int *c, int *a, int *b, int n)
{
    for (int i = 0; i < n; i++)
    {
        c[i] = a[i] + b[i];
    }
}
```

CUDA Vector Addition Code SPMD model

```
// CUDA parallel code
__global__
void vecadd_cuda(int *d_c, int *d_a, int *d_b, int n) //n=3
{
    const int tid = blockIdx.x * blockDim.x + threadIdx.x;
    // const int tid = 0 * 2 + 0 = 0;
    if (tid < n)
    // if (0 < 3)
    {
        d_c[tid] = d_a[tid] + d_b[tid];
        // d_c[0] = d_a[0] + d_b[0];
    }
}
```

```
// CUDA parallel code
__global__
void vecadd_cuda(int *d_c, int *d_a, int *d_b, int n) //n=3
{
    const int tid = blockIdx.x * blockDim.x + threadIdx.x;
    // const int tid = 1 * 2 + 0 = 2;
    if (tid < n)
    // if (2 < 3)
    {
        d_c[tid] = d_a[tid] + d_b[tid];
        // d_c[2] = d_a[2] + d_b[2];
    }
}
```




```
// CUDA parallel code
__global__
void vecadd_cuda(int *d_c, int *d_a, int *d_b, int n) //n=3
{
    const int tid = blockIdx.x * blockDim.x + threadIdx.x;
    // const int tid = 0 * 2 + 1 = 1;
    if (tid < n)
    // if (1 < 3)
    {
        d_c[tid] = d_a[tid] + d_b[tid];
        // d_c[1] = d_a[1] + d_b[1];
    }
}
```

```
// CUDA parallel code
__global__
void vecadd_cuda(int *d_c, int *d_a, int *d_b, int n) //n=3
{
    const int tid = blockIdx.x * blockDim.x + threadIdx.x;
    // const int tid = 1 * 2 + 1 = 3;
    if (tid < n)
    // if (3 < 3)
    {
        d_c[tid] = d_a[tid] + d_b[tid];
        // nothing happens in the loop
    }
}
```

CUDA GEMM Code SPMD model on 2D threadblock

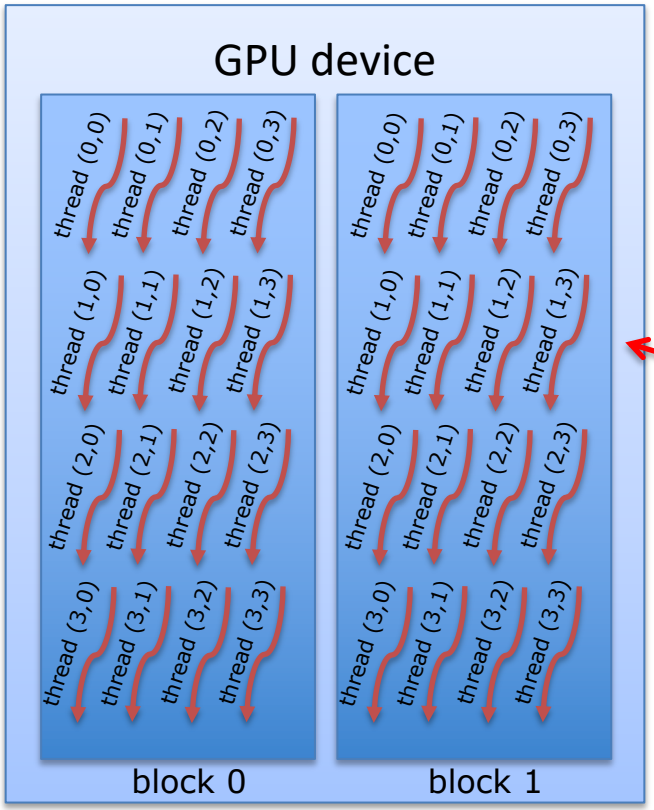
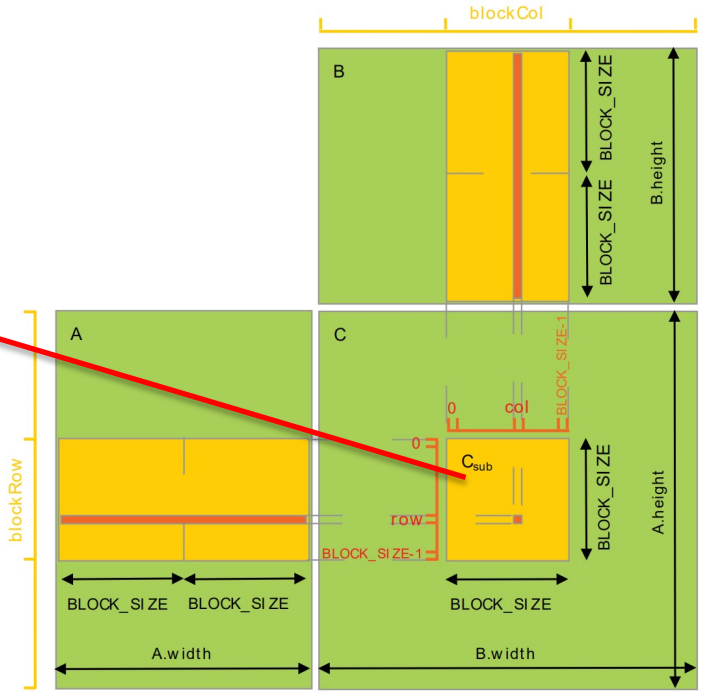


Figure 8. Matrix Multiplication with Shared Memory



Page 32, CUDA C++ Programming Guide (v11.3)

Thanks!