GPU computing discussion group 1/6/12

Introductory thoughts :

- I have ~ 4 months of experience with CUDA C
- I am sold ! (naive programming gives 30–50 x speed)
- I am not an expert !

These meetings :

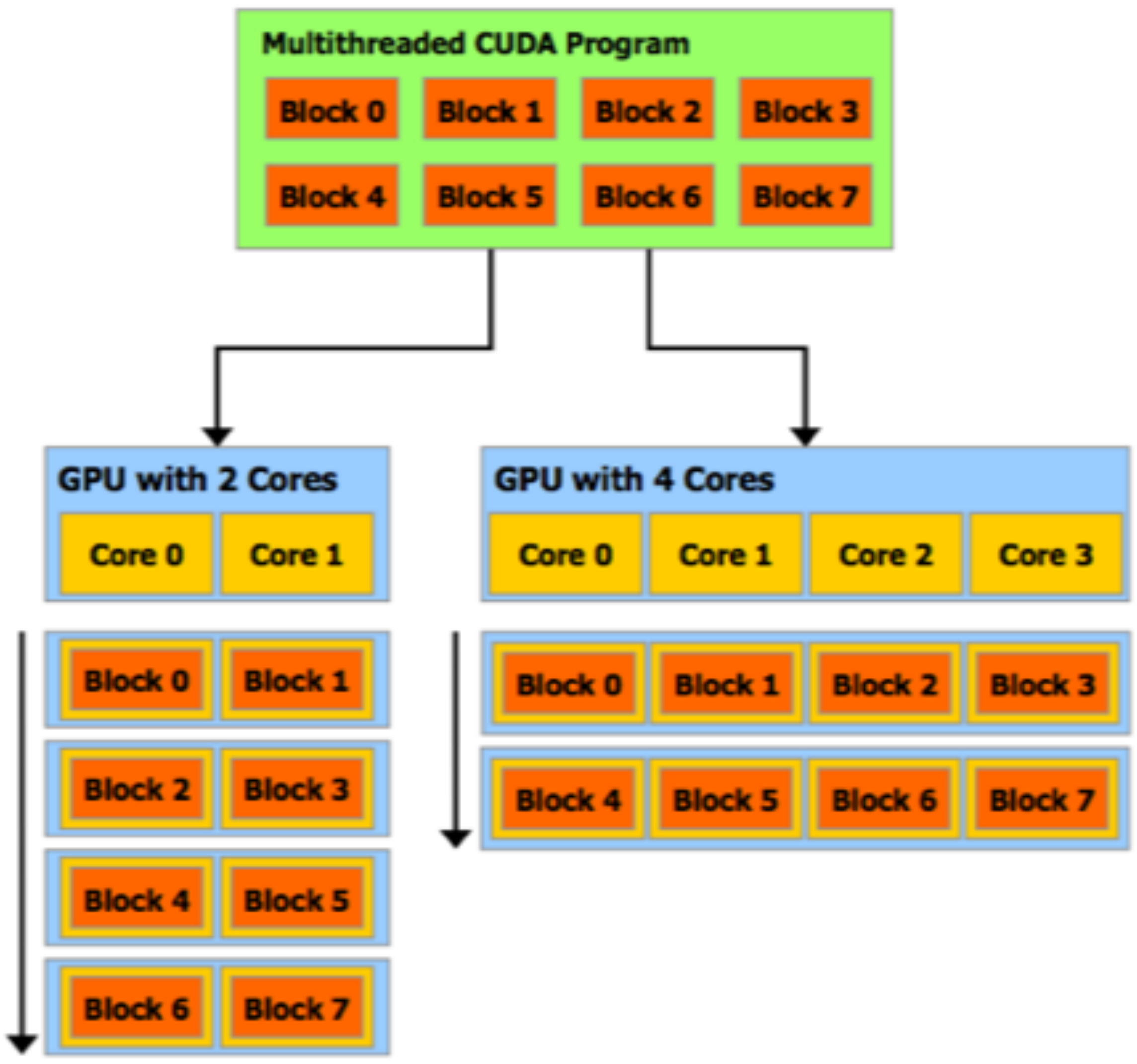
- NOT a class ! (Students are expected to read by themselves)
- NOT about stat methodology !

- about EFFICIENT computing .

GPU: (Graphic Processor Unit)

- parallel
- multithreaded
- many core
- v. high memory bandwidth.

weathertop.stat.osu.edu has 2 NVIDIA Tesla C2050 cards each with 448 cores and 3 GB of mem.

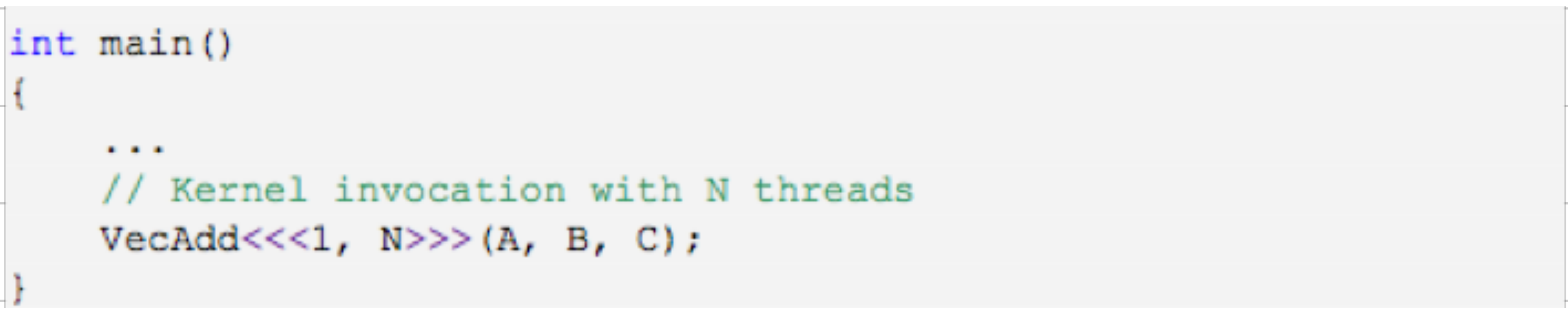


CUDA C:

- introduced by NVIDIA in 2006; open source;
- C-like programming language;
- alternatives: Fortran, OpenCL, etc.

Programming model:

$$C(i) = A(i) + B(i) \qquad i = 1, 2, \dots, N$$



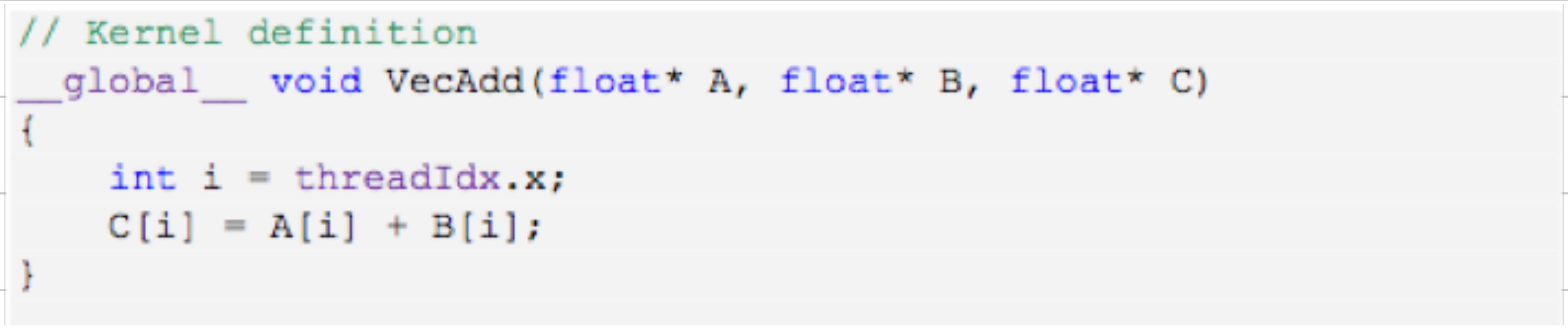
```
int main()
{
    ...
    // Kernel invocation with N threads
    VecAdd<<<1, N>>>(A, B, C);
}
```

VecAdd is a function (user-defined) called a Kernel.

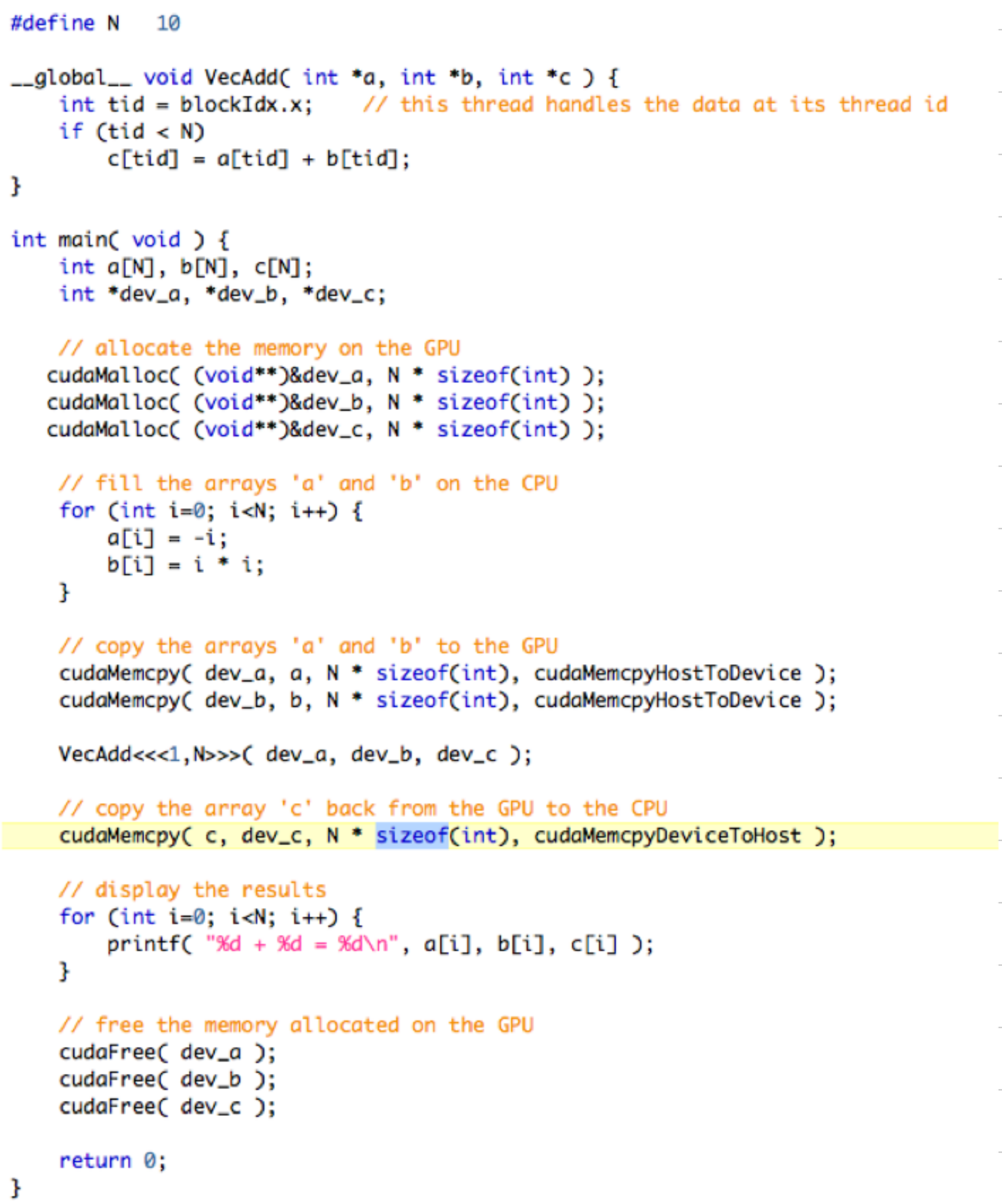
- 3 obvious arguments : A, B, C
- <<<1, N>>> means that N copies of this function are executed in parallel
  - the i'th copy will compute $A(i) + B(i)$ and store it in $C(i)$

- How do we know which one is the i'th copy ?

- At execution, a variable named threadIdx is created
  and it contains the desired information



```
// Kernel definition
__global__ void VecAdd(float* A, float* B, float* C)
{
    int i = threadIdx.x;
    C[i] = A[i] + B[i];
}
```



```
#define N   10

__global__ void VecAdd( int *a, int *b, int *c ) {
    int tid = blockIdx.x;    // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}

int main( void ) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;

    // allocate the memory on the GPU
   cudaMalloc( (void**)&dev_a, N * sizeof(int) );
   cudaMalloc( (void**)&dev_b, N * sizeof(int) );
   cudaMalloc( (void**)&dev_c, N * sizeof(int) );

    // fill the arrays 'a' and 'b' on the CPU
    for (int i=0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }

    // copy the arrays 'a' and 'b' to the GPU
    cudaMemcpy( dev_a, a, N * sizeof(int), cudaMemcpyHostToDevice );
    cudaMemcpy( dev_b, b, N * sizeof(int), cudaMemcpyHostToDevice );

    VecAdd<<<1,N>>>( dev_a, dev_b, dev_c );

    // copy the array 'c' back from the GPU to the CPU
    cudaMemcpy( c, dev_c, N * sizeof(int), cudaMemcpyDeviceToHost );

    // display the results
    for (int i=0; i<N; i++) {
        printf( "%d + %d = %d\n", a[i], b[i], c[i] );
    }

    // free the memory allocated on the GPU
    cudaFree( dev_a );
    cudaFree( dev_b );
    cudaFree( dev_c );

    return 0;
}
```

compile with: `nvcc -o addvectors addvectors.cu`