# CUDA: Основы

# План семинара

1. Обзор CUDA
2. Простая программа: сложение векторов
3. Параллельность. Grid
4. Elapsed time
5. Ручное управление памятью
6. Полезные материалы

# Доступ к кластеру

```
ssh USER@93.175.29.112
```

Можно включить аутентификацию по открытому ключу.

# Обзор

- CUDA — API для вычислений на GPU Nvidia
  - На C, C++, Fortran, Java, Python, DirectX, OpenACC

- С 2006 года. GeForce 8 series и все последующие GPU Nvidia

- SIMT: **Single instruction, multiple threads**
  - GPU состоит из нескольких (~20) **SM**: Streaming Multiprocessor
  - Каждый SM исполняет одновременно **много потоков**
  - Потоки разделены на группы по 32 — **warp'ы**
    - На весь warp инструкция одна! (до Volta, 2017)

- **NUMA**: host (CPU) и device (GPU)
  - Передача между ними — очень медленная (16 vs 898 GB/s)
- У device тоже есть разная память

# Принцип

1. **Отправить** данные на device
2. Выполнить **вычисления**
3. **Отправить** результат на host

# Сложение векторов

```
/* Kernel: исполняется на device, можно вызвать с host */
__global__
void add(int n, float* x, float* y) {
    for (int i = 0; i < n; ++i) {
        y[i] = x[i] + y[i];
    }
}


int main()
{
    int N = 1 << 28;
    float *x, *y;

    cudaMallocManaged(&x, N * sizeof(float));  /* простой API */
    cudaMallocManaged(&y, N * sizeof(float));
```

# Сложение векторов

```
    add<<<1, 1>>>(N, x, y);  /* вызов Kernel-функции */
    CudaDeviceSynchronize();  /* дождаться возврата всех Kernel */

    float maxError = 0.0f;
    for (int i = 0; i < N; i++) {
        maxError = fmax(maxError, fabs(y[i]-3.0f));
    }
    std::cout << "Max error: " << maxError << std::endl;

    cudaFree(x);
    cudaFree(y);

    return 0;
}
```

# Компиляция и запуск

```
nvcc main.cu

./a.out
nvprof ./a.out
```

Существуют и другие способы сборки кода CUDA.

# Параллельность

```
/*
 * Должно делиться на 32 (число потоков в одном warp)
 * Должно быть делителем 1024 (?)
 */
int blockSize = 256;
/*
 * Блоки покрывают всю область определения и образуют grid
 */
int gridSize = (N + (blockSize - 1)) / blockSize;

/* Оба параметра могут быть одно-, двух- и трёхмерными */

add<<<gridSize, blockSize>>>(N, x, y);
```

# Параллельность

Соответственно меняется и функция Kernel:

```
__global__
void add(int n, float* x, float* y) {
    int index = blockIdx.x * blockDim.x + threadIdx.x;
    int stride = blockDim.x * gridDim.x;

    /* Сколько выполнится итераций? */
    for (int i = index; i < n; i += stride) {
        y[i] = x[i] + y[i];
    }
}
```

# Elapsed time

```
cudaEvent_t start, stop;

cudaEventCreate(&start);
cudaEventCreate(&stop);

cudaEventRecord(start, 0 /* Stream ID */);
add<<<gridSize, blockSize>>>(N, x, y);
cudaEventRecord(stop, 0);
cudaEventSynchronize(stop);

float elapsedTime;
cudaEventElapsedTime(&elapsedTime, start, stop);
Std::cout << “Elapsed time is ” << elapsedTime << std::endl;

cudaEventDestroy(start);
cudaEventDestroy(stop);
```

# Ручное управление памятью

```
float *h_x = (float*)malloc(size);
float *h_y = (float*)malloc(size);
…

float *d_x, *d_y;
cudaMalloc(&d_x, size);
cudaMalloc(&d_y, size);

cudaMemcpy(d_x, h_x, size, cudaMemcpyHostToDevice);
cudaMemcpy(d_y, h_y, size, cudaMemcpyHostToDevice);


…
add<<<numBlocks, blockSize>>>(N, d_x, d_y);
…

cudaMemcpy(h_y, d_y, size, cudaMemcpyDeviceToHost);
```

# Полезные материалы

- CUDA C++ Programming guide
  https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html
- CUDA C++ Best practices
  https://docs.nvidia.com/cuda/cuda-c-best-practices-guide/index.html
- Примеры кода на CUDA
  https://github.com/akhtyamovpavel/ParallelComputationExamples