

Лекция 10

Библиотека *Thrust*.

- Обобщенное программирование: контейнеры, обобщенные алгоритмы, итераторы.
- Контейнеры `host_vector` и `device_vector`.
- Алгоритмы *thrust*.
- Преобразование указателей и комбинированный код.
- Алгоритм `transform` и функторы.
- Скалярное произведение векторов с использованием *thrust*.
- Транспонирование матрицы с использованием *thrust*.

Контейнеры `host_vector` и `device_vector`

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
//#include <thrust/copy.h>
int main(void){
    thrust::host_vector<int> h(1 << 8);
    thrust::generate(h.begin(), h.end(), rand);
    thrust::device_vector<int> d=h;
    thrust::sort(d.begin(), d.end());
    //thrust::copy(d.begin(), d.end(), h.begin());
    h=d;
    for(int i=0;i<1<<8;i++)
        printf("%i\t%d\n",i, h[i]);
    return 0;
}
```

```
~> nvcc lab7_0.cu -o 0
```

```
~> nvprof ./lab7_0
```

```
.....  
API calls:
```

```
99.74% 137.32ms 2 68.661ms 3.4070us 137.32ms cudaMalloc
```

```
.....  
0.05% 70.674us 2 35.337us 5.5460us 65.128us cudaFree
```

```
.....  
0.02% 29.026us 2 14.513us 12.875us 16.151us cudaMemcpyAsync
```

```
.....  
0.00% 3.1590us 27 117ns 93ns 571ns cudaGetLastError
```

```
0.00% 1.6800us 5 336ns 212ns 616ns cudaGetDevice
```

Преобразование указателей и комбинированный код

```
#include<thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/fill.h>
#include <thrust/copy.h>
#include <cstdio>

__global__ void gTest(float* d){
    int idx=threadIdx.x+blockDim.x*blockIdx.x;
    d[idx]+=(float)idx;
}
```

```
int main(void){  
    float *raw_ptr;  
#ifdef H2D  
    thrust::host_vector<float> h(1 << 8);  
    thrust::fill(h.begin(), h.end(), 3.1415f);  
    thrust::device_vector<float> d = h;  
    fprintf(stderr, "Host to device\n");  
#else  
    thrust::device_vector<float> d(1<<8);  
    thrust::fill(d.begin(), d.end(), 3.1415f);  
    thrust::host_vector<float> h;  
    fprintf(stderr, "Just on device\n");  
#endif
```

```
raw_ptr = thrust::raw_pointer_cast(&d[0]); //d.data());
```

```
gTest<<<4,64>>>(raw_ptr);  
cudaDeviceSynchronize();
```

```
h=d;  
for(int i=0;i<(1<<8);i++)  
    printf("%g\n",h[i]);
```

```
return 0;
```

```
}
```

Алгоритм *transform* и функторы

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/transform.h>
#include <thrust/sequence.h>
#include <cstdio>
#include <cmath>
```

```
struct range_functor{
    float h;
    range_functor(float _h):h(_h){ }
    __host__ __device__ float operator()(float x){
        return h*x;
    }
};
```

```
struct sin_functor{  
    __device__ float operator()(float x){  
        return __sinf(x);  
    }  
};
```

```
int main(){  
    range_functor R(0.02);  
    sin_functor Sin;  
  
    fprintf(stderr, "%g\n", R(30.0f));  
    fprintf(stderr, "%g\n", Sin(3141592.0f/6.0f));
```



```
thrust::host_vector<float> h1(1 << 8);  
thrust::host_vector<float> h2(1 << 8);  
thrust::device_vector<float> d1(1 << 8);  
thrust::device_vector<float> d2(1 << 8);
```

```
thrust::sequence(thrust::device, d1.begin(), d1.end());  
thrust::transform(d1.begin(), d1.end(), d1.begin(), R);  
thrust::transform(d1.begin(), d1.end(), d2.begin(), Sin);  
h2=d2;  
h1=d1;
```

```
for(int i=0;i<(1<<8);i++)  
    printf("%g\t%g\n", h1[i], h2[i]);
```

```
return 0;
```

```
}
```

Скалярное произведение векторов, алгоритм *transform*, *reduce*, *inner_product*

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/sequence.h>
#include <thrust/inner_product.h>
#include <cstdio>
#define N 16
int main(){
    thrust::host_vector<float> h_V1(N);
    thrust::host_vector<float> h_V2(N);
    thrust::host_vector<float> h_V3(N);
    thrust::device_vector<float> d_V1(N);
    thrust::device_vector<float> d_V2(N);
    thrust::device_vector<float> d_V3(N);
```

```
thrust::sequence(d_V1.begin(), d_V1.end());  
thrust::fill(d_V2.begin(), d_V2.end(), 0.5);
```

```
thrust::transform(d_V1.begin(), d_V1.end(), d_V2.begin(), d_V3.begin(),  
                  thrust::multiplies<float>());
```

```
h_V1=d_V1;  
h_V2=d_V2;  
h_V3=d_V3;
```

```
for(int n=0;n<N;n++)  
    printf("%i\t%g\t%g\t%g\n",n, h_V1[n], h_V2[n], h_V3[n]);
```

```
float sum=thrust::reduce (d_V3.begin(), d_V3.end(), 0.0,  
                           thrust::plus<float>());  
printf("sum=%g\n", sum);  
  
float sp=thrust::inner_product(d_V1.begin(), d_V1.end(), d_V2.begin(), 0.0f);  
    printf("sp=%g\n", sp);  
  
return 0;  
}
```

```
Lecture7/Lab7R> ./lab7r
```

0	0	0.5	0
1	1	0.5	0.5
2	2	0.5	1
3	3	0.5	1.5

..... •

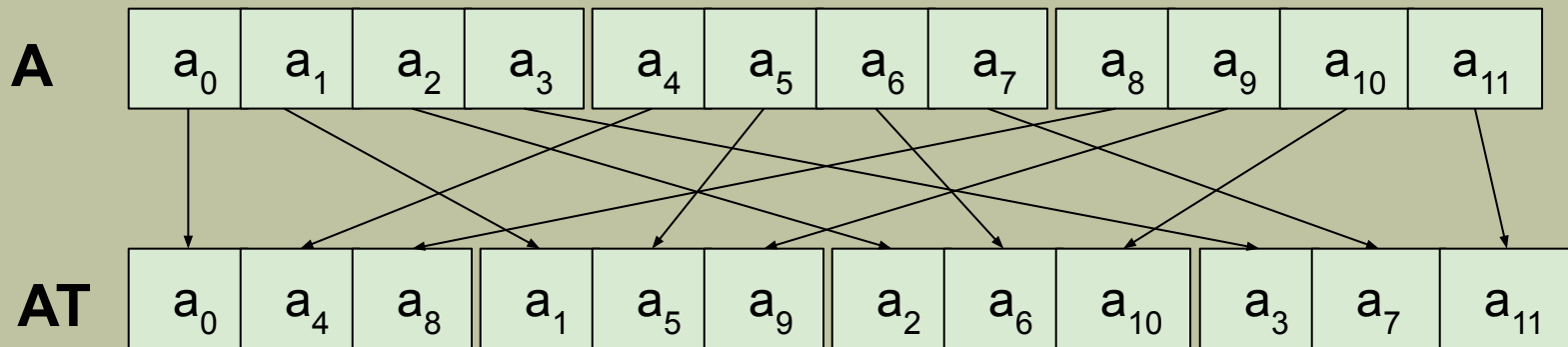
11	11	0.5	5.5
12	12	0.5	6
13	13	0.5	6.5
14	14	0.5	7
15	15	0.5	7.5

```
sum=60
```

```
sp=60
```

Транспонирование матрицы, алгоритм *gather*.

Провести копирование массива a , включающего M векторов длины K по образцу, приведенному на диаграмме:



a_0	a_1	a_2	a_3
a_4	a_5	a_6	a_7
a_8	a_9	a_{10}	a_{11}

Транспонирование



a_0	a_4	a_8
a_1	a_5	a_9
a_2	a_6	a_{10}
a_3	a_7	a_{11}

```
#include <stdio.h>
#include <thrust/generate.h>
#include <thrust/gather.h>
#include <thrust/device_vector.h>
#include <thrust/host_vector.h>

#define M 3
#define K 4

__host__ float rand_f(){
    return ((float)rand()/RAND_MAX);
}
```

```
int main(){
    thrust::host_vector<float> h_A(M*K);
    thrust::host_vector<float> h_AT(K*M);
    thrust::device_vector<float> d_A(M*K);
    thrust::device_vector<float> d_AT(K*M);

    srand(12321);
    thrust::generate(h_A.begin(), h_A.end(), rand_f);
    d_A=h_A;
```



```
int map[K*M];  
for(int i=0; i<K*M;i++)  
    map[i]=(i%M)*K+(i/M);  
thrust::device_vector<int> d_map(map, map + K*M);
```

```
thrust::gather(d_map.begin(), d_map.end(), d_A.begin(), d_AT.begin());
```

```
.....
```

```
// вывод результата
```

```
return 0;
```

```
}
```

```
Lecture7/Lab7t> ./lab7t0
```

```
0.0756    0.4856    0.4013    0.0274
0.6449    0.2524    0.6085    0.9667
0.4870    0.2372    0.6398    0.3343
.....
.....
0.0756    0.6449    0.4870
0.4856    0.2524    0.2372
0.4013    0.6085    0.6398
0.0274    0.9667    0.3343
```

.....
0 4 8 1 5 9 2 6 10 3 7 11
.....
.....
0.0756 0.4856 0.4013 0.0274 0.6449 0.2524 0.6085
 0.9667 0.4870 0.2372 0.6398 0.3343
.....
.....
0.0756 0.6449 0.4870 0.4856 0.2524 0.2372 0.4013
 0.6085 0.6398 0.0274 0.9667 0.3343

ЗАДАНИЕ.

1. Реализовать вычисление скалярного произведения векторов на GPU, используя CUDA API (“сырой код”) и, отдельно, используя библиотеку *Thrust*. Сравнить время выполнения программ при различной длине векторов.
2. Реализовать транспонирование матрицы на GPU, используя CUDA API (“сырой код”) и, отдельно, используя библиотеку *Thrust*. Сравнить время выполнения программ при различной размерности матрицы.