Отчет

Решение уравнения теплопроводности

Выполнил 22931, Чеканов Ярослав Валерьевич

24.05.2024

Реализация кода:

Ядро для расчёта разницы двух матриц:

```
__global__ void subtr_arr(const double *A, const double *Anew, double *subtr_res , int
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    int j = blockDim.y * blockIdx.y + threadIdx.y;
    if ((i >= 0) && (i < m) && (j >= 0) && (j < m))
        subtr_res[OFFSET(i, j, m)] = A[OFFSET(i, j, m)] - Anew[OFFSET(i, j, m)];
}
```

Ядро для расчёта среднего:

```
__global__ void calc_mean(double *A, double *Anew, int m, bool flag) {
    int i = blockDim.x * blockIdx.x + threadIdx.x;
    int j = blockDim.y * blockIdx.y + threadIdx.y;
    if (flag) {
        if ((i > 0) && (i < m - 1) && (j > 0) && (j < m - 1))
            A[OFFSET(j, i, m)] = 0.25 * (Anew[OFFSET(j, i + 1, m)] + Anew[OFFSET(j, i - 1, m)] + Anew[OFFSET(j - 1, i, m)] + Anew[OFFSET(j + 1
    }
    else {
        if ((i > 0) && (i < m - 1) && (j > 0) && (j < m - 1))
            Anew[OFFSET(j, i, m)] = 0.25 * (A[OFFSET(j, i + 1, m)] + A[OFFSET(j, i - 1, m)] + A[OFFSET(j - 1, i, m)] + A[OFFSET(j + 1, i, m)])
    }
}
```

Применение редукции оператора Max:

```
if (iter % 1000 == 0) {
    nvtxRangePushA("calcError");
    err = 0;
    #pragma acc host_data use_device(A, Anew, subtr_temp)
    {
        subtr_arr<<<grid, block, 0, stream>>>(A, Anew, subtr_temp, m);
        cub::DeviceReduce::Max(d_temp_storage, temp_storage_bytes, subtr_temp, d_error_ptr, m * m, st
        // асинхронное копирование из d_error_ptr в err между девайсом и хостом
        cudaErr = cudaMemcpyAsync(&err, d_error_ptr, sizeof(double), cudaMemcpyDeviceToHost, stream);
    }
    nvtxRangePop();
}
```

Применение cuda_unique_ptr:

```
// указатель для управления памятью на устройстве
template<typename T>
using cuda_unique_ptr = std::unique_ptr<T,std::function<void(T*)>>;


// выделение памяти на устройстве
template<typename T>
T* cuda_new(size_t size) {
    T *d_ptr;
    cudaMalloc((void **)&d_ptr, sizeof(T) * size);
    return d_ptr;
}


// освобождение ресурсов
template<typename T>
void cuda_free(T *dev_ptr) {
    cudaFree(dev_ptr);
}
```

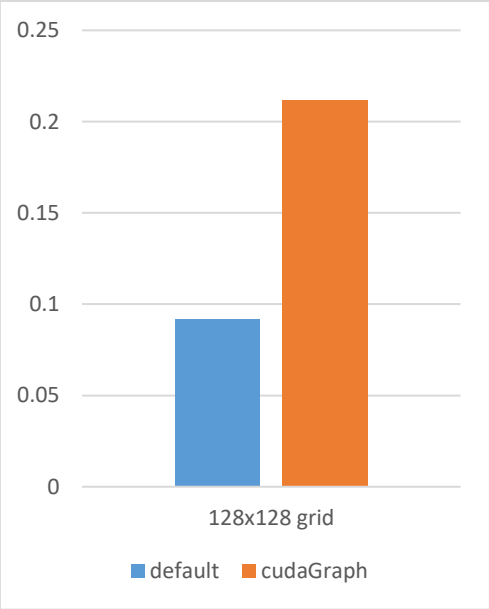
```
cuda_unique_ptr<double> d_unique_ptr_error(cuda_new<double>(0), cuda_free<double>);
cuda_unique_ptr<void> d_unique_ptr_temp_storage(cuda_new<void>(0), cuda_free<void>);

cuda_unique_ptr<double> d_unique_ptr_A(cuda_new<double>(0), cuda_free<double>);
cuda_unique_ptr<double> d_unique_ptr_Anew(cuda_new<double>(0), cuda_free<double>);
cuda_unique_ptr<double> d_unique_ptr_Subtract_temp(cuda_new<double>(0), cuda_free<double>

// выделение памяти и перенос на устройство
double *d_error_ptr = d_unique_ptr_error.get();
cudaErr = cudaMalloc((void**)&d_error_ptr, sizeof(double));

double *d_A = d_unique_ptr_A.get();
cudaErr = cudaMalloc((void **)&d_A, m * m * sizeof(double));

double *d_Anew = d_unique_ptr_Anew.get();
cudaErr = cudaMalloc((void **)&d_Anew, m * m * sizeof(double));

double *d_Subtract_temp = d_unique_ptr_Subtract_temp.get();
cudaErr = cudaMalloc((void **)&d_Subtract_temp, m * m * sizeof(double));

cudaErr = cudaMemcpyAsync(d_A, A, m * m * sizeof(double), cudaMemcpyHostToDevice, stream)
cudaErr = cudaMemcpyAsync(d_Anew, Anew, m * m * sizeof(double), cudaMemcpyHostToDevice, s
```

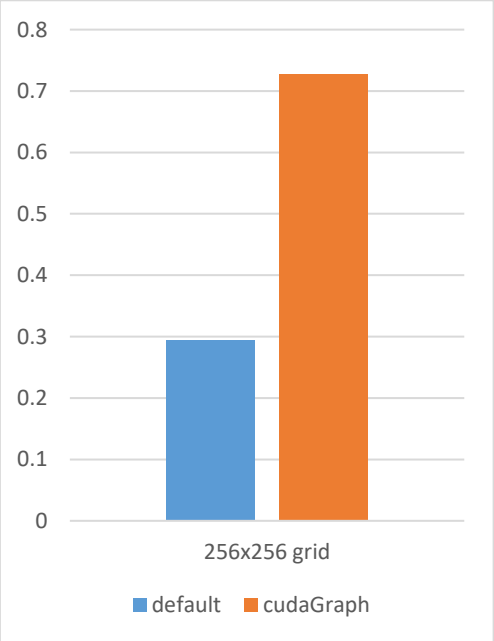
GPU CUB Reduction

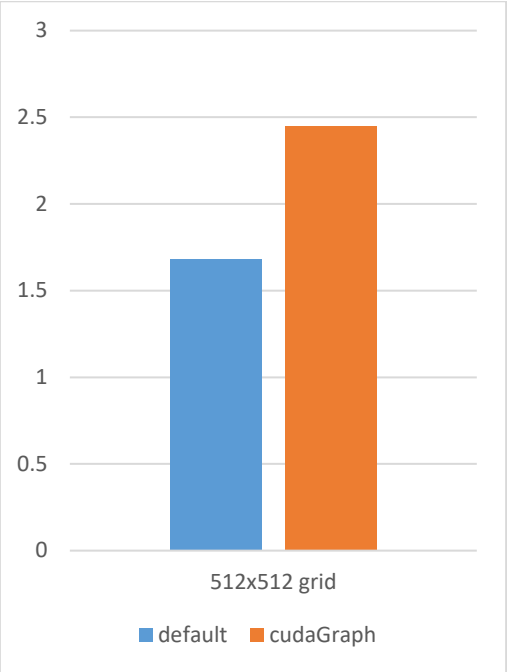
Grid size	Run time, s	Precision	No. of iterations
128	0.092	1.00e-06	31001
256	0.294	1.00e-06	103001
512	1.678	1.00e-06	339600
1024	27.528	1.00e-06	1000000

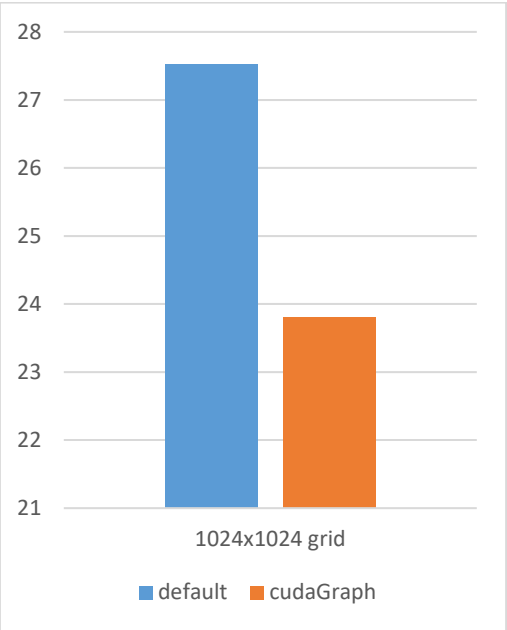
GPU CUB Reduction + Graph

Grid size	Run time, s	Precision	No. of iterations
128	0.212	1.00e-06	30100
256	0.727	1.00e-06	103001
512	2.449	1.00e-06	339600
1024	23.798	1.00e-06	1000000

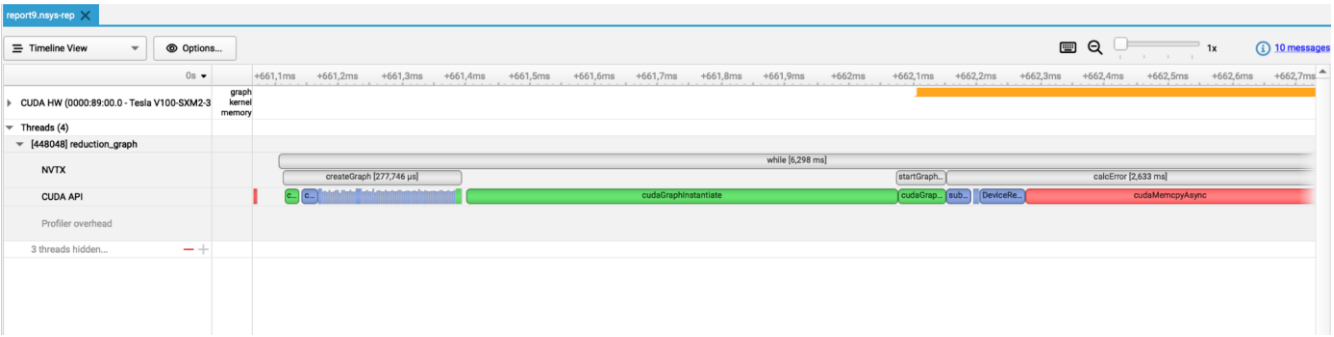


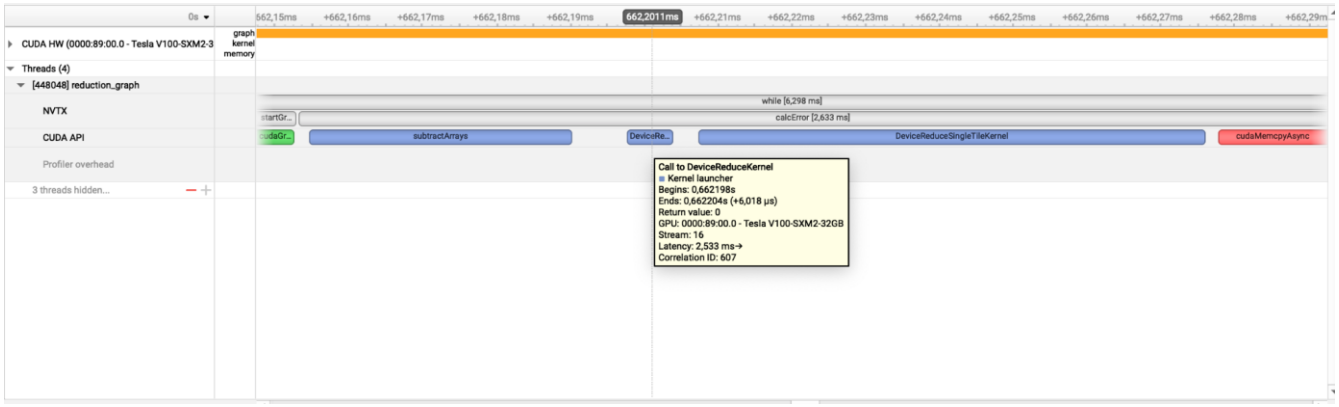


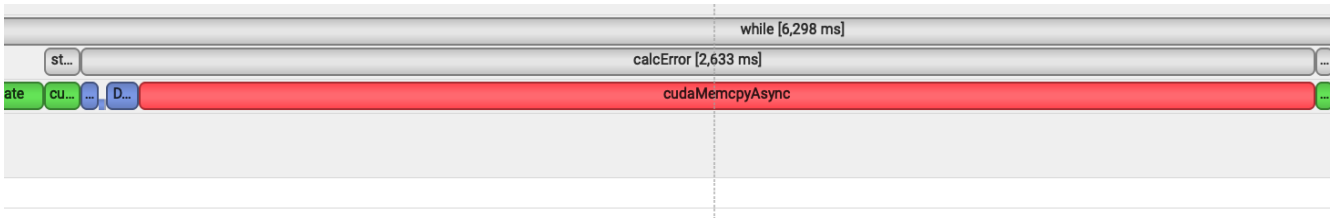


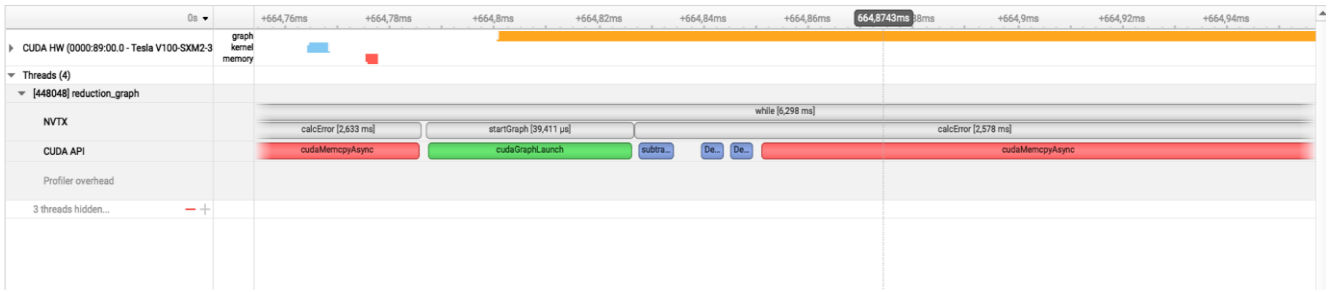


Результаты профилирования:









После реализации cudaGraph не затрачивается время на запуск вычислений, поэтому это дало прирост ускорения на большой сетке.

Вывод: cudaGraph хорошо ускоряет повторяющиеся операции (т.е. в нашем случае – большие сетки).