Отчет

Решение уравнения теплопроводности

Выполнил Фатеев Юрий Сергеевич, 23931

30.05.2025

Реализация кода:

Ядро для расчёта разницы двух матриц:

```
__global__ void sub_mats(const double *A, const double *Anew, double *subtr_res, int m) {
    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[cind(i, j, m)] = fabs(A[cind(i, j, m)] - Anew[cind(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[cind(j, i, m)] = 0.25 * (Anew[cind(j, i + 1, m)] + \
            Anew[cind(j, i - 1, m)] + Anew[cind(j - 1, i, m)] + Anew[cind(j + 1, i, m)]);
    }
    else {
        if ((i > 0) && (i < m - 1) && (j > 0) && (j < m - 1))
            Anew[cind(j, i, m)] = 0.25 * (A[cind(j, i + 1, m)] + \
            A[cind(j, i - 1, m)] + A[cind(j - 1, i, m)] + A[cind(j + 1, i, m)]);
    }
}
```

Применение редукции оператора Max:

```
if (iter % graph_step == 0) {
    nvtxRangePushA("calcError");
    sub_mats<<<grid, block, 0, stream>>>(d_A, d_Anew, d_subtr_temp, n);
    cub::DeviceReduce::Max(d_temp_storage, temp_storage_bytes, d_subtr_temp, d_error_ptr, n * n, stream);
    cudaErr = cudaMemcpy(&error, d_error_ptr, sizeof(double), cudaMemcpyDeviceToHost);
    if (cudaErr != cudaSuccess) f_exception(cudaMemcpy_err);
    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*)>>;
```

```
cuda_unique_ptr<double> d_unique_ptr_error(cuda_new<double>(1), 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>(n*n), cuda_free<double>);
cuda_unique_ptr<double> d_unique_ptr_Anew(cuda_new<double>(n*n), cuda_free<double>);
cuda_unique_ptr<double> d_unique_ptr_subtr_temp(cuda_new<double>(n*n), cuda_free<double>);
```

```
// выделение памяти на устройстве
template<typename T>
T* cuda_new(size_t size) {
    T *d_ptr;
    cudaError_t status;
    status = cudaMalloc((void **)&d_ptr, sizeof(T) * size);
    if (status != cudaSuccess) f_exception(std::string("cudaMalloc error"));
    return d_ptr;
}
// освобождение ресурсов
template<typename T>
void cuda_free(T *dev_ptr) {
    cudaError_t status;
    status = cudaFree(dev_ptr);
    if (status != cudaSuccess) f_exception(std::string("cudaFree error"));
}
```

GPU + CuBLAS

Grid size	Run time, s	Precision	Iterations
128	0.299	1.00e-06	30970
256	1.064	1.00e-06	102898
512	4.054	1.00e-06	339661
1024	35.681	1.00e-06	1000000

GPU CUB Reduction + Graph

Grid size	Run time, s	Precision	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:

Time (%)	Total Time (ns)	Instances	Avg (ns)	Med (ns)	Min (ns)	Max (ns)	StdDev (ns)	Style	Range
50.0	33021563578	1	33021563578.0	33021563578.0	33021563578	33021563578	0.0	PushPop	while
49.2	32463914683	1000	32463914.7	32519679.0	30912265	33958237	395539.8	PushPop	calcError
0.8	550214941	1000	550214.9	504209.0	420294	1501995	120831.4	PushPop	startGraph
0.0	9641848	1	9641848.0	9641848.0	9641848	9641848	0.0	PushPop	init
0.0	2127060	1	2127060.0	2127060.0	2127060	2127060	0.0	PushPop	createGraph

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).