МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

Институт №8 «Информационные технологии и прикладная математика»
Кафедра 806 «Вычислительная математика и программирование»

Лабораторная работа №1
по курсу «Параллельная обработка данных»

Технология MPI и технология CUDA. MPI-IO.

Выполнил: Валов В.В
Группа: М8О-408Б
Преподаватели: А. Ю. Морозов,
К. Г. Крашенинников

Москва, 2022

Условие

Цель работы:

Совместное использование технологии MPI и технологии CUDA. Применение библиотеки алгоритмов для параллельных расчетов Thrust. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в двумерной области с граничными условиями первого рода. Использование механизмов MPI-IO и производных типов данных.

Вариант *MPI_Type_hvector*

Программное и аппаратное обеспечение

GPU:
Compute capability: 7.5;
Графическая память: 4294967296;
Разделяемая память: 49152;
Константная память: 65536;
Количество регистров на блок: 65536;
Максимальное количество блоков: (2147483647, 65535, 65535);
Максимальное количество нитей: (1024, 1024, 64);
Количество мультипроцессоров: 6.
Сведения о системе:
Процессор: Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz 2.59 GHz
Память: 16,0 ГБ;
HDD: 237 ГБ.
Программное обеспечение:
OS: Windows 10;
IDE: Visual Studio 2019;
Компилятор: nvcc.

Метод решения

Для решения задачи сетка делилась на области. За каждую область отвечал один процесс. Сначала происходит обмен граничными данными между процессами, потом обновляются значения во всех ячейках. Потом происходит вычисление погрешности, по результатам которого вычисления останавливаются или продолжаются. В данной работе, в отличие от прошлой все вычисления происходят на gpu, а погрешность

вычисляется при помощи библиотеки thrust. Для вывода данных используется составной тип hvector.

Описание программы

Передача данных всем процессам:

```
MPI_Bcast(&bc_left, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&bc_right, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&bc_up, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&bc_down, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&bc_front, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&bc_back, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&lx, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&ly, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&lz, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&nX, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&nY, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&nZ, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&nb_X, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&nb_Y, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&nb_Z, 1, MPI_INT, 0, MPI_COMM_WORLD);
MPI_Bcast(&epsilon, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
MPI_Bcast(&init_v, 1, MPI_DOUBLE, 0, MPI_COMM_WORLD);
```

Отправка данных:

```
__global__ void send_left_right(double *data, double *buffer, int x, int nX, int nY) {
    int idy = blockIdx.x * blockDim.x + threadIdx.x;
    int offsety = blockDim.x * gridDim.x;

    for (int j = idy; j < nY; j += offsety) {
        buffer[j] = data[_i(x, j)];
    }
}

__global__ void send_up_down(double *data, double *buffer, int y, int nX, int nY) {
    int idx = blockIdx.x * blockDim.x + threadIdx.x;
    int offsetx = blockDim.x * gridDim.x;

    for (int i = idx; i < nX; i += offsetx) {
        buffer[i] = data[_i(i, y)];
    }
}
```

Получение данных:

```
__global__ void recieve_left_right(double *data, double *buffer, int x, int nX, int nY) {
   int idy = blockIdx.x * blockDim.x + threadIdx.x;
   int offsety = blockDim.x * gridDim.x;

   for (int j = idy; j < nY; j += offsety) {
      data[_i(x, j)] = buffer[j];
   }
}

__global__ void recieve_up_down(double *data, double *buffer, int y, int nX, int nY) {
   int idx = blockIdx.x * blockDim.x + threadIdx.x;
   int offsetx = blockDim.x * gridDim.x;

   for (int i = idx; i < nX; i += offsetx) {
      data[_i(i, y)] = buffer[i];
   }
}
```

Контроль погрешности:

```
__global__ void mistake(double* data, double* next, int nX, int nY) {
        int idx = blockDim.x * blockIdx.x + threadIdx.x;
   int idy = blockDim.y * blockIdx.y + threadIdx.y;
   int offsetx = blockDim.x * gridDim.x;
   int offsety = blockDim.y * gridDim.y;

        for (int j = idy - 1; j <= nY; j+=offsety) {
                for (int i = idx - 1; i <= nX; i+=offsetx) {
                        if (i == -1 || j == -1 || i == nX || j == nY) {
                data[_i(i, j)] = 0.0;
                } else {
                data[_i(i, j)] = fabs(next[_i(i, j)] - data[_i(i, j)]);
                }
                }
        }
   }

thrust::device_ptr<double> diff_part = thrust::device_pointer_cast(gpu_data);
```

```
        thrust::device_ptr<double> diff_now = thrust::max_element(thrust::device, diff_part,
diff_part + (nX + 2) * (nY + 2));
      diff =  *diff_now;
```

Результаты:

MPI+GPU	1	2
8	0.1230	0.1994
16	0.1705	0.5512
32	0.4397	1.6501
64	1.9618	6.0013

Выводы

Технология MPI позволяет распараллеливать вычисления, за счет этого возрастает скорость работы программы. Также она хорошо работает в связке с CUDA, что позволяет сделать вычисления еще более эффективными. Но есть и минусы, написание такого кода занимает много времени и сделать это не очень просто. Я использовал операцию Bsend для отправки данных, она является неблокирующей за счет буфера, это позволяет упростить код.