МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №2
по курсу «Параллельная обработка данных»

Технология MPI и технология CUDA. MPI-IO

Выполнил: Г.Н. Хренов
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

1. 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 трехмерной области с граничными условиями первого рода. Использование механизмов MPI-IO и производных типов данных. Запись результатов в файл должна осуществляться параллельно всеми процессами. Необходимо создать производный тип данных, определяющий шаблон записи данных в файл.
2. Вариант 1. MPI_Type_create_subarray

Программное и аппаратное обеспечение

GPU name: NVIDIA GeForce RTX 2060
compute capability 7:5
totalGlobalMem: 6442450944
sharedMemPerBlock: 49152
totalConstMem: 65536
regsPerBlock: 65536
maxThreadsDim: 1024 1024 64
maxGridSize: 2147483647 65535 65535
multiProcessorCount: 30
CPU name: AMD Ryzen 7 3750H with Radeon Vega Mobile Gfx
MaxClockSpeed: 2300
NumberOfCourse: 4
RAM: 8
SSD: 256, HDD: 1024
OS: Windows10
Compiler: nvcc

Метод решения

Задача решается аналогично предыдущей лабораторной. Для записи в файл необходимо производный тип данных с помощью MPI_Type_create_subarray. Для этого нужно отразить структуру расположения сетки при разбиении на процессы. Указывается размер общей сетки, размер одного блока и начало блока в зависимости от его номера. Вычисления сетки, а также копирования в буфер и обратно для передачи данных, переносим на GPU, пишем несколько ядер, избегая ветвлений.

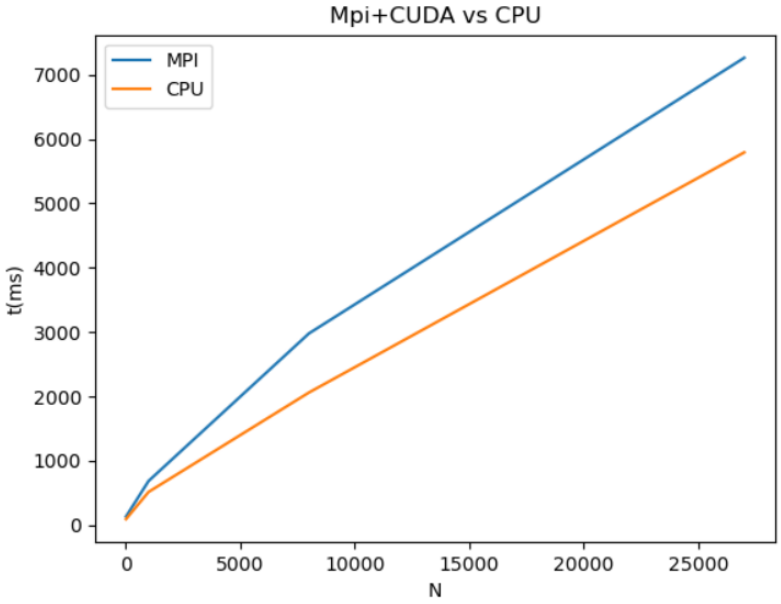
Описание программы

Lab8.cpp:
#define _i(i, j, k) – переход из трехмерной сетки в линейную для элементов
#define _ib(i, j, k) - переход из трехмерной сетки в линейную для блоков
__global__ void kernel_get_vals(…) – вычисление параметров сетки
__global__ void kernel_get_diffs(…) – подсчет разностей с предыдущей итерацией
__global__ void kernel_LR_bc(…) – группа ядер, заполняющая граничные условия

__global__ void kernel_copy_send_LR(…) – группа ядер, заполняющая буфер для отправки границ
__global__ void kernel_copy_recive_LR(…) – группа ядер, заполняющая данные о границах, пришедших с соседних блоков

Результаты(сервер)

блоков\процессов	1*1*1	2*2*1	2*2*2
4*4*4	133.56ms	2284.19 ms	5613.07 ms
10*10*10	686.08 ms	12184.67 ms	30314.7 ms
20*20*20	2978.9 ms	49304.3 ms	119182.03 ms
30*30*30	7262.4 ms	117645.6 ms	281631.9 ms



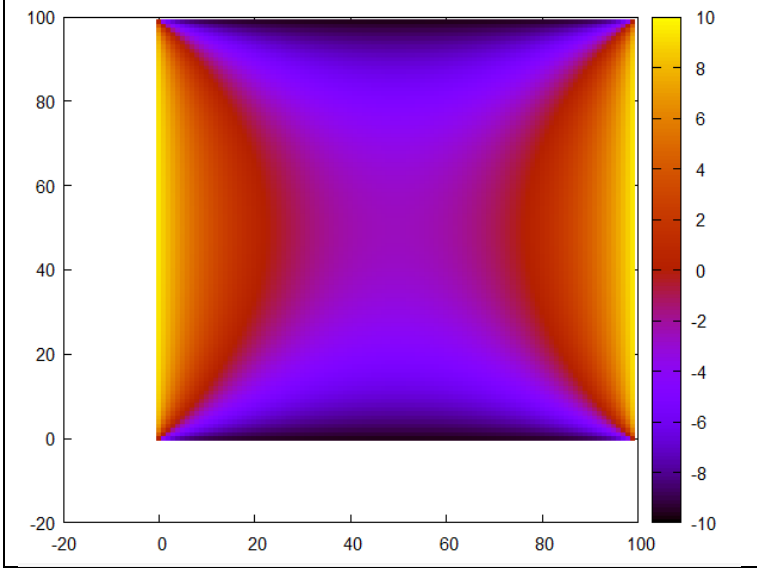
Визуализация

```
2 2 2
100 100 5
mpi.out
1e-10
1.0 1.0 1.0
-5 5 10.0 10.0 -10.0 -10.0
```

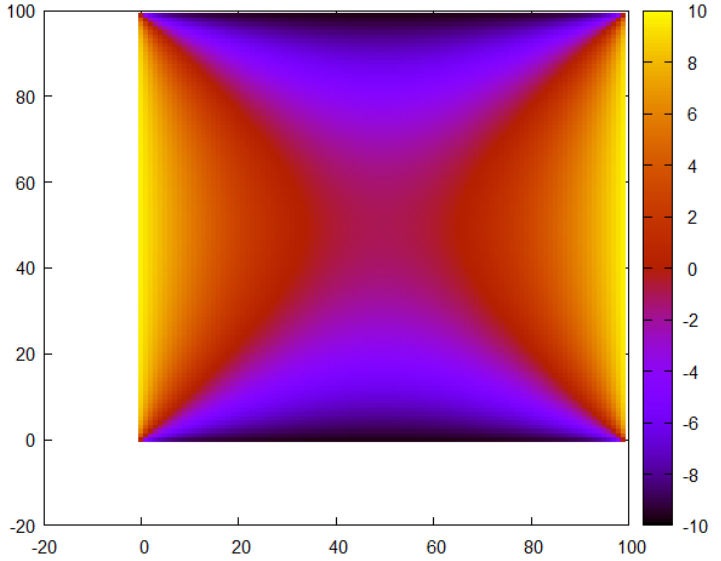
Внизу-холодно, вверху-жарко,
лево-право-жарко
зад-перед-холодно

Срезы по высоте(к)

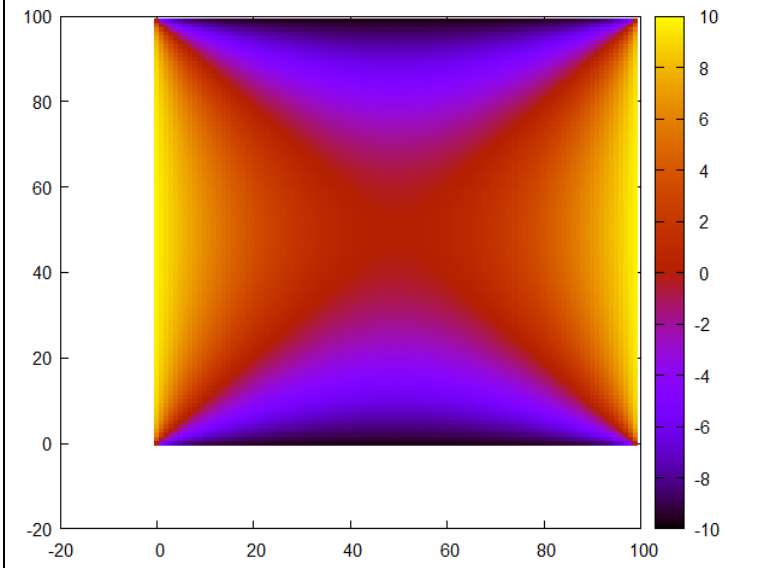
K=1


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

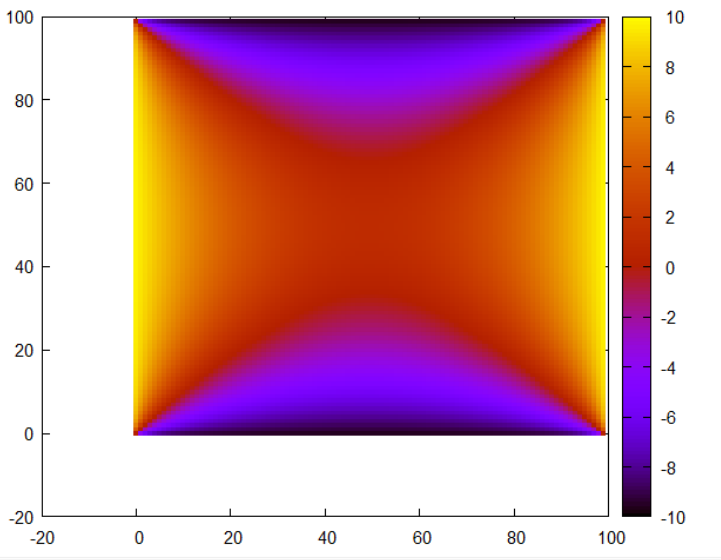
K=2


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

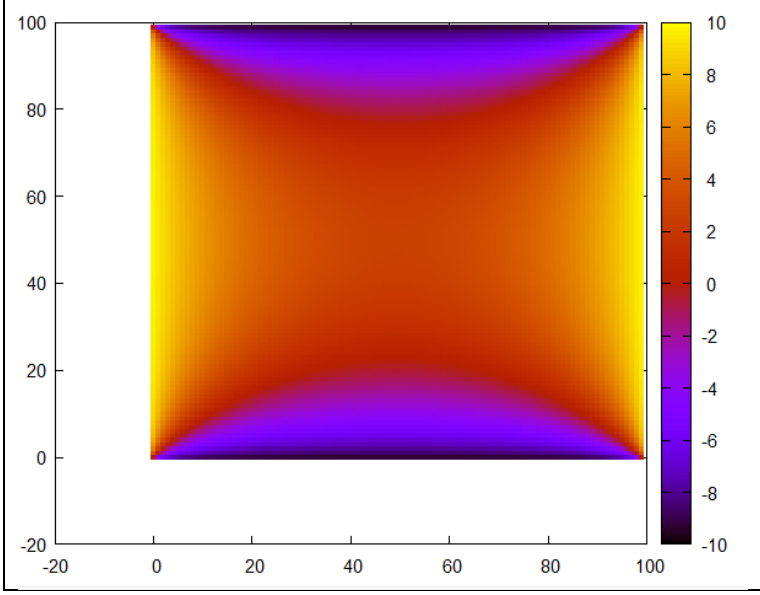
K = 3


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

K = 4


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

K = 5


100
80
60
40
20
0
-20
-20
0
20
40
60
80
100
10
8
6
4
2
0
-2
-4
-6
-8
-10

Выводы

Технология CUDA без проблем сочетается с технологией MPI, что позволяет достигать еще большего распараллеливания задачи. Однако в этом случае необходимо тщательно следить за распределением нагрузки на ресурсы ЭВМ, так как неправильная организация процесса распараллеливания может сильно увеличить время работы программы. Производные типы данных в MPI дают возможность очень удобно распределять данные между процессами. По сути это шаблоны, которые позволяют выделить именно тот участок данных, который нужен для работы.