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: К.О.Вахрамян

Группа: 8О-406Б

Преподаватели: К.Г. Крашенинников,

А.Ю. Морозов

Москва, 2021

**Условие**

Кратко описывается задача:

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.

1. Вариант задания.

MPI\_Type\_hvector

**Программное и аппаратное обеспечение**

**GPU:**

--- General Information for device ---

Name: NVIDIA GeForce GTX 1650

Compute capability: 7.5

Clock rate: 1560000

Device copy overlap: Enabled

Kernel execution timeout : Enabled

--- Memory Information for device ---

Total global mem: 4100521984

Total constant Mem: 65536

Max mem pitch: 2147483647

Texture Alignment: 512

--- MP Information for device ---

Multiprocessor count: 16

Shared mem per mp: 49152

Registers per mp: 65536

Threads in warp: 32

Max threads per block: 1024

Max thread dimensions: (1024, 1024, 64)

Max grid dimensions: (2147483647, 65535, 65535)

**CPU:**

Architecture: x86\_64

CPU op-mode(s): 32-bit, 64-bit

Byte Order: Little Endian

Address sizes: 39 bits physical, 48 bits virtual

CPU(s): 8

On-line CPU(s) list: 0-7

Thread(s) per core: 2

Core(s) per socket: 4

Socket(s): 1

NUMA node(s): 1

Vendor ID: GenuineIntel

CPU family: 6

Model: 158

Model name: Intel(R) Core(TM) i5-9300HF CPU @ 2.40GHz

Stepping: 13

CPU MHz: 1274.759

CPU max MHz: 2400.0000

CPU min MHz: 800.0000

BogoMIPS: 4800.00

Virtualization: VT-x

L1d cache: 128 KiB

L1i cache: 128 KiB

L2 cache: 1 MiB

L3 cache: 8 MiB

**OS:**

Linux Mint 20

**Compiler:**

nvcc

**Code Editor:**

VS Code

**Метод решения**

Задача решается аналогично предыдущей ЛР. Для записи в файл необходим производный тип данных MPI\_Type\_vector. Тип создается как набор блоков из элементов исходного типа, при этом между блоками задаются регулярные промежутки. Все вычисление и обмен граничными условиями производятся на GPU.

**Описание программы**

Хотя мы решаем двухмерную задачу, данные хранятся в одномерном массиве. Для правильной индексации используется макрос:

#define \_i(i, j) (((j) + 1) \* (nx + 2) + (i) + 1)

Для индексации по сетке блоков:

#define \_ib(i, j) ((j) \* nbx + (i))

Ядро для инициализации начальным значением:

\_\_global\_\_ void kernelInit(double\* data, int nx, int ny, double u0)

Ядро записи в буфер граничных значений слева и справа

\_\_global\_\_ void kernelSendLeftRight()

Аналогично, сверху и снизу

\_\_global\_\_ void kernelSendFrontBack()

Ядро для чтения из буфера для левых и правых значений

\_\_global\_\_ void kernelReciveLeftRight()

Аналогично, сверху снизу

\_\_global\_\_ void kernelReciveFrontBack()

Шаг итерационного процесса

\_\_global\_\_ void kernelStep()

Подсчет разницы между новым предыдущим значением

\_\_global\_\_ void kernelDiff()

Данные считывает нулевой процесс и отправляет всем при помощи функции:

MPI\_Bcast();

Отправка данных между процессами осуществляется при помощи буферизованной функции:

MPI\_Bsend();

Данная функция является неблокирующей

Получение про помощи:

MPI\_Recv();

Максимальная погрешность ищется при помощи редукции

MPI\_Allreduce(MPI\_IN\_PLACE, &check, 1, MPI\_DOUBLE, MPI\_MAX, MPI\_COMM\_WORLD);

**Результаты**

**<<<1,32>>>**

|  |  |  |
| --- | --- | --- |
| Сетка | Блок | MPI + CUDA | CPU |
| 1 1 | 10 10 | 0.274s | 0.30s |
| 2 2 | 50 50 | 1.503s | 3.83s |
| 4 2 | 70 70 | 20.267s | 1m10.212s |

**<<<32,32>>>**

|  |  |  |
| --- | --- | --- |
| Сетка | Блок | MPI + CUDA | CPU |
| 1 1 | 10 10 | 1.202s | 0.3s |
| 2 2 | 50 50 | 0.803s | 3.83 |
| 4 2 | 70 70 | 4.112s | 1m10.212s |

**<<<1024,1024>>>**

|  |  |  |
| --- | --- | --- |
| Сетка | Блок | MPI + CUDA | CPU |
| 1 1 | 10 10 | 0.400s | 0.30s |
| 2 2 | 50 50 | 0.523s | 3.83 |
| 4 2 | 70 70 | 2.512s | 1m10.212s |

4 2

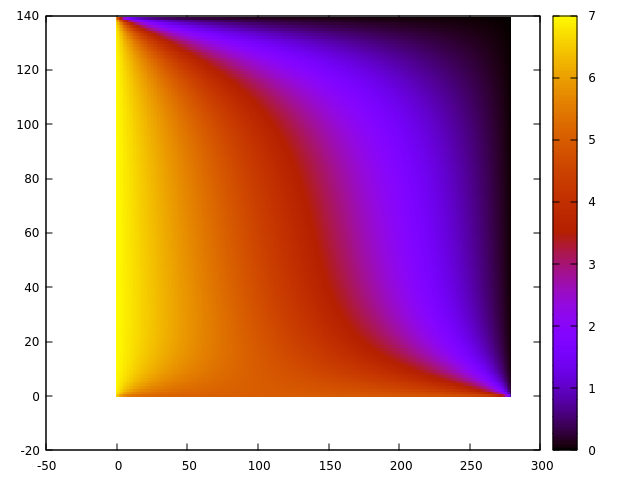
70 70

mpi.out

1e-10

1.0 2.0

7.0 0.0 5.0 0.0

****

**Выводы**

Данная лабораторная работа является доработкой предыдущей. Удалось заметно ускорить код засчет технологии CUDA. Однако, стоит быть аккуратным, т.к. неправильное распараллеливание может только увеличить время работы программы. Также, в рамках работы мною были использованы произвольные типы данных. Запись в файл велась параллельно в каждом процессе.