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»

Message Passing Interface (MPI)

Выполнил: К.О. Вахрамян
Группа: 8О-406Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2021

Условие

1. **Цель работы.** Знакомство с технологией MPI. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в двухмерной области с граничными условиями первого рода.
2. **Вариант задания.** Вариант на 2.

$$\frac{d^2u(x,y)}{dx^2} + \frac{d^2u(x,y)}{dy^2} = 0,$$

$$u(x \le 0, y) = u_{left},$$
$$u(x \ge l_x, y) = u_{right},$$
$$u(x, y \le 0) = u_{front},$$
$$u(x, y \ge l_y) = u_{back}.$$

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

Решение ДУ производится методом конечно-разностных аппроксимаций. Поиск решения сводится к итерационному процессу:

$$u_{i,j}^{(k+1)} = \frac{\left(u_{i+1,j}^{(k)} + u_{i-1,j}^{(k)}\right)h_x^{-2} + \left(u_{i,j+1}^{(k)} + u_{i,j-1}^{(k)}\right)h_y^{-2}}{2\left(h_x^{-2} + h_y^{-2}\right)},$$

где

$$i = 1..n_x, j = 1..n_y,$$
$$h_x = l_x n_x^{-1}, h_y = l_y n_y^{-1},$$
$$u_{0,j}^{(k)} = u_{left}, u_{n_x+1,j}^{(k)} = u_{right},$$
$$u_{i,0}^{(k)} = u_{front}, u_{i,n_y+1}^{(k)} = u_{back},$$
$$u_{i,j}^{(0)} = u^0.$$

Процесс останавливается при

$$max_{i,j} \left| u_{i,j}^{(k+1)} - u_{i,j}^{(k)} \right| < \epsilon$$

Каждый процесс имеет два равных по величине блока, чтобы на основе старых значений, вычислять новые. Также необходимо получать граничные значения, рассчитанные в соседних блоках. Вследствие этого реализован межпроцессорный обмен граничными условиями.
Таким образом получаем алгоритм:

1. Буферизация и обмен граничными значениями между слоями.
2. Подсчет значений в каждом блоке
3. Подсчет погрешности и сравнение максимальной с некоторым наперед заданным eps

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))
```

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:

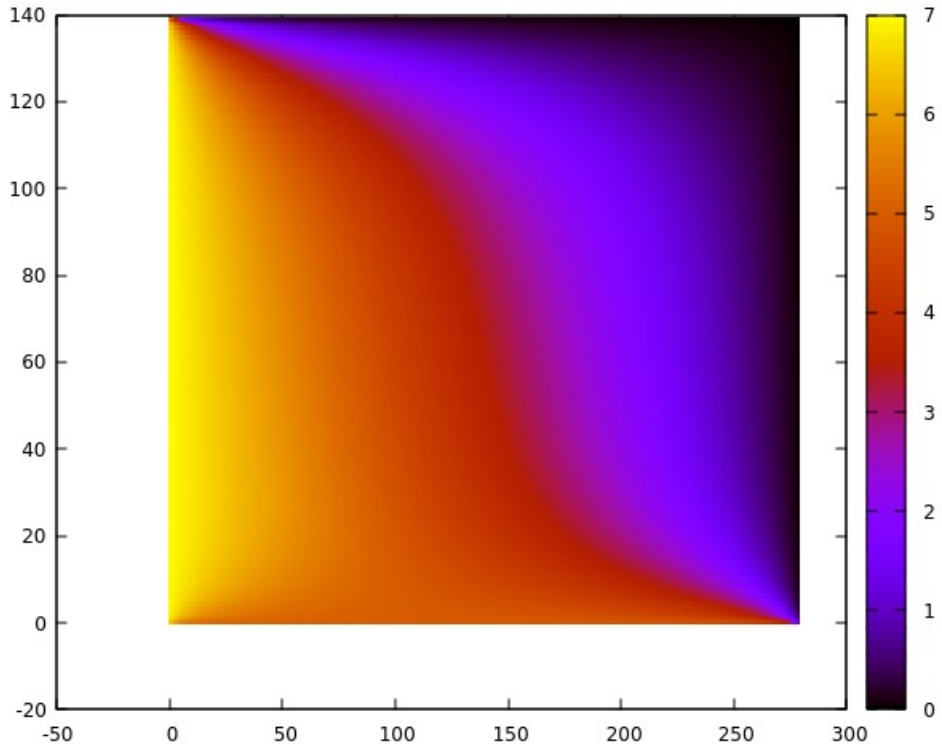
```
MPI_Recv();
```

Максимальная погрешность ищется при помощи редукции
MPI_Allreduce(MPI_IN_PLACE, &check, 1, MPI_DOUBLE, MPI_MAX, MPI_COMM_WORLD);

Результаты

Сетка | Блок	MPI	CPU
1 1 | 10 10	0.065s	0.30s
2 2 | 50 50	2.063s	3.83s
4 2 | 70 70	24.267s	1m10.212s

4 2
70 70
mpi.out
1e-10
1.0 2.0
7.0 0.0 5.0 0.0
5.0



Выводы

Выполнив данную лабораторную работы, я познакомился с необычным методом решения задач численного дифференцирования. Концепция MPI в общем плане немного похожа на CUDA, ведь также один код запускается несколько раз разными процессами / тредами. Однако общение процессов требует написания большого количества кода, который проблематично тестировать. Но в упрощенном варианте передача происходит при помощи функции Bsend, которая является неблокирующей, что заметно упрощает реализацию.