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: А. О. Тояков
Группа: М8О-407Б-18
Преподаватели: К. Г. Крашенинников,
А. Ю. Морозов

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 трехмерной области с граничными условиями первого рода. Использование механизмов MPI-IO и производных типов данных.

Вариант № 2: MPI_Type_hvector

ПРОГРАММНОЕ И АППАРАТНОЕ ОБЕСПЕЧЕНИЕ

Device: GeForce MX250

Размер глобальной памяти: 3150381056

Размер константной памяти : 65536

Размер разделяемой памяти: 49152

Регистров на блок: 32768

Максимум потоков на блок: 1024

Количество мультипроцессоров : 3

OS: Linux Ubuntu 18.04

Редактор: VSCode

Компилятор: nvcc версии 11.4 (g++ версии 7.5.0)

МЕТОД РЕШЕНИЯ

Задача решается почти также, как и предыдущая. Изменения состоят только в том, что все вычисления выполняются на GPU и запись в файл производится одновременно всеми потоками. Для этого используется произвольный тип данных MPI_Type_hvector. Отражаем структуру расположения сетки при разбиении на процессы. Затем указывается размер общей сетки, размер одного блока в зависимости от его номера. Для переноса граничных значений на GPU также пишем несколько ядер.

ОПИСАНИЕ ПРОГРАММЫ

#define _i(i, j, k) – переход из трёхмерной сетки в одномерную для элементов.

#define _ib(i, j, k) – переход из трёхмерной сетки в одномерную для блоков.

__global__ void calculation() – вычисление параметров сетки.

__global__ void calculate_diff() – вычисление разности с предыдущей операцией.

__global__ void extractXPart() – ядра, заполняющие буфер для отправки сообщений.

__global__ void putXPart() – ядра, заполняющие данные о границах, пришедших с соседних блоков.

MPI_Bcast() – передача данных всем процессам.

MPI_Sendrecv – передача данных между процессами.

MPI_Allreduce – контроль сходимости всех процессов.

class Processor – класс для хранения всех функций и переменных для решения задачи.

int main() – создание экземпляра класса Processor, а также инициализация и завершение работы MPI.

ТЕСТЫ ПРОИЗВОДИТЕЛЬНОСТИ

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

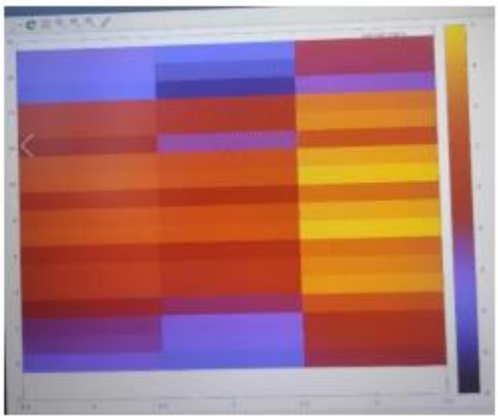
ВИЗУАЛИЗАЦИЯ

Тест:

test.txt

```
1 1 1
3 3 6
mpi.out
1e-10
1.0 1.0 2.0
-7.0 -10.0 5.0 10.0 -3.0 0.0
7.0
```

Срез:



ВЫВОДЫ

Выполнив лабораторную работу № 8, я узнал, как можно использовать возможности CUDA и MPI вместе, получая ещё большее распараллеливание задачи, а, следовательно, меньшее время работы. Однако правильное распределение ресурсов по всем процессам является довольно сложной задачей, и если получить ветвление, то производительность сильно уменьшится. В MPI также есть производные типы данных, которые позволяют распределять данные между процессами. Таким образом можно параллелить даже файловый вывод, который в подобного рода задачах обычно занимает существенное время.