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

Москва, 2020

Условие

Цель работы:

Знакомство с технологией MPI. Реализация метода Якоби. Использование константной памяти. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Вариант 1. Обмен граничными слоями через send/receive, контроль сходимости allgather.

Программное и аппаратное обеспечение

GeForce 940MX

Compute capability:	5.0
Dedicated video memory:	4096 MB
shared memory per block:	49152 bytes
constant memory:	65536 bytes
Total number of registers available per block:	65536
Maximum number of threads per multiprocessor:	2048
Maximum number of threads per block:	1024
(3) Multiprocessors, (128) CUDA Cores/MP:	384 CUDA Cores

Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz

Architecture:	x86_64
Byte Order:	Little Endian
CPU(s):	4
Thread(s) per core:	2
Core(s) per socket:	2
CPU MHz: CPU max MHz: CPU min MHz:	713.848 3100,0000 400,0000
L1d cache: L1i cache: L2 cache: L3 cache:	64 KiB 64 KiB 512 KiB 3 MiB

RAM	8GiB SODIMM DDR4 Synchronous Unbuffered (Unregistered) 2400 MHz (0,4 ns)

SSD(SPCC_M.2_SSD)	223,6G
HDD(ST1000LM035-1RK172)	931,5G

OS: Ubuntu 20.04 focal
IDE: jetbrains clion
compiler: mpic++

Метод решения

В качестве фундамента для решения задачи был взят код с лекции, где была решена упрощенная задача. Главные моменты которые необходимо было переделать, это сделать ввод произвольных значений, изменить индексацию из двухмерной в трехмерную, поменять функцию передачи данных на MPI_Send, сделать остановку итерационного процесса до достижения определенной точности, вместо фиксированного кол-ва итераций и оптимизировать программу.

Описание программы

Индексацию я изменил, используя кодстайл с лекций, изменив макросы и добавив индекс k. Изменения функции на MPI_Send не составил большого труда, так как параметры функции совпадают. Сходимость процесса отслеживалась MPI_Allgather, которая является агрегирующей функцией и складывает значения в буфер. Приведу код проверки сходимости, так как он является самой отличающейся частью от кода из лекций.

```
double *diffs = (double *) malloc(sizeof(double) * nbx * nby * nbz);
MPI_Allgather(&diff, 1, MPI_DOUBLE, diffs, 1, MPI_DOUBLE,
        MPI_COMM_WORLD);
double gather_diff = 0;
for (k = 0; k < nbx * nby * nbz; ++k) {
   gather_diff = std::max(gather_diff, diffs[k]);
}

if (gather_diff < eps) {
   break;
}
```

Результаты

Сравнивать результаты cpu и mpi было бы интересно, будь у меня кластер компьютеров объединенных быстрой сетью. Но все же мы можем провести данный тест, чтобы увидеть отличия.

1.

	MPI	CPU
2 2 2, 20 20 20	5006.32ms	9925.1ms
2 2 4, 20 20 10	5894.24ms	9734.2ms

Выводы

Данная работа была мне интересна по нескольким пунктам. Во-первых, было интересно попробовать программу, которая обменивается данными между процессами. Во-вторых, оптимизация программы была очень интересна за счет жесткого лимита на чекере, произошло понимание того, как блокирующие функции могут задерживать подсчет программы. И в третьих было прикольно увидеть кодстайл с красивым использованием макросов для перевода из одномерного массива в Хмерный массив.