МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа № 4
по курсу «Программирование графических процессоров»

Технология MPI и технология OpenMP

Выполнил: А.В. Скворцов
Группа: 8О-408Б
Преподаватели: К.Г. Крашенинников,
А.Ю. Морозов

Москва, 2020

Условие

Цель работы: Совместное использование технологии MPI и технологии OpenMP. Реализация метода Якоби. Решение задачи Дирихле для уравнения Лапласа в трехмерной области с граничными условиями первого рода.

Требуется решить задачу описанную в лабораторной работе №7, с использованием стандарта распараллеливания openmp в рамках одного процесса.
Вариант 1: распараллеливание основных циклов через parallel for (+директива reduction для вычисления погрешности)

Программное и аппаратное обеспечение

- GPU: Geforce 940MX
  - Compute capability : 5.0
  - Total Global Memory : 2147483648
  - Shared memory per block : 49152
  - Registers per block : 65536
  - Max threads per block : (1024, 1024, 64)
  - Max block : (2147483647, 65535, 65535)
  - Total constant memory : 65536
  - Multiprocessors count : 3
- CPU: Intel Core i5-6200U 2.30GHz
- RAM: 4GB
- Software: Windows 10, Visual Studio Code, nvcc

Метод решения

Над пространством строится регулярная сетка. С каждой ячейкой сопоставляется значение функции в точке соответствующей центру ячейки. Граничные условия u реализуются через виртуальные ячейки, которые окружают рассматриваемую область. Поиск решения сводится к итерационному процессу:

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

процесс останавливается, когда:

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

Когда размер сетки становится настолько большим, что его невозможно поместить в оперативной памяти одного компьютера, либо уже не хватает вычислительной мощности одного процессора имеет смысл распределить нагрузку между несколькими компьютерами. Для этого разобьем сетку на несколько смежных подсеток и будем на каждой итерации обмениваться их граничными элементами, осуществить это можно, например, с помощью технологии MPI. Тогда алгоритм работы распределенной программы будет выглядеть следующим образом:

1. Обмен граничными слоями между процессами
2. Обновление значений во всех ячейках
3. Вычисление локальной погрешности в рамках каждого процесса, а затем обмен погрешностями по всей области для поиска максимальной

Описание программы

Вся машинерия алгоритма происходит внутри одного класса HeatMap:

1. конструктор HeatMap – подготавливает память для сетки, инициализирует ее, вычисляет ранги соседних сеток.

2. int HeatMap::get_rank(int x, int y, int z) — возвращает ранг процесса с заданными координатами
3. void HeatMap::set_limits — задает индексы границ областей, в рамках которых происходит копирование/установка границ
4. void HeatMap::init_boundary(double val, int dir) — проставляет заданное значение на указанную границу.
5. HeatMap::index_func_type HeatMap::get_index_func(int axis) — возвращает функцию для индексации буфера в зависимости от выбранной границы
6. inline size_t HeatMap::index_yz(int i, int j, int k) — индексация буфера вдоль плоскости yz
7. inline size_t HeatMap::index_xz(int i, int j, int k) — индексация буфера вдоль плоскости xz
8. inline size_t HeatMap::index_xy(int i, int j, int k) — индексация буфера вдоль плоскости xy
9. void HeatMap::boundary_to_buff(double *buff, int axis) – копирование заданной границы в буфер
10. void HeatMap::set_boundary(double *buff, int axis) — установка значений буфера на заданную границу
11. void HeatMap::sendrecv_along_axis — неблокирующий обмен граничными условиями вдоль заданной оси
12. void HeatMap::exchange_boundaries — обмен всеми границами вдоль всех осей
13. double HeatMap::approximate — один шаг аппроксимации сетки, включает в себя обмен границами, вычисление новых локальных значений, обмен локальной погрешностью для поиска максимальной
14. void HeatMap::write(char *filepath) — запись всей текущей сетки в файл с указанным именем

Результаты

Скорость работы программы (в миллисекундах) в зависимости от сетки процессов (по горизонтали) от самой сетки (по вертикали) на 4х ядерном процессоре:

	30x30x30	40x40x40
1x1x1	4211	15134
2x1x1	3699	14854
2x2x1	2215	8495
2x2x2	10428	18220

Выводы

Message Passing Interface предоставляет API, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу, применяется в первую очередь при разработке кластеров и суперкомпьютеров. С его помощью можно относительно легко и гибко масштабировать ресурсоемкие программы, которым уже недостаточно вычислительных мощностей одного компьютера. Для дальнейшего повышения производительности таких программ нужно повысить производительность каждого процесса в отдельности. Осуществить это легко и быстро можно, например, с помощью технологии openMP, где за распараллеливание отвечает сама технология, разработчику достаточно лишь указать, где и что нужно параллелить.