МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ
(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ)

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

Лабораторная работа №9
по курсу «Программирование графических процессоров»

Технология MPI и технология OpenMP

Выполнил: Полей-Добронравова Амелия
Группа: 8О-407Б
Преподаватели: К.Г. Крашенинников, А.Ю. Морозов

Москва, 2022

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.

Все **входные-выходные данные** и **варианты заданий** по технологии MPI совпадают с входными-выходными данными и вариантами заданий из лабораторной работы №8. **Обмен граничными слоями** организовать без использования дополнительных буферов, через производный тип данных в соответствии с вариантом лабораторной работы №8.

По технологии OpenMP вводятся **два варианта**:

1. Распараллеливание основных циклов через parallel for (+директива reduction для вычисления погрешности);
2. Распараллеливание в общем виде с разделением работы между нитями вручную (“в стиле CUDA”).

Вариант 2.

Программное и аппаратное обеспечение

Компилятор nvcc версии 7.0(g++ версии 4.8.4) на 64-х битной Ubuntu 14.04 LTS.
Параметры графического процессора:
Compute capability : 6.1
Name : GeForce GTX 1050
Total Global Memory : 2096103424
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 : 5

Метод решения

Аналогичен решению 7ЛР. Отличия:
Циклы пересчета значений сетки распараллелены с помощью OpenMP, но из-за варианта всё удобство инкапсулирования исчезает: нужно параллелить вручную, прописывая какие потоки что делают (через число потоков и их индексы).

Описание программы

Функции по функционалу аналогичны 7ЛР. В **main** OpenMP используется, выясняя количество потоков omp_get_num_threads() и индекс текущего потока

omp_get_thread_num(). Поиск максимального значения погрешности осуществляется с помощью записи значения макс значения блоков потока в общий массив, по которому после идет финальный поиск максимума внутри потока, после чего с помощью MPI_Allgather - по всем потокам.

Результаты

Сетка / Расчет	Результат на MPI+OpenMP, мс	на CPU, мс
1 1 1 / 40 40 40	7574	9843
1 1 2 / 40 40 20	3748	9850
1 2 1 / 40 20 40	3691	9845
2 2 2 / 20 20 20	246017	9952
2 2 4 / 20 20 10	268103	9890

Так как я тестировала на компьютере с 4-х ядерным процессором, физически была возможность запустить параллельно только 4 процесса. Т.е. при задании большего числа процессов показатели заметно ухудшались, потому что процессы выполнялись последовательно.

Выводы

1) Если поддержка OpenMP не будет включена в настройках компилятора, директивы OpenMP будут игнорироваться.
2) Количество параллельных потоков, создаваемых при выполнении приложения, в общем случае не является постоянной величиной. По умолчанию оно равняется числу установленных на компьютере процессоров. Однако, число потоков может также задаваться программистом вручную (с помощью функции omp_set_num_threads, или выражения num_threads).
3) Критические секции замедляют выполнение программы. Из-за критических секций потокам приходится ждать друг друга, это уменьшает приращение производительности. Кроме того, на вход в критические секции и на выход из них также затрачивается некоторое время.
4) OpenMP гораздо проще использования CUDA благодаря инкапсулированию.