Московский Государственный Университет имени М.В. Ломоносова
Факультет вычислительной математики и кибернетики

Отчёт по практичекому заданию в рамках курса «Суперкомпьютерное моделирование и технологии»

Выполнил: Зайцев Игорь Олегович, 624 группа

Постановка и описание задачи

Требовалось написать программу, численно решающую на трехмерной области следующее дифференциальное уравнение:

$$\frac{\partial^2 u}{\partial t^2} = \Delta u$$

С начальными условями:

$$u|_{t=0} = \varphi(x, y, z),$$
$$\left.\frac{\partial u}{\partial t}\right|_{t=0} = 0,$$

И краевыми условиями первого рода по x, z и второго рода по y.

Аналитическое решение данного уравнения имеет следующий вид:

$$\sin(\tfrac{\pi}{L_x}x) \cdot \sin(\tfrac{2\pi}{L_y}y) \cdot \sin(\tfrac{3\pi}{L_z}z) \cdot \cos(a_t \cdot t), a_t = \pi\sqrt{\tfrac{1}{L_x^2} + \tfrac{4}{L_y^2} + \tfrac{9}{L_z^2}}$$

Во втором задании требовалось реализовать параллельное вычисление с блочным разбиением и использованием MPI и openMP, и провести исследование параллельных характеристик на СК bluegene и polus.

Для третьего задания было необходимо реализовать данное задание с использованием MPI+CUDA на системе Polus

Описание решения

Логика программы разделена на те же классы что и во втором задании — Block и Comm

Block отвечает за вычисление и хранение данных на вычислительном узле

Comm – за обмен данными между узлами и их синхронизацию

Обмен данными между узлами реализован следующим образом: перед вычислением следующих значений в своем блоке, каждый узел обменивается со своими 6-ю соседями в пространственной сетке значениями на прилегающих к ним граням.

Для реализации данного алгоритма с использованием CUDA, требовалось изменить класс Block так, чтобы все трудоемкие вычисления производились на GPU – а именно вычисления слоев и вычисления максимальной ошибки на данном слое.

Для вычисления максимальной ошибки используется свертка, предоставленная библиотекой thrust

Одна проблема которую необходимо учитывать при вычислениях связана с тем, что если отношение (T * N) / (K * L) больше 1, то ошибки округления будут накапливаться с каждым шагом вычислений, и приводить к катастрофической потере точности.

В этом отношении

T – граница веменного интервала для переменной t

N – число точек разбиения по пространственным осям

K – число точек разбиения по временной оси

L – гранца пространственных интервалов для переменных x, y, z

Для решения этой проблемы, при вычислениях на СК T выбиралось достаточно малым, чтобы гарантировать что отношение будет меньше или равно ½.

Результаты профилирования

Анализ проводился для запуска программы на 2 MPI процессах с параметрами N=512, K=32.

Анализ показывает, что из 1.75 секунд работы одного процесса, 1.31 секунда приходится на вычисление ядер (порядка 75%), порядка 20 миллисекунд тратится на копирование данных между хостом и гпу и наоборот (около 1% времени), порядка 0.15 секунд тратится на аллокацию и инициализацию необходимых массивов на gpu, (порядка 8%).

Остальные 15% времени (0.26с) преимущественно занимает обмен данными между MPI процессами. Файл со статистикой приложен в архиве.

Результаты

Таблица с результатами расчетов MPI+openMP версии программы на IBM Polus (L=1, K=32)

Число точек по осям	Число процессоров	Погрешность	Время	Ускорение
128	10	1.28e-4	3.601	1
128	20	1.28e-4	2.863	1.26
128	40	1.28e-4	1.490	2.41
256	10	1.08e-5	28.24	1
256	20	1.08e-5	15.352	1.84
256	40	1.08e-5	9.625	2.93
512	10	7.24e-7	205.957	1
512	20	7.24e-7	120.656	1.7
512	40	7.24e-7	64.317	3.2

Таблица с результатами расчетов MPI+CUDA версии программы на IBM Polus (L=1, K=32)

Число точек по осям	Число процессоров	Погрешность	Время	Ускорение	Ускорение по сравнению с MPI+openMP
128	1	1.28e-4	0.16	1	22.5
128	2	1.28e-4	0.16	1	17.8
128	4	1.28e-4	0.22	0.73	6.8
256	1	1.08e-5	0.28	1	100.9
256	2	1.08e-5	0.2	1.4	76.8
256	4	1.08e-5	0.17	1.64	56.6
512	1	7.24e-7	1.45	1	142
512	2	7.24e-7	0.88	1.64	137
512	4	7.24e-7	0.52	2.79	124

(график не приводится, так как из за разницы времени вычисления в несколько порядков, на нем бы не было видно деталей)

Выводы

Как видно из таблиц, MPI+CUDA версия программы работает на 1-2 порядка быстрее MPI+openMP программы, запущенной на соответствующем числе узлов и с соответствующим размером сетки.

Такие результаты получены благодаря переносу на gpu как вычисления слоев, так и вычисления их погрешности (с использованием библиотеки thrust), а так же благодаря минимизации обменов данными между gpu и хостом.

Нужно заметить, что на малых (≤ 128) размерах сетки, вариант с 1 GPU и MPI процессом оказывается предпочтительнее варианта с двумя и более — так как на обмен данными между процессами тратится больше времени чем на вычисление этих достаточно малых слоев и их погрешностей на GPU.