МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Практическое задание по курсу

"Суперкомпьютеры и параллельная обработка данных"

Разработка параллельной версии программы для вычисления интеграла методом Монте-Карло

ОТЧЕТ

о выполненном задании

студента 324 учебной группы факультета ВМК МГУ

Спивакова Андрея Евгеньевича

Москва, 2022 г.

Оглавление

Постановка задачи

В рамках практического задания ставится задача вычисления интеграла методом Монте-Карло.
Требуется:

1. Реализовать параллельные алгоритмы вычисления интеграла методом Монте-Карло с помощью технологий параллельного программирования OpenMP и MPI.
2. Сравнить эффективность OpenMP и MPI-версий параллельной программы.
3. Исследовать масштабируемость полученной параллельной программы: построить графики зависимости времени исполнения от числа ядер/процессоров.
4. Найти количество ядер/процессоров, при котором время выполнения задачи перестаёт уменьшаться.

Описание алгоритма вычисления интеграла методом Монте-Карло

Последовательный алгоритм

Рассмотрим случайную величину u, распределенную равномерно на отрезке [a, b]. Тогда f(u) - тоже случайная величина, ее математическое ожидание выражается как

$$Ef(u) = \int_a^b f(x)\varphi(x)dx = \frac{1}{b-a}\int_a^b f(x)dx$$

Математическое ожидание можно оценить с помощью выборочного среднего, тогда

$$\int_a^b f(x)dx = \frac{b-a}{N}\sum_{i=1}^{N} f(u_i)$$

В многомерном случае величина (b-a) заменяется на n-мерный объем фигуры. Создадим класс генератора псевдослучайных чисел gener, работающего по формуле $U_{n+1} = (a * U_n + c) \bmod m$, где a = 1664525, c = 1013904223, m = 4294967295. Поскольку для работы программы необходимы числа из отрезка [0;1], разделим полученное число на 4294967295.0. Для возможности работы K процессов с одним экземпляром генератора, создадим std::vector длины K для хранения предыдущего сгенерированного числа для каждого процесса. После этого зафиксируем число N, и в цикле на N итераций посчитаем сумму значений исходной функции в очередной сгенерированной точке. Разделим сумму на число итераций, умножим на n-мерный объем и получим оценку интеграла. Код программы для N = 10^8,

$f(x,y,z) = \int_{-3.56}^{-0.49} \int_{-2.15}^{0.72} \int_{-6.13}^{-4.84} cos(-x^2 - y^2 - z^2) + 7sin(x^2 + y^2 + z^2)$ представлен в файле lin.cpp

Параллельный алгоритм

Поскольку вся вычислительная сложность алгоритма заключается в многократном подсчете функции, то, распределив N итераций равномерно по потокам (в случае OpenMP) или по процессам (в случае MPI), мы ускорим вычисление интеграла.
В OpenMP версии модификация заключается в добавлении pragma omp parallel, pragma omp for reduction(+:value). Код программы представлен в файле omp.cpp
В MPI версии в конце работы программы производится reduce результатов подсчета функции. Синхронизация производится с помощью MPI_Barrier. Код программы представлен в файле mpi.cpp

Результаты замеров времени выполнения

Ниже приведены результаты замеров времени выполнения программ на суперкомпьютере Polus: непосредственно в табличной форме и наглядно на 3D-графиках.

Программа была запущена в конфигурациях:

- 1, 2, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 нитей для OpenMP-версии и 1, 2, 4, 8, 16, 32 процесса для MPI

Каждая конфигурация была запущена 3 раза. Ниже приведены усредненные результаты. Ускорение считалось от результатов выполнения линейной версии программы (13 и 128 секунд для N = 10^7 и N = 10^8 соответственно)

Таблицы

OpenMp

Нумерация	Число итераций	Число нитей	Время выполнения, с	Ускорение, %
1	10^7	1	12.74825	1.93653846
2	10^7	2	6.6323	48.98230769

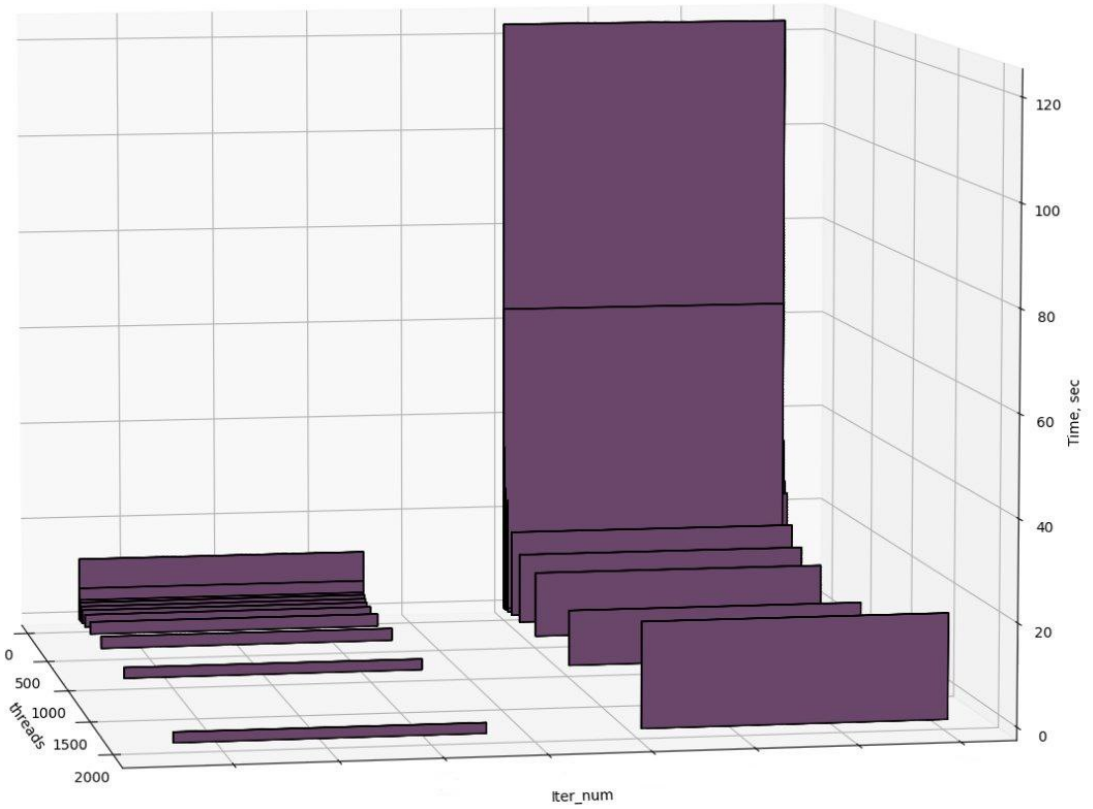
Нумерация	Число итераций	Число нитей	Время выполнения, с	Ускорение, %
3	10^7	4	4.489985	65.46165385
4	10^7	8	4.238565	67.39565385
5	10^7	16	3.805015	70.73065385
6	10^7	32	3.157065	75.71488462
7	10^7	64	2.72581	79.03223077
8	10^7	128	2.51976	80.61723077
9	10^7	256	2.48502	80.88446154
10	10^7	512	2.417345	81.40503846
11	10^7	1024	2.13627	83.56715385
12	10^7	2048	2.039295	84.31311538
13	10^8	1	122.7365	4.11210937
14	10^8	2	63.17055	50.64800781
15	10^8	4	35.0521	72.61554688
16	10^8	8	30.18985	76.41417969
17	10^8	16	34.2378	73.25171875
18	10^8	32	25.7606	79.87453125
19	10^8	64	23.5732	81.5834375
20	10^8	128	17.4554	86.36296875
21	10^8	256	14.0888	88.993125
22	10^8	512	13.07035	89.78878906
23	10^8	1024	11.070445	91.35121484
24	10^8	2048	20.53655	83.95582031

MPI

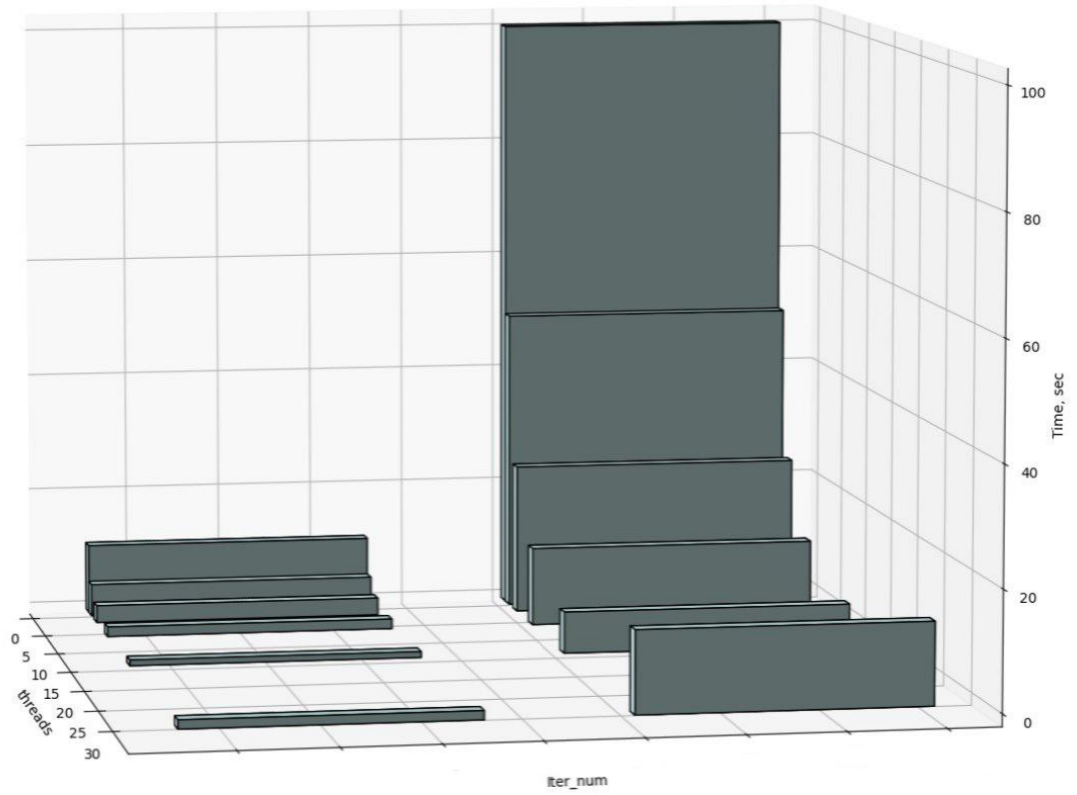
Нумерация	Число итераций	Число процессов	Время выполнения, с	Ускорение, %
1	10^7	1	11.60865	10.70269231
2	10^7	2	5.3773377	58.63586385
3	10^7	4	2.891052	77.76113846
4	10^7	8	1.6460438	87.33812462
5	10^7	16	0.9798	92.46307692
6	10^7	32	1.48387234	88.58559738
7	10^8	1	100.49453333	21.48864583
8	10^8	2	50.47346667	60.56760417
9	10^8	4	25.104	80.3875
10	10^8	8	13.13526667	89.73807292
11	10^8	16	6.94848667	94.57149479
12	10^8	32	13.71613333	89.28427083

Графики

OpenMp



MPI



Анализ результатов

Задача успешно поддалась распараллеливанию, удалось добиться значительного ускорения (вплоть до 90%) времени выполнения. OpenMp версия программы перестала ускоряться при числе нитей = 2048. MPI версия показала наилучший результат при числе процессов = 16, результат при 32 процессах ухудшился.

Выводы

Выполнена работа по разработке параллельной версии программы для вычисления интеграла методом Монте-Карло. Изучены технологии написания параллельных алгоритмов OpenMP и MPI. Проанализировано время выполнения разных версий программы, построены соответствующие таблицы и трехмерные графики.

Технология OpenMP очень проста в использовании и дает колоссальный прирост производительности на рассчитанных на многопоточные вычисления системах.

MPI можно назвать более низкоуровневой технологией: разработка MPI-программы знакомит с основами взаимодействия вычислительных узлов суперкомпьютера.