Московский государственный университет
имени М.В. Ломоносова

Факультет вычислительной математики и кибернетики

Отчет по практическому заданию

Вариант 20

Выполнил:
Студент гр. 321
Астраханцев Дмитрий Андреевич

Москва, 2023

Содержание

1 Введение

1.1 Постановка задачи

Реализовать параллельную версию предложенной программы на языке C с использованием OpenMP для различных входных данных и протестировать работоспособность на различном числе процессоров на суперкомпьютере Polus. Под размером входных данных будем подразумевать число N, равное размерам трехмерного массива A.

1.2 Используемые алгоритмы

Данную задачу отличает регулярная зависимость по данным. Данный класс задач плохо поддается распараллеливанию, по этой причине были использованы особые алгоритмы, которые нацелены именно на такие задачи:

- RedBlack 3D
- Распараллеливание по гиперплоскостям куба

2 Сделанные модификации. OpenMP

2.1 Оптимизации без использования OpenMP

Порядок переменных в циклах был заменен с `k`, `j`, `i` на `i`, `j`, `k`. Это ускорило работу программы на 25% за счет оптимизации доступа к памяти.

2.2 Оптимизации с использованием OpenMP

Т.к. функции `void init(void)` и `void verify(void)` не имеют регулярной зависимости по данным, то в них были добавлены строки

```
#pragma omp parallel for collapse(3) default(none) private(i, j, k) shared (A)
```

и

```
#pragma omp parallel for collapse(3) default(none) private(i, j, k) shared (A) reduction(+:s)
```

соответственно.

2.3 Алгоритм RedBlack3D

Модифицируем алгоритм RedBlack2D для трехмерного случая, высчитывая четность суммы индексов `(i + j + k) % 2` и разбивая вычисления на 2 подцикла. Подробнее реализацию можно посмотреть в файле `openmp_redblack.c`.

2.4 Распараллеливание по гиперплоскостям куба

Изначально данной задачи была написана функция, вычисляющая индексы ячеек, принадлежащих каждой из гиперплоскостей. Данная функция хорошо распараллеливается, однако из-за общей структуры данных она имеет критическую секцию, что замедляет ее работу. Из-за этого данный код пришлость полностью переписать.

Был использован другой подход к реализации. Можно заметить, что

вычисление индексов H-ой гиперплоскости сводится к классической комбинаторной задаче о шариках и перегородках.

Пусть есть N шариков и M перегородок. Перегородки ставятся в промежудках между шариками и образуют "ящики". Рассмотрим пример для $N = 3$ и $M = 2$.

Схема	Индексы
$\|\|\circ\circ\circ$	$(0,0,3)$
$\|\circ\|\circ\circ$	$(0,1,2)$
$\|\circ\circ\|\circ$	$(0,2,1)$
$\|\circ\circ\circ\|$	$(0,3,0)$
$\circ\|\|\circ\circ$	$(1,0,2)$
$\circ\|\circ\|\circ$	$(1,1,1)$
$\circ\|\circ\circ\|$	$(1,2,0)$
$\circ\circ\|\|\circ$	$(2,0,1)$
$\circ\circ\|\circ\|$	$(2,1,0)$
$\circ\circ\circ\|\|$	$(3,0,0)$

Легко заметить, что данные индексы образуют 3-ю гиперплоскость в трехмерном массиве. Используем этот факт в реализации задачи. Перепишем функцию relax так, чтобы обход шел не по индексам, а по элементам гиперплоскости. Подробнее реализацию можно посмотреть в файле `openmp_hyperplain.c`.

3 MPI

Для версии программы, основанной на использовании MPI, код был существенно переработан.

1. Исходный куб размера $N \times N \times N$ был разбит на более мелкие части размера $m \times m \times m$, где m подбирается в зависимости от размера исходных данных.
2. Каждый MPI-процесс получает один или несколько кубов для обработки, после этого передает граничные элементы в следующий процесс, который обрабатывает соседний куб.
3. Отдельно стоит рассмотреть граничные элементы, которые, вообще говоря, могут иметь форму отличную от куба, если m не является делителем N.

Подробнее реализацию можно посмотреть в файле `mpi.c`.

4 Результаты работы

Результатом работы является тестирование, разработанных программ на суперкомпьютере Polus. Для тестирования было выбрано количество процессоров равное 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 20, 40, 60, 80, 100, 120, 140, 160 и входные данные N={200, 500, 700, 1000, 1200}. Из-за ограничения в 15 минут (900 секунд), для некоторых параметров значения так и не были получены. Эти параметры обозначены темно-синим цветом на диаграмме.

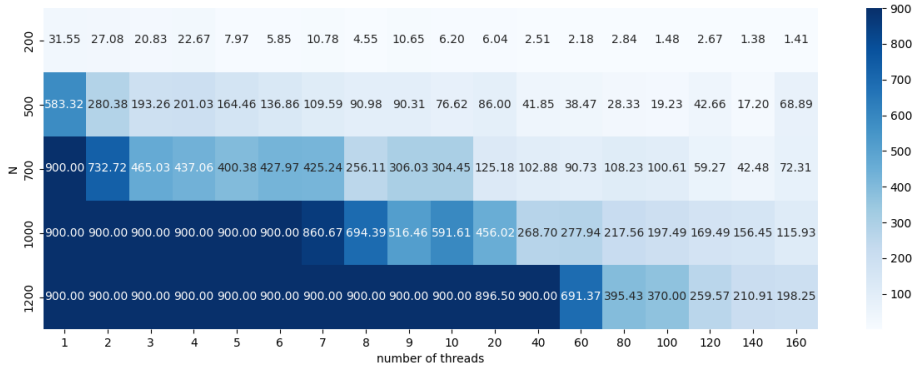

Рис. 1: Алгоритм RedBlack3D с флагом `O0`

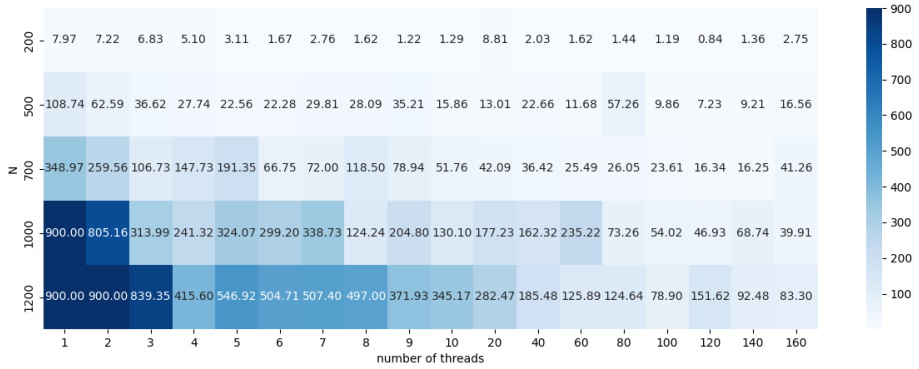

Рис. 2: Алгоритм RedBlack3D с флагом `O2`

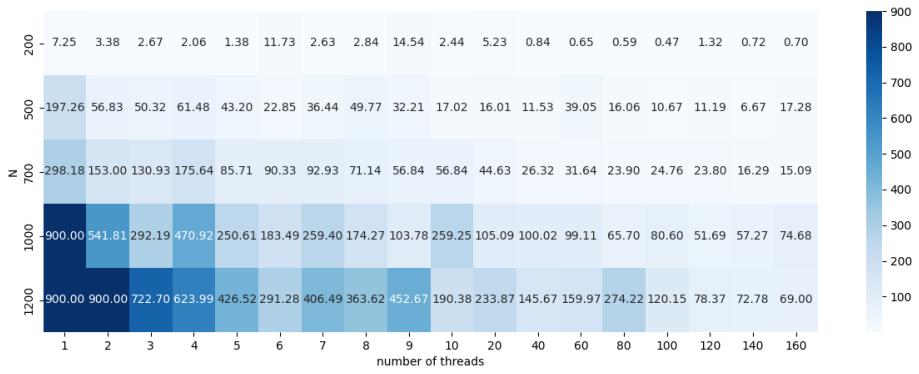

Рис. 3: Алгоритм RedBlack3D с флагом `O3`

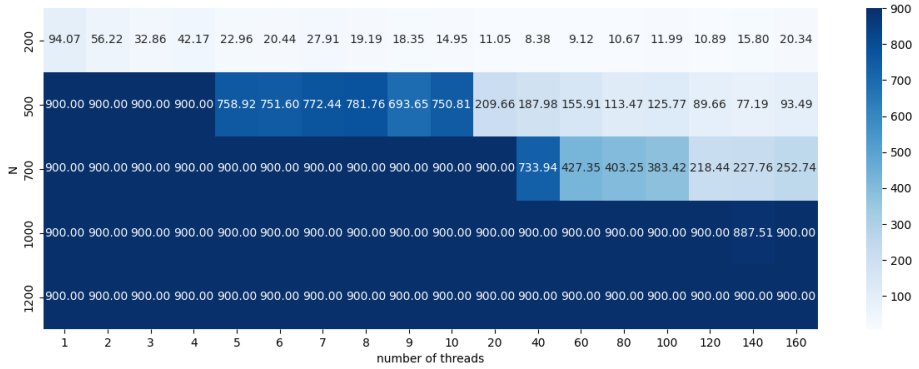

Рис. 4: Распараллеливание по гиперплоскостям куба с флагом `O0`

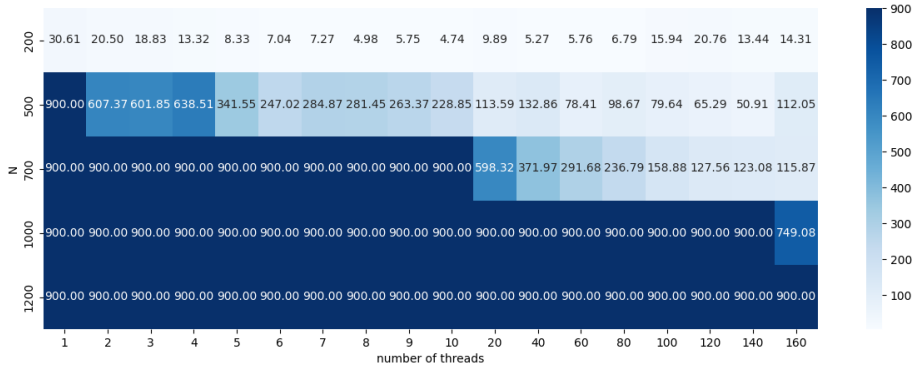

Рис. 5: Распараллеливание по гиперплоскостям куба с флагом `O2`

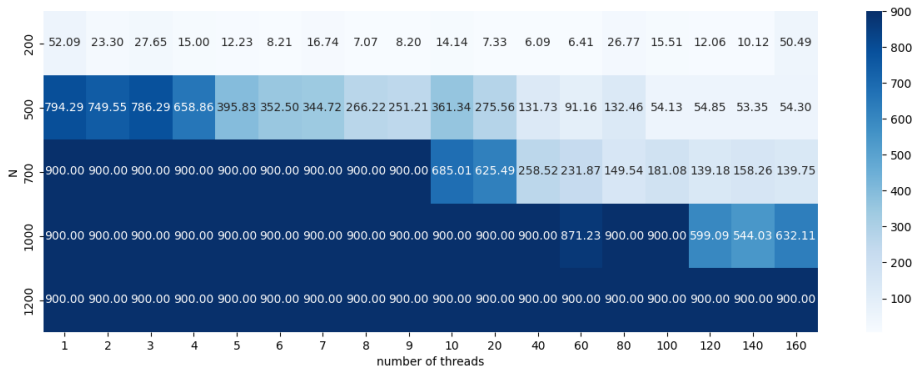

Рис. 6: Распараллеливание по гиперплоскостям куба с флагом `O3`

Для визуализации данных была использована библиотека `seaborn` для `Python 3`.

5 Выводы

Очевидно, что RedBlack3D алгоритм оказался гораздо быстрее, однако это компенсируется тем, что данный алгоритм в принципе не гарантирует какую-либо точность, а лишь позволяет обеспечить сходимость узкого класса методов.

Можно заметить, что с определенного порога при увеличении числа нитей алгоритм распараллеливания по гиперплоскостям дает худший результат. Скорее всего данный эффект обусловлен наличием в программе разделяемых ресурсов. Таким образом большое число нитей увеличивает расходы на передачу управления между ними, что замедляет работу программы.