Московский государственный университет
имени М. В. Ломоносова
Факультет вычислительной математики и кибернетики

Параллельные высокопроизводительные вычисления

Отчёт

О выполненном задании №2
“Параллельная сортировка Бэтчера”

Выполнил:
студент 523 группы
Латыпов Ш. И.
17.12.2023

Москва
2023

Содержание

Описание задачи

Задан массив элементов типа double, при этом считаем, что количество элементов массива достаточно большое, чтобы поместиться в память одного процесса.

На входе: на каждом процессе одинаковое количество элементов массива. (Если на некоторых процессах элементов массива меньше чем во всех остальных, тогда необходимо ввести фиктивные элементы, например, со значением DBL_MAX или -DBL_MAX в зависимости от направления сортировки.)

Цель: разработать и реализовать алгоритм, обеспечивающий параллельную сортировку методом Бэтчера элементов массива в соответствии с заданным направлением. (по возрастанию или по убыванию) Следует реализовать сортировку на каждом отдельном процессе и сеть сортировки Бэтчера.

На выходе: на каждом процессе одинаковое количество элементов массива. Все элементы массива принадлежащие одному процессу отсортированы по возрастанию (убыванию) Каждый элемент массива одного процесса должен быть меньше (больше) по по сравнению с элементами массива любого процесса с большим рангом, за исключением фиктивных элементов.

Требования к программе:

- Программа должна демонстрировать эффективность не менее 50% от максимально возможной, на числе вычислительных ядер, не менее 48 (Запуск программы на параллельном кластере в этом задании обязательно!)

Метод решения

Функция main принимает на вход через параметры командной строки число n.

Далее программа выполянет MPI инициализацию. Процесс с рангом 0 генерирует вектор размера n, и с помощью функции MPI_Scatter(...) рассылает этот буфер по остальным процессам. Каждый процесс сортирует у себя свою локальную часть и начинает сортировку Бэтчера.

В решении данной задачи сортировки со слиянием Бэтчера основными функциями являются рекурсивные функции int B(first, step, count, rank, vector) и int S(first1, first2, step, n, m, rank, vector).

- B(first, step, count, rank, vector) — процедура рекурсивного построения сети сортировки группы линий (first, step, count).
- S(first1, first2, step, n, m, rank, vector), — рекурсивная процедура слияния двух групп линий (first1, step, n) и (first2, step, m).

В функции B(...) массив делится на две части, и далее для каждой из частей рекурсивно запускаются функции B(first, step, count1) и B(first + step $*$ count1, step, count $-$ count1), где count1 - количество элементов в первой половине массива (Вычисление: count1 = count / 2 + count % 2). После этого происходит запуск фукнции S(first, first + step $*$ count1, step, count1, count $-$ count1).

При этом есть дополнительные проверки на размеры массива: если он меньше 2, то функция завершает работу; если он равен двум, то в массив компаратора добавляется новый компаратор со значениями (first, first + step), после этого функция завершает работу.

В функции S(...) объединение элементов массивов с нечетными номерами и отдельно — с четными, после чего, с помощью заключительной группы компараторов, обрабатываются пары соседних элементов с номерами вида (2i, 2i + 1), где i — натуральные числа от 1 до $\frac{p}{2} - 1$.

Рекурсивный запуск фукнций S(first1, first2, 2 $*$ step, n1, m1) и (first1 + step, first2 + step, 2 $*$ step, n $-$ n1, m $-$ m1), где n1 = n $-$ n / 2, m1 = m $-$ m / 2;

В этой функции также присутствует проверка размера массивов, при котором, если размер одного из массивов равен 0, то фукнция завершает работу, а если оба массива имеют размер 1, то в вектор компараторов добавляется новый компаратор.

Обработка компараторов выполняется функцией add_comp(int first, int second, int rank). Когда происходит вызов этой функции, переменные first и second принимают значения рангов процессов, между которыми должен произойти обмен. Эти две перемнные сравниваются со значением rank процесса, и затем, если имеются совпадения, то процессы обмениваются своими локальными буферами для обмена числами в них.

Используемая вычислительная система

Использована система: ПВС «IBM Polus», 2018 год.

Polus - параллельная вычислительная система, состоящая из 5 вычислительных узлов. Основные характеристики каждого узла:

- 2 десятиядерных процессора IBM POWER8 (каждое ядро имеет 8 потоков) всего 160 потоков
- Общая оперативная память 256 Гбайт (в узле 5 оперативная память 1024 Гбайт) с ECC контролем
- 2 x 1 ТБ 2.5” 7K RPM SATA HDD
- 2 x NVIDIA Tesla P100 GPU, 16Gb, NVLink
- 1 порт 100 ГБ/сек

Характеристики вычислительной системы

Коммуникационная сеть – Infiniband / 100 Gb
Система хранения данных – GPFS
Операционная система – Linux Red Hat 7.5
Производительность кластера (Tflop/s): 55,84 (пиковая), 40,39 (Linpack)

Проверка программы

Запуск программы производился с помощью команды запуска задач mpisubmit.pl. Тестировалось на буферах размеров $10^6, 5 * 10^6, 10^7, 5 * 10^7, 10^8$ с использованием 1, 2, 4, 8, 16, 32, 48, 60 процессов. Для каждой пары параметров (размер буфера; количество процессов) было произведено два замера по времени, чтобы вычислить их среднее значение. Для всех пар параметров программы n и p посчитаны время выполнения программы, ускорение программы и эффективность, а также теоретическая максимальная эффективность программы. Все данные отображены в виде таблиц и графиков.

Таблица с замерами времён для всех значений n и p представлены в таблице:

n	1	2	4	8	16	32	48
1000000	0,6181	0,3197	0,1805	0,0974	0,0622	0,0399	0,0274
5000000	2,8676	1,7815	0,8374	0,5104	0,2817	0,2221	0,1151
10000000	6,4197	3,877	2,0208	1,0416	0,5631	0,3295	0,2454
50000000	32,5867	20,8316	10,011	5,4257	2,8897	1,778	1,0902
100000000	66,6127	34,3491	17,7223	9,2966	5,7418	3,2451	2,11

Таблица с показателями ускорения программы:

n	1	2	4	8	16	32	48
1000000	1	1,9332	3,424	6,3482	9,9341	15,4885	22,5818
5000000	1	1,6097	3,4245	5,6189	10,1806	12,9091	24,9236
10000000	1	1,6558	3,1768	6,1633	11,4002	19,4859	26,1553
50000000	1	1,5643	3,2551	6,0059	11,2768	18,3275	29,8907
100000000	1	1,9393	3,7587	7,1653	11,6015	20,527	31,5704

Таблица с показателями эффективности программы:

n	1	2	4	8	16	32	48
1000000	100,00%	96,66%	85,60%	79,35%	62,09%	48,40%	47,05%
5000000	100,00%	80,48%	85,61%	70,24%	63,63%	40,34%	51,92%
10000000	100,00%	82,79%	79,42%	77,04%	71,25%	60,89%	54,49%
50000000	100,00%	78,21%	81,38%	75,07%	70,48%	57,27%	62,27%
100000000	100,00%	96,96%	93,97%	89,57%	72,51%	64,15%	65,77%

Таблица с показателями теоретически-максимальной эффективности программы:

n	1	2	4	8	16	32	48
1000000	100,00%	100,00%	95,22%	86,92%	76,86%	66,59%	60,89%
5000000	100,00%	100,00%	95,70%	88,12%	78,76%	69,00%	63,48%
10000000	100,00%	100,00%	95,88%	88,57%	79,49%	69,93%	64,49%
50000000	100,00%	100,00%	96,24%	89,50%	81,00%	71,89%	66,64%
100000000	100,00%	100,00%	96,37%	89,86%	81,58%	72,66%	67,49%

Соответствующие графики работы программы:

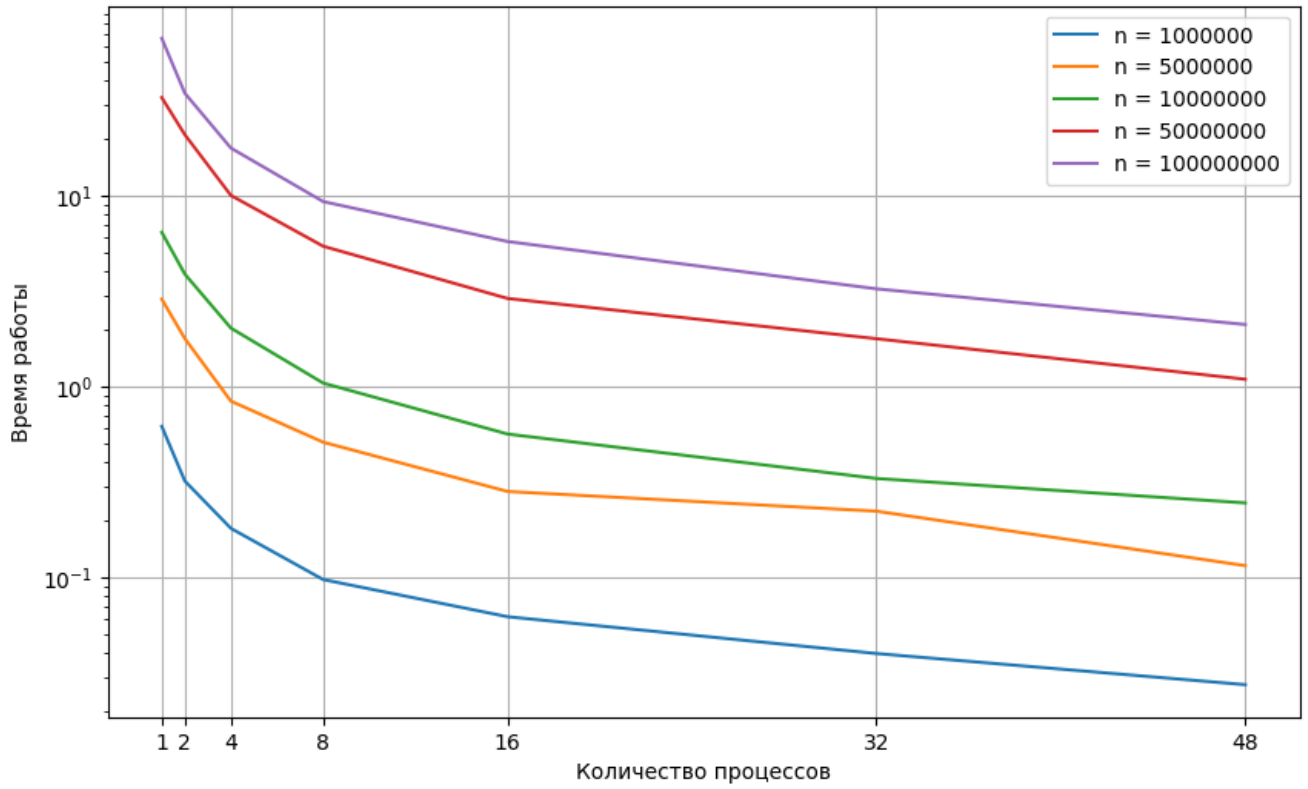

Рис. 1: Время работы программы

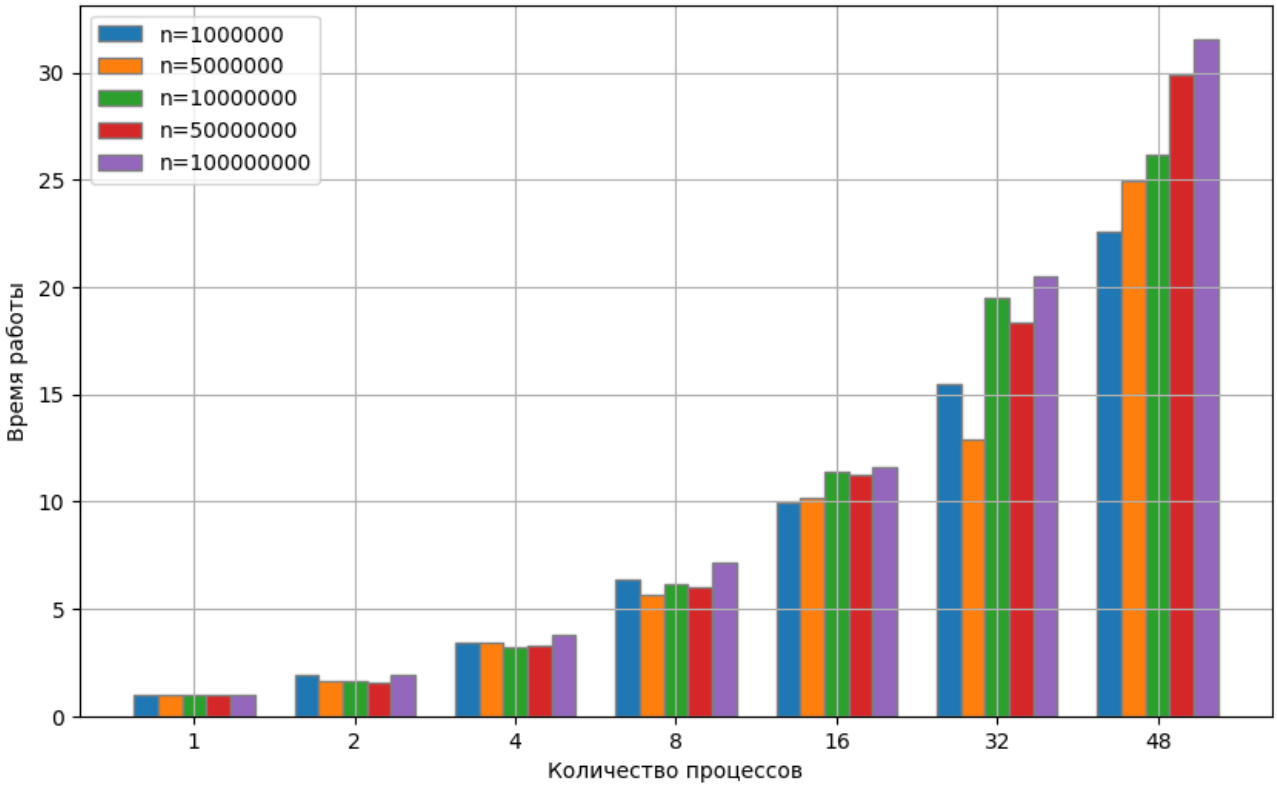

Рис. 2: Ускорение программы

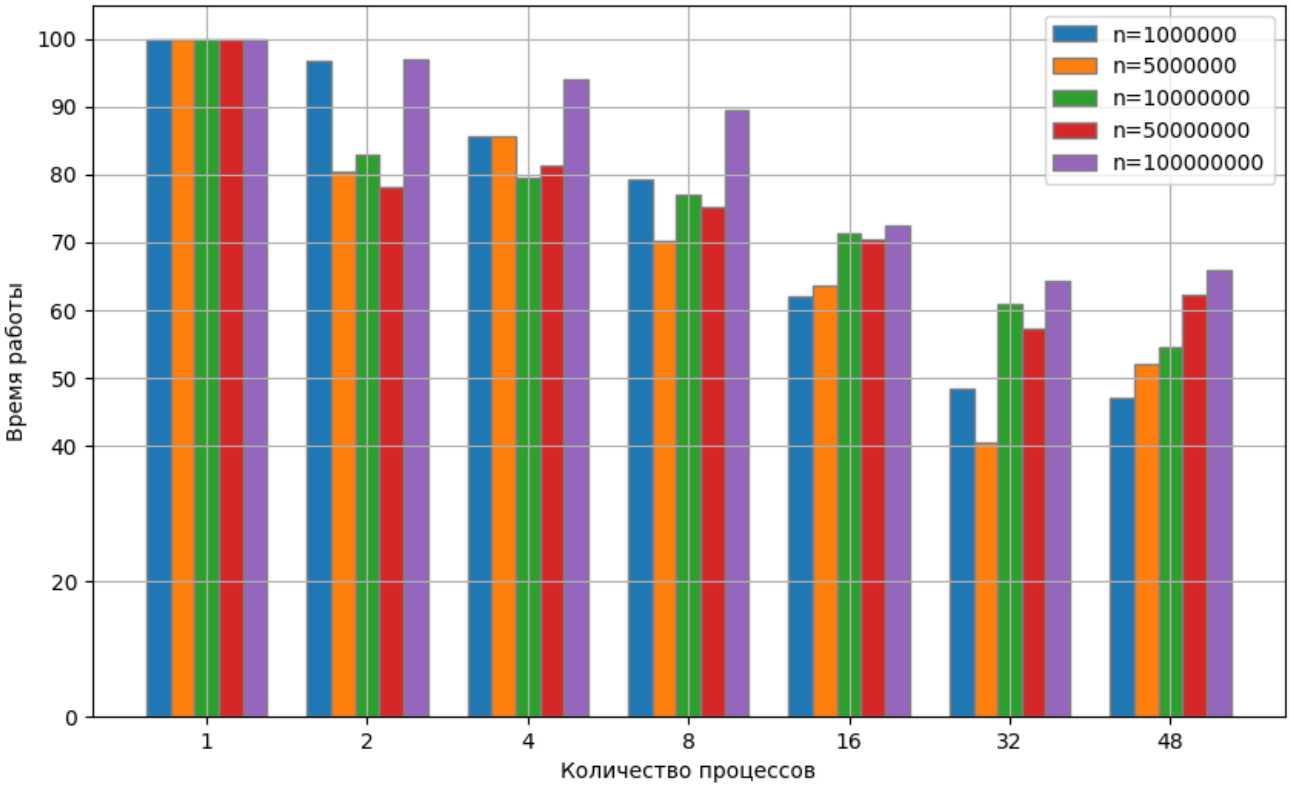

Рис. 3: Эффективность программы

Анализ полученных результатов

Программа демонстрирует эффективность, достаточно близкой к теоретически-максимальной. Эффективность программы сохраняет значения $> 50\%$ даже при количестве процессов $p = 48$.

Значения величины $\frac{T_1}{n*log_2(n)}$ соответственно равны:

$n = 10^6 :$ $3.1 * 10^{-8}$
$n = 5 * 10^6 :$ $2.58 * 10^{-8}$
$n = 10^7 :$ $2.76 * 10^{-8}$
$n = 5 * 10^7 :$ $2.55 * 10^{-8}$
$n = 10^8 :$ $2.5 * 10^{-8}$

Число тактов сортировки Бэтчера:

$p = 2 :$ $tacts = 1$
$p = 4 :$ $tacts = 3$
$p = 8 :$ $tacts = 6$
$p = 16 :$ $tacts = 10$
$p = 32 :$ $tacts = 15$
$p = 48 :$ $tacts = 21$

Аналитические выражения для ожидаемого времени, ускорения и эффективности сортировки:

$$T(n,\,p) = K\frac{n}{p}\left(\log_2\frac{n}{p} + \frac{\lceil\log_2 p\rceil\lceil\log_2 p+1\rceil}{2}\left(1+\frac{\tau_s}{K}\right)\right),$$

$$E(n,\,p) = \left(1 - \log_n p + \frac{\lceil\log_2 p\rceil\lceil\log_2 p+1\rceil}{2\log_2 n}\left(1+\frac{\tau_s}{K}\right)\right)^{-1}.$$

$$E^{max}(n,p) \approx \left(1 + \frac{log_n p}{2} log_2 \frac{p}{2}\right)^{-1}$$

Где:

- $T(n,p)$ - ожидаемое время выполнения,
- $E(n,p)$ - эффективность использования p процессов для обработки n элементов,
- $E^{max}(n,p)$ - максимальная теоретическая эффективность алгоритма,
- n - колчичество элементов массива,
- p - количество процессов,
- τ_s - время переслыки элемента массива
- K - время, необходимое для выполнения операции сортировки (константа). В нашем случае (учитывая сложность std::sort сортировки $O(n * log_2 n)$) значение K в зависимости от наших переменных n и p принимает значение: $K \approx 2.7 * 10^{-8}$

Аналитические значения эффективности работы программы взяты из издания "Параллельные алгоритмы сортировки больших объемов данных", автор: М.В.Якобовский, 2004, 2008.

$$E^{max}(n,p) = \frac{t(n,1)}{pt(n,p)} = \frac{log_2 n}{log_2 n + s_p - log_2 p} \approx \frac{1}{1 + log_n p(log_2 p - 1)/2}$$

Тестирование программы

Для проверки работоспособности алгоритма в функции реализована отдельная секция. Если через параметры командной строки в качестве второго аргумента было вписано 'test', то запускается скрипт, который дополнительно проверяет порядок чисел в массиве, а именно сверяет элементы на $i-1$ и i позициях, проверяя, что $data[i-1] <= data[i]$. Пример запуска тестирующего режима: ./bsort 4 test

Вывод программы при тестировании:

- TESTING - означает, что тестирование началось,
- NOT OK: data[i_1] = <число1>, data[i_2] = <число2> - означает, что элементы не расположены по порядку возврастания, и на соответствующих позициях отображаются хранящиеся значения.
- Test end - означает, что тестирование закончилось.

При успешном прохождении тестирования, в конце программы будет вывод:
TESTING
Test end

В случае неуспешного:
TESTING
NOT OK data[i_1] = <число1>, data[i_2] = <число2>
...
NOT OK data[i_1] = <число1>, data[i_2] = <число2>
Test end

Приложение

К данному отчёту прилагается файл task2.cpp с оригинальным кодом описанной программы.