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

Скрябин Глеб Денисович

Задание 2
Параллельная сортировка Бэтчера

Отчет о работе по курсу "Параллельные высокопроизводительные вычисления"

Автор:
гр. 523

Москва, 2023

Содержание

1 Постановка задачи

1.1 Условие

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 сравнению с элементами массива любого процесса с большим рангом, за исключением фиктивных элементов.

1.2 Требования к программе

1. Программа может быть гибридной: одновременно использовать технологию MPI, для обеспечения взаимодействия вычислительных узлов, и одну из двух технологий Posix threads или OpenMP, для взаимодействия процессов, запущенных на ядрах процессоров

2. 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!)

2 Описание метода решения

2.1 Сети Бэтчера

Сети Бэтчера — наиболее быстродействующие из рассматриваемых масштабируемых сетей сортировки. Для построения сети обменной сортировки со слиянием можно использовать описываемый далее рекурсивный алгоритм или приводящий к аналогичному результату нерекурсивный алгоритм.

Для сортировки массива, содержащего p элементов с номерами [1, ..., p] следует разделить его на две части. В первой оставить $n = \lfloor \frac{p}{2} \rfloor$ элементов с номерами [1, ..., n], а во второй m = p – n элементов с номерами [n + 1, ..., p]. Далее следует отсортировать каждую из частей и объединить результаты сортировки с помощью (n, m)-сети нечетно-четного слияния Бэтчера. В сети нечетно-четного слияния отдельно объединяются элементы 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 $\lfloor \frac{p}{2} \rfloor - 1$.

2.2 Обозначения для программы

Перед разбором работы алгоритма определим некоторые символы:

1. P — количество MPI-процессов;
2. N — длина исходного массива
3. $N_{extended}$ — длина расширенного массива, кратная количеству MPI-процессов P;
4. M — длина части массива, принадлежащая каждому процессу, $M = N_{extended}/P$.

2.3 Этапы работы программы

Разберем этапы работы программы:

1. Определение размера исходного массива;
2. Инициализация MPI. Создаем группу процессов и область связи при помощи функции MPI_Init;

3. Составление расписания сети сортировки Бэтчера;
4. Заполнение части массива длинной M на каждом процессе;
5. Запуск таймера при помощи функции MPI_Wtime;
6. В каждом процессе сортируем полученные части массива процедурой std::sort;
7. Проходим по расписанию сети сортировки и производим параллельное распределение элементов в частях массивов по их величине при помощи функций MPI_Send и MPI_Recv;
8. MPI_Barrier и остановка таймера;
9. Составление итогового массива в мастер-процессе при помощи функции MPI_Gather;
10. Проверка корректности сортировки;
11. Вывод всех результатов в консоль;
12. Завершаем MPI при помощи функции MPI_Finalize.

3 Характеристики вычислительной системы

Все запуски с замерами параметров программы проводились с использованием вычислительной системы IPM Polus

3.1 Parallel HPS "IBM Polus"

- Пиковая производительность – 55.84 TFlop/s;
- Производительность (Linpack) – 40.39 TFlop/s;
- Вычислительных узлов – 5.

На каждом узле:

- Процессоры IBM Power8 – 2;
- NVIDIA Tesla P100 – 2;
- Число процессорных ядер – 20;
- Число потоков на ядро – 8;
- Оперативная память – 256 Гбайт (1024 Гбайт узел 5);
- Коммуникационная сеть – Infiniband / 100 Gb;
- Система хранения данных – GPFS.

Программное обеспечение:

- Операционная система Linux Red Hat 7.5;
- Компиляторы C/C++, Fortran;
- Поддержка OpenMP; Программные средства параллельных вычислений стандарта MPI: библиотека IBM Spectrum MPI, Open MPI;
- Планировщик IBM Spectrum LSF;
- CUDA 9.1; Математическая библиотека IBM ESSL/PESSL.

4 Результаты запусков

4.1 Сводные таблицы

В таблице 1 представлено время работы сортировки в зависимости от длины массива и количества процессов. В таблице 2 представлены рассчитанные параметры для массива размером 100M.

$P \backslash Size$	100K, sec	1M, sec	10M, sec	100M, sec
1	0,0401492	0,470324	5,4124	63,0803
2	0,0312661	0,263613	2,94646	33,6102
4	0,0235204	0,154861	1,71112	19,1179
8	0,0157612	0,115986	1,10143	11,67
16	0,0122145	0,096481	0,704219	7,19171
32	0,0110852	0,107363	0,519218	4,94746
40	0,0110852	0,112043	0,501094	5,16515

Таблица 1: Время работы сортировки в зависимости от длины массива и количества процессов.

P	T, sec	E, %	E_{max}, %	E/E_{max}, %	S	S_{max}
1	63,0803	100,00	100,00	100,00	1,000	1,000
2	33,6102	93,84	100,00	93,84	1,877	2,000
4	19,1179	82,49	96,37	85,59	3,300	3,855
8	11,67	67,57	89,86	75,19	5,405	7,189
16	7,19171	54,82	85,29	64,28	8,771	13,646
32	4,94746	39,84	81,58	48,84	12,750	26,106
40	5,16515	30,53	80,51	37,92	12,213	32,204

Таблица 2: Параметры для массива размером 100M.

4.2 Обозначения для анализа полученных результатов

1. P — количество MPI-процессов;
2. T — время, затраченное на сортировку;
3. E — фактическая эффективность;
4. E_{max} — максимальная эффективность, рассчитанная аналитически;
5. E/E_{max} — отношение фактической эффективности к максимальной;
6. S — фактическое ускорение;
7. S_{max} — максимальное ускорение, рассчитанное аналитически.

4.3 Графики

На логарифмическом графике 1 представлены результаты замеров времени, затраченного на сортировку массивов длиной 100K, 1M, 10M, 100M элементов:

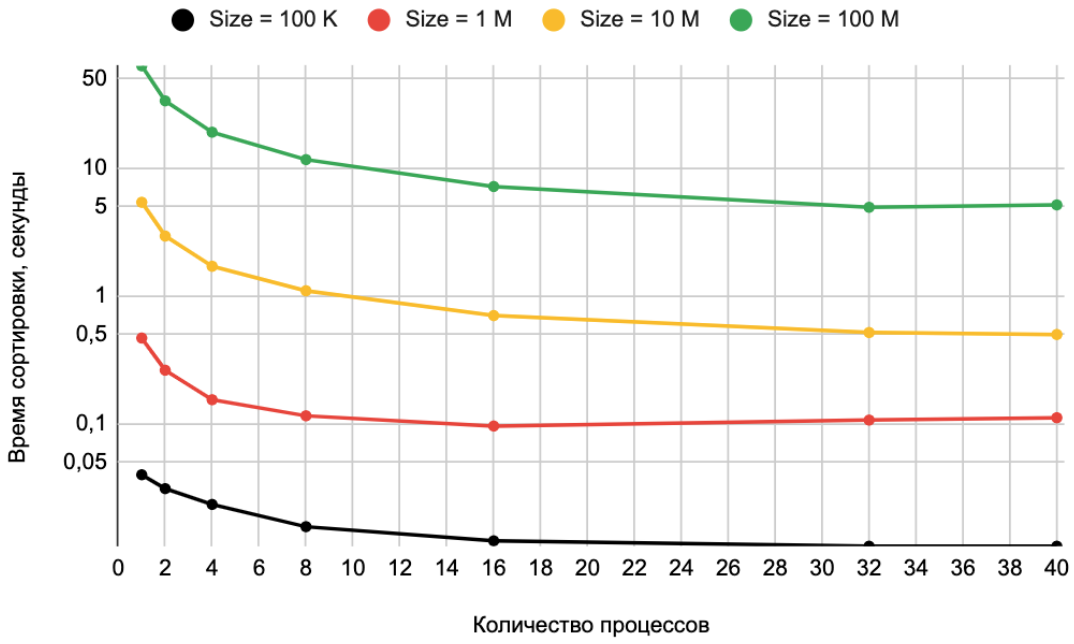

Рис. 1: Время выполнения сортировки при разных размерах массива.

На графике 2 представлены результаты замеров эффективности алгоритма, максимально возможная эффективность, рассчитанная аналитически, а так же их отношение:

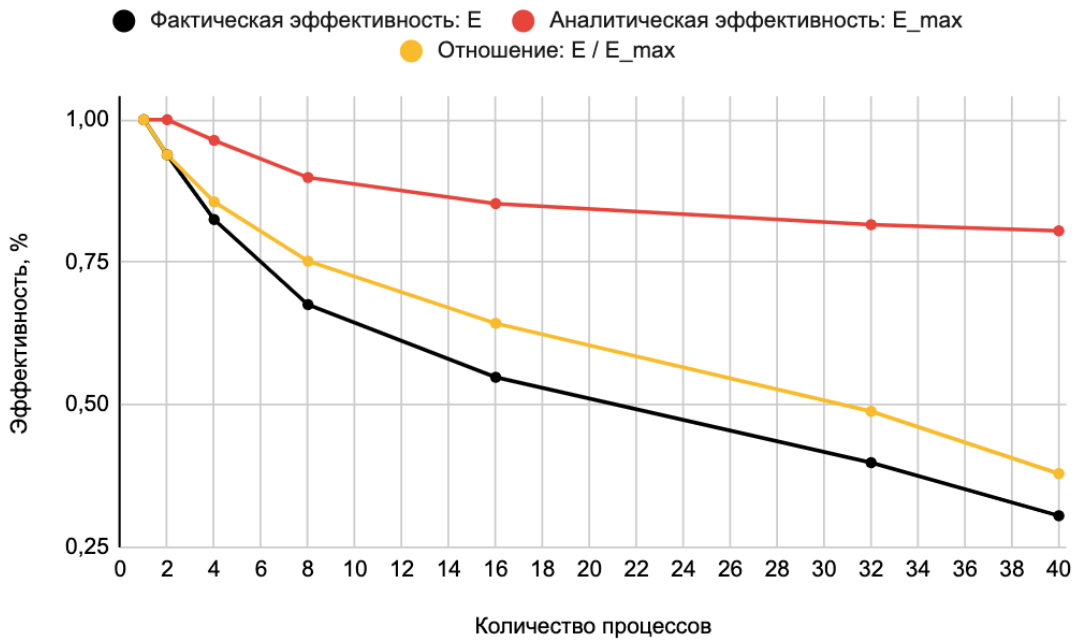

Рис. 2: Показатели эффективности.

5 Анализ полученных результатов

Значение величины $T_1/[N * \log(N)]$ для массивов разного размера:

1. Размер = 100K: 3,487E-08;
2. Размер = 1M: 3,404E-08;
3. Размер = 10M: 3,358E-08;
4. Размер = 100M: 3,424E-08.

Число тактов и компараторов сортировки Бэтчера для разного количества процессов:

1. 2 процесса: 1 такт и 1 компаратор;
2. 4 процесса: 3 такта и 5 компараторов;
3. 8 процессов: 6 тактов и 19 компараторов;
4. 16 процессов: 10 тактов и 63 компаратора;
5. 32 процесса: 15 тактов и 191 компаратор;
6. 40 процессов: 20 тактов и 283 компаратора.

Аналитические выражения для ожидаемого времени, ускорения и эффективности сортировки:

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

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

$$E_{max}(n,p) = \frac{\log_2 n}{\log_2 n + s_p - \log_2 p} \approx \frac{1}{1+\log_2 p\,(\log_2 p - 1)\,/2}$$