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

ОТЧЕТ

По курсу «Параллельные высокопроизводительные вычисления»

Практическое задание №2.

«Параллельная сортировка Бэтчера»

Работу выполнил:
Студент группы 528
Факультета вычислительной математики и кибернетики
Цирунов Леонид Александрович

3 декабря 2023 года

Описание условия

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.

Требования к программе:

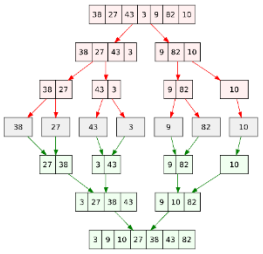
1. Программа может быть гибридной: одновременно использовать технологию MPI, для обеспечения взаимодействия вычислительных узлов, и одну из двух технологий Posix threads или OpenMP, для взаимодействия процессов, запущенных на ядрах процессоров
2. Программа должна демонстрировать эффективность не менее 50% от максимально возможной, на числе вычислительных ядер, не менее 48 (Запуск программы на параллельном кластере в этом задании обязательно!)

Описание метода решения

Для составления расписания сортировки использовался алгоритм и код из первого практического задания.

Алгоритм из первого практического задания:

Для решения поставленной задачи был реализован частичный алгоритм четно-нечетной сортировки Бэтчера, в котором отсутствует логика работы с массивом и логика слияния. Алгоритм используется только для составления расписания. Сортировка массивов в проверке производится непосредственно по построенному расписанию.
Для построения сети использовался рекурсивный алгоритм.
При сортировке массива из n элементов с номерами **[0, …, n-1]** следует разделить его на две части: в первой оставить **p=**$\left\lceil \frac{n}{2} \right\rceil$ элементов с номерами **[0, …, p]**, а во второй **m = n – p** элементов с номерами **[p+1, …, n]**.
Наглядно принцип деления исходного массива и слияния частей представлен на рисунке:



Функции из первого практического задания:

BatcherSort – функция рекурсивного делит массив на два подмассива из **p** и **m** элементов соответственно, после чего вызывает функцию слияния BatcherMerge для этих подмассивов. В реализации отсутствует работа с массивом, оперирование происходит только с индексами условного массива элементов.

BatcherMerge – функция рекурсивного слияния двух групп линий. 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. Данные пары записываются в массив компараторов comparators для дальнейшего использования. В реализации опять же отсутствует работа с массивов, оперирование идет только с индексами, и происходит заполнение массива сравниваемых пар номеров строк, соединяемых i-м компаратором сравнения перестановки.

Описание решения:

Дополнительно реализованы следующие функции:

Comparator – функция отвечающая за коммуникацию процессов и слияния, т.е. обмен элементов локальных данных и перераспределение их по возрастанию, так, чтобы в итоге каждый элемент массива одного процесса был меньше по сравнению с

элементами массива любого процесса с большим рангом. Функция работает согласно построенному расписанию компараторов. Данная функция написана в соответствии с псевдокодом представленным на слайдах 74–75 лекции.
https://lira.imamod.ru/msu202209/L05_20221017_YakobovskiyMV.pdf.

Для формирования массива, содержащего меньшие(большие) элементы использовался алгоритм из книги М.В. Якобовского “Введение в параллельные методы решения задач” стр. 152.

generateRandomDouble – генерирует случайные вещественные числа, используется для инициализации массива случайными данными.

checkSorted – функция для проверки, что полученный массив отсортирован.

Главная функция программы ***main*** выполняет следующие шаги:

- Получение из аргументов командной строки размера вектора.
- Инициализация MPI и определение числа процессов и рангов.
- Генерация локальных данных в каждом процессе.
- Составление расписания сети сортировки Бэтчера.
- Сортировка локальных кусков массива с использованием функции std::sort из библиотеки <algorithm>.
- Выполнение параллельного слияния элементов.
- Сбор итоговых данных на процессе с рангом 0.
- Проверка, что массив отсортирован и вывод результата работы по времени.

Для компиляции программы и постановки в очередь на Polus написан Makefile.

- Компиляция:
  make compile

- Постановка в очередь на polus:
  make run_polus

- Запуск со значением размера массива по умолчанию (100 тыс.):
  make run

- Запуск с явным указанием размера массива:
  mpirun -np P ./pbsort N
  , где *N* – количество элементов конечного массива, *P* – количество процессов.

Описание используемой вычислительной системы

Тестовые прогонки и получение результатов проводились с использованием вычислительной системы IPM Polus

ПВС «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

Численные данные

В таблице представлено время работы сортировки в зависимости от длины массива и количества процессов:

p \ N	2^15	2^18	2^20	2^24	2^27	2^29	2^30
1	0,004155	0,021476	0,087584	1,613090	15,202900	95,631767	134,276000
2	0,002188	0,018407	0,047668	0,843666	7,821850	35,765600	69,265167
4	0,001537	0,014336	0,032781	0,457856	4,168353	17,611467	36,821833
8	0,001031	0,004601	0,020161	0,277641	2,509672	10,612260	21,337150
16	0,000622	0,002954	0,012129	0,202526	1,558143	6,430603	12,846067
32	0,000414	0,003179	0,007456	0,125086	1,040090	4,244545	8,553143
48	0,000338	0,003888	0,005701	0,095391	0,844658	3,385050	6,344230
56	0,000350	0,004130	0,005793	0,092521	0,843487	3,222747	6,175990

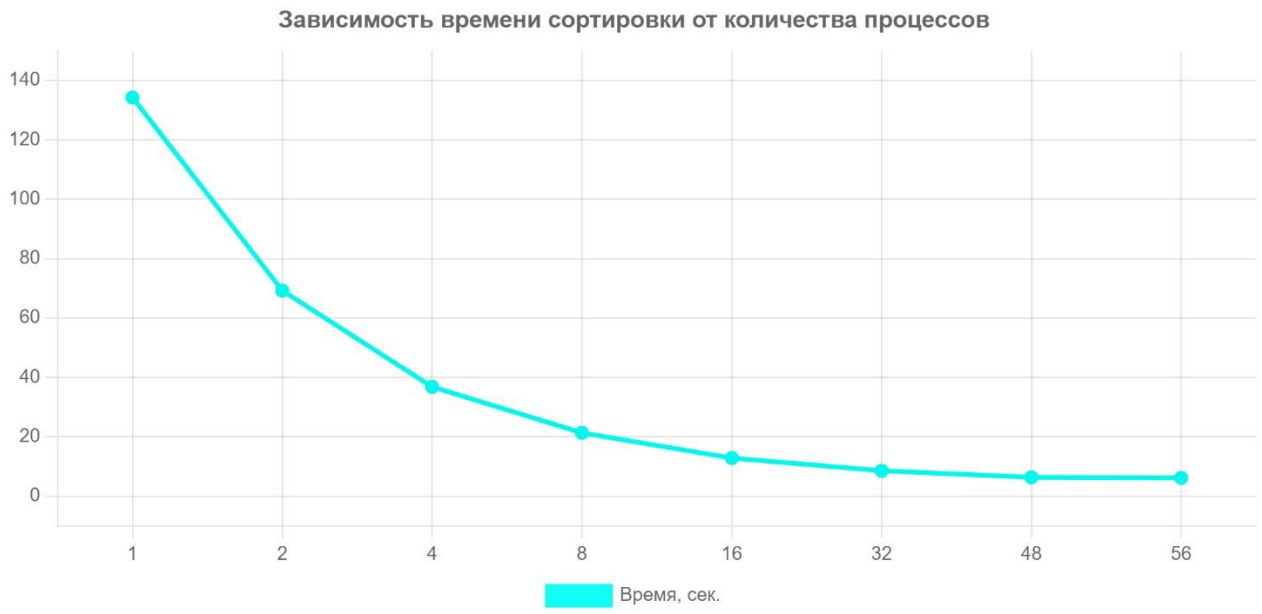
На IBM Polus иногда случаются выбросы – случаи, когда время работы значительно отличается от ожидаемого, поэтому время бралось как среднее значение нескольких запусков. Также время в целом зависит от загруженности кластера в моменте, т. е. в разные дни и время получаются разные результаты для одного и того же решения.

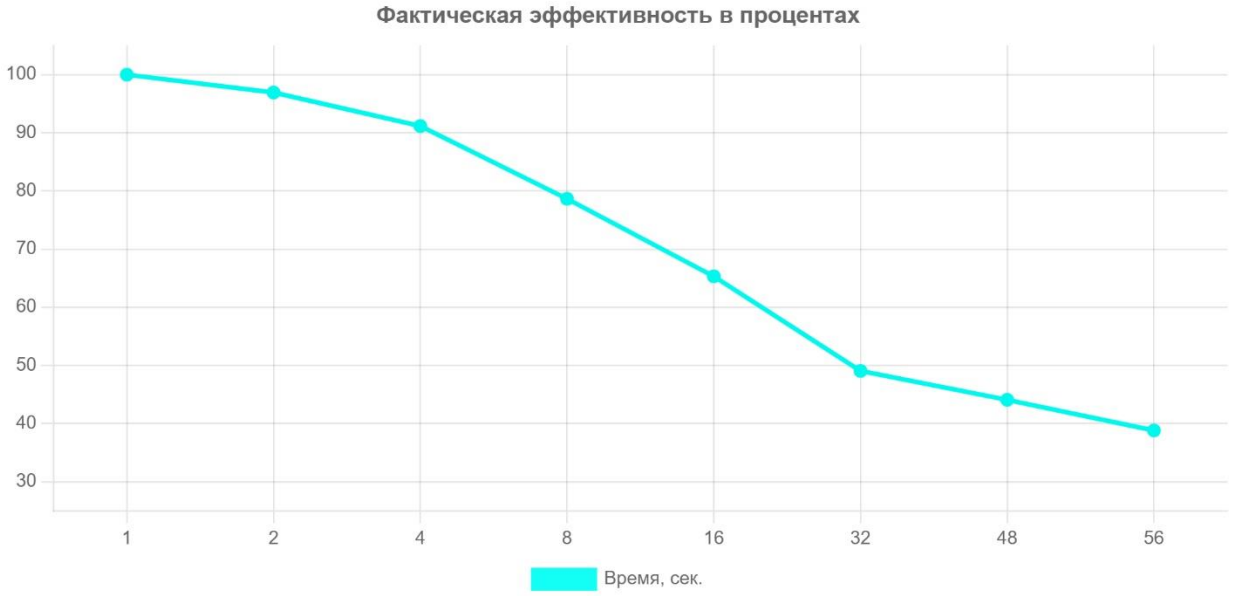
Далее в таблице приведен расчет параметров для массива размером 2^{30}:

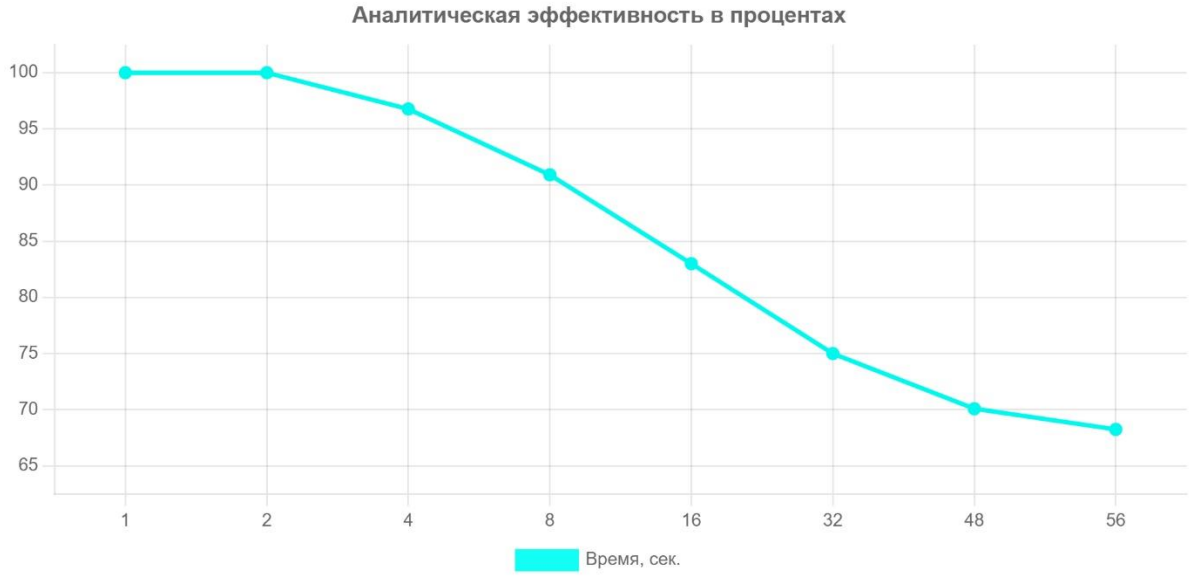
N	p	T	S	E	E_{max}	S_{max}	E / E_{max}
1073741824	1	134,2760	1,0000	100,00%	100,00%	1,0000	100,00%
	2	69,2652	1,9386	96,93%	100,00%	2,0000	96,93%
	4	36,8218	3,6466	91,17%	96,77%	3,8710	94,20%
	8	21,3372	6,2931	78,66%	90,91%	7,2727	86,53%
	16	12,8461	10,4527	65,33%	83,33%	13,3333	78,40%
	32	8,5531	15,6990	49,06%	75,00%	24,0000	65,41%
	48	6,3442	21,1651	44,09%	70,09%	33,6422	62,91%
	56	6,1760	21,7416	38,82%	68,25%	38,2174	56,89%

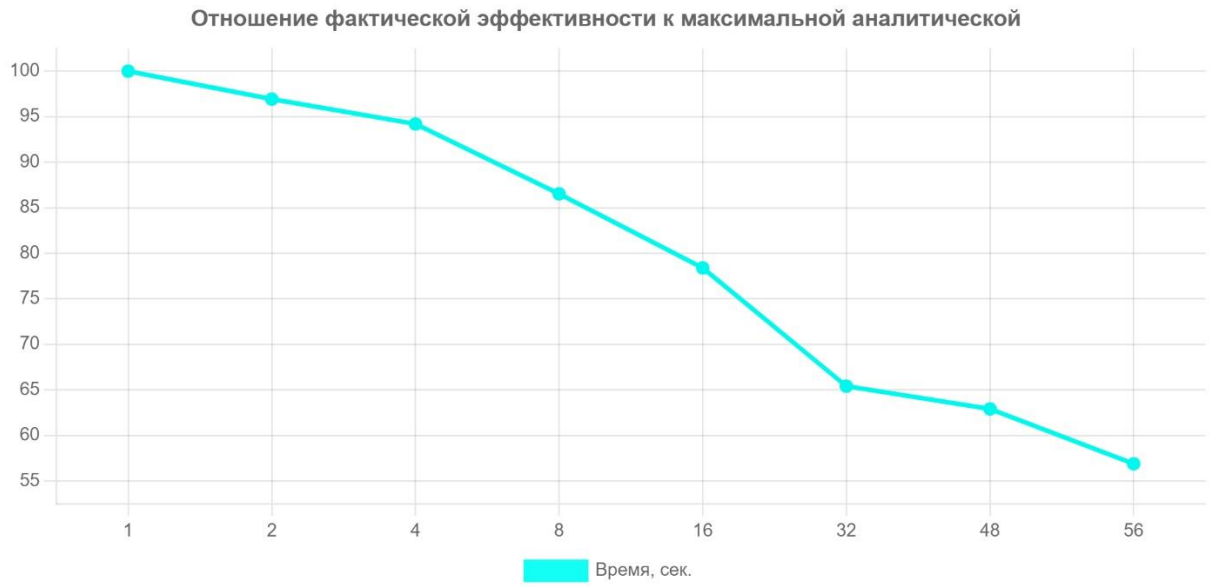
, где:

- p – количество процессов
- T – время затраченное на сортировку
- S – фактическое ускорение
- E – фактическая эффективность
- E_{max} – максимальная эффективность, посчитанная аналитически
- S_{max} – максимальное ускорение посчитанное аналитически
- E / E_{max} – отношение фактической эффективности к теоретической максимальной


Зависимость времени сортировки от количества процессов
140
120
100
80
60
40
20
0
1
2
4
8
16
32
48
56
Время, сек.


Фактическая эффективность в процентах
100
90
80
70
60
50
40
30
1
2
4
8
16
32
48
56
Время, сек.


Аналитическая эффективность в процентах
100
95
90
85
80
75
70
65
1
2
4
8
16
32
48
56
Время, сек.


Отношение фактической эффективности к максимальной аналитической
100
95
90
85
80
75
70
65
60
55
1
2
4
8
16
32
48
56
Время, сек.

Анализ полученных результатов

Тестовые прогонки и получение результатов проводились с использованием вычислительной системы IPM Polus.

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

- 32768 – 1.2680054e-08
- 262144 – 6.827037e-09
- 1048576 – 5.966187e-09
- 16776256 – 5.656068e-09
- 134217728 – 5.961602e-09
- 536870912 – 8.906402e-09
- 1073741824 – 5.954965e-09

Число тактов сортировки Бэтчера для различного количества процессов p:

- p = 2: 1 такт
- p = 4: 3 такта
- p = 6: 6 тактов
- p = 8: 6 тактов
- p =16: 10 тактов
- p = 32 – 15 тактов
- p = 48 – 21 такт

Аналитические выражения для ожидаемого времени, ускорения и эффективности сортировки:

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

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

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

Формулы взяты из работы М.В. Якобовского “Параллельные алгоритмы сортировки больших объемов данных” стр. 14 и книги “Введение в параллельные методы решения задач” стр. 154.

Приложение

- Исходный текст программы в отдельном c++ файле – ***parallel_bsort.cpp***
- ***Makefile*** для компиляции и запуска.