Параллельная сортировка Бэтчера

второе практическое задание по курсу

“Параллельные высокопроизводительные вычисления”

Михальцов Данила Алексеевич

528 группа (1 курс магистратуры)

4 декабря 2023

(с исправлениями от 18 декабря)

Постановка 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.*

Описание метода решения

Для построения сети чётно-нечётной сортировки Бэтчера использовались функции из первого практического задания. Сортировка производилась следующим образом: сначала каждый процессор сортирует свою часть массива, а затем они обмениваются между собой своими частями и производят их слияние согласно упомянутой выше сети сортировки.

Сортировка своих частей производится с помощью функции std::sort, слияние же происходит согласно описанному в лекциях алгоритму, аналогичному слиянию в обычной сортировке слиянием с учетом направления сортировки и того, что количество элементов у разных процессов строго одинаково.

Каждый процесс, когда участвует в сети сортировки, проходя по массиву компараторов (они генерируются на всех процессах), смотрит, относится ли этот компаратор к нему. Если да, то он участвует в блокирующем двустороннем обмене, порядок приема/пересылки определяется положениям номеров процессов в паре компаратора.

Массив создаётся на первом процессе (rank == 0) и раздается всем процессам с помощью MPI_Scatter. В конце массив собирается на первом процессе с помощью MPI_Gather. Проводится проверка корректности сортировки с учётом ее направления.

Для запуска в качестве первого аргумента командной строки нужно задать размер массива, который следует создать и отсортировать. Для корректной работы алгоритма сортировки требуется, чтобы у каждого процесса было одинаковое число элементов массива, поэтому число элементов массива может быть больше, чем N_theoretical:

```
int elements_per_process = (N_theoretical + n_proc - 1) / n_proc;
int N = elements_per_process * n_proc;
```

Фиктивные элементы равны максимальному или минимальному значению типа double (чтобы эти значения располагались в конце массива).

Если указать любой второй аргумент командной строки, то сортировка будет выполнена по убыванию, а не по возрастанию.

Первый процесс с помощью функции MPI_Wtime замеряет время работы программы и выводит его на стандартный поток вывода.

Описание используемой вычислительной системы

IBM Polus - параллельная вычислительная система, состоящая из 5 вычислительных узлов.

На каждом узле:

Процессоры IBM Power 8: 2

NVIDIA Tesla P100: 2

Число процессорных ядер: 20

Число потоков на ядро: 8

Оперативная память: 256 Гбайт (1024 Гбайт на узле 5)

Коммуникационная сеть: Infiniband / 100 Gb

Система хранения данных: GPFS

Операционная система: Linux Red Hat 7.5

Характеристики вычислительной системы

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

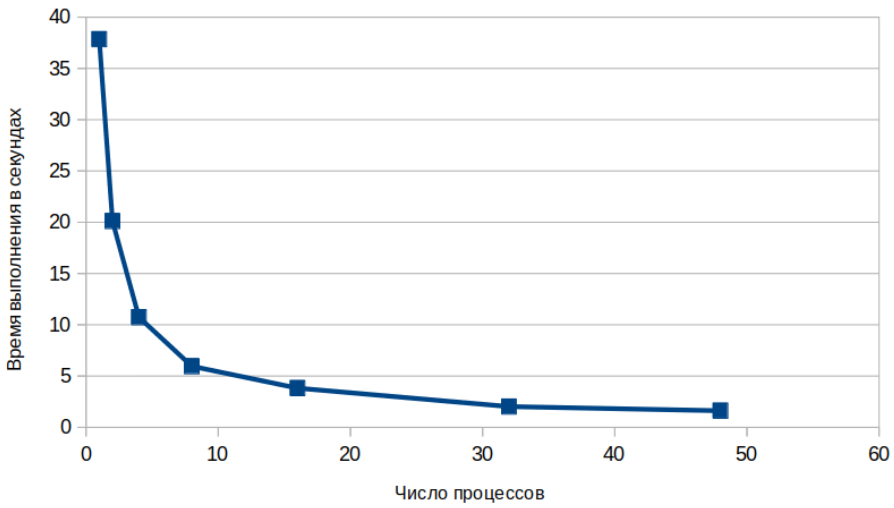
Производительность кластера (Tflop/s): 55,84 (пиковая), 40,39 (Linpack)

Полученные результаты

Результаты внесены в таблицу table.ods (файл прикреплен). Здесь продублированы получившиеся результаты:

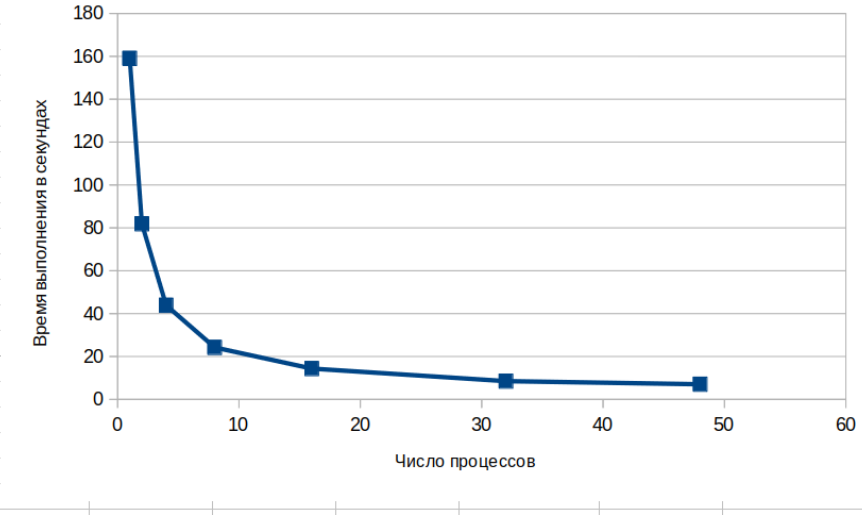
Для массива размером 1e8:

N	N_PROC	T	S = T_1/T	E=S/N_PROC	E_MAX	T1/(N*LOG(N))	Число тактов сортировки
1.00E+08	1	37.8491	1	**1**	1	1.424214E-08	—
1.00E+08	2	20.114	1.88172914	**0.9408645719**	1	1.424214E-08	1
1.00E+08	4	10.7371	3.5250766	**0.8812691509**	0.96373583	1.424214E-08	3
1.00E+08	8	5.96012	6.35039227	**0.7937990342**	0.89856443	1.424214E-08	6
1.00E+08	16	3.82003	9.90806355	**0.6192539718**	0.81581207	1.424214E-08	10
1.00E+08	32	2.02587	18.6828869	**0.5838402143**	0.72659238	1.424214E-08	15
1.00E+08	48	1.62035	23.3585954	**0.4866374034**	0.67486549	1.424214E-08	21



Для массива размером 4e8:

N	N_PROC	T	S = T_1/T	E=S/N_PROC	E_MAX	T1/(N*LOG(N))	Число тактов сортировки
4.00E+08	1	159.014	1	**1**	1	1.391178E-08	—
4.00E+08	2	81.8954	1.94167194	**0.9708359688**	1	1.391178E-08	1
4.00E+08	4	43.8765	3.62412681	**0.9060317026**	0.96618814	1.391178E-08	3
4.00E+08	8	24.2297	6.56277214	**0.8203465169**	0.9049894	1.391178E-08	6
4.00E+08	16	14.3604	11.0730899	**0.6920681179**	0.82646634	1.391178E-08	10
4.00E+08	32	8.514	18.6767677	**0.5836489899**	0.7407676	1.391178E-08	15
4.00E+08	48	7.04369	22.5753831	**0.4703204807**	0.69058051	1.391178E-08	21



Анализ полученных результатов

В таблице N – размер массива, N_PROC – число процессов (и число используемых ядер), T – затраченное время на сортировку, S – ускорение, E – эффективность, E_MAX – теоретическая максимально возможная эффективность.

Предположим, что пересылка любого элемента занимает время tau_s. Тогда приведём аналитические выражения для ожидаемого времени и эффективности сортировки:

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

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

где K(n) = T_1/(n * log2(n)).

Тогда ожидаемое ускорение $S(n, p) = p * E(n, p).$

Приведём оценку теоретической максимально возможной эффективности (E_MAX):

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

Полученная эффективность на 48 процессах составляет 72.1% и 68.1% от максимально возможной.

Примечание: при первоначальном замере без барьеров и с учетом времени на раздачу и сбор данных получить указанные результаты удалось не с первого раза. После исправления результаты получились с первого раза, но и большой очереди на суперкомпьютере не было.

~~Когда на суперкомпьютере была большая очередь job-ов, то результаты при N_PROC=32 и N_PROC=48 сильно отличались от прогнозируемых, хотя при запуске использовалась явное указание вычислительных узлов (50% процессов на 3 узле и 50% процессов на 4 узле) для N_PROC <=32.~~

~~При повторном запуске при меньшей загрузке суперкомпьютера получились приведённые здесь данные.~~

Исходный код программы

Приложен отдельным файлом.