Московский Государственный Университет

имени М.В. Ломоносова

Курс: «Параллельное программирование и суперкомпьютеры»

Параллельная сортировка Бэтчера

**Выполнил:**

**Сактаганов Нуржан (аспирант)**

**Дата подачи: 21.10.2017**

**Москва 2017**

## Постановка задачи

Дана структура (или класс)

Point{

float coord[2];

int index;

}

P[n1\*n2];

Данная структура будет использоваться для работы с регулярной сеткой. Точки данной сетки имеют координаты P[i\*n2+j].coord[0] = x(i, j), P[i\*n2+j].coord[1] = y(i,j), где i = 0, …, n1-1, j = 0, …, n2-1. Индекс определяется соотношением P[i\*n2+j].index = i\*n2+j.

На каждом процессе одинаковое количество элементов структуры Point. (Если на некоторых процессах элементов структуры Point меньше чем во всех остальных, тогда необходимо ввести фиктивные элементы, например, с отрицательным значением индекса).

Надо реализовать параллельную сортировку Бэтчера для структур Point вдоль одной из координат x (или y). То есть с начала необходимо реализовать сортировку на каждом отдельном процессе, а потом реализовать сеть слияния Бэтчера.

На выходе должны получить на каждом процессе одинаковое количество элементов структуры Point. Каждый элемент структуры Point одного процесса находиться левее по координате x ( или y) по сравнению с элементом структуры Point любого процесса с большим рангом, за исключением фиктивных элементов.

Программа должна удовлетворять следующим требованиям:

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 128.

## Метод решения

На входе имеется распределенный массив структур из length элементов на n узлах. Далее считаем, что на каждом узле запущен только один MPI процесс. Без ограничения общности можно предположить, что на каждом узле находится одинаковое количество элементов (если это не так, то нужно дополнить глобальный массив фиктивными элементами).

Для начала, на каждом узле надо построить расписание сети сортировки Бэтчера для n элементов (с целью экономии памяти, в расписание можно добавлять только компараторы вида (x, r) и (r, x), где r-ранг текущего процесса).

На самом первом такте (параллельном) сортировки, сортируем каждый локальный массив. После чего будем производить итерации comp\_exchange, то есть обмениваться локальными массивами с удаленным узлом согласно расписанию, а потом делать слияние локального массива и полученного массива. Новым локальным массивом становится половина слитого массива. Причем, если ранг удаленного узла был выше, то берем первую половину, иначе вторую половину массива.

Когда закончится выполнение расписания, на выходе получим отсортированный распределенный массив.

В качестве последовательного алгоритма сортировки берем dh\_sort. Это гибрид между пирамидальной сортировкой и сортировки слиянием [1]. Этот алгоритм можно распараллелить по аналогии с паралельной сортировкой слиянием.

## Описание вычислительной системы

Вычислительная система представляет собой комплекс IBM Blue Gene/P. В составе имеется две стойки по 1024 четырехъядерных процессоров в каждой стойке. Пиковая производтельность — 27.9 терафлопс. Размер оперативной памяти — 2 Гб на каждый процессор. Каждое ядро модели PowerPC 450 с рабочей частотой 850 МГц, с 32 битной адресацией. Производительность ядра — 3.4 Гфлопс.

Описание коммуникационной сети:

* сеть общего назначения, объединяющие все вычислительные узлы; предназначена для операций типа «точка-точка»
* вычислительный узел имеет двунаправленные связи с шестью соседями
* пропускная способность каждого соединения — 425 MB/s (5,1 GB/s для всех 12 каналов)
* латентность (ближайший сосед):
  + 32-байтный пакет: 0,1 μs
  + 256-байтный пакет: 0,8 μs

## Оценка времени выполнения параллельного алгоритма

Для упрощения модели будем считать, что:

1. все обмены не влияют друг на друга;
2. везде одинаковая латентность;
3. везде одинаквая пропускная способность;
4. между узлами есть двухканальная связь.

Тогда итоговое время выполнения будет состоять из трех основных составляющих T = T1 + T2 + T3:

* T1 — время начальной сортировки локальных массивов;
* T2 — время, затраченное на коммуникации (обмен локальным массивами);
* T3 — время, затраченное на слияние после каждого обмена.

Время начальной сортировки T1 можно обозначить как С\*ls\*log(ls), где C — некая константа, ls — размер локального массива. Константа C вычисляется эмпирически, по сути это время на одну операцию: C = T\_sort / NlogN, T\_sort — время сортировки N элементов последовательным алгоритмом.

Коммуникация происходит после каждого такта кроме последнего. Поэтому, T2 можно оценить как takts \* (T\_latency + T\_bandwithd \* ls \* sizeof(Point)).

Время слияния T3 можно оценить как C \* takts \* ls, т. к. можно сразу слить только нужную половину.

## Анализ полученных результатов

Ниже приведена таблица времен (в секундах) запусков на различном наборе сеток и процессоров.

|  |  |  |  |
| --- | --- | --- | --- |
| Сетка | 1 процессор | 128 процессоров (1 поток или ядро) | 128 процессоров (4 потока или ядра) |
| 80k x 60k | Не доступно | 140.588 | 129.499 |
| 7.5k x 10k | Не доступно | 2.04534 | 1.94928 |
| 7.5k x 5k | 18.6979 | 0.998727 | 0.972783 |
| 4k x 5k | 9.4536 | 0.517129 | 0.507898 |
| 4k x 2.5k | 4.16964 | 0.255948 | 0.254843 |

В случае для одного процессора вместо 7.5k x 10k, была взята сетка 7.5k x 8k (время выполнения — 33.2169c).

Вычислим значение C для разных сеток:

| Сетка | Время (сек) | Количество элементов | C |
| --- | --- | --- | --- |
| 7.5k x 8k | 33.2169 | 60 M | 2.14e-8 |
| 7.5k x 5k | 18.6979 | 37.5 M | 1.98e-8 |
| 4k x 5k | 9.4536 | 20 M | 1.95e-8 |
| 4k x 2.5k | 4.16964 | 10 M | 1.79e-8 |

Есть некая тенденция уменьшения числа C при уменьшении числа элементов. Возможно это разброс от запуска к запуску, или возможно связано с кэшами и локальностью данных при меньшем числе элементов. Но видно, что число С — почти константа. Для дальнейших расчетов возьмем их среднее значение — 1.965e-8.

На 128 процессорах сортировка Бэтчера занимает 28 параллельных тактов. Структруа Point занимает 16 байтов (2 float и 1 long long int).

Рассмотрим параллельные выполнения программы с 1 потоком на процессор. Посчитаем для них теоретические времена выполнения. Пропускная способность между узлами — 425Мбайт/сек. Чтобы получить сколько времени требуется на передачу одного байта, возьмем обратное значение — 0.2353e-8 сек/байт. Латентность берем как 0.8e-6 сек.

| Сетка | Количество элементов | Элементов в узле | T1 | T2 | T3 | T |
| --- | --- | --- | --- | --- | --- | --- |
| 80k x 60k | 4.8 G | 37.5 M | 18.54 | 39.53 | 20.6325 | 78.703 |
| 7.5k x 10k | 75 M | 585938 | 0.2206 | 0.6177 | 0.3224 | 1.1607 |
| 7.5k x 5k | 37.5 M | 292969 | 0.1045 | 0.3089 | 0.1612 | 0.5746 |
| 4k x 5k | 20 M | 156250 | 0.053 | 0.165 | 0.086 | 0.3037 |
| 4k x 2.5k | 10 M | 78125 | 0.025 | 0.082 | 0.043 | 0.1503 |

Посчитаем ускорение и эффективность для теоретических и фактических времен выполнения в однопоточном режиме на 128 процессорах:

| Сетка | Тпосл. | Ттеор. | Тфакт. | Sтеор. | Sфакт. | Eтеор. | Eфакт. |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7.5k x 5k | 18.6979 | 0.5746 | 0.998727 | 32.5407 | 18.7217 | 0.2542 | 0.1463 |
| 4k x 5k | 9.4536 | 0.3037 | 0.517129 | 31.1281 | 18.2809 | 0.2432 | 0.1428 |
| 4k x 2.5k | 4.16964 | 0.1503 | 0.255948 | 27.7421 | 16.291 | 0.2167 | 0.1273 |

Посчитаем процентное соотношение фактической эффективности от теоретической для разных сеток:

| Сетка | Eфакт./Етеор. |
| --- | --- |
| 7.5k x 5k | 57.55% |
| 4k x 5k | 58.72% |
| 4k x 2.5k | 58.75% |

## Список литературы

1. «Параллельные алгоритмы сортировки больших объемов данных», М.В. Якобовский, 2004г.