Параллельно-конвейерные схемы алгоритмов сортировки

Фальфушинский Владислав Владимирович, Гречко Валерий Олегович

Институт кибернетики им. В.М.Глушкова НАН Украины, пр. Глушкова, 40, Киев, Украина

Vladislav.falfushinsky@gmail.com, valeryg52@mail.ru

Аннотация. *Проанализированы параллельно-конвейерные схемы алгоритмов сортировки с интенсивной обработкой сверхбольших объёмов данных. Установлено, что к замедляющим этапам вычислений относятся распределение и сбор данных. Для улучшения алгоритмов предложено применять метод map-reduce ускорения процесса распределения и сбора сверхбольших объёмов данных.*

Ключевые слова

Параллельная обработка больших объёмов данных, HPC, HTC, Cloud-computing, хранилища данных, файловые системы

1 Введение

В 1978 году В.М. Глушков предложил принцип макроконвейерной архитектуры компьютера. Он и его ученики предложили принципы организации вычислений для компьютеров с распределенной памятью, а также подходы к оптимизации вычислений [1]. Суть принципа макроконвейерной обработки данных состоит в том, что при распределении работы вычислительных узлов (ВУ), каждому из них на очередном шаге вычислений дается задание, которое может на длительное время загрузить его работой, не требующей взаимодействия с другими ВУ. С ростом вычислительных мощностей стали использовать другие высокоуровневые системы обработки больших объёмов данных HTC (High-throughput computing) вместо традиционных HPC (High-performance computing). В технологии cloud-computing данные постоянно хранятся на серверах в сети и временно кэшируются на клиентской стороне [2]. А для схем параллельных вычислений алгоритмов характерна реализация по частям на множестве разных ВУ с последующим объединением результатов в корректный итоговый. С развитием таких алгоритмов выделились задачи:

- адаптация последовательных методов расчетов для суперкомпьютеров;
- корректное использование распределенных ресурсов при построении алгоритма.

Если с первой задачей можно работать, оценивая последовательный алгоритм, выделяя части, которые можно распараллелить, то со второй – следует проанализировать классы алгоритмов и выбрать типичные схемы расчетов для типовых задач. Классическая MIMD-характеристика параллельных алгоритмов – соизмеримость объемов расчетов (поток команд) и операций доступа к данным, которые находятся в файлах (поток данных). Для подбора типичных схем параллельных алгоритмов установим набор алгоритмов, требующих интенсивного обмена данными. Такими характеристиками наделены параллельные сортировки. Для примера далее взято несколько алгоритмов сортировки [5].

Все рассмотренные алгоритмы как внешние сортировки оперируют не произвольным, а последовательным (упорядочение строк) доступом, т. е. всегда «видим» только один элемент, а затраты на навигацию в файловой системе по сравнению с объемом оперативной памяти неоправданно велики. Это накладывает дополнительные ограничения на алгоритм и приводит к упорядочению, обычно использующему дополнительное дисковое пространство. Кроме того, доступ к данным на носителе проходит намного медленнее, чем операции с оперативной памятью. Доступ к носителю осуществляется последовательно:

- в каждый момент времени можно считать или записать только элемент, следующий за текущим.
- объём данных не дает возможности разместить их в ОЗУ.

2 Блочная сортировка (bucket sort)

В блочной карманной или корзинной сортировке (*Bucket sort*) сортируемые элементы распределены между конечным числом отдельных блоков (карманов, корзин) так, чтобы все элементы в каждом следующем блоке были всегда больше (или меньше), чем в предыдущем. Каждый блок затем сортируется отдельно либо

рекурсивно тем же методом либо другим. Затем элементы помещают обратно в массив. Для этой сортировки характерно линейное время исполнения.

Алгоритм требует знаний о природе сортируемых данных, выходящих за рамки функций "сравнить" и "поменять местами", достаточных для сортировки слиянием, сортировки пирамидой, быстрой сортировки, сортировки Шелла, сортировки вставкой.

В параллельной версии сортировки каждую группу элементов обрабатывает отдельный узел. Далее:

1) узлы считывают входящие данные из хранилища;
2) хост группирует элементы по определенному признаку;
3) хост распределяет данные между клонами;
4) клоны сортируют свою часть данных;
5) хост собирает данные с клонов.

3 Пузырьковая сортировка (bubble sort)

Сортировка простыми обменами (*bubble sort*) имеет сложность O(n^2). Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравнивают попарно и, если порядок в паре неверный, элементы меняют местами. Проходы по массиву повторяют до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, значит, массив отсортирован. При проходе алгоритма стоящий не на своём месте элемент «всплывает» до нужной позиции как пузырёк, отсюда и название алгоритма. Последовательная версия этой сортировки входит в состав практически всех параллельных алгоритмов сортировки. Описание шагов алгоритма:

6) узлы считывают входные данные из хранилища;
7) хост распределяет данные между клонами;
8) клоны сортируют свою часть данных;
9) каждый клон передает свою часть данных следующему, а тот объединяет их со своими и передает следующему узлу;
10) если количество перестановок равно количеству клонов, то алгоритм завершает работу;
11) хост собирает все данные с клонов.

4 Битоническая сортировка (bitonic sort)

В основе этой сортировки лежит операция B_n (полуочиститель, half-cleaner) над массивом, параллельно упорядочивающая элементы пар x_i и $x_{i+n/2}$. На рис. 1 полуочиститель может упорядочивать элементы пар как по возрастанию, так и по убыванию. Сортировка основана на понятии битонической последовательности и утверждении: если набор полуочистителей правильно сортирует произвольную последовательность нулей и единиц, то он корректно сортирует произвольную последовательность.

Последовательность a_0 , a_1 , …, a_{n-1} называется битонической, если она или состоит из двух монотонных частей (т. е. либо сначала возрастает, а потом убывает, либо наоборот), или получена путем циклического сдвига из такой последовательности. Так, последовательность 5, 7, 6, 4, 2, 1, 3 битоническая, поскольку получена из 1, 3, 5, 7, 6, 4, 2 путем циклического сдвига влево на два элемента.

Доказано, что если применить полуочиститель B_n к битонической последовательности a_0 , a_1 , … , a_{n-1} , то получившаяся последовательность обладает следующими свойствами:

- обе ее половины также будут битоническими.
- любой элемент первой половины будет не больше любого элемента второй половины.
- хотя бы одна из половин является монотонной.

Применив к битонической последовательности a_0, a_1, … , a_{n-1} полуочиститель B_n , получим две последовательности длиной $n/2$, каждая из которых будет битонической, а каждый элемент первой не превысит каждый элемент второй. Далее применим к каждой из получившихся половин полуочиститель $B_{n/2}$. Получим уже четыре битонические последовательности длины $n/4$. Применим к каждой из них полуочиститель $B_{n/2}$ и продолжим этот процесс до тех пор, пока не придем к $n/2$ последовательностей из двух элементов. Применив к каждой из них полуочиститель B_2, отсортируем эти последовательности. Поскольку все последовательности уже упорядочены, то, объединив их, получим отсортированную последовательность.

Итак, последовательное применение полуочистителей B_n, $B_{n/2}$, …, B_2 сортирует произвольную битоническую последовательность. Эту операцию называют битоническим слиянием и обозначают M_n.

Например, к последовательности из 8 элементов a_0, a_1, …, a_7 применим полуочиститель B_2, чтобы на соседних парах порядок сортировки был противоположен. На рис. 2 видно, что первые четыре элемента получившейся последовательности образуют битоническую последовательность. Аналогично последние четыре элемента также образуют битоническую последовательность. Поэтому каждую из этих половин можно отсортировать битоническим слиянием, однако проведем слияние таким образом, чтобы направление сортировки в половинах было противоположным. В результате обе половины образуют вместе битоническую

последовательность длины 8, и ее можно отсортировать.

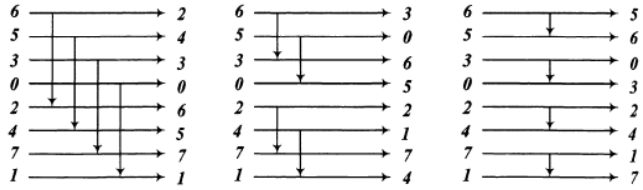

Рис. 1. Примеры полуочистителей B6, B4 и B2.

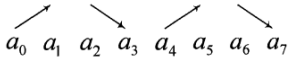

Рис. 2. Результат применения полуочистителя B2.

Битоническая сортировка последовательности из n элементов разбивается пополам и каждая из половин сортируется в своем направлении. После этого полученная битоническая последовательность сортируется битоническим слиянием.

5 Сортировка слиянием (merge sort)

Упорядочивает списки (или другие структуры данных, доступ к элементам которых можно получить только последовательно, например, потоки) в определённом порядке. Эта сортировка — пример использования принципа «разделяй и властвуй». Сначала задача разбивается на несколько подзадач меньшего размера. Затем эти задачи решаются с помощью рекурсивного вызова или непосредственно, если их размер достаточно мал. Наконец, их решения комбинируются, и получаем решение исходной задачи. Эти три этапа выглядят так.

12) сортируемый массив разбивается на две части примерно одинакового размера;
13) каждая из получившихся частей сортируется отдельно, например, тем же алгоритмом;
14) два упорядоченных массива половинного размера соединяют в один.

Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).

Нетривиальный этап – соединение двух упорядоченных массивов в один. Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть две стопки карт лежат рубашками вниз так, что в любой момент видна верхняя карта в каждой из этих стопок. Пусть также карты в каждой из стопок идут сверху вниз в неубывающем порядке. Чтобы из этих стопок сделать одну, на каждом шаге меньшую из двух верхних карт кладём (рубашкой вверх) в результирующую стопку. Когда одна из оставшихся стопок исчерпана, добавляем все оставшиеся карты второй стопки к результирующей стопке.

15) узлы считывают входящие данные из хранилища;
16) хост распределяет данные между клонами;
17) клоны сортируют свою часть данных;
18) каждый клон выделяет индексы из своих частей массива и передает хосту;
19) хост формирует индексную таблицу;
20) хост передает элементы индексной таблицы клонам;
21) каждый клон выбирает из отсортированного массива все элементы, меньшие полученного значения;
22) хост формирует отсортированный массив из полученніх результатов клонов.

6 Сортировка четно-нечетными перестановками (odd-even sort)

Для каждой итерации алгоритма операции сравнения-обмена для всех пар элементов независимы и выполняются одновременно. Рассмотрим случай, когда число процессоров равно числу элементов, т.е. $p=n$ - число процессоров (сортируемых элементов). Предположим, что вычислительная система имеет топологию кольца. Пусть элементы a_i $(i = 1, .. , n)$, первоначально расположены на процессорах p_i $(i = 1, ... , n)$. В нечетной итерации каждый процессор с нечетным номером производит сравнение-обмен своего элемента с элементом, находящимся на процессоре-соседе справа. Аналогично в течение четной итерации каждый процессор с четным номером производит сравнение-обмен своего элемента с элементом правого соседа.

На каждой итерации алгоритма нечетные и четные процессоры выполняют шаг сравнения-обмена с их правыми соседями за время $Q(1)$. Общее количество таких итераций – n; поэтому время выполнения параллельной сортировки – $Q(n)$.

Когда число процессоров p меньше числа элементов n, то каждый из процессов получает свой блок данных n/p и сортирует его за время $Q((n/p)\cdot log(n/p))$. Затем процессоры проходят p итераций ($p/2$ и чётных, и нечётных) и делают сравнивания-разбиения: смежные процессоры передают друг другу свои данные, а внутренне их сортируют (на каждой паре процессоров получаем одинаковые массивы). Затем удвоенный массив делится на 2 части; левый процессор обрабатывает далее только левую часть (с меньшими значениями данных), а правый – только правую (с большими значениями данных). Получаем отсортированный массив после p итераций, выполняя в каждой такие шаги:

23) узлы считывают входные данные из хранилища;
24) хост распределяет данные между клонами;
25) клоны сортируют свою часть данных;
26) клоны обмениваются частями, при этом объединяя четные и нечетные части;
27) если количество перестановок равно количеству клонов, то алгоритм завершен;
28) хост формирует отсортированный массив.

1. Реализация алгоритмов для улучшения результатов

После запуска нескольких видов тестов выявлено, что параллельные алгоритмы с интенсивной обработкой больших объёмов данных можно представить как трехэтапные: (1) распределение данных; (2) обработка; (3) сбор данных. Установлено, что обработка 8Гб данных сейсморазведки занимает 38 часов, из них 3 часа занимает распределение и сбор данных. Если оптимизировать операции распределения и сбора данных, получим прирост производительности алгоритма. Поэтому следует предоставить быстрый доступ каждого клона к его части данных, что целесообразнее сделать за счет использования технологии map-reduce.[6]

На шаге map предварительно обрабатывают входные данные. Для этого один хост-компьютер, получив входные данные задачи, разделяет их на части и передает клонам для предварительной обработки. На шаге reduce происходит свёртка предварительно обработанных данных и, получив ответы от клонов, хост формирует результат — решение искомой задачи.

Преимущество map-reduce заключается в возможности распределено производить операции предварительной обработки и свертки. Операции предварительной обработки исполняются независимо, их можно распараллелить (хотя на практике это ограничено источником входных данных и/или количеством используемых процессоров). Аналогично множество клонов могут осуществлять свертку; для этого необходимо только, чтобы все результаты предварительной обработки с одним конкретным значением ключа обрабатывал один клон в каждый момент времени. Хотя этот процесс может быть менее эффективным по сравнению с более последовательными алгоритмами, map-reduce можно применить к большим объемам данных, обрабатываемых большим количеством клонов. Параллелизм способствует восстановлению после частичных сбоев клонов; если на клоне, выполняющем предварительную обработку или свертку, возникает сбой, то его работу можно передать другому клону (при условии, что доступны входные данные для проводимой операции).

В таблице 1 показано время выполнения разных сортировок. Для демонстрации сортировок взят файл размером 10 Гб (98000000000 строк).

Таблица 1. Сравнение параллельных алгоритмов сортировки.

Количество процессоров	Виды и время выполнения сортировок, сек				
	bitonic	bubble	Bucket	merge	Oddeven
2	204	372	220	708	2060
4	167	312	170	635	1330
8	164	241	140	629	1060
16	153	240	137	610	720
32	147	240	132	584	710
64	140	232	132	560	702
128	135	225	128	530	685

Заключение

Проанализированы параллельно-конвейерные схемы алгоритмов сортировки с интенсивной обработкой сверхбольших объёмов данных. Установлено, что к замедляющим этапам вычислений относятся распределение и сбор данных. Для улучшения работы алгоритмов предложено применять метод map-reduce ускорения распределения и сбора сверхбольших объёмов данных. Для лучшего результата использовано распределенное хранилище, обеспечивающее классическую MIMD-характеристику параллельных алгоритмов сортировки – соизмеримость объемов расчетов (поток команд) и операций доступа к файлам данных (поток данных), в частности параллельных потоков данных. Все результаты получены на кластерном комплексе ИНПАРКОМ-256, созданном Институтом кибернетики им. В.М. Глушкова НАН Украины совместно с НПО «Електронмаш». Все алгоритмы сортировки собраны в одну библиотеку LibPsort и доступны как грид-сервис.

Работы проведены в 2010-2011 гг. по проекту *№69-87* «Интеллектуализация информационных технологий кластерных вычислений в грид-среде» согласно Государственной целевой научно-технической программы «Внедрение и использование грид-технологий на 2009-2013 годы».

Список литературы

[1] Капитонова Ю.В., Летичевский А.А.: Математическая теория проектирования вычислительных систем. – М.: Наука. Гл. ред.физ.-мат.лит., 1988. – 296 с.
[2] Carl Hewitt: ORGs for Scalable, Robust, Privacy-Friendly Client Cloud Computing. *http://www.computer.org/portal/web/csdl/doi/10.1109/MIC.2008.107*, pp. 96-99, September/October 2008 (vol. 12 no. 5).
[3] В.В. Фальфушинский: Использование средств распараллеливания для блочных алгоритмов. *Кибернетика и вычислительная техника, 2011. – 124 с.*
[4] Donald Knuth: The Art of Computer Programming, Volume 3: Sorting and searching, *1983.*
[5] Ralf Lammel: Google's MapReduce Programming Model—Revisited. *Data Programmability Team Microsoft Corp. Redmond, WA, USA*