ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 4(18), 2013, с. 127–142

УДК 004.046

Е. О. Тютляева

Интеграция алгоритма параллельной сортировки Бэтчера и активной системы хранения данных

Аннотация. В статье описан разработанный алгоритм сортировки больших объемов данных при помощи модифицированной версии алгоритма параллельной сортировки Бэтчера. Принципиальной новизной полученного решения является интеграция распространенного и доказавшего свою эффективность алгоритма параллельной сортировки Бэтчера и концепции системы активного хранения на базе библиотеки шаблонных классов TSim и кластерной файловой системы Lustre. В статье представлены результаты тестирования производительности разработанного алгоритма на реальной научной задаче обработки данных сейсмической разведки. Полученные результаты демонстрируют линейное ускорение на задаче, обрабатывающей большой (более 100 Гб) массив данных.

Ключевые слова и фразы: параллельная сортировка, сортировка Бэтчера, обработка больших массивов данных, активное хранилище, распределенная обработка данных.

Введение

Сортировка является одной из базовых операций при обработке данных, которая используется в самом широком спектре задач, включая обработку коммерческих, сейсмических, космических и прочих данных. Часто сортировка является просто вспомогательной операцией для упорядочивания данных, упрощения последующих алгебраических действий над данными и т.п.

В классическом учебнике [1, стр. 21] Д. Кнута упоминается что «по оценкам производителей компьютеров в 60-х годах в среднем более четверти машинного времени тратилось на сортировку. Во многих вычислительных системах на нее уходит больше половины машинного времени. Исходя из этих статистических данных, можно заключить, что либо (i) сортировка имеет много важных применений,

Работа выполнена в рамках проекта «Методы и программные средства разработки параллельных приложений и обеспечения функционирования вычислительных комплексов и сетей нового поколения».

либо (ii) ею часто пользуются без нужды, либо (iii) применяются в основном неэффективные алгоритмы сортировки». В настоящее время, в связи с экспоненциально возросшими объемами данных, вопрос эффективной сортировки данных снова стал актуальным.

Эффективная обработка больших (в несколько десятков Гб) объемов данных на распределенных вычислительных установках требует переработки старых алгоритмов для решения задач минимизации пересылок данных между узлами, эффективного распределения данных по оперативной памяти доступных узлов и эффективного использования доступных вычислительных ресурсов. Наиболее эффективными являются архитектурно-зависимые решения, которые используют преимущества конкретной архитектуры.

В настоящее время разработкой эффективных распределенных алгоритмов и реализаций сортировки занимается значительное количество коммерческих и исследовательских команд. К примеру, на сайте [2] можно найти результаты производительности алгоритмов сортировки для ряда ведущих центров данных. При этом используются различные критерии оценки эффективности, например, «количество данных, которое может быть отсортировано за минуту», «время, затрачиваемое на сортировку 1 миллиона записей» и т.п.

В данной работе был проведен обзор наиболее популярных методов и алгоритмов параллельной сортировки больших объемов данных. На основании проведенного обзора был разработан новый, эффективный алгоритм сортировки интегрированный с системой активного хранения данных, позволяющей приближать вычисления к местам хранения данных [3].

1. Аналитический обзор алгоритмов распределенной сортировки

Рассмотрим часть исследованных методов и проанализируем их с точки зрения применимости и эффективности в распределенной системе активного хранения данных.

1.1. Метод сдваивания

Наиболее простым для понимания и реализации является алгоритм сортировки массива на основе метода сдваивания, состоящий из двух этапов:

(1) На каждом из вычислительных процессоров сортируется локальный фрагмент массива данных.
(2) Отсортированные элементы упорядочиваются с помощью последовательности попарных слияний, определяемых методом сдваивания.

Более подробно этот метод описан в работе [4]. Данный метод возможно объединить с активной системой хранения данных, т.к. он допускает изначальное считывание сортируемых фрагментов всеми доступными вычислительными устройствами независимо. Тем не менее, существенным недостатком данного метода является удвоение количества данных, передаваемых между узлами на каждом шаге и удвоение занимаемой оперативной памяти узлов. Заключительное слияние выполняется всего на одном процессоре, следствием чего является простой всех остальных вычислительных процессоров и необходимость обработки всего объема данных оперативной памятью одного вычислительного узла. Таким образом, данный метод не эффективен для использования в поставленной задаче.

1.2. Терабайтовая сортировка на Hadoop

Терабайтовая сортировка [5] — это наиболее яркий представитель нового поколения подходов к сортировке больших объемов данных. В представленной работе использовался стандартный алгоритм map/reduce сортировки, адаптированный для выполнения в среде Apache Hadoop. В данном алгоритме также использовался усовершенствованный подход к разбиению ключей сортировки. Архитектура Hadoop схожа с архитектурой активной системы хранения данных и полученные результаты демонстрируют, что подобного плана архитектура может быть очень эффективно использована для сортировки больших объемов данных.

1.3. Сортировки для GPU

Для получения наиболее эффективных алгоритмов все большую популярность получает подход разработки архитектурно-зависимых алгоритмов сортировки. Например, ряд исследовательских работ [6, 7] посвящен разработке алгоритмов сортировки использующих вычислительную мощность графических ускорителей. В качестве алгоритма используется специально разработанный Fine Sample Sort, который позволяет эффективно использовать технологические особенности при работе с графическими ускорителями.

1.4. Сравнение параллельных алгоритмов сортировки для различных архитектур

Большой интерес в рамках данной темы представляет технический отчет [8] Техасского A&M университета, посвященный сравнению параллельных алгоритмов сортировки для различных архитектур. Исследователи провели сравнительное тестирование трех параллельных алгоритмов сортировки:

- поразрядной сортировки [9],
- битонической сортировка [10],
- сортировки образцами [11].

Предложенные алгоритмы были реализованы для трех различных архитектур:

- MasPar MP1202,
- nCUBE 2,
- Sequent Balance.

Результаты тестирования убедительно показывают, что для различных архитектур и различных размеров данных эффективно применение различных алгоритмов.

Так, для MasPar MP1202 для небольших объемов данных продемонстрировала свою эффективность битоническая сортировка, а для больших объемов - поразрядная. В то же время на nCUBE 2 лучшие результаты показала сортировка образцами.

К сожалению, архитектуры представленных вычислительных машин сильно отличаются от архитектуры системы активного хранения, и представленные алгоритмы в ходе аналитического исследования не подошли для эффективной реализации.

1.5. Обменная сортировка со слиянием Бэтчера

Данный алгоритм выбран для реализации в системе активного хранения данных. Это рекурсивный алгоритм, на первом этапе которого выполняется считывание и локальная сортировка данных, а затем производится объединение сформированных упорядоченных массивов с помощью (n, m)-сети слияния фрагментов, описанной в [1]. Среди отличительных особенностей данного алгоритма следует отметить:

- сортировку данных, расположенных на каждом узле, можно проводить независимо;

- сеть сортировки, определяющая порядок слияний, также не допускает пересечений;
- к концу процесса сортировки не происходит увеличения объема передаваемых от узла к узлу данных, равно как не происходит и уменьшения числа выполняющих полезную работу процессоров.

Перечисленные свойства алгоритма позволяют эффективно адаптировать его к распределенной вычислительной среде выполнения и интегрировать с активной концепцией хранения данных. В следующем разделе будет более подробно описана данная интеграция и окончательная реализация алгоритма в системе активного хранения.

2. Разработка и реализация алгоритма распределенной сортировки больших объемов данных

Выбранный алгоритм обменной сортировки со слиянием Бэтчера был доработан в соответствии с требованиями системы активного хранения данных.

Далее будут описаны основные технические решения, разработанные для максимального приближения вычислений к местам физического расположения фрагментов сортируемых данных на каждом этапе сортировки.

При реализации в системе активного хранения будет производиться сортировка больших подмассивов данных. Для достижения максимальной эффективности необходимо, чтобы каждый считываемый подмассив не превышал по объему $1/2$ размера оперативной памяти вычислительного узла, при необходимости сортировать большие объемы следует их разбивать на меньшие подмассивы при считывании данных или при загрузке данных в ФС Lustre.

2.1. Чтение

Первым этапом является считывание каждым узлом фрагментов данных с физических устройств хранения (OST) ФС Lustre в оперативную память для последующей обработки.

Каждый узел локально считывает подмассив, физически расположенный на носителе, прикрепленном к этому узлу, и локально сортирует при помощи алгоритма быстрой сортировки.

Для считывания данных реализован класс *ReaderTask*, который наследует базовому типу задач в TSim [12]. Первый параметр конструктора данного класса содержит имя обрабатываемого файла, по которому планировщик активной системы хранения по разработанной на предыдущих этапах и зарегистрированной технологии определяет физическое расположение данных в распределенном вычислительном окружении и отсылает задачу по считыванию на соответствующий узел.

Метод *execute1()* данного класса, который автоматически вызывается планировщиком после того, как задача отправлена на данный узел, содержит описание считывания данных, соответствующее решаемой задаче, локальную сортировку считанного фрагмента и сохранение их в распределенную структуру данных для последующей сортировки.

После того, как задачи по считыванию разосланы на все используемые вычислительные узлы, можно приступать к распределению задач по выполнению попарных сортировок-слияний в соответствии с алгоритмом Бэтчера.

2.2. Структура данных

Основной технологической задачей на данном этапе является использование такой схемы хранения данных в оперативной памяти (или пространстве своппинга), при которой каждый узел будет оперировать только:

- Данными, которые были физически расположены на его носителе.
- Фрагментами, с которыми требуется выполнить слияние тех данных, за которые несет ответственность указанный узел. При выполнении слияния, согласно концепции активных хранилищ, всегда будет использоваться один фрагмент, расположенный на данном узле и один фрагмент с другого узла. После выполнения слияний, отсортированная половина массива остается в памяти текущего вычислительного узла, другая (большая или меньшая, по соответствующему порядковому номеру), вернется на другой узел в замен исходного неотсортированного фрагмента.

Такая схема позволит не только соблюсти исходное условие, заключающееся в том, что к концу процесса сортировки не происходит увеличения объема передаваемых от узла к узлу данных, но также гарантирует, что к концу процесса сортировки не произойдет увеличения объема занятой оперативной памяти каждого из вычислительных узлов, что особенно важно при оперировании большими объемами данных. Таким образом, после выполнения сортировки-слияния двух любых фрагментов данных порождается два новых фрагмента аналогичного размера, глобально отсортированных относительно друг друга, которые размещаются в оперативной памяти тех же узлов, что были расположены исходные фрагменты.

Для реализации описанной схемы хранения данных был использован сложный тип *TVal<TRef <Value> > **.

При использовании данного типа фрагменты сортируемого множества считываются и хранятся в оперативной памяти узла в специальном типе данных языка TSim — *TBuf<Value>*, где *Value* — тип сортируемых данных. Использование *TBuf* связано с тем, что у стандартных векторов C++ есть ограничение по максимальному размеру maxsize, которое может быть превышено при сортировке больших (нескольких десятков Гб) объемов данных. TBuf же базируется на указателях C и ограничен только технологическими особенностями архитектуры вычислительного узла (теоретически — не ограничен).

На фронтальный узел передается неготовая ссылка на считанный массив данных — *TVal<TRef>*. Использование ссылочного типа позволяет вычислительным узлам обращаться к данным, расположенным на любом из доступных устройств хранения. В частности, такая схема позволяет фронтальному узлу рассылать задачи по попарному слиянию тех фрагментов данных, которые не загружены в его оперативную память. Использование концепции неготовых переменных позволяет рассылать параллельные задачи (T-задачи), которые смогут выполняться одновременно в соответствии со стратегий активного планирования.

2.3. Операция упорядоченного слияния отсортированных фрагментов

Для реализации операции упорядоченного слияния отсортированных фрагментов (далее сортировки-слияния) больших фрагментов данных, доступных по указателю *TRef<Value>* реализован специальный класс *MergeTask*, который также наследует базовому типу

задач в TSim. Аналогично классу *Reader*, первый параметр конструктора данного класса содержит имя обрабатываемого файла, по которому планировщик системы активного хранения при помощи API ФС Lustre определяет физическое расположение данных и отсылает задачу по считыванию на соответствующий узел.

Метод *execute1()* данного класса обращается по ссылкам и получает в оперативную память массивы, содержащие считанные фрагменты данных. Если в данный момент задачи по считыванию еще не завершились, то задача по слиянию переходит в режим ожидания, до завершения задачи чтения. После того, как данные получены, выполняется упорядоченное слияние двух отсортированных подмассивов данных, в результате которого порождается два новых TBuf, ссылки на которые возвращаются на узел, породивший задачу слияния.

2.4. Итерации слияний

Согласно алгоритму Бэтчера, для того, чтобы все множество данных было глобально отсортировано, потребуется несколько итераций таких попарных непересекающихся сортировок слияний.

В соответствии с концепцией Т-системы все отправленные планировщику задачи выполняются параллельно и асинхронно. Тем не менее, для корректной работы алгоритма, необходимо, чтобы каждая следующая итерация начиналась тогда и только тогда, когда выполнены все слияния, необходимые на предыдущем шаге.

Для того чтобы получить такой результат был разработан механизм, который позволяет хранить два «поколения» неготовых ссылок на обрабатываемые фрагменты данных:

- результат предыдущего шага (в начале сортировки, это результат считывания и локальной сортировки фрагментов),
- результат текущей итерации (на последнем шаге сортировки это будет последовательность глобально отсортированных фрагментов).

После того, как все задания на сортировку-слияние, необходимые в текущей итерации сформированы и отправлены планировщику, шаблон программы параллельной сортировки переходит в режим ожидания неготовых переменных, которые отражают состояние выполнения задач. Когда соответствующие неготовые переменные перейдут в состояние готовности, можно переходить к следующей итерации.

Важным ограничением при использовании неготовых переменных является запрет на повторное присваивание, т.е. неготовая переменная может получить результат и перейти в стадию готовности лишь один раз за время жизни. Эту особенность следует учитывать при организации поколения неготовых ссылок: контейнер для каждого следующего поколения должен создаваться заново, недопустимо перемещение, требующее явного или неявного присваивания. В связи с этим, мы не можем использовать массив или вектор для хранения поколений, т.к. элементы этих контейнеров хранятся в непрерывной области памяти, и удаление первого поколения приведет к тому, что для последующего будет выполнено копирование в освободившуюся область памяти, что не только снижает эффективность вычисления, но и лишает нас возможности работы с перемещенными неготовыми типами, т.к. для них уже выполнено присваивание.

Решением является использование стандартного контейнера библиотеки STL: list (список), элементы которого хранятся в произвольных кусках памяти. В любом месте контейнера вставка и удаление производятся очень быстро. Доступ к произвольному элементу списка происходит медленнее, чем у вектора, но в реализуемом алгоритме мы рассматриваем список, состоящий только из двух поколений, доступ к которым можно эффективно совершать через выделенные итераторы «первый» front() и «последний» back().

В начале каждой итерации создается новый список неготовых ссылок на фрагменты, в который в ходе выполнения задач слияния записываются ссылки на необработанные фрагменты.

Затем ссылки на те фрагменты, которые не подвергались обработке в ходе текущей итерации, просто переносятся в новый список и программа сортировки переходит в режим ожидания всех задач данной итерации.

После того, как все результаты получены, предыдущее «поколение» ссылок удаляется. Основные положения описанной схемы условно представлены на листинге ниже.

```
list <TVal<TRef <Value> > * > val_list;
TVal<TRef <Value> > * val_lista =
new TVal<TRef <Value> > [NumParts];
val_list.push_back(val_lista);
for all fragments{
    ReaderTask ((val_list.front())[i]);

}
```

```
for all iteration{
    TVal<TRef <Value> > * val_listb =
    new TVal<TRef <Value> > [NumParts];
    val_list.push_back(val_listb);
     for all merges{
        MergeTask ((val_list.front())[i], (val_list.front())[j],
         (val_list.back())[i], (val_list.back())[j]);
    }
    for all unprocessed{
         val.list.back()[i] = (val_list.front())[i];
    }
   wait_for_iteration_end();
    val_list.erase(val_list.begin());
}
```

Представленный подход позволяет параллельно выполнять операции сортировки-слияния внутри каждой итерации и синхронизировать действия между итерациями, а также минимизировать занятую оперативную память.

2.5. Запись

На заключительном этапе каждый узел выполняет запись своего отсортированного фрагмента на устройство хранения. Для записи данных реализован специальный класс WriterTask, который в соответствии с концепцией активного хранения позволяет записывать результат, расположенный в оперативной памяти узла в файл, физически также расположенный на носителе, прикрепленном к данному узлу. Для выполнения записи используется метод, позволяющий при создании файлов в ФС Lustre задавать способ разбиения (striping) файла в хранилище.

Схематически основные положения описанного алгоритма обменной сортировки со слиянием Бэтчера, адаптированного к системе активного хранения данных представлены на рис. 1):

3. Тестирование производительности на задаче обработки сейсмических данных

Для тестирования производительности разработанного шаблона параллельной сортировки больших объемов данных была выбрана реальная задача сортировки сейсмических данных.

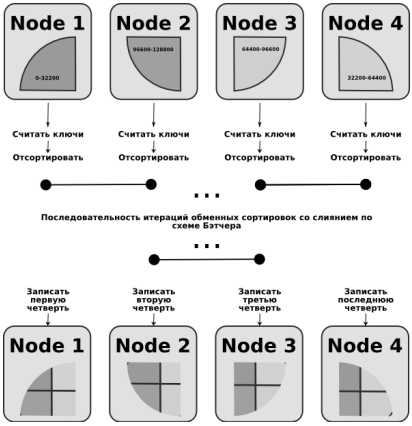


Рис. 1. Основные положения алгоритма сортировки

Для проведения базовых операций по обработке данных сейсмической разведки использовался пакет Seismic Unix [13]. В частности, для проведения последовательной сортировки сейсмических трасс по выбранным ключам был использован базовый модуль susort из этого пакета.

Тестирование проводилось на реальных данных сейсмсической разведки размером 119 Гб. Для размещения данных по вычислительным узлам кластера файл был разбит при помощи модуля suwind пакета Seismic Unix, чтобы не нарушать целостность сейсмических трасс.

При тестировании разработанного алгоритма параллельной сортировки на всех доступных 7-ми узлах вычислительного кластера данные были равномерно распределены по всем физическим носителям (OST) кластерной ФС Lustre. Технические характеристики вычислительного кластера, на котором проводилось тестирование представлены в таблице 1.

Для тестирования на одном узле использовалась последовательная сортировка, которая выполнялась при помощи стандартного модуля susort пакета Seismic Unix. Тестирование проводилось по двум сейсмическим ключам cdp и offset.

Производительность последовательного вычисления была измерена для двух случаев:

Таблица 1. Технические характеристики кластера

Место расположения	ИПС РАН
Число вычислительных узлов	7
Тип процессоров	IntelXeon E5410 2,33 ГГц, 4 ядра – 10 шт, IntelXeon X5570 2,93 ГГц, 4 ядра — 4 шт
Количество ядер	56
Количество процесоров	14
Оперативная память	288 Гб
Дисковая память узлов	29 Тбайт
Тип системной сети	Gigabit Ethernet
Конструктив узла	Full ATX

- Данные физически расположены на одном физическом устройстве, вычисление проводится на соответствующем вычислительном узле, результат записывается на то же устройство. В таблице 2 данный результат записан в графе «Последовательная сортировка».
- Данные физически распределены по всем узлам хранилища ФС Lustre, результат так же записывается на все узлы при помощи средств ФС Lustre. Алгоритм сортировки выполняется на одном узле. (Известно, что преимуществом ФС Lustre является возможность доступа с одного узла-клиента к данным, расположенным во всем хранилище. При этом используются инструменты, обеспечивающие параллельный доступ к данным). В таблице 3 данный результат записан в графе «Последовательная сортировка».

Полученные результаты производительности отражены в таблицах 2 и 3, в первой рассчитывается ускорение по сравнению с первым случаем выполнения последовательного вычисления, во второй —по сравнению с параллельной записью данных средствами ФС Lustre[1].

Результаты производительности демонстрируют сверхлинейное ускорение по сравнению с последовательной версией работающей с

[1]Указанные в таблицах результаты измерения производительности являются усредненными данными по трем запускам. Выполнение измерений проводилось в ночное время, когда на вычислительном кластере отсутствовала всякая посторонняя активность, способная оказать влияние на результаты измерений.

Таблица 2. Результаты тестирования операции сортировки (последовательная версия использует 1 OST)

Количество узлов	1 узел	7 узлов
Время	575 мин 33 сек	75 мин 11 сек
Ускорение	1	7,6

Таблица 3. Результаты тестирования операции сортировки (последовательная версия использует все OST)

Количество узлов	1 узел	7 узлов
Время	521 мин 48 сек	75 мин 11 сек
Ускорение	1	6.9

локальным устройством хранения данных для чтения и записи. Данный результат объясняется тем, что кроме распараллеливания вычислений, в параллельном алгоритме были использованы преимущества параллельного ввода-вывода предоставляемые ФС Lustre при чтении и записи результата; кроме того был использован в 7 раз больший объем оперативной памяти. Объем данных, на которых проводилось тестирование (119 Гб) превышает объем ОЗУ узла (32 Гб), что позволяет получить особенные преимущества при распараллеливании.

При сравнении с последовательной версией, использующей все доступное хранилище ФС Lustre мы получаем ускорение, близкое к линейному. Таким образом, тестирование разработанного алгоритма продемонстрировало очень высокие результаты производительности. Эффективное распараллеливание алгоритма сортировки в распределенном окружении представляет особенную сложность, связанную с неизбежно высокими накладными расходами на передачи по сети больших фрагментов сортируемых массивов данных. Вычислительная сложность алгоритма в неудачных реализациях не всегда способна достаточно эффективно окупить указанные накладные расходы. Полученные результаты показывают, что алгоритм сортировки был успешно интегрирован с планировщиком системы активного хранения данных, что позволило максимально приблизить вычисления к

местам хранения данных и эффективно использовать доступные ресурсы вычислительных узлов.

Другим важным аспектом является проверенная масштабируемость алгоритма — проведенный тест позволил отсортировать массив размером превышающий 100 Гб данных. Полученный алгоритм является очень важным компонентом библиотеки активного хранения данных, т.к. он позволит эффективно выполнять сложные операции обработки больших объемов данных. В частности, алгоритмы сортировки используются в большинстве конвейеров обработки сейсмических данных, чем и был обоснован выбор практической задачи для тестирования разработанного алгоритма.

4. Вывод

В статье представлен разработанный и практически реализованный алгоритм сортировки больших объемов данных при помощи модифицированной версии алгоритма параллельной сортировки Бэтчера. Принципиальной новизной полученного решения является интеграция распространенного и доказавшего свою эффективность алгоритма параллельной сортировки Бэтчера и концепции системы активного хранения . Алгоритм параллельной сортировки Бэтчера был модифицирован таким образом, чтобы максимально эффективно выполнять параллельную сортировку на распределенном вычислительном устройстве и обрабатывать большие (несколько десятков и сотен Гб) объемы данных, распределенные по устройствам хранения установки. При использовании разработанной схемы обработка выполняется не просто на разных вычислительных процессорах, но на разных вычислительных узлах, что соответственно позволяет размещать обрабатываемые данные на различных физических носителях, физически расположенных на разных узлах, а также затем размещать считанные в оперативной памяти различных узлов.

Таким образом, увеличение количества узлов позволяет масштабировать задачу и обрабатывать значительно большие объемы данных, представляющие проблему с размещением в оперативной памяти и пространстве свопинга при обработке на одном узле.

Практическое тестирование производительности разработанного алгоритма на задаче обработки сейсмических данных продемонстрировало линейное ускорение на задаче, обрабатывающей большой (более 100 Гб) массив данных сейсмической разведки.

Список литературы

[1] Кнут Д. Э., Козаченко Ю. В., Красиков И. В. Искусство программирования: Сортировка и поиск. Классический труд, Т. **3**. М. : Вильямс, 2000. — 824 с. ↑[], 1.5

[2] Sorting Benchmarks, http://sortbenchmark.org/, Дата доступа: 12 ноября 2013. ↑[]

[3] Тютляева Е. О., Московский А. А., Курин Е. А. *Применение концепции активных хранилищ в задачах обработки данных сейсмических наблюдений* // Труды Международной суперкомпьютерной конференции «Научный сервис в сети Интернет: поиск новых решений»/ Новороссийск, Россия — М. : Изд-во МГУ, 2012, с. 350–355. ↑[]

[4] Якобовский М.В. *Параллельные алгоритмы сортировки больших объемов данных* // Фундаментальные физико-математические проблемы и моделирование технико-технологических систем: Сб. науч. тр. : Янус-К, 2004, с. 235-249. ↑1.1

[5] O'Malley O. Terabyte Sort on Apache Hadoop/ Yahoo, 2008, http://sortbenchmark.org/YahooHadoop.pdf, Дата доступа: 12 сентября 2013. ↑1.2

[6] Munavalli S. M *Efficient Algorithms for Sorting on GPUs*, Visveswaraiah Technological University, Belgaum, 2012. — 47 p. ↑1.3

[7] Satish N., Kim Ch., Chhugani J., Nguyen A. D., Lee V. W., Kim D., Dubey P. *Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort* // Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data. SIGMOD '10 — New York, NY, USA : ACM, 2010, p. 351–362, http://doi.acm.org/10.1145/1807167.1807207. ↑1.3

[8] Amato N., Iyer R., Sundaresan Sh., Wu Ya. A Comparison of Parallel Sorting Algorithms on Different Architectures/ Texas A & M University College Station. USA, 1998, Technical Report. ↑1.4

[9] Sohn A., Kodama Yu. *Load Balanced Parallel Radix Sort* // International Conference on Supercomputing, 1998, p. 305–312. ↑1.4

[10] Bilardi G., Nicolau A. *Adaptive Bitonic Sorting: An Optimal Parallel Algorithm for Shared-Memory Machines* // SIAM J. Comput., 1989. Vol. **18**, no. 2, p. 216-228. ↑1.4

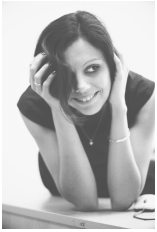
[11] Frazer W. D., McKellar A.C. *Samplesort: A Sampling Approach to Minimal Storage Tree Sorting* // J. ACM, July 1970. Vol. **17**, no. 3, p. 496–507. ↑1.4

[12] Московский А. А. *Реализация библиотеки для параллельных вычислений на основе шаблонных классов языка C++* // Труды Международной научной конференции «Параллельные вычислительные технологии (ПаВТ'2007)». — Челябинск : изд. ЮУрГУ, 2007, с. 256. ↑2.1

[13] Cohen J. K., Stockwell J. J. W. CWP/SU: Seismic Unix Release 43R3: a free package for seismic research and processing, 2012, http://www.cwp.mines.edu/cwpcodes/, Дата доступа: 12 сентября 2013. ↑3

Рекомендовал к публикации *чл.-корр. РАН С. М. Абрамов*

Об авторе:



Екатерина Олеговна Тютляева

Инженер в ИПС им. А.К. Айламазяна РАН с 2006 г., образование высшее, квалификация: математик, системный программист, автор более 11 печатных научных работ. Область научных интересов: системы хранения данных, высокопроизводительные вычисления, обработка больших объемов данных.

e-mail: ordi@xgl.pereslavl.ru

Образец ссылки на эту публикацию:

Е. О. Тютляева. *Интеграция алгоритма параллельной сортировки Бэтчера и активной системы хранения данных* // Программные системы: теория и приложения : электрон. научн. журн. 2013. Т. 4, № 4(18), с. 127–142.

URL: http://psta.psiras.ru/read/psta2013_4_127-142.pdf

E. O. Tyutlyaeva. *Using parallel Batcher sort in Active Storage System.*

Abstract. This paper describes a modified parallel Batcher sort algorithm for big data processing. The main novelty of implemented sort algorithm is to integrate effective parallel batcher sort and Active Storage concept. We use Active Storage based on Lustre File System and TSim C++ template library for parallelization. This paper presents experimental testing results for scientific processing real seismic data. Presented results indicate that described algorithm can reach linear acceleration on sorting big data sets (More then 100 Gb). (*in Russian*)

Key Words and Phrases: Parallel sort, Batcher sort, Big data processing, Active Storage, Distributed data processing.