Вестник Сыктывкарского университета.
Сер.1. Вып.1(19). 2014

УДК 681.3

ПАРАЛЛЕЛЬНЫЕ АЛГОРИТМЫ СОРТИРОВКИ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ ТЕХНОЛОГИИ MPI

В. В. Миронов, В. А. Мартынов

В работе решается задача оптимизации стандартных сортировок с помощью технологии MPI. Используется модель приема–передачи сообщений, являющаяся одной из самых популярных моделей программирования в MPI. Для проведения численных экспериментов написано приложение на языке программирования C++. В работе приведены результаты численного моделирования сортировки данных в параллельном режиме.
Ключевые слова: *параллельные алгоритмы, сортировка, эффективность.*

1. Объектом исследования данной работы являются традиционные алгоритмы сортировки числовых массивов. Сортировка данных — практически важная и теоретически интересная задача, так как сортировка больших объемов данных — неотъемлимая процедура при обработке информации в базах данных, прикладных задачах вычислительной математики и пр. Что же касается разработки алгоритмов, то здесь процесс сортировки также очень важен, так как сортировка является существенной частью многих алгоритмов. В связи с бурным ростом объемов обрабатываемой информации и появлением новых параллельных архитектур компьютеров и сред программирования возникает потребность в разработке алгоритмов сортировки данных, адаптированных под эти архитектуры и среды программирования.

Рассмотрим наиболее известные алгоритмы сортировки с целью их дальнейшей оптимизации для использования на параллельных ЭВМ [1].

Сортировка пузырьком (*bubblesort*), или сортировка простыми обменами, — простой алгоритм сортировки, состоящий из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно, и, если порядок в паре неверный,

выполняется перестановка элементов. Проходы по массиву повторяются $N-1$ раз (N — длина массива) или до тех пор, пока на очередном проходе не окажется, что элементы массива уже отсортированы. Сложность алгоритма сортировки — $O(N^2)$.

Данный алгоритм является учебным и практически не применяется вне учебной литературы. Вместо него применяются более эффективные алгоритмы сортировки. В то же время метод сортировки обменами лежит в основе некоторых более совершенных алгоритмов, таких как шейкерная сортировка, пирамидальная сортировка и быстрая сортировка.

Сортировка Шелла (*Shellsort*) — алгоритм сортировки, являющийся усовершенствованным вариантом алгоритма сортировки вставками. Идея метода состоит в сравнении элементов, стоящих не только рядом, но и на некотором расстоянии друг от друга. Иными словами, это сортировка вставками с предварительными „грубыми“ проходами.

При сортировке Шелла сначала сравниваются и сортируются элементы, отстоящие друг от друга на некотором расстоянии d. После этого процедура повторяется для некоторых меньших значений d. Заканчивается процедура при $d = 1$. Эффективность сортировки Шелла в определенных случаях обеспечивается тем, что элементы „быстрее“ встают на свои места.

Среднее время работы алгоритма зависит от длины промежутков d. Существует несколько подходов для выбора d. Первоначальная последовательность Шелла — $d_1 = N/2$, $d_2 = d_1/2$,..., $d_k = 1$ дает сложность алгоритма в худшем случае $O(N^2)$. Последовательности Хиббарда ($\frac{2^i-1}{2} \leq \frac{N}{2}, i \in N$), Фибоначчи, а также значения $\frac{3^i-1}{2} \leq \frac{N}{2}, i \in N$ — дают сложность алгоритма Шелла $O(N^2)$.

Сортировка вставками — простой алгоритм сортировки. Несмотря на то что данный алгоритм уступает в эффективности более сложным алгоритмам, у него имеется ряд преимуществ:

- эффективен на небольших наборах данных; на наборах данных до нескольких десятков может оказаться лучшим;

- эффективен на наборах данных, которые уже частично отсортированы;

- это устойчивый алгоритм сортировки, то есть не изменяет порядок элементов, которые уже отсортированы;

- может сортировать список по мере его получения.

Суть алгоритма вставками заключается в следующем. На каждом шаге алгоритма выбирается один из элементов входных данных и встав-

ляется на нужную позицию в уже отсортированном списке до тех пор, пока набор входных данных не будет исчерпан. Метод выбора очередного элемента из исходного массива произволен. Обычно и с целью получения устойчивого алгоритма сортировки элементы вставляются по порядку их появления во входном массиве. Временная сложность алгоритма сортировки вставками при худшем варианте входных данных $O(N^2)$.

2. Рассмотрим теперь основные элементы технологии MPI. MPI (Message Passing Interface, интерфейс передачи сообщений) — программный интерфейс (API) для передачи информации, который позволяет обмениваться сообщениями между процессами, выполняющими одну задачу [2].

Наиболее распространенной технологией программирования параллельных компьютеров с распределенной памятью в настоящее время является MPI. Основной способ взаимодействия параллельных процессов в таких системах — передача сообщений друг другу. Это и отражено в названии данной технологии — Message Passing Interface. Интерфейс поддерживает создание параллельных программ в стиле MIMD, что подразумевает объединение процессов с различными исходными текстами. Однако на практике программисты гораздо чаще используют SPMD-модель, в рамках которой для всех параллельных процессов используется один и тот же код.

Все дополнительные объекты: имена функций, константы, предопределенные типы данных и т.п., используемые в MPI, имеют префикс MPI. Описания интерфейса MPI собраны в файле mpi.h, поэтому в начале MPI-программы должна стоять директива include <mpi.h>.

Все функции передачи сообщений в MPI делятся на две группы. В одну группу входят функции, которые предназначены для взаимодействия двух процессов программы. Такие операции называются индивидуальными, или операциями типа точка-точка. Функции другой группы предполагают, что в операцию должны быть вовлечены все процессы некоторого коммуникатора. Такие операции называются коллективными.

MPI-программа — это множество параллельных взаимодействующих процессов. Все процессы порождаются один раз, образуя параллельную часть программы. В ходе выполнения MPI-программы порождение дополнительных процессов или уничтожение существующих не допускается. Каждый процесс работает в своем адресном пространстве, никаких общих переменных или данных в MPI нет. Основным способом взаимодействия между процессами является явная посылка сообщений.

Для локализации взаимодействия параллельных процессов программы можно создавать группы процессов, предоставляя им отдельную среду для общения — коммуникатор. Состав образуемых групп произволен. Группы могут полностью входить одна в другую, не пересекаться или пересекаться частично. При старте программы всегда считается, что все порожденные процессы работают в рамках всеобъемлющего коммуникатора. Этот коммуникатор существует всегда и служит для взаимодействия всех процессов MPI-программы.

Каждый процесс MPI-программы имеет уникальный атрибут — номер процесса, который является целым неотрицательным числом. С помощью этого атрибута происходит значительная часть взаимодействия процессов между собой. Ясно, что в одном и том же коммуникаторе все процессы имеют различные номера. Но поскольку процесс может одновременно входить в разные коммуникаторы, то его номер в одном коммуникаторе может отличаться от его номера в другом. Отсюда становятся понятными два основных атрибута процесса: коммуникатор и номер в коммуникаторе. Если группа содержит n процессов, то номер любого процесса в данной группе лежит в пределах от 0 до $n - 1$.

Основным способом общения процессов между собой является посылка сообщений. Сообщение — это набор данных некоторого типа. Каждое сообщение имеет несколько атрибутов, в частности номер процесса-отправителя, номер процесса-получателя, идентификатор сообщения и другие. Одним из важных атрибутов сообщения является его идентификатор, или тэг. По идентификатору процесс, принимающий сообщение, например, может различить два сообщения, пришедшие к нему от одного и того же процесса. Сам идентификатор сообщения является целым неотрицательным числом, лежащим в диапазоне от 0 до 32767 [3].

3. Рассмотрим теперь программные реализации алгоритмов сортировки для их дальнейшего анализа на возможность распараллеливания [1, 4].

Код процедуры, реализующей сортировку пузырьками, представлен ниже.

```
void BubbleSort (int count, int *pArr)
int trash=0;
for (int i=0; i<count; i++)
for (int j=0; j<count-1-i; j++)
if(pArr[j]>pArr[j+1])
trash=pArr[j]; pArr[j]=pArr[j+1]; pArr[j+1]=trash; / (1)
```

Сортировка вставками реализована в следующем фрагменте программы.

```
void insertsort (T* a, int size)
T tmp;
for (int i=1, j; i<size; i++)
tmp=a[i];
for (j=i-1;j>=0 and a[j]>tmp; j–) a[j+1]=a[j]; / (2)
a[j+1]=tmp;
```

В следующем фрагменте представлена реализация сортировки Шелла.

```
void ShellSort (T a[], long size)
long inc, i,j,seq [40];
int s;
s=increment (seq,size);
while (s>=0)
inc=seq[s–];
for (i=inc;i,size;++i)
T temp=a[i];
for (j=i-inc; (j>=0) and (a[j]>temp);j-=inc) a[j+inc]=a[j]; / (3)
a[j]=temp;
```

Как видно из представленных фрагментов программ итерации основных циклов, сортировки данных являются информационно зависимыми (см. комментарии (1), (2), (3)) [2]. Таким образом, непосредственно распараллелить итерации циклов по процессорам не представляется возможным.

Для реализации идеи параллельной сортировки предлагается модификация алгоритмов по методу крупноблочного распараллеливания. Параллельный алгоритм сортировки состоит в выполнении следующих этапов:

– исходный массив длины n распределяется по p процессорам вычислительной системы по блокам размера n/p;

– каждый из процессов упорядочивает свой блок по одному из рассматриваемых алгоритмов сортировки;

– отсортированные блоки передаются в ведущий процесс, где происходит формирование выходного отсортированного массива.

Для численного моделирования был написан комплекс программ, реализующий алгоритмы сортировки по методу крупноблочного распараллеливания [2]. Результаты эксперимента (коэффициенты ускорения вычислений) приведены в таблицах: 1,2. В таблице N означает длину массива.

Таблица 1

Последовательная сортировка

N	1000	2500	7000	10000
bubblesort	1.12	3.22	8.34	12.97
insertsort	1.01	2.53	6.23	10.21
Shellsort	0.93	2.10	5.57	9.12

Таблица 2

Параллельная сортировка

N	1000	2500	7000	10000
bubblesort	0.72	3.01	6.57	9.64
insertsort	0.62	2.01	4.34	8.21
Shellsort	0.61	1.98	3.94	7.94

Вычисления из таблиц 1 и 2 проводились на параллельной ЭВМ для случая $p = 1$ и $p = 3$ процессоров соответственно. Как видно из представленных таблиц, ускорение работы параллельной программы в зависимости от метода сортировки в среднем составила более 20 %, что свидетельствует о высокой степени параллелизма разработанных алгоритмов.

Список литературы

1. **Кнут Д. Э.** Искусство программирования. Т. 3. Сортировка и поиск. М.: Вильямс, 2007. 800 с.

2. **Воеводин В. В., Воеводин В. В.** Параллельные вычисления. СПб.: БХВ-Петербург, 2002. 602 с.

3. **Антонов А. С.** Параллельное программирование с использованием технологии MPI. М.: Изд-во МГУ, 2004 . 71 с.

4. **Хьюз К., Хьюз Т.** Параллельное и распределенное программирование с использованием C++. М.: Вильямс, 2004. 345 с.

Summary

Mironov V. V., Martynov V. A. Parallel algorithms of sorting data using the MPI technology

The problem of optimization of the standard sorting through the MPI technology is considered. The model of reception and transmission of messages, which is one of the most popular programming models in MPI, is used. For numerical experiments the C++ application is written. In the work results of numerical modeling of data sorting in parallel mode are given.
Keywords: parallel algorithms, sorting, efficiency.

СыктГУ *Поступила 20.02.2014*