*УДК 004.896*

Д.С. Эйзенах. Д.К. Череповский (3 курс, каф. ИУС, СПБГПУ),

В.П. Котляров, к.т.н., проф.

ПРИЛОЖЕНИЕ ДЛЯ СОРТИРОВКИ МАССИВА ИСПОЛЬЗУЮЩЕЕ МОДЕЛЬ ОГРАНИЧЕНОЙ ПАМЯТИ

Постановка задачи:

* Необходимо обеспечить сортировку массива за минимальное время, используя разные средства распараллеливания алгоритма.
* Обеспечить сортировку элементов массива за минимальное время, используя неограниченную область памяти, для представления частей сортируемого массива.
* Обеспечить сортировку элементов массива за минимальное время, используя ограниченную область (размер L ≥ 10\*\*3 элементов – модель ограниченной памяти) для представления частей сортируемого массива.

На первом этапе работы были проверены два алгоритма сортировки – сортировка слиянием и пирамидальная сортировка.

Сортировка слиянием:

В ней мы разделяем задачу на подзадачи и решаем их рекурсивно, а затем объединяем решение подзадач в исходные задачи.

1. Разделяем сортируемый массив на две части примерно одинакового размера.
2. Рекурсивно сортируем элементы в каждой половине.
3. Два упорядоченных массива половинного размера соединяем в один.

Пример:

Разберем алгоритм сортировки слиянием на следующем примере. Имеется неупорядоченная последовательность чисел: 13, 3, 8, 1, 15, 2, 3, 7, 4. После разбиения данной последовательности, процесс сортирующего слияния (по возрастанию) будет выглядеть так:

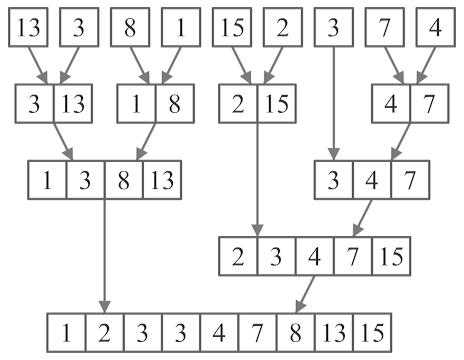


Рис. 1 Алгоритм сортировки слиянием

Пирамидальная сортировка:

1. Построение пирамиды:

Пирамида представляет собой дерево, в котором каждый узел имеет не более двух потомков, причем узел всегда больше или равен своим потомкам (таким образом, на вершине дерева всегда находится наибольший элемент).  
  
 Если в исходном массиве n элементов, то последние (n / 2) элемента становятся основанием пирамиды (эти элементы являются листьями дерева, т.е. у них нет потомков, поэтому для них вышеуказанное правило выполняется автоматически).  
  
 Удобнее всего поместить пирамиду в массив. При этом распределение индексов массива по узлам дерева будет выглядеть так (на этом рисунке все цифры - это индексы элементов массива, а ни в коем случае не значения этих элементов):

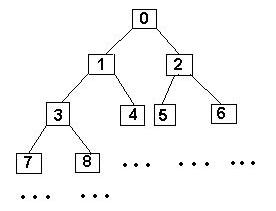


Рис. 2 Представление массива в виде дерева

Таким образом, для того, чтобы каждый узел дерева был больше своих потомков, каждый элемент массива a[i] должен быть больше или равен элементам a[2\*i + 1] и a[2\*i + 2].

1. Сортировка:

В этой части алгоритма мы перемещаем в конец массива максимальный элемент, затем исключаем его из дальнейшего процесса сортировки. Поскольку максимальный элемент всегда находится на вершине пирамиды, мы должны поменять местами элементы a[0] и a[n-1] (т.е. последний элемент). Причем элемент a[n-1] необходимо добавлять так, чтобы не нарушился порядок пирамиды (при этом пирамиду придется частично перестроить). Далее мы будем рассматривать массив только до (n-2)-го элемента.  
 На следующем шаге мы меняем местами a[0] и a[n-2] и далее рассматриваем массив только до (n-3)-го элемента. Повторяем всю эту процедуру до тех пор, пока в рассматриваемой части массива не останется один элемент.

Алгоритм работы программы:

Создается вектор и заполняется случайными числами, далее происходит деление вектора на заданный набор частей, вычисляемый по формуле h = n / p, где h – количество частей на которые делиться начальный вектор, n – размер делимого вектора, p – размер одной части, алгоритм реализован через двумерный вектор, если размер исходного массива делиться с остатком на вычисленное количество частей, происходит деление без остатка, и части складываются в вектор векторов, потом оставшаяся часть помешается в дополнительно созданный вектор в двухмерном векторе.

Пример:

10 – размер исходного массива;

3 – размер одной части;

Вычисляется количество частей для разбиения вектора, в нашем случае 10/3=3, берётся ближайшее число к размеру вектора кратное 3, происходит процедура деления, после которой получаем вектор, состоящий из 3 заполненных числами векторов, для оставшейся части создается еще один вектор, куда отправляется один оставшийся элемент.

После успешного разделения исходного массива отдельно сортируется каждая часть массива, получаем вектор векторов, состоящий из отдельно отсортированных частей.

Затем происходит проверка на пересечение частей.

Пример:

[1, 2, 4]

[3, 5, 6]

Части пересекаются т.к. последний элемент первой части больше первого элемента второй, что не допустимо в отсортированном массиве.

Проверки:

* если первый элемент первой части больше первого элемента второй части;
* если первый элемент первой части больше последнего элемента второй части;
* если последний элемент первой части больше первого элемента второй части;
* если последний элемент первой части больше последнего элемента второй части;

Проходя по двухмерному вектору происходит проверка и если условия верны, то выполняется сортировка 2 частей (на данном этапе модель ограниченной памяти выражена при помощи размера 2 частей исходного массива), берутся две части объединяются в один вектор, сортируются, затем разбиваются обратно на 2 куска и возвращаются на место, проверка и сортировка происходят до тех пор, пока все условия не буду соблюдаться.

После определенного количества итераций на выходе получаем полностью отсортированный вектор.

Для сравнения сортировок использовались входные параметры:

* n = 100000 – размер начального массива;
* p = 100 – размер одной из частей на которые делиться исходный массив;

При сравнении двух сортировок, а именно пирамидальной, и сортировки слиянием были получены следующие результаты:

|  |  |  |
| --- | --- | --- |
| сортировки | время (сек.) | количество итераций |
| Сортировка слиянием | 77.123 | 291 |
| Пирамидальная сортировка | 75.067 | 627 |

На втором этапе при сравнении трех сортировок: 2 начальные и сортировка STL –наиболее быстрая сортировка векторов, были получены следующие результаты:

|  |  |  |
| --- | --- | --- |
| Сортировки | время (сек.) | количество итераций |
| Сортировка слиянием | 78.202 | 292 |
| Пирамидальная сортировка | 74.086 | 629 |
| Сортировка STL | 5.4890 | 290 |

после чего было принято решение перейти к более быстрой сортировке, а именно STL.

После перехода на более быстрый вид сортировки, для уменьшения времени работы, было принято решение распараллелить программу, а именно циклы, отвечающие за сортировку элементов, параллельное программирование в данной работе реализовано через библиотеку Open MP.

|  |  |  |
| --- | --- | --- |
|  | время (сек.) | количество итераций |
| Сортировка STL до параллелизма | 5.48904 | 290 |
| Сортировка STL после параллелизма | 4.3167 | 290 |

В результате была достигнута скорость, намного превышающая начальную, при использовании ограниченной области (размер L 10\*\*3 элементов) для представления частей сортируемого массива, с применением параллелизма.

Список использованной литературы:

1. Роберт Седжвик "Фундаментальные алгоритмы на С++ Часть 1-4.
2. <https://software.intel.com/ru-ru/blogs/2011/11/21/openmp-c>