分段双调排序-解题报告

同济大学电子与信息工程学院 操铄

a)算法描述

0.定义

双调排序是一种并行排序算法，它同时也被用于排序网络的构建。

1.双调序列

双调序列是一个先单调递增后单调递减（或者先单调递减后单调递增）的序列。

即序列表示为

$$x_0 \leq \ldots \leq x_k \geq \ldots \geq x_{n-1} (0 \leq k \leq n)$$

2.Batcher定理

将任意一个长为`2n`的双调序列分为等长的两半`X`和`Y`，将`X`中的元素与`Y`中的元素按原顺序比较，即对于所有的`0<=i<n`，比较`a[i]`与`a[i+n]`，将较大者放入`MAX`序列，较小者放入`MIN`序列。

定理指出，经过这一操作后得到的`MAX`和`MIN`序列仍然是双调序列，并且`MAX`序列中的任意一个元素不小于`MIN`序列中的任意一个元素。

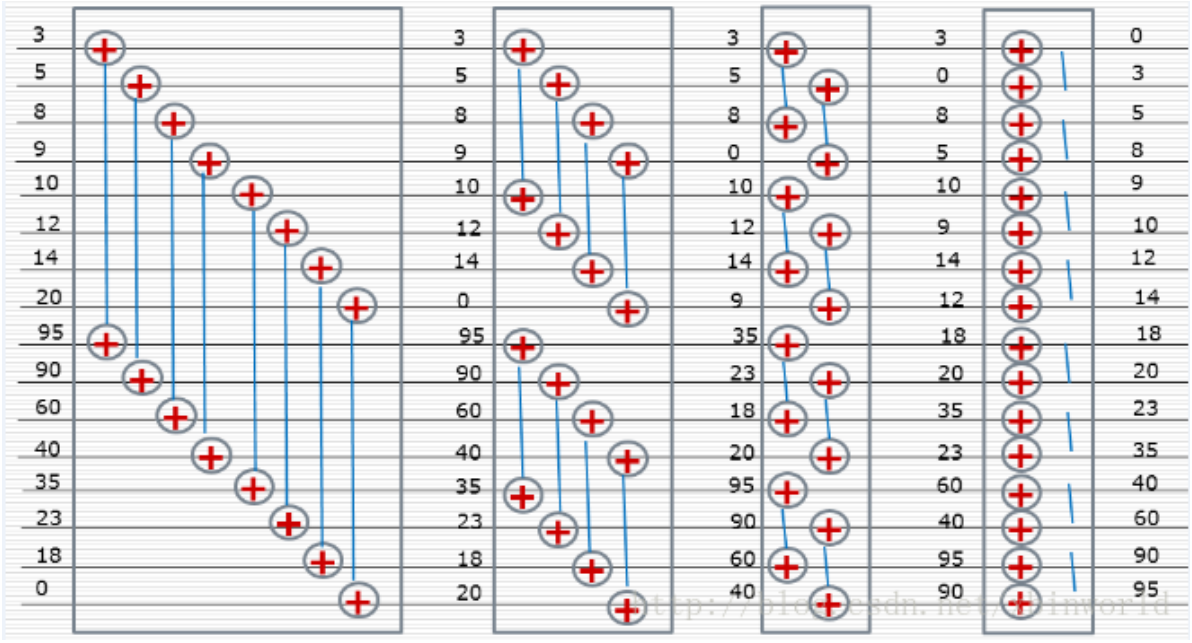
这一操作对应代码如下：

```
for (i=0;i<n;i++)
    if (get(i)>get(i+n)) exchange(i,i+n);
```

3.双调排序

假设有一个双调序列，根据Batcher定理，将该序列划分成2个双调序列，然后继续对每个双调序列递归划分，得到更短的双调序列，直到得到的子序列长度为1为止，于是得到了一个按单调递增顺序排列的输出序列。

具体方法是，把一个序列`(1...n)`对半分，假设`n=2^k`，然后`1`和`n/2+1`比较，小的放上，接下来`2`和`n/2+2`比较，小的放上，以此类推；然后看成两个`(n/2)`长度的序列，因为他们都是双调序列，所以可以重复上面的过程；总共重复`k`轮，即最后一轮已经是长度是`2`的序列比较了，就可得到最终的排序结果。



(图片来源：三十分钟理解：双调排序Bitonic Sort，适合并行计算的排序算法)

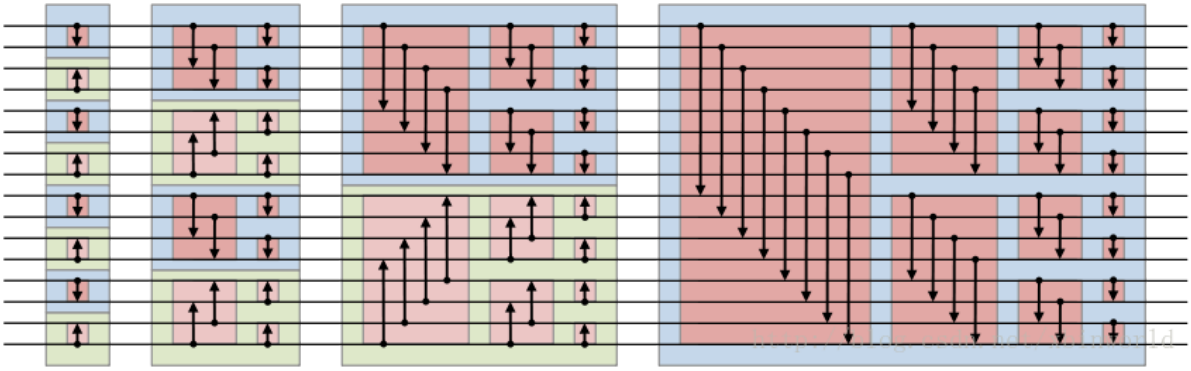
上图为这组元素进行升序双调排序的示意图。可以看到第一轮比较中，只有`20`和`0`发生了交换；第二轮比较中，`9`和`0`、`95`和`35`、`90`和`23`、`60`和`18`、`40`和`20`发生了交换，由于序列长度为`16`，递归到第四轮时得到了正确的升序结果。

4.任意序列生成双调序列

前面所述为双调序列的排序方法，现在讨论如何将任意序列变成一个双调序列。

这个过程和前面排序的思路相反，基本想法是：如果已经有两个相邻、长度为`n`的、单调性相反的序列， 就可以连接得到一个长度为`2n`的双调序列，然后对这个`2n`的序列进行一次双调排序变成有序。

以16个元素的array为例，具体步骤如下：



(图片来源：三十分钟理解：双调排序Bitonic Sort，适合并行计算的排序算法)

1. 相邻两个元素合并形成8个单调性相反的单调序列
2. 两两序列合并，形成4个双调序列，分别按相反单调性排序
3. 4个长度为4的相反单调性单调序列，相邻两个合并，生成两个长度为8的双调序列，分别按相反单调性排序
4. 2个长度为8的相反单调性单调序列，相邻两个合并，生成1个长度为16的双调序列，排序

5.非2的幂次长度序列双调排序

以上讨论的双调排序算法只适合于`n`为`2`的幂次的情况。为了适合于任意长度的数组，可以用一个定义的最大或者最小者来填充数组，让数组的大小填充到`2`的幂长度，再进行排序，最后过滤掉那些最大（最小）值即可。

本次解题中使用的就是这种方式，但是在数组长度较大的时候可能会造成比较大的空间浪费，可以在以后的学习工作中学习更好的解决方案。

b)尝试过和完成了的加分挑战

所有的加分挑战都已完成，下面分别叙述：

1.不递归

`segmentedBitonicSort`函数及其所调用的任何其他函数在程序中都没有进行形式的递归。

2.不调用函数

`segmentedBitonicSort`函数内未调用除标准库以外的其他任何函数。

3.内存高效

`segmentedBitonicSort`及其所调用的任何其他函数都没有进行任何形式的动态内存分配。

4.可并行

`segmentedBitonicSort`涉及到的所有时间复杂度O(n)以上的代码都写在for循环中，而且每个这样的for循环内部的循环顺序可以任意改变，不影响程序结果。

5.不需内存

`segmentedBitonicSort`不调用任何函数（包括C/C++标准库函数，不使用全局变量，所有局部变量都是`int`、`float`或指针类型，C++程序不使用new关键字。

6.绝对鲁棒

包含`NaN`时（例如`sqrt(-1.f)`），保证除`NaN`以外的数据正确排序，`NaN`的个数保持不变。

本代码中利用`var != var`判定是否是`NAN`，若是`NAN`则在比较的时候当作极小值。

c)可以独立运行的源代码

源文件: `BitonicSort.cpp`

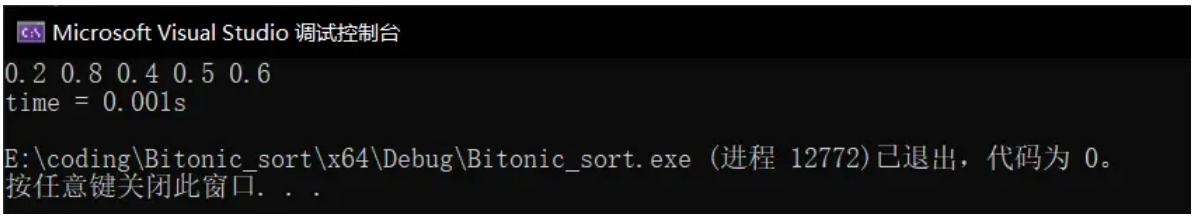
见附件。

d)测试数据

测试数据1：样例输入

```
float data[5] = { 0.8, 0.2, 0.4, 0.6, 0.5 };
int seg_id[5] = { 0,   0,   1,   1,   1 };
int seg_start[3] = { 0,2,5 };
int n = 5;
int m = 2;
```

输出：



测试数据2：加入`sqrt(-1.f)`测试鲁棒性

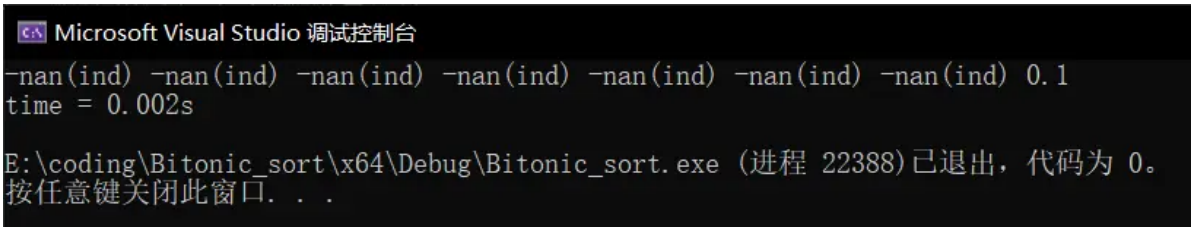
```
float data[12] = { 0.1, sqrt(-1.f), 0.5
,0.7,0.9,0.8,sqrt(-1.f),0.2,0.3,0.6,sqrt(-1.f),0.0 };
int seg_id[12] = { 0,0,0,0,1,1,1,1,1,2,2,2 };
int seg_start[4] = { 0,4,9,12 };
int n = 12;
int m = 3;
```

输出：

测试数据3：大部分为`NAN`

```
float data[8] = {
sqrt(-1.f),sqrt(-1.f),sqrt(-1.f),sqrt(-1.f),0.1,sqrt(-1.f),sqrt(-1.f),sqrt(-1
.f) };
int seg_id[8] = { 0,0,0,0,1,1,1,1 };
int seg_start[3] = { 0,4,8 };
int n = 8;
int m = 2;
```

输出：



e)性能分析

具有`2^n`长度数据的双调排序的时间复杂度为`O(n(logn)^2)`。

当数组长度`n`为任意数字时需要补齐到`2`的幂次方，这个对于任意`n`的新排序网络可以嵌入原始的对于`2^k`的双调排序网络。因此，它仍有`log(n)*(log(n)+1)/2`层，每层最多比较`n/2`次，结果仍然是一个复杂度为`O(nlog(n)^2)`的比较器，跟原始的双调排序网络无区别。

此外由于已经具备了可并行性，引入`openmp`等并行框架可以提升执行效率，在此由于时间原因未加入。

f)测试的起始和完成时间以及实际使用的时间

起始时间：`2022-9-9 15:40`

完成时间：`2022-9-9 23:50`

期间除去吃饭、洗澡等事务实际使用的时间约为8小时，花费了约5小时进行算法学习、资料查找、加分挑战实现以及调试等，约3小时进行解题报告文档撰写。

参考文献

- 三十分钟理解：双调排序Bitonic Sort，适合并行计算的排序算法

  主要参考了其中内容进行了算法理解和描述。

- Wiki pedia: Bitonic sorter

  主要参考了其中的算法描述和并行结构写法。

- Bitonic Sort

  主要参考其中对于算法并行化改进的描述以及并行结构写法。

- 分段双调排序算法 - 黄俊钦

  是曾经做过这项测试的同学的解题博客，在基本完成代码编写后参考了其中的文档结构，并补充了合法性检查以及数组边界设置。

- 各类有关并行计算和位运算的资料若干