**Вступление.**   
**Цель работы**

* Нужно реализовать наиболее быстрый и производительный алгоритм для сортировки BAM файла на ограниченной модели памяти с использованием суперкомпьютера.
* Для сравнения были выбраны два алгоритма сортировки - пирамидальная и сортировка слиянием.

**Алгоритм сортировки на ограниченной модели памяти для большого объема данных**

Алгоритм первой версии программы заключается в следующем:

1. пользователь вводит размер выборки для генерации исходного массива и размер подмассивов, на которые потом будет делиться выборка заданной длины.
2. После генерации чисел, массив разбивается на подмассивы заданной длинны.
3. Каждая часть отдельно сортируется выбранным алгоритмом сортировки.
4. Далее части берутся попарно и проверяются на «непересекаемость» (overlapping).
5. Если части не пересекаются, то рассматриваются следующие, если пересекаются, то два куска массива объединяются, сортируются, «разбиваются» на части прежних размеров и вставляются на прежние места в общем массиве. Так продолжается до тех пор, пока ни одна часть не будет пересекаться с другой.

Для сортировки частей был выбран алгоритм std::sort, т.к. пирамидальная и сортировка слиянием работали очень медленно.

|  |  |
| --- | --- |
| Пирамидальная сортировка | 10.202 сек. |
| Сортировка слиянием | 12.086 сек. |
| Сортировка STL | 5.489 сек. |

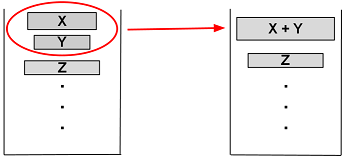
Тест реализован при размере сортируемого массива 100000 и размере одной части при делении 100;

**Timsort**

Однако, полученные результаты были недостаточно быстрыми, поэтому нужно было использовать другой алгоритм. Был выбран и модифицирован алгоритм Timsort. В алгоритме сначала выбирается размер части от 32 до 64. В оригинале для того чтобы сортировать каждый подмассив используется сортировка вставками, в модификации была произведена замена на более эффективный std::sort.

Далее происходит объединение подмассивов для получения результирующего, полностью упорядоченного массива. Алгоритм объединения следующий:

1. Создается пустой стек пар <индекс начала подмассива>-<размер подмассива>. Берётся первый упорядоченный подмассив;
2. В стек добавляется пара данных <индекс начала>-<размер> для текущего подмассива;
3. Определяется, нужно ли выполнять процедуру слияния текущего подмассива с предыдущими. Для этого проверяется выполнение двух правил (пусть X, Y и Z — размеры трёх верхних в стеке подмассивов):   
   X > Y + Z  
   Y > Z
4. Если одно из правил нарушается — массив Y сливается с меньшим из массивов X и Z. Повторяется до выполнения обоих правил или полного упорядочивания данных;
5. Если еще остались не рассмотренные подмассивы — берётся следующий и переходим к пункту 2. Иначе — конец;



Цель этой процедуры — сохранение баланса. То есть слияние подмассивов одинаковой длины.

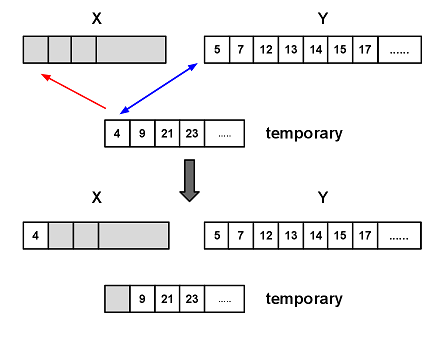
**Пример: (говорить необязательно)**

Изменения будут выглядеть как на картинке справа, а значит, размеры подмассивов в стеке эффективны для дальнейшей сортировки слиянием. В идеальном случае: есть подмассивы размера 128, 64, 32, 16, 8, 4, 2, 2. В этом случае никакие слияния не выполнятся, пока не встретятся 2 последних подмассива, после чего будут выполнены 7 идеально сбалансированных слияний.

**Модификация процедуры слияния подмассивов (Galloping Mode)**

Она был реализован в программе и ее алгоритм следующий:

1. Начинается процедура слияния, как было показано выше.
2. На каждой операции копирования элемента из временного или большего подмассива в результирующий запоминается, из какого именно подмассива был элемент.
3. Если уже некоторое количество элементов (в данной реализации алгоритма это число равно 7) было взято из одного и того же массива — предполагается, что и дальше нам придётся брать данные из него. Чтобы подтвердить эту идею, алгоритм переходит в режим «галопа», то есть перемещается по массиву-претенденту на поставку следующей большой порции данных бинарным поиском (массив упорядочен) текущего элемента из второго соединяемого массива.
4. В момент, когда данные из текущего массива-поставщика больше не подходят (или был достигнут конец массива), данные копируются целиком.



**Режим галопа на примере: говорить необязательно**

Исходные массивы:

A = {1, 2, 3,..., 9999, 10000}

B = { 20000, 20001, ...., 29999, 30000}

Первые 7 итераций сравниваются числа 1, 2, 3, 4, 5, 6 и 7 из массива A с числом 20000, так как 20000 больше — элементы массива A копируются в результирующий. Начиная со следующей итерации алгоритм переходит в режим «галопа»: сравнивает с числом 20000 последовательно элементы 8, 10, 14, 22, 38, n+2^i, …, 10000 массива A. (~log2 N сравнений). После того как конец массива A достигнут и известно, что он весь меньше B, нужные данные из массива A копируются в результирующий.

**Openmp**

Для уменьшения времени работы алгоритма сортировки, было принято решение распараллелить программу, используя библиотеки Omp, а именно распараллелить циклы for(…){…}, при распараллеливании каждая итерация цикла выполняется в определенном потоке, и на определенном ядре, таким образом получается что несколько итераций выполняется не последовательно а параллельно, что дает свои плюсы в скорости работы программы.

В дальнейшем данный алгоритм сортировки будет использован для выравнивания ридов BAM-файла по координате.

**Доп. материал**

**Сортировка слиянием.**Для решения задачи сортировки эти три этапа выглядят так:  
1. Сортируемый массив разбивается на две части примерно одинакового размера;  
2. Каждая из получившихся частей сортируется отдельно, например — тем же самым алгоритмом;  
3. Два упорядоченных массива половинного размера соединяются в один.  
1.1. — 2.1. Рекурсивное разбиение задачи на меньшие происходит до тех пор, пока размер массива не достигнет единицы (любой массив длины 1 можно считать упорядоченным).  
3.1. Соединение двух упорядоченных массивов в один. Основную идею слияния двух отсортированных массивов можно объяснить на следующем примере. Пусть мы имеем два уже отсортированных по неубыванию подмассива. Тогда: 3.2. Слияние двух подмассивов в третий результирующий массив. На каждом шаге мы берём меньший из двух первых элементов подмассивов и записываем его в результирующий массив. Счётчики номеров элементов результирующего массива и подмассива, из которого был взят элемент, увеличиваем на 1.

3.3. «Прицепление» остатка. Когда один из подмассивов закончился, мы добавляем все оставшиеся элементы второго подмассива в результирующий массив.  
Худшее время: O(n log n)  
Лучшее время: O(n log n)  
Среднее время: O(n log n)  
Затраты памяти: O(n) вспомогательных

**Сортировка пирамидальная.**

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):

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].

2. 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.