

NOVOSORT

A Multi-threaded Sort/Merge for BAM files.

V1.04

Contents

A Multi-threaded Sort/Merge for BAM files.....	1
Introduction.....	1
Benefits.....	2
Command Line.....	2
Options.....	2
@PG & @RG Records.....	4
@SQ Records.....	4
Input File(s) Preparation.....	4
Duplicate Removal.....	5
Paired End Reads.....	5
Single End Reads.....	6
Secondary Alignments.....	6
Libraries.....	6
Mark Duplicates Run Log.....	7
Optical Duplicates.....	7
Mark Duplicates Use Cases.....	7
1. Single input Bam file.....	7
2. Multiple BAM files.....	8
2. Multiple BAM files with separate merge.....	8
Picard Differences.....	8
Novosort Performance.....	8
Memory Usage.....	9

Introduction

Novosort is a fast multi-threaded sort for BAM files. In one run it can merge, sort and index a BAM file in either location order or name order. Novosort can also add or change @RG header for a file and reorder the @SQ records.

Novosort is a two phase sort merge, the first phase sorts as many reads as possible in memory and then writes chunks of sorted records to temporary disk files. The second phase merges the sorted

chunks to produce the final sorted file. If the entire BAM file fits into memory then temporary files will not be used.

Novosort can also mark or remove duplicate reads with minimal affect on the runtime of the sort.

Benefits

1. Reduced run times from multi-threading and by combining sort & merge in one program.
2. Option to included strand as part of the sort key.
3. Picard like handling of @PG and @RG identifiers
4. Uses a stable sort/merge algorithm that will not change the order of alignments with the same sort key.
5. Option to add or replace @RG record
6. Smart handling of @SQ records means the order of @SQ does not have to be the same in files being merged.
7. Optionally create the BAM index file.
8. Option to sort by read name.
9. Option to Mark or Remove alignments of duplicate reads including structural variations.

Command Line

```
novosort options bamfile1 [bamfile2....] >outputbamfile 2>runlog.txt
```

Options

Option	Description
[--threads -c] 9	Sets the number of worker threads to be used. Default is the number of CPU cores on the server. At some point novosort will be IO limited so there's a useful upper limit somewhere around 8 to 16 threads. Note. Extra threads are created for IO functions.
[--tmpdir -t] <i>dirname</i>	Sets the folder to be used for temporary files. Defaults to /tmp or \$TMPDIR. There should be sufficient space on this disk drive for one copy of the merged bam file. The temporary folder should be on a high speed file store. Be careful of using /tmp especially if it's on the same drive as the OS.
[--ram -m] 9[G M K]	Sets the amount of memory to be used for first phase of in memory sorting. Defaults to 50% of the RAM on the server. Performance will degrade badly if memory is insufficient, see discussion below.
[--strand -s]	Includes alignment strand as part of the sort key.

NOTE. If a BAM file @HD record indicates it is coordinate sorted it will not be resorted, this could be a problem if the original sort did not include strand as part of the sort key. To ensure all BAM files are sorted

with strand as part of the key add the option --forcesort

<code>[--compression -] [0-9]</code>	Sets the compression level to be used for the final BAM output file. Defaults to 6. Refer to ZLib documentation for information on compression levels.
<code>[--tmpcompression -x] [0-9]</code>	Sets the compression level to be used for temporary files. Using compression on temporary files reduces IO time at expense of CPU time and saves space on the temp device. Defaults to 1.
<code>[--rg -r] "@RG\tID:..."</code>	Defines an @RG record to add or replace existing @RGs
<code>[--forcesort -f]</code>	Sort even if @HD indicates reads are already coordinate sorted.
<code>[--assumesorted -a]</code>	Assume input files are already sorted even if @HD doesn't show coordinate sorted. This can be used to merge already sorted files even if @HD doesn't indicate the files are sorted.
<code>[--index -i]</code>	Create a BAM index file for the final sorted output. Requires -o option. This option is ignored when name sorting.
<code>[--output -o] filename.bam</code>	Final output is written to named file rather than stdout.
<code>[--namesort -n]</code>	Sort on read name rather than alignment coordinate.
<code>--removeduplicates</code> <code>--rd</code>	Remove Alignments for duplicate reads.
<code>--markduplicates</code> <code>--md</code>	Mark Alignments of duplicate reads.
<code>--uniqueTags</code> <code>tag[,tag]</code> <code>--ut</code>	List of SAM tags identifying unique molecules. eg. --ut RX. These are added to read signature when checking for duplicates. They should have the same values on both reads of a pair.
<code>--excludeSecondaries</code> <code>--xs</code>	Secondary alignments are not subject to duplicate removal
<code>--includeSE</code> <code>--ise</code>	Single End reads are also subject to duplicate removal. Default is to pass single end reads with no duplicate check.
<code>--strandSpecific</code> <code>--ss</code>	This option makes Paired end duplicate detection strand aware so that an R1--><--R2 alignment is not a duplicate of an R2--><--R1 alignment even if mapping coordinates are identical. This is useful for sample preparation methods that preserve the strand of the fragments.
<code>--keepTags</code>	Retains signature tags (Zq & Z5) added by the input stage of the

<code>--kt</code>	Novosort mark duplicates function and usually removed during output of the sorted files. Keeping the tags on a sorted file is useful if later you will merge the sorted file with other BAM files from the same library and mark duplicates during the final merge step.
<code>--readNameRegex regex</code> <code>--rnr regex</code>	An extended regular expression for parsing Lane, Tile, X & Y coordinates from the read names. The coordinates are used for detection of optical duplicates. Default is ":[0-9]+):([0-9]+):([0-9]+):([0-9]+)". If the <i>regex</i> has only 3 fields they are interpreted as Tile, X & Y. In the <i>regex</i> , <code>^</code> Matches the beginning of the header <code>\$</code> Matches the end of the header
<code>--opticalDuplicateDistance 9</code> <code>--odd 9</code>	Sets X/Y distance for duplicate reads to be counted as optical duplicates. Set to 0 (zero) to disable optical duplicate detection. Default 100

@PG & @RG Records

When merging BAM files it's possible that multiple input files have @PG records with the same program identifier but with different command lines. In order to preserve the @PG information and to ensure uniqueness of @PG program identifiers, Novosort adds a numeric suffix to the identifiers in form ".999".

Duplicate @RG records are handled in the same way.

@SQ Records

Novosort can merge BAM files that have different @SQ records and different orders for the same @SQ records. The sort order will depend on the order that the @SQ records are first seen. Input BAM files are processed in the order they appear on the command line so effectively the first BAM file defines the order of the @SQ records.

This allows merging to be done on files that may have had different ordering of @SQ records or where, say, one BAM file included Mitochondria and another didn't.

Warning. There is no checking or validation that two @SQ with the same ID refer to the same reference sequence and have the same length. So be careful, don't merge cat & dog without making sure the chromosomes have unique names!

You can use this feature to reorder the @SQ records in a BAM file. If you have a BAM file where @SQ's are not in the correct order then to reorder..

1. Extract the BAM header with `samtools view -H my.bam >headers.sam`
2. Sort @SQ records into the desired order
3. Convert Header to a BAM file with `samtools view -Sb headers.sam >headers.bam`

4. Use Novosort headers.bam my.bam >reorderedSQ.bam

Input File(s) Preparation

SAM files need to be converted to BAM files before sorting and merging. We usually recommend doing this using samtools view with level 1 compression to reduce CPU utilisation. It can also be done by piping the Novoalign SAM report into samtools view. e.g.

```
novoalign ... options ... -o SAM | samtools view -lS - > sample1.bam
```

Once your BAM files are ready you can sort and merge with novosort. Default compression level of 6 is usually used. Here we use the -i option to build an index for the sorted BAM file.

```
novosort -t tmpdir -s -i -o sorted.bam sample*.bam
```

You can also do SAM to BAM conversion at sort time using a pipe and '-' to specify the input file. In this case we use uncompressed option on samtools view or else it will limit performance.

```
samtools view -uS input.sam | novosort -t /tmp -m 8G -s -i -o sorted.bam -
```

If you have multiple SAM files to convert, sort and merge and you want to save on disk space and IO time by not writing the unsorted BAM you can still use pipes with the Linux mkfifo command.

For example to convert, sort & merge the two SAM files a.sam and b.sam.

```
mkfifo a.bam b.bam
samtools view -Su a.sam >a.bam
samtools view -Su b.sam >b.bam
novosort -m 8G a.bam b.bam >sorted.ab.bam
rm a.bam b.bam
```

In this case a.bam and b.bam are FIFO pipes not disk files and they take no disk space and no disk IO time.

Duplicate Removal

Novosort can remove or mark alignments of duplicate reads. When duplicate reads are detected the read with the highest sum of base qualities is retained, and in case where duplicates have the same quality we take the read whose read name compares less using string comparison.

Duplicate detection during sorting is efficient and adds little to the processing time or memory requirements.

Requirements.

1. For **Paired End** reads, the **input BAM files must be grouped by readname**. Usually files straight for aligners are in a suitable format. If in doubt then reads should be name sorted before running novosort –markduplicates. The only exception to this is when merging BAM files that have been already been sorted by novosort –markduplicates –keep-tags.
2. When only one read of a pair is mapped the unmapped read should have mapping coordinates matching the mapped read. If not the unmapped read of the pair will not be marked.
3. Only suitable for single end and paired reads. Results are undefined for templates with more than two read segments.

Novosort calculates a signature for each read. For mapped reads the signature comprises the Library, Strand and Alignment location of the reads 5' most base as calculated from mapping location, CIGAR, strand and the values of any molecular barcode tags. For unmapped reads a 30bit signature is calculated from bases 6-19 of the read sequence in place of the strand and alignment location.

Paired End Reads

During the input phase for unsorted or name sorted alignments, Novosort calculates the signature of each read and adds this as a SAM tag (Z5:i:) to other segments of the template. Later, during the processing of the sorted alignments, we can determine the signature of a read and, from the Z5 tag, the signature of it's pair. Reads are then grouped according to the two signatures, strand & library and duplicates detected within a group.

If you mark duplicates on pre-sorted BAM files that are missing the Z5 tag then novosort will stop with an error message regard “**Missing Z5/ZQ Tag**”.

Proper Pairs, Structural Variations and pairs with only one read mapped.

The paired end process handles proper pairs and structural variations including cross chromosome structural variations. A duplicate read is identified as having the same pair of signatures and from the same library.

Both reads are unmapped

No duplicate detection is performed when both reads of a pair are unmapped.

Single End Reads

Mapped Reads

Single end reads are considered duplicates if they have the same read 5' mapping location.

Unmapped Reads

No duplicate detection is performed on unmapped reads.

Secondary Alignments

By default, duplicate detection treats secondary alignments in the same fashion as primary alignments, so secondaries whose signatures match another read will be considered as duplicates and can be marked or removed.

This process should work well if all secondary alignments for a multi-mapped read are reported. Any duplicates should have the same set of secondary alignments and then at every mapping location Novosort should select the same read to keep. However the process is more prone to false positives because of the higher effective read depth that the secondary alignments produce at repeats.

If only a subset of the secondary alignments were reported (eg. In Novoalign option -r ALL 5 limits reporting to 5 randomly selected alignments for each multi-mapped read) then we will likely have a different set of reads aligning at each location and hence the best read selected as the alignment to keep will vary and this could break the CC/CP chain between secondary alignments and possibly remove the primary alignment for a read while keeping some of the secondary alignments.

Note. Duplicate detection cannot reliably detect duplicate reads for multi-mapped reads when the aligner has randomly selected only one alignment location to report as the duplicates may be mapped to different locations. Other duplicate detection programs likely have the same limitation.

Duplicate detection for secondary alignments can be disabled using the command line option `--excludeSecondaries`.

Note. Multi-mapped reads usually have an alignment quality ≤ 5 and are ignored by most variant callers so this option should have no affect on variant calling.

Libraries

Novosort duplicate removal is sensitive to libraries and for duplicate detection it treats each library as a different set of reads.

The library detection works as follows:

1. @RG records are scanned for LB attributes to construct an initial RG/LB table. If the @RG records search identifies only one library then further processing of LB & RG tags is disabled, if not the following applies...
2. Novosort checks each alignments for an LB tag and if found uses this as the library.
3. If no LB tag and there was more one @RG record then Novosort checks for an RG tag and assigns the corresponding library.
4. Reads are then grouped by library for duplicate detection.

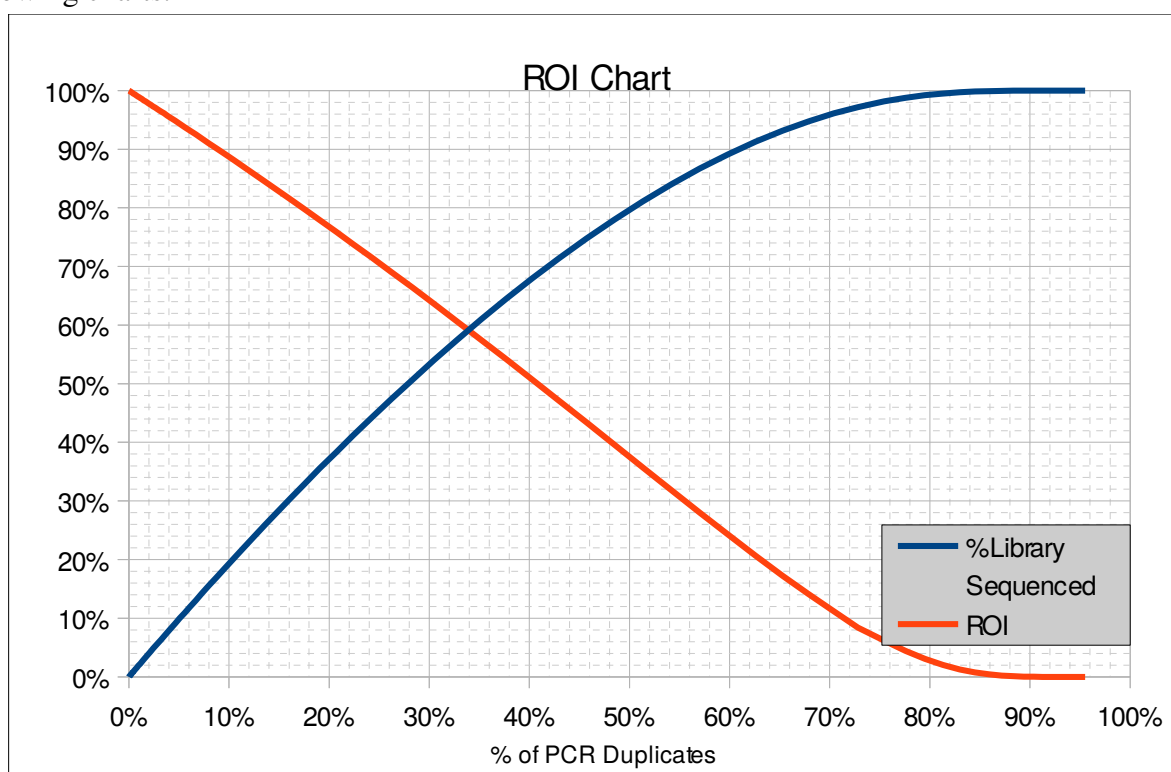
Mark Duplicates Run Log

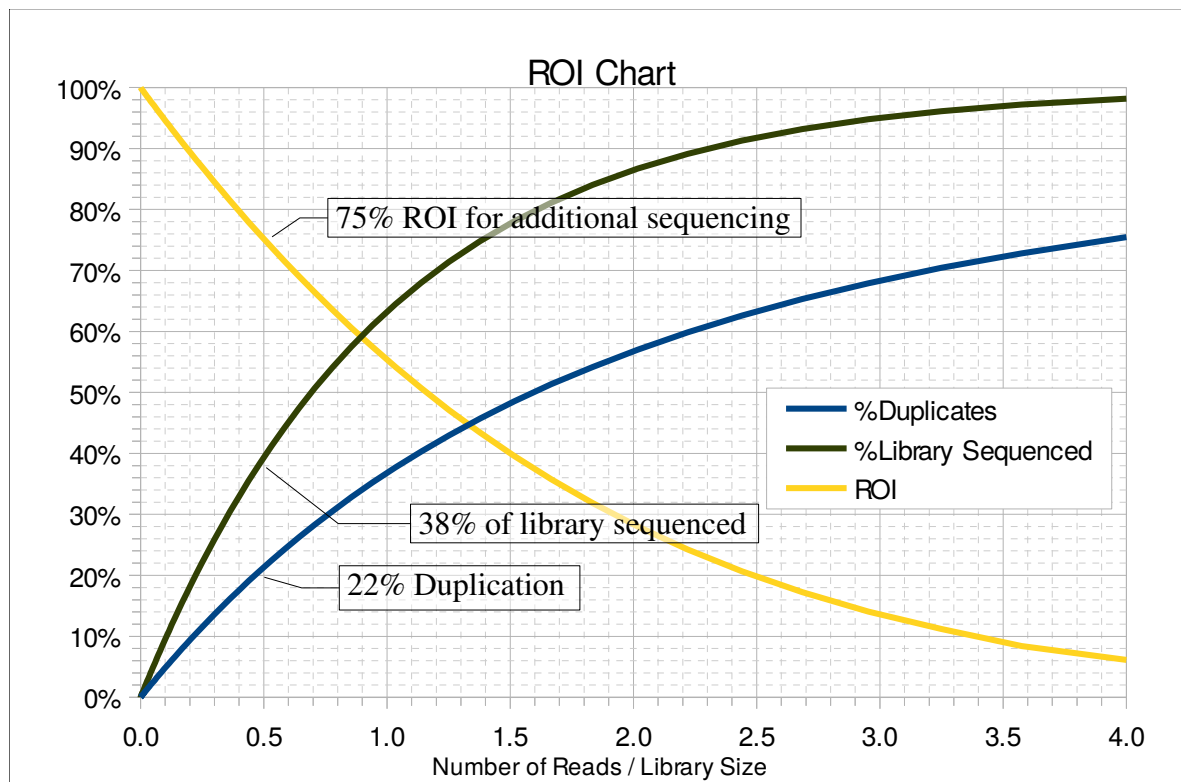
A short report of fragment and duplicate counts is produced at the end. An example is shown below.

Library size is estimated using the binomial distribution and the proper pair counts, as such it is an estimate of the number of unique fragments in the sample that would align as proper pairs.

#	Duplicate Metrics for Library Catch-162057				
#					
#	Paired End Reads				
#	Primary Alignments				
#		Unique	PCR	Optical	Estimated
#		Fragments	Duplicates	Duplicates	Library Size
#	Proper Pairs	23581259	2072418	4648	155853001
#	Improper Pairs	143542	1746	33	
#	Mapped/Unmapped	349118	1925	7	
#	Unmapped	208771	0	0	

The duplication metrics can be used to determine the benefit of additional sequencing using the following charts.





Optical Duplicates

Optical duplicate detection is performed on sets of duplicate reads. If two reads with the same signatures have the same read group, lane and X & Y coordinates that place the reads within the set optical distance limit then one of the reads is counted as an optical duplicate rather than a PCR duplicate. The process is similar to other mark duplicate programs except for the requirement that the two reads are from the same lane. This is designed to handle the case where users assign the same read group to reads from different lanes.

Two command line options affect optical duplicate detection...

- readNameRegex *regex*** An extended regular expression for parsing Lane, Tile, X & Y coordinates from the read names. The coordinates are used for detection of optical duplicates.
- Default is ":[0-9]+):([0-9]+):([0-9]+):([0-9]+)". If the *regex* has only 3 fields they are interpreted as Tile, X & Y.
- In the *regex*,
- ^ Matches the beginning of the header
 - \$ Matches the end of the header
- opticalDuplicateDistance *9*** Sets X/Y distance for duplicate reads to be counted as optical duplicates. Set to 0 (zero) to disable optical duplicate detection. Default 100

Mark Duplicates Use Cases

In all cases we need to start with unsorted or read-name sorted BAM files.

1. Single input Bam file

```
novosort --markduplicates -t . -m 8G input.bam >sorted.bam
```

2. Multiple BAM files

```
novosort --markduplicates -t . -m 8G -i -o sorted.bam input1.bam input2.bam
```

3. Multiple BAM files with separate merge

Some pipelines run per lane processes that produce sorted BAM files and then run a merge step to combine the per lane BAM files. In this situation the Z5 tags need to be added to the reads in the first sort and retained for the merge step.

```
novosort --markduplicates --keeptags -t . -m 8G input1.bam >sorted1.bam
novosort --markduplicates --keeptags -t . -m 8G input2.bam >sorted2.bam
novosort --markduplicates -m 8G -i -o merged.bam sorted1.bam sorted2.bam
```

4. Molecular Barcodes

```
novosort --markduplicates --ut RX -t . -m 8G input.bam >sorted.bam
```

If the input BAM file contains a SAM tag specifying a molecular barcode then novosort adds this to the signature of the read. The barcode tag should have the same value on both reads of a pair so if you are using both 5' and 3' tags each read should have a tag for both barcodes or a single concatenated barcode. (Do let us know if this is an issue for you)

Picard Differences

The main differences between Picard MarkDuplicates and Novosort --MarkDuplicates are:-

1. Picard does not take into account CIGAR insert operations when calculating read 5' mapping location. This leads to incorrect calls if leading/trailing inserts have not been soft clipped (eg. If MarkDuplicates is run after GATK indelRealigner)
2. When selecting the best duplicate to keep Picard sums quality of bases with qualities ≥ 15 , Novosort uses all qualities with no lower limit.
3. For mapped/unmapped pairs Picard uses the Read 5' mapping location of the mapped read to detect duplicates. Novosort also compares a portion of the sequence of the unmapped read so a duplicate needs the same mapping location and the same sequence in the unmapped read.
4. For Mapped/Unmapped pairs Picard will only flag the mapped read as a duplicate, the unmapped read of the pair is not flagged.
5. The Picard implementation of the default regular expression for extracting spot coordinates incorrectly extracts the Y coordinate for Pre-CASAVA 1.8 Illumina headers like "@HWUSI-EAS100R:6:73:941:1973#0" where the extracted Y coordinate would be 19730 rather than 1973.

Novosort Performance

To get best performance out of Novosort ...

1. The folder for temporary files should be on a high speed file store, preferable a striped RAID array or better. It will also help if it is not the same file store as the input and output BAM files.
2. If you don't specify a memory limit, Novosort will take about 50% of the memory on the server. This could cause problems if you have other programs running on the server. As Novosort does not use direct IO it is also a good idea to leave some memory for OS disk cache.
3. If you don't specify a thread limit Novosort will allocate a worker thread for each CPU core on the server.
4. The sort is fastest when the memory allocated is sufficient to fit the full uncompressed BAM file in RAM. Once you go below this, runtime increases by about 60% and is then relatively unaffected by further decreases in memory until you hit minimum memory limit described below.

Using release V1.02 or later we suggest setting -m at 8G and using 16 worker threads (-c 16).

Note. Novosort will create more threads than you specify on the -c option. The -c option specifies the number of threads dedicated to CPU intensive tasks such as sorting and compression. There are extra threads created for IO operations, one per file or merged segment. These IO threads consume only a small amount of CPU time.

Memory Usage

Novosort runs a classic two phase sort/merge algorithm.

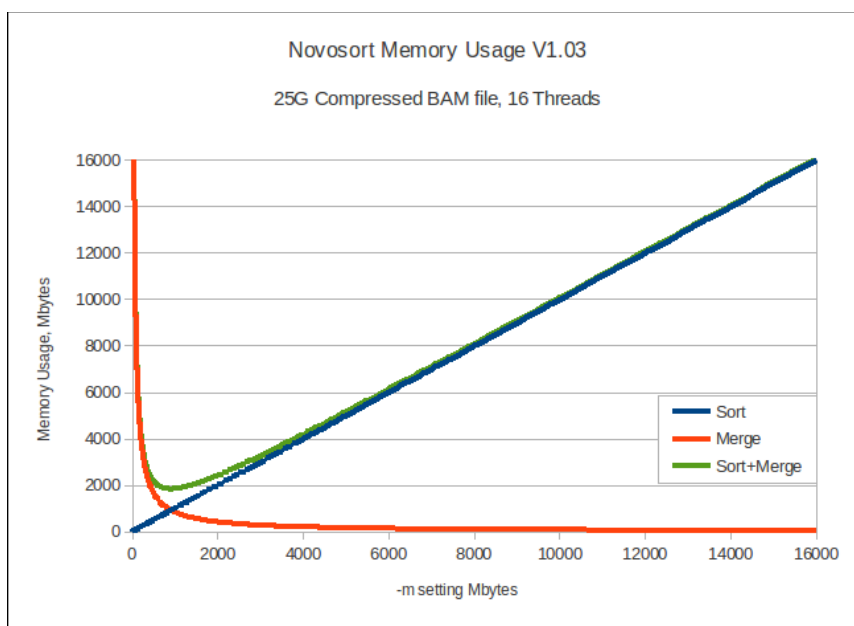
The -m option specifies the total memory to be allocated for the first phase sort buffers. BAM alignments are read into these buffers, sorted and then written to temporary files. The more memory allocated the larger the sort buffers, which leads to less sorted segments or chunks being written to the temporary files.

Actual memory for the sort phase will be about 500Mbyte higher than the -m setting.

Memory for the merge phase depends on how many sorted chunks were created in the initial sort phase. Each sorted chunk has file buffers, heap area and a thread to perform read ahead. This amounts to about 10Mbyte/chunk for V1.01 and earlier. In V1.02 we changed the memory allocator and each chunk will now use approx 0.5Mbyte.

This per chunk memory can add up. For instance if there are 1000 chunks then in V1.01 and earlier the merge phase will use 10G of memory In V1.02 it will use 500Mbyte for the merge.

As sort memory (-m) is reduced the sort produces more sorted segments (chunks) on the temporary files and so the amount of memory required for the merge goes up.



A change was made in V1.03 that reduces run time at expense of merge phase memory. The change takes advantage of the fact that at the end of the sort phase the buffers are still full of alignments so we can use these in the merge rather than using the corresponding temporary files. This means that during the merge phase we still have all the sort buffers and also the memory required for reading the temporary files.

The following table shows the -m setting that minimises merge phase memory for different input BAM sizes and 16 threads. Actual memory used in the merge phase at these settings will be approximately double the -m settings.

Compressed BAM Size	Minimum Memory (-m) with 16 threads		
	V1.01	V1.02	V1.03
200G	9G	2.3G	2.5G
100G	6G	1.6G	1.8G
50G	4G	1G	1.2G
20G	3G	600M	800M
10G	2G	400M	600M
5G	1G	300M	400M

For 16 threads (-c), minimum memory (-m setting) can be calculated using formula $M = 0.18 B^{0.5}$ where B is compressed input BAM size in Gbytes and M is minimum memory setting in Gbytes. The minimum memory is also affected by the number of threads allocated with the --threads option. The above table is based on 16 threads. As the number of threads increases the chunk size gets smaller and hence minimum memory goes up. As rule of thumb, each time the number of threads doubles, the minimum memory increases by 50%.

Notes

1. Going too far below these memory limits may increase allocated memory to the point where the job aborts.
2. Using more memory than the minimum is not a problem, so if in doubt just allocate 8G and it should handle just about any BAM file.