BIOINFORMATICS ORIGINAL PAPER

Vol. 30 no. 7 2014, pages 923–930
doi:10.1093/bioinformatics/btt656

Sequence analysis

Advance Access publication November 13, 2013

featureCounts: an efficient general purpose program for assigning sequence reads to genomic features

Yang Liao[1,2], Gordon K. Smyth[1,3] and Wei Shi[1,2,*]

[1]Bioinformatics Division, The Walter and Eliza Hall Institute of Medical Research, 1G Royal Parade, Parkville, VIC 3052, [2]Department of Computing and Information Systems and [3]Department of Mathematics and Statistics, The University of Melbourne, Parkville, VIC 3010, Australia

Associate Editor: Martin Bishop

ABSTRACT

Motivation: Next-generation sequencing technologies generate millions of short sequence reads, which are usually aligned to a reference genome. In many applications, the key information required for downstream analysis is the number of reads mapping to each genomic feature, for example to each exon or each gene. The process of counting reads is called read summarization. Read summarization is required for a great variety of genomic analyses but has so far received relatively little attention in the literature.

Results: We present *featureCounts*, a read summarization program suitable for counting reads generated from either RNA or genomic DNA sequencing experiments. *featureCounts* implements highly efficient chromosome hashing and feature blocking techniques. It is considerably faster than existing methods (by an order of magnitude for gene-level summarization) and requires far less computer memory. It works with either single or paired-end reads and provides a wide range of options appropriate for different sequencing applications.

Availability and implementation: *featureCounts* is available under GNU General Public License as part of the Subread (http://subread.sourceforge.net) or Rsubread (http://www.bioconductor.org) software packages.

Contact: shi@wehi.edu.au

Received on July 11, 2013; revised on November 6, 2013; accepted on November 7, 2013

1 INTRODUCTION

Next-generation (next-gen) sequencing technologies are revolutionizing biology by providing the ability to sequence DNA at unprecedented speed (Metzker, 2009; Schuster, 2008). The computational problem of mapping short sequence reads to a reference genome has received enormous attention in the past few years (Fonseca *et al*., 2012; Langmead *et al*., 2009; Li and Durbin, 2009; Liao *et al*., 2013; Marco-Sola *et al*., 2012), and the rapid development of fast and reliable aligners is one of the success stories of bioinformatics. Raw aligner output, however, is not usually sufficient for biological interpretation. Read mapping results have to be summarized in terms of read coverage for genomic features of interest before they can be interpreted biologically. One of the most ubiquitous operations that forms part of many next-gen analysis pipelines is to count the number of reads overlapping predetermined genomic features of interest. Depending on the next-gen application, the genomic features might be exons, genes, promotor regions, gene bodies or other genomic intervals. Read counts are required for a wide range of count-based statistical methods for differential expression or differential binding analysis (Oshlack *et al*., 2010).

Despite its importance in genomic research, the read counting problem has received little specific attention in the literature. The problem may appear superficially simple but in practice has many subtleties. Read count programs need to accommodate both DNA and RNA sequencing as well as single and paired-end reads. The reads or paired-end fragments to be counted may incorporate insertions, deletions or fusions relative to the reference genome, and these complications need to be accounted for when comparing the location of each read or fragment to each possible target genomic feature. When the number of features is large, the computational cost of read counting can be comparable with that of the read alignment step.

DNA sequence reads arise from a variety of technologies including ChIP-seq for transcription factor binding sites (Valouev *et al*., 2008), ChIP-seq for histone marks (Park, 2009) and assays that detect DNA methylation (Harris *et al*., 2010). The genomic features of interest for DNA reads can usually be specified in terms of simple genomic intervals. For example, Pal *et al*. (2013) counted reads associated with histone marks by gene promotor regions and by whole gene bodies. Ross-Innes *et al*. (2012) counted reads overlapping with intervals identified by a peak caller (Zhang *et al*., 2008).

Counting RNA-seq reads is somewhat more complex because of the need to accommodate exon splicing. One way is to count reads overlapping each annotated exon, an approach that can be used to test for alternative splicing between experimental conditions (Anders *et al*., 2012; Reyes *et al*., 2013). Another common approach is to summarize counts at the gene level, by counting all reads that overlap any exon for each gene (Anders *et al*., 2013; Bhattacharyya *et al*., 2013; Man *et al*., 2013). Gene annotation from RefSeq (Pruitt *et al*., 2012) or Ensembl (Flicek *et al*., 2012) is often used for this purpose.

Read counts provide an overall summary of the coverage for the genomic feature of interest. In particular, gene-level counts from RNA-seq provide an overall summary of the expression level of the gene but do not distinguish between isoforms when

*To whom correspondence should be addressed.

multiple transcripts are being expressed from the same gene. Reads can generally be assigned to genes with good confidence, but estimating the expression levels of individual isoforms is intrinsically more difficult because different isoforms of the gene typically have a high proportion of genomic overlap. A number of model-based methods have been developed that attempt to deconvolve the expression levels of individual transcripts for each gene from RNA-seq data, essentially by leveraging information from reads unambiguously assigned to regions where isoforms differ (Li and Dewey, 2011; Trapnell *et al*., 2010). This article concentrates on the read count problem, which is generally applicable even when the sequencing depth is not sufficient to make transcript level analysis reliable. Many statistical analysis methods have been developed to detect differential expression or differential binding on the basis of read counts (Anders and Huber, 2010; Auer and Doerge, 2011; Hardcastle and Kelly, 2010; Li *et al*., 2012; McCarthy *et al*., 2012; Wu *et al*., 2013). Recent comparisons have concluded that the read count methods perform well relative to model-based methods for the purposes of gene-level differential expression (Nookaew *et al*., 2012; Rapaport *et al*., 2013) or detection of splice variation (Anders *et al*., 2012).

Only a handful of general purpose read count software tools are currently available. The software packages *GenomicRanges* (Aboyoun *et al*., 2013) and *IRanges* (Pages *et al*., 2013), developed by the core team of the Bioconductor project (Gentleman *et al*., 2004), include functions for counting reads that overlap genomic features. The *countOverlaps* function of *IRanges* is designed for counting reads overlapping exons or other simple genomic regions, whereas the *summarizeOverlaps* function of *GenomicRanges* is designed for counting reads at the gene level. Another tool is the *htseq-count* script distributed with the HT-Seq Python framework for processing RNA-seq or DNA-seq data (Anders, 2013). All of these are popular and well-tested software tools, but all make extensive use of programming in the interpreted computer languages R or Python and none are fully optimized for efficiency and speed. *BEDTools* is a popular tool for finding overlaps between genomic features that can be used to count overlaps between reads and features (Quinlan and Hall, 2010). It is fully implemented in the compiled language C++, making it faster than the aforementioned tools. It is, however, not specifically designed for RNA-seq data, so can count reads for exons or interval features only, similar to *countOverlaps*.

This article presents a highly optimized read count program called *featureCounts*. *featureCounts* can be used to quantify reads generated from either RNA or DNA sequencing technologies in terms of any type of genomic feature. It implements chromosome hashing, feature blocking and other strategies to assign reads to features with high efficiency. It supports multithreading, which provides further speed improvements on large data problems. It is available either as a Unix command or as a function in the R package *Rsubread*. In either case, all the core functionality is written in the C programming language. The R function is a wrapper for the compiled C code that provides the convenience of the R programming environment without sacrificing any of the efficiency of the C implementation.

2 DATA FORMATS AND INPUTS

2.1 Input data

The data input to *featureCounts* consists of (i) one or more files of aligned reads in either Sequence Alignment/Map (SAM) or Binary Alignment/Map (BAM) format (Li *et al*., 2009) and (ii) a list of genomic features in either general feature format (GFF) (Wellcome Trust Sanger Institute, 2013) or simplified annotation format (SAF) (Shi and Liao, 2013b). The read input format (SAM or BAM) is automatically detected and so does not need to be specified by the user. Both the read alignment and the feature annotation should correspond to the same reference genome, which is a set of reference sequences representing chromosomes or contigs. For each read, the SAM or BAM file gives the name of the reference chromosome or contig to which the read mapped, the start position of the read on the chromosome or contig and the so-called Concise Idiosyncratic Gapped Alignment Report (CIGAR) string giving the detailed alignment information including insertions and deletions and so on relative to the start position.

The genomic features can be specified in either GFF or SAF format. The SAF format is the simpler and includes only five required columns for each feature: feature identifier, chromosome name, start position, end position and strand. These five columns provide the minimal sufficient information for read quantification purposes. In either format, the feature identifiers are assumed to be unique, in accordance with commonly used Gene Transfer Format (GTF) refinement of GFF (Brent Lab, 2013).

The number of reference sequences may be small or large depending on the application. For well-established genomes, the number of reference sequences is equal or close to the number of chromosomes. The number of reference sequences can be, however, much larger for genomes with incomplete or low-quality assemblies because each contig becomes a reference sequence. RNA-seq reads are sometimes aligned to the transcriptome instead of to the genome. In this case, there may be hundreds of thousands of transcripts and each transcript becomes a reference sequence.

featureCounts supports strand-specific read counting if strand-specific information is provided. Read mapping results usually include mapping quality scores for mapped reads. Users can optionally specify a minimum mapping quality score that the assigned reads must satisfy.

2.2 Single and paired-end reads

Reads may be paired or unpaired. If paired reads are used, then each pair of reads defines a DNA or RNA fragment bookended by the two reads. In this case, *featureCounts* will count fragments rather than reads. *featureCounts* automatically sorts reads by name if paired reads are not in consecutive positions in the SAM or BAM file.

2.3 Features and meta-features

Each feature is an interval (range of positions) on one of the reference sequences. We also define a meta-feature to be a set of features representing a biological construct of interest. For example, features often correspond to exons and meta-features to genes. Features sharing the same feature identifier in the GFF

or SAF annotation are taken to belong to the same meta-feature. *featureCounts* can summarize reads at either the feature or meta-feature levels.

3 ALGORITHM

3.1 Overlap of reads with features

featureCounts performs precise read assignment by comparing mapping location of every base in the read or fragment with the genomic region spanned by each feature. It takes account of any gaps (insertions, deletions, exon–exon junctions or fusions) that are found in the read. It calls a hit if any overlap (1 bp or more) is found between the read or fragment and a feature.

A hit is called for a meta-feature if the read or fragment overlaps any component feature of the meta-feature.

3.2 Multiple overlaps

A multi-overlap read or fragment is one that overlaps more than one feature, or more than one meta-feature when summarizing at the meta-feature level. *featureCounts* provides users with the option to either exclude multi-overlap reads or to count them for each feature that is overlapped. The decision whether to count these reads is often determined by the experiment type. We recommend that reads or fragments overlapping more than one gene are not counted for RNA-seq experiments because any single fragment must originate from only one of the target genes but the identity of the true target gene cannot be confidently determined. On the other hand, we recommend that multi-overlap reads or fragments are counted for most ChIP-seq experiments because epigenetic modifications inferred from these reads may regulate the biological functions of all their overlapping genes (Pal *et al*., 2013).

Note that, when counting at the meta-feature level, reads that overlap multiple features of the same meta-feature are always counted exactly once for that meta-feature, provided there is no overlap with any other meta-feature. For example, an exon-spanning read will be counted only once for the corresponding gene even if it overlaps with more than one exon.

3.3 Chromosome hashing

The first step of the *featureCounts* algorithm is to generate a hash table for the reference sequence names. This allows the reference sequence names found in the SAM files and GFF annotation to be matched quickly. This is particularly useful when there is a large number of reference sequences. After matching reads and features by reference sequence, subsequent analysis can proceed for each reference sequence separately.

3.4 Genome bins and feature blocks

After hashing the reference sequence names, the features in each reference sequence are sorted by their start positions (leftmost base positions). A two-level hierarchy is then created for each reference sequence. First, the reference sequence is divided into non-overlapping 128 kb bins and features are assigned to bins according to their start positions. Within each bin, equal numbers of consecutive features are grouped into blocks (Fig. 1). The number of blocks in a bin is the square root of number of features in that bin (rounded up to the next whole number). This ensures that the number of features in a block is nearly equal to the number of blocks in a bin, an optimal setting for a hierarchical search.

The use of a hierarchical data structure (features within blocks within bins) is a key component of the *featureCounts* algorithm. It facilitates rapid read assignment by quickly narrowing down the genomic region that could contain features overlapping with the query read. The query read is compared first with genomics bins, then with feature blocks within any overlapping bins and then with features in any overlapping blocks. Instead of using multiple levels of bins (Kent *et al*., 2002; Quinlan and Hall, 2010), the algorithm uses only one level of bins in combination with the feature blocks. Finally, the algorithm decides how to assign the read according to which level of summarization is being performed (feature level or meta-feature level) and whether the read is allowed to overlap with more than one target at that level.

4 IMPLEMENTATION

The `featureCounts` command in the *Subread* package for Unix is written entirely in the C programming language. The memory footprint is minimized by holding in memory only the

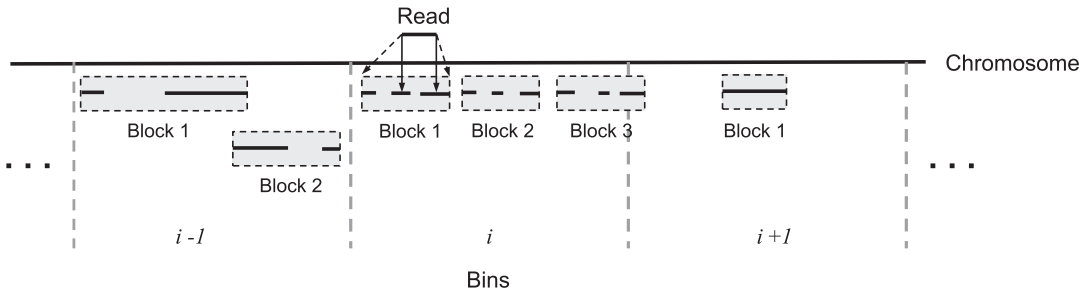

Fig. 1. Genomic bins and feature blocks. Each chromosome is divided into 128 kb bins. Features (solid lines under the chromosome) are assigned to bins according to their start positions and grouped into blocks (gray boxes) within each bin. Query reads are compared with genomic bins, then with blocks (dashed arrows) and finally with features (solid arrows). The query read in the figure overlaps with two features in the first block of bin *i*

feature annotation data required at each stage of the computation. The C code supports multithreading, and the user can specify the number of threads to be used. One thread is the default.

The R function `featureCounts` in the *Rsubread* package for R is a wrapper for the same compiled C code as for the Unix command line. The R function provides the convenience of the R programming environment without sacrificing any of the efficiency of the C implementation. It produces a data object in R that can be input directly into R-based statistical analysis software such as *edgeR* (Robinson *et al*., 2010) or *limma* (Law *et al*., 2013) that are designed to analyze next-gen read counts.

5 PERFORMANCE ON RNA-SEQ DATA

5.1 Data and annotation

First we compare the performance of *featureCounts* with existing software tools for counting RNA-seq reads at the gene level. As an example case study, we use RNA-seq data that were generated as part of the SEQC (SEquencing Quality Control) project, the third stage of the MicroArray Quality Control (MAQC) project (Shi *et al*., 2006). These data consist of 6.8 million pairs of 101 bp reads generated by sequencing a sample of Universal Human Reference RNA on an Illumina HiSeq 2000.

The SEQC RNA-seq dataset was aligned to the human genome GRCh37 using the *Subjunc* aligner included in the *Subread* package (Liao and Shi, 2013; Liao *et al*., 2013; Shi and Liao, 2013a). We used *Subjunc* for this analysis because it explicitly identifies exon–exon junctions and outputs the mapping location of every base of every read including those that span multiple exons. This allowed us to examine rigorously the ability of the read count programs to count reads spanning multiple exons as well as reads falling within exons.

Genes and exons were defined as in the NCBI human RefSeq annotation build 37.2. This included 25 702 genes and 225 071 exons.

Counts were summarized at the gene level. That is, exons were defined to be features, genes were defined as meta-features and quantification was at the meta-feature level. As this is RNA-seq data, reads or fragments that overlapped multiple genes should be excluded from the counts.

5.2 Comparative performance when counting reads

To demonstrate *featureCounts* on single-end reads, the first evaluation uses only the first read from each read pair. Table 1 compares the performance of *featureCounts* to that of the *summarizeOverlaps* function of the *GenomicRanges* package and to the *htseq-count* script. *featureCounts* and *summarizeOverlaps* yielded identical counts for every gene (Table 1, column 2).

htseq-count counted slightly fewer reads than *featureCounts* and *summarizeOverlaps*. We had a close look at the summarization results for each read given by *htseq-count* and *featureCounts* and found that only a small number of reads were assigned to different genes by the two methods (Fig. 2a). By comparing the features regions with the regions these reads were mapped to, we identified the reason causing this discrepancy. *htseq-count* takes the right-most base position of each feature as an open position and excludes it from read summarization, whereas *featureCounts* and *summarizeOverlaps* take it as a closed position and includes it in their summarizations. The GFF specification states that the start and end positions of features are inclusive (Wellcome Trust Sanger Institute, 2013), so the interpretation of *featureCounts* and *summarizeOverlaps* appears to be correct. GFF is the only annotation format supported by *htseq-count*. We modified the annotation file provided to *htseq-count* by adding one to the right-most position of each exon to let *htseq-count* include these positions. After this modification, *htseq-count* yielded identical counts to *featureCounts* and *summarizeOverlaps*.

Here and all subsequent comparisons, the software tools were tested on a HP Blade supercomputer with 64 AMD Opteron

Table 1. Performance results on the SEQC RNA-seq data

Method	Number of reads	Number of fragments	Time (min)	Memory (MB)
featureCounts	4 385 354	4 796 948	1.0	16
SummarizeOverlaps (whole genome at once)	4 385 354	3 942 439	12.1	3400
SummarizeOverlaps (by chromosome)	4 385 354	3 942 439	41.7	661
htseq-count	4 385 207	4 769 913	22.7	101

Note: Results are given for genewise counts of either single-end reads or paired-end fragments. *featureCounts* yields the same read counts as *summarizeOverlaps* but is much faster and memory efficient. *summarizeOverlaps* counts fewer fragments because it excludes read pairs with only one end successfully mapped. *htseq-count* counts slightly fewer reads or fragments than featureCounts because it interprets GFF annotation differently and calls more ambiguously assigned fragments. The table gives the total number of reads counted when using single-end reads and the total number of fragments counted when using paired-end reads. Running time and memory usage are for fragment summarization. *featureCounts* was set to exclude reads or fragments overlapping multiple genes. *summarizeOverlaps* and *htseq-count* were run in 'union' mode. Results are shown for countOverlaps (i) when run on the whole genome at once and (ii) when run chromosome by chromosome.

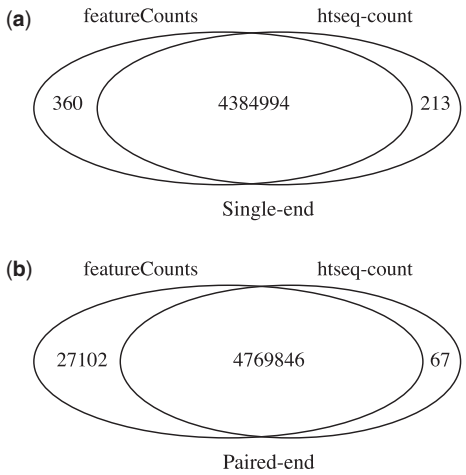

Fig. 2. Concordance between *featureCounts* and *htseq-count* regarding assignment of reads (**a**) or fragments (**b**) to genes. The dataset is the same as for Table 1. The Venn diagram overlap gives the number of reads or fragments assigned by both methods to the same gene. The remaining counts give the number of reads or fragments assigned by one method to some genes but not by the other method

2.3 GHz CPUs and 512 GB of memory. All programs were run using a single CPU without multithreading. Comparisons used software packages *Subread* 1.4.2, *Rsubread* 1.12.2, *GenomicRanges* 1.12.5, *IRanges* 1.18.4, *htseq-count* 0.5.4p3 and *BEDTools* 2.17.0.

5.3 Comparative performance when counting fragments

We went on to compare the same methods for counting paired-end fragments, using the full SEQC paired-end data. *Summarize Overlaps* counted far fewer fragments than *featureCounts* and *htseq-count* (Table 1, column 3). The main reason for this discrepancy is that *summarizeOverlaps* requires fragments to have both end reads successfully mapped before assigning them to genes, whereas *featureCounts* and *htseq-count* do not have such a requirement, i.e. they can assign fragments just one end mapped. With a read length of 101 bp, fragments with only one end mapped can still have relatively high mapping confidence. Counting such fragments seems likely to benefit downstream analyses. Many aligners report fragments that have only one end mapped, including Subread, Subjunc (Liao *et al*., 2013), Bowtie (Langmead *et al*., 2009) and TopHat (Trapnell *et al*., 2009). Almost all (92%) of fragments counted by *featureCounts* but not by *summarizeOverlaps* were assigned to genes that also had at least 100 assigned fragments with both ends mapped. This shows that the fragments were assigned to genuinely expressed genes, giving confidence that the extra fragments have been assigned correctly. Only 0.1% of extra fragments counts by *featureCounts* were assigned to genes not supported by any fragment with both ends mapped.

htseq-count also counts fewer fragments than *featureCounts* in this evaluation (Fig. 2b). Running *htseq-count* in 'IntersectionStrict' or 'IntersectionNotEmpty' modes instead of 'Union' mode did not cause it count more fragments.

featureCounts can distinguish those features that overlap with different numbers of reads from the same fragment. For example, if two genes were found to both overlap with a fragment but one gene was found to overlap with only one read and the other with both reads from that fragment, *featureCounts* will assign that fragment to the gene overlapping with both reads. However, *htseq-count* will take this fragment as ambiguous and will not assign it to any gene. This is the main reason why *featureCounts* counted slightly more fragments than *htseq-count*. *featureCounts* uses the size of overlap (in terms of reads) to recover those 'ambiguous' fragments. For this dataset, >86% of fragments assigned by *featureCounts* but not by *htseq-count* were assigned to genes that already had at least 100 unambiguous fragments assigned by both methods. Only 0.2% of extra fragments assigned by *featureCounts* were not supported by commonly assigned fragments. This again shows that the extra fragments are being assigned to genuinely expressed genes, suggesting that the extra fragments are likely to have been correctly assigned.

Table 1 (columns 4 and 5) shows that *featureCounts* was considerably faster (>10-folds) and more memory efficient than the other programs. *summarizeOverlaps* was also run chromosome by chromosome to save memory. That is, reads were split into groups according to the chromosomes they were mapped to and each group of reads was summarized separately. But it still used 20 times as much memory as *featureCounts*.

6 PERFORMANCE ON CHIP-SEQ DATA

6.1 Data and annotation

Now we compare the performance of *featureCounts* with existing software tools for counting gDNA-seq reads at the feature level. As an example case study, we use a ChIP-seq dataset that was generated as part of a study of global changes in the mammary stem cell epigenome under hormone perturbation (Pal *et al*., 2013). Specifically the dataset was generated to find genomic regions associated with the H3K27me3 epigenetic histone mark (tri-methylation of the histone H3 lysine 27) in mouse mammary stem cells. This dataset consists of 15 million pairs of 35 bp DNA reads generated by an Illumina Genome Analyzer IIx. The study analyzed the total number of fragments mapped to the broad region of each gene, where the broad region is defined to be the entire gene body from first to last base plus the 3 kb region immediately upstream from the transcription start of the gene representing the putative promotor region (Pal *et al*., 2013).

The read mapping and annotation used here follows the original study. Reads were mapped to the mouse genome (mm9) using the *Subread* aligner (Liao *et al*., 2013). Fragments were included in the evaluation only if both paired reads were successfully mapped to the genome and if the fragment defined by the end reads was between 50 and 500 bp long. The transcription start and end positions for each gene were obtained from the NCBI mouse RefSeq annotation (build 37.2).

6.2 Comparative performance

We summarized paired-end fragments at the feature level, where the features represented the broad regions of all annotated genes. In this application, a fragment should be counted multiple times if it overlaps multiple genes.

Table 2 compares the performance of *featureCounts* to that of the *countOverlaps* function of the *IRanges* package, the *htseq-count* script and the *coverageBED* program in the *BEDTools* software suite. *countOverlaps* was used for this comparison instead of *summarizeOverlaps* because it allows multi-overlap reads to be assigned to multiple features.

featureCounts and *countOverlaps* yielded identical counts for every gene, but *featureCounts* was considerably faster and more memory efficient. *countOverlaps* was also run chromosome by chromosome to save memory. This reduced the peak memory usage, although it remained more than a hundred times that used by *featureCounts*. Note that *featureCounts*, unlike *countOverlaps*, can count fragments with only one end successfully mapped, but such fragments were not included in this evaluation to ensure that the timings and memory use for *featureCounts* and *countOverlaps* were for identical operations.

coverageBED assigned slightly fewer fragments than *featureCounts*. We found this was because *coverageBED* used only the first read of each fragment to assign the whole fragment to features. *htseq-count* counted 7–8% fewer fragments, presumably because it does not count multi-overlap fragments. *htseq-count* was run in 'intersection-nonempty' mode as well as in

Table 2. Performance results on the H3K27me3 ChIP-seq dataset

Method	Number of fragments	Time (min)	Memory (MB)
featureCounts	5 392 155	0.9	4
CountOverlaps (whole genome at once)	5 392 155	24.4	7000
CountOverlaps (by chromosome)	5 392 155	36.6	783
htseq-count (union)	4 978 050	36.0	31
htseq-count (intersection-nonempty)	4 993 644	35.7	31
coverageBED	5 366 902	4.4	41

Note: *featureCounts* is the fastest method and uses least memory. It counts the same number of fragments as *countOverlaps* but more than *htseq-count* or *coverageBED*. Table shows the total number of fragments counted, time taken and peak memory used. *featureCounts* was set to count multi-overlap fragments. Results are shown for *countOverlaps* (i) when run on the whole genome at once and (ii) when run chromosome by chromosome. Running by chromosome conserves memory but takes longer. Results are shown for *htseq-count* in two possible counting modes. For *coverageBED*, the BAM input file was converted to a BED file for summarization using *bamToBed* with options '-bedpe' and '-split'.

'union' mode so as to count more fragments, but this did not make up much of the shortfall.

Columns 3 and 4 of Table 2 show that *featureCounts* was about five times faster and used about 10 times less memory than the next most efficient tool.

7 PERFORMANCE WHEN THE NUMBER OF REFERENCE SEQUENCES IS LARGE

7.1 Simulated data

Datasets with large numbers of reference sequences are challenging because the read count software must match the contig names of features to those of reads in an efficient manner. To examine performance under these conditions, we simulated reads from an incompletely assembled genome with relatively large number of scaffolds. We used an assembly of the budgerigar genome generated in the Assemblathon 2 project (Bradnam *et al*., 2013; Howard *et al*., 2013). For this assembly there are 16 204 annotated genes with 153 724 exons located on 2850 scaffolds. Eight million 100 bp single-end reads were randomly extracted from the annotated exonic regions in the assembled scaffolds. The simulated reads were entered into a SAM file. Read mapping information was filled according to the locations from where the reads were extracted.

7.2 Comparative performance

The simulated reads were then summarized at the gene level. Table 3 compares *featureCounts* to *summarizeOverlaps* and *htseq-count* for this dataset. As seen before on the RNA-seq data, *summarizeOverlaps* yields the same counts as *featureCounts*, whereas *htseq-count* yields slightly fewer. *featureCounts* maintained its efficiency advantage over the other methods in this evaluation, increasing its speed advantage over *summarizeOverlaps* in this context.

Table 3. Performance with RNA-seq reads simulated from an annotated assembly of the Budgerigar genome

Methods	Number of reads	Time (mins)	Memory (MB)
featureCounts	7 924 065	0.6	15
summarizeOverlaps (whole genome at once)	7 924 065	12.6	2400
summarizeOverlaps (by scaffold)	7 924 065	53.3	262
htseq-count	7 912 439	12.1	78

Note: The annotation includes 16 204 genes located on 2850 scaffolds. *featureCounts* is fastest and uses least memory. Table gives the total number of reads counted, time taken and peak memory used. *htseq-count* was run in 'union' mode.

8 THEORETICAL ANALYSIS OF ALGORITHMIC COMPLEXITY

This section gives a theoretical analysis of the computational time and memory storage required by *featureCounts* and the other algorithms. The actual time and memory consumed by a computer program depends on the computer hardware, operating system and other factors as well as on the mathematical efficiency of the algorithm used. However, we can derive theoretical expressions for the rate at which time and memory used by any specific algorithm should increase with the number of reads, the number of features and the density of features in the genome. The time complexity of the *featureCounts* algorithm can be derived as $O(f \log f + r\sqrt{k_1})$, where f is the number of features, r is the number of reads and k_1 is the number of features included in a genomic bin. This means that the number of elementary computations used by the algorithm increases linearly with the number of reads, independently of the number of features and somewhat faster than linearly with the number of features. The space complexity of the *featureCounts* algorithm is $O(f + b_1)$, meaning that memory used increases linearly with the number of features plus the number of bins b_1. Time and space complexities for all the algorithms are given in Table 4.

The number of reads is typically large, so rate of increase with r is especially important. The *featureCounts* algorithm has the lowest time complexity of the algorithms being compared. The red-black tree search algorithm used by *htsesq-count* has higher complexity because $\log f$ is typically larger than the square root of the number of features per bin used by *featureCounts*. The hierarchical search within bins used by *featureCounts* is more efficient than the sequential search carried out by *coverageBED* because most reads overlap multiple levels of bins with *coverageBED* causing k_2 to be typically greater than k_1. *countOverlaps* and *summarizeOverlaps* sort reads according to their mapped locations and then use an interval tree to find features overlapping with reads. The sort step is especially expensive and introduces $r \log r$ terms.

The *htseq-count* algorithm has the best theoretical space complexity, but *featureCounts* is not far behind because the number of bins b_1 is usually small compared with f. *BEDTools* has a higher space complexity than *featureCounts* because it uses

Table 4. Theoretical time and space complexity

Method	Time	Memory
featureCounts	$f\log f + r\sqrt{k_1}$	$f + b_1$
countOverlaps	$f\log f + r\log r + r$	$f + r$
summarizeOverlaps	$2f\log f + 2r\log r + r + f$	$f + r$
htseq-count	$f\log f + r\log f$	f
coverageBED	$f\log f + rk_2$	$f + b_2$

Note: The table gives proportionality factors for the number of computations (time complexity) and memory locations (space complexity) required by each algorithm. Time complexities depend on the number of features f, the number of reads r and the number of features included in genomic bins overlapping the query read, k. Space complexity also depends on the number of bins, b. Complexities are interpreted as $O(x)$ where x is the expression given in the table. The number of bins used by *coverageBED*, b_2, is greater than the number of bins used by *featureCounts*, b_1. The number of within-bin features k_2 for *coverageBED* is typically $>k_1$ for *featureCounts*.

more bins. *CountOverlaps* and *summarizeOverlaps* have higher space complexities that depend on the number of reads as well as on the number of features.

In practice, the running time and memory usage of a software program are determined not just by the inherent time and space complexities of the algorithm it adopts but also by the efficiency of the software implementation. The practical timings show that *featureCounts* achieves further efficiency gains from high performance C programming and direct memory manipulation.

9 DISCUSSION

Read summarization is an important step in many next-gen sequencing data analyses. In this study, we developed a new read summarization program called *featureCounts* and compared it with existing methods in terms of efficiency and accuracy. Our results showed a high concordance between alternative methods in summarization accuracy. However, there was a large difference observed in their computational cost. The *featureCounts* method was found to be an order of magnitude faster on average and far more memory efficient than other methods. The high computational efficiency of *featureCounts* is due to its ultrafast feature search algorithm and its highly efficient implementation entirely using the C programming language.

All results presented in this article were produced using a single thread, but *featuresCounts* also supports multithreaded processing, making it particularly useful for summarizing data generated in large sequencing studies. It is the only existing read count method that supports multithreading.

This program provides a wide range of options to allow users to fully control how their read data can be best summarized. Users can choose whether they should count the reads that overlap with more than one feature or meta-feature. This choice is often determined by the experiment type. Reads overlapping with more than one gene (a meta-feature) should not be counted in a RNA-seq experiment because such reads can only originate from one gene, but usually they should be counted in a gDNA-seq experiment such as a histone ChIP-seq experiment. This program also allows users to filter out reads before summarization using a number of metrics such as mapping quality scores, fragment mappability (whether two ends from the same fragment are both successfully mapped or not), fragment length, strandness, chimerism and so on. It can automatically detect either SAM or BAM format read input and sort reads by name if paired reads are not in consecutive positions in the input. It also allows users to specify whether those reads that were reported with more than one mapping location (multi-mapping) should be counted or not. Many of these useful features are not supported by other programs.

The *featureCounts* program has been implemented in both SourceForge *Subread* package (Liao and Shi, 2013) and Bioconductor *Rsubread* package (Shi and Liao, 2013a). The R function provides users with an R interface so that they can access this program from their familiar R environment. It calls the underlying compiled C program to perform all the read summarization operations, and hence has the same speed and memory usage as that of the SourceForge *Subread* package, which is written entirely in C. The implementation of *featureCounts* in R enables complete pipelines to be established for analyzing next-gen sequencing data using Bioconductor software programs. For example, functions included in Bioconductor packages *Rsubread*, *limma* and *edgeR* can be used to perform complete RNA-seq and histone ChIP-seq analyses, starting from read mapping, to read summarization and finally to differential expression analyses or differential histone modification analyses. Owing to its high efficiency and accuracy, we believe the *featureCounts* program will be a useful tool in the bioinformatics toolbox for analyzing next-gen sequencing data.

ACKNOWLEDGEMENT

The authors thank Leming Shi and Charles Wang for providing the SEQC pilot data and Aaron Lun for helpful comments.

Funding: Project grant (1023454) and a Fellowship (to G.K.S.) from the Australian National Health and Medical Research Council (NHMRC). Victorian State Government Operational Infrastructure Support and Australian Government NHMRC IRIIS.

Conflict of Interest: none declared.

REFERENCES

Aboyoun,P. *et al.* (2013) GenomicRanges: representation and manipulation of genomic intervals. R package version 1.12.5. http://www.bioconductor.org (30 April 2013, date last accessed).

Anders,S. (2013) HTSeq: analysing high-throughput sequencing data with Python. http://www-huber.embl.de/users/anders/HTSeq/doc/overview.html (30 April 2013, date last accessed). Version 0.5.4p3.

Anders,S. and Huber,W. (2010) Differential expression analysis for sequence count data. *Genome Biol.*, **11**, R106.

Anders,S. *et al.* (2012) Detecting differential usage of exons from RNA-seq data. *Genome Res.*, **22**, 2008–2017.

Anders,S. *et al.* (2013) Count-based differential expression analysis of RNA sequencing data using R and Bioconductor. *Nat. Protoc.*, **8**, 1765–1786.

Auer,P. and Doerge,R.W. (2011) A two-stage Poisson model for testing RNA-seq data. *Statistical Applications in Genetics and Molecular Biology*, **10**, 1–26.

Bhattacharyya,S. *et al.* (2013) Genome-wide hydroxymethylation tested using the help-gt assay shows redistribution in cancer. *Nucleic Acids Res.*, **41**, e157–e157.

Bradnam,K. *et al*. (2013) Assemblathon 2: evaluating *de novo* methods of genome assembly in three vertebrate species. *Gigascience*, **2**, 10.

Brent Lab. (2013) GTF2.2: a Gene Annotation Format. Washington University, St Louis. http://mblab.wustl.edu/GTF22.html. (27 September 2013, date last accessed).

Flicek,P. *et al*. (2012) Ensembl 2012. *Nucleic Acids Res*., **40**, D84–D90.

Fonseca,N.A. *et al*. (2012) Tools for mapping high-throughput sequencing data. *Bioinformatics*, **28**, 3169–3177.

Gentleman,R. *et al*. (2004) Bioconductor: open software development for computational biology and bioinformatics. *Genome Biol*., **5**, R80.

Hardcastle,T.J. and Kelly,K.A. (2010) baySeq: Empirical Bayesian methods for identifying differential expression in sequence count data. *BMC Bioinformatics*, **11**, 422.

Harris,R.A. *et al*. (2010) Comparison of sequencing-based methods to profile DNA methylation and identification of monoallelic epigenetic modifications. *Nat. Biotechnol*., **28**, 1097–1105.

Howard,J. *et al*. (2013) *De novo* high-coverage sequencing and annotated assemblies of the Budgerigar genome. *GigaScience Database*, http://dx.doi.org/10.5524/100059 (22 July 2013, date last accessed).

Kent,W. *et al*. (2002) The human genome browser at UCSC. *Genome Res*., **12**, 996–1006.

Langmead,B. *et al*. (2009) Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biol*., **10**, Article R25.

Law,C.W. *et al*. (2013) Voom! precision weights unlock linear model analysis tools for RNA-seq read counts. http://www.statsci.org/smyth/pubs/VoomPreprint.pdf (25 September 2013, date last accessed).

Li,B. and Dewey,C. (2011) RSEM: accurate transcript quantification from RNA-Seq data with or without a reference genome. *BMC Bioinformatics*, **12**, 323.

Li,H. and Durbin,R. (2009) Fast and accurate short read alignment with Burrows–Wheeler transform. *Bioinformatics*, **25**, 1754–1760.

Li,H. *et al*. (2009) The sequence alignment/map format and SAMtools. *Bioinformatics*, **25**, 2078–2079.

Li,J. *et al*. (2012) Normalization, testing, and false discovery rate estimation for RNA-sequencing data. *Biostatistics*, **13**, 523–538.

Liao,Y. and Shi,W. (2013) The Subread package: a toolkit for processing next-gen sequencing data. http://subread.sourceforge.net (15 November 2013, date last accessed). SourceForge package version 1.4.2.

Liao,Y. *et al*. (2013) The Subread aligner: fast, accurate and scalable read mapping by seed-and-vote. *Nucleic Acids Res*., **41**, e108.

Man,K. *et al*. (2013) The transcription factor IRF4 is essential for TCR affinity-mediated metabolic programming and clonal expansion of T cells. *Nat. Immunol*., **14**, 1155–1165.

Marco-Sola,S. *et al*. (2012) The GEM mapper: fast, accurate and versatile alignment by filtration. *Nat. Methods*, **9**, 1185–1188.

McCarthy,D.J. *et al*. (2012) Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. *Nucleic Acids Res*., **40**, 4288–4297.

Metzker,M.L. (2009) Sequencing technologiesthe next generation. *Nature Rev. Genet*., **11**, 31–46.

Nookaew,I. *et al*. (2012) A comprehensive comparison of RNA-Seq-based transcriptome analysis from reads to differential gene expression and cross-comparison with microarrays: a case study in *Saccharomyces cerevisiae*. *Nucleic Acids Res*., **40**, 10084–10097.

Oshlack,A. *et al*. (2010) From RNA-seq reads to differential expression results. *Genome Biol*., **11**, 220.

Pages,H. *et al*. (2013) IRanges: infrastructure for manipulating intervals on sequences. R package version 1.18.4. http://www.bioconductor.org (30 April 2013, date last accessed).

Pal,B. *et al*. (2013) Global changes in the mammary epigenome are induced by hormonal cues and coordinated by Ezh2. *Cell Rep*., **3**, 411–426.

Park,P.J. (2009) Chip–seq: advantages and challenges of a maturing technology. *Nat. Rev. Genet*., **10**, 669–680.

Pruitt,K. *et al*. (2012) NCBI Reference Sequences (RefSeq): current status, new features and genome annotation policy. *Nucleic Acids Res*., **40**, D130–D135.

Quinlan,A. and Hall,I. (2010) BEDTools: a flexible suite of utilities for comparing genomic features. *Bioinformatics*, **26**, 841–842.

Rapaport,F. *et al*. (2013) Comprehensive evaluation of differential gene expression analysis methods for Rna-seq data. *Genome Biol*., **14**, R95.

Reyes,A. *et al*. (2013) Drift and conservation of differential exon usage across tissues in primate species. *Proc. Natl Acad. Sci. USA*, **110**, 15377–15382.

Robinson,M. *et al*. (2010) edgeR: a Bioconductor package for differential expression analysis of digital gene expression data. *Bioinformatics*, **26**, 139–140.

Ross-Innes,C.S. *et al*. (2012) Differential oestrogen receptor binding is associated with clinical outcome in breast cancer. *Nature*, **481**, 389–393.

Schuster,S.C. (2008) Next-generation sequencing transforms today's biology. *Nat. Methods*, **5**, 16–18.

Shi,W. and Liao,Y. (2013a) Rsubread: an R package for the alignment, summarization and analyses of next-generation sequencing data. R package version 1.12.2. http://www.bioconductor.org (20 November 2013, date last accessed).

Shi,W. and Liao,Y. (2013b) Subread/Rsubread Users Guide. http://bioinf.wehi.edu.au/subread-package/SubreadUsersGuide.pdf (20 November 2013, date last accessed). Version Subread 1.4.2/Rsubread 1.12.2.

Shi,L. *et al*. (2006) The microarray quality control (MAQC) project shows inter-and intraplatform reproducibility of gene expression measurements. *Nat. Biotechnol*., **24**, 1151–1161.

Trapnell,C. *et al*. (2009) TopHat: discovering splice junctions with RNA-seq. *Bioinformatics*, **25**, 1105–1111.

Trapnell,C. *et al*. (2010) Transcript assembly and quantification by RNA-seq reveals unannotated transcripts and isoform switching during cell differentiation. *Nat. Biotechnol*., **28**, 511–515.

Valouev,A. *et al*. (2008) Genome-wide analysis of transcription factor binding sites based on chip-seq data. *Nat. Methods*, **5**, 829–834.

Wellcome Trust Sanger Institute. (2013) GFF (General Feature Format) specifications document. http://www.sanger.ac.uk/resources/software/gff/spec.html. (7 July 2013, last date accessed).

Wu,H. *et al*. (2013) A new shrinkage estimator for dispersion improves differential expression detection in RNA-seq data. *Biostatistics*, **14**, 232–243.

Zhang,Y. *et al*. (2008) Model-based analysis of ChIP-Seq (MACS). *Genome Biol*, **9**, R137.