Genome analysis

Sambamba: fast processing of NGS alignment formats

Artem Tarasov¹, Albert J. Vilella², Edwin Cuppen³,4, Isaac J. Nijman⁴ and Pjotr Prins⁴,5,*

¹Department of Statistical Simulation, St. Petersburg State University, St. Petersburg, Russia, ²Illumina Cambridge, Cambridge, UK, ³Hubrecht Institute, Royal Netherlands Academy of Arts and Sciences, Utrecht, The Netherlands, ⁴Department of Medical Genetics, Institute for Molecular Medicine, University Medical Centre Utrecht, Utrecht, The Netherlands and ⁵Department of Nematology, Wageningen University, Wageningen, The Netherlands

*To whom correspondence should be addressed.

Associate Editor: Inanc Birol

Received on December 13, 2014; revised on February 2, 2015; accepted on February 10, 2015

Abstract

Summary: Sambamba is a high-performance robust tool and library for working with SAM, BAM and CRAM sequence alignment files; the most common file formats for aligned next generation sequencing data. Sambamba is a faster alternative to samtools that exploits multi-core processing and dramatically reduces processing time. Sambamba is being adopted at sequencing centers, not only because of its speed, but also because of additional functionality, including coverage analysis and powerful filtering capability.

Availability and implementation: Sambamba is free and open source software, available under a GPLv2 license. Sambamba can be downloaded and installed from http://www.open-bio.org/wiki/Sambamba. Sambamba v0.5.0 was released with doi:10.5281/zenodo.13200.

Contact: j.c.p.prins@umcutrecht.nl

1. Introduction

Processing speed matters, not only for diagnostics, but also for analysis and sharing of computational resources. Next-generation sequencing (NGS) is increasingly used as a genetic screening tool in diagnostics (Gullapalli et al., 2012) and reducing time from sample intake to test result/diagnosis potentially saves lives. Introducing multi-core processing can accelerate steps in a pipeline when the CPU is the bottleneck (Trelles et al., 2011)

Since its introduction by the 1000 Genomes Project (Siva, 2008), the sequence alignment/map format (SAM) and its compressed binary counterpart (BAM) have become the de facto file formats used for storing and distributing NGS data. Samtools is the original tool for SAM/BAM files processing, including data extraction and filtering (Li et al., 2009). Recently, samtools added the CRAM format as a compressed alternative to SAM/BAM (Cochrane et al., 2013). While samtools exploits the speed of the low-level C programming language and uses streamed data for efficiency, it has limited support for parallel processing (Fig. 1). Samtools has inspired a number of other BAM processors, notably Picard (Picard, 2009), samblaster (Faust and Hall, 2014), biobambam (Tischler and Leonard, 2014) and Scramble (Bonfield, 2014), each of which is either slower than samtools, or offers a subset of its functionality.

To accelerate analysis pipelines we created Sambamba, a new incarnation of samtools that fully utilizes parallel processing. Sambamba (which means 'parallel' in Swahili) is written in the D programming language, a modern programming language, a modern programming language with runtime performance similar to that of C (Alexandrescu, 2010). D has powerful abstractions for parallel computing which make it possible to scale computations with the number of cores (Fig. 1). When running a Human cancer exome SNV calling pipeline on the results of a single Illumina HiSeq 2500 flowcell in fast mode (2000 genes, 300
Sambamba: fast processing of NGS alignment formats

Fig. 1. Processing speed comparison of samtools and sambamba. Wall-clock time (s) versus number of threads to convert an 11-GB CRAM (1000 genomes HG00110) to 108-GB SAM. With Samtools, view is bound to a single thread at CPU 90%. With Sambamba, IO gets saturated at approximately CPU 250%. When using a faster RAM-disk, IO gets saturated at approximately CPU 350%. For samtools a RAM-disk makes no difference. When adding more threads, performance reproducibly degrades because of CPU cache contention. All timings were performed on a server-class machine with 512 GB of RAM and 48 CPU cores (4 ¥12-core AMD Opteron(tm) Processor 6174 @2.2 Ghz with 6 Mb L2 cache) Samtools version v1.0-15 using htslib v1.0-1 and sambamba v0.5.0 compiled with the LLVM D-compiler v0.14.0.

Table 1. Examples of processing of 31-GB BAM and matching 11-GB CRAM of HG00110 with sambamba and samtools. Wall-clock time (s in seconds) reflects improved analysis time. CPU (×100%) reflects effective multi-core utilisation. See Figure 1 caption for description of hardware, software and measurements.

<table>
<thead>
<tr>
<th></th>
<th>samtools (s)</th>
<th>Sambamba (s)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>BAM view</td>
<td>1506</td>
<td>429</td>
<td>3.5×</td>
</tr>
<tr>
<td>Filter&lt;sup&gt;a&lt;/sup&gt;</td>
<td>195</td>
<td>35</td>
<td>471</td>
</tr>
<tr>
<td>Sort</td>
<td>12 288</td>
<td>396%</td>
<td>1265</td>
</tr>
<tr>
<td>Index</td>
<td>577</td>
<td>137</td>
<td>562</td>
</tr>
<tr>
<td>Markdup&lt;sup&gt;b&lt;/sup&gt;</td>
<td>5220</td>
<td>2296</td>
<td>278</td>
</tr>
<tr>
<td>Merge</td>
<td>3090</td>
<td>571%</td>
<td>2247</td>
</tr>
<tr>
<td>mpileup&lt;sup&gt;c&lt;/sup&gt;</td>
<td>7750</td>
<td>584</td>
<td>4409</td>
</tr>
<tr>
<td>BAM to CRAM</td>
<td>4354</td>
<td>640</td>
<td>796</td>
</tr>
<tr>
<td>CRAM to SAM</td>
<td>1850</td>
<td>729</td>
<td>347</td>
</tr>
<tr>
<td>CRAM index</td>
<td>9</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>

<sup>a</sup>Filter on q > 30 and Chr1.

<sup>b</sup>For markdup samtools v0.19 was used.

<sup>c</sup>mpileup to VCF on 2GB BAM of Chr1 only.

Source code: Sambamba abides by the rules of the ‘Small tools MANIFESTO for Bioinformatics’ (Prins et al., 2014). The sambamba source code is extensible and maintainable. For SAM parsing we opted for Ragel, a finite-state machine compiler, which generates a fast look-ahead parser with input validation, making the code base even more compact (Thurston, 2006). Sambamba uses a unit testing framework with continuous integration testing, so that existing functionality is validated every time the code base is changed.

3. Conclusion

Sambamba is a software engineering example that shows how to make effective use of the D programming language and multi-core computers to reduce the time needed to get from sample to result. Whole genome sequencing and growing sample numbers make such performance improvements increasingly relevant.

Acknowledgements

We thank James Bonfield for his CRAM implementation and for constructive comments. We thank the Google Summer of Code and the Open Bioinformatics Foundation for supporting the original work.

Funding

This work was partially funded by the Google Summer of Code program (AT).

Conflict of Interest: none declared.

References


