The BigWig and BigBed file formats were originally designed for the visualization of next-generation sequenced data through a genome browser. Due to their versatility, these formats have long since become ubiquitous for the storage of processed sequencing data and regularly serve as the basis for downstream data analysis. As the number and size of sequencing experiments continues to accelerate, there is an increasing demand to efficiently generate and query BigWig and BigBed files in a scalable and robust manner, and to efficiently integrate these functionalities into data analysis environments and third-party applications. Here, we present Bigtools, a feature-complete, high-performance, and integrable software library for generating and querying both BigWig and BigBed files. Bigtools is written in the Rust programming language and includes a flexible suite of command line tools as well as bindings to Python. Bigtools is cross-platform and released under the MIT license. It is distributed on Crates.io and the Python Package Index, and the source code is available at https://github.com/jackh726/bigtools.

Introduction

BigWig and BigBed, collectively referred to as Big Binary Indexed (BBI) files, are compressed, binary, and indexed formats for storing reference genome-aligned next-generation sequencing (NGS) information (1). BigBed is the binary equivalent of the Browser Extensible Data (BED) text format that represents features associated with (potentially overlapping) genomic intervals, e.g. peak calls or transcript annotations. Analogously, BigWig is the binary reincarnation of the textual Wiggle and BedGraph formats, which encode a quantitative step-function mapping unique floating point values to bases in the genome, while accommodating missing values, e.g. read pileups from a NGS experiment. Introduced in 2009, these file types are among the most common data artifacts derived from bulk and pseudo-bulk omic experimental modalities.

BBI files were designed for the display of large, distributed datasets in the UCSC Genome Browser; however, these features have made BigWig and BigBed useful beyond genome browsers, and today they are widely used for data analysis. For example, as of the end of 2023, the ENCODE Project data portal (2) provides for analysis purposes approximately 15,000 BigWig files and 17,000 BigBed files comprising 17 different schemas, excluding archived data. In BBI files, the primary data, index, and summary data at several resolutions or “zoom levels” are stored within the same file. The block compression and sparse index design allows for efficient random and remote access. However, because of their complex binary architecture, BBI files require specialized supporting software to enable reading, writing, and random access.

As the availability and use of BBI files grows, there is a growing desire to integrate BBI functionality into third-party bioinformatic applications. Since their inception, the command line interface (CLI) binaries from Kent et al. (1), hereon referred to as the UCSC tools, have served as the canonical software for generating and interacting with BigWig and BigBed files. While it is possible to interface directly with the C code that powers the UCSC tool executables (e.g. bx-python (3), rtracklayer (4), bwtool (5), plastid (6), and pybbi (7)), architectural limitations hinder its suitability as a standalone BBI library. This prevents widespread use of the UCSC reference implementation for application programming or for integration in interactive computing environments, particularly for languages with a runtime like Python, R, or Javascript.

Aside from the need for third-party integration, there is also a growing need for increased computational flexibility to tailor to different environments and use cases. First, while BBI files have traditionally been served statically on web servers, today users commonly require fast and easy querying of data locally for bioinformatic analyses. Similarly, data processing and generation of BBI files must also scale with rising volumes of raw NGS data. While traditionally done on high-performance computing clusters, data processing is also expanding into cloud compute platforms, particularly in large research consortia. In such environments, optimizing compute resources directly translates to decreased operating costs as well as reduced environmental impact.

We introduce Bigtools, a full-featured Rust library for BBI file creation, access, and manipulation that supports flexible resource usage and parallelization. Leveraging this library, Bigtools also includes versatile and high-performance CLI tools and Python bindings.

Results

Bigtools is a full-featured BBI library. Due to the limitations of the UCSC reference implementation, a number of alternative BBI software implementations have emerged in different languages, having varying levels of feature sup-
While alternative implementations for reading and writing Bigtools offers high-performance and flexible CLI tools. While alternative implementations for reading and writing BigWig and BigBed files are popular, none so far offer alternative CLI tools. This limits their use, particularly in pipelines and scripts. Here, we show that the Bigtools CLI has not only substantial performance improvements over the reference implementation but also a number of ergonomic benefits that can make it more versatile and easier to use.

To measure differences in performance between Bigtools and the reference implementation, we have set up a number of benchmarks (Table 2) comparing the processing of real-world data available from the ENCODE consortium (15). Each benchmark primarily measures the run time and memory usage, but also samples CPU usage and memory over time. The benchmarks are repeated three times.

First, we compare single-threaded performance between Bigtools and the UCSC implementation. The wall time of Bigtools varies between approximately 1.5× to 2.5× faster than the UCSC tools. In addition, the Bigtools tools use considerably less memory (approximately 7× to 340×) than the UCSC tools. Notably, for tools like bigWigAverageOverBed and bigWigMerge, where the UCSC implementation loads in whole chromosomes at a time, Bigtools implements lazy computation which reduces the memory consumption by over an order of magnitude.

Second, unlike the UCSC tools, Bigtools offers multi-threaded implementations for nearly all CLI tools (excluding only bigwigmerge). Increasing the number of threads to just two results in between approximately 3% to 41% faster wall time, while scaling up to six threads results in between approximately 24% to 73% faster wall time (Table 2). Memory usage does increase moderately for most of the tools, but remains comfortably below that of the UCSC tools.

Aside from performance, Bigtools offers a number of ergonomic benefits over the UCSC implementation. First, Bigtools offers several options for configuration that allow flexibility over the input representation. For example, while writing a BigWig file, the UCSC implementation reads through the input bedGraph file multiple times, which prohibits creating a BigWig from standard input. By contrast, Bigtools provides the option to write BBI files in a single pass of the input data, including from standard input. These configuration options also allow tuning of performance to the size of the input data: multiple passes are fine for smaller files, but results in significant time overhead for very large files. While writing as a single-pass Bigtools utilizes temporary files to avoid a memory overhead from index and zoom building, though this can be disabled. In addition to various configuration options, Bigtools also works as a drop-in replacement for the equivalent UCSC tools, supporting the most commonly used CLI tools.

Table 1. Comparison of Big Binary Indexed file software implementations

<table>
<thead>
<tr>
<th>Language</th>
<th>UCSC</th>
<th>libBigWig &amp; pyBigWig</th>
<th>Big</th>
<th>bbi.js</th>
<th>bigwig-reader</th>
<th>Bigtools</th>
</tr>
</thead>
<tbody>
<tr>
<td>Read</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Write</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Library</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>Includes bindings</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Includes CLI</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Multithreaded</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

* BigBed writing is not supported
\begin{quote}
    flags. Finally, \textit{Bigtools} can be used either as a single binary with each tool as a subcommand or as separate binaries for each tool.
\end{quote}

\section*{Conclusion}

In summary, we present here a full-featured and high-performance library written in Rust for reading and writing BigWig and BigBed files, along with associated CLI tools and a Python library. \textit{Bigtools} adds to the growing arsenal of core NGS tools in the Rust ecosystem (16, 17). It offers a number of configuration options that allow flexibility depending on file size and resource requirements, including scaling to multiple cores. Thus, \textit{Bigtools} is ideal not only for local use where users may want to prioritize quick, parallel processing in return for higher processor and memory use, but also for cloud or batch workflows where users may want to prioritize minimal system resources in return for slightly longer computation.

\textit{Bigtools} offers substantial benefits over previous library implementations of BigWig and BigBed reading/writing. Namely, \textit{Bigtools} provides: (i) all features available in the UCSC tools, (ii) ease of integration and cross-language compatibility, and (iii) improved performance. In fact, during its development, it has already seen uptake as a dependency in the UCSC tools, (ii) ease of integration and cross-language compatibility, and (iii) improved performance.

The \textit{Bigtools} Rust library and CLI executable are available through crates.io using Rust’s Cargo package manager:

\begin{verbatim}
$ cargo install bigtools
\end{verbatim}

The executables are also available on GitHub. \textit{Pybigtools} is available on PyPI:

\begin{verbatim}
$ pip install pybigtools
\end{verbatim}

\textit{Bigtools} is cross-platform and the executables and Python package are distributed for multiple platforms (including Linux, MacOS, and Windows) and architectures. Documentation for the \textit{Bigtools} library and CLI can be found at https://docs.rs/bigtools and documentation for \textit{Pybigtools} can be found at https://bigtools.readthedocs.io/.

\section*{Acknowledgements}

We thank Vedat Yilmaz, Peter Kerpedjiev, Trevor Manz, Garrett Ng, and the Weng and Maehr labs for helpful discussions and suggestions. JDH and NA acknowledge funding from the NIH Common Fund 4D Nucleome Program (DK107980).

\section*{Bibliography}


<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Wall time (sec)</th>
<th>Max memory (MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UCSC</td>
<td>Bigtools ST</td>
</tr>
<tr>
<td>bigwigaverageoverbed</td>
<td>15.2±0.5</td>
<td>6.2±0.1</td>
</tr>
<tr>
<td>bigwigmerge</td>
<td>113.1±1.6</td>
<td>65.5±1.8</td>
</tr>
<tr>
<td>bedgraphhtobigwig</td>
<td>93.8±1.5</td>
<td>47.2±1.1</td>
</tr>
<tr>
<td>bedtobigbed</td>
<td>61.7±0.2</td>
<td>37.5±1.5</td>
</tr>
</tbody>
</table>

ST: Single-threaded; MT: Multi-threaded (6 threads)