BIOINFORMATICS

Vol. 00 no. 00 2005
Pages ???

Extending BioSeq in Julia: Contributions to an open source project

Kristopher Micinski [1,*], James Parker [2] and Matthew Louis Mauriello [2*]
[1]Department of Computer Science, University of Maryland, College Park

Received on XXXXX; revised on XXXXX; accepted on XXXXX

Associate Editor: XXXXXXX

ABSTRACT
Typically, R is the programming language of choice to perform biological computations. While the dynamic nature of the language allows rapid programming, R suffers from poor performance and is conducive to programming errors. One solution to this problem is the newly developed programming language, Julia. Julia boasts many attractive features such as a JIT compiler and a strong type system, which address many of the shortcomings of R. Specifically, the language produces programs with more guarantees about correctness and runs at speeds approaching that of C. In an effort to improve computational biology and research, we implement **?** parsing of BAM files, representation of genomic ranges, and efficient overlap detection **?** in Julia to take advantage of this new language.
Motivation: Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text.
Results: Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text
Availability: Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text Text
Contact: name@bio.com

1 INTRODUCTION

R is currently the preferred tool of the bioinformatics community. R is very good for rapid programming, however it is also very conducive to programmer error. The language is weakly typed and coercions are implicit, which is often a source of programming error **?**. For instance, consider the following code that attempts to convert a data frame that is read from a file into a matrix.

```
raw <- read.csv(``genotypes.csv'')
genotypes <- data.matrix(raw)
```

At first glance, this appears innocuous, however consider the case when *raw* contains *NAs*. The *NAs* are coerced to *1s* in the matrix. If the researcher is unaware of this behavior, this coercion can lead to incorrect analyses and results.

*to whom correspondence should be addressed

Table 1. Benchmark times relative to C (smaller is better) **?**.

Another issue with R is that it has poor performance. As an interpreted language, all of its code must be evaluated within an interpreter. Running in an interpreter is much slower than native code evaluated directly by a machinefs hardware. One reason for this is that an interpreter must read a line of code, parse it, and evaluate it. On the other hand, native code is compiled and sent as binary instructions to the processor. R is unable to generate native code due to its weak type system. Think about what the interpreter must do when it sees the expression $A * B$ to multiply the objects A and B. It does not know what the types of these objects are so it must figure this out at runtime. Therefore, it cannot know ahead of time whether the operation * corresponds to integer multiplication, matrix multiplication, or many different variants of multiplication. **TODO**: Wrap up sentence that interpretation, R is bad? **TODO**: Kris should revise/elaborate here

To address the shortcomings of R, we investigate the new programming language Julia and implement select functionality from Rfs Bioconductor package. There are many other solutions that perform biological computations. These include Matlabfs Bioinformatics Toolbox, BioPython, and BioLib for FORTRAN **???**.

The Julia programming language actively supports users with writing better code by providing a rich typing system. Types allow us to syntactically specify which programs are well formed. Additionally, performance problems commonly found in R are addressed by the Just In Time (JIT) compiler. This is apparent in Table 1 from **?**, which compares the performance of Julia to many different languages. To highlight the differences in speed between R and Julia, the table shows that Julia is 517.34 times faster than R at performing quicksort.

A strong type system also allows the programmer to make some guarantees about the parameters of program. **TODO**: elaborate here a bit.. **TODO**: Juliafs type system creates convenience for the developer as well. c talk about method dispatch?

Julia has a library for manipulating biological data called BioSeq (Zea, 2013) , however this library does not provide the functionality we are looking for. Currently it handles DNA sequences at the single nucleotide level, but it does not contain data structures for looking at regions of genes. We expand BioSeq and add additional functionality for working with genomic range data. **?** This rest of this paper is laid out as follows. Section 2 discusses specifically the functionality being added to BioSeq and discusses our hypothesis surrounding performance measures we anticipate being provided by Julia over R. Section 3 discuss the particular algorithms used in the countOverlaps() operation as well as discusses the SAMtools API. Section 4 briefly discusses the level of current implementation. Section 5 explores the benchmarking: procedure, data, results, and makes some basic conclusions. Section 6 explains immediate needs before public release. **?**

This project seeks to replicate a portion of BioConductor package that is currently available in R for the Julia language. Several key elements of this package have been selected, which includes: basic support for **?** genomic data classes (i.e. IRanges, GRanges, etc.), support to perform union and intersection operations over genes, the ability to import large BAM files containing genomic data, and support for running overlap queries **?**. With respect to loading BAM files, we have taken advantage of a Foreign Function Interface (FFI) to take advantage of the SAMtools library.

In order to demonstrate that Julia offers performance benefits over R, we will collect system times on the diverse set of tasks that have been replicated in Julia. In particular, this study will look at the total elapse time of the countOverlaps() pipeline as well as the elapse time of each individual set of tasks. Based on the interpreted vs compiled nature of making code comparisons between Julia and R, we make the following hypothesis:

H1. BAM files processing tasks and currently implement data structure operations will show no significant differences in performance times. The use of FFI to call SAMtools is common across both languages and data structure operations are implemented in a naive fashion. Therefore, it is anticipated that the Julia implementation will be faster than R; however, this difference will not be significant.

H2. Performing the countOverlaps() operation on the GRanges interval object will demonstrate significant performance improvements in Julia that result in improvement in the total elapse time of the pipeline. The countOverlaps() operation has been implemented and optimized for running in Julia and is expected to demonstrate performance improvements over R implementation of the same operation.

The purpose of this study is to demonstrate the performance benefits over R that are offered by Julia. These demonstrated benefits will provide support for the continued deve lopment of the bioinformatics package gBioSeqh.

There are no novel algorithms in our implementation, however we do use specific data structures and algorithms to make certain operations efficient. The main example of this is the use of an interval tree. An interval tree can be constructed out of an array of genomic ranges. Then this data structure can be used to efficiently perform queries to find which intervals overlap.

Interval trees are binary trees that contain intervals **?**. Each node of the tree contains a center point, an array of intervals that overlap the center point sorted by the start endpoint, and another array of the same intervals sorted by the finish endpoint. The node also stores a pointer to a left node which is a parent of all intervals that are left of the center point. The same is true for a right node and intervals to the right of the center point. A diagram illustrating interval trees can be seen in Figure **??** where the shown intervals are sorted by the start endpoint.

Intervals trees can be constructed by taking the mean of the given intervals as the center point. Then intervals that overlap the center

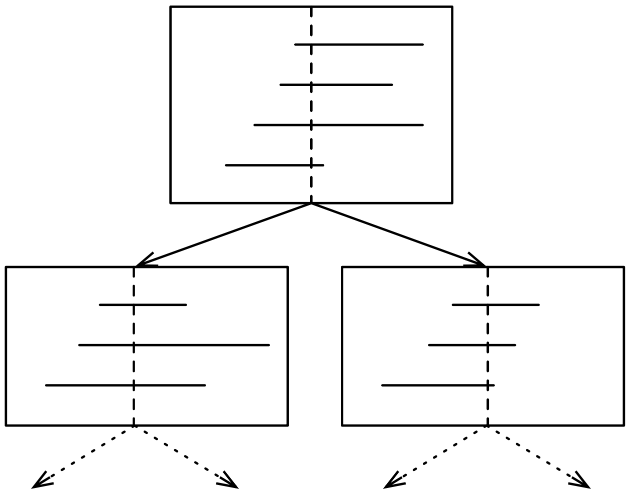

Fig. 1. An interval tree where the intervals in each node are shown as sorted by the start endpoint. The dashed vertical line indicates the center point of the node.

point are stored in the current node. Intervals to the left or right of the center point are then recursively split off to construct the left and right nodes until no intervals remain. This how process of construction takes $O(nlog(n))$ time.

Interval trees are used to find the intervals that overlap a query interval. This is accomplished by checking if the query interval overlaps with a nodefs center point. If this is true then all the intervals in that node are found to overlap. If the query interval is left of the center point, the nodefs forward sorted array of interval is checked if they overlap with the query. Once one of the intervals is found to not overlap the query, the algorithm stops checking more intervals because the intervals are sorted. The same approach is applied to the case when the query interval is right of the center point. After a node is finished, it recursively runs the query against its left or right child nodes as necessary. Performing queries on interval trees takes $O(log(n))$ time as long as the tree is sufficiently balanced.

2 DISCUSSION

these sections/subsections seem a bit weird to me Feel free to change them... the format is loosely based on the guide; hector suggested doing our best with it, but modify as necessary

2.1 Data

Our data was collected using the system timing routines provided by the standard libraries of Julia and R. Table X shows the average time it takes to read in a test data file and map it to memory, run several GRange functions, and complete multiple queries to countOverlaps() routine.

5.2Results

With respect to H1 we found...

With respect to H2 we found...

2.2 Limitations

This implementation developed for this study is very preliminary. The pipeline necessary for performing the countOverlaps() operation has been implemented, but rigorous testing is still required to verify the apparent deterministic behaviour beyond the limited testing that has been performed. Additionally, the data that was used to test has been both randomly generated or provided as sample data for a previous project. The purpose of using randomly generated data was to verify the correctness of the data using simple examples and then testing the performance on various sized data sets. Using past data was done to verify correctness of the FFI interface and SAMtools. Correctness and performance is yet to be verified on large greal datasets. Finally, considerable time was invested into all three components that were constructed for the countOverlaps() pipeline; however, most are not optimized in the same way that countOverlaps() was and therefore are sources of potential performance bottlenecks for future users.

2.3 Conclusions

R might not be the best tool for Bioinformatics. Extending the BioSeq package for Julia will allow researchers to utilize an environment that supports the reduction of programming errors through a typing hierarchy, which provides improved performance over R when combined with the JIT compiler provided by Julia. By implementing a portion of BioSeq, we hope to encourage more development of this package by the Bioinformatics community.

TODO: We propose that the bioinformatics community take advantage of, and begin developing packages for, the Julia Language (http://julialang.org). Julia provides a strong typing architecture that reduces some of the ambiguity in verifying the correctness of a program. Additionally, this information can be provided to a compiler to make some assurances about the code. The JIT compiler of Julia provides performances benefits over R; however, packages for this language are limited making R superior in terms of ease of use in - especially for the bioinformatics community. Without support from developers and researchers, this actively developing language will continue to be overlooked by researchers despite its many benefits.

6Future Work

This study has demonstrated the preliminary development of expansions to the current BioSeq (Zea, 2013) package being developed for Julia. This study showcases that Julia outperforms R in terms of performance during our selected benchmarking operation. This expansion includes the basic data structures, operations, and interfaces associated with genomic ranges and intervals in R and Bioconductor. However, further work on these objects needs to be performed. Specifically, the genomic ranges are not fully implemented as defined by the R specifications found in the genomic ranges vignette (Carlson et al., 2013) and should be. We aim to complete the functions that perform union and intersection operations across genomic ranges and perhaps a few others as necessary..

We have also demonstrated that a simplified API wrapper for accessing SAMtools, and BAM files, through an FFI can be replicated. The API is very minimal, designed produce GRanges of the data specifically to assist in performing the countOverlaps() operation. This API should also be further developed before release **TODO**: expand on this. Finally, general documentation and clean up tasks still need to be performed before a large release of the project can be considered.

ACKNOWLEDGEMENT

The authors would like to thank Dr. Corrada Bravo, of University of Maryland, for his guidance and direction during the course of this project. Additional thanks to the Julia Language and BioSeq communities for providing responsive feedback and assistance with conflict resolution in this early developing environment.