Fall 2016 report

Clark Fitzgerald

December 9, 2016

This is the report for Professor Duncan Temple Lang for 8 units of credit in Fall 2016. The tentative plan is to take the qualifying exam in Spring 2017.

Task

Describe 3 or 4 things to focus on, why they're interesting, what are the challenges, and who cares?

1 Statistically Oriented File Format

Demonstrate a unified file storage / computational model that efficiently processes data larger than memory

The computational model should be flexible enough to allow the user to write a large subset of programs that can easily be done in an expressive language like R. The file format should have the following properties:

1. Fast: Fast read / write operations, hopefully in parallel
2. Typed: Integers are read in as integers, floats are floats, etc. Factors would be nice to have.
3. Efficient: File size no larger than it needs to be, so store as binary rather than text
4. Portable: Works in R, Python, Julia, etc.
5. Iterable: Can take the first k rows and process them sequentially
6. Computationally efficient: Works well for parallel block operations with R / Numpy vectorized computational model.
7. Statistically efficient: Built in features or attributes to facilitate statistical analysis

The last two items are the novel ones. By using a file format that facilitates a chunk based computational strategy it may be possible to realize large gains in performance. One idea

to make it statistically efficient is to enable it to produce iid blocks as in Matloff's software alchemy [10]. Another idea is to store summary statistics or properties of the data or distribution. If it's known that one column is a string with 20 unique values or that another column is distributed approximately $\sim N(0, 1)$ that's useful information.

1.1 Related Work

Since 1998 HDF5 has provided most of the desired properties above [14], including sophisticated chunking and a flexible type system. Mario Annau's recent work on R's h5 package provides R S4 methods for HDF5 data structures. Kane's bigmemory package provides a file backed array.

This year (2016) Wes Mckinney and Hadley Wickham began development on Feather and Apache Arrow. From the documentation for Feather, this "provides binary columnar serialization for data frames." It uses a common C++ implementation with Python and R bindings to support basic types such as numeric, logical, dates, and strings. NA's for all types are also supported. So this may also be a candidate to build from.

R's iotools package provides exciting high performance tools for reading and writing standard text data, ie. CSV files [1]. Pipeline parallelism is explicitly supported. The package is used to integrate R with Hadoop Streaming and Spark to process terabyte to petabyte scale data.

1.2 Challenges

If data exists in memory through both a cached HDF5 chunk and in an R process then it potentially uses twice as much memory as needed.

It would also be possible to simply chunk R objects then serialize them to disk. This would be faster than any other form of serialization but has the disadvantage of not being portable. It may be possible to use serialization for intermediate steps and save writing to the final file format for the last step.

Integrating such a system with typical data sources, ie. plain text files may not be as natural. But maybe we can build off iotools here.

1.3 Applications

Databases and programming environments store tables in an efficient structured format. Yet this structure is often lost when the data is written to a text file. Consequently much time in data analysis is spent loading, cleaning, and inferring the structure of data. Keeping data in a more structured form on disk greatly simplifies the problems and ambiguity associated with 'data munging'.

2 Statistical Calculations on GPU

Develop strategies and tools to use GPU's for data analysis

Ethan Anderes has been working on an application with Joseph Guinness to use Vecchia's approximation to estimate the likelihood function of $X \sim N(0, \Sigma)$ where X is of high dimension, around 1 million. The approximation is used because the actual pdf involves Σ^{-1} which is too expensive to compute [6]. The approximation can be posed as an embarrassingly parallel problem, so this seems like a nice concrete application for a GPU.

The actual computations involve choosing strategies to chunk the data. My research goal is to understand and describe more generally the GPU strategies that can be proven effective.

2.1 Challenges

This application will require understanding the computational model and relevant programming tools for the GPU. This information and experience is very important to have if I'm going to develop further parallel computational ideas.

2.2 Applications

If a GPU can be used from a high level language to accelerate array oriented computations by one or two orders of magnitude this would be very interesting.

3 Code Generation

Automatically generate R bindings for existing C and C++ libraries

This would be an extension of Temple Lang's work with the RCIndex [12] and RCodeGen [13] packages, along with other prior work.

Writing R / C bindings by hand has several problems:

1. Error Prone: Computers do repetitive tasks better than humans.
2. Tedious: Nobody wants to spend their time doing this.
3. Versioning: A minor change in a software dependency version ie. 2.1 to 2.2 often breaks the bindings.

This allows one to very quickly add new capabilities to R as a system. The new capabilities then facilitate rapid prototyping and novel types of data analysis, using R as a 'glue language'. Temple Lang describes further applications and motivations in the paper in the RCIndex package repository [12].

3.1 Related Work

Many packages on CRAN simply wrap existing C/C++ libraries. When done thoughtfully this can be very useful. Ie. the programmer maps the C/C++ API into idiomatic data structures and calls them in the new language, in this case R. The intent with the code generation is to *augment* rather than *replace* such libraries.

The approach in RCIndex differs from existing software like SWIG [2] and Rcpp [5] because it hooks directly into the clang compiler rather than processing the text itself. This means we have a code analysis and generation tool that will be more robust and consistent with behavior of a compiler.

3.2 Challenges

This will require me to spend significant time building expertise in C/C++.

Memory management and garbage collection is potentially difficult. A C routine might allocate memory that R knows nothing about. Then how is that memory protected and freed? A possible solution to this problem is to recursively look through the body of the code itself, which RCIndex doesn't yet do.

3.3 Applications

The mature C++ computer vision library openCV [4] is an example of capabilities that we would like to access from within R. It's possible to directly write computer vision algorithms in R, but this is a huge amount of duplicated work if a mature library already exists.

Another application is quickly providing R bindings to cutting edge specialized machine learning code like Professor Cho-Jui Hsieh's Hogwild++ [16].

4 Modern Data Frames Implementation

Implement data frames with built in support for parallel operations in a modern systems language like Rust or Go

Much of the functionality for data frames is similar across data analysis libraries and also databases. Examples include filtering, basic math, grouping operations and summary statistics. Can common core structures and algorithms be factored out into a common library and implementation?

By starting with a totally new language this gives an opportunity to build data frames in a way that benefits from advances in compiler and parallel processing technologies.

4.1 Related Work

Python's Pandas package is currently considering a large overhaul by rewriting the internals in C++ for a 2.0 version. The proposed changes include copy on write semantics and increased support for missing data, which will make behavior very similar to base R's `data.frame`. It would be nice to have this implementation available in R.

The ArrayFire library provides a backend for high performance arrays, and it may be possible to build on this. The Julia language could also be a candidate for an improved data frame.

4.2 Challenges

Making robust, performant data frames is a very large task. The language interfaces will have to be excellent for this to be usable.

Handling missing or `NA` values within the context of existing vectorized math libraries may be difficult.

4.3 Applications

A portable, efficient data frame implementation removed from the specifics of any particular language could be used in applications that use data frames, for example those currently using SQLite.

5 Chunking Approaches to Large Data

Speculative evaluation of different chunked computation schemes

A well known technique for parallel computation is to split data into chunks and compute on each chunk simultaneously. This approach is useful on large data. Multiplying large block matrices is an example of this.

Many existing software packages for parallel computation requires one to specify the chunk size. Examples include ddR [8] and partools [9] in R and dask in Python. Yet chunk size strongly impacts performance. If it's too small then there will be excessive overhead. If it's too large then it won't fit into memory. In both cases performance will suffer. But how much will it suffer?

Matloff presents compelling reasons to use a chunk based approach for statistical applications [10].

5.1 Related Work

Python's dask library infers appropriate block size for reading CSV files.

R's bigmemory package uses C++ and a memory mapped file which allows shared memory access. [7] It's a tool for representing large data sets, and the package vignette mentions that chunking can and should be used together when possible.

5.2 Challenges

For a given set of hardware and a given task graph in a given computational model there may be an optimal chunking scheme, or a range of chunk sizes with similar performance. We'd like to automatically figure that out so the user doesn't have to specify. It may be possible to gather data be speculatively running parts of the computation and then using this as the inputs to a mathematical optimization problem.

I'd expect that a dynamic language calling vectorized C code has some set of performance characteristics, while standalone C++ code is quite different. Extension: What chunking is optimal for GPU?

Allowing `rechunk()` operations in the middle of the task graph adds much more flexibility and complexity. But it might be useful.

Need to be a little careful because the chunking scheme defines the dask task graph. Then the semantics are more general than the dask graph. It's really similar to optimization.

5.3 Applications

Google's TensorFlow can express large computational graphs on n dimensional arrays (tensors), mostly for machine learning.

Apache Spark does chunking transparently to the user, and it's possible to tune it for performance reasons.

6 Parallelism through pure functions

Programs using R's `Apply` family of functions together with pure functions[1] can be easily parallelized [3]. Then why not provide a mechanism for users to declare and/or detect pure functions?

Consider an R script with the following form:

[1] Pure functions are those that depend only on inputs and have no side effects [15].

```
data = read.table("data.txt")

y = f(data)
fdata = lapply(g, data)
...

write.table(result, "result.txt")
```

`read.table()` and `write.table()` are not pure functions, but suppose that everything in between consists of a mix of `Apply` and pure functions.

Then the user could mark their functions as `f = pfunc(function(x) {...})`. More ambitiously the syntax could be extended as `f = pfunction(x) {...}`. This can then be easily analyzed and parallelized.

Consider a function `f(x) = function(x) x + y`. We could provide a mechanism to automatically make functions pure as in `purify(f) = function(x, .y = y) x + .y`.

6.1 Related Work

This idea came from Böhringer, who describes a relevant algorithm and R implementation in a 2013 paper "Dynamic Parallelization of R Functions" [3].

6.2 Challenges

User settings could affect results in subtle ways, for example a user may change the option for `stringsAsFactors` in their local session and this change does not propagate to the parallel version.

Functions using random sampling require careful handling, because the RNG stream exists in global state. Using a parallel technique like L'Ecuyer's may be a way to overcome this.

6.3 Applications

Removing one of the primary sources of difficulty with parallelism will make it easier to write code that scales to larger data sets and different architectures.

7 What I've learned

Research is about developing new ideas. Engineering is about making things that will be adopted because they work and are useful. Ideally these two activities overlap, but most

research topics don't lead directly to useful software. The line between research and engineering / implementation is sometimes not clear.

As an inexperienced graduate student it's difficult to know whether a topic or approach is new or merits further investigation. Even as I write this I removed a section after reading an article released on Arxiv two days prior doing something very similar [11]. However, the article was encouraging because it motivates the other topics. Motivating applications are extremely important for developing technology and research ideas.

The choices in technology today are vast and often intimidating. Each has their pros and cons. At this point I think it's time for me to get my hands dirty in learning one (such as C++) at a deeper level, since I'll need to have deeper knowledge and experience to push the envelope with research. Classes in school can be a distraction in this regard because they're interesting so I spend time learning that subject when maybe that time would have been better spent on research topics.

References

[1] Taylor Arnold, Michael Kane, and Simon Urbanek. iotools: High-performance i/o tools for r. *arXiv preprint arXiv:1510.00041*, 2015.

[2] David M. Beazley. Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, pages 15–15, Berkeley, CA, USA, 1996. USENIX Association.

[3] Stefan Böhringer. Dynamic parallelization of r functions. *The R Journal*, 5(2):88–96, 2013.

[4] G. Bradski. opencv. *Dr. Dobb's Journal of Software Tools*, 2000.

[5] Dirk Eddelbuettel, Romain Francois, JJ Allaire, Kevin Ushey, Qiang Kou, Douglas Bates, and John Chambers. *Rcpp: Seamless R and C++ Integration*, 2016. R package version 0.12.5.

[6] Joseph Guinness. Permutation methods for sharpening gaussian process approximations. *arXiv preprint arXiv:1609.05372*, 2016.

[7] Michael J Kane and John W Emerson. bigmemory: Manage massive matrices with shared memory and memory-mapped files. *R package version*, 4(3), 2010.

[8] Edward Ma, Indrajit Roy, and Michael Lawrence. *ddR: Distributed Data Structures in R*, 2016. R package version 0.1.3.

[9] Norm Matloff, with contributions by Alex Rumbaugh, and Hadley Wickham. *partools: Tools for the 'Parallel' Package*, 2016. R package version 1.1.5.

[10] Norman Matloff. Software alchemy: turning complex statistical computations into embarrassingly-parallel ones. *arXiv preprint arXiv:1409.5827*, 2014.

[11] Parmita Mehta, Sven Dorkenwald, Dongfang Zhao, Tomer Kaftan, Alvin Cheung, Magdalena Balazinska, Ariel Rokem, Andrew Connolly, Jacob Vanderplas, and Yusra Al-Sayyad. Comparative evaluation of big-data systems on scientific image analytics workloads, 2016.

[12] Duncan Temple Lang. *RCIndex: R Interface to the clang parser's C API*, 2016. R package version 0.3-0.

[13] Duncan Temple Lang. *RCodeGen: Functionality to help creating code for R, C/C++*, 2016. R package version 0.2-0.

[14] The HDF Group. Hierarchical Data Format, version 5, 1997-NNNN. /HDF5/.

[15] Wikipedia. Pure function — wikipedia, the free encyclopedia, 2016. [Online; accessed 28-November-2016].

[16] Huan Zhang, Cho-Jui Hsieh, and Venkatesh Akella. Hogwild++: A new mechanism for decentralized asynchronous stochastic gradient descent. ICDM, 2016.