Duncan asked for a draft / outline of my dissertation by the end of the summer.

These are the main claims / contributions I would like to make in my dissertation. Most of them don't fully hold as I've written; they require more work on my part.

I need to organize these claims into chapters and a coherent narrative. By first figuring out the most important ideas, I can then structure the rest of the outline to support these ideas.

TODO: Add more on data configuration.

0.1 Main Claim

The goal all along has been to make the following claim:

We have developed a system that takes serial R code and generates reasonably efficient parallel R code. It combines knowledge of the data with code analysis to produce code specific to the platform.

What are the restrictions / conditions on this? We can't handle any possible combination of code / data / platform. Can a generated solution do things any better than a hand written solution?

0.2 Claim

keywords: introduction, motivation

It's more efficient for a programmer to write one version of the code, and then have automated tools that can tailor the code to a particular platform and data configuration. This is obviously better. The question is how to do it? That's the whole theme of the dissertation.

- Writing parallel code is difficult, and usually highly specific to the platform.
- Ideas from compilers have applications to dynamic languages like R.

0.3 Claim

keywords: static code analysis, task graph, code comprehension

We can statically infer and represent the task graph implicit in R code.

- The task graph comes from determining which resources a (sub)expression uses and defines. Resources include R level variables, files, options, the current graphing device, and anything else that can be modified and used by an expression.
- The task graph represents the actual constraints for parallelism.
- The task graph is useful in other ways. It shows programmers what their code actually does. Matloff asked about plotting the task graph. It can be used to recompute results when code changes, similar to GNU Make.
- `CodeDepends` is related work that we build on. DTL: What else has been done in other languages?

Conditions: the code analysis doesn't extend into R's C level implementation, which means that if users change do something strange in a call to C code, say setting an option, then we cannot detect it.

0.4 Claim

keywords: optimization, optimal scheduling, algebra

We've precisely described the problem of minimizing the time it takes to perform a GROUP BY calculation for distributed data, including data movement.

- Exact solutions using mixed integer programming are computationally intractable.
- We have evaluated two heuristics that address the GROUP BY scheduling problem.
- The difference in run time between an optimal solution and a pretty good solution is relatively small, so these heuristics are practical.

0.5 Claim

keywords: code analysis, map reduce, algebra

Vectorized and reducible functions in R are the primitives that allow us to build more sophisticated parallel programs. Am I just saying that lots of R code falls into the Map-Reduce paradigm?

- We have algebraically described how long it takes to solve map reduce style problems, including data movement costs.

0.6 Claim

keywords: code analysis, code transformations

Code analysis can detect various idioms in the code and map them into more efficient implementations.

- Examining a script holistically in context allows us to do qualitatively different kinds of code transformations that aren't possible if we strictly follow R's interpreted model.
- For example, we can check which columns of a data frame are used in the code, and generate efficient code that only reads those columns, saving memory.
- Best case scenario we can take code that won't even run because the data size exceeds memory, and filter the data at the source. Thinking about streaming, and not exceeding available memory.
- We can find for loops that can be parallel, and transform them into parallel versions.

0.7 Claim

keywords: software, implementation, use cases

Our implementation of these ideas, `makeParallel`, works on class of real examples, namely computations that involve a GROUP BY.

- It does something completely different for the PEMS example based on the organization of the data.
- It estimates how long the program will take for different parameters of the data / platform. This allows users to reason about what they're doing, for example, there is no point in moving the computation to a cluster if it won't be any faster.

0.8 Claim

keywords: software, implementation, framework, extensibility

Users can experiment with other task scheduling algorithms and code generators using our system.

- We designed `makeParallel` to be customizable and extensible.

Cite something to make the bibliography appear [R Core Team, 2018].

Bibliography

[R Core Team, 2018] R Core Team (2018). R language reference.