This is an abstract for the first chapter of my thesis. This morning I revised it to improve the motivation for a general audience, and to improve the style. I'm not confident the revised version is ideal.

Original

In this chapter we describe how to statically analyze R code to construct a data structure representing the ideal parallelism implicit in the code. It shows the ideal parallelism by capturing all the constraints on execution and identifying map type operations. It represents several types of data dependencies between statements, so it is a task graph. It goes beyond a task graph in a couple ways. First, it identifies vectorized and data parallel statements, which allows us to fuse implicit loops to improve data locality during parallel execution. Second, it represents process dependencies essential for the correct execution of code, for example plotting commands which must run in order on the same process.

Revised

Programs express parallelism through data and tasks. Data parallel code executes the *same* instructions across many data simultaneously. Task parallel code executes two or more *different* instructions simultaneously. Our ultimate goal is to create a more efficient parallel program from one that is not parallel, ie. serial. We need to know where a serial program can use data or task parallelism.

The first chapter describes how to infer parallelism implicit in R code. A program statically analyzes the code to construct a data structure that represents constraints on expression ordering. The data structure represents several types of data dependencies between statements, so it is a task graph that shows all possible ways to group code into tasks and run in parallel. It goes beyond a task graph in two ways. First, it identifies vectorized and apply style functions as data parallel. Second, it represents process dependencies essential for the correct execution of code, for example plotting commands which have no data depenedencies yet must run in order on the same process.