The Utilization of MapReduce for Efficient Protein Sequencing on HPC
Brian D. Knopp; Praveenkumar Kondikoppa; Seung-Jong Park, Ph.D
Center for Computation & Technology, Louisiana State University

Abstract
A growing number of applied biocomputation applications utilize parallelism in several different forms. Most approaches offering a high degree of control require an extensive rewrite of potentially large and complex software programs; we have developed an alternative.

In our approach, we deploy Apache Hadoop as a parallelism framework, which does not require modification of existing software. This allows parallelism to be implemented for tasks with many dependencies, but with few interdependencies. eThread, a protein structure prediction program, was our candidate for parallelization. We have created several scripts encapsulating independent algorithm and program functionality, which eThread depends on. These scripts are executed in parallel utilizing the MapReduce paradigm.

MapReduce and Hadoop
MapReduce is a distributed/cloud computing paradigm. Computation Jobs are broken into two main stages: mapping and reducing. Mappers receive plain text in \(<\text{key},\text{value}>\) format and process this text to produce output. This output is given to reducers, which combine the separate mapper results.

Hadoop is an Open Source implementation of MapReduce[4].

eThread - Streaming Execution
Hadoop is promising for large bioinformatics applications[5]. Our goal was to implement parallelism in an already existing application: eThread. eThread is a protein structure prediction algorithm[1]. It depends on the output of several other programs to predict ligand binding-points.

Funneling Execution
Goal: optimize the execution model. Much time is wasted using only a single compute core. Although eThread has many dependencies, it has few interdependencies. This is conducive to MapReduce utilization.

Implementation
HadoopStreaming is used as an interface, communicating through Unix pipes. Python scripts encapsulate functionality of logically independent programs and algorithms.

Future Work
Future research may focus upon the generalization of our funneling execution approach into an easily applicable framework. Such an implementation could be largely automated, with the user simply specifying necessary dependencies among a set of programs.

Alternatively, the newer Pregel[3] graph-processor is attractive for applications with much more complex (even dynamic) dependencies which are not easily abstracted to MapReduce.

Acknowledgements
Thanks to: My mentor, Seung-Jong Park, Ph.D. Praveenkumar Kondikoppa for daily advice and material review. Michal Brylinski, Ph.D for great assistance in setting up eThread. Daswanth Lingam for pre-implementation consultation.

Research supported by:

This material is based upon work supported by the National Science Foundation under award OCI-1005165 with additional support from the Center for Computation & Technology at Louisiana State University.

References