Lab 6: Data-Driven Futures and Phasers

Instructor: Vivek Sarkar, Co-Instructor: Mackale Joyner

Course Wiki: http://comp322.rice.edu

Staff Email: comp322-staff@mailman.rice.edu

Goals for this lab

- Experimentation with Data-Driven Futures with Abstract Metrics
- Experimentation with Phasers by measuring Real Performance on NOTS

Lab Projects

Today, we will start by using a Cholesky factorization example to learn about using Data Driven Futures. We will compute performance for this part of the lab using abstract metrics.

Next, we will revisit the Iterative Averaging example to implement a point-to-point synchronization variant using phasers. For this part, we will measure real performance on the NOTS cluster at Rice.

The Maven project for this lab is located in the following svn repository:

- `https://svn.rice.edu/r/comp322/turnin/S17/`*`NETID`*`/lab_6`

For instructions on checking out this repository through IntelliJ or through the command-line, please see the Lab 1 handout. The below instructions will assume that you have already checked out the lab_6 folder, and that you have imported it as a Maven Project if you are using IntelliJ.

1 Parallelization of Cholesky (using Data-Driven Tasks)

In linear algebra, the Cholesky factorization is a decomposition of a positive-definite matrix into the product of a lower triangular matrix and its conjugate transpose. This decomposition is useful to efficiently obtain numerical solutions, e.g., for a system of linear equations of the form $A \times x = b$.

We have provided a sequential version of Cholesky Factorization in the `CholeskyFactorization.java` and `CholeskyFactorizationSequential.java` files. Your goal will be to parallelize this computation using data-driven tasks and data-driven futures, and to evaluate your parallelization using abstract metrics. The fact that the sequential version uses a generic `dataStore` container can be leveraged to simplify this conversion.

Your assignment is to create a parallel version of `CholeskyFactorizationSequential.runComputation()` that implements the Cholesky Factorization algorithm by using data-driven tasks:

1. Write a parallel version in the `runComputation()` method of the `CholeskyFactorizationParallel.java` file using data-driven tasks with calls to `asyncAwait()`. The provided file has helpful hints to guide you in this process.
2. For abstract metrics, note that `CholeskyFactorizationParallel.java` is already set up with zero overhead for creating async tasks, as in Homework 1 (but not Homework 2).
3. Run the unit tests to verify whether your parallel computation achieves the required abstract metrics.
4. Record in `lab_6_written.txt` the best total work and the ideal parallelism relative to the sequential version.

2 One-Dimensional Iterative Averaging (with Phasers)

The code provided in `OneDimAveraging.java` performs the iterative averaging computation discussed in the lectures. This code performs a sequential version of the computation in method `runSequential()` and a chunked parallel version in `runChunkedBarrier()`. Iterative averaging is performed on a one-dimensional array of size `(n+2)` with elements 0 and `n+1` initialized to 0 and 1 respectively. The final value expected for each element i at convergence is $i/(n+1)$. However, we limit `iterations` to a constant value to prevent long execution times, so you may not see convergence if you examine the array elements in your lab exercise.

1. *Your assignment is to create a parallel version of `OneDimAveraging.runSequential()` that implements the same one-dimensional iterative averaging algorithm but using phasers instead of barriers.*
2. Update the `runChunkedPhaser()` method by using the approach discussed today in Lecture 15. In particular, pay attention to the phaser registration modes for your tasks. Also, you will need to use chunking to obtain good performance in your programs.
3. You can implement the parallel version of the one-dimensional iterative averaging algorithm locally on your laptop, but to complete the lab you must also run your project on NOTS to evaluate the performance of the different parallelization approaches. You may do so manually using the provided SLURM script, or by using the autograder. On NOTS, you should achieve ∼2x speedup relative to the provided sequential version.

Note that these tests may take some time to run due to the size of the datasets being used.

3 Turning in your lab work

For lab_6, you will need to turn in your work before leaving, as follows.

1. Show your work to an instructor or TA to get credit for this lab. In particular, the TAs will be interested in seeing your code for the phaser-based version.
2. Commit your work to your `lab_6` turnin folder. The only changes that must be committed are your modifications to `CholeskyFactorizationParallel.java` and `OneDimAveraging.java`. Check that all the work for today's lab is in your `lab_6` directory by opening

   `https://svn.rice.edu/r/comp322/turnin/S17/NETID/lab_6/`

   in your web browser and checking that your changes have appeared.