Abstract: We studied the performance and ease of implementation of a concurrent program written in a modern parallel programming language. We compared running times of two different concurrent implementations with a sequential implementation of a standard application. We found that, with minimal time and effort invested, a concurrent implementation of the program could be created that was faster and free of deadlocks and data-races.

The Program

In the image below, the program is still running. Note the progress bars on the lower right corner. Our pipeline was divided into four phases that were able to run concurrently.

The Study

- We wrote a face matching application that takes two large images, each composed of many smaller sub-images of individual faces, and correctly matches each face from one of the images to the other.
- We performed timing tests on three different implementations of this program:
  - frAsync -- uses primarily async constructs
  - frActors -- uses Actor objects
  - HJSeq -- a sequential implementation
- The language we chose was Habanero-Java (HJ)
  - HJ is an extension of the Java language developed at Rice University for the purposes of pedagogy and research.
  - HJ offers constructs and features that encourage experimentation.

Timing Tests and Results

We timed each implementation, varying both the number of sub-images and the size of each sub-image. We recorded the average time to complete over eight iterations for each problem size. Below is a graph displaying the results of timing tests for one of four sets of sizes.

The frActors implementation was consistently the fastest over all problem sizes, followed by frAsync, then HJSeq.

The learning overhead required to implement both the frActors and frAsync implementations was minimal, demonstrating that appreciable speedup can be expected while maintaining a high degree of safety.

Conclusion

Modern parallel programming languages have attained a maturity sufficient that an average programmer can create concurrent implementations of applications with a high degree of expectation in terms of performance improvement and code safety.

References


This research was supported by NSF REU Award 1321642.