1 Methods and Techniques

The bag of tasks implementation created for this report uses a *pull* protocol: Once a client starts, it first sends a request to the server for a new task. The server sends tasks in predefined sized *chunks*. It includes a kill signal for the client to exit in the chunk once it has run out of tasks to send.

This report will present an analysis of process speedup given various chunk sizes and processor counts. The times were measured from when the server process starts until it exits. All timings were measured using the functions defined in the *utilities.h* file on the Ptolemy cluster.

2 Analysis

A rudimentary but useful way for predicting parallel program speedup is Amdahl's Law. Predicting the exact percentage of the program that is parallelizable involves a lot of consideration. For example, the server has to perform relatively slow IO operations every time it sends a new chunk to a client, and it has to do more IO when it logs solutions it receives. The depth first searching used to find solutions is also inherently serial.

Considering these overheads, say only 70% of the program is truly parallelizable. In this case, $f = 0.7$ making the maximum possible speedup $S = 3$.

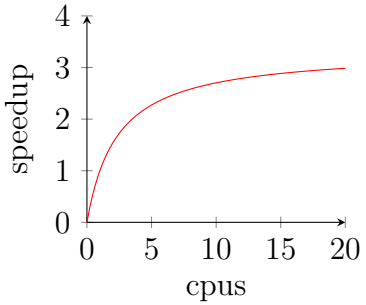

Figure 1: Program speedup prediction using Amdahl's Law

2.1 Chunk Size Consideration

Another overhead that will effect the speedup of the program is the communication overhead. This implementation does not use non-blocking communication functions, so time spent waiting for a communication to finish is idle time.

This communication cost can be modeled using the following formula:

$$t_{comm} = t_s + mt_w \tag{1}$$

Since the communication startup time t_s is much larger than the latency t_w, it is theoretically more efficient to send larger chunks to avoid repeatedly paying this cost. However, by sending large chunks to the clients, there is the chance some clients get a chunk with many difficult tasks causing them to receive less tasks than other clients in the long run. How much this impacts performance is measured in the results.

3 Results

Four different chunk sizes were tested: 1, 5, 10, 20.

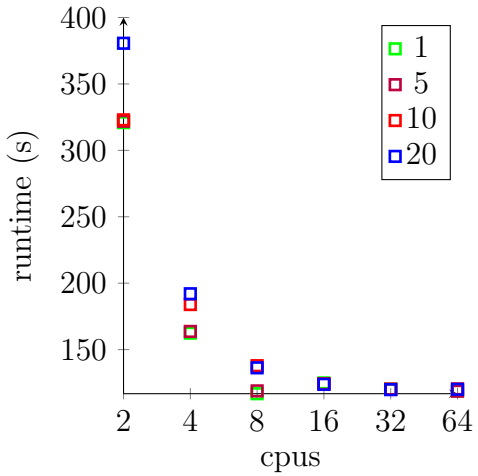

Figure 2: Recorded program runtimes for the different chunk sizes

The runtime recorded for the original, non-parallel version of the program was 421 seconds. The recorded runtimes for chunk size 1 were used to compute speedup.

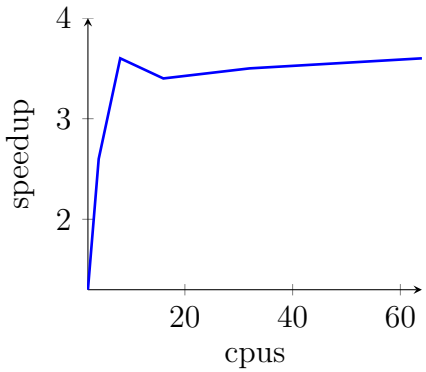

Figure 3: Measured program speedup

4 Synthesis

Compared to the predicted speedup of 3, the result of 4 was off by a fare margin. The predicted overhead must have been less impactful than predicted.

The effect of different sized chunks was only noticeable at low cpu counts. After 8 cpus, the chunk size had little effect on the runtime of the program. Size 20 chunks seemed to have the worst performance regardless, which seems to coincide with the prediction that chunks getting too many hard tasks causes a slowdown.