Alex Sanciangco

704050064

Lab 4 Report

How did you divide the task?

I implemented a waterfall – like system, where the task was broken up into blocks of 1000 sequential elements of a row.

How do you distribute and schedule your subtasks?

Each process is attributed to a different value of ‘i’ in the algorithm’s main loop, meaning that each process is responsible for a row of the score matrix. The loop starts by receiving the data from the previous line, meaning that on the initial iterations, the majority of the processes are waiting to receive data. The first process traverses the row, and after it processes a block’s worth of elements (coded to be 1000), it sends that block to the next process and continues processing the row. Then, while that initial process is continuing, the receiving process begins processing the next row in parallel. This creates a sort of cascade effect, resulting in high parallelization with moderate granularity.

What message are being passed between sub-tasks?

Sub-tasks are sending each other messages that contain blocks of 1000 elements of the score matrix. Lower order rows send them down to higher order rows, since the dynamic programming aspect of the algorithm has dependencies structured in the same way.

What challenges did you meet and how did you solve them?

As I played around with the block size, certain size blocks caused the algorithm to bottleneck and/or deadlock in random places inexplicably. However, a block size of 1000 never had these issues and was therefore hard-coded into the algorithm.

Performance Results:

	Run 1 (sec)	Run 2 (sec)	Run 3 (sec)	Run 4 (sec)
Sequential	5.355728	5.446359	5.377328	5.371646
1 core	6.465296	6.425743	6.407707	6.418666
2 cores	4.666992	4.540532	4.662695	4.659111
4 cores	2.408214	2.411035	2.457294	2.376141
8 cores	1.797251	1.897018	1.796286	1.899192
16 cores	1.850123	1.864166	1.860762	1.896265

It is interesting to note that running at 14 cores, I was able to get a runtime of as low as ~1.3 seconds.

Analysis

It would appear that after 14 cores, the overhead of adding more to the system out-weighs the benefits of the parallel code, resulting again in a slowdown. I was able to achieve a max speed up of ~3x at 8 cores, but ~4x at 14 cores.