Project 5 Technical Analysis – Page 1 of 3

Student Name: Filippo Zallocco

CRN: 71429

**Report Question 1: Add Your Performance Statistics Here**

Sample:

|  |  |
| --- | --- |
| Run # | Results |
| 1 | SEQUENTIAL EXECUTION TIME: 55896489  PARALLEL EXECUTION TIME: 1988112  SPEEDUP: 21.09  NUMBER OF PROCESSORS USED: 25 |

**Your results**

|  |  |
| --- | --- |
| Run # | Results |
| 1) for n = 12 | SEQUENTIAL EXECUTION TIME: 936154  PARALLEL EXECUTION TIME: 96210  SPEEDUP: 9.73  NUMBER OF PROCESSORS USED: 14 |
| 2) for n = 24 | SEQUENTIAL EXECUTION TIME: 14210200  PARALLEL EXECUTION TIME: 714458  SPEEDUP: 19.89  NUMBER OF PROCESSORS USED: 26 |
| 3) for n = 36 | SEQUENTIAL EXECUTION TIME: 27476120  PARALLEL EXECUTION TIME: 1060614  SPEEDUP: 25.91  NUMBER OF PROCESSORS USED: 34 |

Project 5 Technical Analysis – Page 2 of 3

Student Name:

CRN:

**Report Question 2: Your Observations on Performance Across the runs**

What do you notice is different across the runs? Why do you think this is? Use the textbook on pages 204-212 to defend your answer.

|  |
| --- |
| Having tested the Parallel Jacobi Relaxation problem with Matrix sizes, 12, 24, and 32, I noticed that relative speedups and number of processors dramatically increased for the first two runs while they hardly climbed in the last test. From the n = 12 test’s report, we may appreciate that the performance gain is close to 10 units of time and the number of processors used is 14. With the n = 24 test’s report, we learn that the relative speedup went up to nearly 20 units of time and the number of processors grew to 26, each doubling from the previous test. Upon analyzing the difference in performance between the third and the second tests, I noticed that solving the same problem with a matrix of size 32, both speedup and number of processors increment by 30% from the second test.  We can attribute the performance gain across the tests to the local synchronization technique of using Barriers. This feature reduces the time required to ensure the correct execution of parallel algorithm by limiting process synchronization to a given node and its neighbors, thus improving the overall performance. Finally, the use of efficient synchronization in the aggregation function allows for smoother data processing by row of a few arrays of nodes whose values are then evaluated against the maxchange variable. If the value evaluates to true, then the function proceeds to the next row whereas if the value is set to false, then all the processes will be regarded as false, thus extending the operation by an additional iteration. Reducing the global synchronization to a few short arrays, ultimately benefits the parallel execution of the Jacobi Relaxation algorithm including when the size of its matrix grows from 12 to 24 and 32. |

Project 5 Technical Analysis – Page 2 of 3

Student Name:

CRN:

**Report Question 3: How did your program perform compared to the benchmark?**

Benchmark for n = 32 with tolerance = .003 is 28 X Speedup. What is your speedup at this setting? How does your performance compare? Why do you think this is?

|  |
| --- |
| Following the above instructions, my speedup was 26.92, which is a 4% performance slowdown compared to the benchmark. Performing a comparative study with a matrix of size 32 with tolerance .10, the simulator shows that solving the Parallel Jacobi Relaxation with a tolerance of .003 yields a performance improvement of almost 4%, thus leading me to assume that reducing the tolerance from .10 to .003 may not significantly impact the parallel execution of Jacobi Relaxation with local barrier and aggregation functions. |