Luke Johnson

Lab 3

CS 555

**Synthesis Questions**

**1.      Chart the speedup, cost, and efficiency varying the sort algorithm, data set size, and number of processors.**

These charts were made in relation to P=1, N = 160,000, where the sorting function was the C qsort function.

**2.      Estimate the computation to communication ratio for a data set size of 220 and 4 processors. You can use the Big Oh estimates for processing size and can leave the answer in terms of a power of two.**

I don’t even know where to get started…

Assuming we’re talking about the Odd/Even sort, then there is the initial scatter, which costs 220/4, so 218. Then in each iteration, of which there are P/2, each neighbor pair will communicate. They will send over their array of size 218. There are a total of 3 of these communications each iteration, for a total of 6 communications between neighbor pairs. Finally, there is a gather to get the data back to the master node, which again costs 218. So our grand total for communication is 8\*218 multiplied by whatever time constant there is for data transmission.

The computation portion of this algorithm is actually pretty simple. For each neighbor pair communication that a processor is involved in, it has to do 218 comparisons. This makes for a total of 6\*218 comparisons.

The bottom line here is that there is more communication than computation.

**3.      Which law (Amdahl’s or Gustafson’s) is more appropriate in this case?**

Amdahl’s

**4.      Assume that on a single core system, there is insufficient memory to hold the entire array of doubles. How could this be overcome? What modifications in the parallel version might lead to super-linear speedup should we combine a bunch of these single core systems into a cluster?**

The problem could be partitioned so that the memory only had to hold a portion of the array at once.

**5.      What does the speed up results tell you about the effectiveness of parallelizing this application?**

This is a good application of parallel processing, but communication costs start to go up as the number of cores go up.

**6.      Did any of your runs demonstrated super-linear speedup? If so, explain why? How would speedup be impacted if we ran the entire experiment using bubble sort instead of quick sort?**

The two-core runs achieved super-linear speedup. This is possibly because there is less communication overhead required than in runs with more processors. If bubble sort had been used, then the speedup would have been even greater, as each processor has to do a much smaller piece of an n-squared algorithm than the single-core option.

**7.      Suppose the processors simply used a collective gather rather than the tree structured merge. How do you think this would affect speedup? Explain your answer.**

This would actually reduce the amount of speedup achieved, as the receiving processor would have to do much more work to merge the data.

**8.      Do you think the parallel bitonic sorting algorithm would improve speedup compared to the odd-even approach? Justify your answer.**

It seems that it would be slower than the odd/even sort, simply because the odd/even approach requires P/2 loops, while the bitonic algorithm will always require slightly more than that.

**9.      What is a major advantage of partitioning approach compared to divide and conquer? Which sorting algorithm fits this model?**

You don’t end up with a single processor needing to merge all of the data at the end. The partitioning approach only requires a gather once all computations are done.

**10.  How might a shared memory implementation differ from message passing? What are some advantages and disadvantages?**

For the odd/even sort, there would necessarily be synchronization points between iterations, to prevent the shared data being corrupted between neighbors. This means there is more idle time in the loops, but it also does not require passing arrays of data between neighbors.