MISSING DATA

When running merge sort with an array size of 2^28, I ran into “out of memory” issues on Grace even though I was using the maximum amount of memory per node. However, this behavior is expected as merge sort makes local copies of the data when sorting. Therefore, increasing the array size causes a nonlinear increase in memory consumption.

Furthermore, when attempting to run jobs with 1024 processes, I frequently ran into hydra network errors. Because I only had data available for 1024 processes for some configurations, I excluded the 1024 process data points from my plots and subsequent analysis.

ANALYSIS

Array Size:

As the array size increases, the complexity and workload also increase, both for data initialization, computation, and communication. Even with parallelization, larger arrays require more time for sorting simply because there are more elements. However, for smaller array sizes such as 2^16, the computational load is relatively low, so the overhead associated with parallelism dominates the runtime of the algorithm. This results in less noticeable performance gains, as shown in the speedup graphs of the main function for 2^16. However, when we get to the larger array sizes, such as 2^26, parallel processing significantly reduces runtime due to the distribution of workload. This is again seen in the speedup plots for 2^26. However, we do get diminishing returns after a certain number of processes as the communication begins to dominate the runtime. Therefore, it is important to scale the number of processes in accordance with the input size of the array.

Input Types:

For merge sort, the input type does not play a huge role in computation time because regardless of how sorted the input array is, merge swap will always run in O(n log n) time. However, how sorted the array is does play a role in the number of comparisons that are made when merging. Therefore, even though it runs in O(n log n), some inputs may take slightly less or more time than others. This is best seen in the computation plots for sorted vs. reverse sorted runtimes. Sorted inputs generally take a little less time than reverse sorted inputs as fewer comparisons need to be made. Random inputs generally take around the average amount of time to complete and the one percent perturbed inputs take slightly longer than the sorted inputs to complete. Because merging also happens in the communication step between different processes after every process sorts their local array, this observation is also evident in the communication plots.

Number of Processes:

In general, the computation time decreases as more processes are added because the workload is divided into more pieces. However, it does get to a point of diminishing returns, especially when dealing with smaller array sizes. Additionally, for a given input size, when the number of processes gets too large, the communication cost dominates the benefit gained from parallelization. This is because when we have more processes, more merging is required . For example, with 2 processes only one merge step is required for the two local arrays, but with 512 processes, there would be 9 levels of merging. Furthermore, because the amount of

communication necessary increases, the latency or runtime of the communication step also increases.