
Homework #2

(Due: April 6)

Task 1. [250 Points] Distributed Sample Sort with Multithreaded Merge Sort inside Each Compute Node.

This task asks you to implement a distributed sample sort algorithm that runs a multithreaded merge sort algorithm inside each compute node. You will be sorting double precision floating point numbers. We will assume that all numbers in the input array are distinct. For simplicity generate the input array as follows. First initialize the array by storing the number i (as a double precision float) at location i of the array, where $1 \leq i \leq n$. Then choose two random locations of the array and swap their contents, and repeat this swaping step n times. Use this permuted array as input to your sorting implementations. When timing the implementations do not include the time needed to generate the input array.

- (a) [30 Points] Implement and optimize the shared-memory parallel merge sort algorithm (PAR-MERGE-SORT-SM) shown in Figure 1 which uses the standard serial algorithm (MERGE) for merging two sorted sequences.

Optimize for the base case size m_1 , that is, empirically find a value of m_1 that gives you the best or close to the best performance for large values of n , and use that value of m_1 in all subsequent runs of the algorithm. For this task it is OK to check only powers of 2 for the potential best value of m_1 . Use all cores during this base case optimize phase.

Let N be the largest power of 2 such that this algorithm can sort N numbers in less than 5 minutes when run on a single processing core. Find the value of N .

- (b) [30 Points] Implement and optimize the shared-memory parallel merge sort algorithm (PAR-MERGE-SORT-PM) of Figure 2 which uses a parallel merge algorithm (PAR-MERGE).

First optimize for the base case size m_2 of PAR-MERGE, and then use that value of m_2 when you optimize for the base case size m_3 of PAR-MERGE-SORT-PM.

- (c) [30 Points] Plot the running times of PAR-MERGE-SORT-SM and PAR-MERGE-SORT-PM for 10 equispaced points between $n = 1000$ and $n = N$ when run on all cores of the compute node you are using. Compare and explain the results.

- (d) [30 Points] Generate Cilkview scalability plots for PAR-MERGE-SORT-SM and PAR-MERGE-SORT-PM assuming $n = N$. Compare and explain the results.

- (e) [50 Points] Implement the distributed-memory parallel sample sort algorithm (DISTRIBUTED-SAMPLE-SORT) shown in Figure 3. Make two implementations. In one implementation use the merge sort algorithm (PAR-MERGE-SORT-SM) you implemented in part (a) for the local sorts in steps 2 and 5, and call this DISTRIBUTED-SAMPLE-SORT-SM. In another implementation use the merge sort algorithm (PAR-MERGE-SORT-PM) you implemented in part (b) and

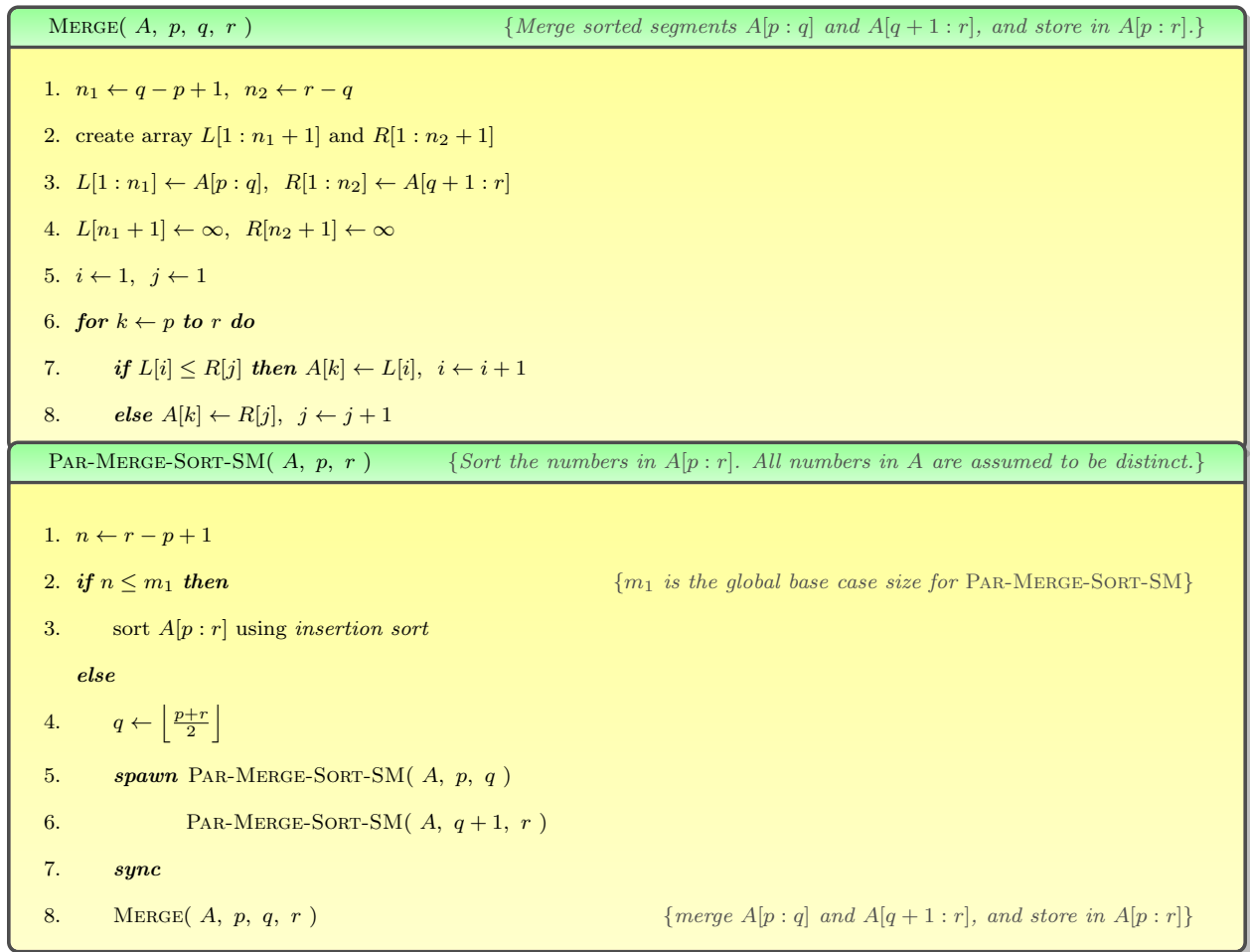


Figure 1: Parallel merge sort with serial merge.

call this DISTRIBUTED-SAMPLE-SORT-PM. In DISTRIBUTED-SAMPLE-SORT-SM, execute the two recursive calls to PAR-MERGE-SORT-SM in steps 5–7 of PAR-MERGE-SORT-SM serially (i.e., remove the **spawn** and **sync** keyword) instead of calling them in parallel. But do not change anything inside the PAR-MERGE-SORT-PM implementation used in DISTRIBUTED-SAMPLE-SORT-PM.

- (f) [40 Points] Plot the running times of DISTRIBUTED-SAMPLE-SORT-SM and DISTRIBUTED-SAMPLE-SORT-PM for 10 equispaced points between $n = 1000$ and $n = N$ when run on 5 compute nodes. For DISTRIBUTED-SAMPLE-SORT-PM use all cores inside each compute node. Compare and explain the results. Also plot the speedups w.r.t. PAR-MERGE-SORT-SM run on a single core of a single compute node.
- (g) [40 Points] Repeat part (f), but without including the time needed for the initial distribution of input (step 1 of DISTRIBUTED-SAMPLE-SORT) and the final collection of output (step 6) in the running time.

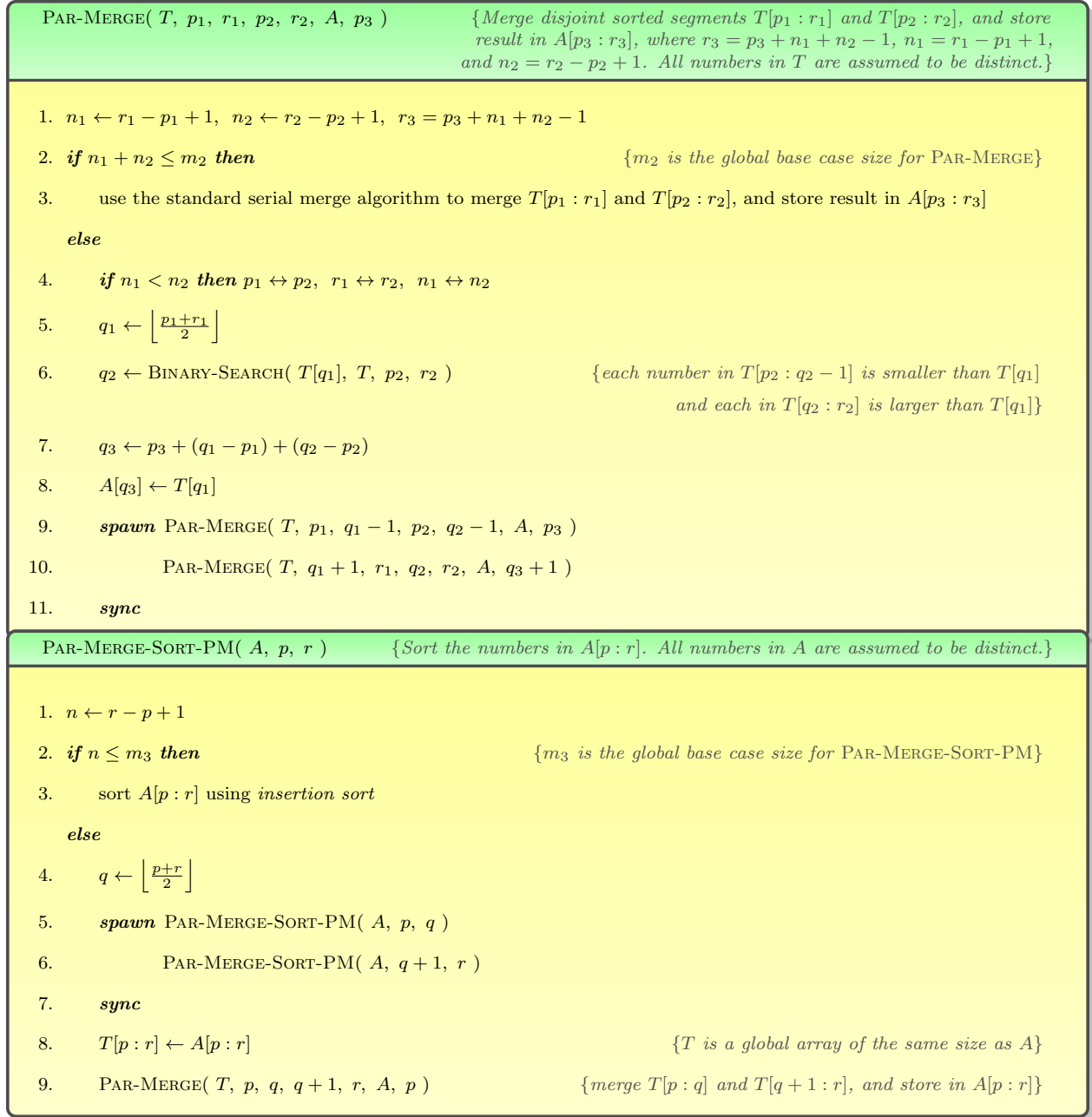


Figure 2: Parallel merge sort with parallel merge.

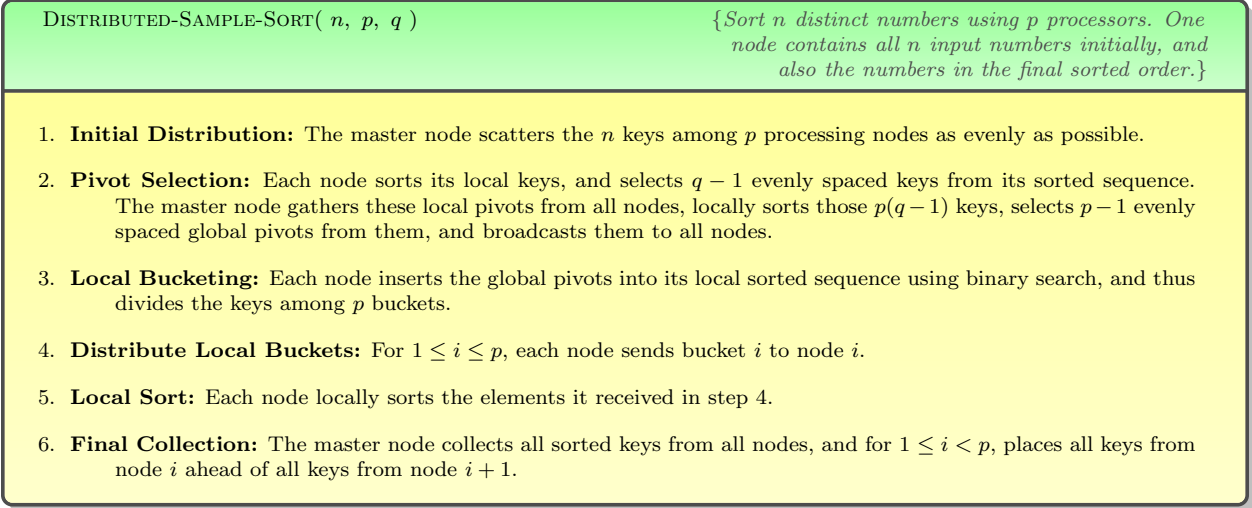


Figure 3: Distributed sample sort.

APPENDIX 1: What to Turn in

One compressed archive file (e.g., zip, tar.gz) containing the following items.

- Source code, makefiles and job scripts.
- A PDF document containing all answers and plots.

APPENDIX 2: Things to Remember

- **Please never run anything that takes more than a minute or uses multiple cores on TACC login nodes.** TACC policy strictly prohibits such usage. They reserve the right to suspend your account if you do so. All runs must be submitted as jobs to compute nodes (even when you use Cilkview or PAPI).
- Please store all data in your work folder (\$WORK), and not in your home folder (\$HOME).
- When measuring running times please exclude the time needed for reading the input and writing the output. Measure only the time needed by the algorithm.