Lab 3

Problem Statement

The goal of this lab is to learn to program in distributed memory model. You are required to implement a Parallel (using MPI) version of the well know Quicksort algorithm.

Parallel Quick Sort

Quicksort has some natural concurrency. The low list and high list can sort themselves concurrently.

Quicksort can be parallelized in a variety of ways. However, instead of considering a naive parallel formulation, you are expected to implement a variant of Quicksort known as **Hyperquicksort** or **Hypersort**.

Hyperquicksort, invented by *Wagar[1]*, assists in efficient movement of values from process to process. Once every process sorts its local list, we will use a pivot value to divide the numbers into two groups: the "lower half" and the "upper half." *Because the list of elements on each process is sorted, the process responsible for supplying the pivot can use the median of its list as the pivot value*. This value is far more likely to be close to the true median of the entire unsorted list than the value of an arbitrarily chosen list element.

The process choosing the pivot broadcasts it to the other processes. Each process uses the pivot to divide its elements into a "low list" of values less than or equal to the pivot and a "high list" of values greater than the pivot. Every process in the upper half swaps its low list for a high list provided by a partner process in the lower half.

After the swap, each process has a sorted sub-list it retained and a sorted sub-list it received from a partner. It merges the two lists it is responsible for so that the elements it controls are sorted. It is important that processes end this phase with sorted lists, because when the algorithm recourses, two processes will need to choose the median elements of their lists as pivots.

After log p such split-and-merge steps, the original hypercube of p processes has been divided into log p single-process hypercubes. Since the processes repeatedly merged lists to keep their

local values sorted throughout the divide-and-swap steps, there is no need for them to call quicksort at the end of the algorithm. Figure I shows the overall idea of hyperquicksort, Figure II gives an example of hyperquicksort in action.

Hyperquicksort assumes the number of processes is a power of 2.

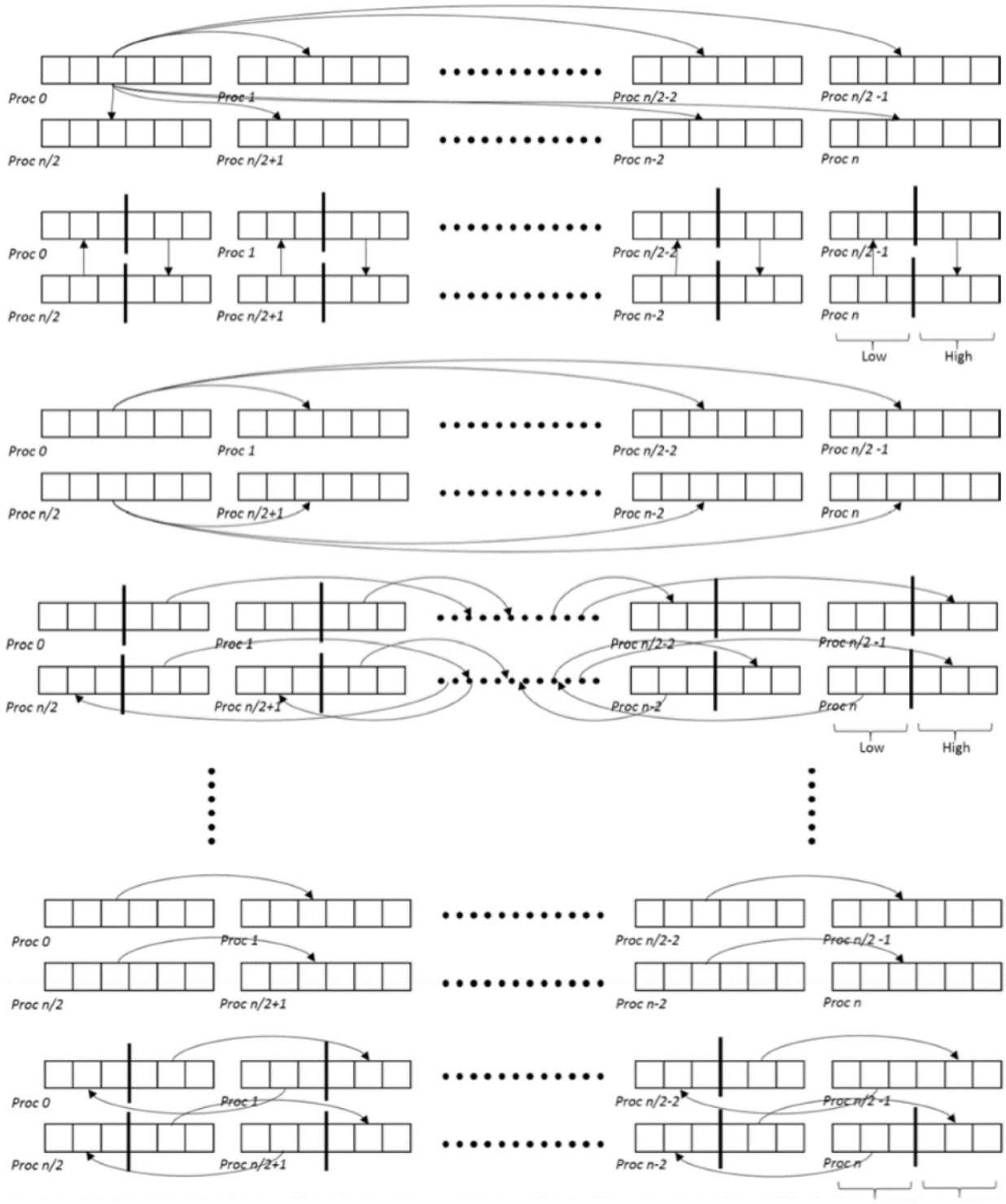

Figure I Hyperquicksort

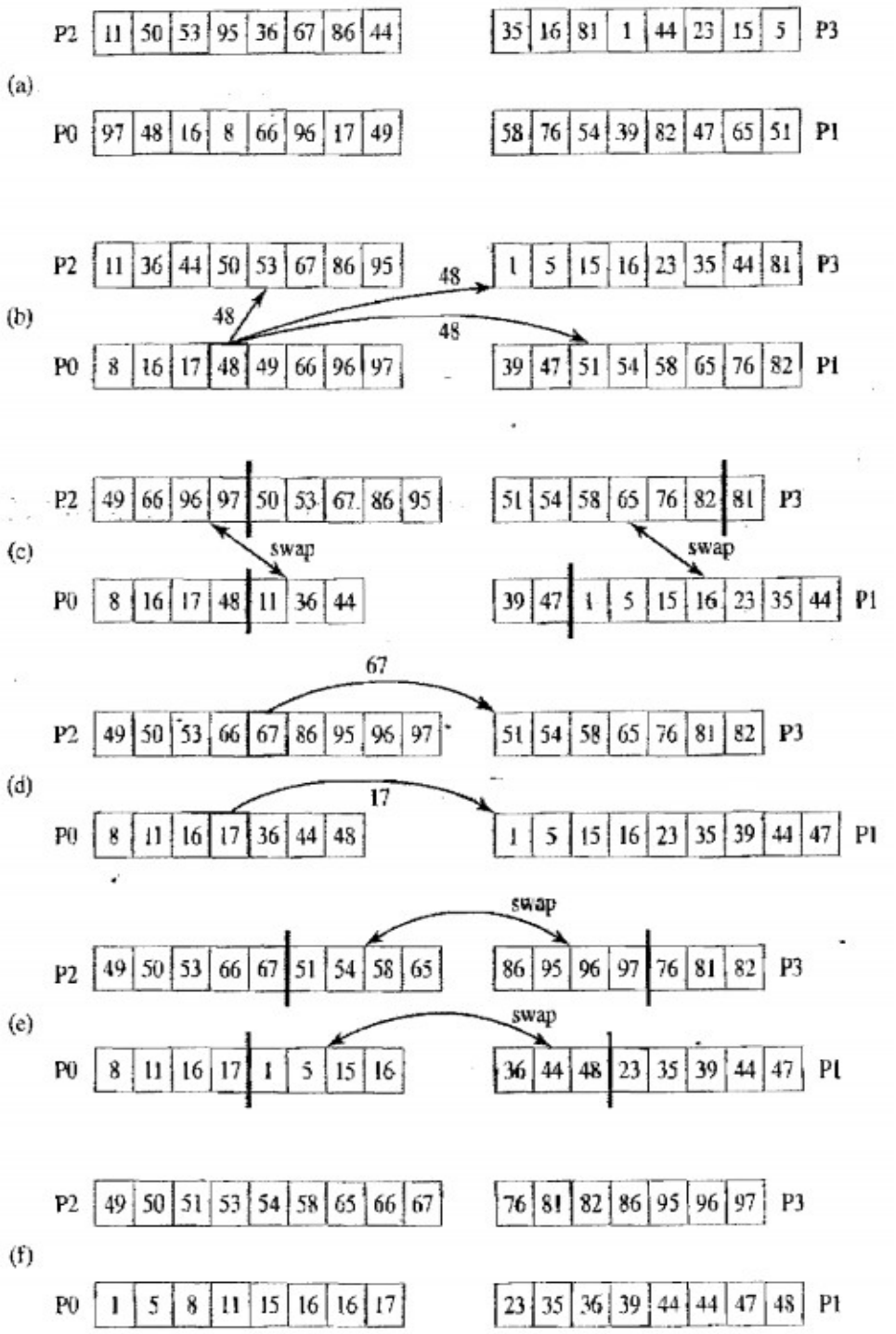

Figure II Hyperquicksort example

Statement of work

1. Implement the Hyperquicksort algorithm using MPI.
    a. Input: list of unsorted signed integers.
    b. Output: list of sorted signed integers
2. You should test sorting for payload sizes upto 2^{26} items and up to 32 nodes, with one processor each.
3. Along with implementation, prepare a lab report (as .txt file) by clearly explaining your design decisions, parallelization strategy, load-balancing strategy. Make sure that explanations are bulleted and are not long paragraphs. Keep the lab report within 800 words.
4. Demonstrate the speedup through graphs:
    a. Payload size vs Time (for fixed no of nodes)
    b. No of nodes vs Time (for fixed payload size)
5. Zip the implementation and report together and name the zip with your entry no.

Testing and Running on HPC

To test for larger no of nodes, you can use the High Performance Computing at IITD (http://supercomputing.iitd.ac.in/).

***Word of caution**:* Test your code for correctness on your local systems before switching to testing on HPC. Use HPC only for testing scalability. Be judicious and responsible when using HPC resources.

To use HPC:

1. Request for an account on HPC
    a. Use your LDAP credentials
    b. Enter
        Faculty supervisor (uid) : svs
        Requested Expiry date of access: 1 May 2017
2. ssh into HPC using your LDAP credentials

```
ssh <entry_no>@hpc.iitd.ac.in
```

3. Copy your files to your HPC client node sing scp

```
scp ./<file> <entry_no>@hpc.iitd.ac.in:
```

4. Write PBS script to submit a job. Follow http://supercomputing.iitd.ac.in/?pbs
    a. Keep a wall time of 30 min.
5. Submit using qsub
    ```
    qsub <script> -q low
    ```
6. Check status of your submitted job using qstat
    ```
    qstat <job_id>
    ```

Grading Policy

1.	Correct execution for at least 2^{10} values:	30 Marks
2.	Scalability upto 2^{20} values:	20 Marks
3.	Scalability upto 2^{26} values:	10 Marks
4.	Efficient speedup:	15 Marks
5.	Clarity in report and discussion of your solutions:	20 Marks
6.	Code modularity, code comments, coding style:	5 Marks

Total 100 Marks

References

[1] Wagar, Bruce, "Hyperquicksort: A fast sorting algorithm for hypercubes:' In Hypercube Multiprocessors 1987, pages 292-299. Philadelphia, PA: SIAM, 1981

[2] Parallel Computing, Theory and Practice, M.J.Quinn, McGraw Hill Publications, 2002.