



Parallel Programming

Peyman Shobeiri

ID: 40327586

Fall 2024




Table of Contents

QUESTION 3	3
QUESTION 4	3
(A)	3
(B)	4
QUESTION 5	4
(A)	4
(B)	6
REFERENCES:	7

Question 3

Calculate the theoretical run-time (T_p) of the parallel quick sort algorithm in Q.1 assuming a physical hypercube topology. You may assume the best-case scenario only, i.e., a balanced pivot selection at each step of the algorithm.

To calculate the total parallel runtime for the quick sort algorithm for N numbers using a P -processor hypercube, we first need to distribute N/P tasks across the processors. Then, one of the processors selects a pivot, and each processor partitions its portion of the list into two parts based on this pivot. This step is followed by an exchange between adjacent pairs of processors.

The selection of the pivot is crucial in this process. If a good pivot is chosen, the load will remain balanced after the exchange. Conversely, if a poor pivot is selected, the exchange can result in load imbalances. In this question, we will only focus on the best-case scenario (as mentioned in the question), where each half is perfectly balanced after every exchange.

Since the total number of such exchanges is $\log(P)$, the best-case parallel time for the hypercube-based solution, assuming parallel I/O, can be summarized as follows:

$$T_p = T_{\text{local_sort}} + T_{\text{array_split}}$$

$$T_p = O\left(\frac{N}{P} \log \frac{N}{P}\right) + O\left(\frac{N}{P} \log P\right) + O(\log^2 P)$$

So, if $P \ll N$ then we have:

$$T_p = O\left(\frac{N}{P} \log \frac{N}{P}\right)$$

Question 4

(a) Is the PSRS algorithm discussed in Q.2. balanced?

In general, this PSRS algorithm is considered balanced because it divides the work evenly among all processors in most steps. For instance, in Step 1, each processor sorts N/P elements, ensuring equal computational effort. In Step 2, every processor selects $P - 1$ regular samples, performing the same amount of work. However, Step 3 requires a single processor (P_0) to gather and sort P^2 regular samples, this overhead is minimal because P is much smaller than N ($P \ll N$), making the sorting of P^2 elements negligible compared to the overall workload. Steps 4 and 5 involve all processors partitioning their data based on the pivots and merging the received partitions, respectively, with each processor handling similar amounts of data, thus maintaining balance.

Therefore, despite the slight imbalance in Step 3, where one processor performs pivot selection, PSRS is considered a balanced algorithm. This little extra overhead at that step would not have noticeable effects on the overall performance, especially for N being much larger than P and this tiny imbalance is tolerable since it has little effect on its efficiency.

(b) **Calculate the theoretical run-time (T_p) of the PSRS algorithm, based on the similar theoretical analyses we did in class.**

The T_p of the PSRS algorithm involves several key steps. First, each process sorts its local sub-list of $\lceil N/P \rceil$ items using sequential quicksort, which takes $O(N/P \log N/P)$ time. Then, each process selects P samples from its sorted list, which takes $O(P)$, and sends them to process P_0 , costing $O(\log P)$ for communication. Process P_0 then sorts the collected P^2 samples and selects $(P-1)$ pivots, taking $O(P^2 \log P^2)$ time, followed by broadcasting the pivots to all processes in $O(\log P)$ time. After receiving the pivots, each process partitions its local list into P partitions, costing $O(N/P)$, and then redistributes the partitions to other processes using all-to-all communication, which takes $O(P \log P)$. Finally, each process merges its P partitions, with a complexity of $O(N/P \log P)$. Thus, the total theoretical run-time T_p is the sum of these components:

$$T_p = T_{\text{local sort}} + T_{\text{pivot selection}} + T_{\text{partitioning}} + T_{\text{communication}}$$

$$T_p = O\left(\frac{N}{P} \log\left(\frac{N}{P}\right)\right) + O(P^2 \log P) + O\left(\frac{N}{P} \log P\right) + O(P \log P)$$

Question 5

(a) **Redesign the algorithm in Q.1., that uses recursive halving, for a 2-D mesh topology. Assume a $p \times p$ mesh comprising of p processors, where processors are numbered from $P(0,0)$ to $P(p-1, p-1)$. You are required to write the pseudo-code only, in a format similar to the pseudo code used in the textbook (e.g., for Q.1.)**

in this code Each processor handles a segment of the data (B). The mesh comprises p processors organized into rows and columns, with each processor identified by a unique row (row_id) and column (column_id) label.

The sorting process occurs in two phases: row-wise sorting and column-wise sorting. During each phase, a designated processor selects a pivot value (x) and broadcasts it to the other processors in the group. Each processor then partitions its data (B) into two parts: B1, which contains values less than or equal to the pivot, and B2, which contains values greater than the pivot. Depending on the value of the i -th bit of its row or column label, each processor exchanges either B1 or B2 with a corresponding processor in the same column or row but with a different i -th bit. This effectively redistributes the data for further improvement.

This process repeats for $1/2 \log p$ iterations in both the row and column phases, gradually reducing the size of the data subsets and refining each processor's segment. The parameters in this algorithm include the row and column labels (row_id and column_id), which determine the communication partner in each iteration, and the pivot (x), which guides the data partitioning. At

the end of both phases, each processor holds a refined portion of the data and then applies sequential quicksort to its local segment to complete the sorting.

```

procedure MESH_QUICKSORT (B, n)    // B is the n/p element subsequence assigned to the process.
begin
  row_id := row label;
  column_id := column label
  for i:=1/2 log p to 1 do
    begin
      x := pivot; // one designated process selects and broadcasts pivot to all other processes
      Partition B into B1 and B2 such that  $B1 \leq x < B2$ ;
      if row_id's i_th bit is 0 then
        Send B2 to the corresponding process at same column and row with row_id that is different at i_th bit;
        C := subsequently receive from corresponding process at same column and row with row_id that is
different at i_th bit;
        B := B1 U C ;
      endif
    else
      Send B1 to the corresponding process at same column and row with row_id that is different at i_th bit;
      C := subsequently receive from corresponding process at same column and row with row_id that is
different at i_th bit;
      B := B2 U C;
    Endelse
  endfor
  for i:=1/2 log p to 1 do
    begin
      x := pivot; // one designated process selects and broadcasts pivot to all other processes
      partition B into B1 and B2 such that  $B1 \leq x < B2$ ;
      if column_id's i_th bit is 0 then
        Send B2 to the corresponding process at same row and column with column_id that is different at i_th bit;
        C := subsequently receive from corresponding process at same row and column with column_id that is
different at i_th bit;
        B := B1 U C;
      endif
    else
      Send B1 to the corresponding process at same row and column with column_id that is different at i_th bit;
      C := subsequently receive from corresponding process at same row and column with column_id that is
different at i_th bit;
      B := B2 U C;
    endelse
  endfor
  sort B using sequential quicksort;
end MESH_QUICKSORT

```

```

1 procedure MESH_QUICKSORT (B, n) // B is the n/p element subsequence assigned to the process.
2 begin
3
4   row_id := row label;
5   column_id := column label
6
7   for i:=1/2 log p to 1 do
8     begin
9       x := pivot; // one designated process selects and broadcasts pivot to all other processes
10      Partition B into B1 and B2 such that B1 ≤ x < B2;
11
12      if row_id's i_th bit is 0 then
13        Send B2 to the corresponding process at same column and row with row_id that is different at i_th bit;
14        C := subsequently receive from corresponding process at same column and row with row_id that is different at i_th bit;
15        B := B1 U C;
16      endif
17
18      else
19        Send B1 to the corresponding process at same column and row with row_id that is different at i_th bit;
20        C := subsequently receive from corresponding process at same column and row with row_id that is different at i_th bit;
21        B := B2 U C;
22      Endelse
23    endfor
24
25    for i:=1/2 log p to 1 do
26      begin
27
28        x := pivot; // one designated process selects and broadcasts pivot to all other processes
29        partition B into B1 and B2 such that B1 ≤ x < B2;
30
31        if column_id's i_th bit is 0 then
32          Send B2 to the corresponding process at same row and column with column_id that is different at i_th bit;
33          C := subsequently receive from corresponding process at same row and column with column_id that is different at i_th bit;
34          B := B1 U C;
35        endif
36
37        else
38          Send B1 to the corresponding process at same row and column with column_id that is different at i_th bit;
39          C := subsequently receive from corresponding process at same row and column with column_id that is different at i_th bit;
40          B := B2 U C;
41        endelse
42      endfor
43
44      sort B using sequential quicksort;
45    end MESH_QUICKSORT

```

(b) **Calculate the theoretical run-time (T_p) of the algorithm.**

In this analysis, we consider the best-case scenario with parallel input/output operations. The mesh network is organized so that processors can communicate efficiently within subgroups defined by rows and columns. In total, there are $\log p$ rounds—comprising $1/2 \log p$ rounds for the rows and $1/2 \log p$ rounds for the columns—during which a balanced pivot is selected and broadcast to all processors within the subgroup. The selection and broadcasting of the pivot in each round take $\Theta(\log p)$ time, resulting in a total time of $\Theta((\log p)^2)$ for these operations.

During each round, processors compare their local elements (each holding n/p elements) with the pivot and partition them into two groups: those that are less than or equal to the pivot, and those that are greater. To ensure proper partitioning across the mesh, elements are exchanged between paired processors. Both the comparison and exchange operations take $\Theta(n/p)$ time per

round, leading to a total time of $\Theta(n/p \log p)$ for all rounds. After all rounds of comparison and exchange, each processor still holds n/p elements due to the balanced nature of the pivot selection. Sorting these local elements using a sequential quicksort algorithm takes $\Theta(n/p \log(n/p))$ time in the best case.

Combining all these components, the total parallel time T_p for the quicksort algorithm on a mesh network is expressed as:

$$T_p = T_{\text{local_sort}} + T_{\text{array_split}}$$

$$T_p = \Theta\left(\frac{N}{p} \log \frac{N}{p}\right) + \Theta\left(\frac{N}{p} \log P\right) + \Theta(\log^2 P)$$

References:

[1]: Grama et al, 2003, Introduction to parallel computing second edition