CENG 478

Assignment 3

Deadline: May 10, 23:59

Full Name: Yasin Fatih ALPUL
Id Number: 2098739

Answer 1

Using the analysis in the course textbook *Introduction to Parallel Computing*, the communication part of the expression should be identical. The difference should be from the multiplication of submatrices.

Assuming p processes are used and the matrices are divided to submatrices of size $\frac{n}{\sqrt{p}}$ each and there will be $\sqrt{p}$ rows and columns in each edge. The algorithm will require two all-to-all broadcast operations each consisting of $\sqrt{p}$ concurrent broadcasts in all rows and columns. Since submatrices contain $\frac{n^2}{p}$ elements, the total communication time is $2(t_s log(\sqrt{p})+t_w\frac{n^2}{p}(\sqrt{p}-1))$, which is approximately $t_s logp + 2t_w\frac{n^2}{\sqrt{p}}$.

For the matrix multiplication part, each process will multiply $\sqrt{p}$ times (by the algorithm) matrices of size $\frac{n}{\sqrt{p}}$, which will require $\sqrt{p} \times \left(\frac{n}{\sqrt{p}}\right)^{2.81}$ using the *Strassen's Algorithm.*

If we add these two expressions, we get the following expression for the parallel run time:

$$\sqrt{p} \times \left(\frac{n}{\sqrt{p}}\right)^{2.81} + t_s logp + 2t_w\frac{n^2}{\sqrt{p}}$$

If we simplify a bit we get,

$$\frac{n^{2.81}}{p^{0.905}} + t_s logp + 2t_w\frac{n^2}{\sqrt{p}}$$

To compute the cost optimality, we multiply that expression by p to get:

$$n^{2.81}p^{0.095} + pt_s logp + 2t_w n^2\sqrt{p}$$

We see that it is asymptotically greater than the sequential algorithm ($\Theta(n^{2.81})$) and hence not cost optimal.

Answer 2

We will use *proof by induction* to show the mathematical relation. Firstly, we will show that a 2×2 matrix is indeed takes $4(n-1) = 4$ steps. This is the basis step. Then, we will assume that the $4(n-1)$ formula is correct for a matrix of size $n \times n$ and use this relation to show it is $4((n+1)-1) = 4n$ for a matrix of size $(n+1) \times (n+1)$.

The below figure shows the steps for $n = 2$ similar to the figure in the textbook.

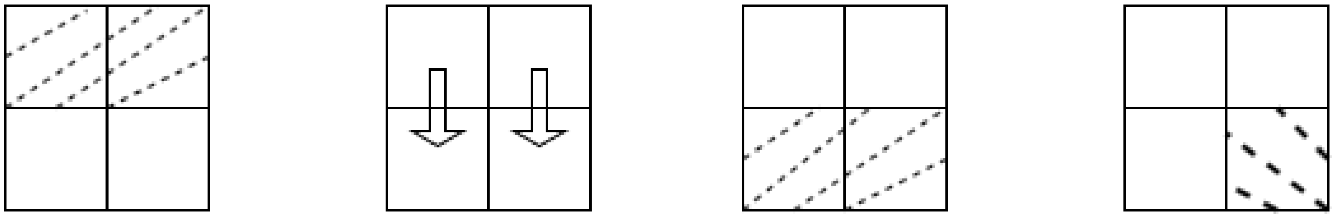

Figure 1: Steps for 2x2 matrix

As the formula has also shown, there are a total of 4 steps. Now, we will show that each increment of the size of the matrix by one corresponds to an increment by 4 in the steps of pipelined Gaussian elimination. This value is equal to the difference $4((n+1)-1) - 4(n-1) = 4$

Now, let's go backwards from this point i.e the point where we are at the first step of the Gaussian elimination of a matrix of size $n \times n$. Any operation can be started only when the row is not used by the previous process anymore. So the step before is where the previous process is using $n+1$ columns of that row i.e using it for normalization (dividing all the row elements by the leading entry). And the step before it is a communication step which is where the content of that row is propagated to the next row. This is because the computation can only be started after the row is propagated otherwise the next row cannot see the original values of the previous row. And the step before it is also a communication step where the row is propagated from the upper row to the current row. Similarly, the step before it is a normalization step where the first row is being normalized. Thus there are a total of 4 steps for an increment of matrix size by 1. The figure below illustrates the steps for a 3×3 matrix. Notice that the lower 4 steps are identical to the 2×2 one.

The figure in the textbook can be seen for the 8 additional steps for a 5×5 matrix.

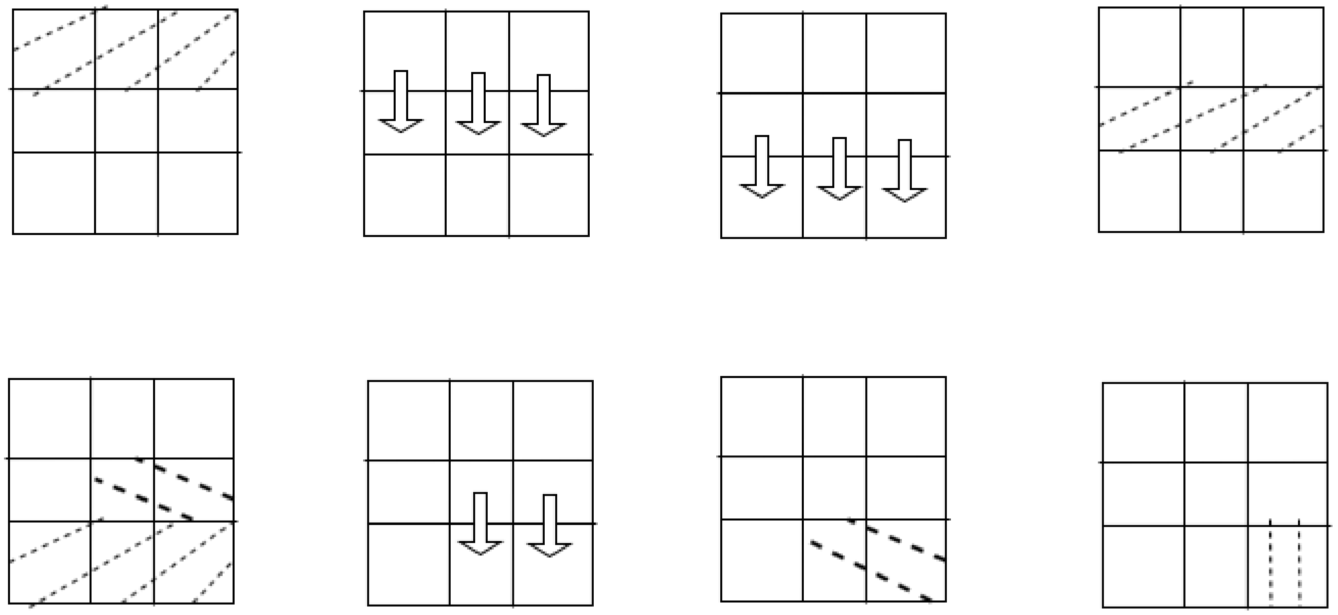

Figure 2: Steps for 3x3 matrix

Hence, we showed the relation of $4(n-1)$ by induction.

Answer 3

To output the shortest paths, we need to add another array in addition to the $l[]$ array denoting the minimum costs. This array will be the same size as $l[]$ and we can denote it as $p[]$ for it will show the parent node for each of the nodes. At the end of the algorithm, we trace back from any given node u to starting node s. For example, we find $n_1 = p[u]$ and then $n_2 = p[n_1]$ etc until we find $s = p[n_i]$. The path $n_1, n_2 \dots n_i$ will be the shortest path. This parent array will be updated when the distance is updated.

The complexity of the Dijkstra's single source shortest path algorithm is $O(n^2)$. However, it becomes $O(V + ElogV)$ with a priority queue data structure if used for determining the vertex with the minimum edge (line 9 of the Algorithm 10.2 from textbook).

The addition of another array $p[]$ does only affect the performance as a constant factor and therefore the overall complexity for run time does not change.

The parallel formulation is also very similar to *Prim's Algorithm* as the textbook said.

Communication part takes $\Theta(logp)$ time because of one-to-all broadcast and all-to-one reduction operations (takes $(t_s + t_w)logp$ time) for each iteration. Computation is $\Theta\left(\frac{n}{p}\right)$ for each iteration because each process computes only n/p part of the $p[]$ and $l[]$ arrays. Multiplying it by the number of iterations we get the expression below.

$$T_p = \underbrace{\Theta\left(\frac{n^2}{p}\right)}_{\text{computation}} + \underbrace{\Theta(nlogp)}_{\text{communication}}$$

After the algorithm ends, the costs can be communicated with a all-to-one reduction and then using the $p[]$ array we can find the shortest path to any given node from the starting node s in $O(n)$ time.

Answer 4

In all cases, the run time of the algorithm has two parts, namely, computation and communication. Moreover, the computation part is the same in all of them since we are using 2D block mapping. According to the Floyd's algorithm in the textbook, each process is assigned $\frac{n^2}{p}$ elements of the $D^{(k)}$ matrix. Therefore, the time to compute the corresponding $D^{(k)}$ values is $\Theta\left(\frac{n^2}{p}\right)$. Since the algorithm will have a total of n iterations, the run time of the computation part is $\Theta\left(\frac{n^3}{p}\right)$.

The algorithm will require two broadcast operations, one along the row and another one along the column at each iteration. The message being broadcasted is of size $\Theta\left(\frac{n}{\sqrt{p}}\right)$ because the process will broadcast the segment which the k^{th} row or column resides. On a mesh with store-and-forward routing, these broadcasts will take $\Theta(n)$ time (we multiply by $\sqrt{p}$ according to section 4.1.2 from textbook). Hence, the total runtime for 2D mest using store-and-forward routing becomes:

$$T_p = \underbrace{\Theta\left(\frac{n^3}{p}\right)}_{\text{computation}} + \underbrace{\Theta(n^2)}_{\text{communication}}$$

For the 2D mesh using cut-through routing, the runtime becomes similar to the formulation in the textbook. Since communication between any two points takes the same amount of time in cut-through routing, a one-to-all broadcast operation simply takes $logp$ times the time it takes to transfer between two nodes. Therefore, assuming the computation part still has the same runtime and we are again transferring a message of size $\Theta\left(\frac{n}{\sqrt{p}}\right)$, we have the following runtime for n iterations:

$$T_p = \underbrace{\Theta\left(\frac{n^3}{p}\right)}_{\text{computation}} + \underbrace{\Theta\left(\frac{n^2}{\sqrt{p}} logp\right)}_{\text{communication}}$$

For the case of p-process hypercube, we can formulate the runtime as the following. Since a hypercube has a recursive structure, we can use recursive doubling algorithm. This approach would not give different results for separate routing algorithms. Therefore, we are transmitting our message *logp* times in the hypercube. Then, the runtime becomes:

$$T_p = \underbrace{\Theta\left(\frac{n^3}{p}\right)}_{\text{computation}} + \underbrace{\Theta\left(\frac{n^2}{\sqrt{p}} logp\right)}_{\text{communication}}$$

The above formulations were for runtime of the parallel algorithm. For speedup and efficiency, we add some more calculations to them. Speedup is defined as $\frac{T_s}{T_p}$ and efficiency is defined as speedup divided by p. Since T_s is $\Theta(n^3)$ for all of them (the sequential Floyd's algorithm), we have the following speedup formulations, in the order we gave the runtimes.

$$\text{Speedup for 2d mesh store-and-forward} = \frac{\Theta(n^3)}{\Theta\left(\frac{n^3}{p}\right) + \Theta(n^2)}$$

$$\text{Speedup for 2d mesh cut-through} = \frac{\Theta(n^3)}{\Theta\left(\frac{n^3}{p}\right) + \Theta\left(\frac{n^2}{\sqrt{p}} logp\right)}$$

$$\text{Speedup for hypercube} = \frac{\Theta(n^3)}{\Theta\left(\frac{n^3}{p}\right) + \Theta\left(\frac{n^2}{\sqrt{p}} logp\right)}$$

For the efficiency, we divide each of them by p and do the algebraic operations to simplify the equations as in the textbook, we get:

$$\text{Efficiency for 2d mesh store-and-forward} = \frac{1}{1 + \Theta\left(\frac{p}{n}\right)}$$

$$\text{Efficiency for 2d mesh cut-through} = \frac{1}{1 + \Theta\left(\frac{\sqrt{p} logp}{n}\right)}$$

$$\text{Efficiency for hypercube} = \frac{1}{1 + \Theta\left(\frac{\sqrt{p} logp}{n}\right)}$$