Concordia University
Gina Cody School of Engineering and Computer Science
Parallel Programming
Name: Nellybett Irahola
ID: 40079991

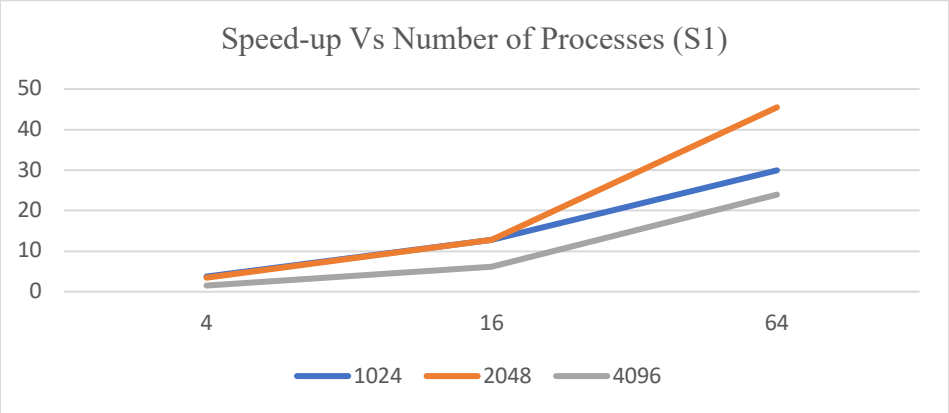
Assignment 3

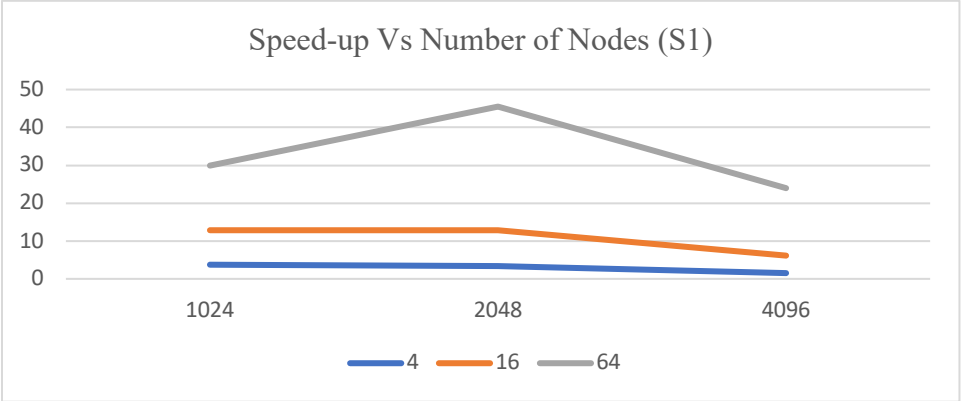
Question 1

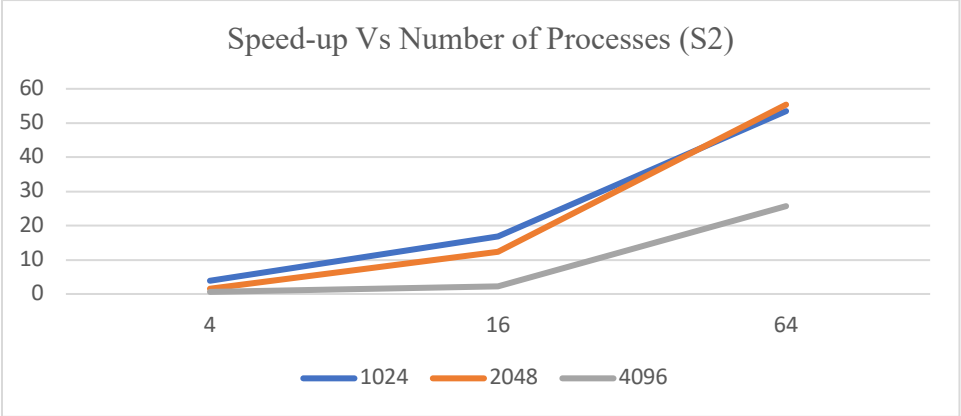
Floyd's shortest path algorithm was implemented in parallel and sequentially. Two parallel implementations were made. Both of them using 2-D block mapping into $\sqrt{p} * \sqrt{p}$ subblocks. The graph was represented using a matrix were "n" represent the number of nodes. The first implementation used broadcasting as a communication strategy and the second pipelines. For the implementation it was assumed that the number of nodes is divisible by $\sqrt{p}$ so each process manages perfect squares of data.

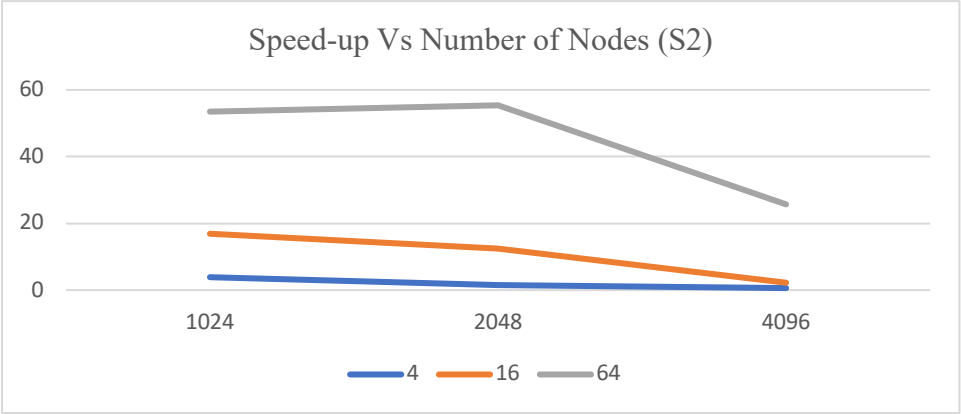
To compile the sequential implementation this command was used "gcc -o sequential_floyd.o sequential_floyd.c", and it can be run using "./sequential_floyd.o 1024" were the argument represents "n". The parallel implementation can be compiled using "mpicc -o parallel_floyd_s2.o parallel_floyd_s2.c" and it can be run using "mpirun -np 4 parallel_floyd_s2.o 1024".

Number of Nodes	Average Sequential Time	Processors	Parallel Time Broadcast (S1)	Speedup Broadcast (S1)	Parallel Time Pipeline (S2)	Speedup Pipeline (S2)
1024	22.46	64	0.75	29.95	0.42	53.48
		16	1.75	12.83	1.33	16.89
		4	6.01	3.74	5.81	3.87
2048	181.09	64	3.98	45.50	3.27	55.38
		16	14.1	12.84	14.58	12.42
		4	52.28	3.46	117.27	1.54
4096	1451.04	64	60.57	23.96	56.49	25.69
		16	236.38	6.14	652.36	2.22
		4	957.55	1.52	2363	0.61









The graphs show that for both strategies the speedup increases with the number of processes. They also demonstrate that the speedup decreases when the number of nodes increases for both strategies. In terms of the better strategy the pipeline shows better results when the number of processes is big. However, for 4 processes the broadcast strategy shows better performance in almost all the cases, this can be explained by the longitude of the pipeline (with 4 processes there are only two columns and two rows).

The results for the pipeline are expected to be always better than the ones obtained by the broadcast strategy, but this only happens when there is a smaller number of nodes. This contradiction can be explained by the way that mpi manages synchronous and asynchronous messages.

Question 2

Advantages:

In dynamic load balancing, when a processor runs out of work, it gets more work from another processor. Although the dynamic distribution of work results in communication overhead for work requests and work transfers, it reduces load imbalance among processors. In this case, if a processor finishes searching the search tree rooted at the node, it is reassigned to its parent node. If the parent has other child nodes still being unexplored, then this processor is allocated to one of them. Otherwise the processor is assigned to its parent. This guarantees that a processor is not idle as far as one of its parents has pendent nodes.

Another advantage is that there is no false termination since the process continues until the entire tree is totally searched. This guarantees that all the possible solutions are considered and evaluated.

Finally, the division of the processors equally among the child nodes is a good strategy in comparison with an unbalance assignment of processors since for a balance tree there will be an equal distribution of work between the processors.

Disadvantages:

Usually sub-trees close to the root are bigger than those close to the leaves, so assigning the pending nodes of the parent and not one of the processors with an explored node provides a bigger subtree the amount of work given is not that small. However, in an unbalanced tree if the parents don't have more nodes to explore all the processors that already finish will be idle even though some processors will be still working on their nodes (if the nodes in the bottom have big sub-trees, because the work assigned by the parent are not explored child nodes).

This strategy doesn't provide a cutoff depth to avoid sending very small amounts of work (nodes beyond a specified stack depth should not be given away).

Another disadvantage is the work-splitting strategy doesn't guarantee an even distribution of work between processors in an unbalanced tree.

There is not direct assignation of work. It is necessary to ask the parent if it has remaining nodes which increases communication overhead and reduces efficiency.

The process assignation requires to transverse the node until there is one process per node which implies that the processes are idle meanwhile.

Question 3

We assume that white is equivalent to green and black to red since those are the terms used in the book. The correctness of the algorithm can be proved by proving that:

- If the token received by the process that initiated the sequence is white, then all processors terminated their operation.
- If all the process terminated their operation, then the token received at the end is white.
- If the processor that initiated receive a white token and it is idle the algorithm terminates.

The first statement can be proved by contradiction. We assume that the processor that initiated the process receive a white token but not all the processors terminated their operation. This assumption leads to a contradiction since the only way that the token will only remain white is if it is passed only by processors that are white processors.

When it becomes idle, processor P_0 initiates termination detection by making itself white and sending a white token to processor P_1. The token transverse the ring processor by processor. If processor P_i has the token and P_i is idle, then it passes the token to P_{i+1}. If the processor that passes the token is white the token remains white if the processor that passes the token is black, then the token is passed as black and after passing the token the processor that sent it becomes white. A black token implies that the token must traverse the ring again.

A processor becomes black when that processor Pj sends work to Pi with j>i which transforms Pj to black, and when the token is passed by this processor the token becomes black, since there is work that is going to be done by Pi (it is necessary to transverse the ring again, since not all the processors are idle). When the token is changed to black, the token cannot be changed to white until it traverses the ring again. Because of that, if a white token is received it means that all processors terminated their operation and since P0 is idle and all the process terminated the algorithm terminates.

To prove the second part by contraposition it is necessary to prove that if the token received by the processor that initiated the operation is black then not all the processors terminated.

If the token that arrives is black it is because it was sent by a black processor. As explained before, a processor Pj becomes black when Pj sends work to Pi with j>i which transforms Pj to black. The token cannot change from black to white until it gets to the processor that initiated the process. That means that when the token is received as black by the processor that initiated the process it is because Pi is performing the work than Pj send it and it would be necessary to traverse the ring again to verify that Pi actually finished the work. Since Pi is not necessarily idle the it means that not all the processors terminated.

Finally, it is necessary to prove that in both terms of the double implication the algorithm will finished only if the processor that initiated the process is idle and it receives a white token. There are two cases, if the token never changes to black then all the processors passed the token when they were idle and the token was sent only by white processors. This means that P0 was idle and passed the white token to P1 and when it received again processor P0 will still be idle and it will receive a white token which means that the algorithm will terminate. In the second case the token was changed to black when it was passed by a black processor which means that the token will traverse the ring a second time until the state of all the processors is verified again and the token can be received as white.

Question 4

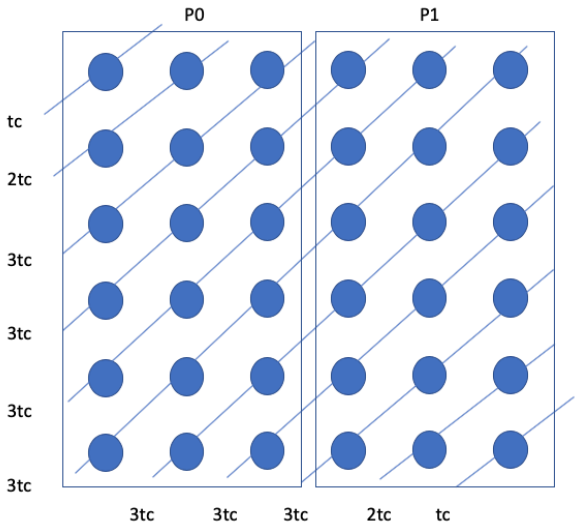
The sequential implementation of this dynamic programming formulation computes the values in a matrix in a row major order. Since there is a constant amount of computation at each entry in the table, the overall complexity of this algorithm is O(nm). Since the problem takes n=m, then the complexity is $O(n^2)$.

When computing the value of F [i, j], processing element P_{j-1} may need either the value of F [i - 1, j - 1] or the value of F [i, j - 1] from the processing element to its left. It takes time $t_s + t_w$ to communicate a single word from a neighboring processing element. To compute each entry in the table, a processing element needs a single value from its immediate neighbor, followed by the actual computation, which takes time t_c. The algorithm makes (2n - 1) diagonal sweeps (iterations) across the table. As the book explains, since each processing element computes a single entry on the diagonal, each iteration takes time $(t_s + t_w + t_c)$, which gives $Tp=(2n-1)(t_s + t_w + t_c)$. The efficiency presented by the book is:

$$E = \frac{n^2 t_c}{p(2n-1)(t_s + t_w + t_c)}$$

$$Emax = \frac{n^2 t_c}{p(2n-1) * t_c}$$

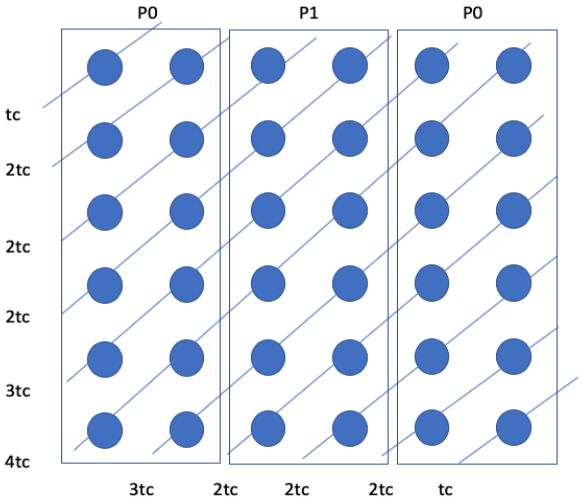
a. For the particular program n=6 which implies a matrix of 6x6 and p=2. Using block distribution (each partition contains a block of n/p consecutive rows) each process gets 3 columns which implies a depreciable communication overhead since the information is only transfer one time. The computation time increased since a diagonal can have n elements (with n processors the computational time is t_c) but with less processors that time is n/p * t_c for a diagonal with n elements. Each processor has to process the elements in its columns and only takes total advantage of parallelism when the diagonal is equal to n (and each processor evaluates n/p elements as previously mentioned). Calculating the efficiency for the specific case taking into consideration each diagonal, for n=6 the number of diagonals is equal to 11 (2n - 1), for maximum efficiency tw=0 and ts=0:



$$Emax = \frac{36\ t_c}{2 * (t_c + 2t_c + 3t_c + 3t_c + 3t_c + 3t_c + 3t_c + 3t_c + 3t_c + 2t_c + t_c)}$$

$$Emax = \frac{36\ t_c}{2 * 27\ t_c} = 0{,}67$$

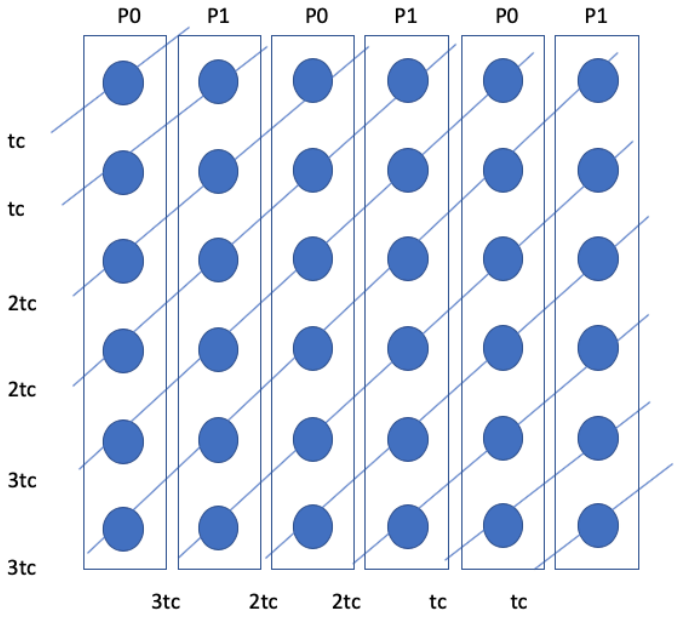
Using the block-cyclic distribution suggested in the question where processor P0 is mapped columns 0 and 1, P1 is mapped columns 2 and 3, and so on.



$$Emax = \frac{36\ t_c}{2 * (t_c + 2t_c + 2t_c + 2t_c + 3t_c + 4t_c + 3t_c + 2t_c + 2t_c + 2t_c + t_c)}$$

$$Emax = \frac{36\ t_c}{2 * 24\ t_c} = 0{,}75$$

b. Using block-cyclic distribution with a different mapping a better efficiency can be achieved since we can assign the first column for the P0 and the second P1 the third P0, etc. In that way in the first diagonal process P0 takes tc in the second processes P0 and P1 take tc since both calculate one element each one, in the third processes P0 and P1 take 2tc, and so on. The maximum efficiency for this approach assuming tw=0 and ts=0 is:



$$Emax = \frac{36\ t_c}{2 * (t_c + t_c + 2t_c + 2t_c + 3t_c + 3t_c + 3t_c + 2t_c + 2t_c + t_c + t_c)}$$

$$Emax = \frac{36\ t_c}{2 * 21\ t_c} = 0{,}86$$