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

Assignment 2

Question 1

The quicksort parallel algorithm was developed assuming as a parameter a number of processes compatible with a hypercube (1,2,4,8,16,32,64) and the number of elements ideally should be divisible by the number of processes. The algorithm uses the sequential quicksort as a sorting algorithm, it imports sequential_quicksort.h. The file quicksort.c contains the parallel implementation, it can be compile using "mpicc -o quicksort.o quicksort.c" and run by "mpirun -np 64 quicksort.o 8388608" where 64 is the number of processes and 8388608 the number of elements that the algorithm should order. These elements are generated randomly and divided between the processes using scatter.

The sequential algorithm can be compiled using "gcc -o sequential_quicksort.o sequential_quicksort.c" and run using "./sequential_quicksort.o 8388608". Since the pivot selection affects the results (worse time if the pivot produces a bad distribution) then the parallel and sequential time used for the speedup was an average.

Speedup= Sequential_Time / Parallel_Time

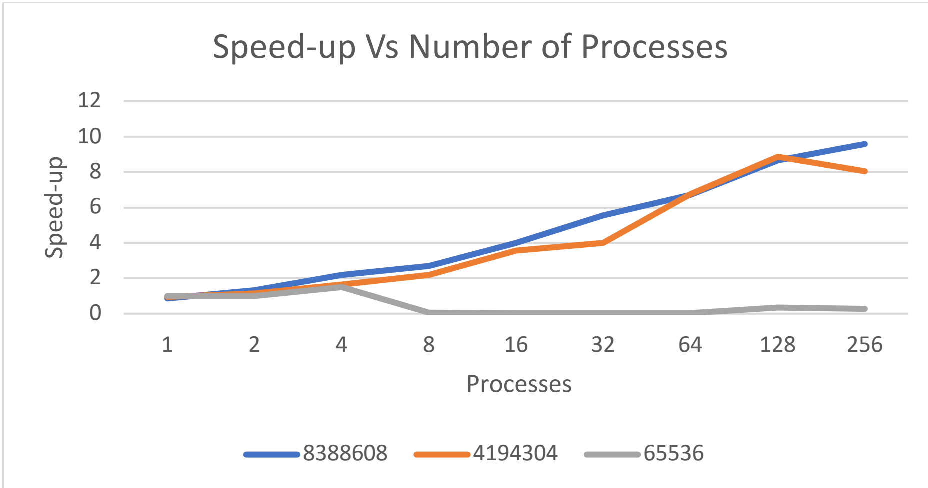
Number of Elements	Average Sequential Time (sec)	Processors	Parallel Time (sec)	Average Parallel Time (sec)	Speedup
8388608	120.39	256	14.62	12.57	9.58
			8.68		
			14.42		
		128	14.63	13.91	8.66
			13.32		
			13.77		
		64	16.46	17.95	6.71
			16.08		
			21.31		
		32	25.45	21.67	5.56
			21.23		
			18.32		
		16	30.8	30.24	3.98
			26.2		
			33.71		
		8	44.15	44.42	2.71
			39.28		
			49.83		
		4	58.42	55.34	2.18
			55.21		

			52.39		
		2	88.55	90.31	1.33
			85.95		
			96.44		
		1	128.46	140.60	0.86
			127.1		
			166.25		
4194304	30.57	256	2.45	3.80	8.05
			5.48		
			3.46		
		128	3.76	3.45	8.86
			3.16		
			3.43		
		64	4.65	4.54	6.73
			4.3		
			4.67		
		32	7.89	7.64	4.00
			8.02		
			7.02		
		16	10.29	8.60	3.55
			7.62		
			7.89		
		8	11.16	14.05	2.18
			16.51		
			14.48		
		4	18.61	18.64	1.64
			16.95		
			20.36		
		2	27.11	26.59	1.15
			23.65		
			29.01		
		1	31.96	32.11	0.95
			32.29		
			32.07		
65536	0.02	256	0.06	0.07	0.27
			0.08		
			0.08		
		128	0.06	0.06	0.35
			0.05		

			0.06		
		64	0.73	0.65	0.03
			0.61		
			0.61		
		32	0.77	0.59	0.03
			0.44		
			0.56		
		16	0.55	0.61	0.03
			0.65		
			0.62		
		8	0.29	0.28	0.07
			0.29		
			0.26		
		4	0.01	0.01	1.50
			0.01		
			0.02		
		2	0.02	0.02	1.00
			0.01		
			0.03		
		1	0.02	0.02	1.00
			0.02		
			0.02		

The summary of the previous table is:

Number of Elements (input)	1	2	4	8	16	32	64	128	256
8388608	0.86	1.33	2.18	2.71	3.98	5.56	6.71	8.66	9.58
4194304	0.95	1.15	1.64	2.18	3.55	4	6.73	8.86	8.05
65536	1	1	1.5	0.07	0.03	0.03	0.03	0.35	0.27



These results can be explained by Amdahl's law (since the problem size is fixed), this law stablishes a maximum for speedup. It says that the speedup is always limited by the sequential part of an algorithm that cannot be parallelized (data setup, reading/writing, etc.) and not the number of processors. Even with infinite number of processors, maximum speedup limited to 1/f where f represents the sequential part of the algorithm. This explains why the speedup decreases after the certain number of processes.

The difference between the results for the different input size can be explained by what is known as parallel slowdown (parallelization beyond a certain point causes the program to run slower) due to a communications bottleneck (communications overhead created by adding another processing node that surpasses the increased processing power that the node provides). The communication overhead can be easily shown in the graph in the gray series with small number of elements, since the peak in speedup occurs before than those with bigger number of elements (even if this one has a smaller number of processes).

Question 2

Given an array of n elements and p processes, the quicksort algorithm needs to perform four steps: (i) determine and broadcast the pivot (ii) locally rearrange the array assigned to each process; (iii) determine the destinations of the various S_i and L_i sub-arrays; (iv) and rearrangement

The first step can be performed in time O(log p) that is the bound for One-To-All broadcast in a hypercube. The second step needs to transverse each n/p partition to divide the elements that are smaller and bigger than the pivot, it can be done in O(n/p). The third and fourth step are determining the process partition sizes and the destinations of the various S_i and L_i sub-arrays; and stablishing the amount of time required for sending and receiving the various arrays. The third has a lower bound of O(log p) and the last step can be compared to an all-to-all communication whose complexity has a lower bound of O(n/p). In consequence, the overall complexity of splitting an n-element array is O(n/p) + O(log p). This process is repeated for each of the two subarrays recursively on half the processes, until the array is split into p parts, at which point each process sorts the elements of the array assigned to it using the serial quicksort algorithm. The repetition is limited by the number of dimensions that is represented by log p in a hypercube. Thus, the overall complexity of the parallel algorithm is:

$$Tp = O\left(\frac{n}{p}\, log\frac{n}{p}\right) + \ O\left(\frac{n}{p}\log p\right) + O(log^2 p)$$

The first element represents the sort and the others the partition previously explained.

$$Ts = O(n \log n)$$

$$Sp = \frac{O(n \log n)}{O\left(\frac{n}{p}\ log\frac{n}{p}\right) + \ O\left(\frac{n}{p}\log p\right) + O(log^2 p)}$$

$$Ef = \frac{O(n \log n)}{p *\ O\left(\frac{n}{p}\ log\frac{n}{p}\right) + \ O\left(\frac{n}{p}\log p\right) + O(log^2 p)}$$

Question 3

For the bitonic sorting algorithm in a hypercube during each step, every process performs a compare-exchange operation (single nearest neighbor communication of one word). Since each step takes O(1) time, the parallel time is TP = $\Theta(log^2$n). The quicksort algorithm is lower bounded by Ω(n) and the parallel time is presented in detail in the previous question for the best scenario which proves that can give better results under optimal circumstances.

From the first question after a multiple test it is evident that for the quicksort algorithm the selection of the pivot produces a big change in performance since some of the processes can have 0 elements which makes them idle (produced by the division of elements according to a bad pivot). This difference in load balance affect the parallel time since some processes has to transverse more than n/p elements. In consequence, the speedup is reduced and also the efficiency.

The bitonic sort algorithm has a load balancing feature that guarantees that each processor under parallel machine maintain equal number of elements. This characteristic accomplishes a better use of the processors which benefits the speedup and efficiency obtain with this algorithm.

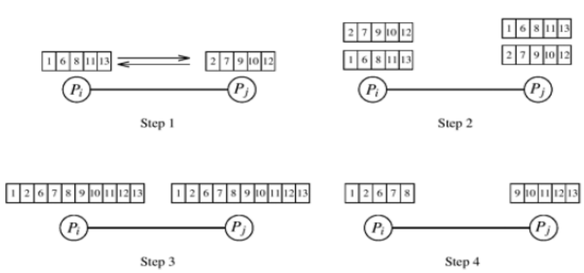
In terms of the input elements. The bitonic algorithm only works with sequences that has a length that is a potency of 2 which is a limitation. The quicksort algorithm doesn't have this limitation but according to the results obtain in question 1 the performance of the algorithm can vary with sequence that have elements with close value (all relatively similar) or a small number of elements since it will be more affected by a bad pivot selection.

Question 4

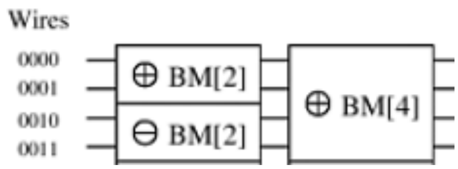
To prove the correctness of the algorithm (coarse grained with compare split) by induction. It is necessary to prove its correctness for d=0 and d=1 which are considered base cases and assuming d=k prove d=k+1. We assume that the sequence is have a length potency of 2. We call BN(K) a bitonic network with a last +BM(K).

For d=0 we only count with one processor. It means that all the elements will be sorted by the local quicksort algorithm before each operation.

For d=1 we have two processors p1 and p2 with sequence of elements P1 and P2 the compare split operation will merge both sequence in one and give p1 the smallest arguments and p2 the biggest ones which will let to an order sequence. It counts only with the application of +BM(2)

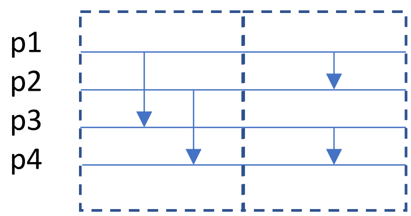


To find a patter for d=k it is important to build BN(4) and prove the -BM(4) of a BN(8) that produce an acceptable patter for a BN(2^k). For a BN(4) the networks will be:

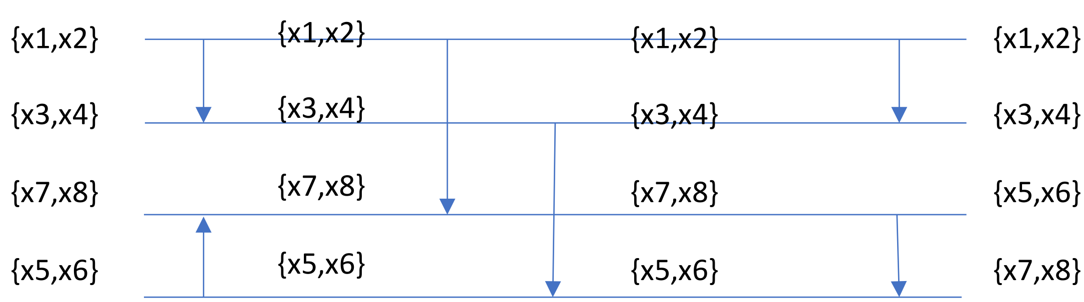


Identifying the wire 00 as p1 and all the other processes respectively. It is a total of p1,p2,p3,p4 and sequence of elements P1,P2,P3,P4 manage by each process, the final sequence will be X=<x1,x2,x3,x4,x5,x6,x7> which means that all the processes manage to elements during the process. It is also necessary to stablish that before +BM(4) comes one +BM(2) and one -BM(2). From the application of the two BM(2) we can deduce that all the elements in P3 are bigger than those in P4 (P3>P4), and all the elements in P2 are bigger that all the elements in P1 (P2>P1).

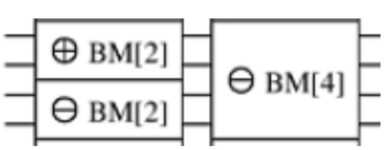
The final BM will always be positive and preceded by a positive and a negative BM of half length. The new conditions for +BM(4) will be P3>P1,P4>P2,P2>P1,P4>P3. This new conditions guarantee that for the final result P4> P1,P2,P3 and P4,P3,P2>P1. The only condition that we still have to prove is the one between P3 and P2 since it is not a clear relation in the network:



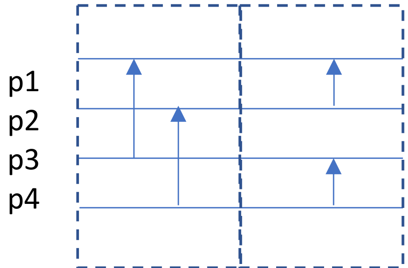
To prove that all the elements in P3 will be bigger than all the elements in P2 it is necessary to evaluate the conditions of the previous +BM(2) and -BM(2). Since P3>P4 and P2>P1, this implies that all the elements that P2 will receive from P4 in the first stage of the +BM(4) will be smaller than all the elements in P3 in the worse scenario all the elements in P2 are bigger than those in P4 which means that the new P2 will be equal to the old P4 which all elements are smaller than P3, this means that in all the possible exchanges P3>P2 since it was assured by the previous condition. In conclusion for BM(4) P4>P3>P2>P1 which gives an order sequence.



To identify the important characteristic of -BM(4) it is necessary to show the conditions that are stablish:

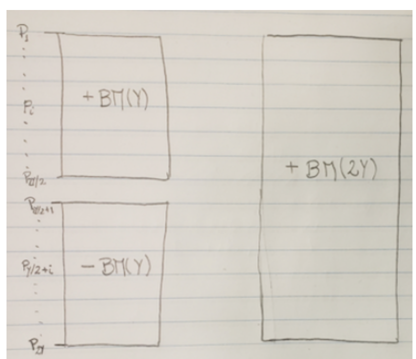


From the network -BM(4):



P1>P3, P2>P4, P3>P4 and P1>P2. From this condition it can be deduced that P1 contains the bigger elements and P4 the smallest elements. To prove that P2>P3 it is important to study the condition from the previous stage in the network from +BM(2) and -BM(2). Since P3>P4 and P2>P1, this implies that all the elements of P1 are smaller than all the elements in P2 and in the worse scenario if all the elements in P3 are bigger than the elements in P1 then the new P3 will be equal to the old P1 which elements are smaller than P2 which proves that all the elements in P2 will be bigger than all the elements in P3 (P2>P3). This proves that P1>P2>P3>P4 which gives a sequence in decreasing order. The final sequences together that +BM(8) will receive from BM(4) and BM(4) will be a bitonic sequence with part of the elements in decreasing order and part in increasing.

For d=k+1 we assume than the conditions previously explained are true for d=k since they are applied recursively. Assuming d=k true then BN(2^k) for simplicity we assume 2^k=Y. Then we assume +BM(Y) and -BM(Y) true and prove BN(2Y).



The previous image reduces the prove to guarantee P2y>P2y/2+i>Pi>P1 knowing that 1<i<2y/2<2y. The first exchange in BM(2Y), will be between Pi and P2y/2+i leaving all the smallest elements at the half in the top and the biggest at the half in the bottom. The second exchange will be between P2y/2+1 with P2y and P2Y/2 with P1 finally leaving the smallest elements with P1, the biggest elements with P2y and replicating the stages of +BM(Y) that will guarantee that all the elements in the middle are sorted in increasing order (since +BM(Y) and -BM(Y) were assume true). The idea of the test is guarantee that before the final BM the previous ones form a bitonic sequence which by the merging theorem will give an ordered sequence at the end.