Comparisons on Process Sharing Solutions

Author: Joe Houng
Western Washington University
houngj@students.wwu.edu

I. Introduction

Suppose that we have n processors and that each processor computes an integer. The goal is for every one of those processors to know the minimum and maximum of all values. I have been presented three methods to accomplish this task.

II. Methods

1.1 Centralized Solution:

One solution would be to have all processes compute their value, and then send it to a designated process, say processor zero, to compute the min and max numbers. After the min and max are computed that designated process zero would send those two values to all other processes.

1.2 Symmetric Solution:

Another solution requires all processes to compute their values just like in the Centralized Solution, but instead of sending all values to one process, all processes sends their value to every other process. This way, All processes know of everyone elses values. From there each process can compute min and max by themselves.

1.3 Ring Solution:

The last solution, unlike the previous two solutions, doesn't need to have all processes compute their values at once. The Ring Solution can be split into two phases.

The first phase starts with process 0 computing its value, assigning min and max to value, and then sending it off to its rank+1, so the next process. So now for every process after 0, the value is compute and then compared to the min and max it recieved from the previous process. After updating min and max accordingly, the two values are sent to the next process.

The second phase starts after the last process has updated min and max, and sends those values to process zero. Now we know the correct min and max from going around to all processes we repeate the same process of going to the next process but this time just passing the min and max without changing it, until we get back to the last process again.

III. Approach

I wrote programs in C for all three solutions using MPI commands MPI_Send, MPI_Recieve, MPI_Barrier, MPI_Wtime, and MPI_Reduce. MPI_Send and MPI_Recieve were used to implement the solutions, and then MPI_Barrier, MPI_Reduce, and MPI_Wtime were used to calculate runtime for each solution. See source code for implementation.

To simmulate the time it takes to compute a value for each process I use the C function, usleep(). And then the value itself is a random value calculated from the C function rand_r(). Each process gets this random value by passing in a seed variable into rand_r() that is set to its rank number as a process.

IV. Math Stuff

To properlly compare each solution, I decided that evaluating both their times and efficiencies is necessary.

$$E(n,P) = \frac{T_{serial}(n)}{P - T_{parallel}(n,P)};$$

P=# of processors, n=# of values, P=n.

T_{serial} is calculated from timing a Serial method I implemented. Look at serialSolution.c.

V. Hypothesis

From implementing the solutions I believe that the Ring method would be most efficient while still be relatively quick. This is because the number of passes between processes is 2n-1 (n = # of nodes), which is significantly less than the passes required for the Symmestric method and does not put computation stress on a single process like the Centralized method does.

VI. Results

In all trials I took the program completion time of 1, 2, 4, 8, 16, 32, 64, and 128 cores for each solution. Also In the bar graphs I have included a Serial method (The black bar), which the number of cores will indicate the number of values to be evaluated

Trial 1

Between 2 to 4 cores:

- Centralized Solution increases only slightly, by .0001 ms

- Symmetric Solution also only increases slightly, by .0001 ms

- Ring Solution jumps dramatically from .000776 to .001075, a .0003 ms jump.

Between 4 to 16 cores

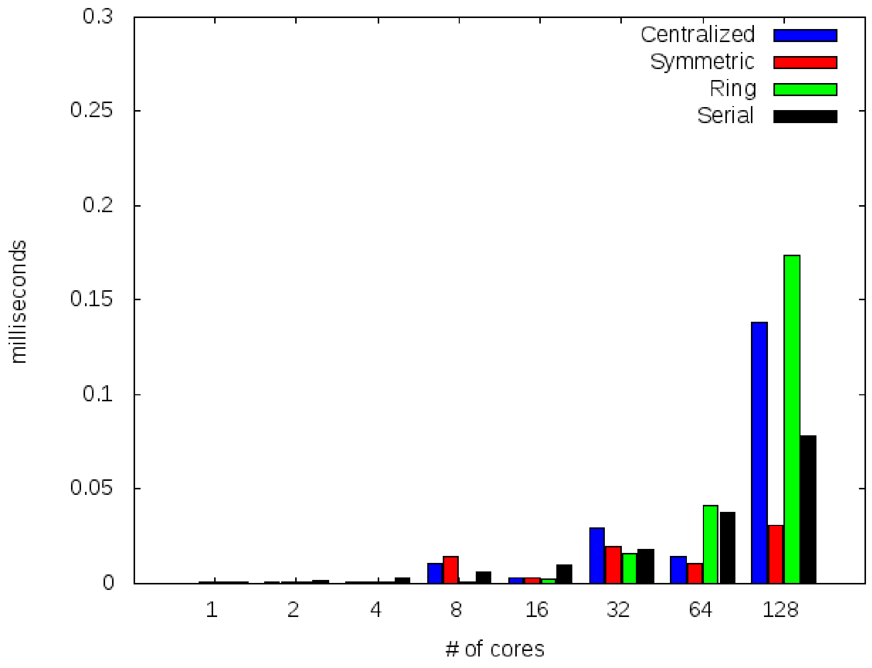
- Centralized Solution increased by .01 ms between 4 to 8 cores but dives back down by .01 at 16 cores. I think this was just a hiccup in take the times for this solution.
- Symmetric Solution increased by .02 between 4 to 8 cores but also dives down by .01 at 16 cores. Perhaps not a hiccup if this happened to both solutions?
- Ring SOlution stays relatively the same between 4 to 8 cores and increases by .001 at 16 cores.

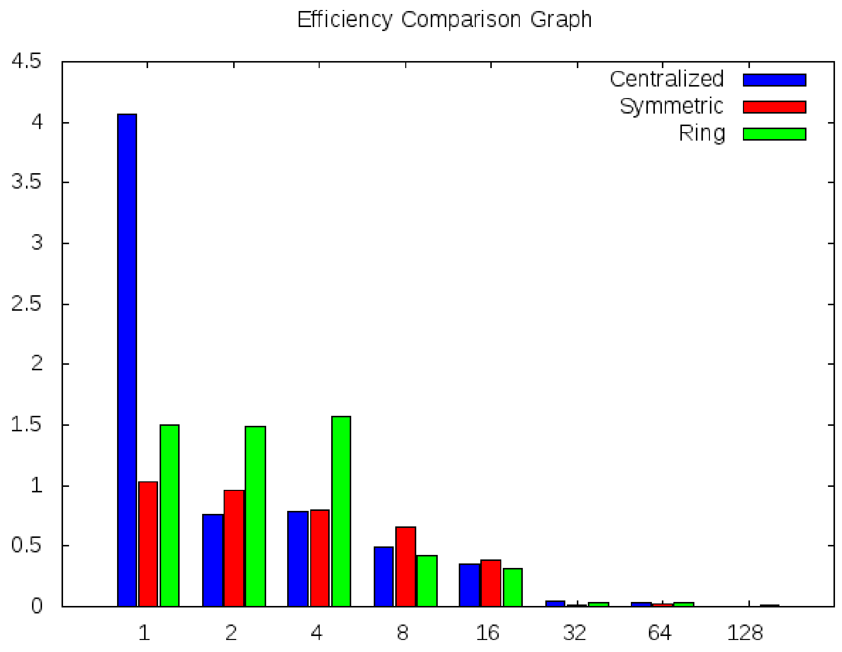
So far the Ring Solution seems to be increasing the most smoothly though almost consistently shows the worst times comparably to the rest of them. The symmetric solution has a few inconsistent jumps in times but it seems that its got the shortest time of them all. This came to a surprise to me because I thought the Ring solution would have the best time, Centralized for the second, and the symmetric solution have the worst time.
The same trend can be seen between 16 and 128 cores.
In terms of efficiency however the Ring solution proved to be the almost consistently greater in all number of cores compared to the others. Which was to my expectations.

	Time Comparisons			
Cores	Centralized Solution	Symmetric Solution	Ring Solution	
1	0.000144	0.00038	0.000549	0.000561
2	0.000977	0.000491	0.000776	0.001509
4	0.001075	0.000522	0.001075	0.003298
8	0.010612	0.013998	0.001064	0.005833
16	0.003315	0.002957	0.002053	
32	0.029127	0.019879	0.015433	
64	0.01409	0.010612	0.041275	
128	0.13853	0.031121	0.173419	

	Efficiency Comparisons		
Cores	Centralized Solution	Symmetric Solution	Ring Solution
1	4.0652173913	1.0293577982	1.5
2	0.7652129817	0.9636015326	1.4940594059
4	0.7882409178	0.7943159923	1.5734732824
8	0.4929851251	0.6521690519	0.425890771
16	0.3521452608	0.391687461	0.3122202884
32	0.048641212	0.0127115585	0.0300237383
64	0.0322436065	0.0245984806	0.0329567037
127	0.0054105626	0.0029320967	0.0131045264

Time Comparison Graph





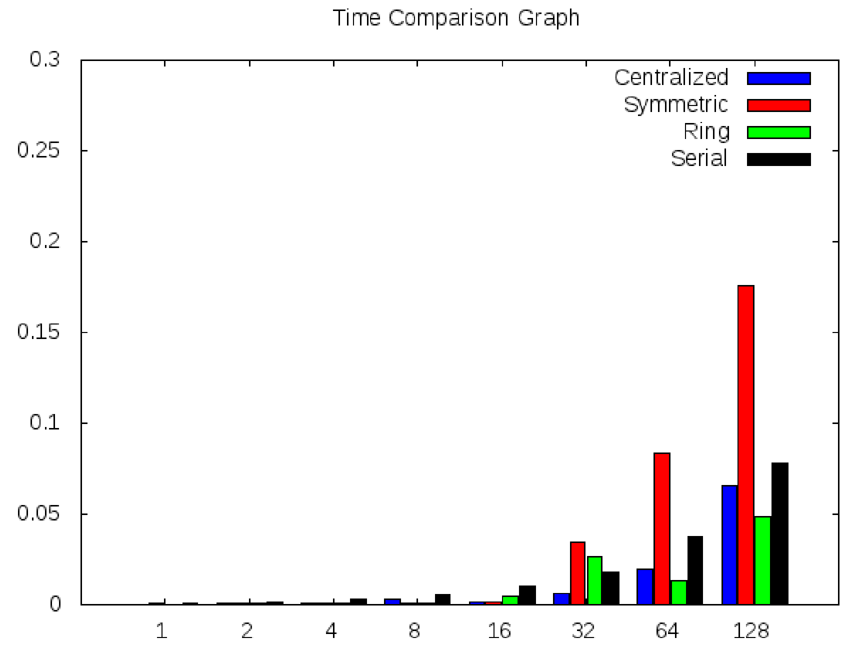
Trial 2

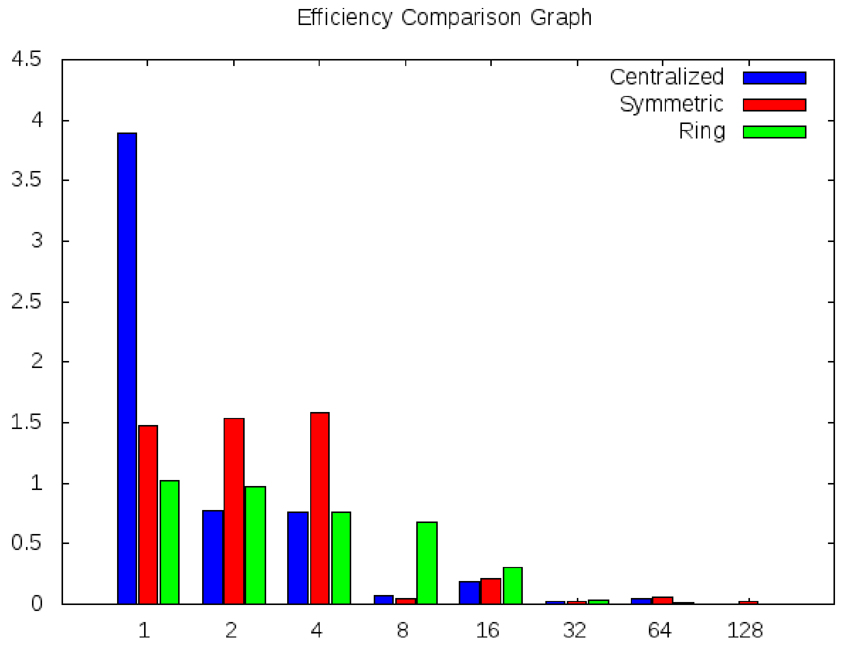
The results of this trial made a lot more sense to me than the first trial. These results indicate that the ring solution becomes compartably faster than the other solutions. However the efficiency of the Ring solution is still not what I expected it to be, lower than the rest.

	Time Comparisons			
Cores	Centralized Solution	Symmetric Solution	Ring Solution	Serial Solution
1	0.000144	0.000551	0.000374	0.000561
2	0.000997	0.000783	0.000488	0.001509
4	0.001033	0.001041	0.000522	0.003298
8	0.002769	0.001087	0.000552	0.005833
16	0.001192	0.001268	0.004648	0.010046
32	0.006378	0.034132	0.026785	0.018314
64	0.019827	0.083811	0.013221	0.037481
128	0.065828	0.175567	0.04867	0.078498

	Efficiency Comparisons		
Cores	Centralized Solution	Symmetric Solution	Ring Solution
1	3.8958333333	1.4763157895	1.0218579235
2	0.7722620266	1.5366598778	0.9722938144
4	0.7669767442	1.5795019157	0.7669767442
8	0.0687075952	0.0520877983	0.6852678571
16	0.1894042232	0.212335137	0.3058329274
32	0.0196488653	0.0287898033	0.0370836843
64	0.0415642743	0.0551866401	0.0141887492
128	0.0044269517	0.0197058457	0.0035363232

Time Comparison Graph





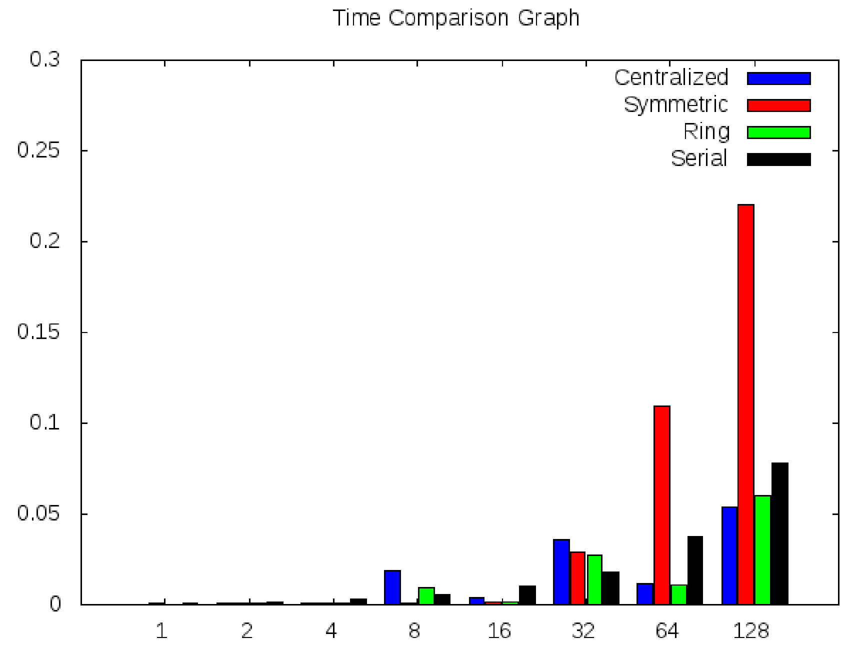
Trial 3

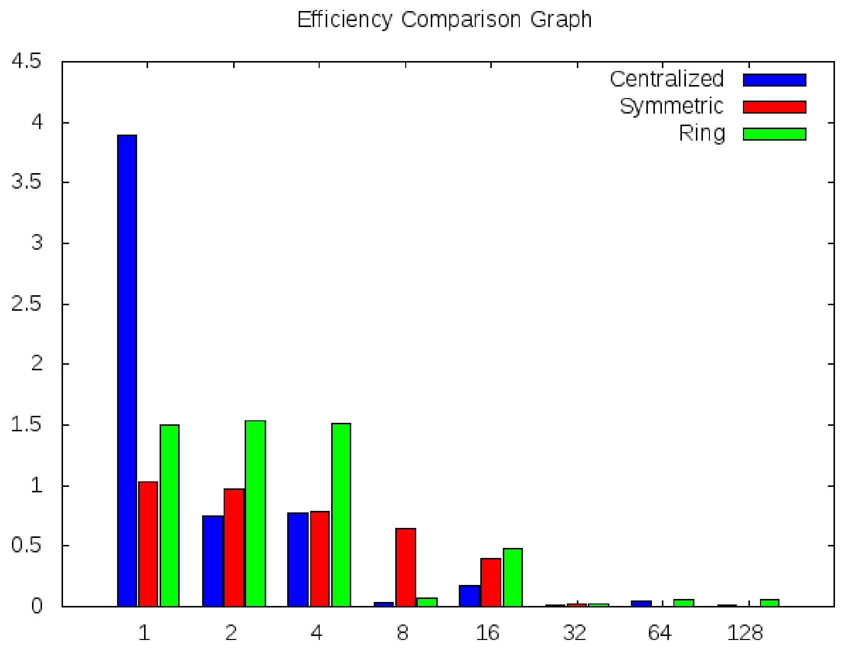
The results of this trial made a lot more sense to me than the first trial. These results indicate that the ring solution becomes compartably faster than the other solutions. However the efficiency of the Ring solution is still not what I expected it to be, lower than the rest.

	Time Comparisons			
Cores	Centralized Solution	Symmetric Solution	Ring Solution	Serial Solution
1	0.000144	0.000545	0.000373	0.000561
2	0.001006	0.000775	0.000491	0.001509
4	0.001067	0.001044	0.000544	0.003298
8	0.018701	0.001139	0.009715	0.005833
16	0.003659	0.001565	0.001321	0.010046
32	0.036216	0.029189	0.027578	0.018314
64	0.012007	0.109691	0.010759	0.037481
128	0.053647	0.220377	0.059783	0.078498

	Efficiency Comparisons		
Cores	Centralized Solution	Symmetric Solution	Ring Solution
1	3.8958333333	1.0293577982	1.5040214477
2	0.75	0.9735483871	1.5366598778
4	0.7727272727	0.7897509579	1.515625
8	0.0389885568	0.6401448639	0.0750514668
16	0.171597431	0.4011980831	0.4753028009
32	0.0158027529	0.0196071294	0.020752502
64	0.0487749334	0.0053390034	0.0544326262
128	0.011431499	0.0053390034	0.0544326262

Time Comparison Graph





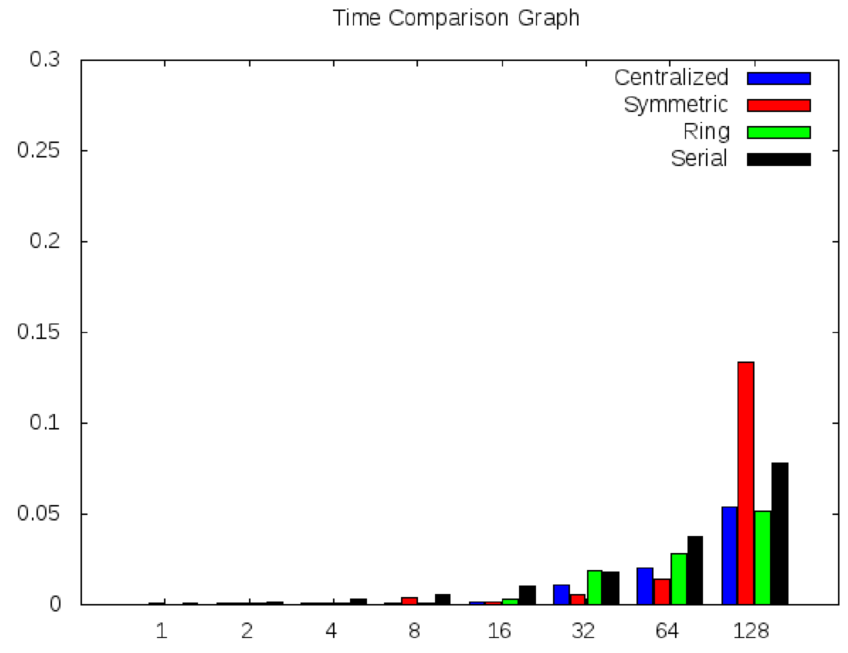
Average of 10 Trial

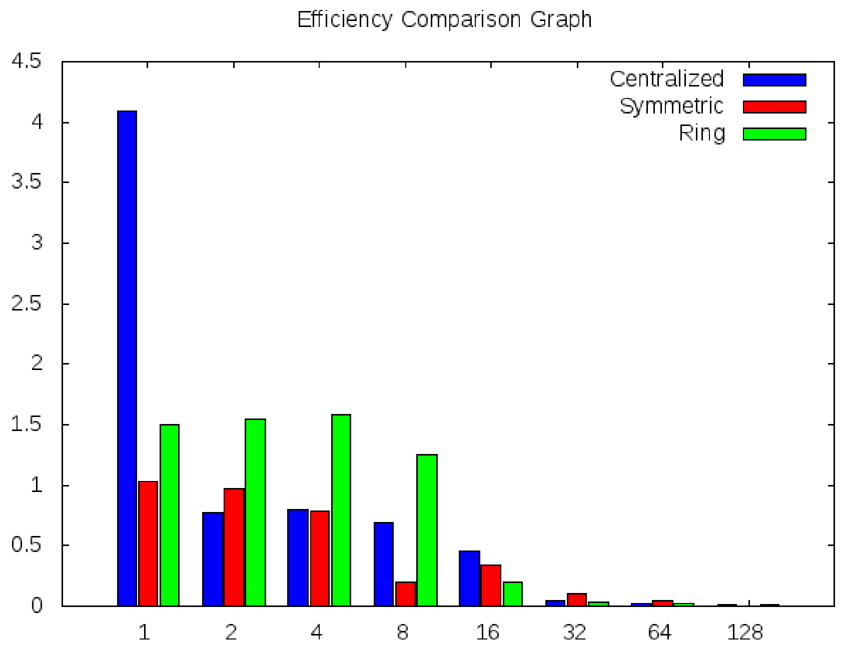
The results of this trial made a lot more sense to me than the first trial. These results indicate that the ring solution becomes compartably faster than the other solutions. However the efficiency of the Ring solution is still not what I expected it to be, lower than the rest.

	Time Comparisons			
Cores	Centralized Solution	Symmetric Solution	Ring Solution	Serial Solution
1	0.000137	0.000545	0.000374	0.000561
2	0.000971	0.000775	0.000489	0.001509
4	0.00103	0.001048	0.000522	0.003298
8	0.00106	0.003675	0.000582	0.005833
16	0.00136	0.001831	0.003237	0.010046
32	0.011218	0.005204	0.019027	0.018314
64	0.020282	0.013842	0.028259	0.037481
128	0.053823	0.133876	0.051176	0.078498

	Efficiency Comparisons		
Cores	Centralized Solution	Symmetric Solution	Ring Solution
1	4.0948905109	1.0293577982	1.5
2	0.7770339856	0.9735483871	1.5429447853
4	0.8004854369	0.7867366412	1.5795019157
8	0.6878537736	0.1984013605	1.2527920962
16	0.4616727941	0.3429137084	0.1939681804
32	0.0510173382	0.1099754996	0.0300789667
64	0.0288748952	0.04230896	0.0207240392
128	0.0113941182	0.0045808481	0.0119834615

Time Comparison Graph





Directions

Included in the github repository is script.sh that can by using the command ./script.sh [nameofexecutable]. This will run the executable for cores 1,2,4,8,16,32,64,128 and spit out a list of computation times for those cores. The executables is also included in the github repository at www.github.com/houngj/Sharing_Computations_Among_Processes. If the executables are ran with any kind of flag (ex. mpiexec -n 12 ./SymmSolution z) it will output the min and max that each core has recieved and the computation time, otherwise it will only output the computation time. To compile each C program you have to run “mpiexec -g -Wall -o cprogramname cprogramname.c”