Ira A. Fulton Schools of Engineering
School of Computing, Informatics and Decision Systems Engineering

SER 450 **COMPUTER ARCHITECTURE**

Your Name: Jacob Hreshchyshyn

FINAL PROJECT WORKSHEET

Step II: Amdahl's Law Analysis

A certain algorithm may be parallelized onto a cluster of N nodes such that each node in the cluster operates on 1/Nth the total data, but must also exchange its results with the (N-1) other nodes in the network. An unusual characteristic of this code is that it is completely parallelizable (there is no serial portion of the code). Based on this characteristic, what would the expected speedup be for 2, 8, and 32 nodes (show your work)?

Since there no serial code, the speedup will linearly with processor nodes (2, 8, 32)
Let's first recall what Amdahl's Law states.

Execution Time After Improvement = (Execution Time Affected By Improvement/Amount Of Improvement) + Execution Time Unaffected

We are given that the code is completely parallelizable. This means that any improvement made in the number of processing nodes will affect all of the execution time of the program. If we assume that our original execution time of the program is 1 second, our formula looks like the following:

Execution Time After Improvement = 1 second/2 nodes + 0 Execution Time Unaffected.
Execution Time After Improvement = 0.5 seconds.

The speedup can then be obtained by dividing the Execution Time Before Improvement by the Execution Time After Improvement:

Speedup = 1 second/0.5 seconds = 2

We clearly see that the speedup is the same as the number of processing nodes added to the cluster. This pattern will be the same as the number of processor nodes increase. So, for 8 nodes, the speedup will be 8. For 32 nodes, the speedup will be 32.

Therefore, the speedup increases linearly with the number of processor nodes.
Based on this analysis, does this algorithm appear to be a good candidate for parallelization (justify your answer)?
Yes, since it scales linearly, this application looks ideal (you cant get better scaling with Amdahl's law)

Because of this linear improvement in speedup as processor nodes are added, this algorithm would be a great candidate for parallelization. In fact, an algorithm without any serial portions could not be better because that means that there is no portion of the code that will not benefit from an increase in processing nodes, meaning that this is the best case for Amdahl's Law.

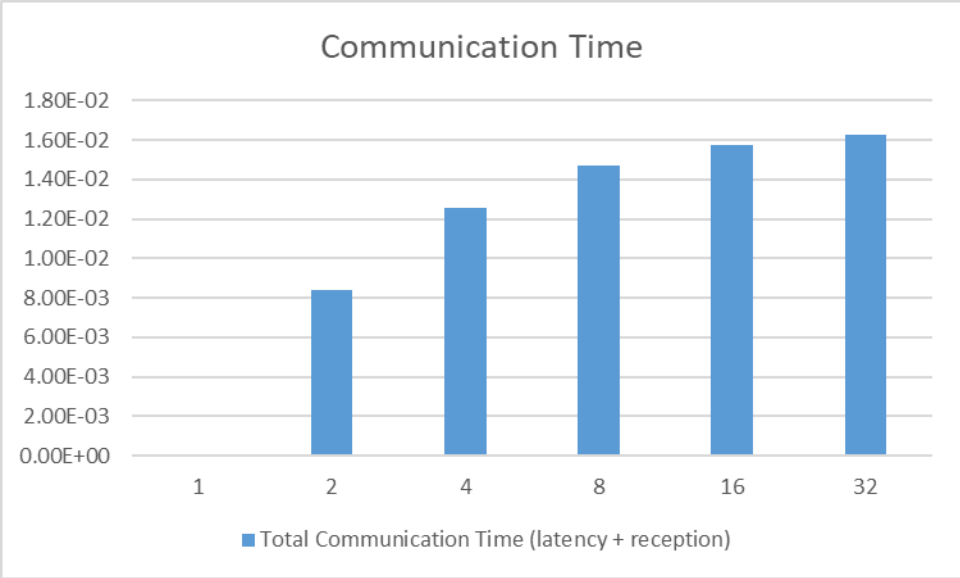
Step III: Initial Time Calculations

A certain algorithm may be parallelized onto a cluster of N nodes such that each node in the cluster operates on 1/Nth the total data, but must also exchange its results with the (N-1) other nodes in the network.

A. Assuming a data set size of 2MiB ($2x2^{20}$ bytes), a communications latency of 90ns and a network throughput of $125x10^6$ bytes per second to each node (note that this is not the same as $125x2^{30}$ bytes/sec), create a spreadsheet tab labeled "Communication Time". On that tab, compute the information below and copy the results into this worksheet.

Number of Nodes	1	2	4	8	16	32
Data operated on by Node (bytes)	2097152	1048576	524288	262144	131072	65536
Data received from other nodes (bytes)	0	1048576	1572864	1835008	1966080	2031616
Reception time in seconds (bytes received/throughput)	0.00E+00	8.39E-03	1.26E-02	1.47E-02	1.57E-02	1.63E-02
Total Communication Time (latency + reception)	9.00E-08	8.39E-03	1.26E-02	1.47E-02	1.57E-02	1.63E-02

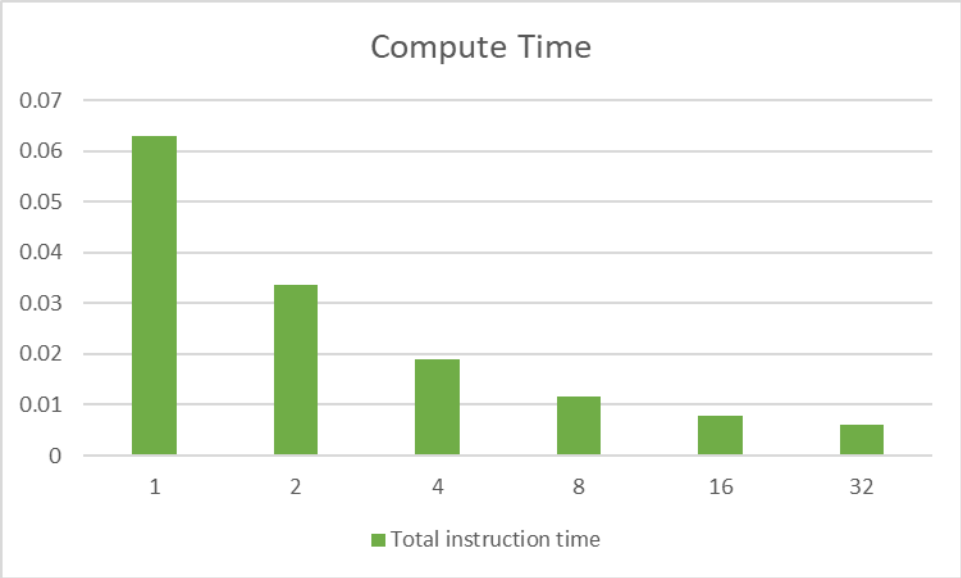
Create a chart that shows the number of nodes on the x-axis, and the communication time on the y axis and include the chart here.



B. Create a second tab in the spreadsheet labeled "Compute Time". On that sheet calculate the following items, assuming the processor requires 30 instructions to operate on each byte of its portion of the data, and 2 instructions to transmit each byte of data to each other node. Copy the results to this worksheet. Assume perfect cache behavior.

Number of Nodes	1	2	4	8	16	32
Average instructions per second	1.00E+09					
Instructions to process share of data	62914560	31457280	15728640	7864320	3932160	1966080
Instructions to send data to 1 other node	N/A	2097152	1048576	524288	262144	131072
Instructions to send data to all other nodes	N/A	2097152	3145728	3670016	3932160	4063232
Total instruction time	0.06291456	0.033554	0.018874	0.011534	0.007864	0.006029

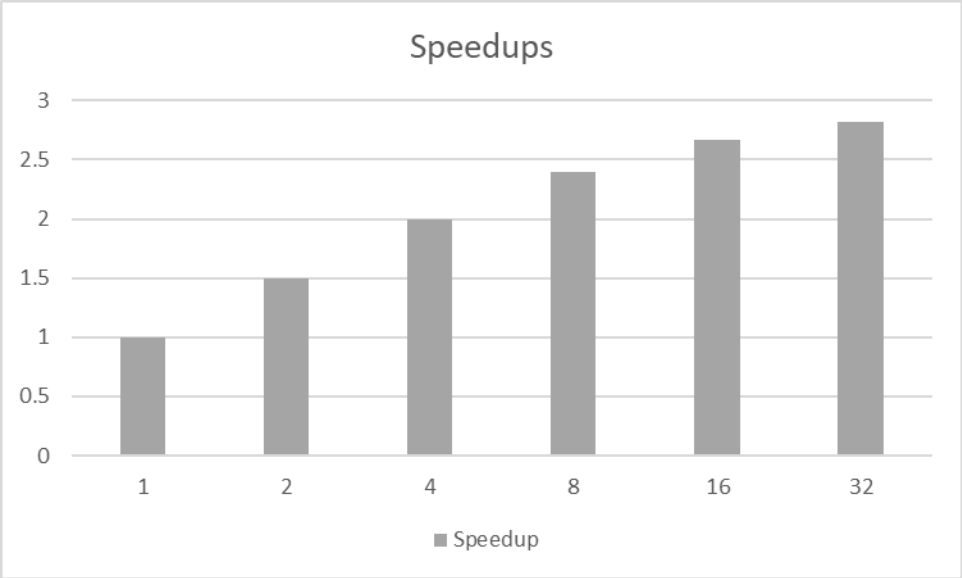
Create a chart that shows the number of nodes on the x-axis, and the compute time on the y axis and include the chart here.



C. Create a new spreadsheet tab labeled "analysis". On the spreadsheet, compute the total time and speedups for each configuration. Assume that communications can be done completely in parallel with the processor's execution. Use the 1 node system as a reference for 1x speedup. Copy your results here.

Number of Nodes	1	2	4	8	16	32
Total Time	0.06291465	0.041943	0.031457	0.026214	0.023593	0.022282
Speedup	1	1.499999	1.999997	2.399995	2.66666	2.823522

Create a chart that shows the number of nodes on the x-axis, and the speedup on the y axis. Include the chart here.



Step IV: Analysis and Business Recommendation

A. If network latency were decreased to 0 and network throughput increased to infinite (a perfect network), what would the approximate maximum speedup be for this application (assume no more than 10,000 nodes)? Please show your work here.

From our previous charts, the Total Communication Time is calculated by adding the network latency and the reception time together. We already know that the network latency is 0. The reception time is calculated by dividing the bytes received from other nodes by the throughput. If the throughput is increased to infinite, the reception time is also effectively 0, meaning that the Total Communication Time would be 0. This means that time would actually accumulate from the Compute Time. To assist in this calculation, we rely on the previous assumption that the average number of instructions per second is 1E+09. This means that our Compute Time can be determined by dividing the sum of the number of instructions to process a node's share of data and the number of instructions to send data to all other nodes by the average number of instructions per second. If we assume that there are to be 10,000 nodes in the network, then the number of instructions worked on by a single node would be $2*2^{20} * (1/10{,}000) = 209.7152$. We are told that it takes 30 instructions to operate on a single byte in the data portion. This means that the number of instructions to process a node's share of data would be 209.7152 * 30 = 6291.456. Next, we must determine the number of instructions to send data to all other nodes. We are told that it takes 2 instructions to transmit each byte of data to each other node. To find the number of instructions to send data to all other nodes, we can multiply the number of bytes operated on by a single node by 2 instructions and multiply that product by the number of nodes - 1 to represent every other node besides the processing node. Doing so gives us (209.7152 * 2)(10,000 – 1) = 419.4304 * 9,999 = 4,193,884. We can now sum this with the number of instructions to process a node's share of data and divide by the average number of instructions per second to give us (6291.456 + 4,193,884)/1E+09 = 0.0042001755 seconds. Based on this calculation, there appears to be a trend towards 0.0042 seconds as the minimum amount of

execution time achievable. When comparing this to our 1-node system with a time of 0.06291465 seconds, our speedup would be 0.06291465/0.0042 = approximately 15 time speedup.

B. What conclusions can you draw for a multi-core multiprocessing configuration instead of a cluster? Please state your reasoning.

As the later question indicates, because no servers currently exist with more than 32 processing cores, a multi-core multiprocessing configuration might not give us better performance when compared to the data found in our cluster analysis. We have already determined that the communication time, despite growing logarithmically as more processing nodes are added to the cluster, is somewhat negligible when factoring in the great reductions in compute time as more processing nodes are added. Additionally, the fact that a theoretical computing speedup of 15 times is possible with up to 10,000 processing nodes in a cluster, it is clear that a cluster can offer much higher performance than a multi-core multiprocessing configuration is physically capable of. In this instance, with this data, the multi-core multiprocessing solution would most likely not be a viable alternative to a cluster. A multi-core multiprocessing configuration would only be justifiable if the communication time cost outweighed the reduction in compute time in a cluster by such an extent that physical CPU cores are preferable.

C. If the L1 data cache is 32kiB in size, at how many nodes will it take so that each node's data set fit completely within its cache? Show your work here.

2MiB/32kiB => 64 nodes

This can be found by dividing the size of our data set by the size of our L1 cache.

$2MiB/32kiB = (2*2^{20})/(32*2^{10}) = 64$ nodes

D. So far, this problem assumed perfect cache behavior. With realistic caching behavior, how will processing times be impacted? Please state your reasoning.

If we are assuming that we are using the same cache as indicated in IV C, I do not think there will be much of an impact in processing times for what we calculated, which was up to 32 nodes. This is, in part, because we know that we can fit up to 64 nodes in the L1 data cache, which means that, once the cache warms up past the initial compulsory misses, which will likely case a negligible detrimental impact on processing times, the cache will have the capacity for each node's data accesses. However, once we start adding nodes greater than 64 in number, additional capacity misses will begin occurring, thereby causing an increasingly detrimental effect on performance.

E. Assume that networking infrastructure costs are negligible and server costs increase by a factor of 1.75x for each doubling of processors within the server (1.75 for 2 processors cores, 3.06x for 4 cores) – no servers are currently available with more than 32 cores. Provide an architectural recommendation (number of nodes and number of cores per server) to FractalFutures that provides the best performance per dollar. Justify your answer and show your work.

There are a few additional things to consider before a recommendation can be made. First, we know that the addition of cores in a system can theoretically improve the performance, thereby affecting the speedup of the system. However, we also know that each core in the system will be limited by the cache since that does not scale up as the number of cores increase. While additional cores will improve performance, the misses resulting from limited cache capacity will affect that speedup, which limits the motivation of doubling processor numbers given that the server cost increases by a factor of 1.75 times. We also know that networking infrastructure costs are negligible, meaning that the number of processing nodes running in a network does not greatly affect the cost of maintaining that network. We also know that we can fit 64 processing nodes completely in our cache, which would eliminate unnecessary performance penalties resulting in capacity misses. With all this in mind, we can begin by comparing the costs per dollar of a server running 64 nodes all on one core, a server running with two cores running 32 nodes each, a server running with 4 cores running 16 nodes each, a server running with 8 cores running 8 nodes each, a server running with 16 cores running 4 nodes each, and a server running with 32 cores running with 2 nodes each. In the Compute Time sheet of my Excel workbook, I calculated the total execution time of the server running 64 nodes. The result is 2.16E-02 seconds. The execution times of the other combinations should be easier to calculate since it is a matter of halving the total instruction times of discovered speeds with each set of nodes for every time the number of cores doubles. Below are the results of those calculations:

32 node 2 core time = 0.022282 seconds/2 cores = 0.011141 seconds

16 node 4 core time = 0.023593 seconds/4 cores = 0.00589825 seconds

8 node 8 core time = 0.026214 seconds/8 cores = 0.00327675 seconds

4 node 16 core time = 0.031457 seconds/16 cores = 0.0019660625 seconds

2 node 32 core time = 0.041943 seconds/32 cores = 0.0013107188 seconds

We can now begin to calculate the performances per dollar. For the combination with only a single processor core, we can consider the cost to be $1. Recall from Project 4B that performance per Watt the new processor relative to the current generation could be found using the following expression:

Performance Speedup / (power_new/power_old)

Similarly, the performance per dollar of the processors we compare relative to the original 1 core 1 node system can be found using the following expression:

Performance Speedup / (cost_new/cost_old)

Thus, our Performances per Dollar (PpD) are represented by the following:

64 node 1 core PpD = (0.06291465/.0216)/($1/$1) = 2.912715278

32 node 2 core PpD = (0.06291465/0.011141)/($1.75/$1) = 3.226930129

16 node 4 core PpD = (0.06291465/0.00589825)/($3.0625/$1) = 3.482992275
8 node 8 core PpD = (0.06291465/0.00327675)/($5.359375/$1) = 3.582567079
4 node 16 core PpD = (0.06291465/0.0019660625)/($9.37890625/$1) = 3.411946954
2 node 32 core PpD = (0.06291465/0.0013107188)/($16.41308594/$1) = 2.924502606

Based on these calculations, the clear winner in terms of performance per dollar is the 8 node 8 core configuration. For this reason, my recommendation is that FractalFutures use this same configuration.