Supercomputers Architecture

Hands on 5

November 24th, 2015

Constantino Gomez

Albert Segura

Cristobal Ortega

Exercise 1:
• Create the Matrix-Vector Multiplication SEQuential program "mvmseq.c" program. ✓
• Compile it ✓

Exercise 2:
• Create the Matrix-Vector Multiplication MPI program "mvmmpi.c" program. ✓
• Compile it ✓

Exercise 3: Taking the time of sequential version.
Timing only the computation part, excluding the time required to populate the matrix and vector. Give your global view of the results

	1024	**2048**	**4096**	**8192**	**16384**	**32768**
T sequential	0,379	2,747	11,313	45,372	198,714	817202.30

It is not increasing linearly, so, if we need to process a matrix much bigger than `32768 x 32768` this approach is unacceptable

Exercise 4: Timing the matrix-vector multiplication program for different number of processes and for different sizes.
Timing only the computation part, excluding the communication part (e.g. the execution time of process 0). Give your global view of the results.
We suggest the following experiments (keep ptile=16):

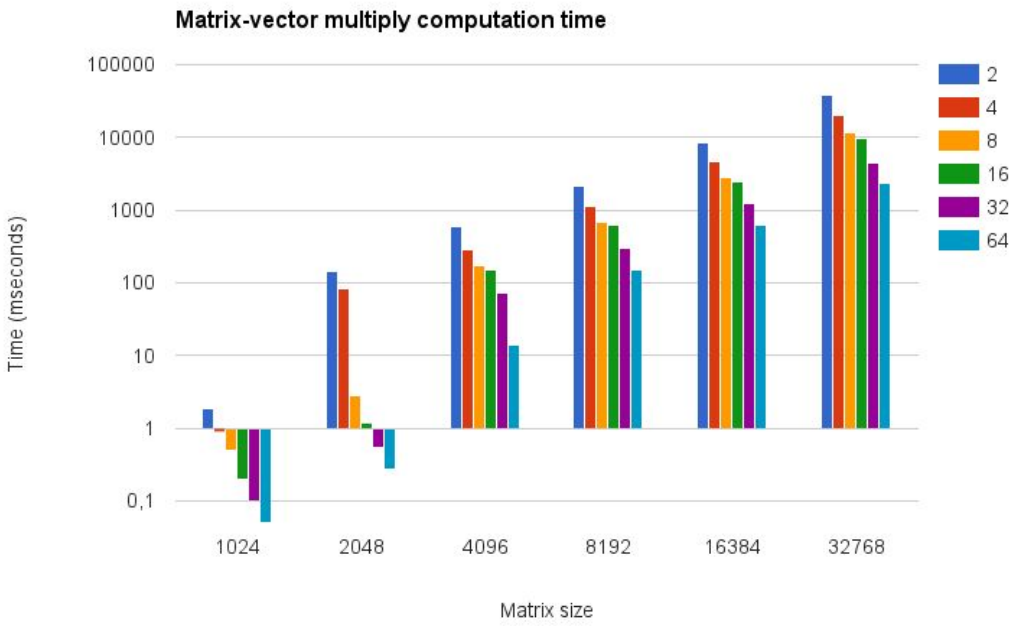
	1024	**2048**	**4096**	**8192**	**16384**	**32768**
2	0,0019	0,144	0,603	2,126	8,509	38,346
4	0,0009	0,084	0,281	1,131	4,629	19,998
8	0,0005	0,0028	0,171	0,689	2,835	11,540
16	0,0002	0,0012	0,153	0,618	2,503	9,780
32	0,000098	0,000531	0,073	0,301	1,215	4,383
64	0,00005	0,000269	0,014	0,151	0,611	2,340

Exercise 5: Timing the matrix-vector multiplication program for different number of processes and for different order of Matrix including the required communication time of the collective communication between processes (do not include the time required to populate the matrix and vector).

	1024	**2048**	**4096**	**8192**	**16384**	**32768**
2	2.385	1.606	66.111	234.876	938.786	4.193.722
4	1.522	10.205	34.235	136.031	555.568	2.921.791
8	1.336	4.729	23.433	92.534	374.527	1.526.126
16	1.638	3.743	22.325	86.542	343.075	1.347.246
32	3.079	4.448	16.503	62.104	231.849	940.251
64	2.212	6.715	13.485	49.275	181.546	703.684

Exercise 6: Plot the results of table 5.2 comparing the size problem with the execution time. Use a logarithmic scale for the time axis.

In this graph there is something strange, What? Why is this? Can you make a guess?



We can observe when we have bigger matrixes is that for 8 and 16 processes the performance is almost the same. This is something we have seen previously, and our guest is an effect of bandwidth, MN3 nodes have 2 sockets of 8 cores each one, meaning that when we spawn 16 processes we actually are in the same node, the effect of this is that we have 16 processes accessing memory at the same time for reading and writing.

Exercise 7: Why when we use 16 processors the execution time is similar to 8 processors? The first hypothesis could be a problem of memory (you can notice that it occurs from a given matrix size). In order to see the memory usage we can generate a log file for example for a N=8192. With a grep (e.g. cat profile.log | grep "Mem:") we can determine the memory usage. What do you think?

Profiling with top we see this while executing:

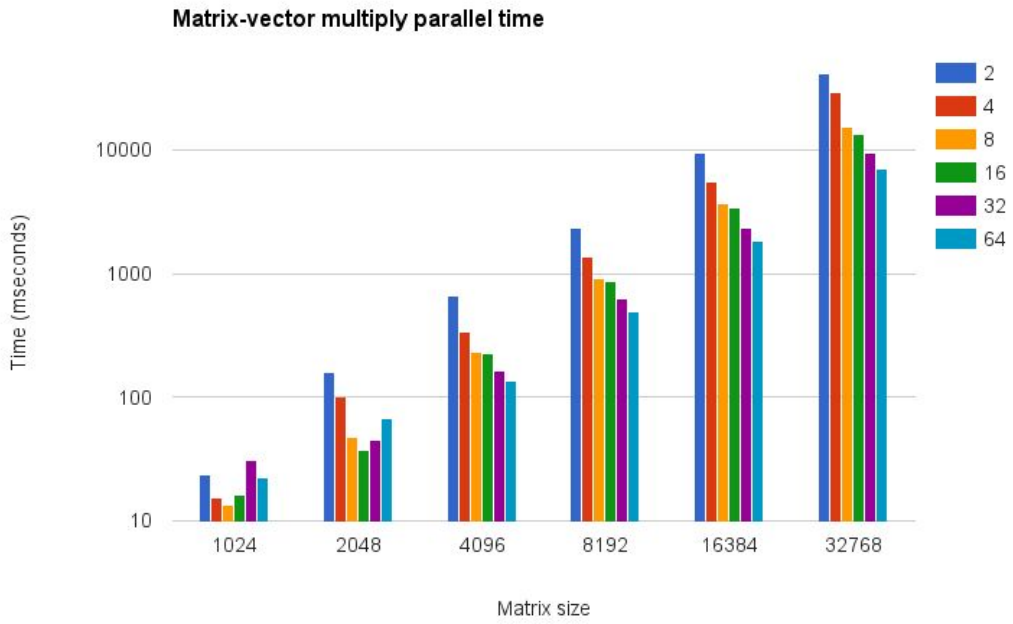
```
Mem:  64532M total,     5105M used,  59427M free,       0M buffers
```

Remember that when we talked about Von Neumann Model Improvements in theory class, the bandwidth between processors and memory can be a bottleneck for some type of applications. Let's try check it

Exercise 8: Execute again the problem with different number of threads per node (e.g. 4,8,16).
What is your conclusion?

	ptile=4	**ptile=8**	**ptile=16**
	8192	8192	8192
16	324.12	312.51	303.32

Exercise 9: Plot the results of table 5.3 comparing the size problem with the execution time. Use a logarithmic scale for the time axis. Why are the results very similar to the previous one plot?

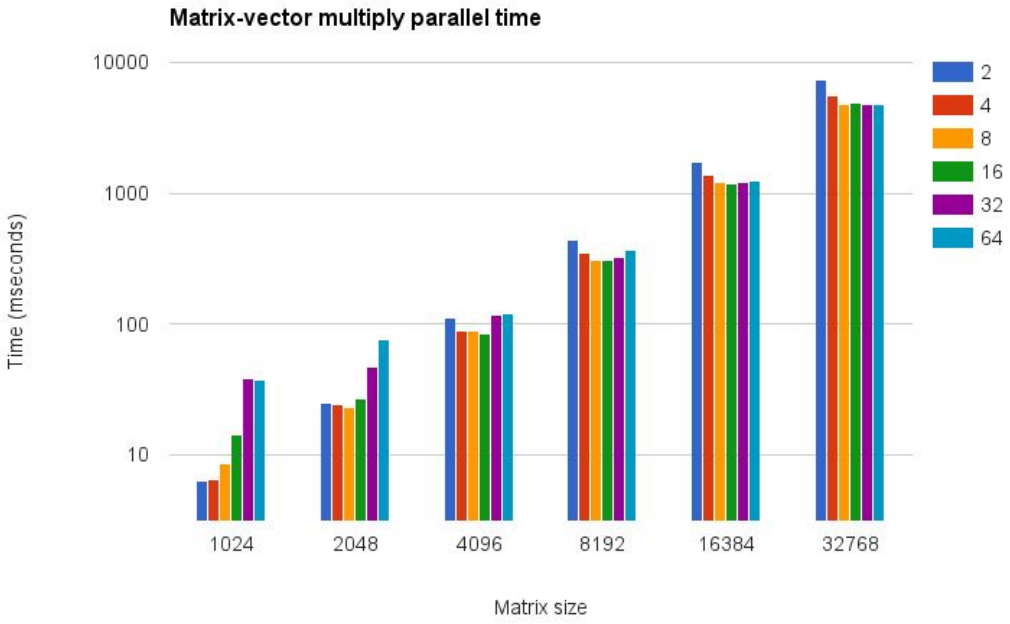


This plot is very similar to the previous one mainly because the computation part of the parallel MPI region is the main component.

Exercise 10: Create a new version of the program mvmmpi.com.c (compting the communication time) but reduce the parallel factor x10 (multiply x100) (mvmmpi100.com.c). Compile with mpicc -o mvmmpi mvmmpi100.com.c and use the same previous JSF file. Timing the matrix-vector multiplication program mvmmpi100.com.c to populate the following matrix.

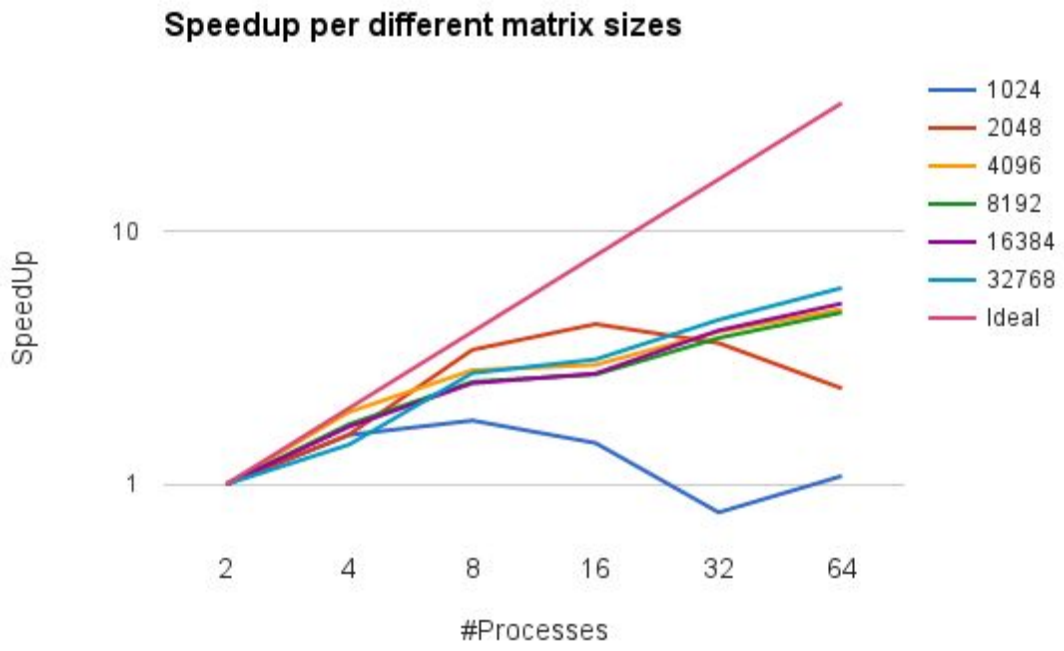
	1024	2048	4096	8192	16384	32768
2	6,23	24,95	111,93	435,44	1739,18	7415,05
4	6,51	24,25	89,51	346,39	1383,4	5624,92
8	8,46	23,14	89,33	306,61	1198,5	4763,94
16	14,32	26,78	84,11	304,7	1171,1	4935,08
32	37,72	46,54	117,8	325,47	1214,06	4746,14
64	36,89	76,85	120	362,69	1247,46	4808,53

Exercise 11: plot the results and compare with the previous one. What happened with small values of N? Why?
What is your conclusion?



As we can see for small values of the matrix size the less processes we use the better, this is due because when increasing the number of processes the communication overhead is bigger than the actual work. Also, when increasing the size of the matrix, getting more processes to work makes no difference for the same reason.

Exercise 12: Plot the graph of the Speedup vs the number of processing elements for matrix multiplication. Also print the linear speedup. Discuss your results with your partner and your teacher during the lab. Include in your report the explanation of its behavior.
What is your conclusion?



We are not close to the ideal speedup that theoretically we can achieve, but we can see that as we increase the matrix size we are get closer to the ideal (32768 elements size is the closest for 64 processes)

This means that we need a very large input to be able to scale to even 16 processes, for 8 processes we are very close to the ideal.

That is why is difficult to code to HPC, because you need to work with large inputs and try to reduce to minimum the overheads.