James Lamb, Pawel Manikowski - Coursework 2 – Exercise 3 – Part E

STRONG SCALING

For strong scaling we modified the `merge_sort_mpi.py`:

```
Nel = int(1e6*1.25)
```

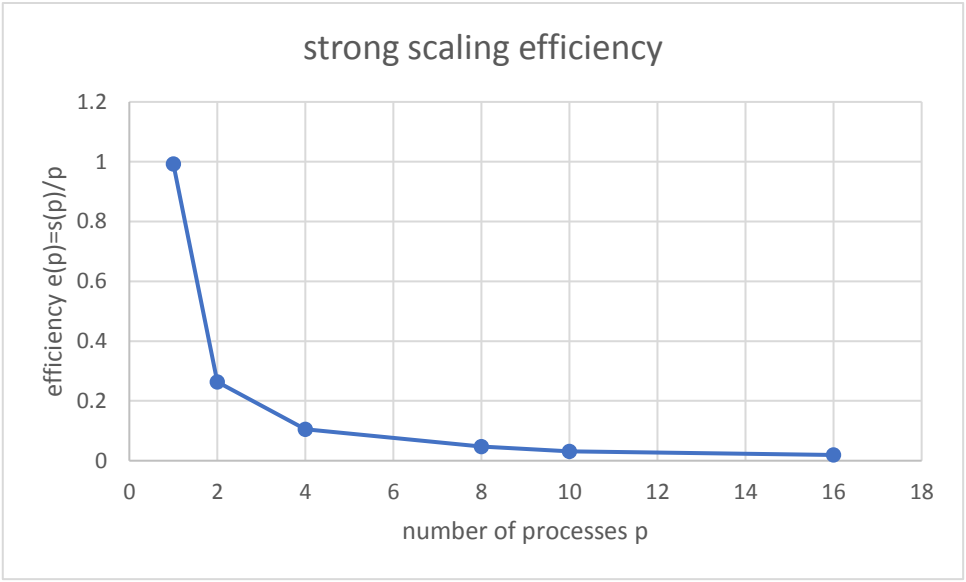
We copied the modified file to room 206 and then run it:

```
> mpirun -N p -f hostfile python myfile.py
```

where p was the number of processes. For each process we calculated the total time three times. We put the obtained values in the table below and calculated the average time, speed up and efficiency:

STRONG SCALING							
	# of processes	total time 1	total time 2	total time 3	**average time**	speed up	**efficiency**
	p	t1	t2	t3	**T = (t1+t2+t3)/3**	s(p) = T_s/T_p	**e(p)= s(p)/p**
serial	**1**	0.238562107	0.236366034	0.240278959	**0.238402367**	0.992716655	**0.992716655**
parallel	**2**	0.473155022	0.432645083	0.440429926	**0.448743343**	0.527397238	**0.263698619**
parallel	**4**	0.587161064	0.544538975	0.555813074	**0.562504371**	0.420736286	**0.105184072**
parallel	**8**	0.621259928	0.637888908	0.621524096	**0.626890977**	0.377523379	**0.047190422**
parallel	**10**	0.808695078	0.791343927	0.688639879	**0.762892962**	0.310221764	**0.031022176**
parallel	**16**	0.730319023	0.778112173	0.80500412	**0.771145105**	0.306902032	**0.019181377**

Using the data from the table we obtained the following graph of efficiency for strong scaling:



Clearly, the program does not have a very high strong scaling efficiency, with a 1% efficiency rate at 16 processes. Even at only 2 processes, the runtime is such that using a serial program would be more efficient. I will discuss parallelisation problems further in the next set of results.

WEAK SCALING

For weak scaling to get the local size of 10MB we needed to modify the `merge_sort_mpi.py`:

```
Nel = size*int(1e6*1.25)
```

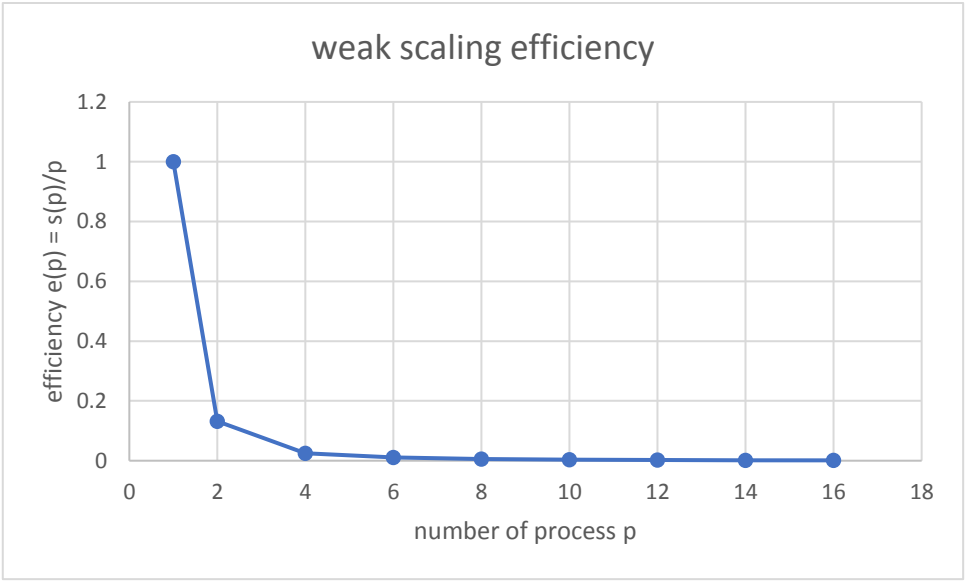
We copied the modified file to room 206 and then run it:

```
> mpirun -N p -f hostfile python myfile.py
```

where p was the number of processes. For each process we calculated the total time three times. We put the obtained values in the table below and calculated the average time, speed up and efficiency:

WEAK SCALING							
	# of processes	total time 1	total time 2	total time 3	**average time**	speed up	**efficiency**
	p	t1	t2	t3	**T = (t1+t2+t3)/3**	s(p) = T_s/T_p	**e(p)= s(p)/p**
serial	**1**	0.24	0.23	0.24	**0.236666667**	0.999997183	**0.999997183**
parallel	**2**	0.898722887	0.869142056	0.918504	**0.895456314**	0.264296534	**0.132148267**
parallel	**4**	2.315710783	2.345202923	2.389524937	**2.350146214**	0.100702671	**0.025175668**
parallel	**6**	3.600500107	3.484977007	3.633229017	**3.572902044**	0.066239152	**0.011039859**
parallel	**8**	5.287915945	5.462336063	5.492663145	**5.414305051**	0.043711242	**0.005463905**
parallel	**10**	7.928426027	7.236097097	7.264084101	**7.476202408**	0.031655911	**0.003165591**
parallel	**12**	8.074198008	7.962728977	8.642838001	**8.226588329**	0.028768426	**0.002397369**
parallel	**14**	9.587563038	9.552712917	9.159389973	**9.433221976**	0.025088565	**0.00179204**
parallel	**16**	12.50679994	12.57130003	12.849751	**12.64261699**	0.0187197	**0.001169981**

Using the data from the table we obtained the following graph of efficiency for weak scaling:



Again there is a clear inability of the code to run efficiently on a parallel basis, in increasing the number of elements to be sorted we have reached less than 1% efficiency at 16 processes. Again even at 2 processes it would seem the serial code would be more efficient.

Although, it is true that the sorting of a vector is much faster using pre-sorted subvectors, we must also consider the communication bottlenecks. By breaking down the individual times for each section of the code, we find that as the process number increases, the time taken to presort the individual subvectors decreases. However, in order for this larger number of processes to effectively work together they must communicate these subvectors across the communicator, which then means both processors involved in the communication must take time to do so hence increasing the overall processing time for the program.

In the case of a weak scaling variant, this problem is exemplified further by the fact that as the number of processes increases, the overall workload of each process does not decrease and hence the impact of communication between the processes is no longer mitigated by the sharing of the workload.