Comp528 Individual Coursework
Assignment 1

By Robert Johnson
200962268

October 26, 2017

Contents

1 Implementation of serial program

Because of the heavy reliance on previous data it stands to reason that each of the steps should be covered in full before moving onto the next step, however this only applies to the mean, when we come to working out the standard deviation as well as the Pearsons coefficient these can be done at the same time, opting to instead do the summation in one for loop instead of multiple reducing the amount of time taken to process. After implementing my Serial version the completion time was 0.008630990982.

2 Initial analysis of the problem (Parallel)

To find the Pearson coefficient as efficiently as possible it required multiple steps, firstly it required the splitting of the global array into equal parts for each process, to do this I used scatter on each array which made use of tree distribution and was therefore more efficient than other methods i.e broadcast or individual send commands. After splitting I decided to calculate the sum of the values sent to each process, this would give me their local sums, which could then be gathered by process 0 using the reduce function. After finding the global sum, it was necessary to use this to work out the global average for each array, this was done by process 0 and then sent to each process by a broadcast command. Each process now having the mean of each array as well as their local chunks of the global array were able to now calculate (2) up to the size of the block of the global array given to each process (n).

$$\sigma_n = \sqrt{\frac{\sum(A)}{n}} \tag{1}$$

$$A_p = (x_1 - mean(X))^2 +_(x_2 - mean(X))^2 .... + (x_{n'} - mean(X))^2 \tag{2}$$

Now that we have A_p for each process we needed to sum all of these to find out A, this was done through a reduce operation. After receiving the global sum it was now possible for process 0 to calculate the standard deviation of each array. This did not need to be sent to each process as it was un-needed due to the fact that Pearson required the sum of local products (3) which could be calculated by each process using their own blocks and sent to process 0 who would then calculate the value of Pearson's and output it.

$$(x_i - mean(C)) * y_i - mean(Y) \tag{3}$$

Overall in this implantation we required:

- 2 Scatters (Splitting the arrays)
- 5 Reduces (reducing the sum of each block for each array, reducing (2) for each array and finally reducing (3))
- 2 Broadcasts (To broadcast the avg for each array)

Resulting in 8 synchronization steps overall.

2.1 Improvements of the initial implementation

Looking into this implementation it was clear to see that there were improvements to be made, for example:

- reducing / broadcasting the values for each array in two steps instead of 1.
  - Grouping these results into one array and reducing/broadcasting them at the same time meant that we could reduce the synchronization steps by 1 for each message, resulting in 3 less synchronization steps overall.
- returning the value of the standard deviation before calculating (3).
  - Due to the standard deviation not being needed till after calculating (3) it would be possible to send the local (2) along with (3) allowing for 1 less synchronization step.

After these changes we were left with:

- 2 Scatters (Splitting the arrays)
- 2 Reduces (Summing A_p for each array as well as Summing (3) for each array)
- 1 Broadcasts (To broadcast the avg for each array)

Resulting in only 5 synchronization steps. However this could be improved upon even more. After noticing that these synchronization steps seemed to be very computationally heavy, it had me wondering if creating extra work for each process would actually reduce the processing time to do this I decided to make use of the "allReduce()" method which summed the values of the reduce and sent it to all process's this was done for the sum of each process's block of the global array, after receiving this data it was to calculate the mean locally, this reduced our overall synchronization steps by 1 resulting in.

- 2 Scatters (Splitting the arrays)
- 1 allReduce (Summing each processors block)
- 1 Reduce (Summing (3) for each array)

Resulting in only 4 synchronization steps overall opposed to the previous 8 synchronization steps required previously. This reduced processing time considerably.

2.2 Problems of the implementation

Using this implementation it occurred to me that scatter only splits array sizes which are divisible by the number of process's properly. For example with an array size of 6 and 4 process's would result in $6/4 = 1.5$ however blocks of data can only be given in whole numbers. Therefore each process would be given blocks of length 2 which would be incorrect resulting in there being repeat/erroneous data. After running this scenario it resulted in a Standard deviation for each array of 0, and a PearsonC of ∞. Because of this it brought me into my second implementation "Padding".

3 Implementation 2 Padding

After realizing that my first implementation had the chance of giving incorrect results it became necessary to look at ways to fix this. My initial thought was "padding" this is where we expand our dataset so that it is divisible, in our previous example we thought about an array size of 6 with 4 process's to make this divisible we would simply expand our dataset by 2, assigning these pieces of data a value of NaN (Not a Number), since these pieces of data have fixed values we are able to check if the data that we sent our processors were NaN, if they were then we ignored them. This however created idle time within our system, going back to our example of an array size of 8 with 4 process's where 2 of these pieces of data were NaN the data would be split to each process like so.

$$A1 = [sin(0), sin(1), sin(2), sin(3), sin(4), sin(5), NaN, NaN] \tag{4}$$
$$A2 = [sin(5), sin(6), sin(7), sin(8), sin(9), sin(10), NaN, NaN] \tag{5}$$
$$P1 = [sin(0), sin(1)], [sin(5), sin(6)] \tag{6}$$
$$P2 = [sin(2), sin(3)], [sin(5), sin(6)] \tag{7}$$
$$P3 = [sin(4), sin(5)], [sin(5), sin(6)] \tag{8}$$
$$P4 = [NaN, NaN], [NaN, NaN] \tag{9}$$

As you can see process 4 is doing no meaningful work at all, and instead is just there for looks, this idle time could be reduced by distributing the data like so.

$$A1 = [sin(0), sin(1), sin(2), sin(3), sin(4), sin(5), NaN, NaN] \tag{10}$$
$$A2 = [sin(5), sin(6), sin(7), sin(8), sin(9), sin(10), NaN, NaN] \tag{11}$$
$$P1 = [sin(0), sin(1)], [sin(5), sin(6)] \tag{12}$$
$$P2 = [sin(2), sin(3)], [sin(7), sin(8)] \tag{13}$$
$$P3 = [sin(4), NaN], [sin(9), NaN] \tag{14}$$
$$P4 = [sin(5), NaN], [sin(10), NaN] \tag{15}$$

Which would reduce the time for P3 to complete its workload however this was not possible by using a "padded" method and would instead need something more complex, "scatterv".

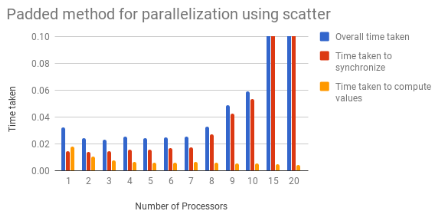
4 Implementation 2 ScatterV

This final implementation found me using a built in function in the MPI package called ScatterV, this specific method allowed for "chunks" to be given dynamically rather than of a fixed size using a displacement value as well as the amount of data that each process would be sent although this took each process loop to calculate it did mean that I no longer had to check if values were NaN as well as having no wasted processors. I decided to give process 0 the least work possible i.e $floor(data/processors)$ due to it having the greatest amount of tasks to do outside of synchronized tasks i.e calculating standard deviation as well as Pearson. After making sure that the results returned by this implementation were correct I decided to do a critical analysis of the runtime of the "padded" and "scatterV" implementation to decide which method scaled better.

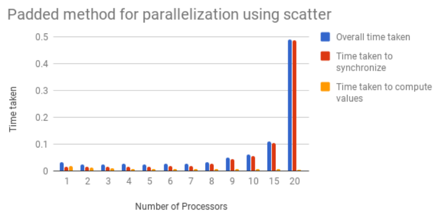
5 Critical analysis

For this section we look at the overall runtime of each system as well as the time taken for communication to take place using these two values it was also possible to determine the time taken for calculations to be made by each process. For each test I have taken the minimum value for an average of 23 different run's of the system, this was due to the platform being unpredictable with the way process time was split, sometimes resulting in differences greater than 1s.

5.1 Padded Implementation



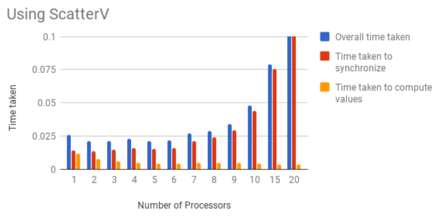
(a) Process number 1-15



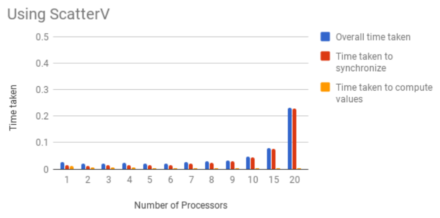
(b) Process number 1-20

Figure 1: Data from Appendix 1

5.2 scatterV Implementation



(a) Process number 1-15



(b) Process number 1-20

Figure 2: Data from Appendix 1

5.3 Comparison

Looking at figure 1 we can see that the overall time taken for the system to finish gets lower as we increase the number of processors to 3 this hold for figure 2 as well, however after this point they both level out till we increase the number of processors to 8 when looking at figure 1 and 7 when looking at figure 2, this is due to the communication only taking slightly more time with n+1 processors however the time taken to calculate the final value within this time decreases substantially therefore the increase in the communication time becomes meaningless, however after increasing the number of processors above 3 the time saved during computation becomes negligible however our communication time becomes significant meaning that the overall time is heavily reliant on the number of processors at higher numbers. However when comparing these two charts we can see that the growth rate of the padded implementation is much higher than that of our scatterv version and therefore hindered more by an increase in the number of processors. Finally in comaparison with out serial version who's completion time was 0.008630990982 we can see that neither of our parrelel versions come near to this I believe this is due to the overhead of not just synchronization but also due to the creation of process's.

6 Appendix

6.1 1: Tables of padded runs.

	1 Process			2 Process's			3 Process's			4 Process's		
#Run	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values
1	0.0324139595	0.01438212395	0.01803183556	0.02405309677	0.01373386383	0.01031923294	0.02535820007	0.01444292068	0.01091527939	0.02698802948	0.01701688766	0.009971141815
2	0.0323779583	0.01436305046	0.01801490784	0.02399897575	0.01372814178	0.01027083397	0.02614402771	0.01836919785	0.007774829865	0.02742600441	0.01735782623	0.01006817818
3	0.0327539444	0.0146021843	0.0181517601	0.02405405045	0.0137360096	0.01031804085	0.02403402328	0.01631903648	0.007714986801	0.02725410461	0.02069497108	0.00655913353
4	0.0331389904	0.01474618912	0.01839280128	0.02399516106	0.01366591454	0.01032924652	0.02393102646	0.01623606682	0.007694959641	0.02774310112	0.01775097847	0.00999212265
5	0.0326628685	0.01452708244	0.01813578606	0.02403283119	0.01370096207	0.01033186913	0.02333903313	0.01557469368	0.007764339447	0.02759289742	0.02118992805	0.00640296936
6	0.03250479698	0.01445531845	0.01804947853	0.02399301529	0.01364588737	0.01034712791	0.02396202087	0.01625704765	0.007704973221	0.02714800835	0.0171380043	0.01001000404
7	0.03247094154	0.01443314552	0.01803779602	0.02410697937	0.01367092133	0.01043605804	0.02544999123	0.01453900337	0.01091098785	0.02824401855	0.01835584641	0.00988817215
8	0.03266191483	0.01449608803	0.0181658268	0.0240778923	0.01365590096	0.01042199135	0.02822089195	0.01608109474	0.01213979721	0.02692723274	0.01695108414	0.009976148605
9	0.03230500221	0.01435399055	0.01795101166	0.02412104607	0.0136756897	0.01044535637	0.0232758522	0.01560020447	0.007675647736	0.02829313278	0.01845192909	0.00984120369
10	0.03306698799	0.01476383209	0.0183031559	0.02399516106	0.01364779472	0.01034736633	0.02412486076	0.01638197899	0.007742881775	0.02694296837	0.02040481567	0.006538152695
11	0.03236103058	0.01434779167	0.01801323891	0.02398300171	0.01363658905	0.01034641266	0.02332282066	0.01555037498	0.007772445679	0.02701711655	0.01702594757	0.009991168976
12	0.0324010849	0.01437497139	0.01802611351	0.02413105965	0.01371502876	0.01041603088	0.02542686462	0.01451897621	0.01090788841	0.02697896957	0.02047324181	0.006505727768
13	0.03231406212	0.01433801651	0.01797604561	0.02397584915	0.01366186142	0.01031398773	0.02393507957	0.01623916626	0.007695913315	0.02672791481	0.02024269104	0.00648522377
14	0.0322470665	0.0143289566	0.01791810989	0.02405309677	0.01372385025	0.01032924652	0.02531909943	0.01438784599	0.01093125343	0.02783608437	0.01791787148	0.009918212891
15	0.03227210045	0.01436400414	0.01790809631	0.02405500412	0.0137360096	0.01031899452	0.02326393127	0.01551795006	0.007745981216	0.02683615685	0.02028989792	0.006546258926
16	0.03268194199	0.01444125175	0.01824069023	0.02404093742	0.01372289658	0.01031804085	0.02630496025	0.01856207848	0.007742881775	0.02705216408	0.02051615715	0.006536006927
17	0.03261804581	0.01446986198	0.01814818382	0.02733397484	0.01693677902	0.01039719582	0.02328491211	0.01553678513	0.007748126984	0.02562308311	0.01915097237	0.006472110748
18	0.03243803978	0.01434469223	0.01809334755	0.03377199173	0.0166079998	0.01716399193	0.02397894859	0.01627993584	0.007699012756	0.02578401566	0.01920580864	0.006578207016
19	0.03225803375	0.01433420181	0.01792383194	0.02738308907	0.01692223549	0.01046085358	0.02553081512	0.01448702812	0.011043787	0.0267829895	0.02036595345	0.006417036057
20	0.03227901459	0.01436209679	0.0179169178	0.03010606766	0.01510024071	0.01500582695	0.02318406105	0.01551222801	0.007671833038	0.02707195282	0.01707696915	0.009994983673
21	0.03258895874	0.01443886757	0.01815009117	0.0240111351	0.01372599602	0.01028513908	0.02406096458	0.01633191109	0.007729053497	0.0264248848	0.01987886429	0.006546020508
22	0.03241014481	0.01437020302	0.01803994179	0.02395796776	0.01362895966	0.0103290081	0.02313899994	0.01546287537	0.007676124573	0.02746415138	0.02106285095	0.00640130043
23	0.03231811523	0.01434206963	0.01797604561	0.02409291267	0.01374006271	0.01035284996	0.02537798882	0.01443767548	0.01094031334	0.02542686462	0.01540708542	0.01001977921
Min	0.0322470665	0.0143289566	0.01790809631	0.02395796776	0.01362895966	0.01027083397	0.02313899994	0.01438784599	0.007671833038	0.02542686462	0.01540708542	0.00640130043
Max	0.0331389904	0.01476383209	0.01839280128	0.03377199173	0.01693677902	0.01716399193	0.02822089195	0.01856207848	0.01213979721	0.02829313278	0.02118992805	0.01006817818
Avg	0.03250195669	0.01443391261	0.01806804408	0.02501409987	0.01416172152	0.01085237835	0.02452040755	0.01576635112	0.008754056433	0.02702547156	0.01886637315	0.008159098418

	5 Process			6 Process's			7 Process's			8 Process's		
#Run	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values
1	0.02606201172	0.01736593246	0.008696079254	0.02536320686	0.01741194725	0.007951259613	0.02766704559	0.0200111866	0.007655858994	0.1397161484	0.1332540512	0.006462097168
2	0.02465415001	0.01595926285	0.008694887161	0.02557015419	0.01761078835	0.007959365845	0.05396103859	0.04699993134	0.006961107254	0.05992507935	0.03249406815	0.0274310112
3	0.02551698685	0.01670098305	0.008816003799	0.02551698685	0.01814389229	0.007373094559	0.04123282433	0.03412485123	0.007107973099	0.05437207222	0.04788780212	0.006484270096
4	0.02462100983	0.01594209671	0.008678913116	0.02550411224	0.01808691025	0.007417201996	0.04476594925	0.02713918686	0.01762676239	0.03290200233	0.02668190002	0.00622010231
5	0.02655506134	0.01794314384	0.008611917496	0.0422809124	0.0366590023	0.005621910095	0.04637098312	0.0393512249	0.007019758224	0.07165288925	0.06568598747	0.005966901779
6	0.07689213753	0.06799292564	0.008899211884	0.02562904358	0.01775813103	0.007870912552	0.06568598747	0.05937981606	0.006306171417	0.08241319656	0.07605099678	0.006362199783
7	0.026045084	0.01745009422	0.008594989777	0.02535796165	0.01740980148	0.007948160172	0.04472398758	0.03815603256	0.006567955017	0.05267119408	0.04631185532	0.00635933876
8	0.02513504028	0.01716947556	0.007965564728	0.02487897873	0.01689720154	0.007981777191	0.1098358631	0.1028289795	0.007006883621	0.04383206367	0.03707885742	0.006753206253
9	0.0247130394	0.01604413986	0.008668899536	0.02542185783	0.01806712151	0.007354736328	0.052973032	0.04650521278	0.006467819214	0.04636979103	0.03961515427	0.006754636765
10	0.02516317368	0.01720190048	0.007961273193	0.02485203743	0.01694703102	0.007905006409	0.03598308563	0.0289478302	0.007035255432	0.09296298027	0.07647776604	0.01648521423
11	0.02526092529	0.01729226112	0.007968664169	0.02526688576	0.01728606224	0.007980823517	0.05302000046	0.04654192924	0.006478071213	0.05667090416	0.04974007607	0.006930828094
12	0.02546596527	0.01751112938	0.007954835892	0.02553200722	0.01810097694	0.007431030273	0.04838013649	0.04147076607	0.006909370422	0.08676099777	0.08055901527	0.006201982498
13	0.02620720863	0.01819396019	0.008013248444	0.02536606789	0.01801729202	0.007348775864	0.0339550972	0.02697610855	0.006978988647	0.6839289665	0.6778190136	0.006109952927
14	0.02601480484	0.01734900475	0.008665800095	0.02530694008	0.01797008514	0.007336854935	0.03994703293	0.03237199783	0.007575035095	0.05744695663	0.05086708069	0.006579875946
15	0.02402091026	0.01812124252	0.00589966774	0.02556395531	0.01767992973	0.007884025574	0.05381894112	0.04681706429	0.007001876831	0.07307600975	0.05592417717	0.01715183258
16	0.02509617805	0.01647996902	0.00861620903	0.02561783791	0.01813793182	0.007479906082	0.3005468845	0.2937688828	0.006778001785	0.06511497498	0.05864310265	0.00647187233
17	0.02601504326	0.01807403564	0.007941007614	0.0254778862	0.0175538063	0.007924079895	0.03775000572	0.0306520462	0.007097959518	0.04708790779	0.03037786484	0.01671004295
18	0.02610301971	0.01747608185	0.008626937866	0.02483892441	0.01733660698	0.007502317429	0.1270968914	0.1201341152	0.006962776184	0.04907798767	0.04260492325	0.006473064423
19	0.02606892586	0.01741695404	0.008651971817	0.02485895157	0.01692080498	0.007938146591	0.05393099785	0.04692602158	0.007004976273	0.06094479561	0.05459284782	0.006351947784
20	0.02551984787	0.01678919792	0.008730649948	0.02479195595	0.0168159008	0.007976055145	0.0419459343	0.03494882584	0.006997108459	0.04525494576	0.03800654411	0.007248401642
21	0.024477005	0.01584911346	0.008627891541	0.02574205399	0.01783275604	0.007909297943	0.02516198158	0.01738715172	0.007774829865	0.06228089333	0.04580187798	0.01647901535
22	0.02423787117	0.01550769806	0.008730173111	0.02554011345	0.01766586304	0.007874250412	0.08674407005	0.07978200912	0.006962060928	0.03349208832	0.02682876587	0.006663322449
23	0.02612996101	0.01748991013	0.008640050888	0.02542090416	0.01750206947	0.007918834686	0.1115260124	0.09456777573	0.01695823669	0.08340382576	0.0769238472	0.006479978561
Min	0.02402091026	0.01550769806	0.00589966774	0.02479195595	0.0168159008	0.005621910095	0.02516198158	0.01738715172	0.006306171417	0.03290200233	0.02668190002	0.005966901779
Max	0.07689213753	0.06799292564	0.008899211884	0.0422809124	0.0366590023	0.007981777191	0.3005468845	0.2937688828	0.01762676239	0.6839289665	0.6778190136	0.0274310112
Avg	0.02765110265	0.0192748049	0.008376297743	0.02607390155	0.01842660489	0.007647296657	0.06682712099	0.05894734548	0.007879775503	0.09049385527	0.0813142424	0.009179612865

	9 Process			10 Process's			15 Process's			20 Process's		
#Run	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values
1	0.08716201782	0.08129811287	0.005863904953	1.106508017	1.100953817	0.005554199219	0.1081531048	0.1035778522	0.004575252533	0.8158159256	0.7979660034	0.01784992218
2	0.09321784973	0.08783912659	0.005378723145	0.08653211594	0.07073020935	0.01580190659	0.5190241337	0.5145430565	0.004481077194	0.6981019974	0.6941149235	0.003987073898
3	0.6473472118	0.6414778233	0.00586938858	0.2028970718	0.1972138882	0.00568318367	0.6222140789	0.6176409721	0.004573106766	0.938615799	0.9345910549	0.004024744034
4	0.06155800819	0.05557179451	0.005986213684	0.3797240257	0.3737971783	0.005926847458	0.1859660149	0.1814770699	0.004488945007	0.5267682076	0.5226540565	0.004114151001
5	0.06598687172	0.06046032906	0.005526542664	0.1986029148	0.1926112175	0.005991697311	0.5999860764	0.5954179764	0.004568099976	0.9530630112	0.9490690231	0.003993988037
6	0.3409409523	0.3350858688	0.005855083466	0.3815960884	0.3659040928	0.01569199562	0.7159979343	0.7114841938	0.00451374054	0.7416050434	0.7375478745	0.004057168961
7	0.104460001	0.09867429733	0.005785703659	0.1202759743	0.1145339012	0.005742073059	0.4215638638	0.4166727066	0.00489115715	0.6998541355	0.6960389614	0.003815174103
8	0.06804800034	0.06204009056	0.006007909775	0.1335449219	0.1282939911	0.005250930786	0.2478599548	0.2432961464	0.004563808441	0.8627309799	0.8587388992	0.003992080688
9	0.3261599541	0.3200259209	0.006134033203	0.1915380955	0.1859369278	0.005601167679	0.2458291054	0.2306818962	0.01514720917	0.5056698322	0.4876768589	0.01799297333
10	0.04848408699	0.04230690002	0.006177186966	0.1122179031	0.1066188812	0.005599021912	0.3483328819	0.3438129425	0.004519939423	0.5209860802	0.5062580109	0.01472806931
11	0.08233904839	0.07622170448	0.006117343903	0.2272820473	0.2216846943	0.005597352982	0.2017068863	0.1971693039	0.004537582397	0.8032000065	0.7992460728	0.003953933716
12	0.09478616714	0.08913803101	0.005648136139	0.06430506706	0.05854392052	0.005761146545	0.1968379021	0.1823511124	0.0144867897	0.6392350197	0.6339428425	0.0052921772
13	0.05705094337	0.05115199089	0.005898952484	0.1223840714	0.1166479588	0.005736112595	0.3627598286	0.3584139347	0.00434589386	0.8940119743	0.8900341988	0.003977775574
14	0.08756113052	0.08125400543	0.006307125092	0.1564850807	0.1506359577	0.005849123001	0.4185590744	0.414018631	0.00454044342	0.5697619915	0.5517909527	0.01797103882
15	0.1950860023	0.1888298988	0.006256103516	0.1838991642	0.1780190468	0.005880117416	0.8077509403	0.7933056355	0.01444530487	0.4901361465	0.4860098362	0.004126310349
16	0.3437259197	0.3379421234	0.00578379631	0.08966112137	0.0836918354	0.005969285965	0.1819360256	0.177601099	0.004334926605	1.655081987	1.651091099	0.003990888596
17	0.1701409817	0.1642849445	0.00585603714	0.1795461178	0.1639568806	0.01558923721	0.2749490738	0.2704532146	0.004495859146	0.9695298672	0.9375326633	0.03199720383
18	0.07740187645	0.07143902779	0.005962848663	0.08079886436	0.07489490509	0.005903959274	0.2271010876	0.2125871181	0.01451396942	1.050199032	1.046234846	0.003964185715
19	0.3132679462	0.3072328568	0.006035089493	0.1933689117	0.1876339912	0.005734920502	0.6220078468	0.617470026	0.004537820816	1.142481089	1.123975277	0.01850581169
20	0.09597396851	0.08979511261	0.006178855896	0.07466316223	0.06900691986	0.005656242371	0.532597065	0.5279436111	0.004653453827	1.128226995	1.124287844	0.003939151764
21	0.334651947	0.3285229206	0.006129026413	0.6317389011	0.6260859966	0.00565290451	0.5045599937	0.5001020432	0.004457950592	0.9481370449	0.930109024	0.01802802086
22	0.08919095993	0.0832130909	0.005977869034	0.05915617943	0.05338215828	0.005774021149	0.3231859207	0.3187229633	0.004462957382	0.7516798973	0.7476918697	0.003988027573
23	0.05132508278	0.04521584511	0.006109237671	0.2756609917	0.2588248253	0.01683616638	0.4580328465	0.453417778	0.004615068436	1.346945047	1.343036652	0.003908395767
Min	0.04848408699	0.04230690002	0.005378723145	0.05915617943	0.05338215828	0.005250930786	0.1081531048	0.1035778522	0.004334926605	0.4901361465	0.4860098362	0.003815174103
Max	0.6473472118	0.6414778233	0.006307125092	1.106508017	1.100953817	0.01683616638	0.8077509403	0.7933056355	0.01514720917	1.655081987	1.651091099	0.03199720383
Avg	0.166776823	0.1608270355	0.005949787472	0.2283646438	0.2208523128	0.007512331009	0.3968222452	0.3905287515	0.006293493768	0.8544277005	0.8456364715	0.008791228999

6.2 2: Tables of ScatterV results.

	1 Process			2 Process's			3 Process's			4 Process's		
#Run	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values
1	0.02640199661	0.01466584206	0.01173615456	0.02151298523	0.01364088058	0.007872104645	0.02384996414	0.0152451992	0.008604764938	0.02123713493	0.01446557045	0.006771564484
2	0.02626109123	0.01450014114	0.01176095009	0.02155303955	0.01366996765	0.007883071899	0.02170109749	0.015843153	0.005857944489	0.02039384842	0.01532292366	0.005070924759
3	0.02635407448	0.01453995705	0.01181411743	0.02151203156	0.01365423203	0.00785779953	0.02369809151	0.0151052475	0.008592844009	0.02253103256	0.01574587822	0.006785154343
4	0.02855014801	0.01672101021	0.0118291378	0.02151703835	0.01364994049	0.007867097855	0.02375483513	0.01504087448	0.008713960648	0.02291512489	0.01606726646	0.006847858429
5	0.02647805214	0.01454591751	0.01193213463	0.02140402794	0.013641119	0.007762908936	0.02152395248	0.01566267014	0.005861282349	0.02296590805	0.01608800888	0.00687789917
6	0.02641510963	0.01458835602	0.01182675362	0.02153897285	0.01371479034	0.00782418251	0.02149200439	0.01563405991	0.005857944489	0.02266788483	0.01590800285	0.006759881973
7	0.02623105049	0.01442909241	0.01180195808	0.02150893211	0.0136449337	0.007863998413	0.02384614944	0.01524376869	0.008602380753	0.02246212959	0.01569890976	0.006763219833
8	0.02645516396	0.01436686516	0.0120882988	0.02147102356	0.01361823082	0.00785279274	0.02389788628	0.01530337334	0.008594512939	0.02063393593	0.01558089256	0.005053043365
9	0.02606487274	0.01430916786	0.01175570488	0.0214869976	0.01366496086	0.007822036743	0.02163696289	0.01575374603	0.005883216858	0.02190208435	0.01691222191	0.004989862442
10	0.02636790276	0.01456212997	0.01180577278	0.02142715454	0.01371479034	0.007712364197	0.0214881897	0.01540803909	0.006080150604	0.0205180645	0.0154440403	0.0050740242
11	0.02642011642	0.01465392113	0.0117661953	0.02151083946	0.01365399361	0.007856845856	0.02144408226	0.01551032066	0.005933761597	0.02252602577	0.01527810097	0.007247924805
12	0.02627491951	0.01442813873	0.01184678078	0.02622485161	0.01844620705	0.007778644562	0.02384018898	0.01521682739	0.008623361588	0.02304506302	0.01604604721	0.006999015808
13	0.02618098259	0.0144598484	0.01172113419	0.02147388458	0.01365184784	0.007822036743	0.02374410629	0.01502013206	0.008723974228	0.02209687233	0.01710891724	0.004987955093
14	0.02677106857	0.01471304893	0.01205801964	0.02153682709	0.01369404793	0.00784277916	0.02166008949	0.01578593254	0.005874156952	0.02344799042	0.01608991623	0.007358074188
15	0.02625489235	0.0144033432	0.01185154915	0.02151203156	0.0136680603	0.007843971252	0.02132105827	0.01543164253	0.005889415741	0.02212190628	0.01713418961	0.004987716675
16	0.02631187439	0.01446986198	0.01184201241	0.02161717415	0.01366972923	0.007947444916	0.02162694931	0.01576399803	0.005862951279	0.02273917198	0.0160009861	0.006738185883
17	0.0262761116	0.01445603371	0.0118200779	0.02133798599	0.01361489296	0.007723093033	0.02154898643	0.01568484306	0.005864143372	0.02126097679	0.01459884644	0.006662130356
18	0.02669501305	0.01461100578	0.01208400726	0.02132487297	0.01358580589	0.007739067078	0.02151107788	0.01562714577	0.005883932114	0.02245402336	0.01534080505	0.007113218307
19	0.02613997459	0.01434707642	0.01179289818	0.02146792412	0.01373577118	0.007732152939	0.02127504349	0.01540803909	0.005867004395	0.02292203903	0.0160586834	0.006863355637
20	0.02614212036	0.01429867744	0.01184344292	0.02133011818	0.01361322403	0.00771689415	0.02126002312	0.01539587975	0.005864143372	0.02285814285	0.01604008675	0.006818056107
21	0.02622008324	0.01443910599	0.01178097725	0.02152013779	0.01365804672	0.007862091064	0.02340292931	0.01541590691	0.0079870224	0.02242612839	0.0152759552	0.007150173187
22	0.026252985	0.01443719864	0.01181578636	0.02144193649	0.01370573044	0.007736206055	0.02375197411	0.01515698433	0.008594989777	0.02304911613	0.01623988152	0.006809234619
23	0.02655291557	0.01464176178	0.01191115379	0.0216190815	0.01373910904	0.007879972458	0.02382588387	0.01525020599	0.008575677872	0.02278804779	0.01599287987	0.006795167923
Min	0.02606487274	0.01429867744	0.01172113419	0.02132487297	0.01358580589	0.007712364197	0.02126002312	0.01502013206	0.005857944489	0.02291703224	0.01607179642	0.004987716675
Max	0.02855014801	0.01672101021	0.0120882988	0.02622485161	0.01844620705	0.007947444916	0.02389788628	0.015843153	0.008723974228	0.02243089676	0.0151860714	0.007358074188
Avg	0.02643793562	0.01459076094	0.01184717469	0.02168912473	0.0138717527	0.007817372032	0.02248267505	0.01543078215	0.007051892903	0.02248191833	0.01530003548	0.006414071373

	5 Process			6 Process's			7 Process's			8 Process's		
#Run	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values
1	0.04358386993	0.03755187988	0.006031990051	0.02115392685	0.01571583748	0.005438089371	0.02711892128	0.02142715454	0.005691766739	0.06187796593	0.05725002289	0.004627943039
2	0.02268600464	0.01653242111	0.006153583527	0.02109098434	0.01566910744	0.005421876907	0.03754281998	0.03232312202	0.005219697952	0.04545021057	0.04023790359	0.005212306976
3	0.03569293022	0.02978682518	0.005906105042	0.0212199688	0.0156879425	0.005532026291	0.03143596649	0.02611303329	0.005322933197	0.05311203003	0.04787898064	0.005233049393
4	0.02160596848	0.01524782181	0.006358146667	0.02136087418	0.01611924171	0.005241632462	0.03557395935	0.03030323982	0.005270719528	0.3786489964	0.3632779121	0.01537108421
5	0.02175188065	0.01527404785	0.006477832794	0.02109599113	0.01559591293	0.005500078201	0.03585386276	0.0304787159	0.005375146866	0.05378603935	0.04865717888	0.005128860474
6	0.02162098885	0.01524090767	0.006380081177	0.02136898041	0.01608014107	0.00528883934	0.08446002007	0.07917380333	0.005286216736	0.08448600769	0.07950520515	0.004980802536
7	0.02174592018	0.01570606232	0.006039857864	0.0211379528	0.01570463181	0.005433320999	0.03586506844	0.0305082798	0.005356788635	0.03927993774	0.03436613083	0.004913806915
8	0.02243900299	0.01595473289	0.006484270096	0.02108216286	0.01552700996	0.005555152893	0.03955483437	0.03444194794	0.005112886429	0.03596401215	0.03089189529	0.005072116852
9	0.0211930275	0.01652598381	0.004667043686	0.02104711533	0.01554107666	0.005506038666	0.0918469429	0.08655714989	0.005289793015	0.12641716	0.1215791702	0.004837989807
10	0.02121305466	0.01651620865	0.004696846008	0.02127599716	0.01579999924	0.005475997925	0.04592180252	0.04098677635	0.004935026169	0.106623888	0.1015648842	0.00505900383
11	0.0216281414	0.01561522484	0.006012916565	0.02176594734	0.01577305794	0.005992889404	0.03045487404	0.02535700798	0.005097866058	0.03472995758	0.0298538208	0.00487613678
12	0.02242588997	0.0158367157	0.006589174271	0.02150702477	0.01559329033	0.005913734436	0.08964800835	0.08436155319	0.005286455154	0.04290485382	0.03824710846	0.004657745361
13	0.03181695938	0.02733016014	0.00448679924	0.1295120716	0.1239469051	0.005565166473	0.04869580269	0.04370141029	0.004994392395	0.05166006088	0.04670095444	0.004959106445
14	0.02170610428	0.01535511017	0.00635099411	0.02187609673	0.01615166664	0.005724430084	0.04360413551	0.03855204582	0.005052089691	0.03291296959	0.02787995338	0.005033016205
15	0.02148318291	0.01508879662	0.006394386292	0.02173304558	0.01581788063	0.005915164948	0.02799105644	0.02271008492	0.005280971527	0.1506619453	0.1457650661	0.004896879196
16	0.0215408802	0.01512694359	0.006413936615	0.0217859745	0.01579904556	0.00598692894	0.04127502441	0.03545880318	0.005816221237	0.2075920105	0.2027311325	0.004860877991
17	0.0223338604	0.01585221291	0.006481647491	0.02163791656	0.01572561264	0.005912303925	0.04658198357	0.04132914543	0.005252838135	0.03577494621	0.03063583374	0.005139112473
18	0.02232694626	0.01585483551	0.006472110748	0.03023910522	0.0261387825	0.004100322723	0.0424349308	0.03667926788	0.005755662918	0.02885603905	0.0238969326	0.004959106445
19	0.02248811722	0.01592588425	0.006562232971	0.0217320919	0.01574516296	0.00598692894	0.04722189903	0.04183506966	0.005386829376	0.03520488739	0.02943992615	0.005764961243
20	0.03272509575	0.02293992043	0.009785175323	0.02168488503	0.01563644409	0.006048440933	0.3996288776	0.3944003582	0.00522851944	0.03608989716	0.03090786934	0.005182027817
21	0.02164196968	0.01575016975	0.005891799927	0.02171516418	0.01572084427	0.005994319916	0.03832888603	0.03299212456	0.005336761475	0.03802800179	0.03271698952	0.005311012268
22	0.03924107552	0.03327488899	0.005966186523	0.02195811272	0.01625108719	0.005707025528	0.03371691704	0.02843284607	0.005284070969	0.05384802818	0.04899430275	0.004853725433
23	0.02612996101	0.01748991013	0.008640050888	0.02164196968	0.01572108269	0.005920886993	0.04216694832	0.03693890572	0.005228042603	0.09683990479	0.09198403358	0.004855871201
Min	0.0211930275	0.01508879662	0.00448679924	0.02173686028	0.01568508148	0.004100322723	0.02711892128	0.02142715454	0.004935026169	0.02885603905	0.0238969326	0.004627943039
Max	0.04358386993	0.03755187988	0.009785175323	0.1295120716	0.1239469051	0.006048440933	0.3996288776	0.3944003582	0.005816221237	0.3786489964	0.3632779121	0.01537108421
Avg	0.02526177531	0.01894685496	0.006314920342	0.02654884173	0.02093312015	0.005615721578	0.06073580617	0.05543747156	0.005298334619	0.07959781522	0.0741288351	0.005468980126

	9 Process			10 Process's			15 Process's			20 Process's		
#Run	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values	Overall time taken	Time taken to synchronize	Time taken to compute values
1	0.2018561363	0.1972072124	0.004648923874	0.06706404686	0.06230688095	0.004757165909	0.2812080383	0.2668330669	0.01437497139	0.4191868305	0.4157910347	0.003395795822
2	0.214951992	0.2104158401	0.004536151886	0.0483880043	0.04378390312	0.004604101181	0.07885503769	0.07544493675	0.003410100937	0.3099229336	0.3065698147	0.003353118896
3	0.05425691605	0.04965591431	0.00460100174	0.08414506912	0.06957888603	0.01456618309	0.3243019581	0.3209691048	0.003332853317	0.4212539196	0.4178769588	0.003376960754
4	0.3359630108	0.3309881687	0.004974842072	0.06176686287	0.05737400055	0.00439286232	0.2822859287	0.2787210941	0.003564834595	0.4681220055	0.4647741318	0.003347873688
5	0.1129369736	0.1079368591	0.005000114441	0.1499090195	0.1455571651	0.004351854324	0.3419809341	0.3382749557	0.003705978394	0.4494380951	0.4460849762	0.003353118896
6	0.0892829895	0.08475708961	0.004525899887	0.04830598831	0.04370498657	0.00460100174	0.1987659931	0.1840832233	0.01468276978	0.5217750072	0.5183858871	0.003389120102
7	0.2530958652	0.2482788563	0.004817008972	0.1552481651	0.1508889198	0.0043592453	0.2624928951	0.2482168674	0.01427602768	0.4531888962	0.449760437	0.003428459167
8	0.03596305847	0.0309741497	0.004988908768	0.05437207222	0.04975390434	0.004618167877	0.3221480846	0.3182780743	0.003870010376	0.5490760803	0.5350980759	0.01397800446
9	0.1412580013	0.1364569664	0.004801034927	0.1376681328	0.1332292557	0.004438877106	0.1567158699	0.1529791355	0.00373673439	0.3441450596	0.3407518864	0.003393173218
10	0.05308294296	0.04818034172	0.004902601242	0.05102705956	0.04649567604	0.004531383514	0.2728588581	0.2690370083	0.003821849823	0.3782758713	0.374894619	0.003381252289
11	0.06465101242	0.06003308296	0.004617929459	0.08987402916	0.08545398712	0.004420042038	0.3138229847	0.3101520538	0.003670930862	0.4891338348	0.4857361317	0.003397703171
12	0.09591698647	0.09096002579	0.004956960678	0.08040595055	0.07580208778	0.004603862762	0.239287138	0.2354791164	0.003808021545	0.5445141792	0.5411372185	0.003376960754
13	0.04590010643	0.04089093208	0.005009174347	0.3248591423	0.3207709789	0.004088163376	0.2391660213	0.2353129387	0.003853082657	0.2323930264	0.2289361954	0.003456830978
14	0.1156449318	0.1110038757	0.004641056061	0.05376386642	0.04913187027	0.004631996155	0.3104760647	0.3067338467	0.003742218018	0.5707700253	0.5673668385	0.003403186798
15	0.05009102821	0.04510688782	0.004984140396	0.1646389961	0.1600060463	0.004632949829	0.3213989735	0.317699194	0.00369977951	0.2921590805	0.2745471001	0.01761198044
16	0.03526902199	0.03060007095	0.004668951035	0.06458282471	0.06043696404	0.004145860672	0.2696819305	0.2658557892	0.003826141357	0.2652790546	0.261661768	0.003617286682
17	0.04555988312	0.04096508026	0.004594802856	0.06347608566	0.05882120132	0.004654884338	0.3209969997	0.3173058033	0.003691196442	0.2703211308	0.2669100761	0.003411054611
18	0.1913621426	0.1865980625	0.004764080048	0.1556949615	0.1513819695	0.004312992096	0.3697030544	0.3547532558	0.01494979858	0.4086670876	0.3914349079	0.01723217964
19	0.04314398766	0.03861379623	0.004530191422	0.09589195251	0.09150290489	0.004389047623	0.2747619152	0.271048069	0.003713846207	0.5454671383	0.5420908928	0.003376245499
20	0.04755806923	0.04266905785	0.004889011383	0.06661391258	0.04902100563	0.01759290695	0.3035228252	0.2890472412	0.01447558403	0.4147801399	0.4113249779	0.003455162048
21	0.05605888367	0.05124926567	0.004809617996	0.05449509621	0.05004310608	0.004451990128	0.3825461864	0.3788638115	0.003682374954	0.4574379921	0.4540278912	0.003410100937
22	0.03416991234	0.02944779396	0.004722118378	0.06143689156	0.05707287788	0.004364013672	0.3142619133	0.3105349541	0.003726959229	0.5615239143	0.5581538677	0.003370046616
23	0.05132508278	0.04521584511	0.006109237671	0.06463098526	0.06016039848	0.004470586777	0.3454620838	0.3307909966	0.01467108727	0.3910498619	0.3739349842	0.0171148777
Min	0.03416991234	0.02944779396	0.004525899887	0.04830598831	0.04370498657	0.004088163376	0.07885503769	0.07544493675	0.003332853317	0.2323930264	0.2289361954	0.003347873688
Max	0.3359630108	0.3309881687	0.006109237671	0.3248591423	0.3207709789	0.01759290695	0.3825461864	0.3788638115	0.01494979858	0.5707700253	0.5673668385	0.01761198044
Avg	0.1030129972	0.09818283371	0.004830163458	0.09557648327	0.09009908593	0.005477397338	0.2837696386	0.2772354147	0.006534223971	0.4242557028	0.4185761161	0.005679586659

6.3 3: Results of serial version.

#Run	Overall time taken
1	0.01038002967834472656
2	0.00904202461242675781
3	0.00889897346496582031
4	0.00878882408142089844
5	0.00949597358703613281
6	0.01051902770996093750
7	0.00870299339294433594
8	0.01037216186523437500
9	0.00868296623229980469
10	0.00995707511901855469
11	0.01038289070129394531
12	0.00872206687927246094
13	0.00863099098205566406
14	0.00888490676879882812
15	0.01048302650451660156
16	0.01039099693298339844
17	0.00893092155456542969
18	0.00884199142456054688
19	0.00870108604431152344
20	0.01048493385314941406
21	0.00873708724975585938
22	0.00890111923217773438
23	0.00866913795471191406
Min	0.00890994071960449219
Max	0.01612114906311035156
Avg	0.00870394706726074219

6.4 4: Code listing

```
// Assignment 1 Comp 528, Robert Johnson, sgrjohn2@student.liverpool.ac.uk, 200962268
// Program to test and compare the runtime of calculating pearsons coefficient
// in parallel and serial, as well as comparing the differences in time between
// padded and unpadded versions of parallel implamentation.

#include <math.h>
#include <stdio.h>
#include <string.h>
#include <mpi.h>
#include <malloc.h>

// create inline functions to each method.
inline void runSerial();
inline void runParallelWithPadding();
inline void runParallelWithScatterV();


// initialize rank, numberofprocessors, ints for sendCount and recievecounts
int numProc, rank, sendCount, recvcount;
// set the size of all arrays
const int SIZE = 2000000;
// store the times of each process
double timeTakenSerial, t1, t2;
// create double arrays.
double *arrayA;
double *arrayB;



int main()
{
        // initialize mpi
        MPI_Init(NULL,NULL);
        // assign rank and processor number
        MPI_Comm_rank(MPI_COMM_WORLD, &rank);
        MPI_Comm_size(MPI_COMM_WORLD, &numProc);
                                                                //Run Serial Version
        if(rank==0)
        {
                printf("\nSerial\n");
                fflush(stdout);
                runSerial();
        }
                                                                //Run Padded Version
        if(rank==0)
        {
                printf("\nPadded\n");
                fflush(stdout);
        }
        MPI_Barrier(MPI_COMM_WORLD);
        runParallelWithPadding();
                                                                //Run ScatterV Version
        if(rank==0)
        {
                printf("\nScatterV\n");
                fflush(stdout);
        }
        MPI_Barrier(MPI_COMM_WORLD);
        runParallelWithScatterV();
        if(rank==0)
        {
                printf("\n");
        }
        MPI_Finalize();
        return 0;
}

//Serial version.
void runSerial()
{
        // allocate space for our array
        double *arrayA = malloc(SIZE*sizeof(double));
        double *arrayB = malloc(SIZE*sizeof(double));
        // fill our array
        for (int i = 0; i < SIZE; i++)
        {
                arrayA[i] = sin(i);
                arrayB[i] = sin(i+5);
        }
        // start timing
        t1 = MPI_Wtime();
        // calculate the sum of each array
        double sumOfA = 0;
        double sumOfB = 0;
    for (int i = 0; i < SIZE; i++)
    {
        double sumOfBTemp = sumOfB;
        double sumOfATemp = sumOfA;
        sumOfB = sumOfB + arrayB[i];
        sumOfA = sumOfA + arrayA[i];
        }
        // work out the means of both arrays
        double meanOfA = sumOfA/SIZE;
        double meanOfB = sumOfB/SIZE;
        // initialize SD and pearsons to 0.
        double standardDevA = 0;
        double standardDevB = 0;
        double pearsons = 0;
        // calculate part of SD and Pearsons
        for(int i = 0; i < SIZE; i++)
        {
                standardDevA = standardDevA + pow((arrayA[i] - meanOfA),2);
                standardDevB = standardDevB + pow((arrayB[i] - meanOfB),2);
                pearsons = pearsons + ((arrayA[i]-meanOfA)*(arrayB[i]-meanOfB));
        }
        // finish calulating SD and pearsons.
        standardDevA = sqrt(standardDevA/SIZE);
        standardDevB = sqrt(standardDevB/SIZE);
        pearsons = ((pearsons/SIZE)/(standardDevA*standardDevB));
        // free it
        free(arrayA);
        free(arrayB);
        // print it out
        printf("sumA = %lf, standDevA = %lf, averageA = %lf\nsumB = %lf, standDevB = %lf, averageB = "
                "%lf\nPearsons = %lf\n", sumOfA, standardDevA, meanOfA, sumOfB, standardDevB,
                meanOfB, pearsons);
        t2 = MPI_Wtime();
        printf("Time taken was %1.20f\n",t2-t1);
        fflush(stdout);
        timeTakenSerial = t2-t1;
}

// Method to calculate the pearsons coefficient using Padding
void runParallelWithPadding()
{
        // How much are we going to buffer the array
        int bufferCount = numProc - SIZE%numProc;
        if (bufferCount == numProc)
        {
                bufferCount = 0;
        }
        if(rank==0)
        {
                // assign the un modified array to size determined and fill it(decided to modify
                // it later to allow for a better comparison of each method)
                arrayA = malloc(SIZE*sizeof(double));
                arrayB = malloc(SIZE*sizeof(double));
                for(int j = 0; j < SIZE; j++)
                {
                        arrayA[j] = sin(j);
                        arrayB[j] = sin(j+5);
                }
                // start timer
                t1 = MPI_Wtime();
                // modify the arrays so that they are padded and assign NaN to extra space.
                if(bufferCount>0)
                {
                        arrayA = (double *)realloc(arrayA, (SIZE+bufferCount)*sizeof(double));
                        arrayB = (double *)realloc(arrayB, (SIZE+bufferCount)*sizeof(double));
                }
                for(int i = SIZE; i < SIZE+bufferCount; i++)
                {
                        arrayA[i] = NAN;
                        arrayB[i] = NAN;
                }
        }
        // set the amount we send to each process to be the padded size / number of processors
        sendCount = (SIZE+bufferCount)/numProc;
        recvcount = (SIZE+bufferCount)/numProc;
        // create space for the recieved arrays
        double *recvbufA = malloc(recvcount*sizeof(double));
        double *recvbufB = malloc(recvcount*sizeof(double));
        // scatter the data
        MPI_Scatter(arrayA,sendCount,MPI_DOUBLE,recvbufA,recvcount,
         MPI_DOUBLE,0,MPI_COMM_WORLD);
        MPI_Scatter(arrayB,sendCount,MPI_DOUBLE,recvbufB,recvcount,
         MPI_DOUBLE,0,MPI_COMM_WORLD);
        // create local array to store the local sums of both arrays
        double localSums[2];
        localSums[0]=0;
        localSums[1]=0;
        // as long as we don't find NaN keep adding, else assign the amount of actual number it was sent to be i.
        for (int i = 0; i < sendCount; i++)
        {
                if(!isnan(recvbufA[i]))
                {
                        localSums[0] += recvbufA[i];
                        localSums[1] += recvbufB[i];
                }
                else
                {
                        sendCount = i;
                }
        }
        // create array to store the global sum and reduce so we have the sum
        double globalSums[2];
        MPI_Allreduce(localSums,globalSums,2,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
        // work out averages on each process
        double averages[2];
        averages[0] = globalSums[0]/SIZE;
        averages[1] = globalSums[1]/SIZE;
        // create a local variable to store local standard deviation as well as local pearsons.
        double finalResults[3];
        for (int i = 0; i < sendCount; i++)
        {
                double TempA = recvbufA[i]-averages[0];
                double TempB = recvbufB[i]-averages[1];
                finalResults[0] += pow(TempA,2);
                finalResults[1] += pow(TempB,2);
                finalResults[2] += TempA*TempB;
        }
        // reduce to global standard deviation and pearsons
        double globalFinalResults[3];
        MPI_Reduce(finalResults,globalFinalResults,3,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
        // free our process buffers
        free(recvbufA);
        free(recvbufB);
        if(rank==0)
        {
                // calculate global standard deviation and pearsons.
                double standDevA = sqrt(globalFinalResults[0]/SIZE);
                double standDevB = sqrt(globalFinalResults[1]/SIZE);
                double pearsons = (globalFinalResults[2]/SIZE)/(standDevA*standDevB);
                // free global arrays.
                free(arrayA);
                free(arrayB);
                t2 = MPI_Wtime();
                // print out results
                printf("sumA = %lf, standDevA = %lf, averageA = %lf\nsumB = %lf, standDevB = %lf, averageB = "
                        "%lf\nPearsons = %lf\n", globalSums[0], standDevA, averages[0], globalSums[1], standDevB,
                        averages[1], pearsons);
                printf("Overall time taken is %1.20f\n",t2-t1);
                fflush(stdout);
                if(t2-t1 > timeTakenSerial)
                {
                        printf("Serial was %d%% quicker\n",(int)floor(((t2-t1)/timeTakenSerial)*100)-100);
                        fflush(stdout);
                }
                else
                {
                        printf("Parallel was %d%% quicker\n",(int)floor((timeTakenSerial/(t2-t1))*100)-100);
                        fflush(stdout);
                }
        }
}



// Method to calculate the pearsons coefficient using ScatterV
void runParallelWithScatterV()
{
        // create arrays for displacement value's and send amounts.
        int displs[numProc];
        int send_counts[numProc];
        // Only create space for arrays on process 0 as well as filling them
        if(rank==0)
        {
                arrayA = malloc(SIZE*sizeof(double));
                arrayB = malloc(SIZE*sizeof(double));
                for(int j = 0; j < SIZE; j++)
                {
                        arrayA[j] = sin(j);
                        arrayB[j] = sin(j+5);
                }
                t1 = MPI_Wtime();
        }
        // create a temporary variable to store the amount of process's
        int numProcTemp = numProc;
        // store the size of the array in temp
        int sizeTemp = SIZE;
        // set the displacement for process 0 to 0.
        displs[0] = 0;
        // set the amount that process 0 recieves to the floor of sizeTemp/numProcTemp
        send_counts[0] = sizeTemp/numProcTemp;
        // for each processor
        for(int i = 1; i < numProc; i++)
        {
                // find out the displacement of process i as well as how much it sends
                displs[i] = displs[i-1] + send_counts[i-1];
                sizeTemp -= send_counts[i-1];
                // we have 1 less processor to give tasks to
                numProcTemp--;
                // set the amount of data to send the process
                send_counts[i] = sizeTemp/numProcTemp;
        }
        // create a recieve buffer for all process's
        double *recvbufA = malloc(send_counts[rank]*sizeof(double));
        double *recvbufB = malloc(send_counts[rank]*sizeof(double));
        // scatter the initial arrays.
    MPI_Scatterv(arrayA, send_counts, displs, MPI_DOUBLE, recvbufA , send_counts[rank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
    MPI_Scatterv(arrayB, send_counts, displs, MPI_DOUBLE, recvbufB , send_counts[rank], MPI_DOUBLE, 0, MPI_COMM_WORLD);
    // create local array to store the local sums of both arrays
        double localSums[2];
        localSums[0] = 0;
        localSums[1] = 0;
        for (int i = 0; i < send_counts[rank]; i++)
        {
                localSums[0] += recvbufA[i];
                localSums[1] += recvbufB[i];
        }
        // create a global array to store the global sums, give it to each process.
        double globalSums[2];
        MPI_Allreduce(localSums,globalSums,2,MPI_DOUBLE,MPI_SUM,MPI_COMM_WORLD);
        // find out the average on each process.
        double averages[2];
        averages[0] = globalSums[0]/SIZE;
        averages[1] = globalSums[1]/SIZE;
        // create a local variable to store local standard deviation as well as local pearsons.
        double finalResults[3];
        finalResults[0]=0;
        finalResults[1]=0;
        finalResults[2]=0;
        for (int i = 0; i < send_counts[rank]; i++)
        {
                double TempA = recvbufA[i]-averages[0];
                double TempB = recvbufB[i]-averages[1];
                finalResults[0] += pow(TempA,2);
                finalResults[1] += pow(TempB,2);
                finalResults[2] += TempA*TempB;
        }
        // reduce to global standard deviation and pearsons
        double globalFinalResults[3];
        MPI_Reduce(finalResults,globalFinalResults,3,MPI_DOUBLE,MPI_SUM,0,MPI_COMM_WORLD);
        // free our process buffers
        free(recvbufA);
        free(recvbufB);
        if(rank==0)
        {
                // calculate global standard deviation and pearsons.
                double standDevA = sqrt(globalFinalResults[0]/SIZE);
                double standDevB = sqrt(globalFinalResults[1]/SIZE);
                double pearsons = (globalFinalResults[2]/SIZE)/(standDevA*standDevB);
                // free global arrays.
                free(arrayA);
                free(arrayB);
                t2 = MPI_Wtime();
                // print out results
                printf("sumA = %lf, standDevA = %lf, averageA = %lf\nsumB = %lf, standDevB = %lf, averageB = "
                        "%lf\nPearsons = %lf\n", globalSums[0], standDevA, averages[0], globalSums[1], standDevB,
                        averages[1], pearsons);
                printf("Overall time taken is %1.20f\n",t2-t1);
                fflush(stdout);
                if(t2-t1 > timeTakenSerial)
                {
                        printf("Serial was %d%% quicker\n",(int)floor(((t2-t1)/timeTakenSerial)*100)-100);
                        fflush(stdout);
                }
                else
                {
                        printf("Parallel was %d%% quicker\n",(int)floor((timeTakenSerial/(t2-t1))*100)-100);
                        fflush(stdout);
                }
        }
}
```