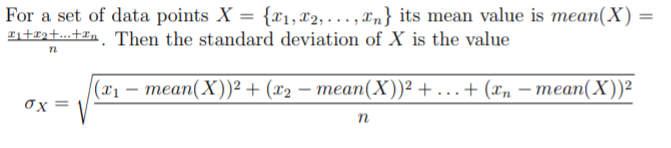
**Report**

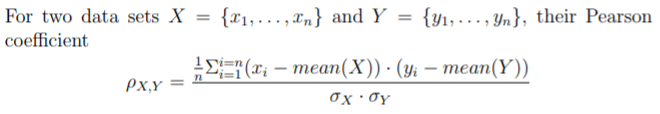
**The problem:**

The input data are given in two arrays a and b of values of type double and the length of each array is at least 2,000,000.

1.Use serial implementation without parallelism that implements the algorithm for computing Pearson correlation coefficient of data presented in arrays a and b;

2.Use OpenMp implementation for computing the same, that works on a varying number of MPI processes

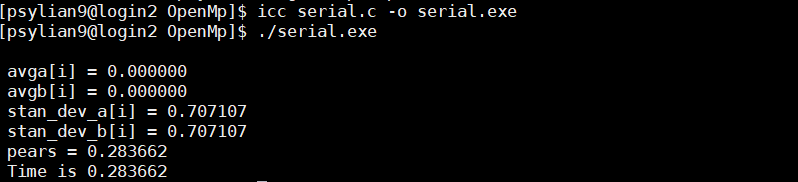




**Brief explanation of serial & OpenMP codes：**

**Serial codes**:

Firstly, I created two arrays which length is 2000000 as global variables and distribute sin( i ) and sin( i+5 ) to them respectively. Then, I defined two functions ( mean, calculateSD ) to calculate each array’s average, standard deviation. Furthermore, I defined and used the function (pearson) to compute the Pearson coefficient of two arrays. Also, I used “clock” function to calculate the time of run the whole serial codes. Finally, in the MobaXterm software, I used icc command to compile the c code to exe file and execute it. Diagram 1 shows the final output of the serial code.



**Diagram 1 the final output of the serial code**

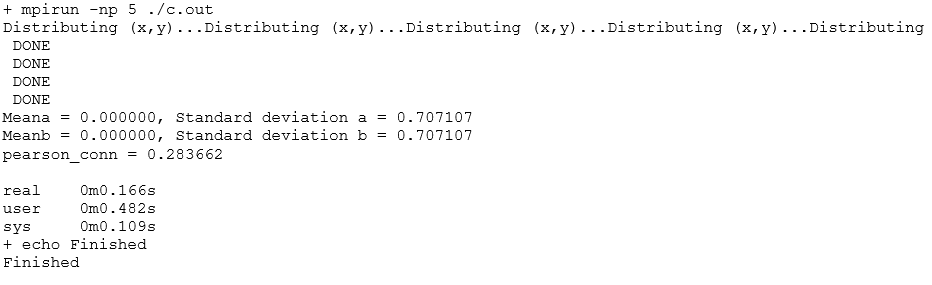
**OpenMp codes:**

I defined an variable n as an array’s length and used function “malloc()” to distribute memory space dynamically to two arrays \*a,\*b. Then, “for” loop in main function is to put “sin(i), sin(i+5)” into the two arrays. In master thread, omp\_get\_num\_threads() is to return the number of threads in the team currently and omp\_get\_wtime() is to get the time as start time of running parallel codes. Then, directives “#pragma omp parallel for default (shared) private (i) schedule (static, chunk) reduction ( +: xxx )” is to calculate the sum of each thread and regard it as the global variable. These directives could help to calculate the averages, standard deviation and the final Pearson coefficient. According to the function “omp\_get\_wtime()”, I counted the running time of OpenMp parallel codes. In the MobaXterm software, I used “ qrsh -l h\_rt=00:05:00 -cwd -V -pe smp 1 -l exclusive=true ./scaling-binding.sh 4 4 ” to run my interactive batch job in Chadwick. The script scaling-binding.sh will run my OpenMp application on a given number of processes (1st arg to script) for a given number of times (2nd argument to script) and output the times for the sequential and parallelized loops. The reason of using scaling-binding.sh is that it could distribute the data to each thread equally. Appendix 1,2 show running application on 1 process with 4 threads and 5 processes with 4 threads.

**Comparison:**

According to Appendix 1 and Diagram 1, the output values of Pearson coefficient of two codes are the same. The time of running OpenMp application on 1 process with 4 threads of 4 times are 0.009967 seconds, 0.009979 seconds, 0.009977 seconds, 0.009967 seconds respectively. Then, the time of running serial codes is 0.283662 seconds. Thus, the speed of running OpenMp parallel application is faster than that of serial application. So, **Speedup S = T(serial) / T(parallel) ≈28.43, Efficiency E = T(serial) / (p\*T(parallel)) ≈7.11**;

On the other hand, appendix 2 show that the time of running the OpenMp application on 5 processes with 4 threads of 4 times are 0.012262 seconds, 0.012096 seconds, 0.012298 seconds and 0.012612 seconds. Thus, the average time of 4 times is 0.012317. Compared to the time of running MPI application on 5 processes, the speed of running OpenMp parallel application is faster than that of MPI parallel application. Figure 1 show that the time of running MPI application on 5 processes.



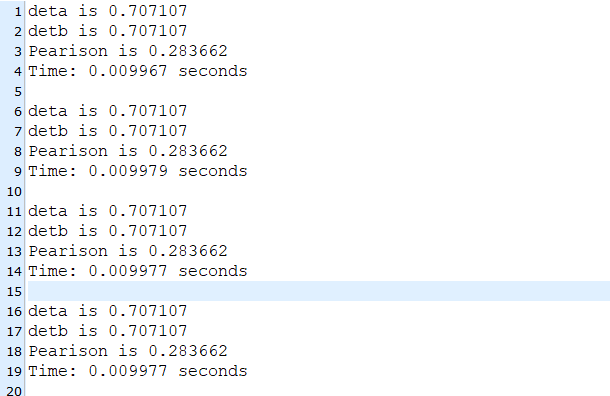
**Figure 1 MPI application running on 5 processes**

**Discussion:**

If the data size is large, using OpenMp parallel codes is more efficient than MPI parallel codes, while serial codes is suitable for computing the small data size because of costing time in distributing data to threads or processes.

I think in my OpenMp codes, it could scale to a few threads finished early, and a few threads lagged, which may improve the performance of OpenMp codes. In MPI codes, data sent from process 0 to other processes, while dataset distributed from master thread to other threads in one process in OpenMp codes. So, that’s maybe the reason why the performance of MPI codes is different from that of OpenMp codes.

**Appendix 1: running application on 1 process with 4 of 4 times.**



**Appendix 2: running application on 5 process with 4 of 4 times.**

