Assignment 1 Report

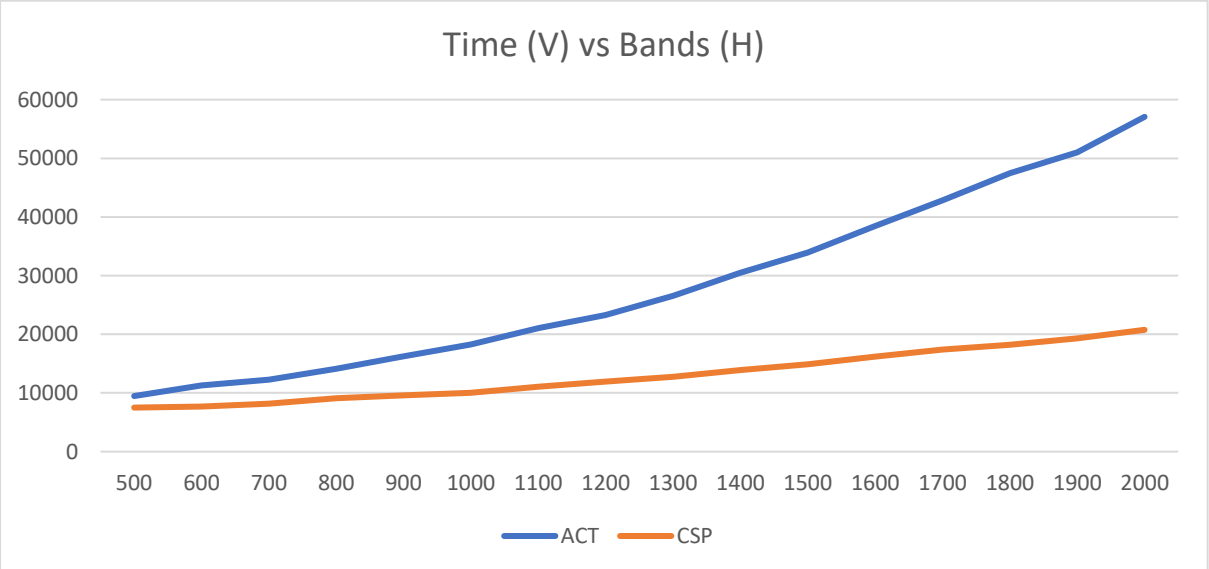
In the assignment one, we need to design a Longest Common Subsequence (as known as LCS) algorithm in sequential version and two parallel version. At the beginning to design the algorithm I choose use the diagonal wave-fronts way to design the algorithm. It will be more difficult to parallel compare with the matrix design because the message passed among grids will be the east row and south row rather than only the one in the south-east corner. Thus, even use the diagonal wave-fronts way to design the algorithm the boundary of each grid still needs to be built. Although it's hard to parallel, the reason why I choose it is that it can save the memory. For the diagonal wave-fronts design only need to save the number on the diagonal and boundary, so the memory needed is O(n). If use the matrix way to implement the algorithm, the memory needed is O(n^2), which means for lager input may cause memory overflow.

To parallel the algorithm, two different techniques (Actor and Channel) are used for different implementation. These two techniques are both asynchronous models of message passing, However the task weights are different. To compare the performance of them, two basic experiments were designed to compare the performance and scalability of the ACT and CSP.

Experimental environment: 8 logical cores 4 physical cores: Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz F# Compiler version 10.2.3 for F# 4.5

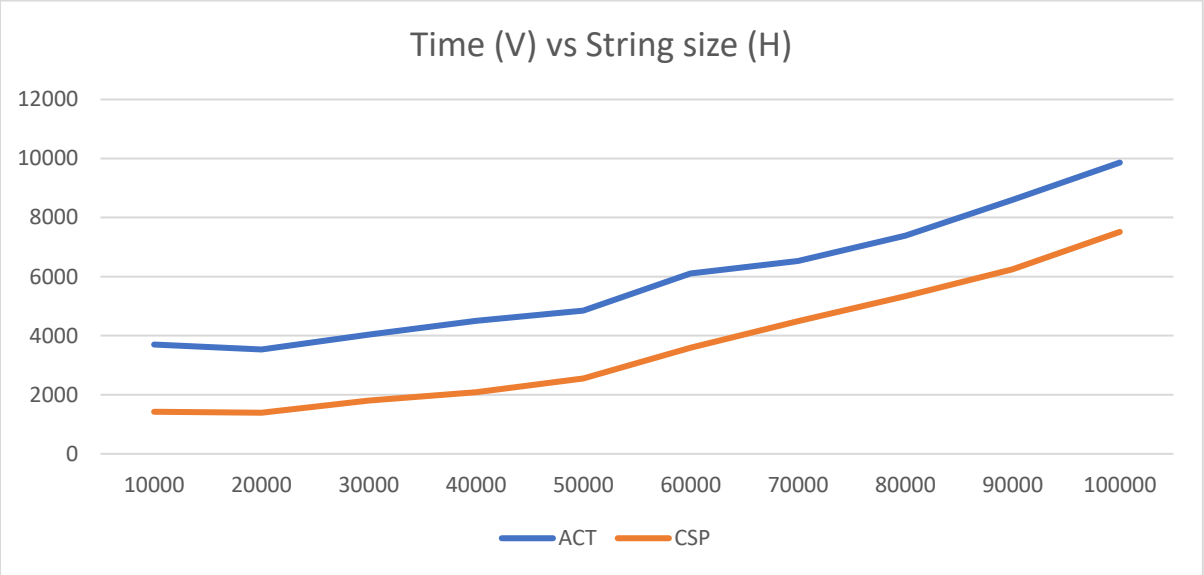
Experiment 1: The size of input string is large and fixed, which has the length of 101,000. The number of the bands increase from 500 to 2000. The result is shown below.

	500	600	700	800	900	1000	1100	1200
ACT	9470	11299	12258	14116	16262	18279	21073	23270
CSP	7504	7702	8188	9116	9563	10014	11048	11950
	1300	1400	1500	1600	1700	1800	1900	2000
ACT	26544	30476	33907	38473	42817	47461	50992	57066
CSP	12752	13900	14865	16186	17404	18220	19280	20761



Experiment 2: The number of bands is fixed, which is 500. The size of the input string increase from 100*100 to 1000*100. The result is shown below.

	10000	20000	30000	40000	50000	60000	70000	80000	90000	100000
ACT	3697	3534	4039	4505	4846	6109	6526	7387	8599	9863
CSP	1422	1392	1809	2085	2556	3595	4495	5339	6248	7516



The experiment results show that the performance of CSP is always better than that of ACT. The scalability of CSP for the size of string is similar with that of ACT. The scalability of CSP for the number of bands is better than ACT.

The performance and scalability of the CSP is better than the ACT is due to their different thread strategies and way of message passing. Actors use async tasks, directly mapped to heavyweight threads from the common thread pool. Channel uses lightweight job tasks, which can be logically blocked without affecting the system. Actors are based on asynchronous post/receive which is implemented with mailbox-type queues. Channel is based on synchronous channels, which do not need queues.