Sample Experimental Results for A#1

We recommend two basic experiments, which will be enough for a good report. In both cases, we empirically investigate the scalability of ACT (Mailbox Actors) vs CSP (Hopac) models. Please feel free to investigate and discuss additional interesting scenarios, but this won't be necessary.

Test #1. Inputs are large and fixed; number of bands is variable.

All experiments use the same two strings, given by input1.txt and input2.txt. We only vary the number of bands: **b1**,**b2** = 50,50; 100,100; 150,150; … 2000,2000 (keeping the same numbers of bands for each string).

Fig.1 shows possible results, if we follow the 1D model of Agents1Mailbox, which uses just one task for each horizontal band, i.e. **b1** tasks.

Fig.3 shows alternate results, if we follow the 2D model of Agents2Mailbox, which uses one task for each grid cell, i.e. **b1**×**b2** tasks - a dramatically larger number of tasks. Explanations? Conclusions?

Note that you were of course free to use any of these two models.

Test #2. Number of bands is fixed; size of strings is variable.

We use a fixed low number of bands - 100,100 – for which both Actors and CSP showed similar performance. We dynamically generate strings of varying size: 5000; 10000; 15000; … 100000, and we use the same string for both sides (sidebar: in this case, the LCS is the string length :).

Fig.2 shows possible results, if we follow the 1D model, i.e. we use 100 tasks. The results based on the 2D model do not essentially differ, just show a wider gap between ACT and CSP. Explanations? Conclusions?

Experimental environment. These experiments have been run on a supercomputer. Your own conclusions need not be the same, e.g. your results will depend on the model you used (1D vs 2D), the size of your inputs, the numbers of bands, the capabilities of your home computer or lab machine.

Our test machine: 48 logical cores
24 physical cores: Intel Xeon Silver 4116 @ 2.10 Gz
F# Compiler version 10.2.3 for F# 4.5

Disclaimer. A professional testing would also require: (i) running a warm-up before doing the measurements (cf. JIT compiling, processor caches); and (ii) averaging test results for several runs. You are welcome to do this, however you are not required to do this for this report.

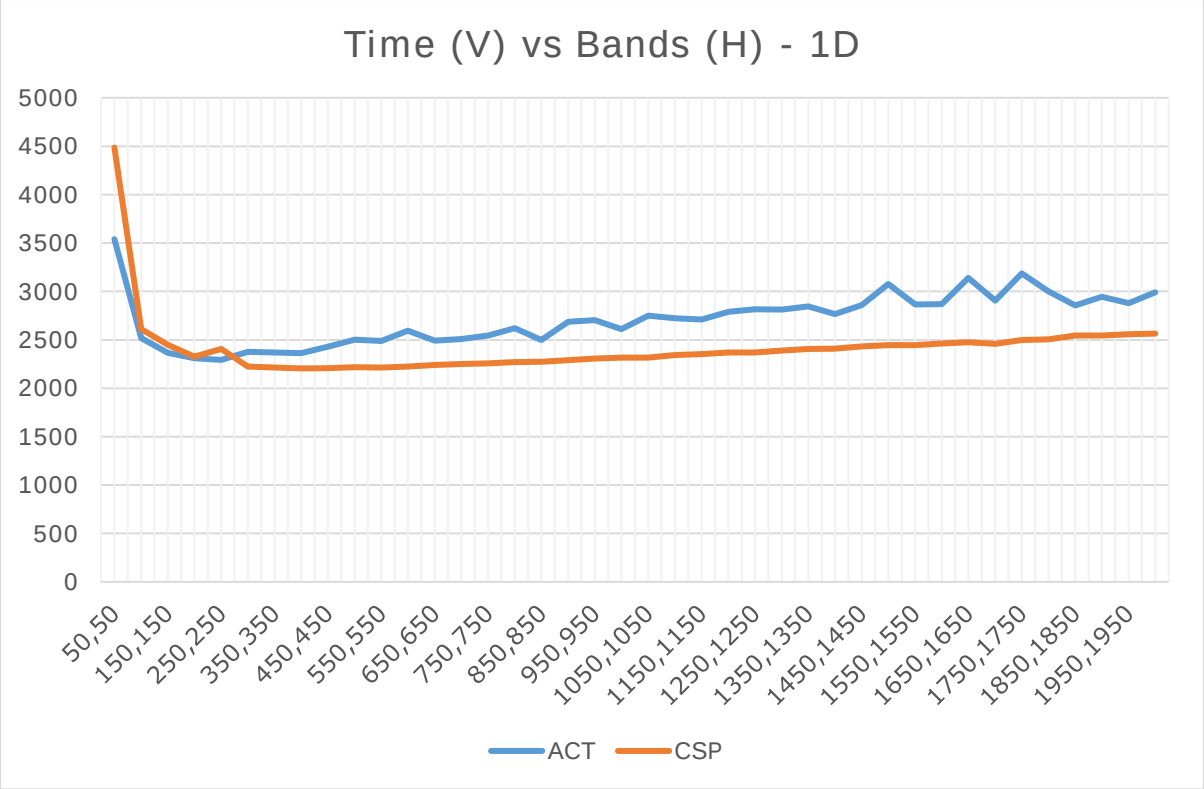

Figure 1 – 1D array of tasks

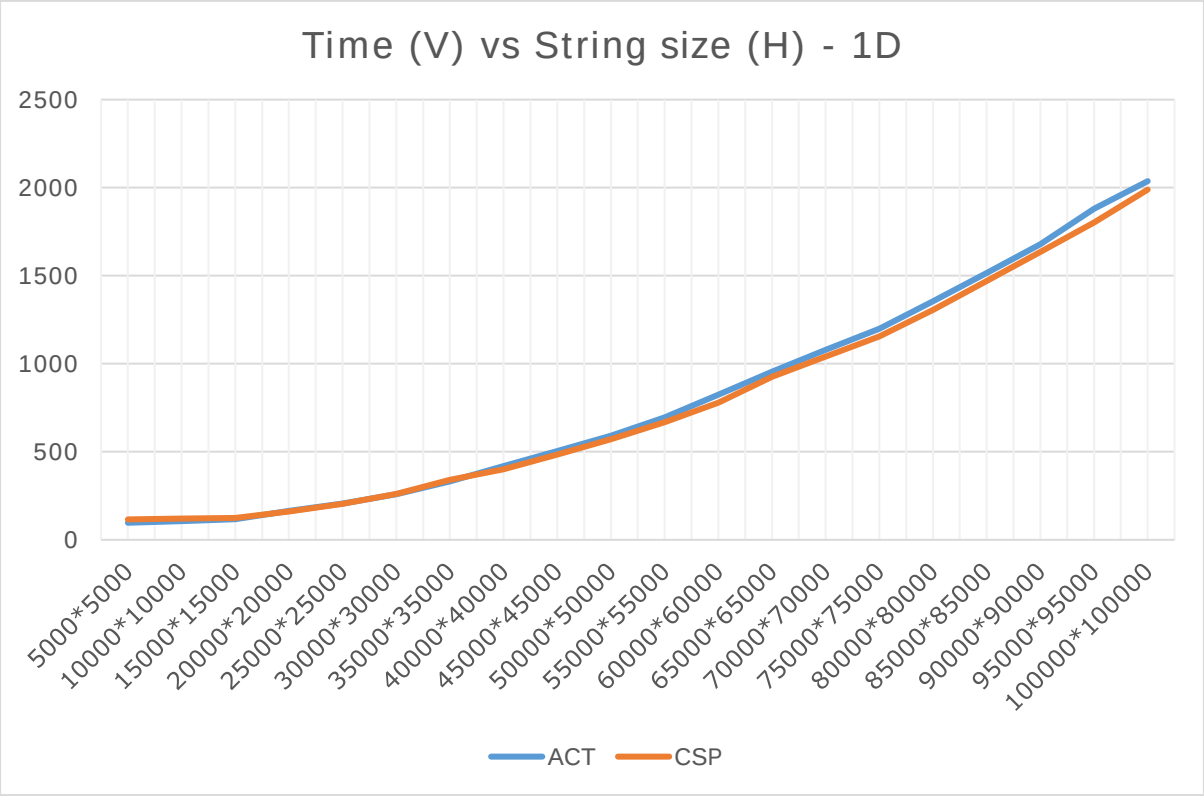

Figure 2 – 1D array of tasks

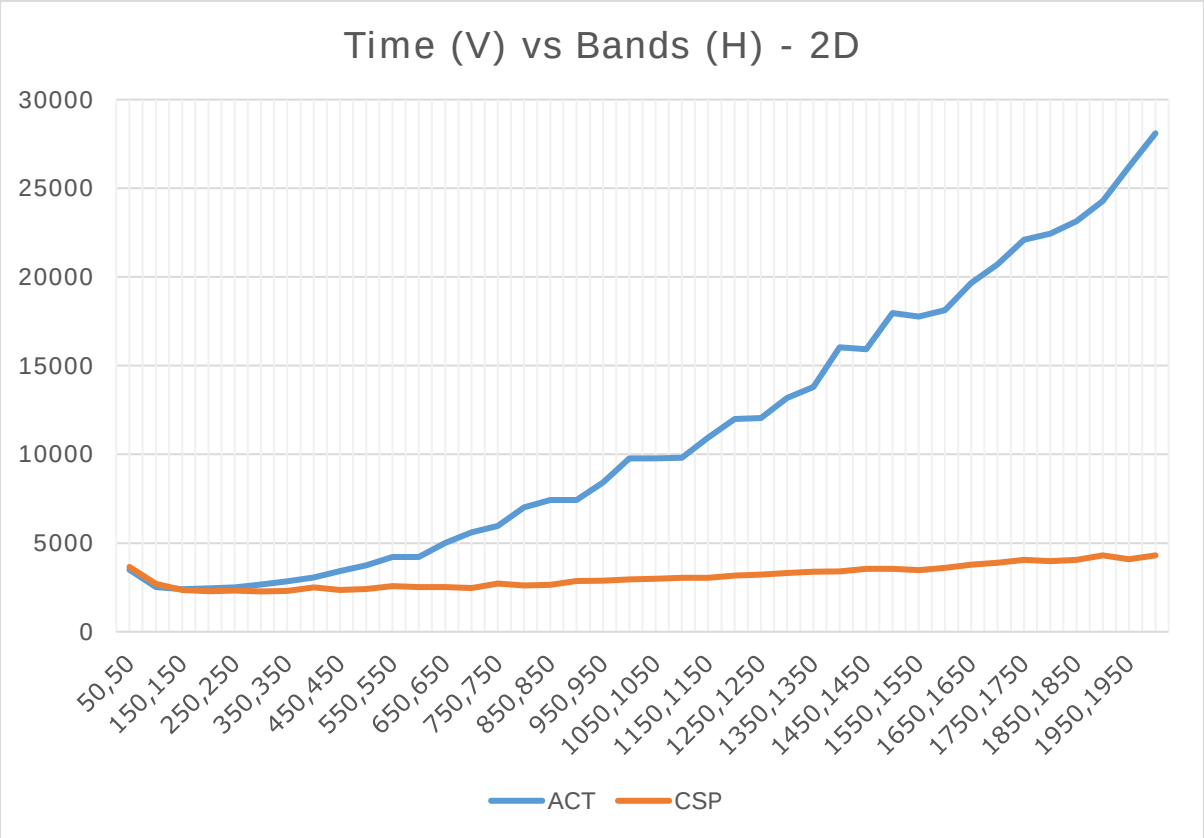

Figure 3 - 2D array of tasks

Appendix

Just added the results of similar runs on a typical lab machine, with 4 physical, 8 logical cores.

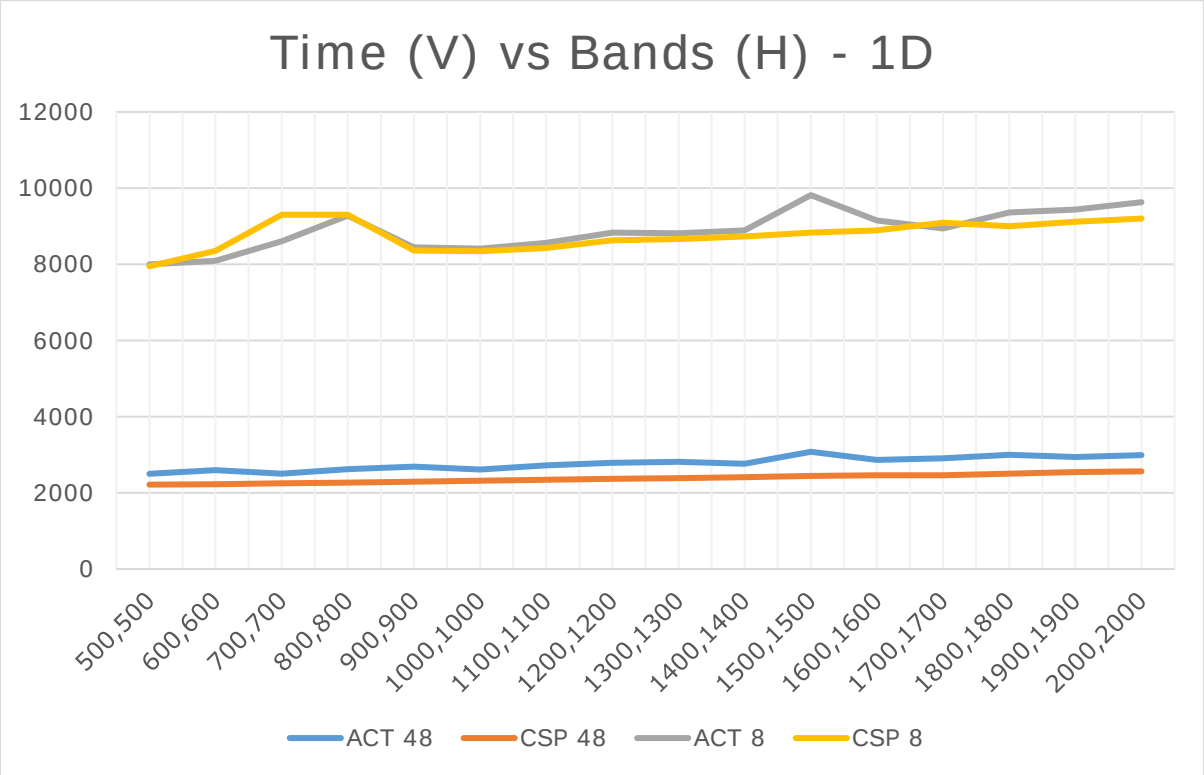

Figure 4

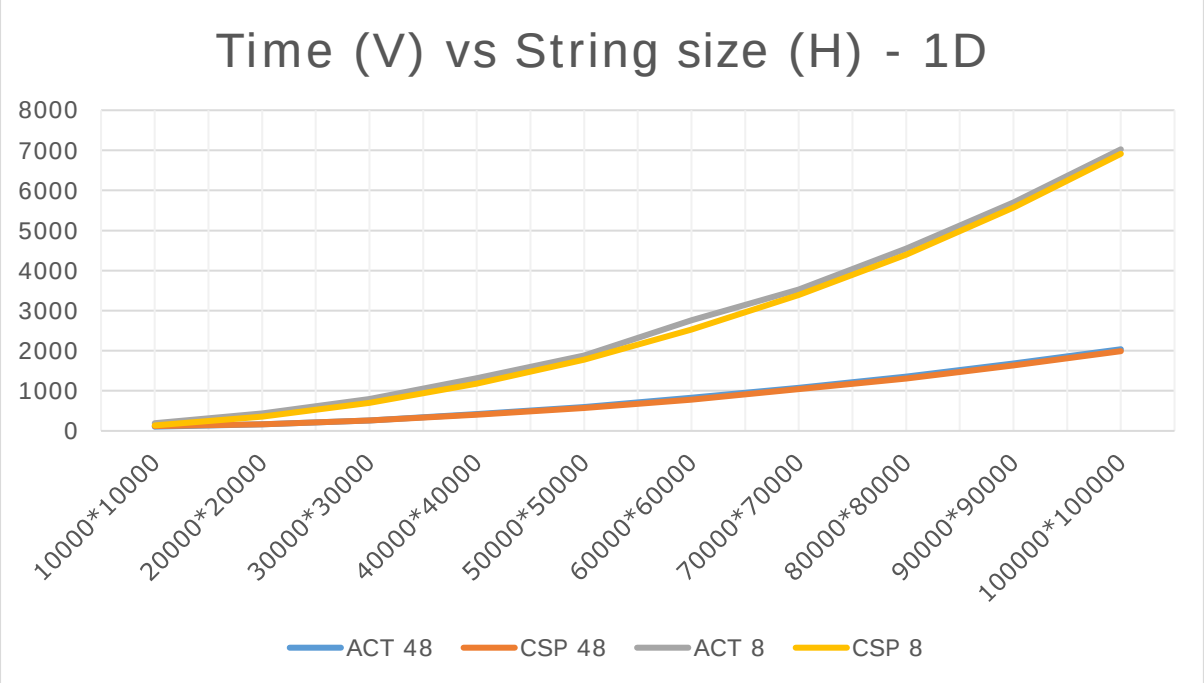

Figure 5

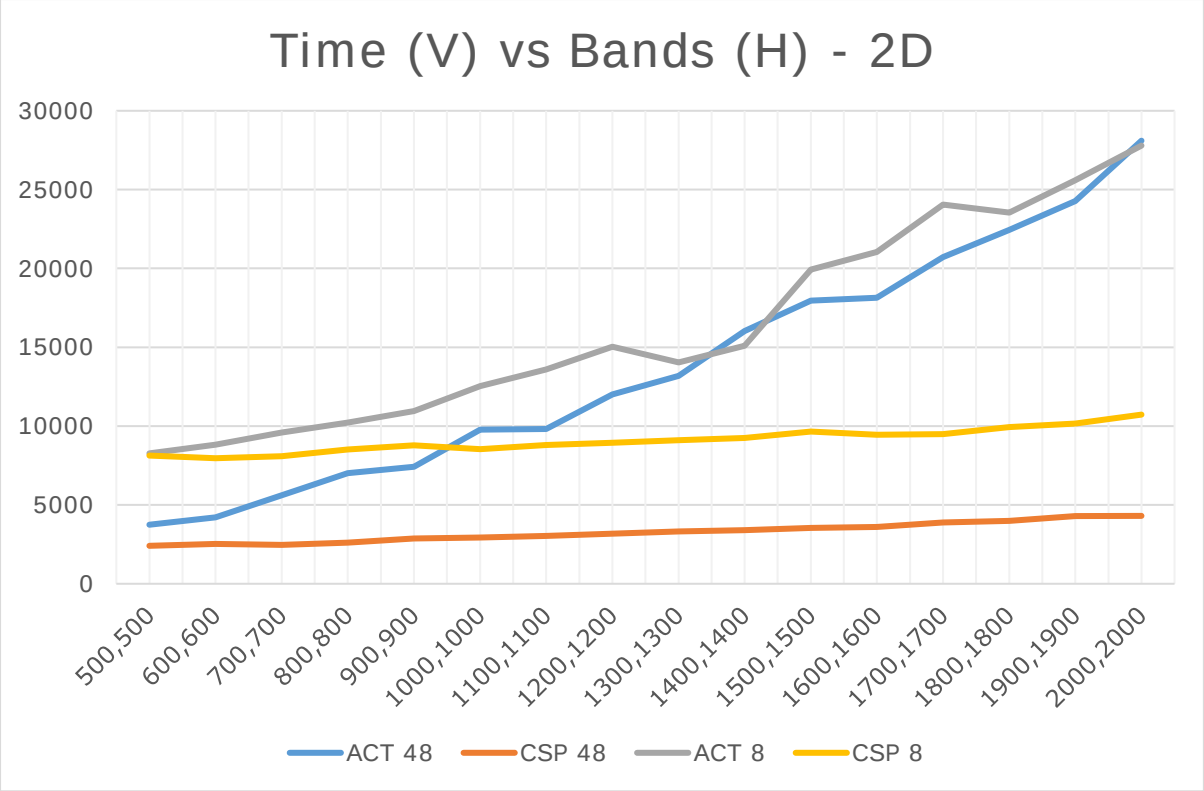
Time (V) vs Bands (H) - 2D
30000
25000
20000
15000
10000
5000
0
500,500
600,600
700,700
800,800
900,900
1000,1000
1100,1100
1200,1200
1300,1300
1400,1400
1500,1500
1600,1600
1700,1700
1800,1800
1900,1900
2000,2000
ACT 48
CSP 48
ACT 8
CSP 8

Figure 6

It is also interesting to look at the **Task Manager's Performance** tab and try to explain the CPU and memory loads. Fig.5 shows a capture for ACT 8 followed by CSP 8 on the lab machine, on the most stressing test used, b1=2000, b2=2000 (2D case, cf. above Fig.6).

Which one is faster? Which one achieves better CPU utilisation? Which one has lower and more constant memory requirements? Conversely, which one seems to need some garbage collection?

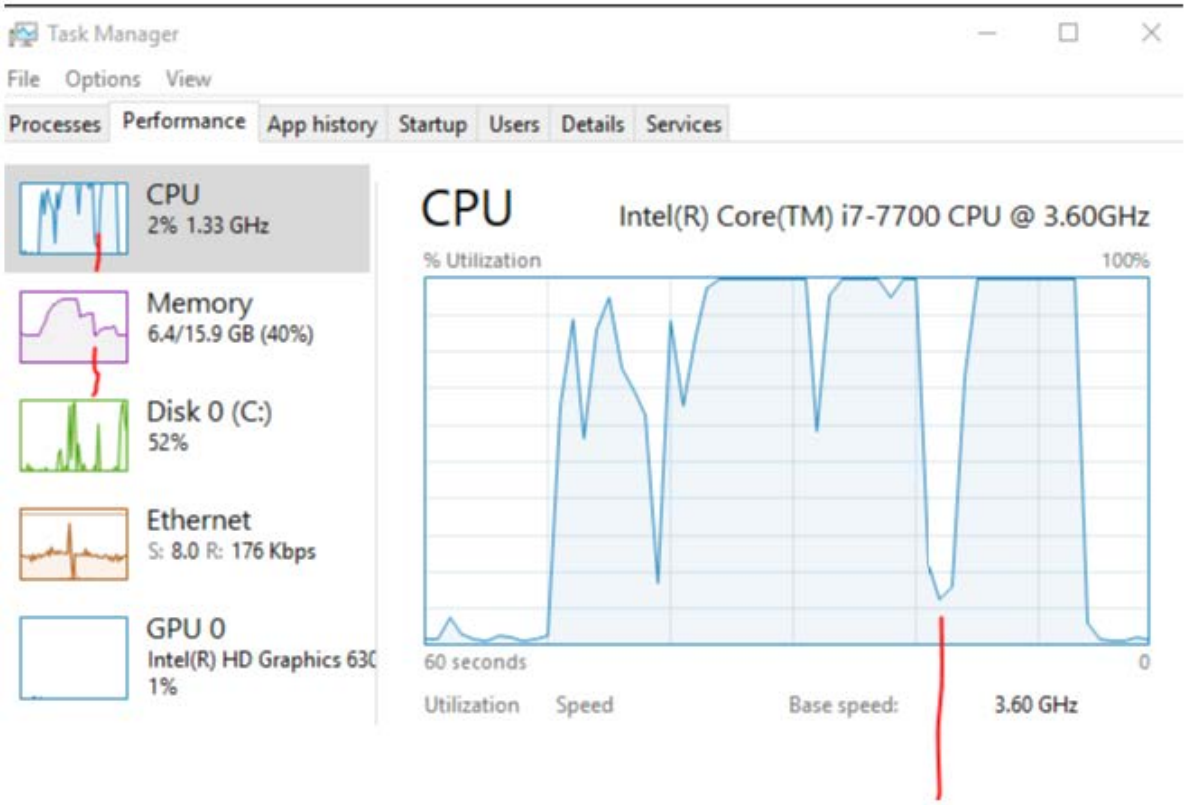

Figure 7

Batch files

Just in case you may find these useful, here are fragments of command-line batch files that we used and you could also adapt and use. We measured the actual algorithm times, using our duration function, w/o the input. The times have been printed to standard error.

Collect runtimes for a series of band numbers: 500, 600, … 2000

```
SET PROMPT=$G$S
echo. > _bands.log
for /L %%B in (500, 100, 2000) do (
        echo. >> _bands.log
        ECHO ... bands %%B,%%B >> _bands.log
        A1F.EXE /S1:input1.txt /S2:input2.txt /ACT:%%B,%%B,0 1>> _bands.log 2>&1
        timeout 2
        A1F.EXE /S1:input1.txt /S2:input2.txt /CSP:%%B,%%B,0 1>> _bands.log 2>&1
        timeout 5
)
```

Timeouts ensure a nice separation of tasks and allow some beneficial chip cooling between successive intensive operations.

Collect runtimes for a series of dynamically generated strings of sizes: 10000, 20000, … 100000:

```
SET PROMPT=$G$S
echo. > _sizes.log
for /L %%L in (100, 100, 1000) do (
        echo. >> _sizes.log
        ECHO ... sizes %%L*100,%%L*100 >> _sizes.log
        @IF EXIST inputs.txt DEL inputs.txt
        @for /L %%I in (1, 1, %%L) do @(
                @ECHO
0123456789012345678901234567890123456789012345678901234567890123456789
012345678901234567890123456>> inputs.txt
        )
        A1F.EXE /S1:inputs.txt /S2:inputs.txt /ACT:500,500,0 1>> _sizes.log 2>&1
        timeout 2
        A1F.EXE /S1:inputs.txt /S2:inputs.txt /CSP:500,500,0 1>> _sizes.log 2>&1
        timeout 5
)
```