CALCULATION OF THE SPEED-UP IN PARALLEL VERSIONS OF CODE_BRIGHT

This document describes a methodology to calculate speed up and efficiency when parallel runs are carried out with CODE_BRIGHT.

1) Run a model in CODE_BRIGHT using, for instance, four different CODE_BRIGHT versions: vn, vnpar2, vnpar4 and vnpar6 (available in the CODE_BRIGHT website). Remember that the parallelization works only when using the iterative solver. It does not make sense to use parallel versions for small problems that are solved using the "direct solver" option.

2) For each model, you will obtain a summary of the calculation, going to:

Calculate → View process info…

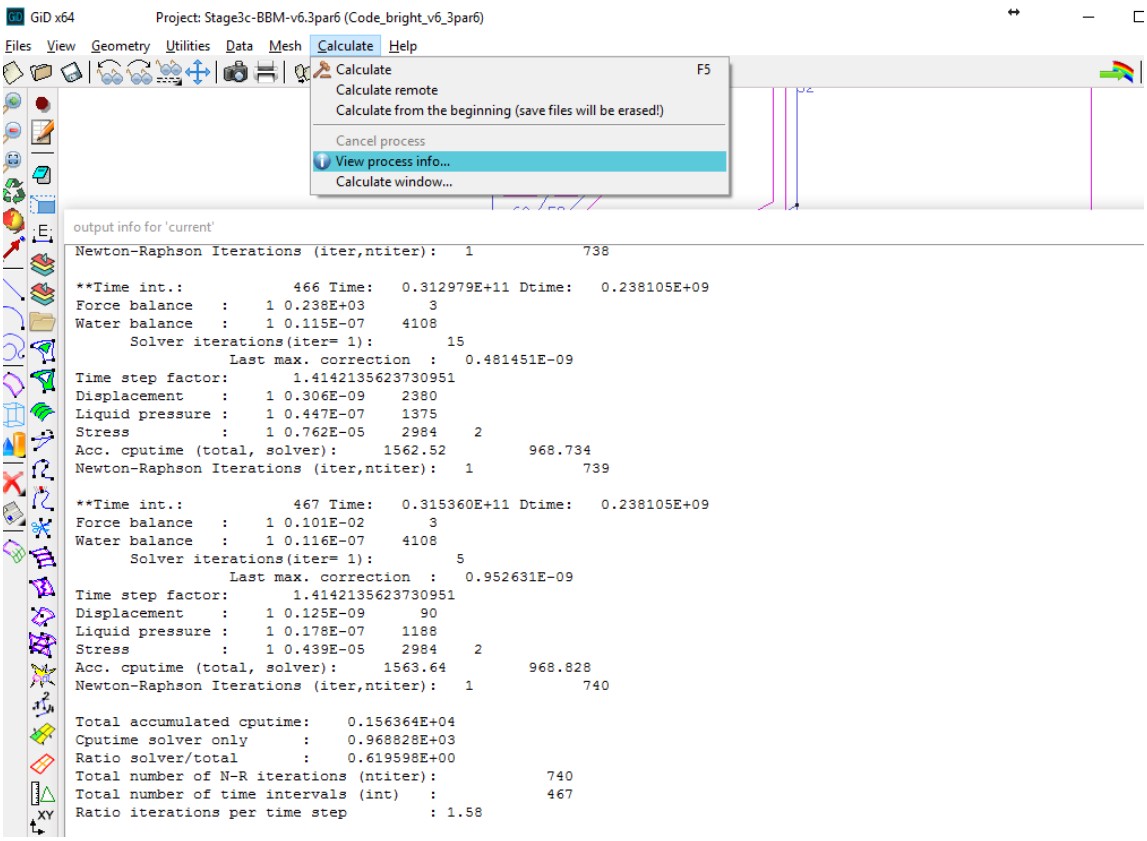


Figure 1. Snapshot of the "View process info…" produced by GiD.
Note that the run has finalized and a summary of information regarding elapsed times, number of time steps and number of iterations is shown.

3) Write down the *Total accumulated cputime* for each case. The cputime devoted to solver is also available and can be used in the same way.

4) Now, the **walk clock time** (real time that the model has taken to run) results from dividing the *Total accumulated cputime* by the number of threads (1 thread for vn, 2 threads for vnpar2, 4 threads for vnpar4 and 6 threads for vnpar6).

5) You may go to the project folder and on the "date modified" column compare the elapsed time between the "msh.dat" file and the "_gen.out" file, for verification.

If these two values are very different, it may mean that processors were not available for CODE_BRIGHT all time. Then, wall clock time from file time and date would be bigger than the internally calculated with CODE_BRIGHT which corresponds to time spent by processors.

6) Finally, to calculate the **speed-up** of each parallelized version, you just have to divide the walk clock time of the non-parallelized version and the walk clock time of each parallel version.

7) You may also calculate the **efficiency** of the parallelization, dividing the speed-up by the number of threads.

Figure 2 shows Amdahl's law, which indicates the expected speed-up that can be obtained. This value depends on the portion of the programs that is able to run in parallel. Figure 3 shows an example of calculation done by the Code_Bright Team.

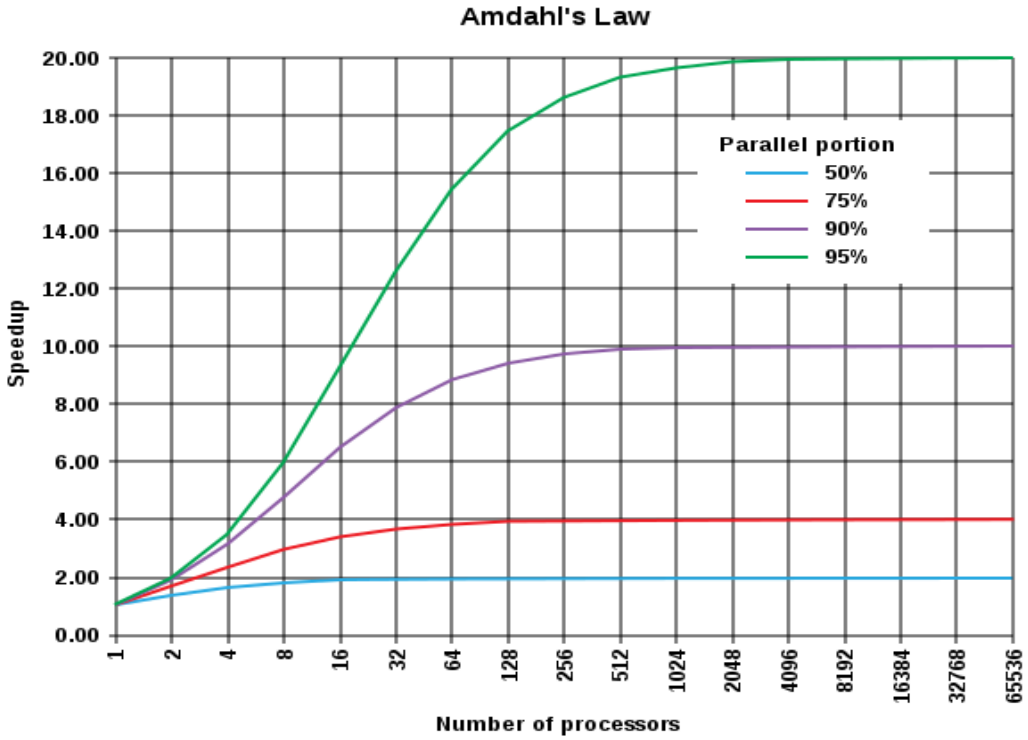
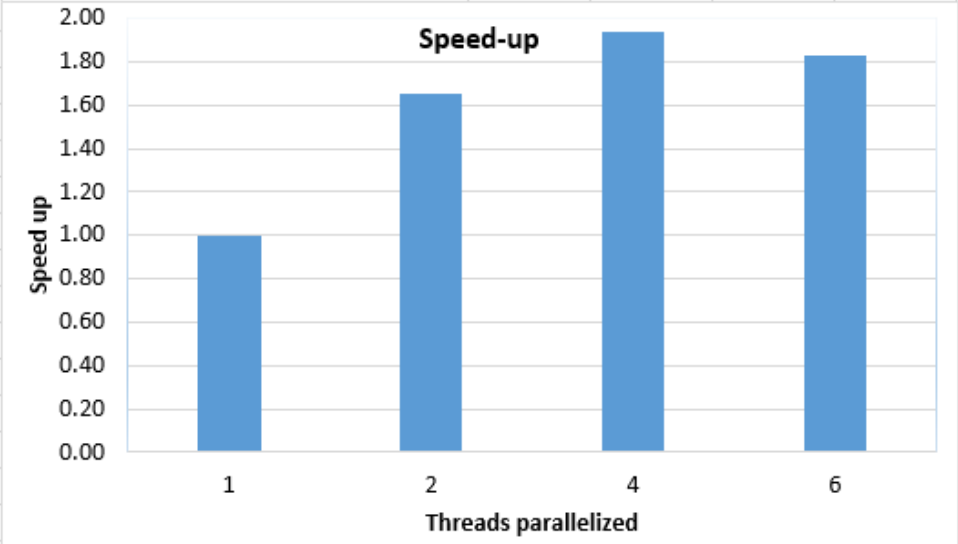


Figure 2. Theoretical evolution of speed-up depending on the parallel portion of a program.

	v7	v7par2	v7par4	v7par6
Number of threads	**1**	**2**	**4**	**6**
Total accumulated cputime	**478**	**579**	**986**	**1564**
Cputime solver only	390	398	626	969
Wall clock time (according to files)	479	290	247	261
Wall clock time calculated	**478**	**290**	**247**	**261**
speed up	**1.00**	**1.65**	**1.94**	**1.84**
speed up solver only	1.00	1.96	2.49	2.41
efficiency cputime	**1.00**	**0.83**	**0.48**	**0.31**
efficiency cputime solver only	1.00	0.98	0.62	0.40
Ratio solver/total:	0.81	0.69	0.63	0.62
Total number of N-R iterations (ntiter):	734	740	740	740
Total number of time intervals (int):	465	467	467	467
Ratio iterations per time step:	1.58	1.58	1.58	1.58



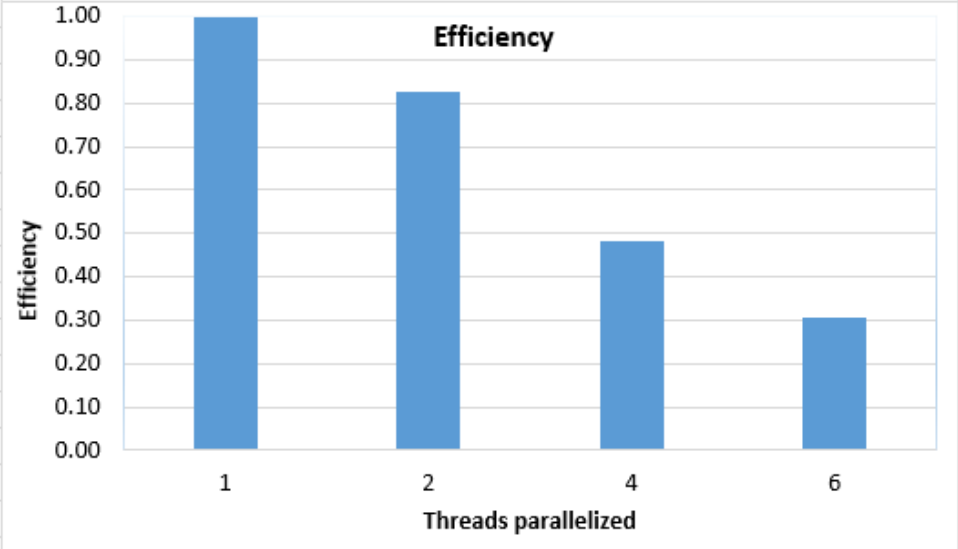


Figure 3. Example of speed-up and efficiency calculation.