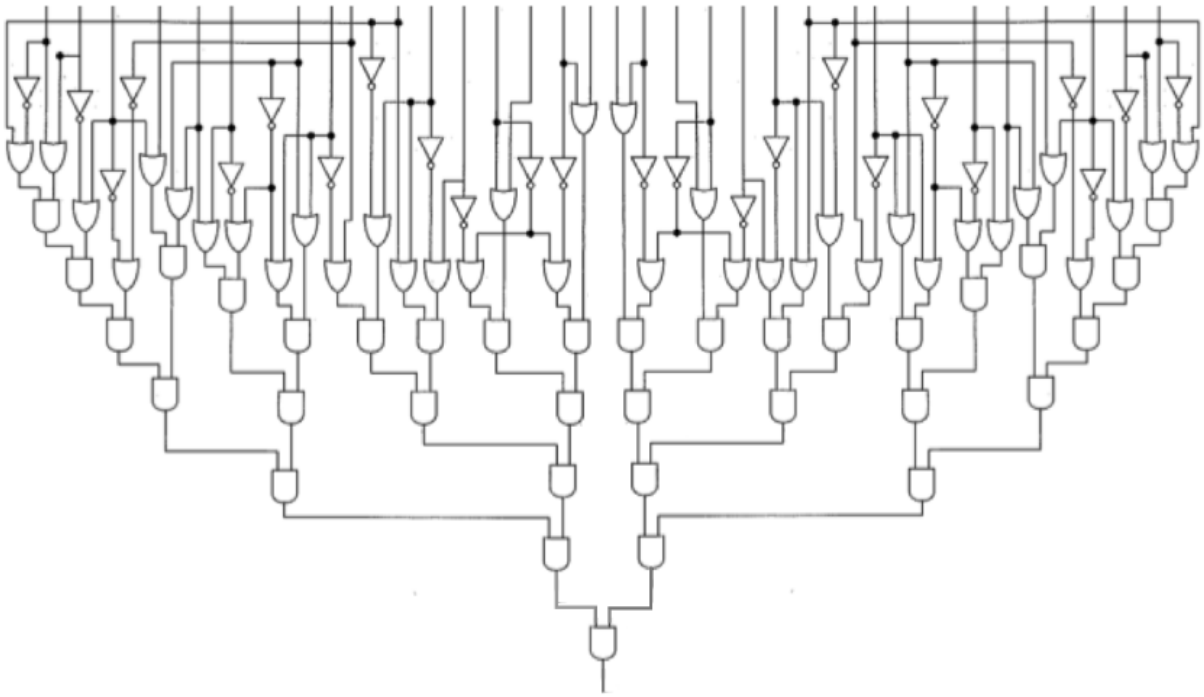
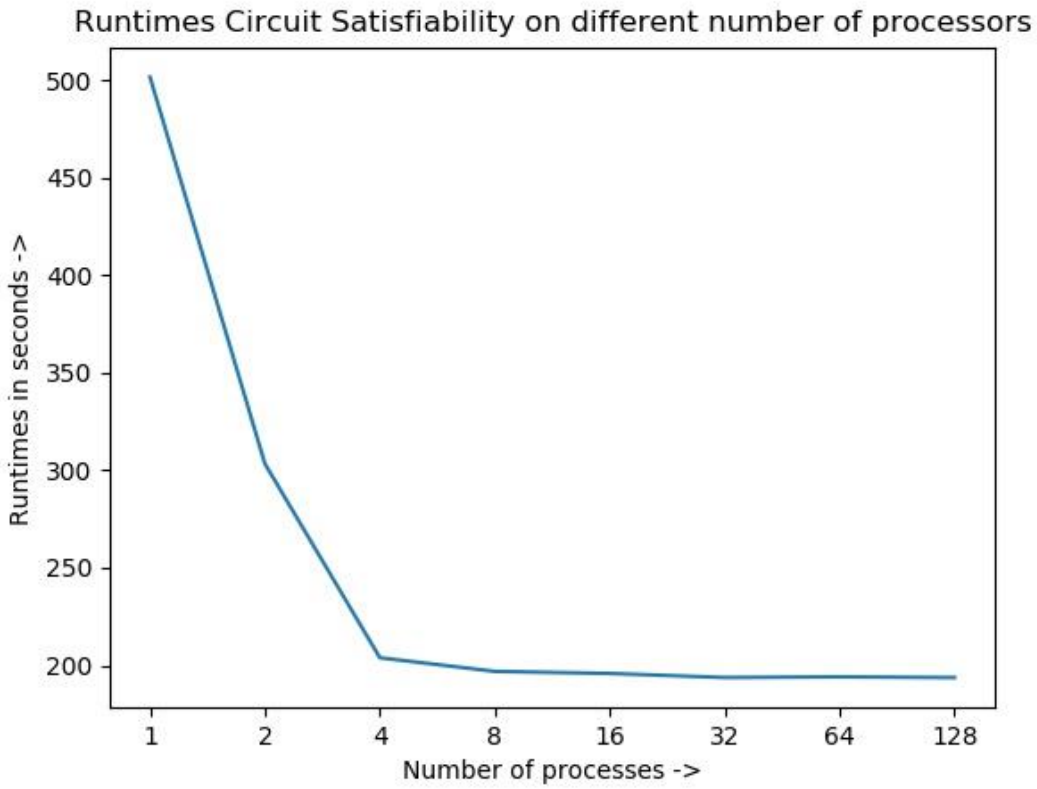
Circuit Satisfiability



Tabel meetwaarden:

Number of processes:	Runtime in seconds:	Number of solutions:
1	501.755280	81
2	303.369553	81
4	203.861919	81
8	196.863818	81
16	195.822393	81
32	193.720058	81
64	194.065160	81
128	193.691350	81
256	ERROR	ERROR

Grafiek meetwaarden:



Korte analyse:

Ik heb het verdelen van processes gedaan aan de hand van de ‘stripes’ methode.

Zoals in de bovenstaande grafiek en tabel te zien is, neemt de runtime af als het aantal gebruikte processoren/processen toeneemt. Dit is ook wat we verwachten, aangezien we het werk opsplitsen in parallelle taken. Echter is de runtime niet onbeperkt te verlagen. Vanaf het moment dat er meer processen worden ingezet dan de computer fysieke processoren aan boord heeft, wordt er geen echte tijdwinst meer behaald. Op mijn computer met Ubuntu subsysteem, kon ik 256 processen tegelijk niet eens draaien en kreeg ik een error.

Het viel mij wel op hoe verschillend de resultaten kunnen zijn bij opnieuw draaien of bij draaien op een andere computer. Daarom heb ik verschillende keren de code gedraaid per aantal processen om er zeker van te zijn dat er geen uitschieters zijn.

Bij verschillende personen uit de klas bleef de runtime omgekeerd relatief gelijk aan het aantal processen. Dus van 1 naar 2 processen werd de runtime gehalveerd. Bij mijn computer loopt de runtime niet relatief aan het aantal processen. Er is amper een verband te vinden, en ik merkte dat andere draaiende processen of geen oplader veel invloed heeft op de runtimes. Ik heb een redelijk oude laptop dus het is niet gek dat ik veel hogere tijden haal. Mogelijk heeft ook de architectuur van de computer invloed hierop.

Al met al kunnen we in ieder geval zeggen dat de runtime afneemt als het aantal parallelle processen toeneemt, tot het limiet van aantal processen is bereikt (in mijn geval i.i.g.). Blijkbaar gebruikt MPI werkelijke fysieke processoren om taken in op te splitsen en geen threads.