Nathan Dow - nmd210001
Leonid Mateush - lmm220016

Number of Threads and Time Efficiency

Introduction

This experiment is to demonstrate the effectiveness of multithreaded programs such as this one, which returns the hash value of an entire file.
Threads are only more productive if they can simultaneously divide work so they don't have to wait for each other's outputs.
In this experiment, we will show that the more threads created, the faster the output will be. We also predicted that eventually creating more threads would not help the speed of the program, as the processor only has a limited number of CPUs.
To conduct this experiment, we will be using multiple files of various sizes, running them through a multithreaded code hash, and using different amounts of threads. The time it takes for each operation will be graphed and analyzed to determine the most efficient number of threads given the information provided by lscpu.

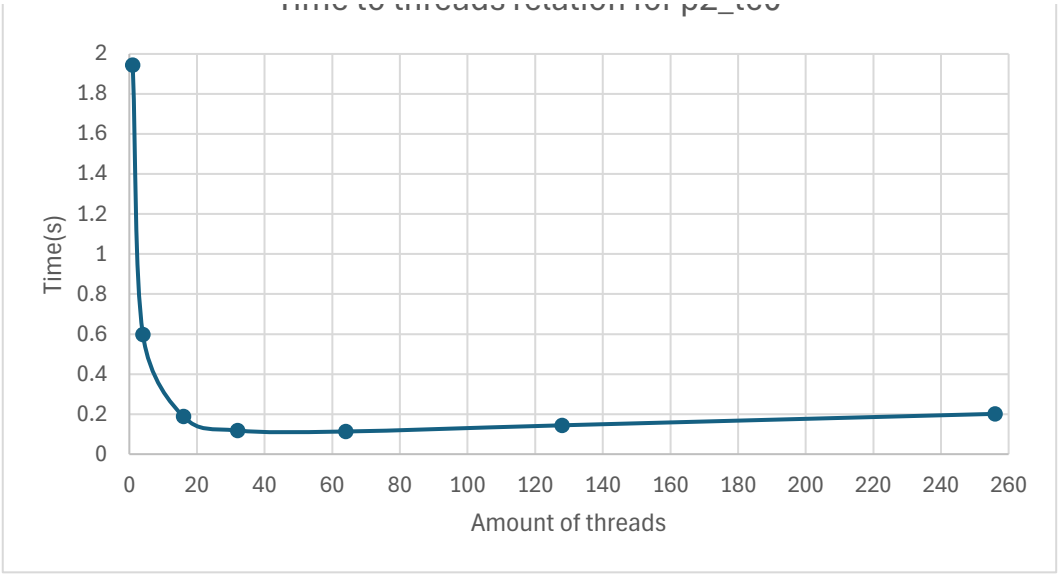
lscpu command can show us many things about our central process, amount of CPU cores, what Hz it can run at, how much cash it has and many more details about our CPU. Using this information, we can try to predict the amount of threads we need to run the program most efficiently. In our case, the CS3 server has 48 CPU cores, and looking at our graphs we can see that roughly at that point our speedup is at its highest.

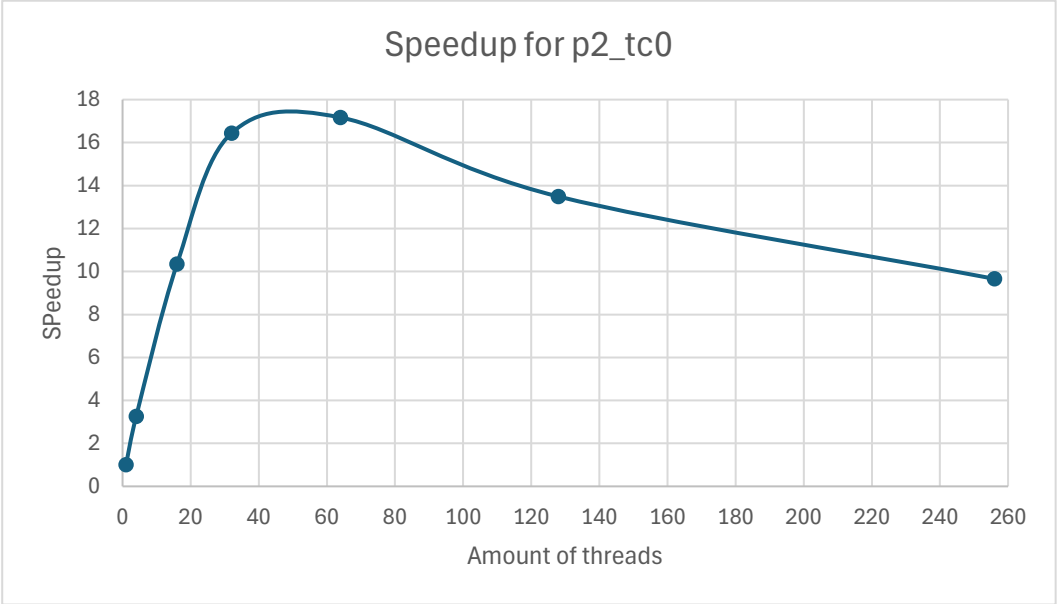
FILES	NUM THREADS:	1	4	16	32	64	128	256
p2_tc0	Time(s):	1.943778	0.598253	0.187882	0.118226	0.113187	0.144084	0.201302
p2_tc1	Time(s):	3.857671	1.129823	0.315818	0.199604	0.184185	0.203392	0.270569
p2_tc2	Time(s):	10.35951	2.23476	0.584305	0.371524	0.333014	0.343541	0.383376
p2_tc3	Time(s):	19.34357	4.09054	1.12976	0.706083	0.615194	0.592011	0.655463
	AVG Time(s):	8.876132	2.013344	0.554441	0.348859	0.311395	0.320757	0.377678
	AVG change in Time:	1	4.41	3.63	1.59	1.12	0.97	0.85

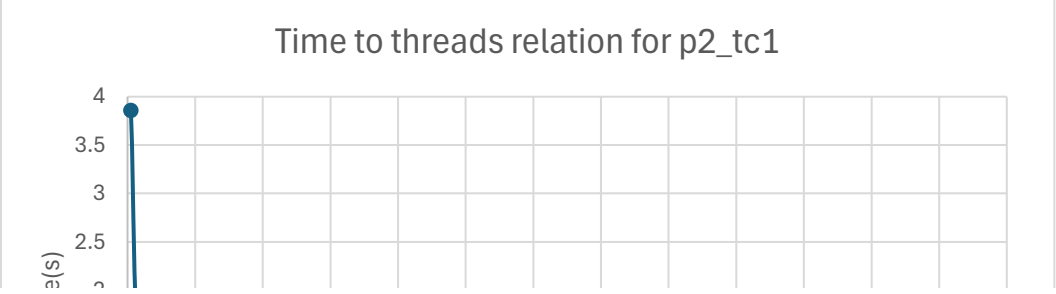
Speedup is (time 1 thread)/(time n threads)

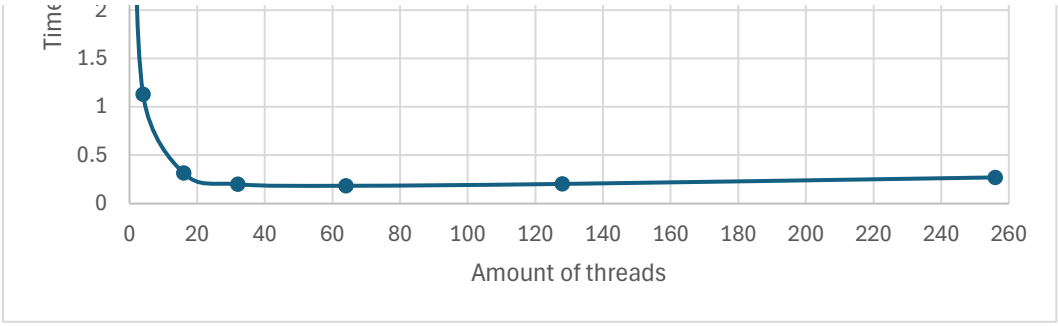
Speedup for p2_tc0:	1	3.25	10.35	16.44	17.17	13.49	9.66
Speedup for p2_tc1:	1	3.41	12.21	19.33	20.94	18.97	14.26
Speedup for p2_tc2:	1	4.64	17.73	27.88	31.11	30.16	27.02
Speedup for p2_tc3:	1	4.73	17.12	27.4	31.44	32.67	29.51

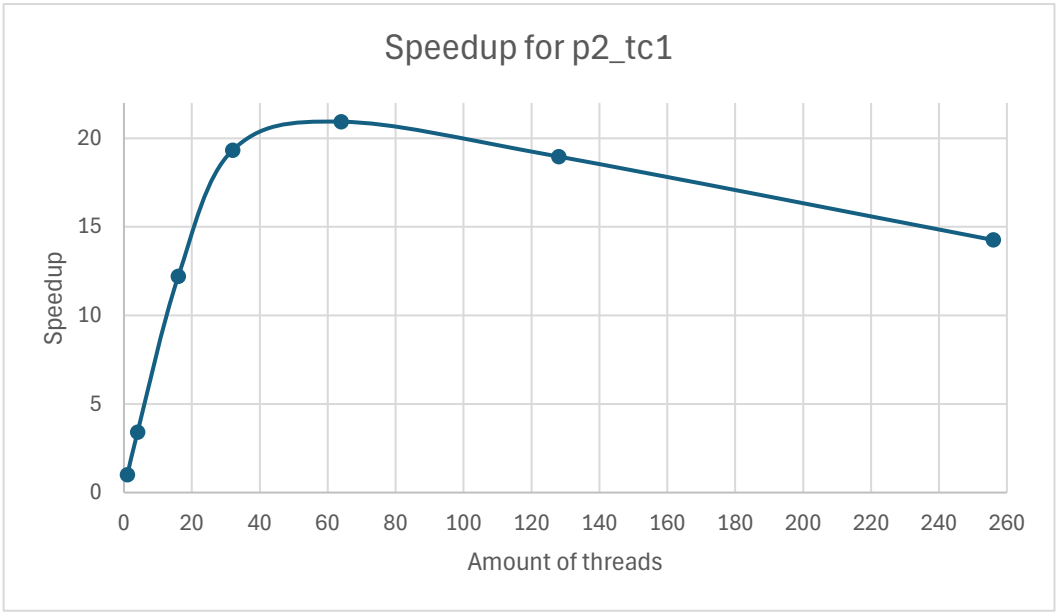
Time to threads relation for p2_tc0

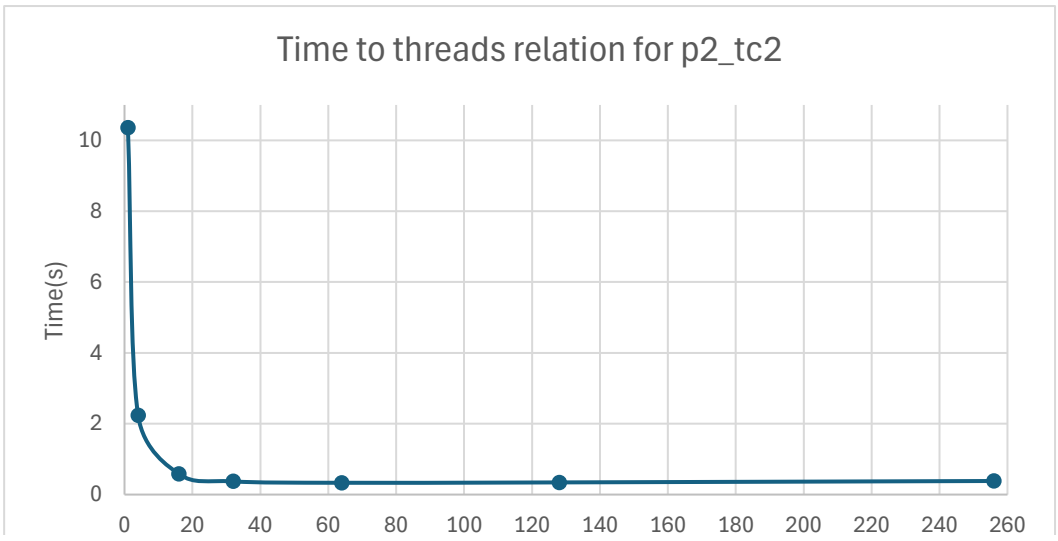

Time(s)
2
1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
0 20 40 60 80 100 120 140 160 180 200 220 240 260
Amount of threads

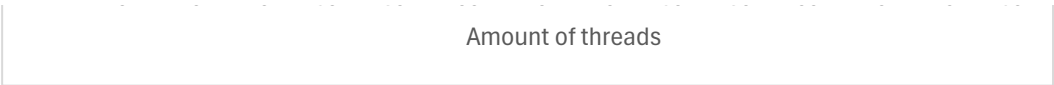

Speedup for p2_tc0
SPeedup
18
16
14
12
10
8
6
4
2
0
0 20 40 60 80 100 120 140 160 180 200 220 240 260
Amount of threads

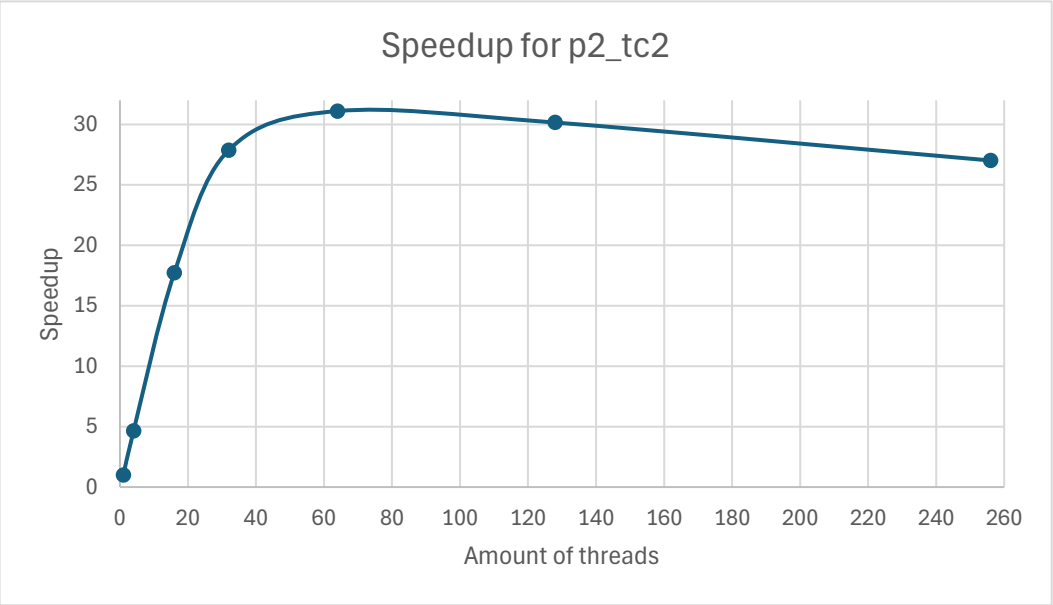

Time to threads relation for p2_tc1
4
3.5
3
2.5

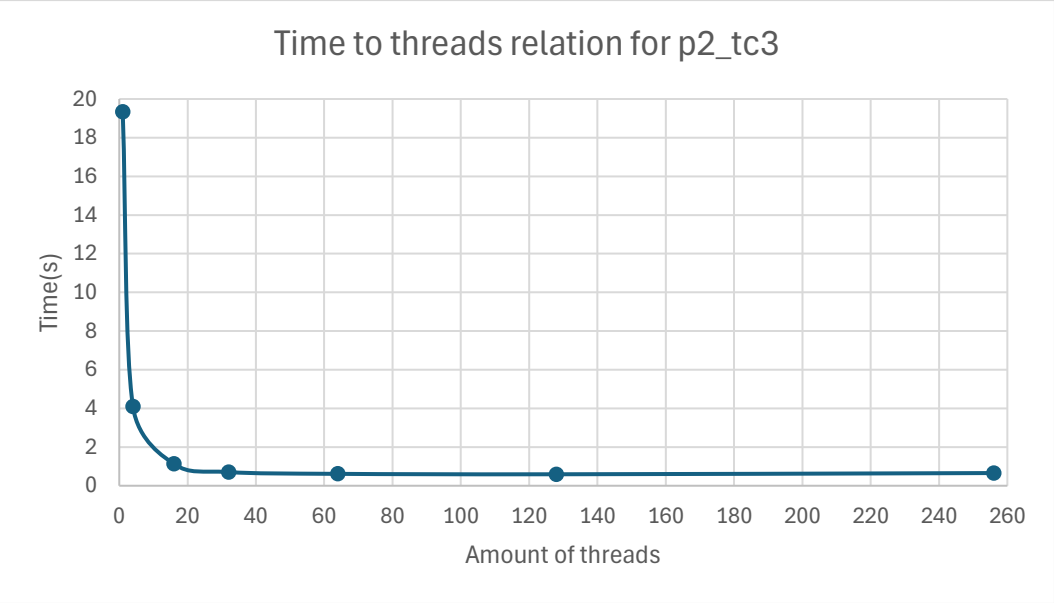

Time
2
1.5
1
0.5
0
0 20 40 60 80 100 120 140 160 180 200 220 240 260
Amount of threads

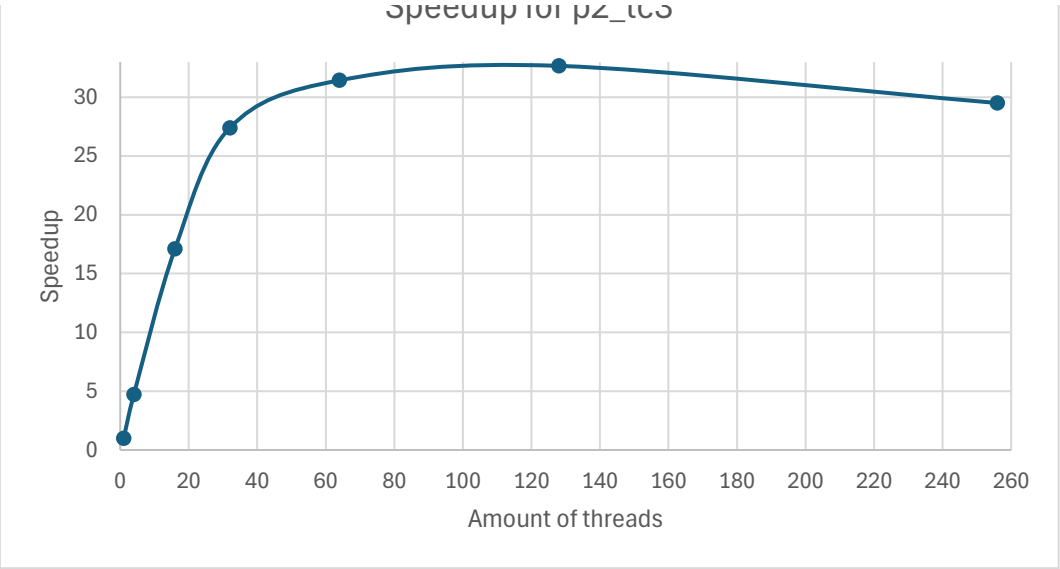

Speedup for p2_tc1
Speedup
20
15
10
5
0
0 20 40 60 80 100 120 140 160 180 200 220 240 260
Amount of threads


Time to threads relation for p2_tc2
Time(s)
10
8
6
4
2
0
0 20 40 60 80 100 120 140 160 180 200 220 240 260


Amount of threads


Speedup for p2_tc2
Speedup
0
5
10
15
20
25
30
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Amount of threads


Time to threads relation for p2_tc3
Time(s)
0
2
4
6
8
10
12
14
16
18
20
0
20
40
60
80
100
120
140
160
180
200
220
240
260
Amount of threads



Conclusion:

It is generally expected that the time to compute a hash value decreases with an increase in the numbe
particularly at low thread counts. However, this decrease in time plateaus as the number of threads ap
certain limit, and with an excessively large number of threads, the computation time may actually incre
files can benefit from more threads before the slowdown becomes noticeable, unlike smaller files whic
stagnation sooner.

Since each thread operates in parallel, increasing the number of threads generally reduces the workloa
thread, resulting in significant speed-ups. The speed-up is quite substantial when moving from a single
multi-threaded configuration.

The "AVG Change in Time" row indicates that, on average, the speedup is proportional up to about 16 th

er of threads,
proaches a
ease. Larger
ch experience

ad for each
e-threaded to a

hreads. Beyond