Lab Report #1

CASSARD Sebastien 19960526-T313

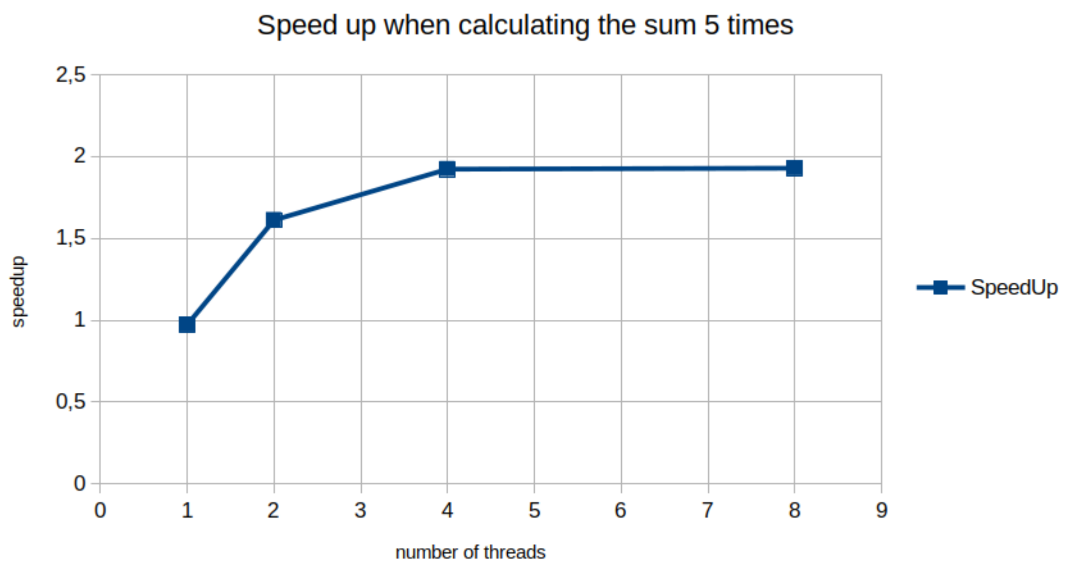
November 11, 2018

NOTE : *For the following results, the program is run for an array size of 500 000 000 elements.*

1 Task 1: Using Pthreads

1.1 Speedup for the Pthreads program

1.1.1 Result when calculating the sum 5 times



Number of threads	Serial	Pthreads	Speed up
1	11.628s	11.956s	0.97
2	11.628s	7.213s	1.61
4	11.628s	6.046s	1.92
8	11.628s	6.028s	1.93

The table above shows the different execution times of the program as well as the speedup for each number of threads. With the help of Amdahl's law it is possible to determine approximately which percentage of the program is serial and which percentage is parallel.

Amdahl's law tells us that:

$$Speedup = \frac{1}{(1-F)+\frac{F}{S}}$$

Where F is the fraction of the programme that is enhanced by a factor S. Here S correspond to the number of threads. So knowing S and $Speedup$ we can deduce $1-F$ and F which are respectively the percentage of the program which run in serial and the percentage which runs in parallel.

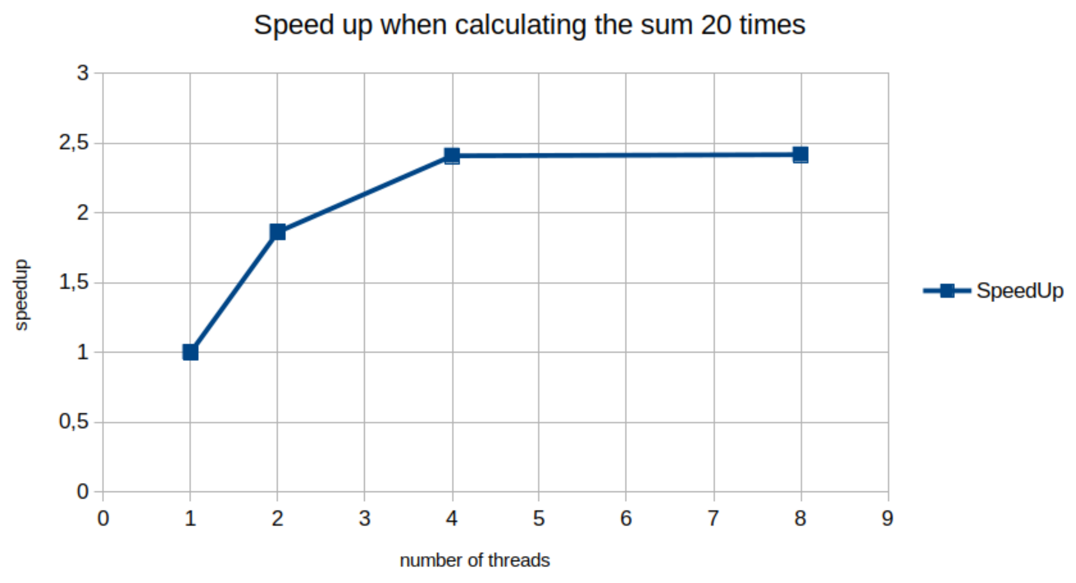
From the above formula, the following equality can be deduced:

$$F = \frac{S(\frac{1}{Speedup}-1)}{1-S}$$

Which gives us the following results :

Number of threads	F	1-F
2	0.76	0.24
4	0.64	0.36
8	0.55	0.45

1.1.2 Result when calculating the sum 20 times



Number of threads	Serial	Pthreads	Speed up
1	38.247s	38.245s	1.00
2	38.247s	20.525s	1.86
4	38.247s	15.874s	2.41
8	38.247s	15.819s	2.42

As in the previous section, the percentages of the serial $(1 - F)$ and parallel (F) fractions are deducted:

Number of threads	**F**	**1-F**
2	0.92	0.08
4	0.78	0.12
8	0.67	0.23

1.2 Conclusions

In the two cases studied above, we notice an improvement in execution time with the increase in the number of threads. However, several elements need to be put into perspective.

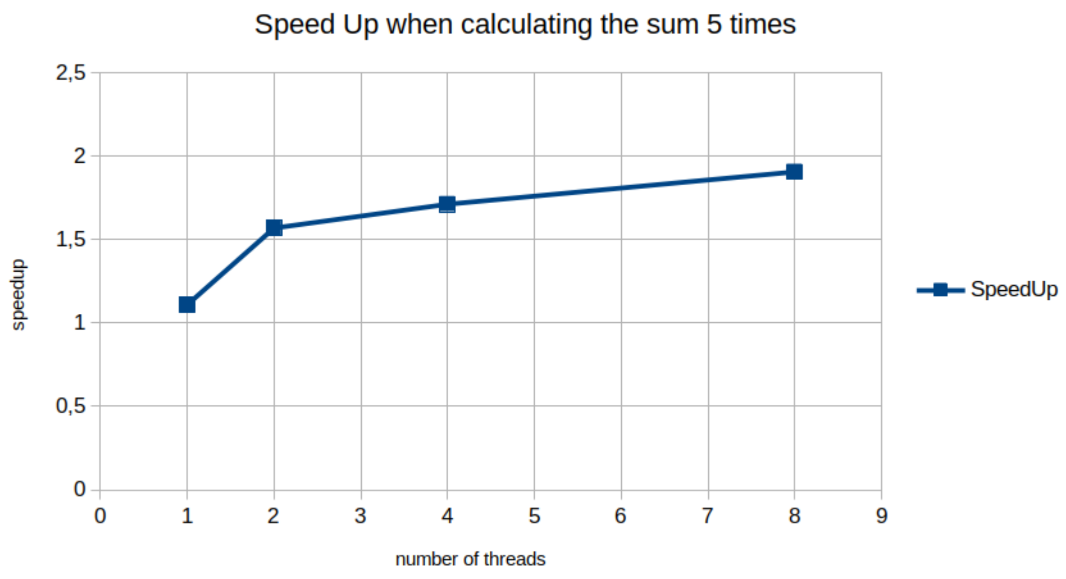
Firstly, we notice that the speedup stagnates for more than 4 threads. This can be explained by the fact that the measurements were made on an Intel Core i3-6100U cpu which has only 4 cores.

Secondly, we can see that the higher the number of threads is, the higher the serial code fraction is. This is probably due to the overhead caused by the creation of the different threads.

Despite these elements, we can observe much better results with a higher number of threads.

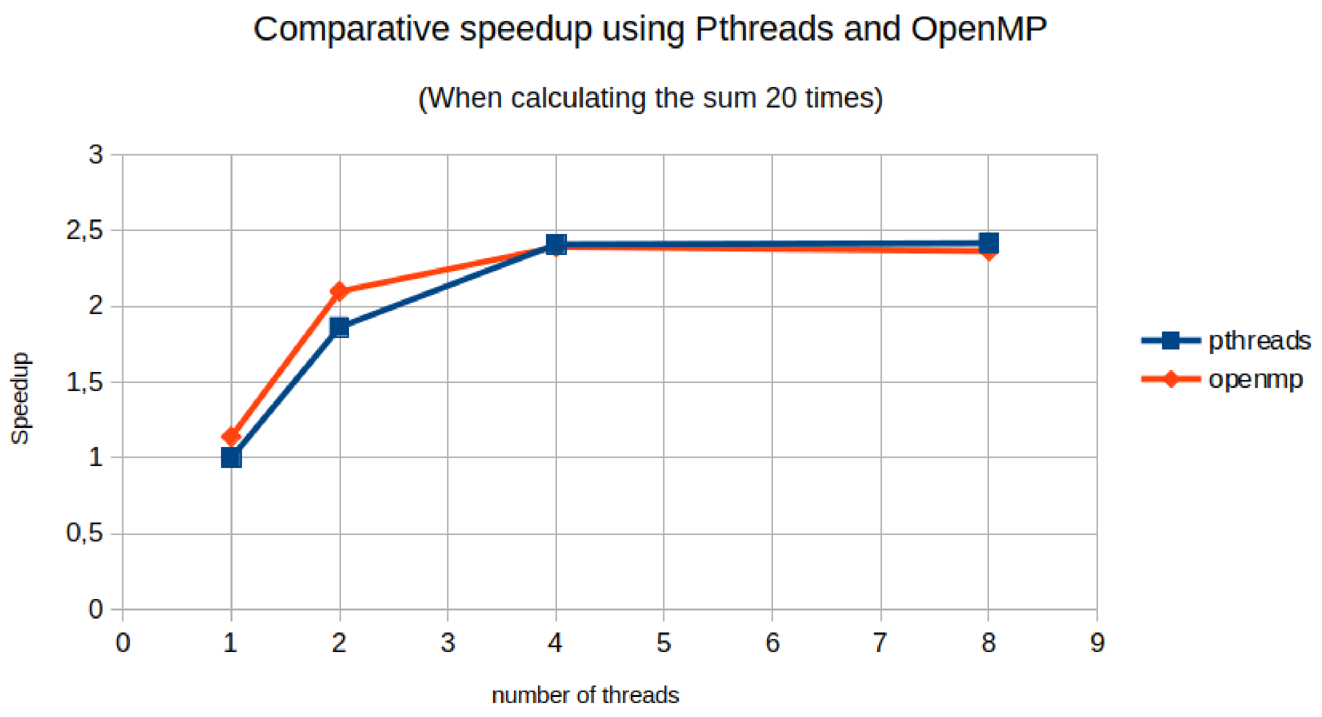
2 Task 2: Using OpenMP

2.1 Speedup for the OpenMP program



Number of threads	Serial	OpenMP	Speed up
1	11.628s	10.474s	1.11
2	11.628s	7.405s	1.57
4	11.628s	6.787s	1.71
8	11.628s	6.096s	1.91

2.2 Comparing OpenMP and Pthreads



Number of threads	Pthreads Speed up	OpenMP Speed up
1	1.00	1.14
2	1.86	2.10
4	2.41	2.39
8	2.42	2.36

We can see that the results obtained with OpenMP and Pthreads are very similar. However, OpenMP only requires one line of code where Pthreads requires about ten. In the case of a loop to be parallelized, the use of OpenMP will be preferred. However, if we need to start a separate process which shouldn't block the main thread, then maybe Pthreads would be a better choice as it allows us to have you have extremely fine-grained control over thread management.