INDIAN INSTITUTE OF INFORMATION TECHNOLOGY DESIGN AND MANUFACTURING KANCHEEPURAM

LAB ASSIGNMENT 3 - REPORT
ON
ADDITION OF N NUMBERS AND VECTOR DOT PRODUCT USING REDUCTION AND CRITICAL SECTION FOR BOTH

SUBMITTED BY

AMAR KUMAR

(CED17I029)

TO

DR. NOOR MAHAMMAD

Specifications : -
Cpu processor - Intel core i5-7200 CPU @ 2.50GHz

L1d cache:	64 KiB
L1i cache:	64 KiB
L2 cache:	512 KiB
L3 cache:	3 MiB

Thread(s) per core:	2
Core(s) per socket:	2
Socket(s):	1

So number of physical cores = 2*1 = 2
And number of logical cores = 2*2*1 = 4

Observation : -

For all the questions, i have taken number of iterations as 10^7 and **question1a** and **question1b** was taking around 1-2 minutes while **question2a** and **question2b** was taking around 3-4 minutes.
These time includes the generation of random numbers as well

ADDITION OF N NUMBERS

(USING REDUCTION)

Strategy

In my program for addition of n numbers, the instruction which is running in parallel is **asum = asum + a[i]**

Here i am using reduction method to add these n numbers otherwise it should lead to racing conditions and thus should give wrong results. Due to reduction, the value of asum is not shared by all the threads and thus it gives the right result.

For example - suppose thread1 has to execute for iteration 1 to 10, then asum is only shared with iteration 1 to 10 not others. Also it is not shared with other threads to avoid racing conditions.

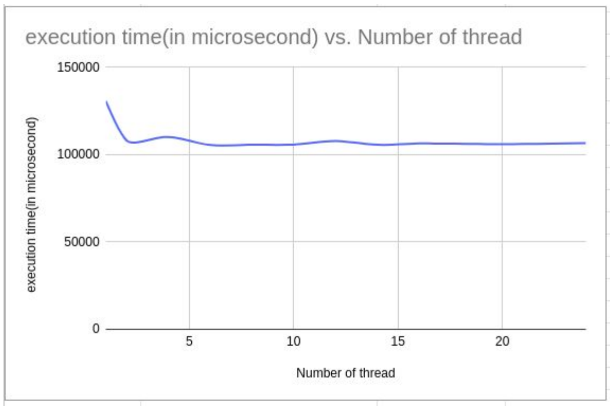
Instead of running the program serially, we can distribute the task of adding n elements among the threads so that my program could run parallely and in turn save the execution time.

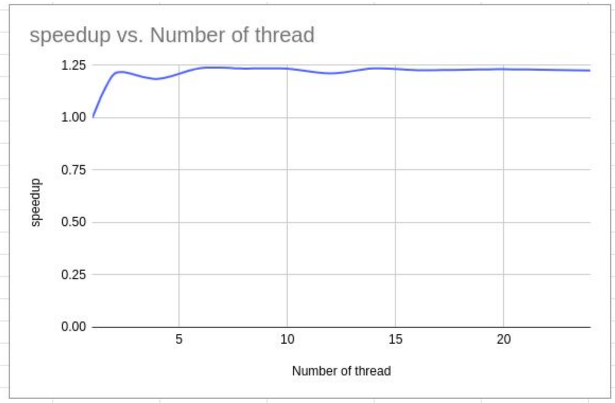
Therefore , in #pragma omp parallel block , for loop consists of these granular instructions. We also have shared a,b and c to run the program in parallel.

Graph and tables

https://docs.google.com/spreadsheets/d/1XfX1qzk0ZGUGZaxF76MaZQ9xRs8rBuHHTQx2Kb8HnT8/edit#gid=0

Number of iterations = 10^7			
Number of thread	execution time(in microsecond)	speedup	parallelization fraction(f)
1	130564	1	0
2	107992	1.209015483	0.3457614656
4	110177	1.18503862	0.2081941934
6	105536	1.237151304	0.2300297172
8	105724	1.234951383	0.2174303133
10	105783	1.234262594	0.2108884872
12	107782	1.211371101	0.1903517885
14	105646	1.235863166	0.2055296194
16	106415	1.226932293	0.1972897072
20	106029	1.231398957	0.1978057948
24	106576	1.225078817	0.1917140753





Calculation of parallelization fraction

T(1) = 130564 microseconds
Here , for P = 6 the execution time is minimum
T(P) = 105536 microseconds

Speedup = $\frac{T(1)}{T(P)}$ = $\frac{130564}{105536}$ = 1.237151304.

From Amdahl's Law,

Speedup = $\frac{1}{(f/P)+(1-f)}$ Where , f = Parallelization factor P = Thread Number

So, f = $\frac{(1-T(P)/T(1))}{(1-(1/P))}$

Therefore, f = 0.2300297172 which means that approx. 23% of the program is parallelizable.

ADDITION OF N NUMBERS

(USING CRITICAL SECTION)

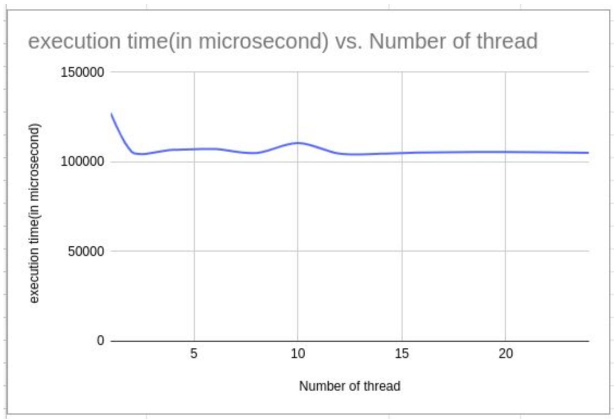
Strategy

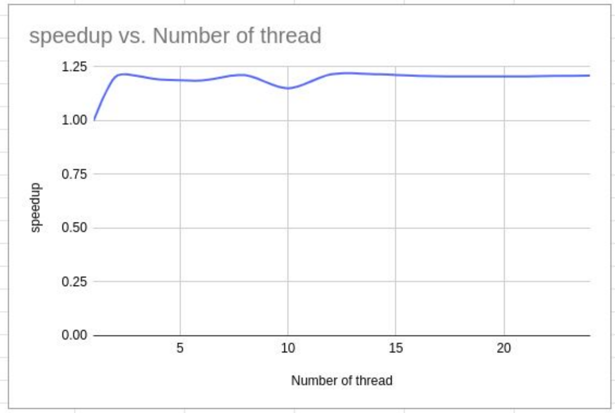
In my program for addition of n numbers, the instruction which is running in parallel is
psum = psum + a[i]
and asum = asum + psum is in critical section.
Here i am using critical section method to add these n numbers otherwise it should lead to racing conditions and thus should give wrong results. Due to critical section, the value of psum is not shared by all the threads and it is used in critical section for finding total psum which is equal to asum and thus it gives the right result.
For example - suppose thread1 has to execute for iteration 1 to 10, then psum is only shared with iteration 1 to 10 not others. Also it is not shared with other threads to avoid racing conditions.
Instead of running the program serially, we can distribute the task of adding n elements among the threads so that my program could run parallelly and in turn save the execution time.
Therefore , in #pragma omp parallel block , for loop consists of these granular instructions. We also have shared a,b and c to run the program in parallel.

Graph and tables

https://docs.google.com/spreadsheets/d/1XfX1qzk0ZGUGZaxF76MaZQ9xRs8rBuHHTQx2Kb8HnT8/edit#gid=0

Number of iterations = 10^7			
Number of thread	execution time(in microsecond)	speedup	parallelization fraction(f)
1	127199	1	0
2	105749	1.202838798	0.3372668024
4	106726	1.191827671	0.2146033643
6	107203	1.186524631	0.1886429925
8	105000	1.211419048	0.1994534997
10	110578	1.150310188	0.1451880736
12	104648	1.215493846	0.1934063232
14	104616	1.215865642	0.1911976812
16	105279	1.208208665	0.1838169587
20	105552	1.20508375	0.1791391111
24	105141	1.209794466	0.1809530223





Calculation of parallelization fraction

T(1) = 127199 microseconds
Here , for P = 14 the execution time is minimum
T(P) = 104616 microseconds
Speedup = $\frac{T(1)}{T(P)}$ = $\frac{127199}{104616}$ = 1.215865642.
From Amdahl's Law,
Speedup = $\frac{1}{(f/P) + (1-f)}$ Where , f = Parallelization factor P = Thread Number

So, f = $\frac{(1-T(P)/T(1))}{(1-(1/P))}$
Therefore, f = 0.1911976812 which means that approx. 19% of the program is parallelizable.

VECTOR DOT PRODUCT

(USING REDUCTION)

Strategy

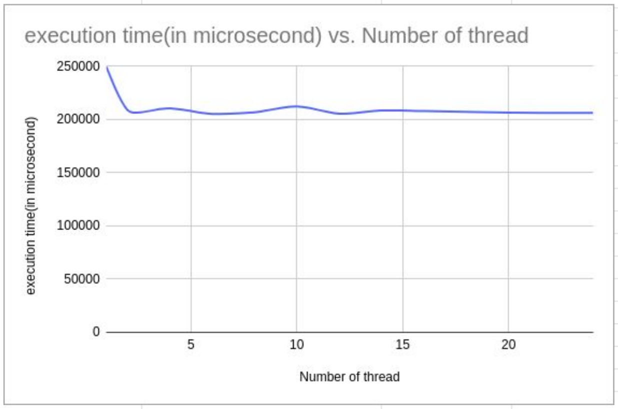
In my program for addition of n numbers, the instruction which is running in parallel is
asum = asum + a[i]*b[i]
Here i am using reduction method to add these product of a and b otherwise it should lead to racing conditions and thus should give wrong results. Due to reduction, the value of asum is not shared by all the threads and thus it gives the right result.
For example - suppose thread1 has to execute for iteration 1 to 10, then asum is only shared with iteration 1 to 10 not others. Also it is not shared with other threads to avoid racing conditions.
Instead of running the program serially, we can distribute the task of adding n multiplied elements among the threads so that my program could run parallely and in turn save the execution time.
Therefore , in #pragma omp parallel block , for loop consists of these granular instructions. We also have shared a,b and c to run the program in parallel.

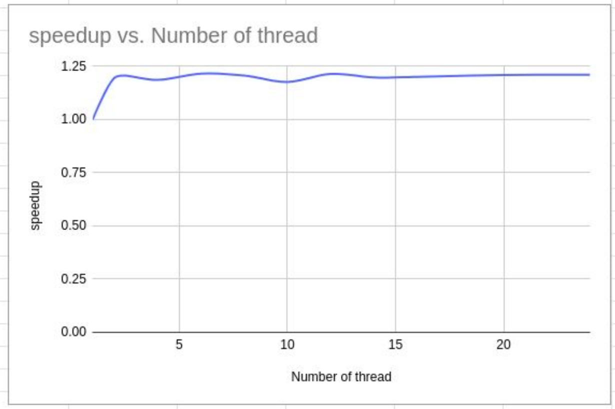
Graph and tables

https://docs.google.com/spreadsheets/d/1XfX1qzk0ZGUGZaxF76MaZQ9xRs8rBuHHTQx2Kb8HnT8/edit#gid=0

Number of iterations = 10^7

Number of thread	execution time(in microsecond)	speedup	parallelization fraction(f)
1	249625	1	0
2	208884	1.195041267	0.3264176264
4	210505	1.185838816	0.2089534301
6	205308	1.215856177	0.2130411617
8	206856	1.206757358	0.1958091423
10	212222	1.176244687	0.1664852835
12	205558	1.214377451	0.1925812355
14	208442	1.197575345	0.1776701976
16	207916	1.200605052	0.1782257386
20	206434	1.209224256	0.1821300372
24	206140	1.21094887	0.1817752716





Calculation of parallelization fraction

T(1) = 249625 microseconds
Here , for P = 6 the execution time is minimum
T(P) = 205308 microseconds
Speedup = $\frac{T(1)}{T(P)}$ = $\frac{249625}{205308}$ = 1.215856177.
From Amdahl's Law,
Speedup = $\frac{1}{(f/P)+(1-f)}$ Where , f = Parallelization factor P = Thread Number

So, f = $\frac{(1-T(P)/T(1))}{(1-(1/P))}$
Therefore, f = 0.2130411617 which means that approx. 21% of the program is parallelizable.

VECTOR DOT PRODUCT

(USING CRITICAL SECTION)

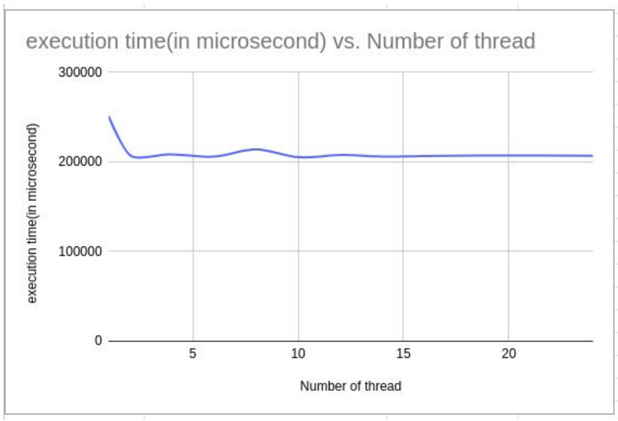
Strategy

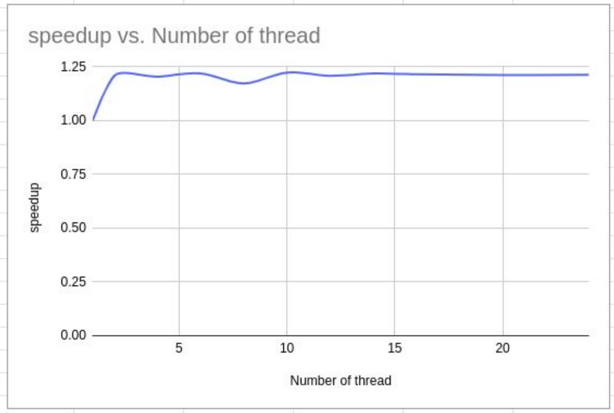
In my program for addition of n numbers, the instruction which is running in parallel is
psum = psum + a[i] * b[i]
and asum = asum + psum is in critical section.
Here i am using critical section method to add these product of a and b otherwise otherwise it should lead to racing conditions and thus should give wrong results. Due to critical section, the value of psum is not shared by all the threads and it is used in critical section for finding total psum which is equal to asum and thus it gives the right result.
For example - suppose thread1 has to execute for iteration 1 to 10, then psum is only shared with iteration 1 to 10 not others. Also it is not shared with other threads to avoid racing conditions.
Instead of running the program serially, we can distribute the task of adding n multiplied elements among the threads so that my program could run parallely and in turn save the execution time.
Therefore , in #pragma omp parallel block , for loop consists of these granular instructions. We also have shared a,b and c to run the program in parallel.

Graph and tables

https://docs.google.com/spreadsheets/d/1XfX1qzk0ZGUGZaxF76MaZQ9xRs8rBuHHTQx2Kb8HnT8/edit#gid=0

Number of iterations = 10^7			
Number of thread	execution time(in microsecond)	speedup	parallelization fraction(f)
1	251055	1	0
2	207855	1.207837194	0.3441476967
4	208401	1.204672722	0.2265320348
6	205888	1.219376554	0.2158905419
8	214121	1.172491255	0.1681316274
10	205234	1.223262228	0.202793102
12	207781	1.208267358	0.1880384776
14	205986	1.218796423	0.1933275424
16	206580	1.215291897	0.1889625779
20	207203	1.211637862	0.1838640935
24	206882	1.213517851	0.1835994711





Calculation of parallelization fraction

T(1) = 251055 microseconds

Here , for P = 10 the execution time is minimum

T(P) = 205234 microseconds

Speedup = $\frac{T(1)}{T(P)}$ = $\frac{251055}{205234}$ = 1.223262228.

From Amdahl's Law,

Speedup = $\frac{1}{(f/P)+(1-f)}$ Where , f = Parallelization factor P = Thread Number

So, f = $\frac{(1-T(P)/T(1))}{(1-(1/P))}$

Therefore, f = 0.202793102 which means that approx. 20% of the program is parallelizable.