Introduction to Parallel Scientific Computing
(CSE504)
Assignment – 2
Roll No : 2019201047

1. Profiling of a Sequential code:

i. Inclusive computing time : the inclusive time (or self time) of a function is the computng time taken by the function along with the time taken by the calls made to other functions(children) by it.
Exclusive computing time : the exclusive time of a function is the computing time taken by the function alone excluding all other children calls.

In our code, the function which takes the most computing time(exclusive) is **funct7**.

a) Speed up :
When only funct7 is parallelized(in best case, assuming the function is parallelized using 260 threads, as it is called 260 times). Thus, the total running time of the function will reduce to : Execution time of function/total calls = 26.69/260 = 0.1 sec. So, the cumulative seconds and Self seconds will change as follows :

Cumuative Seconds	Self Seconds	Name
0.1	0.1	funct7()
8.20	8.10	funct1()
13.96	5.76	funct3()
16.6	2.64	funct8()
18.6	2.00	funct2()
20.6	1.31	funct5()
21.01	0.41	funct4()
21.15	0.14	funct6()
21.25	0.10	main

Hence total running time reducec to 21.25 sec(previously, 46.75 sec).
Thus Speed up = Total running time of sequential code/ Total running time of the parallelized code = 46.75/21.25 = **2.2**

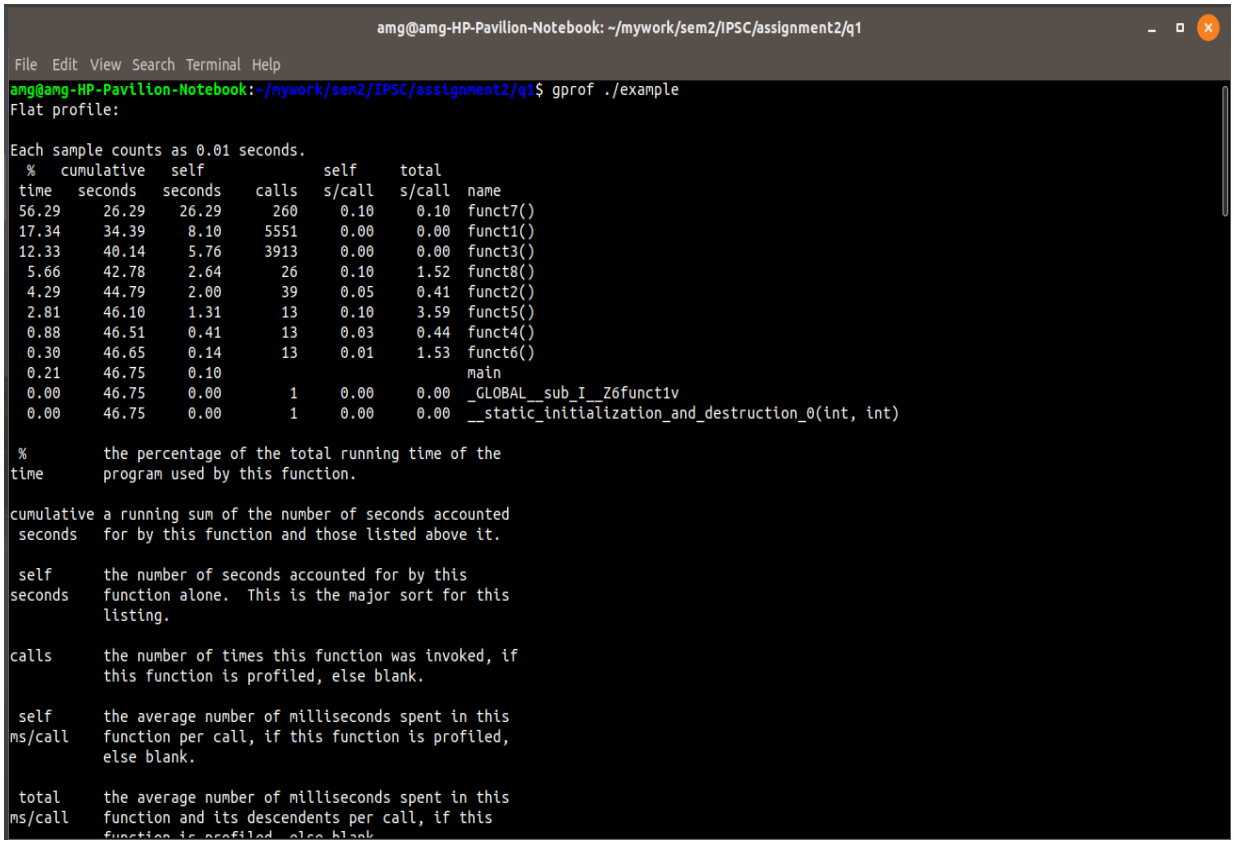
b) The funct7 is called 260 times.
c) The function funct7 does not call any other function, which is evident from the call graph of the code.

ii. It is evident from the flat profile that **funct1**, is the most called function. It is called by funct2() and funct3().

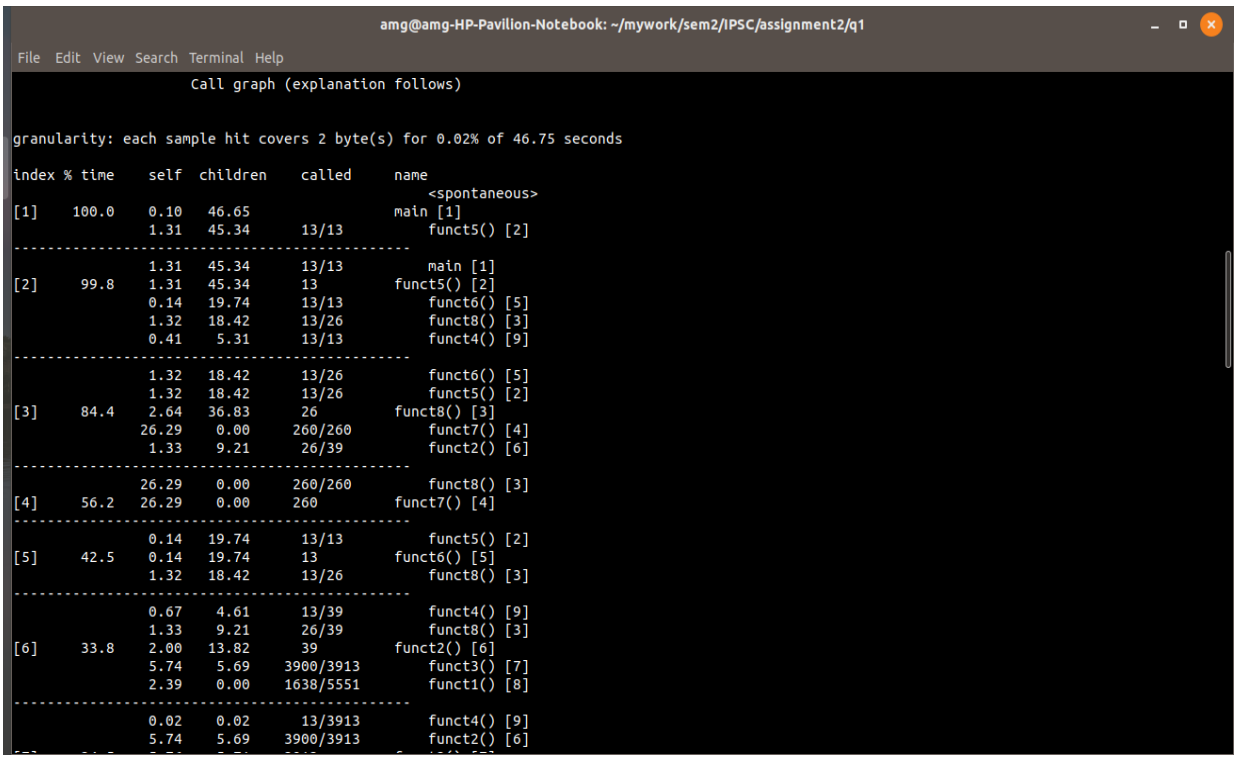
iii. The computation time of funct5 is : 46.65 sec(ie. Self time + Children time)

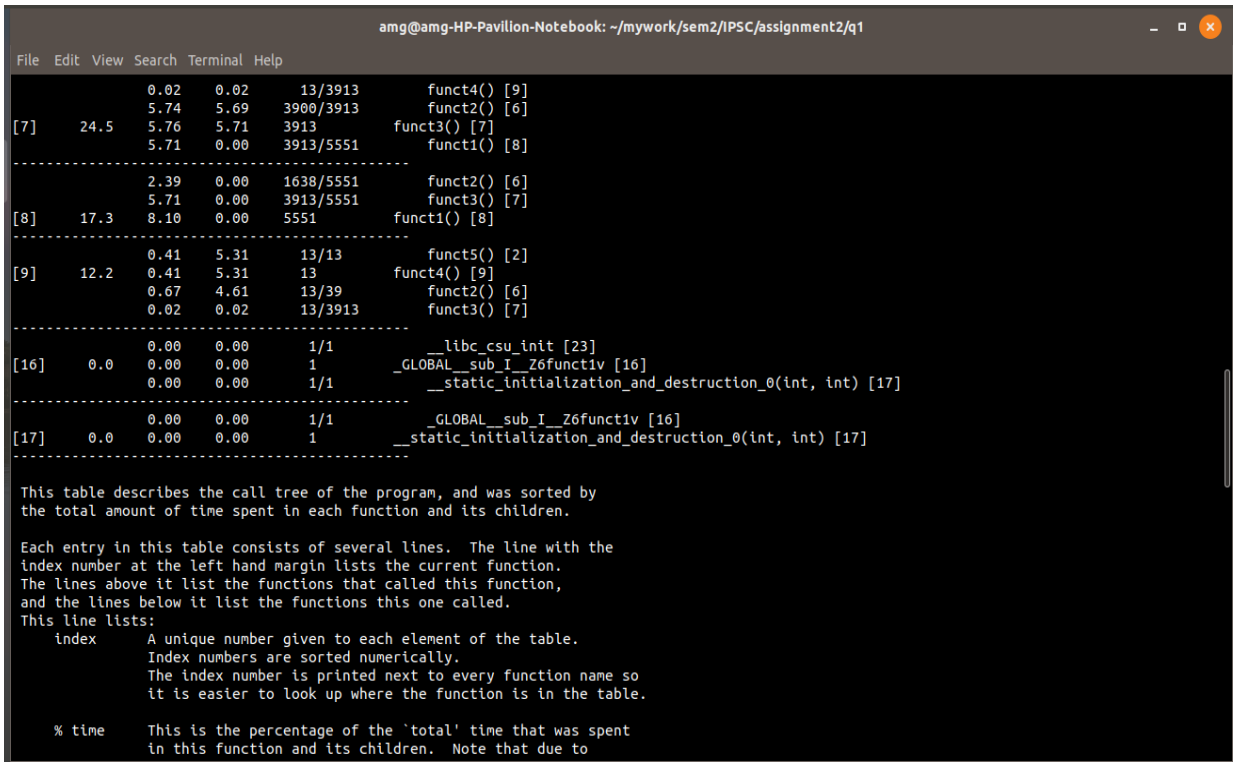
iv. The screenshot of the gprof command output is as follows :

FLAT PROFILE :



CALL GRAPH :





2. Cache Effect :

i. Average running time(real) of co1a.c is : (0.143+0.171+0.143+0.152+0.142)/5 = **0.1502 sec**
Average running time(real) of co1b.c is : (0.098+0.116+0.088+0.114+0.113)/5 = **0.1058 sec**
The second code is faster because there are no cache misses in it, due to the ‘c’ and ‘d’ variables of the struct are not present in it.

3. Cache Effect :

i. Average running time(real) of co2a.c is : (1.453+1.587+1.523+1.557+1.572)/5 = **1.5384 sec**
Average running time(real) of co2b.c is : (0.222+0.254+0.222+0.223+0.244)/5 = **0.233 sec**
The first code is slower because it accesses, the memory in jumps of 1024*8 locations, which causes cache misses and thus slow down the code. The second code on the other hand runs sequentially, as it accesses the memory locations sequentially. Thus no cache misses and hence, it runs considerably faster.

4. Cache Optimization (Cache Blocking) :

i. Average running time(real) of version1(n=1000) is :
(8.389+8.491+8.610+8.459+8.534)/5 = **8.4966 sec**
Average running time(real) of version2(n=1000) is :
(3.825+3.842+3.816+3.828+3.904)/5 = **3.8435 sec**
Average running time(real) of version3(n=1000) is :
(3.842+3.809+3.833+3.872+3.832)/5 = **3.8376 sec**

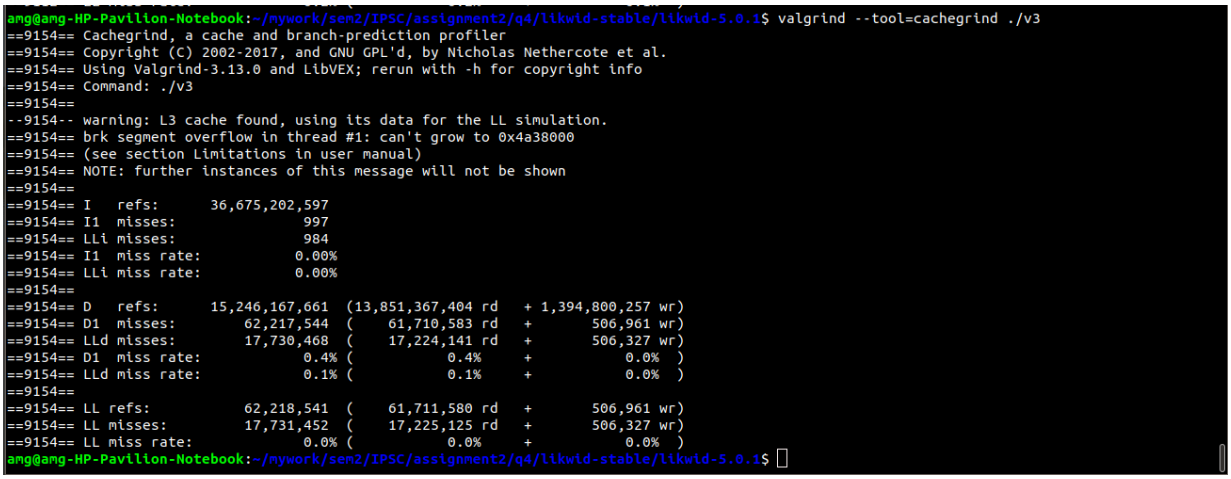
In the first code, the matrices A and C are accessed in row major order whereas the matrix B is accessed in column major order, which causes cache misses, and significantly slows down the code, because it happens inside 3 for-loops.($O(n^3)$)
The second code runs faster because we have transposed matrix B and now all three matrices(A,BT,C) are accessed in row major order fashion inside the 3 for-loops. ($O(n^3)$), making the code faster. There are cache misses in second code also, when we are transposing matrix B, but since it is performed only once, that too outside the 3 for-loops, so it does not affect the running time much.

The third code runs faster than the first two codes as it runs for cache block size used in our cache memory, thus increases the number of hits, hence making it faster.

Cache Report for version 1 ,2 and 3:

```
amg@amg-HP-Pavilion-Notebook:~/mywork/sem2/IPSC/assignment2/q4/likwid-stable/likwid-5.0.1$ valgrind --tool=cachegrind ./v1
==8472== Cachegrind, a cache and branch-prediction profiler
==8472== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==8472== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==8472== Command: ./v1
==8472==
--8472-- warning: L3 cache found, using its data for the LL simulation.
==8472== brk segment overflow in thread #1: can't grow to 0x4a38000
==8472== (see section Limitations in user manual)
==8472== NOTE: further instances of this message will not be shown
==8472==
==8472== I   refs:      37,037,846,019
==8472== I1  misses:               988
==8472== LLi misses:               975
==8472== I1  miss rate:           0.00%
==8472== LLi miss rate:           0.00%
==8472==
==8472== D   refs:      18,021,270,483  (17,017,159,412 rd   + 1,004,111,071 wr)
==8472== D1  misses:     1,252,251,784  ( 1,251,871,959 rd   +       379,825 wr)
==8472== LLd misses:       126,382,181  (   126,002,870 rd   +       379,311 wr)
==8472== D1  miss rate:            6.9% (           7.4%     +           0.0%  )
==8472== LLd miss rate:            0.7% (           0.7%     +           0.0%  )
==8472==
==8472== LL refs:        1,252,252,772  ( 1,251,872,947 rd   +       379,825 wr)
==8472== LL misses:        126,383,156  (   126,003,845 rd   +       379,311 wr)
==8472== LL miss rate:             0.2% (           0.2%     +           0.0%  )
```

```
amg@amg-HP-Pavilion-Notebook: ~/mywork/sem2/IPSC/assignment2/q4/likwid-stable/likwid-5.0.1
File  Edit  View  Search  Terminal  Help
amg@amg-HP-Pavilion-Notebook:~/mywork/sem2/IPSC/assignment2/q4/likwid-stable/likwid-5.0.1$ gprof ./v2 > example.txt
gmon.out: No such file or directory
amg@amg-HP-Pavilion-Notebook:~/mywork/sem2/IPSC/assignment2/q4/likwid-stable/likwid-5.0.1$ gprof ./v2
gmon.out: No such file or directory
amg@amg-HP-Pavilion-Notebook:~/mywork/sem2/IPSC/assignment2/q4/likwid-stable/likwid-5.0.1$ valgrind --tool=cachegrind ./v2
==9112== Cachegrind, a cache and branch-prediction profiler
==9112== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==9112== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==9112== Command: ./v2
==9112==
--9112-- warning: L3 cache found, using its data for the LL simulation.
==9112== brk segment overflow in thread #1: can't grow to 0x4a38000
==9112== (see section Limitations in user manual)
==9112== NOTE: further instances of this message will not be shown
==9112==
==9112== I   refs:      37,057,063,970
==9112== I1  misses:               992
==9112== LLi misses:               979
==9112== I1  miss rate:           0.00%
==9112== LLi miss rate:           0.00%
==9112==
==9112== D   refs:      18,031,341,535  (17,026,197,029 rd   + 1,005,144,506 wr)
==9112== D1  misses:       127,695,045  (   127,188,082 rd   +       506,963 wr)
==9112== LLd misses:       126,752,981  (   126,246,659 rd   +       506,322 wr)
==9112== D1  miss rate:            0.7% (           0.7%     +           0.1%  )
==9112== LLd miss rate:            0.7% (           0.7%     +           0.1%  )
==9112==
==9112== LL refs:          127,696,037  (   127,189,074 rd   +       506,963 wr)
==9112== LL misses:        126,753,960  (   126,247,638 rd   +       506,322 wr)
==9112== LL miss rate:             0.2% (           0.2%     +           0.1%  )
```



Average Running time for version 1(for n = 2000) :
(95.3+96.2+94.6+93.5+95.9)/5 = **95.1 sec**

Average Running time for version 2(for n = 2000) :
(35.338+36.314+35.521+35.323+36.123)/5 = **35.723 sec**

Average Running time for version 3(for n = 2000) :
(32.356+32.452+33.856+32.102+32.312)/5 = **32.615 sec**

Average Running time for version 1(for n = 4000) :
(998+997+1001+997+999)/5 = **998.4 sec**

Average Running time for version 2(for n = 4000) :
(264+271+270+266+269)/5 = **268 sec**

Average Running time for version 3(for n = 4000) :
(242+239+244+249+243)/5 = **243.4 sec**
