Optimizing Matrix chain multiplication

2019101098
pravalika mukkiri

Optimizing the code for Matrix chain multiplication

Matrix chain multiplication

For 4 matrices with result size 1000 1000

Time taken without optimization:
34.32 seconds

Optimizations made in Assignment 1:

1. Changing the loop order to improve the chances of getting cache hit. Previous order was i,j,k. I changed it to i,k,j.

2. Pre increment over post increment Pre-increment is faster than post-increment because post increment keeps a copy of previous (existing) value and adds 1 in the existing value while pre-increment is simply adds 1 without keeping the existing value.

3. Storing in 1D array The best optimisation in terms of time, as now only 1D arrays are used instead of 2D referencing which is most time consuming.

Time taken with the optimizations made in Assignment 1:

9.20 seconds

Optimizations made in this Assignment :

4. Compiler Optimization.
Running the code with -O2 , decreases the time taken for running code.

Time taken: 2.79 seconds.

5. Parlalizing the i th loop.

Parlalizing the loop allows all iterations of the loop to execute in parallel.

Time taken: 1.72 seconds.

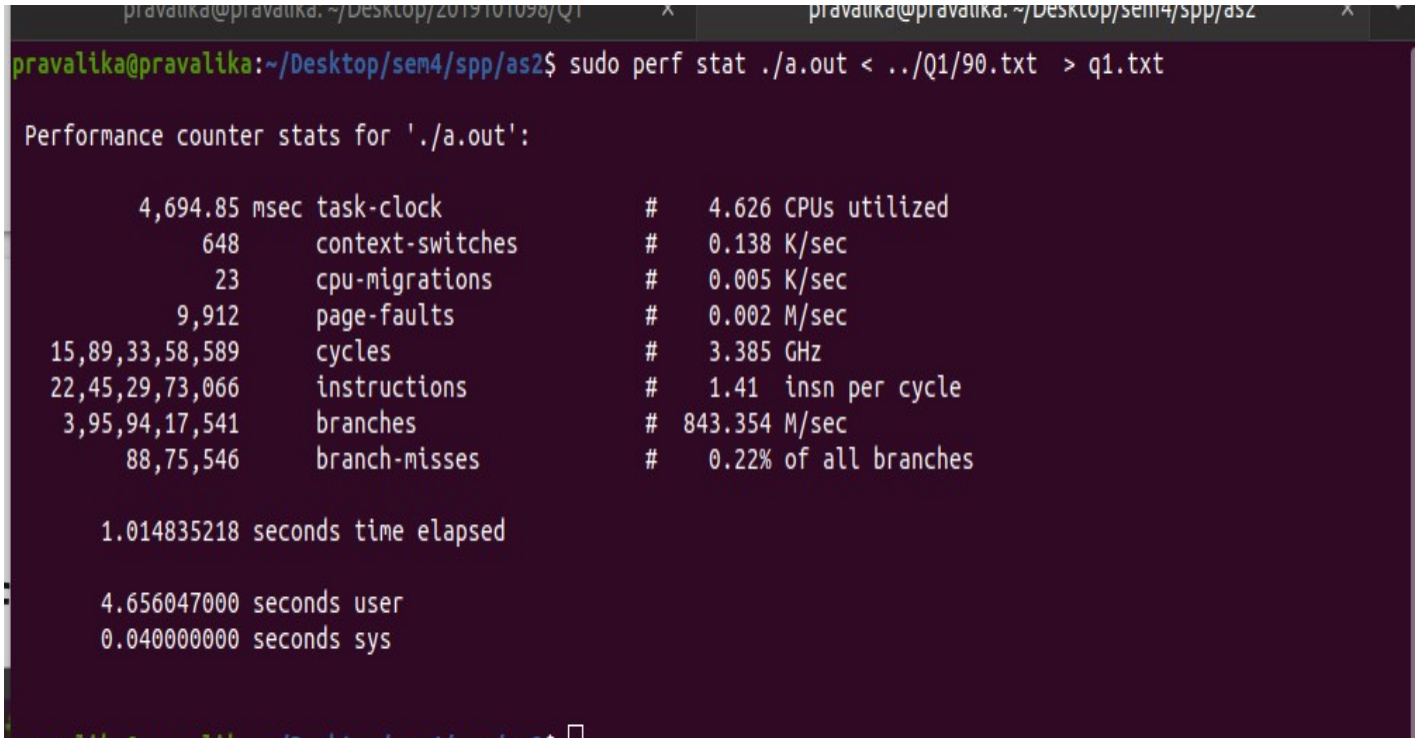
6. Tiled Matrix Multiplication.

We Restructure the computation to reuse data in the cache as much as possible.
In tiled matrix multiplication, cache hits are more and cache misses are low.
We divide the matrix into blockes of size 32 and reuse the data.

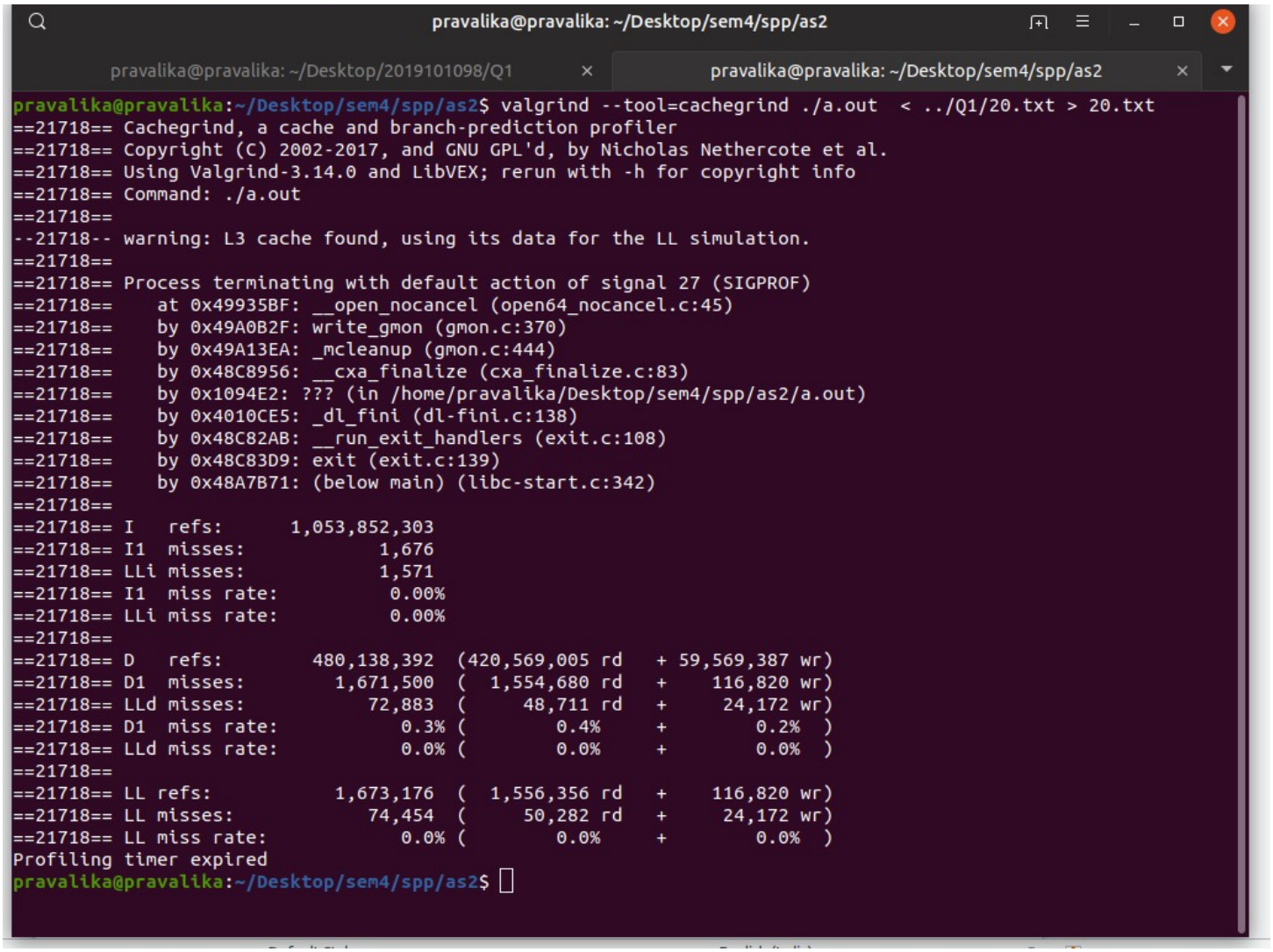
Time taken: 1.22 seconds

Results of perf,cachegrind and gprof for the final optimized code.

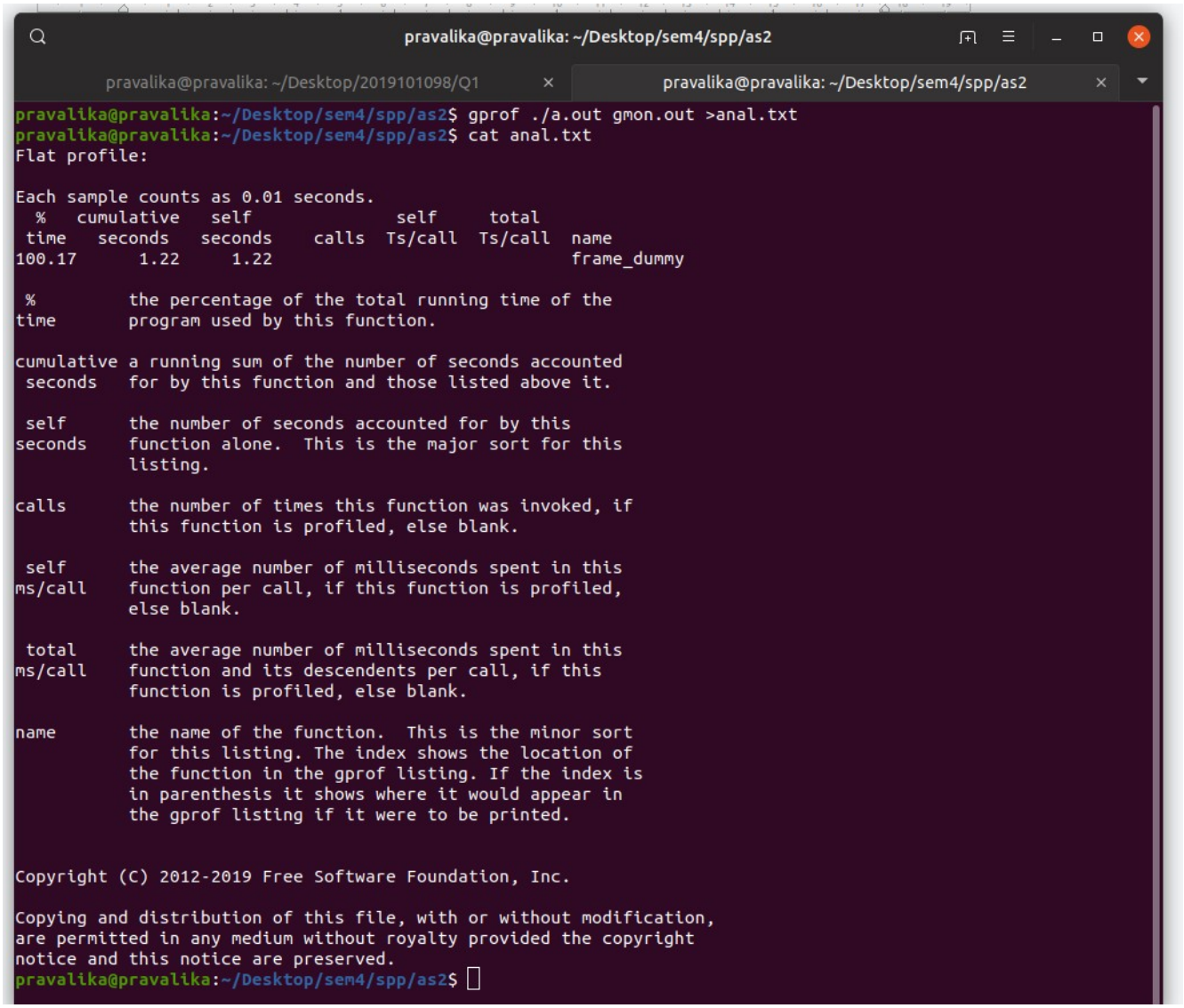
perf



Cachegrind


pravalika@pravalika: ~/Desktop/sem4/spp/as2
pravalika@pravalika: ~/Desktop/2019101098/Q1
pravalika@pravalika: ~/Desktop/sem4/spp/as2
pravalika@pravalika:~/Desktop/sem4/spp/as2$ valgrind --tool=cachegrind ./a.out < ../Q1/20.txt > 20.txt
==21718== Cachegrind, a cache and branch-prediction profiler
==21718== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==21718== Using Valgrind-3.14.0 and LibVEX; rerun with -h for copyright info
==21718== Command: ./a.out
==21718==
--21718-- warning: L3 cache found, using its data for the LL simulation.
==21718==
==21718== Process terminating with default action of signal 27 (SIGPROF)
==21718== at 0x49935BF: __open_nocancel (open64_nocancel.c:45)
==21718== by 0x49A0B2F: write_gmon (gmon.c:370)
==21718== by 0x49A13EA: _mcleanup (gmon.c:444)
==21718== by 0x48C8956: __cxa_finalize (cxa_finalize.c:83)
==21718== by 0x1094E2: ??? (in /home/pravalika/Desktop/sem4/spp/as2/a.out)
==21718== by 0x4010CE5: _dl_fini (dl-fini.c:138)
==21718== by 0x48C82AB: __run_exit_handlers (exit.c:108)
==21718== by 0x48C83D9: exit (exit.c:139)
==21718== by 0x48A7B71: (below main) (libc-start.c:342)
==21718==
==21718== I refs: 1,053,852,303
==21718== I1 misses: 1,676
==21718== LLi misses: 1,571
==21718== I1 miss rate: 0.00%
==21718== LLi miss rate: 0.00%
==21718==
==21718== D refs: 480,138,392 (420,569,005 rd + 59,569,387 wr)
==21718== D1 misses: 1,671,500 (1,554,680 rd + 116,820 wr)
==21718== LLd misses: 72,883 (48,711 rd + 24,172 wr)
==21718== D1 miss rate: 0.3% (0.4% + 0.2%)
==21718== LLd miss rate: 0.0% (0.0% + 0.0%)
==21718==
==21718== LL refs: 1,673,176 (1,556,356 rd + 116,820 wr)
==21718== LL misses: 74,454 (50,282 rd + 24,172 wr)
==21718== LL miss rate: 0.0% (0.0% + 0.0%)
Profiling timer expired
pravalika@pravalika:~/Desktop/sem4/spp/as2$

gprof:


pravalika@pravalika: ~/Desktop/sem4/spp/as2
pravalika@pravalika: ~/Desktop/2019101098/Q1
pravalika@pravalika: ~/Desktop/sem4/spp/as2
pravalika@pravalika:~/Desktop/sem4/spp/as2$ gprof ./a.out gmon.out >anal.txt
pravalika@pravalika:~/Desktop/sem4/spp/as2$ cat anal.txt
Flat profile:
Each sample counts as 0.01 seconds.
% cumulative self self total
time seconds seconds calls Ts/call Ts/call name
100.17 1.22 1.22 frame_dummy
% the percentage of the total running time of the
time program used by this function.
cumulative a running sum of the number of seconds accounted
seconds for by this function and those listed above it.
self the number of seconds accounted for by this
seconds function alone. This is the major sort for this
listing.
calls the number of times this function was invoked, if
this function is profiled, else blank.
self the average number of milliseconds spent in this
ms/call function per call, if this function is profiled,
else blank.
total the average number of milliseconds spent in this
ms/call function and its descendents per call, if this
function is profiled, else blank.
name the name of the function. This is the minor sort
for this listing. The index shows the location of
the function in the gprof listing. If the index is
in parenthesis it shows where it would appear in
the gprof listing if it were to be printed.
Copyright (C) 2012-2019 Free Software Foundation, Inc.
Copying and distribution of this file, with or without modification,
are permitted in any medium without royalty provided the copyright
notice and this notice are preserved.
pravalika@pravalika:~/Desktop/sem4/spp/as2$