CO460 - Assignment 0

Pin, Valgrind, Perf, gprof Tools

Submitted by:
16CO124 Mehnaz Yunus
16CO140 Sharanya Kamath

Submitted to:
Prof. Basavaraj Talawar
CSE Dept. NITK

Q.1. Profile and analyze a family of 4 or more similar programs using the Intel PIN tool.
a) Collect instruction count, Instruction Address Trace, Memory Reference Trace.

Program	Instruction Count
Bubble sort	18589041
Selection sort	9239350
Insertion sort	6336933
Quick sort	584978

Instruction Address Trace and Memory Reference Trace are in the submitted folder.

b) Instruction mix must be collected with the total number of dynamic instructions, integer, floating-point, load, store, branch.

	Bubble sort	Selection sort	Insertion sort	Quicksort
Total dynamic instructions	8651641	3592563	2364973	261325
Branch	1034	1034	1034	1034
Load	6847924	3557942	2087151	186973
Store	1802072	32985	276182	72717
Integer	608	601	605	600

	Bubble sort	Selection sort	Insertion sort	Quicksort
RAW	826	825	820	831
WAW	755	755	750	763
WAR	921	923	923	924

Q.2. Write a program to find the largest eigenvalue of a 1000x1000 real symmetric matrix using the Power Iteration algorithm, then analyze the cache and branch prediction using Valgrind. Report the cache and branch statistics for both, using a matrix size of N=1000. Reason the cache hit/miss behavior in both the variants.

First Variant:

Instruction Cache refs: 149,448,480
First level instruction cache I1 misses: 1,158
Last level instruction cache LLi misses: 1,146
I1 miss rate: 0.00%
LLi miss rate: 0.00%

Data cache D refs: 76,253,366 (63,210,572 rd + 13,042,794 wr)
First level data cache D1 misses: 1,253,521 (190,815 rd + 1,062,706 wr)
Last level data cache LLd misses: 248,336 (173,224 rd + 75,112 wr)
D1 miss rate: 1.6% (0.3% + 8.1%)
LLd miss rate: 0.3% (0.3% + 0.6%)

Last level cache LL refs: 1,254,679 (191,973 rd + 1,062,706 wr)
LL misses: 249,482 (174,370 rd + 75,112 wr)
LL miss rate: 0.1% (0.1% + 0.6%)

Branches: 15,053,264 (14,051,897 cond + 1,001,367 ind)
Mispredicts: 74,290 (74,146 cond + 144 ind)
Mispred rate: 0.5% (0.5% + 0.0%)

Second Variant:

Instruction Cache refs: 149,448,472
First level instruction cache I1 misses: 1,158
Last level instruction cache LLi misses: 1,146
I1 miss rate: 0.00%
LLi miss rate: 0.00%

Data cache D refs: 76,253,358 (63,210,564 rd + 13,042,794 wr)
First level data cache D1 misses: 4,254,859 (3,192,141 rd + 1,062,718 wr)
Last level data cache LLd misses: 247,004 (171,890 rd + 75,114 wr)

D1 miss rate: 5.6% (5.1% + 8.1%)
LLd miss rate: 0.3% (0.3% + 0.6%)

Last level cache LL refs: 4,256,017 (3,193,299 rd + 1,062,718 wr)
LL misses: 248,150 (173,036 rd + 75,114 wr)
LL miss rate: 0.1% (0.1% + 0.6%)

Branches: 15,053,264 (14,051,897 cond + 1,001,367 ind)
Mispredicts: 74,308 (74,164 cond + 144 ind)
Mispred rate: 0.5% (0.5% + 0.0%)

The number of L1 instruction caches misses is insignificant in both variants as the miss rate is always 0%. It means that both programs fit in the L1 instruction cache.
The second part of the output reports information about L1 and LL (last level cache, L3) data caches. Using the *D1 miss rate,* it can be seen that the first variant is more cache efficient as the miss rate is less.

The final part of cachegrind output sums up information about LL (last level cache, L3 in your case) for both instructions and data. It thus gives the number of memory accesses and the percentage of memory requests served by the cache.

Q.3. Write a program using i) recursion, and ii) dynamic programming techniques to find the minimum number of multiplications needed to multiply a chain of matrices. Report the Performance counter stats for both the programs using the perf profiler.

	Recursion	**Dynamic Programming**
Minimum number of multiplications	5250	5250
Time elapsed	1.53 sec	0.878 sec
Cache misses	14845 (53.296%)	13479 (56.756%)
Cache hits	13009 (46.704%)	10270 (43.244%)
Instructions per cycle	0.5	0.61
No. of instructions	7,53,054	7,33,900
Cycles	1.832 GHz	0.896 GHz
CPUs utilized	0.001	0.002

I/D cache hits/misses in recursion method:

```
sharanya@sharanya:~/High-Performance-Computing/A-0/Q.3$ gcc -Wall -pg Q3-recursive.c -o Q3-recursive
sharanya@sharanya:~/High-Performance-Computing/A-0/Q.3$ valgrind --tool=cachegrind --cache-sim=yes --branch-sim=yes ./Q3-recursive
==19644== Cachegrind, a cache and branch-prediction profiler
==19644== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==19644== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==19644== Command: ./Q3-recursive
==19644==
--19644-- warning: L3 cache found, using its data for the LL simulation.
Minimum number of multiplications is 5250
==19644==
==19644== Process terminating with default action of signal 27 (SIGPROF)
==19644==    at 0x4D3BDAF: __open_nocancel (open64.c:69)
==19644==    by 0x4D4F91F: write_gmon (gmon.c:370)
==19644==    by 0x4D500DA: _mcleanup (gmon.c:444)
==19644==    by 0x4C6F040: __run_exit_handlers (exit.c:108)
==19644==    by 0x4C6F139: exit (exit.c:139)
==19644==    by 0x4C4DB9D: (below main) (libc-start.c:344)
==19644==
==19644== I   refs:      203,376
==19644== I1  misses:      1,093
==19644== LLi misses:      1,080
==19644== I1  miss rate:    0.54%
==19644== LLi miss rate:    0.53%
==19644==
==19644== D   refs:       63,879  (49,601 rd   + 14,278 wr)
==19644== D1  misses:      3,271  ( 2,597 rd   +    674 wr)
==19644== LLd misses:      2,685  ( 2,082 rd   +    603 wr)
==19644== D1  miss rate:     5.1% (   5.2%     +    4.7%  )
==19644== LLd miss rate:     4.2% (   4.2%     +    4.2%  )
==19644==
==19644== LL refs:         4,364  ( 3,690 rd   +    674 wr)
==19644== LL misses:       3,765  ( 3,162 rd   +    603 wr)
==19644== LL miss rate:      1.4% (   1.2%     +    4.2%  )
==19644==
==19644== Branches:       43,710  (43,310 cond +    400 ind)
==19644== Mispredicts:     5,351  ( 5,198 cond +    153 ind)
==19644== Mispred rate:     12.2% (  12.0%     +   38.2%   )
```

I/D cache hits/misses in the dynamic programming method:

```
sharanya@sharanya:~/High-Performance-Computing/A-0/Q.3$ gcc -Wall -pg Q3-dynamic.c -o Q3-dynamic
sharanya@sharanya:~/High-Performance-Computing/A-0/Q.3$ valgrind --tool=cachegrind --cache-sim=yes --branch-sim=yes ./Q3-dynamic
==19670== Cachegrind, a cache and branch-prediction profiler
==19670== Copyright (C) 2002-2017, and GNU GPL'd, by Nicholas Nethercote et al.
==19670== Using Valgrind-3.13.0 and LibVEX; rerun with -h for copyright info
==19670== Command: ./Q3-dynamic
==19670==
--19670-- warning: L3 cache found, using its data for the LL simulation.
Minimum number of multiplications is 5250
==19670==
==19670== I   refs:      205,151
==19670== I1  misses:      1,216
==19670== LLi misses:      1,192
==19670== I1  miss rate:    0.59%
==19670== LLi miss rate:    0.58%
==19670==
==19670== D   refs:       64,521  (49,739 rd   + 14,782 wr)
==19670== D1  misses:      3,312  ( 2,623 rd   +    689 wr)
==19670== LLd misses:      2,699  ( 2,088 rd   +    611 wr)
==19670== D1  miss rate:     5.1% (   5.3%     +    4.7%  )
==19670== LLd miss rate:     4.2% (   4.2%     +    4.1%  )
==19670==
==19670== LL refs:         4,528  ( 3,839 rd   +    689 wr)
==19670== LL misses:       3,891  ( 3,280 rd   +    611 wr)
==19670== LL miss rate:      1.4% (   1.3%     +    4.1%  )
==19670==
==19670== Branches:       44,599  (44,208 cond +    391 ind)
==19670== Mispredicts:     5,504  ( 5,336 cond +    168 ind)
==19670== Mispred rate:     12.3% (  12.1%     +   43.0%   )
sharanya@sharanya:~/High-Performance-Computing/A-0/Q.3$
```

Explanation:

Dynamic programming is a solution for a special type of application such as context-free language recognition matrix chain multiplication. Unfortunately, these implementations exhibit poor cache performance because most dynamic programming has high complexity (n^2 ,n^3 and worse).

Q.4. Write a program to solve travelling salesman problem using recursive functions. Profile the program using " gprof " tool to analyse the flat profile and call graphs of the functions used.

Flat Profile:

Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self Ts/call	total Ts/call	name
0.00	0.00	0.00	4	0.00	0.00	tsp
0.00	0.00	0.00	1	0.00	0.00	minimum_cost

Call graphs:

granularity: each sample hit covers 2 byte(s) no time propagated

index	% time	self	children	called	name
		0.00	0.00	4/4	minimum_cost [2]
[1]	0.0	0.00	0.00	4	tsp[1]
---	---	---	---	---	---
				3	minimum_cost [2]
		0.00	0.00	1/1	main [8]
[2]	0.0	0.00	0.00	1+3	minimum_cost [2]
		0.00	0.00	4/4	tsp [1]
				3	minimum_cost [2]
---	---	---	---	---	---