ΕΛΠ 605: Προχωρημένη Αρχιτεκτονική Υπολογιστών

Εργαστήριο 3

Linux Monitoring Utilities (perf,top,mpstat ps, free) and gdb dissasembler, gnuplot

top

Realtime monitoring of:

- CPU and memory utilization for **each process**
- **Total CPU utilization** – average and per core
- **Total Memory utilization** (used and free)

Useful Command switches:

- top –d 1 #set the update interval to 1 second //the default is 3 seconds
- top –b #run in batch mode, top will run until killed, useful for saving top output in a file
- top –H # instruct top to show individual threads
- top –u username # show processes of a specific user only

top

- **>taskset -c 0** ./matrix_serial_ver1 &> /dev/null &
- >top

```
top - 10:28:36 up 4 days, 21:06, 12 users,  load average: 1.14, 0.90, 0.57
Tasks: 382 total,   3 running, 379 sleeping,   0 stopped,   0 zombie
%Cpu(s): 27.3 us,  0.7 sy,  0.1 ni, 71.8 id,  0.1 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7933440 total,  1181480 free,  3488460 used,  3263500 buff/cache
KiB Swap: 10485756 total, 10465824 free,    19932 used.  3514920 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
15112 zhadji01  20   0   15928  11484    348 R  99.7  0.1   0:03.27 matrix_serial_v
 5565 xioann02  20   0 2564768 426032 147408 S   2.3  5.4   3:48.97 firefox
 5629 xioann02  20   0 2168824 351408  81028 S   0.7  4.4   1:17.09 Web Content
```

- CPU utilization explanation: us (user time) sy (system time) ni (processes that run at higher priority) id (idle time) wa (cpu waiting for I/O), hi si (hardware and software interrupts handling)
- User zhadji01 is running matrix_serial_v and xioann02 runs firefox, total main memory is 8GB (7933440KB)
- matrix_serial_v consumes **99.7% CPU time** and 0.1% of total memory
- **Average CPU utilization is 27.3% and the CPU has four cores this means that ~one core is fully utilized**
- Press key 1 to view CPU utilization per core

```
top - 10:29:03 up 4 days, 21:07, 12 users,  load average: 1.09, 0.91, 0.58
Tasks: 381 total,   2 running, 379 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  :  5.3 us,  3.4 sy,  0.0 ni, 91.3 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :  2.0 us,  1.0 sy,  0.0 ni, 96.6 id,  0.5 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :  4.4 us,  1.5 sy,  0.0 ni, 94.2 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7933440 total,  1178116 free,  3491808 used,  3263516 buff/cache
KiB Swap: 10485756 total, 10465824 free,    19932 used.  3511572 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
15458 zhadji01  20   0   15928  10432    348 R 100.0  0.1   0:02.29 matrix_serial_v
```

- Indeed **Core0** is fully utilized at 100%, matrix_serial_v runs at core0 as instructed by taskset

top –H to view threads of multithreaded programs

- >./simpleParallelProgram &
- >top

```
top - 12:13:19 up 4 days, 22:51, 12 users,  load average: 0.84, 0.40, 0.25
Tasks: 378 total,   2 running, 376 sleeping,   0 stopped,   0 zombie
%Cpu(s): 98.5 us,  1.5 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7933440 total,   919868 free,  3737124 used,  3276448 buff/cache
KiB Swap: 10485756 total, 10465824 free,    19932 used.  3258108 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S  %CPU %MEM     TIME+ COMMAND
30013 zhadji01  20   0   33292    604    484 R 400.0  0.0   0:06.92 simpleParallelP
```

400% CPU utilization means it uses 4 cores

top –H to view threads

>top –H

```
top - 12:12:35 up 4 days, 22:50, 12 users,  load average: 0.83, 0.32, 0.22
Threads: 1093 total,   5 running, 1088 sleeping,   0 stopped,   0 zombie
%Cpu0  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu1  : 97.8 us,  2.2 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu2  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
%Cpu3  :100.0 us,  0.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
KiB Mem :  7933440 total,   931660 free,  3725376 used,  3276404 buff/cache
KiB Swap: 10485756 total, 10465824 free,    19932 used.  3269888 avail Mem

  PID USER      PR  NI    VIRT    RES    SHR S %CPU %MEM     TIME+ COMMAND
29481 zhadji01  20   0   33292    600    484 R 99.9  0.0   0:02.49 simpleParallelP
29478 zhadji01  20   0   33292    600    484 R 99.9  0.0   0:02.64 simpleParallelP
29479 zhadji01  20   0   33292    600    484 R 99.9  0.0   0:02.56 simpleParallelP
29480 zhadji01  20   0   33292    600    484 R 97.7  0.0   0:02.60 simpleParallelP
```

Each thread has ~100% cpu utilization meaning it utilizes fully one core

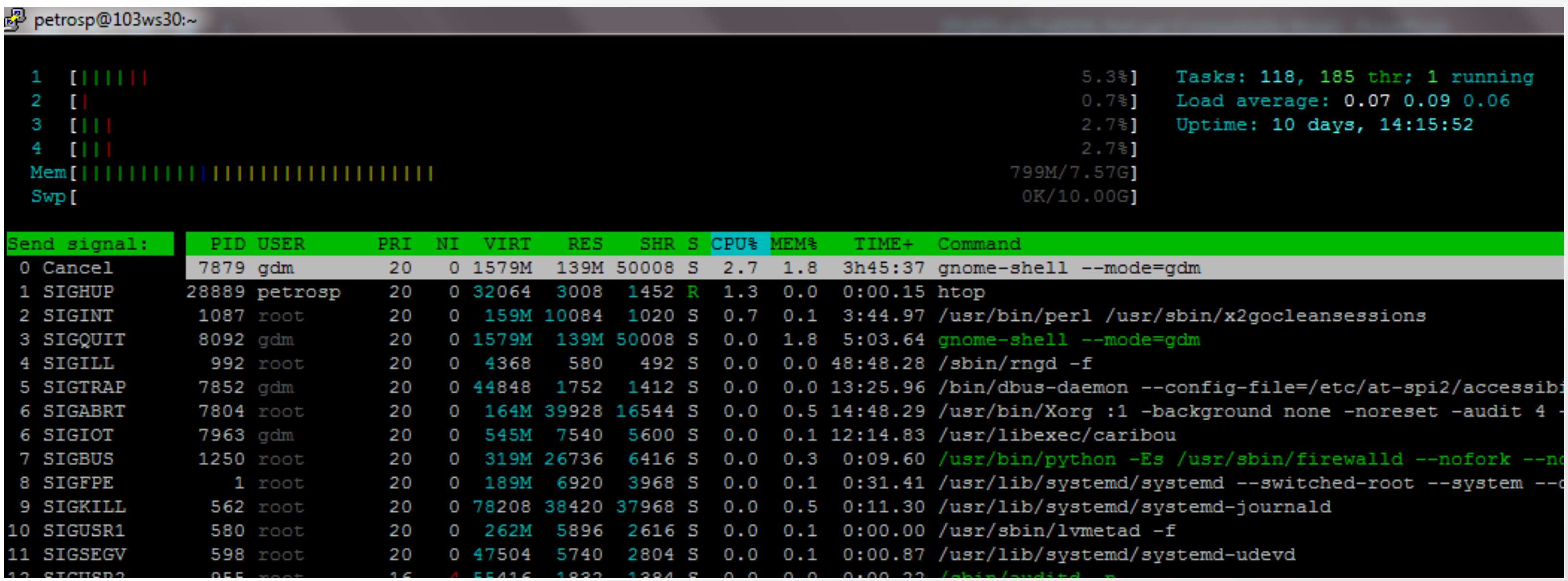
top useful keys in interactive mode

- Press 1 to view per core utilization
- Press Shift+p to sort process from higher CPU utilization to lower
- Press u to view specific user

Htop

htop (https://linux.die.net/man/1/htop)

an interactive system-monitor process-viewer



ps command

- Gives a snapshot of all processes
- >ps aux

```
103ws1:/home/research/zhadji01/EPL605labs>ps aux
USER       PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND
root         1  0.0  0.0 193884  7124 ?        Rs   Sep21   1:15 /usr/lib/systemd/systemd --switched-root --system --deserialize 21
root         2  0.0  0.0      0     0 ?        S    Sep21   0:00 [kthreadd]
root         3  0.0  0.0      0     0 ?        S    Sep21   0:00 [ksoftirqd/0]
root         5  0.0  0.0      0     0 ?        S<   Sep21   0:00 [kworker/0:0H]
root         7  0.0  0.0      0     0 ?        S    Sep21   0:03 [migration/0]
root         8  0.0  0.0      0     0 ?        S    Sep21   0:00 [rcu_bh]
root         9  0.0  0.0      0     0 ?        S    Sep21   2:39 [rcu_sched]
root        10  0.0  0.0      0     0 ?        S    Sep21   0:01 [watchdog/0]
root        11  0.0  0.0      0     0 ?        S    Sep21   0:01 [watchdog/1]
root        12  0.0  0.0      0     0 ?        S    Sep21   0:02 [migration/1]
```

- >ps –ef

```
103ws1:/home/research/zhadji01/EPL605labs>ps -eLF | head
UID        PID  PPID   LWP  C NLWP    SZ   RSS PSR STIME TTY          TIME CMD
root         1     0     1  0    1 48471  7124   0 Sep21 ?        00:01:15 /usr/lib/systemd/systemd --switched-root --system --deserialize 21
root         2     0     2  0    1     0     0   3 Sep21 ?        00:00:00 [kthreadd]
root         3     2     3  0    1     0     0   0 Sep21 ?        00:00:00 [ksoftirqd/0]
root         5     2     5  0    1     0     0   0 Sep21 ?        00:00:00 [kworker/0:0H]
root         7     2     7  0    1     0     0   0 Sep21 ?        00:00:03 [migration/0]
root         8     2     8  0    1     0     0   0 Sep21 ?        00:00:00 [rcu_bh]
root         9     2     9  0    1     0     0   0 Sep21 ?        00:02:39 [rcu_sched]
root        10     2    10  0    1     0     0   0 Sep21 ?        00:00:01 [watchdog/0]
root        11     2    11  0    1     0     0   1 Sep21 ?        00:00:01 [watchdog/1]
```

- ps -eLF # information about threads

mpstat

A good tool to view CPU utilization

mpstat -P ALL 2 1

```
103ws1:/home/research/zhadji01/EPL605labs>mpstat -P ALL 2 1
Linux 3.10.0-693.21.1.el7.x86_64 (103ws1)       09/26/2018      _x86_64_

12:22:11 PM  CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
12:22:13 PM  all    3.01    0.00    0.88    0.13    0.00    0.00    0.00    0.00
12:22:13 PM    0    4.02    0.00    1.01    0.00    0.00    0.00    0.00    0.00
12:22:13 PM    1    5.56    0.00    1.52    0.00    0.00    0.00    0.00    0.00
12:22:13 PM    2    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
12:22:13 PM    3    1.01    0.00    1.01    0.00    0.00    0.00    0.00    0.00

Average:     CPU    %usr   %nice    %sys %iowait    %irq   %soft  %steal  %guest
Average:     all    3.01    0.00    0.88    0.13    0.00    0.00    0.00    0.00
Average:       0    4.02    0.00    1.01    0.00    0.00    0.00    0.00    0.00
Average:       1    5.56    0.00    1.52    0.00    0.00    0.00    0.00    0.00
Average:       2    1.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
Average:       3    1.01    0.00    1.01    0.00    0.00    0.00    0.00    0.00
```

-P ALL show all cores

2 1 show two reports with one second interval between them

free

Tool to view memory utilization

free

```
103ws1:/home/research/zhadji01/EPL605labs>free -g
              total        used        free      shared  buff/cache   available
Mem:              7           3           0           0           3           3
Swap:             9           0           9
```

perf: Linux ***profiling*** with performance counters

Performance counters are CPU hardware registers that count hardware events such as instructions executed, cache-misses suffered, or branches mispredicted.

perf provides rich generalized abstractions over hardware specific capabilities. Among others, it provides per task, per CPU and per-workload counters, sampling on top of these and source code event annotation. Perf gives you visibility where the Hotspots of your program are.

https://perf.wiki.kernel.org/index.php/Main_Page

Intel Core Performance Monitor Unit (PMU)

The core PMU's capability is similar to those described in Section 18.7.1 and Section 18.8, with some differences and enhancements relative to Intel microarchitecture code name Westmere summarized in Table 18-25.

Table 18-25. Core PMU Comparison

Box	Intel® microarchitecture code name Sandy Bridge	Intel® microarchitecture code name Westmere	Comment
# of Fixed counters per thread	3	3	Use CPUID to enumerate # of counters.
# of general-purpose counters per core	8	8	
Counter width (R,W)	R:48 , W: 32/48	R:48, W:32	See Section 18.2.2.3.
# of programmable counters per thread	4 or (8 if a core not shared by two threads)	4	Use CPUID to enumerate # of counters.
Precise Event Based Sampling (PEBS) Events	See Table 18-27	See Table 18-10	IA32_PMC4-IA32_PMC7 do not support PEBS.
PEBS-Load Latency	See Section 18.9.4.2; Data source encoding, STLB miss encoding, Lock transaction encoding	Data source encoding	
PEBS-Precise Store	Section 18.9.4.3	No	
PEBS-PDIR	yes (using precise INST_RETIRED.ALL)	No	
Off-core Response Event	MSR 1A6H and 1A7H; Extended request and response types	MSR 1A6H and 1A7H, limited response types	Nehalem supports 1A6H only.

Limited number of hardware counters (8 counters per core on the above example)

Time multiplexing is performed when selected events > hardware counters. An estimation of actual account is given

e.g. user wants to measure instructions and cycles but only one counter is available, perf will measure half of the time the instructions and half of the time the cycles. The measured instructions and cycles will be multiplied by 2 to give an estimation of the actual total instructions and cycles

Perf Events

```
>perf list
List of pre-defined events (to be used in -e):
  cpu-cycles OR cycles                                [Hardware event]
  instructions                                        [Hardware event]
  cache-references                                    [Hardware event]
  cache-misses                                        [Hardware event]
  branch-instructions OR branches                     [Hardware event]
  branch-misses                                       [Hardware event]
  bus-cycles                                          [Hardware event]
  stalled-cycles-frontend OR idle-cycles-frontend     [Hardware event]
  stalled-cycles-backend OR idle-cycles-backend       [Hardware event]
  ref-cycles                                          [Hardware event]

  cpu-clock                                           [Software event]
  task-clock                                          [Software event]
  page-faults OR faults                               [Software event]
  context-switches OR cs                              [Software event]
  cpu-migrations OR migrations                        [Software event]
  minor-faults                                        [Software event]
  major-faults                                        [Software event]
  alignment-faults                                    [Software event]
  emulation-faults                                    [Software event]

  L1-dcache-loads                                     [Hardware cache event]
  L1-dcache-load-misses                               [Hardware cache event]
  L1-dcache-stores                                    [Hardware cache event]
  L1-dcache-store-misses                              [Hardware cache event]
  L1-dcache-prefetches                                [Hardware cache event]
```

Measuring multiple events

```
perf stat -e instructions,cycles matrix_serial_ver1
Performance counter stats for './matrix_serial_ver1':

    34,058,795,490      instructions              #    2.67  insn per cycle
    12,762,609,265      cycles

       3.487285969 seconds time elapsed
```

To measure more than one event, after -e provide a comma-separated list :

```
perf stat -e cycles,instructions,cache-misses ./matrix_serial_ver1
```

To save the output to a file use –o switch

```
perf stat –o tmp -e cycles,instructions,cache-misses ./matrix_serial_ver1
cat tmp
Performance counter stats for './matrix_serial_ver1':

    34,058,795,490      instructions              #    2.67  insn per cycle
    12,762,609,265      cycles

       3.487285969 seconds time elapsed
```

Attach to already running process

To attach to running process –p (-t to attach to thread)
./matrix_serial_ver1 &

```
perf stat -e instructions,cycles -p $! & ##$! Is the Pid of last launced process
or do
perf stat -e instructions,cycles -p `pgrep matrix_serial` &

pkill -SIGINT perf # send signal interrupt to perf to make perf print statistics
Performance counter stats for './matrix_serial_ver1':

    34,058,795,490      instructions              #    2.67  insn per cycle
    12,762,609,265      cycles

       3.487285969 seconds time elapsed

perf stat -e instructions,cycles -p $! sleep 1 # perf will run only for 1 second
                                                 in this case pkill -SIGINT is
                                                 not required
```

https://perf.wiki.kernel.org/index.php/Tutorial

System wide collection

xg3:/home/root_desktop>./run_NPB.sh ./NPB3.3/NPB3.3-OMP/bin/ sp.C.x 32 &> /dev/null &

we started a multithreaded workload that uses 32 cores, each core runs at 3GHz

##To sum the instructions and cycles executed by all cores

perf stat -e instructions,cycles -a sleep 1

Performance counter stats for 'system wide':

34,812,562,418 instructions # **0.36 insn per cycle**

95,655,967,640 cycles //**Each core at 3GHz should have 3Billion cycles in 1seconds, multiply by 32~96Billion**

1.018783051 seconds time elapsed

0.36 instructions per cycle (IPC) is indicative of single thread performance

To measure the actual IPC of all cores do

perf stat -e instructions,cycles -A -C 0-31 sleep 1

-A disables statistics aggregation –C defines which core statistics to print

Per core stats

```
xg3:/home/root_desktop>perf stat -e instructions,cycles -A -C 0-31 sleep 1

 Performance counter stats for 'CPU(s) 0-31':

CPU0          910,332,242      instructions              #    0.32  insn per cycle           (37.27%)
CPU1          934,146,000      instructions              #    0.33  insn per cycle           (37.27%)
CPU2          931,834,917      instructions              #    0.33  insn per cycle           (37.27%)
CPU3          929,731,042      instructions              #    0.33  insn per cycle           (37.27%)
CPU4        1,014,089,724      instructions              #    0.36  insn per cycle           (37.27%)
CPU5          930,204,464      instructions              #    0.33  insn per cycle           (37.27%)
CPU6          936,039,147      instructions              #    0.33  insn per cycle           (37.27%)
CPU7          932,873,909      instructions              #    0.33  insn per cycle           (37.27%)
CPU8          956,440,472      instructions              #    0.34  insn per cycle           (37.27%)
CPU9        1,032,696,650      instructions              #    0.36  insn per cycle           (37.27%)
CPU10         932,002,615      instructions              #    0.33  insn per cycle           (37.27%)
CPU11         930,507,688      instructions              #    0.33  insn per cycle           (37.27%)
CPU12         929,803,805      instructions              #    0.33  insn per cycle           (37.27%)
CPU13         930,445,510      instructions              #    0.33  insn per cycle           (37.27%)
CPU14         926,390,523      instructions              #    0.33  insn per cycle           (37.27%)
CPU15         937,865,986      instructions              #    0.33  insn per cycle           (37.27%)
CPU16         922,269,780      instructions              #    0.32  insn per cycle           (37.27%)
CPU17         927,120,167      instructions              #    0.33  insn per cycle           (37.27%)
CPU18         926,974,317      instructions              #    0.33  insn per cycle           (37.27%)
CPU19         923,285,989      instructions              #    0.32  insn per cycle           (37.27%)
CPU20         928,399,977      instructions              #    0.33  insn per cycle           (37.27%)
CPU21         925,085,806      instructions              #    0.33  insn per cycle           (37.27%)
CPU22         932,359,502      instructions              #    0.33  insn per cycle           (37.27%)
CPU23         922,616,379      instructions              #    0.32  insn per cycle           (37.27%)
CPU24         922,931,156      instructions              #    0.32  insn per cycle           (37.27%)
CPU25         926,020,376      instructions              #    0.33  insn per cycle           (37.27%)
CPU26         928,917,702      instructions              #    0.33  insn per cycle           (37.27%)
CPU27         925,014,066      instructions              #    0.33  insn per cycle           (37.27%)
CPU28         925,477,302      instructions              #    0.33  insn per cycle           (37.27%)
CPU29         925,664,820      instructions              #    0.33  insn per cycle           (37.28%)
CPU30         921,220,980      instructions              #    0.32  insn per cycle           (37.27%)
CPU31         930,765,008      instructions              #    0.33  insn per cycle           (37.27%)
```

The actual CPU IPC is ~0.33 * 32 =10.56

Per core and system wide collection required root access, or administrator allowing global perf collections (set /proc/sys/kernel/perf_event_paranoid to -1)

Perf top

Shows realtime the most hot function

```
Samples: 351K of event 'cycles:ppp', Event count (approx.): 241386338169
Overhead  Shared Object        Symbol
  37.12%  sp.C.x               [.] compute_rhs_._omp_fn.0
  17.48%  sp.C.x               [.] z_solve_._omp_fn.0
  17.46%  sp.C.x               [.] y_solve_._omp_fn.0
  16.08%  sp.C.x               [.] x_solve_._omp_fn.0
   2.56%  sp.C.x               [.] txinvr_._omp_fn.0
   2.26%  sp.C.x               [.] tzetar_._omp_fn.0
   1.94%  sp.C.x               [.] add_._omp_fn.0
   1.23%  sp.C.x               [.] ninvr_._omp_fn.0
   1.11%  sp.C.x               [.] pinvr_._omp_fn.0
   0.81%  libgomp.so.1.0.0     [.] 0x00000000000185f0
   0.40%  libgomp.so.1.0.0     [.] 0x0000000000018600
   0.10%  libgomp.so.1.0.0     [.] 0x00000000000183cc
   0.09%  [kernel]             [k] arch_cpu_idle
   0.08%  libgomp.so.1.0.0     [.] 0x00000000000183dc
   0.07%  sp.C.x               [.] lhsinit_
   0.05%  [kernel]             [k] finish_task_switch
   0.04%  bash                 [.] 0x000000000003f788
   0.04%  perf                 [.] eprintf
   0.04%  libc-2.17.so         [.] strcmp
   0.04%  sp.C.x               [.] lhsinitj_
   0.03%  [kernel]             [k] copy_page
```

Matrix Multiplication Examples

gcc -Werror -Wall matrix_serial_ver1.c -o matrix_serial_ver1.out
>./matrix_serial_ver1.out
Elapsed Time: 6.32 Sec.

>gcc -Werror -Wall matrix_serial_ver2.c -o matrix_serial_ver2.out
>./matrix_serial_ver2.out
Elapsed Time: 5.38 Sec.

>gcc -Werror -Wall matrix_serial_ver3.c -o matrix_serial_ver3.out
>./matrix_serial_ver3.out
Elapsed Time: 4.77 Sec.

>gcc -Werror -Wall matrix_serial_ver4.c -o matrix_serial_ver4.out
./matrix_serial_ver4.out
Elapsed Time: 4.60 Sec.

Matrix Multiplication Examples

>perf stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver1.out

Elapsed Time: 6.35 Sec.

Performance counter stats for './matrix_serial_ver1.out':

18,929,166,306 cycles	# 0.000 GHz	[50.00%]	
34,062,608,328 instructions	# **1.80 insns per cycle**		[75.01%]
1,066,881,565 cache-references			[74.99%]
83,905 cache-misses	# 0.008 % of all cache refs		[75.02%]

6.362608904 seconds time elapsed

>perf stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver2.out

Elapsed Time: 5.40 Sec.

Performance counter stats for './matrix_serial_ver2.out':

16,114,935,514 cycles	# 0.000 GHz	[50.00%]	
34,051,559,029 instructions	# **2.11 insns per cycle**		[75.01%]
64,741,737 cache-references			[74.99%]
24,675 cache-misses	# 0.038 % of all cache refs		[75.02%]

5.418502175 seconds time elapsed

Both ver1,and ver2 have same executed instructions but ver2 finishes 15% faster, why? because it has 15% faster IPC

Source of faster IPC: less LLC references, better caching behavior in L1 (the cache references in this example refer to last level cache (LLC) which is the slowest)

Cache-miser per 1K instructions (MPKI) = (cache-misses/instructions)*1000

Cache-accesses per 1K instructions (APKI) = (cache-references/instructions)*1000

Ver1 has MPKI 0.002

Ver2 has MPKI 0.0007 #none of the two versions have significant memory references,

Ver 1 has APKI 56 # but the version1 visits much more frequently the LLC cache

Ver2 has 0.001

Matrix Multiplication Examples

```
>gcc -Werror -Wall matrix_serial_ver3.c -o matrix_serial_ver3.out
> stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver3.out
Elapsed Time: 4.79 Sec.
 Performance counter stats for './matrix_serial_ver3.out':

   14,296,893,562 cycles            #   0.000 GHz                [50.00%]
   20,059,297,912 instructions      #   1.40  insns per cycle              [74.99%] CPI=0.71
    1,067,327,910 cache-references                                         [75.02%]
          107,539 cache-misses      #   0.010 % of all cache refs          [74.99%]
      4.804650637 seconds time elapsed

>gcc -Werror -Wall matrix_serial_ver4.c -o matrix_serial_ver4.out
> stat -e cycles -e instructions -e cache-references -e cache-misses ./matrix_serial_ver4.out
Elapsed Time: 4.55 Sec.
 Performance counter stats for './matrix_serial_ver4.out':
   13,582,459,137 cycles            #   0.000 GHz                [50.01%]
   26,075,199,666 instructions      #   1.92  insns per cycle              [75.02%] CPI=0.52
       64,323,587 cache-references                                         [74.99%]
           30,268 cache-misses      #   0.047 % of all cache refs          [75.00%]
      4.566277059 seconds time elapsed
```

Two versions with different IPC and different instructions executed.. Which is faster?

Recall executionTime = CPI * cycleTime * executedInstructions

CycleTime is the same (run on the same CPU with the same frequency)

Ver2 executes ~1.3X more instructions (26B/20B)

but it has ~ 1.36X lower CPI (1/(0.52/0.71))? Therefore ver2 is ~1.05X faster or 5% faster indeed 4.79 - 0.05*4.8 = 4.55

gcc Optimizations and Branch Prediction

```
gcc main.c -O0 -o main.out
>perf stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 40000034998780787062429554585290932224.00
 Performance counter stats for './main.out':
     4,120,356,326 cycles              #    0.000 GHz              [49.99%]
     5,306,382,355 instructions        #    1.29  insns per cycle  [74.96%]
       899,775,764 branches                                        [75.03%]
         8,742,590 branch-misses       #    0.97% of all branches  [75.03%]
       1.387120365 seconds time elapsed

>gcc main.c -O1 -o main.out
>perf stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 40000034998780787062429554585290932224.00
 Performance counter stats for './main.out':

     1,950,230,142 cycles              #    0.000 GHz              [50.07%]
     2,782,472,831 instructions        #    1.43  insns per cycle  [75.05%]
       600,311,976 branches                                        [75.02%]
         1,368,519 branch-misses       #    0.23% of all branches  [74.95%]
       0.662207246 seconds time elapsed
```

gcc Optimizations and Branch Prediction

```
>gcc main.c -O2 -o main.out
>perf stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 40000003499878078706242955458529093224.00
 Performance counter stats for './main.out':
     1,109,248,454 cycles          #   0.000 GHz                 [49.96%]
     2,599,913,162 instructions    #   2.34  insns per cycle     [75.12%]
       400,489,432 branches                                      [75.12%]
            11,973 branch-misses   #   0.00% of all branches     [74.98%]
       0.379008882 seconds time elapsed

main.c -O3 -o main.out
stat -e cycles -e instructions -e branches -e branch-misses ./main.out
sum = 40000003499878078706242955458529093224.00
 Performance counter stats for './main.out':
     1,108,212,355 cycles          #   0.000 GHz                 [50.09%]
     2,603,070,814 instructions    #   2.35  insns per cycle     [75.09%]
       399,947,490 branches                                      [75.11%]
            12,176 branch-misses   #   0.00% of all branches     [74.93%]
       0.378495972 seconds time elapsed
```

https://gcc.gnu.org/onlinedocs/gcc/Optimize-Options.html

Power monitoring

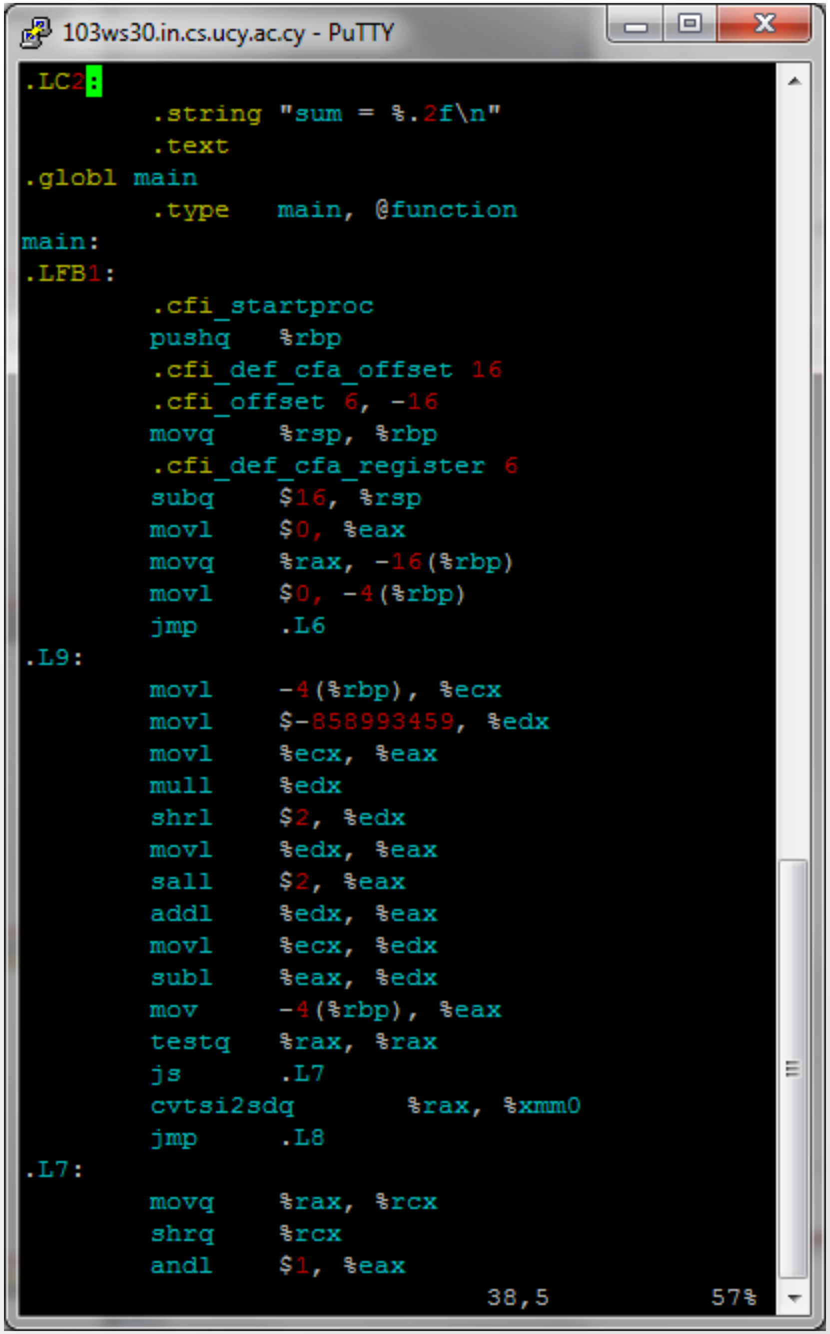
Only on bws103 machines
perf stat -e power/energy-cores/ -e power/energy-ram/ script

Assembly

>gcc –S main.c

>vi main.s

>gcc main.s -o main.s.out



>objdump –d main.out

>hexdump –C main.out

103ws30.in.cs.ucy.ac.cy - PuTTY

```
  40050a:       f2 0f 10 45 d8          movsd  -0x28(%rbp),%xmm0
  40050f:       c9                      leaveq
  400510:       c3                      retq

0000000000400511 <main>:
  400511:       55                      push   %rbp
  400512:       48 89 e5                mov    %rsp,%rbp
  400515:       48 83 ec 10             sub    $0x10,%rsp
  400519:       b8 00 00 00 00          mov    $0x0,%eax
  40051e:       48 89 45 f0             mov    %rax,-0x10(%rbp)
  400522:       c7 45 fc 00 00 00 00    movl   $0x0,-0x4(%rbp)
  400529:       eb 57                   jmp    400582 <main+0x71>
  40052b:       8b 4d fc                mov    -0x4(%rbp),%ecx
  40052e:       ba cd cc cc cc          mov    $0xcccccccd,%edx
  400533:       89 c8                   mov    %ecx,%eax
  400535:       f7 e2                   mul    %edx
  400537:       c1 ea 02                shr    $0x2,%edx
  40053a:       89 d0                   mov    %edx,%eax
  40053c:       c1 e0 02                shl    $0x2,%eax
  40053f:       01 d0                   add    %edx,%eax
  400541:       89 ca                   mov    %ecx,%edx
  400543:       29 c2                   sub    %eax,%edx
  400545:       8b 45 fc                mov    -0x4(%rbp),%eax
  400548:       48 85 c0                test   %rax,%rax
  40054b:       78 07                   js     400554 <main+0x43>
  40054d:       f2 48 0f 2a c0          cvtsi2sd %rax,%xmm0
  400552:       eb 15                   jmp    400569 <main+0x58>
  400554:       48 89 c1                mov    %rax,%rcx
  400557:       48 d1 e9                shr    %rcx
  40055a:       83 e0 01                and    $0x1,%eax
  40055d:       48 09 c1                or     %rax,%rcx
  400560:       f2 48 0f 2a c1          cvtsi2sd %rcx,%xmm0
  400565:       f2 0f 58 c0             addsd  %xmm0,%xmm0
  400569:       89 d7                   mov    %edx,%edi
  40056b:       e8 54 ff ff ff          callq  4004c4 <powern>
  400570:       f2 0f 10 4d f0          movsd  -0x10(%rbp),%xmm1
  400575:       f2 0f 58 c1             addsd  %xmm1,%xmm0
  400579:       f2 0f 11 45 f0          movsd  %xmm0,-0x10(%rbp)
  40057e:       83 45 fc 01             addl   $0x1,-0x4(%rbp)
  400582:       81 7d fc ff e0 f5 05    cmpl   $0x5f5e0ff,-0x4(%rbp)
  400589:       76 a0                   jbe    40052b <main+0x1a>
  40058b:       b8 a8 06 40 00          mov    $0x4006a8,%eax
  400590:       f2 0f 10 45 f0          movsd  -0x10(%rbp),%xmm0
  400595:       48 89 c7                mov    %rax,%rdi
  400598:       b8 01 00 00 00          mov    $0x1,%eax
  40059d:       e8 16 fe ff ff          callq  4003b8 <printf@plt>
  4005a2:       b8 00 00 00 00          mov    $0x0,%eax
  4005a7:       c9                      leaveq
```

GNUPlot

http://www.gnuplot.info/
http://www.gnuplot.info/documentation.html

(log in to **cs6472** or any other machine that has gnuplot, and run **gnuplot**)

```
plot "data.txt" using 1:2 title 'Column 2', "data.txt" using 1:3 title 'Column 3'
```

```
plot "data.txt" using 1:2 title 'Column 2'
gnuplot> set term png                (will produce .png output)
gnuplot> set output "printme.png" (output to any filename you use)
gnuplot> replot
```