Lab Manual

Vishal Rao/Kanishka Lahiri (AMD) and K V Subramaniam (PES)

The objective of this lab is to get familiar with some of the performance measurement tools and then use them to debug performance. We will be using the following tools:

- `htop` - To identify the various threads of a process and the cores on which they run. (apt-get install htop)
- `sar` - monitoring system activity like page faults. (apt-get install sysstat)
- `gprof` - Using a profiler to identify the hotspots in the program
- `perf` - To identify the various architectural parameters such as cache misses. To install:
    - apt-get install linux-tools-common linux-tools-generic linux-tools-`uname -r` (If that fails, apt-get install linux-tools-4.15.0-118-generic)

System Information

```
# lscpu
```

This command lists all the information about the CPU (ex. Threads per Core, Cache Sizes)

```
Architecture:            x86_64
CPU op-mode(s):          32-bit, 64-bit
Byte Order:              Little Endian
CPU(s):                  8
On-line CPU(s) list:     0-7
Thread(s) per core:      2
Core(s) per socket:      4
Socket(s):               1
NUMA node(s):            1
Vendor ID:               GenuineIntel
CPU family:              6
Model:                   94
Model name:              Intel(R) Core(TM) i7-6700HQ CPU @ 2.60GHz
Stepping:                3
CPU MHz:                 1225.916
CPU max MHz:             3500.0000
CPU min MHz:             800.0000
BogoMIPS:                5199.98
Virtualization:          VT-x
L1d cache:               32K
L1i cache:               32K
L2 cache:                256K
L3 cache:                6144K
NUMA node0 CPU(s):       0-7
Flags:                   fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca c
mov pat pse36 clflush dts acpi mmx fxsr sse sse2 ss ht tm pbe syscall nx pdpe1gb
rdtscp lm constant_tsc art arch_perfmon pebs bts rep_good nopl xtopology nonstop_
tsc cpuid aperfmperf pni pclmulqdq dtes64 monitor ds_cpl vmx est tm2 ssse3 sdbg f
ma cx16 xtpr pdcm pcid sse4_1 sse4_2 x2apic movbe popcnt tsc_deadline_timer aes x
save avx f16c rdrand lahf_lm abm 3dnowprefetch cpuid_fault epb invpcid_single pti
 ssbd ibrs ibpb stibp tpr_shadow vnmi flexpriority ept vpid fsgsbase tsc_adjust b
mi1 hle avx2 smep bmi2 erms invpcid rtm mpx rdseed adx smap clflushopt intel_pt x
saveopt xsavec xgetbv1 xsaves dtherm ida arat pln pts hwp hwp_notify hwp_act_wind
ow hwp_epp md_clear flush_l1d
```

```
# cat /sys/devices/system/cpu/cpu*/topology/thread_siblings_list | sort -u
```

```
0,4
1,5
2,6
3,7
```

This command uses the /sys filesystem to determine the cores and their hyperthreads. This output is interpreted as - Every core can run 2 threads, Core 0 is the physical core and its hyperthread is Core 4.

Program 1: Multithreaded PI

This multithreaded program calculates the value of PI.

```
$ gcc -g -lm -pthread -o MT_Pi MT_Pi.c
$ ./MT_Pi <Thread #> [Accuracy]
```

Configurable Parameters -

1. Number of Threads.
2. Accuracy [Default : 1000000000]

Step 1: Observe Multithreaded utilization using `'htop'`.

```
$ ./MT_Pi 4 &
$ PID=$!
$ htop -p $PID
```

Step 2: Observe paging stats using the `'sar'` utility.

```
taskset -c 0-3 ./MT_Pi 4 2> /dev/null & sar -P 0,1,2,3 -B 1 15
```

//We pin the programs the CPUs 0-3 and use `'sar'` to measure paging stats. If your machine has fewer cores, pin them to a fewer number of cores. Also, now observe how htop shows that your pi program threads are only pinned to fewer cores.

```
Linux 4.15.0-128-generic (vishal-Inspiron-7559)         Wednesday 23 December 2020      _x86_64_        (8 CPU)

02:59:37 IST  pgpgin/s pgpgout/s   fault/s  majflt/s  pgfree/s pgscank/s pgscand/s pgsteal/s    %vmeff
02:59:38  IST      0.00    184.00    108.00      0.00   4098.00      0.00      0.00      0.00      0.00
02:59:39  IST      0.00     20.00     47.00      0.00    201.00      0.00      0.00      0.00      0.00
02:59:40  IST      0.00      0.00     64.00      0.00   4929.00      0.00      0.00      0.00      0.00
02:59:41  IST      0.00    156.00    538.00      0.00     71.00      0.00      0.00      0.00      0.00
02:59:42  IST      0.00    148.00   1858.00      0.00   1293.00      0.00      0.00      0.00      0.00
02:59:43  IST      0.00      0.00     45.00      0.00    285.00      0.00      0.00      0.00      0.00
```

Step 3: The /proc filesystem contains all the information about the process. For example, if we wanted to see the currently mapped memory regions relevant to the process -

```
$ ./MT_Pi 4 &
$ PID=$!
$ cat /proc/$PID/maps (Observe the dyn-linked libraries pthread lib)
```

```
00400000-00401000 r-xp 00000000 08:07 20451257                           /home/vishal/Desktop/CCBD/PerfTools/MT_Pi
00600000-00601000 r--p 00000000 08:07 20451257                           /home/vishal/Desktop/CCBD/PerfTools/MT_Pi
00601000-00602000 rw-p 00001000 08:07 20451257                           /home/vishal/Desktop/CCBD/PerfTools/MT_Pi
01c5d000-01c7e000 rw-p 00000000 00:00 0                                  [heap]
7f8cbddf6000-7f8cbddf7000 ---p 00000000 00:00 0
7f8cbddf7000-7f8cbe5f7000 rw-p 00000000 00:00 0
7f8cbe5f7000-7f8cbe5f8000 ---p 00000000 00:00 0
7f8cbe5f8000-7f8cbedf8000 rw-p 00000000 00:00 0
7f8cbedf8000-7f8cbedf9000 ---p 00000000 00:00 0
7f8cbedf9000-7f8cbf5f9000 rw-p 00000000 00:00 0
7f8cbf5f9000-7f8cbf5fa000 ---p 00000000 00:00 0
7f8cbf5fa000-7f8cbfdfa000 rw-p 00000000 00:00 0
7f8cbfdfa000-7f8cbffba000 r-xp 00000000 08:07 4463491                    /lib/x86_64-linux-gnu/libc-2.23.so
7f8cbffba000-7f8cc01ba000 ---p 001c0000 08:07 4463491                    /lib/x86_64-linux-gnu/libc-2.23.so
7f8cc01ba000-7f8cc01be000 r--p 001c0000 08:07 4463491                    /lib/x86_64-linux-gnu/libc-2.23.so
7f8cc01be000-7f8cc01c0000 rw-p 001c4000 08:07 4463491                    /lib/x86_64-linux-gnu/libc-2.23.so
7f8cc01c0000-7f8cc01c4000 rw-p 00000000 00:00 0
7f8cc01c4000-7f8cc01dc000 r-xp 00000000 08:07 4463492                    /lib/x86_64-linux-gnu/libpthread-2.23.so
7f8cc01dc000-7f8cc03db000 ---p 00018000 08:07 4463492                    /lib/x86_64-linux-gnu/libpthread-2.23.so
7f8cc03db000-7f8cc03dc000 r--p 00017000 08:07 4463492                    /lib/x86_64-linux-gnu/libpthread-2.23.so
7f8cc03dc000-7f8cc03dd000 rw-p 00018000 08:07 4463492                    /lib/x86_64-linux-gnu/libpthread-2.23.so
7f8cc03dd000-7f8cc03e1000 rw-p 00000000 00:00 0
7f8cc03e1000-7f8cc0407000 r-xp 00000000 08:07 4463502                    /lib/x86_64-linux-gnu/ld-2.23.so
7f8cc05d2000-7f8cc05d6000 rw-p 00000000 00:00 0
7f8cc0606000-7f8cc0607000 r--p 00025000 08:07 4463502                    /lib/x86_64-linux-gnu/ld-2.23.so
7f8cc0607000-7f8cc0608000 rw-p 00026000 08:07 4463502                    /lib/x86_64-linux-gnu/ld-2.23.so
7f8cc0608000-7f8cc0609000 rw-p 00000000 00:00 0
7fff50e8c000-7fff50ead000 rw-p 00000000 00:00 0                          [stack]
7fff50ec2000-7fff50ec5000 r--p 00000000 00:00 0                          [vvar]
7fff50ec5000-7fff50ec7000 r-xp 00000000 00:00 0                          [vdso]
ffffffffff600000-ffffffffff601000 r-xp 00000000 00:00 0                  [vsyscall]
```

Program 2: Matrix Multiplication

This program calculates the product of two square matrices. Find the baseline execution time using the `'time'` utility.

```
gcc -g -o MatMul MatMul.c
time ./MatMul
```

We then compile and run it with the -pg flag to profile it -

```
gcc -pg -o MatMul MatMul.c; ./MatMul
```

A file named `'gmon.out'` should be created in the current directory

To obtain the profile - `gprof MatMul gmon.out | less`

```
Flat profile:

Each sample counts as 0.01 seconds.
  %   cumulative   self              self     total
 time   seconds   seconds    calls   s/call   s/call  name
100.63     53.86    53.86        1    53.86    53.86  multiplyMatrix
  0.02     53.87     0.01        1     0.01     0.01  populateMatrices
```

The flat profile shows that the Matrix Multiply function is consuming most of the time. Hence, to improve performance, we should ideally start by improving this function.

The perf tool in Linux helps us monitor various software and hardware events. To list all the events available on your system use `'perf list'`

To allow perf to collect these events -

```
echo -1 | sudo tee /proc/sys/kernel/perf_event_paranoid
echo 0 | sudo tee /proc/sys/kernel/kptr_restrict
```

To measure these events for a program -

```
Recompile : gcc -g -o MatMul MatMul.c
perf stat -e <comma separated list of events> ./MatMul
```

Find the cache misses -

```
perf stat -e cache-misses,L1-dcache-load-misses ./MatMul
```

How do you modify the Matrix Multiply function to reduce the time and the cache misses?
Hint: Matrices in C are stored in row-major order
(All columns of a row in sequential addresses followed by the next row).

After you are done using perf, restore defaults -

```
echo 1 | sudo tee /proc/sys/kernel/perf_event_paranoid
echo 1 | sudo tee /proc/sys/kernel/kptr_restrict
```