Indian Institute of Information Technology, Vadodara

Parallel Programming(CS403)

Prof. Reshmi Mitra

Name - Dilip Puri
ID - 201351014
Collaborator - Hemant Kumar(201352026)

Lab 01

Submission Date - August 13, 2016 **Deadline** - Due Aug 15, 4:00 PM

Peak Performance

Problem 1 Consider a memory system with a level 1 cache of 32 KB and DRAM of 512 MB with the processor operating at 1 GHz. The latency to L1 cache is 1 cycle and the latency to DRAM is 100 cycles. In each memory cycle, the processor fetches four words (cache line size is 4 words). What is the peak achievable performance of a dot product of two vectors?

```
/* dot product loop */
for (i = 0; i < dim; i++)
   dot_prod += a[i] * b[i];
```

Answer:

```
/* dot product loop */
for (i = 0; i < dim; i++)
   dot_prod += a[i] * b[i];
```

Problem 2 Now consider the problem of multiplying a dense matrix with a vector using a two-loop dot-product formulation. The matrix is of dimension 4K x 4K. (Each row of the matrix takes 16 KB of storage.) What is the peak achievable performance of this technique using a two-loop dot-product based matrix-vector product?

```
/* matrix-vector product loop */
for (i = 0; i < dim; i++)
   for (j = 0; j < dim; j++)
      c[i] += a[i][j] * b[i];
```

Answer:

```
/* dot product loop */

adf
adf
```

Linux commands

```
$top
```

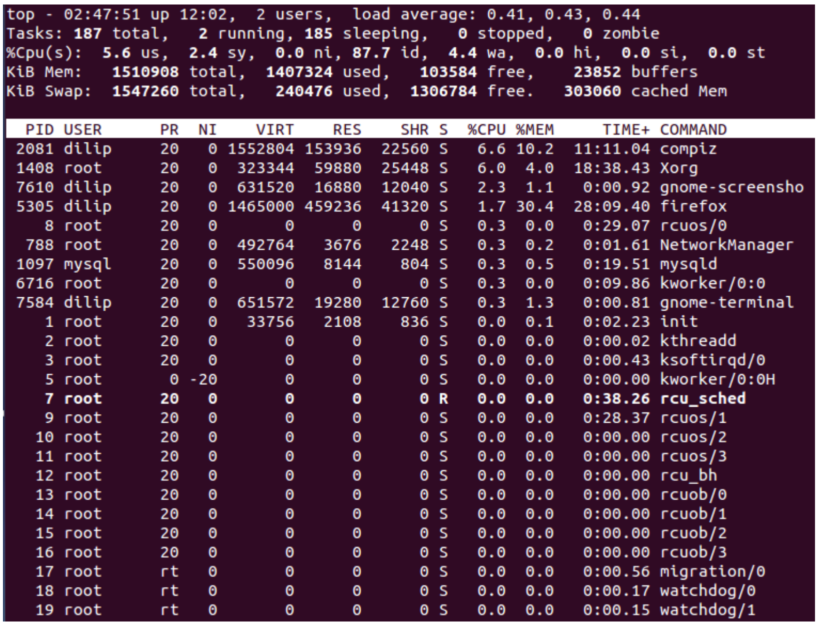

Figure 1: top: provides dynamic real-time view of individual jobs running on the system

```
$gnome-system-monitor
```

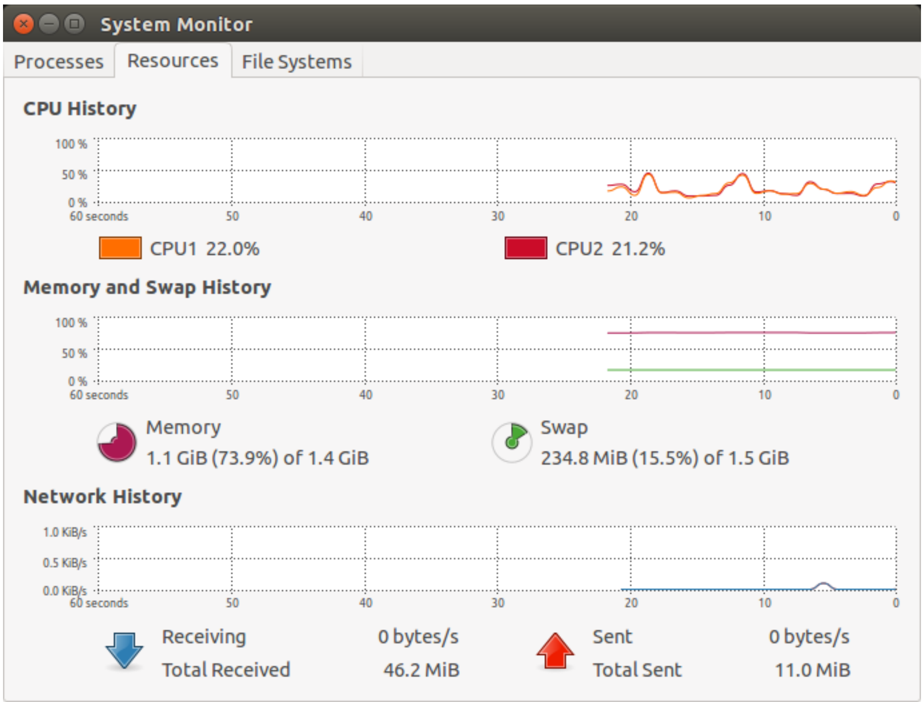

Figure 2: gnome-system-monitor: shows which programs are running and how much processor time, memory, and disk space are being used. This gives an overall system view whereas the "top" instruction represents a detailed perspective

```
$lscpu
We can also get the same information from $cat /proc/cpuinfo
```

```
==================================
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  1
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           AuthenticAMD
CPU family:          18
Model:               1
Stepping:            0
CPU MHz:             800.000
BogoMIPS:            4392.08
Virtualization:      AMD-V
L1d cache:           64K
L1i cache:           64K
L2 cache:            1024K
NUMA node0 CPU(s):   0,1
dilip@dilip-notebook-pc:~$ lscpu
Architecture:        x86_64
CPU op-mode(s):      32-bit, 64-bit
Byte Order:          Little Endian
CPU(s):              2
On-line CPU(s) list: 0,1
Thread(s) per core:  1
Core(s) per socket:  2
Socket(s):           1
NUMA node(s):        1
Vendor ID:           AuthenticAMD
CPU family:          18
Model:               1
Stepping:            0
CPU MHz:             800.000
BogoMIPS:            4392.08
Virtualization:      AMD-V
L1d cache:           64K
L1i cache:           64K
L2 cache:            1024K
NUMA node0 CPU(s):   0,1
```

```
$time ./a.out
```

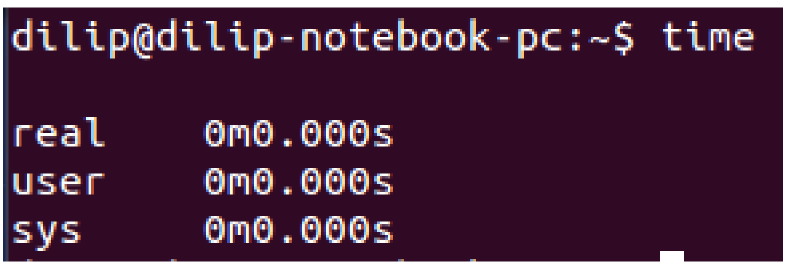

Figure 3: time: Get total program execution time in the shell

Lab Problems

1. Familiarize yourself with the Linux commands and POSIX thread code given in this handout.

2. Solutions for Problems 1-2 on peak performance.

3. Using the basic Linux commands find the cache size, bandwidth number of processors on your system.

4. Write a C-code using POSIX threads to create an unbalanced load using sleep command and hello.c. The sample sleep times are given below:

   (a) thread-1: 1000 sec, thread-2: 5000 sec, thread-3: 20 sec, thread-4: 1200 sec.

   (b) Measure the total time taken for the complete execution of code with and without the additional sleep command.
   Hint: You will require to include unistd.h for successful compilation.

```
hemant@hemant:~/Desktop/sem7/pp/lab$ gcc -pthread -o hello hello.c
hemant@hemant:~/Desktop/sem7/pp/lab$ time ./hello
In main: creating thread 0
In main: creating thread 1
In main: creating thread 2
In main: creating thread 3
Hello World! It's me, thread #2!
In main: creating thread 4
Hello World! It's me, thread #4!
Hello World! It's me, thread #0!
Hello World! It's me, thread #3!
Hello World! It's me, thread #1!

real    0m50.002s
user    0m0.000s
sys     0m0.003s
```

Figure 4: time: Get total program execution time in the shell

5. Write a C-code using POSIX threads about matrix multiplication.

   (a) Take overall execution time measurement using time command for different application size and thread count for the serial and parallel code.

   (b) Observe gnome-system-monitor output as your fire up different thread counts.

   (c) Use the overall execution time measurements to plot and comment upon the speed-up.