8-9 Performance Analysis

Lecture 8-9 Performance Analysis

Standard Definition of Performance

For some program running on machine X,

$\text{Performance}_X = 1/\text{Execution time}_X$

"X is n times faster than Y"

$\text{Performance}_X / \text{Performance}_Y = n$

Speedup

Number of cores = p

Serial run-time = T_{serial}

Parallel run-time = T_{parallel}

$S = \frac{T_{\text{serial}}}{T_{\text{parallel}}}$

Sources of Parallel Overheads

Overhead of creating threads/processes

Synchronization

Load imbalance

Communication

Extra computation

Memory access (for both sequential and parallel!)

Efficiency of a parallel program

$E = \frac{\text{Speedup}}{p} = \frac{\frac{T_{\text{serial}}}{T_{\text{parallel}}}}{p} \ (\leq 1)$

P = can be number of threads, processes, or core

Scalability

Scalability is the ability of a system to handle a growing amount of work efficiently.

If we keep the efficiency fixed by increasing the number of processes/threads and without increasing problem size, the problem is strongly scalable.

If we keep the efficiency fixed by increasing the problem size at the same rate as we increase the number of processes/threads, the problem is weakly scalable.

Execution Time

Elapsed Time (aka wall-clock time)

- counts everything (disk and memory accesses, I/O , etc.)
- a useful number, but often not good for comparison purposes

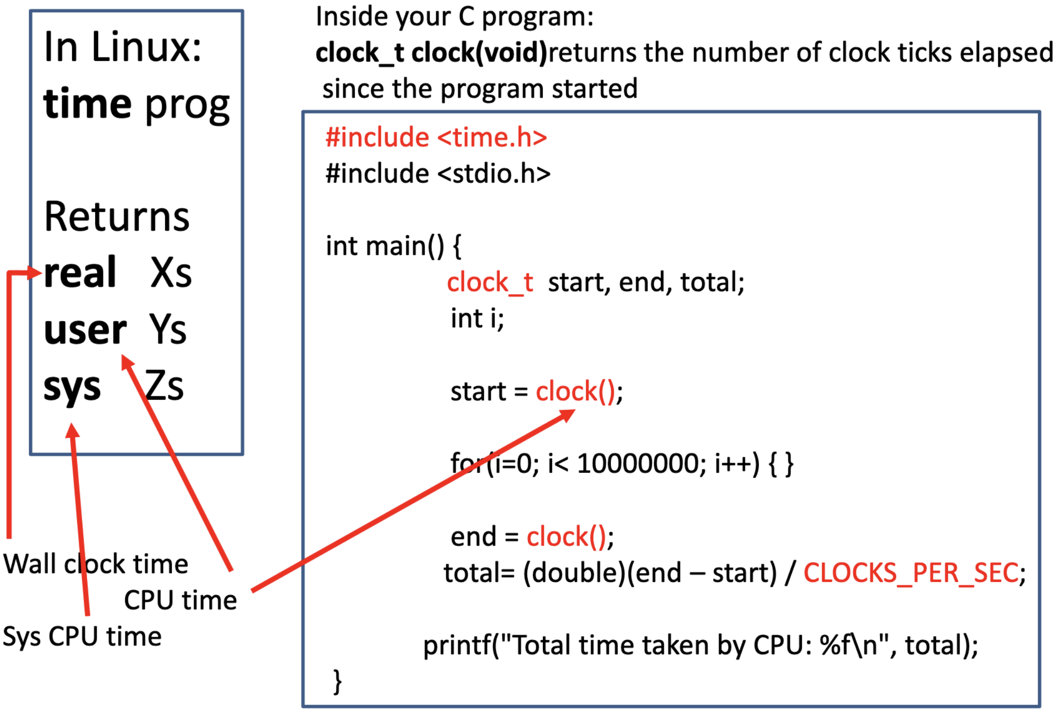
CPU time

- doesn't count I/O or time spent running other programs
- can be broken up into system time, and user time

Our focus: user CPU time

- time spent executing the lines of code that are “in” our program

Taking Timings



In Linux:
time prog

Returns
real Xs
user Ys
sys Zs

Wall clock time
CPU time
Sys CPU time

Inside your C program:
clock_t clock(void)returns the number of clock ticks elapsed since the program started

```
#include <time.h>
#include <stdio.h>

int main() {
            clock_t  start, end, total;
            int i;

            start = clock();

            for(i=0; i< 10000000; i++) { }

            end = clock();
           total= (double)(end – start) / CLOCKS_PER_SEC;

         printf("Total time taken by CPU: %f\n", total);
 }
```

Simple Metrics

Response time (aka Execution Time)

- The time between the start and completion of a task
- e.g How long did the simulation take to finish?

Throughput

- Total amount of work done in a given time
- e.g. How many threads finished in the last two minutes?

Execution time for sequential program:

ET = IC * CPI * CT = total_cycles * CT = total_cycles / clock_rate

ET = Execution Time, IC = Instruction Count, CPI = Cycles Per Instruction, CT = Cycle Time

MIPS = Million Instructions per Second = IC in millions / ET in seconds

Pitfalls in timing in Parallel Machines

The total number of instructions executed may be different across different runs! (This effect increases with the number of cores.)

System-level code account for a significant fraction of the total execution time.

How to check the performance of a parallel machine?

Performance best determined by running a real application

Example of benchmarks: PARSEC (parallel), Rodinia (parallel), SPEC (sequential)

Conclusions

Execution time is what matters: system time, CPU time, I/O and memory time d

Scalability and efficiency measure the quality of your code.