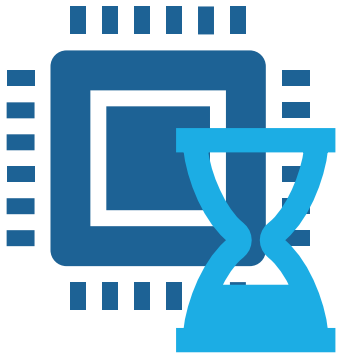
CS3210

Parallel Computing

Changes from Monday in Green



Lab 2

Mon (4pm)
Tues (2pm)

Admin
Updates

- Assignment 1 Part 1 due 11am next Monday
  - Late penalty: 10% per day, up to a week
  - Consider updating your program, measurements and report after this lab
  - See FAQ for questions about compiler choice and flags
  - New installed packages: **`clang`**
  - Request new packages/use of non-standard libraries: drop teaching team a message

- Lab 1 comments to release sometime later this week
  - Will post announcement; look in LumiNUS gradebook

Admin

Lab Machine Info

- Hostnames are: **`soctf-pdc-<node ID>`**
  - Available: 24/7, at least until end of Assignment 1

Node IDs	001 - 008	009 - 016	018 - 019	020 - 021
CPU	Xeon Silver 4114	Intel Core i7-7700	Xeon Silver 4114	Intel Core i9-9700
# sockets	1	1	2	1
# cores/skt	10	4	10	8
SMT?	Yes	Yes	Yes	No
RAM (GB)	32	32	64 (NUMA)	32
SWAP (GB)	2	2	~9	~14

Admin
Roadmap

- Please use only the machines assigned to your lab pair
  - Rotate between the machines with your lab partner
- Check if someone else is on the same machine!
  - Outside of this: free-for-all, but please be considerate
- Today's lab
  - Part 1: Introduction to OpenMP programming
  - Part 2: Introduction to Performance Instrumentation

Admin

Lab 2 Submission Instructions

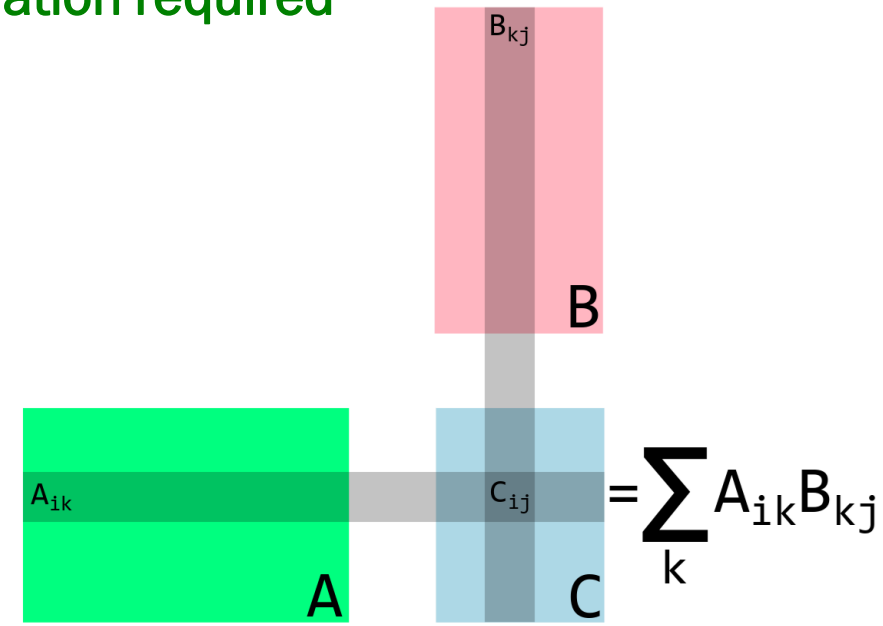
- Due next Tuesday (15 Sep), 11.59pm
  - Each student submits their own PDF writeup (indep. work)
  - Name it A0123456Z.pdf
  - No need to include your programs in writeup

- Contents
  - Your responses to ex4 to ex6 (max 1 paragraph each in addition to any calculations performed)
  - Raw measurements for ex6 with hostnames
  - Graph of results with justifications for observations

Part 1

Matrix Multiplication (1)

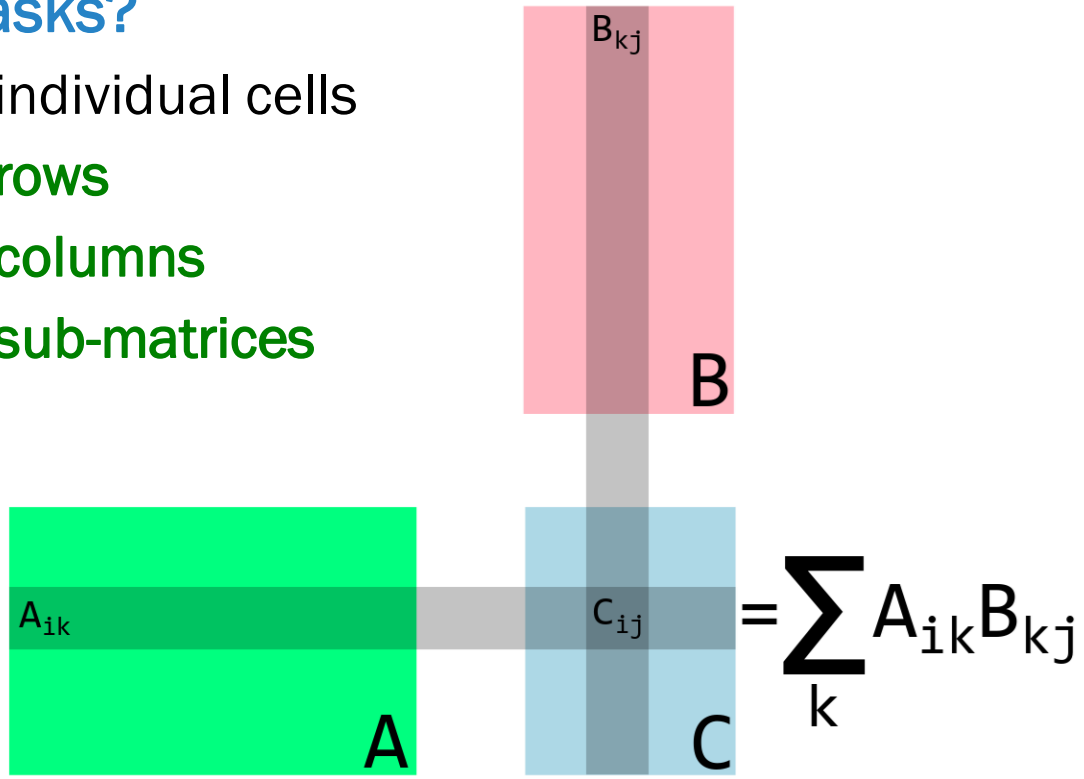
- Classical example of an embarrassingly parallel problem
  - Trivial to parallelise - what kind of parallelism (task or data)?
  - Why is it trivial to parallelise? No synchronization or coordination required



Part 1

Matrix Multiplication (2)

- In what ways can we decompose the computation of C into tasks?
  - By individual cells
  - By rows
  - By columns
  - By sub-matrices

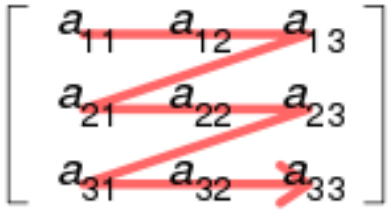


Part 1

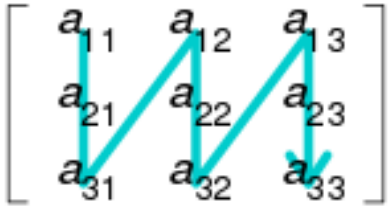
Matrix Multiplication (3)

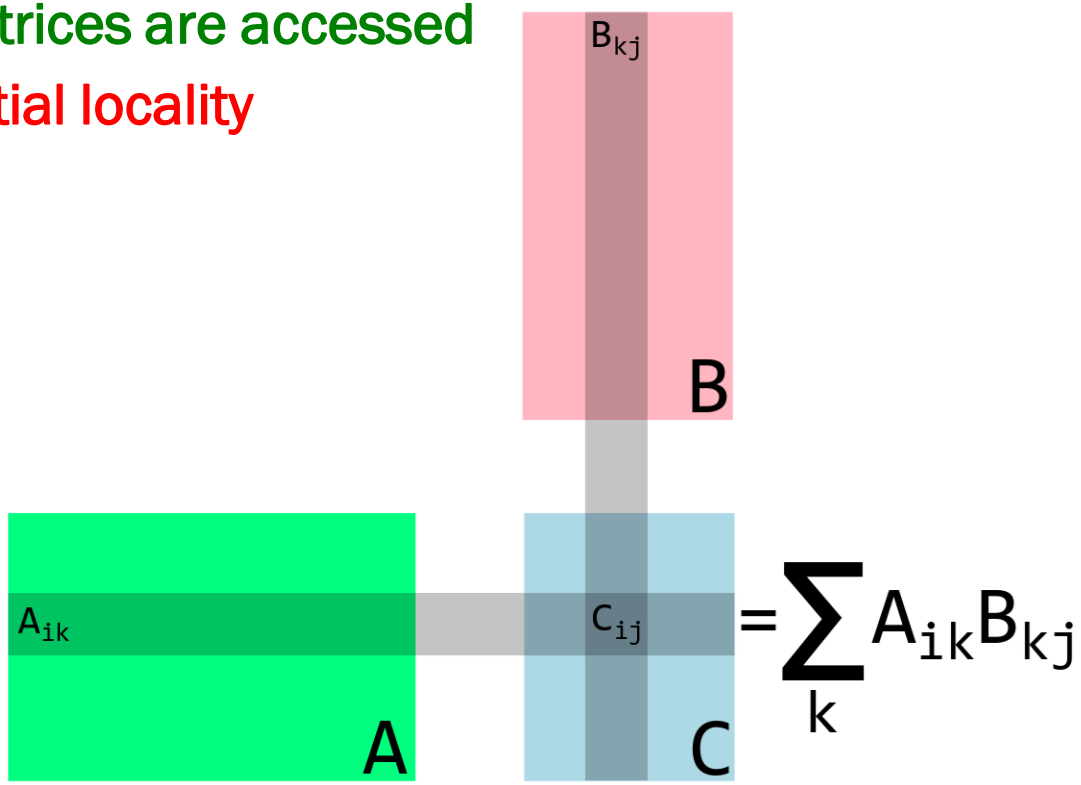
- How can we layout the matrices in memory?
  - Different: how matrices are accessed
  - Key principle: spatial locality

Row-major order



Column-major order





Part 1

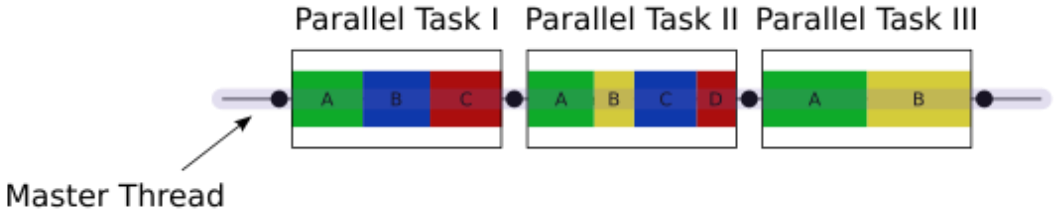
OpenMP (1)

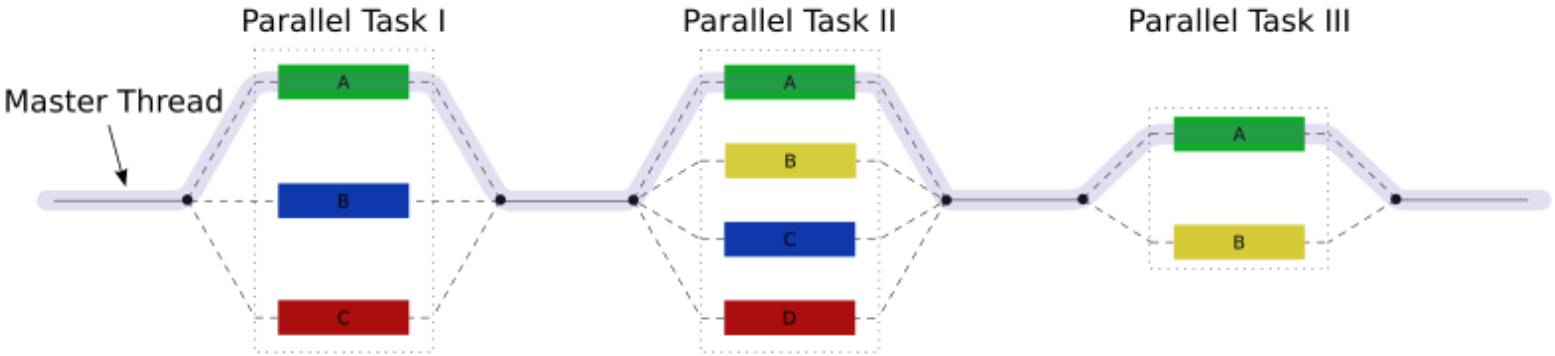
- OpenMP: Open Multi-Processing
  - Set of compiler directives and library routines to specify high-level parallelism in C, C++ and Fortran
  - Shared-memory programming with threads
  - Supports both task and data parallelism
  - C header: **`#include <omp.h>`**
  - Compilation flag: **`-fopenmp`**

- Directive structure
  - **`#pragma omp <directive> [<clause>...] [{ block }]`**

Part 1
OpenMP (2)

- Structure of OpenMP programs
  - Interleaved sequential sections and parallel regions
  - Each parallel region: one master and multiple worker threads





Part 1

OpenMP Directives (1)

- Parallel region directive: **`parallel`**
  - Demarcates a parallel region of code
  - Current thread becomes master thread (with thread ID **`0`**) at start of this block and forks additional worker threads
  - Default number of worker threads forked is 1 less than the number of logical cores on that machine
  - Can change this with **`omp_set_num_threads(int)`** or by setting the environment variable **`OMP_NUM_THREADS`**

```
#pragma omp parallel
{
    structured block
}
```

Part 1

OpenMP Directives (2)

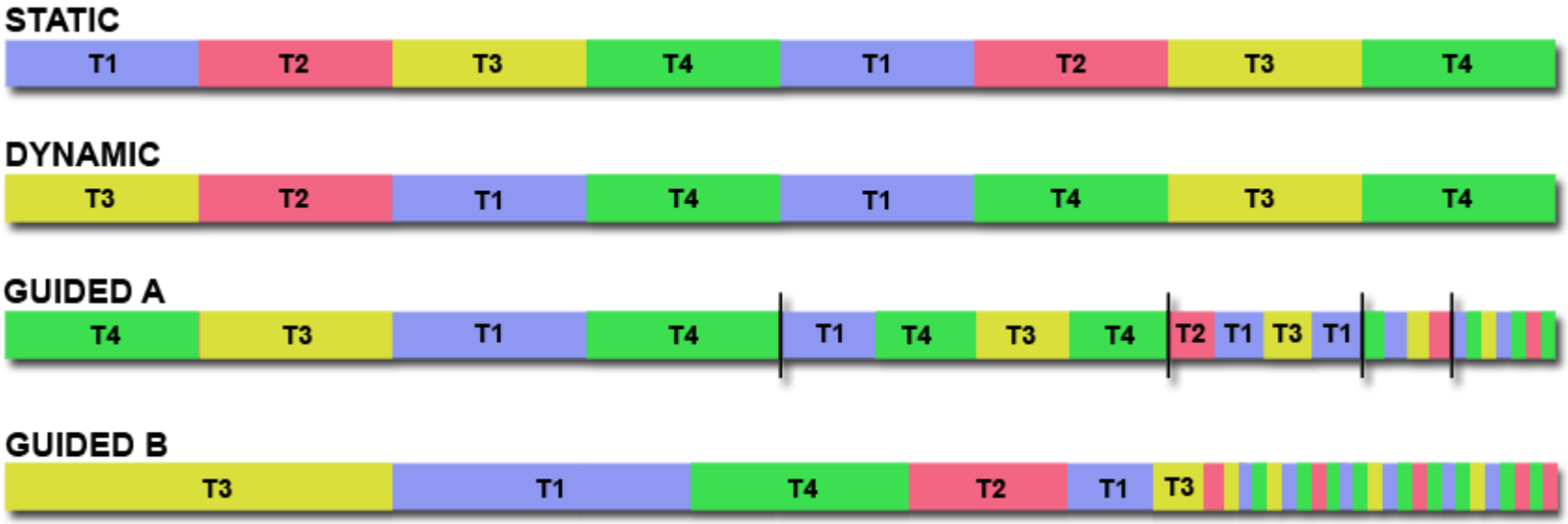
- Loop distribution directive: **`for`**
  - Can only be specified inside a parallel region
  - Can be combined as **`#pragma omp parallel for`**
  - Divides iterations of a `for` loop amongst existing threads
  - Additional clauses allow you to specify order and number of iterations assigned to each thread, e.g. **`schedule`** clause

```
#pragma omp parallel
{
    #pragma omp for [clause...]
    for (...; ...; ...) {
        loop body
    }
}
```

Part 1

OpenMP Directives (3)

- Loop distribution directive: **`for`**
  - Four types of scheduling: **static**, **dynamic**, **guided**, **runtime**
  - If the **`schedule`** clause is not specified - default is implementation-defined



Part 1
OpenMP Directives (4)

- Loop distribution directive: **`for`**
  - What is the expected output of this section of OpenMP code compiled with GCC and run on a 4-core machine?

```
#pragma omp parallel for
for (int i = 0; i < 12; i++) {
    printf("Thread %d - iteration %d", omp_get_thread_num(), i);
}
```

  - Should see: each thread assigned a contiguous "chunk" of 3 loop iterations
  - Are the messages from the same thread printed in order? Yes
  - Are the messages between threads printed in order? No

Part 1

OpenMP Directives (5)

- Task distribution directive: **`sections`**
  - Can only be specified inside a parallel region
  - Specify code sections with **`#pragma omp section`**; each a task that will be assigned to available threads, one at a time

```
#pragma omp parallel {
    #pragma omp sections {
        #pragma omp section {
            structured block
        }

        #pragma omp section {
            structured block
        }
    }
}
```

Part 1

OpenMP Directives (6)

- Barrier synchronization directive: **`barrier`**
  - Some directives (e.g. **`for`**) have an implied barrier at the end
  - Blocks threads that arrive early until all threads reach the barrier

```
#pragma omp barrier
```

- Atomic operation directive: **`atomic`**
  - Specifies a memory location must be updated atomically

```
#pragma omp atomic

    statement expression
```

Part 1

OpenMP Directives (7)

- Master section directive: **`master`**
  - Specifies a region that must only be executed by the master thread

```
#pragma omp master
{
    structured block
}
```

- Critical section directive: **`critical`**
  - Specifies a critical section

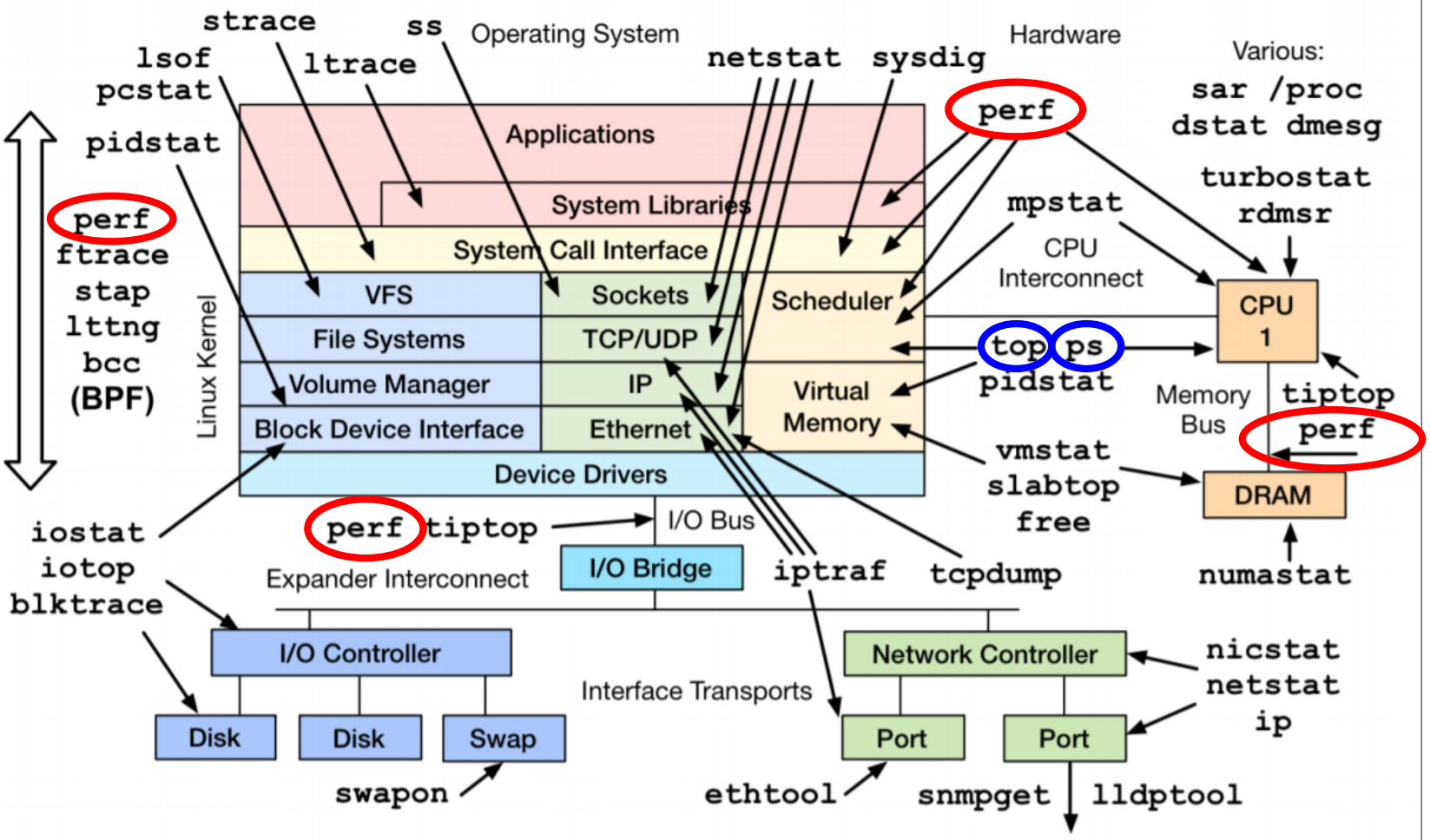
```
#pragma omp critical
{
    structured block
}
```

Part 2

`perf` Utility

- The **`perf`** utility is one of the most popular performance profiling tools on Linux
  - User space utility; not available by default
  - Ubuntu: install **`linux-tools-common`** package with the **`apt`** package manager with **`sudo apt install <package>`**
    - You may be prompted to install other required packages - do so
  - Profiles entire system during execution of a shell command
  - Part of a kernel-based subsystem that exposes performance counters in hardware and software and tracepoints (e.g. for interrupts, scheduling, etc.)
  - Command: **`perf stat <command> <args>…`**

Linux Performance Observability Tools



Utilities introduced in previous labs/tuts

New utilities

Part 2

`perf` Utility Output

- Sample output: **`perf stat <program> <args>…`**

Observe the number of context-switches

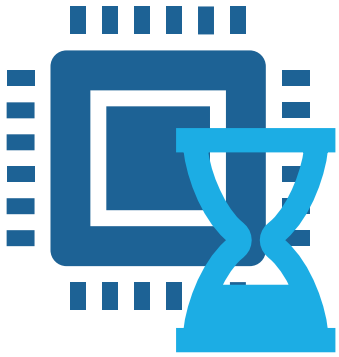
Do you think this is real, kernel or user time?

```
ubuntu-16-04@ubuntu1604-Aspire-4752:~/Desktop$ sudo perf stat ./beh 100000000
[5681]: Step 0
[5682]: Step 0
[5681]: Step 1
[5682]: Step 1
[5681]: Step 2
[5682]: Step 2
[5681]: Step 3
[5682]: Step 3
[5681]: Step 4
[5682]: Step 4
[5682] Child Done!
[5681] Parent Done!

 Performance counter stats for './beh 100000000':

      3238.831006      task-clock (msec)         #    1.994 CPUs utilized
                6      context-switches          #    0.002 K/sec
                0      cpu-migrations            #    0.000 K/sec
               87      page-faults               #    0.027 K/sec
    7,077,783,766      cycles                    #    2.185 GHz                      (83.21%)
    6,071,779,673      stalled-cycles-frontend   #   85.79% frontend cycles idle     (83.31%)
       17,203,110      stalled-cycles-backend    #    0.24% backend cycles idle      (66.76%)
    4,004,748,213      instructions              #    0.57  insn per cycle
                                                 #    1.52  stalled cycles per insn  (83.43%)
      999,786,370      branches                  #  308.687 M/sec                    (83.45%)
           26,545      branch-misses             #    0.00% of all branches          (83.28%)

      1.624490259 seconds time elapsed
```

CS3210

Parallel Computing

Thank you! Any questions?



Lab 2

Mon (4pm)
Tues (2pm)

bit.ly/cs3210-t04-qn