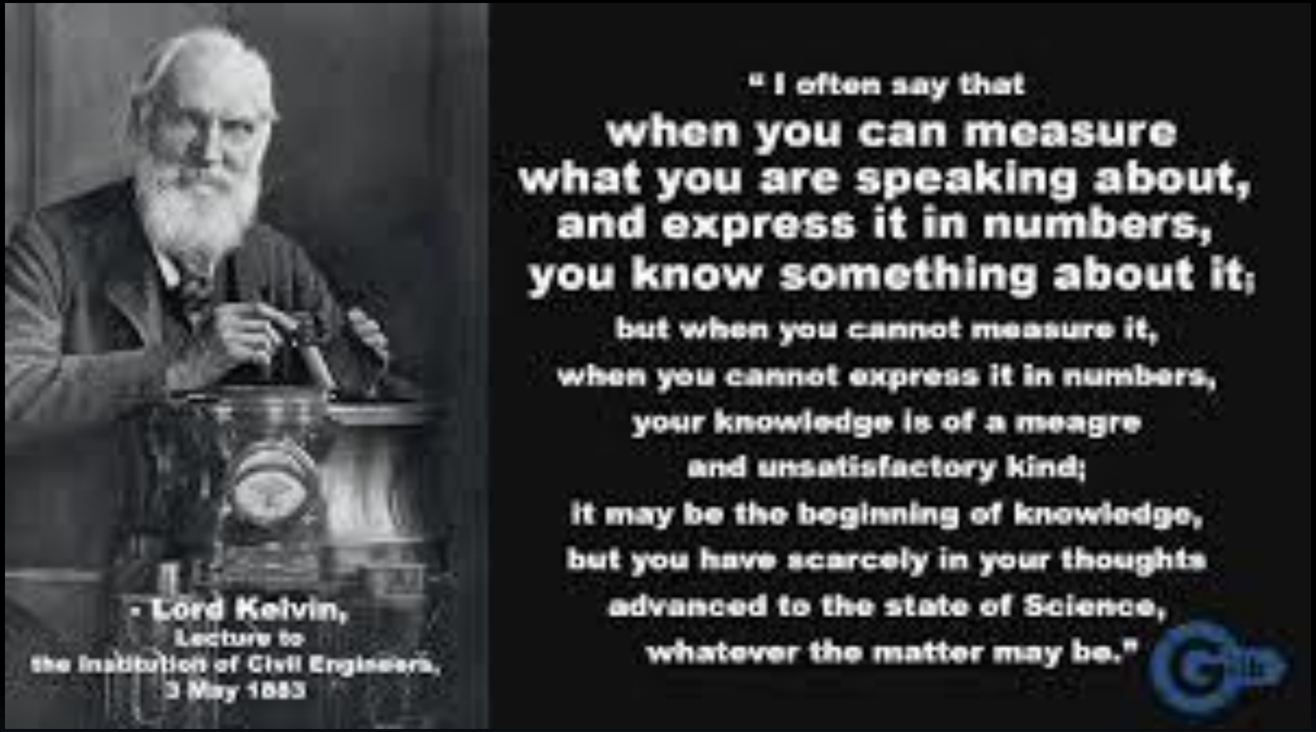
K V Subramaniam
Center for Cloud Computing and Big Data
PES University, Bangalore
subramaniamkv@pes.edu

PERFORMANCE TOOLS

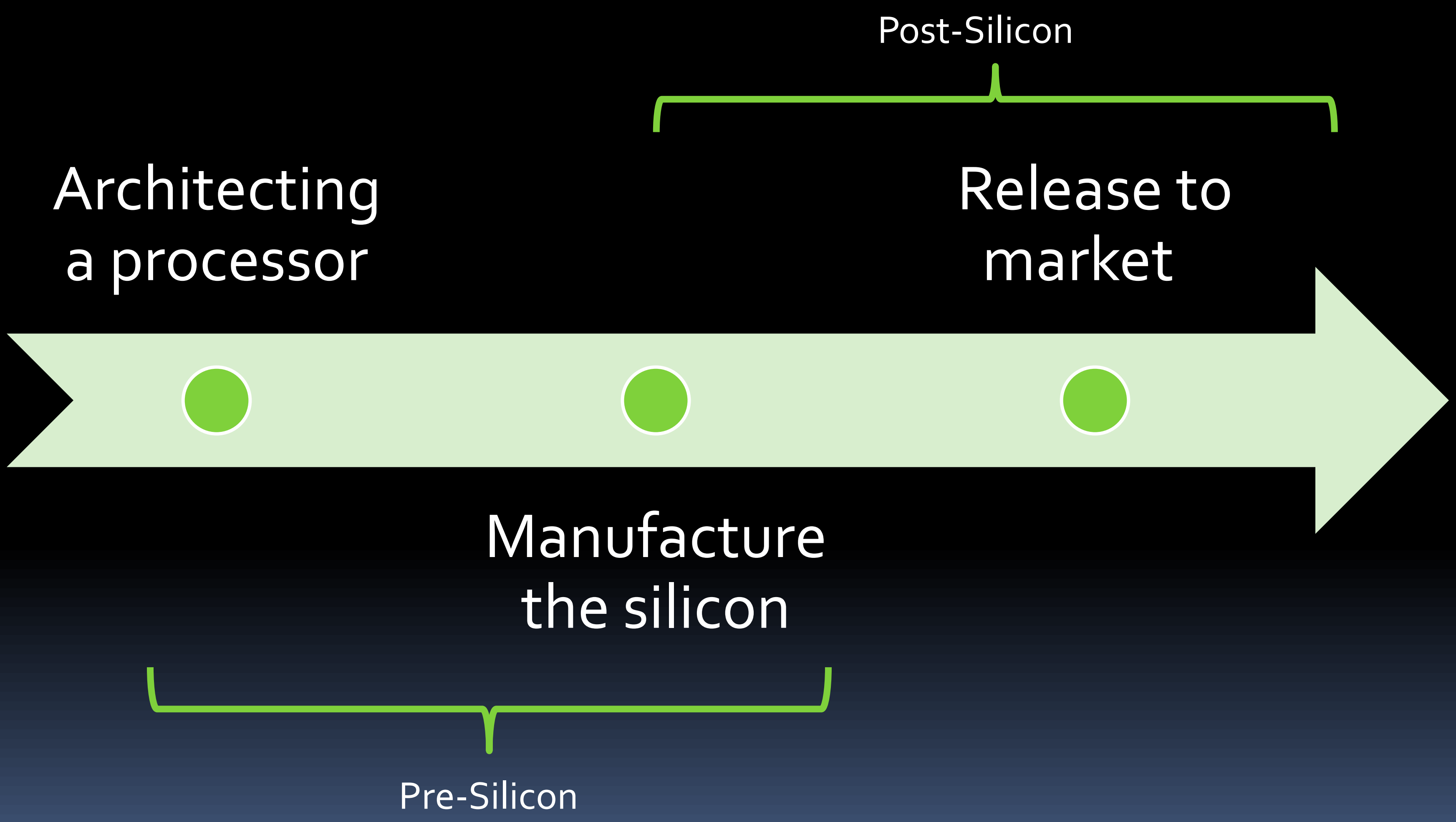
COMPUTER ARCHITECTURE WINTER SCHOOL - 2020


" I often say that
when you can measure
what you are speaking about,
and express it in numbers,
you know something about it;
but when you cannot measure it,
when you cannot express it in numbers,
your knowledge is of a meagre
and unsatisfactory kind;
it may be the beginning of knowledge,
but you have scarcely in your thoughts
advanced to the state of Science,
whatever the matter may be."
- Lord Kelvin,
Lecture to
the Institution of Civil Engineers,
3 May 1883

Overview

- Why performance?
- Performance metrics
- Simulators
- Profiling programs – CPU/Memory
- Measuring Operating system activity
- Measuring microarchitectural activity

Background


Post-Silicon
Architecting a processor
Release to market
Manufacture the silicon
Pre-Silicon

Performance: Goals

- Verify performance goals are met
- Debug performance issues in real world scenarios

Post-Silicon

Architecting a processor

Release market

Manufacture the silicon

Pre-Silicon

- Evaluate alternative designs
- Estimate performance – project how it will work.

Overview

- Why performance?
- Performance metrics
- Simulators
- Profiling programs – CPU/Memory
- Measuring Operating system activity
- Measuring microarchitectural activity

Performance Metrics

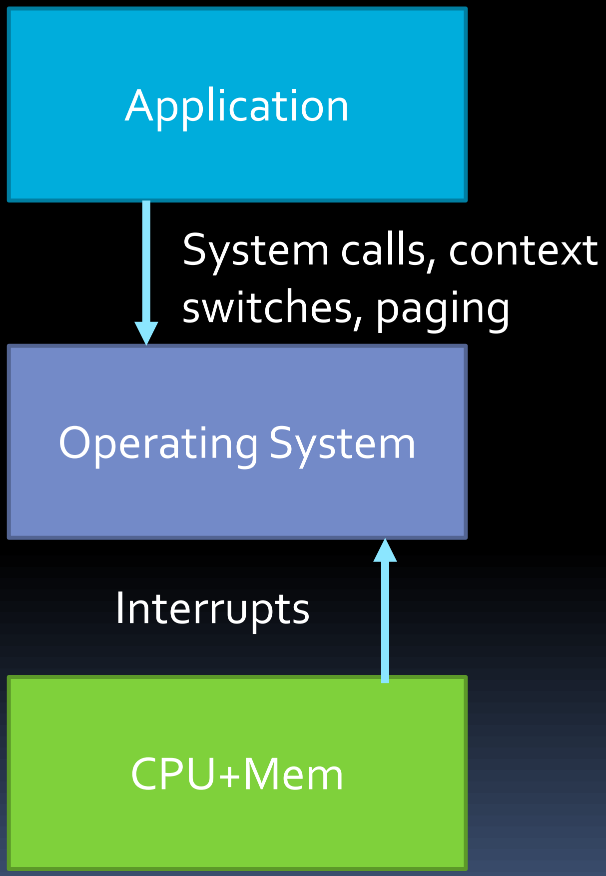
- How **fast** is the application running?
  - #clock cycles
  - Is it using the architecture efficiently?
    - Instructions per Cycle (IPC)
- Pre-silicon – how to check?
- What factors does application performance depend on?
- How do we know that it is using the architecture efficiently?

Application performance factors

- Algorithm used
  - From a theoretical standpoint this is important.
- For us more important
  - Where is time being spent?
  - How does it map to the hardware?

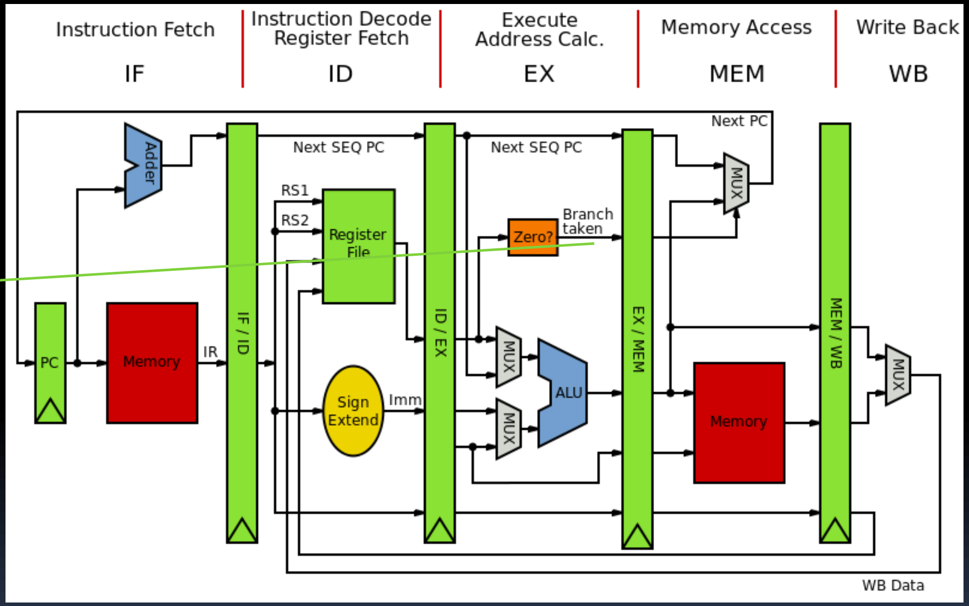
Software stack impact

- Compiler – maps applications to the hardware – multi-dimension arrays
- Runtime systems – OS, libraries, JVM, Garbage collectors
- Applications interact with OS
  - Voluntarily – when they need resources
  - Involuntarily – due to interrupts/context switches.
- Performance can depend on OS activity
  - Other applications..
  - Paging/TLB structures
- Becomes more complex with addition of
  - Virtual machines, containers.
  - And newer programming models – FaaS, Microservices.



Performance Metrics .. 2

- For a computer architect, knowing IPC itself is not enough
  - We need a deeper insight into how the various components are performing
    - e.g – Branch predictors, caches, prefetchers etc.
    - Ferdman et. al ASPLOS 12 – compared scale out workloads with SPEC



Overview

- Why performance?
- Performance metrics
- Simulators
- Profiling programs – CPU/Memory
- Measuring Operating system activity
- Measuring microarchitectural activity

Simulators - Pre-silicon

- Model the system being analyzed
  - Pipeline(s), caches, cores…
  - Both individually and combined together
- Use of simulators to
  - Check alternative designs
  - Interactions between various components
  - Performance projections on standard workloads

Types of simulators

- Functional Simulators
  - Check if the hardware design is indeed working correctly
  - ISA simulators
  - Check if the software stack will run correctly.

Add r1, r2[1000]

Adds content of memory to register r1

Needs to keep track of the state of the system

Types of simulators – Timing Performance simulators

- Timing/Performance simulators
  - How long does an (set of)instruction(s) take to execute?
  - Not so much interested in the results per se.
  - Accuracy – cycle accurate simulators

Add r1, r2[1000]

Compute total cycles..

- Check if location is DRAM/cache
- Move data to cache and then to register
- Perform the add
- Also keep track of how much time is spent in each activity

Overview

- Why performance?
- Performance metrics
- Simulators
- Profiling programs – CPU/Memory
- Measuring Operating system activity
- Measuring microarchitectural activity

Profiling programs

```
Func Add()
{

}

Func Multiply()
{

}

Func Main()
{
  call Add()
  call Multiply()
}
```

- Where is the time being spent?
- Techniques
  - Instrumentation
  - Sampling

Instrumentation based profilers

```
Func Add()
{

}

Func Multiply()
{

}

Func Main()
{
  call Add()
  call Multiply()
}
```

Add additional code
To each function

Log – entry exit points

```
Func Add()
{
  print(....)
}

Func Multiply()
{
  print(....)
}

Func Main()
{
  print(....)
  call Add()
  call Multiply()
}
```

Instrumentation based profilers

- Granularity of instrumentation
  - Function level – prof/gprof
    - High level identification of functions that take time
  - Basic block level
    - Unit of code with a single entry and exit
    - Additional instrumentation added to each basic block
    - detailed instruction execution counts.
    - Code paths

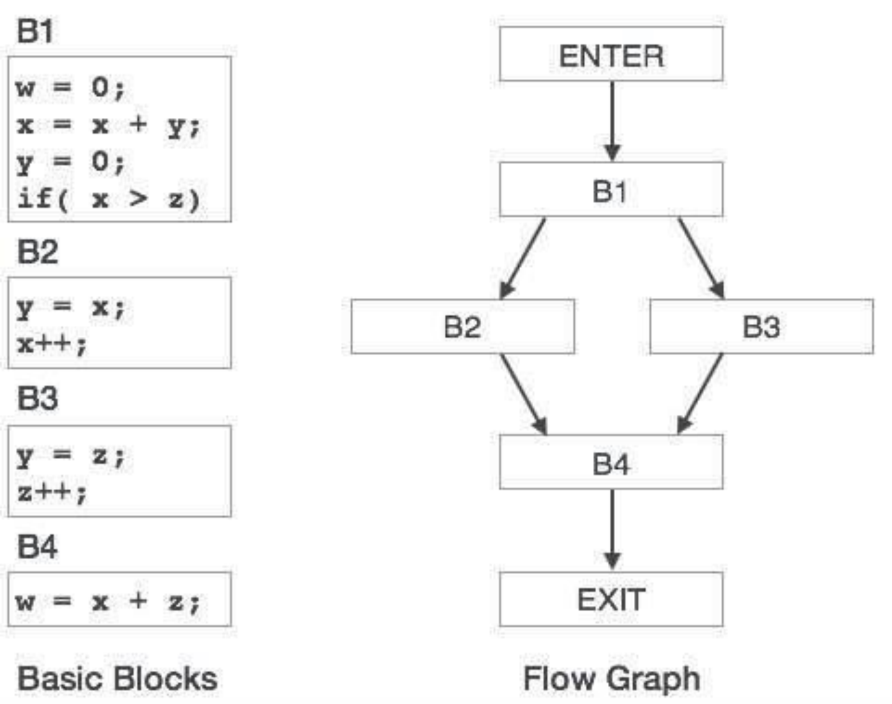
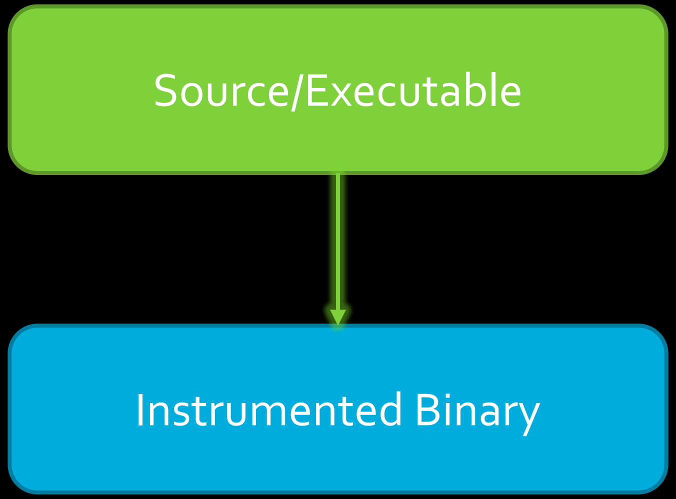


Image source: https://www.tutorialspoint.com/compiler_design/compiler_design_code_optimization.htm

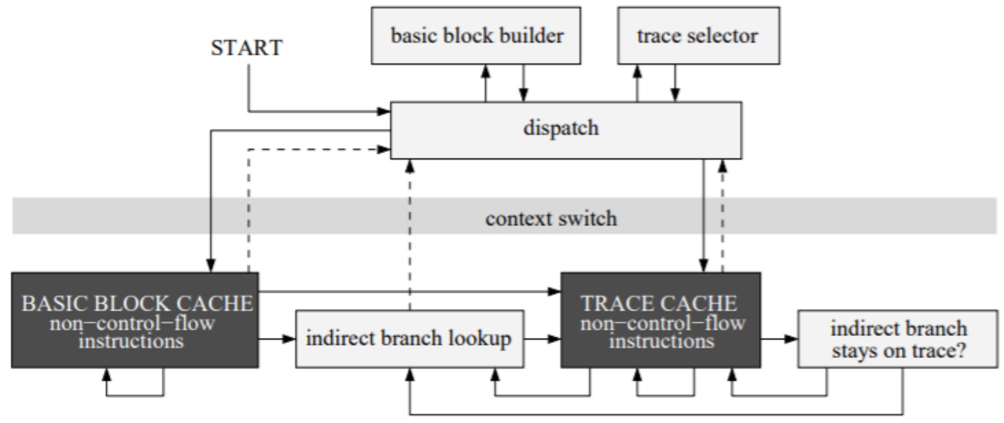
When to instrument?

- Static
  - compile time or prior to execution
    - Gprof: cc helloworld.c –pg
  - Produces an instrumented a.out.
  - Executing this produces run time profile data
  - Analyzed by post processing tools
    - Call graphs



When to instrument?

- Dynamic – at run time.
  - Examine instructions/basic blocks before they run
  - Add instrumentation code and translate to new address space.
  - Execute the translated binary
  - Cache translations
  - E.g: DynamoRIO
  - Valgrind
    - Memory leaks, data races



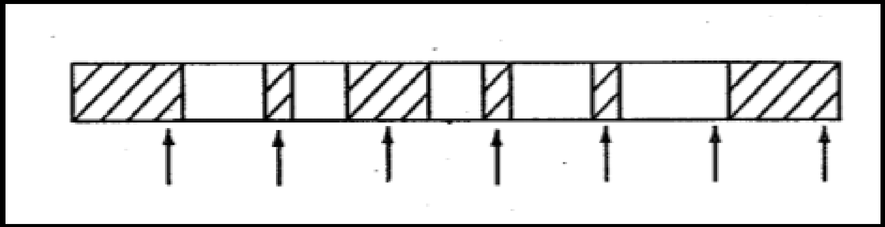
Source: Bruening et al 2003

Caveats: instrumentation based

- Addition of instrumentation code
  - Code bloat
  - Additional code executed dynamically
  - Not really good for measuring efficiency of hardware usage – cache misses?
- Dynamic instrumentation – heavy overheads during run time
  - Good for generating detailed instruction/data traces
  - These can be useful for input to simulators

Sampling based profilers

- Interrupt program at regular intervals
- Examine call stack
- Identify the function currently being executed.
- On completion,
  - Fraction of samples/total #samples → approximated as runtime spent in function.



Sampling based profilers..

- Sampling frequency affects accuracy.
- Short lived functions may not show up in sample
- OS activity – interrupts can affect accuracy
- Good for estimating overall breakup of time
  - Fast, less intrusive
- Not so good if we are looking for detailed breakup or causes of latency.

Overview

- Why performance?
- Performance metrics
- Simulators
- Profiling programs – CPU/Memory
- Measuring Operating system activity
- Measuring microarchitectural activity

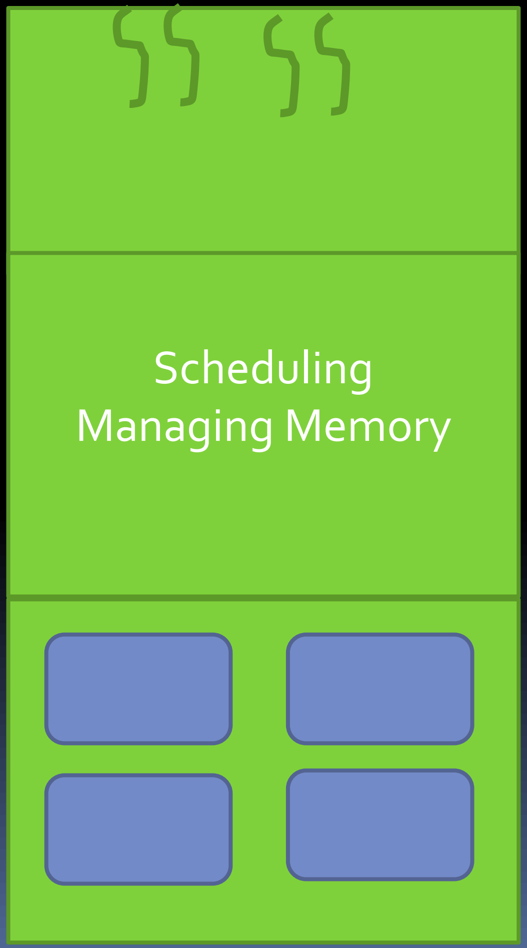
Where does OS impact

Applications are multi threaded
Have large memory footprint

OS – Manages mapping, resources

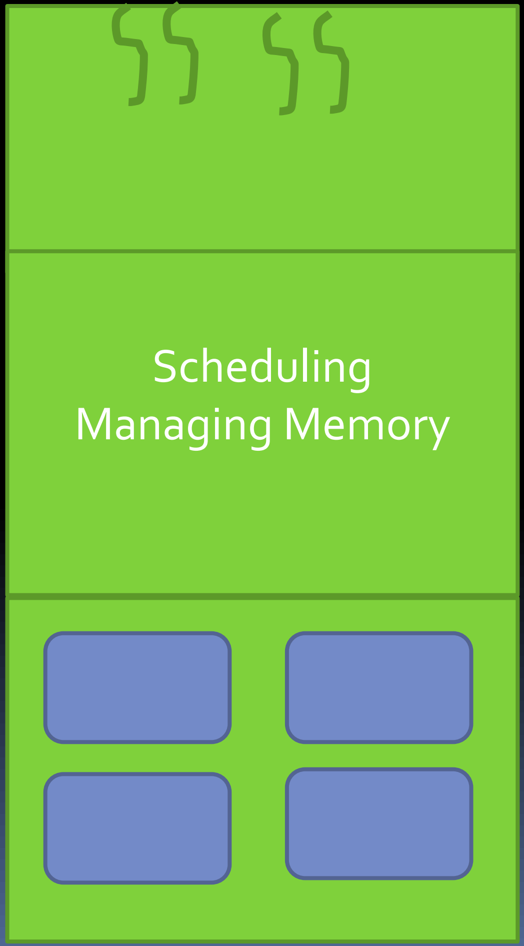
Can lead to inefficient mapping

Architecture: Multicore, shared memory/cache
Resource sharing



Thread Mapping

- A thread executes on a core, it builds up context
  - Cache : enables fast access to data/instruction
  - Memory: pages brought into memory
- On context switch, thread can migrate to different core
  - Context is lost – memory context is important – NUMA machines
- Thread – Core mapping
  - htop
    - Shows the current thread being executed on a core.
- How to measure other activity of a process?



Peeking into the OS

- /proc filesystem
  - A virtual filesystem
  - Gives access to the kernel data structures
  - e.g Virtual memory mapping, open files,
- Sar – system activity report
  - Measure paging activity
  - Disk activity
  - Network activity…

Overview

- Why performance?
- Performance metrics
- Simulators
- Profiling programs – CPU/Memory
- Measuring Operating system activity
- Measuring microarchitectural activity

Measuring Microarchitectural activity

- Sometimes we really need to understand
  - How well application uses hardware.
  - For example:
    - Caches, TLBs,
- Need accurate information about such events

Hardware counters

- Hardware maintains counters of events
  - Introduced in Pentium (http://archive.gamedev.net/archive/reference/articles/article213.html)
- Many events within the hardware are measured
  - Available as MSR (Model Specific Registers)
  - What can be measured depends on the model
  - RDMSR/WRMSR to read/write registers
- Tools: Intel Vtune, perf, AMD uPerf

Using Performance Counters

- Threads from two different cores sharing L3
  - Can interfere with each other
  - Remove cache lines required by the other.
- Perf can be used to identify the number of accesses made by each core.

Instruction Based Sampling

- Given modern processors perform out of order execution
- Many instructions will be simultaneously be active
  - In different stages of execution.
  - Just measuring stats does not allow attributing it to a specific instruction.
- IBS(AMD) – allows sampling certain instructions
  - Collecting all stats about them – during fetch and execution phases.
  - For example: how many times was a speculative fetch for an instruction killed

Conclusion

- Performance
  - Important in many parts of architecture design
  - Needs knowledge of different aspects of system – compilers, OS …
- Reminder – Lab manual will be shared on Channel