Lecture 18: Power Management in Multicore Processors

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Last Lecture (Recap)

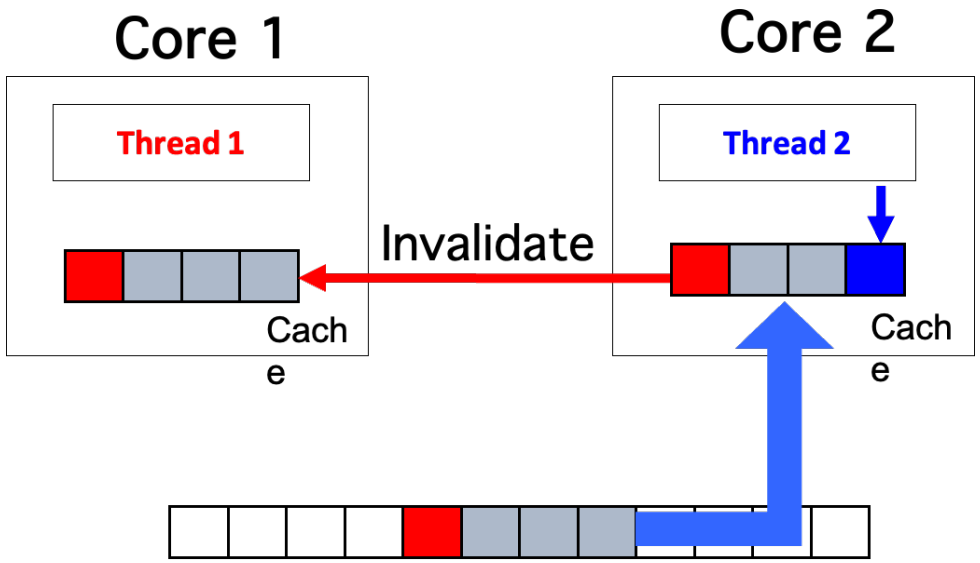
```
me = 1;
you = 1; // globals

me = new Foo;
you = new Bar; // heap

class X {
  int me;
  int you;
}; // fields

arr[me] = 12;
arr[you] = 13; // array indices
```

Two different threads T1 & T2 are involved



1. Initialization – creating mapping of global and heap variables for processes

2. Process creation – creating processes instead of threads

3. Execution – copies of memory pages per process for local updates

4. Synchronization – merging diffs in per-process pages into the global copy

```
1.  /* global variables */
2.  int sum[2];
3.  int main() {
4.    /* heap allocations */
5.    int a = new int[size]; //initialized
6.    T1 = new thread([=]() {
7.      for(int i=0; i<size/2; i++) sum[0]+=a[i];
8.    });
9.    T2 = new thread([=]() {
10.     for(int i=size/2; i<size; i++) sum[1]+=a[i];
11.   });
12.   T1.join(); T2.join();
13.   //Cleanups
14. }
```

- False sharing
- Runtime solutions for detecting/repairing false sharing

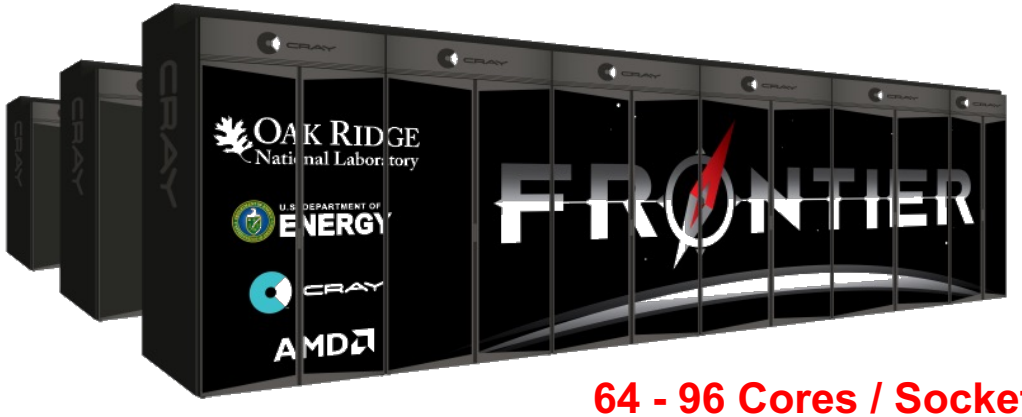
Today's Class

- Power management features on modern processors
- Runtime techniques for achieving energy efficiency
- Quiz-4

The Energy Challenge

Top500 Rank (June 2020)	PFlops/s	Power (MW)	Cores per Socket
1	442	29	48
2	148	10	22
3	94	7	22

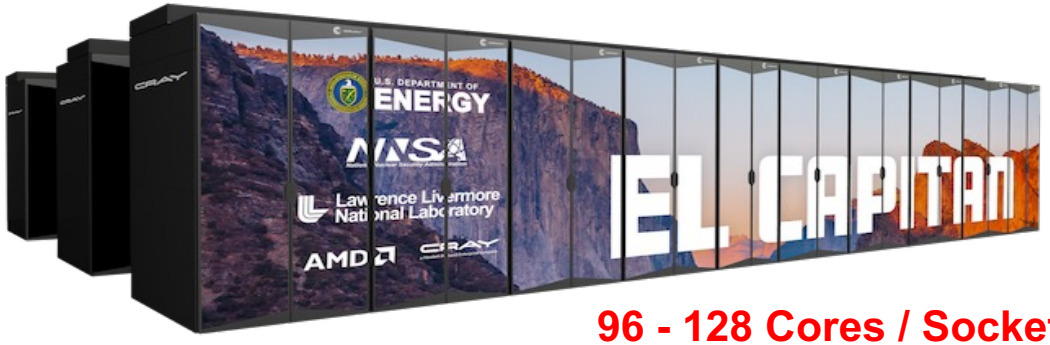
Exascale Systems



64 - 96 Cores / Socket

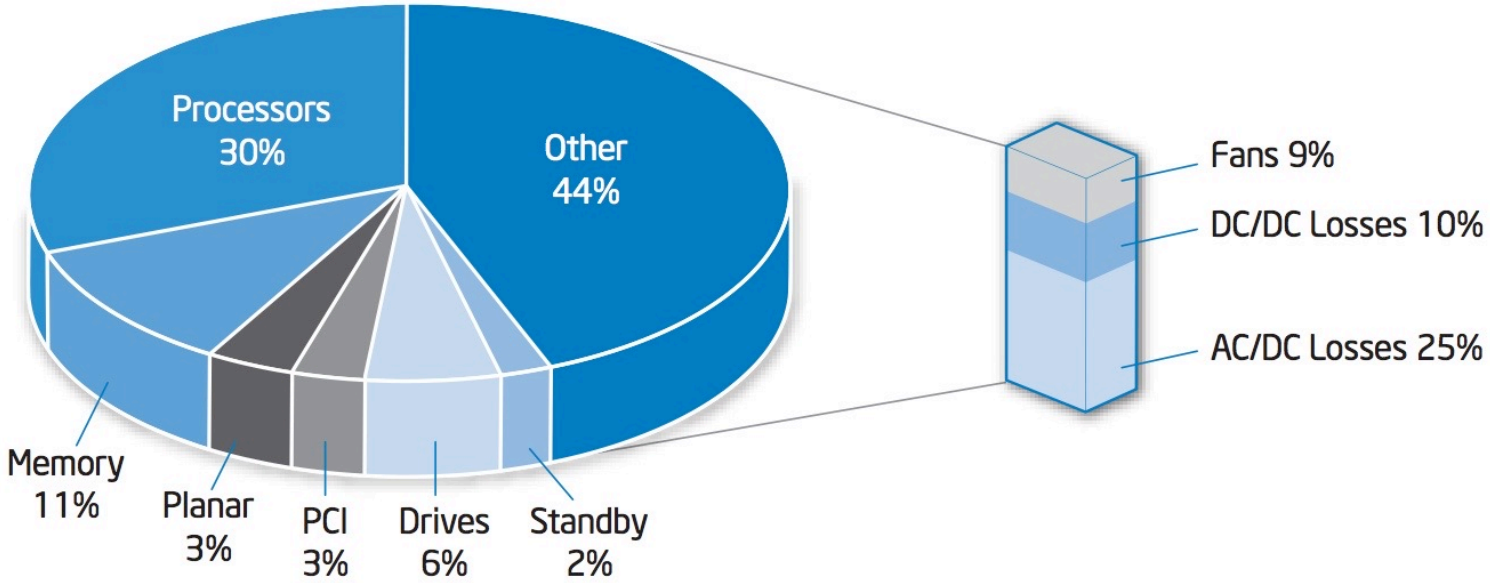
It is crucial to effectively manage and optimise the parallelism to conserve energy

20 MW Power budget



96 - 128 Cores / Socket

Component wise Power Consumption

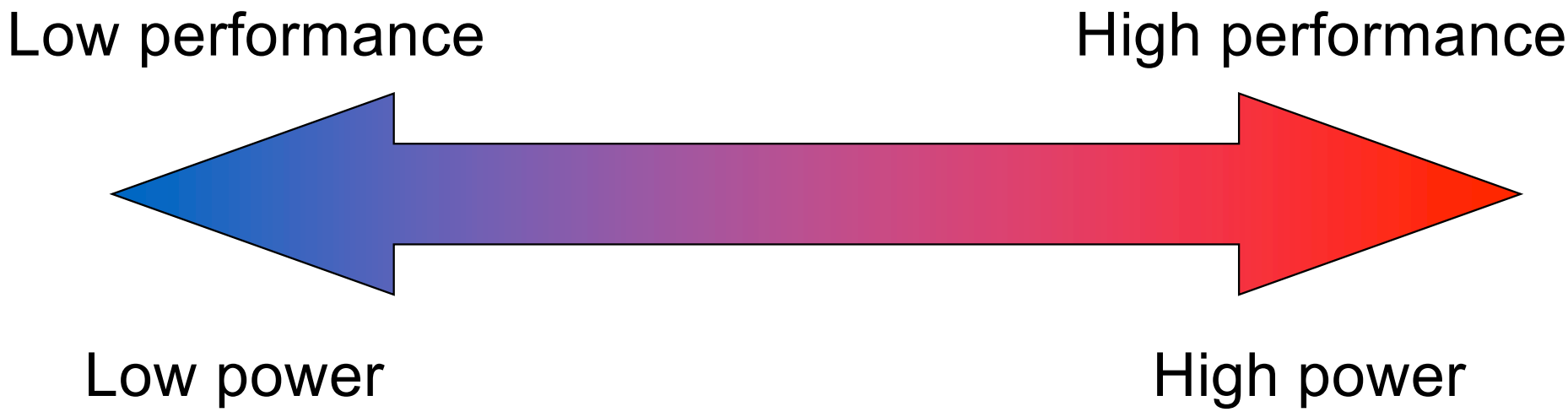


- As per studies of power consumption in a data center
  - 50% of incoming power is consumed by air-conditioning and power-delivery subsystems, even before reaching the servers in a rack
  - Rest 50% consumed by the servers, which can be further broken down into the various elements as shown above

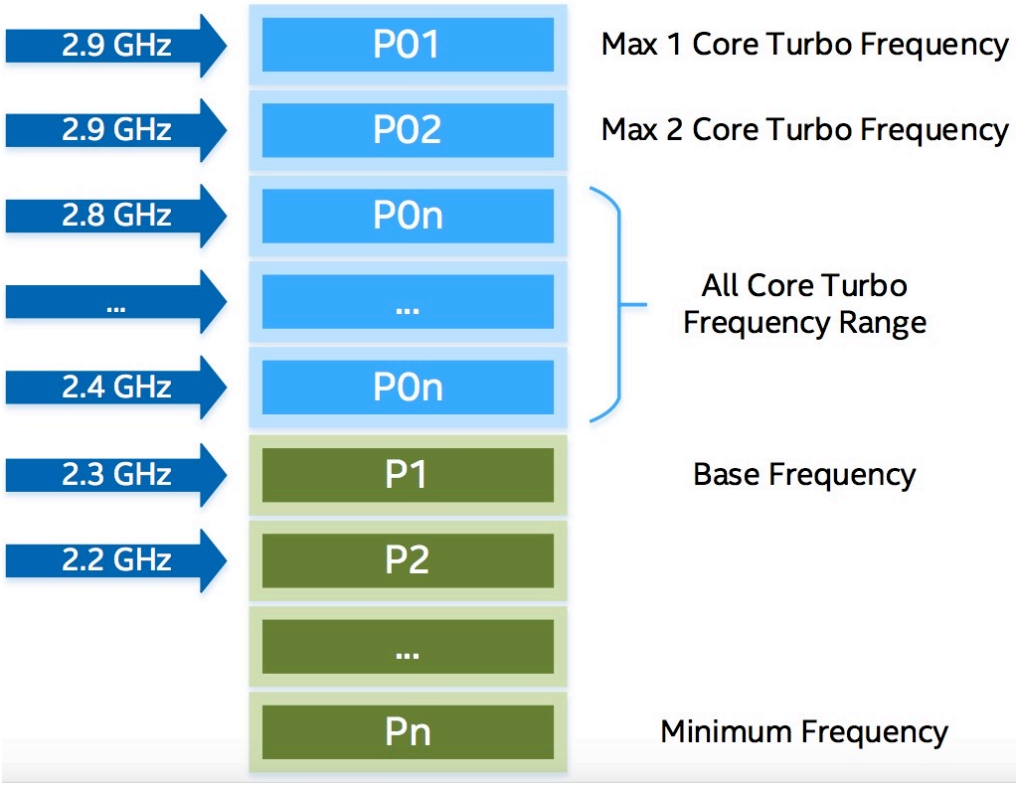
Source: https://www.intel.com/content/dam/support/us/en/documents/motherboards/server/sb/power_management_of_intel_architecture_servers.pdf

Power Management Tradeoff

- Usually, power consumption is proportional to performance



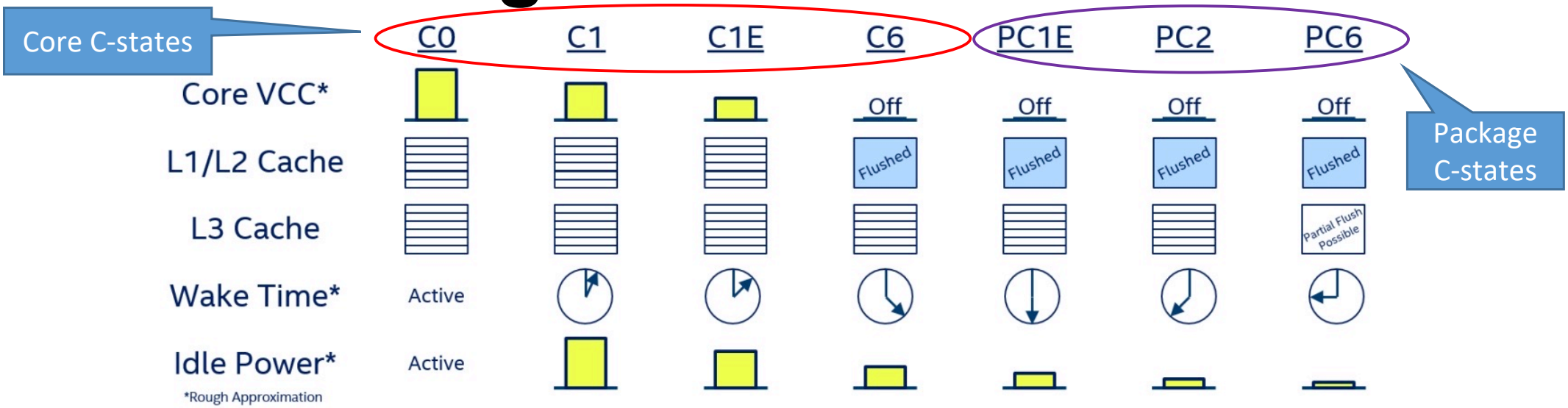
Power Management in Intel Processors



- **Processor States (P-states)**
  - Dynamic Voltage and Frequency Scaling (DVFS) is used by the processor to operate the core at a specific frequency and voltage
    - $P \propto Voltage^2 \times Frequency$
  - Each P-states have an associated frequency
    - Userspace applications are allowed to change the P-states of each core independently on modern processors
  - Multiple levels for turbo frequency depending on the workload
    - $Num_{TurboPstates} = Num_{NonTurboPstates} + 1$
  - During standard operation condition, all cores run at the base frequency
    - Successive P-states below the base frequency differs by 100MHz

Source: https://builders.intel.com/docs/networkbuilders/power-management-technology-overview-technology-guide.pdf

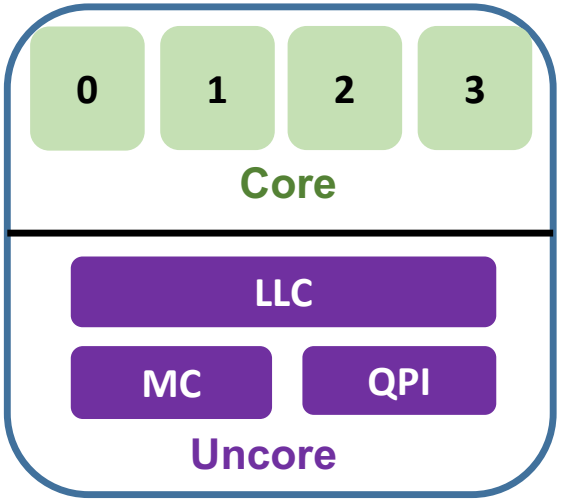
Power Management in Intel Processors



- **CPU States (C-States)**
  - Power states used by the CPUs to reduce the power at **C**ore level or on a CPU **P**ackage **C**ore level (core, private caches, etc.)
  - L1 and L2 are flushed to L3 in C6 (next C0 might require data reload)
  - PC0 is the active state. Rest other PC states require C6 to be active
  - With increasing C number, the CPU sleep mode is deeper, i.e., more circuits and signals are **turned off** (doesn't happen in DVFS) and more time the CPU will require to return to C0 mode, i.e., to wake-up
  - "mwait" and "monitor" instructions can be used to move a core to some C-State

Source: https://builders.intel.com/docs/networkbuilders/power-management-technology-overview-technology-guide.pdf

Power Management in Intel Processors



Multicore Processor

- **Uncore frequency scaling**
  - Changes the frequency of the **uncore elements** in the processor
  - Can be set in the userspace similar to DVFS
  - Currently supported only by Intel processor
    - No information publicly available about AMD processors

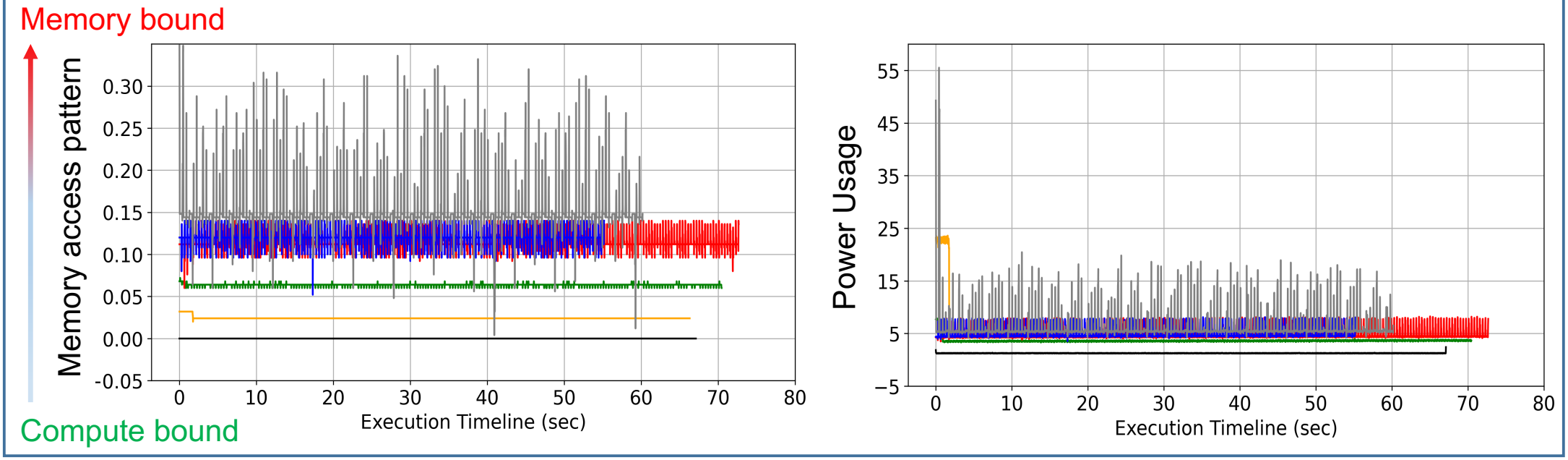
Power Management in Intel Processors

- **Speed Select technology** that provides collection of features for more granular control over processor performance and power usage
  - Allows base frequency of few cores to be increased in exchange of lowering the base frequency of the remaining cores
  - Users can assign priority to each core such that they can get surplus frequency
  - Certain number of cores can run at higher turbo frequency than the rest of the cores
- **Power capping** is another technique where the user specifies the power budget, and the processor dynamically uses DVFS and UFS to balance the core/uncore frequencies such that the power usage of the application stays within the assigned budget

Today's Class

- Power management features on modern processors
- Runtime techniques for achieving energy efficiency
- Quiz-4

Application Type Affects Power Usage



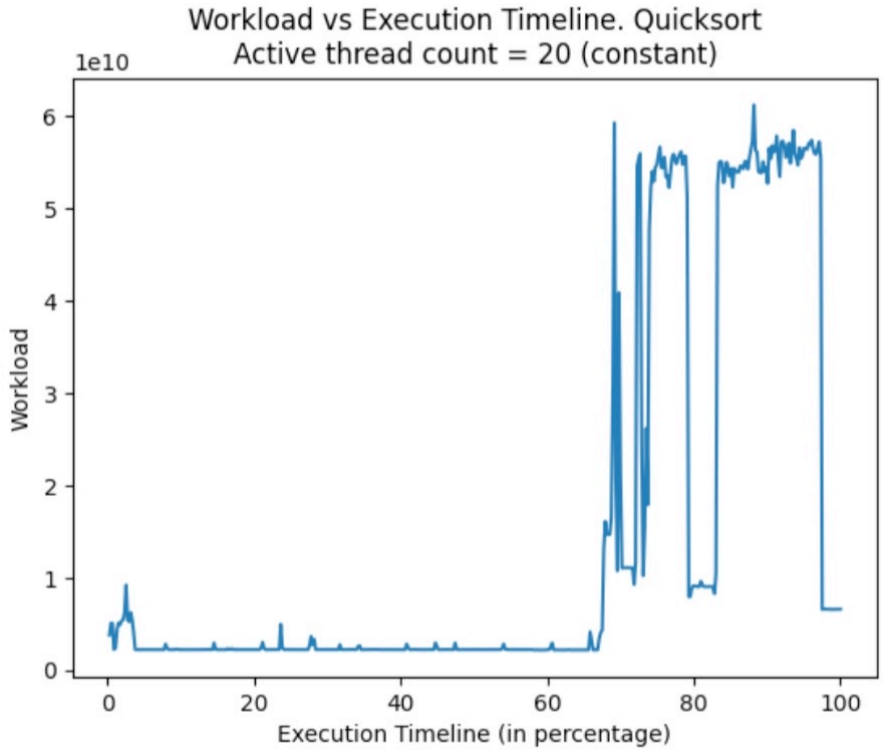
- Compute bound applications requires high core frequency than the uncore frequency
- Memory bound applications require high uncore frequency than the core frequency
- Processor will increase both the core and uncore frequencies as an application moves from compute bound to memory bound, thereby resulting in high power usage

Graph source: https://dl.acm.org/doi/10.1145/3458817.3476163

Class Discussion

- Recall the Energy Aware Scheduler from Lecture #17
  - How can we improve it to improve the energy efficiency even further using frequency scaling?
    - Determine the type of benchmark using hardware performance counters (memory access pattern)
    - Processor frequency throttling based on the type of the benchmark, i.e., compute or memory bound
    - Search for optimal set of frequencies (DVFS, UFS)

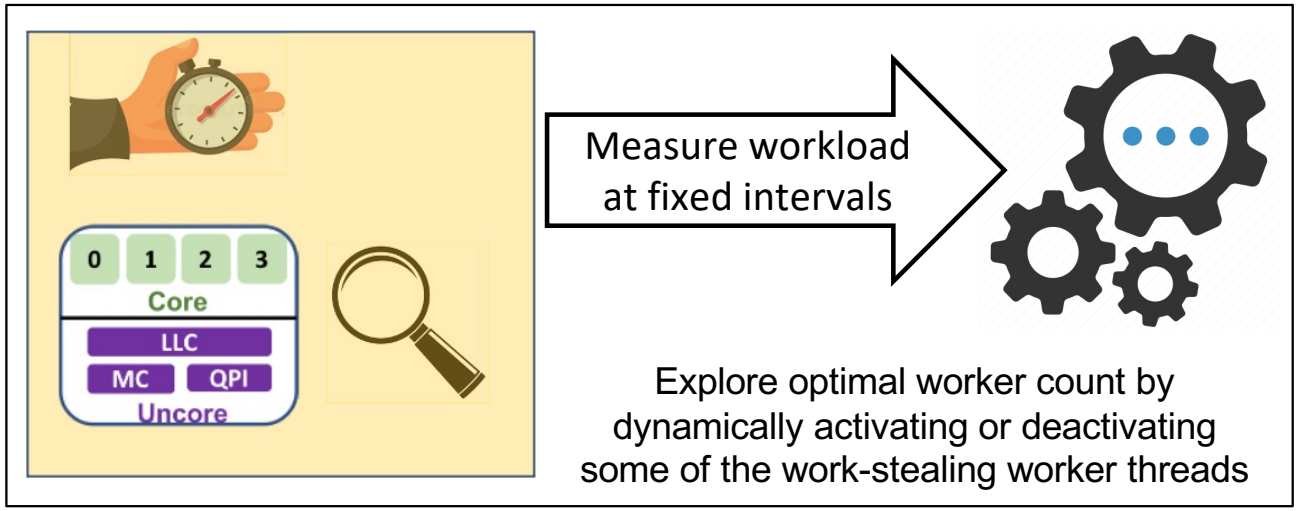
Dynamic Concurrency Throttling



- How about purely multicore execution?
- The graph shows the workload generated by the recursive task parallel quick sort application on a dual socket 10-core Intel Xeon E5-2650v3 processor (using 20 workers)
  - Workload changes dynamically across its execution timeline
    - **Idea:** Dynamically throttling the active thread count can save energy

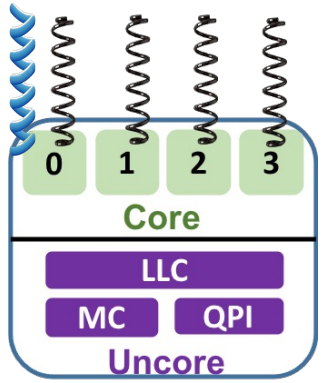
Dynamic Concurrency Throttling

High level architecture



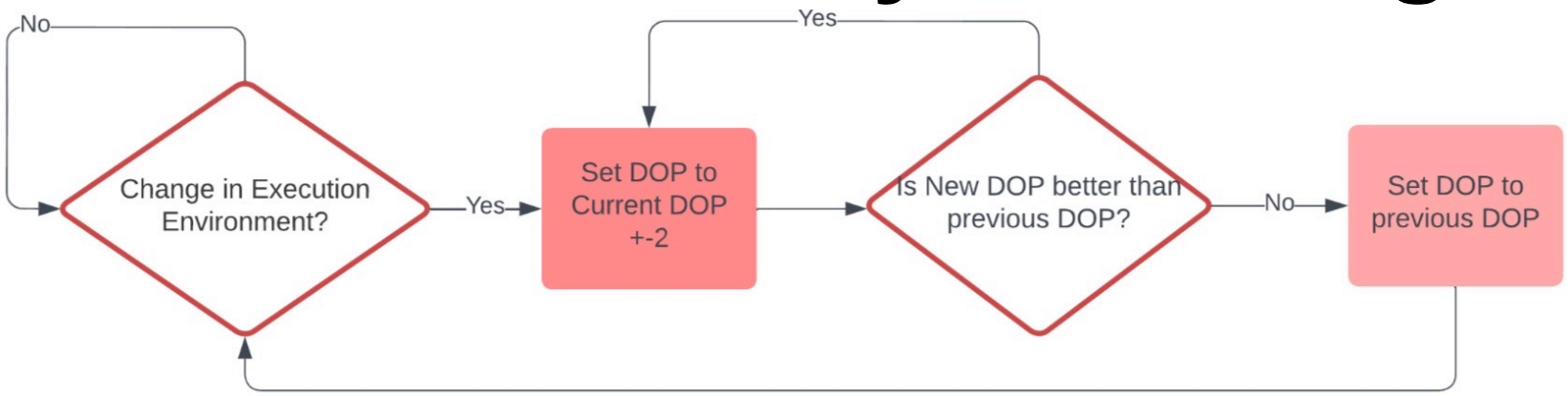

```
#include <iostream>
#include <runtime.h>
int main() {
  runtime::init();
  parallel_computation();
  runtime::finalize();
  return 0;
}
```

Daemon thread running the dynamic concurrency throttling algorithm. Daemon created and destroyed inside init and finalize, respectively



Paper: https://dl.acm.org/doi/10.1145/2594291.2594292

Dynamic Concurrency Throttling



- Daemon thread wakes up at fixed intervals to monitor the workload by reading hardware performance counters
  - E.g., instructions retired per second, or joules per instruction retired
- Measures the difference in workload to calculate the new Degree of Parallelism (DOP), i.e., total number of active workers (less workers if workload decreases and vice-versa)
- Each work-stealing workers checks currently set DOP and total number of workers currently active. If more workers are active than DOP, then extra workers go to sleep (pthread_cond_wait), whereas if less workers active tha DOP then more workers are brought into active state (pthread_cond_signal)
  - Carried out during push, pop, and steal operations

Paper: https://dl.acm.org/doi/10.1145/2594291.2594292

Reading Materials

- Power management technology overview
  - https://builders.intel.com/docs/networkbuilders/power-management-technology-overview-technology-guide.pdf
- Dynamic concurrency throttling
  - https://dl.acm.org/doi/10.1145/2594291.2594292

Next Lecture

- Parallel programming using SIMD vector units