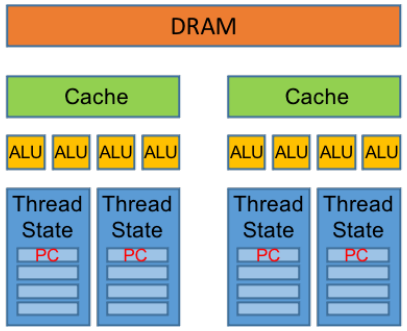
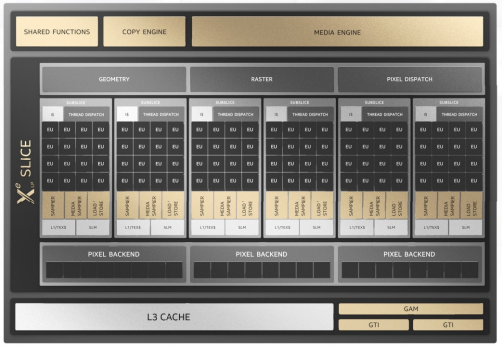
Lecture 21: Power Management in Multicore Processors

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

Last Lecture (Recap)

- Multicore processors are latency oriented, whereas GPUs are throughput oriented

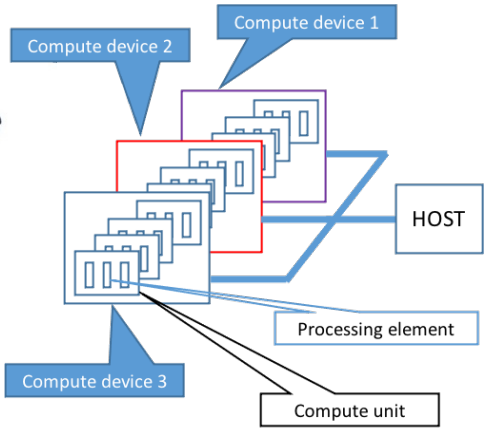





```
#define SIZE 1024
int main() {
   // Step 1 & 2 Setup and create environment
   compute::device device;
   compute::context context;
   compute::command_queue queue = setup(context, device);
   // Step 3a. Manage memory (allocate host data)
   std::vector<float> A(SIZE), B(SIZE), C(SIZE); // Initialize
   // Step 3b. Create device vectors and copy host data to device
   compute::vector<float> dA(A.begin(), A.end(), queue);
   compute::vector<float> dB(B.begin(), B.end(), queue);
  // Step 3c. Create output vector at device
   compute::vector<float> dC(SIZE, context);
  // Step 4. Prepare program (create OpenCL kernel)
   BOOST_COMPUTE_FUNCTION(float, add, (float x, float y), {
      return x + y;
   });
   // Step 5. Execute kernel (queue at device)
   compute::transform(dA.begin(), dA.end(), dB.begin(), dC.begin(), add, queue);
   // Step 6. Get result (copy output vector from device to host vector)
   compute::copy(dC.begin(), dC.end(), C.begin(), queue);
   return 0;
}
```

Executing OpenCL Programs

1. Query host for OpenCL devices
2. Create a context to associate OpenCL devices
3. Create programs for execution on one or more associated devices
4. Select kernels to execute from the programs
5. Create memory objects accessible from the host and/or the device
6. Copy memory data to the device as needed
7. Provide kernels to command queue for execution
8. Copy results from the device to the host



Today's Class

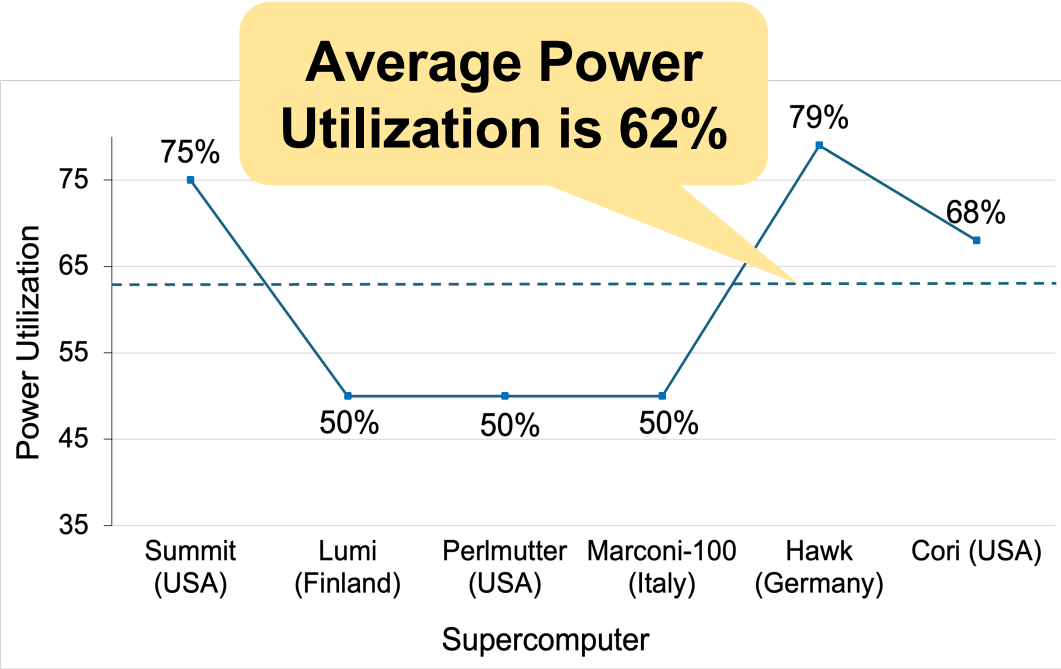
- Power management features on modern processors
- Runtime techniques for achieving energy efficiency
- Quiz-5

Resource Utilization in the Exascale Era

Increasing number of sockets and cores per node

Rank of Top500 (November 2025)	Sockets Per Node	Cores Per Node
1	**4**	96
2	1	64
3	2	104
4	**4**	**288**
5	2	96

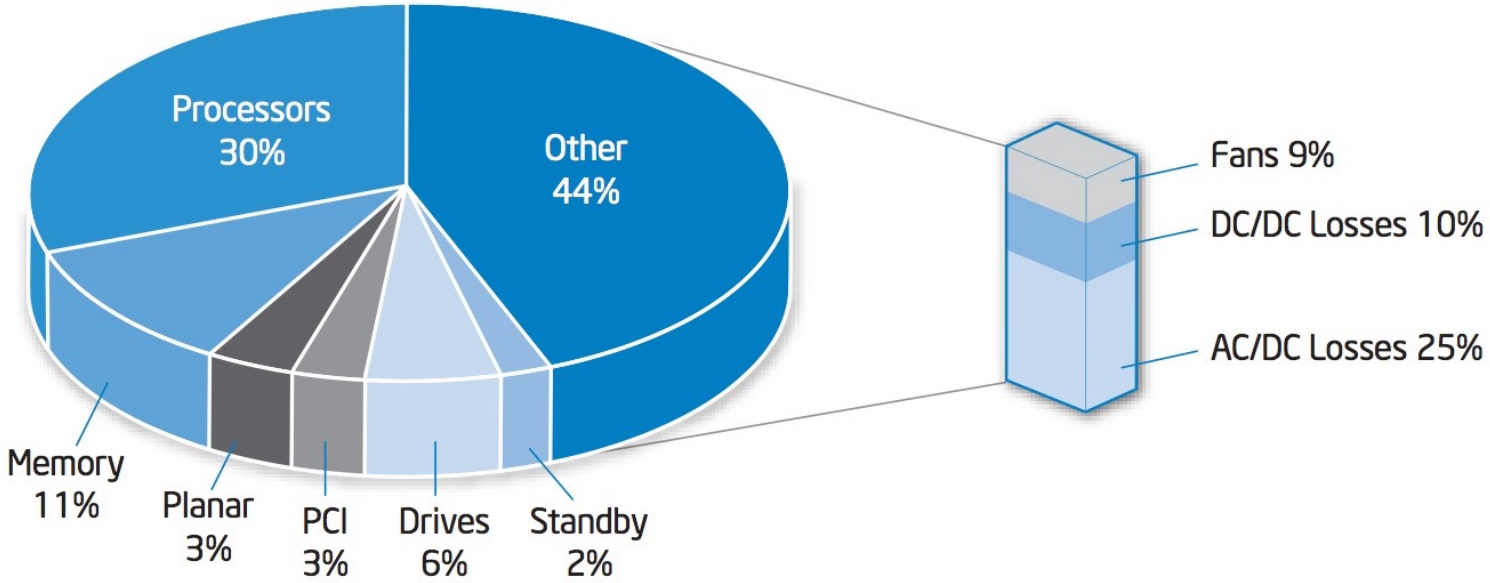
Power usage at supercomputers



It is critical to improve resource utilization for achieving energy efficiency

1. https://top500.org/lists/top500/ 2. Patki et.al. [ICS2025]

Component wise Power Consumption

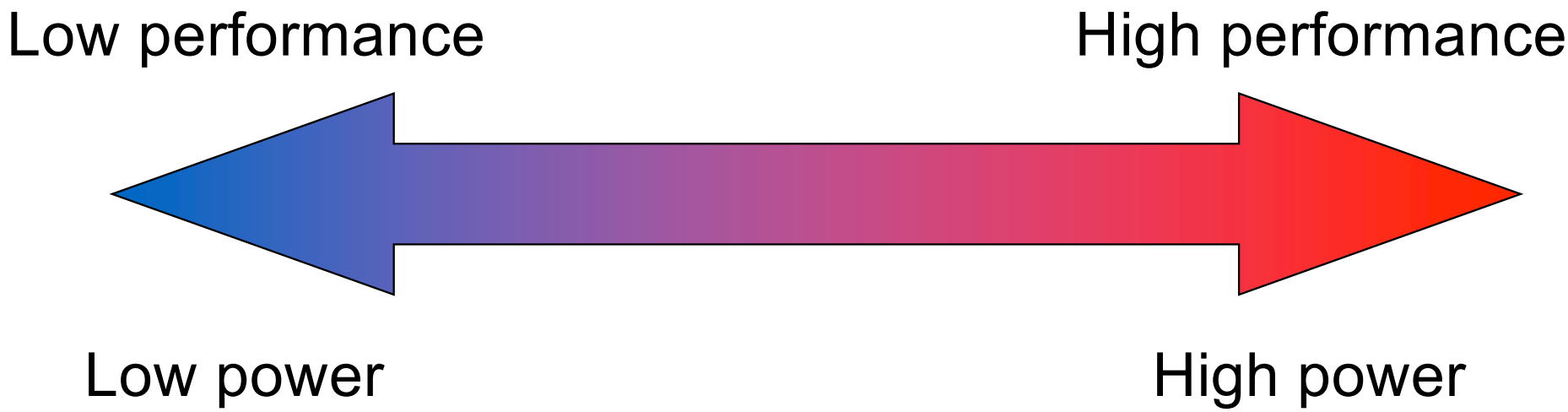


- As per studies of power consumption in a data center
  - 50% of incoming power is consumed by air-conditioning and power-delivery subsystems, even before reaching the servers in a rack
  - Rest 50% consumed by the servers, which can be further broken down into the various elements as shown above

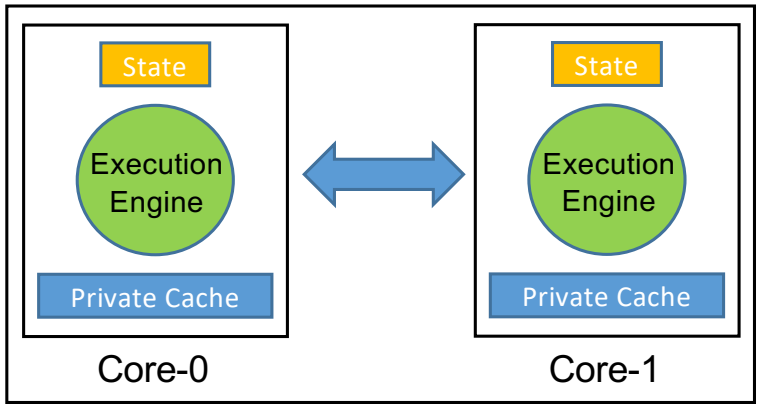
Source: https://www.intel.com/content/dam/support/us/en/documents/motherboards/server/sb/power_management_of_intel_architecture_servers.pdf

Power Management Tradeoff

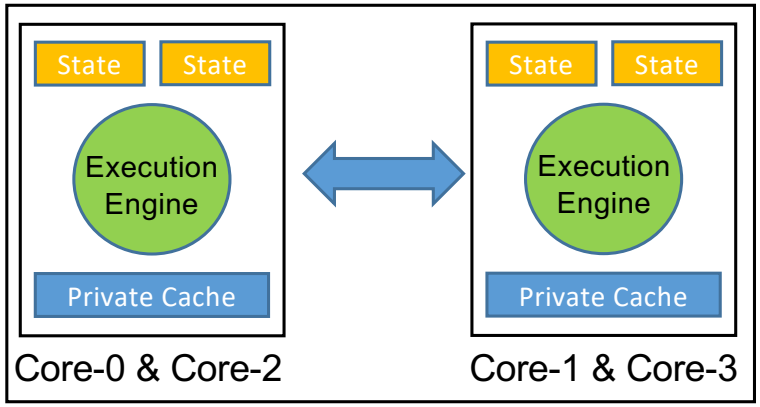
- Usually, power consumption is proportional to performance



Physical VS. Logical Cores



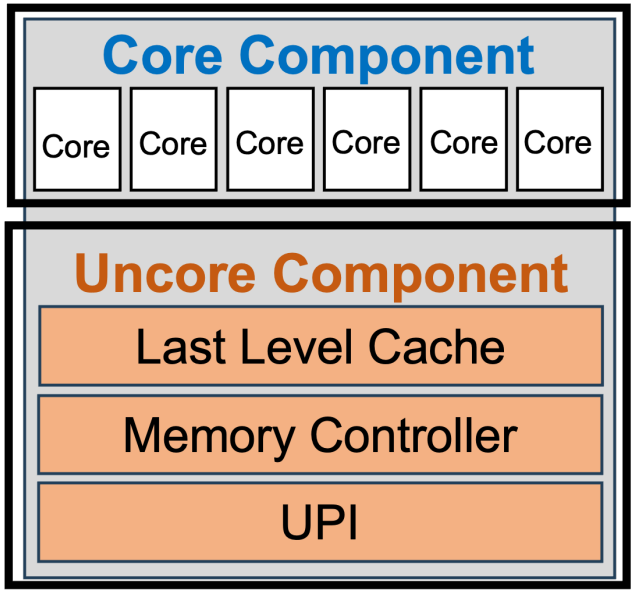
Dual-core processor with hyperthreading **DISABLED**



Dual-core processor with hyperthreading **ENABLED**

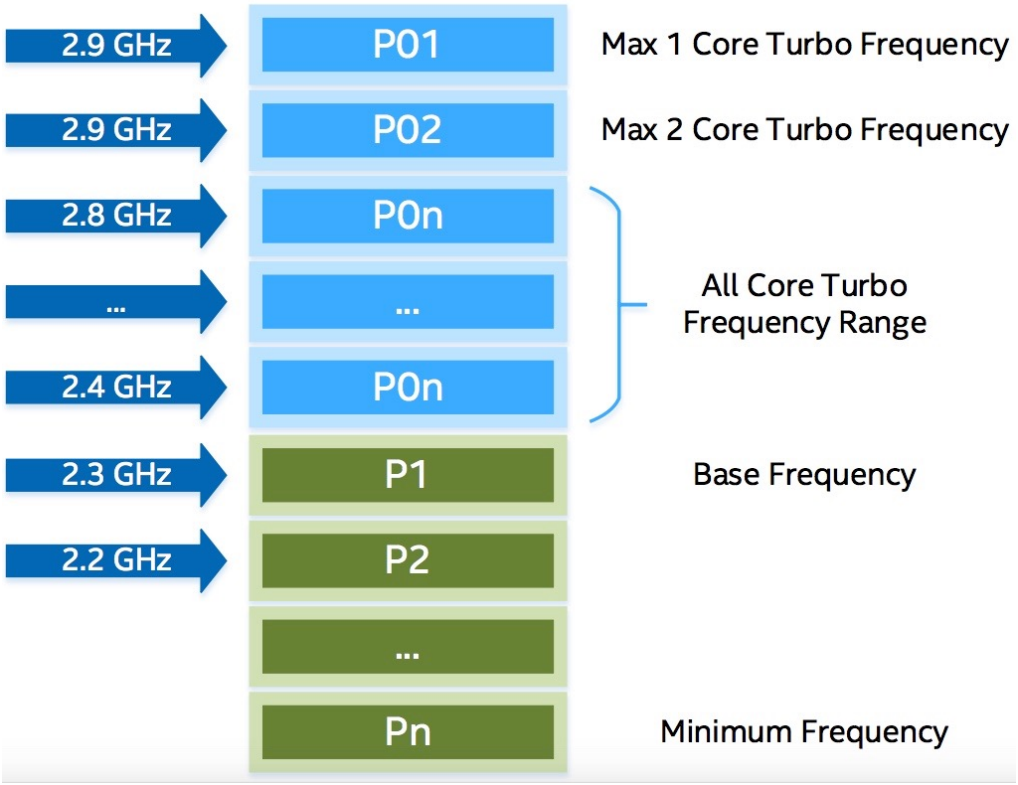
- Architectural state of a core are the registers (EBP, ESP, EIP, etc.)
- Logical cores of a processor share
  - Private cache
  - Execution engine
  - System bus interface
- If the execution of one of the logical core blocks (e.g., Core-0 waiting for a memory fetch from the DRAM) then the other logical core (Core-2) can resume its execution with its own state

Achieving Energy Efficiency on Multicores



- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level

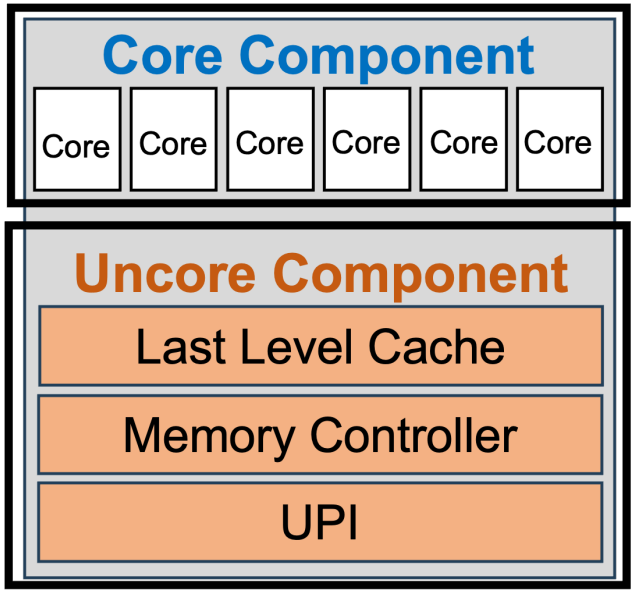
DVFS in Multicore Processors

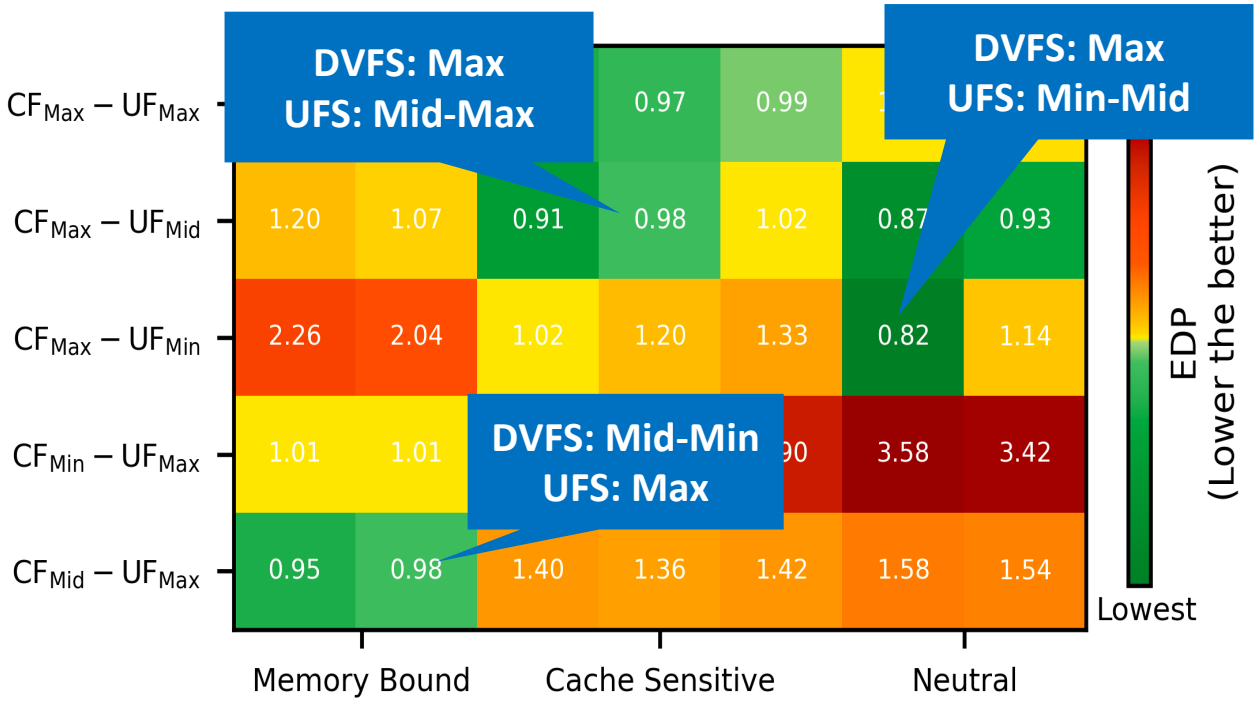


- **Processor States (P-states)**
  - Dynamic Voltage and Frequency Scaling (DVFS) is used by the processor to operate the core at a specific frequency and voltage
    - $P \propto Voltage^2 \times Frequency$
  - Each P-states have an associated frequency
    - Userspace applications are allowed to change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

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

Achieving Energy Efficiency on Multicores





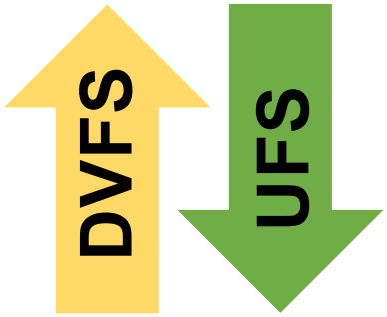
- Dynamic Voltage and Frequency Scaling (DVFS)
  - Core-level
- Uncore Frequency Scaling (UFS)
  - Socket-level
  - Can be set in the userspace similar to DVFS

Heatmap represents the change in EDP with a particular combination of core-uncore frequency relative to default settings

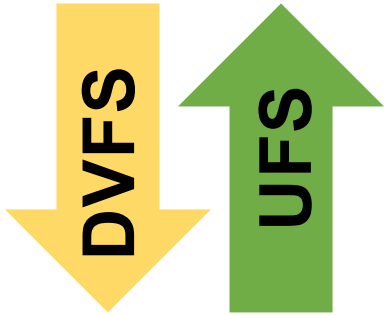
Achieving Energy Efficiency: Insights

- 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

Cache-Sensitive and **Neutral**



Memory-bound

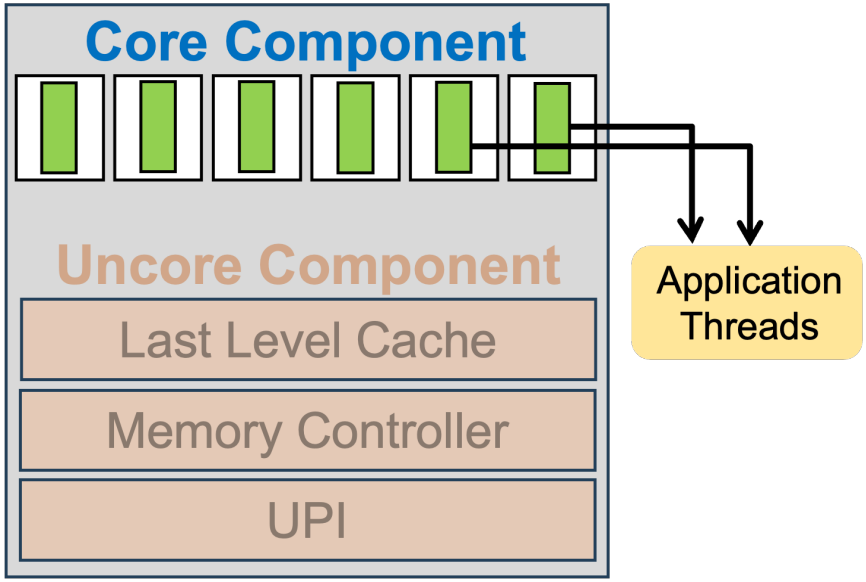


Power Management in Multicore Processors

- **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

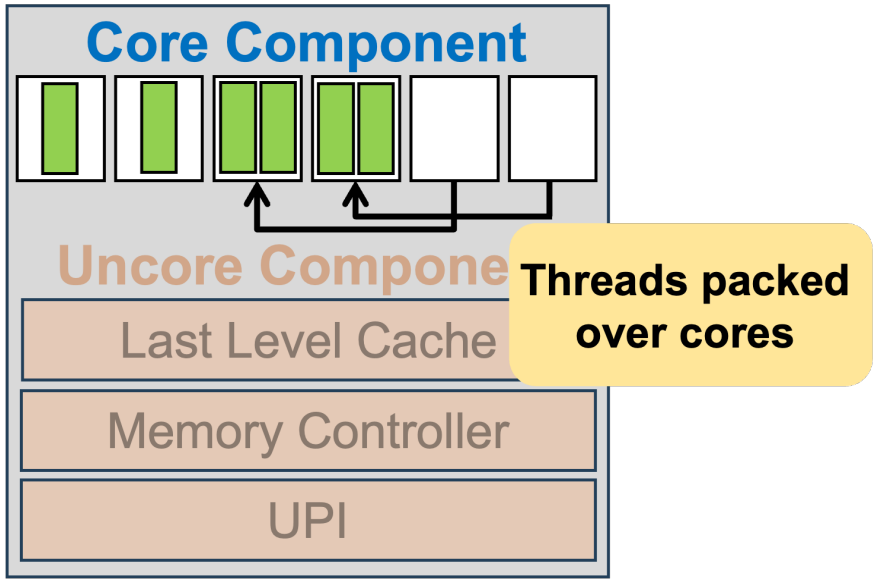
Source: https://builders.intel.com/docs/networkbuilders/power-management-technology-overview-technology-guide.pdf

Achieving Energy Efficiency on Multicores



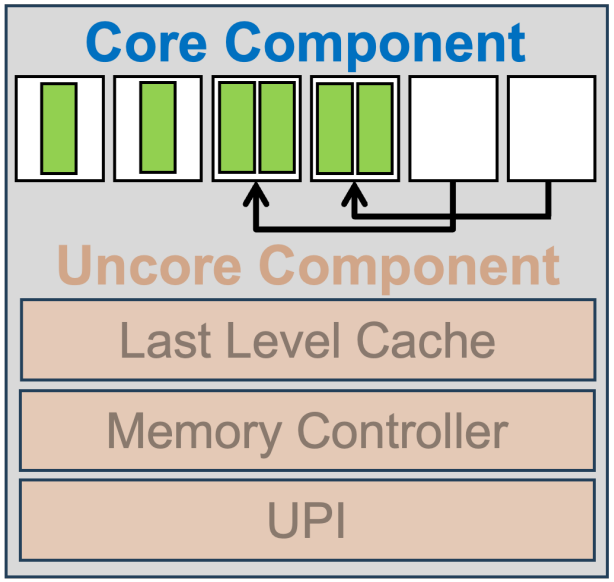
- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation

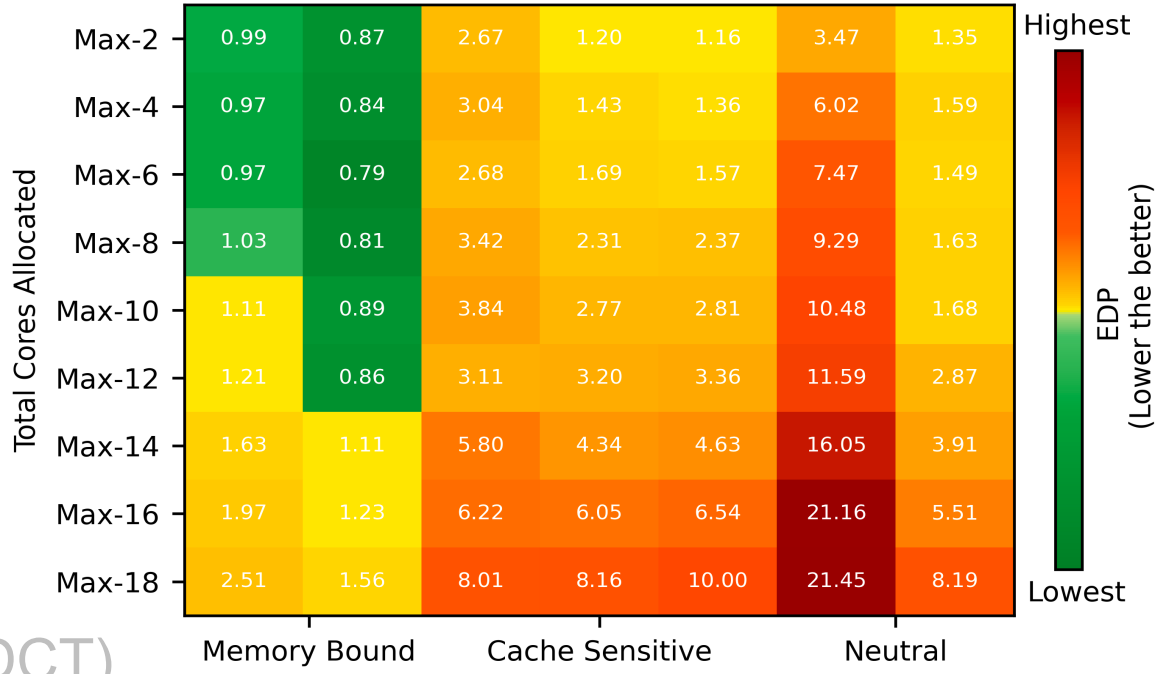
Achieving Energy Efficiency on Multicores



- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation
  - Thread packing and unpacking technique provides runtime independence

Achieving Energy Efficiency on Multicores

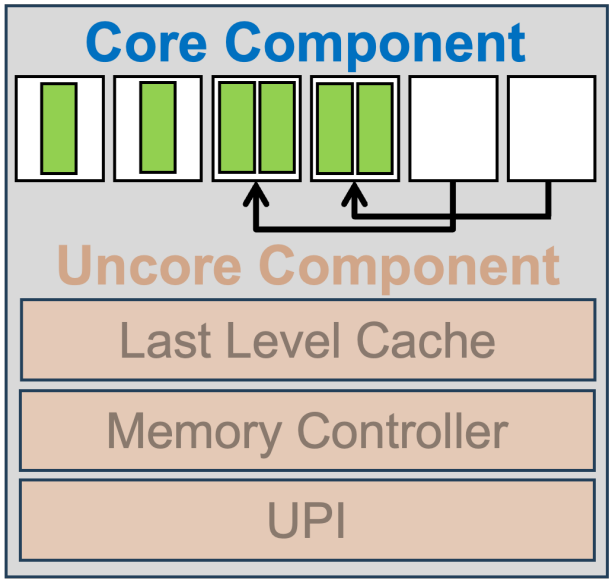


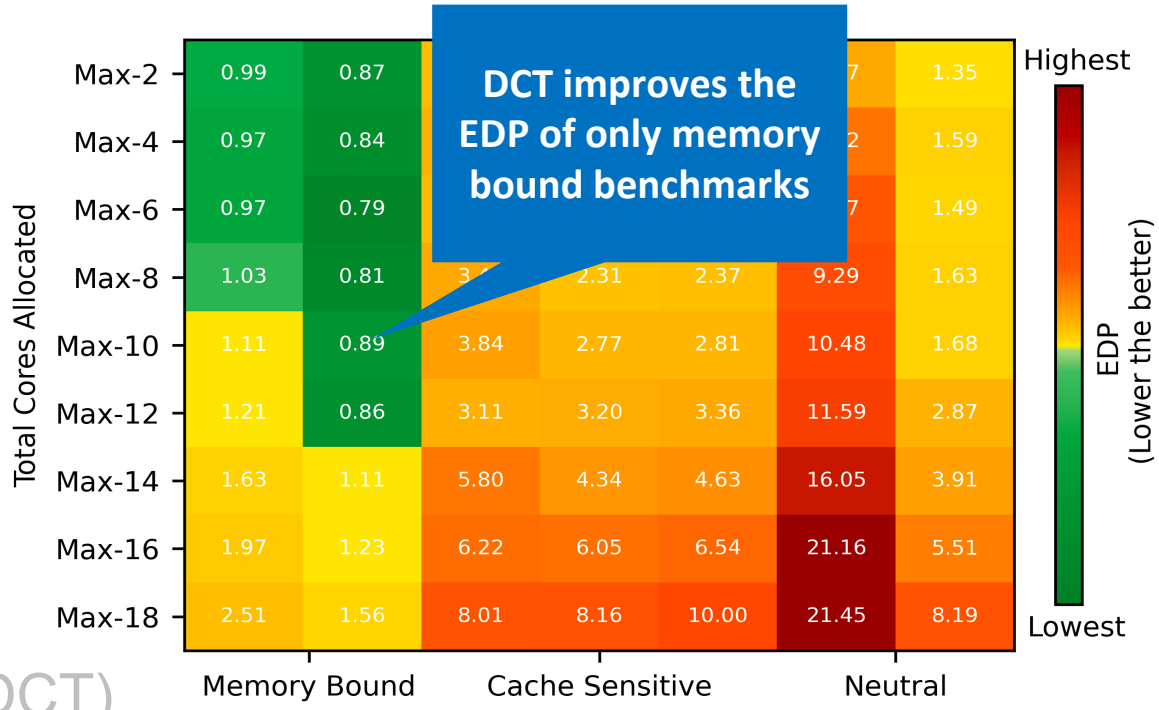


- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation
  - Thread packing and unpacking technique provides runtime independence

Heatmap represents the change in EDP by changing the core count relative to default with maximum core allocation

Achieving Energy Efficiency on Multicores





- Dynamic Concurrency Throttling (DCT)
  - Adjusts the application level parallelism by controlling core allocation
  - Thread packing and unpacking technique provides runtime independence

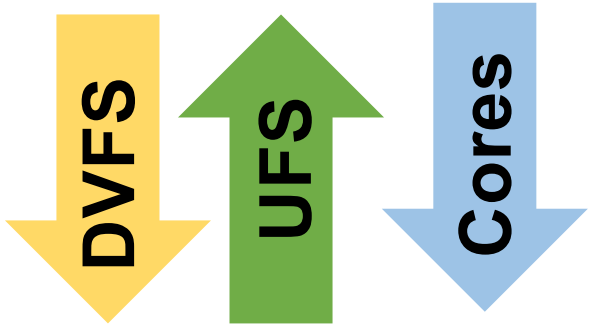
Heatmap represents the change in EDP by changing the core count relative to default with maximum core allocation

Achieving Energy Efficiency: Insights

- 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
- Compute bound applications require large core count than memory bound applications

Cache-Sensitive and **Neutral**

Memory-bound



Today's Class

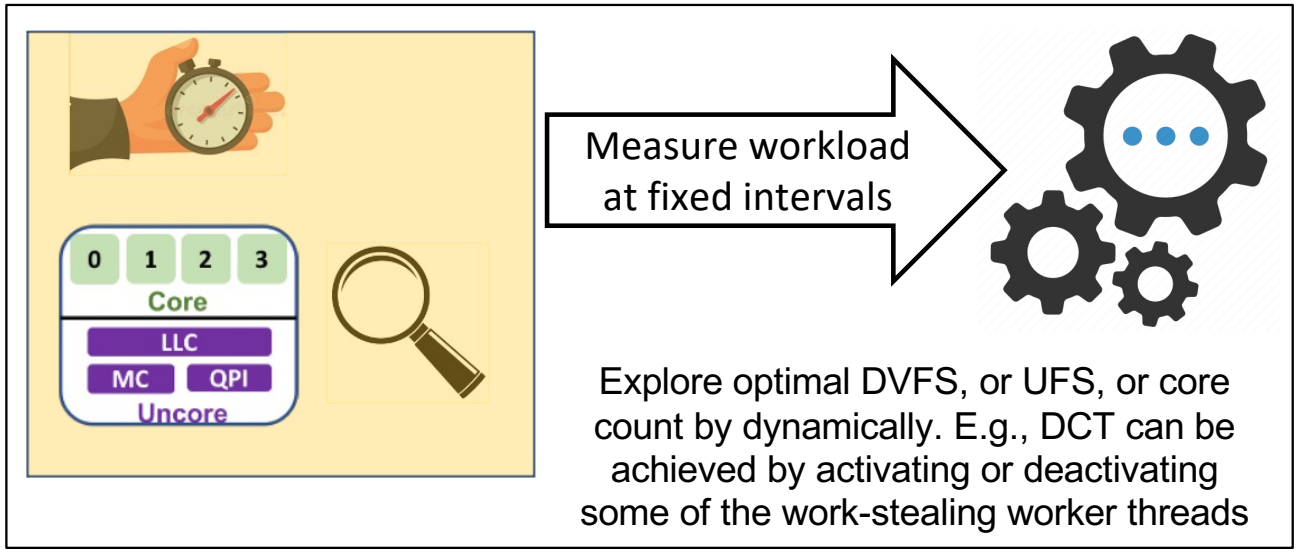
- Power management features on modern processors
- Runtime techniques for achieving energy efficiency
- Quiz-4

Class Discussion

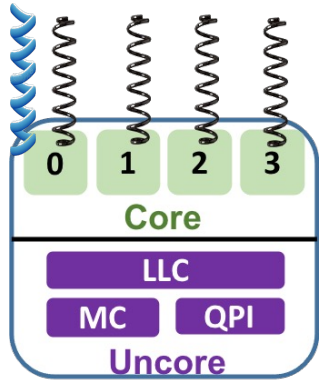
- How can we achieve energy efficiency on multicore processor?
  - Online or by using an offline trained model
  - Determine the type of benchmark using hardware performance counters (memory access pattern)
  - Apply processor frequency throttling or DCT based on the type of the benchmark, i.e., compute or memory bound
    - Search for optimal set of frequencies (DVFS, UFS)
    - Search for optimal core count

Dynamically Achieving Energy Efficiency

High level architecture for applying DVFS, UFS, or DCT



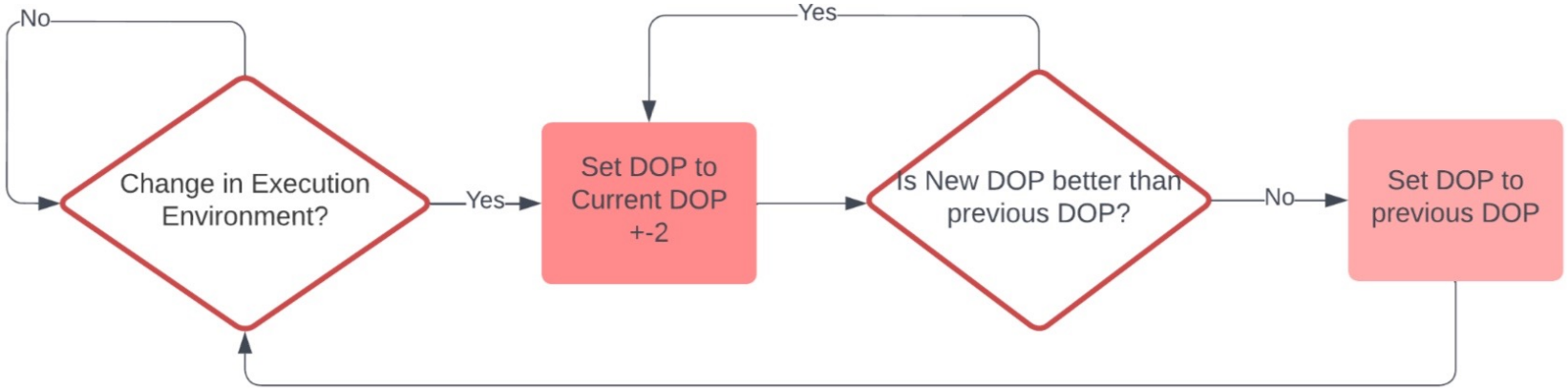

```
#include <iostream>
#include <runtime.h>
int main() {
  runtime::init();
  parallel_computation();
  runtime::finalize();
  return 0;
}
```



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

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

DCT in Work-Stealing Runtime



- 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

- Introduction to memory consistency