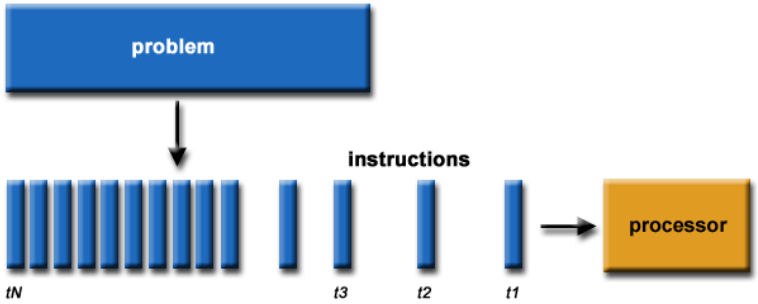
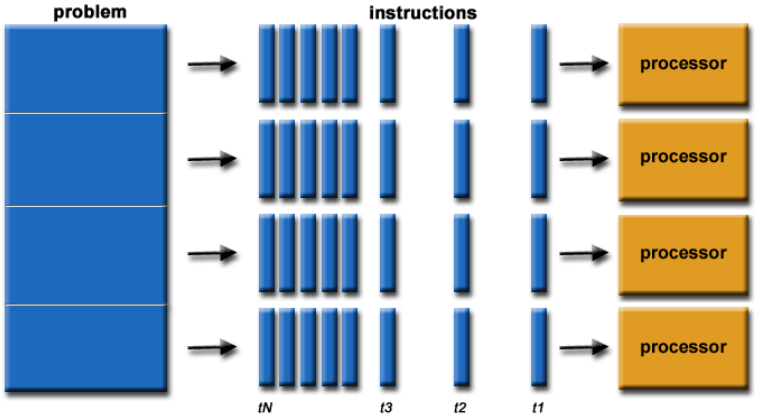
Lecture 01: Introduction to PRMP

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

What is Parallel Programming?



- Serial
  - One instruction at a time

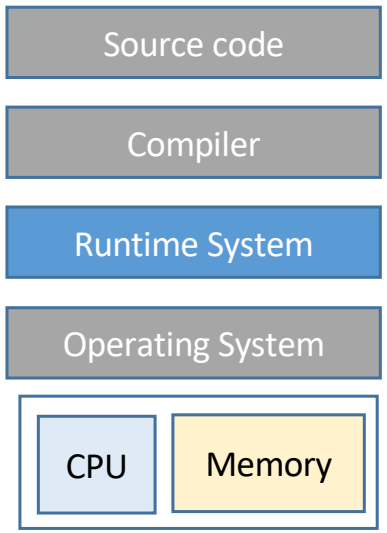


- Parallel
  - Multiple instructions in parallel

Picture source: https://hpc.llnl.gov/documentation/tutorials/introduction-parallel-computing-tutorial

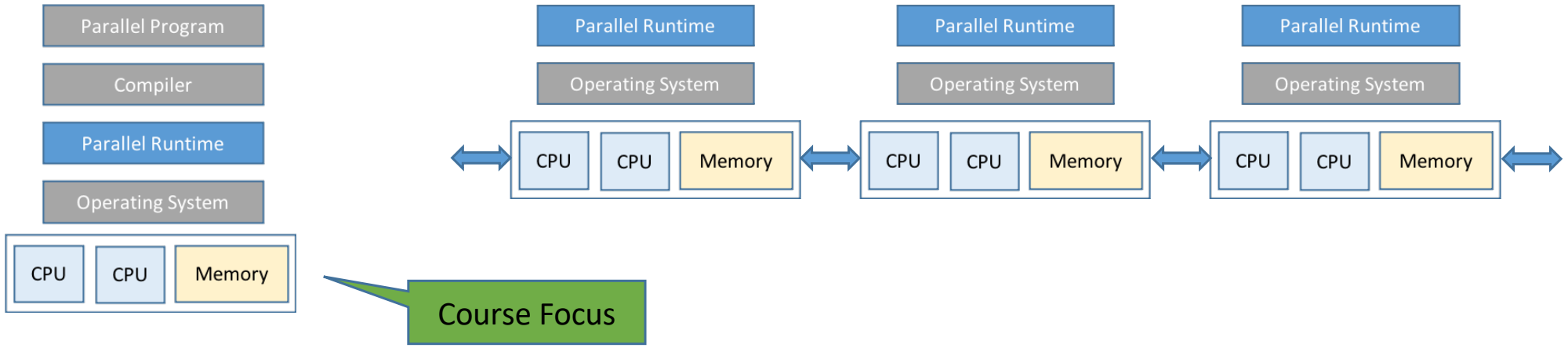
What are Runtime Systems?

- Runtime systems helps in the execution of a program by helping it interact with with the underlying computing resources such as CPU and memory
  - A software implementation that sits above the OS



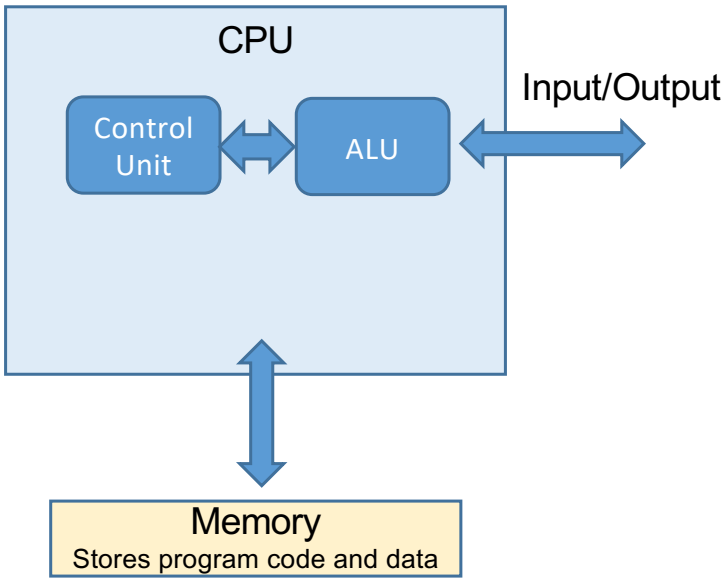
What are Parallel Runtimes?

- Runtime systems to manage the execution of a parallel program over multiple compute resources



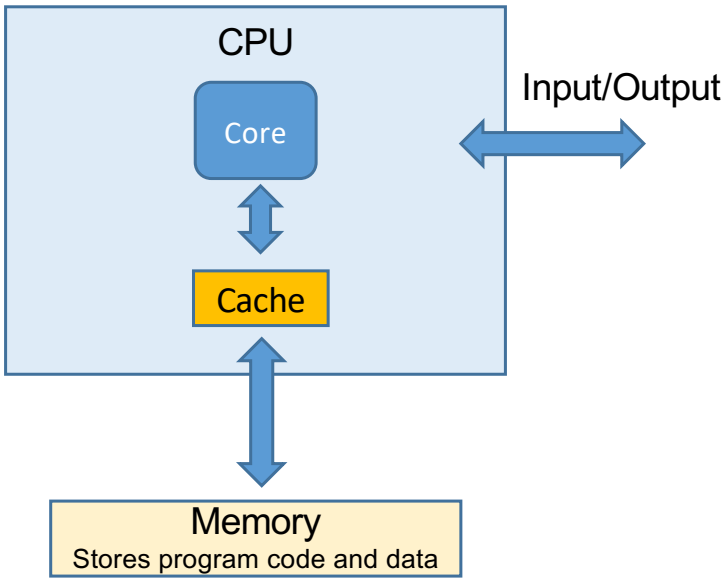
Let us first quickly try to understand the evolution of computers

Von Neumann Architecture



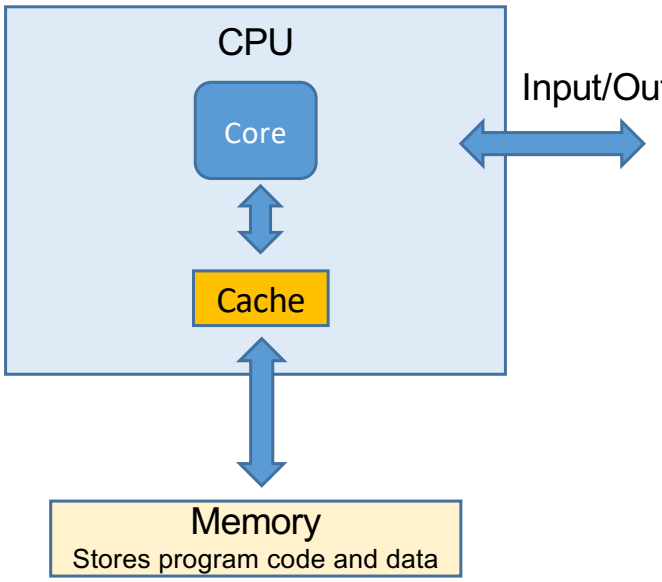
- John Von Neuman in 1945 came up with the architecture for computers that we even use today (albeit with several changes)

Von Neumann Architecture & Associated Issues

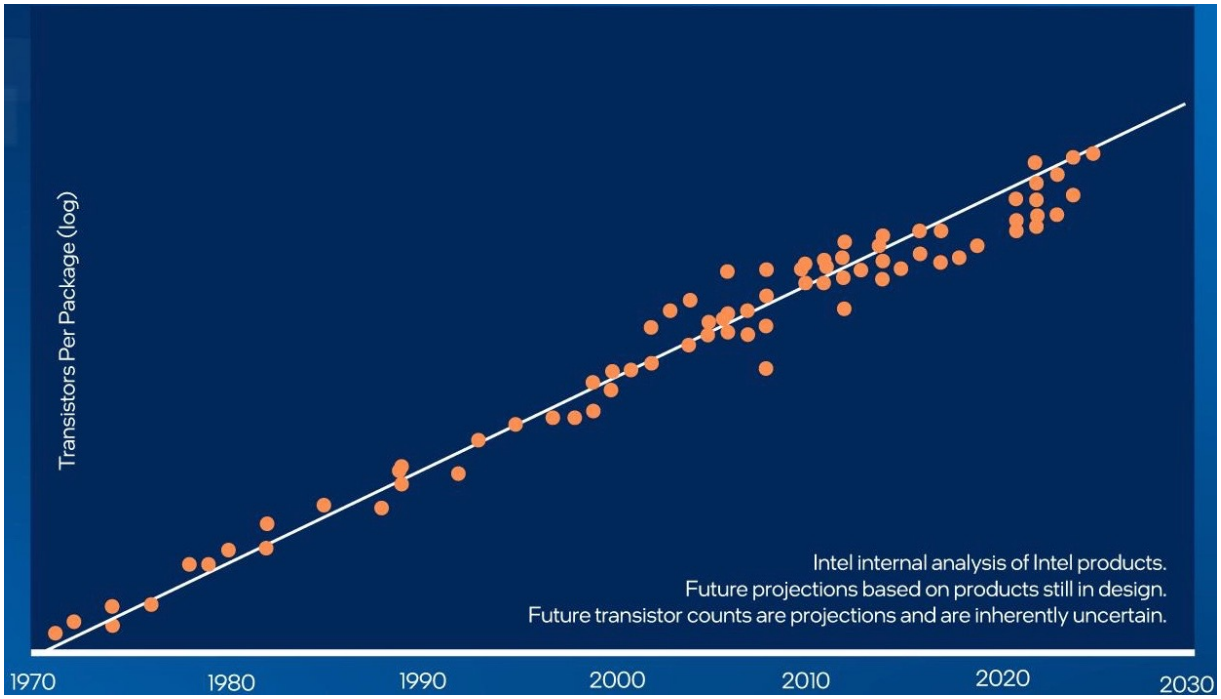


- Memory bottleneck
  - Problem
    - Access latency to memory quite high
      - High CPU stalls while fetching code and data
  - Solution
    - Add cache on the CPU chip to store frequently accessed memory

Von Neumann Architecture & Associated Issues

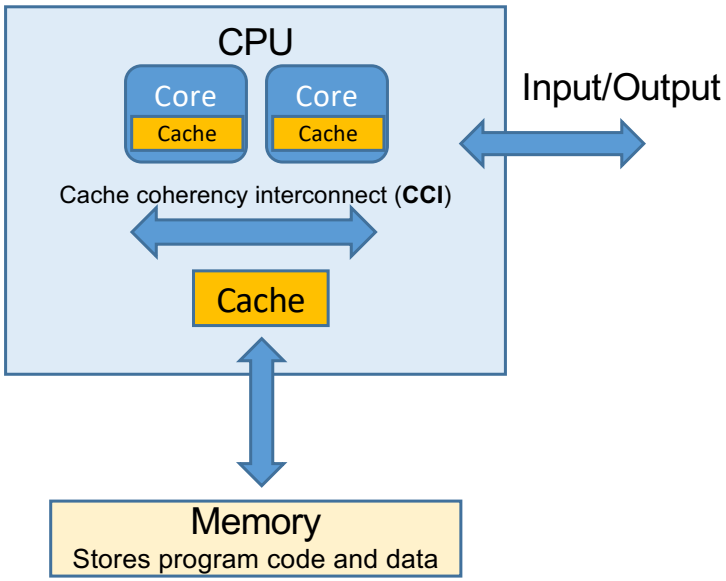


- Performance bottleneck



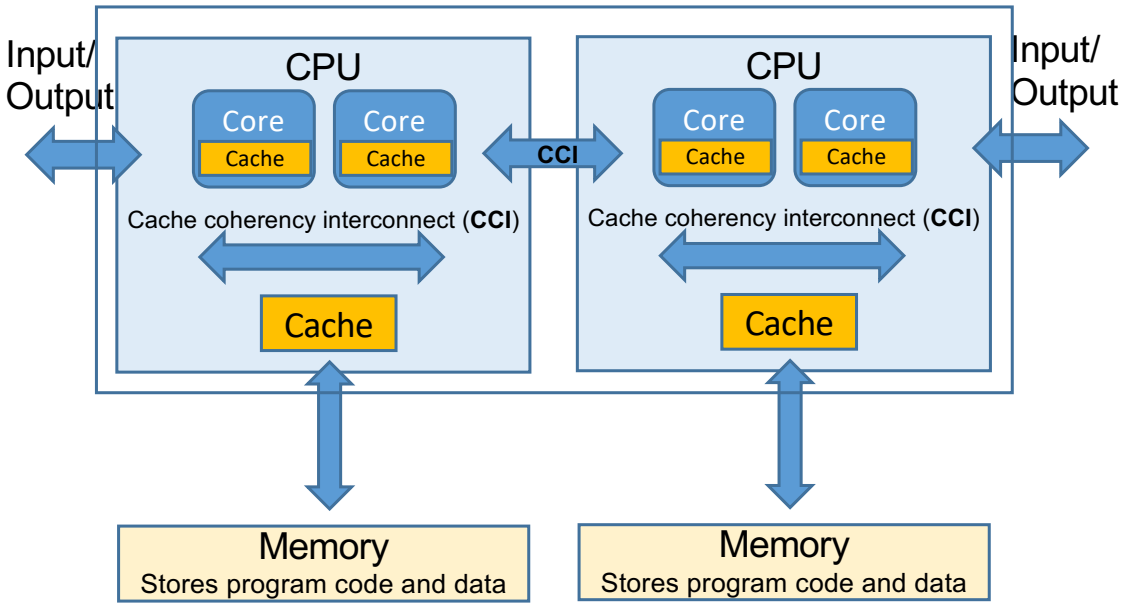
Graph source: https://www.intel.com/content/www/us/en/newsroom/opinion/moore-law-now-and-in-the-future.html

Von Neumann Architecture & Associated Issues



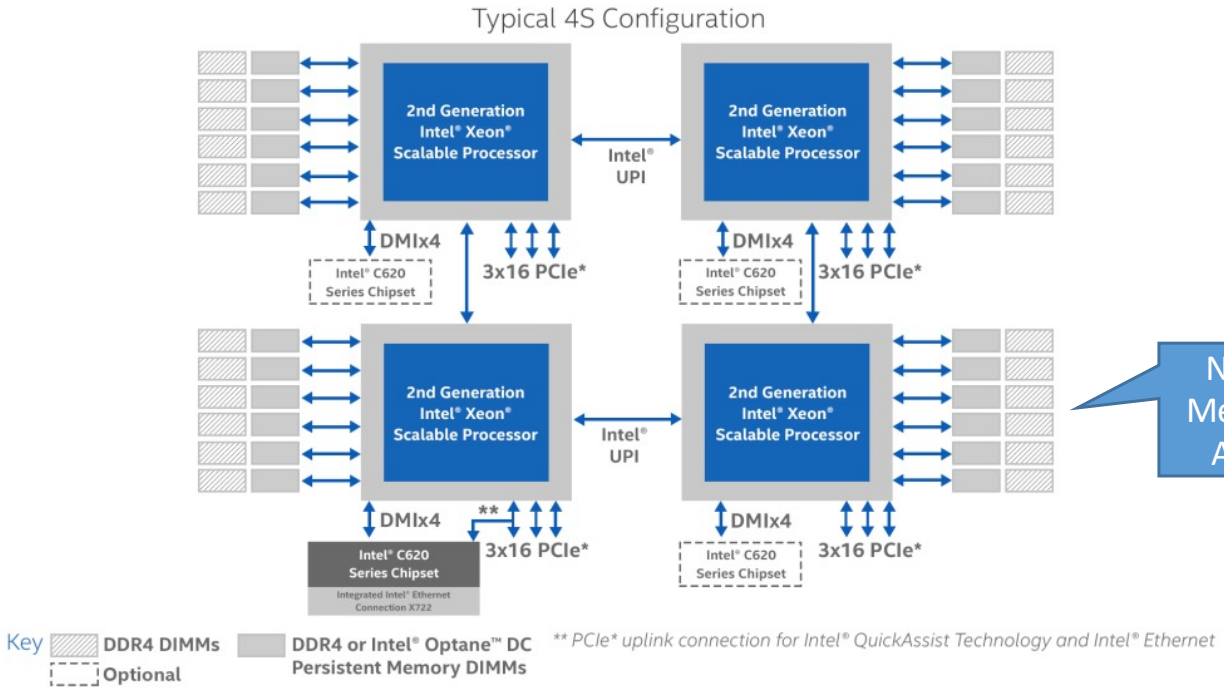
- Performance bottleneck
  - Problem
    - Design issues with increasing the performance of single core processor
      - High heat dissipation
        - Even capable of melting the processor!
      - High power consumption
        - Power $\propto$ (Frequency)^3
  - Solution (around 2004)
    - Add more cores to achieve better performance instead of increasing the performance of a single core
      - Still maintains the Moore's law
    - Add cache coherency interconnect (CCI) to fetch data on one core from the other core's cache instead of going all the way up to main memory

Von Neumann Architecture & Associated Issues



- Still performance bottleneck
  - Problem
    - Modern applications requires more compute power and data
      - How to improve the performance of a single system?
  - Solution
    - Add more processors (a.k.a. sockets) on the same motherboard

Von Neumann Architecture & Associated Issues

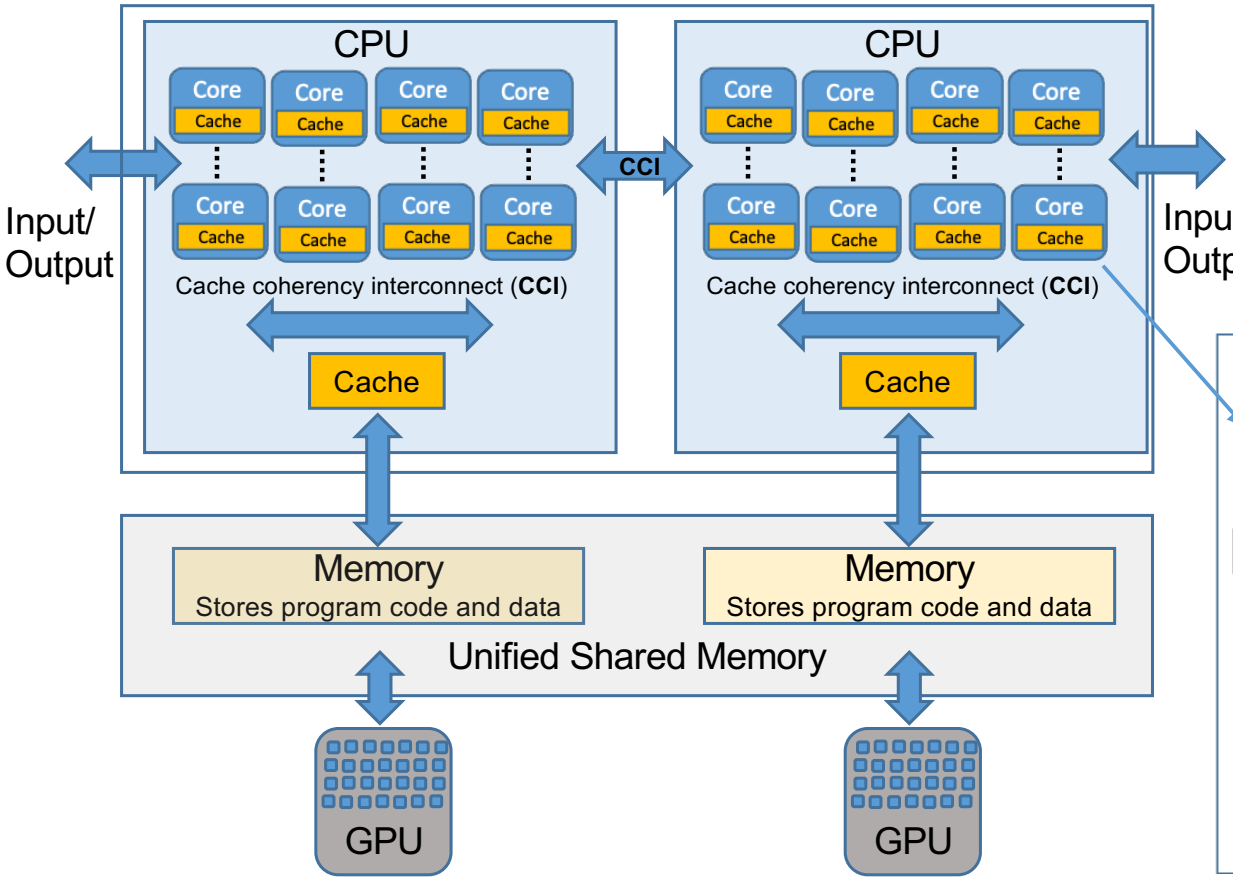


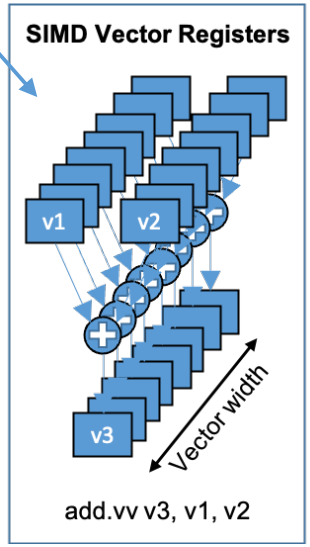
Four socket Intel Cascade Lake processor series

Non-Uniform Memory Access Architecture

- Still performance bottleneck
  - Problem
    - Modern applications requires more compute power and data
      - How to improve the performance of a single system?
  - Solution
    - Add more processors (a.k.a. sockets) on the same motherboard

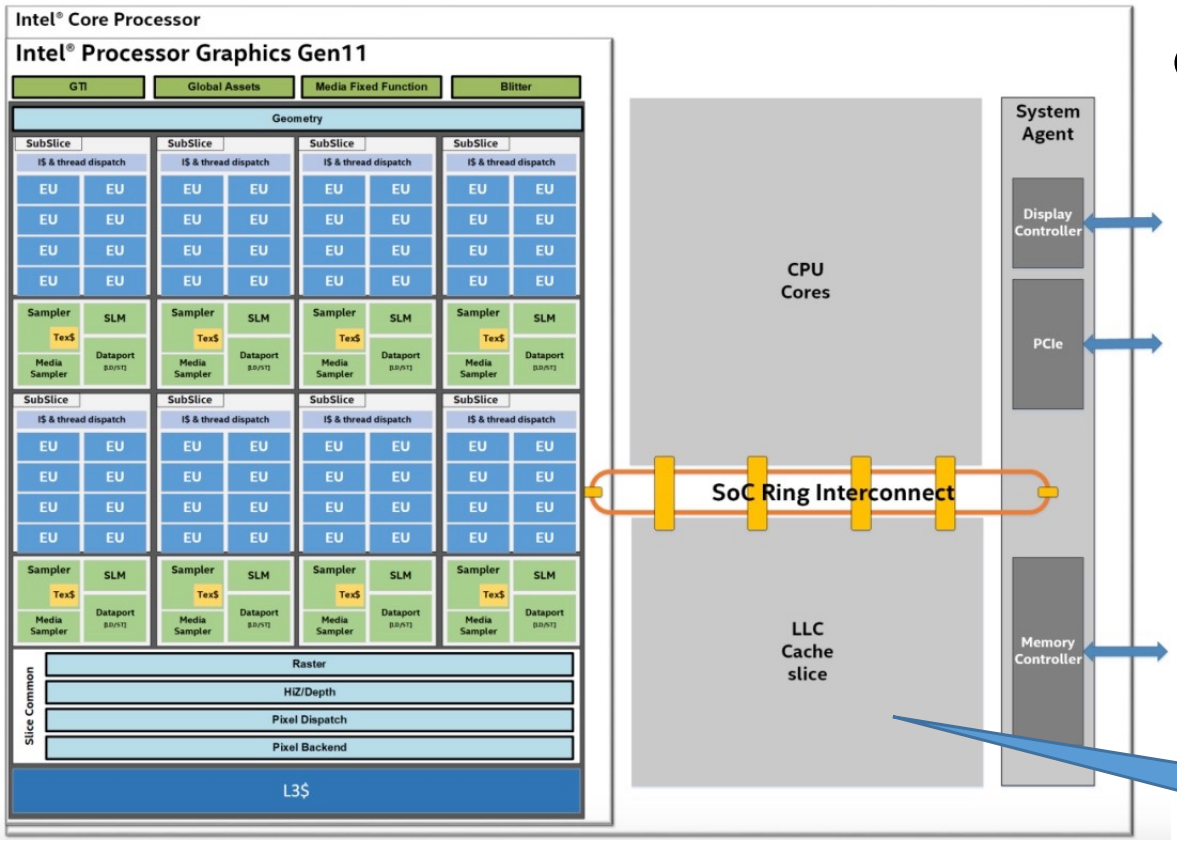
Von Neumann Architecture & Associated Issues





- Still & Still the performance bottleneck
  - Problem
    - Data science / Deep learning / Graphics applications requires more throughput
  - Solution
    - Connect the sockets with GPUs
      - Unified memory architecture that allows sharing the physical memory between CPUs and GPUs

Von Neumann Architecture & Associated Issues

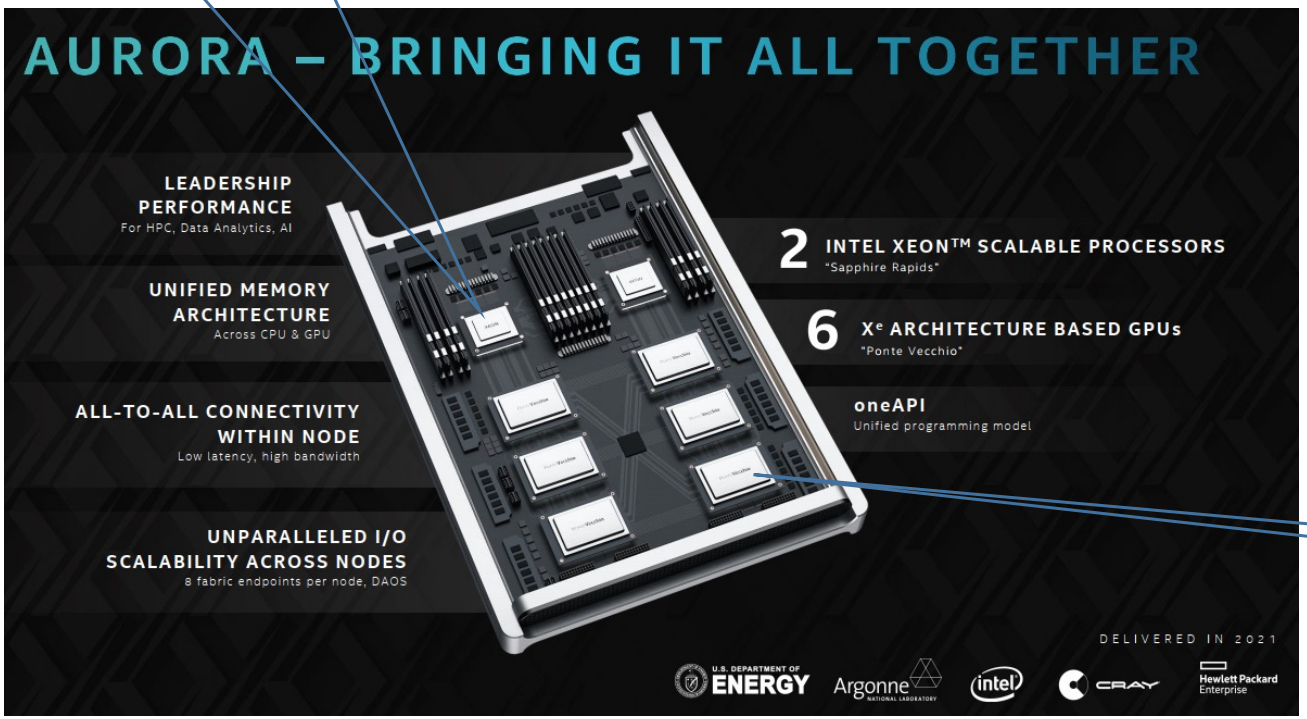


- Still & Still the performance bottleneck
  - Problem
    - Data science / Deep learning / Graphics applications requires more throughput
  - Solution
    - Connect the sockets with GPUs
      - Unified memory architecture that allows sharing the physical memory between CPUs and GPUs
      - No need to copy data across CPU/GPU over PCIe bus

Picture source: https://www.intel.com/content/dam/develop/external/us/en/documents/the-architecture-of-intel-processor-graphics-gen11-r1new.pdf

We are moving into the Exascale Era

More than 100 cores / node

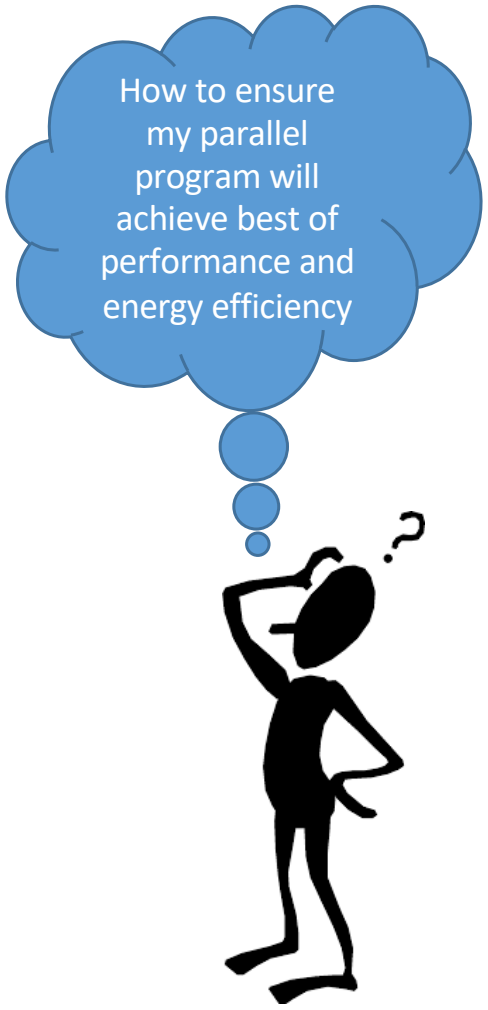


Intel GPUs (6)

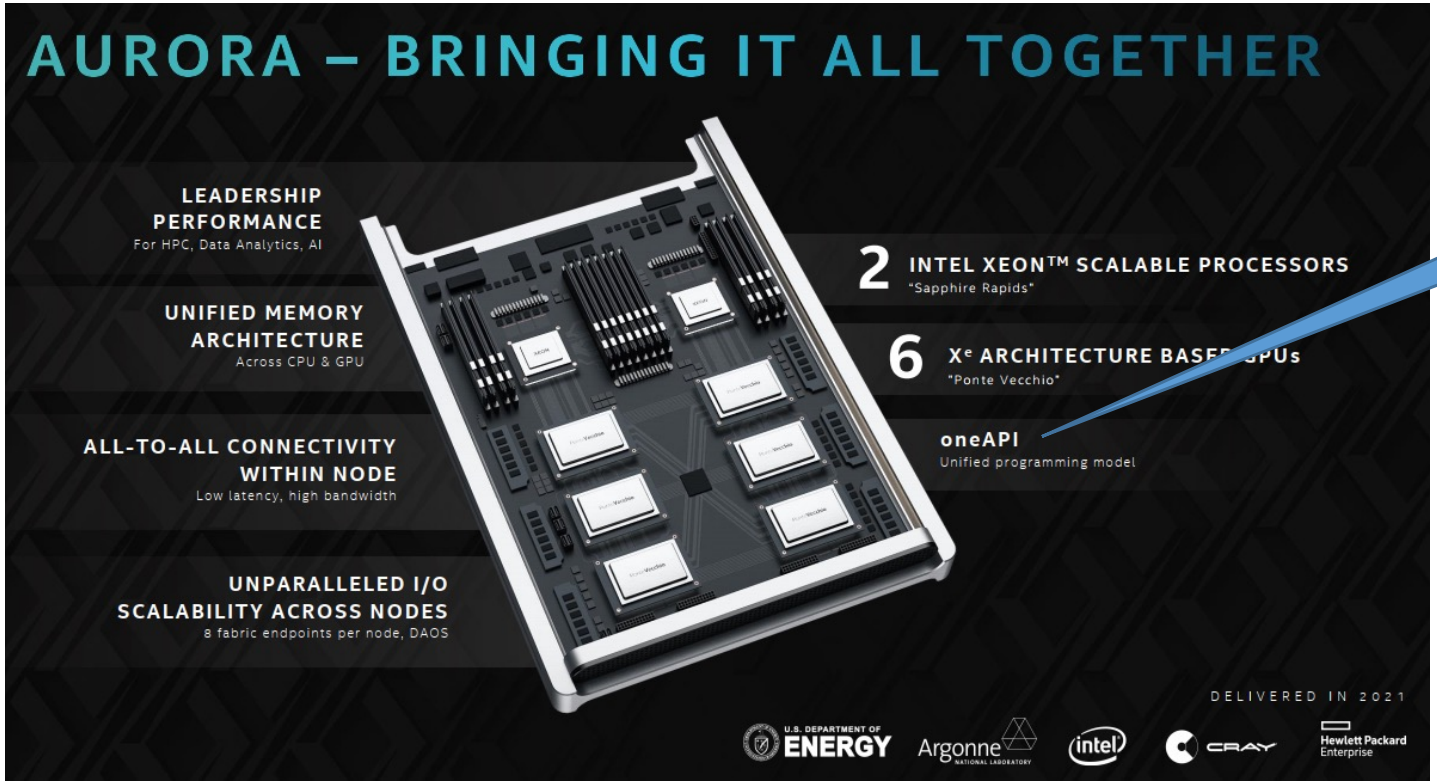
Single node of upcoming Aurora supercomputer at Argonne National Laboratory

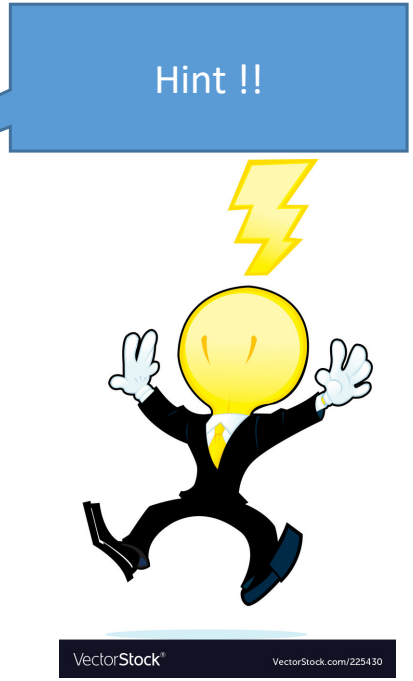
Modern Processors: Summary

- Multiple sockets housed on a single motherboard
- Each socket contains large number of cores
- Cache coherency over entire system (intra/inter socket)
- Deep memory hierarchies (NUMA)
  - Several layers of caches
  - Several DRAMs
- Interfaced with accelerators (e.g., GPU)



Parallel Runtimes for Modern Processors

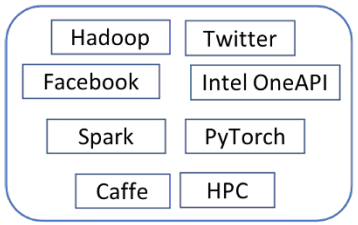


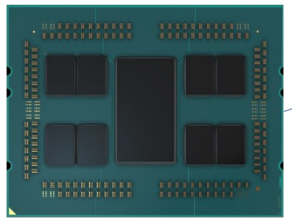


Parallel Runtimes for Modern Processors (PRMP)

PRMP focus, with the content derived from multiple research papers

Improves productivity & performance by abstracting away the hardware complexities





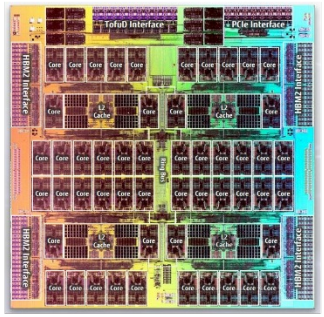
Modern Processor

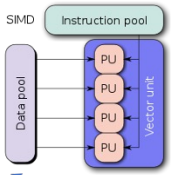
Multicores

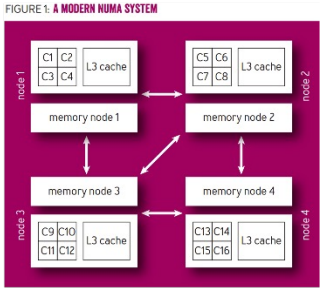
Vector Units

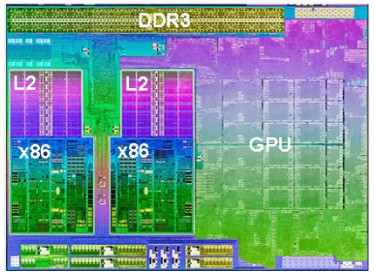
Non Uniform Memory Access

Integrated GPU









Course FAQs

- Theory or programming oriented course?
  - Programming (C/C++) !
- How much of computer architecture?
  - We will only be briefly covering relevant CA/CO theory at the start of some lectures
- How much of load?
  - Moderate, as 60% of marks based on group based activity
- Project details?
  - Constitutes 60% and can be done in a group of two students
  - It will be running throughout the semester with around 4-5 intermediate deadlines
  - You will be implementing a parallel runtime from scratch that provides optimizations/capabilities covered in lecture topics (we will tell you exactly what to implement for each deadline)
- A beginner should take PRMP first or FPP?
  - You can take either of them in any order. In fact you should take both these courses!
- How is PRMP different/similar to FPP course
  - **FPP teach you a variety of parallel programming models** (how to write parallel programs) for shared memory (multicore processors) and distributed memory architectures (clusters) with a brief introduction to parallel runtimes (work-stealing)
  - **PRMP will teach you a variety of runtime techniques** to improve the performance of a single parallel programming model (tasks based programs) over modern processors (**only for shared memory processors**)
  - **Total overlap** of topics taught in PRMP and FPP will be **less than 10%** (common for any parallel computing course that you would encounter)
- Why I should take this course?
  - If you want to get hands-on experience in building systems from scratch
  - If you want to pursue a career in companies working in systems area
  - If you want to improve your C/C++ programming and debugging skills
  - If you want to pursue a research/higher degree in systems area

Course Content*

1. Introduction to variety of computing elements on modern processors (multicore, vector units, GPUs)
   - We will not be teaching OpenCL / CUDA programming. Although, we might use it in a 2-3 lectures, but even then it will be at a basic level
2. Dynamic task parallelism and thread pool runtime for load balancing of these tasks on a multicore processor
   - Automatic management of tasks creation overheads
   - Different types of task queues that could be used in the thread pool, and their respective performance and energy usage
3. Cache coherency on multicore processors
   - Memory consistency models
     - Support by the modern processors
     - How to write processor memory model oblivious programs in C++
   - False sharing
     - Runtime solutions for automatic detection and removing false sharing in parallel programs
4. Runtime for improving the productivity and performance over vector processing units (SIMD)
   - Simultaneous execution over multicores and vector units

** Subjected to minor changes*

Course Content*

4. Runtime solution for trace/replay of asynchronous tasks
5. Non uniform memory access architecture
   - Runtime techniques to improve the performance and productivity
6. Supporting resiliency in a thread pool runtime
10. Performance analysis of parallel programs using hardware performance counters
11. Improving the energy efficiency of parallel programs
    - Exploring a variety of power knobs supported on modern processors
12. Runtime solutions for data race detections and task synchronizations
13. Runtime for hybrid execution of parallel programs over CPU-GPU
    - Automatic tasks partitioning for performance and energy
14. Student based research seminars (2-3 lectures)

** Subjected to minor changes*

Course Evaluation

- Quiz: 10% (N-1 policy)
  - No surprise quizzes
  - Will be held during lecture hours (around 20mins duration)
- Mid semester exam: 10%
- End semester exam: 20%
- Group project: 60%
  - Will start at end of the week-2
  - **Form your groups ASAP**
    - We won't be helping in group formation
    - Group of two students only

Course Prerequisites

- **Programming in C/C++ is a must!**
  - If you don't know C/C++ then you should be confident that you can pick it up on your own
- Basics of Operating Systems

We will strictly follow the IIITD **plagiarism policy**. No excuses if you get caught in plagiarism

Reference Materials

- Course material derived from multiple sources
- Course notes / references will be provided depending on the lecture
- References will also be mentioned on the last slide in each lecture
- Relevant text books will be mentioned during the lectures

Course Logistics

- Course webpage
  - https://classroom.google.com/c/NDk3NzE0NTg5NDEy?cjc=pip63kp
- Machines for your project work
  - You can use your laptop for the intermediate deadlines
    - Would require Linux OS (or any VM running Linux OS)
  - We would provide you multicore servers for final deadline experiments
- In case you have problems please feel free to shoot me an email

Next Class

- Introduction to parallel programming and runtime systems