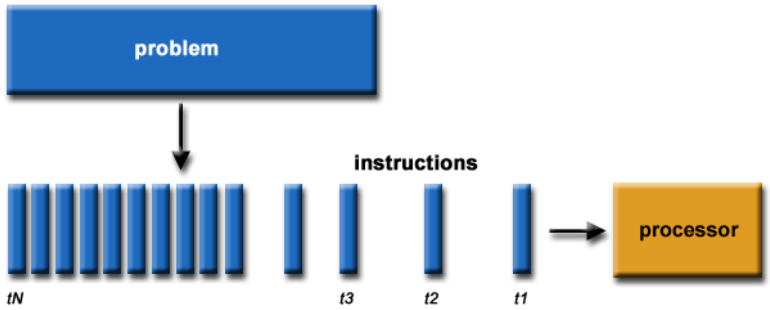
Lecture 01: Course Introduction

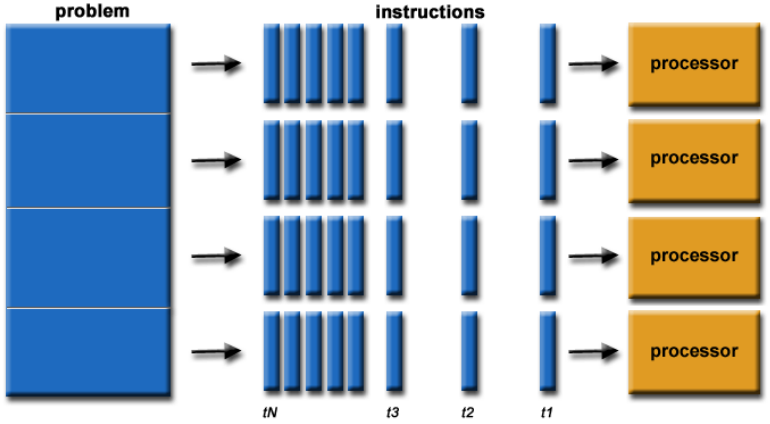
Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

© Vivek Kumar

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

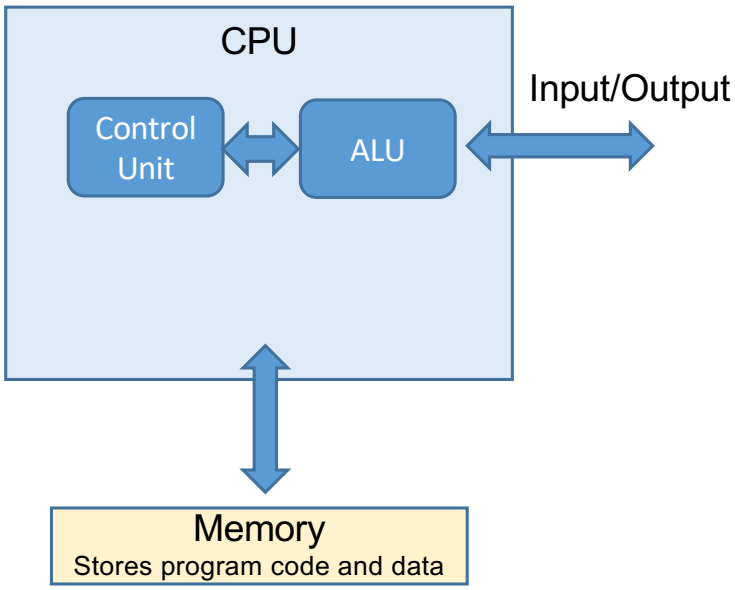
Motivations for Parallel Programming

- Technology push
- Application push

Technology Push for Parallel Programming

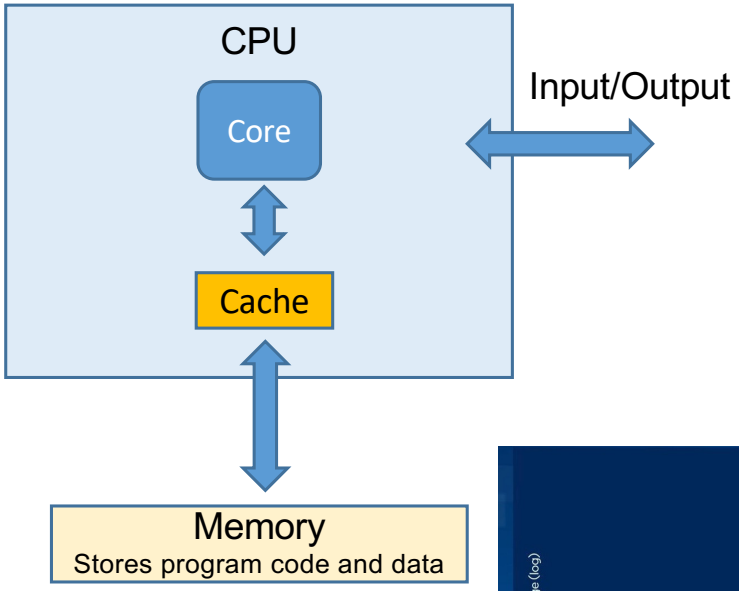
Let us try to understand why the processors are becoming complex?

Von Neumann Architecture

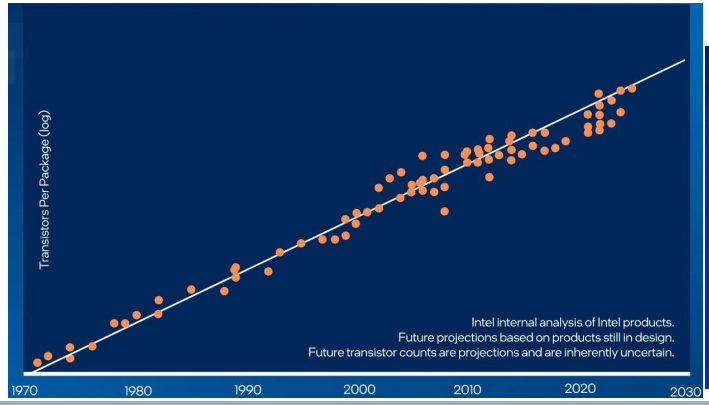


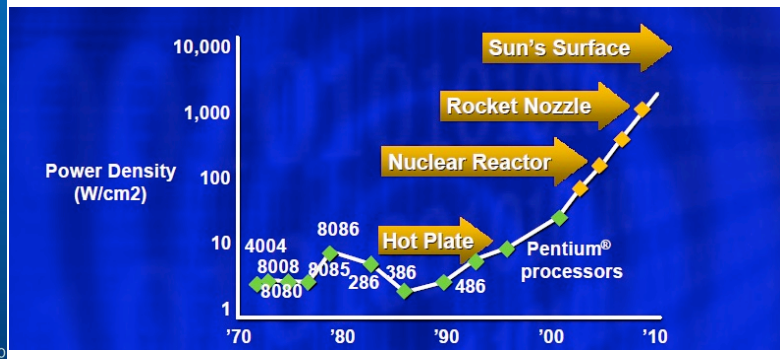
- John Von Neuman in 1945 came up with the architecture for computers that we even use today (albeit with several changes)
- Problem
  - Memory bottleneck
    - Access latency to memory quite high
      - High CPU stalls while fetching code and data

Von Neumann Architecture & Associated Issues

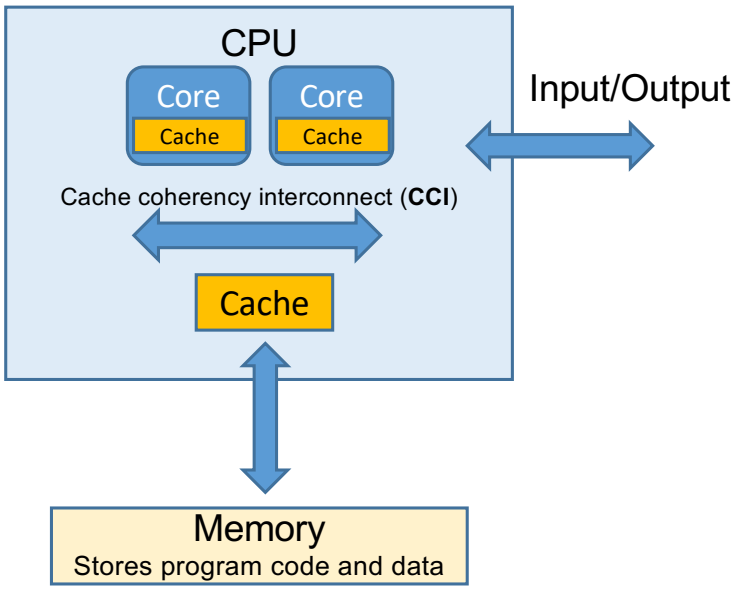


- Memory bottleneck
  - Solution
    - Add cache on the CPU chip to store frequently accessed memory
- Next problem
  - Performance bottleneck
    - Increasing the performance of the processor by adding more and more transistors resulting in high heat dissipation
      - Even capable of melting the processor!



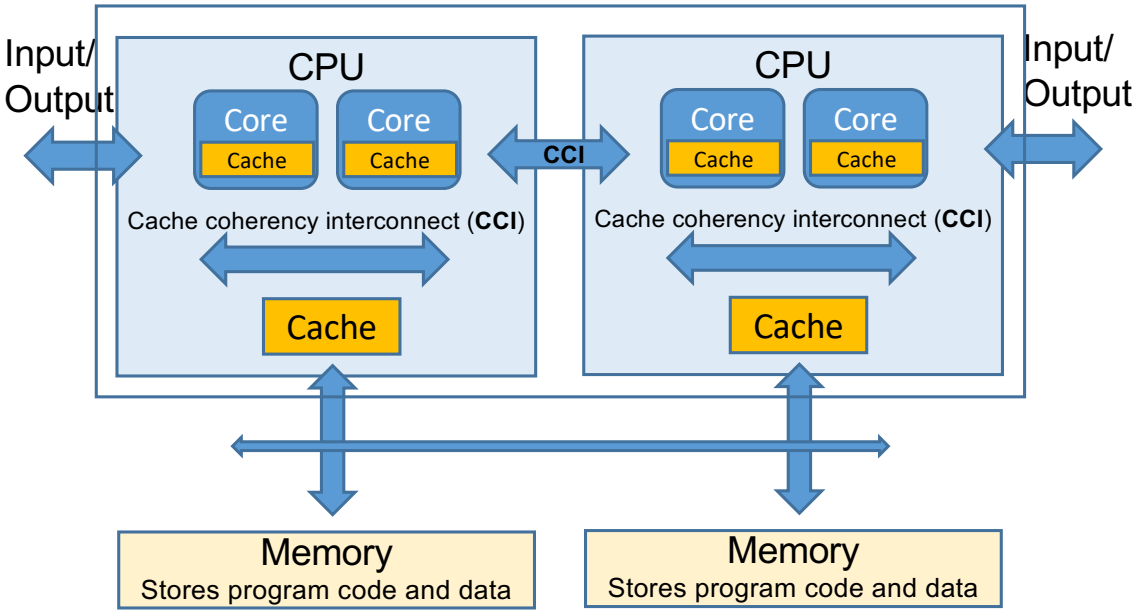


Von Neumann Architecture & Associated Issues



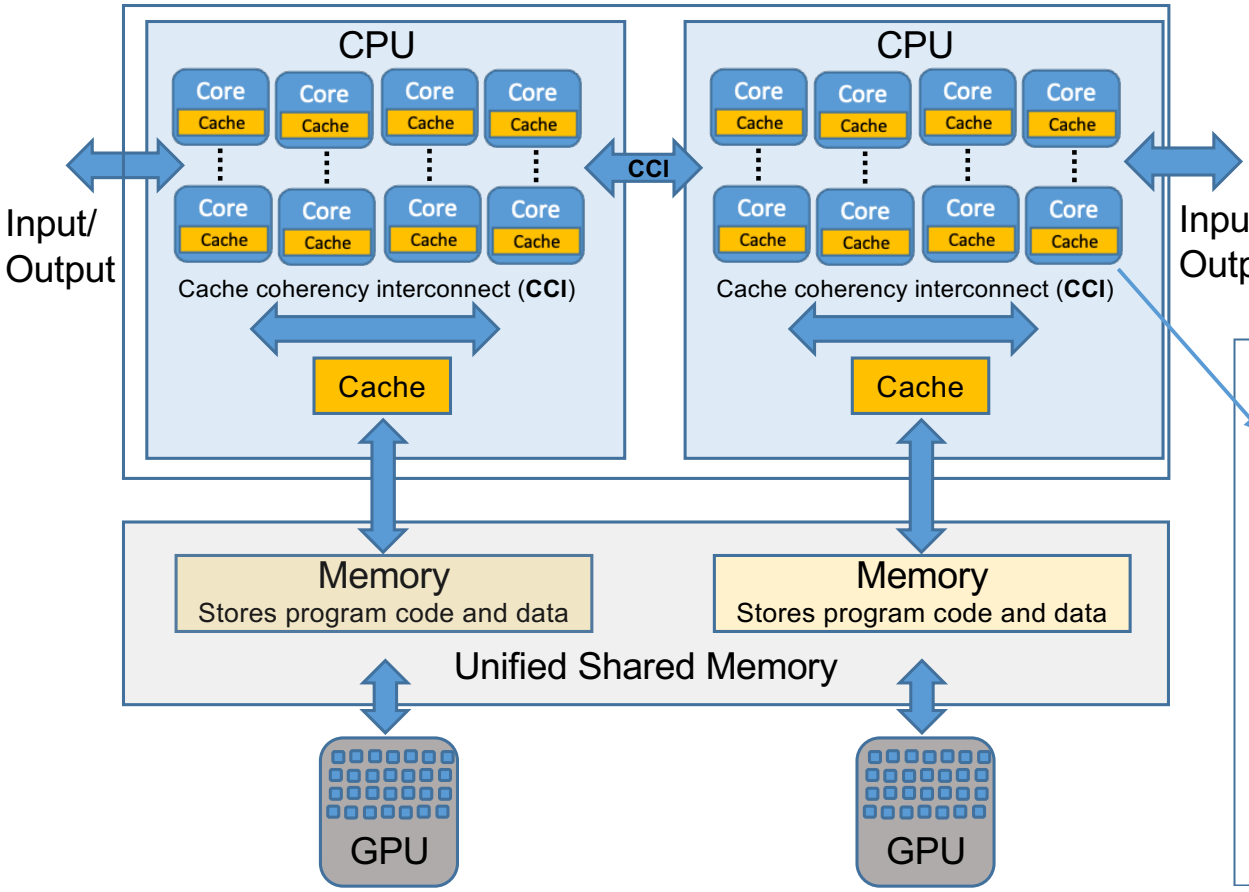
- Performance bottleneck
  - Solution (around 2004)
    - Add more cores to achieve better performance instead of increasing the performance of a single core
    - Add cache coherency interconnect (CCI) to fetch data on one core from the other core's cache instead of going all the way up to main memory
- Next problem
  - Modern applications requires more **performance** and **data**
    - How to improve the performance and data storage of a single system?

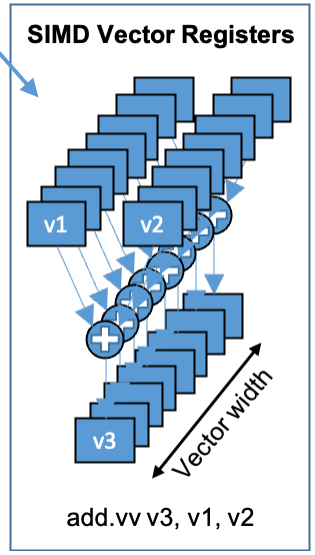
Von Neumann Architecture & Associated Issues



- Performance bottleneck
  - Solution
    - Add more processors (a.k.a. sockets) on the same motherboard
    - Provide local memory banks at each socket
      - Each CPU can access local and remote memory banks
        - Non Uniform Memory Access latency
- Next problem
  - Data science / Deep learning / Graphics applications requires more throughput

Von Neumann Architecture & Associated Issues

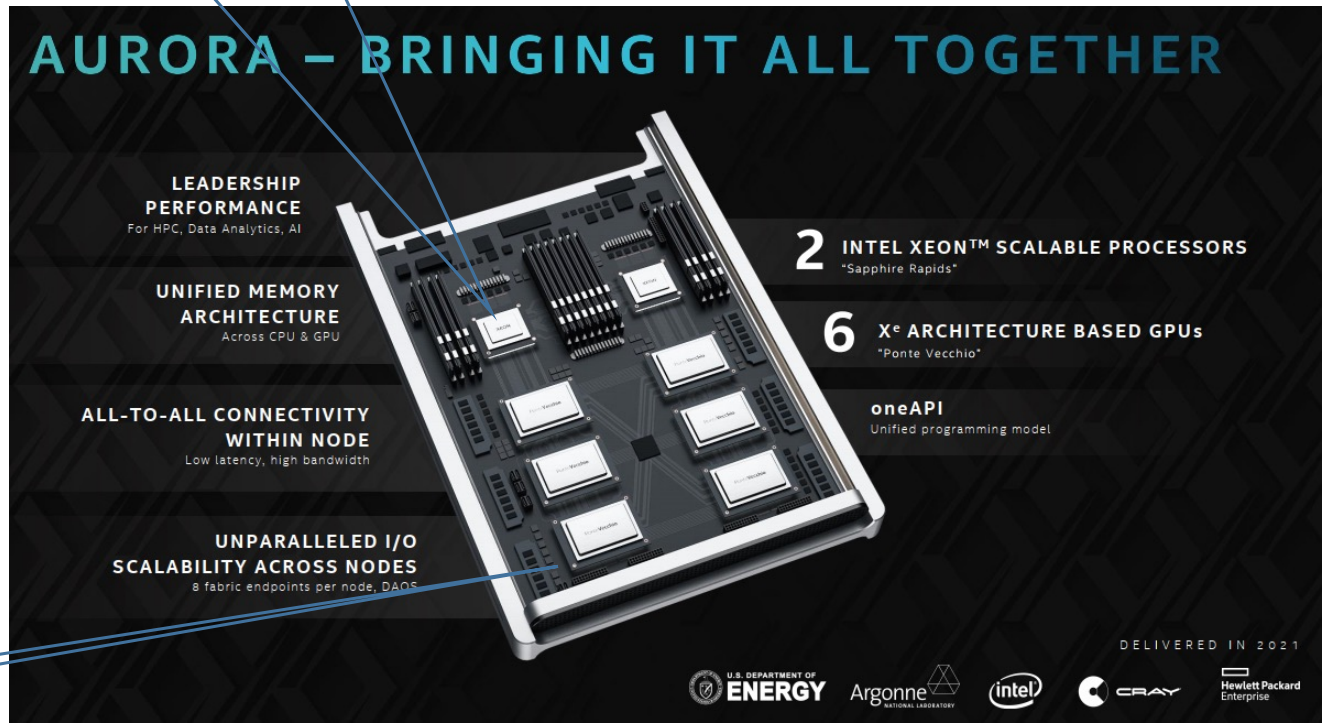




- Solution
  - Connect the sockets with GPUs
    - Unified memory architecture that allows sharing the physical memory between CPUs and GPUs

We are Currently in the Exascale Era

More than 100 cores / node



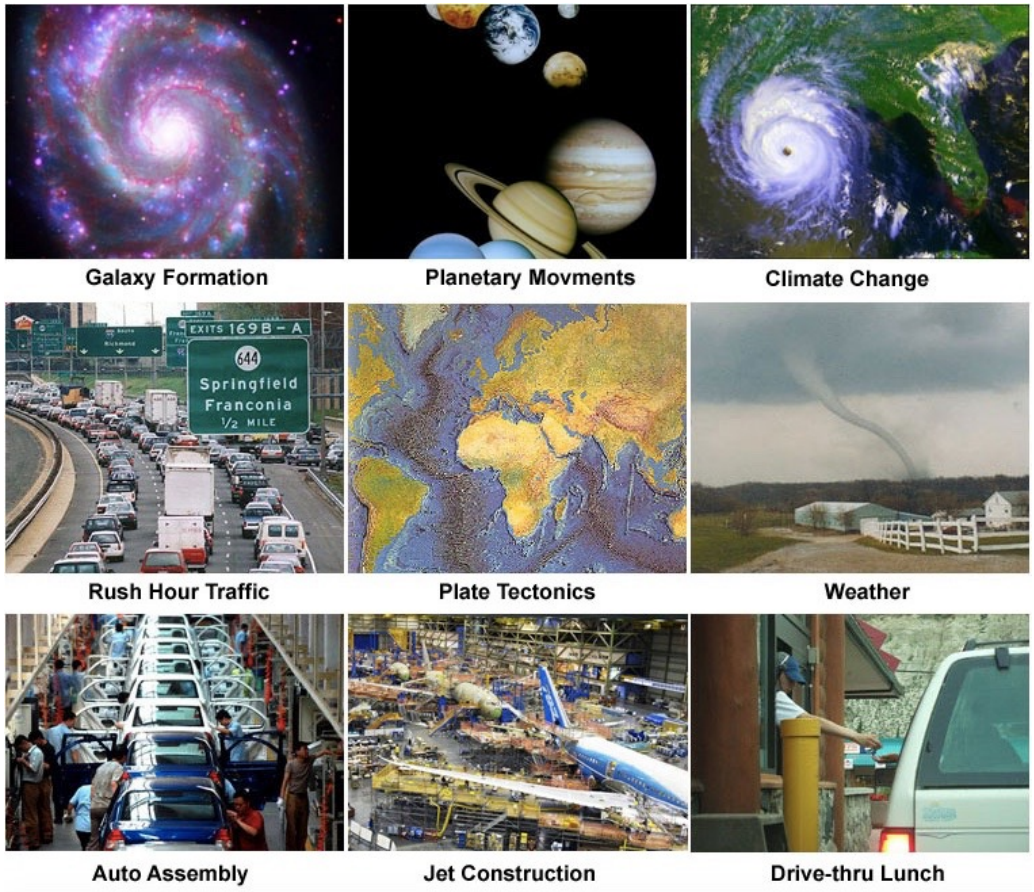
Intel GPUs (6)

Single node of Aurora supercomputer at Argonne National Laboratory

Technology Push: Summary

- Modern processors have become quite complex
  - 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 Programming: Application Push



ChatGPT

Gemini

PyTorch

TensorFlow

Parallel Programming Dilemma

```
uint64_t array_sum(uint64_t* array, uint64_t size) {
  uint64_t sum = 0;
  for(uint64_t i=0; i<size; i++) {
    sum = sum + array[i];
  }
  return sum;
}
```

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}
```

- How to convert a sequential program into parallel program with minimal effort
- How to customize the parallel program for the underlying processor such that it achieves the best performance

Parallel Programming Dilemma

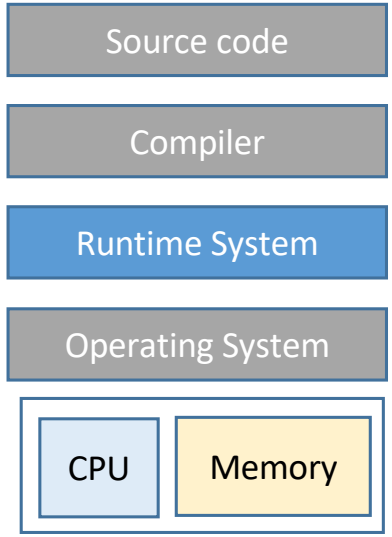
```
uint64_t array_sum(uint64_t* array, uint64_t size) {
  uint64_t sum = 0;
  for(uint64_t i=0; i<size; i++) {
    sum = sum + array[i];
  }
  return sum;
}
```

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}
```

- Solution
  - Write parallel program once but run it anywhere
  - How?
    - With the help of a **Runtime System**

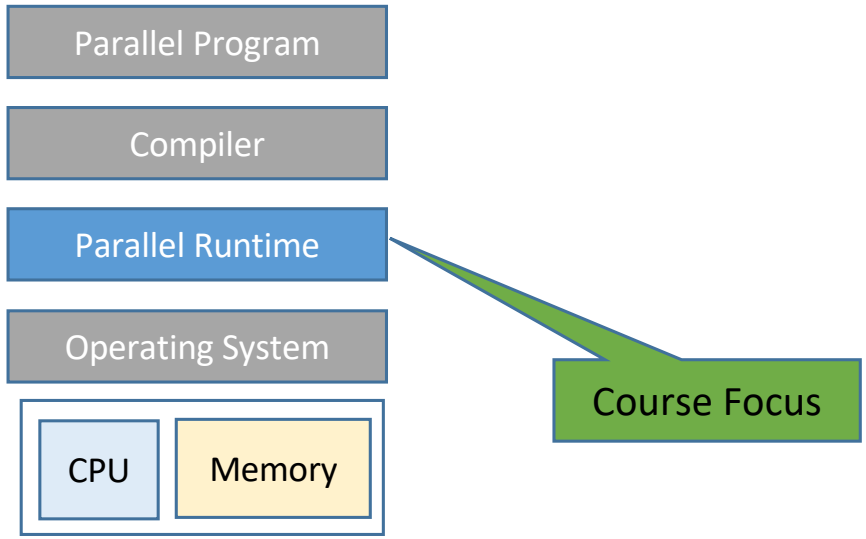
What are Runtime Systems?

- 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 (e.g., GNU C library)

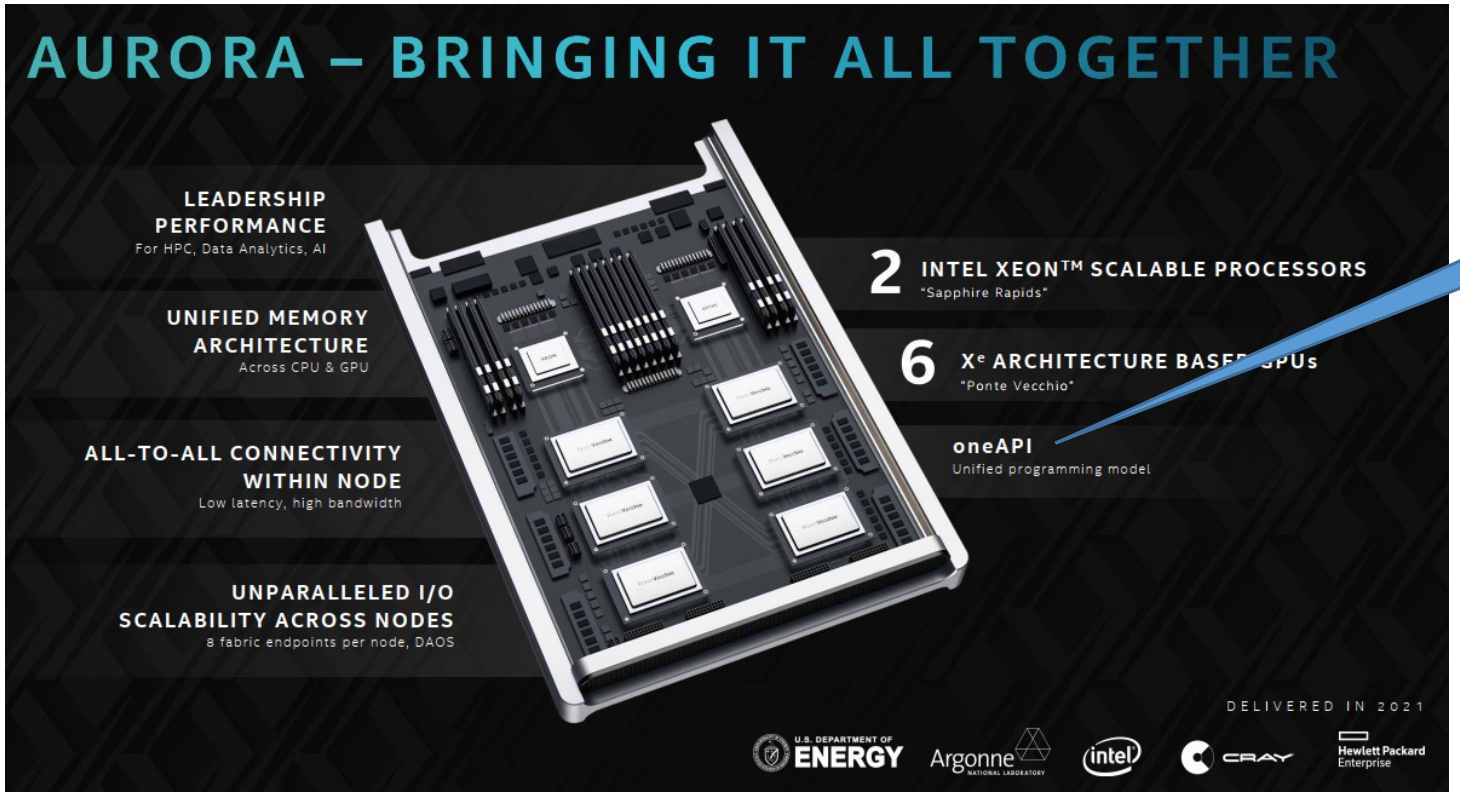


What are Parallel Runtimes?

- Runtime systems to manage the execution of a parallel program over multiple compute resources
  - Abstracts away the challenges with the modern hardware



Parallel Runtimes for Modern Processors



An example of a parallel runtime

Parallel Runtimes for Modern Processors

- Modern processors have become quite complex
  - Deep memory hierarchies (NUMA)
    - Several layers of caches

Improves the performance by abstracting away the hardware complexities

PRMP focus on the solutions to these problems with the content derived from multiple research papers

Course FAQs

- Theory or programming oriented course?
  - **Purely programming (C/C++) !**
- Project details?
  - Constitutes 50% and can be done in a group of two students
  - It will be running throughout the semester with five intermediate deadlines
  - You will be implementing a parallel runtime that provides optimizations/capabilities covered in lecture topics (we will tell you exactly what to implement for each deadline)
- 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
- How much of load?
  - Moderate, as 50% of marks based on group based activity
- How much of computer architecture?
  - We will only be briefly covering relevant CA/CO theory at the start of some lectures

Course Evaluation

- Quiz: 10%
  - Total 4-5 quizzes throughout the semester
  - Will be held during lecture hours (around 20mins duration)
  - (N-1) policy to help you in case you miss any quizzes due to unforeseen events (e.g., medical issue, laziness, etc.)
- Group project: 50%
  - Total **5** deadlines throughout the semester on runtime implementation, and one in the last on paper presentation
  - **Form your groups ASAP**
    - Latest by Monday 13th January
    - We won't be helping in group formation
    - Group of one or two students only
      - Same rubrics/marking scheme irrespective of single/two member group size
- Semester exams: 20% + 20%
- Bonus
  - Project deliverables will have some bonus components
  - Extra 2% bonus on preparing lecture summary notes in a well formatted/readable format
    - **You are allowed to prepare notes only for lectures you were not absent**

Important Information

1. Please don't open-source project implementations even after the course gets over
   - You can't host it on your GitHub public repo
2. We will not upload mid/end semester solution/rubric on Google Classroom
   - Although, we will discuss it in class
3. I will **not** be marking attendance but if you miss the lecture then you are on your own!
   - We will not provide the lecture recordings

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

Next Class

- Introduction to parallel programming and runtime systems