**Activity 1: UMA vs. NUMA Showdown**

**Objective:** Distinguish between UMA (Uniform Memory Access) and NUMA (Non-Uniform Memory Access)

* **Instructions:**
  + Divide the class into two groups (UMA vs. NUMA).
  + Each group lists **5 pros and cons** of their architecture.
  + Groups then present and debate their findings.
* **Solution/Expected Outcomes:**
  + **UMA Pros:** Simpler programming, equal access times.
  + **UMA Cons:** Not scalable, memory bottlenecks.
  + **NUMA Pros:** Scalable, localized memory access is faster.
  + **NUMA Cons:** Complex programming, non-uniform access delays.

**Activity 2: Human SIMD vs. MIMD Simulation**

**Objective:** Understand the difference between SIMD (Single Instruction Multiple Data) and MIMD (Multiple Instruction Multiple Data)

* **Instructions:**
  + Divide the class into two groups.
  + Give each group a task:
    - **SIMD group:** All students perform the **same operation** (e.g., add 5) on different numbers.
    - **MIMD group:** Each student performs a **different operation** (e.g., one adds, one multiplies, etc.) on different numbers.
  + Observe and discuss differences in flexibility and coordination.
* **Solution/Expected Outcomes:**
  + SIMD is efficient for identical operations on large data (e.g., vector processing).
  + MIMD allows task diversity (e.g., multi-core CPU tasks), suitable for general-purpose computing.

**Activity 3: Memory Access Race Game**

**Objective:** Simulate and analyze access in shared address space platforms

* **Instructions:**
  + Give all groups a “memory board” with a list of memory addresses and values.
  + Each group simulates multiple processors accessing memory locations (read/write).
  + Introduce contention by asking multiple processors to access the same address.
* **Solution/Expected Outcomes:**
  + Students will observe **race conditions**, **synchronization needs**, and how **shared address space** leads to challenges with consistency and latency.

**Activity 4: Communication Model Sketch Challenge**

**Objective:** Compare communication models (shared memory vs. message passing)

* **Instructions:**
  + Each group draws a diagram of one communication model (e.g., Shared Memory or Message Passing).
  + Label components (processors, memory, network), and annotate key characteristics.
  + Present the diagram to the class.
* **Solution/Expected Outcomes:**
  + **Shared memory:** Centralized memory, fast access, requires synchronization.
  + **Message passing:** Distributed memory, explicit communication, scalable but complex.

**Activity 5: Match the Platform**

**Objective:** Match real-world systems or use-cases to UMA, NUMA, SIMD, MIMD

* **Instructions:**
  + Give each group a list of real-world systems/applications (e.g., GPU, multi-core CPU, Cray supercomputer, AWS EC2).
  + Groups classify each as UMA, NUMA, SIMD, or MIMD.
  + Justify their classification.
* **Solution/Expected Outcomes:**
  + **GPU:** SIMD
  + **Multi-core CPU (with shared cache):** UMA
  + **NUMA-enabled server (e.g., AMD EPYC):** NUMA, MIMD
  + **AWS EC2 distributed VMs:** Message Passing, MIMD

Let me know if you'd like printable handouts or slides for these activities!

**You said:**

some difficult tasks

**ChatGPT said:**

Great! For more **advanced group activities**, we'll focus on **deep analysis, performance implications, and system design trade-offs** in **shared address space platforms, UMA/NUMA, SIMD/MIMD, and communication models**. These tasks are suitable for more experienced or senior students.

Here are **5 difficult 10-minute group tasks** with **solutions or guidance**:

**Activity 1: Design a NUMA-Optimized Application Flow**

**Objective:** Understand the implications of NUMA architecture on software design.

* **Task:**
  + Each group is given a basic multithreaded application scenario (e.g., parallel matrix multiplication).
  + Design how memory and threads should be allocated to **minimize remote memory access**.
  + Sketch the system layout and explain optimization decisions.
* **Expected Outcome:**
  + Use **thread and memory affinity** (e.g., bind threads to local memory banks).
  + Place data near the threads that use them most.
  + Avoid unnecessary data sharing between NUMA nodes.

**Activity 2: SIMD Performance Limitations Analysis**

**Objective:** Explore when SIMD **fails to deliver performance gains**.

* **Task:**
  + Groups are given different code scenarios (e.g., loop with branching, data dependencies, irregular memory access).
  + Identify which cases are unsuitable for SIMD and explain why.
* **Expected Outcome:**
  + Branching and divergent control flow hurt SIMD.
  + Data dependencies prevent instruction-level parallelism.
  + Misaligned or irregular memory accesses reduce SIMD throughput.

**Activity 3: Model and Compare Parallel Communication Overheads**

**Objective:** Analyze communication cost in shared memory vs. message passing.

* **Task:**
  + Groups are given a small parallel task (e.g., computing dot product across multiple cores).
  + Estimate communication costs using:
    - **Shared memory** (including lock contention, cache coherence)
    - **Message passing** (data transfer time, synchronization)
  + Compare and argue which model is more efficient and under what conditions.
* **Expected Outcome:**
  + **Shared memory:** Faster but affected by contention and coherence traffic.
  + **Message passing:** Scalable and clean, but with latency and setup overhead.
  + Answer depends on number of processes and interconnect topology.

**Activity 4: Design a Parallel System with Hybrid SIMD and MIMD**

**Objective:** Combine SIMD and MIMD in system design.

* **Task:**
  + Design a **parallel system architecture** that utilizes both SIMD and MIMD effectively.
  + Choose a real-world application (e.g., image recognition or scientific simulation).
  + Assign which part of the workload should go to SIMD and which to MIMD.
* **Expected Outcome:**
  + Example: Image processing pipeline:
    - SIMD handles pixel-wise operations (e.g., filters).
    - MIMD manages control flow, scheduling, and I/O.
  + Shows hybrid architectures like CPU+GPU or vector+scalar cores.

**Activity 5: Analyze False Sharing and Cache Coherency**

**Objective:** Understand **false sharing** and its impact in shared memory.

* **Task:**
  + Provide a code snippet with shared variables placed adjacently in memory.
  + Groups must detect potential false sharing issues.
  + Suggest code or memory layout changes to mitigate it.
* **Expected Outcome:**
  + **Problem:** Two threads write to different variables on the same cache line.
  + **Solution:** Pad variables to occupy separate cache lines.
  + Discuss impact on performance and cache coherence traffic.