

Course Code	Course Title	L	T	P	Cr
23CS302PC303	High Performance Computing	3	0	4	5
Program Core					

Unit -2: Shared Memory Programming with OpenMP

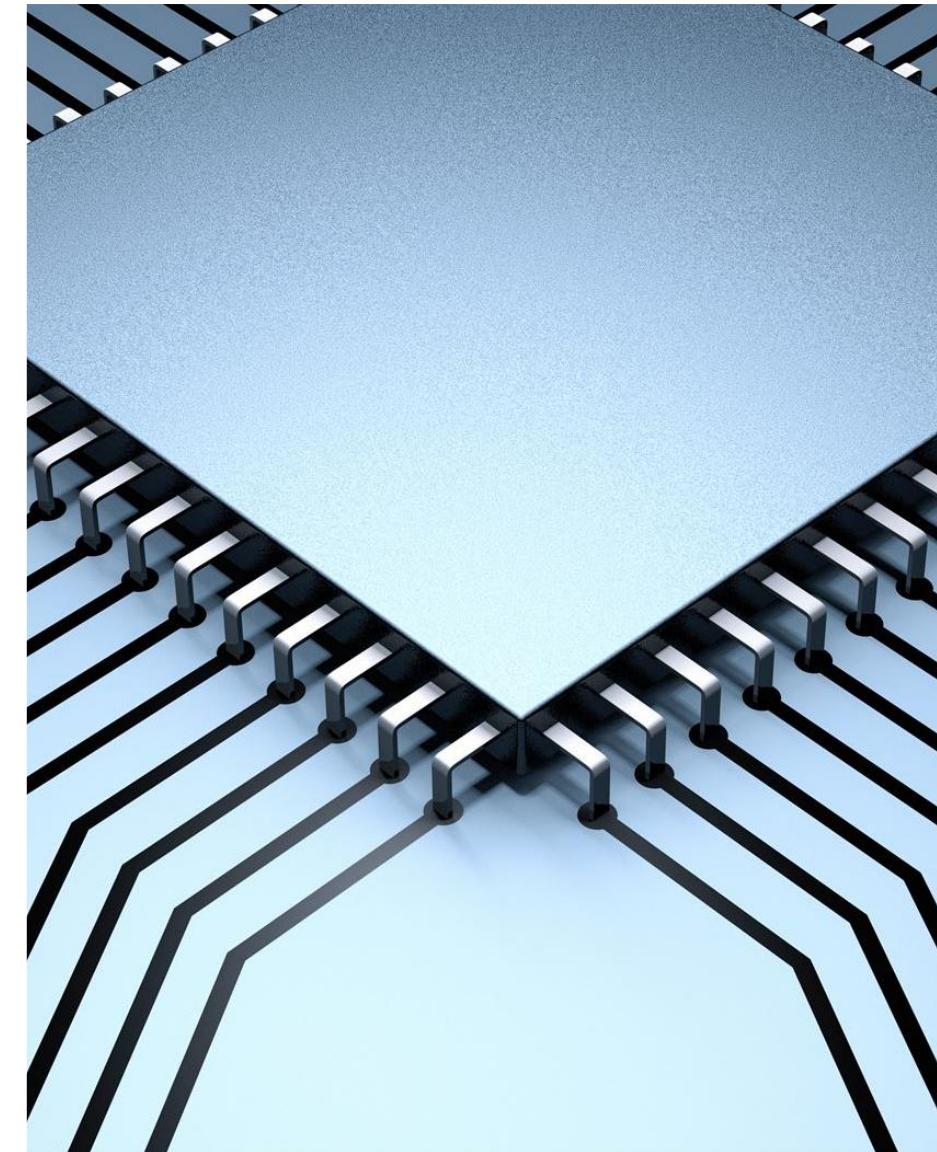
Lesson 12

1. Open MP
2. Work Sharing Constructs:
3. Data Environment:
- 4 Synchronization
5. Tasking with Open MP
6. NUMA Architectues and Open MP:
- 7.Vectorization with Open MP SIMD:
8. Open MP Performance Analysis and Optimization

**Dr. Sri Raman Kothuri
Associate Professor**

Optimizing parallel computing for performance and efficiency

NUMA Architectures and
OpenMP: Affinity, Memory
Placement, Bandwidth, and
STREAM Benchmark



Overview of NUMA in High Performance Computing

NUMA Architecture Basics

NUMA provides each processor with local memory, reducing contention and improving scalability over UMA.

Performance Implications

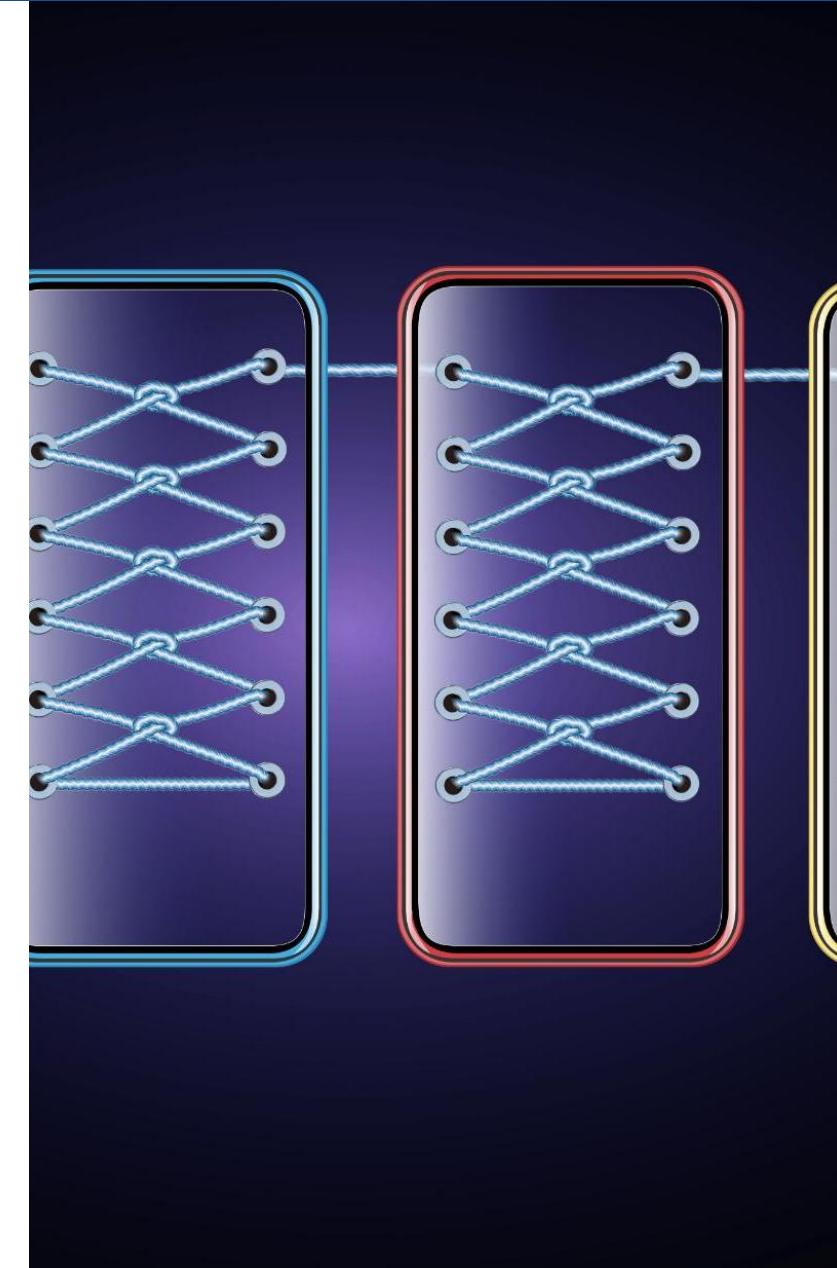
Local memory access latency is minimized, while remote access latency is higher, affecting system performance.

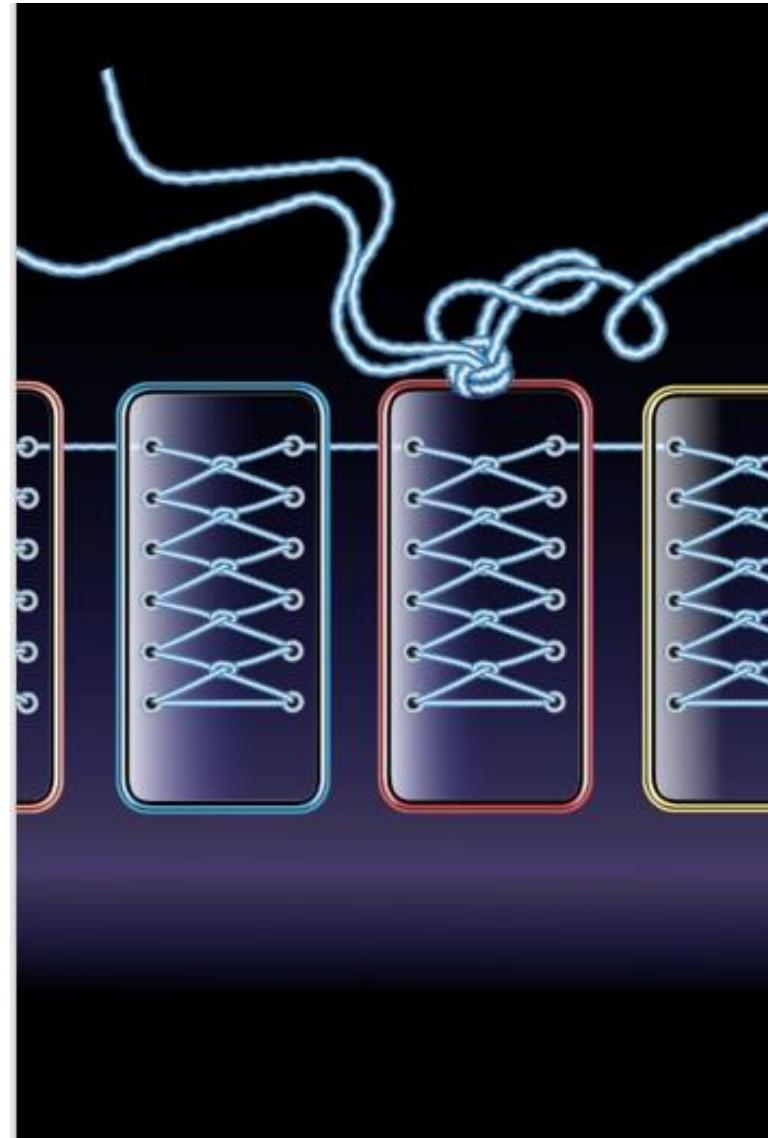
NUMA-aware Programming

Optimizing thread and data placement reduces costly remote memory accesses and improves efficiency.

Use in Modern Systems

NUMA is widely used in servers and supercomputers to achieve peak performance with multiple sockets and cores.





Memory Access Latency

UMA has uniform memory access latency while NUMA provides faster local memory access and slower remote access.

Scalability Differences

NUMA partitions memory across processor nodes allowing better scalability compared to UMA's single memory pool bottleneck.

Programming Complexity

NUMA requires memory locality awareness and optimizations like thread affinity to avoid performance degradation.

Use in HPC Systems

NUMA is preferred in high-performance computing to minimize memory delays and maximize bandwidth utilization.

Thread Affinity in OpenMP

Definition of Thread Affinity

Thread affinity binds threads to specific processors or cores to optimize performance in NUMA systems.

OpenMP Affinity Controls

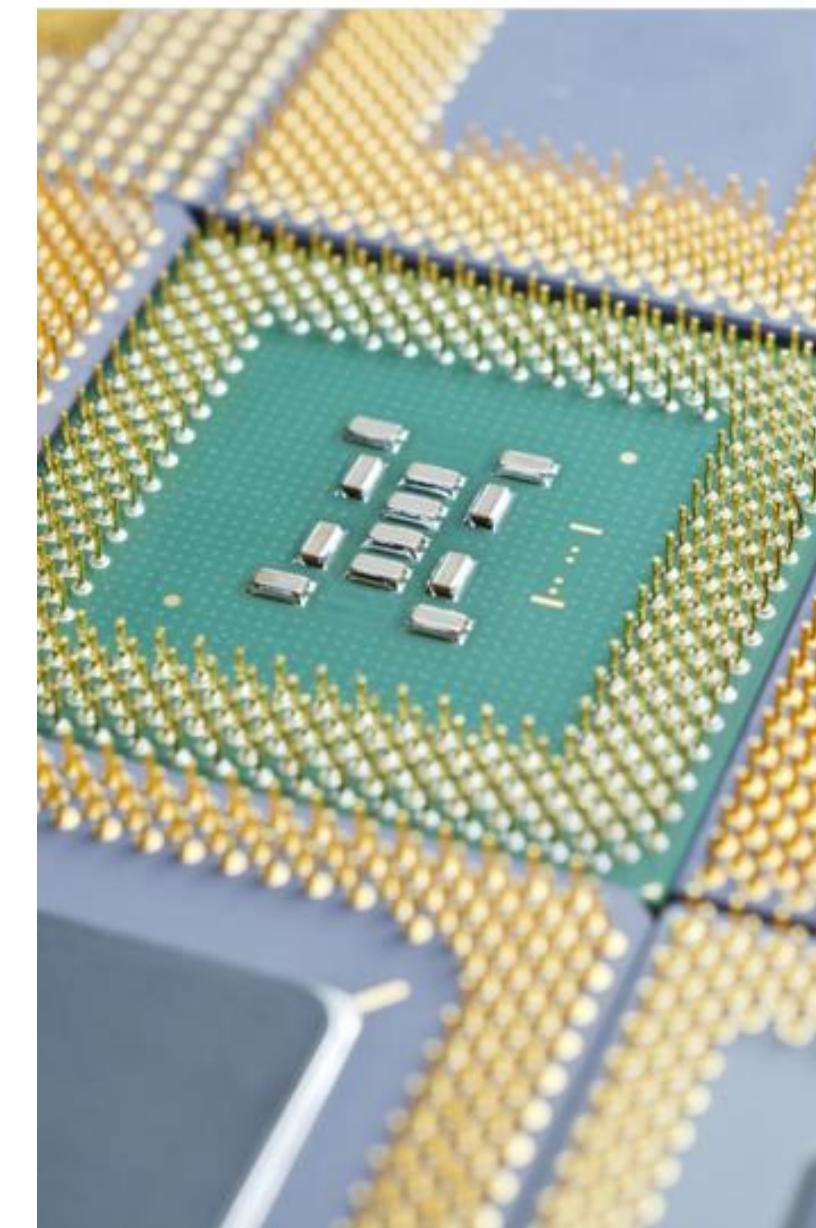
OpenMP uses environment variables like OMP_PROC_BIND and OMP_PLACES to control thread placement across cores and sockets.

Performance Benefits

Proper affinity reduces remote memory access and improves cache utilization, resulting in predictable NUMA performance.

NUMA Architecture Exploitation

Aligning thread execution with memory placement strategies enables effective use of NUMA architecture.





Affinity's Role in Performance

Proper affinity settings keep threads close to their data, reducing latency and improving cache hit rates.

Consequences of Poor Affinity

Poor affinity causes frequent remote memory accesses, increasing latency and lowering bandwidth utilization.

Tools for Affinity Management

Tools like **numactl** and **hwloc** help visualize and manage affinity, optimizing thread placement and memory binding.

Affinity in Memory-Bound Apps

Affinity optimization yields significant performance gains in memory-bound applications by enhancing data locality.

First Touch Policy

Definition of First Touch

The policy allocates memory pages on the NUMA node closest to the thread that first accesses them.

Importance of Data Locality

Allocating memory near the accessing thread improves data locality and reduces latency in memory access.

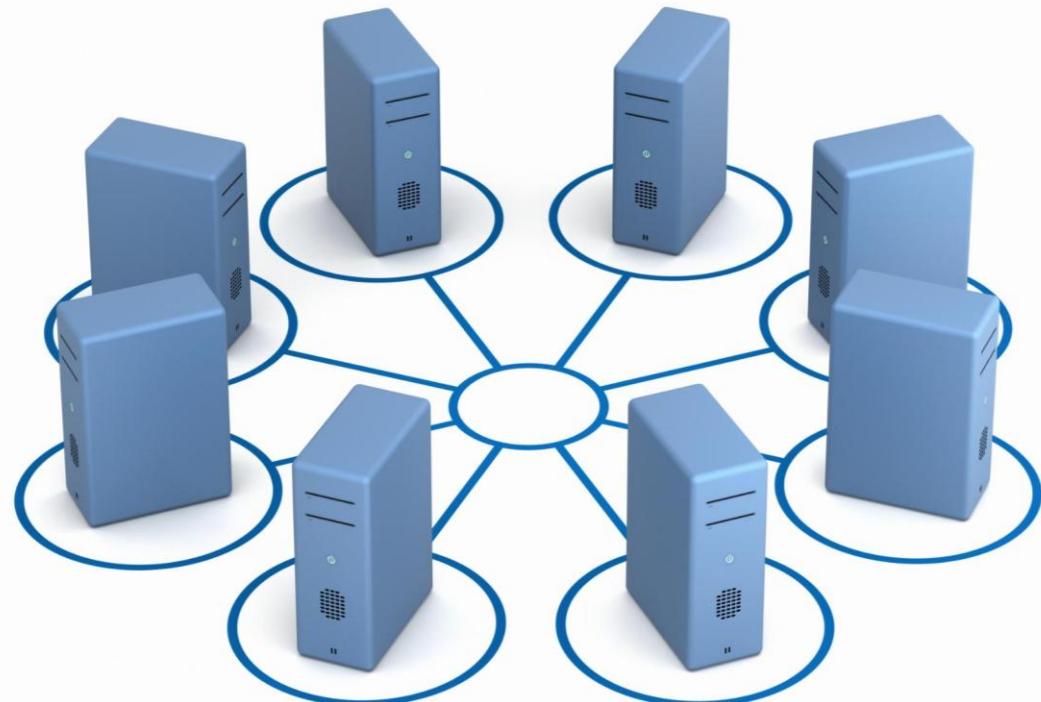
Developer Considerations

Initialization loops must be carefully structured to ensure correct thread touches memory first for optimal placement.

Application in OpenMP

OpenMP applications rely on first touch policy for effective parallel initialization and memory distribution.





Impact of Memory Placement

Local memory access offers higher bandwidth and lower latency than remote memory access across interconnect links.

Interconnect Bandwidth Saturation

Poor memory placement can saturate interconnect bandwidth, causing contention and reduced system performance.

Optimization and Profiling Tools

Tools like Intel VTune and Linux perf help identify bandwidth bottlenecks and guide memory optimization strategies.

Maximizing Throughput in HPC

Effective bandwidth management is essential for high throughput in HPC workloads with large datasets and frequent memory operations.

Purpose and Methodology of STREAM Benchmark

Benchmark Purpose

STREAM measures sustainable memory bandwidth in high-performance computing environments to evaluate system efficiency.

Vector Operations Tested

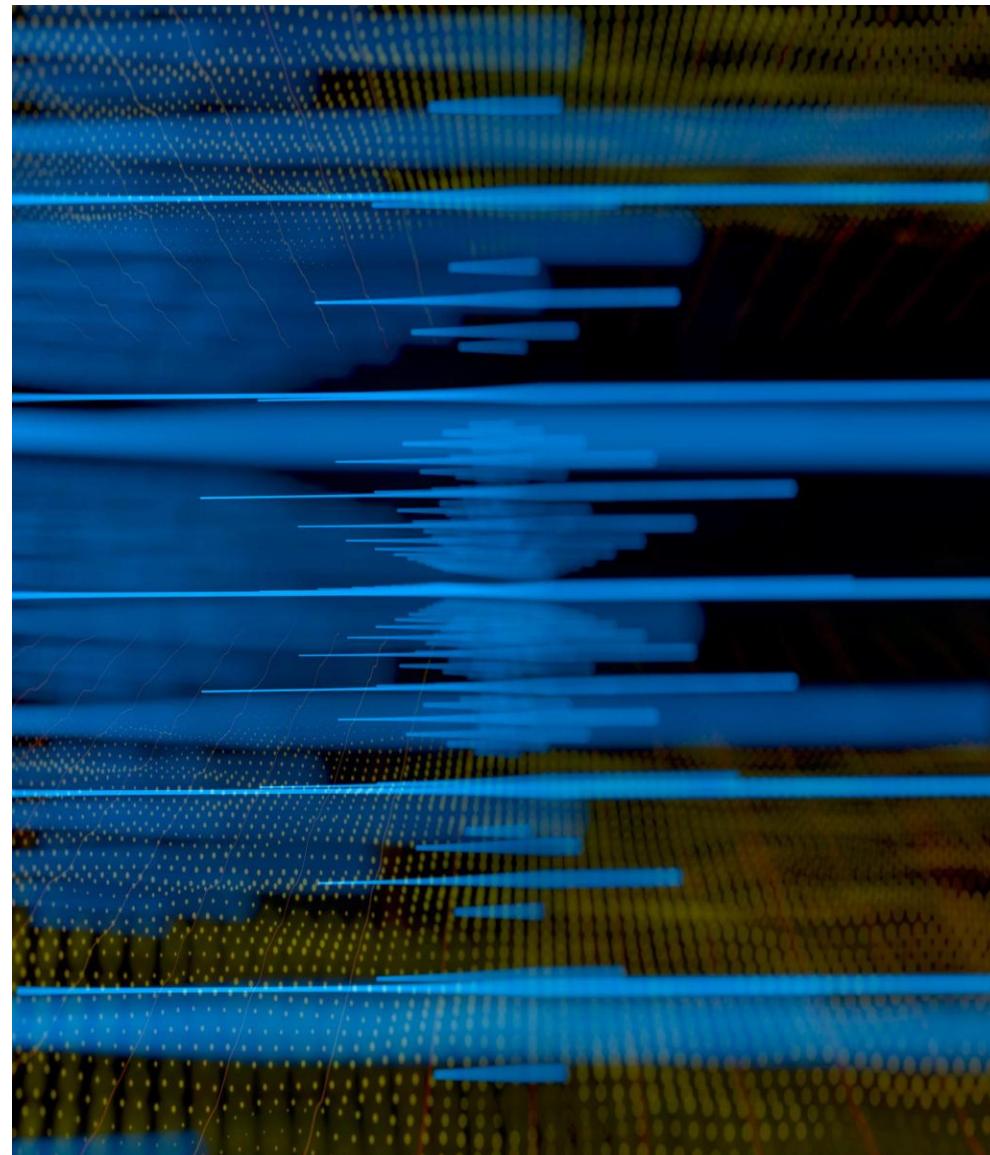
It tests simple vector operations like copy, scale, add, and triad, which reflect memory-intensive workloads.

Focus on Main Memory

STREAM minimizes cache effects to focus on assessing main memory performance realistically.

Applications of Results

Results guide hardware selection, system tuning, and application optimization to enhance computational performance.





NUMA Impact on Bandwidth

STREAM results highlight how NUMA architecture affects memory bandwidth and performance variability across threads and sockets.

Local vs Remote Memory Access

STREAM helps compare local and remote memory access speeds to identify performance bottlenecks and optimize memory placement.

Optimization and Workload Distribution

Analyzing STREAM output guides developers in workload distribution and optimization for efficient bandwidth utilization in NUMA systems.

Summary of Key Concepts

NUMA Performance Benefits

NUMA architectures improve performance by optimizing memory access across multiple sockets, though they add complexity to system design.

Thread Affinity Optimization

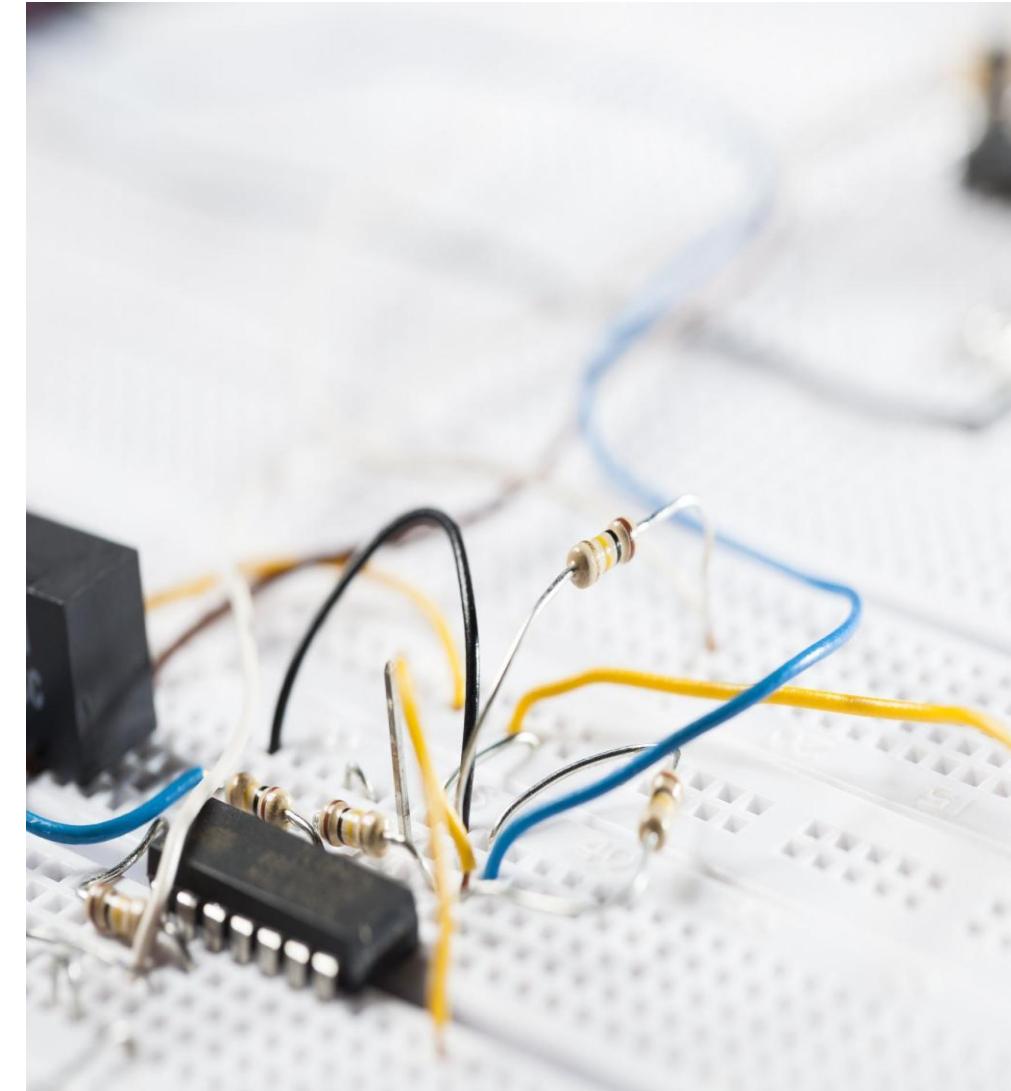
Optimizing thread affinity helps reduce latency by binding threads to specific processors or cores in NUMA systems.

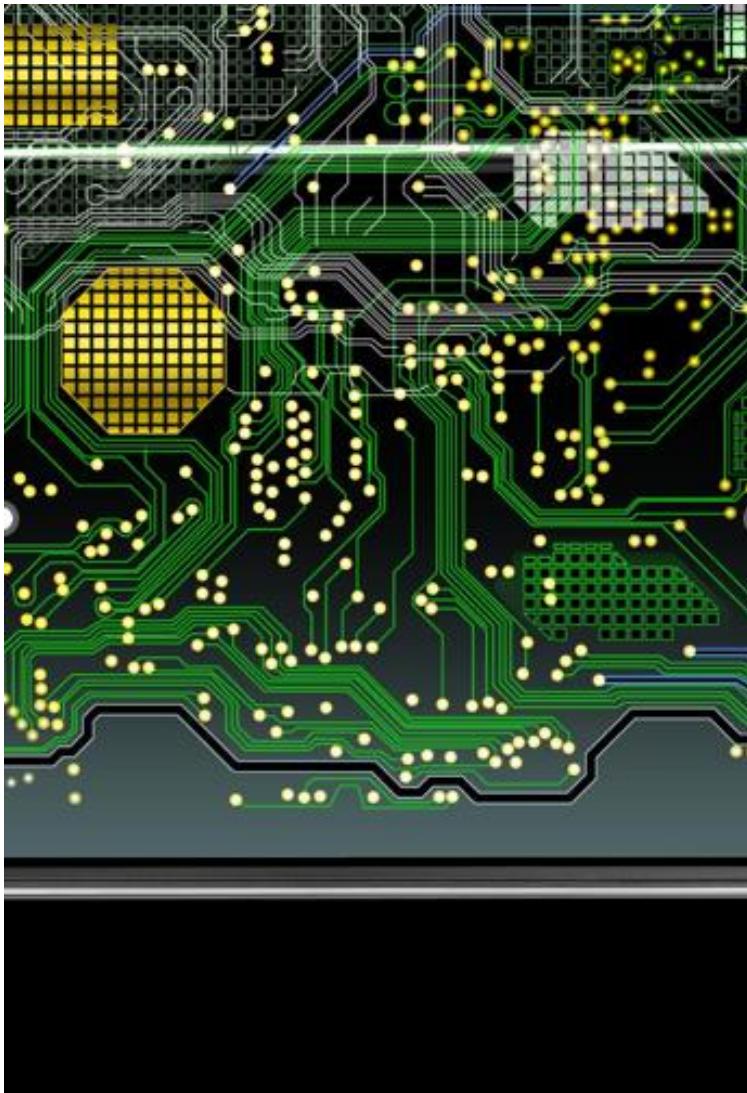
Memory Placement and Monitoring

Leveraging first touch memory placement and monitoring bandwidth utilization ensures efficient memory access and system throughput.

NUMA-aware Programming Tools

Tools like OpenMP and benchmarks such as STREAM help developers control thread placement and evaluate memory performance effectively.





Analyze System Topology

Use tools like `lstopo` or `hwloc` to understand NUMA system layout and optimize resource allocation.

Configure Thread Affinity

Set thread affinity and memory binding using OpenMP and OS utilities to improve data locality and performance.

Optimize Initialization Loops

Structure loops to follow the first touch policy for efficient memory allocation in NUMA systems.

Profile and Validate Performance

Profile bandwidth usage and validate optimizations using STREAM and other benchmarks to ensure scalability.