NUMA (Non-Uniform Memory Access) and Memory Pinning

Modern multi-CPU systems increasingly rely on **NUMA architectures** to scale memory and compute performance. But performance gains are only realized when developers and operating systems are NUMA-aware. Memory pinning further fine-tunes control by locking memory to specific NUMA nodes.

🔷 2. NUMA Implications on Performance

- **Local memory access**: Low latency, high throughput.
- **Remote memory access**: High latency, potential bottlenecks.
- **Memory bandwidth contention**: May occur if many threads access a remote node's memory.

Poor NUMA locality leads to:

- Cache misses.
- Memory access delays.
- Reduced overall throughput.

🔷 4. Tools for NUMA Awareness

Tool	Purpose
`numactl`	Run programs with specific NUMA bindings.
`numastat`	Show memory usage per NUMA node.
`hwloc`	Visualize hardware layout (topology-aware).
`taskset`	Bind process to specific CPUs.
`libnuma`	C library to manage memory/node affinity.

Example using `numactl`:

```
# Run a process on CPU node 0 and allocate memory from node 0
numactl --cpunodebind=0 --membind=0 ./my_app
```

🔷 6. NUMA and Multi-threading

In multithreaded applications (e.g., using `pthread`, OpenMP):

- Threads should be **affinitized to cores**.
- Memory allocations should be **localized to the threads' NUMA nodes**.

Example:

- Bind threads in a worker pool to separate NUMA nodes.
- Use per-node memory pools for data locality.

🔷 8. Performance Tips

Tip	Benefit
Pin threads and memory to same NUMA node	Reduced latency, increased cache hits
Avoid cross-node memory access	Prevents bus congestion and slowdowns
Use per-node memory pools	Better scaling with more cores/sockets
Monitor with `numastat`, `perf`, `htop`	Detect imbalances or remote memory use

---

🔷 9. Conclusion

NUMA and memory pinning are critical for performance on modern multicore/multisocket systems. Developers writing low-latency, high-throughput applications must:

- Be aware of the memory topology.
- Use tools and APIs to enforce memory locality.
- Align thread and memory placement for optimal performance.

Failing to consider NUMA effects can result in underutilized hardware, memory stalls, and unnecessary cross-node traffic.