🚀 🚀 CPU Affinity and Its Impact on Application Performance 🚀 🚀

🔍 What is CPU Affinity?

CPU Affinity is the process of binding a process or a thread to a **specific CPU core or a set of cores** to improve **cache locality, reduce context switching, and optimize scheduling**.

By default, operating systems distribute threads across available CPUs dynamically. However, CPU affinity can **pin** a process or thread to a fixed CPU(s), providing **consistent execution** and preventing unnecessary CPU migrations.

CPU affinity is implemented at two levels:

1. **Process-level Affinity** → Binds an entire process to specific CPU(s).
2. **Thread-level Affinity** → Binds individual threads within a process to specific CPU(s).

🚀 How CPU Affinity Impacts Performance

CPU affinity has a **significant effect on performance, especially for CPU-bound and real-time applications**.

1. Cache Locality Optimization (L1/L2/L3 Caching)

- CPU caches (L1, L2, L3) store frequently accessed memory.
- If a thread moves between different CPUs, it experiences **cache misses**, forcing it to fetch data from **main memory (RAM)**, increasing latency.
- **Binding a thread to a fixed CPU improves cache reuse** and reduces cache misses.

🔬 Example: Measuring CPU Cache Efficiency Using perf

```
perf stat -e cache-references,cache-misses taskset -c 0 ./myapp
```

- **High cache misses** → Frequent CPU migrations **(bad performance)**.
- **Low cache misses** → Good cache locality **(optimized performance)**.

---

2. Reduced Context Switching Overhead

- When a thread switches between CPUs, it incurs:
    - **Register flushes** (saving/restoring CPU registers).
    - **Pipeline flushes** (losing in-flight CPU instructions).

  - **Cache invalidation** (L1/L2/L3 cache eviction).
- High-frequency context switching can **increase CPU overhead** and degrade performance.

🔬 **Example: Measuring Context Switches Using pidstat**

```
pidstat -w -p <PID>
```

- **High voluntary/involuntary context switches** → Threads migrating frequently.
- **Binding affinity to fewer CPUs reduces context switches**.

---

3. NUMA (Non-Uniform Memory Access) Optimization

NUMA architectures (multi-socket CPUs) have **local and remote memory regions**.

- Accessing **local memory** is **fast (low latency)**.
- Accessing **remote memory (cross-socket NUMA access)** incurs **higher latency**.

🔬 **Check NUMA Node Allocation with numactl**

```
numactl --hardware
```

- Use numactl to bind processes to a specific NUMA node:

  numactl --cpunodebind=0 --membind=0 ./myapp

---

4. Load Balancing vs. Core Dedication

- **CPU-intensive applications (e.g., machine learning, video processing)** → Should be assigned **specific CPU cores** to prevent contention.
- **I/O-bound applications (e.g., web servers, databases)** → Should **allow OS scheduling** to distribute workload dynamically.

🔬 **Check CPU Load Using htop**

```
htop
```

- Identify **overloaded CPUs** and adjust affinity accordingly.

⚙️ Where & How to Adjust CPU Affinity?

💻 1. Configuring CPU Affinity in Linux

Using taskset to Bind a Process to Specific CPUs

- **Check Available CPUs**

```
lscpu
```

- **Run a Process on Specific CPUs**

```
taskset -c 2,3 java -jar myapp.jar
```

- **Modify Affinity for a Running Process**

```
taskset -cp 1-3 <PID>
```

- **Check Current CPU Affinity**

```
taskset -p <PID>
```

🛠️ 2. Adjusting CPU Affinity for Java Applications

JVM does not provide direct CPU affinity control, but you can manage it using **OS tools or Java libraries**.

Tuning JVM Garbage Collection Threads

- JVM Garbage Collectors (GC) **spawn multiple threads** that can interfere with application threads.
- Use *-XX:ParallelGCThreads* and *-XX:ConcGCThreads* to control GC CPU usage.

🔬 Example: Configuring G1GC for CPU Optimization

```
-XX:+UseG1GC -XX:ParallelGCThreads=4 -XX:ConcGCThreads=2
```

Using JNI or Libraries to Set Affinity in Java

```
Affinity.setAffinity(2); // Bind thread to CPU 2
```

🐳 3. Adjusting CPU Affinity in Docker/Kubernetes

- **Pin Docker Containers to Specific CPUs**

```
docker run --cpuset-cpus="0,1" -it mycontainer
```

- **Kubernetes Pod CPU Affinity**

```
resources:
 requests:
  cpu: "2"
 limits:
  cpu: "4"
```

💻 4. Configuring CPU Affinity in Windows

- Open **Task Manager** → Right-click Process → **Set Affinity**
- Using wmic in PowerShell:
- wmic process where name="java.exe" CALL setpriority 128

🚀 When Should CPU Affinity Be Adjusted?

Application Type	Recommended CPU Affinity Strategy
Low-latency real-time apps	Pin to a dedicated core to minimize jitter.
High-throughput web servers (Nginx, Apache)	Allow OS scheduling for load balancing.
Garbage Collector (JVM GC tuning)	Spread GC threads across multiple CPUs.
I/O-bound applications (Databases, Redis, Kafka)	Allow OS scheduling for optimal concurrency.
Machine learning / HPC workloads	Pin workloads to cores based on NUMA topology.
Game engines & real-time physics	Bind main loop to a fixed CPU for consistent frame rates.

📝 Summary: Best Practices for CPU Affinity Tuning

✅ **Pin real-time applications to specific cores** to avoid latency spikes.
✅ **Allow OS to manage scheduling for I/O-heavy applications** like web servers.
✅ **Balance CPU-bound applications (e.g., JVM GC, databases) across cores.**
✅ **Optimize NUMA-aware workloads for multi-socket systems.**
✅ **Use taskset, Docker cpuset, numactl, and JVM GC tuning for performance-critical workloads.**

Proper CPU affinity tuning can significantly improve performance, reduce cache misses, and optimize thread execution!