Real-Time on Many-Core Systems

Where real-time and high-performance can benefit from each other

Georg Wassen, Pablo Reble, Stefan Lankes, and Thomas Bemmerl

Chair for Operating Systems, RWTH Aachen University, Germany

Approaches to Real-Time on Multi-Processors

Classic approaches to real-time applications on multi-processor systems are:

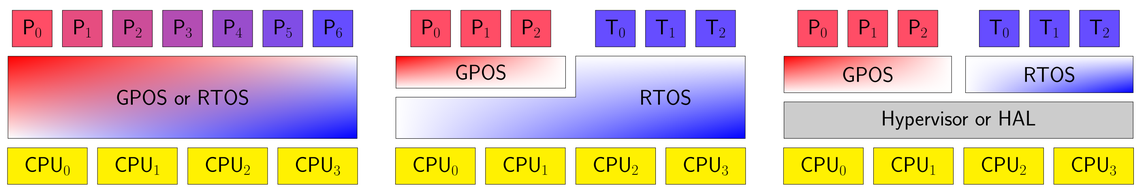
SMP: Single OS, APIs for Scheduling (POSIX), IPC, ...

RTOS: Special Operating Systems

RT-Patch: Modifications for Linux-Kernel (soft RT)

Sub-Kernel: RTOS below GPOS (which runs as least prio. task)

Hypervisor: Dedicated (separate) OS for each CPU



Single OS (SMP) Sub-Kernel Hypervisor (AMP)

Disadvantages:

- Hard real-time requires special RTOS, often with its own API
- Less portable (e. g. RT-Patch and Sub-Kernel only for selected versions)
- Reduced infrastructure (some tools/libraries are not available)

Challenges with Linux and Results

Blocking all interrupts on a CPU will harm the system:

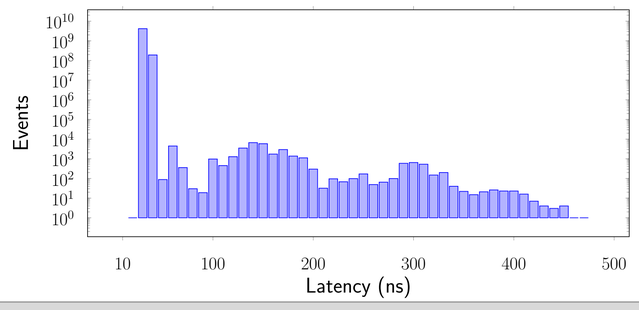
- Timer Interrupts: Clock, Accounting (may drift related to other CPUs)
- Inter-Processor Interrupts: if synchronous IPI not handled, system blocks
- Read-Copy-Update: memory not freed

→ Kernel modification was unavoidable (done for Linux 2.6.31-rt and 3.2)

Kernel-patch includes:

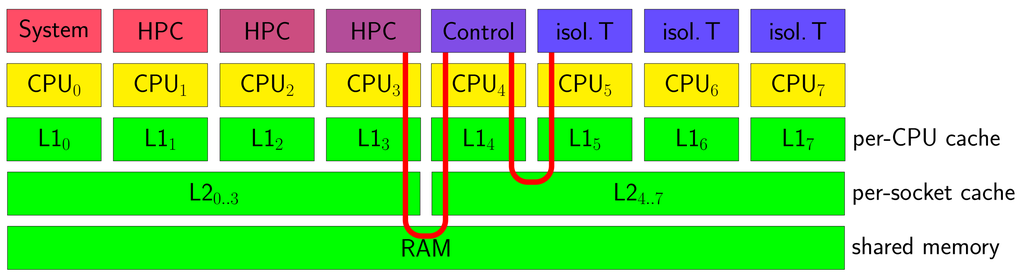
- Flexible run-time configuration via sysfs
- Block/ignore IPIs
- Shut down CPU (similar to hotplugging: notify *most* subsystems)
- Halt timer interrupt (This way: no interrupt-flag needed)
- Can be done from running CPU and remotely (important for recovery)

With these changes, multiple isolations can be started flexibly. Tested with run-times up to 72 hours. System remains stable, max. latency ca. 800 ns.



Outlook to Many-Core

The isolated cores use polling (busy waiting) and are off limits for the load balancer. The overall system load can not use these CPUs, the efficiency is lowered. But with more CPUs, it is less hurtful to reserve some for real-time. With many (even light-weight) CPUs, different domains (e. g. soft and hard real-time or different assignments) can be created. It is important to consider the underlying architecture (caches, memory topology, system busses).

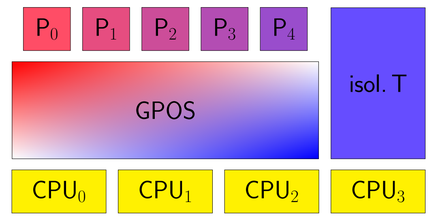


Exemplary software architecture (two sockets)

Here, a control process can communicate via the shared L2 cache with the isolated hard real-time tasks (isol. T) and via RAM with the other processes. The isolated tasks limit themselves to the L1 cache, all other activity is on the other socket.

Isolation

Our project targets *hard real-time tasks* on a *general-purpose operating system (GPOS)*. We realized this via isolation of Linux user-space processes on dedicated CPUs.



Isolation

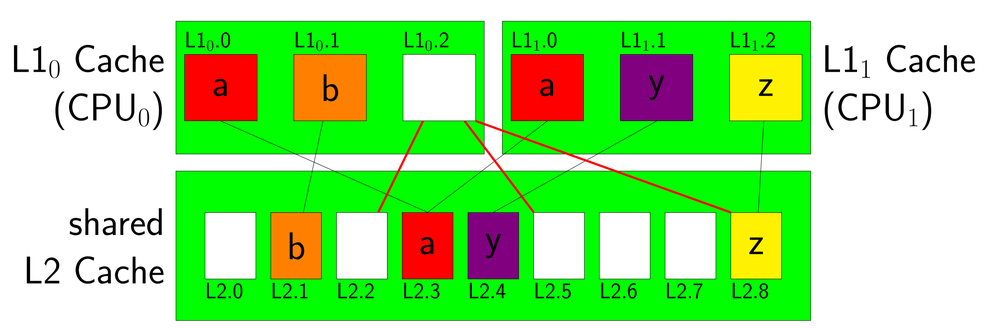
- Stop NMI watchdog, care for SMI (System Management Interrupt)
- Set CPU frequency to fixed maximum value (Power governor)
- Move Interrupts to other CPUs (Interrupt Affinity) and stop IRQ-Balancer
- Move other processes (CPU-Sets)
- Use Interrupt-Flag to block remaining interrupts (Timer and IPI)
- Avoid System Calls, replace with shared-memory IPC (using atomic ops)

→ Can be done with every Operating System without changes to its Kernel.

\+ For P_n: all OS services, libraries, and versions, soft real-time possible
− Isolated Task (isol. T) has limited API

Cache Behavior

Cache effects: shared caches reduce the predictability. A small task limiting itself to the (non-shared) L1 cache is in fact influenced by a memory consuming load on another CPU if they share an inclusive last-level cache.



Eviction in inclusive Caches

Example: An element should be placed in the $L1_0$ cache line $L1_0.2$. Due to associativity, it can be placed in L2 cache lines L2.2, L2.5 or L2.8 (red lines). If the placement strategy chooses L2.8, element (z) will be also evicted from $L1_1.2$, because it is no longer inclusive to the L2 cache.

→ General problem of the x86 architecture, can only be mitigated by careful placement of tasks on multiple sockets (without shared caches).

Benefits

Analyzing real-time effects on a many-core processor yields experience for:

- Deep understanding of low-level architecture
- Synchronization
- Bare-metal programming
- Caching effects

Isolation gains less than 1 % of CPU time by eliminating interrupts, but this avoids system jitter. The more cores work together, the more important is the *lockstep* of all cores between synchronization points.
We plan to apply isolation to the MCPC's `crbif` kernel driver in our cluster of SCCs (Reble: "Connecting the Cloud", 2012) to improve the inter-SCC latency.

Real-time applications also benefit from HPC research.

- Availability of many-core processors
- Cooperation in developing inter-process communication methods
- Non cache-coherrent architectures promise to improve cache behavior

<lastname>@lfbs.rwth-aachen.de http://www.lfbs.rwth-aachen.de