The FERMI@Elettra distributed real-time framework

L. Pivetta, G. Gaio, R. Passuello, G. Scalamera

lorenzo.pivetta@elettra.trieste.it

Sincrotrone Trieste, Trieste, Italy

Outline

- Definition of a real-time system
- Some considerations on PC architecture and Operating Systems
- Performance tuning and tricks
- Real-time subsystem: Adeos/Xenomai
- Real-time performance figures
- Hardware
- Network Reflective Memory
- Use cases
- Conclusions

Real-time system

- A system is said to be real-time if the total correctness of an operation depends not only upon its logical correctness, but also upon the time in which it is performed.
- **hard**: missing a deadline is a total system failure
- **firm**: infrequent deadline misses are tolerable but the result is usless after deadline
- **soft**: the usefulness of a result degrades after deadline

PC architecture & OS

Considerations on PC architecture and GNU/Linux; some features have strong influence on the real-time behaviour:

- system architecture
  - multiple bridges
  - microprocessor cache
- GNU/Linux
  - general purpose OS privileges throughput over responsiveness
  - not suitable to predict precisely the execution of a task

Performance tuning

Methods to improve the OS performance:

- increase the scheduling priority via the `nice` command
- change the scheduling policy programmatically

```
sched.sched_priority = \
            sched_get_priority_max(SCHED_RR)-1;
sched_set_scheduler(0, SCHED_FIFO, &sched);
```

- reserve a CPU core to a task via the `taskset` command
- avoid process memory pages to be swapped via the `mlockall` system call

Preemptible kernel

A *fully preemptible* kernel, based on the PREEMPT_RT patch, allows to:

- reduce the kernel code protected by *big locks*
- implement the priority inheritance mechanism for in-kernel spinlocks and semaphores
- introduce the deferred operations
- enable the support for High Resolution Timers $\rightarrow$ periodic tasks

Pros: native API, existing device drivers.
Cons: no hard real-time behaviour guaranteed.

Interrupt Service Routine

The ISR has the highest pirority in the GNU/Linux kernel. It can't be scheduled and preempts other tasks.
To maximaze the responsiveness some application code could be inserted into an ISR. Good candidates are:

- **data availabile IRQ**: when data is ready an interrupt is rised
- **timer IRQ**: e.g. for periodic tasks

Tune IRQ affinity to minimize CPU cache side effects; e.g. reserve a core to one IRQ. Es. IRQ 25 served by core 0:

```
echo 0x1 > /proc/irq/25/smp_affinity
```

Pros: fast & responsive
Cons: kernel space development, no floatin point, tight constraints on task duration

Real-time subsystem

Adeos/Xenomai is a real-time framework cooperating with the Linux kernel to provide a pervasive hard real-time support to kernel and user space applications.

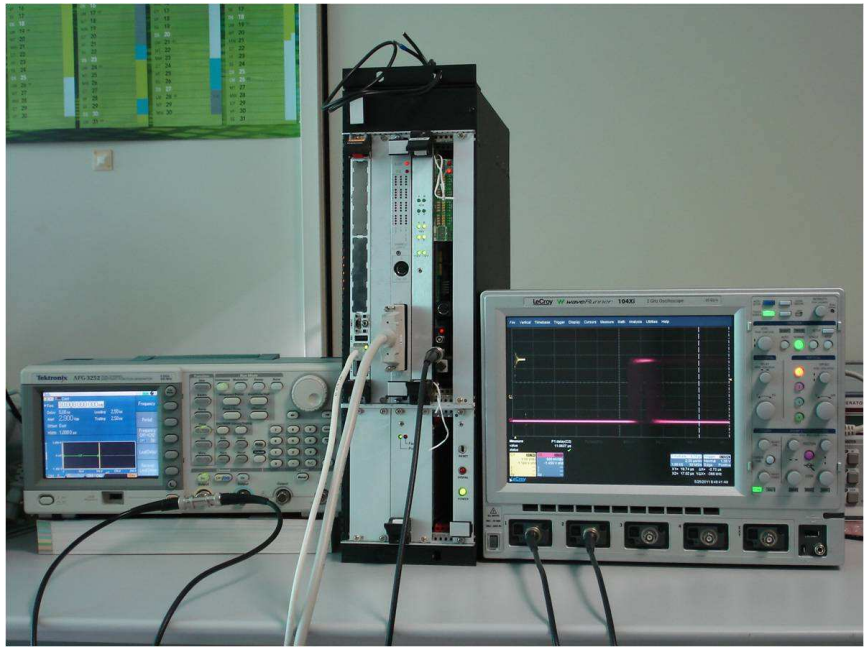
- Several processor architecture support: ARM, Blackfin, NiosII, PowerPC, x86...
- Complete set of API to develop real-time applications in both kernel and user space (IPC, synchronization, ...)

Concept: Xenomai handles all incoming IRQs before the Linux kernel; every IRQ could be managed by Xenomai and/or forwarded to Linux.

Pros: complete set of API to develop real-time applications in both kernel and user space (IPC, synchronization, maiboxes, ...)
Cons: kernel patch; no standard linux system calls in real-time applications; drivers to be adapted

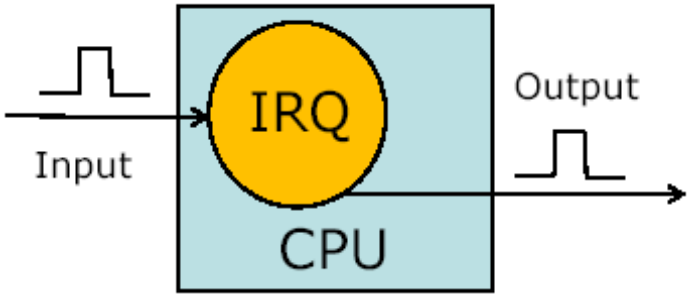
Real-time performance

A pulse, referred below as *input pulse*, produced by a signal generator is acquired by a digital input board which rises an interrupt on the VME bus. The IRQ is managed by the ISR and a digital output line driven producing an *output pulse*. A digital oscilloscope, triggered on the input pulse measures the latency of the output pulse.
System loaded with *ping flood*, heavy network activity, high-speed serial I/O.



Running in the ISR

- The code driving the digital output line is inserted into the ISR of the digital input board.
- Both Linux kernel and Xenomai perform well with a small advantage for the latter.
- The worst case latency decreases when running at higher frequency because the probability of generating a *cache miss* decreases.

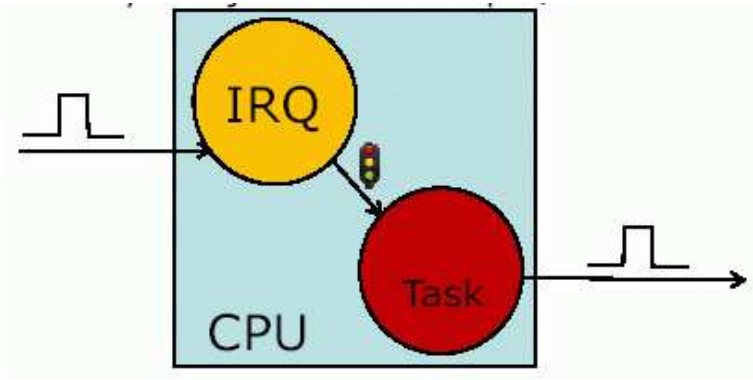


50 Hz rep. rate	MIN	MAX	MEAN	STD
Linux ISR	10 us	33 us	15 us	1.5 us
Xenomai ISR	11 us	23 us	16 us	1.4 us

10 kHz rep. rate	MIN	MAX	MEAN	STD
Linux ISR	10 us	29 us	11 us	0.8 us
Xenomai ISR	11 us	14 us	12 us	0.2 us

Running in a task

The code driving the digital output line is a Xenomai real-time task or a Linux task respectively. In both cases the task is pending on a `ioctl` call which waits on a semaphore. The ISR releases the semaphore and the unlocked task produces the output pulse.



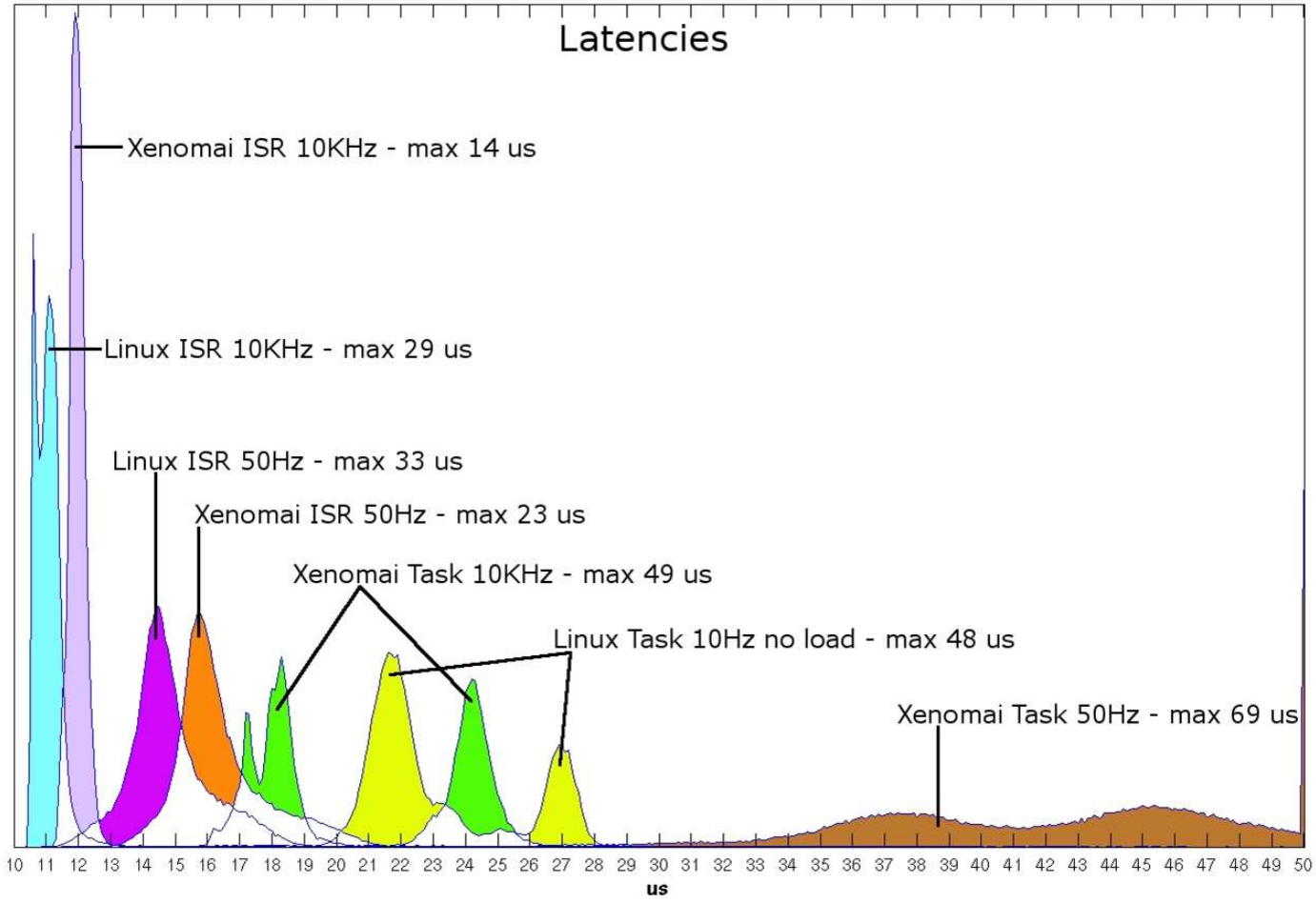
	Rep. rate	MIN	MAX	MEAN	STD
Xenomai task	50 Hz	17 us	69 us	42 us	5.7 us
Xenomai task	10 kHz	16 us	49 us	21 us	3.2 us

Rep. rate 10Hz	MIN	MAX	MEAN	STD
Linux task- no load	15 us	48 us	23 us	2.5 us
Linux task	196 us	37180 us	175 us	1374 us

- Xenomai task → real-time performance in user space!
- Linux task: outstanding performance without load but no guarantee to meet the deadlines when heavy loaded.

Latency measurements

Data sets are composed by few hundred thousand to several million samples depending on repetition rate. Latency distributions.



Real-time test results

- The Adeos/Xenomai real-time framework is reliable and performant
- Tasks with a repetition rate of more than 10KHz are feasible
- The jitter clearly depends on the task code: e.g. memory access, peripheral access through bridges, ...
- Best real-time performances when text and data segments could be kept in cache

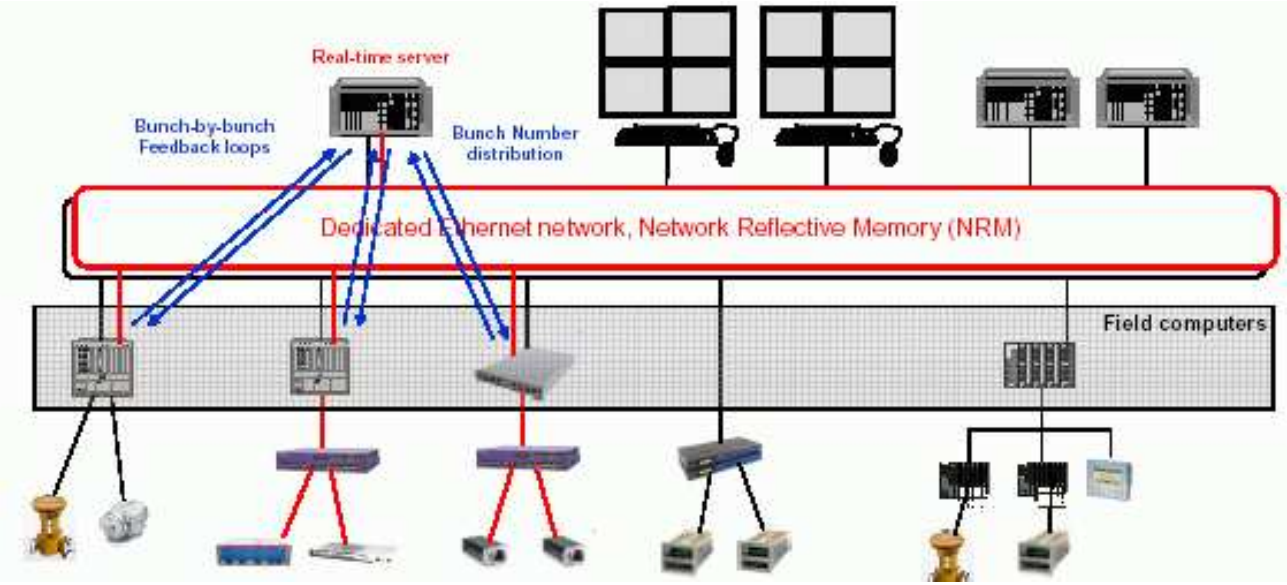
Hardware

Two platforms are currently used as front end computers for real-time tasks at the FERMI@Elettra:

- Emerson MVME7100 PowerPC VME SBC: 1.3 GHz dual core MPC8641D, 2GB DDR2 ECC RAM, 8 GB soldered flash disk, 4 Gigabit Ethernet ports, 5 RS232 serial lines, 4 high resolution timers.
  $\rightarrow$ control system real-time platform.
- Intel based 1U processing servers: two QuadCore 3.0 GHz Xeon processors, 4 to 16 GB DDR3 RAM, up to 6 Gigabit Ethernet ports.
  $\rightarrow$ CCD image processing front-end.
- COTS Gigabit/10Gigabit high performance switched Ethernet network

NRM

- A software infrastructure called Network Reflective Memory has been developed to share data between computers in real-time.
- Each real-time computer is reached by a dedicated Gigabit Ethernet network → restrained traffic
- Star shaped topology with the master station at the centre
- Customized device drivers for Gianfar and PRO/1000 chipsets



Principle of operation

- A master system collects all the messages coming from each slave and broadcasts them to all stations
- The master broadcast is used as NRM trigger:
  - Upon reception each slave updates its local copy of the NRM with the data belonging to the master packet
  - At the same time starts sending new local data to the master
- All the data synchronization is handled by Xenomai in real-time
- The customized Ethernet device driver allows raw access to the ISR and the transmission routines
- Two FIFO queues serve the writing requests coming from kernel space (NRM driver) and user space (application)
- To safeguard the bandwidth shared among stations the maximum packet size is fixed to 256 bytes for the slaves and up to 1MB for the master

Use cases

Several devices are currently interfaced shot by shot:

- **Electron/photon beam diagnostics**: electron beam position monitors (BPM), photon beam position monitors (PHPBM), current monitors (CM), CCDs, bunch lenght monitors (BLM), beam arrival monitors (BAM), laser power meters, i0 monitors (photon counters), experimental station detectors.
- **Power supplies**: corrector power supplies, quadrupole power supplies.
- **Radio frequency systems**: linac low level RF.
- **Machine protection systems**: Cherenkov optic fibers, ionization chambers.

Conclusions

- The Adeos/Xenomai real-time subsystem has been adopted to achieve hard real-time performances on front-end computers
- The real-time behaviour is remarkable even when the system is heavy loaded
- Programming API availabile in user space
- Network Reflective Memory: very effective to share "small" amounts of data with a known and guaranteed latency
- Based on standard Gigabit Ethernet hardware $\rightarrow$ cheap

Thank you