Linux Core Isolation and Tickless Operation

Ola Dahl[1]

[1]Enea

Embedded Conference Scandinavia, November 5, 2014

ENEA

Outline

Outline

Problem formulation

- How to achieve determinism by Linux kernel configuration and scripting?[1]
- Latency, throughput
- Real-time, networking

[1] and a few patches

Goals for this tutorial

- Point to where things can be found
- Give illustrative examples
- Give context and motivation
- Have fun

Outline

Core isolation and tickless operation

- Core isolation - minimize kernel activity on a selected set of cores
- Tickless operation - switch off the kernel tick

Determinism in Linux

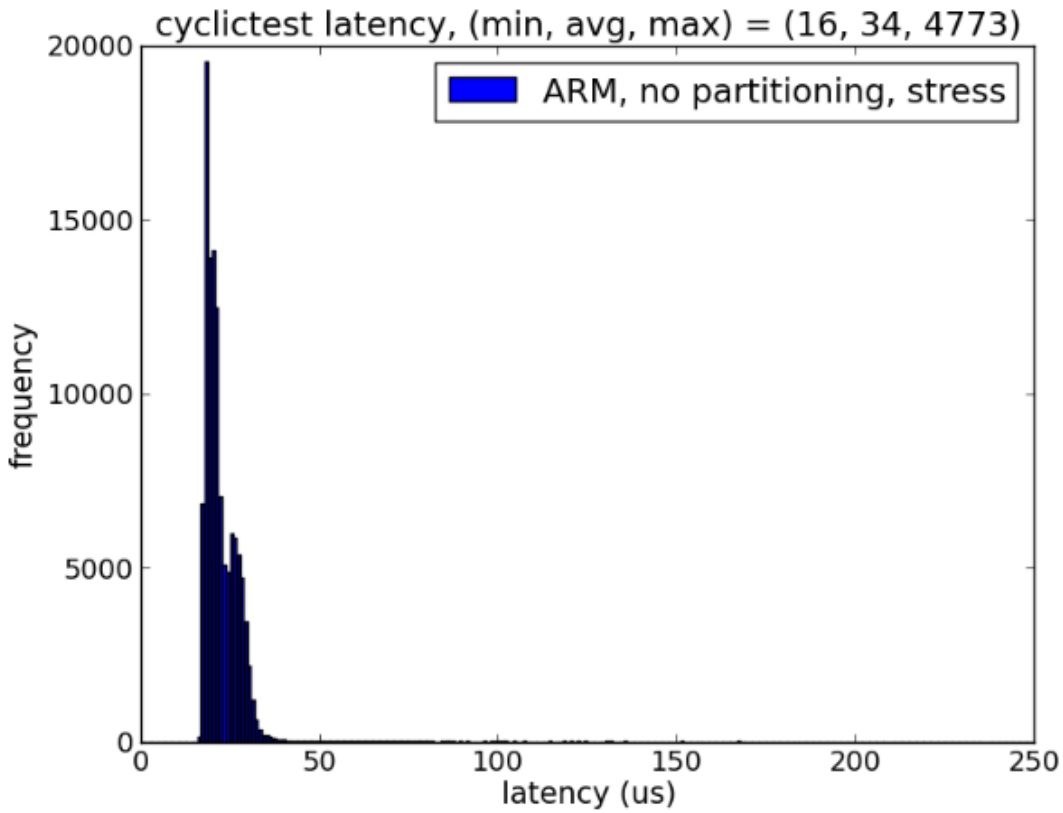
Why not PREEMPT_RT? [PREEMPT_RT, 2014]

- We want minimal modifications of the Linux kernel source code
- We like high throughput (and low latency - if possible) - e.g. [Abeni and Kiraly, 2013]

However - we pay a price by setting aside cores

Latency measurements

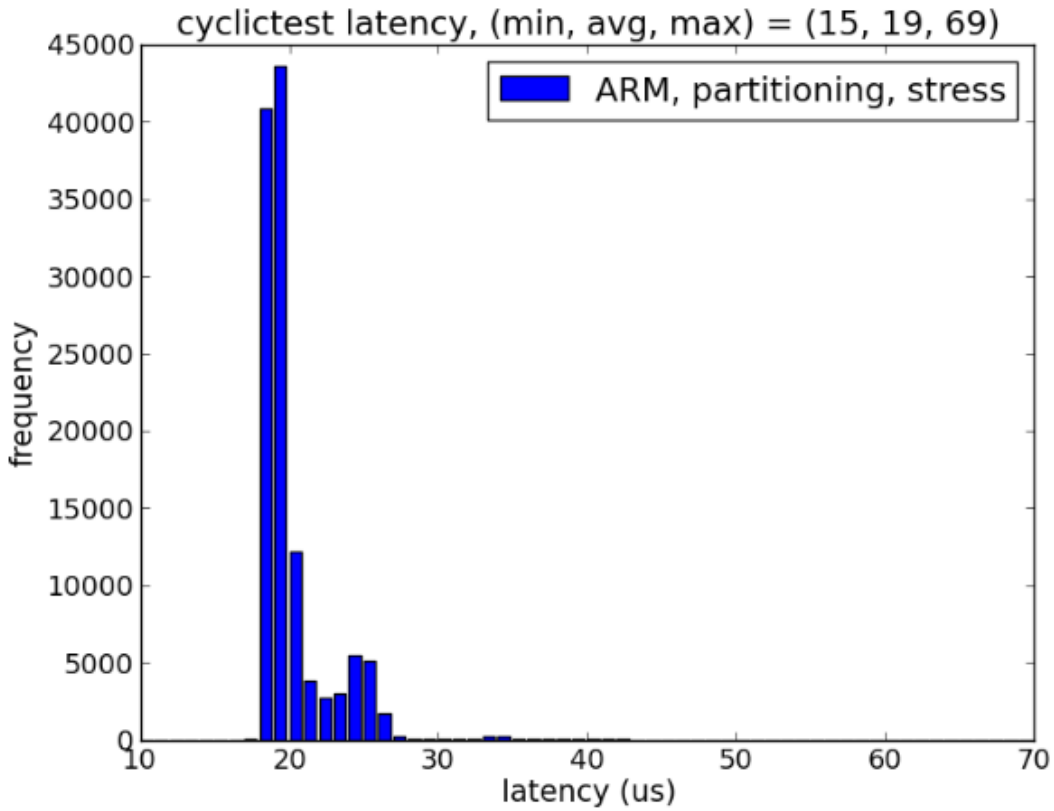
Cyclictest [Williams and Gleixner, 2014] with stress [Waterland, 2014] - unpartitioned system[2]



[2] A system is partitioned when we have applied scripts (and perhaps also patches) to achieve core isolation and tickless operation

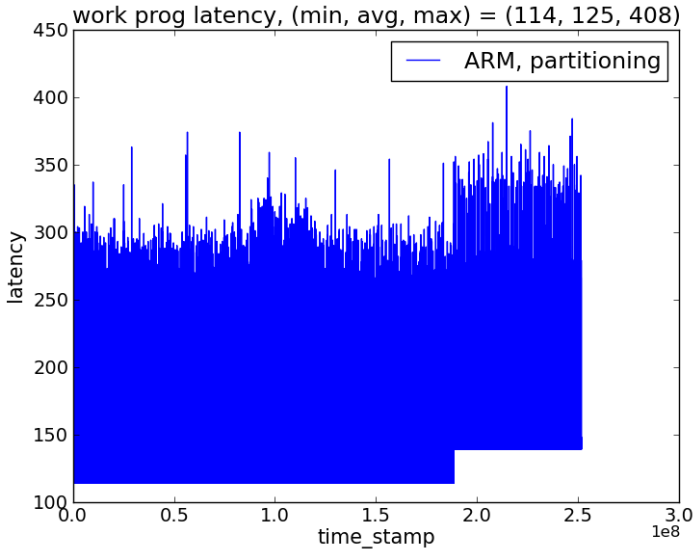
Latency measurements

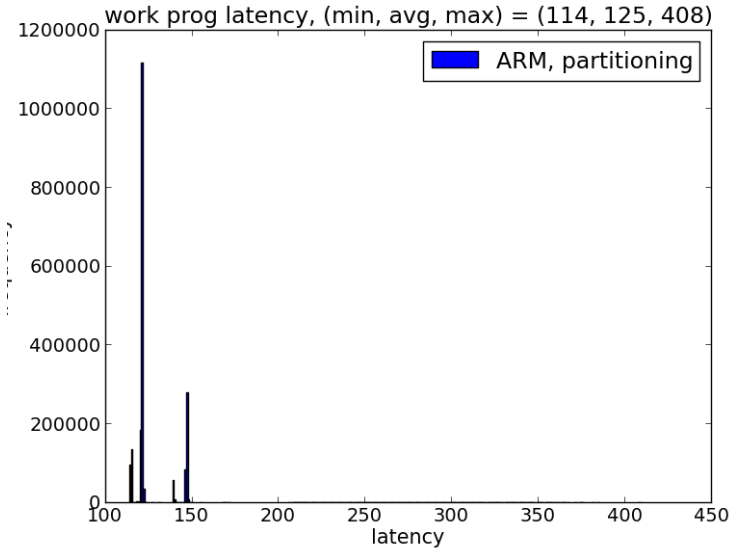
Cyclictest with stress - partitioned system



Execution time measurements

Execution time measurement without stress - partitioned system





A program executing a loop, and measuring time using a performance counter (cycle counter)

Core isolation and tickless operation

Stakeholders

- Networking [Liljedahl, 2013]
- HPC [Akkan et al., 2012]
- Real-time, e.g. [Corbet, 2013]

A very good overview is given by [McKenney, 2014]

Core isolation and tickless operation

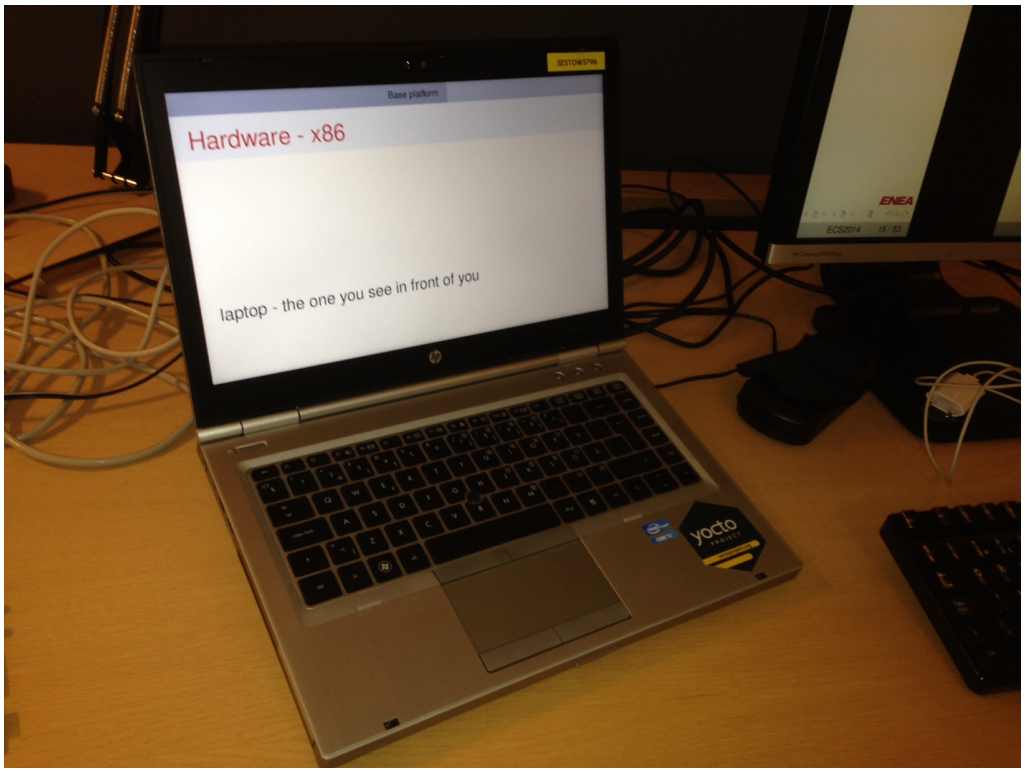
Status as of today

- timers/nohz updates for v.3.18 [Molnar, 2014]
- Linaro nohz upstreaming status [Kumar, 2014]
- Status according to Frederic Weisbecker [Weisbecker, 2014], [Weisbecker, 2013]

Outline

Hardware - x86

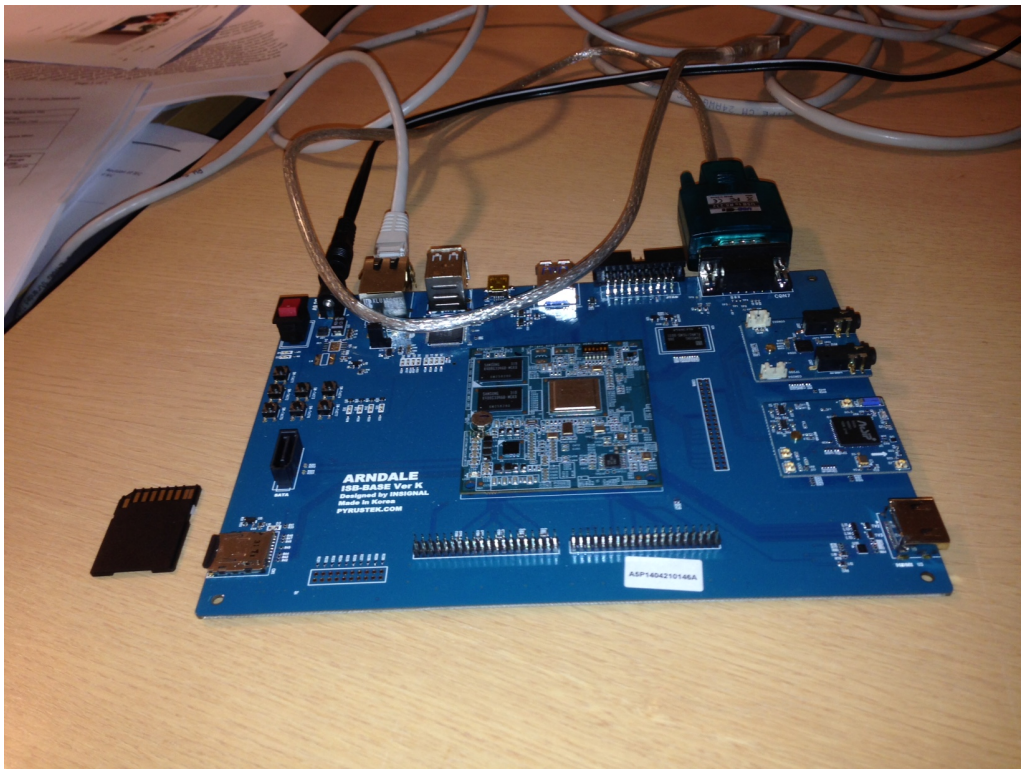
Laptop - the one you see in front of you



Ubuntu 14.04

Hardware - ARM

Arndale [Pyrustek, 2014] - dual Cortex-A15



Linux kernel - x86

Download and decide on which version to use

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/
    torvalds/linux.git
git tag | grep 3.16
git branch linux_3.16 v3.16
git checkout linux_3.16
make kernelversion
```

Linux kernel - x86

Select minimal configuration for the running computer [Rostedt, 2009]

```
make localmodconfig
```

Change some kernel configs

```
sudo apt-get install libncurses5-dev
make menuconfig
```

Build

```
make -j 16
```

Linux kernel - x86

Build and install modules

```
sudo make -j 16 modules_install
```

and look at the printout from the last command, which ends with

```
 INSTALL sound/pci/snd-als300.ko
 INSTALL sound/soundcore.ko
 DEPMOD  3.16.0
```

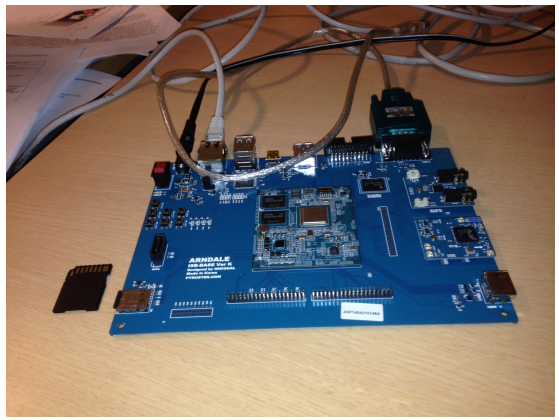
Deploy kernel

```
export VERSION=3.16.0
sudo cp .config /boot/config-$VERSION
sudo cp arch/x86_64/boot/bzImage /boot/vmlinuz-$VERSION
sudo update-initramfs -c -k $VERSION
sudo update-grub2
```

Then reboot, perhaps after having made the grub menu visible, as described in [askubuntu, 2010].

ENEA

Linux kernel - ARM



We need a kernel and a root file system

- Download ready-made SD-card image, or
- create kernel and root file system using Yocto, or
- create kernel and root file system using Linaro script

Linux kernel - ARM

Create kernel and root file system using Linaro script
Prepare, by doing

```
sudo apt-get install curl
sudo apt-get install lib32stdc++6
sudo apt-get install lib32z1
sudo apt-get install u-boot-tools
sudo apt-get install python-html2text
sudo apt-get install python-beautifulsoup
sudo apt-get install linaro-image-tools
sudo apt-get install fakeroot
sudo apt-get install dpkg-dev
```

Clone and run script as

```
git clone https://git.linaro.org/ci/job/linux-lng.git
cd linux-lng
./linux-lng.sh 2>&1 | tee build.log
```

ENEA

Linux kernel - ARM

Deploy image using

```
cd workspace/
gunzip arndale-lng-sd.img.gz
sudo dd if=arndale-lng-sd.img of=/dev/mmcblk0
sync
```

Known problem: you might have to add 'xz' as compression mode to the file

```
/usr/lib/python2.7/dist-packages/debian/debfile.py
```

so that after the edit, we get

```
$ grep xz /usr/lib/python2.7/dist-packages/debian/debfile.py
PART_EXTS = ['gz', 'bz2', 'xz', 'lzma']   # possible extensions
```

Linux kernel - ARM

Boot and check version

```
root@genericarmv7a:~# uname -a
Linux genericarmv7a 3.14.19-linaro-arndale #1 SMP Wed Oct 22
    08:55:40 CEST 2014 armv7l GNU/Linux
```

Outline

Cyclictest

Download

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/
    clrkwllms/rt-tests.git
```

Add required Ubuntu package

```
sudo apt-get install libnuma-dev
```

Build

```
make
```

Run

```
sudo ./rt-tests/cyclictest -q -D 10 -H 100 2>&1 | tee histogram
    .txt
```

Read more, e.g. [Rowand, 2013]

Cyclictest - example parameters

```
$ cyclictest --help
cyclictest V 0.89
Usage:
cyclictest <options>
```

Example options (edited help text)

```
-a Run thread #N on processor #N
-b USEC send break trace command when latency > USEC
-h dump a latency histogram to stdout after the run
-H same as -h except with an additional summary column
-i base interval of thread in us default=1000
-l number of loops: default=0(endless)
-m lock current and future memory allocations
-n use clock_nanosleep
-p priority of highest prio thread
-q print only a summary on exit
-S SMP testing: options -a -t -n and
   same priority of all threads
-t one thread per available processor
```

Counting ticks

Using a script from rt-tools [Enea, 2014]

```
git clone https://github.com/OpenEneaLinux/rt-tools.git
```

Count ticks

```
rt-tools$ sudo ./count_ticks/count_ticks --cpu 1 --start
rt-tools$ sudo ./count_ticks/count_ticks --cpu 1 --end
8 ticks occurred
```

Counting interrupts

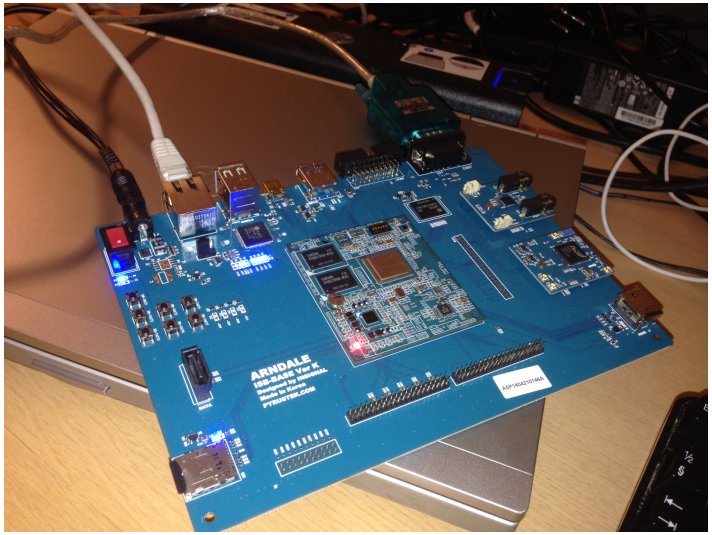
Using a script from Linaro

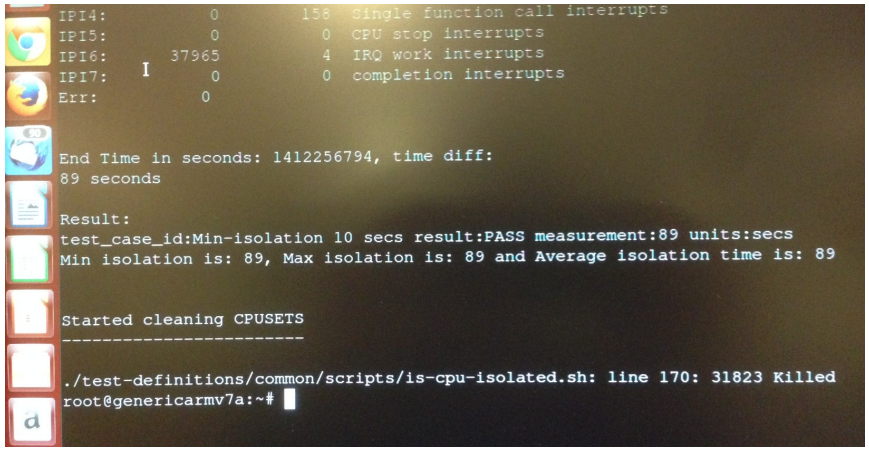
```
git clone git://git.linaro.org/qa/test-definitions.git
```

Count interrupts (check if CPU 1 is isolated)

```
./test-definitions/common/scripts/is-cpu-isolated.sh
```

ARM - Counting interrupts






```
Result:
test_case_id:Min-isolation 10 secs result:PASS measurement:89
    units:secs
Min isolation is: 89, Max isolation is: 89 and Average
    isolation time is: 89
Started cleaning CPUSETS
------------------------
./test-definitions/common/scripts/is-cpu-isolated.sh: line 170:
     3834 Killed                    stress -q --cpu 1 --timeout
    $STRESS_DURATION
```

ftrace

Using ftrace to analyze kernel activity

- An isolated CPU should not have any kernel activity
- Even if there are no ticks, there may be kernel activity - e.g. from timers, work queues, interrupts [Kumar, 2014]
- Read more e.g. by searching with site:lwn.net ftrace, where you might find e.g. [Gregg, 2014]

Outline

partrt

Using a script from rt-tools [Enea, 2014]

```
rt-tools$ sudo PATH=./bitcalc/src:$PATH ./partrt/partrt -v
    create 0xc
...
...
...
System was successfylly divided into following partitions:
Isolated CPUs (rt):2-3
Non-isolated CPUS (nrt):0-1,4-7
```

Use *run* instead of *create* for running programs on isolated CPUs

Linaro

Using the interrupt counting script from Linaro
Count interrupts (check if CPU 1 is isolated)

```
./test-definitions/common/scripts/is-cpu-isolated.sh
```

The script partitions the system, and counts interrupts

Core isolation mechanisms

What do the scripts do?

- cpuset
- load balancing
- interrupt affinity
- real-time throttling
- virtual memory timers
- watchdog
- writeback work queues
- machine check
- CPU hotplug
- CPU governor

See e.g. [Linux, 2014] - look for files named *cgroups/cpusets.txt*, *sysctl/vm.txt*, and *kernel-per-CPU-kthreads.txt*

Core isolation commands

Exctracts from *is_cpu_isolated.sh* and *partrt* scripts
Check if cpuset filesystem is enabled

```
if ! grep -q -s cpuset /proc/filesystems ; then
    echo "Error: Kernel is lacking support for cpuset!"
    exit 1
fi
```

Create cpusets

```
# Create 2 cpusets. One control plane and one data plane
[ -d /dev/cpuset/cplane ] || mkdir /dev/cpuset/cplane
[ -d /dev/cpuset/dplane ] || mkdir /dev/cpuset/dplane
```

Core isolation commands

Remove governor's background timers, i.e. use performance governor

```
if [ -d /sys/devices/system/cpu/cpu$1/cpufreq ]; then
    echo performance > /sys/devices/system/cpu/cpu$1/cpufreq/
        scaling_governor
fi
```

Run program on isolated CPU

```
# Move shell to isolated CPU
echo $$ > /dev/cpuset/dplane/cpu$1/tasks

# Start single cpu bound task
stress -q --cpu 1 --timeout $STRESS_DURATION &

# Move shell back to control plane CPU
echo $$ > /dev/cpuset/cplane/tasks
```

Core isolation commands

Enable tickless operation

```
# Try to disable sched_tick_max_deferment
if [ -d /sys/kernel/debug -a -f /sys/kernel/debug/
    sched_tick_max_deferment ]; then
    echo -1 > /sys/kernel/debug/sched_tick_max_deferment
    echo "sched_tick_max_deferment set to:" `cat /sys/kernel/
        debug/sched_tick_max_deferment`
else
    sysctl -e kernel.sched_tick_max_deferment=-1
fi
```

Core isolation commands

Move interrupts

```
for i in `find /proc/irq/* -name smp_affinity`; do
    echo 1 > $i > /dev/null;
done
```

Core isolation commands

Move tasks

```
for pid in `cat /dev/cpuset/tasks`; do
    if [ -d /proc/$pid ]; then
        echo $pid > /dev/cpuset/cplane/tasks 2>/dev/
...
```

From the ARM system, an example printout of tasks that could not be moved is

```
3 (ksoftirqd/0): Could not be moved to nrt
4 (kworker/0:0): Could not be moved to nrt
5 (kworker/0:0H): Could not be moved to nrt
6 (kworker/u4:0): Could not be moved to nrt
```

Core isolation commands

Handle load balancing

```
# Disable load balancing on top level
echo 0 > /dev/cpuset/$CPUSET_PREFIX"sched_load_balance"

# Enable load balancing withing the cplane domain
echo 1 > /dev/cpuset/cplane/$CPUSET_PREFIX"sched_load_balance"

# But disallow load balancing within the NOHZ domain
echo 0 > /dev/cpuset/dplane/$CPUSET_PREFIX"sched_load_balance"
```

Core isolation commands

Quiesce CPU: i.e. migrate timers/hrtimers away

```
echo 1 > /dev/cpuset/dplane/$CPUSET_PREFIX"quiesce"

# Restart $ISOL_CPUS to migrate all tasks to CPU0
# Commented-out: as we should get good numbers without this
    HACK
#echo 0 > /sys/devices/system/cpu/cpu$ISOL_CPUS/online
#echo 1 > /sys/devices/system/cpu/cpu$ISOL_CPUS/online
```

Outline

Linux kernel - ARM

We need a kernel with git history

Add remote

```
git remote add linux-linaro-lng git://git.linaro.org/kernel/
    linux-linaro-lng
```

Fetch contents

```
git fetch linux-linaro-lng linux-linaro-lng-v3.14
```

Create and checkout branch

```
git checkout -t remotes/linux-linaro-lng/linux-linaro-lng-v3.14
```

Linux kernel - ARM

We need to build and run the kernel
Take .config from the running kernel (the one we built together with the root file system)

```
scp root@172.16.140.16:/proc/config.gz .
gunzip config.gz
```

Setup toolchain path

```
export PATH=/home/olda/arm-tc-14.09/bin:$PATH
```

Linux kernel - ARM

Check kernel version

```
$ make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
    kernelversion
3.14.19
```

Check if we want to change some config flags

```
make ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- menuconfig
```

Build kernel

```
make -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- LOADADDR=0
    x40008000 uImage
```

Linux kernel - ARM

Build modules

```
make -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf- modules
```

Prepare a storage place

```
mkdir ../modules
export INSTALL_MOD_PATH=$PWD/../modules
```

Linux kernel - ARM

Install modules

```
make -j8 ARCH=arm CROSS_COMPILE=arm-linux-gnueabihf-
    modules_install
```

with printouts ending with

```
 INSTALL net/netfilter/xt_state.ko
 INSTALL net/netfilter/xt_tcpudp.ko
 INSTALL net/openvswitch/openvswitch.ko
 DEPMOD  3.14.19+
```

and we realize that the + is there, since we do not have a tag at the current commit

Linux kernel - ARM

Copy modules

```
export VERSION=3.14.19+
sudo mkdir /media/olda/rootfs/lib/modules/$VERSION
sudo cp -r ../modules/lib/modules/$VERSION/* /media/olda/rootfs
    /lib/modules/$VERSION
```

Copy uImage

```
sudo cp /media/olda/boot/uImage /media/olda/boot/uImage_org
sudo cp arch/arm/boot/uImage /media/olda/boot
```

We are done with the card

```
sync
```

and ready to boot

Linux kernel - ARM

Set boot parameters

```
Hit any key to stop autoboot:  0
ARNDALE5250 # setenv bootargs $bootargs isolcpus=1 nohz_full=1
ARNDALE5250 # boot
```

Boot

```
Last login: Thu Oct  2 12:17:21 UTC 2014 on tty1
root@genericarmv7a:~# uname -a
Linux genericarmv7a 3.14.19+ #2 SMP Fri Oct 24 08:53:06 CEST
    2014 armv7l GNU/Linux
```

and look at proc/version

```
root@genericarmv7a:~# cat /proc/version
Linux version 3.14.19+ (olda@olda-HP-EliteBook-8460p) (gcc
    version 4.9.2 20140904 (prerelease) (crosstool-NG linaro
    -1.13.1-4.9-2014.09 - Linaro GCC 4.9-2014.09) ) #2 SMP Fri
    Oct 24 08:53:06 CEST 2014
```

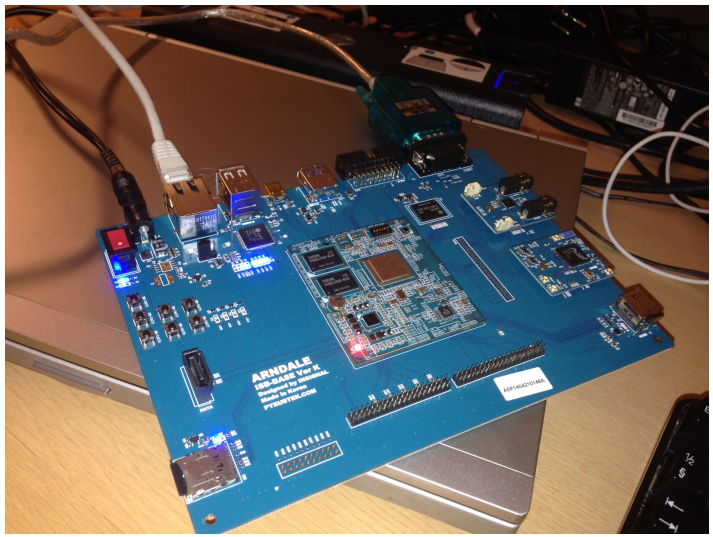
ENEA

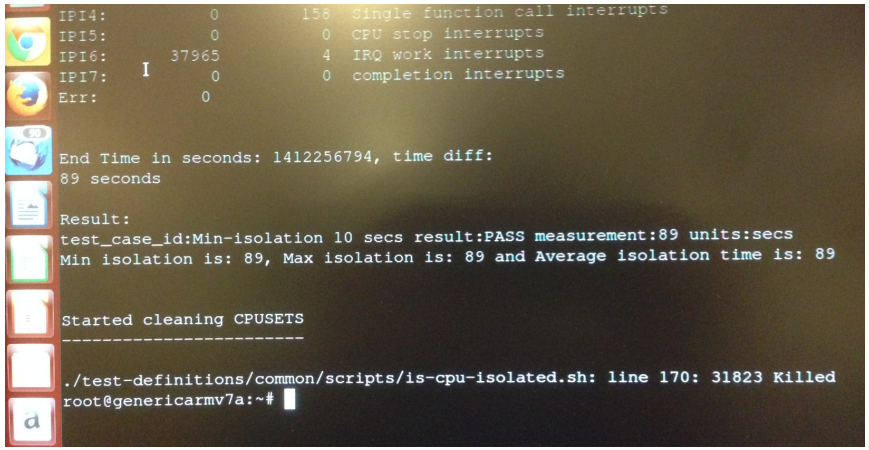
Linux kernel - tickless configuration

```
$ cat linaro/configs/no_hz_full.conf
### config fragment to add NO_HZ_FULL support
CONFIG_NO_HZ_FULL=y
CONFIG_NO_HZ_FULL_ALL=y
CONFIG_NO_HZ_COMMON=y
CONFIG_NO_HZ=y
CONFIG_NO_HZ_IDLE=n
CONFIG_HZ_PERIODIC=n
CONFIG_RCU_USER_QS=y
CONFIG_RCU_NOCB_CPU=y
CONFIG_VIRT_CPU_ACCOUNTING_GEN=y
CONFIG_CONTEXT_TRACKING_FORCE=y
CONFIG_IRQ_WORK=y
CONFIG_CPUSETS=y
CONFIG_CGROUPS=y
CONFIG_MAGIC_SYSRQ=y
CONFIG_DEBUG_FS=y
CONFIG_THUMB2_KERNEL=y
```

ENEA

ARM - Core isolation and tickless operation






```
Result:
test_case_id:Min-isolation 10 secs result:PASS measurement:89
    units:secs
Min isolation is: 89, Max isolation is: 89 and Average
    isolation time is: 89
Started cleaning CPUSETS
------------------------
./test-definitions/common/scripts/is-cpu-isolated.sh: line 170:
     3834 Killed                    stress -q --cpu 1 --timeout
    $STRESS_DURATION
```

Linux kernel - from 1 Hz to 0 Hz

Check branch

```
$ git branch
* linux-linaro-lng-v3.14
  linux_3.16
  master
```

Look for the commit

```
$ git log --pretty=format:"%n%h - %an, %ad %n%s" --grep
    sched_tick_max_deferment --stat

0fae6818 - Kevin Hilman, Tue Dec 17 13:23:07 2013 -0800
sched/nohz: add debugfs control over sched_tick_max_deferment
 kernel/sched/core.c | 16 ++++++++++++++++-
 1 file changed, 15 insertions(+), 1 deletion(-)
```

Linux kernel - 0 Hz for x86 mainline

Change branch

```
$ git checkout linux_3.16
Checking out files: 100% (16546/16546), done.
Switched to branch 'linux_3.16'
```

Check branch

```
$ git branch
  linux-linaro-lng-v3.14
* linux_3.16
  master
```

Look for the commit

```
$ git log --pretty=format:"%n%h - %an, %ad %n%s" --grep
    sched_tick_max_deferment --stat
```

Not there!

Linux kernel - 0 Hz for x86 mainline

Switch to Linaro branch

```
$ git checkout linux-linaro-lng-v3.14
```

Get full hash for the commit

```
$ git log --pretty=format:"%n%H - %an, %ad %n%s" --grep
    sched_tick_max_deferment

0fae6818f1836b08c5882d27c20ad249e2c4e9a0 - Kevin Hilman, Tue
    Dec 17 13:23:07 2013 -0800
sched/nohz: add debugfs control over sched_tick_max_deferment
```

Switch back to mainline branch

```
$ git checkout linux_3.16
```

Linux kernel - 0 Hz for x86 mainline

Apply commit

```
$ git cherry-pick 0fae6818f1836b08c5882d27c20ad249e2c4e9a0
[linux_3.16 394e6c6] sched/nohz: add debugfs control over
    sched_tick_max_deferment
 Author: Kevin Hilman <khilman@linaro.org>
 1 file changed, 15 insertions(+), 1 deletion(-)
```

Look at new log

```
$ git log --pretty=format:"%n%h - %an, %ad %n%s" -2

394e6c6 - Kevin Hilman, Tue Dec 17 13:23:07 2013 -0800
sched/nohz: add debugfs control over sched_tick_max_deferment

19583ca - Linus Torvalds, Sun Aug 3 15:25:02 2014 -0700
Linux 3.16
```

Linux kernel - patches for tickless operation

- We need the *sched_tick_max_deferment* patch to switch off the tick (which otherwise will be at least 1 Hz)
- Do we need more patches?

Linaro Linux kernels

- Linaro LSK - Linaro stable kernel (stable mainline with added ARM features) - `https://wiki.linaro.org/LSK`
- Linaro LNG - Linaro Networking Group kernel (LSK + features for networking[3]) - the one we have used above

[3] general features for networking and features for enabling tickless operation

Linux kernel - patches for tickless operation

Main Linaro developer: Viresh Kumar
Get the 3.14 branch of the Linaro stable kernel

```
git remote add linux-linaro-lsk git://git.linaro.org/kernel/
    linux-linaro-stable
git fetch linux-linaro-lsk linux-linaro-lsk-v3.14
git checkout -t remotes/linux-linaro-lsk/linux-linaro-lsk-v3.14
```

Linux kernel - patches for tickless operation

See how many commits are in LNG kernel but not in Linaro stable kernel

```
$ git log --oneline linux-linaro-lsk-v3.14..linux-linaro-lng-v3
    .14 | wc -l
46
```

Linux kernel - patches for tickless operation

See which of these were developed by Viresh

```
$ git log --oneline linux-linaro-lsk-v3.14..linux-linaro-lng-v3
    .14 --author="Viresh"
fe617ad hrtimer: make sure PINNED flag is cleared after
    removing hrtimer
c817b87 tick: SHUTDOWN event-dev if no events are required for
    KTIME_MAX
c4d6b62 hrtimer: reprogram event for expires=KTIME_MAX in
    hrtimer_force_reprogram()
6b948d0 sched: don't queue timers on quiesced CPUs
936748f cpuset: Create sysfs file: cpusets.quiesce to isolate
    CPUs
```

cont. on next slide

ENEA

Linux kernel - patches for tickless operation

```
5c1c5ba hrtimer: create hrtimer_quiesce_cpu() to isolate CPU
    from hrtimers
260c0d8 hrtimer: update timer->state with 'pinned' information
c7ce415 timer: create timer_quiesce_cpu() to isolate CPU from
    timers
927804e timer: track pinned timers with TIMER_PINNED flag
693d9b4 timer: Remove code redundancy while calling
    get_nohz_timer_target()
6879b1b linaro/configs: Add LNG OpenVswitch config fragment
```

Outline

Summary

- Building and running Linux kernels
- Core isolation and tickless operation
- Kernel configuration, scripts, and patches
- ARM test showed isolation for approx. 90 seconds with Linaro LNG kernel
- More work needs to be done!

The end

Thanks for your attention

Outline

Abeni, L. and Kiraly, C. (2013).
Investigating the network performance of a real-time Linux kernel.
In *15th Real Time Linux Workshop*.
http://csabakiraly.com/files/preprints/kiraly-2013RTLWS-RTNAPI-cr.pdf.

Akkan, H., Lang, M., and Liebrock, L. (2012).
Stepping towards a noiseless Linux environment.
In *International Workshop on Runtime and Operating Systems for Supercomputers - ROSS 2012*.
http://htor.inf.ethz.ch/ross2012/slides/ross2012-akkan.pdf.

askubuntu (2010).
How to get to the grub menu at boot-time?
http://askubuntu.com/questions/16042/how-to-get-to-the-grub-menu-at-boot-time.

Corbet, J. (2013).
(nearly) full tickless operation in 3.10.
LWN.net.
http://lwn.net/Articles/549580/.


Enea (2014).
Linux real-time tools.
https://github.com/OpenEneaLinux/rt-tools.


Gregg, B. (2014).
Ftrace: The hidden light switch.
LWN.net.
http://lwn.net/Articles/608497/.


Kumar, V. (2014).
Isolation / no_hz upstreaming progress.
In *Linaro Connect USA*.
http://lcu14.zerista.com/event/member/137768.


Liljedahl, O. (2013).
Lcu13: Discussion on ODP - networking applications on manycore SoCs.
In *Linaro Connect USA*.
http://www.slideshare.net/linaroorg/lcu13-lng-odpdiscussolavv7.


Linux (2014).
Linux kernel documentation.

ENEA

https://www.kernel.org/doc/Documentation/.

McKenney, P. (2014).
Bare-metal multicore performance in a general-purpose operating system.
In *Linux Collaboration Summit, Napa Valley, CA, USA*.
http://events.linuxfoundation.org/sites/events/files/slides/BareMetal.2014.03.28a.pdf.

Molnar, I. (2014).
[git pull] timers/nohz updates for v3.18.
https://lkml.org/lkml/2014/10/9/30.

PREEMPT_RT (2014).
Real-time Linux wiki.
https://rt.wiki.kernel.org/index.php/Main_Page.

Pyrustek (2014).
Arndaleboard-k (Exynos 5250).
http://bit.ly/1lBtzF9.

Rostedt, S. (2009).
[patch 00/14] kconfig: streamline distro configs for testers.
https://lkml.org/lkml/2009/8/18/507.

ENEA

Rowand, F. (2013).

Using and understanding the real-time cyclictest benchmark.

http://events.linuxfoundation.org/sites/events/files/slides/cyclictest.pdf.

Waterland, A. (2014).

stress.

http://people.seas.harvard.edu/~apw/stress/.

Weisbecker, F. (2013).

Status of Linux dynticks.

In *9th annual workshop on Operating Systems Platforms for Embedded Real-Time applications - OSPERT13*.

http://ertl.jp/~shinpei/conf/ospert13/slides/FredericWeisbecker.pdf.

Weisbecker, F. (2014).

Status of 1 year old full dynticks (aka nohz_full).

In *Linux Plumbers Conference*.

http://www.linuxplumbersconf.org/2014/ocw//system/presentations/2223/original/full_dynticks_lpc_2014.pdf.

Williams, C. and Gleixner, T. (2014).

rt-tests git repository.

http://git.kernel.org/cgit/linux/kernel/git/clrkwllms/rt-tests.git/.