Lessons Learned with SLES® and SLERT at Lockheed Martin Aeronautics

Robert K. Lake
Deputy Chief Engineer – Simulations
Lockheed Martin Aeronautics

Chris Manning
Embedded S/W Engineer Stf
Lockheed Martin Aeronautics
Agenda

What does Lockheed Martin Aeronautics do

Why Lockheed Martin Aeronautics uses SLES® and SLERT
- Indemnification
- Active Collaboration
- Active Participant in the Open Source Community

Lessons learned using SLERT
- BIOS tuning and kernel parameters
- CPU Sets
- IRQ Affinity
- NUMA
- LTTng

Other important aspects when using SLERT
- Large multi-core systems (kernel scalability)
- Third party hardware and drivers
Simulation and Systems Integration Labs Overview
Tactical Simulation Labs Support Concepts of Operation and Tactical Effectiveness

Intercept Control Station (Wingman, Blue or Red)

C4 ISR Stations (GFAC, JSTARS)

Mission Control Room

Full Field-of-View Simulator Display

Advanced Cockpit and Display
Spectrum of Tactical Simulator Configurations

**Laptop Level**
- Simple VS & MS Models
- Medium Fi Battle Model
- Full Cockpit displays
- Add Touch Screen and/or Pilot’s Controls
- Software Reused by:
  - VS Labs
  - MS Labs
  - Air Sys Labs
  - Bus Dev
  - Aero Concept Labs
  - Govt Labs
  - LM Tng Sys Labs

**Mini-MIC**
Desktop Config Plus:
- Correct Pilot’s Ctls Location
- Pedals

**Desktop Configuration**
Basic System Plus:
- Separate HU + HD Display
- More Entities
- PC Computer
- Simple HMD

**Medium Fidelity Station**
- Fully Functional Cockpit
- Wider FOV OTW Display
- HP/SUSE Computers
- More Complex Battlefield
- Links to 3 Stations

**Full MIC**
Desktop Config Plus:
- Separate Heads-Up and Heads-Down Displays
- Correct Pilot’s Ctls Location
- Side Panel Functions
- Pilot Seat
- Also used as Air Threat Station (linked)

**High Fidelity Station**
- Fully Functional Cockpit
- Full FOV OTW Display
- Hi Fidelity Sensor Displays
- Hi Fi Battle Model
- Large Entity Count
- Hi Fi Weapons
- Links to 3 other stations to effect 4-ship F-35
- Links to 8 Threat Stations
- Links to OPFAC stations
- HP/SUSE Computers
- Simulated Full HMD

**Highest Fidelity Station**
- Replace models with A/C S/W and H/W
- Govt Accredited Battle Model and Threat Data
- Highest Fidelity Aero Model
Vehicle Systems Labs Include Comprehensive Sim Stations, A/C Hardware and A/C Software
Our Role in Product Development

Concept Evaluation
- Concept Development
- Handling Qualities
- Full Mission Capability

Design & Development
- HW & SW Development Test Stations
- Simulation Systems
- Special Test Equipment
- Subsystem Testers

Systems Qualification & Verification
- Electromagnetic Testing
- Avionics Systems/Subsystems
- OFP Testing
- Full Mission Simulation

Integration Testing
- Systems/Subsystems
- OFP
- Test Systems

Integrated Lab Capabilities Work Together To Support Weapons Systems Development
Why Lockheed Martin Aeronautics uses SLES® and SLERT
Indemnification

• Why is this important?
  – The Open Source community provides a vast array of experience, technology and developed products
  – Brings with it the GPL license
    – opened the door to inadvertent exposure of key intellectual property and financial liability in the form of derived products
    – Increased cost of vetting all potential software for proper GPL and LGPL
  – As a defense contractor, Lockheed Martin is required by law to protect key intellectual property and defense oriented technology
Indemnification (cont.)

- SUSE® Indemnification Agreement has the following provisions:
  - Is offered for copyright, patent, trade secret or trademark-infringement claims made by third parties against SUSE customers.
  - Includes an agreement designed to protect you financially if a third party files a copyright, patent, trade secret or trademark-infringement claim against you.
  - This protection applies to all products offered by SUSE for customers who:
    - Purchase and maintain ongoing subscription(s) to upgrades and updates
    - Purchase and maintain annual technical support (either separately or included with maintenance)
  - SUSE added a “linking exception clause” to several files in glibc:

https://www.suse.com/partners/cloud-service-providers/technology-assurance-program/
Active Collaboration

- Real-Time features provided under SLERT
  - SUSE\textsuperscript{\textregistered} worked to improve timer performance to meet Lockheed Martin requirements
  - SUSE worked with hardware vendors to verify SLERT on large core count systems

- Real-Time analysis features under LTTng
  - SUSE added the “CPU Graph” view to the eclipse plugin

- Engagement with third parties to collaborate and enhance their drivers and products
  - HP\textsuperscript{\textregistered}, Datapath\textsuperscript{\textregistered}, NVIDIA\textsuperscript{\text™}
  - Certification of products
Active Participant in the Open Source Community

• Kernel Development
  - RT kernel enhancements – **Mike Galbraith**
  - Created a patch for the default and RT kernel that provided significant NFS performance improvements

• LTTng Development
  - Initial discussions about inclusion/improvements
    – **Vojtech Pavlik**
  - Eclipse plugin GUI work
  - Kernel development and validation testing – **Tony Jones**
30us Top-of-Frame Jitter Requirement

Methodology

```c
while(true)
{
    wait_for_timer;
    now = clock_gettime(CLOCK_MONOTONIC);
    error = (now-prev) - 1/frequency;
    prev = now;
}
```
X86 SLES® 11/SLE-RT

idle=poll

Error (usec)

60 Hz Sigma=0.729 usec
X86 SLES® 11 SP3/SLERT

Error (usec)

60 Hz Sigma=0.157 usec
Timer Performance
SLES® 11/SP3

60 Hz:
- Frame = 0
  Time = 0
  softirq_raise

- Frame = 1
  Time = 16.6ms
  softirq_raise

- Frame = 2
  Time = 33.2ms
  softirq_raise

Error:
- Frame = 1
  softirq_raise
- Frame = 2
  softirq_raise

Error - us

60 Hz Sigma=0.068 usec
Lessons Learned Using SLERT
Lessons Learned Using SLERT

• BIOS tuning
  - Disable Hyper-threading
  - Talk to HW vendor about System Management Interrupts (SMIs)
  - HP provided a tuning document for RT deployments

• Kernel parameter tuning
  - SUSE® engineering helped with suggestions
  - idle=poll, not used anymore
  - mce=off, turn off machine check exceptions
    - Or slow them down from the cset script
      
      ```
      echo 300 > /sys/devices/system/machinecheck/machinecheck0/check_interval
      ```
  - hpc_cpusets (see Documentation/kernel-parameters.txt)
CPUSets Configuration

To isolate a set of CPUs for time sensitive processes

```bash
echo -1 > /proc/sys/kernel/sched_rt_runtime_us
# Setup the system CPU set with access to mem node 0

cset set --mem 0 --cpu=0-4 --set=boot
# Setup RT CPU set with access to all mem nodes

cset set --mem 0-7 --cpu=5-24 --cpu_exclusive --set=rtcpus
# Move all running processes to the system CPU set

cset proc --move --kthread --fromset root --toset boot

echo 300 > /sys/devices/system/machinecheck/machinecheck0/check_interval
# Prevent scheduler from load balancing the rtcpus cpuset

echo "0" > $CSET_LOCATION/$RTSET_NAME/sched_load_balance

echo "0" > $CSET_LOCATION/$RTSET_NAME/sched_relax_domain_level

echo 0 > $CSET_LOCATION/sched_load_balance

echo 1 > $CSET_LOCATION/boot/sched_load_balance
# wait for sched_domain rebuild

sleep 5

echo 1 > $CSET_LOCATION/$RTSET_NAME/sched_hpc_rt

for i in $(ls /sys/devices/system/cpu/cpu*/cpufreq/scaling_governor); do
  echo "performance" > $i;
done
```
CPUSets Configuration (cont.)

• Make CPUSets survive reboot
  
  – Modify cset.init.d example script from /usr/share/doc/packages/cpuset
  
  – Add script to /etc/init.d
Working with CPUSets

• To programmatically move processes to an isolated CPU
  - Move the process to the real-time cpuset:

    ```
    stringstream ss;
    ss << "echo " << pid << " > /dev/cpuset/rtcpus/tasks" << ends;
    system(ss.str().c_str());
    ```
  
  - Move the process to the desired CPU:

    ```
    cpu_set_t mask;
    CPU_ZERO(&mask);
    CPU_SET(cpu,&mask);
    sched_setaffinity(pid,sizeof(mask),&mask);
    ```

• To programmatically set the RT scheduler/process priority:

    ```
    sched_param sp;
    sp.sched_priority = priority;
    sched_setscheduler(0,SCHED_FIFO,&sp)
    ```
Working with CPUSets (cont.)

Priority ranges:

99  Reserved for critical kernel threads and should not be used by applications
90 - 98  Hard real-time user threads
60 - 89  High-priority operating system services
40 - 59  Firm real-time user threads
31 - 39  Low-priority operating system services
1 - 30  Soft real-time user threads

/etc/security/limits.conf

* - rtprio 99
IRQ Affinity

• Documented in
  /usr/src/linux/Documentation/IRQ-affinity.txt

• Turn off irq_balancer

• Echo a CPU affinity bitmask to
  /proc/irq/default_smp_affinity
  echo 1 > /proc/irq/default_smp_affinity
  /proc/irq/*/smp_affinity

• Add some additional lines to /etc/init.d/cset to survive reboots

• With irq_balancer off you need to pin IRQ’s to a single CPU not a mask
NUMA Configuration

- See “CPUSets Configuration” Chart
  - Setup the system CPU set with access to memory node 0
    `cset set --mem 0 --cpu=0-4 --set=boot`
  - Setup Real-Time CPU set with access to all memory nodes
    `cset set --mem 0-7 --cpu=5-24 --cpu_exclusive --set=rtcpus`

- Group processes that share data, on the same NUMA node

- Useful commands to monitor NUMA statistics
  - `numactl --H`
  - `numastat`
Paired G7 Nehalem-EX Nodes
Architectural block diagram of the HP® PREMA Architecture in the ProLiant ™ DL980 G7

LTTng Kernel Tracing

- You really can’t identify how the real-time system is performing without this sort of tool
- We have worked closely with SUSE® to improve the usability of this tool
- Eclipse plugin
  - Good for detail visualization
- babeltrace
  - Converts trace to text output
  - When combined with a parsing tool this is good for getting a macroscopic view of the system
LTTng Kernel Tracing (cont.)

• The eclipse plugin can be slow. If running on a system with many core’s it is best to split the trace up by core

• For large core counts you will need to increase the buffer size

  lttng enable-channel channel0 -k --subbuf-size s --num-subbuf n

babeltrace will emit warnings if buffers have overflowed
Sample Babeltrace Output
(From Interrupt Latency Test)

[1407856860.605097580] softirq_raise: { cpu_id = 4 }, { vec = 8 }
[1407856860.605097928] sched_wakeup: { cpu_id = 4 }, { comm = "irq-hrtimer/4", tid = 72, prio = 0, success = 1, target_cpu = 4 }
[1407856860.605098634] sched_switch: { cpu_id = 4 }, { prev_comm = "swapper/4", prev_tid = 0, prev_prio = 20, prev_state = 0, next_comm = "irq-hrtimer/4", next_tid = 72, next_prio = -100 }
[1407856860.605099190] softirq_entry: { cpu_id = 4 }, { vec = 8 }
[1407856860.605099717] sched_wakeup: { cpu_id = 4 }, { comm = "timer", tid = 39351, prio = 1, success = 1, target_cpu = 4 }
[1407856860.605100229] softirq_exit: { cpu_id = 4 }, { vec = 8 }
[1407856860.605100740] sched_switch: { cpu_id = 4 }, { prev_comm = "irq-hrtimer/4", prev_tid = 72, prev_prio = -100, prev_state = 1, next_comm = "timer", next_tid = 39351, next_prio = -99 }
[1407856860.605101709] hrtimer_start: { cpu_id = 4 }, { hrtimer = 18446612167063670048, function = 18446744071579382384, expires = 1793086242679727, softexpires = 1793086242679727 }
[1407856860.605102125] exit_syscall: { cpu_id = 4 }, { ret = 39 }
[1407856860.605102734] sys_clock_gettime: { cpu_id = 4 }, { which_clock = 1, tp = 0x7FFF6FA98930 }
[1407856860.605102999] exit_syscall: { cpu_id = 4 }, { ret = 0 }
[1407856860.605103484] sys_rt_sigtimedwait: { cpu_id = 4 }, { uthese = 0x7FFF6FA98690, utinfo = 0x0, uts = 0x0, sigsetsize = 8 }
[1407856860.605104172] sched_switch: { cpu_id = 4 }, { prev_comm = "timer", prev_tid = 39351, prev_prio = -99, prev_state = 1, next_comm = "swapper/4", next_tid = 0, next_prio = 20 }
Sample Babeltrace Output - Magnified
(From Interrupt Latency Test)

[1407856860.605097580] softirq_raise: { cpu_id = 4 }, { vec = 8 }
[1407856860.605097928] sched_wakeup: { cpu_id = 4 }, { comm = "sirq-hrtimer/4", tid = 72,}
[1407856860.605098634] sched_switch: { cpu_id = 4 }, { prev_comm = "swapper/4", prev_tid
[1407856860.605099190] softirq_entry: { cpu_id = 4 }, { vec = 8 }
[1407856860.605099717] sched_wakeup: { cpu_id = 4 }, { comm = "itimer", tid = 39351, prio =
[1407856860.605100229] softirq_exit: { cpu_id = 4 }, { vec = 8 }
[1407856860.605100740] sched_switch: { cpu_id = 4 }, { prev_comm = "sirq-hrtimer/4",
[1407856860.605101709] hrtimer_start: { cpu_id = 4 }, { hrtimer = 18446612167063670048,
[1407856860.605102125] exit_syscall: { cpu_id = 4 }, { ret = 39 }
[1407856860.605102734] sys_clock_gettime: { cpu_id = 4 }, { which_clock = 1, tp =
[1407856860.605102999] exit_syscall: { cpu_id = 4 }, { ret = 0 }
[1407856860.605103484] sys_rt_sigtimedwait: { cpu_id = 4 }, { uthese = 0x7FFF6FA98690,
[1407856860.605104172] sched_switch: { cpu_id = 4 }, { prev_comm = "itimer", prev_tid =


Our real-time process was interrupted by another
Our real-time process was interrupted by another
Interrupt Latency

User Space

Kernel Space

Interrupt Latency

Mean = 1.068 us
S = 0.084 us

Mean = 3.271 us
S = 0.416 us

Mean = 1.052 us
S = 0.097 us

Mean = 4.750 us
S = 0.752 us

* - itimer is the code under test
Interrupt Latency - Magnified

* - itimer is the code under test
LTTng User Space Tracing

• LTTng supports user space tracing
  - It is cumbersome to bring the kernel trace and user trace into the eclipse plugin

• We use a different approach
  - Make an ioctl call from user code using /dev/null
    
    ```
    int fd = open("/dev/null", O_RDONLY);
    ioctl(fd, cmd, id);
    ```
  - Very low overhead (1us) and we can use the ioctl arguments to identify the call
Other important aspects when using SLERT
Scalability

- Currently with SLES® 11 SP3 we have scalability issues
  - Running systems with 80 cores
  - Bottleneck in the kernel
  - More cores are not better!

- Kernel development is moving scalability fixes into the baseline
  - 8192 logical CPU support
  - SysV IPC and futex enhancements
  - VFS lockref optimizations
Third Party Drivers

- Drivers need to be built for the Real-Time kernel
  - NVIDIA™
  - Datapath®
  - DigiPath
    - SUSE® came up with a procedure to compile the kernel module
  - Creative
    - SUSE added kernel driver to the RT kernel

- Vendor support is spotty