VxWorks on Intel for real time high performance computing

Helmut Tischer

Software Development Division

htischer@eso.org

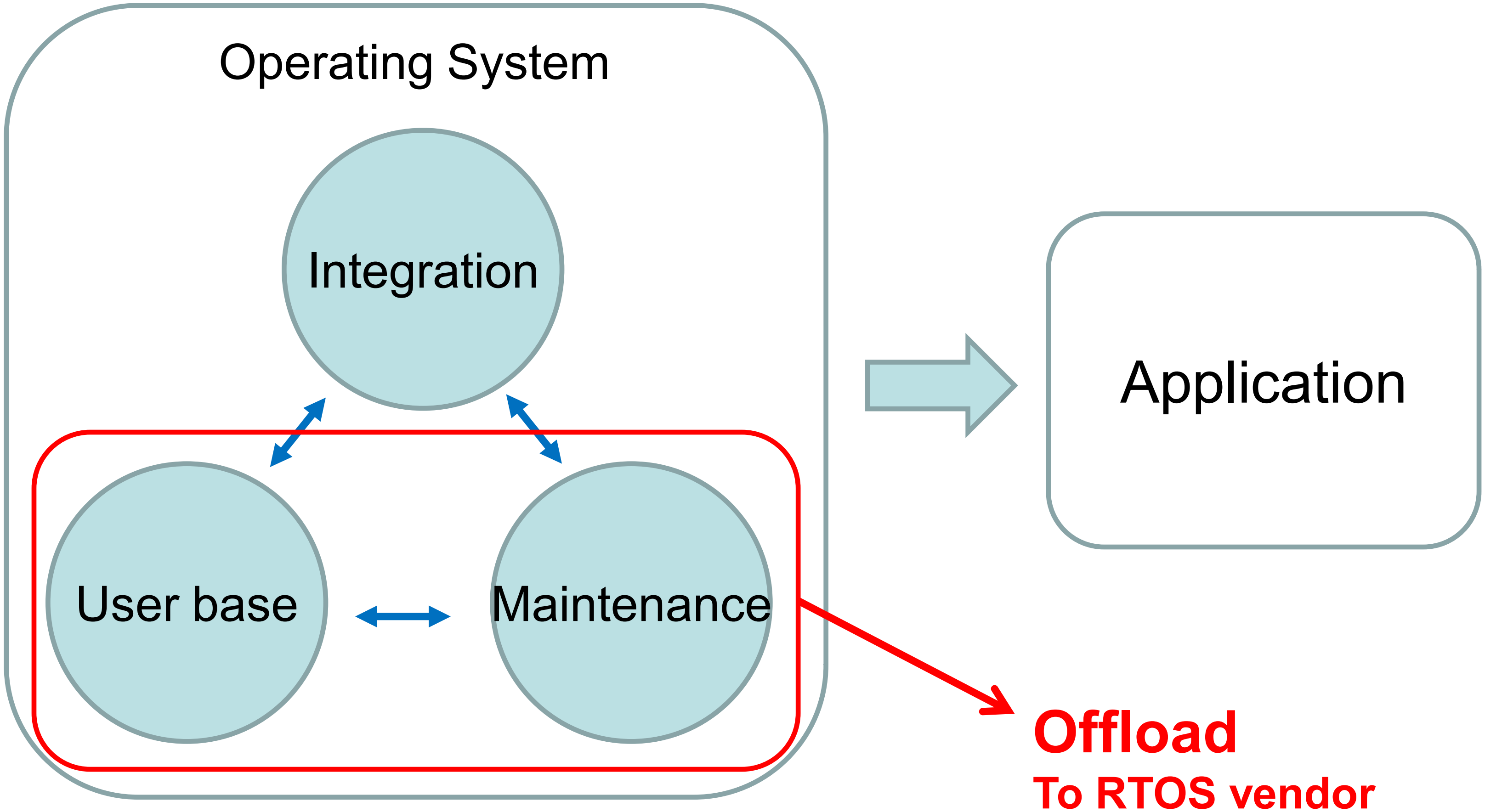
Why a Real Time Operating System?

- Distributed Systems
  - Doing processing at high throughput in time is not enough
  - Need to use operating system facilities for IO and administration
  - For efficient development, need
    - to follow future technology
    - flexibility to add, remove and reassign services or update requirements
    - avoiding dependencies to non Real Time OS behavior or side effects of other jobs.

- RTOS designed for Real Time from ground up
  - Features added in a way which do not violate real time
  - Only needed services are activated

- Real Time Applications
  - Identical treatment like non real time applications
  - Normal operating system APIs can be used
  - Simple communication between real time and non real time
  - Additional facilities to control timing simple and accurate
    - No homegrown limited spinlocks, no delegation to drivers

Focus on Value Add



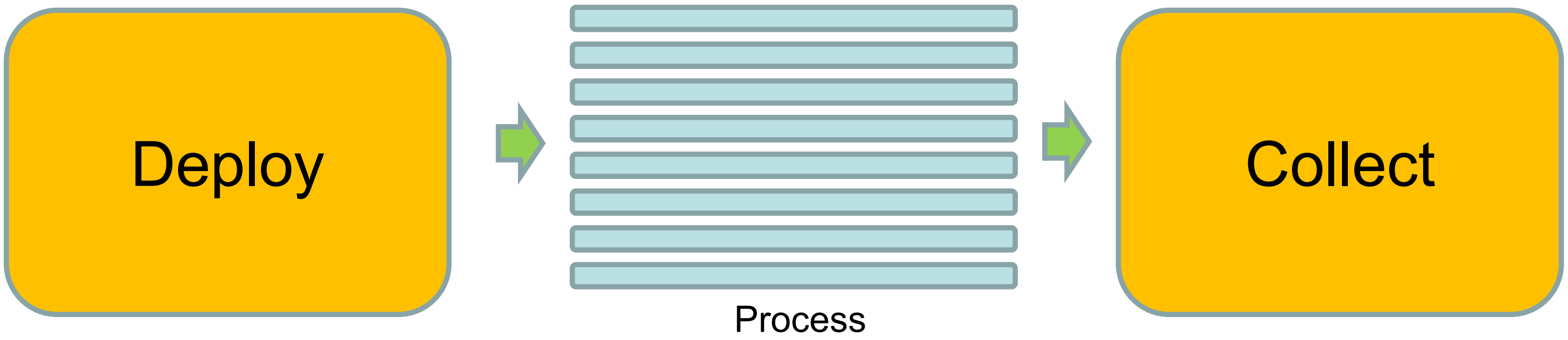
In long term

- Buying commodity is much cheaper as manpower

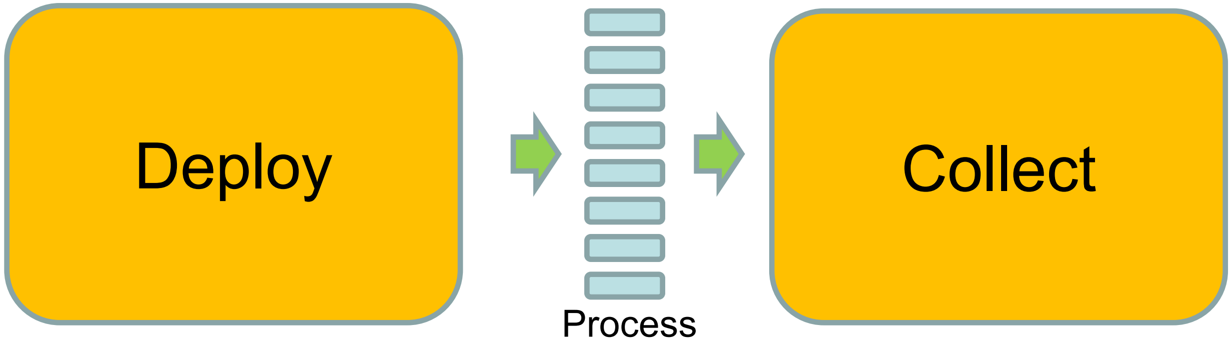
Why Intel x86_64?

- "Too big to fail"
  - Decades to port worldwide existing Software or having fast enough emulators
- Track record of backwards compatibility:
  - 16Bit code of 1978 still executable
  - 32Bit stable since 1985, complete since 1995
  - 64Bit spec stable since 2000, hardware available since 2004 (AMD 2003), ubiquitous since 2008 (Not to confuse with Itanium!)
- Code density & memory bandwidth
- Supercomputers
  - Garching SuperMUC #6 world rank
  - 147456 Sandy Bridge Cores @2.7 GHz
- SIMD - Fully Integrated DSP
  - Guaranteed to be present in 64Bit
  - Auto Vectorization by Compilers
  - Intel understood the challenge of the high performance race
    - Quickly scaling up Vector length, Superscalar, operands.
    - SSE, SSE2, SSE3, SSSE3, SSE4.1 SSE4.2, AVX, AVX2 …
- Compiler ABI redesigned for 64Bit (registers, addressing, SIMD) – efficient!
- Competitors
  - ARM 64Bit multicore and GPUs not mature
  - MIPS, Freescale not following pace, special purpose, declining
  - AMD/VIA follower or low cost

Communication Overhead



Processing dominated



Transport dominated

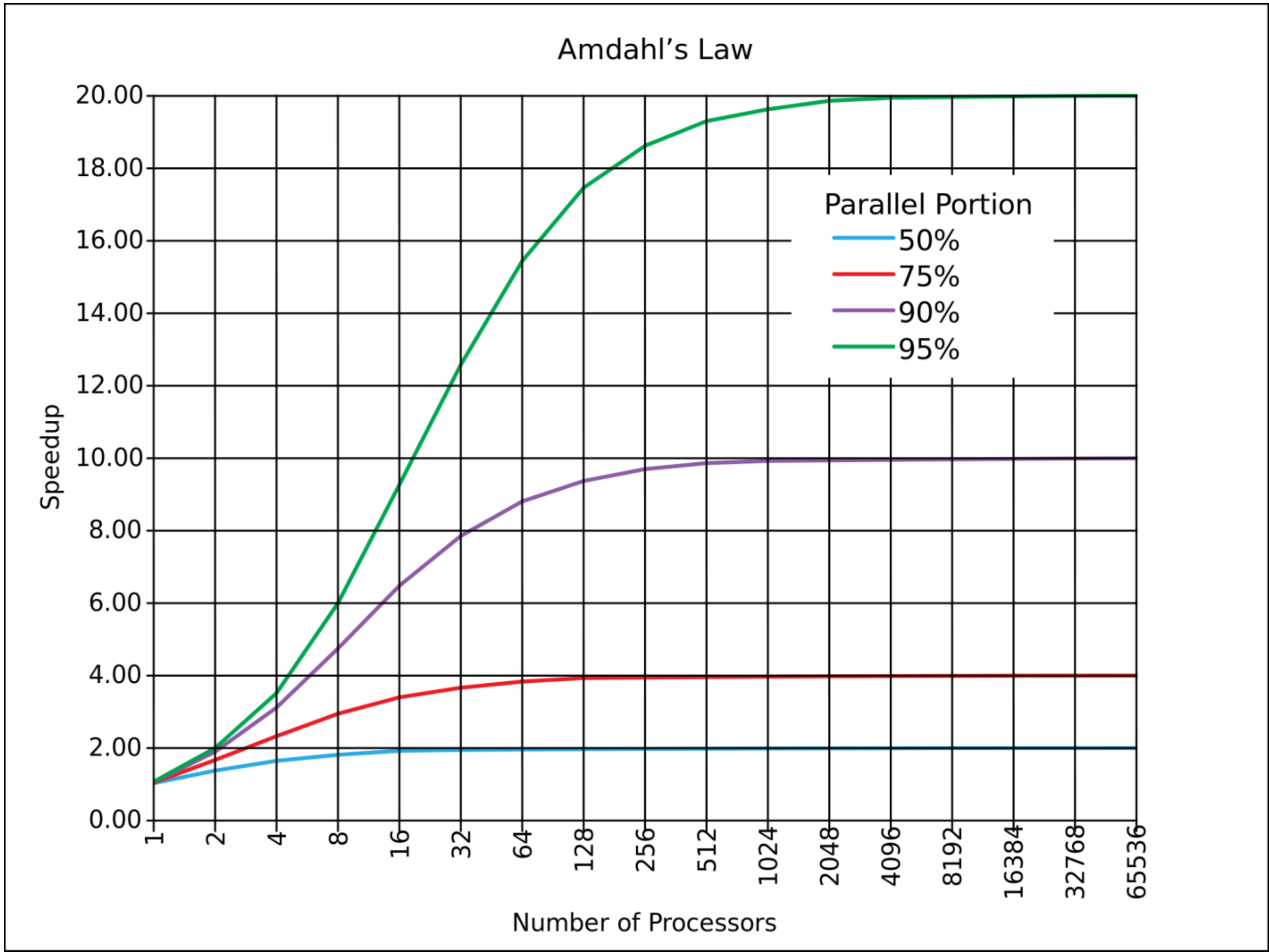
Processing only

Amdahl's Law

- 90% parallelization limits speedup to 10x

  No matter how many CPUs!

- 1967
- A small sequential portion finally dominates!



RTOS Alternatives

- Windows CE
  - Specialized?
  - Becoming Extinct?
- OSE
  - Programming Paradigm?
  - Specialized?
  - CPU support?
- Integrity
  - Specialized?
- QNX
  - Source code?
  - Strategy?
  - CPU support?
- VxWorks
  - In House Competition Linux, but not for RTOS.
  - Successful use in ESO since more as 15 years

VxWorks owned by Intel

- Wind River was purchased 2009
- Sustaining support of x86
- Integration of Intel Software
  - Since VxWorks 6.9 (2011)
  - Intel Integrated Performance Primitives (IPP)
    - algorithm library for signal processing, image processing, matrix operations, etc
  - Intel Compiler (ICC)
    - Highly optimizing for x86
    - SIMD with auto-vectorization (parallelization)
    - SIMD dynamically adjusting to detected CPU features
      - Binary compatibility
      - No recompile

Auto-vectorization Example Code

```
void vmulr( float * __restrict r,
            const float * a,
            const float * b )
{
    int i;

    for (i=0; i<4;++i)
    {
        r[i] = a[i] * b[i];
    }
}
```

```
void vmul( float * r,
           float * a,
           float * b )
{
    int i;

    for (i=0; i<4;++i)
    {
        r[i] = a[i] * b[i];
    }
}
```

Intel - with context hints

```
# icc -c -O2 -m64 -xavx

vmovups (%rsi),%xmm0
vmulps (%rdx),%xmm0,%xmm1
vmovups %xmm1,(%rdi)
retq
```

Accurate Declaration Helps!

Intel - no assumptions

```
# icc -c -O2 -m64 -xavx

vmovss (%rsi),%xmm0
vmulss (%rdx),%xmm0,%xmm1
vmovss %xmm1,(%rdi)
vmovss 0x4(%rsi),%xmm2
vmulss 0x4(%rdx),%xmm2,%xmm3
vmovss %xmm3,0x4(%rdi)
vmovss 0x8(%rsi),%xmm4
vmulss 0x8(%rdx),%xmm4,%xmm5
vmovss %xmm5,0x8(%rdi)
vmovss 0xc(%rsi),%xmm6
vmulss 0xc(%rdx),%xmm6,%xmm7
vmovss %xmm7,0xc(%rdi)
retq
```

GNU - loops

```
# ccpentium -c -O2 -m64

xor    %eax,%eax
movss  (%rsi,%rax,1),%xmm0
mulss  (%rdx,%rax,1),%xmm0
movss  %xmm0,(%rdi,%rax,1)
add    $0x4,%rax
cmp    $0x10,%rax
jne    162 <vmulr+0x2>
repz retq
```

32 Bit code would need 8 instructions more for call frame and setup!

VxWorks Evolution

- 1987 – Introduction
- 199x – VxWorks 5
  - 68K, Sparc, x86, PowerPC, ARM, Mips, SH
  - Mass storage file system
  - kernel configurator
  - Downloadable Kernel Modules
- 2002 – VxWorks 5.5 – still supported in 2012
- 2004 – VxWorks 6 – Application Processes
  - Modeled after Linux
  - private memory, shared lib, resource reclamation
  - Posix PSE52 conformance
  - 68K discontinued, Sparc only for LEON by 3rdparty
- 2005 – VxWorks 6.2 – Hot plug filesys (USB)
- 2007 – VxWorks 6.4 – Long term release
- 2007 – VxWorks 6.5 – Network stack replaced
- 2008 – VxWorks 6.6 – Multicore
- 2009 – VxWorks 6.7 – Compile time configurable OS, source build.
- 2011 – VxWorks 6.9 – 64Bit support
  - Intel Compiler (x86 SIMD Auto vectorization)
  - Performance Primitives (signal processing and math algebra)
- 2012 – VxWorks 6.9.2 – Up to 32 Cores and hyperthreading on Intel CPUs

Conservative

- VxWorks API is remarkably consistent
  - Thousands of APIs same since 1987
  - Very few APIs have changed
    - Multicore: Certain rarely used APIs had to be replaced because of implicit unicore assumptions
      - Workaround available to not being blocked before porting
    - 64Bit: Only pointers and opaque data types changed
      - Backwards compatible types provided
      - No impact if correct types for pointer arithmetic were already used.
      - Compiler warnings added to identify conflicts
- Endian and hardware addresses:
  - Abstraction Layers provided for portability
  - Extremely simple hardware access helpful for rapid prototyping
- Internal Interfaces stable
  - Drivers, BSPs and many OS libraries can be mostly exchanged between versions

➡ Reuse experience as well as large,existing source code

Source Code available

- VxWorks customers are getting full operating system source code
  - Allows keeping control in the future
  - Fixes can be integrated when necessary
  - White box analysis for root of unexpected behavior
  - Scalability
  - Extreme modular
    - Everything is just a library

Tools

- SystemViewer - Logic Analyser
- ProfileScope - Performance
- MemScope - Memory leaks
- StethoScope - Monitor
- CoverageScope

Usage Domains

- Mainline VxWorks:
  - Industrial Control
  - Medical
  - Network Infrastructure
  - Consumer
  - Automotive
  - Military
  - Space
    - James Web Space Telescope
    - various Mars missions, e.g. Curiosity
- Special Versions
  - Airplanes, safety, security
    - (DO-178B, ARINC 653, MILS)

Ecosystem

- Middleware
  - Fieldbusses, graphics, IEEE 1588 PTP, Soft-PLC, algorithms, tools …
- Hardware
  - With BSPs or drivers
- Engineering services
- Consultants
- Support

… by Wind River and 3rd party

ESO AO Test Computing Hardware

- Dell PowerEdge R910
  - System memory size: 128 GB
  - 4 Processor sockets, each with
    - 10 cores, 2.27 GHz, 2.5 MB L2 cache, 24 MB L3 cache
    - Processor name: Intel(R) Xeon(R) CPU E7-4860
- Dell PowerEdge R710
  - 10 Gigabit fibre Intel ethernet card
  - SATA disk
  - System memory size: 24 GB
  - 2 Processor sockets, each with
    - 6 cores, 3.46 GHz, 1.5 MB L2 cache, 12 MB L3 cache
    - Processor name: Intel(R) Xeon(R) CPU X5690
- Both Systems:
  - System memory speed: 1333 MHz
  - CPU family: Westmere Microkernel, 64Bit, SSE4.2
  - 1 Gigabit Copper Intel ethernet card
  - USB disks, CD-ROM drive

Current state

- Reference BSP adapted
- Scripted kernel configuration to have consistency across different BSPs
- ESO VLTSW build system integration
- Porting Guidelines
  - 64Bit, Multicore, endian
- Benchmark lessons:
  1. Take care about transfer of data ownership
     - Small local data set: Scales linear with core count
     - Large heavy used shared memory across all cores:
       - Reaches max performance with only 3 cores
       - More cores as in one processor socket: Time worse as with 1 core ➔ Cache synchronization effect?
  2. On VxWorks, interrupt and task response independent of CPU load!

Task Response Time

```
-> jitterClkStart 4001,0
...
-> jitterReport
...
TASK RESPONSE JITTER
     Samples     Maximum    Average
    ---------  ---------  ---------
       818678     15.993      0.419

    Distribution:
            Time Range          Count              Count (log scale)
      ---------------------  -------  ---------------------------------
          0.000 -      1.117   778096  |********************
          1.117 -      2.234    32066  |***************
          2.234 -      3.352     7087  |*************
          3.352 -      4.469      836  |**********
          4.469 -      5.587      187  |********
          5.587 -      6.704      176  |********
          6.704 -      7.822      124  |*******
          7.822 -      8.939       60  |******
          8.939 -     10.057       44  |******
         10.057 -     11.174       20  |*****
         11.174 -     12.292       16  |*****
         12.292 -     13.409        3  |**
         13.409 -     14.526        2  |**
         14.526 -     15.644        0  |
         15.644 -     16.761        1  |*
         16.761 -     17.879        0  |
```

- While 5 Gbit/s network traffic and full load on all cores
- normal VxWorks configuration – no tuning!

Task Response Time – simple tuning

- While 5 Gbit/s network traffic and full load on all cores
- taskCpuAffinitySet(), vxCpuReserve (), vxbIntReroute(), no USB
- Further tuning candidates
  - Reduce up to 2 µs HPET timestamp hardware overhead by BIOS config for IO prio or replacement

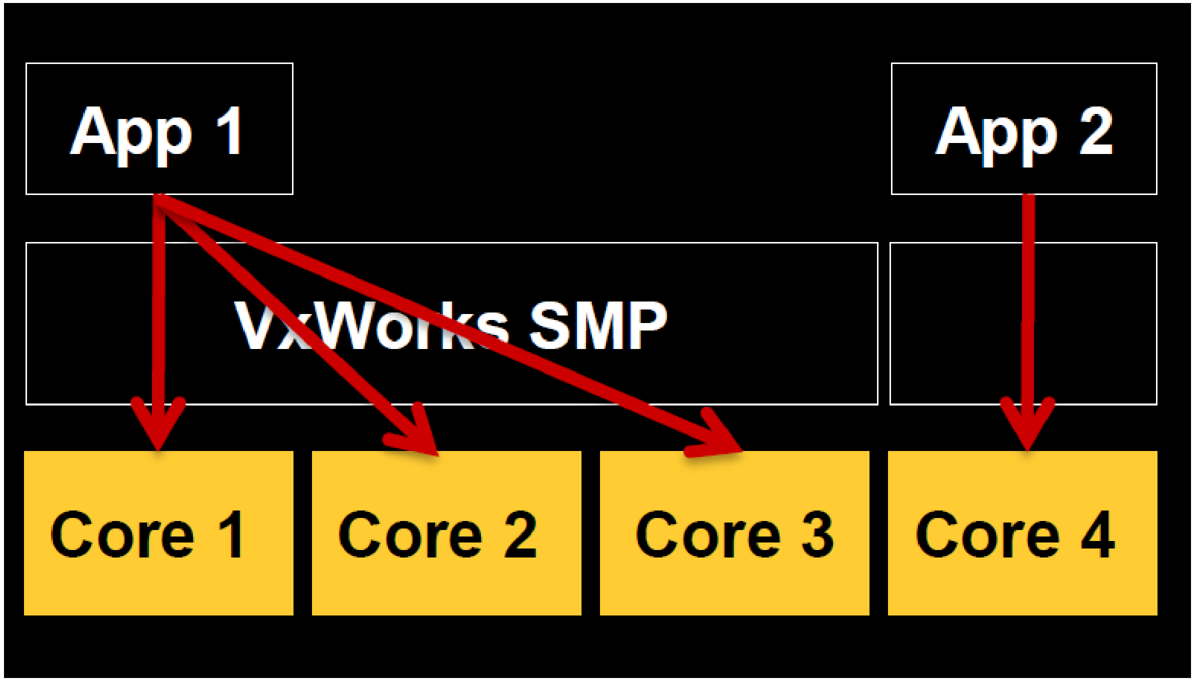
[all Timings in µs]

```
-> jitterClkStart 4001,0
...
-> jitterReport
...
TASK RESPONSE JITTER
      Samples     Maximum    Average
    ---------- ---------- ----------
        812564      3.282      0.349

    Distribution:
            Time Range          Count          Count (log scale)
      ---------------------   -------  ---------------------------------
          0.000 -     0.279   196606  |******************
          0.279 -     0.558   551748  |********************
          0.558 -     0.838    43145  |****************
          0.838 -     1.117     9618  |**************
          1.117 -     1.396     4133  |*************
          1.396 -     1.676     2158  |************
          1.676 -     1.955     1966  |***********
          1.955 -     2.234     1545  |***********
          2.234 -     2.514     1006  |**********
          2.514 -     2.793      524  |**********
          2.793 -     3.073       96  |*******
          3.073 -     3.352       19  |*****
          3.352 -     3.631        0  |
          3.631 -     3.911        0  |
          3.911 -     4.190        0  |
          4.190 -     4.469        0  |
```

Optimizing SMP: CPU Reservation

- Increase CPU specific cache and TLB efficiency by not allowing other tasks to preempt task running on a reserved core and displace cache content



Lessons learned

- Hyperthreading not useful for real time
  - No fair assignment of CPU time, anyway memory bandwidth limit
- Verify driver availability before HW decision
- VxWorks Kernel and OS extremely stable
- ‘Traditional PC’ features working out of the box
- Advanced features for multicore need BSP adaptation
- Multicore systems with one processor sockets are commodity
- For Servers with multiple sockets we were early adopters
  - on processor and interrupt enumeration not all possible cases processed
  - ask Wind River support for debugging assistance, hot fix and integration of the fix into future updates
  - Or contract out the BSP at fixed price.

Thank you

Helmut Tischer

Software Development Division

htischer@eso.org

Abstract

- VxWorks on Intel for real time high performance computing
  - Using a Real Time Operating System (RTOS) is essential to meeting tight timing requirements in a rich application environment without crippling OS and usage in an error prone way on each update.
  - We will see how the widespread RTOS VxWorks including sourcecode access and Tools has kept the pace in the last 25 years and while maintaining broad compatibility has reached support for Posix Processes, up to 32 Cores, 64 Bit, Vector engines on up to date high end hardware.
  - In SPARTA we run VxWorks on COTS Intel Multicore Servers and are porting existing Software to it. We plan to use the easy hardware access to integrate FPGAs, IEEE 1588 Precision Time Protocol and to have a look into available cluster technologies.
  - On x86-64, Vector engines are an inherent part of the mature ABI and compilers. By eliminated communication complexity and transport overhead, and Intel scaling up Vector performancy quickly, we will evaluate how price/effort/performance/portability will compete against GPUs.