RT patch for Celleb

- patch status and performance measurements -

Tsutomu OWA
Corporate Software Engineering Center, TOSHIBA
Nov. 3. 2007

Copyright 2007, Toshiba Corporation.
Contents

• Background
  – realtime-preempt patch (RT patch)

• RT patch status for Celleb/PowerPC64

• Measurements
  – Metrics
    – Measurement Environment
  – Results

• Summary
realtime-preempt patch (RT patch)

- Patch created and maintained by Ingo Molnar, Thomas Gleixner, Steven Rostedt et.al.
  - From at least 2004, steadily merged into mainline
  - Add preemption points in the kernel
    - spinlock → mutex w/ priority inheritance
    - hard/soft interrupts → kernel threads
    - etc.

- URLs
  - “the Wiki Web for the CONFIG_PREEMPT community, and real-time Linux in general”.
  - “A realtime preemption overview”
    - [http://lwn.net/Articles/146861/](http://lwn.net/Articles/146861/)
  - And many other news/materials on the net.
RT patch status - latest version

• patch-2.6.23-rt1 was announced by Steven Rostedt on October 12, 2007
  – Against linux-2.6.23
  – i386, x86_64, arm, mips, sh, powerpc, sparc,…

  patch-2.6.23-rt1 is used in this presentation.

• patch-2.6.23.1-rt5 was released on October 29, 2007
  – patch-2.6.23.1-rt4: October 27, 2007
RT patch status - size

RT patch size

size (byte)

0 500000 1000000 1500000 2000000 2500000 3000000

patch version

14→15 15→16 16→17 17→18 18→19 19→20 20→21 21→22

generic irq

pi mutex

Hrtimer
dyntic
RT patch status - # of files

arch/*

include/*

PowerPC
RT patch status - Celleb/PowerPC64

• **patch-2.6.23-rt1**: Applies cleanly to linux-2.6.23
  – Compile, boot and run tested on Celleb (Cell /B.E., PowerPC64 based)

• **Supported configs for powerpc64**
  – CONFIG GENERIC TIME

  – CONFIG_MCOUNT, CONFIG_FUNCTION_TRACE
    pcsed-1772  0D..2 6867us : deactivate_task <pcscd-1772> (-2 1)
    pcsed-1772  0D..2 6867us : dequeue_task (deactivate_task)
  <idle>-0 0D..2 6870us : __switch_to (__schedule)

  – CONFIG_CLOCKEVENT

  – And others?

  – CONFIG_LOCKDEP, CONFIG_STACKTRACE, CONFIG TRACE_IRQFLAGS
    • Coming…
RT patch status - Celleb/PowerPC64 (Cnt.)

- **Patches not included in 2.6.23-rt1**
  - “RT: fix spin_trylock_irq”
    - From sebastien.dugue@bull.net
    - merged into 2.6.23-rt2
  - “Hook compat_sys_nanosleep up to high res timer code”
    - [http://lkml.org/lkml/2007/10/14/190](http://lkml.org/lkml/2007/10/14/190)
    - merged into 2.6.23-rt?
  - “powerpc: 64 bits irqtrace / lockdep support”
    - From Benjamin Herrenschmidt on 15 Oct 2007
    - [http://patchwork.ozlabs.org/linuxppc/patch?id=14172](http://patchwork.ozlabs.org/linuxppc/patch?id=14172)
    - Not RT specific but useful.
    - merged into 2.6.25?
RT patch status - Celleb/PowerPC64 (Cnt.)

• **Patches not included in 2.6.23-rt1**
  
  – “replace preempt_schedule w/ preempt_schedule_irq”
    • From Tsutomu OWA
  
  – “Implement clockevents driver for powerpc” and its series
    • From Tony Breeds
    • [http://patchwork.ozlabs.org/linuxppc/patch?id=13350](http://patchwork.ozlabs.org/linuxppc/patch?id=13350)
    • Still in discussion? **Not sure…**
Measurement

- Metrics
- Measurement Environment
- Results
Measurements - Metrics

• Previous works
  – Many test results for x86
  – Less test results for other architectures / platforms
  – Adhock-metrics (often) specific to each platform
    • Different / too old kernel base version
    • Different period of time to measure
    • Different load, etc,etc.
    • http://elinux.org/Realtime_Testing_Best_Practices

• What we’d like to have!
  – Common test cases in order to compare results in a consistent manner
    • So that we’ll be on the same ground
  – “IBM Test Cases” and “Cyclictest”
    • are widely used in the RT community
    • could be common testbeds
Measurements - Metrics (Cnt.)

• IBM Test Cases
  – “These test cases for testing a -rt kernel were contributed by IBM's Real-Time Linux development team. They include mostly functional tests, although some performance tests are slowly being added. If you would like to contribute, please use the discussion list above and contact User:dvhart. “

• Cyclictest
  – “Cyclictest is a high resolution test program, written by Thomas Gleixner ”
IBM Test Cases used in this presentation

- **gtod_latency**
  - “to measure the time between several pairs of calls to gettimeofday().” (from gtod_latency.c)
  - On a SCHED_FIFO (99) priority thread.

- **async_handler**
  - “Measure the latency involved with asynchronous event handlers. Specifically it measures the latency of the pthread_cond_signal call until the signalled thread is scheduled.” (from async_handler.c)
  - Two threads with priority set to 89.

- **sched_latency**
  - “A thread is created at a priority of 89. It periodically sleeps for a specified duration(PERIOD).
    - The delay is measured as
      \[
      \text{delay} = (\text{now} - \text{start} - i \times \text{PERIOD}) \text{ converted to microseconds}
      \]
      where, \( \text{now} = \text{CLOCK_MONOTONIC gettime in ns} \), \( \text{start} = \text{CLOCK_MONOTONIC gettime at the start of the test} \), \( i = \text{iteration number} \), \( \text{PERIOD} = \text{the period chosen} \)” (from sched_latency.c)
#if defined(__i386__)
#define rdtscll(val)    __asm__ __volatile__("rdtsc" : "=A" (val))
#elif defined(__x86_64__)
#define rdtscll(val)   
do {
    uint32_t low, high;
    __asm__ __volatile__ ("rdtsc" : "=a" (low), "=d" (high));
    val = (uint64_t)high << 32 | low;
} while(0)
#endif

static inline int atomic_add(int i, atomic_t *v)
{
    int __i;
    asm volatile( 
        "lock; xaddl %0, %1;" 
        :"=r"(i) 
        :"m"(v->counter), "0"(i));
    return i + __i;
}
IBM Test Cases - patch for powerpc

- [http://www.mail-archive.com/linux-rt-users@vger.kernel.org/msg01830.html](http://www.mail-archive.com/linux-rt-users@vger.kernel.org/msg01830.html)
  (local copy)

Re: [RFC] [PATCH] powerpc
Re: [announce] IBM RT Test Cases v.0.3
Darren Hart
Thu, 01 Nov 2007 10:04:33 -0800

On Fri, 2007–10–26 at 13:59 +0900, Tsutomu OWA wrote:
Hello Darren Hart,

> At Fri, 20 Jul 2007 15:40:58 -0700, Darren Hart wrote:
> Please download the tarball
> 〈snip〉 〈snip〉 〈snip〉
> This patch adds powerpc version of rdtscll() macro which actually reads
> the timebase register and powerpc version of atomic_inc() to compile.
> Compile and run tested on a Celleb (a powerpc64 machine).
> Thank you for the patch. I think we should perhaps move all the rdtscl stuff into a header file
> with a more generic name.. rdsystimer or something.
> 〈snip〉 〈snip〉 〈snip〉

> By the way, would you mind if I use and/or refer to your test results found at
> : Please feel free to use the results, but do site the ols2007 publication as the source.
Measurement

• Metrics

• Measurement Environment

• Results
Measurement env.

- **Cell Reference Set**
  - [http://www.toshiba.co.jp/tech/review/2006/06](http://www.toshiba.co.jp/tech/review/2006/06)

- **HW**
  - Cell/B.E. (PowerPC64 based PPU)

- **SW**
  - Linux on a Hypervisor OS
Measurement env. (Cnt.)

- **Kernel Configuration**
  - Linux-2.6.23 vanilla
    - CONFIG_PREEMPT
  - Linux-2.6.23 + patch-2.6.23-rt1
    - CONFIG_PREEMPT_RT

- **Userland Configuration**
  - Fedora 7
    - gcc version 4.1.2, thread model: posix
    - glibc-2.6
    - run level: 3

- **Load**
  - Make linux-2.6.23 kernel (% make)
    - % uptime
      17:43:21 up 3:01, 2 users, load average: 1.64, 0.67, 0.36

More than 15 daemons are running.
Measurement

• Metrics

• Measurement Environment

• Results
cyclic test – vanilla preempt w/o load

Warning: High resolution timers not available
cyclic test – vanilla preempt w load

Warning: High resolution timers not available
cyclic test – rt1 w/o load

![Graph showing latency (us) against iteration for cyclic test – rt1 w/o load. The graph displays a scatter plot with horizontal lines indicating latency values over iterations.](image)
cyclic test – rt1 w load
## gtod_latency

<table>
<thead>
<tr>
<th>latency (us)</th>
<th>vanilla preempt w/o load</th>
<th>vanilla preempt w/ load</th>
<th>rt1 w/o load</th>
<th>rt1 w load</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>842842</td>
<td>842820</td>
<td>168791</td>
<td>167132</td>
</tr>
<tr>
<td>1</td>
<td>156986</td>
<td>157017</td>
<td>830350</td>
<td>831984</td>
</tr>
<tr>
<td>2</td>
<td>13</td>
<td>7</td>
<td>9</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>75</td>
<td>82</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>71</td>
<td>59</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>3</td>
<td>14</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>4</td>
<td>3</td>
<td>255</td>
<td>15</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>2</td>
<td>565</td>
<td>46</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0</td>
<td>12</td>
<td>460</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>331</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>12</td>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>18</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
sched_latency – vanilla preempt w/o load
sched_latency – rt1 w/o load
sched_latency – rt1 w load
async_handler - vanilla preempt w load
async_handler – rt1 w/o load
async_handler – rt1 w load
## Comparison with x86

<table>
<thead>
<tr>
<th>Test Case</th>
<th>2.6.21.5-ols01-LOADED</th>
<th>2.6.21.5-rtl4-ols01-LOADED</th>
<th>2.6.23 vanilla preempt w/ load</th>
<th>2.6.23 rtl w/ load</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cyclic_test</code></td>
<td>Min: 3.0 us Max: 48.0 us Avg: 7.943 us</td>
<td>Min: 4.0 us Max: 22.0 us Avg: 8.065 us</td>
<td>Min: 28 us Max: 1.281 us Avg: 51.0572 us</td>
<td>Min: 9 us Max: 87 us Avg: 23.738 us</td>
</tr>
<tr>
<td><code>gettimeofday</code></td>
<td>Minimum: 1 us Maximum: 243 us Average: 1.242956 us Standard Deviation: 0.593761 us</td>
<td>Minimum: 1 us Maximum: 16 us Average: 1.240043 us Standard Deviation: 0.450524 us</td>
<td>Minimum: 0 us Maximum: 126 us Average: 0.157764 us Standard Deviation: 0.386613 us</td>
<td>Minimum: 0 us Maximum: 53 us Average: 0.337757 us Standard Deviation: 0.412978 us</td>
</tr>
</tbody>
</table>

Celleb specific optimization

- arch/powerpc/mm/ltb_64.c: hpte_need_flush()

```c
#define CONFIG_PREEMPT_RT

/*
 * Since flushing tlb needs expensive hypervisor call(s) on celleb,
 * always flush it on RT to reduce scheduling latency.
 */

if (machine_is(celleb)) {
    __flush_tlb_pending(batch);
    return;
}
#endif /* CONFIG_PREEMPT_RT */
```
Celleb specific optimization (Cnt.)

- Flushing a tlb needs expensive hypervisor calls.

<table>
<thead>
<tr>
<th>Process</th>
<th>CPU</th>
<th>Time</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>3us</td>
<td>.flush_hash_range (.__flush_tlb_pending)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>3us</td>
<td>.flush_hash_page (.flush_hash_range)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>4us</td>
<td>.beat_lpar_hpte_invalidate (.flush_hash_page)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>5us</td>
<td>.__spin_lock_irqsave (.beat_lpar_hpte_invalidate)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..5</td>
<td>6us+</td>
<td>.beat_lpar_hpte_getword0 (.beat_lpar_hpte_invalidate)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..5</td>
<td>13us</td>
<td>.__spin_unlock_irqrestore (.beat_lpar_hpte_invalidate)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>13us</td>
<td>.flush_hash_page (.flush_hash_range)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>14us</td>
<td>.beat_lpar_hpte_invalidate (.flush_hash_page)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>14us</td>
<td>.__spin_lock_irqsave (.beat_lpar_hpte_invalidate)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..5</td>
<td>15us+</td>
<td>.beat_lpar_hpte_getword0 (.beat_lpar_hpte_invalidate)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..5</td>
<td>18us</td>
<td>.__spin_unlock_irqrestore (.beat_lpar_hpte_invalidate)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>19us</td>
<td>.flush_hash_page (.flush_hash_range)</td>
</tr>
<tr>
<td>inetd-358</td>
<td>OD..4</td>
<td>20us</td>
<td>.beat_lpar_hpte_invalidate (.flush_hash_page)</td>
</tr>
</tbody>
</table>

- Was very effective on linux-2.6.21
- But not effective on linux-2.6.22 and later.
Future Work

• **Performance improvement**
  - Try to
    • Figure out reasons for these spikes using latency tracing mechanism.
    • (Back)Port TRACE_IRQFLAGS and STACKTRACE patches to RT.
    • Figure out how to put “same” load between different platforms

• **Real-world applications?**
Summary

• gettimeofday() appears to be slow down for RT on PowerPC.
  – generic version of gettimeofday() needs reference to xtime (and locks) while original powerpc version of gettimeofday() does not.

• sched_latency on vanilla shows the similar result with that of x86 non RT kernel (except for 1ms vs 4ms due to HZ differences).

• Maximum latency of cyclictest and sched_latency for rt1 are less than 90us. Seems fairly good.

• These Metrics could be a good references.

• Some more debugging/investigation is necessary to improve latencies.
Thank you!

Let’s try cyclic test and IBM Test Cases!