Porting RT-preempt to Loongson2F

Wu Zhangjin, Nicholas Mc Guire
Distributed & Embedded System Lab, SISE, Lanzhou University, China
Tianshui South Road 222,Lanzhou,P.R.China
wuzhangjin@gmail.com, der.herr@hofr.at

Abstract

Loonsong2F, is a MIPS compatible single core CPU, it is a low-power, mid-range performance processor and therefore suitable for industrial usage. This paper focus on the effort of porting the RT-preempt extension of Linux to this processor based platform(FuLoong2F). we will briefly introduce the features of this platform, and outline the motivation of selecting RT-preempt, and then discuss the technical issues of porting RT-preempt to a new architecture, and describe the specifics of the platform. Further we cover the non-technical issues of how to integrate such work into mainstream development, and finally present some preliminary benchmark results based on community tool (notably cyclictest).

KeyWords: RT-preempt, Loongson, Real Time, Linux, industrial applications

1 Introduction

The 2nd generation of CPUs from Loongson[5] designed by ICT,CAS in China is currently at the 2F release, with the 2G to be expected this year. This MIPS compatible single core CPU is a low-power, mid-range performance CPU primarily targeting desktop and net-books. The features are listed in table 1.

CPU Core Frequency	800MHz-1GHz
Power Consumption	4 Watt.
Power Management	scalable cpu frequency
Built-in	North bridge,DDR2 Controller

TABLE 1: *Main features of Loongson2F*

The FuLoong(2F)[6] Mini PC made by Lemote, as a Loongson2F based machine, the total power consumption is only 10 watt which ensures quiet operation, and With small size (190x145x37mm) and light weight (0.85KG). It provides a large set of standard interfaces, including 4 USB2.0 Host High-Speed Interface, On-board dual LAN, 1 mini IDE interface, RS232 compliant serial port, 1 IR control receiver interface and it has reserved extended interfaces for wireless network card and LCD display, the form is shown in picture 1.

FIGURE 1: *FuLoong(2F) Mini PC*

Due to its low power-consumption, fan-less operation and integration in form-factors suitable for industrial usage, we have been working on porting suitable RTOS to it, the primary targets for this effort have been the XtratuM hypervisor and mainline RT-preempt[7]. We will not further cover XtratuM (which is in an early development stage) but rather focus on the mainline RT-preempt, to enhance its usability in industrial applications requiring reliable real-time services.

In the following sections, we will briefly outline the motivation of selecting RT-preempt and then focus on the technical issues of porting RT-preempt to a new architecture and the specifics of the FuLoong(2F) platform. Further we cover the attempt of integrating such work into mainstream development, and finally present some preliminary benchmark results based on cyclictest[14]).

2 Why RT-preempt

The FuLoong(2F) machine, as introduced above, utilizes free software, thus is provided with source code for all system software and all standard services. Currently, it only runs Linux kernel based operating systems, including Lemote Loonux, Debian, Gentoo, Gnewsense (though Vxworks, Window CE reportedly can also run on it, but are not in the public market yet).

To meet the basic requirement of real-time services in industrial applications and also to make the RTOS porting project predictable, we selected RT-Preempt, after comparing several RTOSs for MIPS, including Vxworks, RTLinux/GPL, RTAI, Xtraum+PartiKle.

Vxworks is a commercial RTOS which is not easy to get, RTLinux/GPL is bound to the 2.4.X series of kernels and it's migration to XtratuM is not yet completed, RTAI for MIPS is too old, XtratuM+PartiKle is in its early development stage and targets very specific use-cases. But RT-preempt, as an Real Time Extension to mainline Linux, although the response time will likely always be a bit lower than RTLinux/GPL or a comparable dedicated RTOS, but is ready to "control an industrial welding laser"[7]. From the current development of RT-Preempt and its support by OSADL it can be expected that RT-Preempt will be able to cover a majority of the typical industrial real-time demands in the near future. Further one of the most attractive point is that there is an existing MIPS-specific support: 2.6.26.8-rt16, although it can't run on FuLoong(2F) directly.

And other than the realtime environment, RT-preempt is provided with the familiar Linux/POSIX programming environment, file systems, networking, and graphics which will help to expand the application area and also ease the Real Time Programming a lot. Although the current RT-Preempt source is a patch against Linux baseline, the active development of RT-preempt (the latest version is 2.6.31-rc4-rt1)[8] and the success of lots of upstream patches indicate that it can be expected that the whole RT-preempt will go mainstream within the next view mainstream releases. It should be noted that the upcoming 2.6.31 kernel already includes most of the essential RT-Preempt extensions and there are extensive efforts to unify core API (notably threaded interrupts).

3 Porting RT-preempt

The basic procedure of porting RT-preempt to Loongson2F includes,

1. download the original RT-preempt patches from its repository[8].
2. build or download cross-compile environment for MIPS/Loongson2F(only gcc>=4.4 has Loongson2F-specific support)
3. fix the grammar errors with the help of cscope, objdump, nm, readelf...
4. read, analyse the x86 & powerpc specific of RT-preempt
5. port the arch-dependent parts(including basic RT-preempt and debugging tools) to MIPS/Loongson2F platform
6. debug it on Qemu/Malta and FuLoong(2F)
7. optimize with KFT/Ftrace/Perf_counter/Oprofile, find out the hotspots and tune the different latencies(interrupt latency, schedule latency...)
8. benchmark with cyclictest
9. release & try to upstream by posting to the respective mailing lists and contacting the respective maintainers (MIPS and RT-Preempt).
10. fix the code once the first reviews come in and submit again

but this section will mainly focus on three parts, including how to get linux-2.6.26.8-rt16 running on FuLoong(2F) initially, migrate mips-specific support to linux-2.6.29-rc6 and later versions and real time debugging tools(ftrace, perf_counter) porting along with real-time performance optimization.

3.1 Getting linux-2.6.26.8-rt16 onto FuLoong(2F)

The original linux-2.6.26.8-rt16 includes basic MIPS support, but after fixing up a few compiling errors and running it on Qemu/Malta, it could not yet run on FuLoong(2F) directly. The problem is found out with the help of PMON(BIOS & Bootloader), early_printk, watch register of Loongson2F, and serial port, and at last locate the bug in include/asm-mips/atomic.h. The original RT-preempt tried to force all of the MIPS processors to use the *ll,sc* instruction to do the atomic operations if PREEMPT_RT was enabled, but it ignored the following two situations:

- No *ll,sc* instructions in the MIPS processor
- No macro cpu_has_llsc defined to 1

if any of the above condition occurs, the result will be: all of the atomic operations will be always empty or just before the cpu_data[0].options variable is initialized(if the macro is not defined to 1, it will be defined as cpu_data[0].options & MIPS_CPU_LLSC).

Unfortunately, although Loongson implemented the *ll,sc* instructions, but there is no macro cpu_has_llsc defined as 1, so, it failed when booting.

So, the corresponding fix is removing the buggy conditional compilation statement, and at the same time add a Loongson-specific cpu-feature-overrides.h to define the macro. To reduce the size and optimize the performance of Linux, other macros (related to describe the specific CPU features of the Loongson 2F) were also added in (i.e. #define cpu_dcache_line_size() 32, etc).

These macros will save about several hundred Kb of the kernel image, and will help the gcc compiler to remove the condition statement branches and at last optimize the performance.

3.2 Migrate mips-specific support to latest version of RT-preempt

As Linux is a rapidly developing code-base, there is little point in beginning a port with an out of date kernel (any kernel older than two releases is out-of-date) as this will break when trying to upstream any such patch-set. After making linux-2.6.26.8-rt16 working on FuLoong(2F) as a first step, We planned to do some optimization, but found out that ftrace does not works well on it and almost at the same time, found that linux-2.6.29-rcX-rtY was already released. Although 2.6.29-rcX-rtY was without mips-specific support, but the ftrace seems more mature, and there was a new perf_counter debugging tool introduced, so, We stopped to work on linux-2.6.26.8-rt16 and moved on to linux-2.6.29-rcX-rtY.

Since at that moment, the latest Linux for loongson is linux-2.6.27.1, the next step was to migrate Linux for loongson to linux-2.6.29 at first (mainstream not the rt-preempt variant), and then migrate the mips-specific RT-preempt. When doing this job, We have referred to these wonderful resources [9, 10, 2, 1] (one wonders where the claim that the Linux kernel is not well documented comes from).

To simplify the description of this migration, it's better to refer to the commit(70e06e) log of the latest mips-specific RT-preempt in the linux-loongson/2.6.29/rt-preempt branch of the rt4ls git repository[11], The main changes:

1. un-thread the timer and cascade interrupt

   Add IRQF_NODELAY to the irq flags to un-thread the MIPS timer and the south bridge 8259 cascade interrupt. we must use the old spinlock if un-threading them, otherwise there will be deadlock. and although the i8253 timer is also un-threaded, but this MIPS timer is necessary instead of i8253. for it provides with higher precision clock for real time requirement.

2. using mutexed spinlock instead of old spinlock-/BKL in RT-preempt

   Rename old raw_spinlock/rw_t to __raw_spinlock/rw_t, rename old local_irq_disable/enable to raw_local_irq_disable/enable, do real irq disable/enable operation on exception, system call.

3. using mutexed semaphore instead of old semaphore via adding a non-RT-preempt specific rwsem.h

4. reducing scheduling latency via adding a scheduling point in do_signal

Note that the FuLoong-specific part is very small, only need to un-thread the cascade interrupts.

3.3 Porting debugging tools & performance tuning

3.3.1 Porting ftrace

ftrace[13] comes from "function tracer", but recently it becomes a tracing framework, various new tracers have been added via it, such as irqsoff, preemptoff, context switches tracers and so forth.

The original ftrace(static ftrace, dynamical ftrace, function graph tracer and system call tracer) is arch-dependent, which need to implement the arch-specific _mcount relative functions. And about the new tracers, although it is available, but has little value for real time tracing since they use the jiffies-based trace_clock_local(call sched_clock()) to record the local tracing times, so, they have very low timing precision(1/HZ, about 1ms when HZ is 1000), therefore the mips-specific sched_clock() needed to be implemented to get high timing precision.

- mips-specific _mcount

  when enabling HAVE_FUNCTION_TRACER, the -pg option of gcc will be enabled to insert an _mcount function call to the entry of any function without the "notrace" annotation or the functions in the files explicitly compiled without the -pg option(i.e CFLAGS_REMOVE_ftrace.o = -pg), the function call looks as follows.

```
// save call_site's next pc to at
move    at,ra
// save next pc to ra and call _mcount
jal     _mcount
```

and the _mcount function itself works like a wrapper, if ftrace is disabled in user-space, it just return back to the caller.

```
jr      ra
move    ra,at
```

otherwise, it will call a real tracing function, and transfer the arguments in the *at* register and the *ra* register with an negative offset of sizeof(*jal _mcount*).

When enabling HAVE_DYNAMIC_FTRACE and HAVE_FTRACE_MCOUNT_RECORD, the calling sites to the function _mcount are recorded to an ELF section(__mcount_loc), so, there is a possibility to replace this calling sites by a NOP operation (with more or less no impact) or changed back to a real call to _mcount and then we can indicate the functions to trace or filter.

and what about the HAVE_FUNCTION_GRAPH_TRACER? it is actually a simulation of the -finstrument-functions option of gcc which can generate instrumentation calls for entry and exit to functions. since the -pg only insert a call to _mcount at the entry to functions, so, need to simulate an exit to them, how? let's remove it's vail:

```
func:
  jal _mcount -> prepare_function_return
  ...   (replace ra by return_to_handler)
  ...
  jr ra -> return_to_handler
             -> ftrace_return_to_handler
```

As the above description shows, when entering into the _mcount function, it call prepare_function_return and try to save the old return address, replace *ra* by return_to_handler and meanwhile, record the *calltime*, then when return from func, dont really return immediately rather call return_to_handler and then ftrace_return_to_handler, this function will record the *rettime*. Next calculates the function duration via *rettime-calltime*, and at last, does the real return to the saved return address.

- mips-specific sched_clock()

  In x86, there is a *tsc*(64bit long) based high precision timing function sched_clock(), but this function of MIPS is jiffies based since although there is a similar register($9, clock counter) in MIPS, this counter is only 32bit long and thus incurs frequent roll-over.

  But for real time tracing, the old millisecond level timing precision is not enough, and because the clock counter of MIPS has half of the CPU frequency, if the main frequency of CPU is 800MHz, the timing precision could reach 1/(400*10^6Hz) = 2.5ns. So, obviously for RT we'd better implement a new sched_clock() based on this counter and the according roll-over handling needs to be consider. To avoid influencing the other parts of linux-mips, we only enable it for ftrace, here it is.

```
unsigned long long native_sched_clock(void)
{
  u64 current_cycles;
  static unsigned long old_jiffies;
  static u64 time, old_cycles;

  preempt_disable_notrace();
  /* update timestamp to avoid missing
     the timer interrupt */
  if (time_before(jiffies, old_jiffies)) {
    old_jiffies = jiffies;
    time = sched_clock();
    old_cycles = clock->cycle_last;
  }
  current_cycles = clock->read();

  time = (time +
  cyc2ns(clock, (current_cycles - old_cycles)
             & clock->mask));

  old_cycles = current_cycles;
  preempt_enable_no_resched_notrace();

  return time;
}
```

3.3.2 Porting perf_counter

"The performance counters subsystem for Linux adds a new system call (sys_perf_counter_open) and provides a new tool(perf) that makes use of these new kernel capabilities. This subsystem and relative tool is new in that it tries a new approach at integrating all performance analysis related tools under one roof."[12]

It has similar arch-specific APIs as Oprofile, and since Loongson has two performance counters and its Oprofile support is provided in the kernel already it's easy to migrate from Oprofile. At present, this Loongson-specific perf_counter is only implemented in linux-2.6.29-rc6 and maybe out-of-date therefore not suitable to be introduced any more.

3.3.3 Real Time performance tuning

It's turn to tune the RT performance of mips-specific RT-preempt, so suitable tuning strategies need to be consider. When on the tuning procedure, we were very surprised to have found the paper [3], which gave us some very detail tuning methods, so not to repeat that paper, here is just list of the basic strategies:

1. compare compiler version
2. compare kernel versions and config options
3. tuning different latencies via kernel tracers
4. tuning via hotspots

Currently, the last step above is only a placeholder there, no deep try yet for that method is not that easy to enforce. The basic principle is catching the function level hotspots via ftrace or KFT and then getting the source code line level hotspots via Oprofile or kgcov, at last tuning the kernel via algorithm improvement, instruction sequence adjustment and other hardware specific low-level fixups.

4 How to upstream it

We have tried our best to push the whole MIPS/loongson-specific RT-preempt to mainline, but up to now, only a few parts were accepted While this sounds bad, the point is that getting patches into main-line (kernel/MIPS or RT-preempt) require adequate review by the respective subsystem maintainers, testing, coding-style related fixes, documentation and its not easy to get it right the first time. So the submissions to the respective mailing lists have resulted in changes to the patch both content wise and format wise, like moving it all to git - so it can be pulled into the respective tree, which is a standard method in the kernel development community.

This process is still on-going and has not yet resulted in the proposed patches being included (but they have not yet been kicked out either) and we hope to get the mainline support as well as the RT-preempt support to the upstream trees by the end of this year.

Currently, the MIPS/Loongson2F specific RT-preempt and ftrace,KFT,kgcov are maintained in the following git repository[11]. which also include linux-loongson kernel(>=2.6.29) for upstream: git://dev.lemote.com/rt4ls.git

5 Benchmarking

For benchmarking a wide range of tools is available in the Linux kernel, we will not go into the details of each and every tool but rather focus on the most important RT benchmarking tool: cyclictest.

"cyclictest measures the delta from when it's scheduled to wake up from when it actually does wake up"[4].

to cover the mostly possible application environments when benchmarking, we add different load background as follows,

1. idle system
2. I/O load via about 50 "find /" background
3. Network load via "ping -f "(ping flood)
4. X window
5. the above combination
6. X window & Mplayer(Video)
7. the above combination

and the mostly possible interval(us) options of cyclictest are used, including 100, 200, 500, 1000, 2000, 5000, 10000. to ensure the result is more credible, all of the tests are executed 100,000 times(loops). besides, The test thread runs in all cases with SCHED_FIFO, priority 80 and clock_nanosleep.

```
cyclictest -p80 -t1 -n -i$interval -l100000 -v
        > cyclictest_$interval.log
```

and here lists the other testing environment for result reproducible.

1. gcc 4.4 with "-march=loongson2f"
2. binutils 2.19.1 with a patch[15] from Lemote
3. cyclictest with a patch[16]

Table 2 and picture 2 shows the statistic result.

Load	Max	Avg.	Std.Dev.
idle	27	6.986	0.684
ping	32	8.144	1.548
x	34	7.312	1.500
find	68	15.553	5.888
ping,x,find	60	15.163	5.092
x,mplayer	99	13.132	7.678
ping,x,find,mplayer	82	15.902	5.764

TABLE 2: *RT-preempt testing report*

the first column is the load, the following three ones are event response times in microseconds, they are maximal, average, and standard deviation individually. the difference among different intervals are not shown in this picture to avoid the table too long, but they are coped with together(cyclictest_*.log).

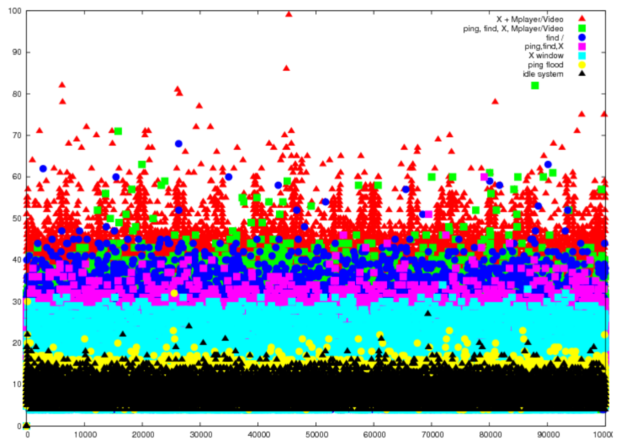

FIGURE 2: *RT-preempt testing report*

The X axes is the loops counts, the Y axes is the event response time in microseconds.

Currently there do not seem to be any data collections on RT-preempt values that allow an easy comparison of the Loongson platform to others, never the less we give the following references [14, 17].

6 Conclusion

The Loongson 2F has favorable hardware properties for the use in industrial applications, its GNU/Linux support made it well suitable for the integration into existing projects - what was missing though is a widely accepted and stable RTOS support, with RT-preempt this is now available for the Loongson 2F. Further it showed that joining an open community like the kernel development community is not as complicated as it might seem and that they are very supportive, allowing an efficient and swift port to a new architecture - which makes GNU/Linux an even more attractive target (RT)OS for hardware companies.

While the RT aspects initially were the main focus it quickly became evident that the tooling capabilities are at least of equal importance and here clearly the capabilities of the Linux kernel and the RT-preempt community proved to be mature and complete, allowing efficiency in the porting job and in the assessment of the results.

We hope that the next generation of Loongson CPUs (2G) will not only support RT-preempt but find its way into mainline Linux from the very start - this will be our next major challenge after concluding the work of up-stream integration of the current Loongson 2F work.

7 Acknowledgment

We would like to thank the MIPS maintainer Ralf Bechle, and the RT-preempt maintainer, Thomas Gleixner, and also the other contributors from the FLOSS communities, for their support and critical review comments. Further we thank the folks at Lemote for supporting this effort, notably Dr. Zhang Fuxin. Last (but not least) the members of DSLab for discussions and support for this undertaking.

References

[1] Steven Rostedt & Darren V. Hart, *Internals of the RT Patch*, P161-172 of Proceedings of the Linux Symposium, June 27-30th, 2007

[2] Michael Opdenacker, Free Electrons, 2003, *Introduction to rtpreempt patches*

[3] Frank Rowand, November 7, 2008, *Adventures In Real-Time Performance Tuning*

[4] Clark Williams, *An Overview of Realtime Linux*, Redhat Summit, June 18-20th, 2008.

[5] Loongson, http://www.loongson.con

[6] FuLoong, http://www.lemote.com/english/fuloong.html

[7] RT-preempt, http://rt.wiki.kernel.org/

[8] http://www.kernel.org/pub/linux/kernel/projects/rt/

[9] http://www.ibm.com/developerworks/cn/linux/l-lrt/part1/

[10] http://www.ibm.com/developerworks/cn/linux/l-lrt/part2/

[11] http://dev.lemote.com/code/rt4ls

[12] perf_counter(v8), http://lwn.net/Articles/336542/

[13] ftrace, Documentation/ftrace.txt

[14] cyclictest, http://rt.wiki.kernel.org/index.php/Cyclictest

[15] http://groups.google.com/group/archlinux-for-loongson/web/binutils-2.19.1-loongson2f-3.patch

[16] http://dev.lemote.com/code/rt4ls/raw-attachment/wiki/WikiStart/0001-cyclictest-ensure-it-work-with-glibc-2.7-in-linux-M.patch

[17] http://rt.wiki.kernel.org/index.php/CONFIG_PREEMPT_RT