June, 2010

Performance Analysis on SMP and Non-SMP for Multicore Technology

FTF-ENT-F0697

TieFei Zang
Principle Software Engineer
Introduction

► Multicore in communication processor technology
► Dual cores boot up process
► Balance the workload of dual-core SMP to improve the IP-forward performance in Linux
► MPC8572, P2020 and P102x
Agenda

► Multicore Technology

► SMP and AMP

► Network Performance on Dual Core SMP in Linux
Moore’s Law: Gate counts have doubled every two years
► CPU clock rate improvements slowing: 40% per year → 12% per year
► Pipelining increased by factor of 4 in past decade → not possible in next decade

Source: UT Dept. Computer Science

8-10 FO4 Pipeline

Relative Performance


1.00

10.00

100.00

1000.00

10000.00

40x

6.6x

10x
Drivers for Multicore

► Multicore architectures becoming mainstream solution to scale processor performance rather than brute force clock-rate increases

► Scaling limitations: Hitting the Wall
  • **Memory Wall**
    Increasing gap between processor and memory speeds, which requires larger cache sizes to mask the latency to memory
  • **Instruction Level Parallelism Wall**
    Increasing difficulty of extracting enough parallelism in the instruction stream to keep a higher performance processor core busy
  • **Power Wall**
    Trend of consuming double the power with each doubling of operating frequency

► Multicore infrastructure needed for connecting CPU cores, I/O interfaces, datapath-accelerators and memory hierarchy
### Why Multicore?

- Bottleneck for clock frequencies on single-core
- Transistor integration
- Software application multithread
- Computer architecture (More parallelism)

![CoreNet Interconnect Fabric Diagram](image)

<table>
<thead>
<tr>
<th>DDR2/3 Memory Controller</th>
<th>DDR2/3 Memory Controller</th>
</tr>
</thead>
<tbody>
<tr>
<td>L2 Cache</td>
<td>L2 Cache</td>
</tr>
<tr>
<td>Power Architecture™ Core</td>
<td>Power Architecture™ Core</td>
</tr>
<tr>
<td>D-Cache</td>
<td>D-Cache</td>
</tr>
<tr>
<td>I-Cache</td>
<td>I-Cache</td>
</tr>
</tbody>
</table>

![CoreNet Interconnect Fabric Diagram](image)

- Front-side L3 Cache
- DDR2/3 Memory Controller
- DDR2/3 Memory Controller
- XAUI/4xSGMII
- XAUI/4xSGMII
- 10GE/4xGE
- 10GE/4xGE

![CoreNet Interconnect Fabric Diagram](image)

- Packet Distribution
- Table Lookup
- Crypto Offload
- Pattern Match Offload
- Perf Mon
- Duart
- 2x PIC
- Timers
- OpenPIC

![CoreNet Interconnect Fabric Diagram](image)

- Queue Management
- Buffer Allocation
- DMA
- PCI Exp
- PCI Exp
- PCI Exp
- SRIO
- SerDes

---

Freescale, the Freescale logo, AllVesc, C-5, CodeWarrior, ColdFire, C-Ware, mobileGT, PowerQUICC, StarCore, and Symphony are trademarks of Freescale Semiconductor, Inc., Reg. U.S. Pat. & Tm. Off. BeeKit, BeeStack, CoreNet, the Energy Efficient Solutions logo, Flexes, MXC, Platform in a Package, Processor Expert, CorIQ, QUICC Engine, SMARTMOS, TurboLink and VortiQa are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. © 2010 Freescale Semiconductor, Inc.
Agenda

► Multicore Technology

► SMP and AMP

► Network Performance on Dual Core SMP in Linux
SMP Concepts

Symmetrical MultiProcessing

- An architecture that provides fast performance by making multiple CPUs available to complete individual processes simultaneously (multiprocessing).

- Any idle CPU can be assigned any task, and additional CPUs can be added to improve performance and handle increased loads.

- Uses a single operating system and shares common memory, all the IO and interrupt resources.

- Processes and threads are distributed among CPUs.
SMP Task Scheduling

► SMP support was introduced with kernel 2.0 and has experienced steady improvement ever since.

► O(1) Scheduler in 2.6 kernel improves the SMP performance by separating task queues to each CPU which improves CPU cache efficiency.

► What does the Linux kernel’s scheduler do for SMP?
  • Assigns the tasks to different CPUs
  • Places the task to the run queue for each CPU
  • Balances the task load by making sure each CPU has relatively equal number of processes running on it

► Kernel/sched.c
The Scheduling Algorithm Review

- **SCHED_FIFO**
  - A First-In, First-Out real-time process. When the scheduler assigns the CPU to the process, it leaves the process descriptor in its current position in the run-queue list. If no other higher-priority real-time process is executable, the process continues to use the CPU as long as it wishes, even if other real-time processes that have the same priority are executable.

- **SCHED_RR**
  - A Round Robin real-time process. When the scheduler assigns the CPU to the process, it puts the process descriptor at the end of the run-queue list. This policy ensures a fair assignment of CPU time to all SCHED_RR real-time processes that have the same priority.

- **SCHED_NORMAL**
  - A conventional, time-shared process
SMP Task Scheduling (cont.)

► One run-queue per CPU
  • When tasks are created in a SMP system, they are placed in a given CPU's run queue.
  • Each CPU handles its own tasks and does not need to wait until other CPU tasks finish their time slices.

► Migration Thread
  • Defined as “struct task_struct *migration_thread”
  • Runs for every CPU
  • Migrate thread from one run-queue to another
SMP Task Scheduling (cont.)

► **load_balance()**

- It attempts to take task from an overloaded CPU and give it to an underloaded one. This is achieved by pulling a task from one CPU and assigning it to another to balance the loads between CPUs.

- It is called:
  - **Explicitly** if run-queues are not balanced.
  - **Periodically** by the time tick. Every 200ms, a CPU checks to see whether the CPU loads are unbalanced. If they are, the CPU performs a cross-CPU balancing of the currently running tasks.
Processes can be made affine to a particular CPU.

CPU affinity is a *scheduler property* that *bonds* a process to a given set of CPUs on the SMP system. The Linux scheduler will honor the given CPU affinity and the process will not run on any of the other CPUs.

Setting CPU affinity for a certain task or process is done by using the `taskset` command

- Install `util-linux` in `ltib`.
- Command example:

```
    # taskset -pc 0 72  /* 0 is CPU id, 72 is process ID (PID) */
```
SMP Interrupt Handling (cont.)

► In a multiprocessor implementation the PIC is replicated for each core.

► Interrupt Destination Register (xIDRn)
  • P0 and P1 bits are used to define which processor the interrupt should be directed

► Inter Processor Interrupt (IPI)
  • The intention is for communication between different processor cores on the same device.
  • We define the IPI action if SMP is defined

► Per-CPU registers
  • Each core has a replicate copy of per-CPU registers

► Linux code reference:
  • arch/powerpc/sysdev/mpic.c
Interrupts on MPC8572DS

- CONFIG_IRQ_ALL_CPUS = “y” in Linux will allow the kernel to distribute IRQs across multiple cores. If “n”, all IRQs will be directed to the first core (core#0) only.

<table>
<thead>
<tr>
<th>CONFIG_IRQ_PER_CPU = n</th>
<th>CONFIG_IRQ_PER_CPU = y</th>
</tr>
</thead>
<tbody>
<tr>
<td>-sh-2.05b# cat /proc/interrupts</td>
<td>-sh-2.05b# cat /proc/interrupts</td>
</tr>
<tr>
<td>CPU0 CPU1</td>
<td>CPU0 CPU1</td>
</tr>
<tr>
<td>5: 6 0 i8259 Level ahci</td>
<td>5: 2 4 i8259 Level ahci</td>
</tr>
<tr>
<td>14: 19 0 i8259 Level libata</td>
<td>14: 8 11 i8259 Level libata</td>
</tr>
<tr>
<td>17: 6 0 OpenPIC Level phy_interrupt</td>
<td>17: 4 5 OpenPIC Level phy_interrupt</td>
</tr>
<tr>
<td>29: 5 0 OpenPIC Level enet_tx</td>
<td>29: 0 5 OpenPIC Level enet_tx</td>
</tr>
<tr>
<td>30: 2 0 OpenPIC Level enet_rx</td>
<td>30: 3 4 OpenPIC Level enet_rx</td>
</tr>
<tr>
<td>34: 0 0 OpenPIC Level enet_error</td>
<td>34: 0 0 OpenPIC Level enet_error</td>
</tr>
<tr>
<td>42: 109 0 OpenPIC Level serial</td>
<td>42: 85 40 OpenPIC Level serial</td>
</tr>
<tr>
<td>43: 25 0 OpenPIC Level i2c-mpc, i2c-mpc</td>
<td>43: 22 3 OpenPIC Level i2c-mpc, i2c-mpc</td>
</tr>
<tr>
<td>251: 3 78 OpenPIC Edge IPI0 (call function)</td>
<td>251: 17 64 OpenPIC Edge IPI0 (call function)</td>
</tr>
<tr>
<td>252: 1 1 OpenPIC Edge IPI1 (reschedule)</td>
<td>252: 4 2 OpenPIC Edge IPI1 (reschedule)</td>
</tr>
<tr>
<td>253: 0 0 OpenPIC Edge IPI2 (unused)</td>
<td>253: 0 0 OpenPIC Edge IPI2 (unused)</td>
</tr>
<tr>
<td>254: 0 0 OpenPIC Edge IPI3 (debugger break)</td>
<td>254: 0 0 OpenPIC Edge IPI3 (debugger break)</td>
</tr>
</tbody>
</table>
Asymmetrical MultiProcessing

- Each CPU runs its own OS which may be same or different from each other.
  - Each CPU can be assigned with specific application to run
  - If Linux, two copies of ulimage are needed, but located at different physical address spaces!
- Both CPUs must cooperate to share the resources.
  - Neither OS can own the whole system
  - I/O and interrupts are separated
  - Static configuration for resources
  - If Linux, resources allocation can be done by two device trees (dts)

Two cores -> two OSs
Multicore Boot Architecture

- ePAPR describes specifics on how secondary CPUs are booted for a system with multiple CPUs

- Default boot architecture
  - The boot program releases all CPUs from hardware reset
  - One CPU is designated to be the client program’s boot CPU
  - All other CPUs are secondary and are placed into loop where the CPUs spin waiting for a spin table field to change that directs them where to go
  - Control is transferred to the client program on the boot CPU
  - When the client program is ready for secondary cores to start, it releases them by writing the spin table field with the desired address

- The architecture allows for other custom-defined secondary CPU release mechanisms as well

- Reference: ePAPR (Power.org Standard for Embedded Power Architecture™ Platform Requirements (ePAPR). power.org, 2008.)
### MPC85xx __spin_table

<table>
<thead>
<tr>
<th>Address Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boot address low</td>
</tr>
<tr>
<td>Boot address high</td>
</tr>
<tr>
<td>R3 Upper</td>
</tr>
<tr>
<td>R3 Lower (pointer to device tree)</td>
</tr>
<tr>
<td>Reserve</td>
</tr>
<tr>
<td>PIR (Processor Identification Register)</td>
</tr>
</tbody>
</table>
Reset configuration cfg_cpu0_boot, cfg_cpu1_boot on board:

- Core0 comes out of reset at 0x0_FFFF_FFFC then branch to _start_e500() at 0x0_FFFF_F000, Core1 is set to boot holdoff mode

  _start_e500()
  
  1. Enable L1 caches
  2. Config interrupt vectors
  3. Config MMU and LAWs
  4. Config L1D RAM
  5. Config local bus
  6. Config DDR
  7. Relocate to DDR
  ...............
MP U-Boot Process (cont.)

- `setup_mp()`
  - Determine secondary CPU boot address
  - Update TLB of reset page to the boot address

- `plat_mp_up()`
  - Set boot page in BPTR
  - Set EEBPCR to release CPUs

- Secondary CPUs start at `__secondary_reset_vector()` then branch to `__secondary_start_page()`
  - Secondary CPUs start spinning in `__secondary_start_page()`
<table>
<thead>
<tr>
<th>Offset</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>b__secondary_start_page</td>
<td>Start of boot code</td>
</tr>
<tr>
<td>4k</td>
<td>__secondary_start_code_end</td>
<td>End of boot code</td>
</tr>
<tr>
<td>4k</td>
<td>__secondary_reset_vector</td>
<td>Reset vector</td>
</tr>
</tbody>
</table>

__secondary_start_page:
Enable cache
Spin for boot_addr change
Setup r3,r6,PIR
Setup 64M TLB for boot code
Jump to the boot code
Use *bootm* command

- Copy *device tree* to DDR
- Copy *uImage* to DDR
- Copy *Ramdisk* to DDR (optionally)
- Launch kernel
AMP Boot Process (Core1)

- setenv bootm_low 0x20000000  /* set memory range */
- setenv bootm_size 0x10000000
- tftp 21000000 ulimage.core1
- tftp 22000000 ramdiskfile
- tftp 20c00000 mpc8572ds_camp_core1.dtb
- interrupts off  /* disable interrupts */
- bootm start 21000000 22000000 20c00000
- bootm loados  /* load image and uncompress to load addr*/
- bootm ramdisk  /* relocate init ramdisk; set env variables*/
- bootm fdt  /* relocate fdt to bootstrap range */
- fdt boardsetup  /* Update device tree of board info */
- fdt chosen $initrd_start $initrd_end  /* Update chosen */
- bootm prep  /* Flush dcache */
- cpu 1 release $bootm_low - $fdtaddr -  /* fill __spin_table: bootaddr=$bootm_low; r3 = $fdtaddr */
AMP Boot Process (Core 0)

- setenv bootm_low 0
- setenv bootm_size 0x20000000
- tftp 1000000 uImage.core0
- tftp 2000000 ramdiskfile
- tftp c00000 mpc8572ds_camp_core0.dtb
- bootm 1000000 2000000 c00000
Agenda

- Multicore Technology
- SMP and AMP

Network Performance on Dual Core SMP in Linux
IP Forward on Linux

Network 1

PowerPC Target with Linux

Network 2
Test Environment

Test Center

IP: 192.168.1.1
MAC: 00:00:00:00:00:0C

IP: 192.168.2.1
MAC: 00:00:00:00:00:0D

Port 1

Port 2

Eth0

Eth1

Positive Flow

Negative Flow

IP: 192.168.1.00
MAC: 00:E0:0C:02:10:FD

IP: 192.168.2.100
MAC: 00:E0:0C:02:01:FD
Core1 can process other tasks
Core1 can process other tasks

1 Core 2 Opposite Flow

Test Center

Port 1

Port 2

Rx

Tx

Eth0

Eth1

cpu0

cpu1
Different Core Process Different Flow

Test Center
Port 1
Port 2

Rx
Tx

Eth0
cpu0

Eth1
cpu1

SMP
Dedicate Core

2 Core 2 Opposite Flow
Fast Path Routing

```
Ethernet (1000 Mbit)

Arrow keys navigate the menu. <Enter> selects submenus --->.
Highlighted letters are hotkeys. Pressing <Y> includes, <N> excludes,
<M> modularizes features. Press <Esc><Esc> to exit, <?> for Help, </> for Search. Legend: [*] built-in  [ ] excluded  <M> module < >

^(-)

< > VIA Velocity support
< > Broadcom Tigon3 support
< > Broadcom NetXtremeII support
< > Broadcom CNIC support
-<-- Freescale PQ MDIO
<*> Gianfar Ethernet
[*] Socket Buffer Recycling Support (v1.1.5) (EXPERIMENTAL)
[*] Fast Path routing
<*> Freescale QE Gigabit Ethernet
[ ] Transmit on Demand support

<Select> < Exit > < Help >
```
Fast Path Routing on SMP

► Add fast path entry to device’s routing table cache
  • IPv4
  • Regular header
  • Not a multicast packet
  • Going to a valid destination

► Improve ipfw performance
E500 Dual Core Family

- MPC8572
- P2020
- P102x
Freescale Product Longevity Program

► The embedded market needs long-term product support

► Freescale has a longstanding track record of providing long-term production support for our products

► Freescale offers a formal product longevity program for the market segments we serve
  
  • For the automotive and medical segments, Freescale will make a broad range of program devices available for a minimum of 15 years
  
  • For all other market segments in which Freescale participates, Freescale will make a broad range of devices available for a minimum of 10 years
  
  • Life cycles begin at the time of launch

► A list of participating Freescale products is available at: www.freescale.com/productlongevity