Chapter 25
A Framework for Porting Linux OS to a cc-NUMA Server Based on Loongson Processors

Kun Zhang, Hongyun Tian, Li Ruan, Limin Xiao, Yongnan Li and Yuhang Liu

Abstract In order to make the Linux operating system available on a cache coherence NUMA (cc-NUMA) server based on Loongson processors, a family of general-purpose MIPS64 CPUs developed by the Institute of Computing Technology in China, this paper proposes a framework for porting Linux operating system to this cc-NUMA server. Researchers present the overall port scheme after analyzing the framework of the Linux kernel and the architecture of the hardware platform, and then they discuss the transplantation in details with processor-level transplantation, memory management transplantation, interrupt and trap transplantation. The performance evaluation shows that the whole system works stable and the ported operating system could reach about 30 % of the theoretical peak value of floating-point calculation. The method could port Linux OS to the target board successfully and can be used on other platforms. The research has great significance to the development of the domestic Loongson processor and the cc-NUMA platform based on Loongson processors.

Keywords High performance computer · cc-NUMA · Loongson · Linux kernel

25.1 Introduction

Loongson is a family of general-purpose MIPS64 CPUs developed by the Institute of Computing Technology (ICT) in China. The Loongson-3B processor is an 8-coreprocessor with 1 GHz frequency [1]. Non-Uniform Memory Architecture

K. Zhang (✉) · H. Tian · L. Ruan · L. Xiao · Y. Li · Y. Liu
State Key Laboratory of Software Development Environment, Beihang University, Beijing, China
e-mail: zhangkun2441@126.com

K. Zhang · L. Ruan (✉)
School of Computer Science and Engineering, Beihang University, Beijing, China
e-mail: ruanli@buaa.edu.cn

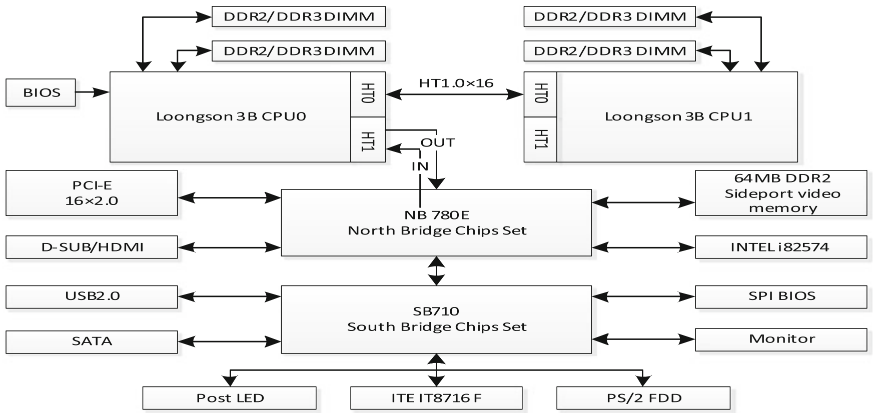


Fig. 25.1 The logic building-block of the cc-NUMA system based on Loongson CPU

(NUMA) has been more and more popular in the field of high performance computer as it has better scalability than Uniform Memory Architecture (UMA) [2]. The NUMA system has the feature that for any given region of physical memory, some processors are closer to it than the other processors.

cc-NUMA is a kind of NUMA system. cc-NUMA server based on Loongson processors is devised by the Institute of Computer Architecture of BeiHang University. Figure 25.1 shows the logic building-block view of the target platform. There are two nodes on the board with a processor per node. As depicted in Fig. 25.1, the CPU0 can access to its own memory faster than access to the memory on the other node. Frequently remote memory access would degrade the system performance seriously. So we need to avoid remote memory access in the porting scheme. CPU0 on the node0 is the boot CPU of the system and the other processors need to be initialized by it. Therefore, we need to solve this problem during the system initialization. Besides, the North Bridge chipset connects the peripheral component and South Bridge chipset with the boot CPU. Then we need to program the under-layer functions of the PCI device handler to make them work properly.

The rest of this paper is organized as follows. Section 25.2 analyzes the Linux kernel and puts forward an overall porting scheme. Section 25.3 discusses the transplantation in detail and we evaluate the modified kernel in Sect. 25.4. Some related work is introduced in Sect. 25.5 befor the conclusion in Sect. 25.6.

25.2 Linux Kernel Analysis and Overall Porting Scheme

Linux is one of the most widely ported operating system kernels. In this section, we first analyze the kernel and then put forward an overall porting scheme. As shown in Fig. 25.2, the Linux kernel lies between the user space and the under-layer hardware.

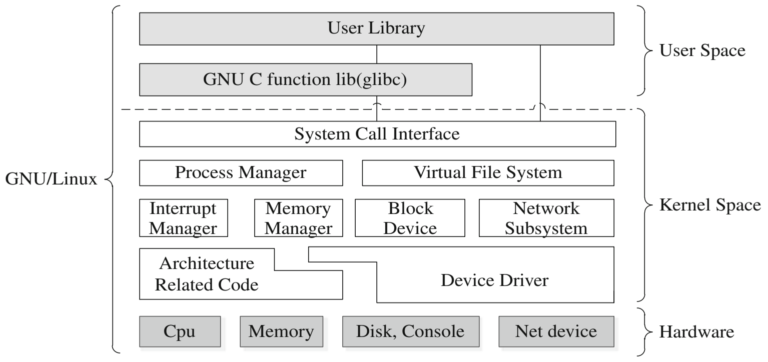


Fig. 25.2 A simplified view of Linux operating system

Kernel is the key part of the operating system with different modules responsible for monitoring and management of the entire computer system [3]. As presented in Fig. 25.2, the Linux operating system can be divided into three layers. Kernel layer lies under the system call interface and can be further divided into architecture independent module and architecture relevant module. Architecture independent module is common to all Linux support architectures. Architecture relevant module is called as Board Support Package (BSP), which is relevant to the hardware architecture and the processors [4].

The most important modules in the kernel are Process management module (PMM), memory management module (MMM), virtual file system (VFS) and network interface module (NIM) [5]. These modules cooperate with each other to make the kernel runs properly. PMM chooses the next-running process during the clock interrupt and it is also responsible for the load balance of the system [6]. As introduced in Sect. 25.1, it must be careful to choose the next-running process because of the problem of remote memory access. To make sure the scheduler works well on the cc-NUMA architecture, some information of the under-layer needs to be provided to the scheduler.

The MMM is responsible for deciding the region of memory that each process can use and determining what to do when not enough memory is available [7]. The memory management module can be logically divided into two parts, architecture-independent module and architecture-relevant module. The architecture-relevant part contains codes of memory initialization and some handler functions of memory management. The virtual file system provides a unified interface for all devices, and hides the specifics of the under-layer hardware. To perform useful functions, processes need to access to the peripherals connected to the North Bridge and South Bridge, which are controlled by the kernel through device drivers. The device driver module is device specific [8]. We don't need to rewrite the device driver code, as the AMD Inc. has already provided the driver for the peripheral devices.

We get the overall porting scheme according to our analysis to the kernel code as presented in Fig. 25.3. Details will be discussed in the next section.

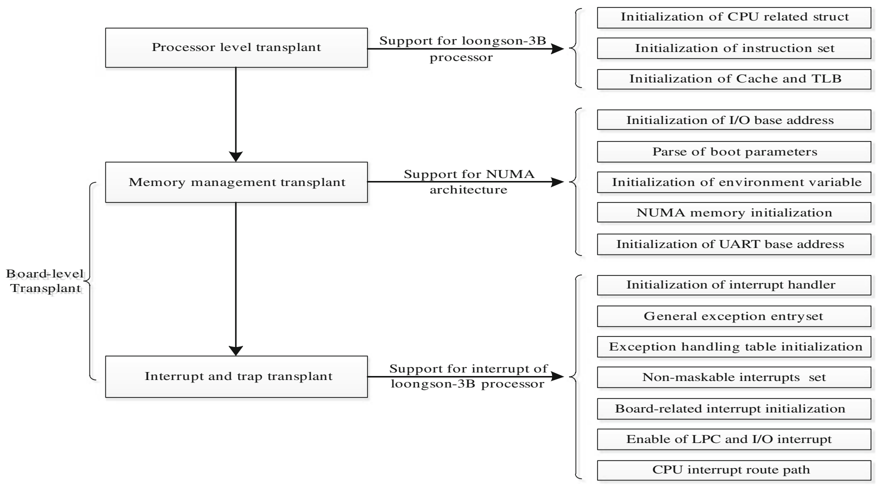

Processor level transplant
Support for loongson-3B processor
Initialization of CPU related struct
Initialization of instruction set
Initialization of Cache and TLB
Memory management transplant
Support for NUMA architecture
Initialization of I/O base address
Parse of boot parameters
Initialization of environment variable
NUMA memory initialization
Initialization of UART base address
Board-level Transplant
Interrupt and trap transplant
Support for interrupt of loongson-3B processor
Initialization of interrupt handler
General exception entryset
Exception handling table initialization
Non-maskable interrupts set
Board-related interrupt initialization
Enable of LPC and I/O interrupt
CPU interrupt route path

Fig. 25.3 The overall porting scheme

25.3 Kernel Porting

In order to port Linux to the Loongson cc-NUMA platform, we take the stable version of Linux 2.6.36 as our orignal edition. As schematically described in Fig. 25.3, we will concretely discuss the kernel porting in detail in this section.

25.3.1 Processor-Level Transplantation

Processor-level transplantation includes the initialization of CPU related structure, cache volumes and TLB volumes. Specifically, it contains the following steps:

1. Initialize the CPU related structure. Such as the processor_id of Loongson-3B, the machine type, the instruction cache, the data cache and the second cache.
2. Get the architecture type of the CPU according to the processor_id. Loongson-3B processor is based on MIPS 4KC.
3. Get the cpu_id, fpu_id and the type of the CPU based on the front two steps.
4. Check the virtual memory bits with EntryHi register to prepare for the memory management transplantation, Table 25.1.

25.3.2 Memory Management Transplantation

Memory is the key component of the system, process can't work without memory. The memory management transplantation mainly contains 5 parts.

Table 25.1 Example code of processor-level transplantation

```
#define PRID_IMP_LOONGSON3A          0x6305
#define PRID_IMP_LOONGSON3B          0x6306 //definition of the processor_id
#define enable_fpu()              \
do {                              \
        if (cpu_has_fpu)          \
                __enable_fpu();   \
} while (0);
```

1. Initialize the base address of I/O space. The Loongson-3B processor unified the whole physical space. As a result, the I/O memory address is a part of them.
2. Parse the boot command. The default command is "console = tty, 115200, root = /dev/sda1". We parse the boot command to determine the frequency of the console and the path of the root file system. In addition, some other environment variables can be passed by the boot command.
3. Parse the environment variables. These parameters are transferred by the bootloader according to hardware registers, including the frequency of bus clock, cpu clock and the size of the memory, high memory size et al.
4. Initialize memory subsystem supporting for NUMA architecture. As we introduced in Sect. 25.1, it needs to avoid remote memory access as much as possible to be efficiently. We distinguish the memory between NUMA nodes with the memory size of each NUMA node. Each NUMA node has its own memory and the kernel can be conscious about the system memory.
5. Initialize the UART base address. The UART port is very important for the system debug as it can print out the debug information for the developer. The UART base address varies between the different processors, which should be set according to the datasheet.

25.3.3 *Interrupt and Trap Transplantation*

Interrupt subsystem is the essential composition of multithread system. This part can be divided into three parts as follows.

1. Initialize the system interrupts. First, the kernel gets the active cpu_id from the CP0_coprocessor, then it sets the interrupt registers to mask all the interrupt flags and clear the interrupts hanged up. Then the kernel enables the LPC and I/O interrupt controller and other interrupt related registers.
2. Map the interrupt handler to the irq number. We need to rewrite the hardware related interrupt handlers such as shown in Table 25.2.

Table 25.2 Example code of interrupt and trap transplantation

```
void __init mach_init_irq(void)
{
        lpc_interrupt_route(); //Route the LPC interrupt to Core0 INT0
        ht_interrupt_route(); // Route the HT interrupt to Core0 INT1
        mips_cpu_irq_init();// Route the cpu related interrupt to Core0
        init_i8259_irqs(); // Route the serial interrupt on the south bridge to Core0
}
```

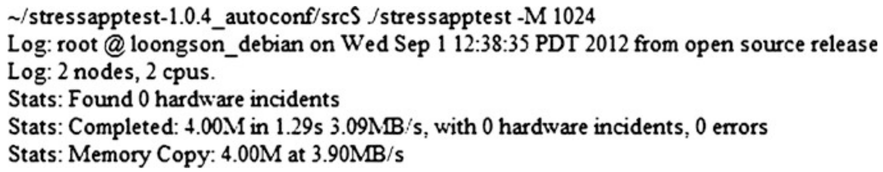
3. Set the trap base address and initialize the exception handling table. Some under layer related handlers need to be rewrite.

25.4 Evaluation

In this section, we evaluate our ported system with *stressapptest 1.0.4* to test the memory subsystem. Besides, we use *linpack* to test the peak value of floating-point calculation to evalute the system performance. The *linpack* test case is “mpiexec –np 8./xhpl” with HUGE_TLB configured in kernel. The test score of *stressapptest* showed in Fig. 25.4 demonstrates that the memory subsystem works stable. While the test score of *linpack* in Fig. 25.5 shows that our ported system can reach about 30 % of the theoretical peak value of floating-point calculation.

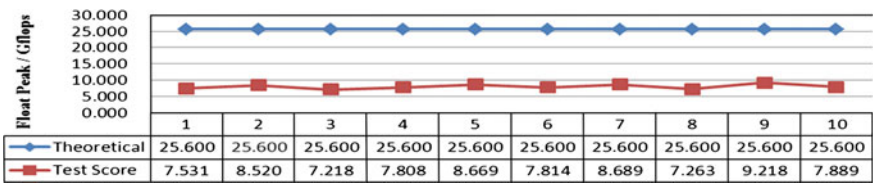
25.5 Related Work

It has been a long time of work to port Linux OS to other platforms such as ARM, MIPS. Hu Jie [9] has transplanted Linux OS to the ARM platform. As of Loongson platform, Cheng xiao-yu [10, 11] has transplanted the μC/OS to the Loongson based platform, and Qian Zheng-jiang [12] discussed about the development of Linux distribution on Loongson platform. Besides, the ICT has made a lot of work



```
~/stressapptest-1.0.4_autoconf/src$ ./stressapptest -M 1024
Log: root @ loongson_debian on Wed Sep 1 12:38:35 PDT 2012 from open source release
Log: 2 nodes, 2 cpus.
Stats: Found 0 hardware incidents
Stats: Completed: 4.00M in 1.29s 3.09MB/s, with 0 hardware incidents, 0 errors
Stats: Memory Copy: 4.00M at 3.90MB/s
```

Fig. 25.4 Evaluation with *stressapptest 1.0.4*



	1	2	3	4	5	6	7	8	9	10
◆ Theoretical	25.600	25.600	25.600	25.600	25.600	25.600	25.600	25.600	25.600	25.600
■ Test Score	7.531	8.520	7.218	7.808	8.669	7.814	8.689	7.263	9.218	7.889

Fig. 25.5 Evaluation with *linpack*

on the Loongson-3A processor. Our research can make up for the current situation about lacking of research to the Loongson-3B platform.

25.6 Conclusion

In this paper, Researchers analyzed the Linux kernel architecture and the hardware platform based on Loongson processor. Besides, they discussed porting Linux to Loongson NUMA platform concretely. Processor-level transplantation, memory management transplantation and interrupt related transplantation were introduced in detail. The evaluate score shows that the ported system runs smoothly on the Loongson platform. The research provides an example of porting Linux to Loongson platforms and can be easily used on other platforms. The research has great significance to the development of domestic Loongson processor and the cc-NUMA platform based on Loongson processors.

Acknowledgments Our research is sponsored by the Hi-tech Research and Development Program of China (863 Program) under Grant NO.2011AA01A205, the National Natural Science Foundation of China under Grant NO.61232009, the Doctoral Fund of Ministry of Education of China under Grant NO.20101102110018, the Beijing Natural Science Foundation under Grant NO.4122042, the fund of the State Key Laboratory of Software Development Environment under Grant NO.SKLSDE-2012ZX-07.

References

1. Weiwu, H., Ru, W., Baoxia, F., et al.: Physical implementation of the eight core Loongson-3B microprocessor. J. Comput. Sci. Technol. **26**(3), 520–527 (2011)
2. Bolosky, W., Fitzgerald, R.: The development of computer architecture in HPC. In: Proceedings of the 19th International Conference on Parallel Architectures, pp. 557–561
3. Sanjeev, K.: Reliability estimation and analysis of linux kernel. Int. J. Comput. Sci. Technol. **12**(2), 529–533 (2011)
4. Bowman, T.: Linux as a case study: its extracted software architecture [EB/OL]. http://pIg.uwater-Ioo. ca/∼ itbowman/papers
5. Jones, T.: Inside the Linux 2.6 Completely Fair Scheduler. [EB/OL]. http://www.ibm.com/deve-loperworks/linux/library/l-completely-fair-scheduler
6. Galvin, S.: Operating System Concepts, 4th edn. pp. 458–460 (1994)

7. Eranian, S.: Virtual Memory in the MIPS Linux Kernel, pp. 320–331. Prentice Hall PTR, Upper Saddle River (2005)
8. Choi, J., Baek, S., Shin, S.Y.: Design and implementation of a kernel resource protector for robustness of Linux module programming. In: Proceedings of the 2006 ACM Symposium on Applied Computing, pp. 1477–1481 (2006)
9. Jie, H., Genbao, Z.: Research transplanting method of embedded Linux kernel based on ARM platform. International Conference of Information Science and Management Engineering. Xi'an, China, pp. 424–432 (2010)
10. Qian, Z., Fujian, W., Boyan, L.: Transplant Method and Research of μC/OS_II on Loongson paltform. 2011 Fourth International Conference on Intelligent Computation Technology and Automation, pp. 291–301. Xi'an, China (2011)
11. Xiao yu, C., Duyan, B., Ye, C et al.: Transplantation of μC/OS on loongson processor and its performance analysis. Comput. Eng. **05**(02), 372–379 (2009)
12. Zhengjiang, Q., Jin yi, C.: Development of Linux release based on Loongson mipsel architecture. J. Chang Shu Inst. Technol. **22**(10), 87–91 (2008)