

多路处理器计算机 教学实验系统 用户使用手册

v 1.27

目 录

第一部分	实验平台系统简介	1
一	产品背景	1
二	产品特点	2
三	实验平台	3
第二部分	实验箱硬件系统介绍	6
一	处理板	6
1	主板硬件及配置	6
二	控制板	8
1	控制板介绍	8
2	控制板操作说明	8
第三部分	实验平台软件系统介绍	10
一	实验平台运行流程	10
二	软件系统	10
1	配置文件系统服务器	11
1.1	基于 Ubuntu 系统	11
1.2	基于 Windows 系统	14
1.2.1	VMware 虚拟机	14
2	启动处理单元	19
2.1	单处理单元系统	19
2.2	多处理单元系统	21
2.3	分布式系统	23
3	登录处理单元	24
4	处理单元连接外网	24
第四部分	实验箱并行结构配置	25
一	单处理器	25
1	对称多处理机 (SMP)	25
2	4 核 SMP/CMP 系统	25
二	16 核 SMP 系统	27
1	系统介绍	27
2	系统硬件配置	27
三	CC-NUMA 系统	28
1	CC-NUMA 系统	28
2	两节点 CC-NUMA 系统	29
3	四节点 CC-NUMA 系统	30
四	分布式存储多处理机	30
第五部分	并行实验指导实例	32
一	LU 分解实验指导案例	32
1	单处理器串行算法	33
2	单处理器 OpenMP 并行算法	35

3	16 核系统 OpenMP 并行算法	38
4	分布式存储多处理机 MPI 并行算法.....	41
5	实验结果及分析.....	49
附 1	处理板硬件说明.....	52
附 2	拨码开关设置说明.....	54
附 3	控制板接口说明.....	56
附 4	并行算法实践课程实验表.....	57

第一部分 实验平台系统简介

根据高等院校计算机、软件工程等本科及研究生专业开设的《计算机体系结构》(含并行)、《并程序序设计》和《计算机操作系统》(含并行)等课程的实验教学需要,采用多路国产龙芯多核芯片,自主研发,具有自主知识产权的,基于Linux 操作系统平台的集成实验开发平台环境。该实验系统可以满足不同层次院校开设上述课程实验的验证型、综合型、创新型实验教学要求。

一 产品背景

《计算机体系结构》(含并行)、《并程序序设计》和《计算机操作系统》(含并行)是计算机、软件工程等专业的重要课程,但上述课程的实验教学环境非常贫乏。目前,仅仅是引用国外大学及研究机构的虚拟机及其相应模拟或仿真器,硬件实验平台几乎可说是空白。这种状况造成了学生在理解、巩固和加深理论知识上受到制约,在复杂多处理器架构面前缺乏相应实践能力,在理论研究与核心技术上缺乏科研创新思维。

随着计算机软硬件技术的不断发展,处理器目前已经从单核(Single-core)进入多核(Multi-core)应用时代,基于多处理器的并行计算机也应用越来越广泛,云计算、大数据等新的应用,不单单意味着处理器核数量的增多,更是对单片处理器本身体系架构、多片处理器的计算机整体架构,对操作系统、编译器、应用软件提出了更多架构上的挑战。因此,高等院校计算机相关专业的教学必须与时俱进,不仅要在教材上,而且要在实验环境上跟上这一技术发展。否则,教学将脱离现实与实践,阻碍高科技创新人才的培养。

目前各高校采用的传统实验方案主要有两种:

一)采用模拟的方法,其设备一般采用普通PC机+软件模拟器,该方式虽成本低但实验效果不理想,并对学生操作缺乏硬件直观性,无法体现软件设计在硬件平台上的真实效果;

二)采用远程登录机房服务器,该方式成本高,平台可控性与灵活性低,导

致实验效率效果差，而且占用远程机的许多宝贵时间，并不适用于教学场景。

由于没有合适的实验教学设备，目前上述课程的实验课程在实时性、实验内容、教师现场交互指导、自由与独立自主等方面受到许多限制。

基于上述现状，中国科学院计算技术研究所联合国家高性能计算中心深圳分中心等相关计算机科研与教育实力单位，以提高我国计算机体系结构、并行计算等领域的教学与应用水平为目标，立足国产“龙芯”CPU 平台，开发多路处理器计算机教学实验系统，弥补上述课程缺乏实验教学仪器的不足，进行上述课程的配套实验设备开发及市场推广。

二 产品特点

- 国内首创，技术结构先进

该系统架构国内首创，高度集成小型化，通过软件可以配置系统架构，多个实验台可以连接构成机群，做具有更多的处理器的程序设计与运算。系统架构设计先进，既独立完整，又灵活可配置。同时，该平台适应 Linux 操作系统，可以独立作为一台工作站，相比用 PC 机，可以提供更多的处理器核，更多的互连方式。尤其在并行计算方面，虽然国内外从事多核多处理器并行机研制的公司、高校较多，但专门研制给学生实验的多核多处理器硬件实验平台，尚属首次。

- 自主开放、可调试性强

实验平台采用基于我国自主研发的龙芯多核处理器，基于标准 MIPS 架构与 Linux 通用操作系统，全系统国内自主研发，具有高性能、自主可控、系统开放等特点。该系统从底层处理器芯片到主板，以及系统软件都由我国自主研发或移植，并根据教学课程需要进行精简与定制，同时可提供 CPU 级别的 EJTAG 调试接口，具备比 Intel X86 更好的底层开放性和可调试性。

- 实验课程丰富、适用于多种课程教学

该系统可以满足《计算机体系结构》（含并行）、《并行程序设计》、和《计算机操作系统》（含并行）等多种课程的实验教学。通过配合教师日常授课，设计配套较为直观的实验案例，通过安排学生对教师所教学知识点进行及时上机操

作，起到辅助教师教学、降低教师授课负担并有效提高教学质量。同时，由于该系统具有丰富的硬件资源，还可以开展计算机相关的编译系统、应用程序、网络通讯等课程实验。使学生达到巩固提高、融会贯通、能力训练、知识创新的教学效果。

- **操作方便、成本低廉**

与模拟器相比，本实验设备使得学生可以直接针对具体硬件进行操作，实操性与直观性更强。与远程登录大的商用机器相比，可以在低成本的情况下让学生独占机器资源进行实验，互相不干扰，并且不需要排队，学生可充分发挥的聪明智慧和创造思维。

- **实验环境简单、部署方便**

采用独立的试验箱，可与现有机房功能相结合、部署安装简便，无须特殊的用房、空调和水电，对环保和安全也无特殊要求。本实验室既是实验平台，又是多处理器互连与并行程序设计开发、研究验证、培养和训练人才多功能平台。实验突出设计型、开放性。本实验平台扩展性强，多个试验箱即可独立使用，也可连接起来结合后台管理与存储服务器，构成超算/云计算/大数据实验环境，形成全套国产化科研与教学实验平台。

- **提供业界交流与服务平台**

该实验室平台由国家高性能计算中心深圳分中心（深圳大学计算机与软件学院）陈国良院士团队设计和研制，并由龙芯中科技术有限公司负责产品开发与推广，在提供教学实验平台的同时还提供培训、科研与教学合作的机会，提供课程的教学老师和学生有机会与国内最前沿的计算机芯片和研发人员直接交流，从而能够最大程度上对计算机系统课程进行深入理解，并达到高校教学与企业需求的有效对接。

三 实验平台

多路处理器计算机教学实验系统是由龙芯 3A 处理器构成的、内可配置外可扩展结构的实验硬件平台。实验系统特点如下：

- 多种并行层次：多发射、多核、多路、多机
- 多种互连方式：片上网络、HyperTransport、以太网
- 多种存储结构：CMP/SMP、CC-NUMA
- 多种编程模式：Pthread、MPI、OpenMP

该实验平台由多路处理器教学实验箱和一台文件系统服务器组成。多路处理器教学实验箱上有对称的 4 个处理单元，每个处理单元包含一颗龙芯 3A 四核处理器。4 个处理单元既可通过网络互连为多处理机集群架构，也可通过 HT 总线互连为 CC-NUMA 架构。并且多个实验平台可通过网络和 HT 实现更多的处理器互连。文件系统服务器用来统一存放 NFS 系统，为处理单元提供内核和文件系统。在教学中，教师办公用机即可配置为文件系统服务器。实验平台的基本结构和软硬件结构图如下所示。

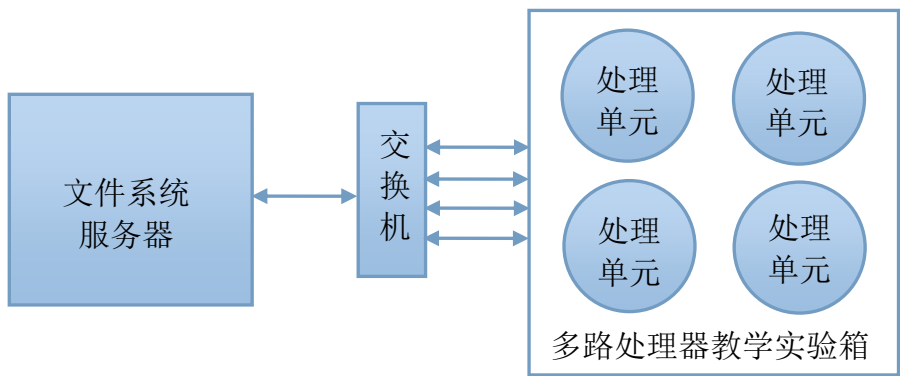


图 1-1 实验平台基本结构图



图 1-2 多路处理器教学实验系统软硬件结构框图

多路处理器教学实验箱由处理板和控制板两部分组成，整体实物如图 1-3 所示。



图 1-3 多路处理器教学实验箱实物图

第二部分 实验箱硬件系统介绍

多路处理器教学实验箱由处理板和控制板组成，如图 2-1 所示。处理板是实验平台的主要部分，可运行多处理器程序，实现并行程序的设计与开发。控制板主要负责把处理板上的网络、串口等资源扩展成标准接口，同时负责给处理板供电和进行相应的控制。

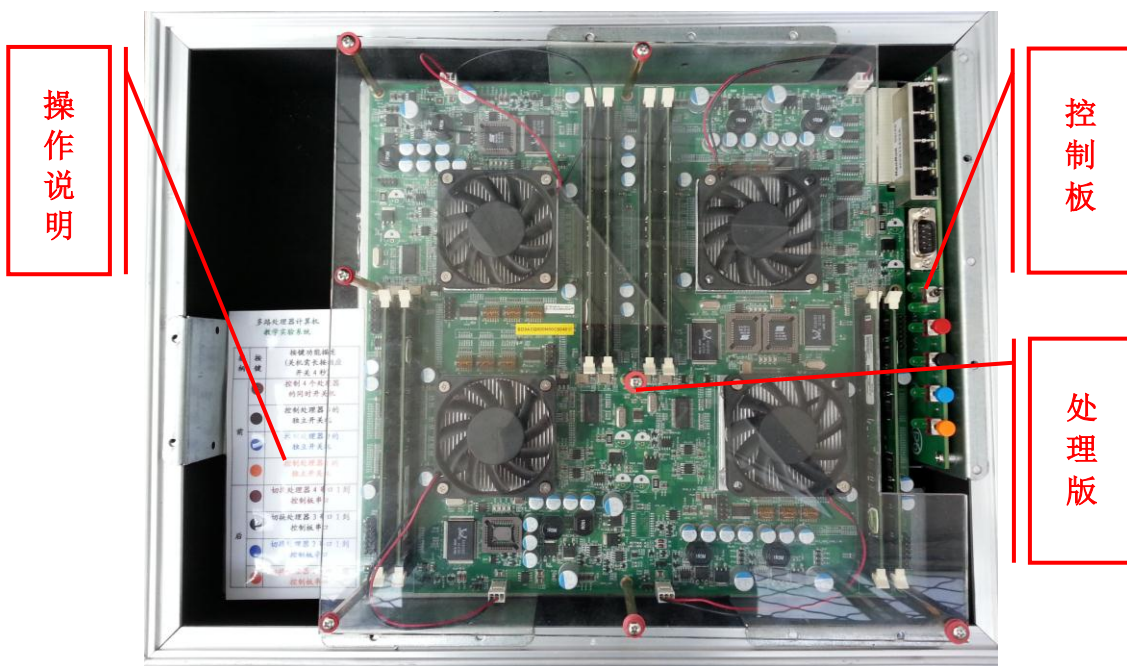


图 2-1 处理主板

一 处理板

1 主板硬件及配置

处理板承载 4 个处理单元，编号如图 2-2 所示，每个处理单元包括一颗龙芯 3 号四核处理器、DDR2 内存（内存数量和单条容量可根据需要进行配置）、RTL8110 千兆以太网卡芯片、BIOS Flash、串口收发芯片以及电源变换电路等。四个龙芯 3A 处理器在处理板上通过 HT 总线实现互连。另外，通过以太网四个处理单元还可以在板外进行互连。处理器板的结构示意图如图 2-2 所示。

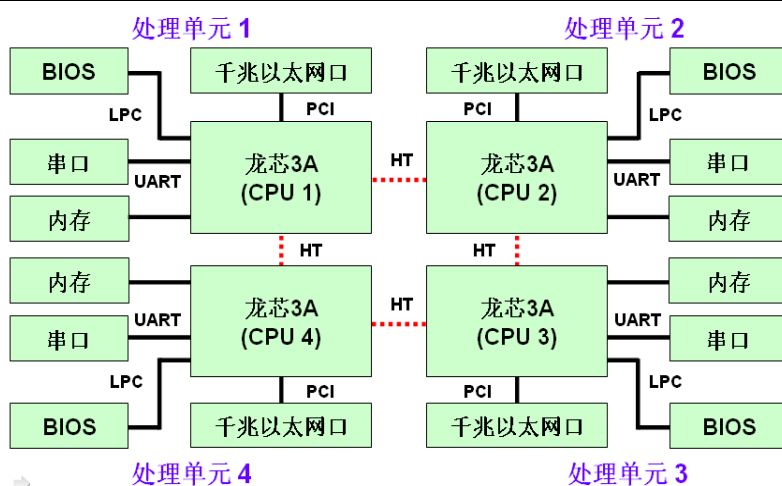


图 2-2 四路龙芯 3A 教学实验箱处理板框图

实物图为图 2-3，其中标注出一个处理单元所包括的主要器件。本主板 CPU 频率为 800MHz，内存频率 200MHz，HT 拨码开关采用默认配置，各处理单元配置参数一致，具体设置方式和功能描述参见附录 2。

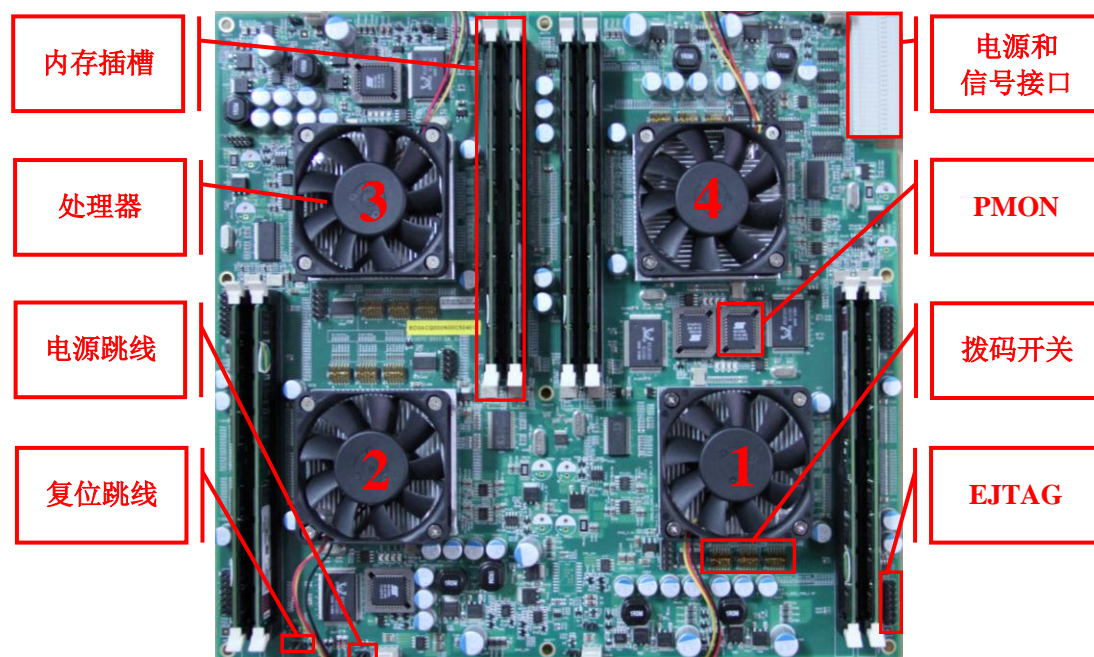


图 2-3 处理主板

处理板的设计可满足多种并行层次应用上的需求。在 HT 总线不使用时，实现单处理器结构、SMP 结构和 SMP 集群结构等；在处理器支持 Cache 一致性的 HT 通信并使用 HT 总线的情况下，可以实现 4 路 4 核的 CC-NUMA 结构和 CC-NUMA 集群结构。因此，利用该处理板可以构建实验平台，进行计算机体系

结构教学在多种并行层次上的教学实验。

二 控制板

1 控制板介绍

控制板为用户提供了操作主板的接口，控制板主要包括与每个处理器对应的千兆 RJ45 网口、可 4 选 1 的 DB9 串口、一个切换按键功能的摇柄开关、4 个功能按键和一个控制 4 个处理器开关的 AT89S52 单片机，控制板结构如图 2-4 所示：

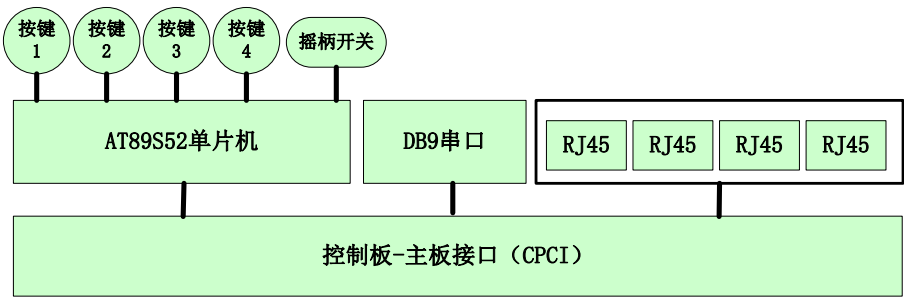


图 2-4 控制板结构

实物图为下图 2-5 所示：

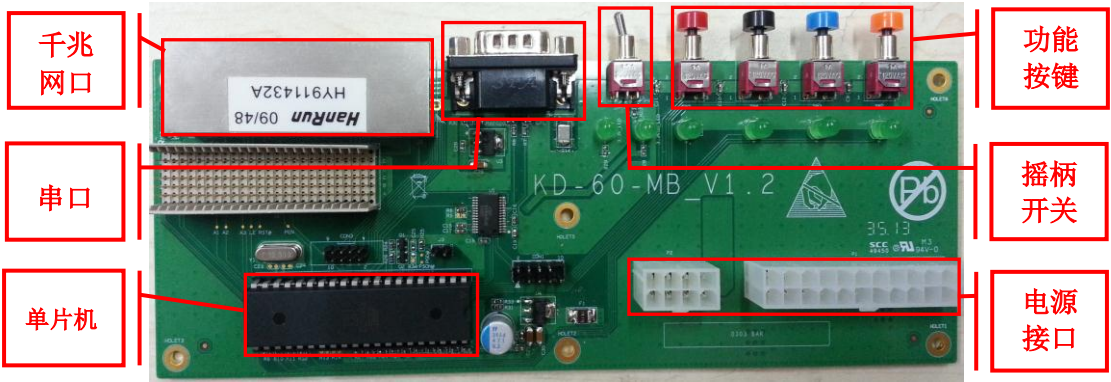


图 2-5 控制板实物图

2 控制板操作说明

控制板提供了四个按键开关和一个摇柄开关来操作不同处理器，按键说明见表 2-1。每个按键开关下配置一个 LED 来指示相应的状态，LED 灯指示说明见

表 2-2。

表 2-1 按键功能说明

摇柄开关	按键	指示灯	按键功能描述
前	红	LED1	控制计算板 4 个处理器的同时开关机
	黑	LED2	控制处理器 3 的独立开关机
	蓝	LED3	控制处理器 2 的独立开关机
	黄	LED4	控制处理器 1 的独立开关机
后	红	LED1	切换处理器 4 串口 1 到控制板串口
	黑	LED2	切换处理器 3 串口 1 到控制板串口
	蓝	LED3	切换处理器 2 串口 1 到控制板串口
	黄	LED4	切换处理器 1 串口 1 到控制板串口

注：关机动作需长按相应开关 4 秒。

表 2-2 控制板 LED 指示说明

摇柄开关	指示灯	指示灯状态描述	
前	LED1、LED2、LED3、LED4	亮	所对应的处理器处于开机状态
		灭	所对应的处理器处于关机状态
后	LED1、LED2、LED3、LED4	亮	控制板串口连通于所对应的处理器
		灭	/

第三部分 实验平台软件系统介绍

一 实验平台运行流程

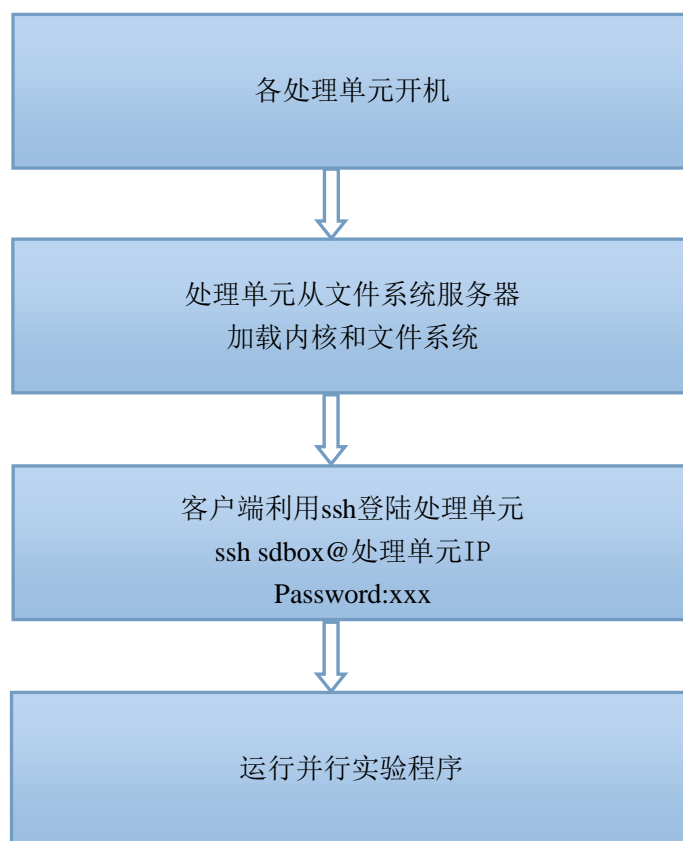


图 3-1 实验平台运行流程

二 软件系统

文件系统服务器和多路处理器综合实验箱的连接方案如下：

文件系统服务器和多路处理器综合实验箱都连接至局域网。文件系统服务器需要绑定固定的 IP 地址，开启多路处理器综合实验箱，系统会根据实验箱号和计算节点号设定本机 IP，此 IP 地址和文件系统服务器的在同一网段。具体方案如图 3-2 所示。

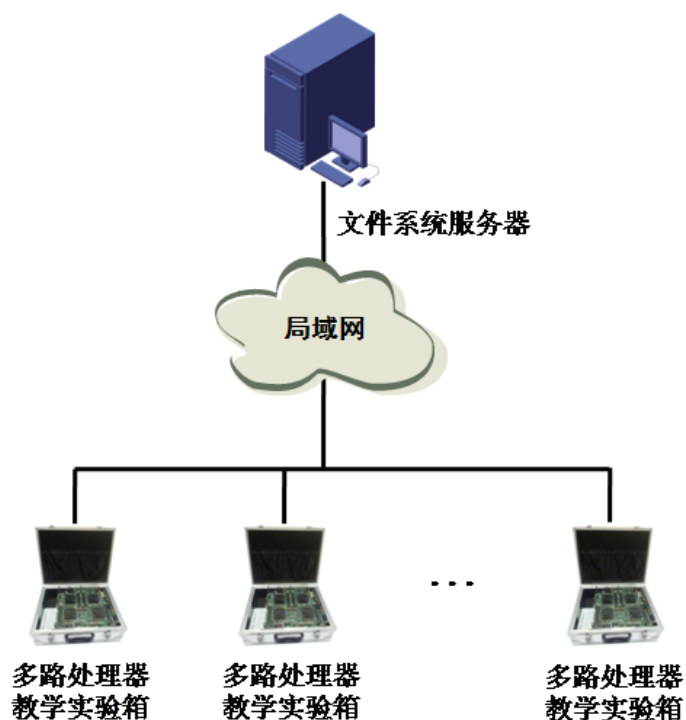


图 3-2 搭建实验室方案

下面详细介绍文件系统服务器的配置和多路处理器综合实验箱的启动流程。

1 配置文件系统服务器

配置文件系统服务器有两种方案，第一种是将一台装有 Ubuntu 操作系统的计算机配置为文件系统服务器，安装相关服务，具体操作请查考 1.1 节内容。第二种是在 windows 操作系统下，打开已经配置好的虚拟机，直接作为文件系统服务器使用，具体操作参考 1.2 节。

1.1 基于 Ubuntu 系统

(1) 安装 Ubuntu 系统

建议安装 ubuntu 12.04 LTS 系统，默认安装，将系统全部安装到一个分区，该分区应大于 60G。

(2) IP 地址配置

本文将文件系统服务器 IP 配置为 192.168.100.250

(3) 配置 NFS

文件系统服务器需支持网络文件系统（NFS），配置成 NFS 服务器。配置步骤如下(在 root 权限下):

- 安装三个软件包，命令如下：

```
apt-get install nfs-kernel-server
```

```
apt-get install portmap nfs-common
```
- 系统配置
 1. 创建 NFS 共享目录
 - 建立 NFS 共享目录 /home/nfs-debian6/rootfs/

```
mkdir /home/nfs-debian6/rootfs -p
```
 - 修改 NFS 共享目录的属性

```
chmod 777 /home/nfs-debian6/rootfs
```
 - 把提供的根文件系统解压到/home/nfs-debian6/rootfs/目录下

```
tar -xmf rootfs.tar.gz -C /home/nfs-debian6/rootfs/
```
 2. 配置/etc/exports 文件，添加语句

```
/home/nfs-debian6/rootfs    *(rw,no_root_squash,async)
```
 3. 关闭系统防火墙

```
iptables -F
```
- 重启 NFS 服务

```
/etc/init.d/nfs-kernel-server restart
```

```
/etc/init.d/portmap restart
```
- 测试本机 NFS 服务
 1. 输入指令 `showmount -e`，显示本机的 NFS 服务目录
 2. 挂载该 NFS 服务目录至/mnt，可在/mnt 目录下看到 NFS 服务目录下的内容，说明 NFS 服务搭建成功。

```
mount 192.168.100.250:/home/nfs-debian6/rootfs /mnt
```
- NFS 的启动和停止指令
启动服务：

```
/etc/init.d/nfs-kernel-server start
```


停止服务： `/etc/init.d/nfs-kernel-server stop`

(4) 配置 TFTP

处理单元的 linux 内核需要通过网络 TFTP 服务从文件系统服务器上下载，故需在文件系统服务器上配置 TFTP 服务，具体配置步骤如下：

- 安装 tftp

`sudo apt-get install tftp-hpa tftpd-hpa xinetd`

- 创建 tftp 服务目录

`mkdir /srv/tftp -p`

- 修改权限

`sudo chmod -R 777 /srv/tftp`

- 修改 tftp 配置文件/etc/xinetd.d/tftp，没有则创建该文件

```
service tftp
{
    disable          = no
    socket_type      = dgram
    protocol         = udp
    wait             = yes
    user             = root
    server            = /usr/sbin/in.tftpd
    server_args      = -s /srv/tftp -c
    source           = 11
    cps              = 100 2
    flags =IPv4
}
```

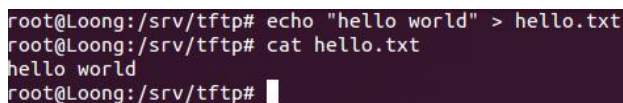
- 重启 xinetd 服务

`/etc/init.d/xinetd restart`

- 测试 tftp

1. 在本机 tftp 服务目录下建立测试文件 hello.txt

`echo "hello world" > hello.txt`



```
root@Loong:/srv/tftp# echo "hello world" > hello.txt
root@Loong:/srv/tftp# cat hello.txt
hello world
root@Loong:/srv/tftp#
```

图 3-3 创建测试文件

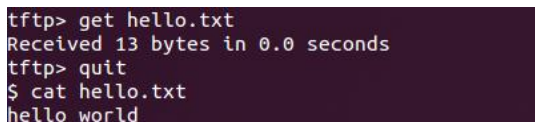
2. 在可以连接到该主机的计算机上使用 tftp 协议连接

```
tftp 192.168.100.250
```

```
tftp> get hello.txt
```

```
tftp> quit
```

```
cat hello.txt
```



```
tftp> get hello.txt  
Received 13 bytes in 0.0 seconds  
tftp> quit  
$ cat hello.txt  
hello world
```

图 3-4 tftp 测试

- 测试成功后将处理单元使用的 linux 内核镜像压缩包压缩至/srv/tftp 即可。

1.2 基于 Windows 系统

1.2.1 VMware 虚拟机

(1) 安装 Windows 7 系统

安装 Windows 7 系统，默认安装即可。

(2) 安装 VMware

拷贝多路处理器实验箱虚拟文件系统服务器 DVD 光盘上的 VMware-workstation-full-9.0.1-894247.exe，点击运行，选择经典安装。

(3) 拷贝解压 VMware 虚拟机文件

拷贝 DVD 盘中的 mj-Ubuntu-IOT-32.tar 至本地，点击右键，解压到当前文件夹。解压完成后，出现名为 Ubuntu-IOT-32 的文件夹。

(4) 打开 Ubuntu-IOT-32 虚拟机

1. 运行 VMware，点击首页上的 Open a Virtual Machine，如图 3-5，弹出打开对话框，进入到第 3 步解压的 Ubuntu-IOT-32 的文件夹，选中 Ubuntu-IOT-32.vmx，打开。

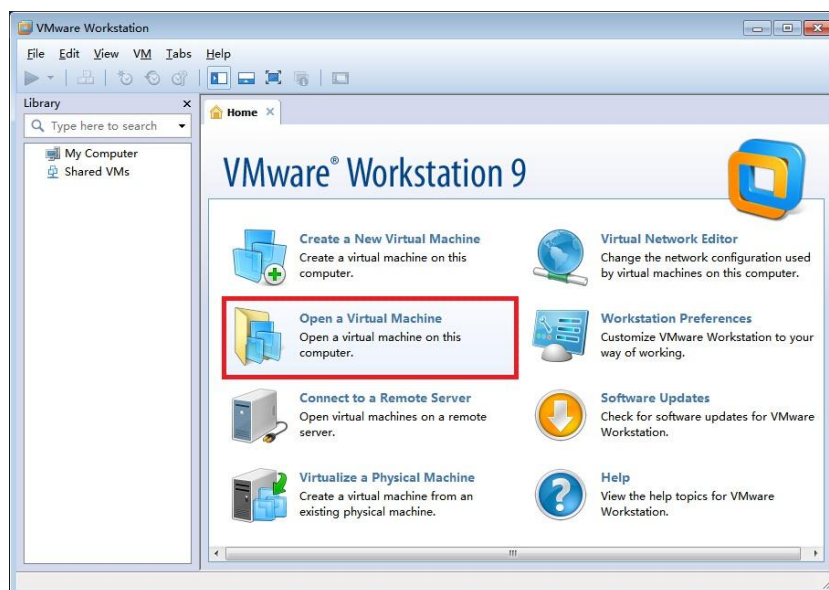


图 3-5 VMware 虚拟机

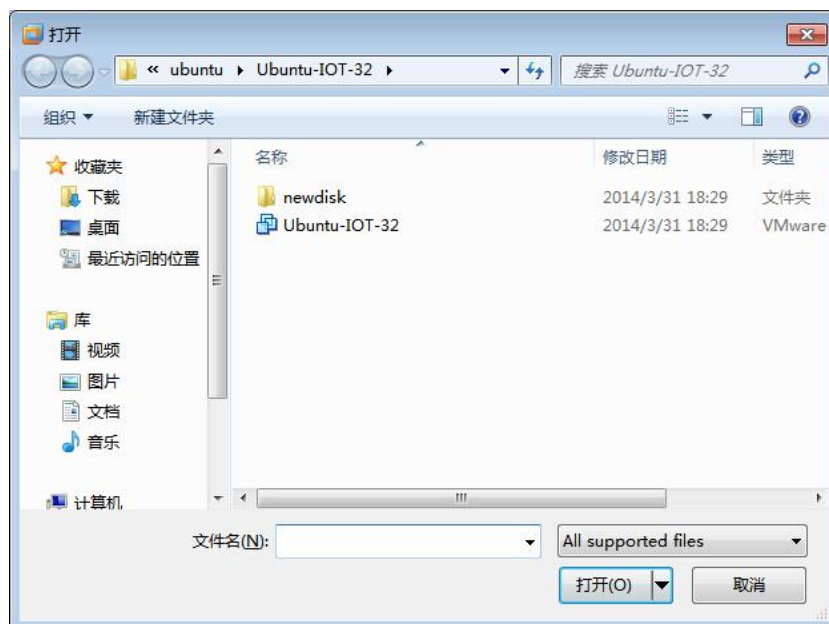


图 3-6 添加虚拟机文件

2. 启动虚拟机，点击 Ubuntu-IOT-32 的 Power on，登陆密码为 loongson

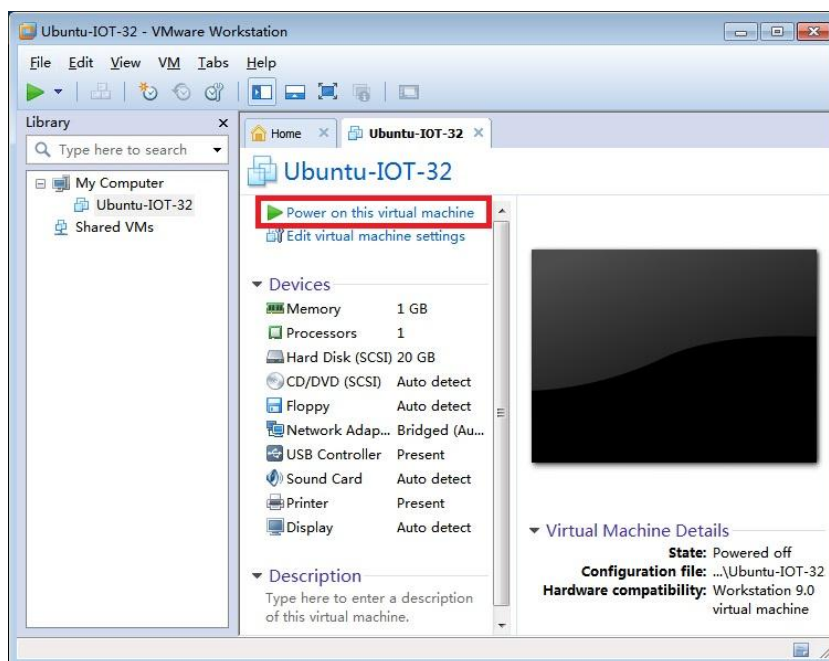


图 3-7 启动 Ubuntu 虚拟机

该虚拟机已经配置好 NFS 和 TFTP 服务。

NFS 的服务目录为: /home/nfs-debian6/rootfs/ 如图 3-8 所示:

```
loongson@ubuntu:/home/nfs-debian6/rootfs$ pwd
/home/nfs-debian6/rootfs
loongson@ubuntu:/home/nfs-debian6/rootfs$ ls
bin      dev      include  libexec  opt      sbin     share    tmp
boot     etc      kd60-230 media     proc     selinux  srv      usr
boot.sh  home     lib      mnt      root     set      sys      var
loongson@ubuntu:/home/nfs-debian6/rootfs$
```

图 3-8 NFS 服务目录和文件系统

TFTP 的服务目录为: /srv/tftp/ 该目录下有启动多路处理器实验箱的五个内核, 如图 3-9 所示:

```
loongson@ubuntu:/srv/tftp$ pwd
/srv/tftp
loongson@ubuntu:/srv/tftp$ ls
vmlinux-numa-16core vmlinux-smp-16core vmlinux-smp-8core
vmlinux-numa-8core vmlinux-smp-4core
loongson@ubuntu:/srv/tftp$
```

图 3-9 TFTP 服务目录和内核

3. 配置双 IP

将文件系统服务器的网卡设置为双 IP, IP 设置为 192.168.100.250。

该 IP 是文件系统独有的, 配置命令如下:

```
sudo ifconfig eth0:1 192.168.100.250
```

配置后可以用 `ifconfig` 命令查看，若文件系统服务器重启后，需要再次设置双 IP。

(5) 导入共享文件

文件系统服务器已配置了 `samba`，即共享文件夹，可以在 win7 系统下直接访问该文件夹，将需要拷贝至文件系统服务器的文件直接复制到该文件下。操作步骤如下：

1. 查询虚拟文件系统的原始 IP

在虚拟文件系统服务器中，用 `ifconfig` 指令查看系统网络信息，如图 3-10 所示，`eth0` 对应的 IP（192.168.1.101）为虚拟文件系统的原始 IP，用于访问 `samba` 共享文件夹；`eth0:1` 为虚拟文件系统绑定的固定双 IP。

```
loongson @ ubuntu : ~ $ ifconfig
eth0      Link encap:Ethernet  HWaddr 00:0c:29:66:c2:ed
          inet addr:192.168.1.101  Bcast:192.168.1.255  Mask:255.255.255.0
          inet6 addr: fe80::20c:29ff:fe66:c2ed/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:326257 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1503995 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:31117592 (31.1 MB)  TX bytes:2142400707 (2.1 GB)
          Interrupt:19 Base address:0x2024

eth0:1    Link encap:Ethernet  HWaddr 00:0c:29:66:c2:ed
          inet addr:192.168.100.250  Bcast:192.168.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:19 Base address:0x2024

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          inet6 addr: ::1/128 Scope:Host
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:7845 errors:0 dropped:0 overruns:0 frame:0
          TX packets:7845 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:532666 (532.6 KB)  TX bytes:532666 (532.6 KB)
```

图 3-10 虚拟文件系统 IP

2. 访问 samba 共享文件

在 Windows 7 系统下，打开运行，输入虚拟文件系统的原始 IP（192.168.1.101），如图 3-11，点击确定。

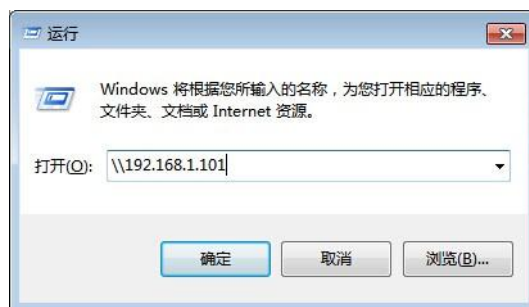


图 3-11 访问共享文件夹

系统弹出文件夹，如下图所示：

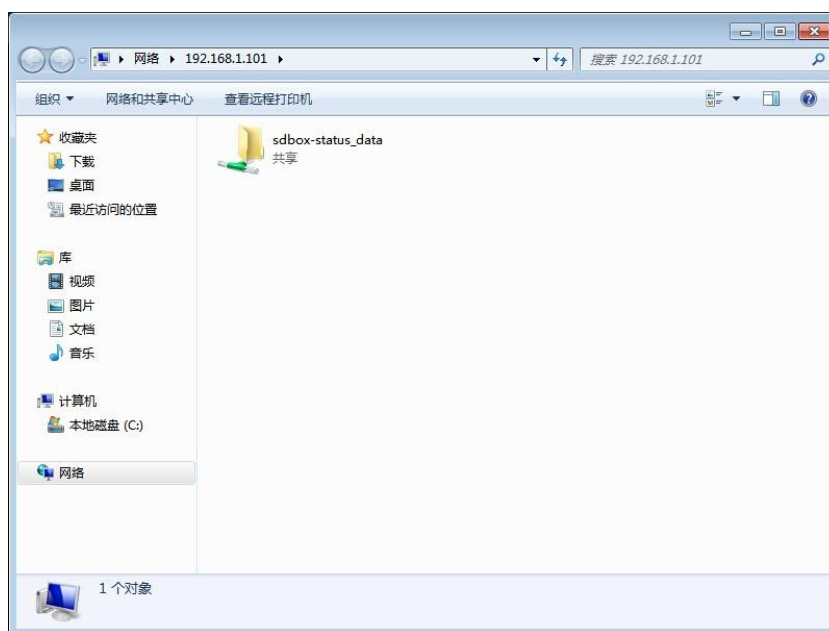


图 3-12 共享文件夹

sdbx-status_data 即为共享文件夹，双击该文件夹，弹出登陆窗口。

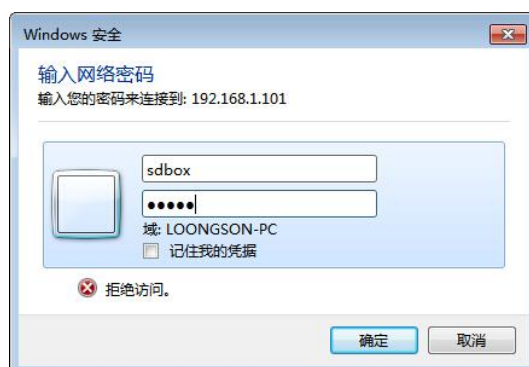


图 3-13 登陆共享文件夹

用户名和密码均为 **sdbox**，输入后点击确定，即进入该共享文件夹。

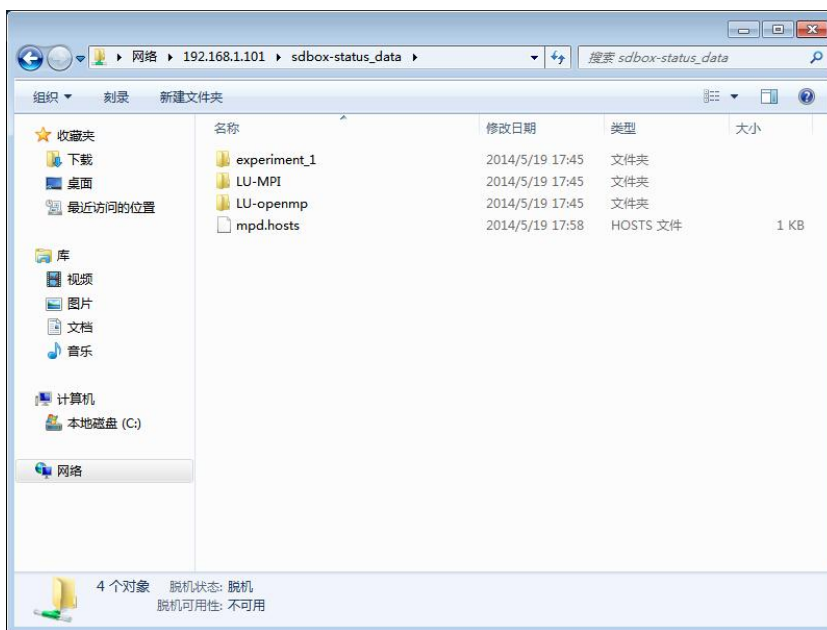


图 3-14 共享文件夹

将需要共享给学生的文件直接拷贝至该目录。学生在登陆实验箱节点后，在 `/student_data` 下可看到共享文件，拷贝至自己的私有目录即可。

```
sdbox@loongsonbox-n1:~$ ls /student_data/  
experiment_1  LU-MPI  LU-openmp  mpd.hosts
```

图 3-15 学生用户查看共享文件

2 启动处理单元

2.1 单处理单元系统

单处理单元系统是指每个处理单元作为独立的系统，可单独启动运行。硬件上需要设置每个处理单元的拨码开关，将信号位 `NODE_ID0`、`NODE_ID1`（见附表 3）统一设置为 00。实验箱硬件默认设置为四个单处理单元系统。

软件设置需要按照以下步骤操作。实验箱上电后，按控制板按钮说明启动处理单元，通过该处理单元的串口进行配置。本文以 1 号处理单元为例说明启动步骤。

（1）选择启动模式

进入 pmon 后，系统提示设置启动模式，0 代表启动 4 核系统，即当前处理单元；1 代表启动 8 核系统，启动主处理单元和从处理单元；3 代表启动 16 核系统，启动实验箱四个处理单元；f 代表保持设置。

若当前设置为 0，则保持设置，等待倒计时结束。

```
Boot mode is 0
Checkout boot one node, change it? (1: boot node0 and node1. 3: boot four node. f: skip)
04
```

图 3-16 设置启动模式

(2) 配置 NFS IP

首次使用实验箱，系统会提示配置 NFS 的 IP 地址，本文配置为 192.168.100.250。设置完毕，输入 reboot 命令重启

命令格式：[set nfsip 192.168.100.250](#)

```
Check the ENV named "nfsip" is NULL.
Please set it, follow:
"set nfsip IPADDR"
The IPADDR is your NFS server's IP.
PMON> set nfsip 192.168.100.250
```

图 3-17 设置 nps ip 地址

若 NFS 的 IP 地址有变动，需要在 pmon 下重新设置，命令同上。如果 NFS 的 IP 地址不变，之后启动处理单元则默认上次设置。

(3) 自动加载内核和文件系统

系统重启后，出现下列内容，如图 3-18 所示，192.168.100.1 是本处理单元的 IP 地址。启动单个处理单元时，只有唯一的内核，pmon 会自动加载内核。

```
00
Boot mode is 0
Checkout boot one node, change it? (1: boot node0 and node1. 3: boot four node. f: skip)
00

ifconfig rtk0 192.168.100.1
rtl_chip_info: RTL8169s/8110s, 4
r8110: link up : PHY status:0xb

Which kernel you want to start?
Input "1" to boot 4 core system.
00

Choosing 4 core system, now booting...
Loading file: tftp://192.168.100.250/vmlinux-smp-4core (elf)
(elf)
0x80300000/7449984-
```

图 3-18 加载内核

加载完内核后，系统自动加载网络文件系统，出现图 3-19 所示语句，表

明系统启动成功，进入 linux 系统。可通过 ifconfig 命令获得处理单元的 IP 地址，从其他终端用 ssh 远程登录。




图 3-19 进入系统

启动其他三个处理器单元步骤相同。首次使用实验箱时，建议按照上述步骤配置每一个处理单元，配置文件系统服务器地址，可减少后续模式切换的操作步骤。

2.2 多处理单元系统

多处理单元系统包括：8 核 SMP 系统、16 核 SMP 系统、8 核 CC-NUMA 系统和 16 核 CC-NUMA 系统。本文以 8 核系统为例，说明启动 8 核系统的步骤。请先按照 2.1 配置每个处理单元的 NFS IP。

（1）修改软件模式

选定主处理单元，启动主处理单元，在选择启动模式时，即在出现图 3-16 时，输入 1。长按主处理单元的开机键，关闭主处理单元。

（2）修改硬件配置

在硬件上，需要将实验箱配置为多处理器互连系统。

每个处理单元都配有三个黄色拨码开关，分布在处理器周围，功能请参考附表 3。实验箱配置为 8 核系统时，需要确定两个处理单元，并设置这两个处理单元拨码开关信号位 NODE_ID0、NODE_ID1。主处理单元设置为 00，从处理单元设置为 01，其中 NODE_ID0 对应低位，NODE_ID1 对应高位。例如 01 表示则 NODE_ID0 设置为 1，NODE_ID1 设置为 0。

同理，实验箱配置为 16 核系统时，同样需设置处理单元拨码开关信号位 NODE_ID0、NODE_ID1。一般选择 1 号处理单元作为主处理单元，依次将处理单元 1、2、3、4 设置为 00、01、11、10。

多路处理器实验箱可以配置为两个 8 核系统，一般情况下，1 号和 2 号处理单元组成一个 8 核系统，1 号处理单元作为主处理单元；3 号和 4 号处

理单元组成一个 8 核系统，4 号处理单元作为主处理单元。

本文以 1 号和 2 号处理单元组成一个 8 核系统为例，将 1 号和 2 号处理单元分别配置为 00，01。

（3）选择内核

硬件配置完后，启动 1 号和 2 号处理单元，串口切换至 1 号处理单元，等待设置启动模式的倒计时结束，系统提示选择内核，如下图 3-20 所示：

```
Which kernel you want to start?
Input "2" to boot 8 core SMP system.
Input "4" to boot 8 core CC-NUMA system.
00
```

图 3-20 进入系统

8 核系统下有两种内核可选择，分别为 8 核 SMP 系统，8 核 CC-NUMA 系统。在倒计时结束之前，手动输入待加载内核对应的数字，系统将自动到文件系统服务器上加载内核。例如输入 4，系统加载 8 核 numa 内核，如图 3-21。

```
Choosing 8 core CC-NUMA system, now booting...
Loading file: tftp://192.168.100.250/vmlinux-numa-8core (elf)
(elf)
0x80300000/7499424\
```

图 3-21 加载所选内核

系统启动成功后，进入 linux 系统，出现如图 3-22 所示命令行。通过 ifconfig 命令可以获得计算单元的 IP 地址，从其他终端用 ssh 远程登录。

```
root@loongsonbox-n1:/#
```

图 3-22 进入系统

在以后的使用中，若所有设置均未改变，系统将默认按照上次配置启动。

若文件系统服务器 IP 改变，则需要出现设置启动模式的倒计时时，按 Ctrl+c 退出，进入 pmon，用 set nfsip ip 指令修改服务器 IP，重启系统即可。

注意：切换启动模式时，必须首先修改软件配置，即执行第（1）步，然后再修改硬件配置。若顺序执行错误，先修改硬件配置，再配置软件，则会在启动处理单元时，出现错误，串口显示如图 3-23 的信息。系统无法正常启动。

```
0xbfe00190 : 0000824280f8f8f8
CPU CLK SEL : 00000002
CPU clk frequency = SYSCLK x 0x00000020 / 1
MEM CLK SEL : 00000012
DDR clk frequency = MEMCLK x 0x00000020 / 4
```

图 3-23 系统输入错误的打印信息

若出现这种提示，请关掉所有处理单元，将实验箱的硬件配置还原，再按照上述步骤重新配置即可。

2.3 分布式系统

实验箱可搭建为分布式系统，由相互独立的处理单元通过互联网络连接而成。实验箱的硬件配置如同 2.1 节单处理单元系统的配置。将需要启动的处理单元用网线连接在同一网络中。

开启需要连入分布式系统的处理单元，等待所有处理进入系统后，即可以运行 MPI 程序。

(1) 建立 mpd.hosts 文件

在主处理单元的家目录下，新建 mpd.hosts 文件，添加需要连接的主机名。保存退出。

例如，在 loongsonbox-n1 的家目录下，创建 mpd.hosts，连接 loongsonbox-n1 和 loongsonbox-n2 两台主机。mpd.hosts 内容如图 3-24 所示：

```
sdbox@loongsonbox-n1:~$ cat mpd.hosts
loongsonbox-n1
loongsonbox-n2
sdbox@loongsonbox-n1:~$
```

图 3-24 mpd.hosts 文件

(2) 启动分布式集群

在主处理单元的家目录下启动分布式集群，采用如下命令：

```
mpdboot -n 2 -f mpd.hosts
```

-n 后的数字表示要启动的处理单元个数，一般是不大于 mpd.hosts 文件中的机器数，比如本文上述的例子就是两台机器。

`mpdtrace`

测试命令，若输出两个主机名，说明启动成功，这样就可以执行 MPI 程序。

(3) 执行程序

在各个主机的家目录下分别拷贝待运行的 MPI 程序。在创建 `mpd.hosts` 文件的处理单元上，输入运行命令

`mpirun -np 2 ./`

`-np` 后的数字表示要运行的处理器核数，应该小于分布式系统的核总数。若启动两个处理单元，该系统中共有 2 个处理器，8 个 `cpu` 核，所以运行时，`-np` 后的数字应该不大于 8。

3 登录处理单元

从文件系统服务器或其他终端远程登陆各处理单元，指令格式：

`ssh sdbox@192.168.100.xxx`

Password: `sdbox`

- 192.168.100.xxx 为处理单元 IP
- 登陆密码为：`sdbox`

4 处理单元连接外网

若处理单元需要连接外网，按照如下指令在系统中配置网关：

`route add default gw IP`

IP 为本地网关地址

第四部分 实验箱并行结构配置

一 单处理器

1 对称多处理机（SMP）

对称多处理机是一种并行计算机的机构模型，它的结构图如图 4-1 所示。P/C（Microprocessor and Cache）是微处理器和高速缓存，SM（Shared Memory）是共享存储器，I/O 是 I/O 总线。

SMP 系统使用商用微处理器（具有片上或外置高速缓存），它们经由高速总线（或交叉开关）连向共享存储器。这种 SMP 结构具有以下特性：①对称性：系统中任何处理器均可以对称的存取任何存储单元和 I/O 设备；②单一物理地址空间：所有处理器的存储单元按单一地址空间编址；③高速缓存及其一致性：多级高速缓存可支持数据局部性，而其一致性可以由硬件来实现。

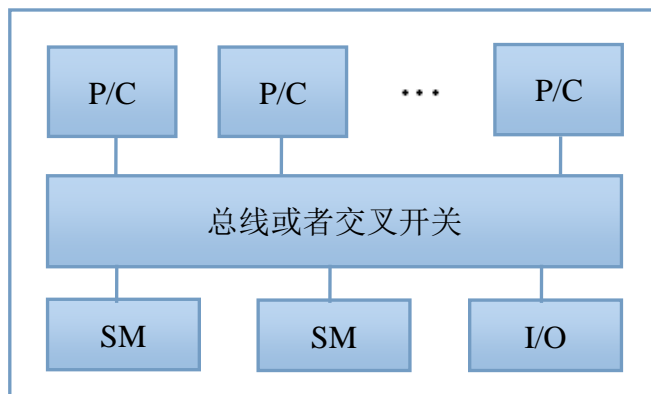


图 4-1 对称多处理机结构模型

SMP 系统的优势就是对称，每个处理器可以同等地访问共享存储器、I/O 设备和操作系统服务。正是由于对称，系统才能开拓较高的并行度。

2 4 核 SMP/CMP 系统

多路处理器计算机教学实验箱有 4 个处理单元，每个处理单元包含一个龙芯 3A 处理器、DDR2 内存、以太网卡、BIOS、串口收发芯片以及电源变换电路等。

每个处理单元可以独立控制，是一个单芯片多处理器（CMP）或对称多处理器系统（SMP）。龙芯 3A 处理器是一款配置为单节点 4 核的处理器，采用 65nm 工艺制造，最高工作主频为 1GHz，主要技术特征如下：

- 片内集成 4 个 64 位的四发射超标量 GS464 高性能处理器核；
- 片内集成 4MB 的分体共享二级 Cache(由 4 个体模块组成，每个体模块容量为 1MB)；
- 通过目录协议维护多核及 I/O DMA 访问的 Cache 一致性；
- 片内集成 2 个 64 位 400MHz 的 DDR2/3 控制器；
- 片内集成 2 个 16 位 800MHz 的 HyperTransport 控制器；
- 片内集成 2 个 16 位 800MHz 的 HyperTransport 控制器；
- 片内集成 32 位 100MHz PCIX/66MHz PCI；
- 片内集成 1 个 LPC、2 个 UART、1 个 SPI、16 路 GPIO 接口。

龙芯 3A 号芯片整体架构基于两级互连实现，结构如下图 4-2 所示。根据龙芯 3A 处理器的系统结构，处理单元集成 1 片龙芯 3A 处理器，集成 4 个高性能处理器核和共享二级 Cache，存储单元按单一地址空间编址，是一个 4 核 SMP/CMP 系统。

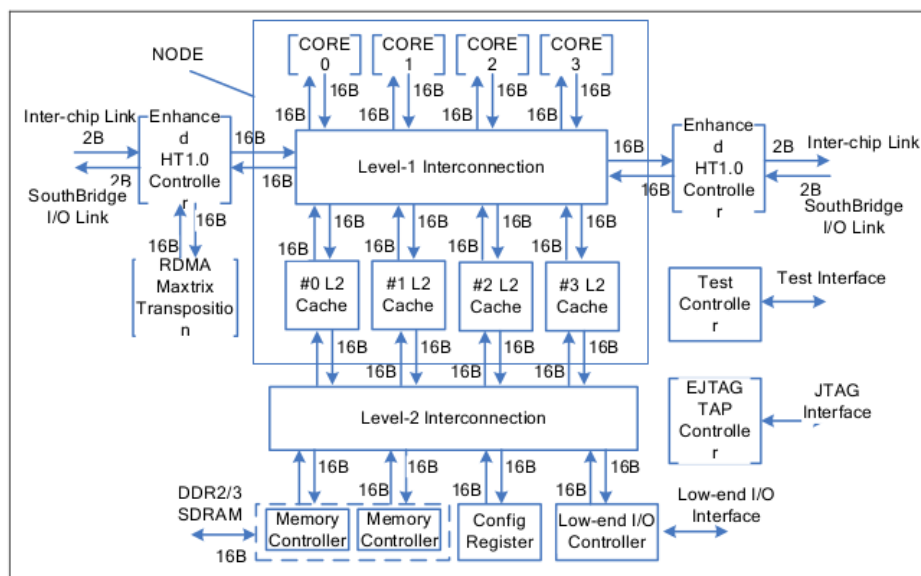


图 4-2 龙芯 3A 芯片结构

二 16 核 SMP 系统

1 系统介绍

多路处理器计算机教学实验箱的 4 个处理单元可以通过 HT 总线互连，组成 16 核的主从式（master-slave）SMP 系统。该系统由一台主处理单元记录、控制其它从处理单元的状态，并分配任务给从处理单元。

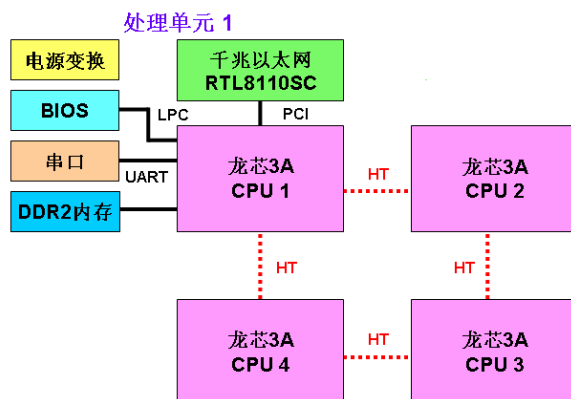


图 4-3 16 核主从式 SMP 结构

结构图如图 4-3 所示，处理单元 1 为主，其他三个处理单元为从。操作系统在主处理单元上运行，所有处理单元共享处理单元 1 的存储器、I/O 设备等。从处理单元的请求通过陷入传送给主处理机，然后主处理机回答并执行相应的服务操作。主从式 SMP 系统硬件和软件结构简单，不存在存取冲突和访问阻塞问题，可高效的分配各处理单元的任务，并行执行效率高。

但由于主处理单元的责任重大，当它来不及处理进程请求时，其它从属处理单元的利用率就会随之降低。主处理单元出现严重故障时，很容易引起整个系统的崩溃。

2 系统硬件配置

多路处理器计算机教学实验箱配置为 16 核 SMP 系统，HT0 接口硬件支持处理器核间 Cache 一致性协议，可以使用 HT0 接口构成 4 个处理单元的互连系统。图 4-4 中给出了互连的方式：

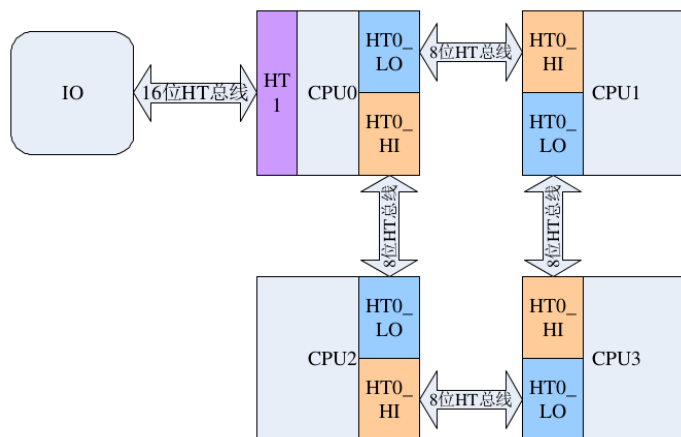


图 4-4 多处理器系统 HT 接口连接

设置处理单元拨码开关信号位 NODE_ID0、NODE_ID1（见附表 3）。依次将处理单元 0、1、2、3 设置为 00、01、10、11。NODE_ID0 对应低位，NODE_ID1 对应高位。例如 2 号处理单元，对应二进制 10，则 NODE_ID0 设置为 0，NODE_ID1 设置为 1。

三 CC-NUMA 系统

1 CC-NUMA 系统

CC-NUMA（Coherent-Cache Nonuniform Memory Access）是高速缓存一致性非均匀存储访问模型的简称，CC-NUMA 是并行计算机访存模型 NUMA 系统的一种特例。CC-NUMA 能够处理多个连接起来的处理器，每个处理器能存取一个公共的存储器组。这种结构把处理器分成几个节点，在每个节点中所有处理器互连在一起，互相通信，并可与节点内的本地存储器通信以减轻总线阻塞情况。处理器也可存取其它所有节点中的存储器组，这种存取时间随着节点的距离远近而异。处理器访问离其“近”的存储器比“远”的存储器速度要快。

实际上，CC-NUMA 系统就是将一些 SMP 机器作为一个单节点，彼此连接起来形成的一个较大的系统。CC-NUMA 既保持了 SMP 系统单一操作系统拷贝、简便的应用程序编程模式和易管理的特色，又能有效地扩充系统的规模。

图 4-5 示出 CC-NUMA 多处理机的模型结构。M（Memory）是存储器，NIC（Network Interface Circuitry）是网络接口电路，RC 是远程高速缓存。其结构的

特点是：①绝大多数商用 CC-NUMA 多处理机系统都使用基于目录的高速缓存一致性协议；②它在保留 SMP 机构易于编程的优点的同时，也改善了常规 SMP 的可扩展性问题；③它最显著的优点是程序员无需明确地在节点上分配资源，系统的硬件和软件会在开始时自动将数据分配到各节点处，在运行期间，高速缓存一致性硬件会自动地将数据移植需要使用该数据的节点上。

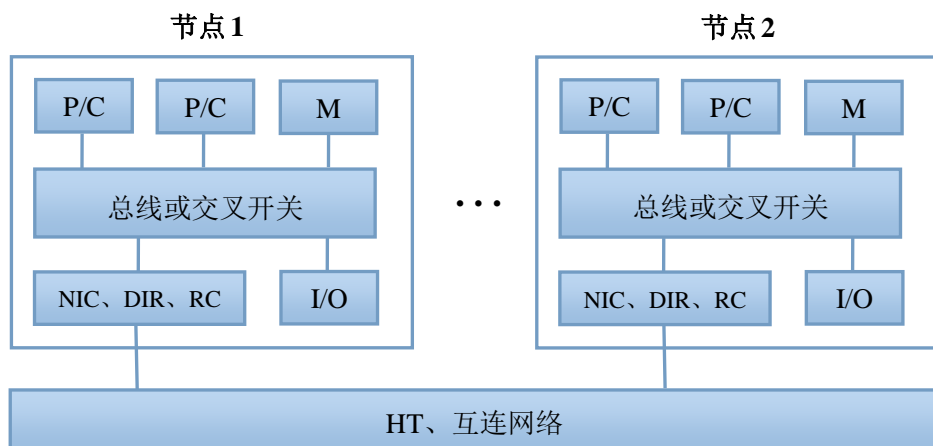


图 4-5 CC-NUMA 模型结构

2 两节点 CC-NUMA 系统

多路处理器计算机教学实验箱的处理单元可以通过 HT 总线互连，配置为两路两节点 CC-NUMA 系统。结构图如图 4-6 所示：

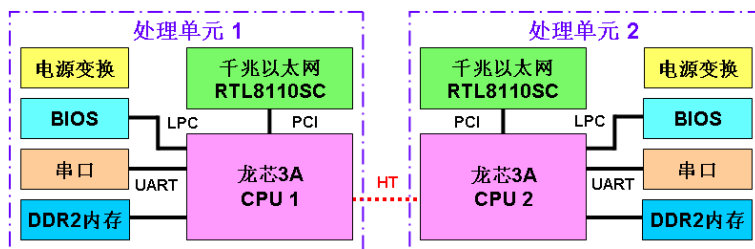


图 4-6 两节点 CC-NUMA 模型

每个是处理单元看作一个对称多处理器系统（SMP），将两个处理单元通过 HT 总线互连，形成的两节点 CC-NUMA 系统。本实验箱使用 HT0 接口构成两节点 CC-NUMA 系统，图 4-7 给出了互连的方式。

设置处理单元拨码开关信号位 NODE_ID0、NODE_ID1。将处理单元 0、1 分别设置为 00、01。本实验箱可配置两路两节点 CC-NUMA 系统，配置方式一

致。

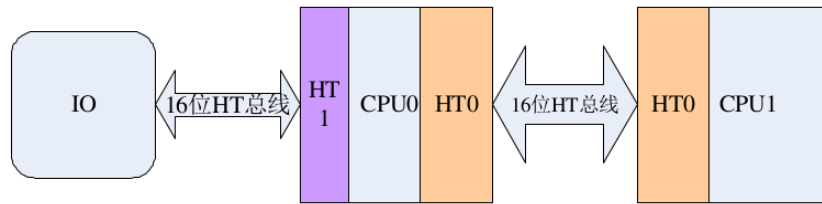


图 4-7 两节点 CC-NUMA 模型

3 四节点 CC-NUMA 系统

多路处理器计算机教学实验箱的处理单元可以通过 HT 总线两两互连，配置为四节点 CC-NUMA 系统。结构图如图 4-8 所示：

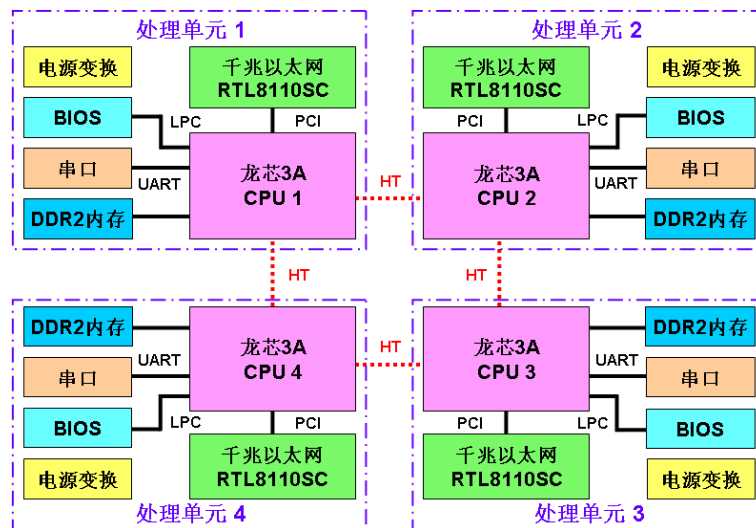


图 4-8 四节点 CC-NUMA 模型

四节点 CC-NUMA 系统的硬件配置同 16 核 SMP 系统，具体配置方法参见 16 核 SMP 系统的第二小节系统硬件配置。

四 分布式存储多处理机

分布式存储多处理机也叫做多计算机，是由多个具有本地存储模块的相互独立的处理机通过互连网络连接而成的。这种处理机特点是结构灵活、易扩充，具有较高的可扩充性和计算性。不同节点上的进程间通信要使用消息传递模型，即通过显式的收发原语来完成。在程序设计的过程中需要考虑数据分配和消息通信，通常要设计专有算法。多路处理器计算机教学实验箱的处理单元可以通过网

络互连组成分布式存储多处理机。系统结构图如图 4-9 所示：

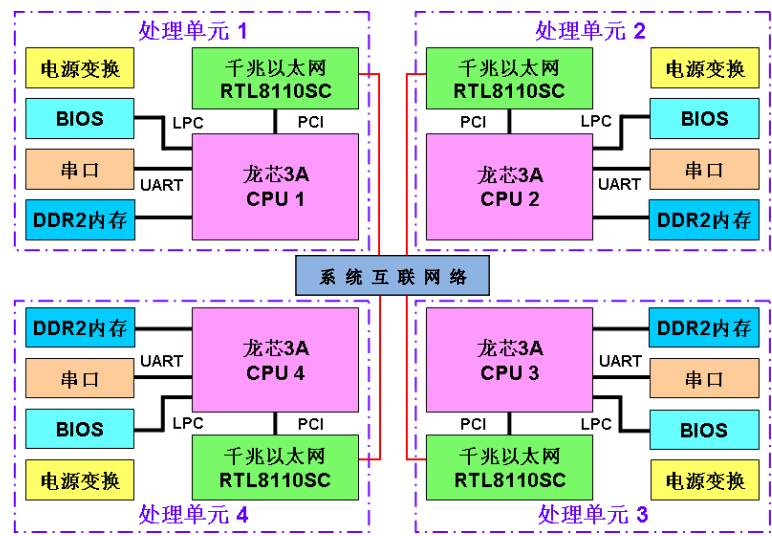


图 4-9 分布式存储多处理机模型

第五部分 并行实验指导实例

在该教学实验平台的基础上，我们编制了相应课程的实验大纲与教材，并应用于计算机本科与研究生的实际实验教学过程中，取得了良好的效果。具体实验课程表参见附件 4。下面以矩阵的 LU 分解为实例进行讲解，说明不同结构的系统配置和程序设计。用户可根据实际教学需要进行修改。

一 LU 分解实验指导案例

在线性代数中，LU 分解是矩阵分解的一种，可以将一个矩阵分解为一个下三角矩阵和一个上三角矩阵的乘积。LU 分解主要应用在数值分析中，用来解线性方程、求反矩阵或计算行列式。

对于一个 n 阶非奇异方阵 $A[a_{ij}]$ ，对 A 进行 LU 分解是求一个主对角元素全为 1 的下三角方阵 $L[l_{ij}]$ 与上三角方阵 $U[u_{ij}]$ ，使 $A=LU$ 。设 A 的各阶主子行列式皆非零， U 和 L 的元素可由下面的递推式求出：

$$a_{ij}^{(1)} = a_{ij}$$

$$a_{ij}^{(k+1)} = a_{ij}^{(k)} - l_{ik} u_{kj}$$

其中：

$$l_{ik} = \begin{cases} 0 & i < k \\ 1 & i = k \\ a_{ij}^{(k)} u_{kk}^{(-1)} & i > k \end{cases}$$

$$u_{kj} = \begin{cases} 0 & k > j \\ a_{kj}^{(k)} & k \leq j \end{cases}$$

在计算过程中，首先计算出 U 的第一行元素，然后算出 L 的第一列元素，修改相应 A 的元素；再算出 U 的第二行， L 的第二列 \cdots ，直至算出 u_m 为止。

下面在试验箱不同的配置模式下实现 LU 分解。

1 单处理器串行算法

(1) 编译、运行环境配置

1. 硬件配置

默认配置，开启一个处理节点

2. 软件配置

依赖环境：GCC

编译选项：gcc xx.c -o app

(2) 程序设计

LU 分解算法按照上述递推法则求得，具体实现如下 lu_bx1.c：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <omp.h>
#define MaxN 1010
#define infilename "LU.in"
#define outfilename "LU_bx.out"

FILE *fin,*fout;           //fin 为输入文件 fout 为输出文件
double A[MaxN][MaxN];      //A 为原矩阵
double L[MaxN][MaxN],U[MaxN][MaxN]; //L 和 U 为分解后的矩阵
int n;                     //n 为矩阵行数
int nthreads,tid;

int init()
{
    int i,j;
    fscanf(fin,"%d",&n);
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fscanf(fin,"%lf",&A[i][j]);
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
```

```
        L[i][j]=0;
        U[i][j]=0;
    }
    for (i=0; i<n; i++) L[i][i]=1;
    return 0;
}

int factorize()
{
    int i,j,k;
    for (k=0; k<n; k++)
    {
        for (i=k+1; i<n; i++)
            A[i][k]=A[i][k]/A[k][k];
        for (i=k+1; i<n; i++)
            for (j=k+1; j<n; j++)
            {
                A[i][j]=A[i][j]-A[i][k]*A[k][j];
            }
    }
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
            if (i<=j) U[i][j]=A[i][j];
            else L[i][j]=A[i][j];
        }
    return 0;
}

int output()
{
    int i,j;
    //输出 L 矩阵
    fprintf(fout,"Matrix L:\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fprintf(fout,"% .10lf%c",L[i][j],(j==n-1)?'\n':' ');
    //输出 U 矩阵
    fprintf(fout,"Matrix U:\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
```

```
        fprintf(fout, "%.10lf%c", U[i][j], (j==n-1)?'\n':' ');
    return 0;
}

int main()
{
    double ts, te;
    fin=fopen(infile, "r");
    fout=fopen(outfile, "w");
    //初始化
    init();
    //矩阵分解
    ts = omp_get_wtime(); //统一用 openmp 的计时函数，记录时间差。
    factorize();
    te = omp_get_wtime();
    printf("time = %f s\n", te - ts);
    //输出结果
    output();
    fclose(fin);
    fclose(fout);
    return 0;
}
```

2 单处理器 OpenMP 并行算法

OpenMP 是用于多线程程序设计的编译处理方案。OpenMP 支持的 C 语言编程。OpenMP 提供了对并行算法的高层的抽象描述，通过在源代码中加入专用的 `pragma` 语句，编译器就可以自动将程序进行并行化，并在必要之处加入同步互斥以及通信。

(1) 编译、运行环境配置

1. 硬件配置

默认配置，开启一个处理节点。

2. 软件配置

依赖环境：GCC >= 4.2、OpenMP 库

编译选项：gcc -o xx lu_bx2.c -fopenmp

(2) 程序设计

OpenMP 是基于线程的编程模型，设计基于多线程的 OpenMP 的 LU 分解算法，其主要思想为：外层设置一个列循环，在每次循环中开设 THREAD_NUMS 个线程，每个线程处理的矩阵 A 的行为上述的 m ，一次循环过后则完成对应列的变换，这样在 n 次循环过后便可完成矩阵 A 的 LU 分解。即 L 为 $A[k][j]$ 中 $k > j$ 的元素，其对角线上元素为 1，其它为 0， U 为 $A[k][j]$ 中 $k \leq j$ 的元素，其余为 0。

宏定义 NUM_OF_THREADS 取值范围为 [1, 4]，具体实现如下 lu_bx2.c：

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <omp.h>
#define MaxN 1010
#define infilename "LU.in"
#define outfilename "LU_bx.out"
#define NUM_OF_THREADS 4

FILE *fin,*fout;           //fin 为输入文件 fout 为输出文件
double A[MaxN][MaxN];      //A 为原矩阵
double L[MaxN][MaxN],U[MaxN][MaxN]; //L 和 U 为分解后的矩阵
int n;                      //n 为矩阵行数
int nthreads,tid;

int init()
{
    int i,j;
    fscanf(fin,"%d",&n);
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fscanf(fin,"%lf",&A[i][j]);
    omp_set_num_threads(NUM_OF_THREADS);
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
            L[i][j]=0;
            U[i][j]=0;
        }
    for (i=0; i<n; i++) L[i][i]=1;
```

```
        return 0;
    }

int factorize()
{
    int i,j,k;
#pragma omp parallel shared(A) private(tid,i,j,k)
    {
        for (k=0; k<n; k++)
        {
#pragma omp for
            for (i=k+1; i<n; i++)
                A[i][k]=A[i][k]/A[k][k];
#pragma omp for
            for (i=k+1; i<n; i++)
                for (j=k+1; j<n; j++)
                {
                    A[i][j]=A[i][j]-A[i][k]*A[k][j];
                }
        }
    }
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
            if (i<=j) U[i][j]=A[i][j];
            else L[i][j]=A[i][j];
        }
    return 0;
}

int output()
{
    int i,j;
    //输出 L 矩阵
    fprintf(fout,"Matrix L:\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fprintf(fout,"% .10lf%c",L[i][j],(j==n-1)?'\n':' ');
    //输出 U 矩阵
    fprintf(fout,"Matrix U:\n");
    for (i=0; i<n; i++)
```



```
        for (j=0; j<n; j++)
            fprintf(fout, "%.10lf%c", U[i][j], (j==n-1)?'\n':' ');
    return 0;
}

int main()
{
    double ts, te;
    fin=fopen(infile, "r");
    fout=fopen(outfile, "w");
    //初始化
    init();
    //矩阵分解
    ts = omp_get_wtime();
    factorize();
    te = omp_get_wtime();
    printf("time = %f s\n", te - ts);
    //输出结果
    output();
    fclose(fin);
    fclose(fout);
    return 0;
}
```

3 16 核系统 OpenMP 并行算法

OpenMP 可以用于共享内存并行系统的多线程程序设计，本实验系统可配置为 16 核大系统，有两种配置方式，四个处理单元可以看为一个整体单元，运行矩阵 LU 分解的多线程算法。

(1) 编译、运行环境配置

1. 硬件配置

- 16 核 SMP 系统
- 四节点 CC-NUMA

2. 软件配置

依赖环境：GCC >= 4.2、OpenMP 库

编译选项: gcc -o xx lu_bx3.c -fopenmp

(2) 程序设计

将宏定义 NUM_OF_THREADS 重新设置, 取值范围为[1, 16]。具体实现如下 lu_bx3.c :

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <sys/time.h>
#include <omp.h>
#define MaxN 1010
#define infilename "LU.in"
#define outfilename "LU_bx.out"
#define NUM_OF_THREADS 16

FILE *fin,*fout;           //fin 为输入文件  fout 为输出文件
double A[MaxN][MaxN];      //A 为原矩阵
double L[MaxN][MaxN],U[MaxN][MaxN]; //L 和 U 为分解后的矩阵
int n;                     //n 为矩阵行数
int nthreads,tid;

int init()
{
    int i,j;
    fscanf(fin,"%d",&n);
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fscanf(fin,"%lf",&A[i][j]);
    omp_set_num_threads(NUM_OF_THREADS);
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
            L[i][j]=0;
            U[i][j]=0;
        }
    for (i=0; i<n; i++) L[i][i]=1;
    return 0;
}

int factorize()
```

```
{
    int i,j,k;
#pragma omp parallel shared(A) private(tid,i,j,k)
    {
        for (k=0; k<n; k++)
        {
#pragma omp for
            for (i=k+1; i<n; i++)
                A[i][k]=A[i][k]/A[k][k];
#pragma omp for
            for (i=k+1; i<n; i++)
                for (j=k+1; j<n; j++)
                {
                    A[i][j]=A[i][j]-A[i][k]*A[k][j];
                }
        }
    }
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
        {
            if (i<=j) U[i][j]=A[i][j];
            else L[i][j]=A[i][j];
        }
    return 0;
}

int output()
{
    int i,j;
    //输出 L 矩阵
    fprintf(fout,"Matrix L:\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fprintf(fout,"% .10lf%c",L[i][j],(j==n-1)?'\n':' ');
    //输出 U 矩阵
    fprintf(fout,"Matrix U:\n");
    for (i=0; i<n; i++)
        for (j=0; j<n; j++)
            fprintf(fout,"% .10lf%c",U[i][j],(j==n-1)?'\n':' ');
    return 0;
}
```

```
int main()
{
    double ts,te;
    fin=fopen(infile,"r");
    fout=fopen(outfile,"w");
    //初始化
    init();
    //矩阵分解
    ts = omp_get_wtime();
    factorize();
    te = omp_get_wtime();
    printf("time = %f s\n", te - ts);
    //输出结果
    output();
    fclose(fin);
    fclose(fout);
    return 0;
}
```

4 分布式存储多处理机 MPI 并行算法

MPI (Message Passing Interface) 是一个基于消息传递的并行编程工具，用于开发基于消息传递的并程序，提供实际可用的、可移植的、高效的消息传递接口库，支持 C 语言。MPI 适用于分布式存储系统的并行编程，系统通过互连网络将多个处理器连接起来，每个处理器均有自己的局部存储器，所有的局部存储器构成整个地址空间，系统中各局部存储器独立编址，用户程序空间则是多地址空间，远程存储器的访问通过消息传递库程序，即 MPI 实现。

(1) 编译、运行环境配置

1. 硬件配置

配置为分布式存储多处理机。

2. 软件配置

依赖环境：GCC >= 4.2、MPI 库

编译选项: mpicc lu_bx4.c -o xx

运 行: mpirun -np 4 ./xx

(2) 程序设计

本实验采用 MPI 编程技术实现 LU 分解算法, 主要的计算是利用主行 i 对其余各行 j ($j > i$) 作初等行变换, 各行计算之间没有数据相关关系, 因此可以对矩阵 A 按行划分来实现并行计算。考虑到在计算过程中处理器之间的负载均衡, 对 A 采用行交叉划分: 设处理器个数为 p , 矩阵 A 的阶数为 n , $m = \lceil n/p \rceil$, 对矩阵 A 行交叉划分后, 编号为 i ($i = 0, 1, \dots, p-1$) 的处理器存有 A 的第 $i, i+p, \dots, i+(m-1)p$ 行。然后依次以第 $0, 1, \dots, n-1$ 行作为主行, 将其广播给所有处理器, 各处理器利用主行对其部分行向量做行变换, 这实际上是各处理器轮流选出主行并广播。若以编号为 `my_rank` 的处理器的主行元素作为主行, 并将它广播给所有处理器, 则编号大于等于 `my_rank` 的处理器利用主行元素对其第 $i+1, \dots, m-1$ 行数据做行变换, 其它处理器利用主行元素对其第 $i, \dots, m-1$ 行数据做行变换。具体实现如下 `lu_bx4.c` :

```
#include "stdio.h"
#include "stdlib.h"
#include "mpi.h"
#include "math.h"
#define a(x,y) a[x*M+y]
#define b(x) b[x]
#define A(x,y) A[x*M+y]
#define B(x) B[x]
#define floatsize sizeof(float)
#define intsize sizeof(int)
int M;
int N;
int m;
float *A;
float *B;
double starttime;
double time1;
```

```
double time2;
int my_rank;
int p;
int l;
MPI_Status status;

void fatal(char *message)
{
    printf("%s\n",message);
    exit(1);
}

void Environment_Finalize(float *a,float *b,float *x,float *f)
{
    free(a);
    free(b);
    free(x);
    free(f);
}

int main(int argc, char **argv)
{
    int i,j,t,k,my_rank,group_size;
    int i1,i2;
    int v,w;
    float temp;
    int tem;
    float *sum;
    float *f;
    float lmax;
    float *a;
    float *b;
    float *x;
    int *shift;
    FILE *fdA,*fdB;

    MPI_Init(&argc,&argv);
    MPI_Comm_size(MPI_COMM_WORLD,&group_size);
    MPI_Comm_rank(MPI_COMM_WORLD,&my_rank);
    p=group_size;
```

```

if (my_rank==0)
{
    starttime=MPI_Wtime();

    fdA=fopen("dataIn.txt","r");
    fscanf(fdA,"%d %d", &M, &N);
    if (M != N-1)
    {
        printf("the input is wrong\n");
        exit(1);
    }

    A=(float *)malloc(floatsize*M*M);
    B=(float *)malloc(floatsize*M);

    for(i = 0; i < M; i++)
    {
        for(j = 0; j < M; j++)
        {
            fscanf(fdA,"%f", A+i*M+j);
        }
        fscanf(fdA,"%f", B+i);
    }
    fclose(fdA);
}

MPI_Bcast(&M,1,MPI_INT,0,MPI_COMM_WORLD);    /* 0 号处理机将 M 广播给
所有处理机 */
m=M/p;
if (M%p!=0) m++;

f=(float*)malloc(sizeof(float)*(M+1));    /* 各处理机为主行元素建立发送和接收
缓冲区(M+1) */
a=(float*)malloc(sizeof(float)*m*M);    /* 分配至各处理机的子矩阵大小为
m*M */
b=(float*)malloc(sizeof(float)*m);    /* 分配至各处理机的子向量大小为 m */
sum=(float*)malloc(sizeof(float)*m);
x=(float*)malloc(sizeof(float)*M);
shift=(int*)malloc(sizeof(int)*M);

if (a==NULL||b==NULL||f==NULL||sum==NULL||x==NULL||shift==NULL)

```

```

        fatal("allocate error\n");

    for(i=0;i<M;i++)
        shift[i]=i;
    /*
    0 号处理机采用行交叉划分将矩阵 A 划分为大小为 m*M 的 p 块子矩阵,将 B 划分为大
    小
    为 m 的 p 块子向量, 依次发送给 1 至 p-1 号处理机
    */
    if (my_rank==0)
    {
        for(i=0;i<m;i++)
            for(j=0;j<M;j++)
                a(i,j)=A(i*p,j);

        for(i=0;i<m;i++)
            b(i)=B(i*p);
    }

    if (my_rank==0)
    {
        for(i=0;i<M;i++)
            if ((i%p)!=0)
            {
                i1=i%p;
                i2=i/p+1;

                MPI_Send(&A(i,0),M,MPI_FLOAT,i1,i2,MPI_COMM_WORLD);
                MPI_Send(&B(i),1,MPI_FLOAT,i1,i2,MPI_COMM_WORLD);
            }
    }
    /* my_rank==0 */
    else
    /* my_rank !=0 */
    {
        for(i=0;i<m;i++)
        {
            MPI_Recv(&a(i,0),M,MPI_FLOAT,0,i+1,MPI_COMM_WORLD,&status);
            MPI_Recv(&b(i),1,MPI_FLOAT,0,i+1,MPI_COMM_WORLD,&status);
        }
    }

    time1=MPI_Wtime();
    /* 开始计时 */

```



```
for(i=0;i<m;i++)                                /* 消去 */
    for(j=0;j<p;j++)
    {
        if (my_rank==j)                          /* j 号处理机负责广播主行元素 */
        {
            v=i*p+j;                             /* 主元素在原系数矩阵 A 中的行
号 and 列号为 v */
            lmax=a(i,v);
            l=v;

            for(k=v+1;k<M;k++)                    /* 在同行的元素中找最大元，并确
定最大元所在的列 l */
                if (fabs(a(i,k))>lmax)
                {
                    lmax=a(i,k);
                    l=k;
                }

            if (l!=v)                             /* 列交换 */
            {
                for(t=0;t<m;t++)
                {
                    temp=a(t,v);
                    a(t,v)=a(t,l);
                    a(t,l)=temp;
                }

                tem=shift[v];
                shift[v]=shift[l];
                shift[l]=tem;
            }

            for(k=v+1;k<M;k++)                    /* 归一化 */
                a(i,k)=a(i,k)/a(i,v);

            b(i)=b(i)/a(i,v);
            a(i,v)=1;

            for(k=v+1;k<M;k++)
                f[k]=a(i,k);
```

```
f[M]=b(i);

/* 发送归一化后的主行 */
MPI_Bcast(&f[0],M+1,MPI_FLOAT,my_rank,MPI_COMM_WORLD);
/* 发送主行中主元素所在的列号 */
MPI_Bcast(&l,1,MPI_INT,my_rank,MPI_COMM_WORLD);
}
else
{
    v=i*p+j;
    MPI_Bcast(&f[0],M+1,MPI_FLOAT,j,MPI_COMM_WORLD);
    MPI_Bcast(&l,1,MPI_INT,j,MPI_COMM_WORLD);

    if (l!=v)
    {
        for(t=0;t<m;t++)
        {
            temp=a(t,v);
            a(t,v)=a(t,l);
            a(t,l)=temp;
        }

        tem=shift[v];
        shift[v]=shift[l];
        shift[l]=tem;
    }
}

if (my_rank<=j)
    for(k=i+1;k<m;k++)
    {
        for(w=v+1;w<M;w++)
            a(k,w)=a(k,w)-f[w]*a(k,v);
        b(k)=b(k)-f[M]*a(k,v);
    }

if (my_rank>j)
    for(k=i;k<m;k++)
    {
        for(w=v+1;w<M;w++)
            a(k,w)=a(k,w)-f[w]*a(k,v);
```

```

        b(k)=b(k)-f[M]*a(k,v);
    }
}                                     /* for i j */

for(i=0;i<m;i++)
    sum[i]=0.0;

for(i=m-1;i>=0;i--)                 /* 回代 */
    for(j=p-1;j>=0;j--)
        if (my_rank==j)
        {
            x[i*p+j]=(b(i)-sum[i])/a(i,i*p+j);

            MPI_Bcast(&x[i*p+j],1,MPI_FLOAT,my_rank,MPI_COMM_WORLD);

            for(k=0;k<i;k++)
                sum[k]=sum[k]+a(k,i*p+j)*x[i*p+j];
        }
        else
        {
MPI_Bcast(&x[i*p+j],1,MPI_FLOAT,j,MPI_COMM_WORLD);

            if (my_rank>j)
                for(k=0;k<i;k++)
                    sum[k]=sum[k]+a(k,i*p+j)*x[i*p+j];

            if (my_rank<j)
                for(k=0;k<=i;k++)
                    sum[k]=sum[k]+a(k,i*p+j)*x[i*p+j];
        }

if (my_rank!=0)
    for(i=0;i<m;i++)
        MPI_Send(&x[i*p+my_rank],1,MPI_FLOAT,0,i,MPI_COMM_WORLD);
else
    for(i=1;i<p;i++)
        for(j=0;j<m;j++)
            MPI_Recv(&x[j*p+i],1,MPI_FLOAT,i,j,MPI_COMM_WORLD,&status);

if (my_rank==0)
{

```

```
printf("Input of file \"dataIn.txt\\n");
printf("%d\\t%d\\n", M, N);
for(i=0;i<M;i++)
{
    for(j=0;j<M;j++) printf("%f\\t",A(i,j));
    printf("%f\\n",B(i));
}
printf("\\nOutput of solution\\n");
for(k=0;k<M;k++)
{
    for(i=0;i<M;i++)
    {
        if (shift[i]==k) printf("x[%d]=%f\\n",k,x[i]);
    }
}

time2=MPI_Wtime();

if (my_rank==0)
{
    printf("\\n");
    printf("Whole running time      = %f seconds\\n",time2-starttime);
    printf("Distribute data time    = %f seconds\\n",time1-starttime);
    printf("Parallel compute time = %f seconds\\n",time2-time1);
}

MPI_Finalize();
Environment_Finalize(a,b,x,f);
return(0);
}
```

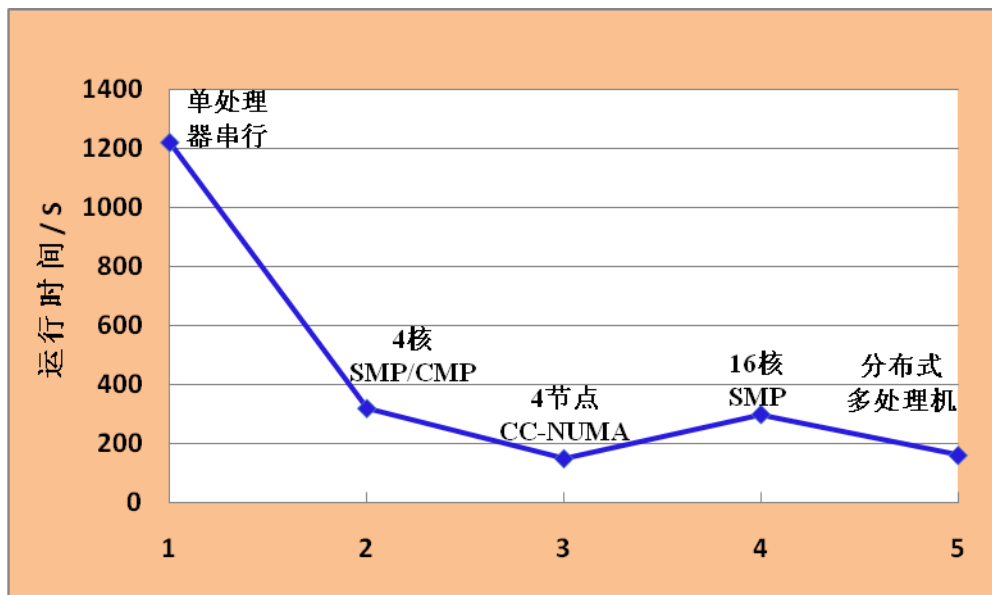
5 实验结果及分析

采用上述系统配置，在实验箱上实现非奇异方阵的 LU 分解。本对比实验中，矩阵应大于 2400 阶，小矩阵的运行结果无法真实的体现系统性能。

各种系统下，2720 阶方阵的 LU 分解运行时间和加速比如下表所示：

	1	2	3	4	5
系统类型	单处理器串行	4 核 SMP/CMP	4 节点 CC-NUMA	16 核 SMP	分布式多 处理机
运行时间 (s)	1218.3	317.39	147.21	296.96	158.14
加速比	1x	3.83x	8.27x	4.10x	7.70x

对应的曲线图如下图所示：



从图中可以看出，4 核 SMP/CMP 系统下，程序运行时间基本上是单处理器串行模式下的四分之一。单处理器串行模式下，参与运算的只有一个处理器核，而 4 核 SMP/CMP 系统下有四个处理器核并行运算，运算时间缩短到四分之一。同理，4 节点 CC-NUMA 系统是 16 个处理器核参与并行运算，其运算时间随着处理器核数基本呈线性减少。

4 节点 CC-NUMA 系统，16 核 SMP 系统和分布式多处理机都是 16 核系统，从上图曲线可以看出，4 节点 CC-NUMA 系统运行时间最短，性能最优。16 核 SMP 系统的运行时间比 4 节点 CC-NUMA 系统长，是因为该系统的 I/O 资源集

中在主处理单元，在运算过程中从处理单元需要访问主处理单元的 I/O 资源（主存），所以需要更多的交互时间。

分布式多处理的运行时间介于 4 节点 CC-NUMA 系统和 16 核 SMP 系统之间。其运行时间比 4 节点 CC-NUMA 系统长，是因为分布式多处理机中各处理单元通过网络互连，而 4 节点 CC-NUMA 系统是通过内部 HT 互连，网络互连的效率低于 HT 互连。分布式多处理机的性能优于 16 核 SMP 系统，是因为分布式多处理机的 I/O 资源在本地，即每个处理单元都有自己独立的 I/O 资源，而 16 核 SMP 系统的 I/O 资源只集中在主处理单元。

附 1 处理板硬件说明

表 1 处理板硬件脚位表

<div>处 理 器 号</div> <div>器 件 描 述</div>	处理器单元 1	处理器单元 2	处理器单元 3	处理器单元 4
处理器	U1	U13	U25	U37
内存插槽	CON2 CON3	CON5 CON6	CON8 CON9	CON11 CON12
PMON	U3	U15	U27	U39
处理器串口 0	CON1	CON4	CON7	CON10
EJTAG	CON25	CON26	CON27	CON28
拨码开关	SW1 SW2 SW3	SW5 SW6 SW7	SW9 SW10 SW11	SW13 SW14 SW15
电源跳线	J1	J2	J3	J4
复位跳线	J10	J11	J12	J13
电源和信号接口	J5			
网络指示灯	CON13			

- 说明：
1. 处理器为实际处理器脚位标号，每个主板提供 4 个处理器，编号为 1- 4。
 2. 内存插槽为每个处理器对应的两个通道的内存插槽。
 3. 每个处理器提供两个串口，一个通过电源和信号接口链接到控制板供用户使

用，一个在主板内部供其它用途，以 9 针排针形式提供。

4. EJTAG 用于处理器调试，详细用法请参考 EJTAG 功能介绍。
5. 拨码开关用于设定处理器频率，内存频率，HT 工作配置，处理器 NODE-ID 等状态，详细说明请参照拨码开关设置表格。
6. 电源跳线用于锁定处理器上电状态，仅供调试使用，不建议用户使用。
7. 复位跳线用于复位处理器，不建议用户使用。
8. 电源和信号接口用于处理器主板连接外围接口和控制板。
9. 网络指示灯排针为网络工作状态监视灯连接排线。

附 2 拨码开关设置说明

表 1 各处理单元拨码开关脚位表

<div>处 理 器 号</div> <div>拨 码 开 关</div>	处理器单元 1	处理器单元 2	处理器单元 3	处理器单元 4
CPU 频率设定	SW1	SW5	SW9	SW13
内存频率设定	SW2	SW6	SW10	SW14
HT 配置设定	SW3	SW7	SW11	SW15

表 2 拨码开关 6 位信号表

<div>处 理 器 号</div> <div>拨 码 开 关</div>	拨码开关信号位					
	1	2	3	4	5	6
CPU 频率设定	CLKSEL0	CLKSEL1	CLKSEL2	CLKSE3	CLKSE4	NODE_ID0
内存频率设定	CLKSEL5	CLKSEL6	CLKSEL7	CLKSE8	CLKSE9	NODE_ID1
HT 配置设定	CLKSE10	CLKSE11	CLKSE12	CLKSE13	CLKSE14	CLKSE15

说明：

1. CPU 频率设置

CLKSEL[4:0]对应 CPU CORE 频率设定，计算公式为：

$$\text{sysclk} * (\text{clkssel}[3:0] + 30) / (\text{clkssel}[4] + 1)$$

本 主 板 使 用 $\text{sysclk}=25\text{MHz}$ ， 并 且 $\text{sysclk} * (\text{clkssel}[3:0] + 30)$ 必 须 为 $600\text{MHz}\sim 1.36\text{GHz}$ ， sysclk 必须为 $10\sim 40\text{MHz}$ 。

2. MEM 频率设置

CLKSEL[9:5]对应 MEM 频率设定，计算公式为：

$$\text{memclk} * (\text{clkssel}[8:5] + 30) / (\text{clkssel}[9] + 3)$$

本主板使用 memclk=25MHz，并且 memclk*(clkssel[8:5]+30) 必须为 600MHz~1.36GHz，memclk必须为10~40MHz。

3. HT 配置设定功能描述见表 3。（请不要随意修改 HT 配置设定，可能造成系统不正常运行）

表 3 处理板 HT 配置设定表

信号	作用
CLKSEL[15]	1：表示采用内部参考电压，推荐使用
	0：表示采用外部参考电压
CLKSEL[14]	1：表示 HT PLL 采用差分时钟输入，即 HT0/HT1 分别使用 HT0_CLKp/HT0_CLKn 或 HT1_CLKp/HT1_CLKn 作为参考时钟。
	0：表示 HT PLL 采用普通时钟输入，即使用 HTCLK 作为参考时钟
CLKSEL[13:12]	00：表示 PHY 时钟为 1.6GHZ/1，推荐使用
	01：表示 PHY 时钟为 3.2GHZ/2
	10：表示 PHY 时钟为普通输入时钟
	11：表示 PHY 时钟为差分输入时钟
CLKSEL[11:10]	00：表示 HT 控制器时钟 200MHz，频率与参考时钟源无关
	01：表示 HT 控制器时钟 400MHz，频率与参考时钟源无关
	1x：表示 HT 控制器时钟为普通输入时钟

附 3 控制板接口说明

表 1 控制板相关接口芯片定义表

功能	标号	备注
CPCI 接口	J1	与主板对接
4 个千兆网口	J2	分别与主板 4 个网卡对应
单片机复位接口	J3	上电默认单片机复位完成，可不用。
主板串口 232 转换芯片	U1	/
单片机芯片	U2	/
电源转换芯片	U3、U4	U3 为 3.3V 转 1.8V，U4 为 5VSB 转 3.3VSB
单片机串口	CON1	保留
串口	CON2	可用于选通主板 4 个处理器的 UART1。
千兆网 LED 指示灯接口	CON3	用排线可与主板 CON3 对接
24PIN ATX 电源接口	P1	用于控制板和主板供电
8PIN 12V 电源接口	P2	/
功能按键	SW1~SW5	/

附 4 并行算法实践课程实验表

表 1 “并行算法实践”课程实验列表

序号	实验项目名称	实验基本方法和内容
实验 1	LU 分解的 openmp 实现	编写 LU 分解的 openmp 程序
实验 2	KMP 算法的 openmp 实现	编写 KMP 算法的 openmp 程序
实验 3	高斯消元法解线性方程组的 openmp 实现	编写高斯消元法解线性方程组的 openmp 程序
实验 4	计算 pi 值的 openmp 实现	编写计算 pi 值的 openmp 程序
实验 5	高斯消元法解线性方程组的 MPI 实现	编写高斯消元法解线性方程组的 MPI 程序
实验 6	约当消元法解线性方程组的 MPI 实现	编写约当消元法解线性方程组的 MPI 程序
实验 7	雅可比迭代法解线性方程组的 MPI 实现	编写雅可比迭代法解线性方程组的 MPI 实现
实验 8	LU 分解的 MPI 实现	编写 LU 分解的 MPI 程序
实验 9	随机串匹配算法的 MPI 实现	编写随机串匹配算法的 MPI 程序
实验 10	顶点倒塌算法求连通分量的 MPI 实现	编写顶点倒塌算法的 MPI 程序
实验 11	快速排序算法的 MPI 实现	编写快速排序算法的 MPI 程序

具体实验内容参见《并行算法实践》实验指导书。