实验一 Amdahl 定量原理

一、 实验平台

（1）系统简介

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。该实验平台由多路处理器教学实验箱和一台前端控制单元组成。实验平台的基本结构和软硬件结构图如下所示。

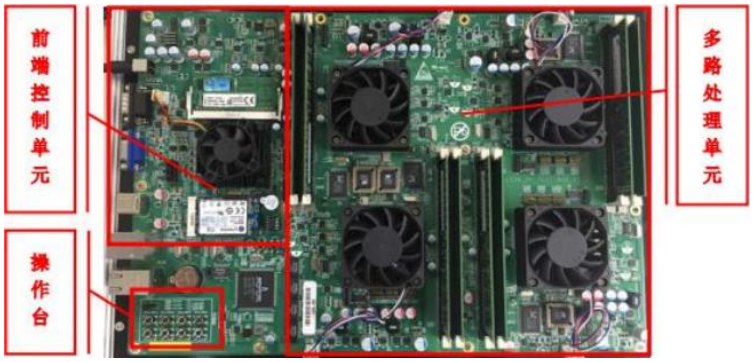


图 1-2 多路处理器教学实验箱实物图

（2）前端控制单元

前端控制单元为多路处理单元的计算节点提供网络文件系统和内核文件，组成有：固态硬盘 x1、内存 x1、2H 处理器 x1、千兆网口 x1、USB2.0 接口 x4、VGA 接口 x1、串口 x1，

图 2-2 前端控制单元（文件服务器或控制主机）

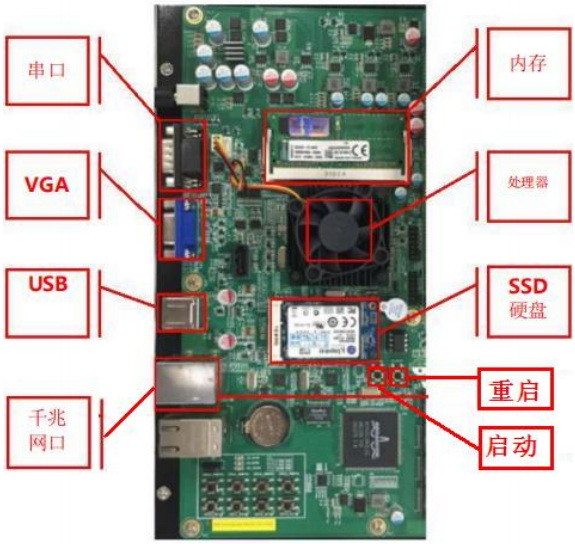


图 2-2 前端控制单元

（3）硬件系统组成

前端控制单元只有处理器、固态硬盘和内存，组成硬件系统还需要配备 USB 鼠标 x1、USB 键盘 x1 和 VGA 显示器 x1。

使用前将 USB 鼠标、USB 键盘接到 USB2.0 接口上、将显示器接到 VGA 接口、接好电源。

二、系统启动

（1）实验平台运行流程

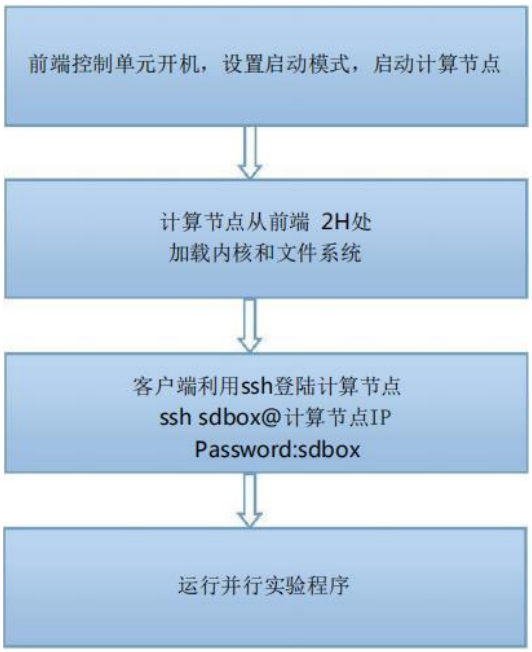


图 2-1 实验平台运行流程

（2）启动前端服务器（控制主机，前端控制单元）

按前端服务器中的启动按钮，系统进入启动页面。出现界面：

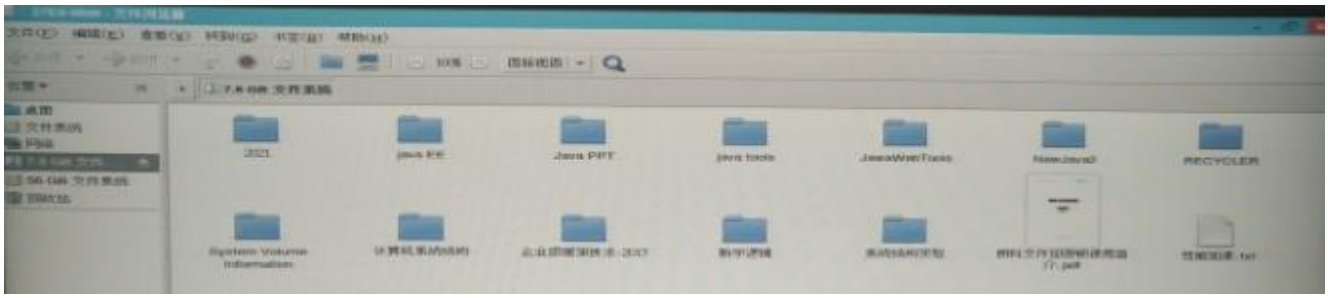
按任意键系统跳过自检过程，选择“其他”项，出现登录界面，输入用户名称：**root**

出现密码界面，输入密码：**loongson**（全部小写）
出现系统桌面

前端服务器启动完成。

（3）将实验程序考入前端文件服务器硬盘

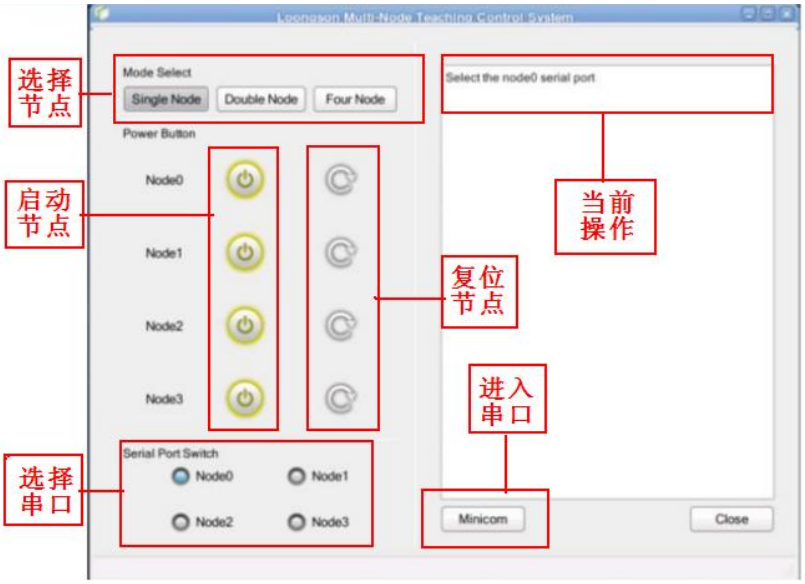
将装有实验程序的 U 盘插入 USB 口，系统自动进入 U 盘的目录。找到保持实验程序的目录，复制实验程序文件。



启动桌面的文件管理，进入 home 文件夹、将实验程序文件考入；后续从主控机向某节点机复制文件可以使用 scp 命令或拷入计算节点的共享目录 nfa-debian6。

（4）启动多路处理器教学实验箱

龙芯实验箱设置软件是用来操作和设置多路处理单元的辅助工具，集成了模式切换功能，计算节点开关机和重启功能，串口展示和切换功能。同时软件用可视化图形实时显示各计算节点的开关状态。在桌面点击“Loongson Teaching control Systen”进入。



① 模式切换（Mode Select）

三个按钮分别对应多路处理单元的三种启动模式，从左到右依次是 单节点模式（4 核 x4），双节点模式（8 核 x2，2、4 节点被 1、3 节点管理）和四节点模式（16 核 x1，2、3、4 节点被 1 节点管理）。当前实验箱启动模式为深灰色按钮，点击其他按钮可切换启动模式。

② 节点控制开关（Power Button）

每个计算节点都有对应的开关机按键和重启按键。计算节点的开关机键有三种状态：
黄色代表此节点为准备状态，设备关闭，可启动。
点击黄色的开机键后，该键变为绿色，设备开启，同时红色的重启键可用。
灰色代表此节点为兼容状态，被其他设备包含，不可启动。

③ 串口（Serial Port Switch）

点击界面右侧信息提示窗口下的 Minicom 按键，打开一个串口信息终端，用于查看节点启动过程中的系统打印信息。在打开一个串口信息终端前，应在左侧的按钮中选择要查看的节点，只能四选一，并且有信息终端打开时，必须先关闭该终端，才能选择并打开另一个串口信息终端。另：终端只适合查看节点启动过程中的系统打印信息。不能进行节点操作，否则可能产生操作错误。

注意：端口启动时间较长，应用串口信息终端查看启动过程和结果，出现问题及时纠正.

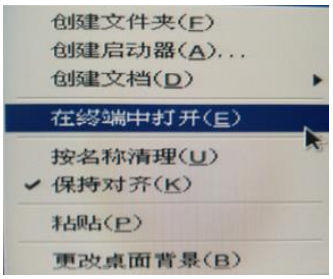
出现下列提示节点 n2 启动完成

否则，该节点启动失败，可以点重启按钮重启。或关闭后重新打开。

如本例在单节点模式下进入任一节点（如节点 1）进行实验。

（5）将 c 源程序拷贝到 n1

在桌面点击鼠标右键，出现对话框：



选择“在终端中打开”，进入终端，使用 ls 显示目录内容，就会发现当前目录是在 root 下的桌面中，使用目录命令：cd..，退到 root 目录，使用 ls 显示，进入保存 c 源程序的目录，如 diao：cd diao。将 c 源程序拷贝到 n1：

方法 1：scp 文件名.c root@192.168.100.1:/~ #拷贝文件到节点 1 的 root 用户的家目录中。

方法 2：在文件服务器文件管理器下复制文件到计算节点的共享目录 nfa-debian6 中的相应文件夹。

（6）登录处理单元

4 个处理单元的编号为 n1、n2、n3 和 n4，对用的地址为 192.168.100.1-192.168.100.4。

鼠标右键选择“在终端中打开”，出现下面内容

```
[root@localhost ~]#
```

输入命令

– ssh root@192.168.100.1

– Password: loongson

可进入 n1 节点。

三、实验原理

（1） π 的计算

本实验需要进行并行计算，这里选择循环计算 π 值，利用 Machin 公式，公式：

π ≈1 - 1/3 + 1/5 - 1/7 + 1/9 + 1/11 + 1/13 ...

计算 π 值还可以用积分求面积占比方法，具体看 C 程序代码用的是哪种方法，对实验结果的验证是一样的。

本例是利用 1/4 圆的面积积分法求解的。

使用 OpenMP 编程技术实现 π 值的计算，在循环叠加的运算过程中，开设 THREAD_NUMS 个线程，以线程数为间隔，每个线程计算对应分量，最后累加所有分量，得到 π 值，并使得 π 值满足设定的精确度。

（2）Amdahl 定量原理

并行计算中的加速比是用并行前的执行速度和并行后的执行速度之比来表示的，它表示了在并行化之后的效率提升情况。加速比（SpeedUP）可用公式：

$$S=\frac{W_s+W_p}{W_s+\frac{Wp}{P}}$$

来表示。式中 W_s,W_p 分别表示问题规模的串行分量（问题中不能并行化的那一部分）和并行分量，P 表示处理器数量。

只要注意到当 $P\to\infty$ 时，上式的极限是 WWs ，其中，$W=W_s+W_p$ 。这意味着 无论我们如何增大处理器数目，加速比是无法高于这个数的。 Amdahl（阿姆达尔）定理是固定负载（计算总量不变时）时的加速比量化标 准。如果，f 表示计算机执行某个任务的总时间不可被改进部分的执行时间所占百分比，则 $Ws+Wp$ 可以相应地表示为 $f+（1-f）$，上述公式改为:

$$S=\frac{f+(1-f)}{f+\frac{1-f}{P}}=\frac{P}{1+f(P-1)}$$

当 $P\to\infty$时，极限为 1/f。

四、实验内容

实验思想为：固定运算总量和程序中串行任务和并行任务的比例，通过不断提 高处理器的数量，并计算出相应的加速比，观测最终的加速比是否趋向于理论加 速比的极限。

样例代码使用 PI 值计算的 OpenMP 程序，编程中固定计算 Pi 的迭代次数 *NUM_ITERS*，再把迭代过程按比例 *p* 拆分成两部分(*0* 至 *LOOP1* 和 *LOOP1* 至 *LOOP2*)，前半部分使用串行计算，后半部分使用 openmp 进行并行计算。

(1)把任务全部分给串行代码，运行得到串行运行时间 T_s。

(2)把任务按 *1:4* 的串/并比例划分，再分别用不同的处理器数运行，得到 T_p，

除以 T_s得到相应的加速比。

(3)尝试按其他比例划分任务，再重复(2)，并记录运行时间。

五、**编译和运行的方法:**

编译 openmp 程序的时候需要加*-fopenmp* 参数，同时因为程序使用了数学库，所以命令中需要加 *-lm* 参数。另外，运行程序的时候，我们通过 *OMP_NUM_THREADS* 和 *PERCENTAGE* 环境变量分别控制运行的并行计算的线程数和串并任务划分百分比。

编译命令如为：gcc－lm ***-fopenmp*** file1.c －o file1

执行命令如为：*OMP_NUM_THREADS*=4 *PERCENTAGE=25* ./file1

#OMP_NUM_THREADS=4 PERCENTAGE=25 为环境变量设置执行参数，可以更改，线程最大是 16，比例最大 100.

```
sdbox@loongsonbox-n1:~$ gcc -lm -fopenmp Amdah1.c -o amdah1
sdbox@loongsonbox-n1:~$ OMP_NUM_THREADS=4 PERCENTAGE=25 ./amdah1
pi ~ 3.141593
serial time: 10.567035
parallel time: 7.905465
omp_get_max_threads(): 4
Total time: 18.472500
sdbox@loongsonbox-n1:~$ OMP_NUM_THREADS=8 PERCENTAGE=25 ./amdah1
pi ~ 3.141593
serial time: 10.566905
parallel time: 3.974850
omp_get_max_threads(): 8
Total time: 14.541755
sdbox@loongsonbox-n1:~$ OMP_NUM_THREADS=16 PERCENTAGE=25 ./amdah1
pi ~ 3.141593
serial time: 10.567010
parallel time: 2.000749
omp_get_max_threads(): 16
Total time: 12.567759
sdbox@loongsonbox-n1:~$
```

对不同的 percent 和 threads 通过计算得到：serial_time、parallel_time、total_time，再通过原理给出的公式，计算每个 speedup

实验数据： 表 1-1 不同处理器和任务串并比下的 Pi 值计算时间

percent	threads	serial_time	parallel_time	total_time	speedup
25	1				
	2				
	4				

	6				
	8				
	12				
	16				
50	1				
	2				
	4				
	6				
	8				
	12				
	16				
75	1				
	2				
	4				
	6				
	8				
	12				
	16				
100	1				

Percent 串行任务占并行任务的百分比

threads 线程数

serial_time 串行任务的运行时间

parallel_time 并行任务的运行时间

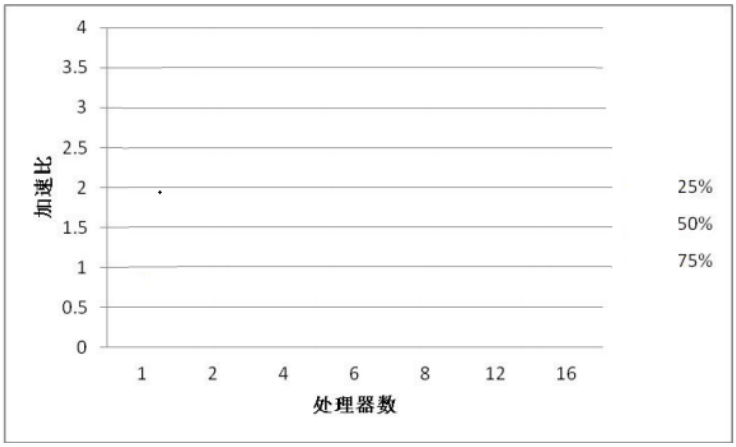
total_time 任务运行的总时间

speedup 加速比

5、 实验分析

将表 1-1 的数据绘制图表如图 1-1 所示。

图 1-1 加速比与处理器数量、任务串并比的关系

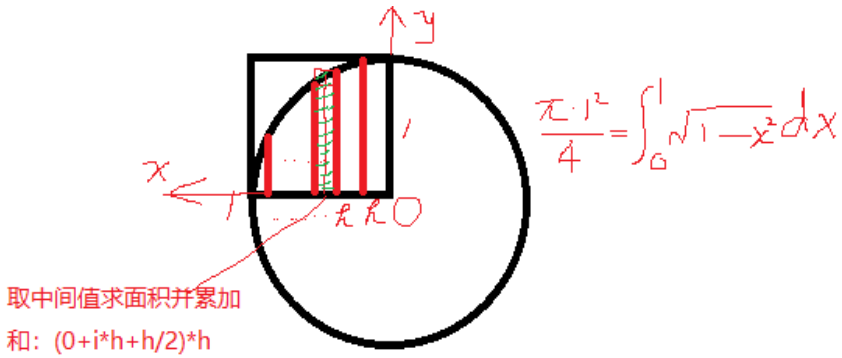


（1）通过观察上图，我们可以发现？

（2）本实验与 Amdahl 定理的描述相一致吗？

（3）谈谈你所理解的 Amdahl 定理在多处理机下的应用。

附件：程序代码原理



即：$\frac{\pi}{4} = \int_0^1 \sqrt{1-x^2}\,dx \approx \sum_{i=0}^{N-1}(x_0 + ih + h/2)h = h\sum_{i=0}^{N-1}(x_0 + ih + h/2)$

```
/*
* Calculate $\pi = 4 \int_0^1 \sqrt{1 - x^2} \, dx$
* using a rectangle rule $\sum_{i=1}^N f(x_0 + i h - h/2) h$
*
* Compile as $> gcc -Wall -Wextra -fopenmp -lm
* Run as $> OMP_NUM_THREADS=2 PERCENTAGE=30 ./a.out
*/
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <omp.h>
const long long NUM_ITERS = 100000000;
int percentage = 30;
long long LOOP1;
long long LOOP2;
void init_percentage()
{
char *s = getenv("PERCENTAGE");
if (s)
sscanf(s, "%d", &percentage);
LOOP1 = NUM_ITERS / 100. * percentage;
LOOP2 = NUM_ITERS;
}
int main(int argc, char *argv[])
{
const double L = 1.0;
const double h = L / NUM_ITERS;
const double x_0 = 0.0;
double pi;
double s_t, p_t, t_1, t_2;
double total_time = 0;
int i;
double sum = 0.0;
init_percentage();
/* serial */
t_1 = omp_get_wtime();
for (i = 0; i < LOOP1; ++i)
{
double x = x_0 + i * h + h/2;
sum += sqrt(1 - x*x);
}
t_2 = omp_get_wtime();
s_t = t_2 - t_1;
/* parallel */
t_1 = omp_get_wtime();
#pragma omp parallel for reduction(+: sum)
schedule(static)
for (i = LOOP1; i < LOOP2; ++i)
{
double x = x_0 + i * h + h/2;
sum += sqrt(1 - x*x);
}
t_2 = omp_get_wtime();
p_t += t_2 - t_1;
total_time += s_t + p_t;
pi = sum * h * 4.0;
if (argc == 1) {
printf("pi ~ %f\n", pi);
printf("serial time: %f\n", s_t);
printf("parallel time: %f\n", p_t);
printf("omp_get_max_threads(): %d\n",
omp_get_max_threads());
printf("Total time: %f\n", total_time);
} else {
printf("%d\t%d\t%f\t%f\t%f\n",percentage,omp_
get_max_threads(),s_t,p_t,total_time);
}
return 0;
}
```

实验准备：本实验事先熟悉一下 linux 基本命令有：cd, ls, cp, scp, cat, vi, su, sudo, who, ssh 等命令及 linux 目录配置等