快排算法实验手册

版本：1.0



华为技术有限公司

目录

[1 课程介绍 1](#_Toc46492263)

[1.1 简介 1](#_Toc46492264)

[1.2 内容描述 1](#_Toc46492265)

[1.3 读者知识背景 1](#_Toc46492266)

[1.4 实验环境说明 1](#_Toc46492267)

[2 快排算法实验 2](#_Toc46492268)

[2.1 实验介绍 2](#_Toc46492269)

[2.1.1 关于本实验 2](#_Toc46492270)

[2.1.2 教学目标 2](#_Toc46492271)

[2.2 实验任务操作指导 2](#_Toc46492272)

[2.2.1 创建快排算法源码 2](#_Toc46492273)

[2.2.2 创建makefile 4](#_Toc46492274)

[2.2.3 进行编译 4](#_Toc46492275)

[2.2.4 建立主机配置文件 5](#_Toc46492276)

[2.2.5 运行监测 5](#_Toc46492277)

[2.3 思考题及答案 6](#_Toc46492278)

# 课程介绍

## 简介

本书适用于学习并行计算课程的学生进行实验练习，完成本实验手册后，您将能更加充分理解集群快排算法，掌握在华为鲲鹏上如何运行。

## 内容描述

本实验指导书通过在华为鲲鹏上，编译运行快排算法程序。完成实验操作后，读者会掌握快排算法程序的编写。

## 读者知识背景

本课程为并行计算基础课程，为了更好地掌握本书内容，阅读本书的读者应首先具备以下基本条件：

* 具备基本的Linux命令能力；

## 实验环境说明

* 华为鲲鹏云主机、openEuler 20.03操作系统；
* 每套实验环境可供1名学员上机操作。

# 快排算法实验

## 实验介绍

### 关于本实验

实现多台主机快排算法的编译运⾏。

### 教学目标

掌握多台主机快排算法的编写。

## 实验任务操作指导

### 创建快排算法源码

以下步骤均在ecs-hw-0001上，以zhangsan用户执行。

执行以下命令，创建quicksort目录存放该程序的所有文件, 并进入quicksort目录（四台主机都执行）

mkdir /home/zhangsan/quicksort

cd /home/zhangsan/quicksort

执行以下命令，创建快排算法源码quick\_sort.cpp（四台主机都执行）

vim quick\_sort.cpp

代码内容如下：

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <time.h>

#include <iostream>

#include "omp.h"

using namespace std;

void QuickSort(int \*&array, int len) {

if (len <= 1) return;

int pivot = array[len / 2];

int left\_ptr = 0;

int right\_ptr = len - 1;

while (left\_ptr <= right\_ptr) {

while (array[left\_ptr] < pivot) left\_ptr += 1;

while (array[right\_ptr] > pivot) right\_ptr -= 1;

if (left\_ptr <= right\_ptr) {

swap(array[left\_ptr], array[right\_ptr]);

left\_ptr += 1;

right\_ptr -= 1;

}

}

int \*sub\_array[] = {array, &(array[left\_ptr])};

int sub\_len[] = {right\_ptr + 1, len - left\_ptr};

#pragma omp task default(none) firstprivate(sub\_array, sub\_len)

{ QuickSort(sub\_array[0], sub\_len[0]); }

#pragma omp task default(none) firstprivate(sub\_array, sub\_len)

{ QuickSort(sub\_array[1], sub\_len[1]); }

// for (int i = 0; i < 2; i++) QuickSort(sub\_array[i], sub\_len[i]);

}

int main(int argc, char \*argv[]) {

srand(time(NULL));

if (argc != 3) {

cout << "Usage: " << argv[0] << " thread-num array-len\n";

exit(-1);

}

int t = atoi(argv[1]);

int n = atoi(argv[2]);

int \*array = new int[n];

omp\_set\_num\_threads(t);

unsigned int seed = 1024;

#pragma omp parallel for

for (int i = 0; i < n; i++) array[i] = rand\_r(&seed);

struct timeval start, stop;

gettimeofday(&start, NULL);

#pragma omp parallel default(none) shared(array, n)

{

#pragma omp single nowait

{ QuickSort(array, n); }

}

gettimeofday(&stop, NULL);

double elapse = (stop.tv\_sec - start.tv\_sec) \* 1000 +

(stop.tv\_usec - start.tv\_usec) / 1000;

cout << elapse << " " << n << endl;

for (int i = 0; i < n - 1; i++) {

if (array[i] > array[i + 1]) {

cerr << "quick sort fails! \n";

break;

}

}

return 0;

}

### 创建makefile

执行以下命令，创建Makefile（四台主机都执行）

vim Makefile

代码内容如下：

CC = g++

CCFLAGS = -I . -O2 -fopenmp

LDFLAGS = # -lopenblas

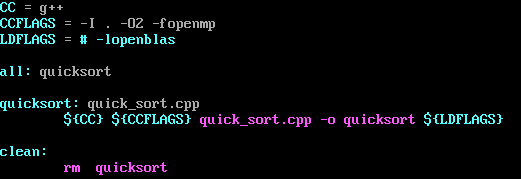
all: quicksort

quicksort: quick\_sort.cpp

${CC} ${CCFLAGS} quick\_sort.cpp -o quicksort ${LDFLAGS}

clean:

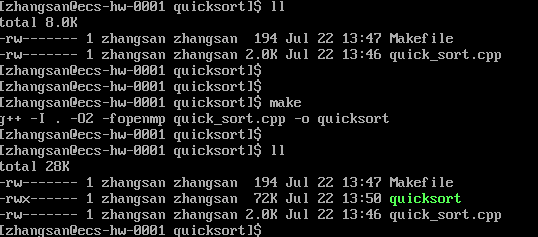
rm quicksort



### 进行编译

执行以下命令，进行编译（四台主机都执行）

make



### 建立主机配置文件

执行以下命令，建立主机配置文件（四台主机都执行）

vim /home/zhangsan/quicksort/hostfile

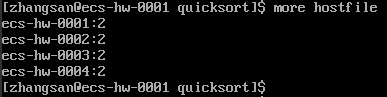
添加内容如下：

ecs-hw-0001:2

ecs-hw-0002:2

ecs-hw-0003:2

ecs-hw-0004:2



### 运行监测

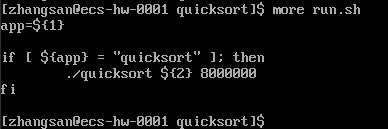
编写run.sh脚本，内容如下：

app=${1}

if [ ${app} = "quicksort" ]; then

./quicksort ${2} 8000000

fi



分别执行以下命令，查看快排算法运行结果（只需要在ecs-hw-0001上执行）

bash run.sh quicksort 1

bash run.sh quicksort 2

bash run.sh quicksort 3

bash run.sh quicksort 4

bash run.sh quicksort 5

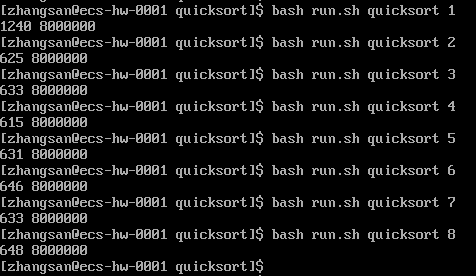
bash run.sh quicksort 6

bash run.sh quicksort 7

bash run.sh quicksort 8

1-8数字表示启动处理的进程数量。

结果如下：



通过上述运行，可以看出快排算法程序已经在集群中并行运行起来。大致从整体可以看出，随着进程数量的增加，耗时越来越少。从开始的1240减少到630左右。

## 思考题及答案

* 链接过程进行了什么操作？静态链接器和动态链接器的区别是什么？

参考答案：链接就是将不同部分的代码和数据收集和组合成为一个单一文件的过程,这个文件可被加载或拷贝到存储器执行，链接是由链接器自动执行的。

静态链接器以一组可重定位目标文件和命令行参数作为输入，生成一个完全链接的可以加载和运行的可执行目标文件作为输出。

共享库是一个目标模块,在运行时,可以加载到任意的存储器地址，并在存储器中和一个程序链接起来。这个过程称为动态链接，是由动态链接器完成的。