WordCount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](#_Toc53471547)

[1.1 简介 1](#_Toc53471548)

[1.2 内容描述 1](#_Toc53471549)

[1.3 读者知识背景 1](#_Toc53471550)

[1.4 实验环境说明 1](#_Toc53471551)

[2 WordCount实验 2](#_Toc53471552)

[2.1 实验介绍 2](#_Toc53471553)

[2.1.1 关于本实验 2](#_Toc53471554)

[2.1.2 教学目标 2](#_Toc53471555)

[2.2 实验任务操作指导 2](#_Toc53471556)

[2.2.1 创建WordCount源码 2](#_Toc53471557)

[2.2.2 创建makefile 8](#_Toc53471558)

[2.2.3 进行编译 9](#_Toc53471559)

[2.2.4 建立主机配置文件 10](#_Toc53471560)

[2.2.5 运行监测 10](#_Toc53471561)

[2.3 思考题及答案 11](#_Toc53471562)

# 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WordCount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WordCount程序。完成实验操作后，读者会掌握简单的程序编写，如WordCount中的getWords、countWords、treeMerge。

## 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；
* 安装mpich-3.3.2.tar.gz；
* 安装OpenBLAS-0.3.8.tar.gz；
* 每套实验环境可供1名学员上机操作。

# WordCount实验

## 实验介绍

### 关于本实验

实现多台主机WordCount算法的编译运⾏。

### 教学目标

掌握多台主机WordCount算法的编写运行。

## 实验任务操作指导

### 创建WordCount源码

实验说明：实验将提供两个文件夹，第一个文件夹包含100个小文件，第二文件夹包含一个大文件。请针对这两种情况分别实现WordCount算法，同时把结果打印到屏幕上。WordCount算法可分解为三步，分别为getWords 、countWords 、treeMerge。

以下步骤均在ecs-hw-0001上，以zhangsan用户执行。

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

mkdir /home/zhangsan/wordcount

cd /home/zhangsan/wordcount

执行以下命令，创建存放测试数据的目录（四台主机都执行）

mkdir –p project\_file/big\_file

mkdir –p project\_file/small\_file

执行以下命令，创建WordCount源码wordcount.cpp（四台主机都执行）

vim wordcount.cpp

代码内容如下：

#include <math.h>

#include <stdio.h>

#include <stdlib.h>

#include <sys/time.h>

#include <string>

#include <cstring>

#include <fstream>

#include <sstream>

#include <iterator>

#include <vector>

#include <map>

#include <unordered\_map>

#include <dirent.h>

#include <iostream>

#include "mpi.h"

using namespace std;

void getFiles(string path, vector<string>& filenames)

{

DIR \*pDir;

struct dirent\* ptr;

if(!(pDir = opendir(path.c\_str()))){

return;

}

while((ptr = readdir(pDir))!=0) {

if (strcmp(ptr->d\_name, ".") != 0 && strcmp(ptr->d\_name, "..") != 0){

filenames.push\_back(path + "/" + ptr->d\_name);

}

}

closedir(pDir);

}

std::string readFile(std::string filename) {

std::ifstream in(filename);

in.seekg(0, std::ios::end);

size\_t len = in.tellg();

in.seekg(0);

std::string contents(len + 1, '\0');

in.read(&contents[0], len);

return contents;

}

std::vector<std::string> split(std::string const &input) {

std::istringstream buffer(input);

std::vector<std::string> ret((std::istream\_iterator<std::string>(buffer)),

std::istream\_iterator<std::string>());

return ret;

}

std::vector<std::string> getWords(

std::string &content, int rank, int worldsize) {

std::vector<std::string> wordList = split(content);

std::vector<std::string> re;

std::string tmp;

for (int i = 0 ; i < wordList.size(); i++) {

if (i % worldsize) {

// re.push\_back(wordList[i]);

tmp += " " + wordList[i];

}

}

re.push\_back(tmp);

return re;

}

std::vector<pair<std::string, int>> countWords(

std::vector<std::string> &contentList) {

// split words

std::vector<std::string> wordList;

std::string concat\_content;

for (auto it = contentList.begin(); it != contentList.end(); it++) {

std::string content = (\*it);

concat\_content += " " + content;

}

wordList = split(concat\_content);

// do the word count

std::map<std::string, int> counts;

for (auto it = wordList.begin(); it != wordList.end(); it++) {

if (counts.find(\*it) != counts.end()) {

counts[\*it] += 1;

} else {

counts[\*it] = 1;

}

}

std::vector<pair<std::string, int>> res;

for (auto it = counts.begin(); it != counts.end(); it++) {

res.push\_back(std::make\_pair(it->first, it->second));

}

return res;

}

std::vector<pair<std::string, int>> mergeCounts(

std::vector<pair<std::string, int>> &countListA,

std::vector<pair<std::string, int>> &countListB) {

std::map<std::string, int> counts;

for (auto it = countListA.begin(); it != countListA.end(); it++) {

counts[it->first] = it->second;

}

for (auto it = countListB.begin(); it != countListB.end(); it++) {

if (counts.find(it->first) == counts.end())

counts[it->first] = it->second;

else

counts[it->first] += it->second;

}

std::vector<pair<std::string, int>> res;

for (auto it = counts.begin(); it != counts.end(); it++) {

res.push\_back(std::make\_pair(it->first, it->second));

}

return res;

}

void sendLocalCounts(int from, int to,

std::vector<pair<std::string, int>> &counts) {

int num = counts.size();

MPI\_Send(&num, 1, MPI\_INT, to, from, MPI\_COMM\_WORLD);

if (num) {

int \*counts\_array = new int[num];

int i = 0;

for (auto it = counts.begin(); it != counts.end(); it++, i++) {

counts\_array[i] = it->second;

}

MPI\_Send(counts\_array, num, MPI\_INT, to, from, MPI\_COMM\_WORLD);

delete counts\_array;

}

std::string words = " ";

for (auto it = counts.begin(); it != counts.end(); it++) {

words += it->first;

words += " ";

}

num = words.length();

MPI\_Send(&num, 1, MPI\_INT, to, from, MPI\_COMM\_WORLD);

if (num) {

char \*\_words = new char[num];

words.copy(\_words, num);

MPI\_Send(\_words, num, MPI\_CHAR, to, from, MPI\_COMM\_WORLD);

delete \_words;

}

}

void recvCounts(int from, int to, std::vector<pair<std::string, int>> &counts) {

MPI\_Status status;

int \_num = 0, num = 0;

int \*counts\_array;

char \*\_words;

std::string words;

MPI\_Recv(&\_num, 1, MPI\_INT, from, from, MPI\_COMM\_WORLD, &status);

if (\_num) {

counts\_array = new int[\_num];

MPI\_Recv(counts\_array, \_num, MPI\_INT, from, from, MPI\_COMM\_WORLD, &status);

}

MPI\_Recv(&num, 1, MPI\_INT, from, from, MPI\_COMM\_WORLD, &status);

if (num) {

\_words = new char[num];

MPI\_Recv(\_words, num, MPI\_CHAR, from, from, MPI\_COMM\_WORLD, &status);

for (int \_i = 0; \_i < num; \_i++) words+=\_words[\_i];

delete \_words;

}

if (\_num) {

std::vector<std::string> word\_vec = split(words);

for (int i = 0; i < \_num; i++) {

counts.push\_back(std::make\_pair(word\_vec[i], counts\_array[i]));

}

delete counts\_array;

}

}

void treeMerge(int id, int worldSize,

std::vector<pair<std::string, int>> &counts) {

int participants = worldSize;

while (participants > 1) {

MPI\_Barrier(MPI\_COMM\_WORLD);

int \_participants = participants / 2 + (participants % 2 ? 1 : 0);

if (id < \_participants) {

if (id + \_participants < participants) {

std::vector<pair<std::string, int>> \_counts;

std::vector<pair<std::string, int>> temp;

recvCounts(id + \_participants, id, \_counts);

temp = mergeCounts(\_counts, counts);

counts = temp;

}

}

if (id >= \_participants && id < participants) {

sendLocalCounts(id, id - \_participants, counts);

}

participants = \_participants;

}

}

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

int rank;

int worldSize;

MPI\_Init(&argc, &argv);

MPI\_Comm\_size(MPI\_COMM\_WORLD, &worldSize);

MPI\_Comm\_rank(MPI\_COMM\_WORLD, &rank);

/\*

\* Word Count for big file

\*/

{

struct timeval start, stop;

gettimeofday(&start, NULL);

std::string big\_file = "./project\_file/big\_file/big\_100.txt";

auto content = readFile(big\_file);

auto partContent = getWords(content, rank, worldSize);

auto counts = countWords(partContent);

treeMerge(rank, worldSize, counts);

gettimeofday(&stop, NULL);

if (rank == 0) {

cout << "word count: "

<< (stop.tv\_sec - start.tv\_sec) \* 1000.0 +

(stop.tv\_usec - start.tv\_usec) / 1000.0

<< " ms"<< endl;

}

/\*

if (rank == 0) {

for (auto it = counts.begin(); it != counts.end(); it++) {

std::cout << it->first << " : " << it->second << endl;

}

}

\*/

}

/\*

\* Word Count for small file

\*/

{

struct timeval start, stop;

gettimeofday(&start, NULL);

std::string small\_file\_folder = "./project\_file/small\_file/";

std::vector<std::string> \_small\_files, small\_files;

getFiles(small\_file\_folder, \_small\_files);

for (auto it = \_small\_files.begin(); it!= \_small\_files.end(); it++) {

std::size\_t \_hs = std::hash<std::string>{}(\*it);

if (int(\_hs % worldSize) == rank) {

small\_files.push\_back(\*it);

}

}

std::vector<std::string> wdlst;

for (int i = 0; i < small\_files.size(); i++) {

auto content = readFile(small\_files[i]);

wdlst.push\_back(content);

}

auto counts = countWords(wdlst);

treeMerge(rank, worldSize, counts);

gettimeofday(&stop, NULL);

if (rank == 0) {

cout << "word count: "

<< (stop.tv\_sec - start.tv\_sec) \* 1000.0 +

(stop.tv\_usec - start.tv\_usec) / 1000.0

<< " ms"<< endl;

}

// if (rank == 0) {

// for (auto it = counts.begin(); it != counts.end(); it++) {

// std::cout << it->first << " : " << it->second << endl;

// }

// }

}

MPI\_Finalize();

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 = mpic++

CCFLAGS = -O2 -fopenmp

LDFLAGS = -lopenblas

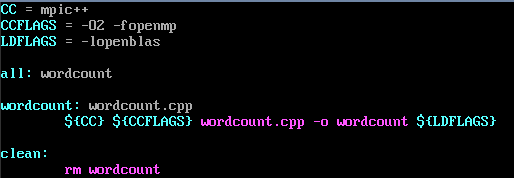
all: wordcount

wordcount: wordcount.cpp

${CC} ${CCFLAGS} wordcount.cpp -o wordcount ${LDFLAGS}

clean:

rm wordcount



### 进行编译

执行以下命令，获取包（四台主机都执行）

wget https://github.com/xianyi/OpenBLAS/archive/v0.3.8.tar.gz

执行以下命令，进行编译（四台主机都执行）

tar -zxvf v0.3.8.tar.gz && cd OpenBLAS-0.3.8

make -j2

sudo make PREFIX=/usr/local/openblas install

sudo chmod -R 777 /usr/local/openblas/

执行以下命令，完成安装（四台主机都执行）

sudo ln -s /usr/local/openblas/lib/libopenblas.so /usr/lib/libopenblas.so

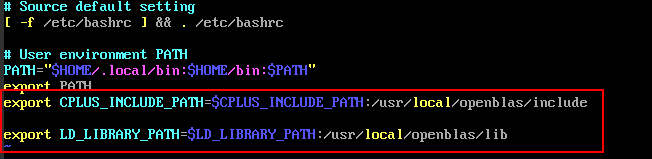
执行以下命令，配置OpenBLAS环境（四台主机都执行）

vim ~/.bashrc

添加下列内容

export CPLUS \_INCLUDE\_PATH=$CPLUS\_INCLUDE\_PATH: /usr/local/openblas/include

export LD\_LIBRARY\_PATH=$LD\_LIBRARY\_PATH:/usr/local/openblas/lib

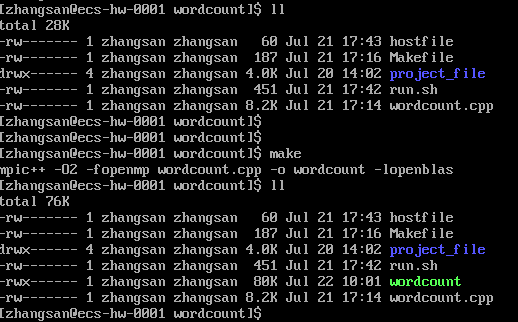


执行以下命令，使环境变量均生效（四台主机都执行）

source ~/.bashrc

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/wordcount/hostfile

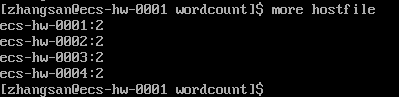
添加内容如下：

ecs-hw-0001:2

ecs-hw-0002:2

ecs-hw-0003:2

ecs-hw-0004:2



### 运行监测

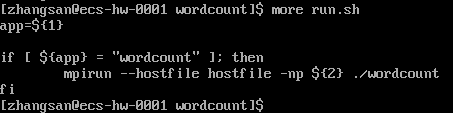
编写run.sh脚本，内容如下：

app=${1}

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

mpirun --hostfile hostfile -np ${2} ./wordcount

fi



分别执行以下命令，查看wordcount运行结果（只需要在ecs-hw-0001上执行）

bash run.sh wordcount 2

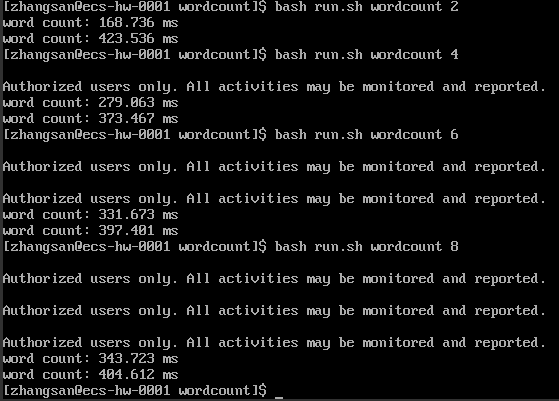
bash run.sh wordcount 4

bash run.sh wordcount 6

bash run.sh wordcount 8

2、4、6、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wordcount算法程序已经在集群中并行运行起来。其中第一行输出代表的是大文件统计耗时，第二行输出代表的是小文件统计耗时。

## 思考题及答案

实现WordCount算法中比较关键的三步是哪些？

参考答案：getWords 、countWords 、treeMerge。