Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра компьютерных систем и программных технологий

**Отчёт по лабораторной работе**

**Дисциплина**: Параллельные вычисления

**Тема**: Разработка программ с использованием pthread и MPI в языке С++

Выполнил студент гр. 13541/4 Степанов Д.С.

(подпись)

Руководитель Стручков И.В.

(подпись)

“ ” 2017 г.

Санкт - Петербург

2017

**Цель работы**

Научиться создавать программы с использованием многопоточных технологий. Познакомиться с работой библиотек pthread и OpenMP для языка C++. Проанализировать прирост производительности при использовании многопоточных библиотек.

**Выполнение работы**

Естественные языки могут быть охарактеризованы распределением частот встречаемости своих элементов, таких как слова, отдельные буквы или последовательности букв (N-граммы). Формально N-граммой называется строка из N символов, принадлежащих некоторому алфавиту, состоящему из конечного числа символов.

Вероятность появления n-граммы в тексте подсчитывается формулой:

F=n/L,

где n – число появлений целевой n-граммы, а L – число всех возможных n-грамм.

Технические характеристики:

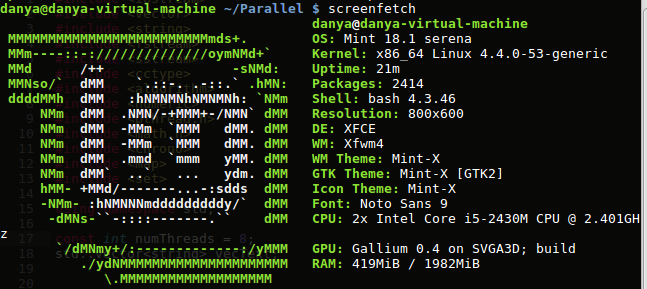


Рис.1. Технические характеристики

**Однопоточная программа**

Ниже приведен код однопоточной программы:

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <sstream>

#include <cctype>

#include <algorithm>

#include <numeric>

#include <pthread.h>

#include <math.h>

#include <chrono>

#include <map>

#include <set>

using namespace std;

std::vector<string> vecText;

vector<string> readFileByWords(string& fileName)

{

vector <string> vec;

ifstream file(fileName.c\_str());

string word;

while (file >> word){

vec.push\_back(word);

}

return vec;

}

std::string deleteZn(std::string& s){

string ret;

for (auto&& ch : s){

if (!iswpunct(ch)){

ret.push\_back(tolower(ch));

}

}

return ret;

}

typedef struct thread\_data {

int index;

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

} thread\_data;

map<string, int> calculateFreq(){

map<string, int> map;

for (int i = 0; i < vecText.size()-2; ++i){

std::string s = vecText[i] + " " + vecText[i+1] + " " + vecText[i+2];

auto&& it = map.find(s);

if(it != map.end()){

++it->second;

}

else{

map.insert(std::make\_pair(s, 0));

}

}

return map;

}

template<class InputIterT1, class InputIterT2, class OutputIterT, class Comparator, class Func>

OutputIterT merge\_apply(

InputIterT1 first1, InputIterT1 last1,

InputIterT2 first2, InputIterT2 last2,

OutputIterT result, Comparator comp, Func func) {

while (true)

{

if (first1 == last1) return std::copy(first2, last2, result);

if (first2 == last2) return std::copy(first1, last1, result);

if (comp(\*first1, \*first2) == 0 ) {

\*result = func(\*first1, \*first2);

++first1;

++first2;

} else if (comp(\*first1, \*first2) > 0) {

\*result = \*first2;

++first2;

} else {

\*result = \*first1;

++first1;

}

++result;

}

}

template<class T>

int compare\_first(T a, T b) {

return a.first.compare(b.first);

}

template<class T>

T sum\_pairs(T a, T b) {

if(a.second == b.second && a.second == 0)

return std::make\_pair(a.first, 1);

else

return std::make\_pair(a.first, a.second + b.second);

}

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

{

std::vector<float>times;

auto&& reps = 20;

for (int rep = 0; rep < reps; ++rep){

cout << "rep=" << rep << endl;

setlocale(LC\_ALL, "rus");

string fileName("War\_and\_piece.txt");

auto&& num = 2;

vecText = readFileByWords(fileName);

/\*filter ---\*/

char c = (int) -30;

char d = (int) -128;

char e = (int) -109;

std::string fi;

fi.push\_back(c);

fi.push\_back(d);

fi.push\_back(e);

vecText.erase(std::remove\_if(vecText.begin(), vecText.end(), [fi](string& word){

if (word.size() == 1 && !iswalpha((unsigned char)word[0]))

return true;

else if(word == fi)

return true;

else

return false; }), vecText.end());

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

vecText[i] = deleteZn(vecText[i]);

}

std::chrono::time\_point<std::chrono::system\_clock> start, end;

start = std::chrono::system\_clock::now();

auto map = calculateFreq();

int L=0;

cout<<"size="<<map.size()<<endl;

for(auto&& it:map){

if(it.second == 0) continue;

if(it.second != 0) L=L+it.second;

}

end = std::chrono::system\_clock::now();

int elapsed\_seconds = std::chrono::duration\_cast<std::chrono::milliseconds> (end-start).count();

float frequency = ((float)L / (float)map.size());

cout << "N=" << map.size() << " L= " << L << " freq= " << frequency << endl;

vecText.clear();

unsigned int workingTime = elapsed\_seconds;

times.push\_back(workingTime);

}

float m = std::accumulate(times.begin(), times.end(), 0) / reps;

float disp = 0;

for (auto&& num : times){

cout << num << " m= " << m << endl;

disp += std::pow((num - m), 2);

}

disp = disp / (reps - 1);

float sigma = std::sqrt(disp);

float t = 1.984;

float interHigh = m + t\*(sigma / (sqrt(reps)));

float interLow = m - t\*(sigma / (sqrt(reps)));

cout << "M=" << m << endl;

cout << "disp=" << disp << endl;

cout << "inter= [" << (long) interLow<<";"<<(long) interHigh <<"]"<< endl;

return 0;

}

Результаты:

N=215132 L= 15125 freq= 0.0703057

|  |  |
| --- | --- |
| Мат. ожидание | Доверительный интервал |
| 1086 | [1074;1097] |

**Многопоточная программа с использованием pthread**

Код:

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <sstream>

#include <cctype>

#include <algorithm>

#include <numeric>

#include <pthread.h>

#include <math.h>

#include <chrono>

#include <map>

#include <set>

using namespace std;

const int numThreads = 8;

std::vector<string> vecText;

vector<string> readFileByWords(string& fileName)

{

vector <string> vec;

ifstream file(fileName.c\_str());

string word;

while (file >> word){

vec.push\_back(word);

}

return vec;

}

std::string deleteZn(std::string& s){

string ret;

for (auto&& ch : s){

if (!iswpunct(ch)){

ret.push\_back(tolower(ch));

}

}

return ret;

}

typedef struct thread\_data {

int index;

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

} thread\_data;

void \*calculateFreq(void \*arg){

thread\_data \*tdata=(thread\_data \*)arg;

long threadid = tdata->index;

int n = 0;

int step = vecText.size() / numThreads;

auto tid = threadid;

auto from = tid\*step;

auto to = (tid + 1)\*step;

if (tid == numThreads - 1)

to = vecText.size();

long count = 0;

for (int i = from; i < to-2; ++i){

std::string s = vecText[i] + " " + vecText[i+1] + " " + vecText[i+2];

auto&& it = tdata->map.find(s);

if(it != tdata->map.end()){

++it->second;

++count;

}

else{

tdata->map.insert(std::make\_pair(s, 0));

}

}

pthread\_exit(0);

}

template<class InputIterT1, class InputIterT2, class OutputIterT, class Comparator, class Func>

OutputIterT merge\_apply(

InputIterT1 first1, InputIterT1 last1,

InputIterT2 first2, InputIterT2 last2,

OutputIterT result, Comparator comp, Func func) {

while (true)

{

if (first1 == last1) return std::copy(first2, last2, result);

if (first2 == last2) return std::copy(first1, last1, result);

if (comp(\*first1, \*first2) == 0 ) {

\*result = func(\*first1, \*first2);

++first1;

++first2;

} else if (comp(\*first1, \*first2) > 0) {

\*result = \*first2;

++first2;

} else {

\*result = \*first1;

++first1;

}

++result;

}

}

template<class T>

int compare\_first(T a, T b) {

return a.first.compare(b.first);

}

template<class T>

T sum\_pairs(T a, T b) {

if(a.second == b.second && a.second == 0)

return std::make\_pair(a.first, 1);

else

return std::make\_pair(a.first, a.second + b.second);

}

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

{

std::vector<float>times;

auto&& reps = 10;

for (int rep = 0; rep < reps; ++rep){

cout << "rep=" << rep << endl;

setlocale(LC\_ALL, "rus");

string fileName("War\_and\_piece.txt");

auto&& num = 2;

vecText = readFileByWords(fileName);

std::vector<string> vecGr;

/\*filter ---\*/

char c = (int) -30;

char d = (int) -128;

char e = (int) -109;

std::string fi;

fi.push\_back(c);

fi.push\_back(d);

fi.push\_back(e);

vecText.erase(std::remove\_if(vecText.begin(), vecText.end(), [fi](string& word){

if (word.size() == 1 && !iswalpha((unsigned char)word[0]))

return true;

else if(word == fi)

return true;

else

return false; }), vecText.end());

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

vecText[i] = deleteZn(vecText[i]);

}

pthread\_t threads[numThreads];

pthread\_attr\_t attr;

pthread\_attr\_init(&attr);

int size = vecText.size();

int step = size / numThreads;

std::vector<thread\_data> tdata(numThreads);

std::chrono::time\_point<std::chrono::system\_clock> start, end;

start = std::chrono::system\_clock::now();

for (long ind = 0; ind<numThreads; ++ind){

thread\_data td;

td.index = ind;

td.map = std::map<std::string, int>();

tdata[ind]=td;

pthread\_create(&threads[ind], &attr, calculateFreq, (void \*)&tdata[ind]);

}

int rc;

void\* status;

long t;

long n = 0;

for (t = 0; t<numThreads; t++) {

rc = pthread\_join(threads[t], &status);

}

end = std::chrono::system\_clock::now();

auto genMap = tdata[0].map;

auto cmap = std::map<std::string, int>();

for(int i = 1; i<numThreads; ++i){

merge\_apply(tdata[i].map.begin(), tdata[i].map.end(), genMap.begin(), genMap.end(), inserter(cmap, cmap.begin()),

compare\_first<pair<std::string, int> >, sum\_pairs<pair<std::string, int> >);

genMap = cmap;

cmap.clear();

}

int L=0;

for(auto&& it:genMap){

if(it.second == 0) continue;

if(it.second != 0) L=L+it.second;

}

int elapsed\_seconds = std::chrono::duration\_cast<std::chrono::milliseconds> (end-start).count();

float frequency = ((float)L / (float)genMap.size());

cout << "N=" << genMap.size() << " L= " << L << " freq= " << frequency << endl;

vecText.clear();

times.push\_back(elapsed\_seconds);

}

float m = std::accumulate(times.begin(), times.end(), 0) / reps;

float disp = 0;

for (auto&& num : times){

cout << num << " m= " << m << endl;

disp += std::pow((num - m), 2);

}

disp = disp / (reps - 1);

float sigma = std::sqrt(disp);

float t = 1.984;

float interHigh = m + t\*(sigma / (sqrt(reps)));

float interLow = m - t\*(sigma / (sqrt(reps)));

cout << "M=" << m << endl;

cout << "disp=" << disp << endl;

cout << "inter= [" << (long) interLow<<";"<<(long) interHigh <<"]"<< endl;

return 0;

}

|  |  |  |
| --- | --- | --- |
| Потоков | Мат. ожидание | Доверительный интервал |
| 1 | 1223 | [1186;1259] |
| 2 | 641 | [624;657] |
| 4 | 406 | [384;427] |
| 8 | 388 | [349;426] |

**Многопоточная программа с использованием MPI**

Исходный код приведен ниже:

#include <iostream>

#include <vector>

#include <string>

#include <fstream>

#include <sstream>

#include <cctype>

#include <algorithm>

#include <numeric>

#include <pthread.h>

#include <math.h>

#include <chrono>

#include <map>

#include <mpi.h>

#include <boost/archive/text\_oarchive.hpp>

#include <boost/archive/text\_iarchive.hpp>

#include <boost/serialization/list.hpp>

#include <boost/serialization/map.hpp>

using namespace std;

int numproc = 1;

std::vector<string> vecText;

template<class InputIterT1, class InputIterT2, class OutputIterT, class Comparator, class Func>

OutputIterT merge\_apply(

InputIterT1 first1, InputIterT1 last1,

InputIterT2 first2, InputIterT2 last2,

OutputIterT result, Comparator comp, Func func) {

while (true)

{

if (first1 == last1) return std::copy(first2, last2, result);

if (first2 == last2) return std::copy(first1, last1, result);

if (comp(\*first1, \*first2) == 0 ) {

\*result = func(\*first1, \*first2);

++first1;

++first2;

} else if (comp(\*first1, \*first2) > 0) {

\*result = \*first2;

++first2;

} else {

\*result = \*first1;

++first1;

}

++result;

}

}

template<class T>

int compare\_first(T a, T b) {

return a.first.compare(b.first);

}

template<class T>

T sum\_pairs(T a, T b) {

if(a.second == b.second && a.second == 0)

return std::make\_pair(a.first, 1);

else

return std::make\_pair(a.first, a.second + b.second);

}

vector<string> readFileByWords(string& fileName)

{

vector <string> vec;

ifstream file(fileName.c\_str());

string word;

while (file >> word){

vec.push\_back(word);

}

return vec;

}

std::string deleteZn(std::string& s){

string ret;

for (auto&& ch : s){

if (!iswpunct(ch)){

ret.push\_back(tolower(ch));

}

}

return ret;

}

typedef struct thread\_data {

int index;

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

} thread\_data;

map<std::string, int> calculateFreq(int id){

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

int n = 0;

int step = vecText.size() / numproc;

auto tid = id;

auto from = tid\*step;

auto to = (tid + 1)\*step;

if (tid == numproc - 1)

to = vecText.size();

long count = 0;

for (int i = from; i < to-2; ++i){

std::string s = vecText[i] + " " + vecText[i+1] + " " + vecText[i+2];

auto&& it = map.find(s);

if(it != map.end()){

++it->second;

++count;

}

else{

map.insert(std::make\_pair(s, 0));

}

}

return map;

}

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

{

std::vector<float>times;

auto&& reps = 1;

for (int rep = 0; rep < reps; ++rep){

cout << "rep=" << rep << endl;

setlocale(LC\_ALL, "rus");

string fileName("War\_and\_piece.txt");

auto&& num = 2;

vecText = readFileByWords(fileName);

std::vector<string> vecGr;

/\*filter ---\*/

char c = (int) -30;

char d = (int) -128;

char e = (int) -109;

std::string fi;

fi.push\_back(c);

fi.push\_back(d);

fi.push\_back(e);

vecText.erase(std::remove\_if(vecText.begin(), vecText.end(), [fi](string& word){

if (word.size() == 1 && !iswalpha((unsigned char)word[0]))

return true;

else if(word == fi)

return true;

else

return false; }), vecText.end());

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

vecText[i] = deleteZn(vecText[i]);

}

int rank;

MPI\_Init(&argc, &argv);

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

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

std::chrono::time\_point<std::chrono::system\_clock> start, end;

start = std::chrono::system\_clock::now();

auto&& map = calculateFreq(rank);

//end = std::chrono::system\_clock::now();

//int elapsed\_seconds = std::chrono::duration\_cast<std::chrono::milliseconds> (end-start).count();

//cout<<"elapsed "<<rank<<" "<<elapsed\_seconds<<endl;

std::stringstream ss;

MPI\_Status st;

if(rank != 0){

std::stringstream ss;

boost::archive::text\_oarchive oa(ss);

oa & map;

cout<<rank<<" send size="<<map.size()<<endl;

int size = ss.str().size();

MPI\_Send(&size, 1, MPI\_INT, 0, 0, MPI\_COMM\_WORLD);

MPI\_Send(ss.str().c\_str(), ss.str().size(), MPI\_CHAR, 0, 0, MPI\_COMM\_WORLD);

}

else {

auto genMap = map;

auto cmap = std::map<std::string, int>();

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

int size;

MPI\_Recv(&size, 1, MPI\_INT, MPI\_ANY\_SOURCE, 0, MPI\_COMM\_WORLD, &st);

const int buf\_size = size;

char buf[buf\_size];

MPI\_Recv(buf, buf\_size, MPI\_CHAR, st.MPI\_SOURCE, 0, MPI\_COMM\_WORLD, MPI\_STATUS\_IGNORE);

std::stringstream ss;

ss.str(buf);

boost::archive::text\_iarchive ia(ss);

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

ia >> recmap; // No size/range needed

cout<<"rec from "<<st.MPI\_SOURCE<<" size="<<recmap.size()<<endl;

merge\_apply(recmap.begin(), recmap.end(), genMap.begin(), genMap.end(), inserter(cmap, cmap.begin()),

compare\_first<pair<std::string, int> >, sum\_pairs<pair<std::string, int> >);

}

end = std::chrono::system\_clock::now();

int elapsed\_seconds = std::chrono::duration\_cast<std::chrono::milliseconds> (end-start).count();

int L = 0;

for(auto&& it:genMap){

if(it.second == 0) continue;

if(it.second != 0) L=L+it.second;

}

float frequency = ((float)L / (float)genMap.size());

cout << "N=" << genMap.size() << " L= " << L << " freq= " << frequency << endl;

vecText.clear();

unsigned int timeEnd = clock();

unsigned int workingTime = elapsed\_seconds;

cout<<"workingTime="<<workingTime<<endl;

}

}

MPI\_Finalize();

return 0;

}

|  |  |  |
| --- | --- | --- |
| Потоков | Мат. ожидание | Доверительный интервал |
| 1 | 1190 | [1175;1204] |
| 2 | 1137 | [1122;1151] |
| 4 | 1074 | [1058;1089] |
| 8 | 1453 | [1343;1562] |

Общая таблица:

Таблица 1. Общая таблица времени работы программ

|  |  |  |
| --- | --- | --- |
| Количество потоков | Мат. ожидание | Доверительный интервал |
| 1 | 1086 | [1074;1097] |
| pThread | | |
| 1 | 1223 | [1186;1259] |
| 2 | 641 | [624;657] |
| 4 | 406 | [384;427] |
| 8 | 388 | [349;426] |
| MPI | | |
| 1 | 1190 | [1175;1204] |
| 2 | 1137 | [1122;1151] |
| 4 | 1074 | [1058;1089] |
| 8 | 1453 | [1343;1562] |

**Вывод**

В ходе работы создано три программы на языке С++ для решения задачи поиска n-грамм в тексте. Предложенные решения позволяют, решать задачу с любым количеством слов в n-грамме (т.е. менять n). В ходе работы изучены и использованы библиотеки многопоточного программирования pThread и MPI.

Полученные решения протестированы на большом входном файле. Для одного набора входных данных проведен анализ вероятностных характеристик (табл.1). В результате экспериментов установлено, что наиболее эффективным оказалось использование библиотеки pThread. Использование библиотеки mpi не привело к положительным результатам, из-за накладных расходов при передаче результатов работы процесса.

Использование библиотеки MPI оказалось очень удобным благодаря встроенным функциям получения ранга процесса и синхронизации результатов. Из-за того что библиотека MPI не умеет автоматически пересылать STL контейнеры, была использована библиотека boost для сериализации словаря – результата работы каждого процесса.