Київський національний університет імені Тараса Шевченка

Кафедра теоретичної кібернетики факультету комп’ютерних наук та кібернетики

ЗВІТ

ПАРАЛЕЛЬНЕ ПРОГРАМУВАННЯ

Виконав студент групи ТК-3

Ігнашов Денис

Київ-2017

1. **Постановка задачі**

Створити перелік слів, які містяться у наданих книжках (приблизно 160-240 Кб кожна), застосовуючи технології паралельних обчислень; порівняти такі технології між собою.

1. **Отримані результати**

|  |  |  |  |
| --- | --- | --- | --- |
| **Кількість книжок** | **Послідовна програма** | **MPI**  **(n=4)** | **OpenMP**  **(n=4)** |
| **1** | **21** | **35** | **30** |
| **4** | **97** | **67** | **75** |
| **10** | **1813** | **551** | **986** |
| **20** | **2401** | **1144** | **2011** |

1. **Висновок**

Використання технологій паралельного програмування зменшило час виконання алгоритму у декілька разів(1.5-3.5 рази, залежно від конкретної технології та обсягу даних). Проте варто помітити, що при роботі в одному потоці, послідовна програма бере гору над обома паралельними.

Під час тестування використовувалась обчислювальна машина Intel Core i5-3230M CPU @ 2.60GHz. З великою імовірністю, використання більш потужного процесора, наприклад з більшою кількістю ядер, значно збільшило б приріст швидкості виконання.

Всі тести проводились до запуску основних процесів операційної системи, що дозволило заміряти достатньо точні показники. Також на момент тестування усі виводи у консоль були закоментовані.

OpenMP виявився значно простішим у реалізації, проте MPI виявився більш потужною технологією для вирішення даної задачі і на останніх тестах, де кількість книжок досягла 20, OpenMP почала працювати значно повільніше і навіть наблизилась до швидкості послідовно програми. В свою чергу, швидкість виконання програми з MPI фактично не змінилась і до останнього показувала гарні результати.

Лістинг послідовного коду:

#include <iostream>

#include <string>

#include <set>

#include <fstream>

#include <ctime>

using namespace std;

void printResults(int numOfWords, int numOfWordsWithoutRepetitions, string bookName) {

cout << "Nomber of words in " << bookName << " = " << numOfWords << endl;

cout << "Nomber of words without in repetitions" << bookName << " = " << numOfWordsWithoutRepetitions << endl;

}

set<string> split(string book)

{

string word = "";

unsigned count = 0;

set<string> words;

ifstream in(book);

while (!in.eof())

{

string segment;

in >> segment;

for (unsigned i = 0; i < segment.size(); i++)

{

if (isalpha(segment[i]))

word += segment[i];

else

break;

}

if (word.size() > 0 && word.size() < 30) {

words.insert(word);

count++;

}

word = "";

}

in.close();

printResults(count, words.size(), book);

return words;

}

int main()

{

set<string> dict;

set<string> books;

books.insert("texts/alice.txt");

books.insert("texts/constitution.txt");

books.insert("texts/tom.txt");

books.insert("texts/wonderland.txt");

books.insert("texts/ulysses.txt");

books.insert("texts/hugo.txt");

books.insert("texts/kjv.txt");

books.insert("texts/dracula.txt");

books.insert("texts/whittier.txt");

books.insert("texts/men.txt");

long st\_time = clock();

for (string book : books)

for (string word : split(book))

dict.insert(word);

/\*for (string i : dict)

cout << i << endl;\*/

cout << "Time: " <<clock() - st\_time << endl;

system("pause");

}

Лістинг коду з OpenMP:

#include <iostream>

#include <string>

#include <set>

#include <fstream>

#include <ctime>

#include <omp.h>

#include <vector>

using namespace std;

void printResults(int numOfWords, int numOfWordsWithoutRepetitions, string bookName) {

cout << "Nomber of words in " << bookName << " = " << numOfWords << endl;

cout << "Nomber of words without in repetitions" << bookName << " = " << numOfWordsWithoutRepetitions << endl;

}

set<string> split(string book)

{

string word = "";

unsigned count = 0;

set<string> words;

ifstream in(book);

while (!in.eof())

{

string segment;

in >> segment;

for (unsigned i = 0; i < segment.size(); i++)

{

if (isalpha(segment[i]))

word += segment[i];

else

break;

}

if (word.size() > 0 && word.size() < 30) {

words.insert(word);

count++;

}

word = "";

}

in.close();

printResults(count, words.size(), book);

return words;

}

int main()

{

set<string> dict;

vector<string> books;

books.push\_back("texts/alice.txt");

books.push\_back("texts/constitution.txt");

books.push\_back("texts/tom.txt");

books.push\_back("texts/wonderland.txt");

books.push\_back("texts/ulysses.txt");

books.push\_back("texts/hugo.txt");

books.push\_back("texts/kjv.txt");

books.push\_back("texts/dracula.txt");

books.push\_back("texts/whittier.txt");

books.push\_back("texts/men.txt");

books.push\_back("texts/alice.txt");

books.push\_back("texts/constitution.txt");

books.push\_back("texts/tom.txt");

books.push\_back("texts/wonderland.txt");

books.push\_back("texts/ulysses.txt");

books.push\_back("texts/hugo.txt");

books.push\_back("texts/kjv.txt");

books.push\_back("texts/dracula.txt");

books.push\_back("texts/whittier.txt");

books.push\_back("texts/men.txt");

double st\_time = omp\_get\_wtime();

#pragma omp parallel for num\_threads(8)

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

for (string word : split(books[i]))

#pragma omp critical

{

dict.insert(word);

}

}

/\*for (string i : dict)

cout << i << endl;\*/

cout << "Time: " << omp\_get\_wtime() - st\_time << endl;

system("pause");

}

Лістинг коду з MPI:

#include <iostream>

#include <string>

#include <set>

#include <fstream>

#include <ctime>

#include "mpi.h"

#include <vector>

using namespace std;

void printResults(int numOfWords, int numOfWordsWithoutRepetitions, string bookName) {

cout << "Nomber of words in " << bookName << " = " << numOfWords << endl;

cout << "Nomber of words without in repetitions" << bookName << " = " << numOfWordsWithoutRepetitions << endl;

}

set<string> split(string book)

{

string word = "";

unsigned count = 0;

set<string> words;

ifstream in(book);

while (!in.eof())

{

string segment;

in >> segment;

for (unsigned i = 0; i < segment.size(); i++)

{

if (isalpha(segment[i]))

word += segment[i];

else

break;

}

if (word.size() > 0 && word.size() < 30) {

words.insert(word);

count++;

}

word = "";

}

in.close();

printResults(count, words.size(), book);

return words;

}

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

{

int rank;

int size;

set<string> dict;

vector<string> books;

books.push\_back("texts/alice.txt");

books.push\_back("texts/constitution.txt");

books.push\_back("texts/tom.txt");

books.push\_back("texts/wonderland.txt");

books.push\_back("texts/ulysses.txt");

books.push\_back("texts/hugo.txt");

books.push\_back("texts/kjv.txt");

books.push\_back("texts/dracula.txt");

books.push\_back("texts/whittier.txt");

books.push\_back("texts/men.txt");

books.push\_back("texts/alice.txt");

books.push\_back("texts/constitution.txt");

books.push\_back("texts/tom.txt");

books.push\_back("texts/wonderland.txt");

books.push\_back("texts/ulysses.txt");

books.push\_back("texts/hugo.txt");

books.push\_back("texts/kjv.txt");

books.push\_back("texts/dracula.txt");

books.push\_back("texts/whittier.txt");

books.push\_back("texts/men.txt");

double st\_time = MPI\_Wtime();

MPI\_Init(&argc, &argv);

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

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

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

for (string word : split(books[i]))

dict.insert(word);

}

MPI\_Finalize();

/\*for (string i : dict)

cout << i << endl;\*/

cout << "Time: " << MPI\_Wtime() - st\_time << "ms" << endl;

system("pause");

}