МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**«Вятский государственный университет»**

**(ФГБОУ ВО «ВятГУ»)**

Факультет автоматики и вычислительной техники

Кафедра электронных вычислительных машин

**Параллельное программирование**

Реализация вычислительно сложного алгоритма с применением программного пакета PVM

Вариант 7

Выполнил студент группы ИВТ-31 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Кудяшев Я.Ю./

Проверил преподаватель\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_/Долженкова М.Л./

Киров 2022

1. Задание

Познакомиться с программным пакетом PVM, получить навыки реализации параллельных приложений с его использованием.

Этапы работы:

1. Изучить основные принципы работы PVM.
2. Реализовать параллельную версию алгоритма с помощью языка C++ и пакета PVM, используя возможности конструирования гетерогенной платформы с поддержкой передачи сообщений.
3. Показать корректность полученной реализации путем осуществления тестирования на построенном в ходе первой лабораторной работы наборе тестов.
4. Провести доказательную оценку эффективности PVM-реализации алгоритма, в том числе с использованием инструментов профилирования.
5. Метод распараллеливания алгоритма

В качестве областей участков для распараллеливания при помощи PVM были выбраны те же участки, что и при простом распараллеливании. Это было сделано для наиболее точного сравнения результатов тестов.

Из исследований алгоритма для перемножения полиномов с помощью быстрого преобразованья Фурье удалось выяснить, что время в большей степени зависит от количества входных векторов, нежели от размерности. Было принято решение переложить работу по умножению каждой пары векторов на потоки. Помимо этого, для более эффективного использования потоков, было принято решение провести дополнительные тесты, где в дополнение к предыдущему алгоритму распараллеливания было добавлено параллельное нахождение ДПФ для каждого вектора в паре. Результаты тестов с дополнительных тестов приведены в таблице 2.

1. Программная реализация

Листинг программной PVM-реализации алгоритма приведен в приложении А.

1. Тестирование

Тестирование проводилось на сети из 1 и 2-х ЭВМ. ЭВМ под управлением 64-разрядной OC Ubuntu, с 1 ГБ выделенной оперативной памяти. Виртуальной машине выделялись все доступные ядра процессора хост-системы. Для подключения виртуальных машин в одну локальную сеть использовался сетевой мост, хост-системы подключались к общей Wi-Fi-сети.

В ходе тестирования использовались ЭВМ:

* Процессор Intel Core i5-8250U с частотой 1.80 ГГц (8 логических или 4 физических ядра);
* Процессор Intel Core i3-7100 (8 логических или 4 физических ядра).

Для тестирования на 1 ЭВМ использовалась 1-я ЭВМ.

В предыдущих лабораторных работах сравнительно худший результат показал алгоритм, реализованный при помощи OpenMP, поэтому в таблице 1 приведено сравнении PVM именно с ним.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исходные данные (кол-во полиномов, размерность) | Последовательная реализация, мс | OpenMP-реализация, мс | PVM-реализация | Ускорение в сравнении с последовательной реализацией |
| 2, 100000 | 210 мс | 208 мс | 806 мс | 0,26 |
| 4, 100000 | 1183 мс | 590 мс | 1543 мс | 0,77 |
| 6, 100000 | 5485 мс | 1217 мс | 1975 мс | 2,78 |
| 8, 100000 | 24990 мс | 4308 мс | 5510 мс | 4,53 |
| 2, 1000000 | 1493 мс | 1175 мс | 2246 мс | 0,66 |
| 4, 1000000 | 10734 мс | 4101 мс | 6894 мс | 1,56 |
| 6, 1000000 | 50241 мс | 10171 мс | 12683 мс | 3,96 |
| 8, 1000000 | 259036 мс | 40150 мс | 44987 мс | 5,76 |
| 2, 10000000 | 26759 мс | 21103 мс | 22985 мс | 1,16 |
| 8, 10000 | 2640 мс | 502 мс | 2020 мс | 1,31 |
| Максимальное | 5,76 |
| Среднее | 2,28 |
| Минимальное | 0,26 |

Таблица 1 – Результаты тестирования

Исходя из результатов тестирования можно сказать, что в реализации данного алгоритма PVM не даёт ускорения. О причинах такого поведения сказано в выводе.

Предыдущее тестирование проводилось под руководством 1 ЭВМ. Для полноты представления были проведены дополнительные тестирования уже на 2-х ЭВМ. Результаты приведены в таблице 2.

Таблица 2 – Результаты тестирования для 1 и 2 ЭВМ

|  |  |  |  |
| --- | --- | --- | --- |
| Исходные данные (кол-во полиномов, размерность) | PVM с 1 ЭВМ | PVM с 2 ЭВМ | Ускорение |
| 2, 100000 | 806 мс | 1023 мс | 0,78 |
| 4, 100000 | 1543 мс | 1904 мс | 0,81 |
| 6, 100000 | 1975 мс | 2854 мс | 0,69 |
| 8, 100000 | 5510 мс | 5387 мс | 1,02 |
| 2, 1000000 | 2246 мс | 2690 мс | 0,83 |
| 4, 1000000 | 6894 мс | 6031 мс | 1,14 |
| 6, 1000000 | 12683 мс | 12853 мс | 0,98 |
| 8, 1000000 | 44987 мс | 42901 мс | 1,05 |
| 2, 10000000 | 22985 мс | 22707 мс | 1,01 |
| 8, 10000 | 2020 мс | 2790 мс | 0,72 |
| Максимальное | 1,14 |
| Среднее | 0,90 |
| Минимальное | 0,69 |

В большинстве случаев ускорение не наблюдается. Однако в некоторых случаях, при обработке большого количества векторов, имеется небольшое ускорение. О причинах такого поведения говорится в выводе.

1. Вывод

В ходе выполнения лабораторной работы был реализован алгоритм быстрого преобразования Фурье для параллельной обработки с применением программного пакета PVM. Из-за привязанности PVM к операционной системе Linux было принято решение установить на виртуальную машину дистрибутив GNU/Linux под названием Ubuntu.

После проделанных действий были проведены тесты, которые показали, что параллельная реализация алгоритма при помощи PVM практически не даёт никакого ускорения. Данная аномалия связана с несколькими причинами: отсутствие поддержки библиотеки PVM и её устарелость, запуск алгоритма на виртуальной машине и обмен данными между потоками при помощи Wi-Fi, а не сетевого кабеля.

Приложение А

(обязательное)

Листинг программы

**main\_thread.cpp**

#include <iostream> //Быстрое преобразование Фурье

#include <chrono> //Нахождение ДПФ

#include <vector>

#include <complex>

#include <fstream>

#include <thread>

#include <pvm3.h>

using namespace std;

typedef complex<double> base;

vector<int> information[30]; //data vector

vector<int> result(10000000); //result vector

int counter = 1;

int range = 1000000

int thread\_counter = 8;

std::thread threads[8]; //8 threads were created

void enter(string first, int number, int size, string path){

fill\_from\_file(path, number, size); //number, size

}

void fill\_from\_file(string path, int number\_of\_vectors, int size\_of\_vectors) { //reading data from the file

ifstream vectorr("C:\\Programming\\Parallel programming\\Lab 1\\" + path);

for (int i = 0; i < number\_of\_vectors; i++) {

information[i].resize(size\_of\_vectors);

}

for (int i = 0; i < number\_of\_vectors; i++) {

for (int j = 0; j < size\_of\_vectors; j++) {

vectorr >> information[i].at(j);

}

}

vectorr.close();

}

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

{

int ntask = 4; /\* число дочерних задач \*/

int info; /\* возвращаемый код завершения для функций PVM \*/

int mytid; /\* собственный идентификатор задачи \*/

int myparent; /\* идентификатор родительской задачи \*/

int child [ 50 ] ; /\* массив идентификаторов дочерних задач \*/

int i, mydata, buf, len, tag, tid; /\* остальные переменные \*/

enter("The third", 8, 100000, "int\_0-100 8\_100000.txt");

cout << "8 vectors of size 100000";

unsigned int start\_time = clock();

mytid = pvm\_mytid(); /\* определить собственный идентификатор \*/

info = pvm\_spawn ("slaves", information, PvmTaskDefault, NULL, 4, child);

ntask = info; /\* запись в ntask числа успешно запущенных задач \*/

for (int rank = 0; rank < ntask; ++rank) {

pvm\_recv(ntask, -1);

pvm\_upkint(information, range, 1);

}

info = pvm\_spawn ("slaves", information, PvmTaskDefault, NULL, 2, child);

ntask = info; /\* запись в ntask числа успешно запущенных задач \*/

for (int rank = 0; rank < ntask; ++rank) {

pvm\_recv(ntask, -1);

pvm\_upkint(information, range, 1);

}

pvm\_recv(1, -1);

pvm\_upkint(information, range, 1);

pvm\_exit();

unsigned int end\_time = clock();

unsigned int search\_time = end\_time - start\_time;

std::cout << search\_time << " mc\n";

cout << '\n';

cout << "8 vectors of size 1000000";

}

**slaves.cpp**

#include <iostream>

#include <chrono>

#include <vector>

#include <complex>

#include <fstream>

#include <thread>

#include <pvm3.h>

using namespace std;

typedef complex<double> base;

int range = 100000

int rev(int num, int lg\_n) { //begining of good realisation

int res = 0;

for (int i = 0; i < lg\_n; ++i)

if (num & (1 << i))

res |= 1 << (lg\_n - 1 - i);

return res;

}

void good\_realisation(vector<base>& a, bool invert) {

int n = (int)a.size();

int lg\_n = 0;

while ((1 << lg\_n) < n) ++lg\_n;

for (int i = 0; i < n; ++i)

if (i < rev(i, lg\_n))

swap(a[i], a[rev(i, lg\_n)]);

for (int len = 2; len <= n; len <<= 1) {

double ang = 2 \* 3.14 / len \* (invert ? -1 : 1);

base wlen(cos(ang), sin(ang));

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

base w(1);

for (int j = 0; j < len / 2; ++j) {

base u = a[i + j], v = a[i + j + len / 2] \* w;

a[i + j] = u + v;

a[i + j + len / 2] = u - v;

w \*= wlen;

}

}

}

if (invert)

for (int i = 0; i < n; ++i)

a[i] /= n;

}

void good\_multiplication(const vector<int>& a, const vector<int>& b, vector<int>& res, int number) { //multiplication of two vectors

vector<base> fa(a.begin(), a.end()), fb(b.begin(), b.end());

int n = 1;

while (n < max(a.size(), b.size())) n <<= 1;

n <<= 1;

fa.resize(n), fb.resize(n);

good\_realisation(fa, false);

good\_realisation(fb, false);

for (int i = 0; i < n; ++i)

fa[i] \*= fb[i];

good\_realisation(fa, true);

res.resize(n);

for (int i = 0; i < n; ++i)

res[i] = int(fa[i].real() + 0.5);

int buffer[range] = {time, result};

pvm\_initsend(PvmDataRaw);

pvm\_pkint(buffer, range, 1);

pvm\_send(pvm\_parent(), 1);

}

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

int wr = pvm\_mytid();

int cnt = get\_tasks\_count(argc, argv);

good\_multiplication(wr, cnt);

pvm\_exit();

}