МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
«Вятский государственный университет»
Факультет автоматики и вычислительной техники
Кафедра электронных вычислительных машин

Лабораторная работа № 5 по курсу «Параллельное программирование»

Выполнил студент группы ИВТ-31 _____/Седов М. Д./
Проверил доцент кафедры ЭВМ _____/Долженкова М. Л./

Киров 2020

1. Задание

Знакомство с программным пакетом PVM, получение навыков реализации параллельных приложений с его использованием.

1. Изучить основные принципы работы с пакетом PVM.

2. Реализовать параллельную версию алгоритм 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.

2. Словесное описание процесса выделения распараллеливаемых фрагментов

В реализованном алгоритме есть лишь одна процедура, распараллеливание которой может принести выигрыш в скорости нахождения решения поставленной задачи. Данный участок кода отвечает за порождение новых состояний игрового поля за счет сдвига пустой фишки во всех возможных направлениях.

Таким образом, основной процесс отвечает за взаимодействие с пользователем через консоль, а непосредственно решение задачи выполняется несколькими процессами. Количество порождаемых состояний может варьироваться от 2 до 4 включительно, в зависимости от местоположения пустой фишки.

Целесообразным способом распараллеливания является одновременное выполнение сдвига фишка во всех возможных направлениях и запись новых порожденных состояний в дерево.

Каждый процесс порождает новое состояние из изначального путем сдвига пустой фишки в одну из сторон, а затем в каждом процессе происходит поиск решения задачи на основе порожденного состояния.

Как только в одном из процессов было найдено решение, об этом информируются другие процессы и поиск решения завершается.

3. Схема взаимодействия процессов

На рисунке 1 представлена схема, иллюстрирующая взаимодействие процессов.

Рисунок 1 – Схема взаимодействия процеесов

4. Тестирование

Тестирование проводилось на сети из 2 ЭВМ. Для упрощения процесса развертывания вычислительной сети тестируемая программа запускалась на виртуальных машинах с 64 разрядной ОС Ubuntu версии 18.04 с 2 Гб ОЗУ. Для подключения машин в одну сеть использовался сетевой мост, хост системы подключались к общей WiFi — сети.

- Процессор Intel Core i5 6200u с частотой 2.3ГГц, 2 ядра
- Процессор Intel Core i5 6200u с частотой 2.3ГГц, 2 ядра

Количество строк и столбцов каждой матрицы пятнашек и результаты тестирования приведены в таблице 1.

Таблица 1 – Результаты тестирования

Строки и столбцы матрицы	Линейная реализация, с	Параллельная реализация, с	OpenMP,с	MPI, с	PVM, с
2x3	0,0000734	0,00006502	0,00004302	0,0000420	0,0000408
3x3	0,0011579	0,00015598	0,00017598	0,0001730	0,0001701
3x4	0,0126833	0,0036612	0,0020612	0,0020342	0,0019784
4x4	0,112261	0,008361	0,008197	0,0079215	0,006874
4x5	2,50988	0,216	0,204	0,1723	0,15097
5x5	61,3944	4,82699	4,5	4,254	3,975

5 Вывод

В ходе выполнения лабораторной работы были изучен программный пакет PVM, принципы создания многопроцессорных приложений для запуска в вычислительной сети. Получены навыки в построении вычислительных сетей. Разработан параллельный алгоритм поиска разрешающей последовательности. Данная многопроцессорная реализация алгоритма показала наилучшее время выполнение из всех рассмотренных ранее.

Приложение А
(обязательное)
Листинг программной реализации

Мастер

master.cpp

```
#include <iostream>
#include <cmath>
#include <vector>
#include <chrono>
#include "pvm3.h"

class Map
{
public:
	int* map;
	int lines;
	int cols;
public:
	Map(int lines, int cols) {
		this->map = new int[lines * cols];
		this->lines = lines;
		this->cols = cols;
	}

	int getLines() {
		return this->lines;
	}

	int getCols() {
		return this->cols;
	}

	int find(int n) {
		for (int i = 0; i < lines * cols; ++i) {
			if (map[i] == n) return i;
		}
		return  -1;
	}

	Map* shift(int angle) {
		if ((angle < 0) || (angle > 3)) return this;

		int len = lines * cols;
		Map * mapc = new Map(lines, cols);

		int ind = 0;
```

```
                for (int i = 0; i < len; ++i) {
                        mapc->map[i] = map[i];
                        if (map[i] == 0) ind = i;
                }

                switch (angle) {
                        //Вверх
                case 0: {
                        mapc->map[ind] = mapc-
>map[ind - cols];
                        mapc->map[ind - cols] = 0;
                        break;
                }
                                //Вправо
                case 1: {
                        mapc->map[ind] = mapc-
>map[ind + 1];
                        mapc->map[ind + 1] = 0;
                        break;
                }
                                //Вниз
                case 2: {
                        mapc->map[ind] = mapc-
>map[ind + cols];
                        mapc->map[ind + cols] = 0;
                        break;
                }
                                //Влево
                case 3: {
                        mapc->map[ind] = mapc-
>map[ind - 1];
                        mapc->map[ind - 1] = 0;
                        break;
                }
                }
                return mapc;
        }
};


int costFunc(Map* map);
unsigned int calcHash(Map* map);

class State
{
public:
        int cost;
        State* parent;
        Map* map;
        int hash;
```

```
public:
	State(Map* map, State* parent) {
		this->cost = costFunc(map);
		this->parent = parent;
		this->map = map;
		this->hash = calcHash(map);
	}

	State() {}

	const State* copy() const {
		return this;
	}

	Map* getMap() const {
		return this->map;
	}

	int getHash() const {
		return this->hash;
	}

	int getCost() const {
		return this->cost;
	}

	State* getParent() {
		return this->parent;
	}
};

int costFunc(Map* map) {
	int len = map->lines * map->cols;
	int sum = 0;
	for (int i = 0; i < len; ++i)
	{
		if (map->map[i] == 0) continue;
		int dx = abs((i % map->cols) - ((map-
>map[i] - 1) % map->cols));
		int dy = abs((i / map->cols) - ((map-
>map[i] - 1) / map->cols));
		sum += dx + dy;
	}

	return sum;
}


unsigned int calcHash(Map * map) {
	int len = map->lines * map->cols;
```

```
	unsigned int h = 0;
	for (int i = 0; i < len; ++i) {
		h += std::hash<int>{}(map->map[i] * (1
<< i));
	}

	return h;
}


struct Node {
	Node* left;
	Node* right;
	State* elem;
};

class BinTree
{
private:
	Node* node;
public:
	int len;

	BinTree(State* s) {
		this->node = new Node();
		this->node->elem = s;
		len = 1;
	}

	BinTree() {
		this->node = new Node();
		len = 0;
	}

	~BinTree() {
	}

	int getLen() {
		return len;
	}

	void add(State* s) {
		Node* n = this->node;

		len += 1;

		if (n->elem == NULL) {
			n->elem = s;
			return;
		}
```

```
                do
                {
                        if (s->getCost() >= n->elem-
>getCost()) {
                                if (n->right == NULL) {
                                        n->right = new
Node();
                                        n->right->elem =
s;
                                        break;
                                }
                                else {
                                        n = n->right;
                                }
                        }
                        else {
                                if (n->left == NULL) {
                                        n->left = new
Node();
                                        n->left->elem = s;
                                        break;
                                }
                                else {
                                        n = n->left;
                                }
                        }

                } while (true);
        }

        void del(State* s) {
                Node* n = this->node;
                Node* p = NULL;

                while (n->elem != s)
                {
                        p = n;
                        if (s->getCost() > n->elem-
>getCost()) {
                                n = n->right;
                        }
                        else {
                                n = n->left;
                        }

                        if (n == NULL) return;
                }

                len -= 1;
```

```
		if ((n->left == NULL) && (n->right ==
NULL)) {
			if (p == NULL) {
				n->elem = NULL;
				return;
			}
			if (p->left == n) p->left = NULL;
			else p->right = NULL;
			delete n;
			return;
		}

		if (n->left == NULL) {
			if (p == NULL) {
				this->node = n->right;
				delete n;
				return;
			}
			if (p->left == n) p->left = n-
>right;
			else p->right = n->right;
			delete n;
			return;
		}

		if (n->right == NULL) {
			if (p == NULL) {
				this->node = n->left;
				delete n;
				return;
			}
			if (p->left == n) p->left = n->left;
			else p->right = n->left;
			delete n;
			return;
		}

		if (n->right->left == NULL) {
			n->elem = n->right->elem;
			n->right = n->right->right;
			return;
		}
		else {
			Node* k = n->right;

			while (k->left->left != NULL) {
				k = k->left;
			}

			n->elem = k->left->elem;
```

```
                        k->left = k->left->right;
                }

        }
        //Компаратор при коллизии хешей
        bool cmp(State * a, State * b) {
                Map* am = a->getMap();
                Map* bm = b->getMap();
                int len = am->getCols() * am-
>getLines();
                for (int i = 0; i < len; i++) {
                        if (am->map[i] != bm->map[i])
return false;
                }
                return true;

        }
        //Поиск элемента в дереве
        State* find(State * s) {
                Node* n = this->node;

                if (n->elem == NULL) return NULL;

                while ((n->elem->getHash() != s-
>getHash()) || (!cmp(n->elem, s))) {
                        if (s->getCost() >= n->elem-
>getCost()) {
                                n = n->right;
                        }
                        else {
                                n = n->left;
                        }

                        if (n == NULL) return NULL;
                }
                return n->elem;
        }
        //Поиск минимального элемента дерева
        State* min() {
                Node* n = this->node;

                while (n->left != NULL) n = n->left;

                return n->elem;
        }
};

using namespace std;

using namespace std;
```

```
bool check(Map*);
void printMap(Map*);

//Генератор игрового поля
Map* generateMap(int lines, int cols, int n_of_sh) {
	int len = lines * cols;
	Map* map = new Map(lines, cols);

	for (int i = 0; i < len; ++i)
	{
		map->map[i] = i + 1;
	}
	map->map[len - 1] = 0;
	int i = 0;
	int shift_pos;
	srand(time(0));
	while (i <= len*n_of_sh) {
		//Находим пустую клетку (3,5 > 3x3,
20 <= 3x3)
		int zero = map->find(0);
		shift_pos = rand() % 4;
		switch (shift_pos) {
		case 0:
			//Если свехру есть квадрат
			if (zero / map->getCols() != 0) {
				map = map-
>shift(shift_pos);
				i++;
			}
			continue;
		case 1:
			//Если справа есть квадрат
			if (zero % map->getCols() !=
map->getCols() - 1) {
				map = map-
>shift(shift_pos);
				i++;
			}
			continue;
		case 2:
			//Если снизу есть квадрат
			if (zero / map->getCols() !=
map->getLines() - 1) {
				map = map-
>shift(shift_pos);
				i++;
			}
			continue;
		case 3:
```

```
                        //Если слева есть квадрат
                        if (zero % map->getCols() != 0)
{
                                map = map-
>shift(shift_pos);
                                i++;
                        }
                        continue;
                }
        }
        return map;
}

// Функция для вывода матрицы на экран
void printMap(Map* map) {
        cout << endl;
        for (int i = 0; i < map->lines; ++i) {
                for (int j = 0; j < map->cols; ++j) {
                        cout << map->map[i * map-
>cols + j] << '\t';
                }
                cout << endl;
        }
        cout << endl;
}

vector<State*> resultP2;

vector<State*> thread_func2(Map* map, State* min,
BinTree* close, BinTree* open, int flag_active_arr[4],
int rank) {
        vector<State*> lol;

        return lol;
}

vector<State*> aPar2(Map* map, int rank) {
        return resultP2;

}


   /****  Секция, использовавшаяся для отладки
примера  ****/

#ifdef EXDEBUG

#define DBG(x) x
```

```
   /* отсюда и до #endif : поддержка отладочных
выводов в файл */

#include <stdarg.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <time.h>

#define T(x) log_printf(timestr()), x

char *timestr(void)    /* возвращает строку с
текущим */
{                      /* временем в формате ЧЧ:ММ:СС */
  time_t t = time(0);
  struct tm *tt = localtime(&t);
  static char buf[32];
  sprintf (buf, "%02d:%02d:%02d    ", tt->tm_hour, tt-
>tm_min, tt->tm_sec);
  return buf;
}

void log_printf (char *format, ...)
{
  char s[1024], fn[32];
  int fd;
  va_list ap;
  va_start (ap, format);
  sprintf (fn, "pvmll.%08X", pvm_mytid());
  fd = open (fn, O_WRONLY|O_APPEND|O_CREAT,
0644);
  write (fd, s, vsprintf (s, format, ap));
  close (fd);
  va_end (ap);
}

#else  /* !EXDEBUG */

#define DBG(x)

#endif /* !EXDEBUG */

#define  SIZETAG 5       /* идентификатор для
других сообщений */
#define MAPTAG 6
#define CHILDTAG 8
#define FINALTAG 7
#define MY_GROUP "MY_GROUP"

int main(int argc, char** argv)
```

```
{
	int rank, size;
	int lines = atoi(argv[1]);
	int cols = atoi(argv[2]);
	int TESTS = atoi(argv[3]);
	int n_of_sh = atoi(argv[4]);
	Map* map;
	vector<State*> ans;
	long double tParNew = 0;
	int  child[4];  /* массив идентификаторов
дочерних задач */

	int mytid = pvm_mytid();   /* определить
собственный идентификатор задачи */
	//int mygid = pvm_joingroup (MY_GROUP);

	int numt = pvm_spawn("Slave", argv,
PvmTaskDefault, NULL, 4, child);

	int bufid = pvm_initsend(PvmDataDefault);
	int info = pvm_pkint(child, 4, 1);
	info = pvm_mcast(child, 4, CHILDTAG);

	int i = 0;
	for (i; i < TESTS; i++) {
		srand(i);
		map = generateMap(lines, cols,
n_of_sh);

		bufid = pvm_initsend(PvmDataDefault);
		info = pvm_pkint(map->map, lines*cols,
1);
		info = pvm_mcast(child, 4, MAPTAG);

		//Спавн, упаковать данные и
отправить слейвам

	chrono::high_resolution_clock::time_point t11
= chrono::high_resolution_clock::now();
		pvm_recv(     /* получить от
родительской задачи значения m и blksize. */
			-1,  /* от кого : идентификатор
задачи, которая */

			/* в группе MY_GROUP имеет
номер 0. */
			FINALTAG		/*
идентификатор ожидаемого сообщения */
		);
		//pvm_upkint(map->map, lines * cols,
1);	/* распаковать принятые значения m */
```

```
	chrono::high_resolution_clock::time_point t22
= chrono::high_resolution_clock::now();

		long double duration2 =
chrono::duration_cast<chrono::milliseconds>(t22 -
t11).count();
		//cout << "\n" << "Time of NEW
PARALLEL = " << duration2;
		printf ("%Lf--/n",duration2);
		tParNew += duration2;
	}

	//cout << "\n" << "---------------" << endl;
	//cout << "Average time NEW PARALLEL = "
<< (tParNew / (double)TESTS) << endl;
	printf ("Average time NEW PARALLEL =
%Lf------/n",(tParNew / (double)TESTS));
	//cout << "---------------" << endl;
	//system("pause");
	pvm_exit();	/* отключиться от PVM и
завершить работу дочерней задачи */

	return 0;
}
```

Slave.cpp

```
#define  SIZETAG 5
#define MAPTAG 6
#define CHILDTAG 8

int main(int argc, char** argv)
{
	int rank, size;
	int lines = atoi(argv[2]);
	int cols = atoi(argv[3]);
	int TESTS = atoi(argv[4]);

	int side = -1;
	Map* map;
	vector<State*> ans;
	long double tParNew = 0;
	int  child[4];

	int mytid = pvm_mytid();
	int myparent = pvm_parent();

	pvm_recv(
		myparent,
```

```
                CHILDTAG
        );
        pvm_upkint(child, 4, 1);

        map = new Map(lines, cols);

        if (mytid == child[0])
                side = 0;
        else
        if (mytid == child[1])
                side = 1;
        else
        if (mytid == child[2])
                side = 2;
        else
        if (mytid == child[3])
                side = 3;

        for (int i = 0; i < TESTS; i++) {
                pvm_recv(
                        myparent,


                        MAPTAG
                );
                pvm_upkint(map->map, lines * cols, 1);
                ans = aPar2(map, side, child, myparent);
        }
        pvm_exit();
        return 0;
}
```