Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет) (МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

НАПРАВЛЕНИЕ ПОДГОТОВКИ «09.03.04 Программная инженерия»

ОТЧЕТ

по практикуму №1 по обработке графов знаний на вычислительной платформе «Тераграф»

Название: Обработка графов знаний на вычислительной платформе «Тераграф»

Дисциплина: Архитектура ЭВМ

Студент ИУ7и-54Б _____ Шавиш тарек

Группа Подпись, дата И. О. Фамилия

Преподаватель _____ А. Ю. Попов

Подпись, дата И. О. Фамилия

Москва, 2024 г.

Цели работы

В ходе практикума ознакомиться с архитектурой и принципами работы вычислительного комплекса Терапраф, выполнить практические задания по программированию гетерогенных ядер обработки графов, ознакомиться с библиотеками для обработки и визуализации графов. Доступ к вычислительному комплексу осуществляется с использование облачной платформы Терапраф Cloud, обеспечивающей одновременный доступ многих пользователей к гетерогенным ядрам обработки, входящим в состав микропроцессора Леонард Эйлер.

На основе изложенных сведений необходимо разработать распределенное приложение обработки и визуализации графов, функционирующее в системе Терапраф.

Практикум состоит из трех этапов:

- Исследование принципов функционирования вычислительного комплекса Терапраф

- Практикум по программированию гетерогенного вычислительного узла

- Командный практикум по генерации музыкальных композиций на основе графов знаний

Практакум №1. Разработка и отладка программ в вычислительном комплексе Тераграф

Практикум посвящен освоению принципов работы вычислительного комплекса Тераграф и получению практических навыков решения задач обработки множеств на основе гетерогенной вычислительной структуры. В ходе практикума необходимо ознакомиться с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра sw_kernel.

Пример взаимодействия устройств: система определения ролей.

Рассмотрим следующие примеры кода подсистемы и программного ядра, которые мы будем использовать в практикуме. Система определения ролей пользователя сохраняет в памяти микропроцессора DISC тестовый набор ролей пользователей (1K пользователей с 1K ролями каждого). Далее система отвечает на запросы вида: какие роли пользователя были использованы начиная с момента времени time. Рассматриваемый пример выполняет следующие действия:

• Хост подсистема инициализирует gpc программным ядром sw_kernel.rawbinary.

```
gpc64_inst = new gpc();
log<<"Открывается доступ к "<<gpc64_inst->gpc_dev_path<<endl;
if (gpc64_inst->load_swk(argv[1])==0) {
		log<<"Программное ядро загружено из файла "<<argv[1]<<endl;
}
else {
		log<<"Ошибка загрузки sw_kernel файла << argv[1]"<<endl;
		return -1;
}
```

• Если программное ядро успешно загружено, хост подсистема запускает в gpc обработчик update, выполняющий прием и запись ключей и значений в SPE (ins_async). Код обработчика, функцуионирующего в sw_kernel представлен ниже:

```
void update() {
	while(1){
		users::key key=users::key::from_int(mq_receive());
		if (key==-1ull) break;
		users::val val=users::val::from_int(mq_receive());
		USERS.ins_async(key,val); //Вставка в таблицу с типизацией uint64_t
```

```
    }
}
```

• Далее хост-подсистема инициализирует поток сообщений к программному ядру. Для этого могут быть использованы два способа:

1. Последовательная пересылка ключей и значений unsigned long long короткими сообщениями.

```
for (uint32_t user=0;user<TEST_USER_COUNT;user++) {
   for (uint32_t idx=0;idx<TEST_ROLE_COUNT;idx++,offs+=2) {
      gpc64_inst->mq_send(users::key{.idx=idx,.user=user}); //запись о роли #idx
      gpc64_inst->mq_send(users::val{.role=idx,.time=time_t(0)}); //роль и время доступа
   }
}
```

2. Заполнение буфера данных и передача его драйверу (блочная передача). Данный способ обеспечивает большую пропускную способность передачи, так как реализуется через механизм прямого доступа к памяти. Передача данных из буфера выполняется в асинхронном режиме (процесс запускается по команде mq_send). Для ожидания момента завершения передачи метод mq_send возвращает указатель на поток передачи. Далее, если требуется ожидание завершения процесса передачи, необходимо использовать синхронизирующую команду join (send_buf_th->join()). Пример кода блочной передачи приведен ниже:

```
unsigned long long *buf = (unsigned long long*)malloc(sizeof(unsigned long long)*TEST_USER_COUNT*TEST_ROLE_COUNT*2);
for (uint32_t user=0,offs=0;user<TEST_USER_COUNT;user++) {
   for (uint32_t idx=0;idx<TEST_ROLE_COUNT;idx++,offs+=2) {
      buf[offs]=users::key{.idx=idx,.user=user};
      buf[offs+1]=users::val{.role=idx,.time=time_t(idx*3600)};
   }
}
auto send_buf_th = gpc64_inst->mq_send(sizeof(unsigned long long)*TEST_USER_COUNT*TEST_ROLE_COUNT*2,(char*)buf);
send_buf_th->join();
free(buf);
```

• По завершению передачи посылается терминальный символ (0xffffffffffffffff):

```
//Терминальный символ
gpc64_inst->mq_send(-1ull);
```

• В ответ на терминальный символ sw_kernel завершает обработчик update, и код хост-подсистемы запускает обработчик запросов поиска select.

• Система готова к приему запросов пользователя. Формат таблицы, представленной в SPE микропроцессоре следующий common.sh

```
//Запись для формирования ключей (* - наиболее значимые биты поля)
STRUCT(key)
{
   uint32_t   idx              :idx_bits;  //Поле 0:
   uint32_t   user   :32;                  //Поле 1*
};
STRUCT(val)
{
   uint32_t   role          :32;                    //Поле 0:
   time_t                   time   :32;             //Поле 1*
};
```

Поле ключа состоит из полей: user (поле идентификатора пользователя, старшая часть ключа) и idx (поле индекса записи о роли пользователя, младшая часть ключа). Поле значения состоит из полей: role (поле идентификатора роли) и time (поле времени последнего доступа).

• Запрос состоит в выборе тех ролей пользователя из таблицы users, которые были использованы позднее момента времени time, заданного в запросе. Например:

```
select role from users where user=5 and time>100000;
```

• Программный код хост-системы использует регулярные выражения (regex) для выделения полей в запросе select.

• Поля запроса user и time передаются в sw_kernel:

```
gpc64_inst->mq_send(stoi(match_query1[4])); //пользователь
gpc64_inst->mq_send(stoi(match_query1[6])); //время доступа
```

• Микропроцессор DISC выбирает из ассоциативной памяти все роли пользователя user и определяет те из них, которые соответствуют условию запроса (например, time>100000). Найденные роли передаются в хост-подсистему.

• В итоге, хост подсистема выдает сообщение о результатах поиска в поток cout.

Индивидуальное задание

Вариант 25

Ассоциативная память. Сформировать в хост-подсистеме и передать в SPE 256 случайных ключей и значений (по 64 бит). Выполнить поиск случайного значения ключа. Если результат найден, выдать его на консоль. Если результат не найден, то записать искомый ключ и случайное значение в SPE. Выполнить тестирование работы SPE, сравнив результат с ожидаемым.

Для выполнения данного задания я использовал 2 демо проект, который демонстрирует принципы взаимодействия устройств в системе и примеры хранения и обработки множеств в микропроцессоре Lnh64.

```
// #include "common_struct.h"
// #include "compose_keys.hxx"
#include "gpc_handlers.h"
#include "gpc_io_swk.h"
// #include "iterators.h"
#include "lnh64.hxx"
// #include <ctime>
// #include <iostream>
#include <stdlib.h>

#define __fast_recall__

extern lnh lnh_core;
volatile unsigned int event_source;

/*
Микроархитектура lnh64 допускает обращение к одной из семи независимых
структур (1..7). Структура с инексом 0 не используется для хранения и
зарезервирована.
То есть в этой микроархитектуре уже где то там на глубине созданы
какие то 7 структур, которые мы тупо используем, и именно что по номеру
*/
// Определяем структуру данных
// (захотелось использовать первую, мог любую от 1 до 7)
#define A 1 // Структура A
/*
ключом будет выступать беззнаковое целое число типа uint64_t
значением будет выступать беззнаковое целое число типа uint64_t
Это не композитные ключи -> доп. структур для них не надо
*/

int main(void)
{
  ////////////////////////////////////////////////////////
  //                  Main Event Loop
  ////////////////////////////////////////////////////////
  // Leonhard driver structure should be initialised
    lnh_init();
    for (;;)
    {
    // Wait for event
        event_source = wait_event();
        switch (event_source)
        {
            ////////////////////////////////////////////
            //  Measure GPN operation frequency
            ////////////////////////////////////////////
            case __event__(insert): insert(); break;
            case __event__(search): search(); break;
            case __event__(printA): printA(); break;
        }
        set_gpc_state(READY);
    }
}

//-------------------------------------------------------------
//      Вставка ключа и значения в структуру
//-------------------------------------------------------------

void insert()
{
    while (1)
```

```
    {
        uint64_t key = mq_receive(); // получили ключ от хоста
        if (key == -1ull)
          break;
        uint64_t val = mq_receive(); // получаем значение от хоста

        /*
         lnh_ins_sync(str,key,value)     Вставка ключа key и значения value в
         структуру str.
        */
        lnh_ins_sync(A, key, val);
    }
}

//-------------------------------------------------------------
//      Поиск значения по случайному ключу
//-------------------------------------------------------------

void search()
{
    // получили случайный ключ
    uint64_t key = mq_receive();
    //Поиск по ключу
          if (lnh_search(A, key)) // если значение по ключу найдено
    {
        //Отправка ответа
        mq_send(lnh_core.result.value);
    }
    else // если значение по ключу не найдено
    {
        //Отправка ответа
        mq_send(-1);

        uint64_t val = rand();

        if (lnh_ins_sync(A, key, val))
            // отправка сообщения об успешном добавлении в структуру
            mq_send(0);
        else
            mq_send(-1);
    }

}

void printA()
{
    // получили количество записей в структуре A
    uint32_t count_records = lnh_get_num(A);

    mq_send(count_records);

    for (int i = 0; i < count_records; ++i)
    {
        uint64_t key = mq_receive();

        lnh_search(A, key);

        mq_send(lnh_core.result.value);
    }

}
```

Листинг 1. Измененный код sw-kernel

```
#include "host_main.h"
#include <ctime>
#include <fstream>
#include <iostream>
#include <iterator>
#include <regex>
#include <sstream>
#include <string>
#include <vector>

using namespace std;

/*
Вариант 14

Ассоциативная память. Сформировать в хост-подсистеме и передать в SPE 256
случайных ключей и значений (по 64 бит). Выполнить поиск случайного значения
ключа. Если результат найден, выдать его на консоль. Если результат не
найден, то записать искомый ключ и случайное значение в SPE. Выполнить
тестирование работы SPE, сравнив результат с ожидаемым.
```

```
*/

// согласно заданию, нужно 256 записей
#define COUNT_RECORDS 256

void printStructure(gpc *gpc64_inst, vector<uint64_t> &vector_keys)
{
    // обработчик для вывода записей структуры A
    gpc64_inst->start(__event__(printA));

    auto count_records = gpc64_inst->mq_receive();

    cout << "\nЗаписей в структуре  A: " << count_records << endl;

    for (int i = 0; i < count_records; ++i)
    {

        gpc64_inst->mq_send(vector_keys[i]);
        auto val = gpc64_inst->mq_receive();

        // if (vector_keys[i] == vector_keys[vector_keys.size() - 1])
        cout << "Ключ:     " << vector_keys[i] << endl;
        cout << "Значение: " << val << endl;
    }

}

int main(int argc, char **argv)
{
    srand(time(NULL));

    ofstream log("task1.log"); // поток вывода сообщений
    unsigned long long offs = 0ull;
    gpc *gpc64_inst; // указатель на класс gpc

    // вектор ключей структуры A для удобного вывода на экран
    vector<uint64_t> vector_keys;

    // Инициализация gpc
    if (argc < 2)
    {
        log << "Использование: host_main <путь к файлу rawbinary>" << endl;
        return -1;
    }

    // Захват ядра gpc и запись sw_kernel
    gpc64_inst = new gpc();
    log << "Открывается доступ к " << gpc64_inst->gpc_dev_path << endl;
    if (gpc64_inst->load_swk(argv[1]) == 0)
    {
        log << "Программное ядро загружено из файла " << argv[1] << endl;
    }
    else
    {
        log << "Ошибка загрузки sw_kernel файла << argv[1]" << endl;
        return -1;
    }

    gpc64_inst->start(__event__(insert)); // обработчик вставки

    for (uint64_t i = 0; i < COUNT_RECORDS; ++i)
    {
        uint64_t rand_key = rand();

        // сохраняем ключ для возможного вывода на экран
        vector_keys.push_back(rand_key);

        uint64_t rand_val = rand();

        // передали ключ и значение
        gpc64_inst->mq_send(rand_key);
        gpc64_inst->mq_send(rand_val);
    }

    gpc64_inst->mq_send(-1ull); // терминальный символ

    printStructure(gpc64_inst, vector_keys);

    gpc64_inst->start(__event__(search)); // обработчик запроса поиска

    // // случаный ключ (в диапазоне от 0 до 511 -  чтобы были шансы найти)
    uint64_t rand_key = rand() % 512;
    // передали случайный ключ
    gpc64_inst->mq_send(rand_key);

    // получаем ака найденный ключ
    auto found_val = gpc64_inst->mq_receive();
```

```
    if (found_val == -1) // если ключ был не найден
    {
        cout << "Значение по ключу " << rand_key << " не было найдено" << endl;

        auto rc = gpc64_inst->mq_receive();

        if (rc == 0)
        {
            cout << "Значение по ключу " << rand_key << " было успешно добавлено в структуру A" << endl;
            vector_keys.push_back(rand_key);
            printStructure(gpc64_inst, vector_keys);
        }
        else
            cout << "Ошибка добавления ключа в структуру" << endl;
    }
    else
    {
        cout << "\n Значение по ключу " << rand_key << " было найдено" << endl;
        cout << "Ключ:     " << rand_key << endl;
        cout << "Значение: " << found_val << endl;

    }

    log << "Выход!" << endl;

    delete (gpc64_inst);

    return 0;
}
```

Листинг 2. Измененный код host

```
/*
 * gpc_handlers.h
 *
 * host and sw_kernel library
 *
 * Macro instantiation for handlers
 *
 */
#ifndef DEF_HANDLERS_H_
#define DEF_HANDLERS_H_
#define DECLARE_EVENT_HANDLER(handler) \
            const unsigned int event_ ## handler =__LINE__; \
            void handler ();
#define __event__(handler) event_ ## handler
//  Event handlers declarations by declaration line number
DECLARE_EVENT_HANDLER(insert);
DECLARE_EVENT_HANDLER(search);
DECLARE_EVENT_HANDLER(printA);
#endif
```

Измененный код gpc_handlers.h