**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 ИУ7-53Б Н. Д. Лысцев

Группа Подпись, дата И. О. Фамилия

Преподаватель А. Ю. Попов

Подпись, дата И. О. Фамилия

Москва, 2023 г.

**Цели работы**

В ходе практикума ознакомиться с архитектурой и принципами работы вычислительного комплекса Тераграф, 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 Тераграф 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 Тераграф.

Практикум состоит из трех этапов:

* **Исследование принципов функционирования вычислительного комплекса Тераграф**
* **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 взаимодествия 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 (1К 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

}

}

1. 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](https://latex.bmstu.ru/gitlab/hackathon2023/lab2/lab2/-/blob/main/include/common_struct.h?ref_type=heads#L55)

//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 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.

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. Изменнный 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 А: " << 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 А 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