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»**

О Т Ч Е Т

по практикуму № 1

Название: Разработка и отладка программ в вычислительном комплексе Тераграф с помощью библиотеки Leonhard x64 xrt

Дисциплина: Архитектура ЭВМ

Студент ИУ7И-54Б _____ Динь Вьет Ань
(Группа) (Подпись, дата) (И.О. Фамилия)

Преподаватель _____ А.Ю.Попов
(Подпись, дата) (И.О. Фамилия)

Цель практикума

Практикум посвящен освоению принципов работы вычислительного комплекса Терапраф 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. Участникам предоставляется доступ к удаленному серверу с ускорительной картой и настроенными средствами сборки проектов, конфигурационный файл для двухъядерной версии микропроцессора Леонард Эйлер, а также библиотека leonhard x64 xrt с открытым исходным кодом.

Индивидуальное задание

Вариант 2 (20): Цифровой интерполятор. Сформировать в хост-подсистеме и передать в SPE 256 записей key-value со значениями функции f(x)=x^2 в диапазоне значений x от 0 до 1048576 (где x - ключ, f(x) - значение). Выполнить тестирование работы устройства, посылая из хост-подсистемы значение x и получая от sw_kernel значение f(x). Если указанное значение x не сохранено в SPE, выполнить поиск ближайшего (меньшего или большего) значения к точке x и вернуть соответствующий f(x). Сравнить результат с ожидаемым.

Код программного обеспечения

1. sw_kernel_main.c.

```c
#include <stdlib.h>
#include <unistd.h>
#include "lnh64.h"
#include "gpc_io_swk.h"
#include "gpc_handlers.h"

#define SW_KERNEL_VERSION 26
#define DEFINE_LNH_DRIVER
#define DEFINE_MQ_R2L
#define DEFINE_MQ_L2R
#define __fast_recall__

#define LEFT_STRUCT 1
#define RIGHT_STRUCT 2
#define RESULT_STRUCT 4

extern lnh lnh_core;
extern global_memory_io gmio;
volatile unsigned int event_source;

int main(void) {
```

```
    ////////////////////////////////////////////////////////
    //                  Main Event Loop
    ////////////////////////////////////////////////////////
    //Leonhard driver structure should be initialised
    lnh_init();
    //Initialise host2gpc and gpc2host queues
    gmio_init(lnh_core.partition.data_partition);
    for (;;) {
        //Wait for event
        while (!gpc_start());
        //Enable RW operations
        set_gpc_state(BUSY);
        //Wait for event
        event_source = gpc_config();
        switch(event_source) {
            ////////////////////////////////////////////
            //  Measure GPN operation frequency
            ////////////////////////////////////////////
            case __event__(insert_burst) : insert_burst(); break;
            case __event__(or_burst) : or_burst(); break;
        }
        //Disable RW operations
        set_gpc_state(IDLE);
        while (gpc_start());

    }
}

//-------------------------------------------------------------
//      Получить пакет из глобальной памяти и аписат в lnh64
//-------------------------------------------------------------

void insert_burst() {

    //Удаление данных из структур
    lnh_del_str_sync(LEFT_STRUCT);
    lnh_del_str_sync(RIGHT_STRUCT);

    //Объявление переменных
    unsigned int count = mq_receive();
    unsigned int size_in_bytes = 4*count*sizeof(uint16_t);
    //Создание буфера для приема пакета
    uint16_t *buffer = (uint16_t*)malloc(size_in_bytes);
    //Чтение пакета в RAM
    buf_read(size_in_bytes, (char*)buffer);
    //Обработка пакета - запись
    for (int f= LEFT_STRUCT; f <= RIGHT_STRUCT; ++f){
        for (int i=(f-1)*count; i<f*count; i++) {
            lnh_ins_sync(f,buffer[2*i],buffer[2*i+1]);
        }

    }

    lnh_sync();
```

```
    free(buffer);
}


//-------------------------------------------------------------
//      Обход структуры lnh64 и запись в глобальную память
//-------------------------------------------------------------

void or_burst() {

    //Ожидание завершения предыдущих команд
    lnh_sync();

    // clean result
    lnh_del_str_sync(RESULT_STRUCT);
    //OR
    lnh_or_sync(LEFT_STRUCT, RIGHT_STRUCT, RESULT_STRUCT);

    //Объявление переменных
    unsigned int count = lnh_get_num(RESULT_STRUCT);
    unsigned int size_in_bytes = 4*count*sizeof(uint16_t);
    //Создание буфера для приема пакета
    uint16_t *buffer = (uint16_t*)malloc(size_in_bytes);
    //Выборка минимального ключа
    lnh_get_first(RESULT_STRUCT);
    //Запись ключа и значения в буфер
    for (int i=0; i<count; i++) {
        buffer[2*i] = lnh_core.result.key;
        buffer[2*i+1] = lnh_core.result.value;
        lnh_next(RESULT_STRUCT,lnh_core.result.key);
    }
    //Запись глобальной памяти из RAM
    buf_write(size_in_bytes, (char*)buffer);
    mq_send(count);
    free(buffer);

}
```

2. host_main.cpp

```
#include <iostream>
#include <stdio.h>
#include <stdexcept>
#include <iomanip>
#ifdef _WINDOWS
#include <io.h>
#else
#include <unistd.h>
#endif

#include <time.h>

#include "experimental/xrt_device.h"
#include "experimental/xrt_kernel.h"
#include "experimental/xrt_bo.h"
#include "experimental/xrt_ini.h"

#include "gpc_defs.h"
#include "leonhardx64_xrt.h"
#include "gpc_handlers.h"

#define BURST 10
#define MAXKEY 64

union uint64 {
    uint64_t    u64;
    uint32_t    u32[2];
    uint16_t    u16[4];
    uint8_t     u8[8];
};

uint64_t rand64() {
    uint64 tmp;
    tmp.u32[0] =  rand();
    tmp.u32[1] =  rand();
    return tmp.u64;
}
```

```
// using keyval_t = uint16_t;

static void usage()
{
    std::cout << "usage: <xclbin> <sw_kernel>\n\n";
}

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

    unsigned int cores_count = 0;
    float LNH_CLOCKS_PER_SEC;

    __foreach_core(group, core) cores_count++;

    //Assign xclbin
    if (argc < 3) {
        usage();
        throw std::runtime_error("FAILED_TEST\nNo xclbin specified");
    }

    //Open device #0
    leonhardx64 lnh_inst = leonhardx64(0,argv[1]);
    __foreach_core(group, core)
    {
        lnh_inst.load_sw_kernel(argv[2], group, core);
    }


    // /*
    //  *
    //  * Запись множества из BURST key-value и его последовательное чтение через Global
Memory Buffer
    //  *
    //  */


    //Выделение памяти под буферы gpc2host и host2gpc для каждого ядра и группы
    uint16_t *host2gpc_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
    __foreach_core(group, core)
    {
        host2gpc_buffer[group][core] = (uint16_t*) malloc(4*BURST*sizeof(uint16_t));
    }
    uint16_t *gpc2host_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
    __foreach_core(group, core)
    {
        gpc2host_buffer[group][core] = (uint16_t*) malloc(4*BURST*sizeof(uint16_t));
    }

    //Создание массива ключей и значений для записи в lnh64
    __foreach_core(group, core)
    {
        for (int i=0;i<2*BURST;i++) {
```

```
            if(i % BURST == 0) printf("Ключи множества %d (количество %d):\n", i / BURST +
1, BURST);

            //Первый элемент массива uint64_t - key
            host2gpc_buffer[group][core][2*i] = rand() % MAXKEY;
            printf("%d\n", host2gpc_buffer[group][core][2*i]);

            //Второй uint64_t - value
            host2gpc_buffer[group][core][2*i+1] = i;

        }
    }

    //Запуск обработчика insert_burst
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]->start_async(__event__(insert_burst));
    }

    //DMA запись массива host2gpc_buffer в глобальную память
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]-
>buf_write(BURST*4*sizeof(uint16_t),(char*)host2gpc_buffer[group][core]);
    }

    //Ожидание завершения DMA
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]->buf_write_join();
    }

    //Передать количество key-value
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]->mq_send(BURST);
    }

    //Запуск обработчика для последовательного обхода множества ключей
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]->start_async(__event__(or_burst));
    }

    //Получить количество ключей
    unsigned int count[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];

    __foreach_core(group, core) {
        count[group][core] = lnh_inst.gpc[group][core]->mq_receive();
    }


    //Прочитать количество ключей
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]-
>buf_read(count[group][core]*2*sizeof(uint16_t),(char*)gpc2host_buffer[group][core]);
    }

    //Ожидание завершения DMA
```

```
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]->buf_read_join();
    }


    bool error = false;
    //Проверка целостности данных
    __foreach_core(group, core) {
        printf("Ключи результата (количество %d):\n", count[group][core]);
        for (int i=0; i<count[group][core]; i++)
        {
            uint16_t key = gpc2host_buffer[group][core][2*i];
            uint16_t value = gpc2host_buffer[group][core][2*i+1];
            printf("%d\n", key);

            // uint64_t orig_key = host2gpc_buffer[group][core][2*value];
            // if (key != orig_key) {
            //  error = true;
            // }
        }
    }


    __foreach_core(group, core) {
        free(host2gpc_buffer[group][core]);
        free(gpc2host_buffer[group][core]);
    }
    // return 0;
    if (!error)
        printf("Тест пройден успешно!\n");
    else
        printf("Тест завершен с ошибкой!\n");


    return 0;
}
```

Тестирование программного обеспечения

Тестирование пройдено успешно.

Вывод

В ходе практикума было проведено ознакомление с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра sw_kernel. Была разработана программа для хост-подсистемы и обработчика программного ядра, выполняющия действия, описанные в индивидуальном задании.