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 **КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

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 Электронно-вычислительных машин

Студент **ИУ7 - 53Б** _____ **А.А. Светличная**
(Группа) (Подпись, дата) (И.О. Фамилия)

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

Москва, 2022

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.

```
#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 TEST_STRUCTURE 1

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

int main(void) {
```

```
    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__(search_burst) : search_burst(); break;
        }
        //Disable RW operations
        set_gpc_state(IDLE);
        while (gpc_start());

    }
}

void insert_burst() {

    //Удаление данных из структур
    lnh_del_str_sync(TEST_STRUCTURE);
    //Объявление переменных
    unsigned int count = mq_receive();
    unsigned int size_in_bytes = 2*count*sizeof(uint64_t);
    //Создание буфера для приема пакета
    uint64_t *buffer = (uint64_t*)malloc(size_in_bytes);
    //Чтение пакета в RAM
    buf_read(size_in_bytes, (char*)buffer);
    //Обработка пакета - запись
    for (int i=0; i<count; i++) {
        lnh_ins_sync(TEST_STRUCTURE,buffer[2*i],buffer[2*i+1]);
    }
    lnh_sync();
    free(buffer);
}

void search_burst() {
    //Ожидание завершения предыдущих команд
    lnh_sync();
```

```
    //Объявление переменных
    unsigned int count = lnh_get_num(TEST_STRUCTURE);
    //Передать количество key-value
    mq_send(count);
    //Получить ключ
    auto key = mq_receive();
    //Поиск по ключу
        lnh_search(1, key);
    //Отправка ответа
        mq_send(lnh_core.result.value);
    // mq_send(buffer[2*1+1]);
}
```

2. host_main.cpp

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

#include <cmath>

#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 256

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;
}

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

int main(int argc, char** argv)
{
        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);
        }

        // Выделение памяти под буферы gpc2host и host2gpc для каждого ядра и группы
        uint64_t *host2gpc_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
        __foreach_core(group, core) {
                host2gpc_buffer[group][core] = (uint64_t*) malloc(2*BURST*sizeof(uint64_t));
        }

        uint64_t *gpc2host_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
        __foreach_core(group, core) {
                gpc2host_buffer[group][core] = (uint64_t*) malloc(2*BURST*sizeof(uint64_t));
        }
```

```
	uint64_t keys[BURST], values[BURST];
	uint64_t num = 0;
	uint64_t start_key = 0;

	for (size_t i = 0; i < BURST; i++)
	{
		keys[i] = num;
		values[i] = num * num;
		num += 1;
	}

	uint64_t user_key = 0;
	long double tmp_key = 0;

	printf("Enter X value: ");
	scanf("%llf", &tmp_key);

	// Создание массива ключей и значений для записи в lnh64
	__foreach_core(group, core) {
		for (int i=0;i<BURST;i++)
		{
			//Первый элемент массива uint64_t - key
			host2gpc_buffer[group][core][2*i] = start_key + i;
			//Второй uint64_t - value
			host2gpc_buffer[group][core][2*i+1] = tmp_key * tmp_key;

		}
	}

	//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*2*sizeof(uint64_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 и наш ключ
```

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

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

        //Получить количество ключей и значение по переданному ключу
        unsigned int count[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
        unsigned int answer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];

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


        //Прочитать количество ключей
        __foreach_core(group, core) {
                lnh_inst.gpc[group][core]-
>buf_read(count[group][core]*2*sizeof(uint64_t),(char*)gpc2host_buffer[group][co
re]);
        }

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


        //Чтение значения, полученного по ключу и проверка целостности данных
        __foreach_core(group, core) {
                uint64_t value = answer[group][core];
                uint64_t orig_value = host2gpc_buffer[group][core][2*user_key+1];
                printf("Result: %llu ", value);

                if (value == orig_value) {
                        printf("(CORRECT)\n");
                }
                else {
                        printf("(INCORRECT)\n");
```

```
            }
        }


        __foreach_core(group, core) {
            free(host2gpc_buffer[group][core]);
            free(gpc2host_buffer[group][core]);
        }

        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.