



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 09.03.01 Информатика и вычислительная техника

О Т Ч Е Т

по практикуму № 1

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

Дисциплина: Организация ЭВМ и систем

Студент

ИУ6-72Б

(Группа)

(Подпись, дата)

С.В. Астахов

(И.О. Фамилия)

Преподаватель

(Подпись, дата)

(И.О. Фамилия)

Цель работы: освоение принципов работы вычислительного комплекса Тераграф и получению практических навыков решения задач обработки множеств на основе гетерогенной вычислительной структуры. Ознакомление с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра sw_kernel.

Задание

Сетевой коммутатор на 128 портов. Сформировать в хост-подсистеме и передать в SPE таблицу коммутации из 254 ip адресов 195.19.32.1/24 (адреса 195.19.32.1 .. 195.19.32.254). Каждому адресу поставить в соответствие один из 128 интерфейсов (целые числа 0..127). Выполнить тестирование работы коммутатора, посылая из хост-подсистемы ip адреса и сравнивая полученный от GPC номер интерфейса с ожидаемым.

Исходный код решения задачи представлен в листингах 1 и 2.

Листинг 1 — host_main.cpp

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

#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
#define TESTSET_SIZE 10

uint64_t orig_values[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);
    }

    // /*
    // *
    // * Запись множества из BURST key-value
    // *
    // */

    //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));
    }

```

```

    }

    //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] = i;
            //Второй uint64_t - value
            orig_values[i] = rand64() % 128;
            host2gpc_buffer[group][core][2*i+1] = orig_values[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*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
    __foreach_core(group, core) {
        lnh_inst.gpc[group][core]->mq_send(BURST);
    }

    unsigned int value[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];

    bool error = false;
    //Проверка целостности данных
    __foreach_core(group, core) {
        for (int i=0; i<TESTSET_SIZE; i++) {
            int key = rand64() % 256;
            lnh_inst.gpc[group][core]->start_async(__event__(search_burst));
//Запустить обработчик
            lnh_inst.gpc[group][core]->mq_send(key); // Запрос
            value[group][core] = lnh_inst.gpc[group][core]->mq_receive(); // Ответ
            if (value[group][core] != orig_values[key]) {
                error = true;
                printf("[x] ip: 195.19.32.%d int: %d orig_int: %d \n", key,
value[group][core], orig_values[key]);
            } else {
                printf("[v] ip: 195.19.32.%d int: %d orig_int: %d \n", key,
value[group][core], orig_values[key]);
            }
        }
    }

```

```

        }
    }
}

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

if (!error)
    printf("Тест пройден успешно!\n");
else
    printf("Тест завершен с ошибкой!\n");

return 0;
}

```

Листинг 2 — sw_kernel.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) {
    //////////////////////////////////////
    //          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__(search_burst) : search_burst(); break;
    //case __event__(search_interface) : search_interface(); break;
}
//Disable RW operations
set_gpc_state(IDLE);
while (gpc_start());

}
}

//-----
//   Получить пакет из глобальной памяти и аписат в Inh64
//-----

void insert_burst() {

    //Удаление данных из структур
    Inh_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++) {
        Inh_ins_sync(TEST_STRUCTURE,buffer[2*i],buffer[2*i+1]);
    }
    Inh_sync();
    free(buffer);
}

//-----
//   Обход структуры Inh64 и возврат по ключу
//-----

void search_burst() {

    //Ожидание завершения предыдущих команд
    Inh_sync();
    //Объявление переменных
    unsigned int count = Inh_get_num(TEST_STRUCTURE);
    unsigned int size_in_bytes = 2*count*sizeof(uint64_t);

    //Выборка минимального ключа

```

```

Inh_get_first(TEST_STRUCTURE);

//Получение ключа
unsigned int key = mq_receive();

char search_complete = 0;
int i = 0;

/*
while ((i<count) && (search_complete == 0)) {
    if(key == Inh_core.result.key){
        search_complete = 1;
        mq_send((unsigned int) Inh_core.result.value);
    }
    Inh_next(TEST_STRUCTURE,Inh_core.result.key);
}

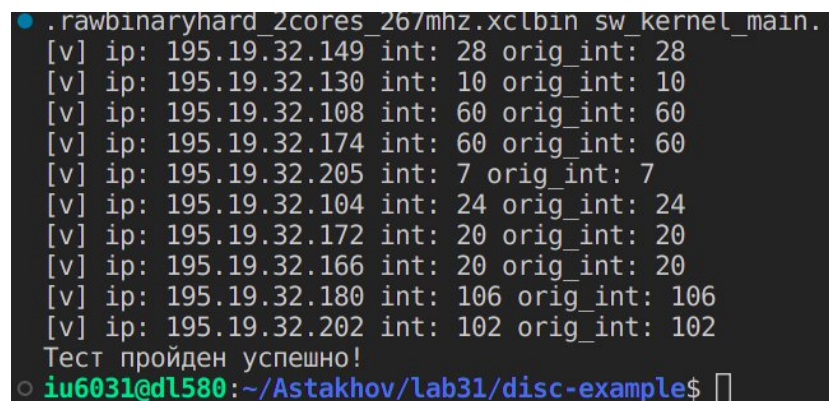
// Если элемент не найден
if(search_complete == 0){
    mq_send(404);
}
*/

if(Inh_search(TEST_STRUCTURE, key)){
    mq_send((unsigned int) Inh_core.result.value);
} else {
    mq_send(404);
}

}

```

Результаты тестирования программы представлены на рисунке 1.



```

● .rawbinaryhard_2cores_26/mhz.xclbin sw kernel_main.
[v] ip: 195.19.32.149 int: 28 orig_int: 28
[v] ip: 195.19.32.130 int: 10 orig_int: 10
[v] ip: 195.19.32.108 int: 60 orig_int: 60
[v] ip: 195.19.32.174 int: 60 orig_int: 60
[v] ip: 195.19.32.205 int: 7 orig_int: 7
[v] ip: 195.19.32.104 int: 24 orig_int: 24
[v] ip: 195.19.32.172 int: 20 orig_int: 20
[v] ip: 195.19.32.166 int: 20 orig_int: 20
[v] ip: 195.19.32.180 int: 106 orig_int: 106
[v] ip: 195.19.32.202 int: 102 orig_int: 102
Тест пройден успешно!
○ iu6031@ed1580:~/Astakhov/lab31/disc-example$ █

```

Рисунок 1 — результаты тестирования программы

Вывод: в результате выполнения лабораторной работы были освоены принципы работы вычислительного комплекса Тераграф и получены практические навыки решения задач обработки множеств на основе гетерогенной вычислительной структуры.