|  |  |
| --- | --- |
|  | **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**

**Отчет**

|  |  |
| --- | --- |
| **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 |  |  | С.В. Астахов | |
|  | (Группа) |  | (Подпись, дата) | | (И.О. Фамилия) |
|  |  |  |  | |  |
| Преподаватель |  |  |  | |  |
|  |  |  | (Подпись, дата) | | (И.О. Фамилия) |

Москва, 2022

**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 lnh64  //-------------------------------------------------------------    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);  }  //-------------------------------------------------------------  // Обход структуры lnh64 и возврат по ключу  //-------------------------------------------------------------    void search\_burst() {  //Ожидание завершения предыдущих команд  lnh\_sync();  //Объявление переменных  unsigned int count = lnh\_get\_num(TEST\_STRUCTURE);  unsigned int size\_in\_bytes = 2\*count\*sizeof(uint64\_t);  //Выборка минимального ключа  lnh\_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 == lnh\_core.result.key){  search\_complete = 1;  mq\_send((unsigned int) lnh\_core.result.value);  }  lnh\_next(TEST\_STRUCTURE,lnh\_core.result.key);  }  // Если элемент не найден  if(search\_complete == 0){  mq\_send(404);  }  \*/  if(lnh\_search(TEST\_STRUCTURE, key)){  mq\_send((unsigned int) lnh\_core.result.value);  } else {  mq\_send(404);  }  } |

Результаты тестирования программы представлены на рисунке 1.

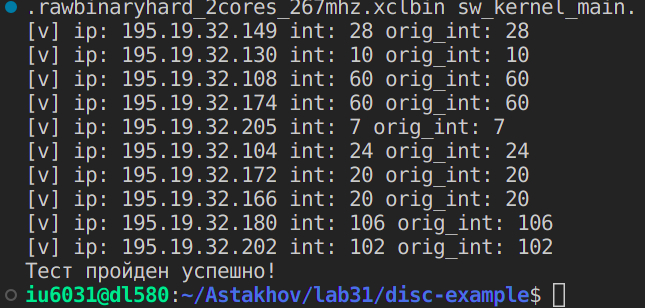


Рисунок 1 — результаты тестирования программы

**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.