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»

ОТЧЕТ ПО ПРАКТИКУМУ №1
по курсу «Архитектура ЭВМ»

«Разработка и отладка программ в вычислительном комплексе Тераграф с помощью библиотеки leonhard x64 xrt»

Студент Золотухин Алексей Вячеславович

Группа ИУ7-54Б

Оценка (баллы) _____

Преподаватель Ибрагимов С.В.

2022 г.

Содержание

Введение

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. 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.

1 Сведения о взаимодействии хост-подсистемы и программного ядра sw_kernel

Рассмотрим следующие примеры кода подсистемы и программного ядра, которые мы будем использовать в практикуме. Пример выполняет следующие действия:

1. Хост-подсистема инициализирует ядра GPC:
   lnh_inst.load_sw_kernel(argv[2], group, core);, после чего становится возможным запуск обработчиков программного ядра sw_kernel. В практикуме используется версия микропроцессора Леонард Эйлер с одной группой и двумя ядрами GPC, при этом в файле gpc_defs.h разрешено использование только ядра #0 группы #0 (настройка может быть изменена пользователем).

2. Хост-подсистема выделяет память под буферы
   gpc2host_buffer и host2gpc_buffer

3. В буфере host2gpc_buffer инициализируется массив ключей и значений для записи в GPC.

4. Запускается обработчик insert_burst и запускается механизм прямого доступа к памяти для записи host2gpc_buffer в глобальную память группы #0. Последовательность указанных действий может быть обратной: сначала может быть запущен механизм DMA, после чего запускается обработчик.

5. Хост-подсистема ожидает завершения копирования памяти
   (buf_write_join) для синхронизации процессов.

6. Хост-подсистема передает сообщение с количеством ключей и значений (m_send(...)).

7. Программное ядро в обработчике insert_burst получает сообщение и выделяет буфер (buffer) для хранения данных в RAM CPE.

8. Программное ядро копирует данные в буфер и последовательно вызывает команду INS микропроцессора lnh64:
lnh_ins_sync(TEST_STRUCTURE,buffer[2*i],buffer[2*i+1]).

9. Последовательность действий для реализации пакетной обработки Последовательность действий для реализации пакетной обработки

10. Далее происходит запуск обработчика для последовательного обхода множества ключей и выдачи их обратно в хост-подсистему.

11. Программное ядро в обработчике search_burst получает количество ключей в структуре
(unsigned int count = lnh_get_num(TEST_STRUCTURE))

12. Обход структуры начинается с выборки минимального ключа
(lnh_get_first(TEST_STRUCTURE))

13. Далее осуществляется запись ключа и значения в буфер для обратной передачи в хост-подсистему.

14. Используя итерационный цикл, или же проверяя результат выполнения функции lnh_next(TEST_STRUCTURE,lnh_core.result.key) цикл повторяется до обхода всей структуры TEST_STRUCTURE.

15. Альтернативно может быть использован следующий код для последовательного обхода структуры.

Листинг 1.1 – Код последовательного обхода структуры

```
1       lnh_get_first(TEST_STRUCTURE);
2       do {
3           //lnh_core.result.key   - found key
4           //lnh_core.result.value - found value
5           ...
6       } while (lnh_next(TEST_STRUCTURE, lnh_core.result.key));
```

16. По завершению обхода программное ядро посылает сообщение с количеством переданных ключей и значений.

17. Хост-подсистема ожидает получения сообщения, после чего последовательно проверяет, что ключ в буфере совпадает с ожидаемым.

18. В итоге выдается сообщение о результате тестирования.

2 Выполнение практикума

Вариант 6

2.1 Индивидуальное задание

Разработать программу для хост-подсистемы и обработчики программного ядра, выполняющие следующие действия.
Сформировать в хост-подсистеме и передать в SPE 256 записей с ключами x и значениями $f(x) = x^2$ в диапазоне значений x от 0 до 1048576. Передать в sw_kernel числа $x1$ и $x2$ ($x2 > x1$). В хост-подсистему вернуть сумму значений $f(x)$ на диапазоне $(x1, x2)$. Сравнить результат с ожидаемым.

2.2 Выполнение индивидуального задания

Для выполнения индивидуального задания были написаны программы для хост-подсистемы и программного ядра sw_kernel.

В листингах 2.1 – 2.2 показан код двух программ – для хост подсистемы и программного ядра sw_kernel.

Листинг 2.1 – Код программы host_main.cpp

```
1  #include <iostream>
2  #include <stdio.h>
3  #include <stdexcept>
4  #include <iomanip>
5  #ifdef _WINDOWS
6  #include <io.h>
7  #else
8  #include <unistd.h>
9  #endif
10
11
12 #include "experimental/xrt_device.h"
13 #include "experimental/xrt_kernel.h"
14 #include "experimental/xrt_bo.h"
15 #include "experimental/xrt_ini.h"
16
17 #include "gpc_defs.h"
18 #include "leonhardx64_xrt.h"
19 #include "gpc_handlers.h"
20
21 #define BURST 256
22
23 union uint64 {
24     uint64_t    u64;
25     uint32_t    u32[2];
26     uint16_t    u16[4];
27     uint8_t     u8[8];
28 };
29
30 uint64_t rand64() {
31     uint64 tmp;
32     tmp.u32[0] =  rand();
33     tmp.u32[1] =  rand();
34     return tmp.u64;
35 }
36
37 static void usage()
38 {
39     std::cout << "usage: <xclbin> <sw_kernel>\n\n";
40 }
```

```
41
42 const uint64_t start_ip = 1;
43
44 int main(int argc, char** argv)
45 {
46     unsigned int cores_count = 0;
47     float LNH_CLOCKS_PER_SEC;
48
49
50     //Assign xclbin
51     if (argc < 3) {
52         usage();
53         throw std::runtime_error("FAILED_TEST\nNo␣xclbin␣
               specified");
54     }
55
56     //Open device #0
57     leonhardx64 lnh_inst = leonhardx64(0,argv[1]);
58     lnh_inst.load_sw_kernel(argv[2], 0, 0);
59
60
61     //Выделение памятиподбуферы  gpc2host и host2gpc
          длякаждогоядраигруппы
62     uint64_t *host2gpc_buffer = (uint64_t*)
          malloc(2*BURST*sizeof(uint64_t));
63     uint64_t *gpc2host_buffer = (uint64_t*)
          malloc(2*BURST*sizeof(uint64_t));
64
65     uint64_t x1, x2;
66     std::cout << "Input␣x1,␣x2␣(256␣>=␣x2␣>␣x1␣>=␣0):" <<
          std::endl;
67     std::cin >> x1 >> x2;
68
69     if (!(x1 >= 0 && x1 < x2 && x2 < 256))
70     {
71         std::cout << "Error:␣incorrect␣x1,␣x2" << std::endl;
72         return 1;
73     }
74
75     //Создание массиваключейизначенийдлязаписив  lnh64
76     for (int i = 0; i < BURST ; i++)
```

```
77      {
78          host2gpc_buffer[2*i] = i;
79          //Второй uint64_t - value
80          host2gpc_buffer[2*i+1] =  i * i;
81          // std::cout << host2gpc_buffer[2*i] << " " <<
                host2gpc_buffer[2*i+1] << std::endl;
82      }
83
84      //Запуск обработчика insert_burst
85      lnh_inst.gpc[0][0]->start_async(__event__(insert_burst));
86
87      //DMA записьмассива  host2gpc_buffer вглобальнуюпамять
88
            lnh_inst.gpc[0][0]->buf_write(BURST*2*sizeof(uint64_t),(char*)host
89
90      //Ожидание завершения DMA
91      lnh_inst.gpc[0][0]->buf_write_join();
92
93      //Передать количество key-value инашключ
94      lnh_inst.gpc[0][0]->mq_send(BURST);
95      lnh_inst.gpc[0][0]->mq_send(x1);
96      lnh_inst.gpc[0][0]->mq_send(x2);
97
98      //Запуск обработчикадляпоследовательногообходамножестваключей
99      lnh_inst.gpc[0][0]->start_async(__event__(search_burst));
100
101     //Получить количествоключейизначениепопереданномуключу
102     unsigned int count;
103     uint64_t result;
104
105     count = lnh_inst.gpc[0][0]->mq_receive();
106     result = lnh_inst.gpc[0][0]->mq_receive();
107
108     //Прочитать количествоключей
109     lnh_inst.gpc[0][0]->buf_read(count * 2 * sizeof(uint64_t),
            (char*)gpc2host_buffer);
110
111     //Ожидание завершения DMA
112     lnh_inst.gpc[0][0]->buf_read_join();
113
114     //Чтение значения, полученногопоключуипроверкацелостностиданных
```

```
115     x2--;
116     uint64_t answer = x2 * (x2 + 1) * (2 * x2 + 1) / 6 - x1 * (x1
            + 1) * (2 * x1 + 1) / 6;
117     std::cout << "Your integral: " << result << std::endl <<
            "Expected: " << answer << std::endl;
118
119     if (result == answer)
120         std::cout << "(CORRECT)" << std::endl;
121     else
122         std::cout << "(INCORRECT)" << std::endl;
123
124     free(host2gpc_buffer);
125     free(gpc2host_buffer);
126     return 0;
127 }
```

Листинг 2.2 – Код программы sw_kernel_main.c

```
1  /*
2   * gpc_test.c
3   *
4   * sw_kernel library
5   *
6   *  Created on: April 23, 2021
7   *      Author: A.Popov
8   */
9
10 #include <stdlib.h>
11 #include <unistd.h>
12 #include "lnh64.h"
13 #include "gpc_io_swk.h"
14 #include "gpc_handlers.h"
15
16 #define SW_KERNEL_VERSION 26
17 #define DEFINE_LNH_DRIVER
18 #define DEFINE_MQ_R2L
19 #define DEFINE_MQ_L2R
20 #define __fast_recall__
21
22 #define TEST_STRUCTURE 1
23
24 extern lnh lnh_core;
25 extern global_memory_io gmio;
26 volatile unsigned int event_source;
27
28 int main(void) {
29     ////////////////////////////////////////////////////////
30     //                    Main Event Loop
31     ////////////////////////////////////////////////////////
32     //Leonhard driver structure should be initialised
33     lnh_init();
34     //Initialise host2gpc and gpc2host queues
35     gmio_init(lnh_core.partition.data_partition);
36     for (;;) {
37         //Wait for event
38         while (!gpc_start());
39         //Enable RW operations
40         set_gpc_state(BUSY);
```

```
41          //Wait for event
42          event_source = gpc_config();
43          switch(event_source) {
44              ////////////////////////////////////////////
45              //  Measure GPN operation frequency
46              ////////////////////////////////////////////
47              case __event__(insert_burst) : insert_burst(); break;
48              case __event__(search_burst) : search_burst(); break;
49          }
50          //Disable RW operations
51          set_gpc_state(IDLE);
52          while (gpc_start());
53
54      }
55  }
56
57  //-------------------------------------------------------------------------
58  // Получитьпакетизглобальнойпамятииаписатв                    lnh64
59  //-------------------------------------------------------------------------
60
61  void insert_burst() {
62
63      //Удаление данныхизструктур
64      lnh_del_str_sync(TEST_STRUCTURE);
65      //Объявление переменных
66      unsigned int count = mq_receive();
67      unsigned int size_in_bytes = 2*count*sizeof(uint64_t);
68      //Создание буферадляприемапакета
69      uint64_t *buffer = (uint64_t*)malloc(size_in_bytes);
70      //Чтение пакетав  RAM
71      buf_read(size_in_bytes, (char*)buffer);
72      //Обработка пакета - запись
73      for (int i=0; i<count; i++) {
74          lnh_ins_sync(TEST_STRUCTURE, buffer[2*i], buffer[2*i+1]);
75      }
76      lnh_sync();
77      free(buffer);
78  }
79
80
81  void search_burst() {
```

```
82      //Ожидание завершенияпредыдущихкоманд
83      lnh_sync();
84      //Объявление переменных
85      unsigned int count = lnh_get_num(TEST_STRUCTURE);
86      //Передать количество key-value
87      mq_send(count);
88      //Получить ключ
89      auto x1 = mq_receive();
90      auto x2 = mq_receive();
91      //Поиск поключу
92
93      lnh_grls_sync(x1, x2, 1, 2);
94      uint64_t n = lnh_get_num(2);
95      lnh_get_first(2);
96      uint64_t sum = lnh_core.result.value;
97      for (int i = 0; i < n - 1; i++)
98      {
99          lnh_next(2, lnh_core.result.key);
100         sum += lnh_core.result.value;
101     }
102     //Отправка ответа
103     mq_send(sum);
104     // mq_send(buffer[2*1+1]);
105 }
```

В результате запуска данных программ выводится сумма квадратов между введёнными числами.

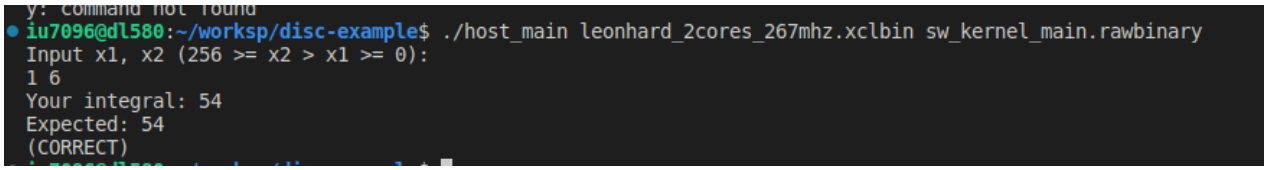

Рисунок 2.1 – Результат работы программ

Заключение

В данном практикуме были освоены принципы работы вычислительного комплекса Терраграф и получены практические навыки решения задач обработки множеств на основе гетерогенной вычислительной структуры.

В ходе практикума была изучена структура двух взаимодействующих программ: хост-подсистемы и программного ядра sw_kernel.

Работа проводилась на удаленном сервере с ускорительной картой, микропроцессором Леонард Эйлер, а также библиотекой leonhard x64 xrt с открытым исходным кодом.

Было выполнено задание согласно индивидуальному варианту.