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
Задание №1

Тема практикума «Обработка и визуализация графов.»

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

Дисциплина «Архитектура элекронно-вычислительных» машин

Студент: _____ Чыонг В. Х.
подпись, дата Фамилия, И.О.

Преподаватель: _____ Попов А. Ю.
подпись, дата Фамилия, И. О.

Москва — 2022 г.

Содержание

Цель работы

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 Основные теоретические сведения

Основная вычислительная системы (так называемая хост-подсистема) берет на себя функции управления запуском вычислительных задач, поддержкой сетевых подключений, обработкой и балансировкой нагрузки. В хост-подсистему входят два многоядерных ЦПУ по 26 ядер каждый, оперативная память на 1 Тбайт и дополнительная энергонезависимая память на 8 Тбайт, где хранятся атрибуты вершин и ребер графа, буферизируются поступающие запросы на обработку и визуализацию графов, хранятся временные данные об изменениях в графах. В хост-подсистеме используется процессор с архитектурой x86 для обеспечения сетевого взаимодействия и связи системы с внешним миром. Указанные функции реализованы в Программном ядре хост-подсистемы (host software kernel) – программном обеспечении, взаимодействующим с подсистемой обработки графов через шину PCIe.

Основу взаимодействия подсистем при обработке графов составляет передача блоков данных и коротких сообщений между GPC и хост-подсистемой. Для передачи сообщений для каждого GPC реализованы два аппаратных FIFO буфера на 512 записей: Host2GPC для передачи от хост-подсистемы к ядру, и GPC2Host для передачи в обратную сторону.

Обработка начинается с того, что собранное программное ядро (software kernel) загружается в локальное ОЗУ одного или нескольких CPE (микропроцессора riscv32im). Для этого используется механизм прямого доступа к памяти со стороны хост-подсистемы. В свою очередь, GPC (один или несколько) получают сигнал о готовности образа software kernel в Глобальной памяти, после чего вызывается загрузчик, хранимый в ПЗУ CPE. Загрузчик выполняет копирование программного ядра из Глобальной памяти в ОЗУ CPE и передает управление на начальный адрес программы обработки. Предусмотрен режим работы GPC, при котором во время обработки происходит обмен данными и сообщениями. Эти два варианта работы реализуется через буферы и очереди соответственно. На рисунке 7 представлена диаграмма последовательностей первого сценария работы – вызов обработчика с передачей параметров и возвратом значения через очередь сообщений.

2 Экспериментальная часть

2.1 Индивидуальное задание

Задание практикума выполнялось по варианту 11: Устройство формирования индексов SQL EXCEPT. Сформировать в хост-подсистеме и передать в SPE 256 записей множества A (случайные числа в диапазое 0..1024) и 256 записей множества B (случайные числа в диапазоне 0..1024). Сформировать в SPE множество C = A not B. Выполнить тестирование работы SPE, сравнив набор ключей в множестве C с ожидаемым.

2.2 Результаты выполнения задания

2.2.1 Host

Листинг 2.1 – Измененный код хост-системы под индивидульное задание

```
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 #include <time.h>
12
13 #include "experimental/xrt_device.h"
14 #include "experimental/xrt_kernel.h"
15 #include "experimental/xrt_bo.h"
16 #include "experimental/xrt_ini.h"
17
18 #include "gpc_defs.h"
19 #include "leonhardx64_xrt.h"
```

```
20 #include "gpc_handlers.h"
21
22 #define BURST 10
23 #define MAXKEY 64
24
25 union uint64 {
26     uint64_t    u64;
27     uint32_t    u32[2];
28     uint16_t    u16[4];
29     uint8_t     u8[8];
30 };
31
32 uint64_t rand64() {
33     uint64 tmp;
34     tmp.u32[0] =  rand();
35     tmp.u32[1] =  rand();
36     return tmp.u64;
37 }
38
39 // using keyval_t = uint16_t;
40
41 static void usage()
42 {
43     std::cout << "usage: <xclbin> <sw_kernel>\n\n";
44 }
45
46 int main(int argc, char** argv)
47 {
48     srand(time(NULL));
49
50     unsigned int cores_count = 0;
51     float LNH_CLOCKS_PER_SEC;
52
53     __foreach_core(group, core) cores_count++;
54
55     //Assign xclbin
56     if (argc < 3) {
57         usage();
58         throw std::runtime_error("FAILED_TEST\nNo xclbin
               specified");
59     }
```

```
60
61      //Open device #0
62      leonhardx64 lnh_inst = leonhardx64(0,argv[1]);
63      __foreach_core(group, core)
64      {
65          lnh_inst.load_sw_kernel(argv[2], group, core);
66      }
67
68
69      // /*
70      //  *
71      //  * Запись множества из BURST key-value и его последовательн
            ое чтение через Global Memory Buffer
72      //  *
73      //  */
74
75
76      //Выделение памяти под буферы gpc2host и host2gpc для каждого
            ядра и группы
77      uint16_t
            *host2gpc_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
78      __foreach_core(group, core)
79      {
80          host2gpc_buffer[group][core] = (uint16_t*)
               malloc(4*BURST*sizeof(uint16_t));
81      }
82      uint16_t
            *gpc2host_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
83      __foreach_core(group, core)
84      {
85          gpc2host_buffer[group][core] = (uint16_t*)
               malloc(4*BURST*sizeof(uint16_t));
86      }
87
88      //Создание массива ключей и значений для записи в lnh64
89      __foreach_core(group, core)
90      {
91          for (int i=0;i<2*BURST;i++) {
92              if(i % BURST == 0) printf("Ключи␣множества␣%d␣
                   (количество␣%d):\n", i / BURST + 1, BURST);
93
```

```
94              //Первый элемент массива uint64_t - key
95              host2gpc_buffer[group][core][2*i] = rand() % MAXKEY;
96              printf("%d\n", host2gpc_buffer[group][core][2*i]);
97
98              //Второй uint64_t - value
99              host2gpc_buffer[group][core][2*i+1] = i;
100
101         }
102     }
103
104     //Запуск обработчика insert_burst
105     __foreach_core(group, core) {
106         lnh_inst.gpc[group][core]->start_async(__event__(insert_burst));
107     }
108
109     //DMA запись массива host2gpc_buffer в глобальную память
110     __foreach_core(group, core) {
111         lnh_inst.gpc[group][core]->buf_write(BURST*4*sizeof(uint16_t),(ch
112     }
113
114     //Ожидание завершения DMA
115     __foreach_core(group, core) {
116         lnh_inst.gpc[group][core]->buf_write_join();
117     }
118
119     //Передать количество key-value
120     __foreach_core(group, core) {
121         lnh_inst.gpc[group][core]->mq_send(BURST);
122     }
123
124     //Запуск обработчика для последовательного обхода множества кл
        ючей
125     __foreach_core(group, core) {
126         lnh_inst.gpc[group][core]->start_async(__event__(or_burst));
127     }
128
129     //Получить количество ключей
130     unsigned int count[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
131
132     __foreach_core(group, core) {
```

```
133         count[group][core] =
                lnh_inst.gpc[group][core]->mq_receive();
134     }
135
136
137     //Прочитать количество ключей
138     __foreach_core(group, core) {
139         lnh_inst.gpc[group][core]->buf_read(count[group][core]*2*sizeof(u
140     }
141
142     //Ожидание завершения DMA
143     __foreach_core(group, core) {
144         lnh_inst.gpc[group][core]->buf_read_join();
145     }
146
147
148     bool error = false;
149     //Проверка целостности данных
150     __foreach_core(group, core) {
151         printf("Ключи␣результата␣(количество␣%d):\n",
                count[group][core]);
152         for (int i=0; i<count[group][core]; i++)
153         {
154             uint16_t key = gpc2host_buffer[group][core][2*i];
155             uint16_t value = gpc2host_buffer[group][core][2*i+1];
156             printf("%d\n", key);
157
158             // uint64_t orig_key =
                    host2gpc_buffer[group][core][2*value];
159             // if (key != orig_key) {
160             //  error = true;
161             // }
162         }
163     }
164
165
166     __foreach_core(group, core) {
167         free(host2gpc_buffer[group][core]);
168         free(gpc2host_buffer[group][core]);
169     }
170     // return 0;
```

```
171     if (!error)
172         printf("Тест пройден успешно!\n");
173     else
174         printf("Тест завершен с ошибкой!\n");
175
176
177     return 0;
178 }
```

2.2.2 sw_kernel

Листинг 2.2 – Измененный код sw_kernel под индивидульное задание

```
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 LEFT_STRUCT 1
23 #define RIGHT_STRUCT 2
24 #define RESULT_STRUCT 4
25
26 extern lnh lnh_core;
27 extern global_memory_io gmio;
```

```
28 volatile unsigned int event_source;
29
30 int main(void) {
31     ////////////////////////////////////////////////////
32     //                  Main Event Loop
33     ////////////////////////////////////////////////////
34     //Leonhard driver structure should be initialised
35     lnh_init();
36     //Initialise host2gpc and gpc2host queues
37     gmio_init(lnh_core.partition.data_partition);
38     for (;;) {
39         //Wait for event
40         while (!gpc_start());
41         //Enable RW operations
42         set_gpc_state(BUSY);
43         //Wait for event
44         event_source = gpc_config();
45         switch(event_source) {
46             ////////////////////////////////////////////
47             //  Measure GPN operation frequency
48             ////////////////////////////////////////////
49             case __event__(insert_burst) : insert_burst(); break;
50             case __event__(or_burst) : or_burst(); break;
51         }
52         //Disable RW operations
53         set_gpc_state(IDLE);
54         while (gpc_start());
55
56     }
57 }
58
59 //-------------------------------------------------------------------
60 //      Получить пакет из глобальной памяти и аписат в lnh64
61 //-------------------------------------------------------------------
62
63 void insert_burst() {
64
65     //Удаление данных из структур
66     lnh_del_str_sync(LEFT_STRUCT);
67     lnh_del_str_sync(RIGHT_STRUCT);
68
```

```
69      //Объявление переменных
70      unsigned int count = mq_receive();
71      unsigned int size_in_bytes = 4*count*sizeof(uint16_t);
72      //Создание буфера для приема пакета
73      uint16_t *buffer = (uint16_t*)malloc(size_in_bytes);
74      //Чтение пакета в RAM
75      buf_read(size_in_bytes, (char*)buffer);
76      //Обработка пакета - запись
77      for (int f= LEFT_STRUCT; f <= RIGHT_STRUCT; ++f){
78          for (int i=(f-1)*count; i<f*count; i++) {
79              lnh_ins_sync(f,buffer[2*i],buffer[2*i+1]);
80          }
81
82      }
83
84      lnh_sync();
85      free(buffer);
86  }
87
88
89  //-------------------------------------------------------------------
90  //      Обход структуры lnh64 и запись в глобальную память
91  //-------------------------------------------------------------------
92
93  void or_burst() {
94
95      //Ожидание завершения предыдущих команд
96      lnh_sync();
97
98      // clean result
99      lnh_del_str_sync(RESULT_STRUCT);
100     //OR
101     lnh_or_sync(LEFT_STRUCT, RIGHT_STRUCT, RESULT_STRUCT);
102
103     //Объявление переменных
104     unsigned int count = lnh_get_num(RESULT_STRUCT);
105     unsigned int size_in_bytes = 4*count*sizeof(uint16_t);
106     //Создание буфера для приема пакета
107     uint16_t *buffer = (uint16_t*)malloc(size_in_bytes);
108     //Выборка минимального ключа
109     lnh_get_first(RESULT_STRUCT);
```

```
110     //Запись ключа и значения в буфер
111     for (int i=0; i<count; i++) {
112         buffer[2*i] = lnh_core.result.key;
113         buffer[2*i+1] = lnh_core.result.value;
114         lnh_next(RESULT_STRUCT,lnh_core.result.key);
115     }
116     //Запись глобальной памяти из RAM
117     buf_write(size_in_bytes, (char*)buffer);
118     mq_send(count);
119     free(buffer);
120 }
```