



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»

## Отчёт по практикуму №1 по курсу «Архитектура ЭВМ»

Тема Разработка и отладка программ в вычислительном комплексе

Тераграф с помощью библиотеки leonhard x64 xrt

Студент Ляпина Н.В.

Группа ИУ7-52Б

Преподаватель Дубровин Е.Н.

Москва — 2022 г.

# Оглавление

<b>Введение</b>	<b>3</b>
<b>1 Аналитическая часть</b>	<b>4</b>
1.1 Структура вычислительного комплекса Тераграф . . . . .	4
1.2 Принципы взаимодействия микропроцессора Леонард Эйлер и хост-подсистемы . . . . .	4
1.3 Взаимодействие CPE(riscv32im) и SPE(lnh64) . . . . .	5
<b>2 Ход выполнения работы</b>	<b>7</b>
<b>Заключение</b>	<b>13</b>

## 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 Аналитическая часть**

## **1.1 Структура вычислительного комплекса Тераграф**

Комплекс «Тераграф» предназначен для хранения и обработки графов сверхбольшой размерности и будет применяться для моделирования биологических систем, анализа финансовых потоков в режиме реального времени, для хранения знаний в системах искусственного интеллекта, создания интеллектуальных автопилотов с функциями анализа дорожной обстановки, и в других прикладных задачах. Он способен обрабатывать графы сверхбольшой размерности до  $10^{12}$  (одного триллиона) вершин и  $2 \cdot 10^{12}$  ребер. Комплекс состоит из 3-х однотипных гетерогенных узлов, которые взаимодействуют между собой через высокоскоростные сетевые подключения 100Gb Ethernet. Каждый узел состоит из хост-подсистемы, подсистемы хранения графов, подсистемы коммутации узлов, а также подсистемы обработки графов. Структурная схема одного узла представлена на рисунке ( 1.1).

## **1.2 Принципы взаимодействия микропроцессора Леонард Эйлер и хост-подсистемы**

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.

Если код программного ядра уже загружен в ОЗУ CPE, хост-подсистема может вызвать любой из содержащихся в нем обработчиков. Для этого в GPC передает оговоренный UID обработчика (handler), после чего передается сигнал запуска (сигнал START). В ответ CPE устанавливает состояние BUSY и начинает саму обработку. В ходе обработки ядро может обмениваться сообщениями с хост-подсистемой через очереди (команды mq\_send и mq\_receive). По завершении обработки устанавливается состояние IDLE и вырабатывается прерывание, которое перехватывается хост-подсистемой. Далее, пользовательское приложение хост-подсистемы уведомляется о завершении обработки и готовности результатов.

Если во время работы над кодом обработчика программному ядру software kernel требуется осуществить передачу больших блоков данных между CPE и хост-подсистемой, то может быть задействована Глобальная память и внешняя память большого размера (External Memory, до 16ГБ).

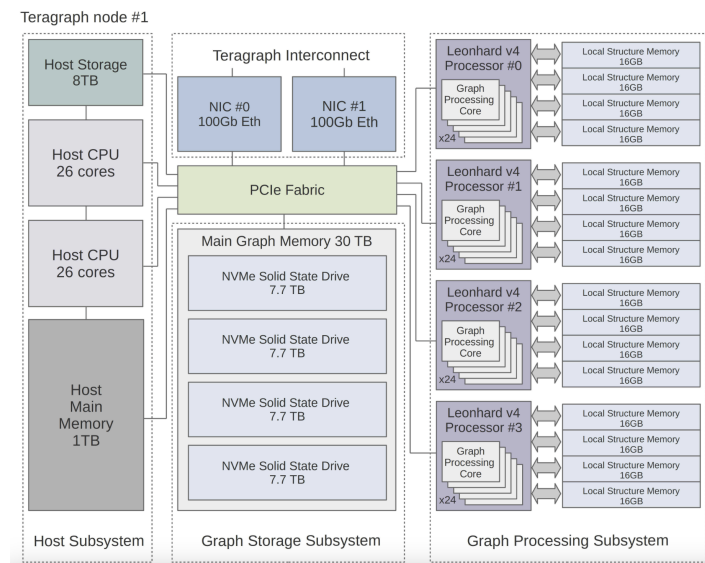


Рисунок 1.1 – Схема одного узла Телеграф

### 1.3 Взаимодействие CPE(riscv32im) и SPE(lnh64)

Микропроцессор lnh64 с набором команд дискретной математики (Discrete Mathematics Instruction Set Computer) является ассоциативным процессором, т.е. устройством, выполняющим операции обработки над данными, хранящимися в ассоциативной памяти (так называемой Локальной памя-

ти структур). В качестве таковой выступает адресная память DDR4, причем для каждого ядра lnh64 доступны 2.5 ГБ адресного пространства в ней. Для организации ассоциативного способа доступа к адресному устройству микропроцессор lnh64 организует на аппаратном уровне структуру В+дерева. Причем 512МБ занимает древовидная структура от верхнего и до предпоследнего уровня, 2048МБ занимает последний уровень дерева, на котором и хранятся 64х разрядные ключи и значения. Каждый микропроцессор lnh64 может хранить и обрабатывать до 117 миллионов ключей и значений.

Исходя из этого, обработка множеств или графов представляется в DISC наборе команд, как работа со структурами ключей и значений (key-value). Однако, как было показано ранее при описании набора команд DISC, в отличие от общепринятых key-value хранилищ, доступны такие операции как ближайший больший (NGR), ближайший меньший (NSM), команды объединения множеств (OR) и ряд других. Это и позволяет использовать lnh64 в качестве устройства, хранящего большие множества (для графов это множества вершин и ребер).

Доступ к микропроцессору lnh64 (Structure Processing Element) осуществляется чтением и записью в пространство памяти микропроцессора riscv32im (Computing Processing Element) в диапазоне 0x60000000 - 0x60001000.

## 2 Ход выполнения работы

Все задания практикума выполнялись по варианту 11.

Устройство формирования индексов SQL EXCEPT. Сформировать в хост-подсистеме и передать в SPE 256 записей множества А (случайные числа в диапазоне 0..1024) и 256 записей множества В (случайные числа в диапазоне 0..1024). Сформировать в SPE множество  $C = A \text{ not } B$ . Выполнить тестирование работы SPE, сравнив набор ключей в множестве С с ожидаемым.

В листинге 2.1 представлен код программы по индивидуальному варианту из файла *host\_main.cpp*. В листинге 2.2 представлен код программы по индивидуальному варианту из файла *sw\_kernel\_main.c*.

Листинг 2.1 – Код программы по индивидуальному варианту

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 int main(int argc, char** argv)
43 {
44
45     unsigned int cores_count = 0;
46     float LNH_CLOCKS_PER_SEC;
47
48     __foreach_core(group, core) cores_count++;
49
50     if (argc < 3) {
51         usage();
52         throw std::runtime_error("FAILED_TEST\nNo <xclbin> specified");
53     }
54
55     leonhardx64 lnh_inst = leonhardx64(0, argv[1]);
56     __foreach_core(group, core)
57     {
58         lnh_inst.load_sw_kernel(argv[2], group, core);
59     }
60
61     uint64_t *host2gpc_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
62     __foreach_core(group, core)
63     {
64         host2gpc_buffer[group][core] = (uint64_t*)
65             malloc(2*BURST*sizeof(uint64_t));
66     }
67     uint64_t *gpc2host_buffer[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
68     __foreach_core(group, core)
69     {
70         gpc2host_buffer[group][core] = (uint64_t*)
71             malloc(2*BURST*sizeof(uint64_t));
72     }
73
74     __foreach_core(group, core)
75     {
76         for (int i=0; i<BURST; i++)
77         {

```



```

76         uint64_t a = rand64()%1025;
77         uint64_t b = rand64() %1025;
78         host2gpc_buffer[group][core][2*i] = a;
79         host2gpc_buffer[group][core][2*i+1] = b;
80
81     }
82 }
83
84 __foreach_core(group, core) {
85     lnh_inst.gpc[group][core]->start_async(__event__(insert_burst));
86 }
87
88 __foreach_core(group, core) {
89     lnh_inst.gpc[group][core]->buf_write(BURST*2*sizeof(uint64_t),(char*)host2gpc_buffer[group][core]);
90 }
91
92 __foreach_core(group, core) {
93     lnh_inst.gpc[group][core]->buf_write_join();
94 }
95
96 __foreach_core(group, core) {
97     lnh_inst.gpc[group][core]->mq_send(BURST);
98 }
99
100 __foreach_core(group, core) {
101     lnh_inst.gpc[group][core]->start_async(__event__(search_burst));
102 }
103
104
105 unsigned int count[LNH_GROUPS_COUNT][LNH_MAX_CORES_IN_GROUP];
106
107 __foreach_core(group, core) {
108     count[group][core] = lnh_inst.gpc[group][core]->mq_receive();
109 }
110
111
112
113 __foreach_core(group, core) {
114     lnh_inst.gpc[group][core]->buf_read(count[group][core]*2*sizeof(uint64_t));
115 }
116
117
118 __foreach_core(group, core) {
119     lnh_inst.gpc[group][core]->buf_read_join();
120 }
121
122
123 bool error = false;

```

```

124
125     __foreach_core(group, core) {
126         for (int i=0; i<count[group][core]; i++) {
127             uint64_t elem = gpc2host_buffer[group][core][i];
128             for(int j = 0; j < BURST;j++)
129                 {
130                     uint64_t orig_key = host2gpc_buffer[group][core][2*j+1];
131                     if (elem == orig_key)
132                         error = true;
133                 }
134         }
135     }
136
137
138     __foreach_core(group, core) {
139         free(host2gpc_buffer[group][core]);
140         free(gpc2host_buffer[group][core]);
141     }
142
143
144
145     if (!error)
146         printf("????_???????_???????!\n");
147     else
148         printf("????_???????_?_???????!\n");
149
150
151     return 0;
152 }

```

Листинг 2.2 – Код программы по индивидуальному варианту  
sw\_kernel\_main.c

```

1
2 #include <stdlib.h>
3 #include <unistd.h>
4 #include "lnh64.h"
5 #include "gpc_io_swk.h"
6 #include "gpc_handlers.h"
7
8 #define SW_KERNEL_VERSION 26
9 #define DEFINE_LNH_DRIVER
10 #define DEFINE_MQ_R2L
11 #define DEFINE_MQ_L2R
12 #define __fast_recall__
13
14 #define A_STRUCTURE 1
15 #define B_STRUCTURE 2

```

```

16 #define R_STRUCTURE 3
17
18 extern lnh lnh_core;
19 extern global_memory_io gmio;
20 volatile unsigned int event_source;
21
22 int main(void) {
23     //////////////////////////////////////
24     // Main Event Loop
25     //////////////////////////////////////
26     //Leonhard driver structure should be initialised
27     lnh_init();
28     //Initialise host2gpc and gpc2host queues
29     gmio_init(lnh_core.partition.data_partition);
30     for (;;) {
31         //Wait for event
32         while (!gpc_start());
33         //Enable RW operations
34         set_gpc_state(BUSY);
35         //Wait for event
36         event_source = gpc_config();
37         switch(event_source) {
38             //////////////////////////////////////
39             // Measure GPN operation frequency
40             //////////////////////////////////////
41             case __event__(insert_burst) : insert_burst(); break;
42             case __event__(search_burst) : search_burst(); break;
43         }
44         //Disable RW operations
45         set_gpc_state(IDLE);
46         while (gpc_start());
47     }
48 }
49 }
50
51 //-----
52 // ???????? ????? ?? ?????????? ?????? ? ?????? ? lnh64
53 //-----
54
55 void insert_burst() {
56
57     lnh_del_str_sync(A_STRUCTURE);
58     lnh_del_str_sync(B_STRUCTURE);
59     lnh_del_str_sync(R_STRUCTURE);
60     unsigned int count = mq_receive();
61     unsigned int size_in_bytes = 2*count*sizeof(uint64_t);
62     uint64_t *buffer = (uint64_t*)malloc(size_in_bytes);
63     buf_read(size_in_bytes, (char*)buffer);

```

```

64     for (int i=0; i<count; i++) {
65         lnh_ins_sync(A_STRUCTURE,buffer[2*i],0);
66         lnh_ins_sync(B_STRUCTURE, buffer[2*i+1],0);
67     }
68     lnh_sync();
69     free(buffer);
70 }
71
72
73 //-----
74 // ????? ????????? lnh64 ? ?????? ? ?????????? ??????
75 //-----
76
77 void search_burst() {
78
79     lnh_sync();
80     lnh_not_sync(A_STRUCTURE, B_STRUCTURE,R_STRUCTURE);
81     unsigned int count = lnh_get_num(R_STRUCTURE);
82     unsigned int size_in_bytes = count*sizeof(uint64_t);
83     uint64_t *R = (uint64_t*)malloc(size_in_bytes);
84
85     for (int i=0; i<count; i++) {
86         R[i] = lnh_core.result.key;
87         lnh_next(R_STRUCTURE,lnh_core.result.key);
88     }
89
90     buf_write(size_in_bytes, (char*)R);
91     mq_send(count);
92     free(R);
93
94 }

```

В листинге 2.3 представлена команда для сборки проекта.

Листинг 2.3 – Код программы по индивидуальному варианту

sw\_\_kernel\_\_main.c

```

1 ./host_main leonhard_2cores_267mhz.xclbin sw_kernel_main.rawbinary

```

## Заключение

В данном практикуме были получены практические навыки решения задач обработки множеств на основе гетерогенной вычислительной структуры. В ходе практикума было проведено знакомство с типовой структурой двух взаимодействующих программ: хост-подсистемы и программного ядра `sw_kernel`.