Feasibility Study:

FPGA Parallel Heterogeneous Computing

https://github.com/yuwen41200/fpga-computing

Why FPGAs ?

- FPGA = Field Programmable Gate Array
- FPGAs are essentially massively parallel processors
- Fully customized hardware
  → Maximized optimization
- Past research has also showed that using FPGAs as accelerators can realize better performance than using GPGPUs
- If the data transmission interface is also PCIe, the cost is almost the same as GPGPUs

Drawbacks

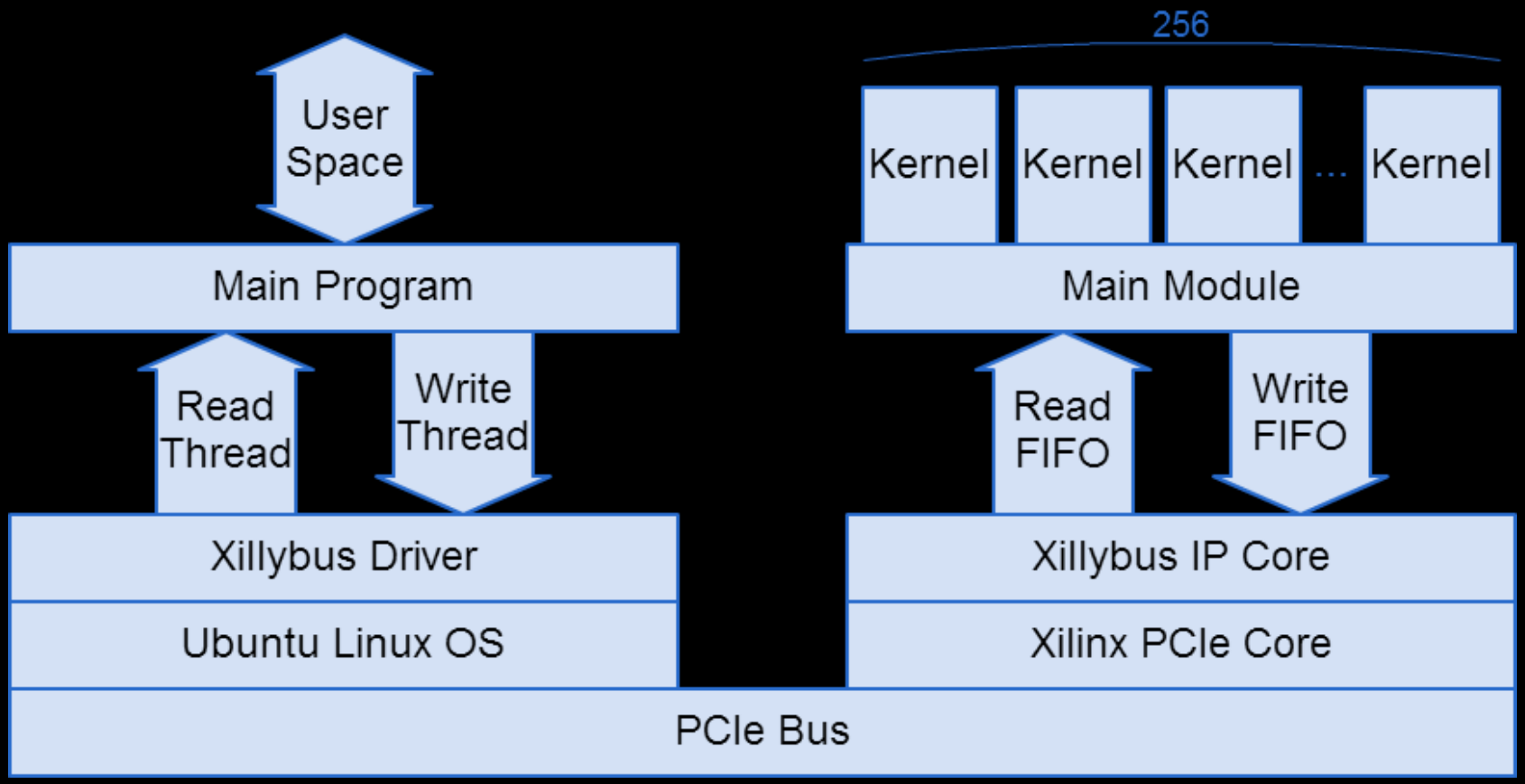
- Circuit design is hard, especially for experienced software developers
- The performance of the circuit is highly dependent on the quality of the circuit design and the available resources on the FPGA development board
- Though some EDA (electronic design automation) tools, like Xilinx SDAccel, support high-level synthesis that can convert OpenCL C/C++ codes into schematics, these techniques are not mature yet, and may lead to poor performance

Proposed Implementation

- Employ RTL (register-transfer level) design by the Verilog HDL (hardware description language)
- Use the Xillybus IP core for data transmission over the PCIe interface
- Targeted board: Xilinx Virtex-5 ML506 Evaluation Platform
- Use C++ for software design
- The host programs run on 64-bit Linux distributions

PC

FPGA


256
User Space
Kernel
Kernel
Kernel
...
Kernel
Main Program
Main Module
Read Thread
Write Thread
Read FIFO
Write FIFO
Xillybus Driver
Xillybus IP Core
Ubuntu Linux OS
Xilinx PCIe Core
PCIe Bus

Problem Statement

```
14  always @(posedge clk) begin
15      if (in_valid && counter == 0) begin
16          result <= in_data;
17          counter <= counter + 1;
18          out_valid <= 0;
19      end
20      else if (in_valid && counter < 1025) begin
21          result <= result + (result >> 3);
22          counter <= counter + 1;
23          out_valid <= 0;
24      end
25      else if (in_valid && counter == 1025) begin
26          out_valid <= 1;
27      end
28      else if (~in_valid) begin
29          counter <= 0;
30          out_valid <= 0;
31      end
32  end
```

$$color \times 1.125^{1024}$$

← Hardware Version

↓ Software Version

```
30      for (int i = 0; i < 1920; ++i)
31          for (int j = 0; j < 1080; ++j)
32              for (int k = 0; k < 1024; ++k)
33                  frame0[i][j] = frame0[i][j] + (frame0[i][j] >> 3);
```

Kernel Instantiation

```
283  generate
284      genvar thread_no;
285      for (thread_no = 0; thread_no < THREAD_NUMBER; thread_no = thread_no + 1) begin:warp
286          kernel kernel_ins (
287              .clk(bus_clk),
288              .in_data(in_data[thread_no]),
289              .in_valid(in_valid[thread_no]),
290              .out_data(out_data[thread_no]),
291              .out_valid(out_valid[thread_no])
292          );
293      end
294  endgenerate
```

FSM (Finite-State Machine)

```
191  always @(*) begin
192      case (curr_state)
193          IDLE_STATE:
194              if (user_w_write_32_open && user_r_read_32_open)
195                  next_state = RECV_STATE;
196              else
197                  next_state = IDLE_STATE;
198          RECV_STATE:
199              if (recv_counter == THREAD_NUMBER-2)
200                  next_state = EXEC_STATE;
201              else
202                  next_state = RECV_STATE;
203          EXEC_STATE:
204              if (exec_done)
205                  next_state = SEND_STATE;
206              else
207                  next_state = EXEC_STATE;
208          SEND_STATE:
209              if (send_counter == THREAD_NUMBER-2)
210                  next_state = IDLE_STATE;
211              else
212                  next_state = SEND_STATE;
213      endcase
214  end
```

Data Interface

```
 9  const char *readDeviceFile = "/dev/xillybus_read_32";
10  const char *writeDeviceFile = "/dev/xillybus_write_32";
11
12  int initRead() {
13      int fileDescriptor = open(readDeviceFile, O_RDONLY);
14      if (fileDescriptor < 0) {
15          if (errno == ENODEV)
16              fprintf(stderr, "Maybe %s is a write-only file.\n", readDeviceFile);
17          perror("Failed to open the device file for read");
18          exit(1);
19      }
20      return fileDescriptor;
21  }
```

Multithreading on Software

```
21  void sender() {
22      int fd = initWrite();
23      for (int i = 0; i < 1920; ++i)
24          fpgaWrite(fd, (unsigned char*) frame0[i], 2160);
25      for (int i = 0; i < 1920; ++i)
26          fpgaWrite(fd, (unsigned char*) frame1[i], 2160);
27      for (int i = 0; i < 1920; ++i)
28          fpgaWrite(fd, (unsigned char*) frame2[i], 2160);
29  }
30
31  void receiver() {
32      int fd = initRead();
33      for (int i = 0; i < 1920; ++i)
34          fpgaRead(fd, (unsigned char*) frame3[i], 2160);
35      for (int i = 0; i < 1920; ++i)
36          fpgaRead(fd, (unsigned char*) frame4[i], 2160);
37      for (int i = 0; i < 1920; ++i)
38          fpgaRead(fd, (unsigned char*) frame5[i], 2160);
39  }
```

Platform Specification

- **Serial:** Intel Core i5-3570 CPU @ 3.40GHz 4C4T, 8 GB Memory, Ubuntu 14.04 LTS 64-bit
- **CPU:** Intel Core i5-3570 CPU @ 3.40GHz 4C4T, 8 GB Memory, Ubuntu 14.04 LTS 64-bit, using up to 4 threads
- **GPGPU:** Intel Core i7-3770 CPU @ 3.40Ghz 4C8T, 16 GB Memory, Windows 10 Enterprise 64-bit, NVIDIA GeForce GTX 670
- **FPGA:** Intel Core i5-3570 CPU @ 3.40GHz 4C4T, 8 GB Memory, Ubuntu 14.04 LTS 64-bit, Xilinx Virtex-5 ML506 Evaluation Platform

Results

Data Size	Serial (Singlethread)	CPU Multithreading (OpenMP)	GPGPU Accelerator (CUDA)	FPGA Accelerator
Small (128 KiB)	0.179096 s	0.067667 s	0.891333 s	0.000534 s
Speedup	**1.000 x**	**2.647 x**	**0.201 x**	**335.386 x**
Large (~12 MiB)	15.873200 s	4.678667 s	1.799333 s	0.016104 s
Speedup	**1.000 x**	**3.393 x**	**8.822 x**	**985.668 x**

Attribution

Special thanks to Chun-Jen Tsai,
assistant professor at Dept. of C.S., NCTU