University of Pennsylvania
Department of Electrical and System Engineering
System-on-a-Chip Architecture

ESE532, Fall 2019

HW3: Thread Parallel

Wednesday, September 11

Due: Friday, September 20, 5:00PM

In this assignment, we will map the application from homework 2 on the ARM Cortex A53 cores of the Ultra96 platform. We will explore different parallel implementations and analyze their impact on performance.

Collaboration

You can find the partner assignment on Canvas in the *Partners* map under the *Files* section. In the event that the partner assignment does not work out, contact the instructor or TA as soon as possible. Partners may share code and results and discuss analysis, but each writeup should be prepared independently. Outside the assigned groups, only sharing of tool knowledge is allowed. See the course policies on the course web page http://www.seas.upenn.edu/~ese532 for full details of our policies for this course.

Communication

We will divide the work into threads that run on different processors. To distribute and coordinate the work, the processors must communicate. We will let the processors communicate via the SDRAM. This is possible because the SDRAM is mapped into the address spaces of both processors. In order to share data, the processors must agree on the location and organization of the data. In the provided code, we have mapped a small structure (`com_area`) at a fixed address in memory, which we use for communicating pointers to shared memory areas and synchronization. We must also make sure that both processors respect each other's private memory areas. We have already done that in the provided code by mapping private code and data of both processors at different locations. Sharing SDRAM is complicated by the fact that SDRAM is cached in L1 and L2 caches. Data that one processor attempts to write to SDRAM may not have been written to SDRAM yet, but instead remain in the private L1 cache of the processor. When another processor reads the same memory location, it may observe an old value. Fortunately, our Zynq has a Snoop Control Unit, which bypasses data directly between processors as needed to maintain a consistent view of the SDRAM. Therefore, this is no concern.

Another problem that we face when we communicate via shared memory is that the reading processor should not start reading the memory until the writing processor has completed writing the data. In other words, we need a form of synchronization between the cores. Design of synchronization functions is a rather complex subject, which is dealt with in other courses such as CIS 501, so we will just provide a few functions for this purpose without discussing their implementation:

- `Initialize` prepares a processor core for communication.
- `Wait_for_start` blocks processor core 1 until core 0 calls `Start_core_1`.
- `Start_core_1` stops a blocking `Wait_for_start` function on core 1.
- `Wait_for_core_1` blocks processor core 0 until core 1 calls `Return_to_core_0`.
- `Return_to_core_0` stops a blocking `Wait_for_core_1` function on core 1.

These functions have known shortcomings, but they should be sufficient for this assignment.

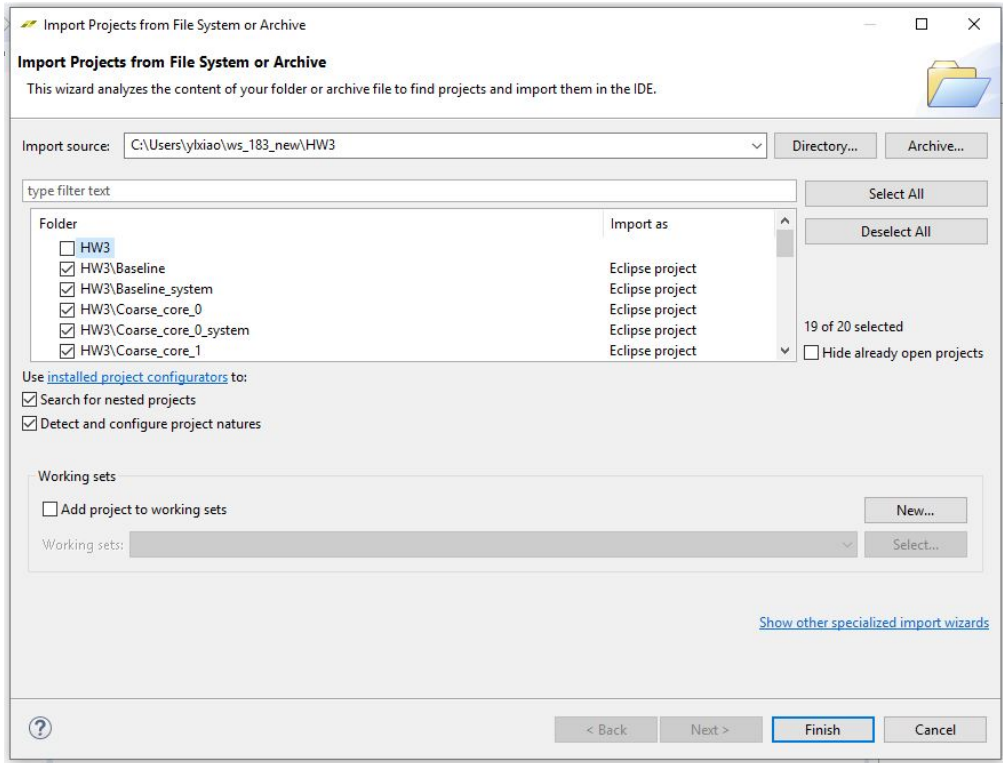
Note that you may have to adapt `Initialize` and `com_area` if you want to communicate more pointers between the processors. Other than that, you should not have to change these functions.

Obtaining and Running the Code

In the previous homework, we dealt with a streaming application that compressed only one picture. For this homework, we will use the same application, except that it will take a video stream instead of a single picture. The input and golden data is available here. Extract it and copy them into SD card. Download the project archive from here. Here's a video showing how to extract the code and setup the projects.

The parallel implementations, which start with `Coarse` and `Pipeline`, are split over two projects. Each project has the suffix `core_0` or `core_1`, which indicate on which ARM core they will run. We have provided debug configurations as well. Synchronization in the initialization function guarantees that core 1 does not start processing data before core 0 has finished initializing.

You may notice that there are several more projects, such as `ultra96V2_wrapper_hw_platform_0` and `Coarse_core_1_bsp`. The reason is that we created the application for core 1 as an application project because an SDSoC project does not allow us to adapt the memory layout of the application. We have to change the memory layout to prevent code and data for core 1 from overlapping with code and data for core 0. Every application needs a Board Support Package (BSP), which provides low-level routines to access the hardware. An SDSoC project automatically includes a BSP. For an application project, we have to create one manually. In addition, we need a platform description, which is in the `ultra96V2_wrapper_hw_platform_0` project.


Import Projects from File System or Archive

Import Projects from File System or Archive

This wizard analyzes the content of your folder or archive file to find projects and import them in the IDE.

Import source: C:\Users\ylxiao\ws_183_new\HW3

type filter text

Folder	Import as
☐ HW3	
☑ HW3\Baseline	Eclipse project
☑ HW3\Baseline_system	Eclipse project
☑ HW3\Coarse_core_0	Eclipse project
☑ HW3\Coarse_core_0_system	Eclipse project
☑ HW3\Coarse_core_1	Eclipse project

19 of 20 selected

☐ Hide already open projects

Use installed project configurators to:

☑ Search for nested projects

☑ Detect and configure project natures

Working sets

☐ Add project to working sets

Working sets:

Figure 1: Projects in the Workspace

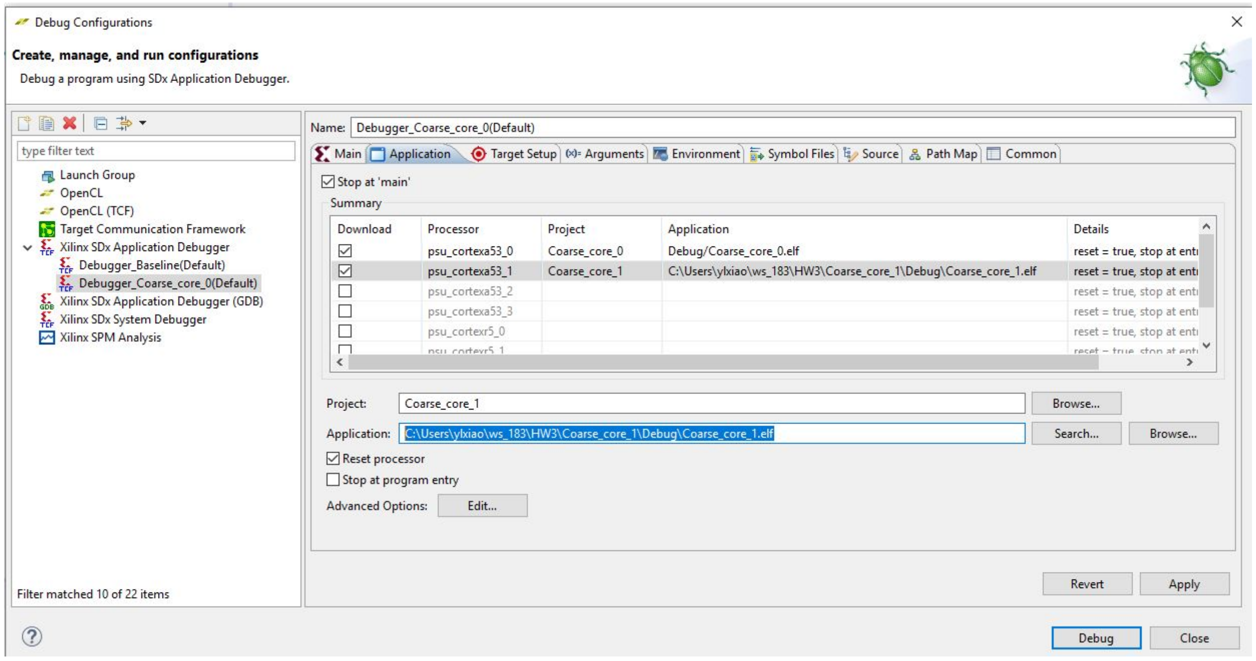
Homework Submission

Your writeup should follow http://www.seas.upenn.edu/~ese532/writeup_guidelines.pdf. Your writeup should include your answers to the following questions:

1. **Baseline** Extract the project archive, and click *File→Open Projects from File System..* Check all the project directories as Figure 1.

   (a) Determine the throughput of `Baseline` in pictures per second. This is your baseline. We use -O2 for the baseline, so you should keep using -O2 for the rest of the homework. Ignore overhead such as loading and storing pictures for this and the following questions. (1 line)

2. **Coarse-grain parallelism** We will parallelize the application by processing half of each picture on core 0 and the other half on core 1, a form of coarse-grain, data-level parallelism. There are two projects, `Coarse_core_0` and `Coarse_core_1` in the provided workspace that form the starting point of your implementation. We have parallelized `Scale` and some parts of `Filter` already for you. We use debug mode to run the 2 cores in parallel. Select *Run → Debug Configurations* from the menu. Configure Debugger_Coarse_core_0(Default) as Figure 2. In the *Application* tab, check *psu_cortexa53_1*, and choose the right ELF file for Core_1. Click apply and Debug.


Debug Configurations

Create, manage, and run configurations

Debug a program using SDx Application Debugger.

type filter text
Launch Group
OpenCL
OpenCL (TCF)
Target Communication Framework
Xilinx SDx Application Debugger
Debugger_Baseline(Default)
Debugger_Coarse_core_0(Default)
Xilinx SDx Application Debugger (GDB)
Xilinx SDx System Debugger
Xilinx SPM Analysis
Filter matched 10 of 22 items
Main
Application
Target Setup
Arguments
Environment
Symbol Files
Source
Path Map
Common
Browse...
Search...
Browse...
Edit...
Revert
Apply
Debug
Close

Name: Debugger_Coarse_core_0(Default)

☑ Stop at 'main'

Summary

Download	Processor	Project	Application	Details
☑	psu_cortexa53_0	Coarse_core_0	Debug/Coarse_core_0.elf	reset = true, stop at ent
☑	psu_cortexa53_1	Coarse_core_1	C:\Users\ylxiao\ws_183\HW3\Coarse_core_1\Debug\Coarse_core_1.elf	reset = true, stop at ent
☐	psu_cortexa53_2			reset = true, stop at ent
☐	psu_cortexa53_3			reset = true, stop at ent
☐	psu_cortexr5_0			reset = true, stop at ent
☐	psu_cortexr5_1			reset = true, stop at ent

Project: Coarse_core_1

Application: C:\Users\ylxiao\ws_183\HW3\Coarse_core_1\Debug\Coarse_core_1.elf

☑ Reset processor

☐ Stop at program entry

Advanced Options:

Figure 2: Debugger Configurations for 2 Cores

(a) Can we parallelize all streaming functions in our application, i.e. `Filter_horizontal`, `Filter_vertical`, `Differentiate`, and `Compress` in the same way as `Scale`? Motivate your answer. Assume that we synchronize our cores between each producer-consumer pair. (3 lines)

(b) What speedup do you expect from parallelizing the functions that you considered parallelizable in the previous question? [Include an equation for the expected parallel runtime and show the equation you use for computing the speedup as well as your final, numeric result.] (3 lines)

(c) Complete the implementation by parallelizing the functions that you considered parallelizable in the previous question. Provide the relevant sections of code in your report.

(d) Measure the throughput of your parallel implementation. (1 line)

(e) Validate your results. Make sure that your parallel version produces the same answers as the original serial version. Explain how you validated your results; report any discrepancies in your final implementation. (3–5 lines)

(f) Compare your measurement with your ideal, expected speedup. (1 line)

(g) If your speedup is different from ideal, expected, what effects are likely to be responsible for the difference? (1-3 lines)

3. **Pipelining** As an alternative to coarse-grain, data-level parallelism, we will investigate a pipelined implementation in this question. The initial implementation, which can be found in the projects `Pipeline_core_0` and `Pipeline_core_1`, maps `Scale` and

parts of `Filter` on core 1, and parts of `Filter`, `Differentiate` and `Compress` on core 0. The provided stream has only 10 frames, but assume in your performance computations that you are dealing with a stream of infinite length. You can run the 2 cores as the previous section.

(a) Report the throughput of the pipelined implementation in pictures per second. (1 lines)

(b) What is the best performance that one could theoretically achieve with a pipelined mapping of the streaming application?

(c) Describe the mapping that achieves the best performance.

(d) Reviewing the provided code, explain how it is able to deal with filling and draining the pipeline of operators? That is, when the application starts, there is only data for the first stage in the pipeline (`Scale`) and no data for the later stages. After the input data has been consumed by the `Scale` stage, the later stages will still have data to process. How does the code assure the program runs correctly to completion on all data? (4–6 lines)

(e) Review the provided code. Explain how you can adjust the *PIPELINE_PAR* parameter to maximize throughput. (2–3 lines)

(f) Adapt the implementation by changing the parameter *PIPELINE_PAR* in `Filter.c` to optimize the pipeline task or implement your own mapping to optimize the pipeline tasks. Include the sections of the code that you modified in your report.

(g) Validate that your results. Report on how you validated and any disrepancies. (1–3 lines)

(h) Report the throughput of your new application in pictures per second. (1–2 lines)

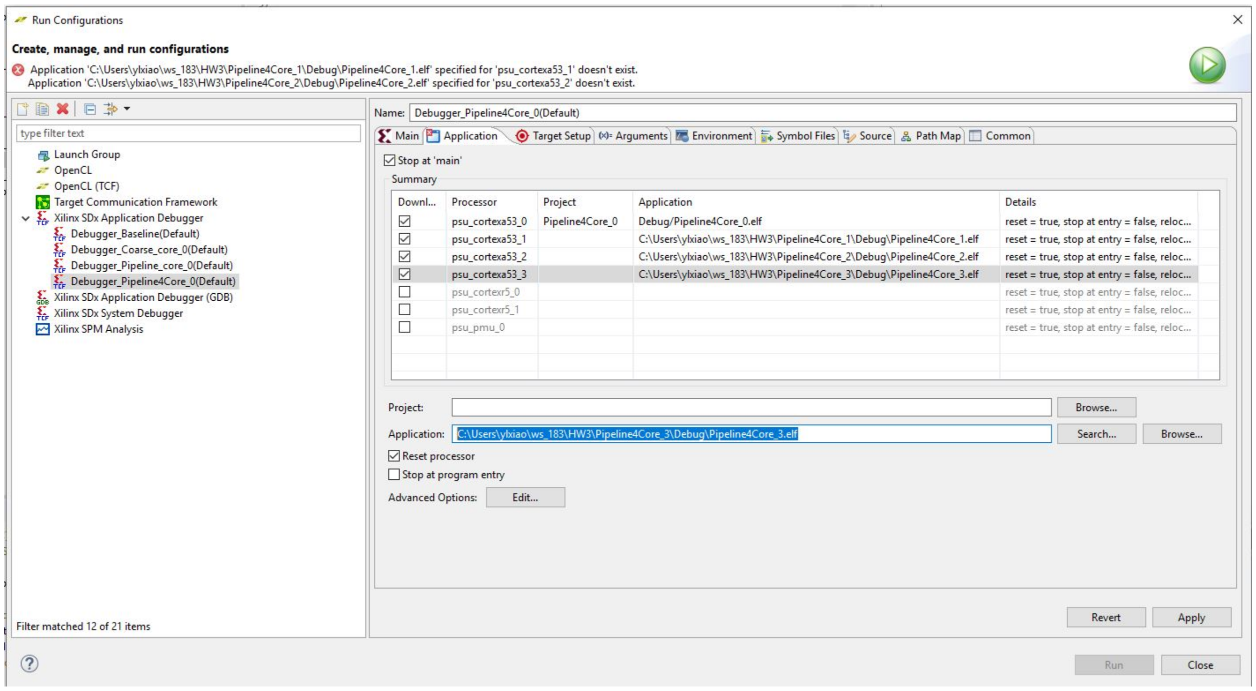
(i) Let's investigate the performance if we incorporate the optimized pipeline in a video broadcast server. The input data is read from an interface with 100 MB/s throughput. 75% of traffic is video traffic that is compressed using our pipeline (running on 2 processors). Assume the 2 cores can pipeline the process perfectly. The remaining 25% is other traffic that we protect with an error correction code (ECC) running on a dedicated hardware unit that adds 10% overhead in size. The hardware ECC unit processes 30 MB/s. The output of the ECC unit and compression pipeline are output to a single 2-Gigabit/s Ethernet port.

   i. Draw a streaming dataflow diagram for the network server. Indicate throughput and data transfer ratios where applicable.

   ii. What is the maximum throughput that the server can achieve? (10 lines)

   iii. Where is the bottleneck? (1 line)

   iv. How much smaller do we have to make the kernel (FILTER_LENGTH) of `Filter` to move the bottleneck? (7 lines)



Run Configurations

Create, manage, and run configurations

Application 'C:\Users\ylxiao\ws_183\HW3\Pipeline4Core_1\Debug\Pipeline4Core_1.elf' specified for 'psu_cortexa53_1' doesn't exist.
Application 'C:\Users\ylxiao\ws_183\HW3\Pipeline4Core_2\Debug\Pipeline4Core_2.elf' specified for 'psu_cortexa53_2' doesn't exist.

- Launch Group
- OpenCL
- OpenCL (TCF)
- Target Communication Framework
- Xilinx SDx Application Debugger
  - Debugger_Baseline(Default)
  - Debugger_Coarse_core_0(Default)
  - Debugger_Pipeline_core_0(Default)
  - Debugger_Pipeline4Core_0(Default)
- Xilinx SDx Application Debugger (GDB)
- Xilinx SDx System Debugger
- Xilinx SPM Analysis

Filter matched 12 of 21 items

Name: Debugger_Pipeline4Core_0(Default)

[✓] Stop at 'main'

Summary

Downl...	Processor	Project	Application	Details
✓	psu_cortexa53_0	Pipeline4Core_0	Debug/Pipeline4Core_0.elf	reset = true, stop at entry = false, reloc...
✓	psu_cortexa53_1		C:\Users\ylxiao\ws_183\HW3\Pipeline4Core_1\Debug\Pipeline4Core_1.elf	reset = true, stop at entry = false, reloc...
✓	psu_cortexa53_2		C:\Users\ylxiao\ws_183\HW3\Pipeline4Core_2\Debug\Pipeline4Core_2.elf	reset = true, stop at entry = false, reloc...
✓	psu_cortexa53_3		C:\Users\ylxiao\ws_183\HW3\Pipeline4Core_3\Debug\Pipeline4Core_3.elf	reset = true, stop at entry = false, reloc...
☐	psu_cortexr5_0			reset = true, stop at entry = false, reloc...
☐	psu_cortexr5_1			reset = true, stop at entry = false, reloc...
☐	psu_pmu_0			reset = true, stop at entry = false, reloc...

Project:

Application: C:\Users\ylxiao\ws_183\HW3\Pipeline4Core_3\Debug\Pipeline4Core_3.elf

[✓] Reset processor

[ ] Stop at program entry

Advanced Options: Edit...

Figure 3: Debugger Configurations for 4 Cores

4. **More Parallelism**

   Building on techniques and observations from previous parts, create a revised implemetation that uses all four 64b ARM Cortex A53 cores to achieve additional speedup. We have already offered you the 4-cores platform.[1] We add the sync functions in each core. However, we only move `Scale` into Core_2, move `Filter_horizontal` into Core_1 and use Core_0 to run the rest of the tasks in pipeline style. Set the right ELF files for each cores as shown in Figure 3. Try to achieve a 4× speedup over the single ARM core solution.

   (a) Describe your solution strategy (1 paragraph)

   (b) Include your code in your report.

   (c) Report speedup obtained and relate it to your solution. (3–5 lines)

   (d) Validate your design and report on any disrepancies.

[1]In the future we will provide some notes on platform creation. For now, the platform we provide you should be sufficient.