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

ESE532, Fall 2018 — Analysis Milestone — Wednesday, October 24

Due: Friday, Nov. 2, 5:00PM

Group: Forming your team, assessing the requirements and parallelism, and developing an initial functional implementation of the computation are group tasks for this milestone. You should discuss and refine your understanding of the task as a project team. You may share pseudocode and diagrams and collaborate on performance models within your team. You may not share or collaborate with anyone outside of your team.

Individual: Writeup is an individual task.

1. Report the partners you have agreed to work with for the project duration.
2. Specifically considering processing the input stream at 1 Gb/s:
   (a) Assuming hardware gathers up 64b words from the input stream, how many 667 MHz cycles do you have to process each 64b of data and maintain full throughput?
   (b) Assuming hardware gathers up 64b words from the input stream, how many 200 MHz cycles do you have to process each 64b of data and maintain full throughput?
3. Working from the high-level description of the computations involved, assess the computational and memory requirements for each of the coarse-grained operations (Content-Defined Chunking, SHA-256, chunk matching for deduplication, LZW encoding).
   (a) Summarize the computation of each coarse-grained operation in pseudocode. (write in your own words; credit sources)
   (b) What memory is needed to support each task? (Identify what functions the memory serves and the size for each of the memory components.)
   (c) What computational work is required per byte of input (or per chunk, where appropriate)? (How many adds, compares, multiplies, etc.)
   (d) What memory operations are required per byte of input (or per chunk, where appropriate)?
   (e) Based on this analysis and a simple model of processor execution (define as needed), what throughput can a single ARM processor achieve on this task? [This is a resource bound question.]

4. Identify and characterize parallelism available.

   (a) What parallelism exists among the operations in the coarse-grained task flow?

   (b) Within each operation, characterize opportunities for or inhibitions to thread-level data parallelism (i.e., what can be processed independently and what must be processed in a sequence).

   (c) Within an operation thread, what opportunity is there for data-level parallelism?

   (d) Within an operation thread, what opportunities exist for pipelined computations? What II is achievable? What dependencies determine the II? What is the depth of the pipelines?

   (e) What is the latency bound for the entire deduplication/compression flow for a single chunk working with a maximum chunk size of 8KB.

Questions 3 and 4 should give you enough information to begin reasoning about how you can achieve the throughput goals (1Gb/s, 10Gb/s). We are deliberately asking you to consider these computations from the high-level description rather than a particular piece of sequential code so that you can reason about the fundamental requirements and opportunities apart from the specific artifacts of a particular, sequential implementation. As you begin to develop and benchmark your solutions, you should refer back to your analysis here. Is your solution performing worse than your analysis would predict? Is this because your solution is inefficient or encounters some unanticipated bottleneck? or is this because your original analysis missed something?

5. **Sample Acceleration**

In this section, we will look at another application that has been restructured for HLS acceleration as an example of the kinds of restructuring necessary and the kinds of speedup possible. Specifically, we will look at the digit recognition Rosetta Stone Benchmark. You can find the full set of Rosetta Stone Benchmarks here. The digit recognition benchmark is in the digit-recognition subdirectory.

You will need to do this part of the assignment on linux. We provide you a set of files to use with the Rosetta Stone.

(a) Clone the Rosetta Stone git repository.
`git clone https://github.com/cornell-zhang/rosetta`

(b) In order to build and run the project, you need to follow the instructions. (All the instructions assume that you are working in the folder `digit-recognition`.)

i. Replace the `/src/host/digit_recognition.cpp` and `../harness/harness.mk` with files we provided.

ii. Set the enviorment variable `XILINX_SDX` to your SDx installation directory. You can use the following command:
`export XILINX_SDX=/opt/XILINX/SDx/2018.2`

iii. Source the SDx script using the command:
`source $XILINX_SDX/settings64.sh`

iv. Replace the training and test data sets with the training data we provide to make sure it can fit the ZedBoard. The training data set has 5400 data point and the testing has 400.

v. In the file `/host/src/typedef.h`, change the definition of the parameter K to 2 and PAR_FACTOR to 10.

vi. There are some functions with same name in `/src/sdsoc/digitrec.cpp` and `/src/sw/digitrec_sw.cpp`. You may want to change some of them to avoid multiple defination.

vii. Use the commands `make clean` and `make sdsoc` to build the project.

viii. Instead of using the GUI to launch the project on the ZedBoard, you could use sdx batch mode from command line. First you need to find your jtag_cable_name using the following command:
`sdx -batch -source ./List_cables.tcl | grep jtag_cable_name`
You will need to replace the jtag cable name in the `Launch_project.tcl` with yours.

ix. Use the edited TCL script (`Launch_project.tcl`) with the command to launch the bitstream and exectuable:
`sdx -batch -source ./Launch_project.tcl`

x. To receive the information sent back from ZedBoard, you can either use the GUI as before or use `cat` in a terminal. To use `cat` in a terminal:

- start a new terminal

- configure your serial port using:
  `stty -F /dev/ttyACM0 115200 raw cs8 -clocal icrnl`
- Use `cat` to look at the serial output received:
  `cat /dev/ttyACM0 | tee output.log`
- Launch your board.

(c) Examining the SDSoC version of the code, explain how the parameter `PAR_FACTOR` impacts the SDSoC mapping. (2-3 lines)

(d) With `PAR_FACTOR` set to 10 in `host/typedefs.h`, build and benchmark the SDSoC version. Report the speedup achieved. (1 line)

(e) Tune `PAR_FACTOR` so the design still fits on the ZedBoard platform while achieving as high a throughput as possible. Report the `PAR_FACTOR` used and the throughput achieved. Report what prevents you from using a higher `PAR_FACTOR`. (2-3 lines)

(f) Explain how the restructured SDSoC version achieves its speedup compared to the software version.

i. Identify how the code was restructured and why.
(The comments in the README can serve as general hints to you, but do not answer this question. For this answer, we want you to identify specific code changes and their rationale.)

ii. Explain how the FPGA-mapped version of the restructured code obtains its speedup. Be specific and quantitative. Write equations as appropriate to support your explanation.
(The code changes may not be 1:1 with speedup, so we're asking this piece separately. This is about describing how the resulting code is mapped to the hardware and how that results in the speedup obtained.)

As you see, there are several other Rosetta Stone benchmarks of varying complexity with SDSoC mappings. These can serve as further examples for you to illustrate how code is restructured, optimized, and tuned for acceleration on the Zynq SoC platform.