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

ESE5320, Fall 2023

Analysis Milestone

Wednesday, October 23

Due: Friday, Nov. 1, 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.

For this milestone, writeup is a group task.

1. Report the partners you have agreed to work with for the project duration.
2. Specifically considering processing the input stream at 400 Mb/s:
   (a) Assuming hardware gathers up 64b words from the input stream, how many 1.2 GHz 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. According to the AMD/Xilinx White Paper `https://www.xilinx.com/support/documentation/white_papers/wp512-accel-crypto.pdf`, what throughput should a single ARM core provide performing SHA3-384 on 8KB blocks?
4. 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, 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) What data must be communicated between coarse-grain operations?

(f) Assuming the input arrives at 400 Mb/s, what are the data rates between coarse-grain operators (including SHA)?

(g) Based on this analysis and a simple model of processor execution (define as needed), what throughput (in Mb/s) can a single ARM processor achieve on this task? [This is a resource bound question.]

5. Identify and characterize parallelism available.

   (a) What parallelism exists among the operations in the coarse-grained task flow (Content-Defined Chunking, SHA, chunk matching for deduplication, LZW encoding)?

   (b) Within each operation (Content-Defined Chunking, chunk matching for deduplication, LZW encoding), 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 (Content-Defined Chunking, chunk matching for deduplication, LZW encoding), what opportunity is there for data-level parallelism?

   (d) Within an operation thread (Content-Defined Chunking, chunk matching for deduplication, LZW encoding), what opportunities exist for pipelined computations? Is an II of 1 achievable? What dependencies prevent an II of 1?

Questions 4 and 5 should give you enough information to begin reasoning about how you can achieve the throughput goals (400 Mb/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?

6. Develop placeholder encoder solution in C.
   (a) Run on single ARM Cortex A53 processor. Use the Ultra96 platform from the recent homeworks.
   (b) Work with sample input files we've provided and the provided Decoder.
   (see project handout.)
   (c) Remember that you implemented CDC on Homework 2; you will likely need to change how this communicates with the other components.
   (d) Focus on decomposing into components and how those components communicate with each other.
   - What data structures, data format, and data sequence do you send between components?
   - What data rates are required between components?
   - What is the unit of data you send between components? (What composes a unit of data? how many bits?)
   - What synchronization is required?

   (e) To scope this down, do **not** focus on full functionality of the components. Instead look for minimal, placeholder functionality. These should be simple—as simple as possible to handle the data. As a result, they will likely not be particularly good (e.g. poor compression) or fast (e.g. slow algorithms).
   - For example, you might use addition modulo 2^{64} as a placeholder hash for SHA for initial integration as a placeholder to be replace with the proper SHA-256 later.
     (using a simplistic hash like this will require that you check for false matches.)
   - Reminder: we have pointed you to NEON-based SHA software versions you might consider using.
   - Hint: how can you minimize the initial functionality of each of the other components?
   - The idea is that we will refine these with proper functionality for the next milestone. This helps enable you to always have something that works and allows you to work on components independently.

   (f) Include a description in your report of:
   - your interfaces between components
   - your placeholder functions (what they do, how they are simplified)
   - status of the code – what overall functionality does it provide

   (g) Turn in a tar file with your minimal functionallity code to the designated assignment component in canvas.