CS 560: Homework 3: Advanced Foundations

S. Rajopadhye, Colorado State University

The goal of this assignment is to increase your familiarity with equational reasoning/programming, to extend our methods for manipulating affine functions and domains to equational programs using a Change-of-Basis (CoB) transformation. You will use this to derive by hand, a parallel hardware sorter (the same one we saw in the first lecture, but with a small twist).

You should be able to (i) figure out a legal schedule and processor allocation, (2) use that to build a CoB, (3) and apply the CoB to the SURE and rewrite it as a new SURE

Problem I: AlphaZ [30 pts]

In the AlphaZ system, after the "Checking out Sample Project" step, create a new folder called HW3, and develop your Alpha/AlphaZ answers in this directory—one subdirectory per sub problem. You should have a separate test-out directory within HW3, so it is submitted with your code.

- **Subproblem 1: LUD tutorial** [10 pts]

  Complete the LUD tutorial and understand everything—all the steps through code generation and testing, as well as the array notation vs standard notation. You must finish this by start of class Thursday September 17. No need to submit anything, we can find out when you get it done it.

- **Subproblem 2: Fibonacci** [20 pts]

  1. Write an Alpha program with a parameter N that computes the N-th Fibonacci number but returns it as a double (we want to test it out on large inputs, and want to avoid integer overflow exceptions). This is different from the one that was checked out and done in the recitation. That one returned an array of the first N Fibonacci numbers. Generate code and test it out on a number of values, by hand. [5 pts]
  2. Carefully study the Makefile generated and the wrapper generated and be prepared to answer questions about it. In particular you should know all the compile time flags (search for lines with `#ifdef`) that can be passed to it, and all the use cases of the produced code. Experiment with it, and in your report, write a short summary of the options. [5 pts]
  3. Write a function Fib_verify in a file Fib_verify.c that has exactly the same signature as the AlphaZ-generated one. It should also compute the N-th Fibonacci number, but using the standard recursive definition (as in the recitation). Use this function and the "make verify" command to build a program that checks the correctness of the AlphaZ generated code. [5 pts]
  4. Once you are sure that the code works for a decent range of values, empirically study the asymptotic complexity of the generated code and report your result. To do this, you may have to modify the wrapper, since the program is "too fast." [5 pts]

Problem II: (Manipulating Equations [60 pts] We are going to define an SURE which computes (a part of) the sorting algorithm and derive a parallel hardware implementation for it. Let $D_1 = \{i, j \mid 0 \leq i < P \wedge 0 \leq j < N - i\}$, and $D_2 = \{i, j \mid 0 \leq i < P \wedge 0 \leq j < N - i - 1\}$. And assume that $P < N$. Eqns 1–2 below define an SURE with two variables as follows:

$$X[i,j] \;=\; \begin{cases} D_2 \cap \{|i = 0\} & : \; \min(a_{N-j-1}, Y[i, j+1]) & (\texttt{left}) \\ D_2 \cap \{|i > 0\} & : \; \min(X[i-1, j], Y[i, j+1]) & (\texttt{rest}) \end{cases} \tag{1}$$

$$Y[i,j] \;=\; \begin{cases} D_1 \cap \{|i = 0 \wedge j = N-1\} & : \; a_0 & (\texttt{top left corner}) \\ D_1 \cap \{|i = 0 \wedge j < N-1\} & : \; \max(a_{N-j-1}, Y[i, j+1]) & (\texttt{left}) \\ D_1 \cap \{|i > 0 \wedge i + j = N-1\} & : \; X[i-1, j] & (\texttt{top} - \texttt{diagonal}) \\ D_1 \cap \{|i > 0 \wedge i + j < N-1\} & : \; \max(X[i-1, j], Y[i, j+1]) & (\texttt{interior}) \end{cases} \tag{2}$$

The final goal of this Problem is to systematically parallelize this SURE. Our parallelization may eventually be implemented as a hardware accelerator circuit or even as an OpenMP program, but that is not important here.

1. Draw a neat dependence graph in this SURE for $N = 12, P = 4$. Make sure you label the axes, mark the vertices of the domains of each variable, and clearly show the inputs and outputs. Since there are two variables, X and Y, at each index point, $[i, j]$, you should use a clear convention (e.g., colors, or shapes) to distinguish between the two. Show the dependences in the graph (to avoid clutter, you may show just a few instances). [15 pts]

2. For parts 2-4, you may view the entire computation (of both X and Y) at an index point, $[i, j]$ by a single node. Analyze the dependences and choose linear/affine functions for the schedule and processor allocation. Illustrate these functions on your diagram by drawing families of lines through the graph. [15 pts]

3. Write down (in the syntax of Alpha dependences, standard notation, as well as in matrix form) these two functions and their combination as a space-time transformation T. [5 pts]

4. Apply a CoB transformation by T to the SURE, in a "pictorial/intuitive way:" Redraw/sketch the transformed dependence graph (use a single node at each index point), the transformed dependences, and rewrite the SURE using the pictorial intuition of where the different subregions have "moved." [15 pts]

5. Now we will do the more extended CoB using polyhedral machinery. To do this, we will apply the transformation $(i, j \rightarrow i, i + j)$ to *only* the variable X in the SURE of Eqns 1–2. In order to get full credit, please show your work: how did you compute each dependence and the domains of each branch of each case in the transformed SURE. [10 pts]

Report [10 pts] Write a short summary of what you learnt in doing this assignment. Submit everything as a single pdf file called `HW3.pdf` via Checkin.