CS 560: Homework 6: Hand Parallelization in AlphaZ

S. Rajopadhye, CSU. Due Thursday Oct 29, 2:00 pm

The goal of this assignment is to develop the skills to analyze polyhedral programs, determine (i) schedules, (ii) the space-time mapping, the memory map, and to Use AlphaZ to generate sequential and parallel code. You will also compare the performance of the code to manually parallelized codes.

Problem I: HW0 Revisited **[30 pts]**

Consider the program that you saw in HW0: `syr2k.c`.

1. First of all, study the code of the function `kernel_syr2k` and write it as an Alpha program. Generate demand-driven code, and study the execution time for a range of "large enough values," In your report, we would like to see plots for a *single independent parameter* using an appropriate scale, so you should use a few specific values of the "aspect ratio" (e.g., $N = M$, $N = 2M$, $N = 5M$, $M = 100$, etc.)

2. Next, generate different *sequential programs*, and compare the performance with the best one that you wrote by hand (in HW0). To do this, you will write and execute AlphaZ scripts that appropriately choose a *space-time map*, a *memory map*, and invoke the *scheduled-C* code generator. You should explore different possibilities, including the ones you tried for HW0, but also other schedules—since CoB is more general than just loop permutations.

   Be choosy in what you report—no need to show a plot just because you have the data. What is the story that you want to tell? Is there a message? How many different plots do you need to tell the story?

3. Next generate parallel code by using the set-parallel command, and explore the performance of different parallelizations. In your report, you should plot the execution time, for a varying number of processors/threads, but, the independent variable should be the problem size parameter, N: we are, first and foremost, even before speedup, interested in asymptotic behavior of the program (scalability). After you are convinced that the scalability is good, select an appropriate slice of the data for which you will show us the speedup, this time, with the number of processors/threads as the independent variable.

4. Use equational reasoning to reduce the complexity by a factor of two.[1] Repeat the previous code generation steps for improved program and report your observations.

Problem II: Simple 2D Jacobi Stencil **[30 pts]**

Many of you have written for HW1 that you parallelized a simple Jacobi stencil computation and obtained "good" performance on it. We are going to revisit this and leant a few thing: how to systematically generate these parallel programs using AlphaZ, how to quantify the performance and to understand what good really means, and how to compare it to the "machine peak." Repeat the process that you did for *Problem I* for the 2-D Jacobi stencil. Please use the code out of the most recent release of Polybench (4.0). Follow the same or similar steps:

[1] Hint: You don't need anything more sophisticated than the sophomore level linear algebra (e.g., MATH229). Your program should then run twice as fast as the original "for free."

1. First, study the C code and write the kernel as an Alpha program. Generate demand-driven code, and check whether it provides the same results as the code in Polybench (you may use the verify-rand option if you like. Then study the execution time. In your report, we would like to see plots for a *single independent parameter* using an appropriate scale, so you should use a few specific values of the "aspect ratio" (e.g., $N = T$, $N = 2T$, $2N = T$, $T = 100$, etc.)

2. Next, generate different *sequential programs*, and compare their relative performance. To do this, write and execute AlphaZ scripts that appropriately choose a *space-time map*, a *memory map*, and invoke the *scheduled-C* code generator. As before, your report should first demonstrate scalability by plotting the execution time for a varying number of processors/threads, but, with the problem size parameter, N as the independent variable. Only then should you select an appropriate slice of the data for which you show the speedup.

Problem III: UTMI: (Unit) Upper Triangular Matrix Inversion **30 pts]**

Do the same thing for the triangular matrix inversion program/algorithm for an upper triangular matrix, with unit diagonals. Write the Alpha program, generate demand-driven code, check that it works correctly (say, by simply confirming that the product of the answer and the input is indeed the identity matrix), and then keeping this system aside as a baseline (the `_verify.c` file). Now write the serialized version of the Alpha program, choose appropriate target and memory mappings, both sequential and parallel, explain/argue why they are legal, and compare their performance.

Report Write your report using the guidelines above. **10 pts]**