CS 610 Semester 2020–2021-I: Assignment 2

21th September 2020

Due Your assignment is due by Sep 30 2020, 11:59 PM IST.

General Policies

- You should do this assignment ALONE.
- Do not plagiarize or turn in solutions from other sources. You will be PENALIZED if caught.
- We MAY check your submission(s) with plagiarism checkers.

Submission

- Submission will be through mooKIT.
- Write your programs in C++ .
- Submit a compressed file with name "<roll-no>.tar.gz". The compressed file should have the following structure.

  ```
  -- roll-no
  -- -- report.pdf
  -- -- <problem4-dir>
  -- -- -- <source-files>
  -- -- <problem5-dir>
  -- -- -- <source-files>
  ```

  Submit a PDF file with name "report.pdf". We encourage you to use the LaTeX typesetting system for generating the PDF file. The file should include your name and roll number, and results and explanations for the following problems. Include the exact compilation instructions for each programming problem, for example, `g++ -O3 problem4.cpp -pthread`, and any other additional information and assumptions that you think will help the evaluation.
- You will get up to TWO LATE days to submit your assignment, with a 25% penalty for each day.

Evaluation

- Please write your code such that the EXACT output format (if any) is respected.
- We will evaluate your implementations on a Unix-like system, for example, a recent Debian-based distribution.
- We will evaluate the implementations with our OWN inputs and test cases, so remember to test thoroughly.

Problem 1 **[10 marks]**

Consider the following code:

```
for i = 1, N-2
  for j = i+1, i+N-2
  A(i, i-j) = A(i, i-j-1) - A(i-1, i-j) + A(i+1, i-j+1)
```

List all flow, anti, and output dependences, if any, using the Delta test. Assume all array subscript references are valid.

Problem 2 **[70 marks]**

Consider the following code:

```
  int i, j, t, k;
  for (t = 0; t < 2048; t++) {
    for (i = t; i < 1024; i++) {
      for (j = t; j < i; j++) {
        for (k = 1; k < j; k++) {
          S(t, i, j, k);
        }
      }
    }
  }
```

The data dependences for the loop are given to be (1,0,-1,1), (1,-1,0,1), and (0,1,0,-2).

(a) Which loops, if any, are valid to unroll? Why?

(b) What are valid permutations of the loop? Why?

(c) What tiling is valid, if any?

(d) Which loops, if any, are parallel?

(e) Show code for the *tikj* form of the code. For this part, ignore the above dependences and assume *tikj* permutation is allowed.

Problem 3 **[50 marks]**

Consider the following code:

```
#define N 4096
double A[N][N];
int t,i,j;
for (t = 0; t < N-1; t++) {
  for (i = 1; i < N-1; i++) {
    for (j = 1; j < N-1; j++) {
      A[i][j] = 0.2*(A[i-1][j] + A[i][j] + A[i+1][j] + A[i][j-1] + A[i][j+1]);
    }
  }
}
```

(a) List all data dependences, stating the kind of dependence and the distance vector.

(b) What permutations (if any) are valid? Why?

(c) Which loops (if any) are valid to unroll? Why?

(d) What tiling (if any) is valid? Why?

Problem 4 **[50 marks]**

Write a Pthread program to simulate a customer-unfriendly bank system! Assume the bank can service two customers concurrently at a time. There are two tellers in the bank, and each teller handles one customer request ata time.

Assume a pool of `N` customers who arrive at almost the same time and compete to get service from the tellers. Each customer gets a monotonically increasing token upon entry to the bank. *After* getting a token, the customer takes her place in a shared FIFO queue in the waiting area. Two customers get serviced at a time, and the others wait for their turn in the queue. The next customer to get serviced is selected from the head of the queue.

A bank transaction is represented by a customer *atomically* writing eight integer values (her token) to a shared array (of size 16). A teller services a customer by atomically reading the eight numbers from the shared array, prints the numbers to standard out in the same order, and takes a break for five seconds. Once done, the lazy teller services the next customer thread if any.

The employee-friendly bank shuts down for the day when there are no more customers.

Assumptions. Your implementation should meet the basic correctness requirements (no data races, no atomicity violations, and no deadlocks).

- Assume the bank processes 24 customers a day.
- Your program should take the number of customers as an argument to the program.
- We do not have unruly and impatient customers! A customer does not overwrite another customer's request.
- A teller does work as long as there are customers.
- Do not explicitly control the schedule of consumer threads, it is fine to have nondeterminism because of synchronization (for example, the enqueue order of customers can vary and need not be in the order of the tokens).

Problem 5 **[50 marks]**

Assume a naïve O(n^3) sequential matrix multiplication kernel (*ijk* form). Write a program to introduce parallelism in the kernel for performance improvement.

(a) Assume the matrices are of dimensions 4096×4096. Report speedup numbers for 2, 4, and 8 threads over the sequential implementation for the same matrix dimensions. This is a study of strong scaling (i.e., problem size remains the same but you increase compute resources).

Your report should include speedup plots, x-axis should be the number of threads, and y-axis should be the speedup compared to the sequential version.

(b) Implement optimized versions of your sequential and parallel matrix multiplication implementations. You are free to apply any valid data partitioning and optimization strategies to get the BEST PERFORMANCE FOR YOUR SETUP. For example, you can apply blocking and unrolling. You are free to choose which loops to block and unroll, and the block dimensions and loop unrolling factors.

Experiment with reasonable block sizes (for e.g., 4×4 or 64×64), and report the best performing variant that you find. Compare the results with your parallel threaded implementation. Include a brief justification for the results you obtained. Note that the best performing configurations could vary across different architectures.

Assumptions.

- Populate your matrices with all ones of type `uint64_t`.
- Create separate functions for your optimized kernels for ease of evaluation and debugging.