Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Parallel Programming
Assignment 4: Parallel Models
Spring Semester 2017

Assigned on: **14.03.2017** Due by: **20.03.2017**

Overview

This assignment aims to introduce you to basic theoretical concepts of parallel programming. This exercise does not include a programming part. Please hand in your solutions by uploading a file called `report.pdf` into your SVN repository as described in the Submission section.

Task 1 – Amdahl's and Gustafson's Law

Assuming a program consists of 50% non-parallelizable code.

a) Compute the speed-up when using 2 and 4 processors according to Amdahl's law.

b) Now assume that the parallel work per processor is fixed. Compute the speed-up when using 2 and 4 processors according to Gustafson's law.

c) Explain why both speed-up results are different.

Task 2 – Pipelining

Bob, Mary, John and Alice share a flat. In this flat they share a washing machine, a dryer and an ironing board. The washing machine takes 50 minutes for one wash cycle. The dryer takes 90 minutes. Everyone of them takes roughly 15 minutes to iron their laundry.

a) Assuming they would do their laundry in strictly sequential order (one person starts only after the other finished ironing), calculate how long would it take to finish the laundry?

b) Are there any better options? If yes, describe them and calculate the improved laundry time, as well as the speedup of this version compared to the strictly sequential version

c) Can you devise a better strategy assuming that the four roommates bought another dryer? If yes, calculate the new laundry time

Task 3 – Pipelining II

Instruction pipelining is a form of parallelism (also called instruction level parallelism) commonly used to improve program performance. The main idea is that the execution of multiple instructions can be partially overlapped which leads to reduction of the overall time required to complete the execution.

a) Let us consider a following for loop:

```
for (int i = 0; i < data.length; i++) {
    data[i] = data[i] * data[i];
}
```

Assume that the multiplication has a throughput 1 instruction per cycle and latency 6 cycles. That is, the processor can in each cycle start executing single new multiplication instruction and it will take the processor 6 cycles to compute the result. Further, consider only arithmetic operations performed in the loop body (i.e., in this case `data[i] * data[i]`) and ignore all other operations (e.g, storing the result, incrementing the loop counter, evaluating loop termination condition). However, all the computation in the loop body must be finished in order to start new loop iteration.

Calculate how many cycles the processor needs to execute the loop a) given the above simplifying assumptions.

b) Let us consider a different loop that calculates the same result as the loop in a), assuming that the array length is divisible by two:

```
for (int i = 0; i < data.length; i += 2) {
    j = i + 1;
    data[i] = data[i] * data[i];
    data[j] = data[j] * data[j];
}
```

Assume that the addition has a throughput 1 instruction per cycle and latency 3 cycles. Note that it is allowed for the processor to issue multiplication instruction even if addition instruction is still being computed and vice versa (i.e., we can issue addition in first cycle followed by multiplication in cycle 2). Calculate how many cycles the processor needs to execute the loop b).

c) Finally, let us consider following loop that calculates the same results as loops in a) and b), assuming that the array length is divisible by four:

```
for (int i = 0; i < data.length; i += 4) {
    j = i + 1;
    k = i + 2;
    l = i + 3;
    data[i] = data[i] * data[i];
    data[j] = data[j] * data[j];
    data[k] = data[k] * data[k];
    data[l] = data[l] * data[l];
}
```

Calculate how many cycles the processor needs to execute the loop c).

Note: The above optimization is called loop unrolling and is typically performed automatically by the compiler. You should not write such code manually unless you are writing performance critical code and are sure that the compiler cannot perform this optimization automatically.

Task 4 – Identify Potential Parallelism

a) Inspect the code snippets of the two `for` loops below. For each loop explain if it is OK to use a parallel `for` loop instead.

Note: A parallel `for` loop is a parallel programming construct covered in Exercise 2 that allows different loop iterations to be performed in parallel. This should not be confused with instruction level parallelism.

(a) Loop-1

```
for (int i=1; i<size; i++) {    // for loop: i from 1 to (size-1)
    if (data[i-1] > 0)          // If the previous value is positive
        data[i] = (-1)*data[i]; // change the sign of this value
}                               // end for loop
```

(b) Loop-2

```
for (int i=0; i<size; i++) {      // for loop: i from 0 to (size-1)
    data[i] = Math.sin(data[i]);  // calculate sin() of the value
}                                 // end for loop
```

Submission

This exercise does not include a programming part. Please hand in your solutions by uploading a file called `report.pdf` into your SVN repository as described below:

- **Check-in your project for the first time**
  - Create new empty project (**New** → **Other...** → **General** → **Project**) called **assignment4**.
  - Put all your written answers in a single file named `report.pdf` and place it in the base directory of the project.
  - Right click your created project called **assignment4**.
  - In the menu go to **Team**, then click **Share Project**.
  - In the dialog that now appears, select **SVN** as a repository type, then click **Next**.
  - In case you have submitted Exercise 1, choose **Use existing repository location** and select the pre-defined URL in the dialog that should look like this
    https://svn.inf.ethz.ch/svn/vechev/pprog17/students/NETHZ_USERNAME
    Click Finish. Otherwise follow the steps in Exercise 1 to set-up a repository location.
- **Commit changes in your project**
  - Now that your project is connected with the SVN server, you need to make sure that every time you change your code or your report, at the end you submit it to the SVN server as well.
  - Right click your project called **assignment4**.
  - In the menu go to **Team**, then click **Commit**.
  - In the Comment field, enter a comment that summarizes your changes.
  - Then, click on **Ok**.