Laboratory 6

(Due date: Nov. 9th)

OBJECTIVES

- Compile and execute C++ code using the TBB library in Ubuntu 12.04.4 using the Terasic DE2i-150 Development Kit.
- Execute applications using TBB: *parallel_pipeline*

REFERENCE MATERIAL

- Refer to the board website or the Tutorial: Embedded Intel for User Manuals and Guides.
- Refer to the Tutorial: High-Performance Embedded Programming with the Intel® Atom™ platform → *Tutorial 7* for associated examples.

ACTIVITIES

FIRST ACTIVITY: COMPUTATION ON A STREAM OF INCOMING VECTORS (100/100)

- Given a vector $\vec{x}$, we want to transform it into a vector $\vec{r}$, whose elements are specified by:

$$r(i) = \frac{1}{1 + e^{-x(i)}}, i = 0, \dots, n-1$$

- Then, we want to get the minimum value out of all the elements in vector $\vec{r}$, i.e. $result = min(\vec{r})$

- This is a relatively simple vector operation. However, in a real application, we want to process a set of incoming vectors. This is depicted in Fig. 1. For every incoming vector $\vec{x}$, we compute $\vec{r}$, and then get the minimum value (the results are stored in an array).

Figure 1. Description of application: A set (NV) of incoming vectors (each vector represented by $\vec{x}$) is processed by the software application. Out of each vector, one scalar result is generated. The final result is a vector of NV elements.

- A straightforward sequential implementation can then be quickly implemented. But if we can divide the computations into a linear sequence of stages (operation on vector $\vec{x}$, and then get the minimum out of vector $\vec{r}$), we might be able to take advantage of pipelining. This allows multiple vectors to be in different stages of processing at the same time. This overlapping of computations might lead to overall execution time improvements when we process a stream of incoming vectors.

INSTRUCTIONS

- In this experiment, you are asked to implement a pipeline (via TBB *parallel_pipeline*) for the application of Fig. 1.
  - ✓ To simplify the problem, we assume that the incoming vectors are extracted from a matrix.
- You are asked to implement a serial-parallel-serial pipeline (3 stages) as depicted in Fig. 2.
  - ✓ First Stage: It receives items (vectors) and feeds them into the pipeline.
  - ✓ Second Stage: It generates the vector $\vec{r}$. This is a parallel stage since the computations are relatively complex.
  - ✓ Third Stage: It gets the minimum value of the vector $\vec{r}$ and places them in an output array `ci`.

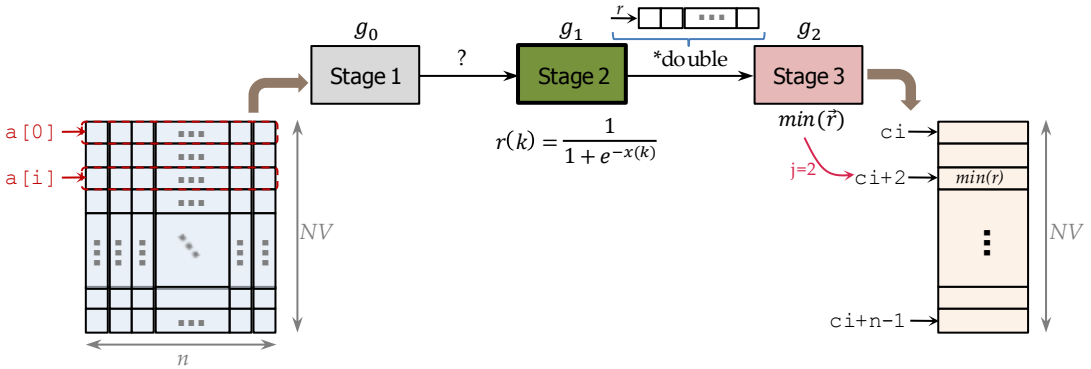

Figure 2. Serial-parallel-serial pipeline. Data is read from a 2D array. Stage 1 feeds input data (a n-element vector) into the pipeline. Parallel Stage 2 performs the element-wise operation. Stage 3 computes the minimum of the incoming vector and places the result in an output array.

- Write a `.cpp` program that reads the parameters n (length of vectors) and NV (number of vectors), and then performs the specified computations.
  - ✓ Your code should implement the software pipeline in Fig. 2 using TBB *parallel_pipeline*.
  - ✓ Include a standard sequential implementation in order to compare processing times.
  - ✓ Your code should measure the computation time (only the actual computation portion) in us for both the pipelined and the sequential implementation.

- **Verification**: Use the following numbers to initialize:

```
for (i=0; i < NV; i++)
    for (j=0; j < n; j++)
       a[i][j] = i*0.4 - j*0.5;
```

  - ✓ To verify the correctness of your result, have your program print out the result for n=5, NV=20. The result should match the following 20-element result vector:

```
Result vector: [ 0.119203      0.167982      0.231475      0.310026      0.401312
                 0.5           0.598688      0.689974      0.768525      0.832018
                 0.880797      0.916827      0.942676      0.960834      0.973403
                 0.982014      0.987872      0.991837      0.994514      0.996316]'
```

- Compile the code and execute the application on the DE2i-150 Board. Complete Table I and Table II (use an average of 10 executions in order to get the computation time for each case).
  - ✓ Example: `./pip_lab6 100 300 ↵`
    - It will process 300 100-element vectors and return a 300-element result.

TABLE I. COMPUTATION TIME (US) – SEQUENTIAL IMPLEMENTATION

	NV						
n	100	200	500	1,000	5,000	10,000	20,000
20							
50							
100							
500							
1,000							

TABLE II. COMPUTATION TIME (US) – IMPLEMENTATION WITH TBB PIPELINE

	NV						
n	100	200	500	1,000	5,000	10,000	20,000
20							
50							
100							
500							
1,000							

- Comment on your results in Tables I and II. Does increasing NV and/or n result in a consistent processing time improvement?
  - ✓ At what (approximate) point (in terms of NV and n) does time improvement occur?
  - ✓ At what (approximate) point (in terms of NV and n) doe time improvement stop?

- Take a screenshot of the software running in the Terminal for n=5, NV=20. It should show the output array and the processing times for both the pipelined and the sequential implementation.

SUBMISSION

- Demonstration: In this Lab 6, the requested screenshot of the software routine running in the Terminal suffices.
  - ✓ If you prefer, you can request a virtual session (Webex) with the instructor and demo it (using a camera).

- Submit to Moodle (an assignment will be created):
  - ✓ One .zip file:
    - 1st Activity: The .zip file must contain the source files (`.cpp, .h, Makefile`) and the requested screenshot.
  - ✓ The lab sheet (a PDF file) with the completed Tables I and II as well as your comments.

TA signature: _____ **Date:** _____