Laboratory 3

(Due date: Oct. 10th)

OBJECTIVES

- Compile and execute multi-threaded C code in Ubuntu 12.04.4 using the Terasic DE2i-150 Development Kit.
- Learn multi-threading implementation using *pthreads* in C.
- Compare computation time of multi-threaded implementations using different number of threads.

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® AtomTM platform → *Tutorial 3 and 4* for associated examples.

ACTIVITIES

FIRST ACTIVITY: CENTERED MOVING AVERAGE (WINDOW SIZE = 7)

- Given an n-element vector $\vec{a}$, where $a(i)$ is an element of the vector ($i = 0,1, \dots, n-1$), the elements of the 7-element moving average $\vec{f}$ are given by:

$$f(i) \leftarrow \frac{a(i-3)+a(i-2)+a(i-1)+a(i)+a(i+1)+a(i+2)+a(i+3)}{7}$$

  - ✓ The moving average is usually a central moving average that can be computed using data equally spaced on either side of a central value (this needs the number of elements in the window to be odd).
  - ✓ In the formula, $i = 0,1, \dots n-1$. When the elements are not available (at the borders), we only use the available elements:

$$f(0) \leftarrow \frac{a(i)+a(i+1)+a(i+2)+a(i+3)}{4}$$
$$f(1) \leftarrow \frac{a(i-1)+a(i)+a(i+1)+a(i+2)+a(i+3)}{5}$$
$$f(2) \leftarrow \frac{a(i-2)+a(i-1)+a(i)+a(i+1)+a(i+2)+a(i+3)}{6}$$
$$f(n-1) \leftarrow \frac{a(i-3)+a(i-2)+a(i-1)+a(i)}{4}$$
$$f(n-2) \leftarrow \frac{a(i-3)+a(i-2)+a(i-1)+a(i)+a(i+1)}{5}$$
$$f(n-3) \leftarrow \frac{a(i-3)+a(i-2)+a(i-1)+a(i)+a(i+1)+a(i+2)}{6}$$

- Fig. 1 depicts an example. The original data (102 data points) is plotted as a series of dots. The 7-element moving average smooths short-term fluctuations and highlight longer-term trends.

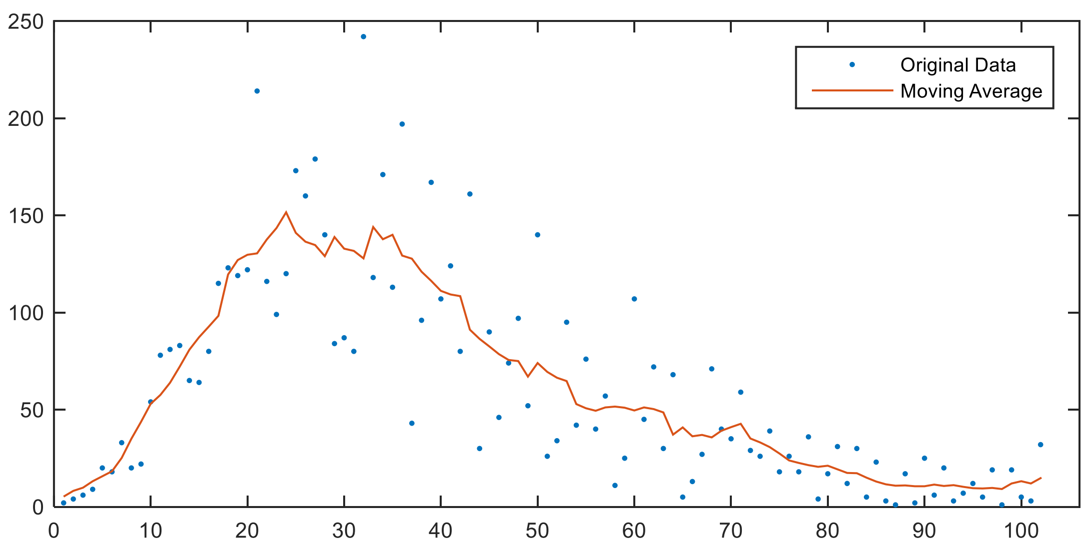

Figure 1. Seven-element moving average

INSTRUCTIONS

- Write a `.c` program that reads in the parameter *nthreads*, reads the input data set from a binary input file (`.bif`), computes the 7-element centered moving average and displays the result.
  - ✓ Your code should measure the computation time (only the actual computation portion) in us.

- **Considerations**:
  - ✓ Input dataset: 100,000 elements of type `int32`. This is available in the provided `mydata.bif` file.
    - You can use this code snippet to read data from a binary file (use `typ=1` since each element is of type `int32`).

```
int read_binfile (int *data, int Length, char *in_file, int typ) {
// data: array where the data read from file is placed
// type: type = 0: each element is unsigned 8-bit integer. ==> 'unsigned char'
//       type = 1: each element is a signed integer (32 bits) ==> 'int'
// Length: # of elements to read (if type =1 --> number of 32-bit words)
  FILE *file_i;
  int i;
  size_t result, ELEM_SIZE;

  if (typ != 0 && typ != 1) { printf ("Wrong modifier (only 0, 1 accepted)\n"); return -1; }

  file_i = fopen(in_file,"rb");
  if (file_i == NULL) { printf ("Error opening file!\n"); return -1; }

  if (typ == 0) { // each element is an unsigned integer of 8 bits
      unsigned char *IM;
      IM = (unsigned char *) calloc (Length, sizeof(unsigned char));

      ELEM_SIZE = sizeof(unsigned char);
      result = fread (IM, sizeof(unsigned char), Length, file_i);

      for (i = 0; i < Length; i++)  data[i] = (int) IM[i];
      free (IM); }
  else { //  if (typ == 1) // each element is a signed 32-bit integer
      int *IM;
      IM = (int *) calloc (Length, sizeof(int));

      ELEM_SIZE = sizeof(int);
      result = fread (IM, sizeof(int), Length, file_i);

      for (i = 0; i < Length; i++)  data[i] = IM[i];
      free (IM); }

  fclose (file_i);
  printf ("(read_binfile) Input binary file '%s': # of elements read = %ld\n", in_file, result);
  printf ("(read_binfile) Size of each element: %ld bytes\n", ELEM_SIZE);
  return 0;
}
```

  - ✓ Strategy for parallelization: Given *nthreads* threads, the index i represents each thread from *0* to *nthreads-1*.
    - Each thread i is in charge of processing a slice of the input vector in order to generate a slice of the output vector.
    - The thread i computes the slice of the output vector $\vec{f}$ with the following indices:
      - From $\left\lfloor \frac{i \times n}{nthreads} \right\rfloor$ to $\left\lfloor \frac{(i+1) \times n}{nthreads} \right\rfloor - 1$.
    - Note that $nthreads \in [1, n]$.

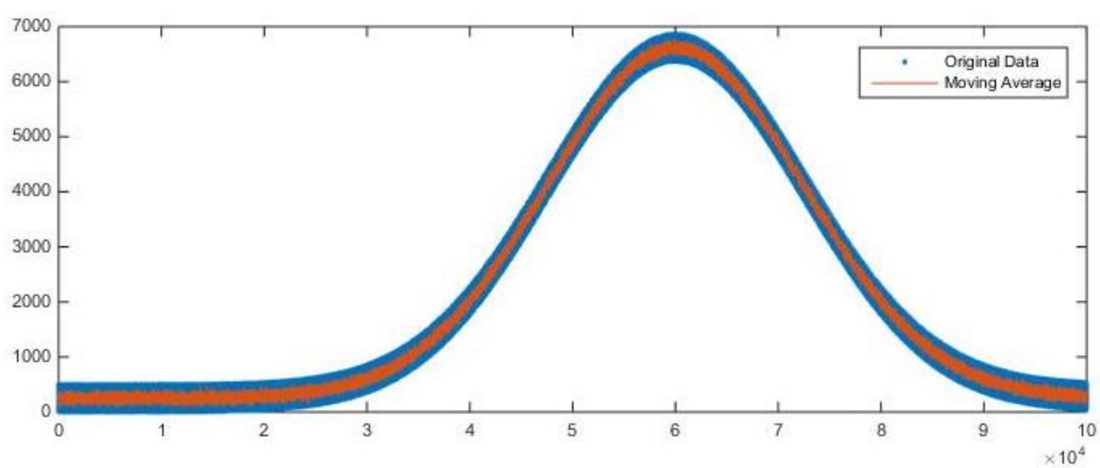

Figure 2. Seven-element moving average for the 100,000-element input dataset.

- **Verification**: Fig. 2 depicts the input dataset along with the 7-element moving average.
  - ✓ The dataset is relatively large, so to verify the correctness of your result, have your program print out the following indices of output vector $\vec{f}$:
    - `f(0:19), f(1000:1019), f(99980:99999)`
  - ✓ Fig. 3 shows a screenshot of the execution in the Terminal with the three 20-element sets of values.

```
daniel@daniel-Inspiron-1545: ~/Dropbox/mystuff/work_ubuntu/labs/lab3
daniel@daniel-Inspiron-1545:~/Dropbox/mystuff/work_ubuntu/labs/lab3$ ./movavg_pthreads 5
(read_binfile) Input binary file 'mydata.bif': # of elements read = 100000
(read_binfile) Size of each element: 4 bytes

Creating 5 Threads
Thread 0 computes slice 0 (indices: 0 to 19999)
Thread 1 computes slice 1 (indices: 20000 to 39999)
Thread 2 computes slice 2 (indices: 40000 to 59999)
Thread 3 computes slice 3 (indices: 60000 to 79999)
Thread 4 computes slice 4 (indices: 80000 to 99999)
odata[0] = 218.7500     odata[1000] = 303.2857  odata[99980] = 397.7143
odata[1] = 273.6000     odata[1001] = 281.2857  odata[99981] = 406.1429
odata[2] = 310.8333     odata[1002] = 246.2857  odata[99982] = 384.2857
odata[3] = 329.2857     odata[1003] = 246.7143  odata[99983] = 405.2857
odata[4] = 277.7143     odata[1004] = 250.8571  odata[99984] = 381.5714
odata[5] = 244.8571     odata[1005] = 281.2857  odata[99985] = 320.4286
odata[6] = 310.8571     odata[1006] = 238.0000  odata[99986] = 287.5714
odata[7] = 311.0000     odata[1007] = 264.0000  odata[99987] = 245.7143
odata[8] = 252.1429     odata[1008] = 228.5714  odata[99988] = 194.7143
odata[9] = 220.1429     odata[1009] = 267.5714  odata[99989] = 179.5714
odata[10] = 216.8571    odata[1010] = 263.2857  odata[99990] = 178.7143
odata[11] = 210.7143    odata[1011] = 249.8571  odata[99991] = 181.0000
odata[12] = 238.1429    odata[1012] = 234.0000  odata[99992] = 234.4286
odata[13] = 235.8571    odata[1013] = 254.7143  odata[99993] = 262.2857
odata[14] = 272.8571    odata[1014] = 275.4286  odata[99994] = 319.8571
odata[15] = 300.0000    odata[1015] = 317.5714  odata[99995] = 374.8571
odata[16] = 302.2857    odata[1016] = 340.1429  odata[99996] = 376.7143
odata[17] = 252.2857    odata[1017] = 319.1429  odata[99997] = 371.5000
odata[18] = 316.5714    odata[1018] = 335.1429  odata[99998] = 391.0000
odata[19] = 335.0000    odata[1019] = 378.0000  odata[99999] = 377.2500
(write_binfile) Output binary file 'mydata.bof': # of (int32) elements written = 100000
start: 310084 us
end: 312837 us
Elapsed time (only actual computation): 2753 us
daniel@daniel-Inspiron-1545:~/Dropbox/mystuff/work_ubuntu/labs/lab3$
```

Figure 3. Execution of 7-element moving average showing three 20-element sets of values (the computation time corresponds to an execution on a Dell Inspiron laptop)

- Compile the code and execute the application on the DE2i-150 Board. Complete Table I (use an average of 10 executions in order to get the computation time for each case).
  - ✓ Example: `./my_movavg 10`
    - It will compute the moving average of the input dataset using 10 threads.

TABLE I. COMPUTATION TIME (US) VS. NUMBER OF THREADS

	nthreads									
	1	2	3	4	5	6	7	8	9	10
Computation Time (us)	19270.4	11164.8	10634.2	9907.4	11181.6	11497	11584.1	11034.9	12393	12187.8

✓ Comment on your results in Table I. Is there an optimal number of threads? At what point increasing the number of threads causes an increase in processing time?

> The optimal number of threads is reached at n = 4. At this point the average computation time reaches its lowest time of 9907.7 us. After this point the average computation time reaches above 12000 us.

- Take a screenshot of the software running in the Terminal for $nthreads=5$. It should show the computation time along with the three 20-element sets of values for the output vector $\vec{f}$ (like in Fig. 3).

- Provided file: `mydata.bif`.

SUBMISSION

- Demonstration: In this Lab 3, the requested screenshot of the software routine running in the Terminal suffices.
  - ✓ If you prefer, you can request a virtual session (Zoom) with the instructor and demo it.

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

TA signature: ______________________________ **Date:** ___________________________