

Midterm Exam

(October 19th @ 7:30 pm)

- Implement SAXPY (Single-Precision A.X Plus Y), also called Scaled Vector Addition with both *pthread*s and TBB.

$$\vec{y} \leftarrow a\vec{x} + \vec{y}$$
- SAXPY is a combination of scalar multiplication and vector addition. It takes as input two n -element input vectors \vec{x} and \vec{y} (whose elements are 32-bit floating point numbers), and a scalar value a . A simple C implementation looks like this:

```
void saxpy(int n, float a, float *x, float *y) {
    for (int i = 0; i < n; i++)
        y[i] = a*x[i] + y[i];
}
```

PROBLEM 1 (60 PTS)

- Implement SAXPY using *pthread*s in C (30 pts)
 - Your code should read the parameter *nthreads* (number of threads) and the length of the vectors (n).
 - Note that $nthreads \in [1, n]$.
 - Parallelization: each thread i ($i \in [1, n]$) computes a slice of the output vector \vec{y} with the following indices:
 - From $\lfloor \frac{i \times n}{nthreads} \rfloor$ to $\lfloor \frac{(i+1) \times n}{nthreads} \rfloor$.
 - Input data:** Given the length n , your code should initialize the vectors \vec{x} and \vec{y} as per the following pseudo-code:


```
a = 1.618
for i = 0:n-1
    x[i] = sinh(i*3.416/n);    y[i] = cosh(i*3.416/n);
```
 - Verification:** To be fully sure that your results are correct, you need to create a sequential implementation and then compare the results with those of your multi-threaded implementation. This can be achieved by computing the sum of absolute differences (SAD), which should be 0.0:

$$diff = \sum_{i=0}^{n-1} |y_p(i) - y_s(i)|$$

where \vec{y}_p and \vec{y}_s are the output vectors of the multi-threaded and sequential implementations respectively.

- Compile the code and execute the application on the DE2i-150 Board. Complete Table I (take an average of ~ 10 executions in order to get the computation time for each case). (20 pts)
 - Example:** `./mysaxpy 1000 10`
 - It will compute SAXPY on 1000-element vectors \vec{x} and \vec{y} using 10 threads.

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

n	<i>nthreads</i>									
	1	2	3	4	5	6	7	8	9	10
1,000										
10,000										
100,000										
1,000,000										
2,000,000										

- 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?

- Take (and attach) a screenshot of the software running in the terminal for $nthreads=5$, $n=20$. It should show the computation times (for both the sequential and the *pthread*s implementations), the input vectors \vec{x} and \vec{y} , the output vector \vec{y} , and the sum of absolute differences (SAD). Fig. 1 shows an execution example. (10 pts)

```

daniel@daniel-Inspiron-1545: ~/Dropbox/mystuff/work_ubuntu/labs/midterm/saxpy_pthreads
x(input)      y(input)      y(output)
x[0]=0.0000   y[0]=1.0000   y[0]=1.0000
x[1]=0.1716   y[1]=1.0146   y[1]=1.2923
x[2]=0.3483   y[2]=1.0589   y[2]=1.6224
x[3]=0.5351   y[3]=1.1342   y[3]=2.0000
x[4]=0.7376   y[4]=1.2426   y[4]=2.4360
x[5]=0.9617   y[5]=1.3874   y[5]=2.9433
x[6]=1.2138   y[6]=1.5727   y[6]=3.5367
x[7]=1.5015   y[7]=1.8040   y[7]=4.2335
x[8]=1.8331   y[8]=2.0881   y[8]=5.0541
x[9]=2.2183   y[9]=2.4333   y[9]=6.0224
x[10]=2.6683  y[10]=2.8496  y[10]=7.1670
x[11]=3.1964  y[11]=3.3492  y[11]=8.5210
x[12]=3.8180  y[12]=3.9468  y[12]=10.1243
x[13]=4.5512  y[13]=4.6598  y[13]=12.0237
x[14]=5.4175  y[14]=5.5091  y[14]=14.2746
x[15]=6.4423  y[15]=6.5194  y[15]=16.9430
x[16]=7.6554  y[16]=7.7205  y[16]=20.1069
x[17]=9.0924  y[17]=9.1473  y[17]=23.8588
x[18]=10.7953 y[18]=10.8416 y[18]=28.3084
x[19]=12.8139 y[19]=12.8529 y[19]=33.5859
Sum of absolute differences: 0.0000

Time measurements
*****
pthread implementation - nthreads = 10
start: 555010 us      end: 555766 us
Elapsed time: 756 us
Sequential implementation
start: 556037 us      end: 556038 us
Elapsed time: 1 us
daniel@daniel-Inspiron-1545:~/Dropbox/mystuff/work_ubuntu/labs/midterm/saxpy_pthreads$

```

Figure 1. SAXPY execution showing three 20-element sets of values. Computation times obtained from execution on a Dell Inspiron laptop.

PROBLEM 2 (40 PTS)

- Implement SAXPY using TBB *parallel_for* in C++ (15 pts)
 - Follow the same procedure as in Problem 1, but instead of using *pthread*s to implement slices of the output vector, use *parallel_for* to fully parallelize the sequential SAXPY. Make sure to include a sequential implementation in C++.
 - Your code should read the parameter input data set size (n).
- Compile the code and execute the application on the DE2i-150 Board. Complete Table II (take an average of ~10 executions for each case). (15 pts)
 - Example: `./mysaxpy_tbb 1000`
 - It will compute SAXPY on 1000-element vectors \vec{x} and \vec{y} .

TABLE II. COMPUTATION TIME (US) VS. VECTORS LENGTH

Implementation	n				
	10,000	100,000	1,000,000	2,000,000	5,000,000
Sequential					
TBB					

- Comment on your Table II results. Is there any point at which the TBB implementation is faster than the sequential one? Yes or No? If No, can you venture a guess as to why this is happening?

- Take (and attach) a screenshot of the software running in the terminal for $n=20$. It should show the computation times (both sequential and the TBB implementations), the input vectors \vec{x} and \vec{y} , the output vector \vec{y} and the SAD (as in Fig. 1). (10 pts)

SUBMISSION

- Demonstration: In this Midterm, the requested screenshots of the software routines running in the Terminal suffices.
- Submit to Moodle (an assignment will be created):
 - ✓ Two .zip files (one for Problem 1 and one for Problem 2).
 - Problem 1: The .zip file must contain the source files (.c, .h, Makefile).
 - Problem 2: The .zip file must contain the source files (.cpp, .h, Makefile).
 - ✓ Your Midterm work (a PDF file): This must include the completed Tables I and II, your comments, as well as the requested screenshots (2).