Ways to SAXPY

Prof. Naga Kandasamy
ECE Department
Drexel University

April 14, 2020

This assignment, worth twenty points, is due April 26, 2020, by 11:59 pm. You may work on it in a group of up to two people. Please submit original code.

You are asked to compare two different ways of parallelizing the SAXPY loop using the pthread library. SAXPY is a function within the standard Basic Linear Algebra Subroutines (BLAS) library and stands for "Single-Precision AX Plus Y." The implementation is very simple, involving a combination of scalar multiplication and vector addition. The routine takes as inputs two vectors of 32-bit floating-point values x and y with n elements each, and a scalar value a. It multiplies each element $x[i]$ by a and adds the result to $y[i]$. The serial implementation looks like this:

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

Using the provided sequential program `saxpy.c` as a starting point, develop two versions that parallelize SAXPY.

- **(10 points)** Complete the function *compute_using_pthreads_v1()* to parallelize the SAXPY loop using the "chunking" method. That is, given vectors x and y of some arbitrary length n, and when k threads are used, individual threads calculate SAXPY in parallel on smaller chunks of these vectors.
- **(10 points)** Complete the function *compute_using_pthreads_v2()* to parallelize the SAXPY loop using the "striding" method. That is, each thread strides over elements of the vectors with some stride length, calculating SAXPY along the way. For example, given $k = 4$ threads, thread 0 calculates SAXPY for elements $y[0], y[4], y[8], \ldots$, thread 1 calculates SAXPY for elements $y[1], y[5], y[9], \ldots$, and so on. The pseudo-code for this method looks like this for each thread:

```
/* tid is the thread ID and k is the number of threads created */
int stride = k;
while (tid < n) {
    y[tid] = a * x[tid] + y[tid];
    tid = tid + stride;
}
```

The program given to you accepts command-line arguments for the length of the vectors and the number of threads to use. For example, executing the program as

```
$ ./saxpy 1000000 8
```

will create x and y vectors with 10^6 elements each. The solutions provided by the pthread implementations using eight threads will be compared to that generated by the reference code.

Upload all source files needed to run your code as a single zip file on BBLearn. One submission per group is sufficient. You may add additional data structures and functions as you deem necessary. The code must compile and run on the xunil cluster. Also, include a brief report describing: (1) the design of your multi-threaded implementations, using code or pseudocode to clarify the discussion; (2) the performance achieved by 4, 8, and 16 threads, for 10^4, 10^6, and 10^8 elements, relative to the serial version. As part of your report, provide a clear reasoning for any differences that you might observe in the performance achieved by your parallel versions.