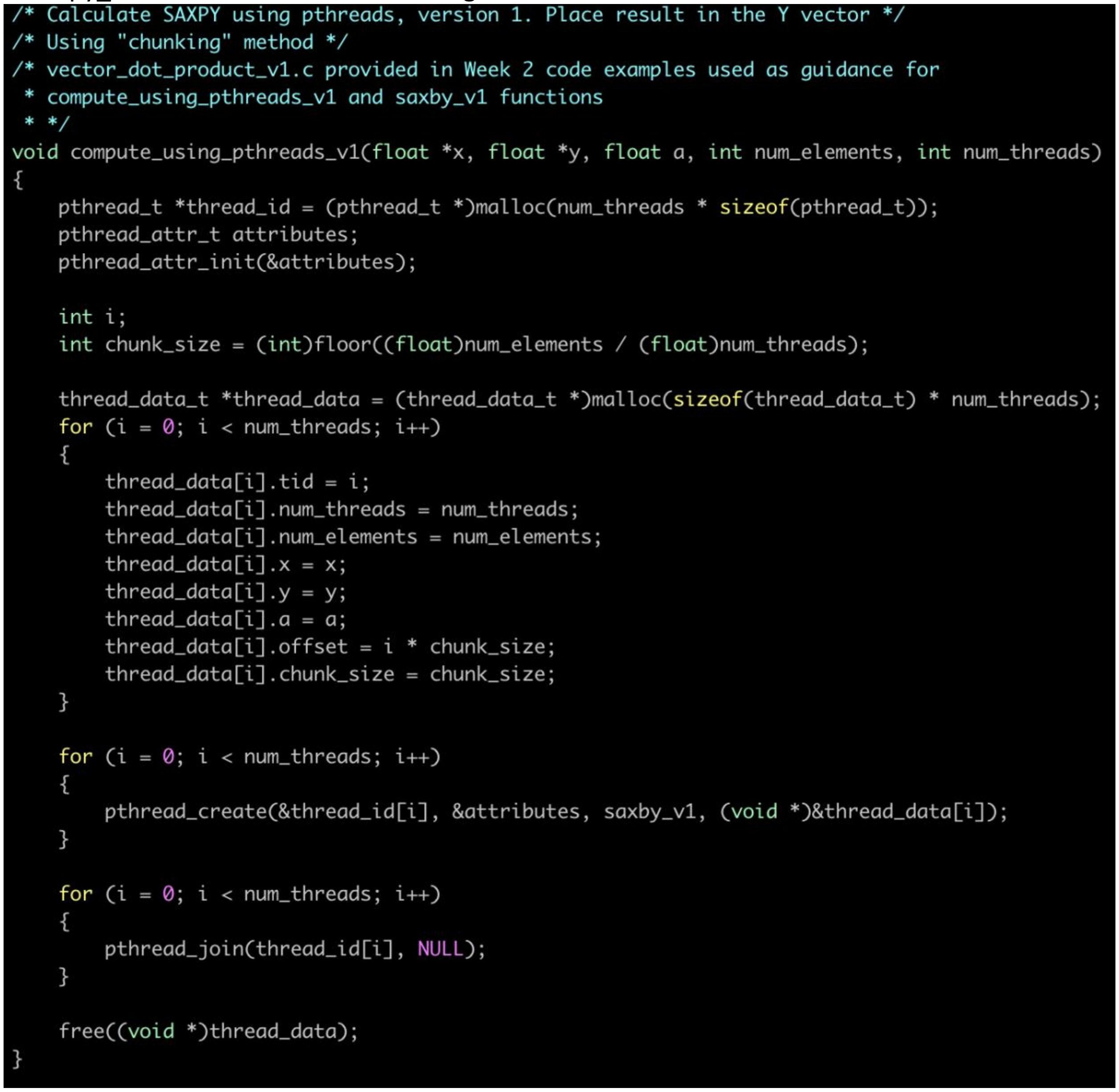
ECEC 622
Assignment 1 Report – January 19, 2022
Sherry Lu
Atandrila Anuva

SAXPY Version 1 – Brief Description

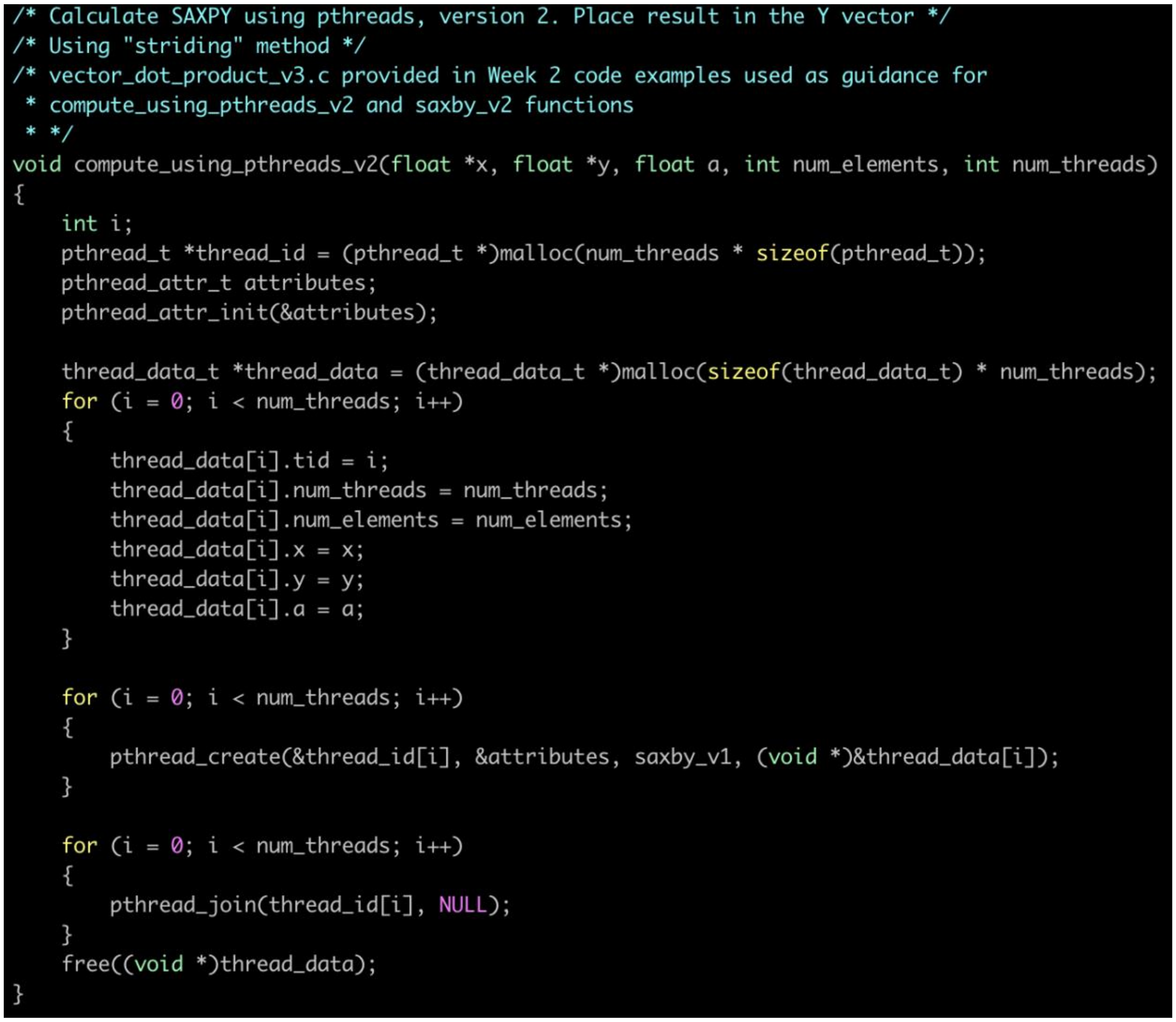
For the version 1 of the assignment, we used the code examples provided in Week 2 (vector_dot_product_v1.c) as a guide. The code uses a chunking method where elements of the data array are split into chunks and distributed among the threads to do their computations. To accomplish this, an offset and chunk size were used. The implementation uses two functions: compute_using_pthreads_v1, and saxby_v1 (helper function). The saxby_v1 function is called from compute_using_pthreads_v1 to create the threads. The offset and chunk size is then used in saxpy_v1 function to distribute among the threads.



```c
/* Calculate SAXPY using pthreads, version 1. Place result in the Y vector */
/* Using "chunking" method */
/* vector_dot_product_v1.c provided in Week 2 code examples used as guidance for
 * compute_using_pthreads_v1 and saxby_v1 functions
 * */
void compute_using_pthreads_v1(float *x, float *y, float a, int num_elements, int num_threads)
{
    pthread_t *thread_id = (pthread_t *)malloc(num_threads * sizeof(pthread_t));
    pthread_attr_t attributes;
    pthread_attr_init(&attributes);

    int i;
    int chunk_size = (int)floor((float)num_elements / (float)num_threads);

    thread_data_t *thread_data = (thread_data_t *)malloc(sizeof(thread_data_t) * num_threads);
    for (i = 0; i < num_threads; i++)
    {
        thread_data[i].tid = i;
        thread_data[i].num_threads = num_threads;
        thread_data[i].num_elements = num_elements;
        thread_data[i].x = x;
        thread_data[i].y = y;
        thread_data[i].a = a;
        thread_data[i].offset = i * chunk_size;
        thread_data[i].chunk_size = chunk_size;
    }

    for (i = 0; i < num_threads; i++)
    {
        pthread_create(&thread_id[i], &attributes, saxby_v1, (void *)&thread_data[i]);
    }

    for (i = 0; i < num_threads; i++)
    {
        pthread_join(thread_id[i], NULL);
    }

    free((void *)thread_data);
}
```

SAXPY Version 2 – Brief Description

For the version 2 of the assignment, we used the code examples provided in Week 2 (vector_dot_product_v3.c) as a guide. The code uses a striding method which assigns a starting element for each thread. For example, thread 0 calculates SAXPY for elements 0, 4, 8, …, thread 1 calculates for elements 1, 5, 9, …, and so on. The implementation uses two functions: compute_using_pthreads_v2, and saxby_v2 (helper function). The saxby_v2 function is called from compute_using_pthreads_v2 to create the threads. Using the stride (int) in saxpy_v2 function, the program moves from element to element.



```c
/* Calculate SAXPY using pthreads, version 2. Place result in the Y vector */
/* Using "striding" method */
/* vector_dot_product_v3.c provided in Week 2 code examples used as guidance for
 * compute_using_pthreads_v2 and saxby_v2 functions
 * */
void compute_using_pthreads_v2(float *x, float *y, float a, int num_elements, int num_threads)
{
    int i;
    pthread_t *thread_id = (pthread_t *)malloc(num_threads * sizeof(pthread_t));
    pthread_attr_t attributes;
    pthread_attr_init(&attributes);

    thread_data_t *thread_data = (thread_data_t *)malloc(sizeof(thread_data_t) * num_threads);
    for (i = 0; i < num_threads; i++)
    {
        thread_data[i].tid = i;
        thread_data[i].num_threads = num_threads;
        thread_data[i].num_elements = num_elements;
        thread_data[i].x = x;
        thread_data[i].y = y;
        thread_data[i].a = a;
    }

    for (i = 0; i < num_threads; i++)
    {
        pthread_create(&thread_id[i], &attributes, saxby_v1, (void *)&thread_data[i]);
    }

    for (i = 0; i < num_threads; i++)
    {
        pthread_join(thread_id[i], NULL);
    }
    free((void *)thread_data);
}
```

SAXPY Version 1 vs Version 2 – Performance

Version 1

	Threads		
Number of Elements	4	6	8
10^4	0.000217s	0.000223s	0.000409s
10^6	0.002371s	0.001838s	0.002281s
10^8	0.328708s	0.175810s	0.175036s

Version 2

	Threads		
Number of Elements	4	6	8
10^4	0.000179s	0.000241s	0.000380s
10^6	0.002523s	0.003433s	0.003139s
10^8	0.575856s	0.308693s	0.291660s

From the performance difference observed in the tables above, there is a significant difference between Version 1 – Chunking and Version 2 – Striding. In the striding method, invalidation of caches leads to the larger execution time as seen from the data above. As we increase the number of threads, we are distributing the elements either through chunking or striding method.