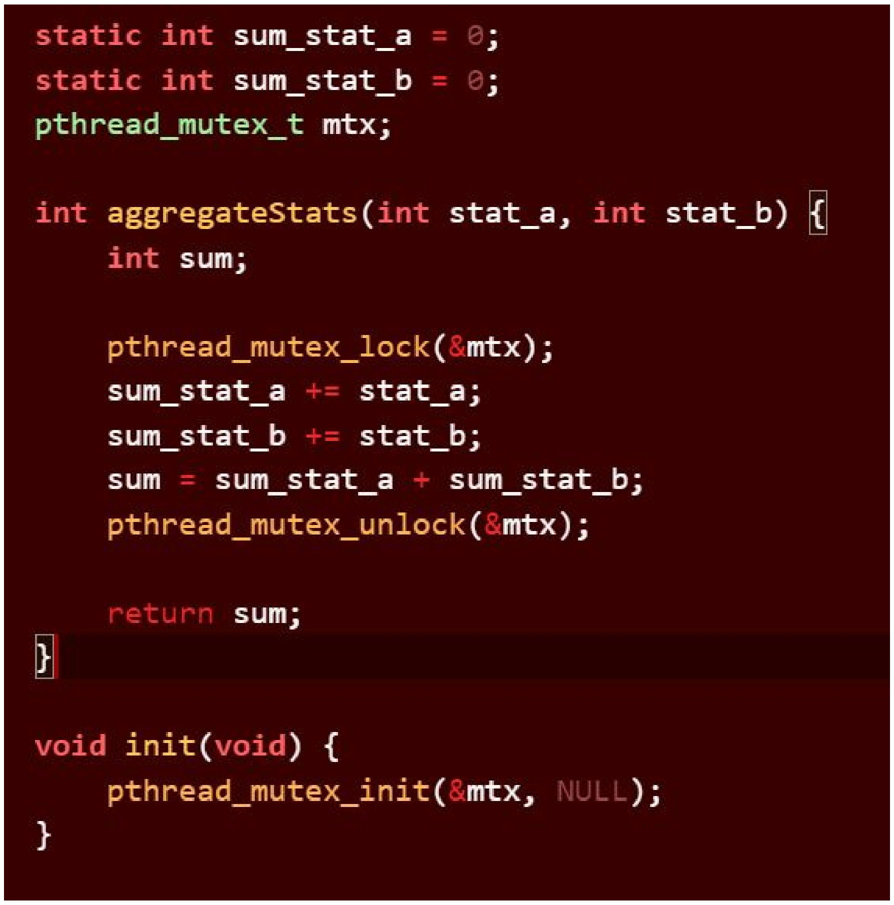
Michael Laucella
Multicore Programming
HW2

1. Threads should equal 2 times the number of cores on the machine assuming the data set given is enough to warrant the need of this many threads, if the dataset given is not this large then the number of threads should be at a point where the efficiency is reasonable given the speedup.

2. The way Peterson's algorithm works is by taking advantage of cache consistency, because the two threads will write to the victim only one whole value can be stored not malformed values. Hence if both threads set the victim flag it will only be P0 or P1, so even though both threads may set each others flag to true only one will have the victim flag set allowing the other to proceed to the critical section. After the critical section completes the thread sets the others flag to false allowing it to break the while loop and proceed. This process will fail for 3 locks because the victim flag, for instance imagine P0 set itself as victim then P1 sets itself as victim allowing P0 to pass now P2 comes sets himself as victim allowing P1 to pass but P0 might still be in the critical section meaning that both P0 and P1 will be in the critical section together.

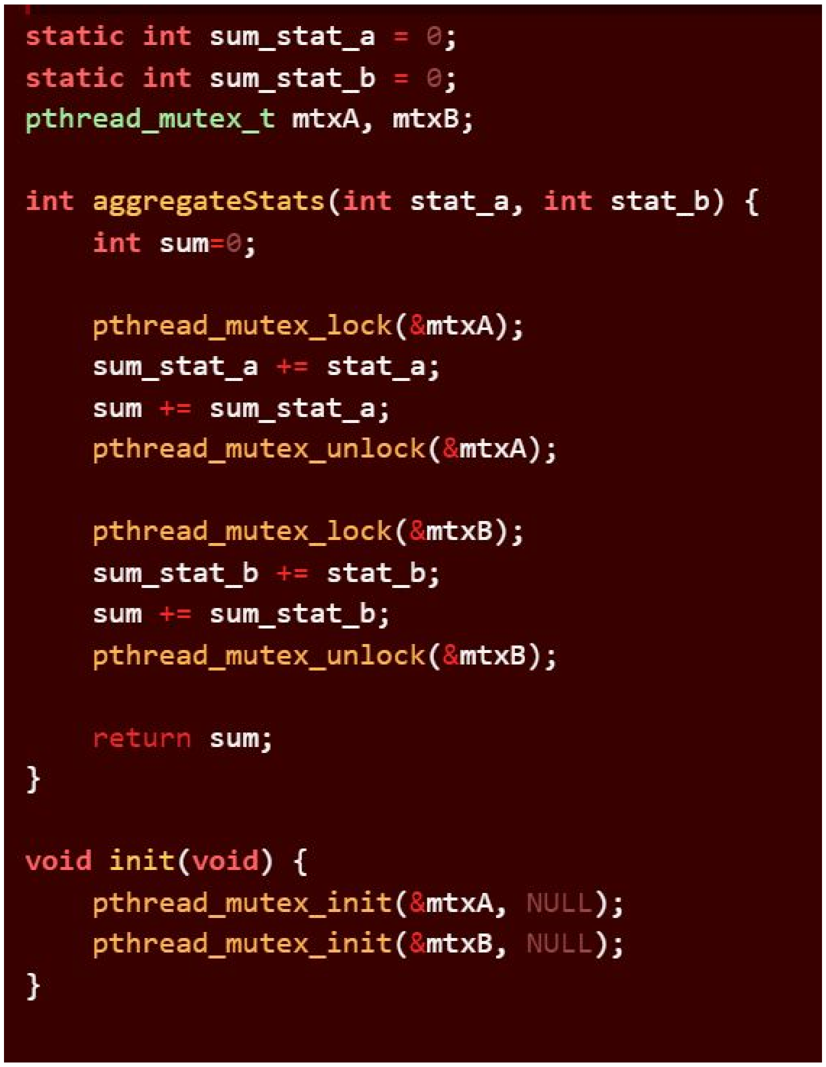
3. L1 cache is fast but also very small, as a result often things will need to be accessed further down the memory chain and although L2 and L3 are slower than L1 they store more and are considerably faster than an access to main memory, so having L2 and L3 allow for less frequent access to main memory.

4. Because adding to A and B must be exclusive they are thrown in the mutex also in order to return the sum of the two they must also not change while this is occurring so we wrap the sum with the mutex as well this can be seen in the photo.



```c
static int sum_stat_a = 0;
static int sum_stat_b = 0;
pthread_mutex_t mtx;

int aggregateStats(int stat_a, int stat_b) {
    int sum;

    pthread_mutex_lock(&mtx);
    sum_stat_a += stat_a;
    sum_stat_b += stat_b;
    sum = sum_stat_a + sum_stat_b;
    pthread_mutex_unlock(&mtx);

    return sum;
}

void init(void) {
    pthread_mutex_init(&mtx, NULL);
}
```

5. If we now have two mutices we can surround As addition and Bs addition with locks however the sum still needs to be taken so we wrap the sum of 0 and A in As mutex and then the sum and B in Bs mutex as can be seen in the photo.



```c
static int sum_stat_a = 0;
static int sum_stat_b = 0;
pthread_mutex_t mtxA, mtxB;

int aggregateStats(int stat_a, int stat_b) {
    int sum=0;

    pthread_mutex_lock(&mtxA);
    sum_stat_a += stat_a;
    sum += sum_stat_a;
    pthread_mutex_unlock(&mtxA);

    pthread_mutex_lock(&mtxB);
    sum_stat_b += stat_b;
    sum += sum_stat_b;
    pthread_mutex_unlock(&mtxB);

    return sum;
}

void init(void) {
    pthread_mutex_init(&mtxA, NULL);
    pthread_mutex_init(&mtxB, NULL);
}
```

6. Pthread_detach tells the system that after the thread terminates to free the threads memory while allowing the caller to proceed, whereas Pthread_join suspends the caller until the thread it's waiting for terminates.