**Problem statement:**

Divide a simple task between threads. The task can easily be divided in sub-tasks requiring no cooperation at all. See the effects of false sharing, and the costs of creating threads and of switching between threads.

Requirement: write two problems: one for computing the sum of two matrices, the other for computing the product of two matrices.

Divide the task between a configured number of threads (going from 1 to the number of elements in the resulting matrix). See the effects on the execution time.

**Solution:**

For computing the sum of the two matrices the user provide the number of threads. Based on that number I compute the dimension of the matrix that should be computed by every thread and every thread simple add the values for a given area of the matrix. This operation doesn’t require any lock because every thread just sum its part.

For computing the product of the two matrices the user also provide the number of threads but in the program base on the dimension of the matrix I compute the number of thread that I should really use and also compute the dimension of every block of the number. I choose to implement this using the Cannon algorithm so every thread should multiply the same block from the matrix but the matrix will change its state after every iteration. There are two type of threads one which do the computation and another one which change the state of the matrix. Every thread takes the lock associated with its block and check if it already compute or not this part. If not do the computation and mark in *auxProd[i]* (i is the index of the current thread) that it done. The checker thread will iterate over the *auxProd* in order to see if all the thread are done if yes it just reset the matrix.