```
#pragma omp parallel for
for (i = 0; i < ncolsa; i++)
   for (j = 0; i < nrowsb; j++) {
      sum = 0;
      for (k = 0; i < ncolsa; i++)
         sum += a[i][k] * b[k][j];
   }


__global__ void matmul(float *ma,float *mb,float *mc,int nrowsa,int ncolsa,i
nt ncolsb)
{  int k;
   sum = 0;
   for (k = 0; i < ncolsa; i++)
      sum += a[i*ncolsa+k] * b[k*ncols+j];
}

// from http://astro.pas.rochester.edu/~aquillen/gpuworkshop/AdvancedCUDA.pd
f

__global__ void MultiplyOptimise(const float *A, const float *B, float *C) {
  // Extract block and thread numbers
  int bx = blockIdx.x; int by = blockIdx.y;
  int tx = threadIdx.x; int ty = threadIdx.y;

  // Index of first A sub-matrix processed by this block
  int aBegin = dc_wA * BLOCK_SIZE * by;
  // Index of last A sub-matrix
  int aEnd = aBegin + dc_wA - 1;
  // Stepsize of A sub-matrices
  int aStep = BLOCK_SIZE;
  // Index of first B sub-matrix
  // processed by this block
  int bBegin = BLOCK_SIZE * bx;
  // Stepsize for B sub-matrices
  int bStep = BLOCK_SIZE * dc_wB;
  // Accumulator for this thread
  float Csub = 0;
  for(int a = aBegin, b = bBegin; a <= aEnd; a += aStep, b+= bStep) {
     // Shared memory for sub-matrices
     __shared__ float As[BLOCK_SIZE][BLOCK_SIZE];
     __shared__ float Bs[BLOCK_SIZE][BLOCK_SIZE];
     // Load matrices from global memory into shared memory
     // Each thread loads one element of each sub-matrix
     As[ty][tx] = A[a + (dc_wA * ty) + tx];
     Bs[ty][tx] = B[b + (dc_wB * ty) + tx];
     // Synchronise to make sure load is complete
     __syncthreads();
     // Perform multiplication on sub-matrices
     // Each thread computes one element of the C sub-matrix
     for( int k = 0; k < BLOCK_SIZE; k++ ) {
        Csub += As[ty][k] * Bs[k][tx];
     }
    // Synchronise again
     __syncthreads();
   }
   // Write the C sub-matrix back to global memory
   // Each thread writes one element
   int c = (dc_wB * BLOCK_SIZE * by) + (BLOCK_SIZE*bx);
   C[c + (dc_wB*ty) + tx] = Csub;
}

#define BLOCK_SIZE 16

__constant__ int dc_wA;
__constant__ int dc_wB;

```