```
/*** Sequential code: Single thread on GPU ***/
__global__  void add(int n, int *a, int *b, int *c)
{
   int i ;
   for(i=0; i<n; i++) c[i] = a[i]+b[i] ;
}


void main() {
   int n, size, i, *a, *b, *c, *d_a, *d_b, *d_c;
   n = 1024 ;
   size = n*sizeof(int) ;
   a = (int *)malloc(size);
   b = (int *)malloc(size);
   c = (int *)malloc(size);
   for(i=0;i<n; i++) {// Enter data into a[]& b[]
      a[i] = ... ;      b[i] = ... ;
   }
   cudaMalloc( (void **)&d_a, size ) ;
   cudaMalloc( (void **)&d_b, size ) ;
   cudaMalloc( (void **)&d_c, size ) ;
   cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

   add<<<1, 1>>>(n, d_a, d_b, d_c) ;
   cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
 // print c[] ...

  free(a); free(b);  free(c);
  cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
}
```

```
/*** n threads on one thread block ***/
__global__  void add(int *a, int *b, int *c) {
   c[threadIdx.x] = a[threadIdx.x]+b[threadIdx.x] ;
}

void main() {

   . . .

   add<<<1, n>>>(n, d_a, d_b, d_c) ;

   . . .

}
```

```
/*** 16 threads on one thread block ***/

__global__ void add(int n, int *a, int *b, int *c)
{
   int i, i_start, i_end, subset_size = n/blockDim.x ;
// blockDim.x: threads per block
// i_end = ( (i_end<n) ? i_end : n ) ;

   i_start = threadIdx.x* subset_size ;
   i_end   = i_start + subset_size ;
   for(i=i_start; i<i_end; i++)
      c[i] = a[i] + b[i] ;
}


void main() {

   ...

   add<<<1, 16>>>(n, d_a, d_b, d_c) ;

   ...

}
```

```
/*** 2-dimensional thread block ***/
// a, b c are m x n matrices
__global__ void add(int m, int n, int *a, int *b,
                    int *c)
{
   int i, i_start, i_end, j_start, j_end, kx, ky;

   kx = m/ blockDim.x ;
   ky = n/ blockDim.y ;
   i_start = threadIdx.x*kx ;  i_end = i_start + kx ;
   j_start = threadIdx.y*ky ;  j_end = j_start + ky ;
   for(i=i_start; i<i_end; i++)
      for(j=j_start; j<j_end; j++)
         c[i*n+j] = a[i*n+j] + b[i*n+j] ;
}


void main() {
   int m, n, size, i, *a, *b, *c, *d_a, *d_b, *d_c;

   m = 32 ; n = 32 ;
   size = m*n*sizeof(int) ;
   a = (int *)malloc(size);
   b = (int *)malloc(size);
   c = (int *)malloc(size);
   for(i=0;i<m*n; i++) {// Enter data into a[]& b[]
      a[i] = ... ;      b[i] = ... ;
   }
   cudaMalloc( (void **)&d_a, size ) ;
   cudaMalloc( (void **)&d_b, size ) ;
   cudaMalloc( (void **)&d_c, size ) ;
   cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

   dim3 myBlock(4, 4) ;
   add<<<1, myBlock >>>(m, n, d_a, d_b, d_c) ;
   cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
 // print c[] ...

  free(a); free(b);  free(c);
  cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
}
```

```
/*** Hierarchical Thread Structure: multiple blocks ***/

__global__ void add(int n, int *a, int *b, int *c)
{
   int i, i_start, i_end,
       subset_size = n/( gridDim.x*blockDim.x) ;

   i_start = (blockIdx.x * blockDim.x + threadIdx.x)*
              subset_size ;
   i_end   = i_start + subset_size ;
   for(i=i_start; i<i_end; i++)
      c[i] = a[i] + b[i] ;
}


void main() {
   int n, size, i, *a, *b, *c, *d_a, *d_b, *d_c;
   n = 1024 ;
   size = n*sizeof(int) ;
   a = (int *)malloc(size);
   b = (int *)malloc(size);
   c = (int *)malloc(size);
   for(i=0;i<n; i++) {// Enter data into a[]& b[]
      a[i] = ... ;      b[i] = ... ;
   }
   cudaMalloc( (void **)&d_a, size ) ;
   cudaMalloc( (void **)&d_b, size ) ;
   cudaMalloc( (void **)&d_c, size ) ;
   cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
   cudaMemcpy(d_b, b, size, cudaMemcpyHostToDevice);

   add<<<16, 16>>>(n, d_a, d_b, d_c) ;
   cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
 // print c[] ...

  free(a); free(b);  free(c);
  cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);

}
```