```
/*** Sequential code: Single thread on GPU ***/
                                                         /*** n threads on one thread block ***/
 _global__ void add(int n, int *a, int *b, int *c)
                                                         global void add(int *a, int *b, int *c) {
                                                            c[threadIdx.x] = a[threadIdx.x]+b[threadIdx.x];
  int i ;
  for(i=0; i< n; i++) c[i] = a[i]+b[i];
                                                         void main() {
void main() {
  int n, size, i, *a, *b, *c, *d_a, *d_b, *d_c;
                                                            add <<<1, n>>>(n, 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[]</pre>
      a[i] = ...; b[i] = ...;
                                                          /*** 16 threads on one thread block ***/
  cudaMalloc( (void **)&d_a, size );
                                                          __global__ void add(int n, int *a, int *b, int *c)
  cudaMalloc( (void **)&d b, size );
  cudaMalloc( (void **)&d c, size );
                                                            int i, i_start, i_end, subset_size = n/blockDim.x ;
  cudaMemcpy(d a, a, size, cudaMemcpyHostToDevice);
                                                         // blockDim.x: threads per block
  cudaMemcpy(d b, b, size, cudaMemcpyHostToDevice);
                                                         // i end = ((i end < n) ? i end : n);
  add<<<1, 1>>>(n, d_a, d_b, d_c);
                                                            i start = threadIdx.x* subset size ;
  cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
                                                            i end = i start + subset size ;
 // print c[] ...
                                                            for(i=i start; i<i end; i++)</pre>
                                                               c[i] = a[i] + b[i];
 free(a); free(b); free(c);
 cudaFree(d a); cudaFree(d b); cudaFree(d c);
                                                         void main() {
                                                            add<<<1, 16>>>(n, d a, d b, d c);
```

```
/*** 2-dimensional thread block ***/
                                                         /*** Hierarchical Thread Structure: multiple blocks ***/
// a, b c are m x n matrices
__global__ void add(int m, int n, int *a, int *b,
                                                          global void add(int n, int *a, int *b, int *c)
                    int *c)
                                                            int i, i start, i end,
                                                                subset_size = n/(gridDim.x*blockDim.x);
  int i, i start, i end, j start, j end, kx, ky;
  kx = m/blockDim.x;
                                                            i start = (blockIdx.x * blockDim.x + threadIdx.x)*
                                                                       subset size ;
  ky = n/ blockDim.y;
  i_start = threadIdx.x*kx ; i_end = i_start + kx ;
                                                            i end = i start + subset size ;
                                                            for(i=i_start; i<i_end; i++)</pre>
  j_start = threadIdx.y*ky ; j_end = j_start + ky ;
  for(i=i start; i<i end; i++)</pre>
                                                               c[i] = a[i] + b[i];
      for(j=j_start; j<j_end; j++)</pre>
        c[i*n+j] = a[i*n+j] + b[i*n+j] ;
                                                         void main() {
                                                            int n, size, i, *a, *b, *c, *d_a, *d_b, *d_c;
void main() {
                                                            n = 1024;
  int m, n, size, i, *a, *b, *c, *d_a, *d_b, *d_c;
                                                            size = n*sizeof(int) ;
                                                            a = (int *)malloc(size);
                                                            b = (int *)malloc(size);
  m = 32 ; n = 32 ;
  size = m*n*sizeof(int) ;
                                                            c = (int *)malloc(size);
  a = (int *)malloc(size);
                                                            for(i=0;i<n;i++) {// Enter data into a[]& b[]
                                                               a[i] = ...;
  b = (int *)malloc(size);
                                                                               b[i] = ...;
  c = (int *)malloc(size);
                                                            cudaMalloc( (void **)&d_a, size );
  for(i=0;i<m*n;i++) {// Enter data into a[]& b[]
                                                            cudaMalloc( (void **)&d b, size );
      a[i] = ...; b[i] = ...;
                                                            cudaMalloc( (void **)&d c, size );
                                                            cudaMemcpy(d_a, a, size, cudaMemcpyHostToDevice);
  cudaMalloc( (void **)&d_a, size );
                                                            cudaMemcpy(d b, b, size, cudaMemcpyHostToDevice);
  cudaMalloc( (void **)&d b, size );
  cudaMalloc( (void **)&d c, size );
                                                            add<<<16, 16>>>(n, d_a, d_b, d_c);
  cudaMemcpy(d a, a, size, cudaMemcpyHostToDevice);
                                                            cudaMemcpy(c, d c, size, cudaMemcpyDeviceToHost);
  cudaMemcpy(d b, b, size, cudaMemcpyHostToDevice);
                                                          // print c[] ...
  dim3 myBlock(4, 4);
  add<<<1, myBlock >>>(m, n, d a, d b, d c);
                                                           free(a); free(b); free(c);
  cudaMemcpy(c, d_c, size, cudaMemcpyDeviceToHost);
                                                           cudaFree(d_a); cudaFree(d_b); cudaFree(d_c);
 // print c[] ...
 free(a); free(b); free(c);
 cudaFree(d a); cudaFree(d b); cudaFree(d c);
```