

Answer 1- a :

```
ami - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM NSIGHT TOOLS TEST ANALYZE WINDOW HELP
Local Windows Debugger Debug
add_loop_gpu.cu -b X
(Global S C:\Windows\system32\cmd.exe
/*
 * 0 + 0 = 0
 * 1 + 1 = 0
 * 2 + 4 = 2
 * 3 + 9 = 6
 * 4 + 16 = 12
 * 5 + 25 = 20
 * 6 + 36 = 30
 * 7 + 49 = 42
 * 8 + 64 = 56
 * 9 + 81 = 72
 *
 * Press any key to continue . . .
 */
#include
#define
global void add( int *a, int *b, int *c ) {
    int tid = blockIdx.x; // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + b[tid];
}
int main( void ) {
    int a[N], b[N], c[N];
    // ...
}
```

Answer 1- b:

Answer 1- c:

The number of threads created when $N = 70000000$, is too high. Which is impossible for the hardware to be able to create. Thus the output is nothing since the compiler simply exits because it is not able to overcome the hardware limitations.

Answer 2 :

Code :

```
#include "J:\ami\ami\common\book.h"
#define N 10
#define scale 10
__global__ void add( int *a, int *b, int *c ) {
    int tid = blockIdx.x; // this thread handles the data at its thread id
    if (tid < N)
        c[tid] = a[tid] + scale * b[tid];
}
int main( void ) {
    int a[N], b[N], c[N];
    int *dev_a, *dev_b, *dev_c;
    // allocate the memory on the GPU
    HANDLE_ERROR( cudaMalloc( (void**)&dev_a, N * sizeof(int) ) );
    HANDLE_ERROR( cudaMalloc( (void**)&dev_b, N * sizeof(int) ) );
    HANDLE_ERROR( cudaMalloc( (void**)&dev_c, N * sizeof(int) ) );
    // fill the arrays 'a' and 'b' on the CPU
    for (int i=0; i<N; i++) {
        a[i] = -i;
        b[i] = i * i;
    }
    // copy the arrays 'a' and 'b' to the GPU
    HANDLE_ERROR( cudaMemcpy( dev_a, a, N * sizeof(int),
                              cudaMemcpyHostToDevice ) );
    HANDLE_ERROR( cudaMemcpy( dev_b, b, N * sizeof(int),
                              cudaMemcpyHostToDevice ) );
    add<<<N,1>>>>( dev_a, dev_b, dev_c );
    // copy the array 'c' back from the GPU to the CPU
    HANDLE_ERROR( cudaMemcpy( c, dev_c, N * sizeof(int),
                              cudaMemcpyDeviceToHost ) );
    // display the results
    for (int i=0; i<N; i++) {
        printf( "%d + %d = %d\n", a[i], scale*b[i], c[i] );
    }
    // free the memory allocated on the GPU
```

```

HANDLE_ERROR( cudaFree( dev_a ) );
HANDLE_ERROR( cudaFree( dev_b ) );
HANDLE_ERROR( cudaFree( dev_c ) );
return 0;
}

```

Screenshot:

