PROBLEM SET 5

Zhiqiang Xie 77892769

Problem 1:

- False. The SIMD is more likely in one warp. It's SIMT parallelism in CUDA, which is quite different compared to SIMD, here are 3 key features that SIMD doesn't have:
  - Single instruction, multiple register sets
  - Single instruction, multiple addresses
  - Single instruction, multiple flow paths
  - For more information, refer this blog
- False. All threads inside a SM share the registers, the number of registers available varies by the workload of the SM.
- False, threads can share data and synchronize with each other via the constant memory and global memory either.

Problem 2

- Block size: 16×16 threads, SM size: 8 blocks, 2048 threads.
- $n = 256 - (400 * 900)\%256 = 192$ threads will be idle, which may suggest the small block is better?

Problem 3

```
__global__
void vecAddKernel(float* A, float* B, float* C, int n, int p) {
    int i = threadId.x + blockDim.x * blockId.x;
    for (int j=0,j<p;j++){ // where p is the elements to process
        if (i*j<n) C[i*j] = A[i*j] + B[i*j];
    }
}
```

Problem 4

A naive implementation is following, and maybe I'll add the shared-memory implementation:

```
__global__ void kernel(float *mat, float *vec, float *out, const int N){
    int tid=threadIdx.x+blockIdx.x*blockDim.x;
    float res=0;
    if(tid<N){
        for(int i=0; i<N; i++)
            sum += mat[(i*N)+tid]*vec[i];
        out[tid]=sum;
    }
}

int main (void) {
    float *a, *b, *c;
    float *cu_a, *cu_b, *cu_c;
    int N=64;
    a=(float*)calloc(N*N, sizeof(float));
    b=(float*)calloc(N, sizeof(float));
    c=(float*)calloc(N, sizeof(float));

    cudaMalloc((void**)&cu_a, sizeof(float)*N*N);
    cudaMalloc((void**)&cu_b, sizeof(float)*N);
    cudaMalloc((void**)&cu_c, sizeof(float)*N);

    cudaMemcpy(cu_a, a, sizeof(float)*N*N, cudaMemcpyHostToDevice);
    cudaMemcpy(cu_b, b, sizeof(float)*N, cudaMemcpyHostToDevice);

    kernel<<<M/256+1, 256>>>(cu_a, cu_b, cu_c, N);

    cudaMemcpy(c, cu_c, sizeof(float)*N, cudaMemcpyDeviceToHost);
    cudaFree(cu_a);
    cudaFree(cu_b);
    cudaFree(cu_c);

    return 0;
};
```