# key points

Reduction: make a series of reduction to parallel calculation.

reduction tree: takes $log_2 N$ times rather than N-1 times. the maximum number of calcualtor in the same step is N/2, the average number is $\frac{N-1}{log_2 N}$

shared memory for can decrease the times of reading and writing data.

for low number of blocks, utilizing coarsening factor can help this problem. This will take less parallel workers, which should be paid attention to.

# 1

## 1.1

~~that is $\sum_{i=1}^{5}(1024/(2*32*2^{i-1}))$~~ $\frac{1024}{2^5} = 32 active warps$

## 1.2

that is 1 warp

## 1.3

```
__global__ void ConvergentSumReductionKernel(float* input, float* output){
    unsigned int i = threadIdx.x;
    for (unsigned int stride = blockDim.x; stride >= 1; stride /= 2) {
        if (threadIdx.x > stride && threadIdx.x < stride*2){
            input[i] += input[i-stride];
        }
        __syncthreads();
    }
    if(threadIdx.x == 0) {
        *output = input[blockDim.x];//this order help decrease competition
            between different blocks.
    }
}
```

## 1.4

```
__global__ CoarsenedMaxReductionKernel(const float* input, float* output){
    __shared__ float input_s[BLOCK_DIM];
    unsigned int segment = COARSE_FACTOR*2*blockDim.x*blockIdx.x;
    unsigned int i = segment + threadIdx.x;
    unsigned int t = threadIdx.x;
    float maximum = input[i];
    for(unsigned int tile = 1; tile < COARSE_FACTOR*2; ++tile) {
        maximum = max(maximum, input[i+tile*BLOCK_DIM]);
    }
    input_s[t] = maximum;
    for (unsigned int stride = blockDim.x/2; stride >= 1; stride /= 2){
        __syncthreads();
        if ( t < stride) {
            input_s[t] = max(input_s[t], input_s[t+stride]);
        }
    }
    if (t == 0){
        output[blockIdx.x] = input_s[0];
    }
}
```

## 1.5

### 1.5.1

for each iteration, it should be: 8 11 13 4 19 17 36

### 1.5.2

it should be: 11 10 10 5 21 15 36