Reduction

De lo que se trata es de efectuar lo mas rapidamente posible la suma de todos los valores de un vector. Si cada *block* lanza un determinado numero de *threads* y la suma de los calculos se almacena en una variable temporal despues podemos hacer *folding* de este array para calcular la suma.

Para calcular lanzamos un cierto numero de *threads* y en cada uno acumulamos la porcion del resultado que le corresponde,

```
__global__ void network (float *matrix, float *result){
    __shared__ float tvalue[NPROC];
    int i = blockIdx.x + blockIdx.y*shape.imd[0]+ blockIdx.z*shape.plane;
    int j = threadIdx.x;
    int index = threadIdx.x;
    float temp = 0;

    while (j<matrix.vol) {
        temp += matrix_function(i,j);
        j += blockDim.x;
    }
    tvalue[index] = temp;

    __syncthreads();
        // Aqui iria la suma por reduction

    return;
}
```

Despues de sincronizar los *threads* se hace la suma de todos los valores

```
    size_t k = blockDim.x/2;
    while (k!=0) {
        if (index < k) {
            tvalue[index] += tvalue[index + k];
        }
        __syncthreads();
        k/=2;
    }
    if (index == 0){
        result[i] = tnet[0];
    }
```

El codigo completo seria algo asi

```
__global__ void network (float *matrix, float *result){
    __shared__ float tvalue[NPROC];
    int i = blockIdx.x + blockIdx.y*shape.imd[0]+ blockIdx.z*shape.plane;
    int j = threadIdx.x;
    int index = threadIdx.x;
    float temp = 0;
```

```
    while (j<matrix.vol) {
        temp += matrix_function(i,j);
        j += blockDim.x;
    }
    tvalue[index] = temp;

    __syncthreads();
    size_t k = blockDim.x/2;
    while (k!=0) {
        if (index < k) {
            tvalue[index] += tvalue[index + k];
        }
        __syncthreads();
        k/=2;
    }
    if (index == 0){
        result[i] = tnet[0];
    }
    return;
}
```