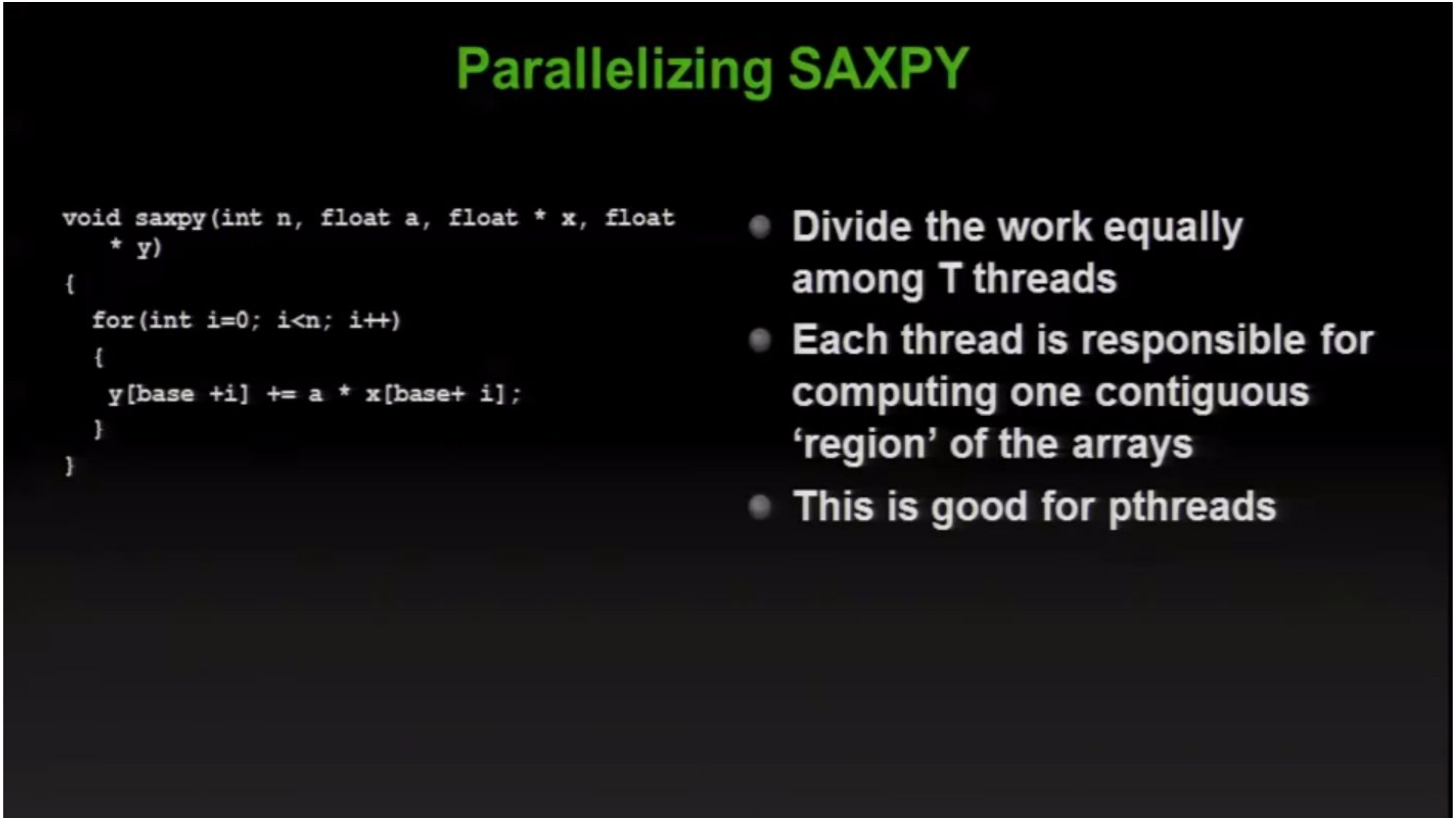
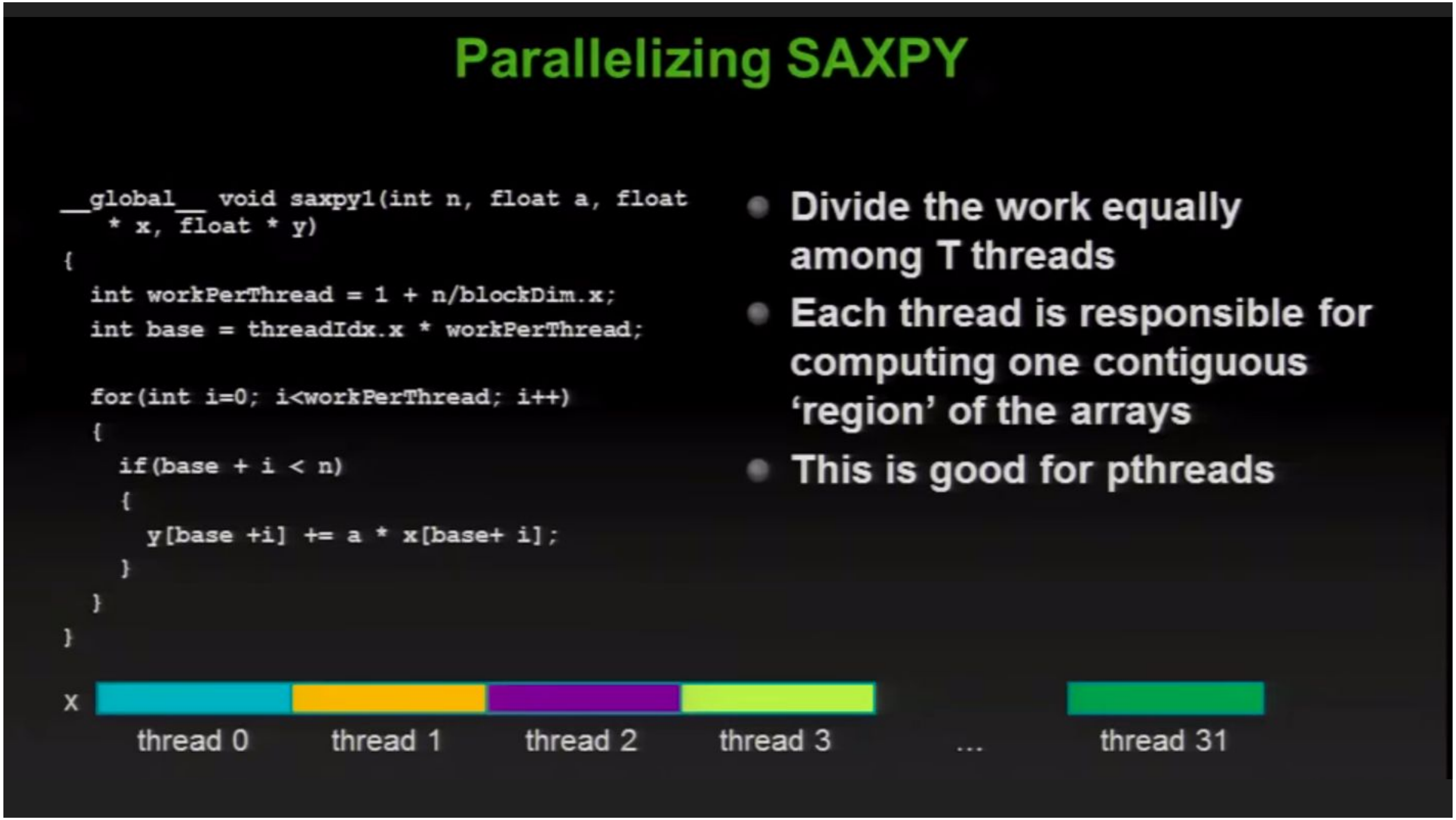


Parallelizing SAXPY

```
void saxpy(int n, float a, float * x, float
   * y)
{
  for(int i=0; i<n; i++)
  {
   y[base +i] += a * x[base+ i];
  }
}
```

- Divide the work equally among T threads
- Each thread is responsible for computing one contiguous ‘region’ of the arrays
- This is good for pthreads

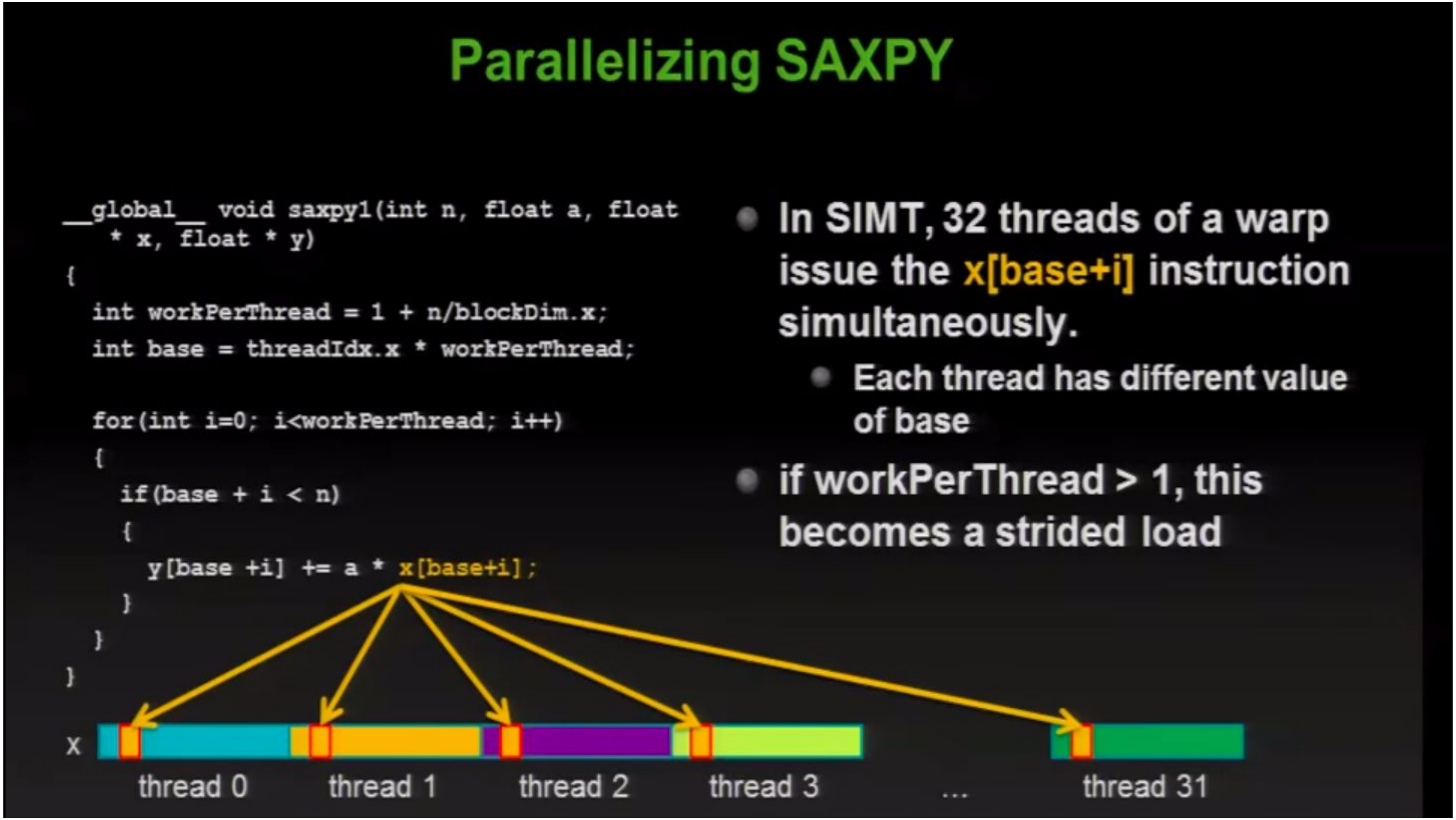


Parallelizing SAXPY

```
__global__ void saxpy1(int n, float a, float
  * x, float * y)
{
  int workPerThread = 1 + n/blockDim.x;
  int base = threadIdx.x * workPerThread;

  for(int i=0; i<workPerThread; i++)
  {
    if(base + i < n)
    {
      y[base +i] += a * x[base+ i];
    }
  }
}
```

- Divide the work equally among T threads
- Each thread is responsible for computing one contiguous ‘region’ of the arrays
- This is good for pthreads

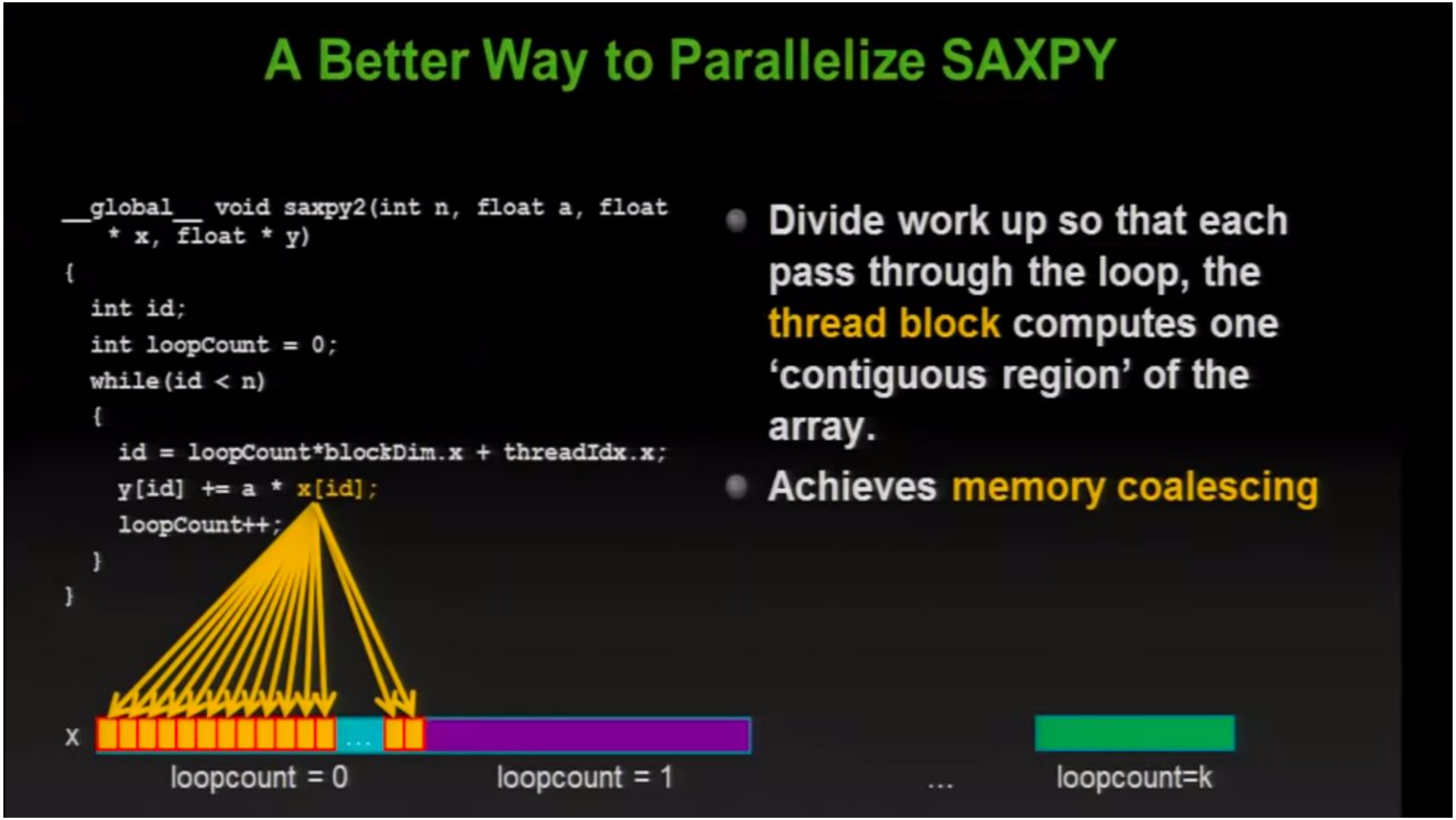


Parallelizing SAXPY

```
__global__ void saxpy1(int n, float a, float
  * x, float * y)
{
  int workPerThread = 1 + n/blockDim.x;
  int base = threadIdx.x * workPerThread;

  for(int i=0; i<workPerThread; i++)
  {
    if(base + i < n)
    {
      y[base +i] += a * x[base+i];
    }
  }
}
```

- In SIMT, 32 threads of a warp issue the x[base+i] instruction simultaneously.
  - Each thread has different value of base
- if workPerThread > 1, this becomes a strided load



A Better Way to Parallelize SAXPY

```
__global__ void saxpy2(int n, float a, float
  * x, float * y)
{
  int id;
  int loopCount = 0;
  while(id < n)
  {
    id = loopCount*blockDim.x + threadIdx.x;
    y[id] += a * x[id];
    loopCount++;
  }
}
```

- Divide work up so that each pass through the loop, the thread block computes one ‘contiguous region’ of the array.
- Achieves memory coalescing