## 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];
   }
}</pre>
```

- 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];
        }
    }
}</pre>
```

X

thread 0

- Divide the work equally among T threads
- Each thread is responsible for computing one contiguous 'region' of the arrays

thread 31

This is good for pthreads

thread 1 thread 2 thread 3 ...

## Parallelizing SAXPY

```
global void saxpy1(int n, float a, float
                                           In SIMT, 32 threads of a warp
 * x, float * y)
                                            issue the x[base+i] instruction
int workPerThread = 1 + n/blockDim.x;
                                            simultaneously.
int base = threadIdx.x * workPerThread:
                                              Each thread has different value
                                                of base
for(int i=0; i<workPerThread; i++)
                                         if workPerThread > 1, this
 if(base + i < n)
                                            becomes a strided load
   y[base +i] += a * x[base+i];
   thread 0
                                       thread 3
                                                               thread 31
               thread 1
                           thread 2
```

## 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++;
}
}</pre>
```

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

loopcount=k