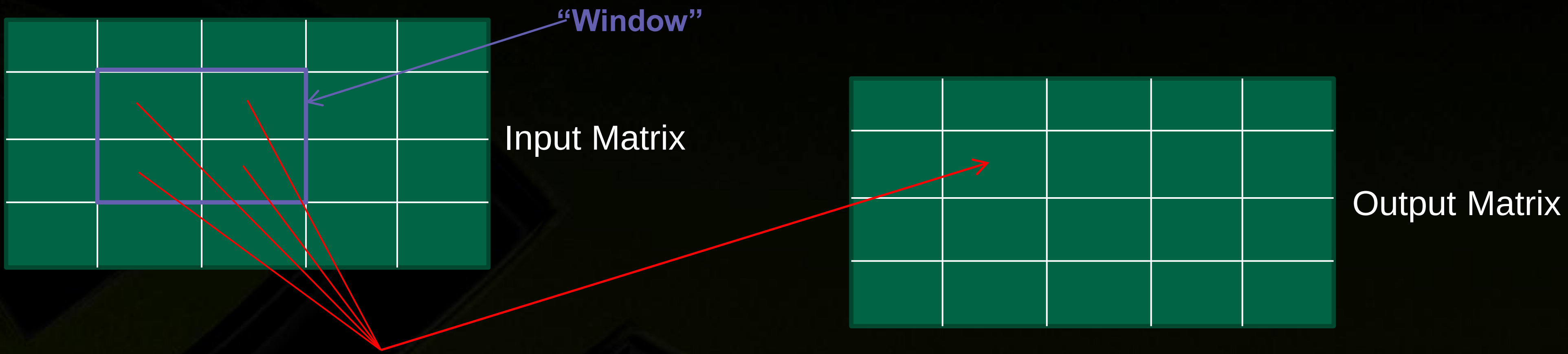
Accelerate your Application with CUDA C

2D Minimum Algorithm

- Consider applying a 2D window to a 2D array of elements
  - Each output element is the minimum of input elements within the window



- 2D operations like this are found in many fundamental algorithms
  - Interpolation, Convolution, Filtering
- Applications in seismic processing, weather simulation, image processing, etc

2D Minimum: C Version

```c
#define WIN_SIZE 16
#define N 20000

int main() {
  int size=N*N*sizeof(int);
  int i, j, x, y, temp;
  //allocate resources
  int *cell=(int *)malloc(size);  //input
  int *node=(int *)malloc(size);  //output

  initializeArray(cell,N);
  initializeArray(node,N);
```

Allocate Resources

Initialize data

```c
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
    {
   //find minimum in window
      temp = node[i][j];
      for(x=0;x<WIN_SIZE;x++)
        for(y=0;y<WIN_SIZE;y++)
             if (temp> cell[i+x][j+y])
                temp = cell[i+x][j+y];
       node[i][j] = temp;
     }

 //free resources
  free(cell); free(node);
}
```

Loop over dataset

Loop over window

Find min

Cleanup

http://developer.nvidia.com/cuda

2D Minimum: C Version

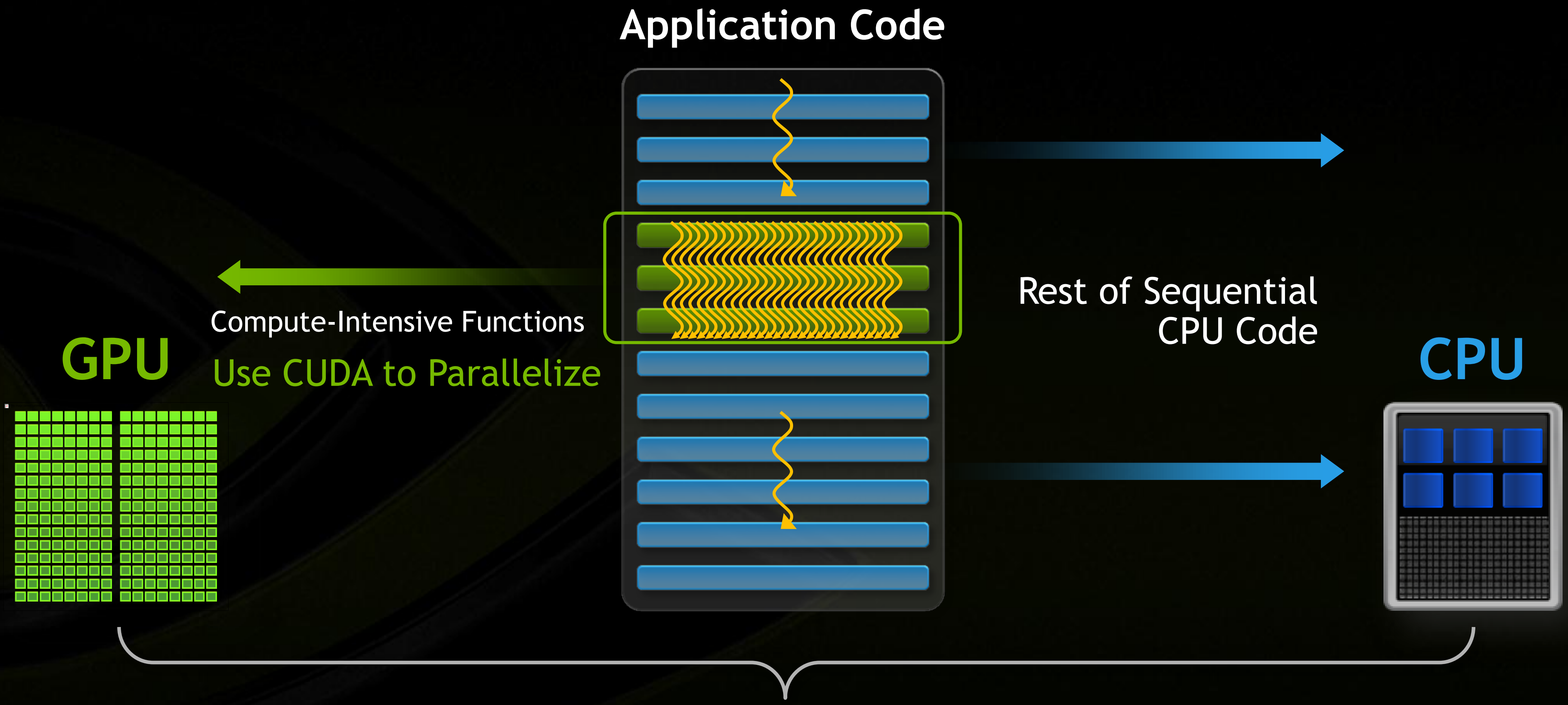
```
#define WIN_SIZE 16
#define N 20000

int main() {
  int size=N*N*sizeof(int);
  int i, j, x, y, temp;
  //alloca
  int *cel
  int *no

  initializeArray(cell,N);
  initializeArray(node,N);
```

```
for(i=0;i<N;i++)
     for(j=0;j<N;j++)
     {
    //find minimum in window
        temp = node[i][j];

                       temp = cell[i+x][j+y];
         node[i][j] = temp;
       }

 //free resources
  free(cell); free(node);
}
```

Algorithm	Device	Compute	Speedup
C (serial)	Xeon X5650	96.1sec	--

http://developer.nvidia.com/cuda

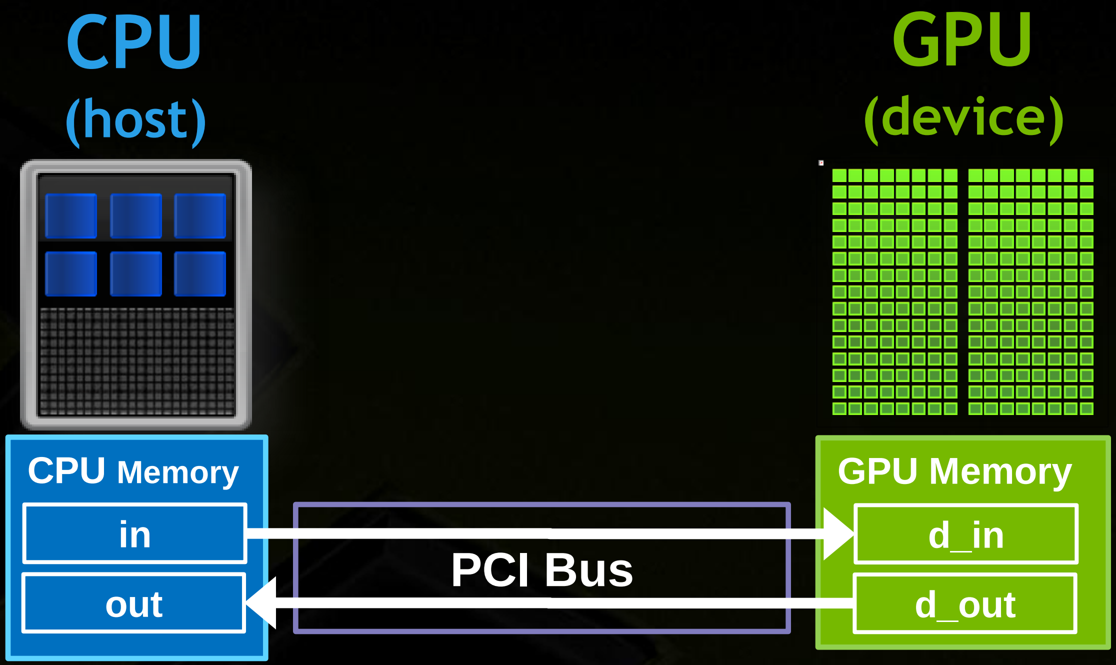
CPU + GPU = Big Speedup



+

http://developer.nvidia.com/cuda

Explicit Data Movement



http://developer.nvidia.com/cuda

2D Window: CUDA – Main Program

C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(int*)malloc(size); //input
  int *node=(int*)malloc(size); //output

  initializeArray(cell,N);
  initializeArray(node,N);
// nested for loops
  for…
    for …
     for …
       for ….
//free resources
  free(in); free(out);
}
```

CUDA C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(float*)malloc(size);
  int *node=(float*)malloc(size);
  int *d_cell; cudaMalloc(&d_cell,size);
  int *d_node; cudaMalloc(&d_node,size);
   initializeArray(cell,N); initializeArray(node,N);

   cudaMemcpy(d_cell,cell,size,cudaMemcpyHostToDevice);
   cudaMemcpy(d_node,node,size,cudaMemcpyHostToDevice);
   compute_win2D<<<nblocks, nthreads>>>(d_node,d_cell);
  //free resources
  free(cell); free(node);
  cudaFree(d_cell); cudaFree(d_node);
}
```

http://developer.nvidia.com/cuda

2D Window: CUDA – Main Program

C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(int*)malloc(size); //input
  int *node=(int*)malloc(size); //output

  initializeArray(cell,N);
  initializeArray(node,N);
// nested for loops
  for…
    for …
     for …
       for ….
//free resources
  free(in); free(out);
}
```

CUDA C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(float*)malloc(size);
  int *node=(float*)malloc(size);
  int *d_cell; cudaMalloc(&d_cell,size);
  int *d_node; cudaMalloc(&d_node,size);
   initializeArray(cell,N); initializeArray(node,N);

   cudaMemcpy(d_cell,cell,size,cudaMemcpyHostToDevice);
   cudaMemcpy(d_node,node,size,cudaMemcpyHostToDevice);
   compute_win2D<<<nblocks, nthreads>>>(d_node,d_cell);
  //free resources
  free(cell); free(node);
  cudaFree(d_cell); cudaFree(d_node);
}
```

Allocate Device Memory

http://developer.nvidia.com/cuda

2D Window: CUDA – Main Program

C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(int*)malloc(size); //input
  int *node=(int*)malloc(size); //output

  initializeArray(cell,N);
  initializeArray(node,N);
// nested for loops
  for…
    for …
     for …
       for ….
//free resources
  free(in); free(out);
}
```

CUDA C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(float*)malloc(size);
  int *node=(float*)malloc(size);
  int *d_cell; cudaMalloc(&d_cell,size);
  int *d_node; cudaMalloc(&d_node,size);
   initializeArray(cell,N); initializeArray(node,N

   cudaMemcpy(d_cell,cell,size,cudaMemcpyHostToDevice);
   cudaMemcpy(d_node,node,size,cudaMemcpyHostToDevice);
   compute_win2D<<<nblocks, nthreads>>>(d_node,d_cell);
  //free resources
  free(cell); free(node);
  cudaFree(d_cell); cudaFree(d_node);
}
```

Copy Data to the Device

http://developer.nvidia.com/cuda

2D Window: CUDA – Main Program

C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(int*)malloc(size); //input
  int *node=(int*)malloc(size); //output

  initializeArray(cell,N);
  initializeArray(node,N);
// nested for loops
  for…
    for …
     for …
       for ….
//free resources
  free(in); free(out);
}
```

CUDA C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(float*)malloc(size);
  int *node=(float*)malloc(size);
  int *d_cell; cudaMalloc(&d_cell,size);
  int *d_node; cudaMalloc(&d_node,size);
   initializeArray(cell,N); initializeArray(node,N);

   cudaMemcpy(d_cell,cell,size,cudaMemcpyHostToDevice);
   cudaMemcpy(d_node,node,size,cudaMemcpyHostToDevice);
   compute_win2D<<<nblocks, nthreads>>>(d_node,d_cell);
  //free resources
  free(cell); f
  cudaF                                   _node);
}
```

Call Cuda Function

Launch Parameters

Device Pointers

http://developer.nvidia.com/cuda

2D Window: CUDA – Main Program

C

```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(int*)malloc(size); //input
  int *node=(int*)malloc(size); //output

  initializeArray(cell,N);
  initializeArray(node,N);
// nested for loops
  for…
    for …
     for …
       for ….
//free resources
  free(in); free(out);
}
```

CUDA C

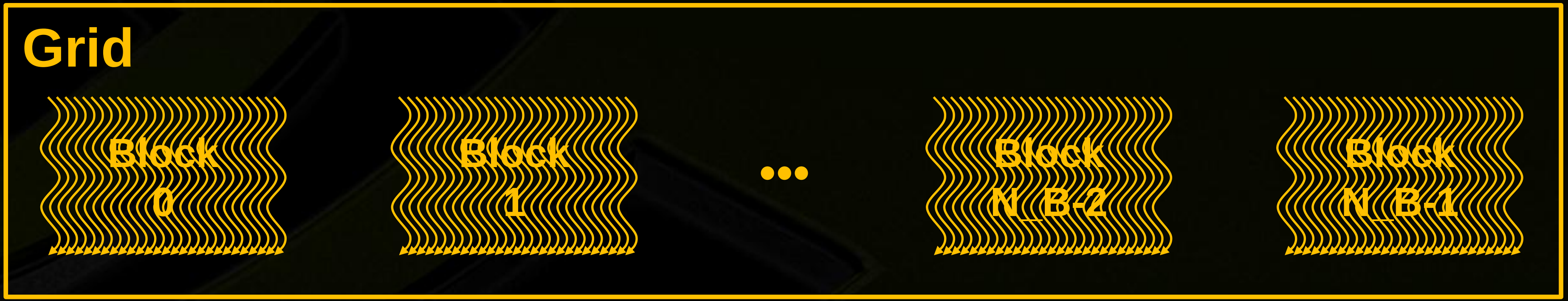
```
int main() {
  int size=N*N*sizeof(int);
  //allocate resources
  int *cell=(float*)malloc(size);
  int *node=(float*)malloc(size);
  int *d_cell; cudaMalloc(&d_cell,size);
  int *d_node; cudaMalloc(&d_node,size);
   initializeArray(cell,N); initializeArray(node,N);

   cudaMemcpy(d_cell,cell,size,cudaMemcpyHostToDevice);
   cudaMemcpy(d_node,node,size,cudaMemcpyHostToDevice);
   compute_win2D<<<nblocks, nthreads>>>(d_node,d_cell);
  //free resources
  free(cell); free(node);
  cudaFree(d_cell); cudaFree(d_node);
}
```

Cleanup

http://developer.nvidia.com/cuda

Parallel Execution Model

- A CUDA C function is executed by many parallel threads
- Threads are organized as a grid of independent thread blocks



http://developer.nvidia.com/cuda

2D Window: CUDA – Kernel Function

C

```
for(i=0;i<N;i++)
     for(j=0;j<N;j++)
     {
    //find minimum in window
        temp = node[i][j];
        for(x=0;x<WIN_SIZE;x++)
          for(y=0;y<WIN_SIZE;y++)
                 if (temp> cell[i+x][j+y])
                     temp = cell[i+x][j+y];
         node[i][j] = temp;
       }
```

CUDA C

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {
 //find minimum in window
        temp = knode[idx][idy];
        for(x=0;x<WIN_SIZE;x++)
          for(y=0;y<WIN_SIZE;y++)
                 if (temp> kcell[idx+x][idy+y])
                     temp = kcell[idx+x][idy+y];
         knode[i][j] = temp;
}
```

http://developer.nvidia.com/cuda

2D Window: CUDA – Kernel Function

C

```
for(i=0;i<N;i++)
     for(j=0;j<N;j++)
     {
    //find minimum in window
        temp = node[i][j];
        for(x=0;x<WIN_SIZE;x++)
          for(y=0;y<WIN_SIZE;y++)
                 if (temp> cell[i+x][j+y])
                     temp = cell[i+x][j+y];
         node[i][j] = temp;
       }
```

Add __global__ Keyword

CUDA C

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {
 //find minimum in window
        temp = knode[idx][idy];
        for(x=0;x<WIN_SIZE;x++)
          for(y=0;y<WIN_SIZE;y++)
                 if (temp> kcell[idx+x][idy+y])
                     temp = kcell[idx+x][idy+y];
         knode[i][j] = temp;
}
```

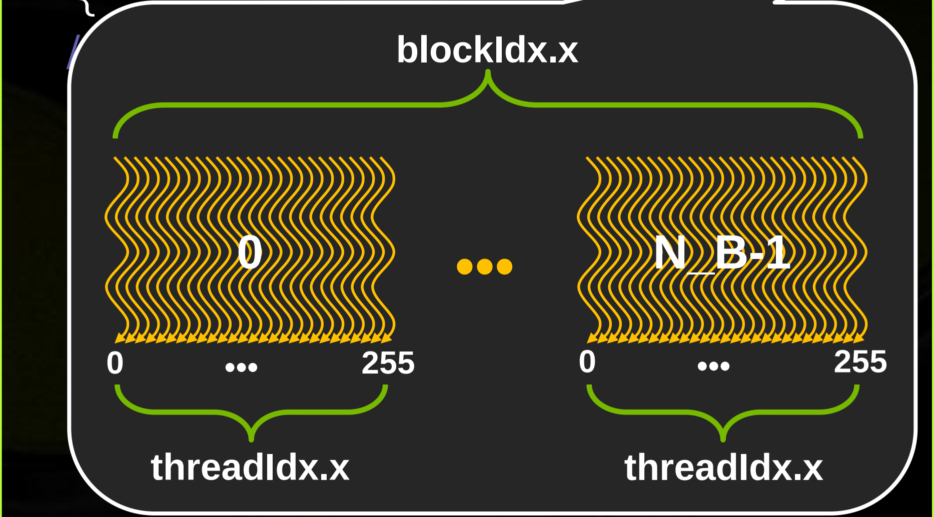
http://developer.nvidia.com/cuda

2D Window: CUDA – Kernel Function

C

Replace Outer Loops With an Index Calculation

```
for(i=0;i<N;i++)
    for(j=0;j<N;j++)
    {
    /
```



CUDA C

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {
 //find minimum in window
       temp = knode[idx][idy];
       for(x=0;x<WIN_SIZE;x++)
         for(y=0;y<WIN_SIZE;y++)
                if (temp> kcell[idx+x][idy+y])
                    temp = kcell[idx+x][idy+y];
        knode[i][j] = temp;
}
```

2D Window: CUDA – Kernel Function

C

```
for(i=0;i<N;i++)
     for(j=0;j<N;j++)
     {
    //find minimum
        temp = nod
        for(x=0;x<V
          for(y=0;y<
               if (ten
                   temp = cell[i+x][j+y];
         node[i][j] = temp;
       }
```

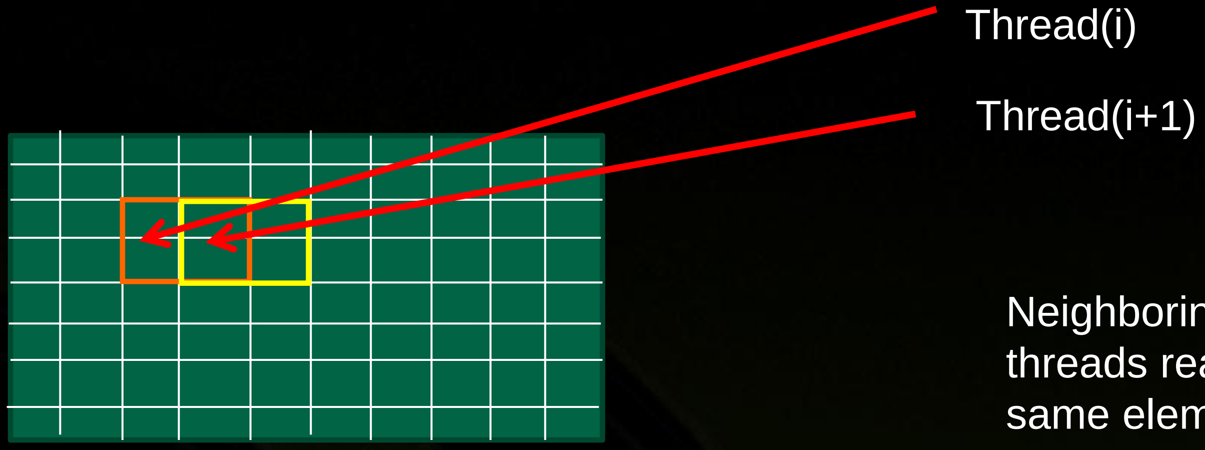
CUDA C

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp, x, y;
                if (temp> kcell[idx+x][idy+y])
                     temp = kcell[idx+x][idy+y];
          knode[i][j] = temp;
}
```

Algorithm	Device	Compute	Speedup
C (serial)	X5650	96.1sec	--
CUDA	M2090	8.33sec	11.5x

http://developer.nvidia.com/cuda

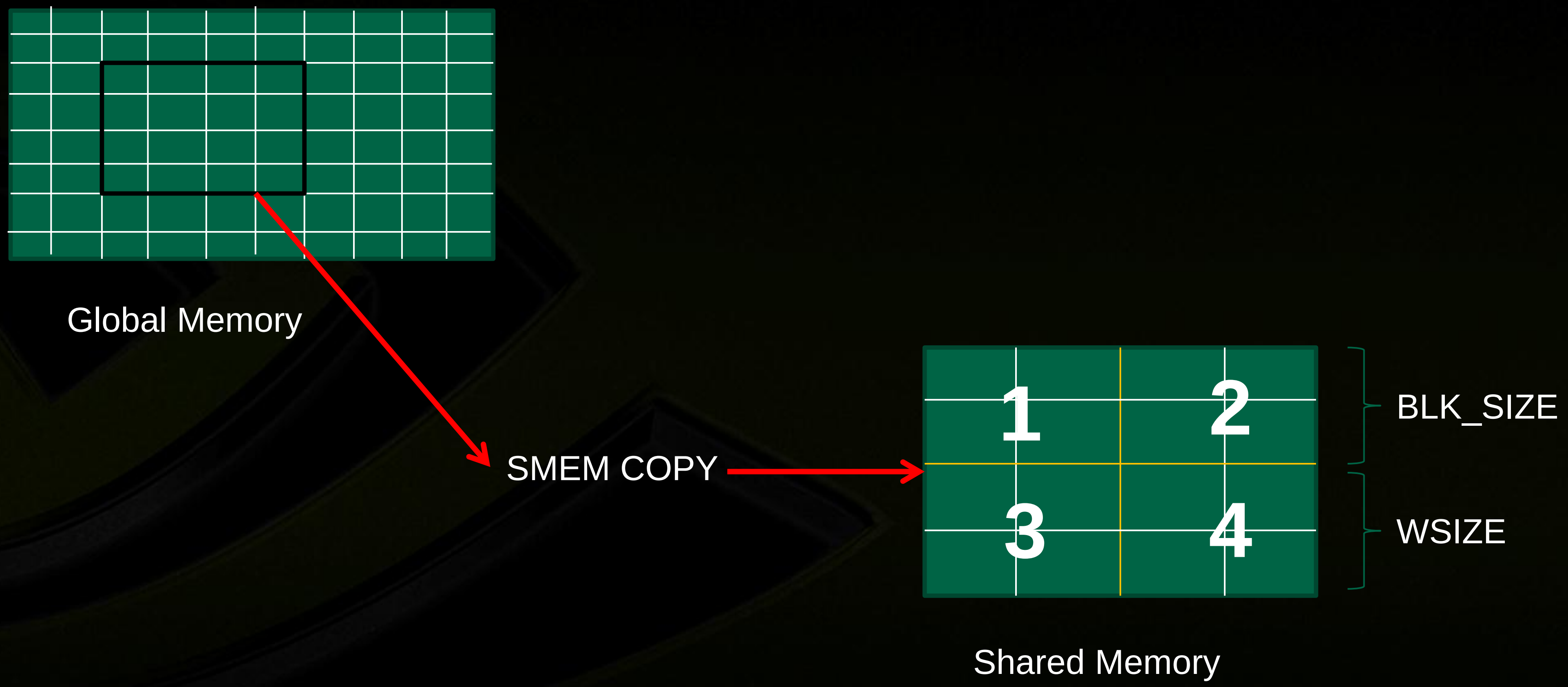
Observation: Data Reuse



Global Memory

Neighboring threads read the same elements.

Optimization: Use Shared Memory



http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                      kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

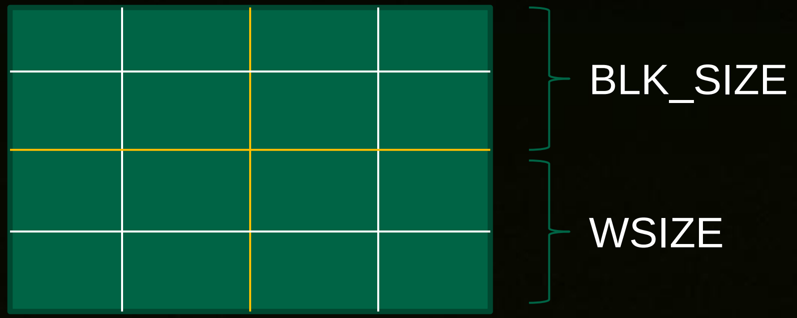
```
 //find minimum in window
    temp = knode[idx][idy];
    for(x=0;x<WSIZE;x++)
      for(y=0;y<WSIZE;y++)
        if (temp> smem[threadIdx.x+x][threadIdx.y+y])
          temp = smem[threadIdx.x+x][threadIdx.y+y];
        knode[i][j] = temp;
    }
}
```

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                          kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Allocate Shared Memory for Each Block



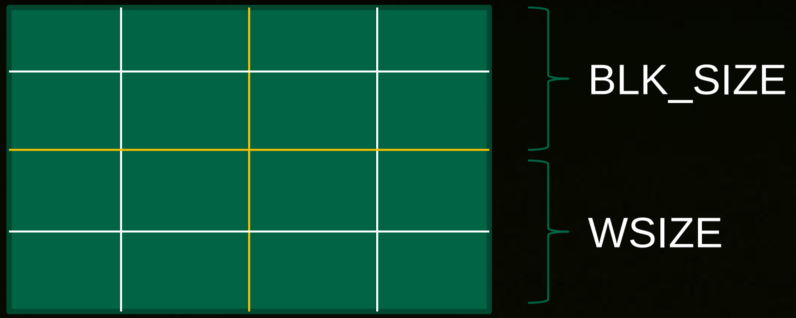
Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                         kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Calculate Global Memory Indices



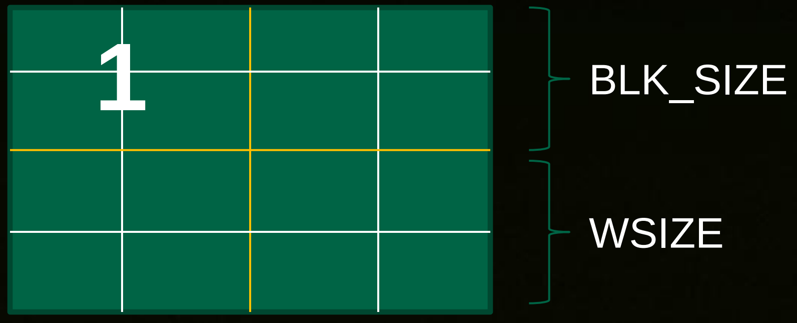
Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                        kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Load Interior Region 1 to Shared Memory



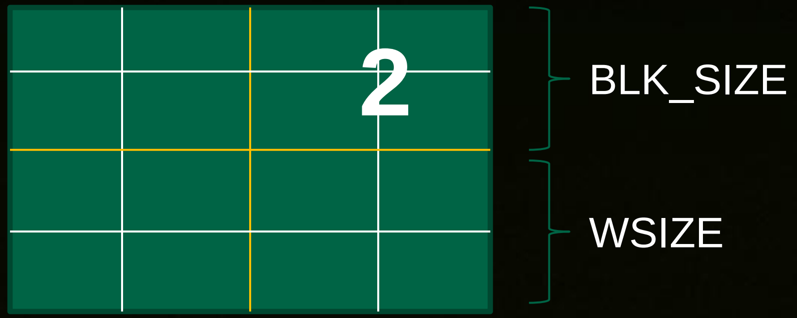
Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                          kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Load Halo Region 2 to Shared Memory



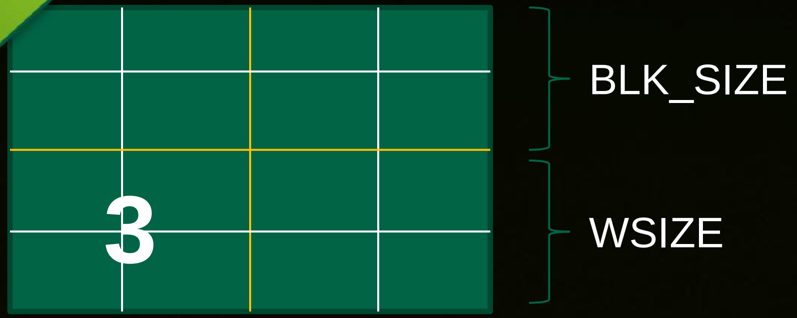
Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                         kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Load Halo Region 3 to Shared Memory



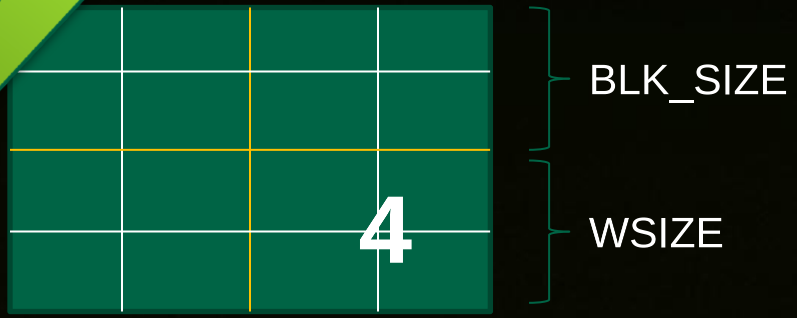
Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                        kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Load Halo Region 4 to Shared Memory



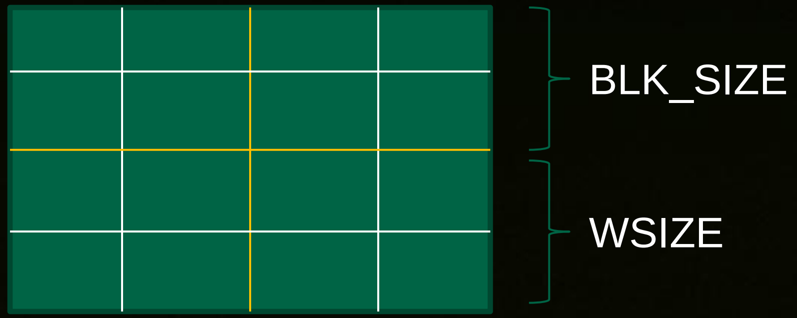
Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                      kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

Wait for All Threads to Finish Writing to Shared Memory



Shared Memory

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                        kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();

 //find minimum in window
   temp = knode[idx][idy];
   for(x=0;x<WSIZE;x++)
     for(y=0;y<WSIZE;y++)
       if (temp> smem[threadIdx.x+x][threadIdx.y+y])
         temp = smem[threadIdx.x+x][threadIdx.y+y];
       knode[i][j] = temp;
   }
}
```

Find Minimum: Read Input from Shared Memory, Accumulate into a Register

http://developer.nvidia.com/cuda

CUDA C: Optimized Kernel

```
__global__ void compute_win2D(int knode[][N], int kcell[][N])
{
 __shared__ int smem[BLK_SIZE+WSIZE][BLK_SIZE+WSIZE];
  int  idx=blockIdx.x*blockDim.x+threadIdx.x;
  int  idy=blockIdx.y*blockDim.y+threadIdx.y;
  int  temp,  x, y;
  if((idx<N)&&(idy<N)) {

  smem[threadIdx.x][threadIdx.y]=kcell[idx][idy];
  if (threadIdx.y > (N-WSIZE))
    smem[threadIdx.x][threadIdx.y + WSIZE]=kcell[idx][idy+WSIZE];
  if (threadIdx.x >(N-WSIZE))
    smem[threadIdx.x + WSIZE][threadIdx.y]=kcell[idx+WSIZE][idy];
  if ((threadIdx.x >(N-WSIZE)) && (threadIdx.y > (N-WSIZE)))
   smem[threadIdx.x+WSIZE][threadIdx.y+WSIZE]=
                       kcell[idx+WSIZE][idy+WSIZE];
 //wait for all threads to finish read
  __syncthreads();
```

```
 //find minimum in window
    temp = knode[idx][idy];
    for(x=0;x<WSIZE;x++)
      for(y=0;y<WSIZE;y++)
         if (temp> smem[threadIdx.x+x][threadIdx.y+y])
           temp = smem[threadIdx.x+x][threadIdx.y+y];
         knode[i][j] = temp;
    }
}
```

Write Minimum to Global Memory

Questions?