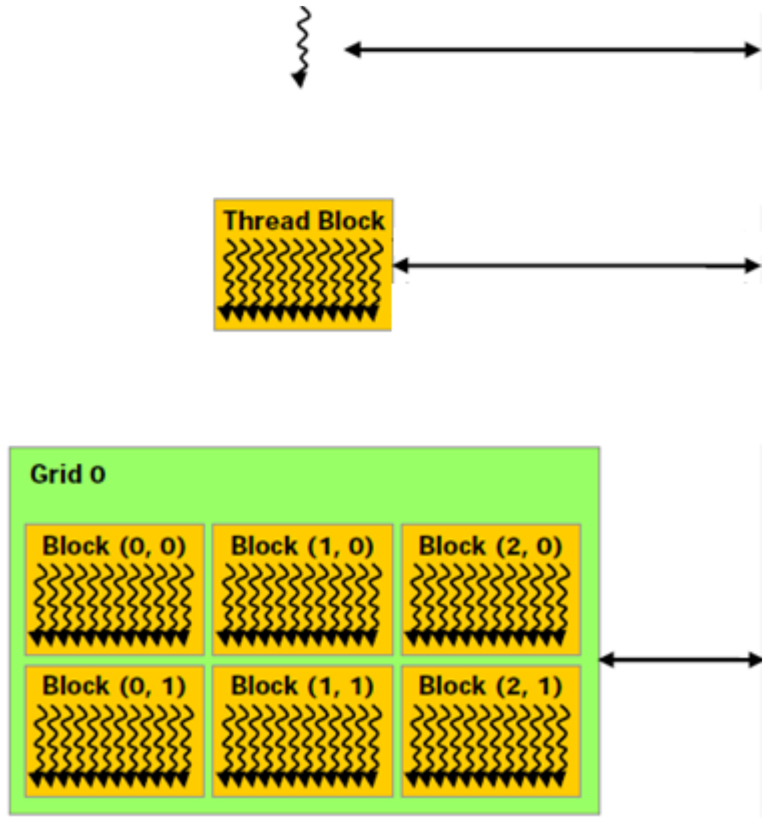
Introduction to CUDA

T.V. Singh
Institute for Digital Research and Education
UCLA
tvsingh@ucla.edu

Threads Execution and Warps

- Blocks and Grids of Blocks



WARP

- SIMD hardware executes all threads of a warp as a bundle.
- Divergence in warp
  - If loop condition is based on thread index
    - for (i=0, I < threadIdx.x; i++){}
  - If conditions
    - If(threadIdx > 2) {}

UCLA IDRE

Types of Memories

- Registers
  - fadd r1, r2, r3
- Shared memory
  - load r1, r2, offset
  - fadd r1,r2,r3
- Global memory
  - foad r1,r2, offset
  - fadd r1, r2, r3

Memory coalescing

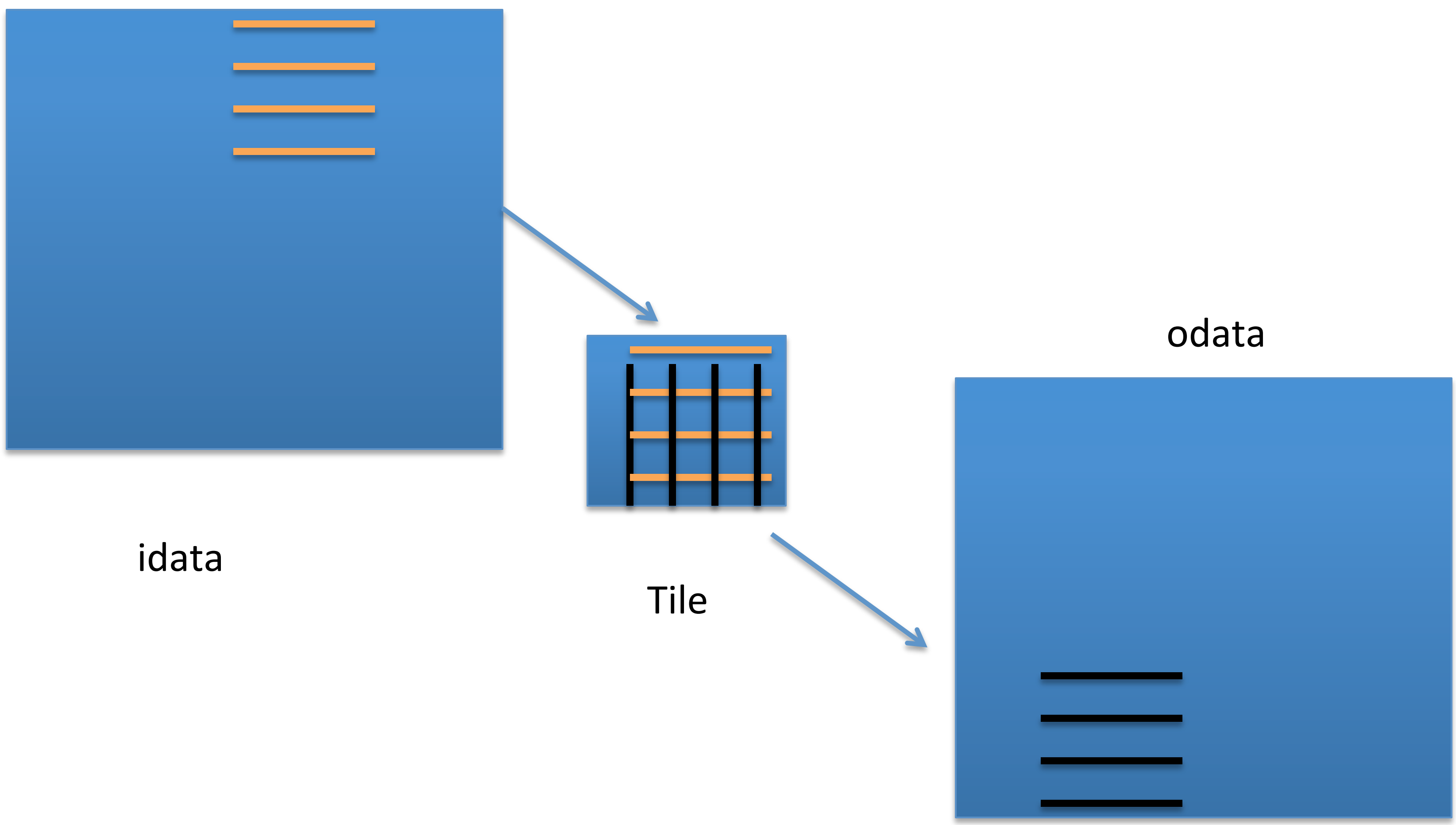
- Global memory access by a half warp coalesced in to a single transaction if:
  - It accesses contiguous region of global memory:
    - 64 bytes - each thread reads 4 bytes.
    - 128 bytes - each thread reads 8 bytes
    - 256 bytes – each thread reads 16 bytes
- This defines how the optimum memory bandwidth can be achieved on GPU

UCLA IDRE

Example-With shared memory

```
#include <stdlib.h>
#include <stdio.h>
#include "cuda.h“
#define BLOCK_DIM  16

__global__ void transpose(float *odata, float *idata, int W, int H)
{
    __shared__ float block[BLOCK_DIM][BLOCK_DIM+1];

    // read the matrix tile into shared memory
    unsigned int xIndex = blockIdx.x * BLOCK_DIM + threadIdx.x;
    unsigned int yIndex = blockIdx.y * BLOCK_DIM + threadIdx.y;

    if((xIndex < W) && (yIndex < H)){
        unsigned int index_in = yIndex * W + xIndex;
        block[threadIdx.y][threadIdx.x] = idata[index_in];}

    __syncthreads();

    // write the transposed matrix tile to global memory
    xIndex = blockIdx.y * BLOCK_DIM + threadIdx.x;
    yIndex = blockIdx.x * BLOCK_DIM + threadIdx.y;

    if((xIndex < H) && (yIndex < W)){
        unsigned int index_out = yIndex * H + xIndex;
        odata[index_out] = block[threadIdx.x][threadIdx.y];}
}
```

Transpose using Shared Memory



Sum reduction example-1

	Thread -0	Thread -1	Thread -2	Thread -3	Thread -4	Thread -5	Thread -6	Thread -7
Step-1	0	1	2	3	4	5	6	7
Step-2	0+1		2+3		4+5		6+7	
Step-3	0+2				4+7			
Step-4	0+4							

Sum reduction example-1

```
__shared__ float psum[];

unsigned int t = threadIdx.x;
i = blockIdx.y*blockDim.y+threadIdx.y;

psum[t] = x[i]
__syncthreads();

for (int stride =1; stride < blockDim.x; stride *=2){

__syncthreads();

If (t % (2*stride) == 0)pum[t] += psum[t+stride];
}
```

UCLA IDRE

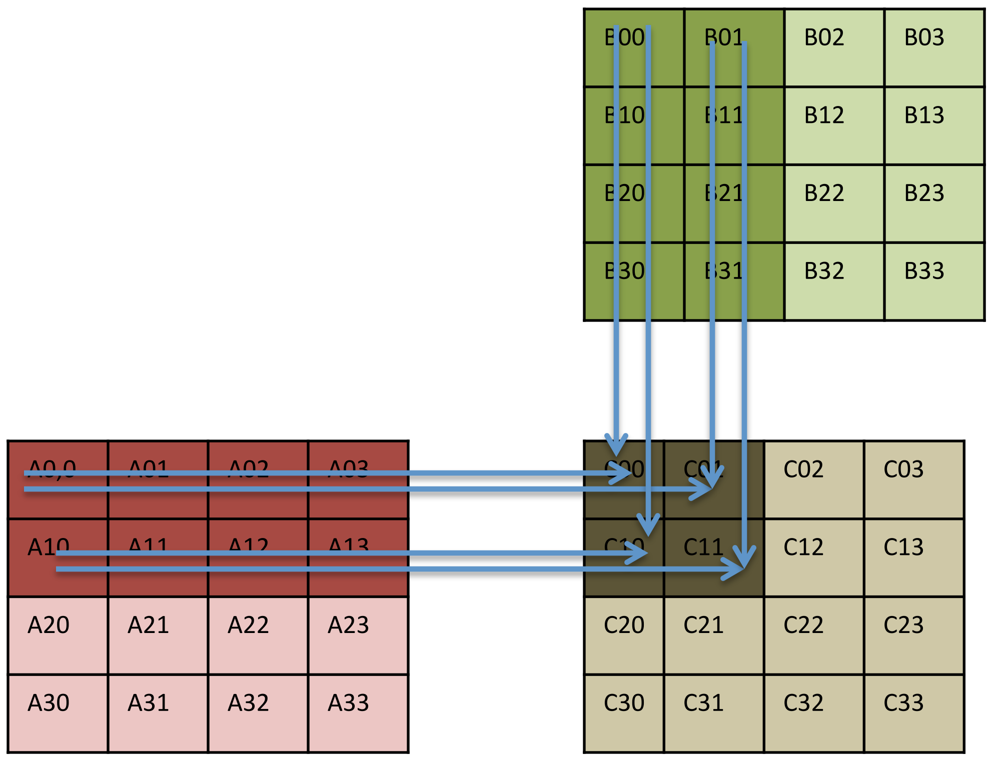
Sum reduction example-2

Thread-0	Thread-1	Thread-2	Thread-3	Thread-4	Thread-5	Thread-6	Thread-7
0	1	2	3	4	5	6	7
0+8	1+9	2+10	3+11	4+12	5+13	6+14	7+15
0+4	1+5	2+6	3+7				
0+2	1+3						
0+2							

Sum reduction example-2

```
__shared__ float psum[];

unsigned int t = threadIdx.x;
i = blockIdx.y*blockDim.y+threadIdx.y;

psum[t] = x[i]
__syncthreads();

for (int stride =blockDim.x; stride > 1; stride /=2){

__syncthreads();

If ( t < stride ) pum[t] += psum[t+stride];
}
```

Example – Matrix multiplication



Code-host

```
__host__ void matrixmul_h(float A[N][N], float B[N][N], float C[N][N])
{
for( int row =0; row < N; ++row){
  for( int col =0; col < N; ++col){
        float element = 0;
        for (int k = 0; k < N; ++k) element += A[row][k] * B[k][col];
        C[row][col] = element;
       }
     }
return;
}
```

UCLA IDRE

Code-GPU

```
__global__ void matrixmul_d(float* A, float* B, float* C)
{
unsigned int row   = blockIdx.y*blockDim.y+threadIdx.y;
unsigned int col   = blockIdx.x*blockDim.x+threadIdx.x;

if( (row < N) && (col < N) ) {
        float element = 0;
        for (int k = 0; k < N; ++k) element += A[row*N+k] * B[k*N+col];
        C[row*N+col] = element;
     }
return;
}
```

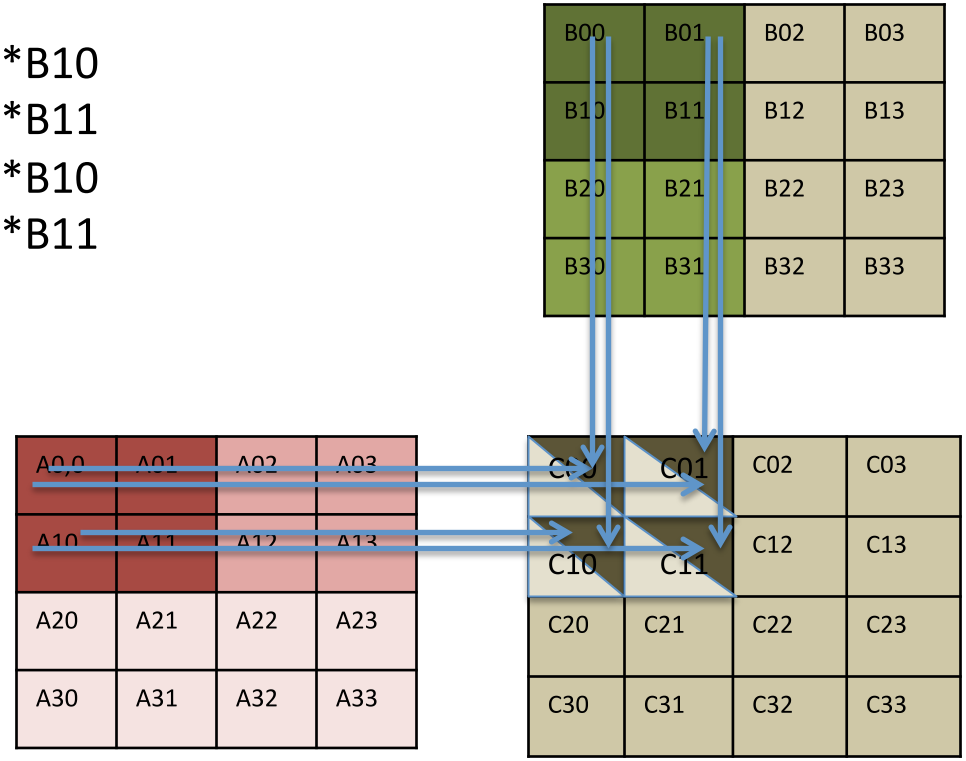
Access Order

C00=A00, B00+A01,B10+ A02,B20+A03,B 03	C01=A00, B01+A01,B11+ A02,B21+A03,B 31		
C10=A10,B01+ A11,B10+A12,B 20+A13,B03	C11=A10, B01+A11,B11+ A12,B21+A13,B 31		

Tiled Matrix multiplication

C00-1=A00*B00+A01*B10
C01-1=A00*B01+A01*B11
C10-1=A10*B00+A11*B10
C11-1=A10*B01+A11*B11

C00=C00-1+C00-2
C01=C01-1+C01-2
C10=C10-1+C10-2
C11=C11-1+C11-2



UCLA IDRE

Code- Tiled

```
__global__ void matrixmul_d(float* A, float* B, float* C)
{

  __shared__ float A_ds[TILE_WIDTH][TILE_WIDTH];
  __shared__ float B_ds[TILE_WIDTH][TILE_WIDTH];
   unsigned int bx= blockIdx.x;
   unsigned int tx= threadIdx.x;
   unsigned int by= blockIdx.y;
   unsigned int ty= threadIdx.y;
   unsigned int row   = by*TILE_WIDTH+ty;
   unsigned int col   = bx*TILE_WIDTH+tx;

        float element = 0;
        for (int m = 0; m < N/TILE_WIDTH; ++m) {
                A_ds[tx][ty] = A[N*(m*row+TILE_WIDTH)+tx];
                B_ds[tx][ty] = B[N*col+m*TILE_WIDTH+ty];
        __syncthreads();
        for ( int k = 0; k < TILE_WIDTH; ++k) element += A_ds[tx][k]*B_ds[k][ty];
        __syncthreads();
      }
      C[row*N+col] = element;
      return;
}
```

Communication between host and device

- The bandwidth between host and device is much less than the memory bandwidth on the device
    - The communication between host and device is costly
    - Less data transfer between host and device is preferable for better performance
- Streams could help in computing while communicating

Stream

- Applications manage concurrency through streams.
- A sequence of commands that execute in order.
- Different streams may execute their commands out of order.
- Defined by creating a stream object.
    - Objects can be used as parameters to sequence of kernels and memory copies.

Stream example

```
#include<cuda.h>
#include<stdio.h>
# define N 1024
# define BLOCK_SIZE 64
__global__ void VecAdd(float* A, float* B, float* C){
                int j   = blockIdx.x*blockDim.x+threadIdx.x;
                if(j<N)C[j]=A[j]+B[j];
                return; }
main(){
float *a, *b, *ab;
int  vecsize = sizeof(float)*N;
        cudaMallocHost((void **)&a , 2*vecsize);
        cudaMallocHost((void **)&b , 2*vecsize);
        cudaMallocHost((void **)&ab ,2*vecsize);
float *a_d, *b_d, *ab_d;
        cudaMalloc((void **)&a_d , 2*vecsize);
        cudaMalloc((void **)&b_d , 2*vecsize);
        cudaMalloc((void **)&ab_d , 2*vecsize);
int i, j;
        for (j=0; j< 2*N; j++){
                        a[j] = rand()%100;
                        b[j] = rand()%100;
                        ab[j]=0.0; }
```

```
cudaStream_t stream[2];
for (i=0; i<2; ++i) cudaStreamCreate(&(stream[i]));

for (i=0; i<2; ++i) cudaMemcpyAsync(a_d+i*N, a+i*N, vecsize,
        cudaMemcpyHostToDevice, stream[i]);

for (i=0; i<2; ++i) cudaMemcpyAsync( b_d+i*N, b+i*N,
        vecsize   ,cudaMemcpyHostToDevice,stream[i]);

dim3 dimBlock(BLOCK_SIZE);
dim3 dimGrid (N/dimBlock.x);
if( N % BLOCK_SIZE != 0 ) dimGrid.x+=1;
for (i=0; i<2; ++i) VecAdd<<<dimGrid,dimBlock,0,stream[i]>>> (a_d
        +i*N,b_d+i*N,ab_d+i*N);

for (i=0; i<2; ++i)cudaMemcpyAsync(ab + i * N, ab_d + i * N, vecsize,
        cudaMemcpyDeviceToHost, stream[i]);
 cudaThreadSynchronize();
for (i=0; i<2; i++) cudaStreamDestroy(stream[i]);

cudaFree(a_d);   cudaFree(b_d);  cudaFree(ab_d);
cudaFreeHost(a);  cudaFreeHost(b);  cudaFreeHost(ab);
```

Stream..

- cudaStreamCreate(cudaStream_t &stream);
    - Creates Stream
- cudaMallocHost((void**)&hostPtr, size);
    - Page locked memory allocation on host
- cudaMemcpyAsync(to,from,size,direction,stream);
    - Asyncronous memory copy
- Kernel<<<gridDim, blockDim,0,stream>>>(…);
    - Asynchronous kernel launch
- cudaStreamDestroy(stream);
    - Delete the stream object
- cudaStreamQuery(stream);
    - Query stream
- cudaStreamSynchronize(stream);
    - Synchronize a single stream
- cudaThreadSynchronize();
    - Wait till all stream finishes

Event management

- Events could created in CUDA code
  - For example
    - To measure the time
    - To block CPU until a CUDA call finish
  - Functions:
    - cudaEvent_t start, stop;
    - cudaEventCreate(&start);
    - cudaEventCreate(&stop);
    - cudaEventRecord(start, 0);
    - kernel<<<gridDim, blockDim>>>(...);
    - cudaEventRecord(stop, 0);
    - cudaEventSynchronize(stop);
    - float etime;
    - cudaEventElapsedTime(&etime, start, stop);
    - cudaEventDestroy(start);
    - cudaEventDestroy(stop);