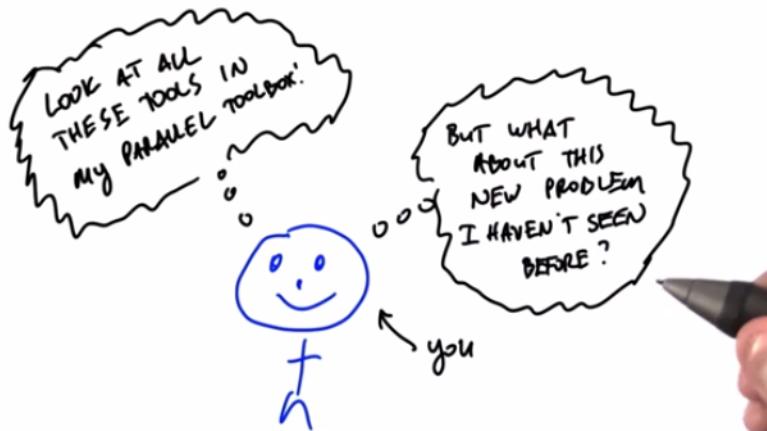


# CS344 Introduction to Parallel Programming

## Lesson 6.1: Parallel Computing Pattern Part A

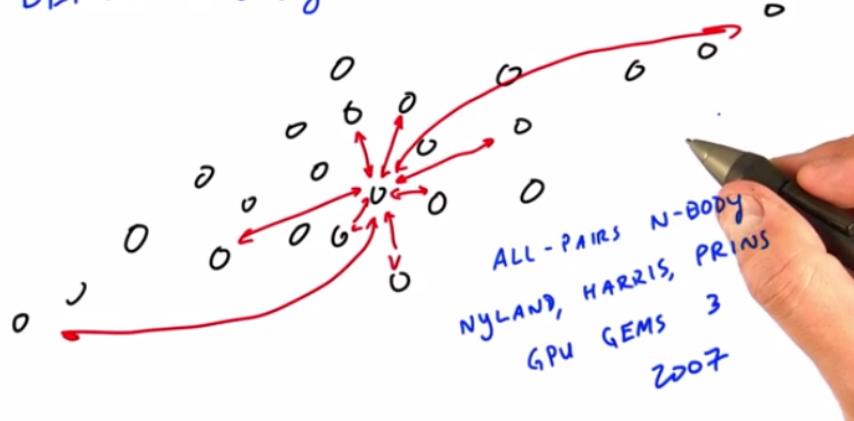
### L6.1-6.2-Adding to Your GPU Computing Toolbox

#### UNIT 6 · PARALLEL COMPUTING GRAB BAG



### L6.1-6.3-All Pairs N-Body

#### DENSE N-BODY



#### ALL-PAIRS N-BODY

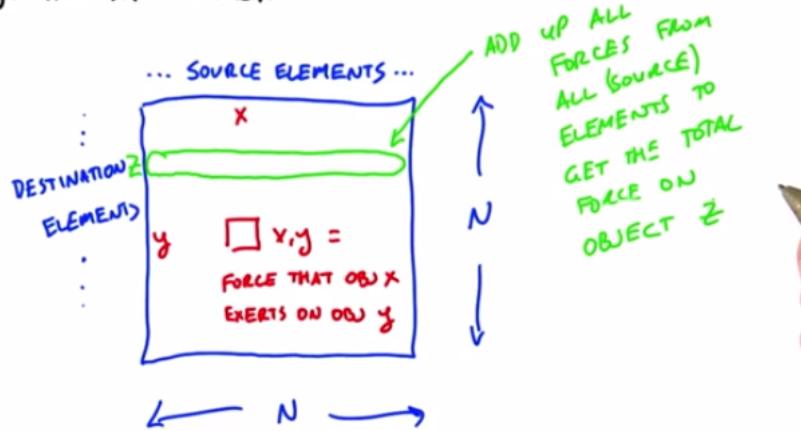
EACH OBJECT COMPUTES THE FORCES ON IT  
FROM EACH OTHER OBJECT.  $N$  OBJECTS.

TOTAL AMOUNT OF WORK? PROPORTIONAL TO ...

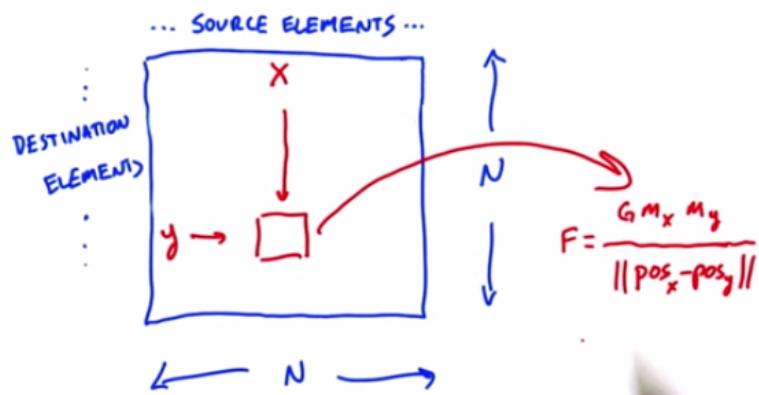
- $N$
- $N \log N$
- $N^2$
- $N^3$

### L6.1-6.4-How to Implement Dense N-Body as Simply As Possible

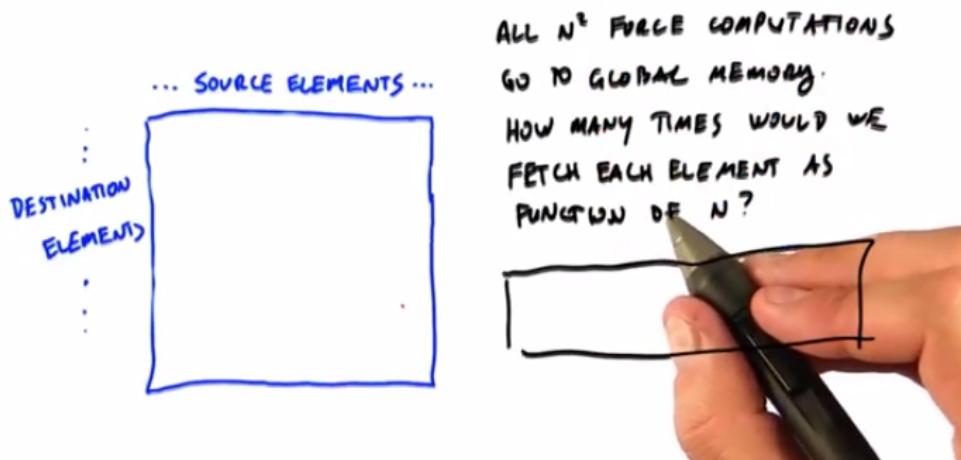
N-BODY: AN NXN MATRIX



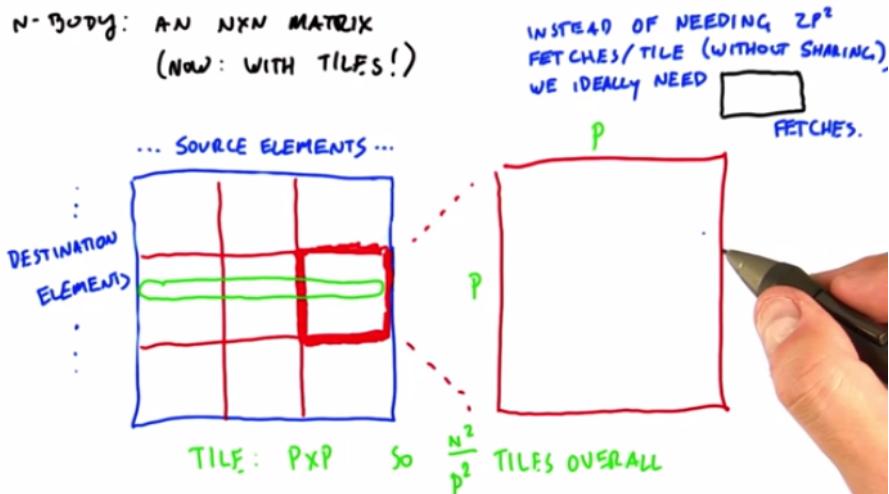
N-BODY: AN NXN MATRIX



N-BODY: AN NXN MATRIX



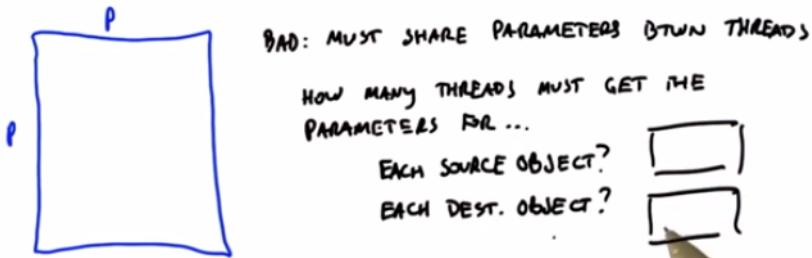
### L6.1-6.5-Dividing N by N Metric into Tiles



### L6.1-6.6-Downside to Using Tiles

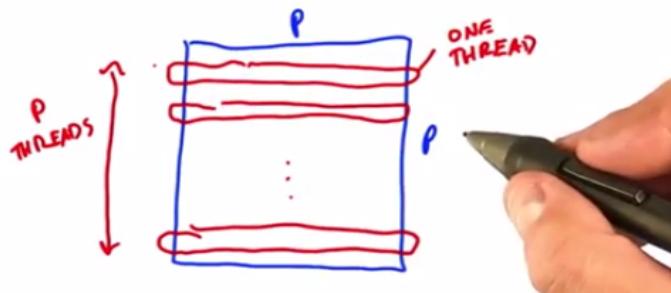
$P^2$  THREADS TO COMPUTE 1  $P \times P$  TILE

GOOD:  $P^2$  PARALLEL OPERATIONS



### L6.1-6.7-Using on P Threads

USING  $P$  THREADS TO COMPUTE 1  $P \times P$  TILE

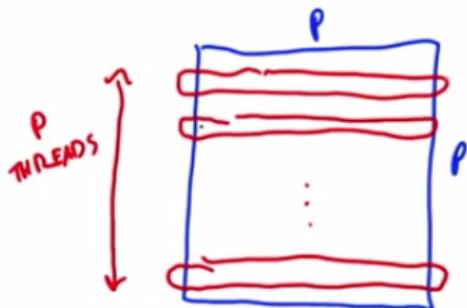


```
device__ float3
tile_calculation(Params myParams, float3 force) {
    int i;
    extern __shared__ Params[] sourceParams;
    for (i = 0; i < blockDim.x; i++) {
        force += bodyBodyInteraction(myParams, sourceParams[i]);
    }
    return force;
}
```

### L6.1-6.8-Why is This a Good Idea

#### USING $P$ THREADS TO COMPUTE 1 PKP TILE

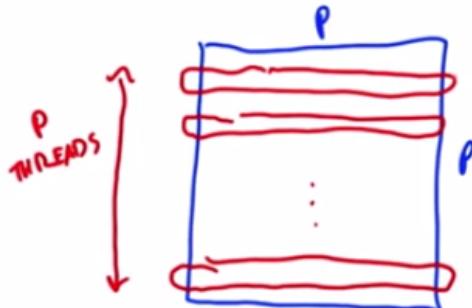
WHY IS THIS A GOOD IDEA?



- STILL MUST SHARE SOURCE OBJECT PARAMS ACROSS ALL THREADS
- NO SHARING NECESSARY FOR DESTINATION OBJECTS!
- NO COMMUNICATION NECESSARY TO SUM UP RESULTS!

#### USING $P$ THREADS TO COMPUTE 1 PKP TILE

HAVE WF:



- INCREASED
- DECREASED
- KEPT CONSTANT

THE AMOUNT OF PARALLELISM?

#### USING $P$ THREADS TO COMPUTE 1 PKP TILE

BIG PICTURE:

WE CHOSE

- FEWER THREADS
- MORE WORK/THREAD

THIS CONVERTED COMMUNICATION BETWEEN THREADS TO COMMUNICATION WITHIN A THREAD.

## L6.1-6.9-How Big Should We Make a Tile

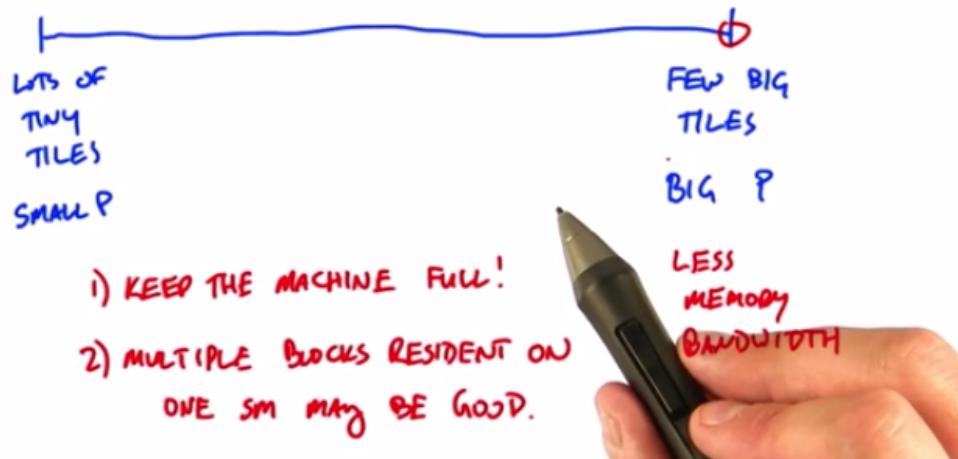
## HOW BIG SHOULD WE MAKE A TILE?



## L6.1-6.10-What If P is Too Big or Small

## L6.1-6.11-Determining the Size of the Tiles

## CONSIDERATIONS ON TILE SIZING



L6.1-6.12-All Pairs N-Body Summary

## All-Pairs N-Body : Summary

- 1) BRUTE FORCE, SIMPLE, HIGH PERFORMANCE.
  - 2) TRADEOFF BETWEEN:
    - MAXIMIZING PARALLELISM
    - MORE WORK/THREAD → MORE EFFICIENT COMMUNICATION

### L6.1-6.13-SpMV

SPARSE MATRIX-VECTOR MULTIPLY

DENSE

$$\begin{bmatrix} x & x & x & x \\ x & x & x & x \\ x & x & x & x \\ x & x & x & x \end{bmatrix} \cdot \begin{bmatrix} y \\ y \\ y \\ y \end{bmatrix}$$

SPARSE

$$\begin{bmatrix} x & & x \\ & x & x \\ & & x \end{bmatrix} \cdot \begin{bmatrix} y \\ y \\ y \end{bmatrix}$$

① LESS STORAGE

② LESS COMPUTATION

BLANKS ARE ZEROS

### L6.1-6.14-Role of Thread in SpMv

WHAT IS THE ROLE OF A THREAD?

THREAD  
PER  
ROW

$$\begin{bmatrix} x & & x \\ & x & x \\ & & x \end{bmatrix} \cdot \begin{bmatrix} y \\ y \\ y \end{bmatrix} = \begin{bmatrix} z \end{bmatrix}$$

WHICH APPROACH WILL ...

TPR TPE

LAUNCH MORE THREADS?

THREAD  
PER  
ELEMENT

$$\begin{bmatrix} x & & x \\ & x & x \\ & & x \end{bmatrix} \cdot \begin{bmatrix} y \\ y \\ y \end{bmatrix}$$

REQUIRE COMMUNICATION  
BETWEEN THREADS?

DO MORE WORK PER  
THREAD?

### L6.1-6.15-Thread Per Row

THREAD PER ROW - DATA  
STRUCTURE

CSR

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix}$$

VALUE [A B C D E F]

INDEX [0 2 1 0 2 3]

ROWPTR [0 2 3 5]

UDACITY

```
__global__ void
spmv_csr_scalar_kernel(
    const int num_rows, const int * rowptr,
    const int * index, const float * value,
    const float * x, float * y) {
    int row = blockDim.x * blockIdx.x +
        threadIdx.x;
    if (row < num_rows) {
        float dot = 0;
        int row_start = rowptr[row];
        int row_end = rowptr[row+1];
        for (int jj = row_start ;
            jj < row_end ; jj++)
            dot += value[jj] * x[index[jj]];
        y[row] += dot;
    }
}
```

### L6.1-6.16-Performance Analysis:TPR

THREAD PER ROW · DATA STRUCTURE  
CSR

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix}$$

VALUE [A B C D E F]

INDEX [0 2 1 0 2 3]

ROWPTR [0 2 3 5]

QUIZ:

- 32 ADJACENT THREADS IN WARP
- EACH THREAD ASSIGNED TO ROW
- ROWS ARE DIFFERENT LENGTHS

WILL RUNTIME BE ~

- SHORTEST
  - AVERAGE
  - LONGEST
- row?

### L6.1-6.17-Thread Per Element

THREAD PER ELEMENT

CSR

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix}$$

VALUE [A B C D E F]

INDEX [0 2 1 0 2 3]

ROWPTR [0 2 3 5]

CSR REPRESENTATION OF MATRIX.

- (1) USE ROWPTR TO GENERATE SEGMENTED VALUE ARRAY
- (2) THREAD PER ELEMENT: MAP  
VALUE[n] · X[INDEX[n]]
- (3) BACKWARDS INCLUSIVE SEGMENTED SUM-SCAN
- (4) USE ROWPTR TO GATHER SPARSE OUTPUT INTO DENSE VECVR.

### L6.1-6.18-Performance Analysis:TPE

THREAD PER ELEMENT

CSR

$$\begin{bmatrix} A & B \\ C & D \\ E & F \end{bmatrix}$$

VALUE [A B C D E F]

INDEX [0 2 1 0 2 3]

ROWPTR [0 2 3 5]

QUIZ:

32 ADJACENT THREADS / WARP

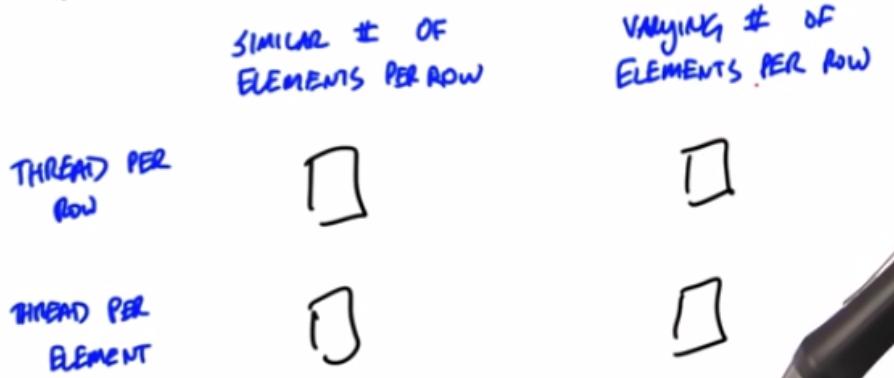
THREAD / ELEMENT

IS RUNTIME:

- PROPORTIONAL TO LONGEST ROW?
- " " " SHORTEST ROW?
- INSENSITIVE?

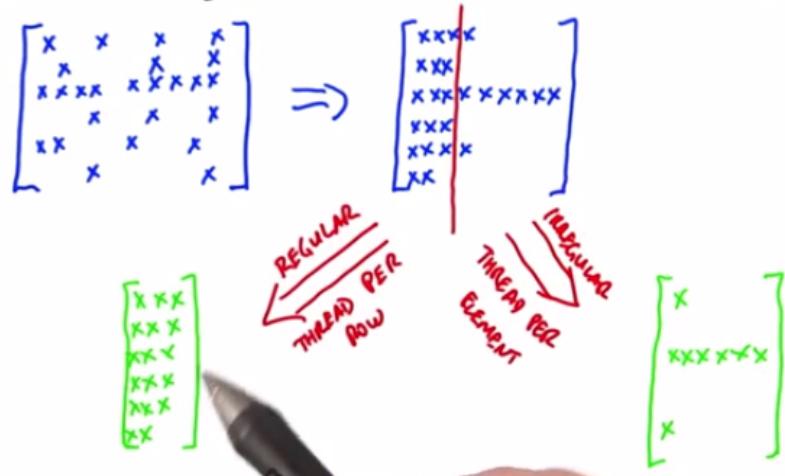
### L6.1-6.19-Thread Per Row vs. Thread Per Element

#### COMPARISON



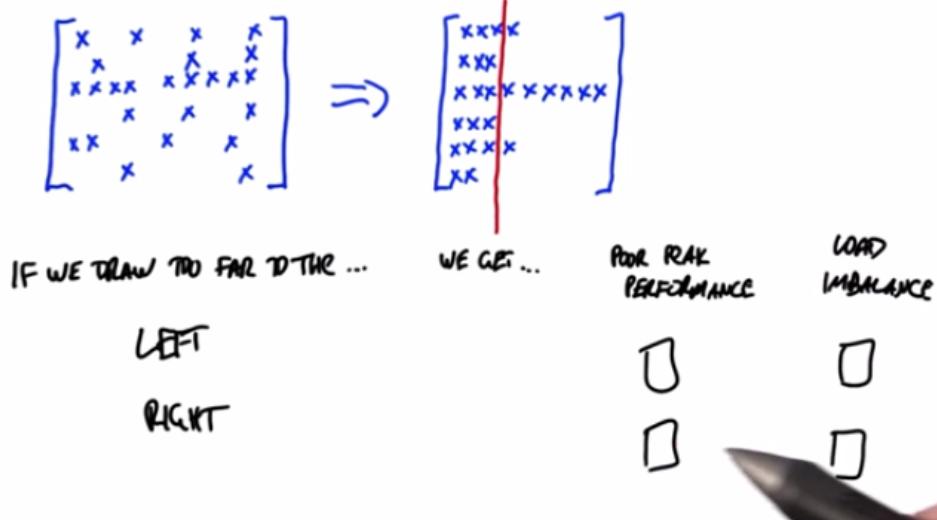
### L6.1-6.20-Hybrid Approach

#### HYBRID APPROACH [BELL & GARLAND, SUPERCOMPUTING 2009]



### L6.1-6.21-A Quiz on the Hybrid Approach

#### HYBRID APPROACH [BELL & GARLAND, SUPERCOMPUTING 2009]



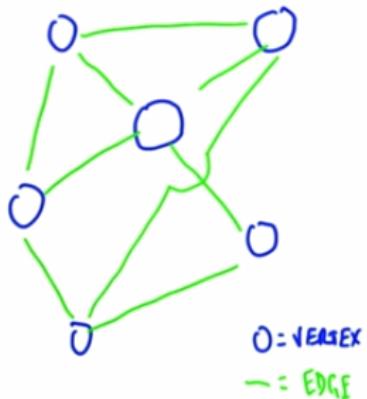
## L6.1-6.22-SpMV-The Big Picture

### BIG PICTURE

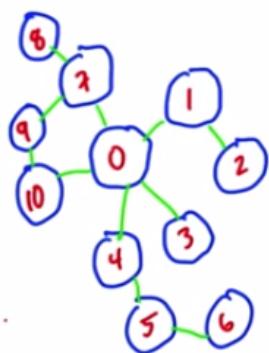
- (1) KEEP YOUR THREADS BUSY
- (2) CLOSER COMMUNICATION IS CHEAPER

## L6.1-6.23-Traversal of Graphs

### BREADTH-FIRST TRAVERSAL OF GRAPHS

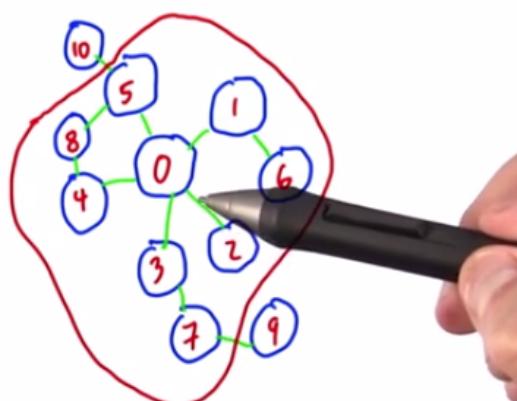


### DEPTH FIRST TRAVERSAL



- LESS STATE

### BREADTH-FIRST TRAVERSAL

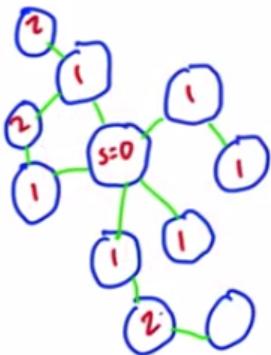


- MORE PARALLELISM

[Correction] On the right, 9 and 10 should be switched, and 9 should be included inside the red boundary.

### L6.1-6.24-Calculating Distance From the Root

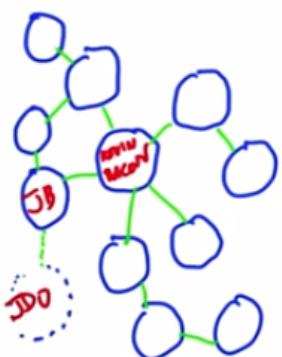
BREADTH-FIRST TRAVERSAL : CALCULATING DISTANCE FROM ROOT



[Correction] far right 1 should be 2.

### L6.1-6.25-Minimum and Maximum Possible Depth From Root

BREADTH-FIRST TRAVERSAL : CALCULATING DISTANCE FROM ROOT



GRAPH WITH N NODES:

WHAT'S THE:

- MINIMUM POSSIBLE MAX. DEPTH?



- MAXIMUM POSSIBLE MAX. DEPTH?



### L6.1-6.26-BFS Try #1

BFS TRY #1

WHAT DO WE WANT?

- LOTS OF PARALLELISM
- COALESCED MEMORY ACCESS
- MINIMAL EXECUTION DIVERGENCE
- EASY TO IMPLEMENT

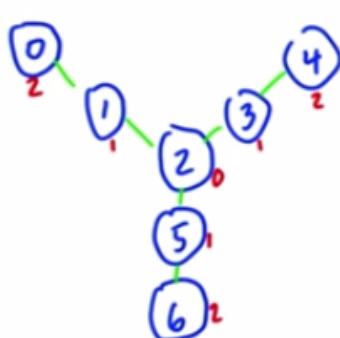
### BFS: HIGH LEVEL ALGORITHM

- PARALLELIZE OVER LIST OF EDGES
- ITERATE  $n$  TIMES,  $n = \text{MAX DEPTH OF GRAPH}$
- PER EDGE:
  - IF ONE VERTEX HAS DEPTH  $d$  AND OTHER VERTEX HAS NOT BEEN VISITED:
  - OTHER VERTEX GETS  $d+1$



### L6.1-6.27-BFS Try #1-An Example

EXAMPLE



VERTICES

0	1	2	3	4	5	6
2	1	0	1	2	2	1

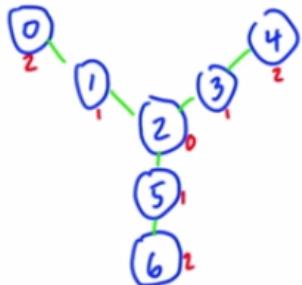
EDGES

0	1	2	3	5	6
1	2	3	4	6	2

[Correction]: Edges 012356/123425, vertices depth 2101212

### L6.1-6.28-What is the Work Complexity

EXAMPLE



QUIZ

$$\text{WORK COMPLEXITY} = f(V, E)$$

(V)	<input type="checkbox"/>	(E)	<input type="checkbox"/>
1	<input type="checkbox"/>	1	<input type="checkbox"/>
$\log V$	<input type="checkbox"/>	$\log E$	<input type="checkbox"/>
V	<input type="checkbox"/>	E	<input type="checkbox"/>
$V \log V$	<input type="checkbox"/>	$E \log E$	<input type="checkbox"/>
$V^2$	<input type="checkbox"/>	$E^2$	<input type="checkbox"/>

### L6.1-6.29-BFS Code Part 1

```
// Initialization kernel: launch v threads
__global__ void
initialize_vertices( Vertex * vertices,
                     int starting_vertex,
                     int num_vertices )
{
    int v = blockDim.x * blockIdx.x + threadIdx.x;
    if (v == starting_vertex) vertices[v] = 0 else vertices[v] = -1;
}
```

### L6.1-6.30-BFS Code Part 2

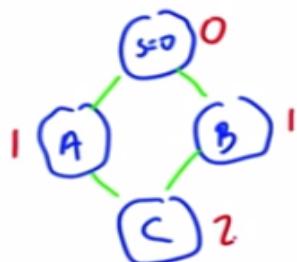
```
// BFS kernel: launch e threads
__global__ void
bfs( const Edge * edges,
      Vertex * vertices,
      int current_depth )
{
    int e = blockDim.x * blockIdx.x + threadIdx.x;
    int vfirst = edges[e].first;
    int dfirst = vertices[vfirst];
    int vsecond = edges[e].second;
    int dsecond = vertices[vsecond];
    if ((dfirst == current_depth) && (dsecond == -1)) { // 1st visited, 2nd hasn't
        vertices[vsecond] = dfirst + 1;
    }
    if (                                         ) { // 2nd visited, 1st hasn't
    }
    // other two cases: {visited, visited}, {not, not} can be ignored
}
```

### L6.1-6.31-BFS Code Part 3

```
// BFS kernel: launch e threads
__global__ void
bfs( const Edge * edges,
      Vertex * vertices,
      int current_depth )
{
    int e = blockDim.x * blockIdx.x + threadIdx.x;
    int vfirst = edges[e].first;
    int dfirst = vertices[vfirst];
    int vsecond = edges[e].second;
    int dsecond = vertices[vsecond];
    if ((dfirst == current_depth) && (dsecond == -1)) { // 1st visited, 2nd hasn't
        vertices[vsecond] = dfirst + 1;
    }
    if ((dsecond == current_depth) && (dfirst == -1)) { // 2nd visited, 1st hasn't
        vertices[vfirst] = dsecond + 1;
    }
    // other two cases: {visited, visited}, {not, not} can be ignored
}
```

### L6.1-6.32-Do We Have to Worry About Race Condition

#### IMPLEMENTATION NOTES



```

// snippet 1
while (!h_done) {
    bfs(edges, vertices)
    cudaMemcpy(&h_done, &d_done, sizeof(bool), cudaMemcpyDeviceToHost);
}

// snippet 2
while (!h_done) {
    cudaMemcpy(&d_done, &h_true, sizeof(bool), cudaMemcpyHostToDevice);
    bfs(edges, vertices)
    cudaMemcpy(&h_done, &d_done, sizeof(bool), cudaMemcpyDeviceToHost);
}

// snippet 3
if ((vfirst != -1) && (vsecond == -1)) { // first visited, second hasn't
    vertices[vsecond] = vfirst + 1;
    done = false;
}
if ((vfirst == -1) && (vsecond != -1)) { // second visited, first hasn't
    vertices[vfirst] = vsecond + 1;
    done = false;
}

```



#### L6.1-6.33-BFS TRY #1 Summary

Summary: BFS Try #1

- PARALLEL? ☺
  - MEMORY BEHAVIOR? ☺
  - THREAD DIVERGENCE? ☹
  - COMPLEXITY? :: QUADRATIC!  $O(v^2)$
- 

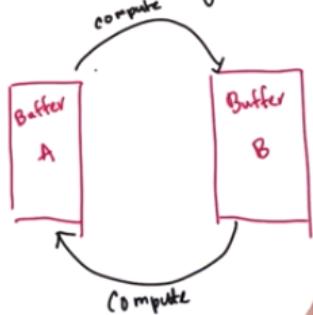
#### L6.1-6.34-Problem Set 6

Problem Set #6 - Seamless Image Cloning



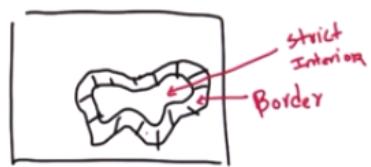
## Problem Set #6

Double Buffering



## Problem Set #6

Inputs



## Problem Set #6



$$I_0 = 0$$

$$I_K$$

Compute  $I_{K+1}$  as follows:

for  $i_{KL}$  in  $I_{K+1}$ ,

$$i_{K+1} = \frac{A + B + C}{D}$$

A = sum of  $i_K$ 's  
neighbors

B = sum of  $i_L$ 's  
neighbors

C = difference between  
 $S$  and  $S'$ 's neighbors

D = number of  $i_K$ 's  
neighbors