

CS344 Introduction to Parallel Programming

Lesson 4: Fundamental GPU Algorithms (Applications of Sort and Scan)

L4-4.2-Scan Recap

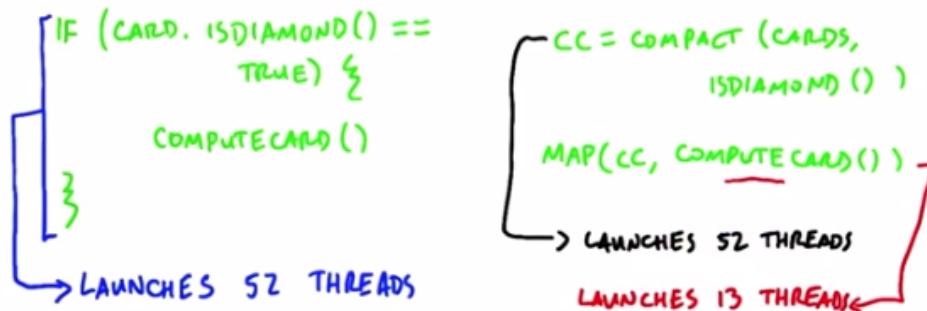
SCAN RECAP (AS A QUIZ)

ON A SCAN OF N ELEMENTS :

AMOUNT OF WORK	# OF STEPS
$O(\log n)$	□
$O(n)$	□
$O(n \log n)$	□
$O(n^2)$	□



L4-4.3-What is Compact



COMPACT (MORE FORMALLY)

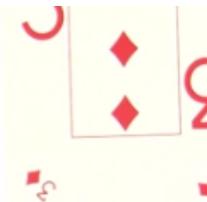
INPUT	$s_0 s_1 s_2 s_3 s_4 \dots$
PREDICATE	T F T F T ... "is my index even?"
OUTPUT	$s_0 - s_2 - s_4 \leftarrow \text{SPARSE}$ OR $s_0 s_2 s_4 \leftarrow \text{DENSE}$

- SELECT 13 DIAMONDS FROM 52 CARDS
- RUN COMPUTECARD() ON DIAMONDS

```

    ↓ SPARSE      ↓ DENSE
  IF (CARD.ISDIAMOND() == TRUE) {
    COMPUTECARD()
}
  CC = COMPACT(CARDS, ISDIAMOND())
  MAP(CC, COMPUTECARD())
}

```



L4-4.4-When To Use Compact

QUIZ: WHEN TO USE COMPACT?

COMPACT IS MOST USEFUL WHEN WE COMPACT

AWAY A SMALL NUMBER OF ELEMENTS
 LARGE

AND THE COMPUTATION ON EACH SURVIVING ELEMENT

IS CHEAP ?
 EXPENSIVE

L4-4.5-Core Algorithm to Compact

CORE ALGORITHM FOR COMPACT

PRED	T	F	F	T	T	F	T	F
ADDRESSES	0	-	-	1	2	-	3	-

PRED	1	0	0	1	1	0	1	0
ADDRESSES	0	1	1	1	2	3	3	4

WHAT IS THIS OPERATION?

L4-4.6-Steps to Compact

STEPS TO COMPACT

- 1) PREDICATE
- 2) SCAN-IN ARRAY: TRUE 1
FALSE 0
- 3) EXCLUSIVE-SUM-SCAN (SCAN-IN)

OUTPUT IS SCATTER ADDRESSES FOR COMPACTED ARRAY

- 4) SCATTER INPUT INTO OUTPUT USING ADDRESSES

QUIZ COMPACT IN ELEMENTS ($1 \rightarrow 1^M$)

A: IS DIVISIBLE BY 17 [KEEPS FEW ITEMS]

B: IS NOT DIVISIBLE BY 31 [KEEPS MANY ITEMS]

	A RUNS FASTER	SAME	B RUNS FASTER
PREDICATE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SCAN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/> ↘
SCATTER	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>

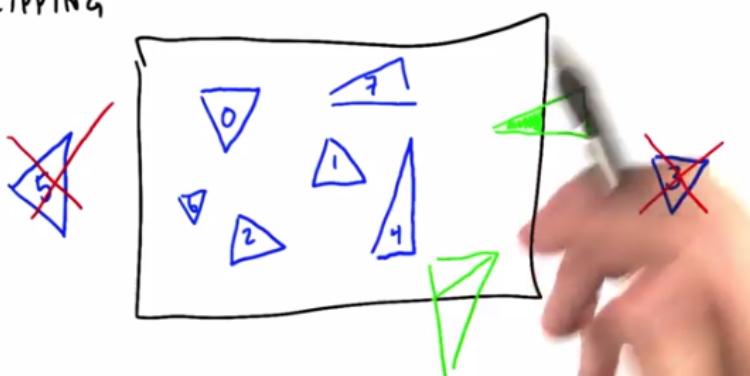
L4-4.7-Allocate

ALLOCATE

COMPACT GENERATES ↗ 1 OUTPUT FOR TRUE INPUTS
∅ FOR FALSE

CAN WE GENERALIZE?

CLIPPING



QUIZ WHAT IS THE MAXIMUM # OF TRIANGLES
THAT CAN BE PRODUCED BY A
TRIANGLE CLIPPED BY A RECTANGLE?



L4-4.8-Possible Allocate Strategy

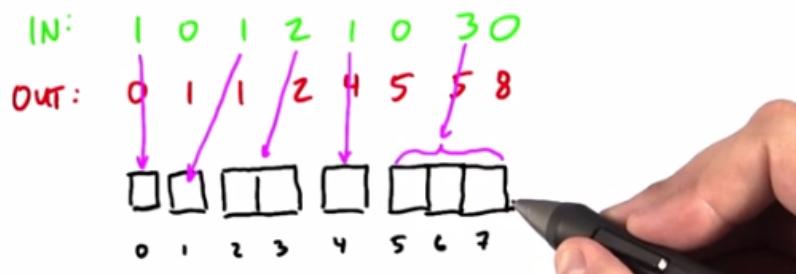
POSSIBLE ALLOCATE STRATEGY

- ALLOCATE MAXIMUM SPACE IN INTERMEDIATE ARRAY
- COMPACT RESULT
- WASTEFUL IN SPACE
- REQUIRES SCANNING LARGE INTERMEDIATE ARRAY



A GOOD STRATEGY FOR ALLOCATE

- INPUT: ALLOCATION REQUEST PER INPUT ELEMENT
- OUTPUT: LOCATION IN ARRAY TO WRITE YOUR THREAD'S OUTPUT



A GOOD STRATEGY FOR ALLOCATE

- INPUT: ALLOCATION REQUEST PER INPUT ELEMENT
- OUTPUT: LOCATION IN ARRAY TO WRITE YOUR THREAD'S OUTPUT

IN: 1 0 1 2 1 0 3 0] SCAN!

OUT: 0 1 1 2 4 5 5 8



L4-4.9-Segmented Scan

SEGMENTED SCAN

- MANY SMALL SCANS?
- LAUNCH EACH INDEPENDENTLY
- COMBINE AS SEGMENTS \Rightarrow SEGMENTED SCAN

EXCLUSIVE SUM SCAN:

$$(1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8) \rightarrow (0 \ 1 \ 3 \ 6 \ 10 \ 15 \ 21 \ 28)$$

$$(1 \underline{2} | \underline{3} \underline{4} \underline{5} | \underline{6} \underline{7} \underline{8}) \rightarrow (\underline{0} \underline{1} | \underline{0} \underline{3} \underline{7} | \underline{0} \underline{6} \underline{13})$$

(1 0 1 0 0 1 0 0) : SEGMENT HEADS

QUIZ

INCLUSIVE SEGMENTED SUM SCAN ON SAME ARRAY:

$$(1 \ 2 | 3 \ 4 \ 5 | 6 \ 7 \ 8)$$



L4-4.10-SpMV

SPARSE MATRIX / DENSE VECTOR MULTIPLICATION [SpMV]

- DENSE MATRICES: STORE ALL ELEMENTS
- SPARSE : DON'T STORE ZEROES



MATRIX \times VECTOR

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + by + cz \\ dx + ey + fz \\ gx + hy + iz \end{bmatrix}$$

3×3 3×1 3×1

QUIZ: MATRIX \times VECTOR

$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 1 & 3 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} \square \\ \square \\ \square \end{bmatrix}$$

L4-4.11-Sparse Matrices

SPARSE MATRICES

COMPRESSED SPARSE ROW:

$$\rightarrow \begin{bmatrix} 0 & 1 & 2 \\ a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

VALUE [a b c d e f]

COLUMN [0 2 0 1 2 1]

ROWPTR [0 2 5]

L4-4.12-Generate CSR

QUIZ: GENERATE CSR FOR

$$\begin{bmatrix} 1 & 0 & 2 \\ 2 & 1 & 0 \\ 0 & 1 & 3 \end{bmatrix}$$

VALUE
INDEX
ROWPTR

L4-4.13-Actually Doing the Matrix

VALUE $[a \ b \ c \ d \ e \ f]$
 COLUMN $[0 \ 2 \ 0 \ 1 \ 2 \ 1]$
 ROWPTR $[0 \ 2 \ 5]$

VECTOR
 $\begin{bmatrix} x \\ y \\ z \end{bmatrix}$

- 1 CREATE SEGMENTED REP'N FROM VALUE + ROWPTR
- 2 GATHER VECTOR VALUES USING COLUMN
- 3 PAIRWISE MULTIPLY 1-2 (BACKWARDS)
- 4 EXCLUSIVE SEGMENTED SUM SCAN

Diagram illustrating the computation of a matrix-vector product:

$$\begin{bmatrix} a & b \\ c & d \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + bx + bz \\ cx + dy + cz \\ 0x + 0y + fz \end{bmatrix}$$

Intermediate results after pairwise multiplication:

$$\begin{bmatrix} a & b \\ x & z \\ ax + bx + bz & \end{bmatrix} \begin{bmatrix} c & d \\ x & y \\ cx + dy + cz & \end{bmatrix} \begin{bmatrix} f \\ fy \\ fx + fy + fz \end{bmatrix}$$

Final result after an exclusive segmented sum scan:

$$\begin{bmatrix} a & b \\ x & z \\ ax + bx + bz & \end{bmatrix} \begin{bmatrix} c & d \\ x & y \\ cx + dy + cz & \end{bmatrix} \begin{bmatrix} f \\ fy \\ fx + fy + fz \end{bmatrix}$$

- 1 CREATE SEGMENTED REP'N FROM VALUE + ROWPTR
- 2 GATHER VECTOR VALUES USING COLUMN
- 3 PAIRWISE MULTIPLY 1-2 (BACKWARDS)
- 4 EXCLUSIVE SEGMENTED SUM SCAN

Diagram illustrating the computation of a matrix-vector product:

$$\begin{bmatrix} a & b \\ c & d \\ 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + bx + bz \\ cx + dy + cz \\ 0x + 0y + fz \end{bmatrix}$$

Intermediate results after pairwise multiplication:

$$\begin{bmatrix} a & b \\ x & z \\ ax + bx + bz & \end{bmatrix} \begin{bmatrix} c & d \\ x & y \\ cx + dy + cz & \end{bmatrix} \begin{bmatrix} f \\ fy \\ fx + fy + fz \end{bmatrix}$$

Final result after an exclusive segmented sum scan:

$$\begin{bmatrix} a & b \\ x & z \\ ax + bx + bz & \end{bmatrix} \begin{bmatrix} c & d \\ x & y \\ cx + dy + cz & \end{bmatrix} \begin{bmatrix} f \\ fy \\ fx + fy + fz \end{bmatrix}$$

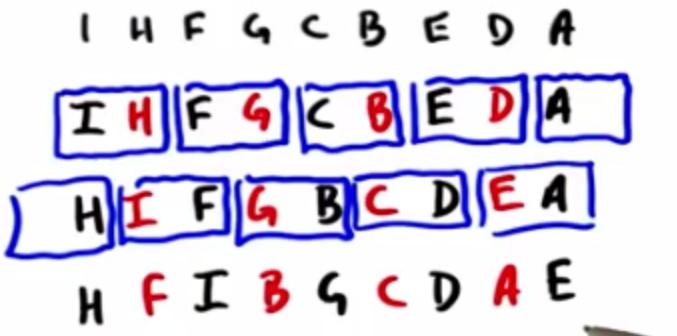
$$\begin{bmatrix} a & 0 & b \\ c & d & e \\ 0 & 0 & f \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} ax + bx + bz \\ cx + dy + cz \\ fx + fy + fz \end{bmatrix}$$

L4-4.14-Sort

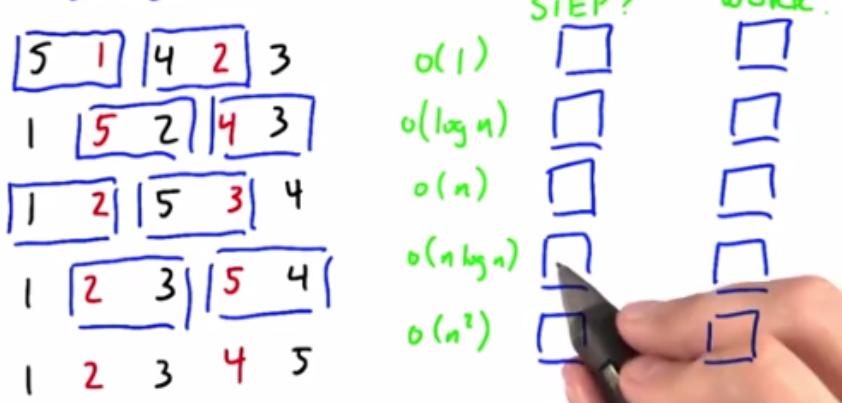
SORT

- LOTS OF SERIAL ALGORITHMS! FEWER PARALLEL.
- FIND EFFICIENT PARALLEL ALGORITHMS
 - KEEP HARDWARE Busy (LOTS OF THREADS)
 - LIMIT BRANCH DIVERGENCE
 - PREFER COALEXED MEMORY ACCESS

ODD-EVEN SORT (BRICK SORT)



BRICK SORT SAMPLE

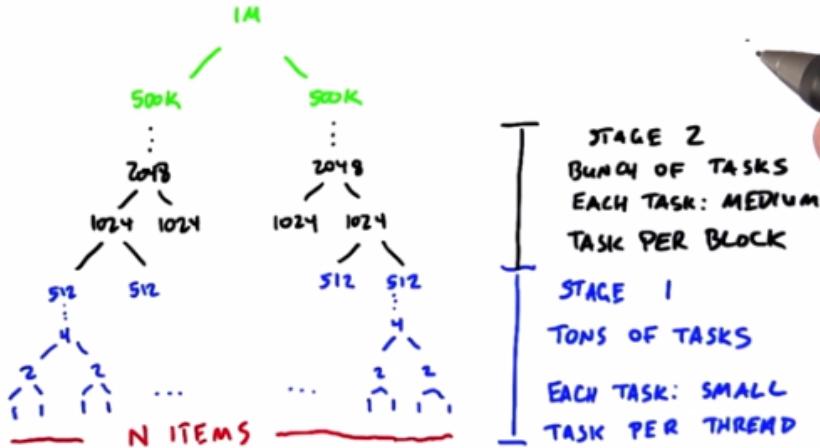


L4-4.15-Merge Sort

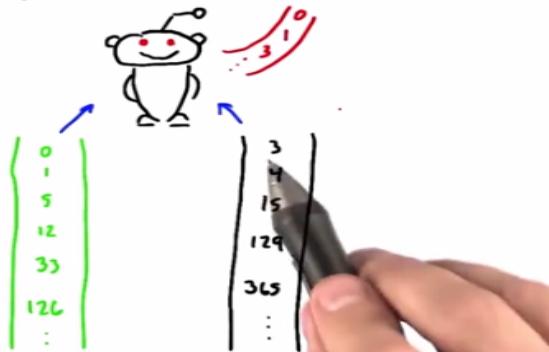
MERGE SORT



MERGE SORT



INTERMEDIATE MERGE: SERIAL ALGORITHM

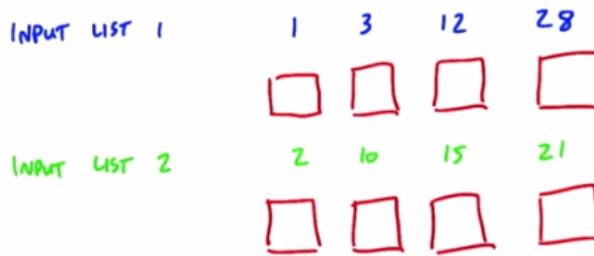


L4-4.16-Parallel Merge

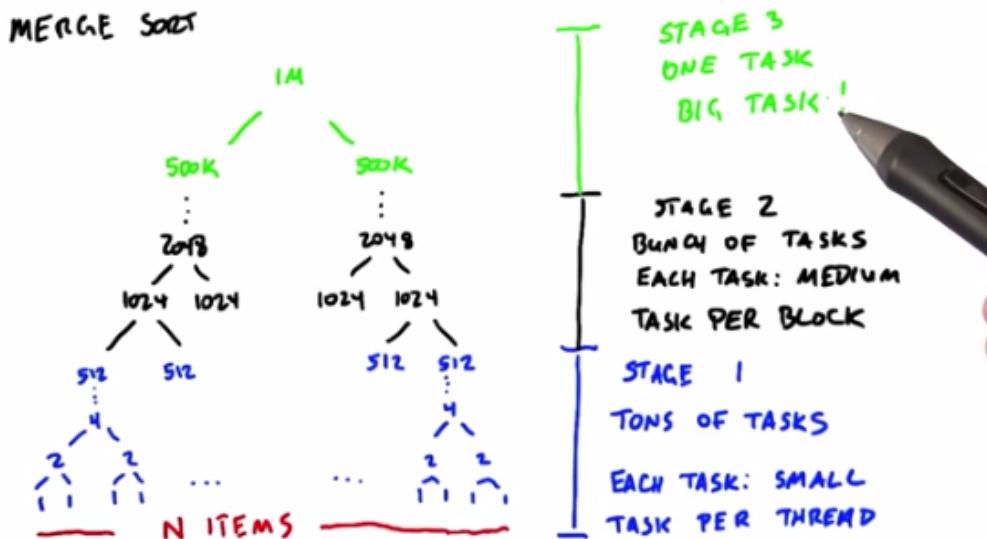
REVIEW: COMPACT

IN	1	1	2	3	5	8	13	(21)
PRED	T	T	F	T	T	F	T	T
SHUFFLE ADDRESS	0	1	2	3			4	5
OUT	1	1	3	5	13	21		

PARALLEL MERGE



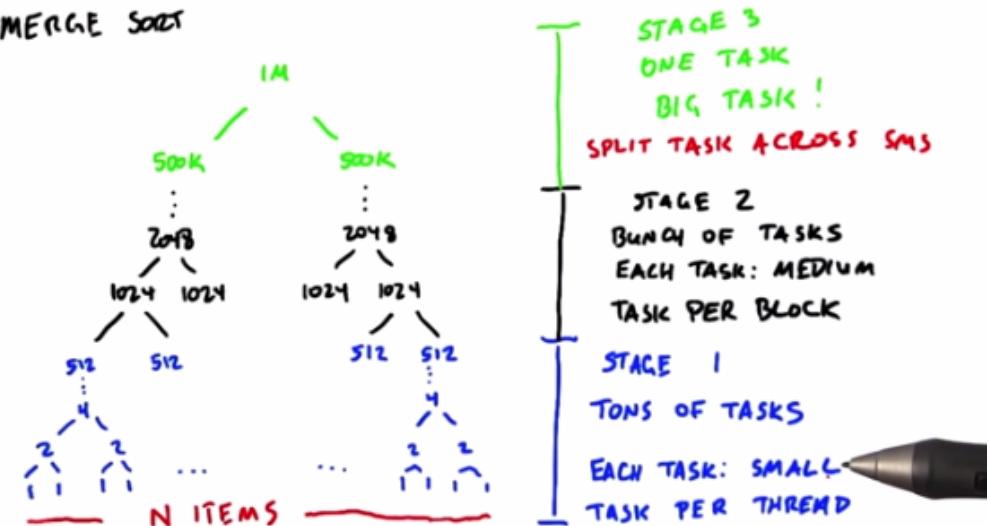
L4-4.17-Bad To Have One big Merge



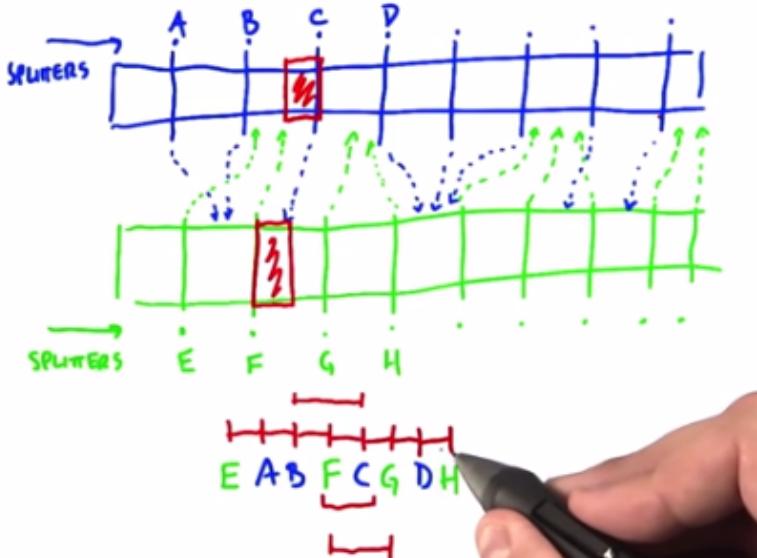
QUIZ: WHY IS IT BAD TO HAVE ONLY ONE MERGE
TASK REMAINING?

- LOTS OF THREADS IDLE PER SM
- LOTS OF SMs IDLE
- VERY HIGH BRANCH DIVERSION

L4-4.18-Split Tasks Across SMs



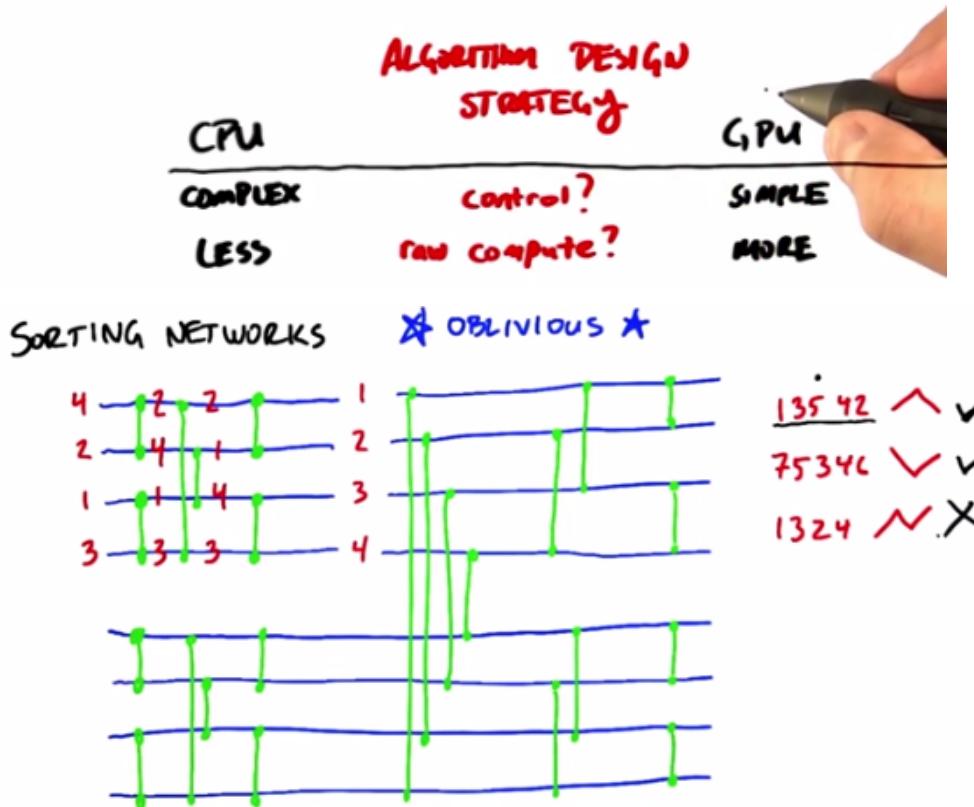
BREAKING UP A SINGLE BIG MERGE

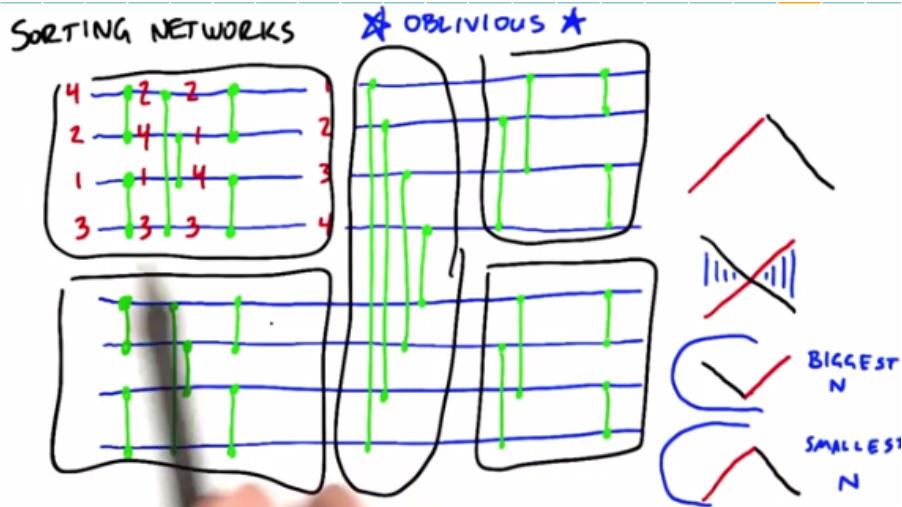


L4-4.19-Sorting Networks Part 1

OBLIVIOUSNESS + GPUs

"OBLIVIOUS": BEHAVIOR IS INDEPENDENT OF
SOME ASPECT OF THE PROGRAM





QUIZ RUNTIME FOR BITONIC SORTER TO SORT ..

- A) COMPLETELY SORTED
- B) ALMOST SORTED
- C) REVERSED
- D) RANDOM

- A < B < C < D
 C < D < B < A
 A < B < C < D
 A = B = C = D



L4-4.20-Sorting Networks Part 2

L4-4.21-Radix Sort Part 1

RADIX SORT

- 1 START WITH LSB
- 2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
- 3 MOVE TO NEXT MSB, REPEAT

0	000	000	000	000
5	101	010	100	001
2	010	110	101	010
7	111	100	001	011
1	001	101	010	100
3	011	111	110	101
6	110	001	111	110
4	100	011	... 011	111

$O(kn)$
 k: bits in representation
 n: items to sort

RADIX SORT

- 1 START WITH LSB
- 2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
- 3 MOVE TO NEXT MSB, REPEAT

0	000	000
5	101	010
2	010	110
7	111	100
1	001	101
3	011	111
6	110	001
4	100	011

WHAT IS THIS ALGORITHM?

L4-4.22-Radix Sort Part 2 (

RADIX SORT

- 1 START WITH LSB
- 2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
- 3 MOVE TO NEXT MSB, REPEAT

0	000	000
5	101	010
2	010	110
7	111	100
1	001	101
3	011	111
6	110	001
4	100	011

WHAT IS THIS ALGORITHM?

COMPACT

WHAT IS THE PREDICATE ON UINT i?

L4-4.23-Radix Sort Part 3

RADIX SORT

- 1 START WITH LSB
- 2 SPLIT INPUT INTO 2 SETS BASED ON BIT. OTHERWISE PRESERVE ORDER.
- 3 MOVE TO NEXT MSB, REPEAT

0	000	000	1	0
5	101	010	0	1
2	010	110	1	1
7	111	100	0	2
1	001	101	0	2
3	011	111	0	2
6	110	001	1	2
4	100	011	1	3

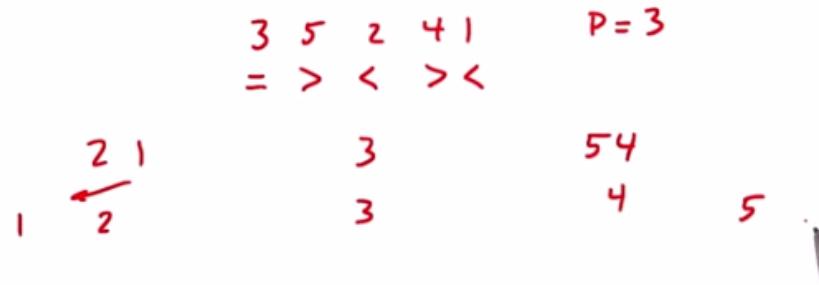
SCAN

0
1
2
3

L4-4.24-Quick Sort

QUICK SORT

- 1 CHOOSE PIVOT ELEMENT
- 2 COMPARE ALL ELEMENTS VS PNOT
- 3 SPLIT INTO 3 ARRAYS: $<P$ $=P$ $>P$
- 4 RECURSE ON EACH ARRAY

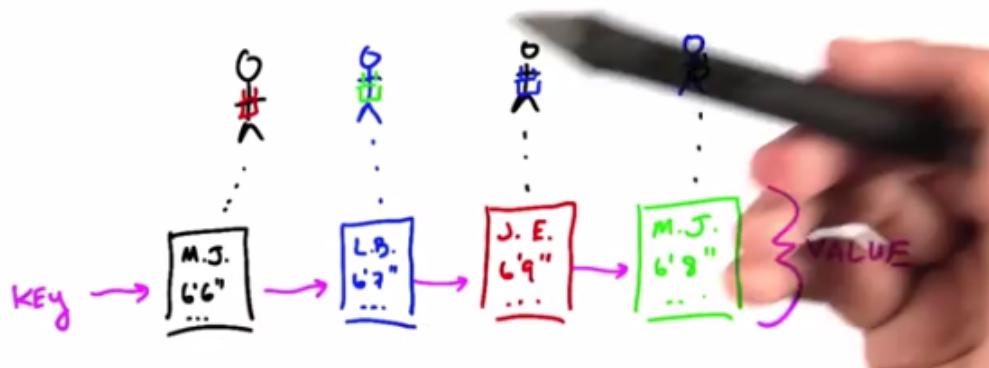


QUICK SORT + SEGMENTS

DISTRIBUTE (SEG.) $[3 \ 5 \ 2 \ 4 \ 1]$
MAP 3 3 3 3 3
COMPACT < 2 1
COMPACT = 3
COMPACT > 5 4
[2 1 | 3 | 5 4]

L4-4.25- Key Value Sort

KEY VALUE SORTS



L4-4.26-Summary

Summary : WHAT WE LEARNED

- COMPACT
- ALLOCATE
- SEGMENTED SCAN
- ODD EVEN SORT
- MERGE SORT
- SORTING NETWORKS
- QUICK SORT
- RADIX SORT

L4-4.28-Problem Set #4

Problem Set #4 - Red Eye Removal



(1) Stencil

(2) Sort

(3) Map

Problem Set #4

