ESE5320: System-on-a-Chip Architecture

Day 23: November 20, 2023
(was Day 21: November 14, 2022)
Reduce

Today

- Part 1
  - Reduce
  - Associative Operations
  - Model
- Part 2
  - Latency Bound Implications and Implementations
- Part 3
  - Parallel Prefix
  - Broad Application
- Part 4: Binary Arithmetic

Message

- Aggregation is a common need that is not strictly data parallel
- …but admits to parallel computation with a slightly different pattern that is worth knowing

Reduce

- Reduce – combining a collection of data into a single value
  - Converting a vector into a scalar
    - E.g. sum elements

Sum Reduce

- Simplest and most common
  - Add up all the values in a vector or array

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```


Sum Reduce

- What's II? (unit delay add)

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```

Sum Reduce

- What's latency bound?
  - Assuming associativity holds for addition

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```


Associative Operations

- Associativity means can group together operations in any way
- Addition is associative for
  - Natural numbers
  - Real Numbers
  - Modulo Arithmetic

Associative Operations

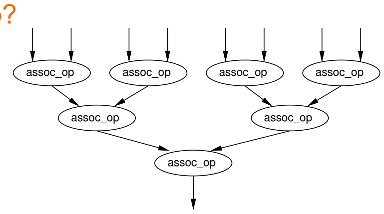
- Associativity means can group together operations in any way
- Normal sequential:
  (((a[0]+a[1])+a[2])+a[3])+…
- Associative regroup:
  (a[0]+((((a[1]+(a[2]+a[3]))+a[4])+(…)))

Associative Operations

- Associativity means can group together operations in any way
- Normal sequential:
  (((a[0]+a[1])+a[2])+a[3])+…
- Regroup parallelism:
  (((a[0]+a[1])+(a[2]+a[3]))+((a[4]+a[5])+(a[6]+a[7])))

Associative Tree Reduce

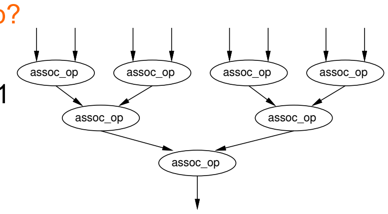
- Add pairs – cut numbers in half
- Repeat adding pairs until single value
- How deep?



Associative Tree Reduce

- Add pairs – cut numbers in half
- Repeat adding pairs until single value
- How deep?

- $N*(1/2)^k=1$
- $N=2^k$
- $k=\log_2(N)$



Latency Bounds

- Associative reduces typically contribute log terms to latency bounds
  - …as you've seen on many previous midterms and finals

Sum Reduce

- Data Parallel?

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```


Sum Reduce

- How exploit 4 cores to compute?
  - (assume a very large, like 1 million)

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```


Model: Data Parallel+Reduce

- Data Parallel + Reduce
  - Very common to perform a data parallel operation then a reduce on results
- Example: dot product
  (core in DNN, Matrix-Multiply)

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i]*b[i];
```


Dot Product

- Latency bound for dot product
  - Assume 1 cycle add, 3 cycle multiply
- Example: dot product

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i]*b[i];
```


Model: Data Parallel+Reduce

- Data Parallel + Reduce
  - Very common to perform a data parallel operation then a reduce on results
- General form

```
int res=0;
for (int i=0;i<N; i++)
  res=assoc_op(res,f(a[i],b[i], …))
```

What else Associative?

- Beyond modulo addition, what other associative operations do we often see as reductions?

Associative Operations

- Add
- Multiply
- Max
- Min
- AND
- OR

- Max/min
  – And keep associated position
- Find First

Optimization Loop

```
int minval=f(0);
int min=0;
for (i=1;i<N;i++) {
   int val=f(i);
   if (val<minval) {
       minval=val; min=i;
      }
}
```


Min keeping Position

```
if (val<minval) {
       minval=val; min=i;
      }
```

Each operation:

min1,val1,min2,val2 → min,val

if(val1<=val2) // keep first position found

// if equal, should be first

{min=min1; val=val1;}

else

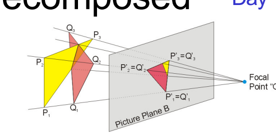
{min=min2; val=val2;}

Optimization Loop

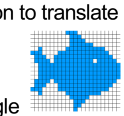
```
int minval=f(0);
int min=0;
for (i=1;i<N;i++) {
   int val=f(i);
   if (val<minval) {
       minval=val; min=i;
      }
}
```


Rendering Decomposed

Day 15



- Pipeline of
  – Projection
    - Where do the points of this triangle end up in the viewed image?
    - Matrix-multiplication to translate points
  – Rasterization
    - Turn into pixels
    - Fill pixels for triangle
  – Z-buffer
    - Keep only the ones on top (not hidden)
      – 2D image + Z-depth – keep smallest



Figures from: https://commons.wikimedia.org/wiki/File:Perspective_Projection_Principle.jpg https://en.wikipedia.org/wiki/Rasterisation#/media/File:Raster_graphic_fish_20x23squares_sdtv-example.png

Z-Buffering

- Storing into Z–buffer is an associative reduce operation
  - Min reduce (keep nearest pixel) on depth with an associated value
- Parallel strategy
  - Split triangles into sets
  - Project, rasterize, Z-buffer in parallel
  - Assoc. reduce Z-buffer pixels across parallel Z-buffers

25

Not Associative: Floating Point

- Floating-Point Addition
  - Due to rounding
    - (1+1E100)-1E100 = 0
    - 1+(1E100-1E100) = 1

26

Not Associative: Saturated Addition

- Saturated Addition

```
tmp=a+b;
if (tmp>MAXVAL) sum=MAXVAL;
          else sum=tmp;
```

- MAXVAL=255
  - 254+(20-3) = 255
  - (254+20)-3 = 252

27

Majority Associative?

- Carry=MAJ=majority
  = A&&B || B&&C || A&&C
- Is Majority Associative ?
- Hint: What are each of following?
  - MAJ(1,1,MAJ(1,1,MAJ(1,0,0)))
  - MAJ(MAJ(MAJ(1,1,1),1,1),0,0)

28

Teaser

- Can recast into associative operations
  - saturated add
  - Majority (Section 4)
- Can still use ideas with Floating Point

29

Part 2: Data Parallel+Reduce

IMPLEMENTATIONS

30

Threaded: Data Parallel+Reduce

- Break into P threads
  - 0 to N/P-1, N/P to 2N/P-1, …
- Run fraction of data and reduce on each
- Then bring results together to sum
  - P small, on one processor
  - P large, as tree

Vector: Data Parallel + Reduce

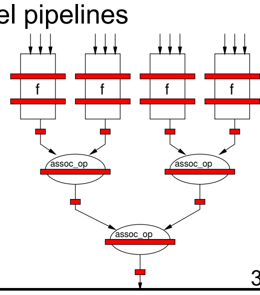
- Some vector/SIMD machines will have dedicated reduce hardware
- E.g. vector-add operator
- NEON
  - Not have vector reduce
  - Does have VPADAL
    - Pairwise adds
- Use VL adds for coarse-grained reduce (data parallel)

```
for (i=0;i<N;i+=VL) {
  avl=a[i]…a[i+VL-1]
  VADD(res,avl, res);
}
```

- Use VPADAL to complete
- Cycles?

Unrolled Pipeline: Data Parallel + Reduce

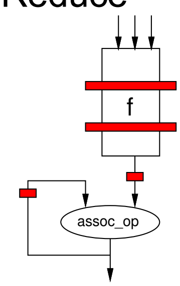
- Unroll computation
- Perform f ops in parallel pipelines
- Pipelined tree reduce
- Latency?
  - N f ops
  - Delay f – 3
  - Delay assoc -- 2



Model: Data Parallel+Reduce

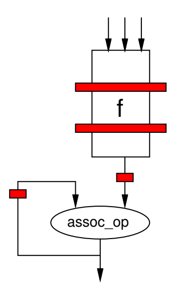
- What's cycle → what's II?
  - (concrete: assoc_op delay=2)
- General form

```
int res=0;
for (int i=0;i<N; i++)
  res=assoc_op(res,f(a[i],b[i], …))
```



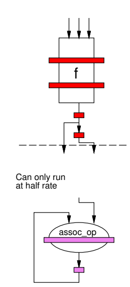
Pipeline: Data Parallel + Reduce

- Pipeline f
- Cycle on assoc_op



Pipeline: Data Parallel + Reduce

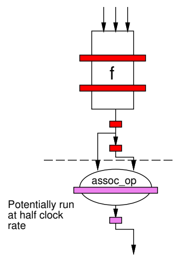
- Pipeline f
- Cycle on assoc_op
  - Cannot take input on every cycle



Shown II=2

Pipeline: Data Parallel + Reduce

- Pipeline f
- Cycle on assoc_op
  - Cannot take input on every cycle
- Can use assoc. reduce to combine groups of original II
  - Allow cycle to run at lower frequency

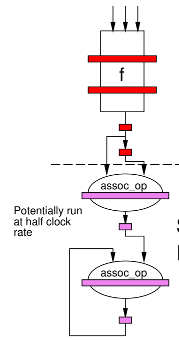


Shown II=2

37

Pipeline: Data Parallel + Reduce

- Pipeline f
- Cycle on assoc_op
- Avoid cycle, II=1 for associative
  - Gather up II values
  - Run through pipelined assoc. reduce tree
  - Drop into assoc_op cycle every II cycles



Shown II=2

38

Model: Data Parallel+Reduce

- **Conclude:** associative reduce can achieve II of 1
- General form

```
int res=0;
for (int i=0;i<N; i++)
  res=assoc_op(res,f(a[i],b[i], …))
```


40

Implement Reduce

- Can exploit with all of our parallel implementation forms
  - Multi-thread (multi-processor)
  - SIMD/Vector
  - Instruction
  - Pipeline
  - Spatial (unrolled)

44

Part 3

PARALLEL PREFIX

45

What if want Prefix?

Sum Reduce

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```

Sum Prefix

```
int sum[N];
sum[0]=a[0];
for (int i=1;i<N; i++)
    sum[i]=a[i]+sum[i-1];
```


46

Integers 1--5

Sum Reduce

```
int sum=0;
for (int i=0;i<N; i++)
    sum+=a[i];
```

→ sum=1+2+3+4+5=15

Integers 1--5

Sum Reduce = 15

Sum Prefix

```
int sum[N];
sum[0]=a[0];
for (int i=1;i<N; i++)
    sum[i]=a[i]+sum[i-1];
```

→ {1, 3, 6, 10, 15}

Prefix

- Aggregate (vector) output where item i is the reduce of the input vector 0 through i

```
prefix[0]=a[0];
for (int i=1;i<N; i++)
  prefix[i]=op(prefix[i-1],f(a[i]…));
```

- ``Prefix'' because given reduce of each prefix subset 0 to i

Latency Bound

- What's the latency bound for the prefix when op is associative?
  - Assume op is 1 cycle

```
prefix[0]=a[0];
for (int i=1;i<N; i++)
  prefix[i]=op(prefix[i-1],f(a[i]…));
```


Latency Bound

- Simple (not area efficient) answer:
  - Compute reduce for each prefix[i] in parallel
  - Latency bound? (single cycle op)

```
prefix[0]=a[0];
for (int i=1;i<N; i++)
  prefix[i]=op(prefix[i-1],f(a[i]…));
```

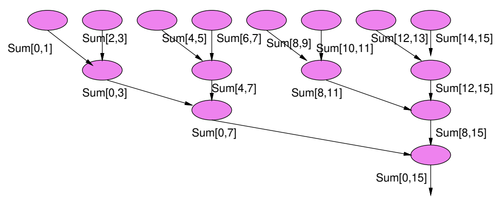

Resources?

- How much hardware to achieve within 2x latency bound?
  - Hint: can do better than simple case previous slide

```
prefix[0]=a[0];
for (int i=1;i<N; i++)
  prefix[i]=op(prefix[i-1],f(a[i]…));
```

Reduce Tree

- While computing Sum[0,N-1] compute many Sum[0,j]'s
  - Sum[0,1], Sum[0,3], Sum[0,7] ….

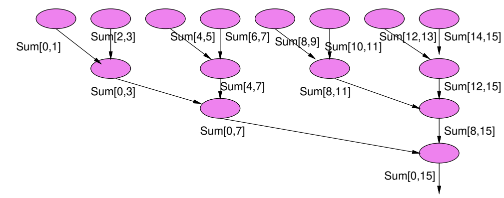


Penn ESE5320 Fall 2022 -- DeHon 53

53

Prefix Tree

- While computing Sum[0,N-1] only get
  - PG[0,2^n-1]
- How fillin holes?
- – e.g. how get Sum[0,11]?

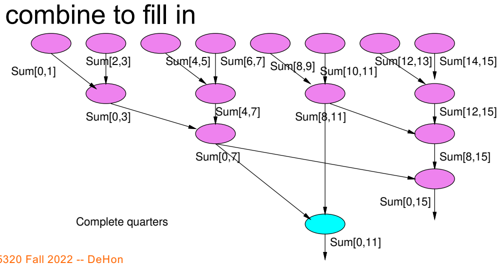


Penn ESE5320 Fall 2022 -- DeHon 54

54

Prefix Tree

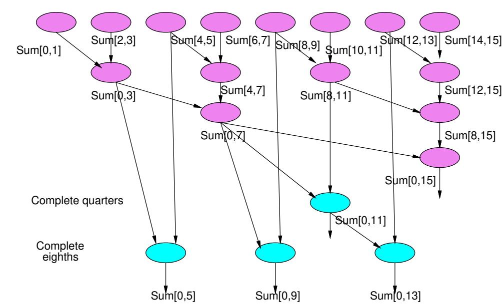
- Look at Symmetric stage (with respect to middle=Sum[0,N-1] stage) and combine to fill in



Penn ESE5320 Fall 2022 -- DeHon 55

55

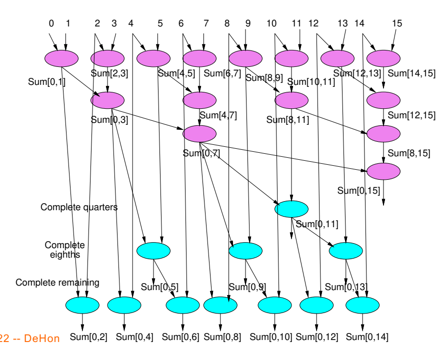
Prefix Tree



Penn ESE5320 Fall 2022 -- DeHon 56

56

Prefix Tree

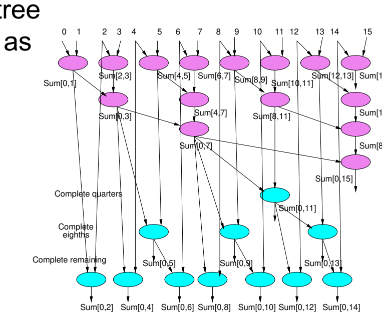


Penn ESE5320 Fall 2022 -- DeHon 57

57

Prefix Tree

- Note: prefix-tree is same size as reduce tree

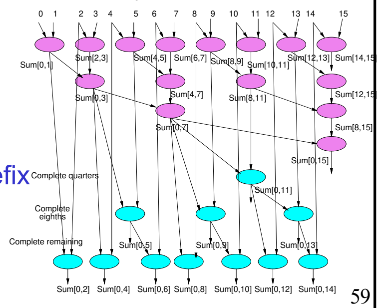


Penn ESE5320 Fall 2022 -- DeHon 58

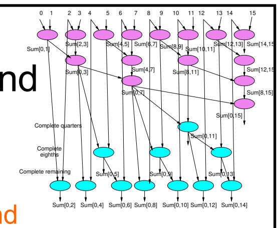
58

Parallel Prefix
Area and Delay?

- Roughly twice the area/delay
- Area= 2N
- Delay = $2\log_2(N)$
- Conclude: can compute prefix in log time with linear area.



Latency Bound



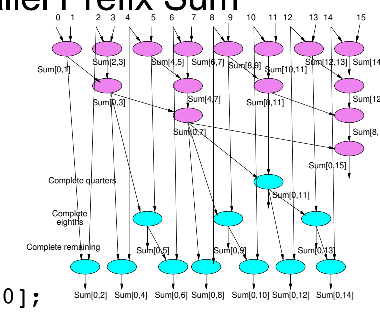
- What's the latency bound for the prefix when op is associative?
  - When cycles(op)>1?

```
prefix[0]=a[0];
for (int i=1;i<N; i++)
 prefix[i]=op(prefix[i-1],f(a[i]…));
```


Parallel Prefix

- Important **Pattern**
- Applicable any time operation is *associative*
  - Or can be made associative
- Function Composition is always associative
  - (Section 4)
- Logarithmic delay
- Linear area

Parallel Prefix Sum




```
prefix[0]=a[0];
for (int i=1;i<N; i++)
 prefix[i]=op(prefix[i-1],f(a[i]…));
```


BROADER APPLICATION

Cast Associative

- If you can cast it into an associative operation, you can apply
  - Associative Reduce
  - Parallel Prefix

Examples

- Saturated Addition
  - Not associative
- Floating-Point Addition
- Finite Automata Evaluation

- (papers in supplemental reading)

65

Categorization

- To minimize confusion, will typically ask you to characterize:
  - Data parallel
  - Reduce
  - Sequential

66

Part 4

BINARY ADDITION

67

Majority Associative?

- Carry=MAJ=majority
  = A&&B || B&&C || A&&C

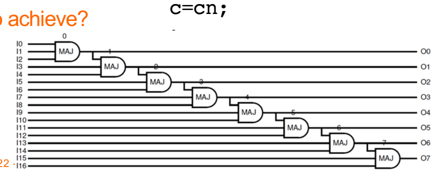
- Is Majority Associative ?
- Hint: What are each of following?
  - MAJ(1,1,MAJ(1,1,MAJ(1,0,0)))
  - MAJ(MAJ(MAJ(1,1,1),1,1),0,0)

68

Binary Addition

- Binary addition needs parallel prefix on majority
- Adding 2 W-bit numbers
  - What's the latency bound?
  - Area to achieve?

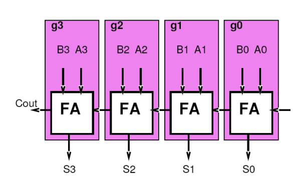
```
• boolean a[i],b[i],s[i]
• for (i=0;i<W;i++) {
   cn=(a[i]&&b[i])||
          (a[i]&&c)||
          (b[i]&&c);
   s[i]=a[i] ^ b[i] ^ c;
   c=cn;
```



69

Carry Computation

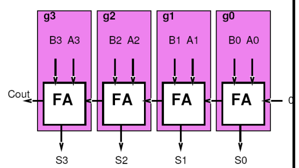
- Think about each adder bit as a computing a function on the carry in
  - C[i]=g(c[i-1])
  - Particular function f will depend on a[i], b[i]
  - g=f(a,b)



70

Functions

- Carry=MAJ=majority
  = A&&B || B&&C || A&&C
- What are the functions g(c[i-1])?
  - g(c)=carry(a=0,b=0,c)
  - g(c)=carry(a=1,b=0,c)
  - g(c)=carry(a=0,b=1,c)
  - g(c)=carry(a=1,b=1,c)



71

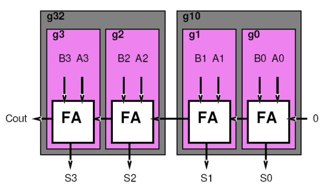
Functions

- What are the functions g(c[i-1])?
  - g(x)=1 Generate
    - a[i]=b[i]=1
  - g(x)=x Propagate
    - a[i] xor b[i]=1
  - g(x)=0 Squash
    - a[i]=b[i]=0

72

Combining

- Want to combine functions
  - Compute $c[i]=g_i(g_{i-1}(c[i-2]))$
  - Compute compose of two functions
- What functions will the compose of two of these functions be?
  - Same as before
    - Propagate, generate, squash



73

Compose Rules (LSB MSB)

- GG
- GP
- GS
- PG
- PP
- PS
- SG
- SP
- SS

[work on board]

74

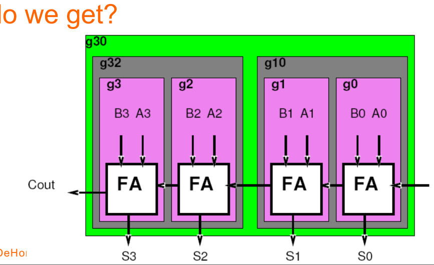
Compose Rules (LSB MSB)

- GG = G
- GP = G
- GS = S
- PG = G
- PP = P
- PS = S
- SG = G
- SP = S
- SS = S

75

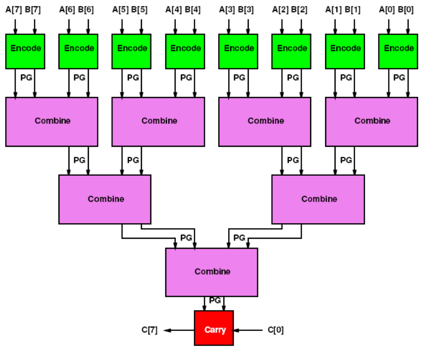
Combining

- Do it again…
- Combine g[i-3,i-2] and g[i-1,i]
- What do we get?



76

Associative Reduce Tree

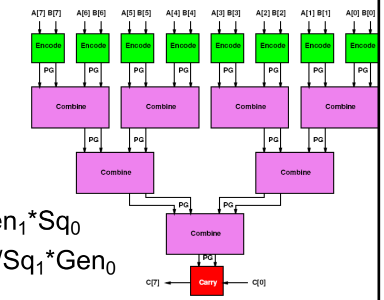


77

Reduce Tree

- Sq=/A*/B
- Gen=A*B

- Sq_{out}=Sq_1+/Gen_1*Sq_0
- Gen_{out}=Gen_1+/Sq_1*Gen_0

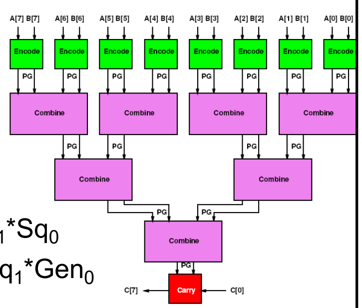


78

Reduce Tree

- Sq=/A*/B
- Gen=A*B

- Sq_{out}=Sq_1+/Gen_1*Sq_0
- Gen_{out}=Gen_1+/Sq_1*Gen_0



- Delay and Area? (work next few slides)

79

Reduce Tree

- Sq=/A*/B
- Gen=A*B

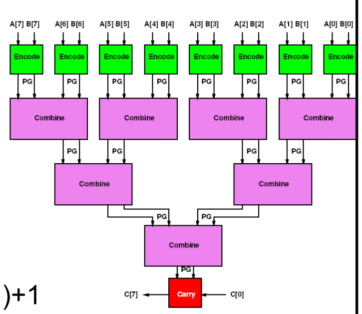
- Sq_{out}=Sq_1+/Gen_1*Sq_0
- Gen_{out}=Gen_1+/Sq_1*Gen_0

- A(Encode)=2
- D(Encode)=1
- A(Combine)=4
- D(Combine)=2
- A(Carry)=2
- D(Carry)=1

80

Reduce Tree: Delay?

- D(Encode)=1
- D(Combine)=2
- D(Carry)=1

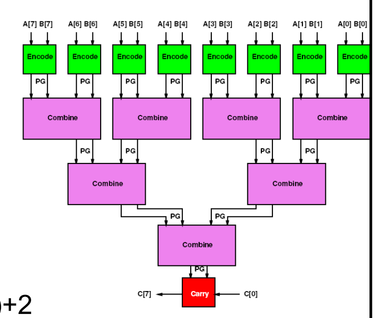


Delay = 1+2$\log_2$(N)+1

81

Reduce Tree: Area?

- A(Encode)=2
- A(Combine)=4
- A(Carry)=2

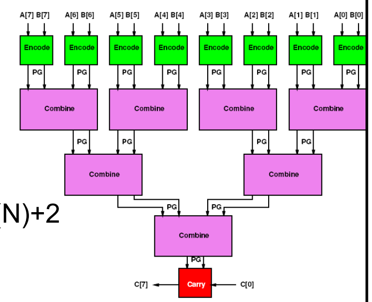


Area= 2N+4(N-1)+2

82

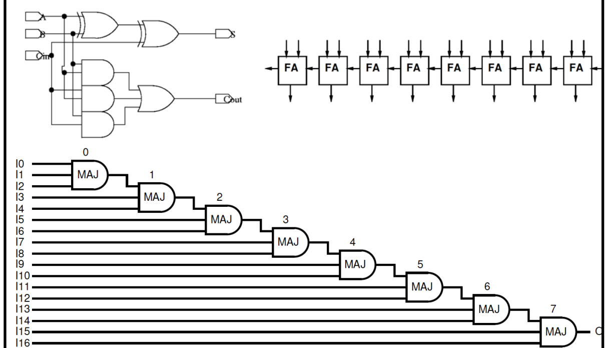
Reduce Tree: Area & Delay

- Area(N) = 6N-2
- Delay(N) = $2\log_2(N)+2$



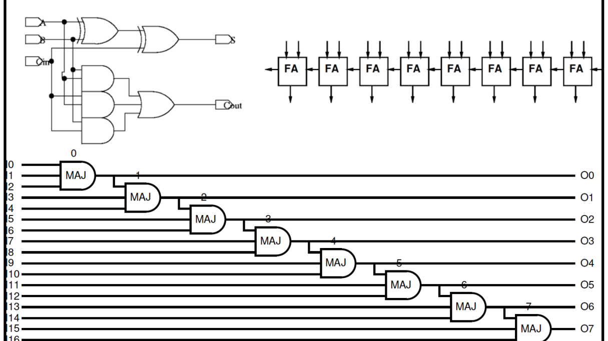
83

Compute Carry[N]



84

Need Prefix

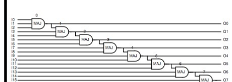


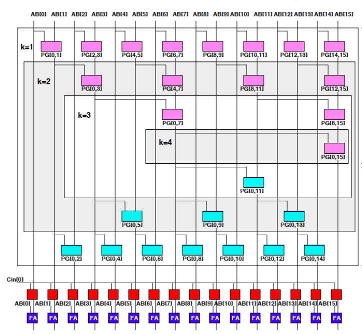
Need PG[i:0] forall i

85

Prefix Tree

- Bring in Carry and compute each intermediate carry-in



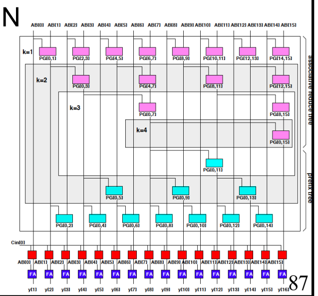


86

Parallel Prefix
Area and Delay?

- Roughly twice the area/delay
- Area= 2N+4N+4N+2N
  = 12N
- Delay = $4\log_2(N)+2$

- Conclude:
  can add in log time with linear area.



87

Big Ideas:

- Reduce from aggregate to scalar
  - is a common operation
  - not strictly data parallel
  - Associative reduce admits to parallelism
    - log(N) latency bound
    - II=1
    - Linear area
- Prefix when want reduce of all prefixes
  - Also log(N) latency bound
  - Linear area

88

Admin

- Wednesday is a virtual Friday – no lecture