ESE532: System-on-a-Chip Architecture

Day 25: November 25, 2019
Reduce

Today

- Reduce
- Associative Operations
- Model
- Latency Bound Implications and Implementations
- Parallel Prefix
- Broad Application

Message

- Aggregation is a common need that is not strictly data parallel
- …but admits to parallel computation

Reduce

- Reduce – combining a collection of data into a single value
  - Converting a vector into a scalar

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?

```
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];
```

Penn ESE532 Fall 2019 -- DeHon 7

Associative Operations

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

Penn ESE532 Fall 2019 -- DeHon 8

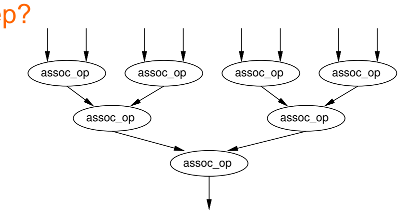
Associative Operations

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

Penn ESE532 Fall 2019 -- DeHon 9

Associative Tree Reduce

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



Penn ESE532 Fall 2019 -- DeHon 10

Latency Bounds

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

Penn ESE532 Fall 2019 -- DeHon 11

Sum Reduce

- Data Parallel?

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

Penn ESE532 Fall 2019 -- DeHon 12

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

```
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 addition, what other associative operations do we often see as reductions?

Associative Operations

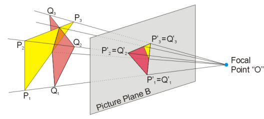
- Add
- Multiply
- Max
- Min
- AND
- OR
- Max/min
  - And keep associated values
- Find First

Optimization Loop

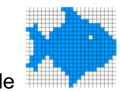
```
int minval=f(0);
int min=0;
for (i=1;i<N;i++) {
    int val=f(i);
    if (val<minval) {
        minval=val; min=i;
      }
}
```


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

Data Parallel+Preduce

IMPLEMENTATIONS

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

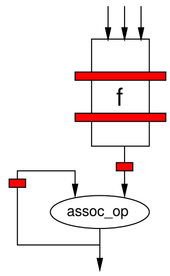
Model: Data Parallel+Reduce

- What's cycle → what's II?
- General form

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

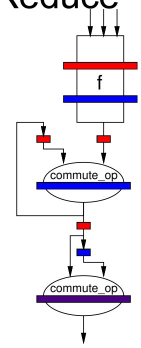
Pipeline: Data Parallel + Reduce

- Pipeline f
- Cycle on assoc_op



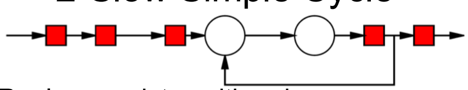
Pipeline: Data Parallel + Reduce

- Pipeline f
- Cycle on assoc_op
- Avoid cycle, II=1 for commutative
  - Run interleaved
    - C-slow (C=II)
  - Combine at end
- Commutative operation
  - Can reorder operands



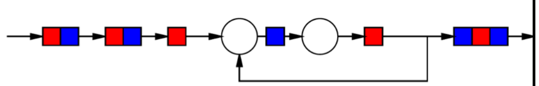
2-Slow Simple Cycle

Day 7



- Replace register with pair

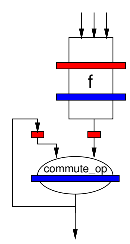
- Retime



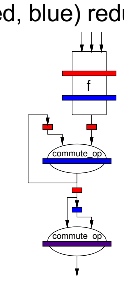
- Observe independence of red/blue computations

II=2 Commutative Pipelined Reduce

- 2-slow transformation

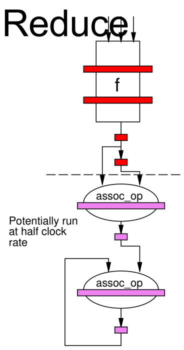


- Combine independent (red, blue) reduces



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



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], …))
```

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

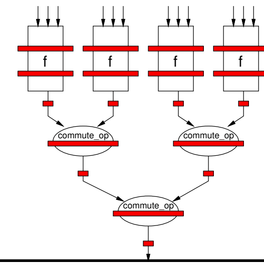
- Use VL adds for course-grained reduce

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

- Use VPADAL to complete

Unrolled Pipeline: Data Parallel + Reduce

- Unroll computation
- Perform f ops in parallel pipelines
- Pipelined tree reduce



PARALLEL PREFIX

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];
```


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]…));
```


Latency Bound

- What's the latency bound for the prefix when op is associative?
  - Assume op is 1 cycle
  - How cylces(op)>1 change?

```
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 latency bound?

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


Latency Bound?

- What's the latency bound for this operation?

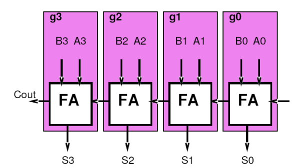
```
• boolean a[i],b[i],s[i]
• for (i=0;i<N;i++) {
   cn=(a[i]&&b[i])||
        (a[i]&&c)||
        (b[i]&&c);
   s[i]=a[i] ^ b[i] ^ c;
   c=cn;
   }
```


Associative

- Is the carry operation in addition an associative operation?
- Operation:
  - MAJ=majority = A&&B || B&&C || A&&B

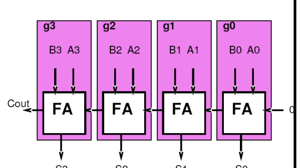
Carry Computation

- Think each adder bit as a function 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)



Functions

- 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)

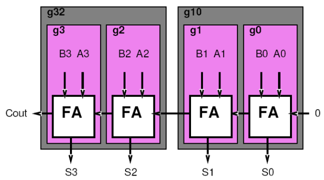


Functions

- 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

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



Compose Rules (LSB MSB)

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

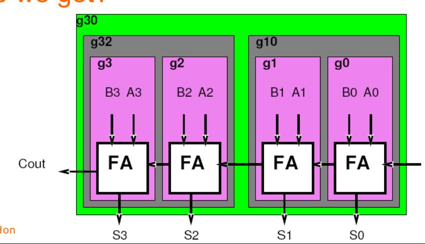
[work on board]

Compose Rules (LSB MSB)

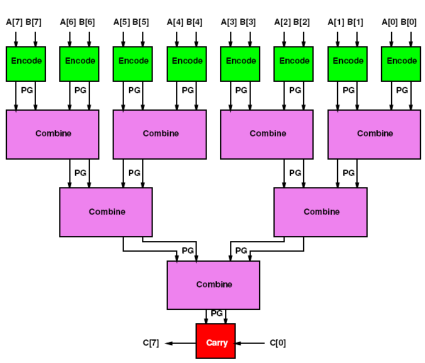
- GG = G
- GP = G
- GS = S
- PG = G
- PP = P
- PS = S
- SG = G
- SP = S
- SS = S

Combining

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

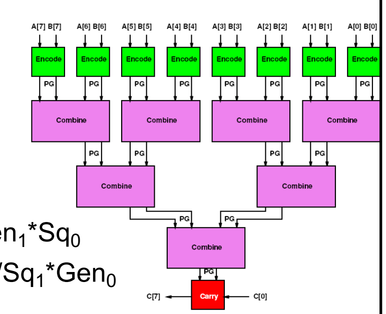


Associative Reduce Tree



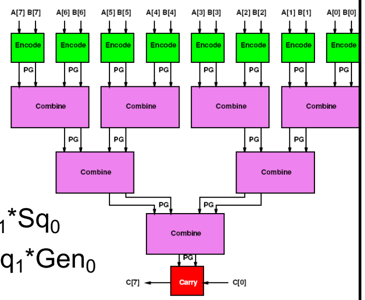
Reduce Tree

- Sq=/A*/B
- Gen=A*B
- Sq_{out}=Sq_1+/Gen_1*Sq_0
- Gen_{out}=Gen_1+/Sq_1*Gen_0



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)

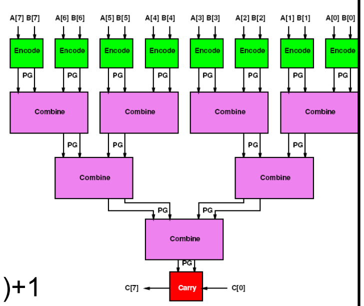
Reduce Tree

- 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

Reduce Tree: Delay?

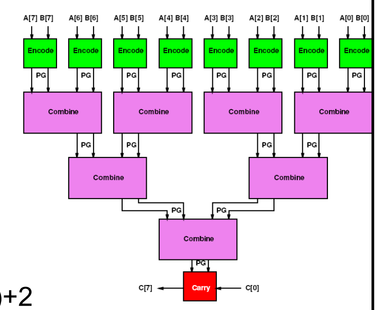
- D(Encode)=1
- D(Combine)=2
- D(Carry)=1



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

Reduce Tree: Area?

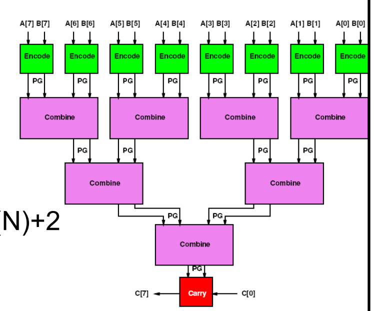
- A(Encode)=2
- A(Combine)=4
- A(Carry)=2



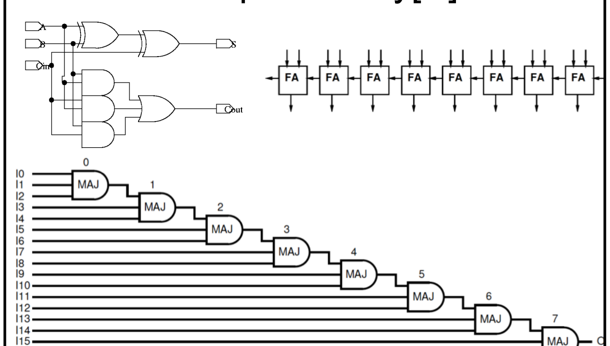
Area= 2N+4(N-1)+2

Reduce Tree: Area & Delay

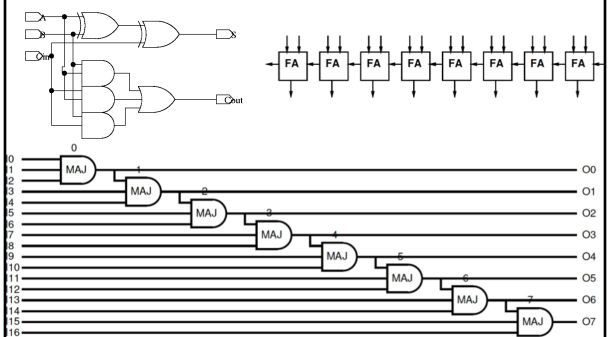
- Area(N) = 6N-2
- Delay(N) = $2\log_2(N)+2$



Compute Carry[N]



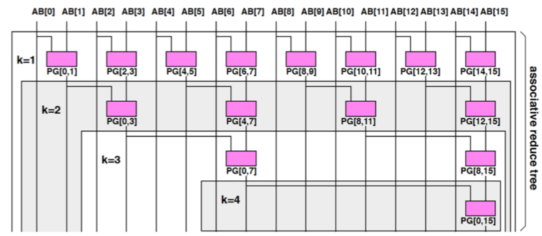
Need Prefix



Need PG[i:0] forall i

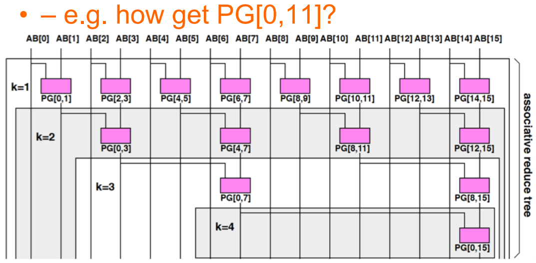
Reduce Tree

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



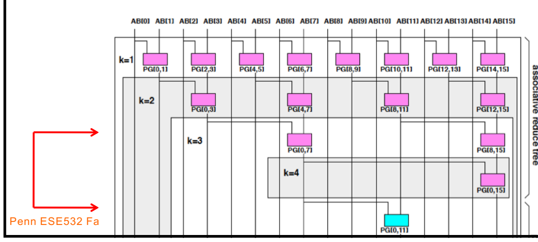
Prefix Tree

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

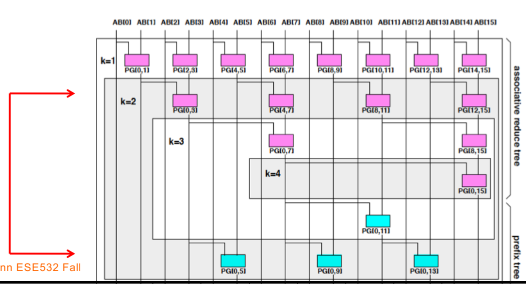


Prefix Tree

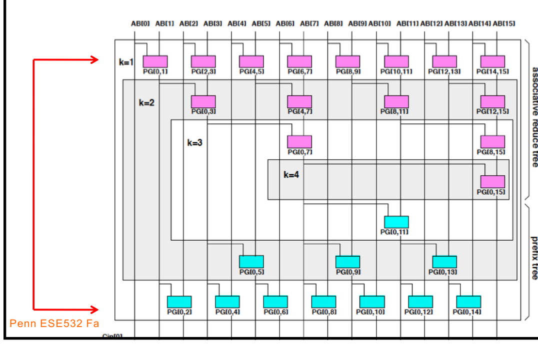
- Look at Symmetric stage (with respect to middle=PG[0N] stage) and combine to fillin



Prefix Tree

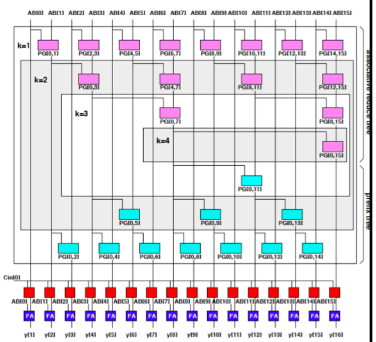


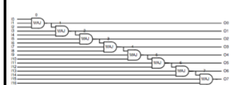
Prefix Tree



Prefix Tree

- Bring in Carry and compute each intermediate carry-in

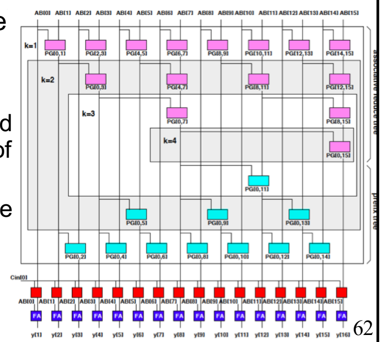




Penn ESE532 Fall 2019 -- DeHon

Prefix Tree

- Note: prefix-tree is same size as reduce tree
  - Always matched same number of elements in symmetric stage

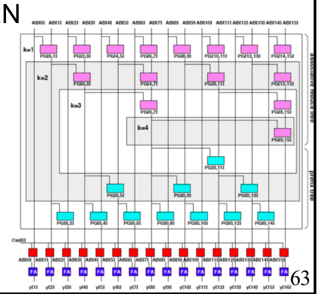


Penn ESE532 Fall 2019 -- DeHon

62

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.



Penn ESE532 Fall 2019 -- DeHon

63

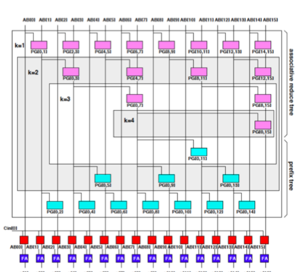
Parallel Prefix

- Important **Pattern**
- Applicable any time operation is *associative*
  - Or can be made assoc. as in MAJ case
- Function Composition is always associative
- Logarithmic delay
- Linear area

Penn ESE532 Fall 2019 -- DeHon

64

Parallel Prefix Sum



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

Penn ESE532 Fall 2019 -- DeHon

65

BROADER APPLICATION

Penn ESE532 Fall 2019 -- DeHon

66

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)

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

Admin

- Wednesday is a virtual Friday
  - Will not meet
  - Happy Thankgiving
  - Nothing due Friday (virtual or real)
- Project due **following** Friday (12/6)