ESE532: System-on-a-Chip Architecture

Day 2: September 5, 2018
Analysis, Metrics, and Bottlenecks

Work Preclass
Lecture start 10:35pm

Today: Analysis

- How do we quickly estimate what's possible?
  - Before (with less effort than) developing a complete solution
- How should we attack the problem?
  - Achieve the performance, energy goals?
- When we don't like the performance we're getting, how do we understand it?
- Where should we spend our time?

Today: Analysis

- Throughput
- Latency
- Bottleneck
- Computation as a Graph, Sequence
- Critical Path
- Resource Bound
- 90/10 Rule
- Amdahl's Law (time permitting)

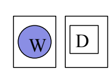
Message for Day

- Identify the Bottleneck
  - May be in compute, I/O, memory, data movement
- Focus and reduce/remove bottleneck
  - More efficient use of resources
  - More resources
- Repeat

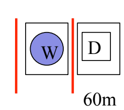
Latency vs. Throughput

- **Latency:** Delay from inputs to output(s)
- **Throughput:** Rate at which can produce new set of outputs
  - (alternately, can introduce new set of inputs)

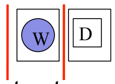
Preclass Washer/Dryer Example



- 10 shirt capacity
- 1 Washer Takes 30 minutes
- 1 Dryer Takes 60 minutes
- How long to do one load of wash?
  - → Wash latency
- Cleaning Throughput?



Pipeline Concurrency



- Break up the computation graph into stages
  - Allowing us to
    - reuse resources for new inputs (data),
    - while older data is still working its way through the graph
      - Before it has exited graph
  - Throughput > (1/Latency)
- Relate liquid in pipe
  - Doesn't wait for first drop of liquid to exit far end of pipe before accepting second drop

Escalator

Image Source: https://commons.wikimedia.org/wiki/File:Tanforan_Target_escalator_1.JPG

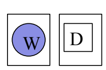
Escalator

- Moves 2 ft/second
- Assume for simplicity one person can step on escalator each second
- Escalator travels 30 feet (vertical and horizontal)
- Latency of escalator trip?
- Throughput of escalator: people/hour ?

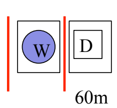
Bottleneck

- What is the rate limiting item?
  - Resource, computation, ….

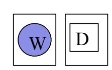
Preclass Washer/Dryer Example



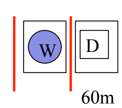
- 1 Washer Takes 30 minutes
  - Isolated throughput 20 shirts/hour
- 1 Dryer Takes 60 minutes
  - Isolated throughput 10 shirts/hour
- Where is bottleneck in our cleaning system?



Preclass Washer/Dryer Example

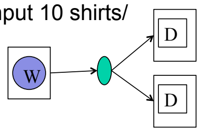


- 1 Washer $500
  - Isolated throughput 20 shirts/hour
- 1 Dryer $500
  - Isolated throughput 10 shirts/hour
- How do we increase throughput with $500 investment



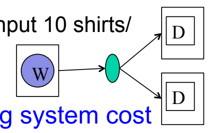
Preclass
Washer/Dryer Example

- 1 Washer $500
  – Isolated throughput 20 shirts/hour
- 2 Dryers $500
  – Isolated single dryer throughput 10 shirts/hour
- Latency?
- Throughput?

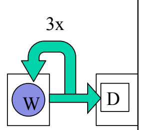


Preclass
Washer/Dryer Example

- 1 Washer $500
  – Isolated throughput 20 shirts/hour
- 2 Dryers $500
  – Isolated single dryer throughput 10 shirts/hour
- Able to double the throughput without doubling system cost



Preclass
Stain Example

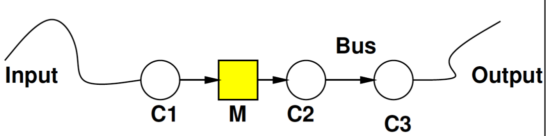


- 1 Washer Takes 30 minutes
  – Isolated throughput 20 shirts/hour
- 1 Dryer Takes 60 minutes
  – Isolated throughput 10 shirts/hour
- Shirt need 3 wash cycles
- Latency?
- Throughput (assuming share)?

Beyond Computation

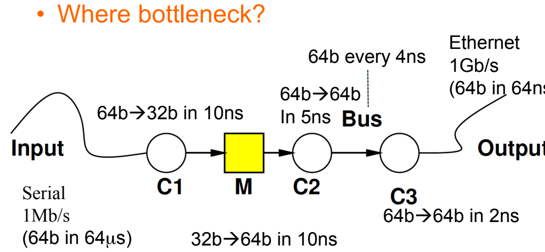
Bottleneck

- May be anywhere in path
  – I/O, compute, memory, data movement



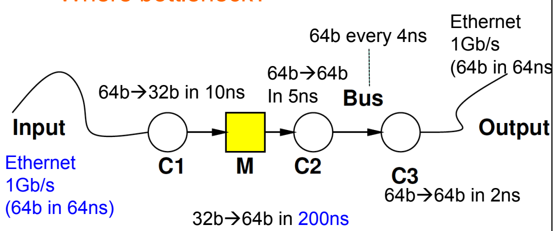
Bottleneck

- Where bottleneck?



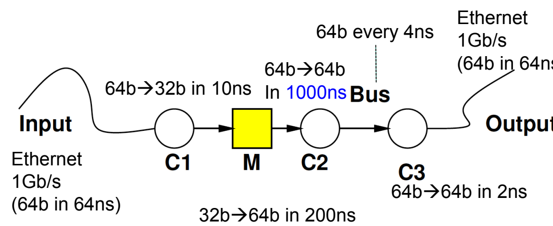
Bottleneck

- Where bottleneck?



Bottleneck

- Where bottleneck?



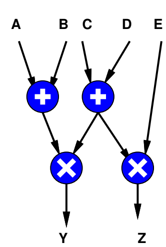
Feasibility / Limits

- First things to understand
  - Obvious limits in system?
- Impossible?
- Which aspects will demand efficient mapping?
- Where might there be spare capacity

Generalizing

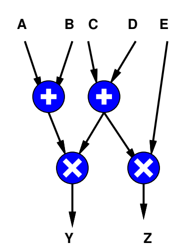
Computation as Graph

- Shown “simple” graphs (pipelines) so far
- Y=(A+B)*(C+D)
- Z=(C+D)*E



Computation as Graph

- Nodes have multiple input/output edges
- Edges may fanout
  - Results go to multiple successors



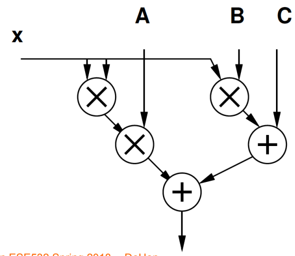
Computation as Sequence

- Shown "simple" graphs (pipelines) so far
- Y=(A+B)*(C+D)
- Z=(C+D)*E

T1=A+B
T2=C+D
Y=T1*T2
Z=T1*E

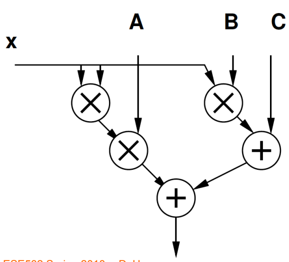
Computation as Graph

- $Y=Ax^2+Bx+C$



T1=x*x
T2=A*T1
T3=B*x
T4=T2+T3
Y=C+T4

Computation as Graph

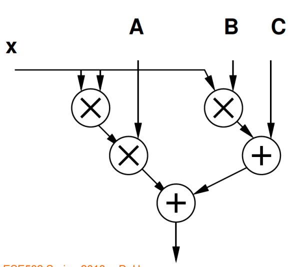


- Latency multiply = 3
- Latency add = 1
- Latency from B to output?
- Latency from x to output?
  - Through Ax^2 ?
  - Through Bx ?

Delay in Graphs

- There are multiple paths from inputs to outputs
- Need to complete all of them to produce outputs
- Limited by longest path
- **Critical path:** longest path in the graph

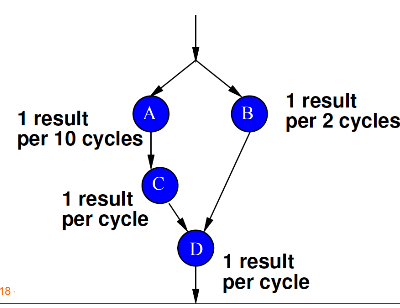
Computation as Graph



- Latency multiply = 3
- Latency add = 1
- Critical Path?

Bottleneck

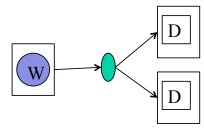
- Where is the bottleneck?



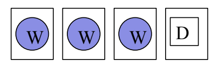
Time and Space

Space-Time

- In general, we can spend resources to reduce time
  - Increase throughput

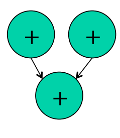


Three wash stain removal case



Space Time

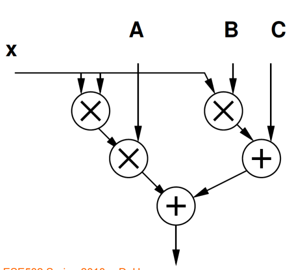
- Computation
  - A=x0+x1
  - B=A+x2
  - C=B+x3
- Adder takes one cycle
- Throughput on one adder?
- Throughput on 3 adders?



Dependencies and S-T

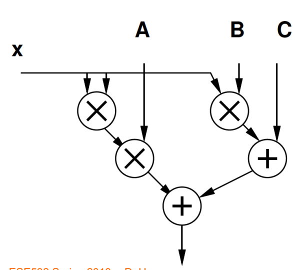
- Dependencies may limit throughput acceleration
  - Give benefit less than 1/space

Computation as Graph



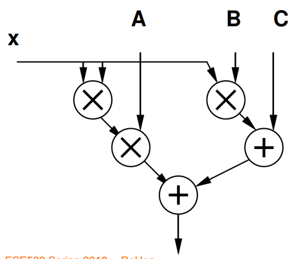
- Latency multiply = 1
- Thput mult = 1
- Space multiply = 3
- Latency add = 1/3
- Space add = 1
- Thput and Space
  - 3 mul, 2 add

Computation as Graph



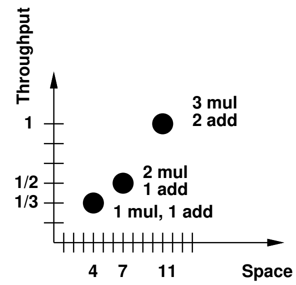
- Latency multiply = 1
- Thput mult = 1
- Space multiply = 3
- Latency add = 1/3
- Space add = 1
- Thput and Space
  - 1 mul, 1 add
  - Where is bottleneck?

Computation as Graph



- Latency multiply = 1
- Thput mult = 1
- Space multiply = 3
- Latency add = 1/3
- Space add = 1
- Thput and Space
  - 2 mul, 1 add
  - (no algebraic optimizations, iterations may overlap)

Space-Throughput Graph



Two Bounds

(still in Time and Space)

Bounds

- Quick lower bounds can estimate
- Two:
  - CP: Critical Path
    - Sometimes call it “Latency Bound”
  - RB: Resource Bound
    - Sometimes call it “Throughput Bound” or “Compute Bound”

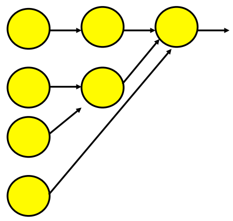
Critical Path Lower Bound

- Critical path assuming infinite resources
- Certainly cannot finish any faster than that

Resource Capacity Lower Bound

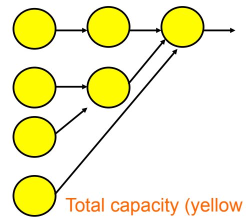
- Sum up all capacity required per resource
  - E.g. number of multiplications, additions, memory lookups
- Divide by total resource (for type)
  - E.g., number of multipliers, adders, memory ports
- Lower bound on compute
  - (best can do is pack all use densely)
  - Ignores data dependency constraints

Example



Critical Path

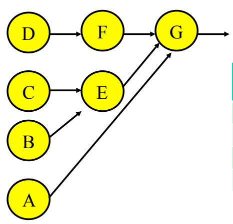
Example



Total capacity (yellow circle evaluations) needed?

Resource Bound (2 resources)?

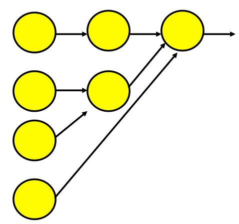
Example



Cycle	Resource 1	Resource 2
0	A	B
1	C	D
2	E	F
3	G	

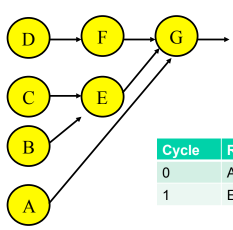
Resource Bound (2 resources)?

Example



Resource Bound (4 resources)?

Example



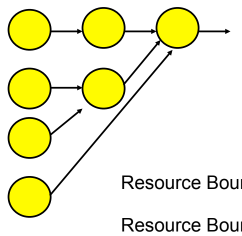
Cycle	R1	R2	R3	R4
0	A	B	C	D
1	E	F	G	

Legal Schedule?

Resource Capacity Lower Bound

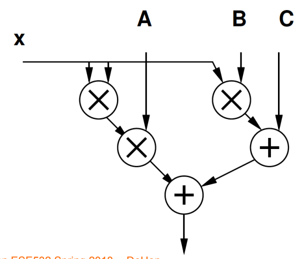
- Sum up all capacity required per resource
  - E.g. number of multiplications, additions, memory lookups
- Divide by total resource (for type)
  - E.g., number of multipliers, adders, memory ports
- Lower bound on compute
  - (best can do is pack all use densely)
  - Ignores data dependency constraints

Example



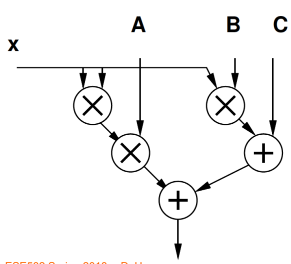
Critical Path	3
Resource Bound (2 resources)	7/2=4
Resource Bound (4 resources)	7/4=2

Critical Path



- Latency multiply = 3
- Thput mult = 1/3
- Space multiply = 3
- Latency add = 1
- Space add = 1
- Critical Path?

Resource Bound



- Latency multiply = 3
- Thput mult = 1/3
- Space multiply = 3
- Latency add = 1
- Space add = 1
- Resource Bound
  - 1 mul, 1 add
  - 2 mul, 1 add
  - 3 mul, 2 add

90/10 Rule (of Thumb)

- Observation that code is not used uniformly
- 90% of the time is spent in 10% of the code
- Knuth: 50% of the time in 2% of the code
- Implications
  - There will typically be a bottleneck
  - We don't need to optimize everything
  - We don't need to uniformly replicate space to achieve speedup
  - Not everything needs to be accelerated

Amdahl's Law

- If you only speedup Y(%) of the code, the most you can accelerate your application is 1/(1-Y)
- T_{before} = 1*Y + 1*(1-Y)
- Speedup by factor of S
- T_{after}=(1/S)*Y+1*(1-Y)
- Limit S→infinity T_{before}/T_{after}=1/(1-Y)

Amdahl's Law

- T_{before} = 1*Y + 1*(1-Y)
- Speedup by factor of S
- T_{after}=(1/S)*Y+1*(1-Y)
- Y=70%
  - Possible speedup (S→infinity) ?
  - Speedup if S=10?

Amdahl's Law

- If you only speedup Y(%) of the code, the most you can accelerate your application is 1/(1-Y)
- Implications
  - Amdhal: good to have a fast sequential processor
  - Keep optimizing
    - T_{after}=(1/S)*Y+1*(1-Y)
    - For large S, bottleneck now in the 1-Y

Big Ideas

- Identify the Bottleneck
  - May be in compute, I/O, memory ,data movement
- Focus and reduce/remove bottleneck
  - More efficient use of resources
  - More resources

Admin

- Reading for Day 3 on web
- HW1 due Friday
- HW2 out
  - Partner assignment and board shuffle (see canvas)
- Remember feedback