ESE535: Electronic Design Automation

Day 12: February 23, 2011
Routing 2
(Pathfinder)

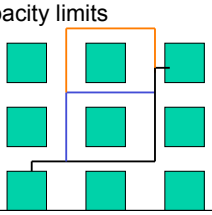
Today

- Routing
  - Pathfinder
    - graph based
    - global routing
    - simultaneous global/detail

Behavioral
(C, MATLAB, …)
↓ Arch. Select Schedule
RTL
↓ FSM assign, Two-level, Multilevel opt., Covering, Retiming
Gate Netlist
↓ Placement, Routing
Layout
↓
Masks

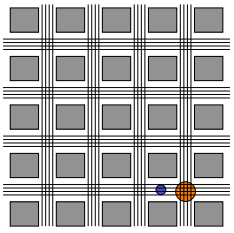
Global Routing

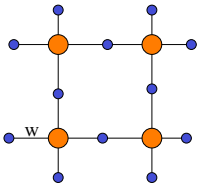
- **Problem:** Find sequence of channels for all routes
  - minimizing channel sizes
  - minimize max channel size
  - meeting channel capacity limits



Global→Graph

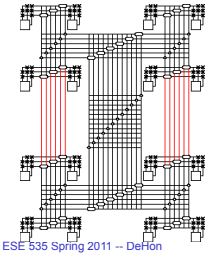
- Graph Problem on routes through regions

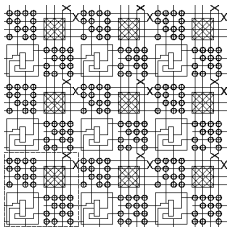


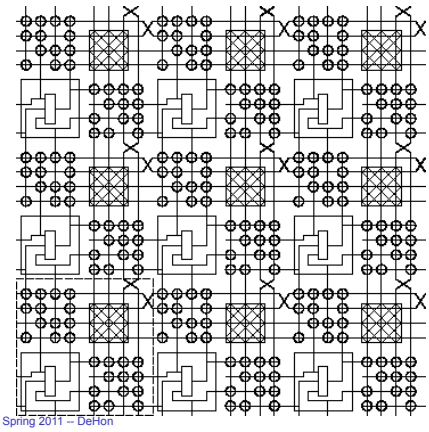


Global/Detail

- With limited switching (*e.g.* FPGA)
  - can represent routing graph exactly

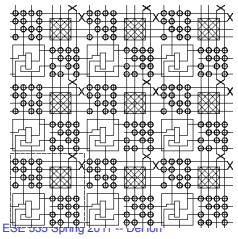






Routing in Graph

- Find (shortest/available) path between source and sink
  - search problem (e.g. BFS, A*)



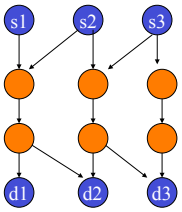
Breadth First Search (BFS)

- Start at source src
- Put src node in priority queue with cost 0
  - Priority queue orders by cost
- While (not found sink)
  - Pop least cost node from queue
    - Get: current_node, current_cost
  - Is this sink? → found
  - For each outgoing edge from current_node
    - Push destination onto queue
    - with cost current_cost+edge_cost

Easy?

- Finding a path is moderately easy
- What's hard?
- Can I just iterate and pick paths?

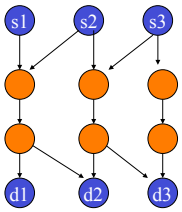
Example



All links capacity 1

$s_i \rightarrow d_i$

Challenge



- Satisfy *all* routes simultaneously
- Routes share potential resources
- Greedy/iterative
  - not know who will need which resources
    - E.g. consider routing s3->d3 then s2->d2 then s1->d1
  - *i.e.* resource/path choice looks arbitrary
  - …but earlier decisions limit flexibility for later
    - like scheduling
  - order effect result

Negotiated Congestion

- Old idea
  - try once
  - see where we run into problems
  - undo problematic/blocking allocation
    - rip-up
  - use that information to redirect/update costs on subsequent trials
    - retry

Negotiated Congestion

- Here
  - route signals
  - allow overuse
  - identify overuse and encourage signals to avoid
    - reroute signals based on overuse/past congestion

Basic Algorithm

- Route signals along minimum cost path
- If congestion/overuse
  - assign higher cost to congested resources
- Repeat until done

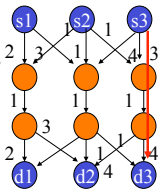
Key Idea

- Congested paths/resources become expensive
- When there is freedom
  - future routes, with freedom to avoid congestion will avoid it
- When there is less freedom
  - must take congested routes
- Routes which must use congested resources will, while others will chose uncongested paths

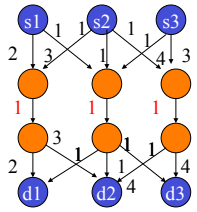
Cost Function (1)

- PathCost=Σ (link costs)
- LinkCost = base × f(#routes using, time)
- Base cost of resource
  - *E.g.* delay of resource
  - Encourage minimum resource usage
    - (minimum length path, if possible)
  - minimizing delay = minimizing resources
- Congestion
  - penalizes (over) sharing
  - increase sharing penalty over time

3+1+4=8



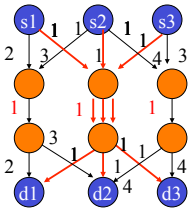
Example (first order congestion)



Capacity

Base costs (delays)

Example (first order congestion)

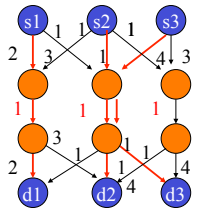


Capacity

Base costs (delays)

All, individual routes prefer middle; create congestion.

Example (first order congestion)

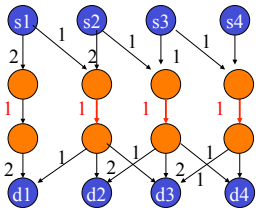


Capacity

Base costs (delays)

Reroute, avoid congestion.

Example (need for history)

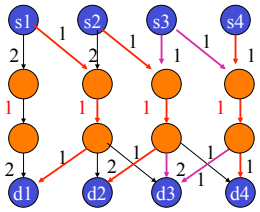


Capacity

Base costs (delays)

Need to redirect uncongested paths; how encourage?

Example (need for history)



Local congestion alone won't drive in right directions.

Both paths equal cost …neither resolves problem.

May ping-pong back and forth.

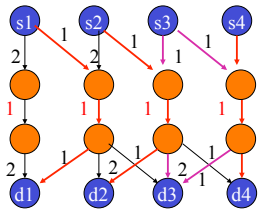
(can imagine longer chain like this)

Cannot route s3→d3

Cost Function (2)

- Cost = (base + history)*f(#resources,time)
- History
  - avoid resources with history of congestion

Example (need for history)



S3→d3 and s4→d4 initially ping-pong

Builds up congestion history on path 3 and 4

Eventually makes path 3 and 4 more expensive than path 1; …resolves conflict…

➔ Adaptive cost scheme.

Delay

What about delay?

- Existing formulation uses delay to reduces resources, but doesn't directly treat
- Want:
  - prioritize critical path elements for shorter delay
  - allow nodes with slack to take longer paths

Cost Function (Delay)

- Cost=
  - (1-W(edge))*delay + W(edge) *congest
  - congest as before
    - (base+history)*f(#signals,time)
- W(edge) = Slack(edge)/Dmax
  - 0 for edge on critical path
  - >0 for paths with slack
- Use W(edge) to order routes
- Update critical path and W each round

Cost Function (Delay)

- Cost=
  - (1-W(edge))*delay + W(edge) *congest
  - congest as before
    - (base+history)*f(#signals,time)
- W(edge) = Slack(edge)/D_{max}
- What happens if multiple slack 0 nets contend for edge?
- W(edge)=Max(minW,Slack(edge)/D_{max})
  - minW > 0

VPR

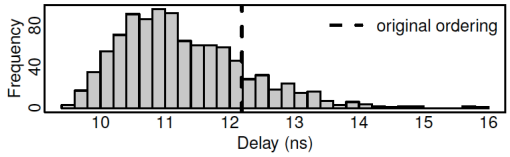
- If doesn't uncongest, weight congestion more
- Cost=

  (1-W(e))*delay + W(e) *$PF^{(iter)}$*congest

  PF=Pressure Factor
- Eventually congest dominates delay

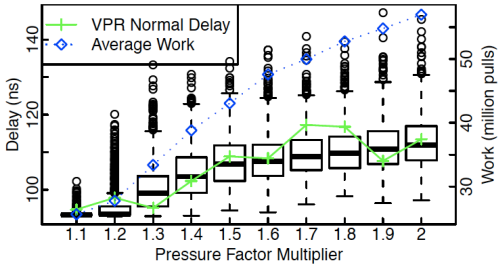
VPR Pressure Factor

- Converges quickly
- But may "freeze" with higher delay than necessary
- Netlist Shuffle experiment



[Rubin / FPGA 2011]

VPR Pressure Factor Tuning



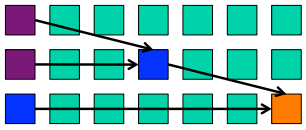
[Raphael Rubin 2010]

Alternate Delay Approach

- Believe pathfinder can resolve congestion
- Pathfinder has trouble mixing delay and congestion
- **Idea:** Turn delay problem into congestion problem
  - Reject paths that are too long
  - All signals compete only for resources that will allow them to meet their timing goals

Outlaw Long Paths

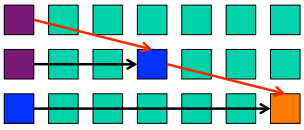
- Issue: Critical path may go through multiple gates
  - Contain more than one gate→gate path
  - How allocate slack among paths?



Target 12
Gate 1
Manhattan hop 1

Outlaw Long Paths

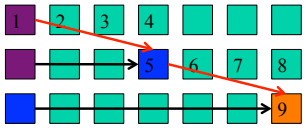
- Issue: Critical path may go through multiple gates
  - Contain more than one gate→gate path
  - How allocate slack among paths?



Target 12
Gate 1
Manhattan hop 1

Outlaw Long Paths

- Issue: Critical path may go through multiple gates
  - Contain more than one gate→gate path
  - How allocate slack among paths?

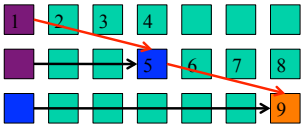


Target 12
Gate 1
Manhattan hop 1

Total Slack?

Slack Budgeting

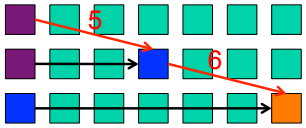
- Divide slack among the paths
  - Slack of 3
  - Example: give slack 1 to first link
    2 to second



 [So / FPGA 2008]

Slack Budgeting

- Divide slack among the paths
- Each net now has delay target
- Reject any path exceeding delay target
- Reduce to congestion negotiation



 [So / FPGA 2008]

Slack Budgeting

- Can often find lower delay routes that VPR
- Takes 10x as long
  - Mostly in slack budgeting
- Solution depends on slack budget
  - Not exploiting full freedom to re-allocate slack among links

[So / FPGA 2008]

Delay Target Routing

- Similar high-level idea
- Just set target for Pathfinder cost
  - Rather than allowing to float

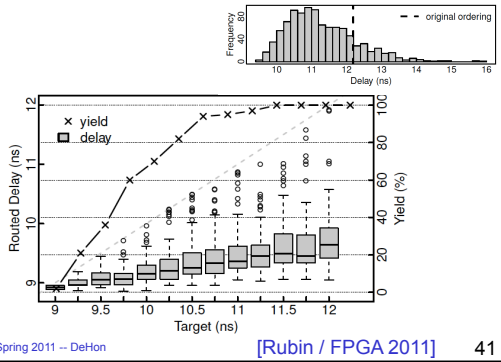
Delay Target

- Cost=
  (1-W(edge))*delay + W(edge) *congest
- W(edge) = Slack(edge)/D_{target}
  - Previously: denominate was D_{max}
- Compute Slack based on D_{target}
  - can be negative
- W(edge)=Max(minW,Slack(edge)/D_{target})
  - minW > 0

Delay Target Routing

- Does allow slack to be used on any of the gate→gate connections on path
  - …but not being that deliberate/efficient about the allocation
- Doesn't require time for slack allocation

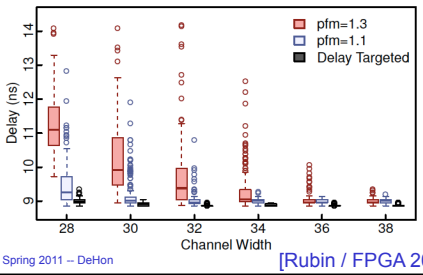
Delay Target Routing



[Rubin / FPGA 2011]

Delay Target Routing

- Less sensitive to initial conditions



[Rubin / FPGA 2011]

Rerouting

Rerouting

- Default: reroute everything
- Can get away rerouting only congested nodes
  - if keep routes in place
  - history force into new tracks
    - causing greedy/uncongested routes to be rerouted

Rerouting

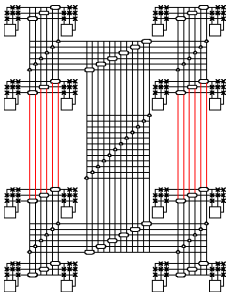
- Effect of only reroute congested?
  - maybe more iterations
    - (not reroute a signal until congested)
  - less time
  - Convergence?
    - Faster? … prevent convergence?
  - ? Hurt quality?
    - (not see strong case for)
  - …but might hurt delay quality
    - Maybe followup rerouting everything once clear up congestion?

Run Time?

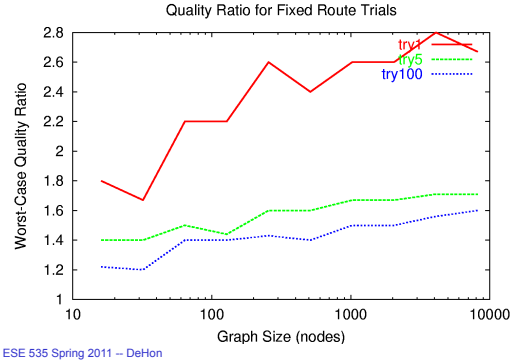
- Route |E| edges
- Each path search O($|E_{graph}|$) worst case
  - …generally less
- Iterations?

Quality and Runtime Experiment

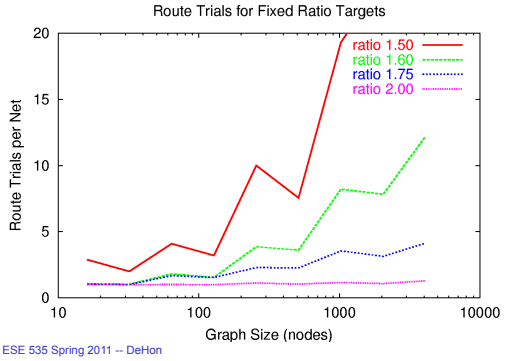
- For Synthetic netlists on HSRA
  - Expect to be worst-case problems
- Congestion only
  - Quality = # channels
- Number of individual route trials limited (measured) as multiple of nets in design
  - (not measuring work per route trial)



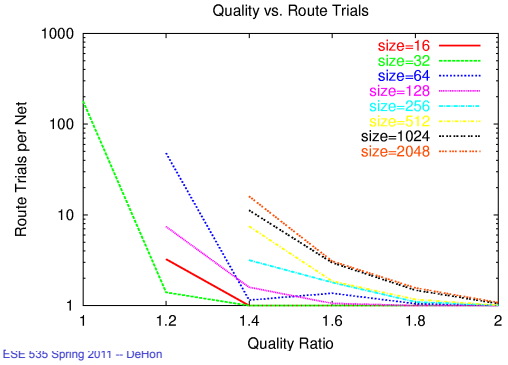
Quality: fixed runtime



Quality Target

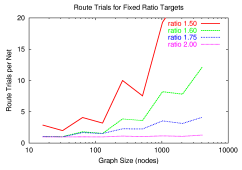


Quality vs. Time



Conclusions?

- Iterations increases with N
- Quality degrade as we scale?



Techniques to Accelerate

(already in use in data just shown)

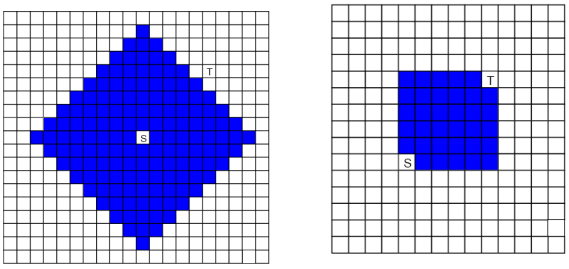
Search Ordering

- Default: breadth first search for shortest
  - O(total-paths)
  - $O(N^p)$ for HSRA
- Alternately: use A*:
  - estimated costs/path length, prune candidates earlier
  - can be more depth first
    - (search promising paths as long as know can't be worse)

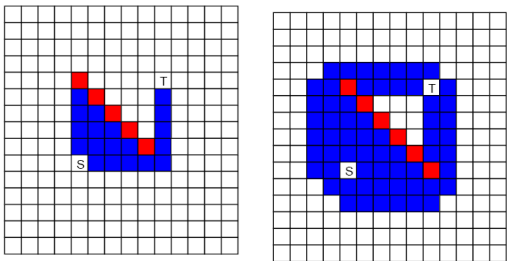
BFS → A*

- Start at source
- Put src node in priority queue with cost 0
  - Priority queue orders by cost
  - Cost = Σ (path so far) + min path to dest
- While (not found sink)
  - Pop least cost node from queue
    - Get: current_node, current_cost
  - Is this sink? → found
  - For each outgoing edge
    - Push destination onto queue
    - with cost current_cost+edge_cost

BFS vs. A*

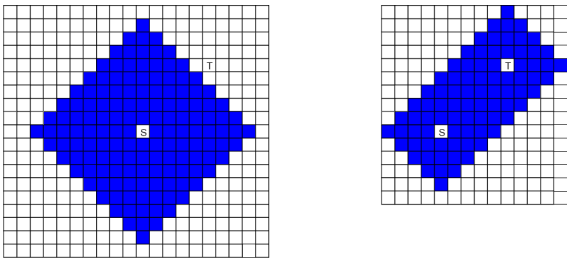


Single-side, Directed (A*)

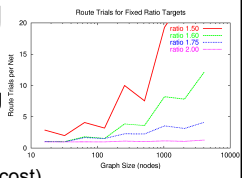


Only expand search windows as prove necessary to have longer route.

Search: one-side vs. two-sides



Searching



- In general:
  - greedy/depth first searching
    - find a path faster
    - may be more expensive
      - (not least delay, congest cost)
  - tradeoff by weighting
    - estimated delay on remaining path vs. cost to this point
    - control greediness of router
  - More greedy is faster at cost of less optimal paths (wider channels)
    - 40% W → 10x time reduction [Tessier/thesis'98]

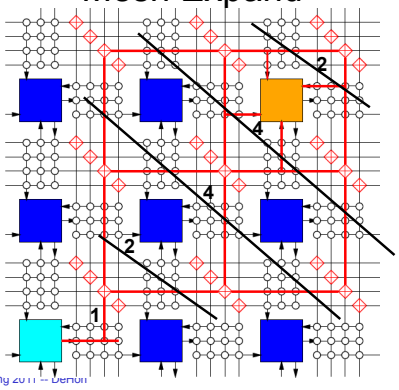
Searching

- Use A* like search
  - Always expanded (deepen) along shortest …as long as can prove no other path will dominate
  - Uncongested: takes O(path-length) time
  - Worst-case reduces to breadth-first
    - O(total-paths)
    - $O(N^p)$ for HSRA

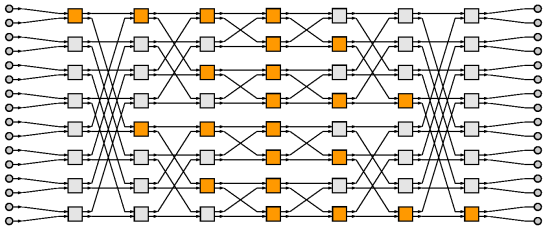
Domain Negotiation

- For Conventional FPGAs (and many networks)
  - path freedom
    - bushy in middle
    - low on endpoints

Mesh Expand



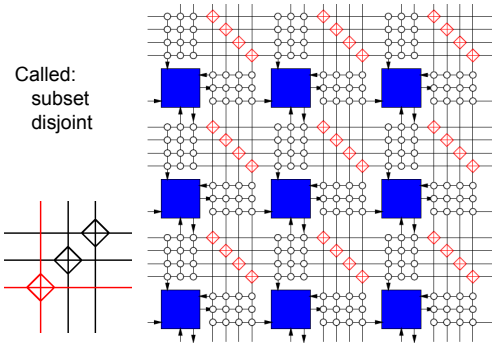
Multistage/Benes



Switches in all paths 0000 to 1111

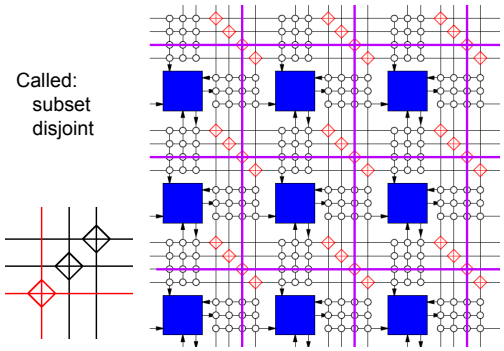
Conventional FPGA Domains

Called:
subset
disjoint



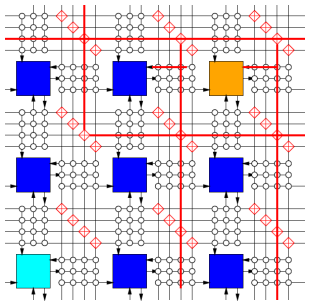
Conventional FPGA Domains

Called:
subset
disjoint



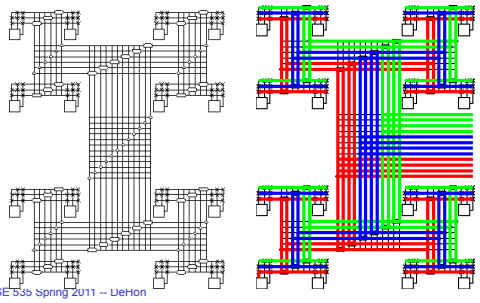
Domain Routing

- No point in searching along an entire path from source
- Just to find it's heavily congested at sink



(SRC)

HSRA Domains



Domain Negotiation

- Path bottlenecks exist at **both** endpoints
- Most critical place for congestion
- Most efficient: work search from both ends
  - more limiting in A* search
  - focus on paths with least (no) congestion on endpoints first
  - FPGAs -- picking "domain" first
  - otherwise paths may look equally good up to end (little pruning)

Summary

- Finding short path easy/well known
- **Complication:** need to route set of signals
  - who gets which path?
  - Arbitrary decisions earlier limit options later
- **Idea:** iterate/relax using congestion history
  - update path costs based on congestion
    - Cost adaptive to route
  - reroute with new costs
- Accommodate delay and congestion

Admin

- No class next Monday 2/28
  - work on assign 4
- Reading for next Wednesday online
- Assignment 4 due Wed. 3/2
- Andre away Monday and Tuesday

Big Ideas

- Exploit freedom
- Technique:
  - Graph algorithms (BFS, DFS)
  - Search techniques: A*
  - Iterative improvement/relaxation
  - Adaptive cost refinement