Today

• Routing
  – Pathfinder
    • graph based
    • global routing
    • simultaneous global/detail

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, Bellman Ford, A*)
**Breadth First Search (BFS)**

- Start at source
- 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? \( \rightarrow \) found
  - For each outgoing edge
    - Push destination onto queue
    - with cost current_cost+edge_cost

**Bellman Ford**

- For \( k \leftarrow 0 \) to \( N \)
  - \( u_i \leftarrow \infty \) (except \( u_i=0 \) for IO)
- For \( k \leftarrow 0 \) to \( N \)
  - for \( e_{ij} \in E \)
    - \( u_i \leftarrow \min(u_i, u_j+w(e_{ij})) \)
- For \( e_{ij} \in E \) //still update \( \rightarrow \)negative cycle
  - if \( u_i > u_j+w(e_{ij}) \)
    - cycles detected

**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
  - 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

Example (first order congestion)

All, individual routes prefer middle; create congestion.
**Example** (first order congestion)

Base costs (delays)

Reroute, avoid congestion.

**Example** (need for history)

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.

Cannot route s₃ → d₃

(cost can imagine longer chain like this)

**Cost Function (2)**

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

**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

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.
Cost Function (Delay)

- Cost = \((1-W(\text{edge})) \cdot \text{delay} + W(\text{edge}) \cdot \text{congest} - \text{congest as before}
  - (\text{base+history}) \cdot f(\#\text{signals}, \text{time})
- W(\text{edge}) = \frac{\text{Slack(\text{edge})}}{D_{\text{max}}}
  - 0 for edge on critical path critical path
  - >0 for paths with slack
- Use W(\text{edge}) to order routes
- Update critical path and W each round

Convergence

- Chan+Schlag [FPGA'2000]
  - cases where doesn’t converge
  - special case of bipartite graphs
    - converge if incremental
    - or if prefer uncongested to least history cost
- theory (continuous)
  - only reroute overflow
  - converge in O(|E|) reroutes
  - But then have fractional routes…

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
  - ? Better 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_{\text{graph}}|) worst case
  - …generally less
  - Iterations?
Quality and Runtime Experiment
- For Synthetic netlists on HSRA
  - Expect to be worst-case problems
- Number of individual route trials limited (measured) as multiple of nets in design
  - (not measuring work per route trial)

Quality Target

Quality vs. Time

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

Search Ordering
- Default: breadth first search for shortest
  - $O(\text{total-paths})$
  - $O(N^p)$ 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 \( \rightarrow \) A**

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

---

**BFS vs. A**

---

**Single-side, Directed (A**

**Search: one-side vs. two-sides**

---

**Search: Oblivious vs. Directed (BFS vs. A**)
Searching

• Use A* like search
  – Always expanded (deepen) along shortest ...
    ...as long as can prove no other path will dominate
  – Uncongested: takes $O(\text{path-length})$ time
  – Worst-case reduces to breadth-first
    • $O(\text{total-paths})$
    • $O(N^2)$ for HSRA

Domain Negotiation

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

Mesh Expand

Multistage/Benes

Conventional FPGA Domains

Conventional FPGA Domains

Switches in all paths 0000 to 1111

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

- Online Course Evaluations
  - http://www.upenn.edu/eval
- Reading: online
- Assignment 5: Due Wednesday
- Assignment 6: now online

Big Ideas

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