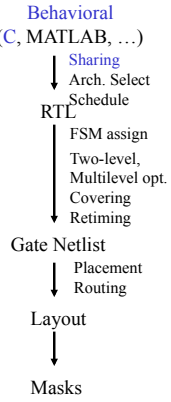
ESE535: Electronic Design Automation

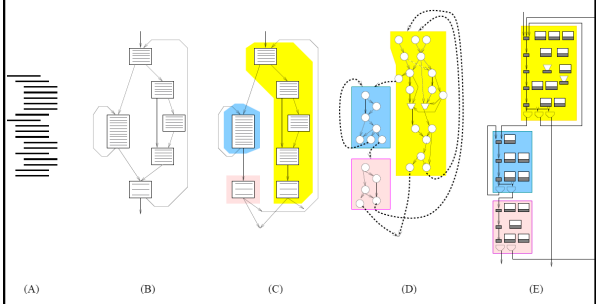
Day 17: March 30, 2015
High Level Synthesis II
Dataflow Graph Sharing

Today

Sharing
- Dataflow subgraph
  - Pattern identification
  - Pattern selection



Flow Review

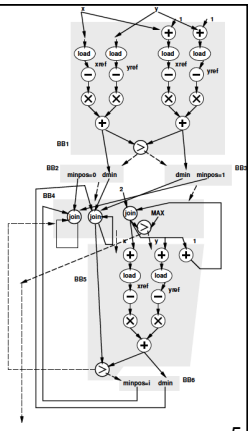


Additional Concerns?

What are we still not satisfied with?
- Parallelism in hyperblock
  - Especially if memory sequentialized
    - Disambiguate memories?
    - Allow multiple memory banks?
- Only one hyperblock active at a time
  - Share hardware between blocks?
- Data only used from one side of mux
  - Share hardware between sides?
- Most logic in hyperblock idle?
  - Couldn't we pipeline execution?

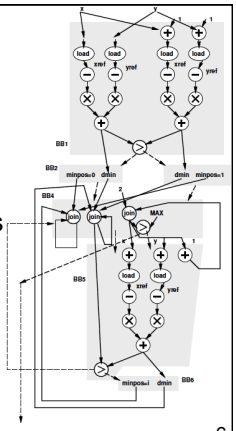
Preclass

- Common subgraphs?
- How would we like to share?
  - If trying to avoid slowdown
  - If willing to make area-time tradeoffs?



Subgraph Sharing

- Can potentially share identical subgraphs
- Can share similar subgraphs

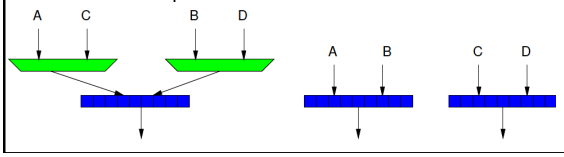


Evaluating Subgraph Sharing

- What do we have to do to share subgraphs?
- When is it worthwhile?
  - How big does graph need to be?
  - How much overhead to share?

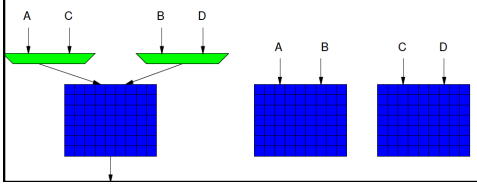
Example

- Muxes on inputs to an adder
  - Probably bigger than just having two adders
  - 2(Amux) + Aadd > 2(Aadd)
  - On FPGA:
    - ~LUT per Adder bit
    - ~LUT per Mux bit



Example

- Muxes on input to mulitipler
  - Probably smaller than two multipliers
  - 2(Amux)+Ampy < 2(Ampy)
  - General
    - Area(Amux) ~ O(N)
    - Area(Ampy) ~ O(N^2)

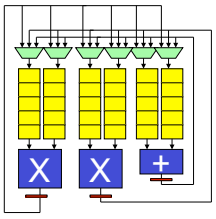


Extreme Case

- If ignored multiplexing overhead, what would we get?
  - What would we select at the resources and how connected?

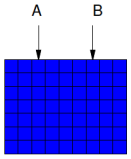
VLIW Extreme

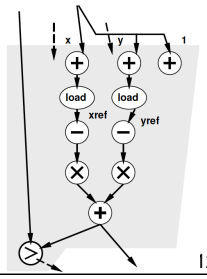
- Sketch
  - Each basic block requires a set of operators to achieve minimum path length
  - Union sets over all basic blocks
    - Keep track of max number of each operator type
  - Build VLIW with that operator set
- Why unsatisfying?



Favorable Subgraphs

- Particularly beneficial when I/O into subgraph small
  - Overhead for muxing proportional to inputs





Approach

- Find candidate, reusable subgraphs→patterns
- Select a cover set of patterns
- Assign original graph to patterns
  - Assess benefits of sharing
- Patch together pattern cover with control and multiplexing

Terms

- Subgraph
  - A piece of original computational graph
- Pattern
  - Common (resuable) subgraph
- Want to find small set of patterns that can efficiently cover the original graph

Approach

- Find **patterns**
- Select a cover set of patterns
- Assign original graph to patterns
  - Assess benefits of sharing
- Patch together pattern cover with control and multiplexing

Find Recurring Patterns

- How might we identify the set of candidate patterns?

Finding Subgraphs

- Keep set of subgraphs of size k
- Create subgraphs of size k+1 from subgraphs of size k
  - By adding a neighboring node
    - Maybe several such expansions for each k-subgraph
- Careful: can end up with exponential subgraphs

Optimization

- Compute candidate graph patterns during subgraph generation
  - Each subgraph may become a candidate
  - Keep track of subgraphs that might match with candidate patterns
  - As add subgraph, compare it with candidate patterns and add to list if “close” enough
  - At end of a given graph size, prune out patterns with too few potential matches

Close enough?

- **Conceptually:** not too expensive to use the candidate pattern
- **Concretely:** compute a distance metric between graph and pattern
  - Minimum cost of edits to morph one graph into another
    - E.g. relabel nodes, remove nodes
  - Want to capture potential cost of adding muxes and control

Algorithm 1 HPR Algorithm

```
1: ℙ          → set of discovered patterns
2: 𝕊_k        → set of size k subgraph
3: INST(P)    → instances of a pattern P
4: l_dist     → edit distance limit
5: l_count    → frequency limit
6:
7: travel all DFGs, add size 1 patterns and instances to ℙ and 𝕊_1
8: for k ← 2, N do
9:   for all s_k ∈ 𝕊_k do
10:    adding a neighbor to expand s_k to s_{k+1}
11:    if s_k is the primary subgraph of s_{k+1} and convex then
12:      calculate CV(s_{k+1})
13:      get list of patterns P_i s.t. ||CV(P_i) − CV(s_{k+1})||_1 ≤ 4 *
         l_dist using LSH
14:      calculate edit distance of s_{k+1} with each P_i
15:      if d(P_i, s_{k+1}) < l_dist then
16:        add s_{k+1} to INST(P_i)
17:      else
18:        create a new pattern based on s_{k+1}, add to ℙ
19:      end if
20:      add s_{k+1} to 𝕊_{k+1}
21:    end if
22:  end for
23:  for all new pattern P_i ∈ ℙ do
24:    if |INST(P_i)| < l_count && size(P_i) ≤ (k + 1 − l_dist) then
25:      remove P_i from ℙ
26:      remove INST(P_i) from 𝕊_{k+1}
27:    end if
28:  end for
29: end for
```


[Cong & Jiang / FPGA 2008]

Potential Optimization

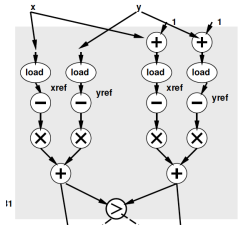
- Canonicalize subgraphs so recognize when encounter same subgraph again
  - Keep set of subgraphs small
- How might we identify/match subgraphs?

Subgraph Canonicalization (similar to Common Subexpression)

- In topological order (inputs to outputs)
- Give name for single operator
- Each node, need name for subgraph rooted at this node
  - Since named/canonicalize all predecessors
    - Looking for name for a pattern with same operator at the output, and the same subgraph on inputs
    - Compare existing patterns end with output operator
      - Hash operator+inputs → only check things that match hash
      - Match → use that name, else allocate name

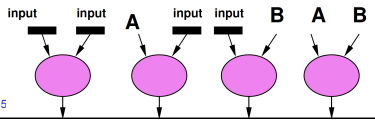
Subgraph Canonicalization

- Problem
  - Matches partial patterns from inputs
  - Not match partial pattern omit part of inputs



Subgraph Canonicalization

- Problem
  - Matches partial patterns from inputs
  - Not match partial pattern omit part of inputs
- Approach
  - Also create/name patterns at each node with a subset of the inputs
  - Means each node has multiple pattern candidates (could explode here)



Cover Subgraphs

- One have candidate patterns, need to cover the original graph.
- What's our goal?
  - (cost function)

Cover Goal

- Minimize area

$$\sum_{P} A(p) + \sum_{BB} Ause(p \in P)$$

- Minimum added latency
  - Delay of BB covered by p in P
- Minimize energy?

Cover Subgraph

- Given a proposed set of pattern graphs, how can we cover?

Cover Subgraph

- How many sets if we explored them all?

Greedy Cover Subgraph

- How might we cover greedily?

Greedy Cover Subgraph

- Select "most beneficial" pattern
- Assign it to the stuff it covers
  - Add logic to share accommodate
  - Remove those as things that need to be covered
- Repeat until all covered or no benefit

Most Beneficial Pattern

- How would we define pattern benefit?

Beneficial Pattern

- N – number of patterns can apply to
- Area: save muxes inside pattern

$$\frac{N*(mux(io)+mux(inside))+area(P)}{N*mux(io)+area(P)}$$

- Latency: prefer parallel (low depth)

$$\frac{|P|}{latency(P)}$$

 [Cong & Jiang / FPGA 2008]

Pattern and Graph Statistics

Size	#Subgraph	#Pattern	#Inst	#Calc
2	62	3	62	0.96
3	108	12	108	1.08
4	195	20	161	1.48
5	366	26	248	1.49
6	701	35	404	1.9
7	1357	58	579	2.6
8	2533	76	714	3.18
9	4517	86	762	3.82
10	7800	94	793	4.43
11	13112	101	668	7.04
12	21365	73	348	7.89
13	33316	32	87	5.03
14	49040	3	6	1.7

#Calc – Average number of edit-distance calculations per subgraph match

 [Cong & Jiang / FPGA 2008]

Energy Impact?

- What are the energy impacts of sharing?

Big Ideas:

- Sharing
- Estimation
- Techniques
  - Graph Matching
  - Covering
  - Greedy

Admin

- Project Formulation Proposal Due Thursday
  - Office Hours or schedule time if want to discuss
- Reading for Wednesday online