11. High-level synthesis steps

Binding and resource sharing
Binding

• Scheduling determines:
  – When ops will execute
  – How many resources are needed

• Binding determines which ops execute on which resources
  – If multiple ops use the same resource
    • Resource Sharing
Basic Idea: given a schedule, map operations onto resources such that operations in same cycle do not use same resource.

2 ALUs (+/-), 2 Multipliers

Cycle1
1 * 2 + 3 +
Cycle2
5 * 6 * 4 -
Cycle3
7 +
Cycle4

Mult1 ALU1 ALU2 Mult2
• Many binding possibilities and many side effects
  – Bad binding may increase resources, require huge steering logic, reduce clock rate, etc.
• Can’t bind same resource to more than op in a cycle
  – 1 resource can’t perform multiple ops simultaneously!

2 ALUs (+/-), 2 Multipliers

Incorrect binding
• How to automate binding?
  – More graph theory

• Binding uses compatibility graphs

• Compatibility graph, definition:
  – Each node is an operation
  – Edges represent compatible operations between a pair nodes
    • Compatible means that two ops can share a resource
      – Compatible ops use same type of resource (ALU, etc.) and are scheduled to different cycles
Compatibility Graph

2 and 3 not compatible (same cycle)

5 and 6 not compatible (same cycle)
Note - Fully connected subgraphs can share one resource (all involved nodes are compatible), e.g. nodes {2,4,7,8} in ALU subgraph can share a single ALU.
Fully connected subgraphs can share a resource; nodes \{1,5\} in Mul subgraph can share a single Mul
Other examples of fully connected subgraphs
Compatibility Graph

- Binding goal: Find minimum number of fully connected subgraphs that cover entire graph
  - Well-known problem in graph theory: Clique partitioning (NP-complete)

- Cliques = \{ \{2,8,7,4\}, \{3\}, \{1,5\}, \{6\} \}
  - ALU1 executes 2,8,7,4
  - ALU2 executes 3
  - MUL1 executes 1,5
  - MUL2 executes 6
Compatibility Graph

- Final Binding:
Compatibility Graph

- Alternative Final Binding:
Translation to data path – Now we begin to see hardware

1) Add resources and registers
2) Add mux for each input
3) Add input to left mux for each left input in DFG
4) Do same for right mux
5) If only 1 input, remove mux
Left Edge Algorithm

- Alternative to clique partitioning
  - Name ‘left edge’ adopted from graph theory,
  - Each node in scheduled DFG has a ‘left edge’ – start point and a ‘right-edge’ end point:

2 ALUs (+/-), 2 Multipliers
Left Edge Algorithm

Let's rotate the scheduled DFG by 90 deg:

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

2 ALUs (+/-), 2 Multipliers

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound
Left Edge Algorithm

1) Initialize right_edge to 0
2) Find a node N whose left edge is >= right_edge
3) Bind N to a particular resource
4) Update right_edge to the right edge of N
5) Repeat from 2) for nodes using the same resource type until right_edge passes all nodes
6) Repeat from 1) until all nodes bound

2 ALUs (+/-), 2 Multipliers

right_edge

Cycle1  Cycle2  Cycle3  Cycle4  Cycle5  Cycle6  Cycle7
1  +  2  +  3  +  4  *  5  *  6  *  7  +  8  
Extensions

• Algorithms presented here will find a valid binding
  – But they do not consider amount of steering logic required
  – Different bindings can require significantly different # of muxes

• Possible solution
  – Extend compatibility graph
    • Use weighted edges/nodes to introduce cost function representing steering logic
    • Perform clique partitioning, finding the set of cliques that minimize weight
Binding Summary

• Binding maps operations onto physical resources
  – Determines sharing among resources

• Binding may greatly affect steering logic

• Binding becomes trivial for fully pipelined circuits
  – 1 resource per operation

• Translation from bound DFG to data path hardware is straightforward
Summary of main HLS steps

• HLS tool front-end (parsing/compiling) converts user code into intermediate representation
  – We considered CDFG
• Scheduling assigns a start time for each operation in DFG
  – CFG node start times are defined by control dependencies
  – Resource allocation determined by schedule
• Binding maps scheduled operations onto physical resources
  – Determines how resources are shared
• Final result:
  – Scheduled/Bound DFG can be translated into a data path
  – CFG can be translated to a controller FSM
  – This way High-Level Synthesis can create a custom circuit for any CDFG.