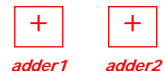
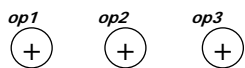


Lecture 14 Resource Sharing & Binding

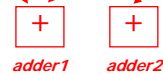
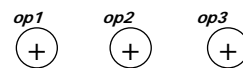
Sharing vs. Binding

- Resource Sharing
 - Assignment of a resource to more than one operation
 - Goal – reduce area by allowing multiple non-concurrent operations to share the same hardware operator
- Resource Binding
 - Explicit mapping between operations and resources



Resource Sharing

- We have 3 add operations and 2 adder units

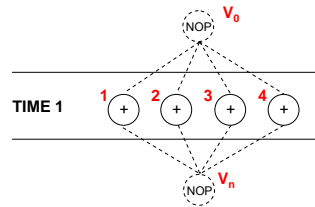


Resource Binding

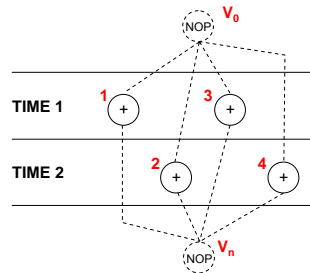
- Add op1 and op2 executes on adder unit 1
- Add op3 executes on adder unit2

Resource Binding

- Resource binding can be applied to scheduled or non-scheduled sequencing graphs
 - Scheduled sequencing graphs provides limitation on possible sharing



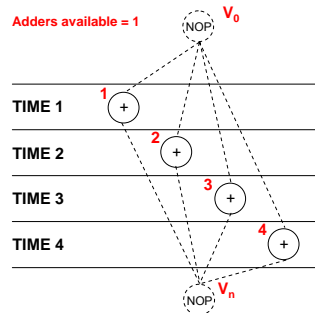
Requires 4 adders to meet the time constraint
(upper bound = 1)



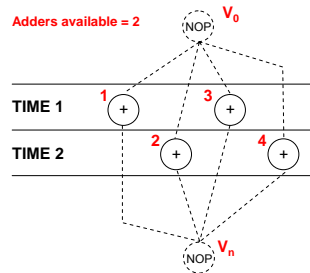
Requires 2 adders to meet the time constraint
(upper bound = 2)

Resource Binding

- Resource binding can be applied to scheduled or non-scheduled sequencing graphs
 - Scheduled sequencing graphs provides limitation on possible sharing
 - Non-scheduled sequencing graphs, the limitation of resource sharing effects the latency by limiting the concurrency of operations (LIST_L scheduling)



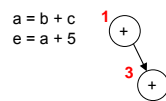
At most 1 add operation can be executed in a
time slice, latency = 4



At most 2 add operations can be executed in a
time slice, latency = 2

Sharing and Binding for Resource Dominated Circuits

- We are interested in the set of vertices of the sequencing graph (omit source/sink nodes)
- How much sharing is possible?
- Two or more operations can be bound to the same resource if they are **compatible**
 - Not concurrent
 - Can be implemented with the same resource type



v1 and v3 are not concurrent

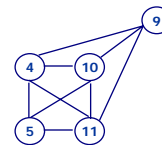
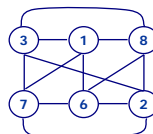
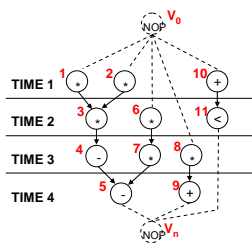


v7 and v10 are not concurrent

- Two operations are NOT concurrent if
- Either one starts after the other has finished
 - Alternative choices (mutually exclusive) of a branching decision

Resource Compatibility Graph

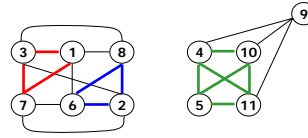
- Graph whose set of vertices is a one-to-one correspondence with operations in the sequencing graph and whose edges denotes the compatible operations pairs



- 3, 1 – Same op, 3 starts after 1
 - 3, 2 – Same op, 3 starts after 2
 - 3, 4 – Different ops
 - 3, 5 – Different ops
 - 3, 6 – Same op, BUT neither starts after the other and not alternative choices of branch1
 - 3, 7 – Same op, 3 starts in Time 2 and 7 starts in Time 3
 - 3, 8 – Same op, 3 starts in Time 2 and 8 starts in Time 3
 - 3, 9 – Different ops
 - 3, 10 – Different ops
 - 3, 11 – Different ops
- Repeat for each node*

Compatibility Graph Shows Resource Sharing

- As many disjoint (no common elements) components as resource types
 - A multiply operations is not compatible with an add operation
- Clique - group of mutually compatible operations correspond to subset of vertices that are mutually connected
 - Each vertices connected to every other vertices
- Maximal set of mutually compatible operations are represented by maximal clique
- The optimum resource sharing is on that minimizes the number of required resource instances
 - Resource instance relates to cliques
 - Partitioning graph into minimum number of cliques yields optimal sharing



Maximize size of cliques, must ensure all vertices included

{1, 3, 7}
 {2, 6, 8}
 {1, 8}
 {4, 5, 10, 11}
 {9}

Resources = # cliques

We need 2 adders, 2 multipliers

Clique Partitioning

```

CLIQUE_PARTITION( G(v, e) ){
    Π = Φ // initial set of partitions to empty
    while( G(v,e) not empty ) do{ // while the graph is not empty, keep iterating
        C = MAX_CLIQU( G(v,e) ) // compute a maximal clique in graph
        Π = Π U C // add max clique to set of partitions
        delete C from G(v,e) // remove max clique from graph
    }
}

MAX_CLIQU( G(v, e) ){
    C = vertex with largest degree
    repeat {
        repeat {
            U = { v ∈ V : v ∉ C and adjacent to all vertices of C }
            if ( U ≠ Φ ){ // no such vertices exist
                return C
            }
        } else{
            select v ∈ U // pick one
            C = C U v // add to clique
        }
    }
}
    
```

Clique Partitioning

Example 1

$\Pi = \Phi$ // set of partitions is initially empty

Is G empty? No.

Find max clique

$C = 1$ // vertex with largest degree, anything with 4 will do

$U = \{3, 7, 6, 8\}$ // these vertices are connected to 1

$V = 3$

$C = \{1\} \cup \{3\} = \{1, 3\}$

$U = \{7, 8\}$ // these vertices are connected to 1 and 3

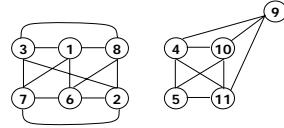
$C = \{1, 3\} \cup \{7\} = \{1, 3, 7\}$

$U = \{\Phi\}$ // no others vertices connect to 1, 3, and 7

Return $\{1, 3, 7\}$

$\Pi = \{1, 3, 7\}$

Remove $\{1, 3, 7\}$ from G



Vertices	Degree
1	4
2	4
3	4
4	4
5	3
6	4
7	4
8	4
9	3
10	4
11	4

Clique Partitioning

Example 1

$\Pi = \{1, 3, 7\}$

Is G empty? No.

Find max clique

$C = 4$ // vertex with largest degree, anything with 4 will do

$U = \{5, 9, 10, 11\}$ // these vertices are connected to 4

$V = 5$

$C = \{4\} \cup \{5\} = \{4, 5\}$

$U = \{10, 11\}$ // these vertices are connected to 4 and 5

$C = \{4, 5\} \cup \{10\} = \{4, 5, 10\}$

$U = \{11\}$ // these vertices are connected to 4 and 5

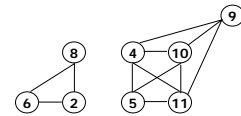
$C = \{4, 5, 10\} \cup \{11\} = \{4, 5, 10, 11\}$

$U = \{\Phi\}$ // no others vertices connect to 4, 5, 10, and 11

Return $\{4, 5, 10, 11\}$

$\Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}$

Remove $\{4, 5, 10, 11\}$ from G



Vertices	Degree
2	2
4	4
5	3
6	2
8	2
9	3
10	4
11	4

Clique Partitioning

Example 1

$\Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}$

9

Is G empty? No.

Find max clique

$C = 2$ // vertex with largest degree, anything with 2 will do

$U = \{6, 8\}$ // these vertices are connected to 4

$V = 6$

$C = \{2\} \cup \{6\} = \{2, 6\}$

$U = \{8\}$ // these vertices are connected to 1 and 3

$C = \{2, 6\} \cup \{8\} = \{2, 6, 8\}$

$U = \{\Phi\}$ // no others vertices connect to 1, 3, and 7

Return $\{2, 6, 8\}$

$\Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}$

Remove $\{2, 6, 8\}$ from G



Vertices	Degree
2	2
6	2
8	2
9	0

Clique Partitioning

Example 1

$\Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}$

9

Is G empty? No.

Find max clique

$C = 9$

$U = \{9\}$

$U = \{\Phi\}$ // no others vertices connect to 1, 3, and 7

Return $\{9\}$

$\Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}, \{9\}$

Remove $\{9\}$ from G

Vertices	Degree
9	0

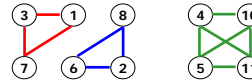
Clique Partitioning

Example 1

$\Pi = \{1, 3, 7\}, \{4, 5, 10, 11\}, \{2, 6, 8\}, \{9\}$

Is G empty? Yes!

- What does clique partition tell us?
 - $\{1, 3, 7\}$ – multiplier
 - $\{4, 5, 10, 11\}$ – alu
 - $\{2, 6, 8\}$ – multiplier
 - $\{9\}$ – alu



- We know how much sharing AND binding

Clique Partitioning

Example 2

$\Pi = \Phi$

Is G empty? No.

Find max clique

$C = 4$

$U = \{1, 2, 3, 5, 6\}$

$V = 1$

$C = \{4\} \cup \{1\} = \{1, 4\}$

$U = \{5, 6\}$

$C = \{1, 4\} \cup \{5\} = \{1, 4, 5\}$

$U = \{6\}$

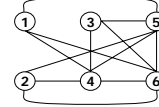
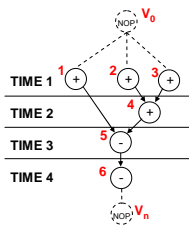
$C = \{1, 4, 5\} \cup \{6\} = \{1, 4, 5, 6\}$

$U = \{ \Phi \}$

Return $\{1, 4, 5, 6\}$

$\Pi = \{1, 4, 5, 6\}$

Remove $\{1, 4, 5, 6\}$ from G



Vertices	Degree
1	3
2	3
3	3
4	5
5	5
6	5

Clique Partitioning

Example 2

$\Pi = \{1, 4, 5, 6\}$

Is G empty? No.

Find max clique

$C = 2$

$U = \{\Phi\}$

Return {2}

$\Pi = \{1, 4, 5, 6\}, \{2\}$

Remove {2} from G

③

②

Vertices	Degree
2	0
3	0

$\Pi = \{1, 4, 5, 6\}, \{2\}$

Is G empty? No.

Find max clique

$C = 3$

$U = \{\Phi\}$

Return {3}

$\Pi = \{1, 4, 5, 6\}, \{2\}, \{3\}$

Remove {3} from G

③

Vertices	Degree
3	0

ECE 474a/575a
Susan Lysecky

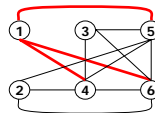
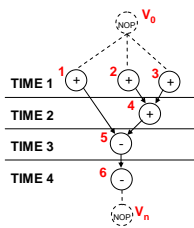
15 of 25

Clique Partitioning

Example 2

$\Pi = \{1, 4, 5, 6\}, \{2\}, \{3\}$

Is G empty? Yes



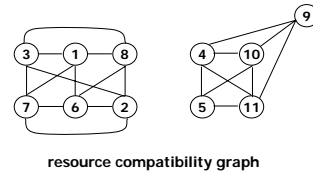
- Need 3 ALUs
 - ALU 1 executes ops 1, 4, 5, 6
 - ALU 2 executes op 2
 - ALU 3 executes op 3

ECE 474a/575a
Susan Lysecky

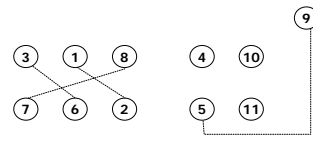
16 of 25

Resource Conflict Graph

- Instead of compatibility we can instead look at conflicts
 - May simplify the graph
- Resource conflict graph
 - Graph whose set of vertices is a one-to-one correspondence with operations in the sequencing graph and whose edges denotes the **conflicting** operations pairs
 - To simplify graph, we consider conflicts between each resource type independently



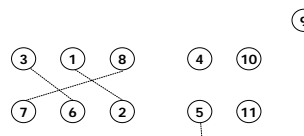
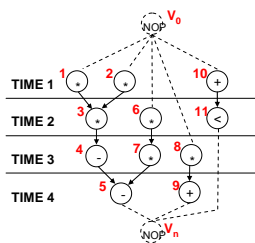
resource compatibility graph



resource conflict graph

Building Resource Conflict Graph

- To simplify graph, we consider conflicts between each resource type independently



Multipliers

ALUs

Consider Multipliers (1, 2, 3, 6, 7, 8)

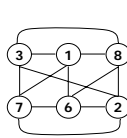
- 1, 2 – concurrent
- 3, 6 – concurrent
- 7, 8 – concurrent

Consider ALUs (4, 5, 9, 10, 11)

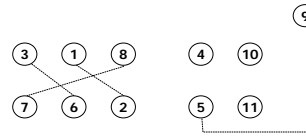
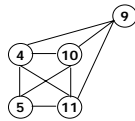
- 5, 9 – concurrent

Building Resource Conflict Graph

- Conflict graph is the complement of the compatibility graph
- In conflict graph, looking for set of mutually compatible operations
 - Subset of vertices that are NOT connected by edges
 - Also called independent set of G



resource compatibility graph



resource conflict graph

Graph Coloring

```
VERTEX_COLOR ( G(v, e) ){  
  for( i=1 to |V| ){  
    C = 1 // use number to represent color  
    while( there exists a vertex adjacent to vi with color c) do{  
      C = C + 1  
    }  
    label vi with C  
  }  
}
```

- Use graph coloring to find independent sets
 - Each color represents a resource instance (two adders will be represented by two different colors)
- Optimal resource sharing corresponds to vertex coloring with minimal amount of colors

Graph Coloring

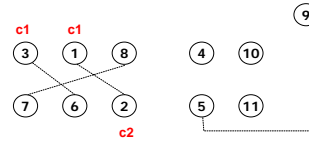
Example 1

$i = 1$ // look at vertex 1

$C = c1$ // represents first color

Is there any adjacent vertices with color = 1? No.

$v_1 = c1$



$i = 2$ // look at vertex 2

$C = c1$

Is there any adjacent vertices with color = 1? Yes.

$C = c2$

Is there any adjacent vertices with color = 2? No.

$v_2 = c2$

$i = 3$ // look at vertex 3

$C = c1$ // represents first color

Is there any adjacent vertices with color = 1? No.

$v_3 = c1$

ECE 474a/575a
Susan Lysecky

21 of 25

Graph Coloring

Example 1

$i = 4$ // look at vertex 4

$C = c1$ // represents first color

Is there any adjacent vertices with color = 1? Yes – remember conflict is implied across different resource types.

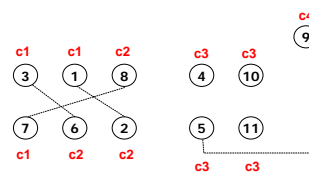
$C = c2$

Is there any adjacent vertices with color = 2? Yes.

$C = c3$

Is there any adjacent vertices with color = 3? No.

$v_4 = c3$



Similarly repeat for remaining

▪ Four colors required – need four resources

- $c1$ is used for multiply
- $c2$ is used for multiply
- $c3$ is used for alu
- $c4$ is used for alu

ECE 474a/575a
Susan Lysecky

22 of 25

Graph Coloring

Example 2

i = 1

C = c1
 Adjacent vertices with color = 1? No.
 $v_1 = c1$

i = 2

C = c1
 Adjacent vertices with color = 1? Yes.
 C = c2
 Adjacent vertices with color = 2? No.
 $v_2 = c2$

i = 3

C = c1
 Adjacent vertices with color = 1? No.
 $v_3 = c1$

i = 4

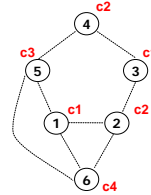
C = c1
 Adjacent vertices with color = 1? Yes.
 C = c2
 Adjacent vertices with color = 2? No.
 $v_4 = c2$

i = 5

C = c1
 Adjacent vertices with color = 1? Yes.
 C = c2
 Adjacent vertices with color = 2? Yes.
 C = c3
 Adjacent vertices with color = 3? No.
 $v_5 = c3$

i = 6

C = c1
 Adjacent vertices with color = 1? Yes.
 C = c2
 Adjacent vertices with color = 2? Yes.
 C = c3
 Adjacent vertices with color = 3? Yes.
 C = c4
 Adjacent vertices with color = 4? No.
 $v_6 = c4$

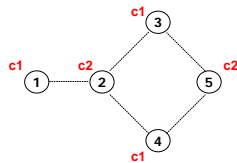


Four colors required – need four resources

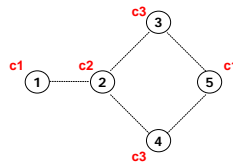
- c1 for node 1, 3 op
- c2 for node 2, 4 op
- c3 for node 5 op
- c4 for node 6 op

Graph Coloring

- VERTEX_COLOR algorithm sensitive to ordering of vertices explored - variety of modifications available
 - Switching pair assignment of colors
 - Backtracking to switching larger number of vertices



- Node ordering 1, 2, 3, 4, 5
- Requires 2 colors



- Node ordering 1, 5, 2, 3, 4
- Requires 3 colors

Conclusion

- Considered several types ways to find resource sharing and binding
 - Compatibility Graph / Max Clique
 - Conflict Graph / Vertex color
- Again, many other methods available
 - Golumbic's algorithm
 - Left-edge algorithm
 - ILP formulation
- Idea of sharing and binding not limited to adders and multipliers
 - Registers
 - Determining minimal number of memory ports
 - Bus sharing