Datapath Synthesis Using Clique Partitioning

A graph is said to be *complete* if and only if for every pair of its vertices there is an edge linking them

- A complete subgraph of graph *G* is called a *clique*
- The *clique partitioning problem* is the problem of partitioning a graph into a minimal number of cliques such that each node belongs to exactly one clique
  - This is a classic NP-complete problem, for which heuristics are usually used to solve

Using clique partitioning for datapath synthesis:

- Build a graph, with vertices representing operations, values, etc.
- Connect two vertices with an edge if they can share a functional unit, register, etc.
- Find the minimum number of cliques that cover the graph = finding the minimum number of functional units, registers, etc.
  - This is an NP-complete problem, so heuristics must be used to prioritize the grouping

CMU's Facet / Emerald (Facet is the methodology, which was implemented as the Emerald system):

- First, uses clique partitioning to bind registers
- Then, uses clique partitioning to bind functional units
- Finally, uses clique partitioning to bind interconnection

A Clique Partitioning Algorithm — Facet / Emerald (Tseng)

Initially, each vertex $v_i \in V$ of *G* is placed in a separate super-node $s_i \in S$ of *G'*

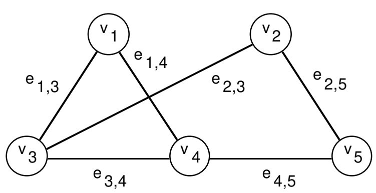
At each step, the algorithm finds the super-nodes s_{index1} and s_{index2} in *S* such that they are connected by an edge and have the maximum number of common neighbors

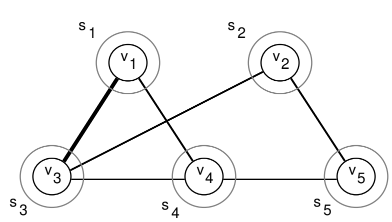
- These two super-nodes are merged into a single super-node $s_{index1index2}$, which contains all the vertices of s_{index1} and s_{index2}
- The set *common_set* contains all the common neighbors of s_{index1} and s_{index2}
- All edges originating from s_{index1} and s_{index2} in *G'* are deleted, and new edges are added from $s_{index1index2}$ to all the super-nodes in *common_set*

This process repeats until there are no edges left in the graph *G'*

When this process is finished, the vertices contained in each super-node $s_i \in S$ form a clique of the graph *G*

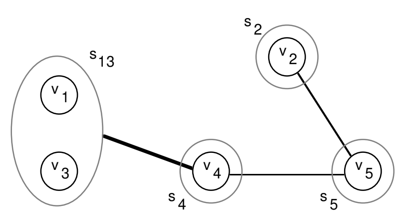
A Clique Partitioning Algorithm — Facet / Emerald (cont.)



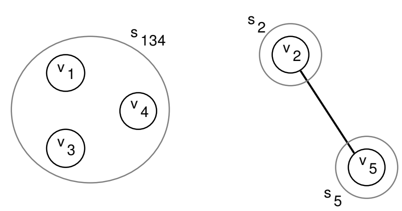


Edge	Common Neighbors
$e_{1,3}$	1
$e_{1,4}$	1
$e_{2,3}$	0
$e_{2,5}$	0
$e_{3,4}$	1
$e_{4,5}$	0

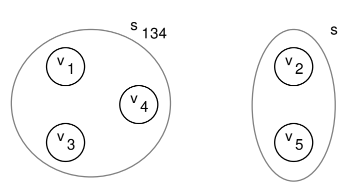
A Clique Partitioning Algorithm — Facet / Emerald (cont.)



Edge	Common Neighbors
$e_{13,4}$	0
$e_{2,5}$	0
$e_{4,5}$	0



Edge	Common Neighbors
$e_{2,5}$	0



Cliques
$s_{134} = \{v_1, v_3, v_4\}$
$s_{25} = \{v_2, v_5\}$

A Clique Partitioning Algorithm — Facet / Emerald (cont.)

while $E' \neq \emptyset$ **do**

/* find s_{index1}, s_{index2} having most common neighbors */
mostcommons = –1;
for each $e'_{ij} \in E'$ **do**
c_{ij} = | common_neighbor(G', s_i , s_j) |;
if c_{ij} > *mostcommons* **then**
mostcommons = c_{ij} ;
index1 = *i* ; *index2* = *j* ;
endif
endfor

common_set = common_neighbor(G', s_{index1} , s_{index2});

/* delete all edges linking s_{index1} or s_{index2} */
E' = delete_edge(E', s_{index1});
E' = delete_edge(E', s_{index2});

/* merge s_{index1} and s_{index2} into $s_{index1index2}$ */
$s_{index1index2} = s_{index1} \cup s_{index2}$;
$S = S - s_{index1} - s_{index2}$;
$S = S \cup \{ s_{index1index2} \}$;

/* add edge from $s_{index1index2}$ to super-nodes in common_set */
for each $s_i \in$ *common_set* **do**
$E' = E' \cup \{ e'_{e,\, index1index2} \}$;
endfor
endwhile

Using Clique Partitioning for Register Assignment — Facet / Emerald

Construct a *compatability graph* as follows:

Each vertex represents a variable

An edge connects two vertices if and only if the two variables can be stored in the same register

That is,

Their lifetime intervals do not overlap, or

They are used as both source and destination in the same statement

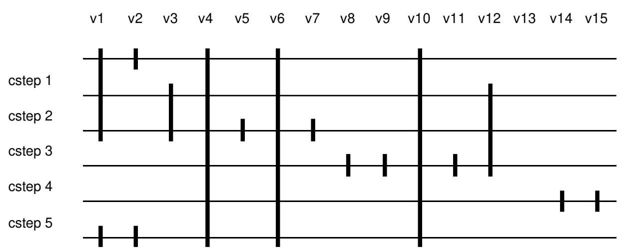
A clique partitioning of this graph yields the minimum number of registers needed to store the values

Given the following code sequence (assumed to be inside a loop):

v3 = v1 + v2;	v12 = v1	
v5 = v3 – v4;	v7 = v3 * v6;	v13 = v3;
v8 = v3 + v5;	v9 = v1 + v7;	v11 = v10 / v5;
v14 = v11 and v8;	v15 = v12 or v9;	
v1 = v14;	v2 = v15;	

Register Assignment Example

We have the following lifetime intervals:



Which gives us the following compatibile variable pairs:

(1,9)	(1,13)	(1,14)	(2,3)	(2,5)	(2,7)
(2,8)	(2,9)	(2,11)	(2,13)	(2,15)	(3,8)
(3,13)	(3,14)	(3,15)	4,13)	(5,8)	(5,11)
(5,13)	(5,14)	(5,15)	(6,13)	(7,9)	(7,13)
(7,14)	(7,15)	(8,13)	(8,14)	(9,13)	(9,15)
(10,13)	(11,13)	(11,14)	(12,13)	(12,15)	(13,14)
(13,15)					

Eventually, we get the following cliques:

{1, 14} {2, 7, 9, 15} {3, 8, 13}

{4} {5, 11} {6} {10} {12}

Using Clique Partitioning for Functional Unit Assignment — Facet / Emerald

Construct a *compatability graph* as follows:

Each vertex represents an operation

An edge connects two vertices if and only if (1) the two operations are in different control steps, and (2) there exists a functional unit that is capable of carrying out both operations

A clique partitioning of this graph yields the minimum number of functional units needed to perform the operations

Prioritize the collection of edges according to these 8 cases:

G8: the operations and the three pairs of variables are the same

G7: the operations are different but the three pairs of variables are the same

G6: the operations and two pairs of variables are the same, but the third pair is different

G5: two pairs of variables are the same, but the operations and one pair of variables are different

G4: the operations and one pair of variables are the same, but the other two pairs of variables are different

G3: one pair of variables is the same, but the operations and the other two pairs of variables are different

G2: the operations are the same, but all three pairs of variables are different

G1: the operations and all three pairs of variables are different

Functional Unit Assignment Example

Assume we have an infinite number of ALUs available, where each ALU can perform: + – * / and or

Given the following code sequence:

$v3 = v1 +_1 v2;$	$v12 = v1$	
$v5 = v3 -_1 v4;$	$v2 = v3 *_1 v6;$	
$v3 = v3 +_2 v5;$	$v2 = v1 +_3 v2;$	$v5 = v10 /_1 v5;$
$v1 = v5 \text{ and}_1 v3;$	$v2 = v12 \text{ or}_1 v2;$	

Assign the following indices to each opertion:

1	2	3	4	5	6	7	8
$+_1$	$-_1$	$*_1$	$+_2$	$+_3$	$/_1$	and_1	or_1

The edges of the compatability graph:

$(1,2)^1$	$(1,3)^1$	$(1,4)^4$	$(1,5)^6$	$(1,6)^1$	$(1,7)^1$	$(1,8)^3$
		$(2,4)^3$	$(2,5)^1$	$(2,6)^1$	$(2,7)^3$	$(2,8)^1$
		$(3,4)^3$	$(3,5)^3$	$(3,6)^1$	$(3,7)^3$	$(3,8)^3$
					$(4,7)^3$	$(4,8)^1$
					$(5,7)^1$	$(5,8)^5$
					$(6,7)^3$	$(6,8)^1$

Functional Unit Assignment Example (cont.)

Considering the single C6, nodes 1 and 5 are combined, two categories are updated (for edges (15,3) and (15,8)) resulting in:

$(15,2)^1$	$(15,3)^3$	$(15,4)^4$	$(15,5)^6$	$(15,6)^1$	$(15,7)^1$	$(15,8)^5$
		$(2,4)^3$	$(2,15)^1$	$(2,6)^1$	$(2,7)^3$	$(2,8)^1$
		$(3,4)^3$	$(3,15)^3$	$(3,6)^1$	$(3,7)^3$	$(3,8)^3$
					$(4,7)^3$	$(4,8)^1$
					$(15,7)^1$	$(15,8)^5$
					$(6,7)^3$	$(6,8)^1$

Eventually, we get the following cliques:

{1, 3, 5, 8} {2, 4, 7} {6}