Resource sharing

Giovanni De Micheli
Integrated Systems Centre
EPF Lausanne

This presentation can be used for non-commercial purposes as long as this note and the copyright footers are not removed

© Giovanni De Micheli – All rights reserved
Module 1

Objectives

- Motivation and problem formulation
- Flat and hierarchical graphs
- Functional and memory resources
- Extension to module selection
Allocation and binding

◆ Allocation:
  ▲ Number of resources available

◆ Binding:
  ▲ Relation between operations and resources

◆ Sharing:
  ▲ Many-to-one relation

◆ Optimum binding/sharing:
  ▲ Minimize the resource usage
Bounding

**Limiting cases:**

▲ **Dedicated resources**
  ▼ One resource per operation
  ▼ No sharing

▲ **One multi-task resource**
  ▼ ALU

▲ **One resource per type**
Optimum sharing problem

◆ Scheduled sequencing graphs
  ▲ Operation concurrency well defined

◆ Consider operation types independently
  ▲ Problem decomposition
  ▲ Perform analysis for each resource type
Compatibly and conflicts

◆ Operation compatibility:
  ▲ Same type
  ▲ Non concurrent

◆ Compatibility graph:
  ▲ Vertices: operations
  ▲ Edges: compatibility relation

◆ Conflict graph:
  ▲ Complement of compatibility graph

<table>
<thead>
<tr>
<th></th>
<th>x=a+b</th>
<th>y=c+d</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>s=x+y</th>
<th>t=x-y</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>t1</td>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td>t2</td>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td>t3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th></th>
<th>t1</th>
<th>t2</th>
<th>t3</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x=a+b</td>
<td>s=x+y</td>
<td>z=a+t</td>
</tr>
<tr>
<td></td>
<td>y=c+d</td>
<td>t=x-y</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Conflicts:

- 1 and 2
- 3 and 4
- 5

Compatibilities:

- 1 and 2
- 3 and 4
- 5

Coloring:

ALU1: 1, 3, 5
ALU2: 2, 4

Partitioning:
Compatibility and conflicts

◆ Compatibility graph:
  ▲ Partition the graph into a minimum number of cliques
  ▲ Find \textit{clique cover number} \( k (G_+) \)

◆ Conflict graph:
  ▲ Color the vertices by a minimum number of colors.
  ▲ Find the \textit{chromatic number} \( x (G_-) \)

◆ NP-complete problems:
  ▲ Heuristic algorithms
The compatibility/conflict graphs have special properties:

- Compatibility
  - Comparability graph
- Conflict
  - Interval graph

Polynomial time solutions:

- Golumbic’s algorithm
- Left-edge algorithm
Perfect graphs

◆ **Comparability graph:**

△ Graph $G (V, E)$ has an orientation $G (V, F)$ with the transitive property

$$(v_i, v_j) \in F \quad \text{and} \quad (v_j, v_k) \in F \rightarrow (v_i, v_k) \in F$$

◆ **Interval graph:**

△ Vertices correspond to *intervals*

△ Edges correspond to interval intersection

△ Subset of *chordal* graphs

▽ Every loop with more than three edged has a chord
Example
Example
Left-edge algorithm

Input:

- Set of intervals with *left* and *right edge*
- A set of *colors* (initially one color)

Rationale:

- Sort intervals in a *list* by *left edge*
- Assign non overlapping intervals to first color using the list
- When possible intervals are exhausted, increase color counter and repeat
Example

Conflict graph

Colored conflict graph

Intervals

Coloring
Left-edge algorithm

```plaintext
LEFT_EDGE(I) {
    Sort elements of I in a list L in ascending order of \( l_i \);
    \( c = 0; \)
    while (some interval has not been colored) do {
        \( S = \emptyset; \)
        \( r = 0; \)
        while (exists \( s \in L \) such that \( l_s > r \)) do {
            \( s = \) First element in the list \( L \) with \( l_s > r; \)
            \( S = S \cup \{s\}; \)
            \( r = r_s; \)
            Delete \( s \) from \( L; \)
        }
        \( c = c + 1; \)
        Label elements of \( S \) with color \( c; \)
    }
}
```
Hierarchical sequencing graphs

- Hierarchical conflict/compatibility graphs:
  - Easy to compute
  - Prevent sharing across hierarchy

- Flatten hierarchy:
  - Bigger graphs
  - Destroy nice properties
Example

(a)

(b)

(c)
Example
Register binding problem

☀ Given a schedule:
  ▲ Lifetime intervals for variables
  ▲ Lifetime overlaps

☀ Conflict graph (interval graph):
  ▲ Vertices ↔ variables
  ▲ Edges ↔ overlaps
  ▲ Interval graph

☀ Compatibility graph (comparability graph):
  ▲ Complement of conflict graph
Register sharing in data-flow graphs

◆ Given:
  ▲ Variable lifetime conflict graph

◆ Find:
  ▲ Minimum number of registers storing all the variables

◆ Key point:
  ▲ Interval graph
    ▼ Left-edge algorithm (polynomial-time complexity)
Example

TIME 1
- 1
  z1
  z2
- 2

TIME 2
- 3
  z3
  z4
- 4
- 5

TIME 3
- 6

TIME 4
- 7
  z5
  z6

(a)

(b)

(c)
Register sharing
general case

◆ Iterative conflicts:
  ▲ Preserve values across iterations
  ▲ Circular-arc conflict graph
    ▼ Coloring is intractable

◆ Hierarchical graphs:
  ▲ General conflict graphs
    ▼ Coloring is intractable

◆ Heuristic algorithms
Example

(c) Giovanni De Micheli
Example

Variable-lifetimes and circular-arc conflict graph
Bus sharing and binding

- Find the *minimum number of busses* to accommodate all data transfer
- Find the *maximum number of data transfers* for a fixed number of busses
- Similar to memory binding problem
Example

◆ One bus:
  ▲ 3 variables can be transferred

◆ Two busses:
  ▲ All variables can be transferred
Module selection problem

◆ Extension of resource sharing
  ▲ Library of resources:
  ▲ More than one resource per type

◆ Example:
  ▲ Ripple-carry adder
  ▲ Carry look-ahead adder

◆ Resource modeling:
  ▲ Resource subtypes with
    ▼ (area, delay) parameters
Module selection solution

◆ ILP formulation:
  ▲ Decision variables
    ▼ Select resource sub-type
    ▼ Determine (area, delay)

◆ Heuristic algorithm
  ▲ Determine minimum latency with fastest resource subtypes
  ▲ Recover area by using slower resources on non-critical paths
Example

- Multipliers with:
  - ▲ (Area, delay) = (5,1) and (2,2)
- Latency bound of 5

Slow multipliers save area
Example 2

- **Latency bound of 4**
  - Fast multipliers for \( \{ v_1, v_2, v_3 \} \)
  - Slower multiplier can be used elsewhere
    - Less sharing
- **Minimum-latency design uses fast multipliers only**
  - Impossible to use slow multipliers

(c) Giovanni De Micheli
Module 2

◆ Objectives
  ▲ Data path generation
  ▲ Control synthesis
Data path synthesis

◆ Applied after resource binding

◆ Connectivity synthesis:
  ▲ Connection of resources to *multiplexers busses* and *registers*
  ▲ Control unit interface
  ▲ I/O ports

◆ Physical data path synthesis
  ▲ Specific techniques for regular datapath design
    ▼ Regularity extraction
Example

REGISTERS

DATA-PATH

CONTROL-UNIT

(c) Giovanni De Micheli
Control synthesis

- Synthesis of the control unit

- Logic model:
  - Synchronous FSM

- Physical implementation:
  - Hard-wired or distributed FSM
  - Microcode
Example
Resource sharing is reducible to vertex coloring or to clique covering:

- Simple for flat graphs
- Intractable, but still easy in practice, for other graphs
- Resource sharing has several extensions:
  - Module selection

Data path design and control synthesis are conceptually simple but still important steps:

- Generated data path is an interconnection of blocks
- Control is one or more finite-state machines