ARCHITECTURAL-LEVEL SYNTHESIS

©Giovanni De Micheli

Stanford University
Outline

- Motivation.

- Compiling language models into abstract models.

- Behavioral-level optimization and program-level transformations.

- Architectural synthesis: an overview.
Synthesis

- Transform behavioral into structural view.

- Architectural-level synthesis:
  - Architectural abstraction level.
  - Determine *macroscopic* structure.
  - Example: major building blocks.

- Logic-level synthesis:
  - Logic abstraction level.
  - Determine *microscopic* structure.
  - Example: logic gate interconnection.
Synthesis and optimization

**Language Models**

**Abstract Models**

**Behavioral View**

**Structural View**

**Architectural Level**

**Logic Level**

- **HDL**
  - Compilation
  - Translation

- **Operations and dependencies**
  - (Data-flow & sequencing graphs)
  - Architectural synthesis & optimization

- **FSMs – Logic functions**
  - (State-diagrams & logic networks)
  - Logic synthesis & optimization

- **Interconnected logic blocks**
  - (Logic networks)
Example

diffeq {
    read \((x, y, u, dx, a)\);
    repeat {
        \(xl = x + dx\);
        \(ul = u - (3 \cdot x \cdot u \cdot dx) - (3 \cdot y \cdot dx)\);
        \(yl = y + u \cdot dx\);
        \(c = x < a\);
        \(x = xl\); \(u = ul\); \(y = yl\);
    }
    until (c);
    write \((y)\);
}
Example of structures

© GDM
Example
Architectural-level synthesis
motivation

- Raise input abstraction level.
  - Reduce specification of details.
  - Extend designer base.
  - Self-documenting design specifications.
  - Ease modifications and extensions.

- Reduce design time.

- Explore and optimize macroscopic structure:
  - Series/parallel execution of operations.
Architectural-level synthesis

- Translate HDL models into sequencing graphs.

- Behavioral-level optimization:
  - Optimize abstract models independently from the implementation parameters.

- Architectural synthesis and optimization:
  - Create macroscopic structure:
    * data-path and control-unit.
  - Consider area and delay information of the implementation.
Compilation and behavioral optimization

- **Software compilation:**
  - Compile program into intermediate form.
  - Optimize intermediate form.
  - Generate target code for an architecture.

- **Hardware compilation:**
  - Compile HDL model into sequencing graph.
  - Optimize sequencing graph.
  - Generate gate-level interconnection for a cell library.
Hardware and software compilation.
Compilation

• Front-end:
  – Lexical and syntax analysis.
  – Parse-tree generation.
  – Macro-expansion.
  – Expansion of meta-variables.

• Semantic analysis:
  – Data-flow and control-flow analysis.
  – Type checking.
  – Resolve arithmetic and relational operators.
Parse tree example

\[ a = p + q \times r \]
Behavioral-level optimization

- Semantic-preserving transformations aiming at simplifying the model.

- Applied to parse-trees or during their generation.

- Taxonomy:
  - *Data-flow* based transformations.
  - *Control-flow* based transformations.
Data-flow based transformations

- Tree-height reduction.
- Constant and variable propagation.
- Common subexpression elimination.
- Dead-code elimination.
- Operator-strength reduction.
- Code motion.
Tree-height reduction

• Applied to arithmetic expressions.

• Goal:
  – Split into two-operand expressions to exploit hardware parallelism at best.

• Techniques:
  – Balance the expression tree.
  – Exploit commutativity, associativity and distributivity.
Example of tree-height reduction using commutativity and associativity

\[ x = a + b \times c + d \Rightarrow x = (a + d) + b \times c \]
Example of tree-height reduction using distributivity

\[
\begin{align*}
\text{(a)} & \quad x = a \cdot (b \cdot c \cdot d + e) \Rightarrow x = a \cdot b \cdot c \cdot d + a \cdot e;
\end{align*}
\]
Examples of propagation

• Constant propagation:
  
  – $a = 0; \ b = a + 1; \ c = 2 \times b;$
  
  – $a = 0; \ b = 1; \ c = 2;$

• Variable propagation:
  
  – $a = x; \ b = a + 1; \ c = 2 \times a;$
  
  – $a = x; \ b = x + 1; \ c = 2 \times x;$
Sub-expression elimination

- Logic expressions:
  - Performed by logic optimization.
  - Kernel-based methods.

- Arithmetic expressions:
  - Search isomorphic patterns in the parse trees.
  - Example:

\[
\begin{align*}
* a &= x + y; & b &= a + 1; & c &= x + y; \\
* a &= x + y; & b &= a + 1; & c &= a;
\end{align*}
\]
Examples of other transformations

- Dead-code elimination:
  \[ a = x; \quad b = x + 1; \quad c = 2 \times x; \]
  \[ a = x; \text{can be removed if not referenced.} \]

- Operator-strength reduction:
  \[ a = x^2; \quad b = 3 \times x; \]
  \[ a = x \times x; \quad t = x \ll 1; \quad b = x + t; \]

- Code motion:
  \[ \text{for} \ (i = 1; i \leq a \times b) \{ \} \]
  \[ t = a \times b; \quad \text{for} \ (i = 1; i \leq t) \{ \} \]
Control-flow based transformations

- Model expansion.
- Conditional expansion.
- Loop expansion.
- Block-level transformations.
Model expansion

- Expand subroutine – flatten hierarchy.

- Useful to expand scope of other optimization techniques.

- Problematic when routine is called more than once.

- Example:

  - $x = a + b; \quad y = a \times b; \quad z = \text{foo}(x, y);$

  - $\text{foo}(p, q)\{ t = q - p; \quad \text{return}(t); \}$

  - By expanding $\text{foo}$:

  - $x = a + b; \quad y = a \times b; \quad z = y - x;$
Conditional expansion

- Transform conditional into parallel execution with test at the end.

- Useful when test depends on late signals.

- May preclude hardware sharing.

- Always useful for logic expressions.

- Example:

  \[- y = ab; \quad \text{if} \ (a) \  \{x = b + d; \} \quad \text{else} \  \{x = bd; \}\]

  can be expanded to: \[x = a(b+d) + a'bd\]

  and simplified as: \[y = ab; \ x = y + d(a + b)\]
Loop expansion

- Applicable to loops with data-independent exit conditions.

- Useful to expand scope of other optimization techniques.

- Problematic when loop has many iterations.

- Example:

  \[- x = 0; \\
  \textbf{for} \ (i = 1; i \leq 3; i++) \ \{x = x + 1; \} \]

- Expanded to:

  \[- x = 0; \ x = x+1; \ x = x+2; \ x = x+3 \]
Architectural synthesis and optimization

• Synthesize macroscopic structure in terms of building-blocks.

• Explore area/performance trade-off:
  – maximum performance implementations subject to area constraints.
  – minimum area implementations subject to performance constraints.

• Determine an optimal implementation.

• Create logic model for data-path and control.
Design space and objectives

• Design space:
  – Set of all feasible implementations.

• Implementation parameters:
  – Area.
  – Performance:
    * Cycle-time.
    * Latency.
    * Throughput (for pipelined implementations)
  – Power consumption
Design evaluation space
Hardware modeling

- Circuit behavior:
  - Sequencing graphs.

- Building blocks:
  - Resources.

- Constraints:
  - Timing and resource usage.
Resources

- Functional resources:
  - Perform operations on data.
  - Example: arithmetic and logic blocks.

- Memory resources:
  - Store data.
  - Example: memory and registers.

- Interface resources:
  - Example: busses and ports.
Functional resources

• Standard resources:
  – Existing macro-cells.
  – Well characterized (area/delay).
  – Example: adders, multipliers, ...

• Application-specific resources:
  – Circuits for specific tasks.
  – Yet to be synthesized.
  – Example: instruction decoder.
Resources and circuit families

- **Resource-dominated circuits.**
  - Area and performance depend on few, well-characterized blocks.
  - Example: DSP circuits.

- **Non resource-dominated circuits.**
  - Area and performance are strongly influenced by sparse logic, control and wiring.
  - Example: some ASIC circuits.
Implementation constraints

- Timing constraints:
  - Cycle-time.
  - Latency of a set of operations.
  - Time spacing between operation pairs.

- Resource constraints:
  - Resource usage (or allocation).
  - Partial binding.
Synthesis in the temporal domain

- **Scheduling:**
  - Associate a *start-time* with each operation.
  - Determine latency and parallelism of the implementation.

- **Scheduled sequencing graph:**
  - Sequencing graph with start-time annotation.
Example
Synthesis in the spatial domain

- **Binding:**
  - Associate a resource with each operation with the same type.
  - Determine area of the implementation.

- **Sharing:**
  - Bind a resource to more than one operation.
  - Operations must not execute concurrently.

- **Bound sequencing graph:**
  - Sequencing graph with resource annotation.
Example
Binding specification

- Mapping from the vertex set to the set of resource instances, for each given type.

- Partial binding:
  - Partial mapping, given as design constraint.

- Compatible binding:
  - Binding satisfying the constraints of the partial binding.
Example

TIME 1

TIME 2

TIME 3

TIME 4

NOP

NOP n

NOP 0

NOP 10

NOP 9

NOP 8

NOP 7

NOP 6

NOP 5

NOP 4

NOP 3

NOP 2

NOP 1
Estimation

- Resource-dominated circuits.
  - Area = sum of the area of the resources bound to the operations.
    * Determined by binding.
  - Latency = start time of the sink operation (minus start time of the source operation).
    * Determined by scheduling

- Non resource-dominated circuits.
  - Area also affected by:
    * registers, steering logic, wiring and control.
  - Cycle-time also affected by:
    * steering logic, wiring and (possibly) control.
Approaches to architectural optimization

- *Multiple-criteria* optimization problem:
  - area, latency, cycle-time.

- Determine *Pareto optimal* points:
  - Implementations such that no other has all parameters with inferior values.

- Draw trade-off curves:
  - discontinuous and highly nonlinear.
Approaches to architectural optimization

- Area/latency trade-off,
  - for some values of the cycle-time.

- Cycle-time/latency trade-off,
  - for some binding (area).

- Area/cycle-time trade-off,
  - for some schedules (latency).
Area/latency trade-off
Area-latency trade-off

- Rationale:
  - Cycle-time dictated by system constraints.

- Resource-dominated circuits:
  - Area is determined by resource usage.

- Approaches:
  - *Schedule* for minimum latency under resource constraints
  - *Schedule* for minimum resource usage under latency constraints
    * for varying constraints.
Summary

- Behavioral optimization:
  - Create abstract models from HDL models.
  - Optimize models without considering implementation parameters.

- Architectural synthesis and optimization.
  - Consider resource parameters.
  - Multiple-criteria optimization problem:
    * area, latency, cycle-time.