Reconfigurable Computing

Partitioning

Chapter 5

Prof. Dr.-Ing. Jürgen Teich
Lehrstuhl für Hardware-Software-Co-Design
Partitioning - Motivation

- A design implementation is often too big to allow its implementation on a single FPGA.

- Possible solutions are:
  - **Spatial partitioning**: The design is partitioned into many FPGAs. Each partition block is implemented in one single FPGA. All the FPGAs are used simultaneously.
  - **Temporal partitioning**: The design is partitioned into blocks, each of which will be executed in sequence on one single FPGA.

- We will give a short overview on spatial partitioning in the first part of the chapter. Temporal partitioning algorithms will be considered in detail in the second part of this chapter.
Partitioning – definitions

- Dataflow graph: A dataflow or sequencing graph or task graph $G = (V, E)$ is a directed acyclic graph in which
  - each node $v_i$ in $V$ represents a task with execution time $d_i$
  - An edge $e = (u, v)$ represents a data dependency between the nodes $u$ and $v$.

- Scheduling and ordering relation: Given a DFG $G = (V, E)$ with a precedence relation among the nodes
  - A schedule is a function $s: V \rightarrow \mathbb{N}$.
  - A schedule defines for each node, the time at which the node will be executed on the reconfigurable device.
    - A schedule is feasible iff $\forall (u, v) \in E: s(u) \leq s(v)$
  - We define an ordering relation $\leq$ induced by any schedule $s$ as follows:
    - $u \leq v \iff s(u) \leq s(v)$
Partitioning – definitions

- The relation $\leq$ can be extended to sets as follows:
  $(A \leq B) \iff \forall a \in A, b \in B: \text{either } a \text{ is not in relation with } b \text{ or } a \leq b.$

- Partition: Given a DFG $G=(V,E)$ and a set $R=\{R_1, R_2, \ldots, R_k\}$ of reconfigurable devices. A partition $P$ of a graph $G$ toward $R$ is its division into some disjoint subsets $P_1, P_2, \ldots, P_r$:
  $\forall P_i \exists R_j: S(P_i) \leq S(R_j) \land T(P_i) \leq T(R_j)$
  where $S(X) =$ size of $X$ and $T(X) =$ # terminals of $X$

- A partition is called *spatial* iff $(p_{ij}=1$ iff $P_i$ will be implemented in $R_j)$ \(|\{ P_i \in P: p_{ij} = 1\}| \leq 1 \ \forall R_j \in R$

- A partition is *temporal* iff $\exists R_j \in R: |\{P_i \in P: p_{ij} = 1\}| > 1$

- If all the devices in $R$ are of the same type, then the partition is said to be uniform.

- If $|R|=1$, we have a single device partition
Spatial partitioning
Spatial partitioning – Problem

- **Partitioning Constraints:** Each FPGA is characterized by:
  - The size, i.e., the number of LUTs, FFs available
  - The terminals, i.e., the number of I/O pins available on the device
  - A partition is valid iff for a block B produced by the partition, we have:
    - $S(B) \leq S(\text{device})$ where $S(X) = \text{size of } X$
    - $T(B) \leq T(\text{device})$ where $T(X) = \# \text{ terminals of } X$
Objectives: The following objectives are possible:
- Minimize the number of cut nets
- Minimize the number of produced blocks
- Minimize the delay

Difficult problem due to all the constraints which are not always compatible.

Solution approaches:
- Use of heuristics for automatic partitioning
- Manual intervention
Spatial partitioning – Approaches – Hierarchical

Goal:
- Partition a netlist into a minimal number of subgraphs (partitions) where each subgraph fits into one FPGA (capacity) and also satisfies I/O pin constraints (so-called valid blocks).

Problem:
- The size of the flat netlist is too big in general.

Solution:
- Create a hierarchy of objects.
- Use methods to change the hierarchy, e.g., to avoid non-valid blocks.
Spatial partitioning – Approaches – Hierarchical

- **Approach:**
  - Apply an algorithm for clustering a flat netlist (creates green rectangles)
  - Flatten the hierarchy except created (green) clusters
  - Partition this flat netlist (reduced problem size)
Spatial partitioning – Approaches – Hierarchical

- Removing hierarchy of non-valid blocks may produce a big amount of glue logic in the final problem.

- Some non-valid blocks may be partitioned separately by applying a divide-and-conquer strategy.

- **ST quality** is used to determine how good a partition block is:
  \[
  ST = \frac{S}{T} \quad (S=\text{Size}, \ T=\text{Terminal})
  \]
  defines the ratio size/terminal.

- **Poor ST-quality**: Blocks having many connections with other hierarchy blocks.
  - Removing hierarchy is preferable.
Spatial partitioning – Transformations

Remove hierarchy

Splitting
Spatial partitioning – Approaches – Hierarchical

- **Good ST-quality**: Blocks having few connections with other hierarchy blocks
  - Splitting is preferable
- **Average ST-quality**: calculated recursively in a bottom-up fashion (for a global view)
- **Device ST-quality**: ST(D).
  - Device filling is good when the ST-quality of the assigned block is larger or equal to the device quality.

<table>
<thead>
<tr>
<th>Blocks</th>
<th>ST-quality</th>
<th>ST &lt; ST(D)</th>
<th>ST &gt;= ST(D) and ST &gt;= average ST</th>
<th>ST &gt;= ST(D) and ST &lt; average ST</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Leaf block</strong></td>
<td></td>
<td>Remove</td>
<td>Split</td>
<td>Split</td>
</tr>
<tr>
<td><strong>Non leaf block with big amount of glue logic</strong></td>
<td></td>
<td>Remove</td>
<td>Split</td>
<td>Split</td>
</tr>
<tr>
<td><strong>Non leaf block with small amount of glue logic</strong></td>
<td></td>
<td>Remove</td>
<td>Split</td>
<td>Remove</td>
</tr>
</tbody>
</table>

Big size, small I/O pin number, good ST-quality

![Diagram](image-url)
Spatial partitioning – User intervention

- Fully automatic partitioning never satisfies designers
- User intervention may lead to more efficient results
- A mixture of manual and automatic strategies is therefore common
- User intervention:
  - Assignment of hierarchy blocks to devices
  - Hierarchy modification
  - Manual guidance of the automatic partitioning
  - Invoking automatic partitioning on selected blocks (splitting)
Spatial partitioning – Timing – Block replication

Critical path optimization

Reducing the number of I/O pins
Temporal partitioning
Temporal partitioning – Problem definition

- **Temporal partitioning:**
  - We consider a single device temporal partitioning of a DFG G=(V,E) for a device R.
  - A temporal partition can also be defined as an ordered partition of G with the constraints imposed by R.
  - With the ordering relation imposed on the partition, we reduce the solution space to only those partitions which can be scheduled on the device for execution.
  - Therefore, cycles are not allowed in the dataflow graph. Otherwise, the resulting partition may not be schedulable on the device.
Temporal partitioning - Problem

- **Goal:**
  - Computation and scheduling of a Configuration graph

- **In a configuration graph,**
  - Nodes are partitions or bitstreams
  - Edges reflect the precedence in a given DFG
  - The partition blocks communicate by means of inter-configuration registers usually mapped into the processor address space
  - The configuration sequence is controlled by a host processor
  - On configuration, save register values. This requires a given amount of memory
  - After reconfiguration, copy values back
Temporal partitioning - Problem

- **Objectives:**
  - Minimize the number of interconnections. This is one of the most important objectives since it will minimize:
    - the amount of exchanged data
    - the amount of memory for temporally storing the data
  - Minimize the number of produced blocks
  - Minimize the overall computation delay

- **Quality of the result:** Provides a means to measure how good an algorithm performs
  - Connectivity of a graph $G=(V,E)$:
    $$\text{con}(G) = \frac{2|E|}{|V|^2 - |V|}$$
  - Quality of Partitioning $P = \{P_1,...,P_n\}$: Average connectivity over $P$
  - High (low) quality means algorithm performs well (poor).
Temporal partitioning vs Scheduling

- **Scheduling**: Given is a DFG and an architecture which is a set of resources
  - Compute the starting time of each node on a given resource
- **Temporal partitioning**: Given is a DFG and a reconfigurable device
  - The starting time of each node is the starting time of the partition to which it belongs!
  - Compute the starting time of each node on the device
- **Solution approaches:**
  - List scheduling
  - Integer Linear Programming
  - Network Flow
  - Spectral method
Unconstrained Scheduling

- **ASAP** (as soon as possible)
  - Defines the earliest starting time for each node in the DFG
  - Computes the minimal latency (lower bound)

- **ALAP** (as late as possible)
  - Defines the latest starting time for each node in the DFG according to a given latency

- The mobility of a node is the difference between the ALAP-starting time and ASAP-starting time
  - Mobility is 0 → node is on a critical path
Unconstrained scheduling with optimal latency: $L = 4$
ASAP(\(G(V,E),d\)) { 
  FOREACH (\(v_i\) without predecessor) 
  \(s(v_i) := 0;\) 
  REPEAT { 
    choose a node \(v_i\) whose predecessors 
    are all planned; 
    \(s(v_i) := \max_{j: (v_j, v_i) \in E} \{s(v_j) + d_j\};\) 
  } 
  UNTIL (all nodes \(v_i\) are planned); 
  RETURN \(s\) 
}
Unconstrained scheduling with optimal latency: $L = 4$
ALAP\textbf{-Algorithm}

\begin{verbatim}
ALAP(G(V,E),d, L) {
    FOREACH (v_i without successor)
        s(v_i) := L - d_i;
    REPEAT {
        Choose a node v_i whose successors are all planned;
        s(v_i) := \min_{j: (v_i,v_j) \in E} \{s(v_j)\} - d_i;
    }
    UNTIL (all nodes v_i are planned);
    RETURN s
}
\end{verbatim}
Mobility
Constrained scheduling

- **Extended ASAP, ALAP**
  - Compute ASAP or ALAP
  - Assign the tasks earlier (ASAP) or later (ALAP) such that the resource constraints are always fulfilled by construction

- **Listscheduling**
  - A list \( L \) of ready to run tasks is created
  - Tasks are placed in \( L \) in decreasing priority order
  - At a given step, the task with highest priority is assigned to the free resource.
  - Criteria can be: number of successors, mobility, connectivity, etc.
Extended ASAP, ALAP

2 Multiplier, 2 ALUs (+, −, <)

Time 0

Time 1

Time 2

Time 3

Time 4

Reconfigurable Computing
Constrained scheduling

Criterion: number of successors

Resource: 1 multiplier, 1 ALU (+, −, <)
Constrained scheduling

Time 0

Time 1

Time 2

Time 3

Time 4

Time 5

Time 6

Time 7
List Scheduling (LS) for partitioning

1. Construct a list $L$ of all nodes with priorities

2. Create a new empty partition $P_{\text{act}}$
   
2.1 Remove a node from the list and place it in the partition

2.2 If $\text{size}(P_{\text{act}}) \leq \text{size}(R)$ and $T(P_{\text{act}}) \leq T(R)$ goto 2.1, else goto 2.3

2.3 If empty(list), stop
    else goto 2.
Temporal partitioning vs constrained scheduling

Criterion: number of successors

\[ \text{size(FPGA)} = 250, \text{size(mult)} = 100, \text{size(add)} = \text{size(sub)} = 20, \text{size(comp)} = 10 \]
Temporal partitioning vs constrained scheduling

Connectivity: \( c(P1) = \frac{1}{6} \), \( c(P2) = \frac{1}{3} \), \( c(P3) = \frac{1}{3} \)

Quality: \( \frac{5}{18} = 0.27 \)
Temporal partitioning vs constrained scheduling

Connectivity: $c(P1) = \frac{1}{5}$, $c(P2) = \frac{2}{3}$, $c(P3) = \frac{2}{3}$

Quality: 0.51

Connectivity is better.
List scheduling – list construction

- **ASAP**
  - Place the currently processed node in the list if all its predecessors are already in the list.
  - This corresponds to:
    - Assigning a level number to nodes
    - Scheduling the nodes for execution according to the level number

- **Drawback**
  - “Levelization”: Nodes are assigned to partitions only on the basis of their level-number (no consideration of amount of data exchange)

- **Advantage**
  - Fast (polynomial run-time)
  - Local optimization possible
List scheduling - Improvement

- Local optimization by configuration switching (Bobda)
- If two consecutive partitions P1 and P2 share a common set of operators, then:
  - We implement the minimal set of operators needed for the two partitions.
  - We use signal multiplexing to switch from one partition to the next one.

**Drawbacks**: More resources are needed to implement the signal switching

**Advantages**:
- Reconfiguration time is reduced
- Device operation is not interrupted
List scheduling – config switching

Configuration 1

Configuration 2

Inter configuration register
List scheduling – config switching

Configuration 1

Configuration 2

Inter configuration register
List scheduling – config switching

Configuration 1

Configuration 2

Inter configuration register
List scheduling - Improvement

- Improved List Scheduling algorithm
  1. Generate the list of nodes `node_list`
  2. Build a first partition P1
  3. While (`!node_list.empty()`)
     4. Build a new partition P2
  5. If `union(P1, P2)` fits on the device, then
     implement configuration switching with P1 and P2
  6. Else set P1 = P2 and goto 3
  7. Exit
Temporal partitioning – ILP

- With the ILP (Integer Linear Programming),
  - the temporal partitioning constraints are formulated as inequalities.
  - The system of inequalities is then solved using an ILP-solver.

- The constraints usually considered are:
  - Uniqueness constraint
  - Precedence (temporal order) constraint
  - Memory constraint
  - Resource constraint
  - Latency constraint

- Notations:
  \[ y_{vi} = 1 \iff v \in P_i \]
  \[ w_{uv} = 1 \iff (u,v) \in E, u \in P_i \land v \in P_j \land P_i \neq P_j \]

Reconfigurable Computing
40
Temporal partitioning – ILP

- **Unique assignment constraint**: Each task should be placed in exactly one partition: \( \forall v \in V: \sum_{i=1,\ldots,m} y_{vi} = 1 \)

- **Precedence constraint**: for each edge \((u,v)\) in the graph, node \(u\) must be placed either in the same partition as \(v\) or in an earlier partition than that in which \(v\) is placed:
  \[
  \forall (u,v) \in E: \sum_{i=1}^{m} i y_{ui} \leq \sum_{i=1}^{m} i y_{vi}
  \]

- **Resource constraint**: The sum of the resources needed to implement the modules in one partition should not exceed the total amount of available resources:
  - **Device area constraint**
    \[
    \forall P_i \in P: \sum_{u \in V} y_{ui} s(u) \leq S(\text{device})
    \]
  - **Device terminal constraint**
    \[
    \forall P_i \in P: \sum_{u \in P_i, v \notin P_i} w_{uv} + \sum_{u \notin P_i, v \in P_i} w_{uv} \leq T(\text{device})
    \]
Temporal partitioning – ILP

- **Device terminal constraint:**

\[
\forall P_i \in P: \sum_{u \in P_i, v \notin P_i} w_{u,v} + \sum_{u \notin P_i, v \in P_i} w_{u,v} \leq T_{\text{device}}
\]

\[
\sum_{u \in P_i, v \notin P_i} g_{u,v} y_{u_i} (1 - y_{v_i}) + \sum_{u \notin P_i, v \in P_i} g_{u,v} (1 - y_{u_i}) y_{v_i} \leq T_{\text{device}}
\]

\( g_{u,v} = 1 \), if there is a data dependency from \( u \) to \( v \), 0 else.

\[
\sum_{u \in P_i, v \notin P_i} g_{u,v} y_{u_i} - g_{u,v} y_{u_i} y_{v_i}
\]

\[
+ \sum_{u \notin P_i, v \in P_i} g_{u,v} y_{v_i} - g_{u,v} y_{v_i} y_{u_i} \leq T_{\text{device}}
\]
Temporal partitioning – ILP

- This is not a linear constraint due to the multiplication of two variables but can be linearized by introducing a new variable $z_{u_i,v_i} = y_{u_i}y_{v_i}$ such that

$$
\sum_{u \in P_i, v \in \overline{P_i}} g_{u,v}(y_{u_i} - z_{u_i,v_i}) + \sum_{u \in \overline{P_i}, v \in P_i} g_{u,v}(y_{u_i} - z_{u_i,v_i}) \leq T_{device}
$$

$z_{u_i,v_i}$ is subject to the following constraints:

- $z_{u_i,v_i} \leq y_{u_i}$
- $z_{u_i,v_i} \leq y_{v_i}$
- $y_{u_i} + y_{v_i} - z_{u_i,v_i} \leq 1$

<table>
<thead>
<tr>
<th>$y_{u_i}$</th>
<th>$y_{v_i}$</th>
<th>$z_{u_i,v_i}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Recursive bi-partitioning:

- The goal at each step is the generation of a uni-directional bi-partition.
- The goal at each step is to compute a bi-partition which minimizes the edge-cut size between the two partition blocks.
- Network flow methods are used to compute the bi-partition with minimal edge-cut size.
- Directly applying the min-cut max-flow theorem may lead to non-unidirectional cuts.
- Therefore, the original G is first transformed into a new graph G' in which each cut will be unidirectional in an optimal solution.
Network-flow – graph transformations

- **Two-terminal net transformation**
  - Replace an edge \((v_1, v_2)\) by two edges \((v_1, v_2)\) with capacity 1 and \((v_2, v_1)\) with infinite capacity.

- **Multi-terminal net transformation**
  - For a multi-terminal net \(\{v_1, v_2, \ldots, v_n\}\), introduce a dummy node \(v\) with no weight and a bridging edge \((v_1, v)\) with capacity 1.
  - Introduce the edges \((v, v_2), \ldots, (v, v_n)\), each of which is assigned a capacity of 1.
  - Introduce the edges \((v_2, v_1), \ldots, (v_n, v_1)\), each of which is assigned an infinite capacity.
  - Having computed a min-cut in the transformed graph \(G\), a min-cut can be derived in \(G\): for each node of \(G'\) assigned to a partition, its counterpart in \(G\) is assigned to the corresponding partition in \(G\).