Compilation for Embedded Reconfigurable Computing Architectures: Part B

João M. P. Cardoso, Pedro C. Diniz

3rd Summer School on Generative and Transformational Techniques in Software Engineering

6–11 July, 2009, Braga, Portugal
Outline

• First part of the Tutorial: Architectures
  – Motivation
  – Technology Trends
  – Reconfigurable Computing
  – Embedded and Reconfigurable Architectures
  – Reconfiguration
  – Improving Performance

• Second part of the Tutorial: Compiling
  – Main Compiler and Execution Concepts
  – Compiling to Fine-Grained Reconfigurable Architectures
  – Optimizations

• Conclusions
PROGRAMMING
“The basic parameters of the application are known - there will be a signal stream coming in, a datapath that processes that stream, a controller that sequences the datapath, some memory elements for buffering and storage, and some output stream.”

in “Akya Reconfigurable Logic Roll Your Own FPGA - Sort of”

by Kevin Morris, FPGA and Structured ASIC Journal
Compilation Steps

1. **Imperative Programming**
   - Software Language
   - Hardware/Software Partitioning
     - Code for Microprocessor
     - Compile to Microprocessor
     - Object Code for Microprocessor

2. **Hardware Accelerator**
   - Code for Hardware Accelerator
   - Compile to Hardware Accelerator
   - Binaries for Configuring/Programming the Hardware Accelerator

3. **Compilation from High-Level Language**
   - HDL code at RTL
   - RTL Synthesis
   - Placement and Routing

4. **Memory**
   - Microprocessor
   - Hardware Accelerator
Programming

- Years of efforts on parallelizing compilers yielded meager returns on the potential for concurrent execution
- Movement in industry for new concurrent programming paradigms and languages (upc, X-10, Fortress, etc.)

**Showstopper:** Programming is excruciatingly painful... How to make devices like FPGAs easily programmable is a hard research problem, still.
Programming

• Future reconfigurable architectures will exacerbate all the programming problems
• Issues:
  – How can programming languages help the compiler?
  – How can architectures help the compiler and tools?
FROM IMPERATIVE PROGRAMS TO APPLICATION-SPECIFIC ARCHITECTURES
Hardware Compilation

- Hardware structures to control execution of a data-path
  - data-path: basic operations
  - control unit: derived from CFG
- Increase ILP
  - use data-dependence analysis
  - control flow evaluates multiple branches
- Important execution technique:
  - pipelining for data-path and memory accesses
Data Dependence

• Data-Dependence Analysis:
  - determines operations data and control dependences
  - allows compilers to build data flows
  - schedule their execution

Analysis Results:
instruction “e = a + f” can execute concurrently with other two instructions
Hardware Compilation

- **IF statements** ⇒ two alternatives:
  
  „data flow“

  if (x < 5) 
  y = a;
  else
  y = a + 3;

  „control flow“

  x < 5 a 3
  + y

  5
  x

  < a
Hardware Compilation

• Loops:

a = x;
for (i = 1; i <= 10; i++)
a = a + i;
Hardware Compilation

- Memory (port) access
  - $A[\text{index}_1], B[\text{index}_k],...$
Hardware Compilation: Resource Sharing

• Example:
Hardware Compilation: Resource Sharing

- Example:

```
load

x

+ load

x load

select; load

load

x

select; load

load

+ load
```
Hardware Compilation

• Example

- function $f_1 = \{1, 3\}$
- function $f_2 = \{2\}$
- memory $M_1 = \{A, B\}$, single port

• Allocation

Perform Scheduling considering:
- $f_1$ requires 2 clock cycles (ccs)
- $f_2$ requires 1 ccs
- load/stores require 1 ccs
- one load/store each time
Hardware Compilation

• (List) Scheduling

1 - DFG nodes ready:


\[ cs \leftarrow 1; \]

2 - Sort nodes ready by priorities (e.g., ALAP)

Perform Scheduling considering:

- \( f1 \) requires 2 clock cycles (ccs)
- \( f2 \) requires 1 ccs
- load/stores require 1 ccs
- one load/store each time
Hardware Compilation

- ASAP (As Soon As Possible) and ALAP (As Late As Possible): <ASAP, ALAP>

Perform Scheduling considering:
- f1 requires 2 clock cycles (ccs)
- f2 requires 1 ccs
- load/stores require 1 ccs
- one load/store each time
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- \( f_1 \) requires 2 clock cycles (ccs)
- \( f_2 \) requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

\[ cs \leftarrow 1; \]
1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

\[
\begin{align*}
\end{align*}
\]

Table:

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>f1</th>
<th>f2</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Compilation

• (List) Scheduling

1 - DFG nodes ready Sorted by priorities (e.g., ALAP):


2 – start scheduling nodes ready

Perform Scheduling considering:
- f1 requires 2 clock cycles (ccs)
- f2 requires 1 ccs
- load/stores require 1 ccs
  one load/store each time
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
  f1 requires 2 clock cycles (ccs)
  f2 requires 1 ccs
  load/stores require 1 ccs
  one load/store each time

\[ \text{cs} \leftarrow 1; \]
1 - DFG nodes ready Sorted by priorities (e.g., ALAP):
\[ \begin{align*}
\end{align*} \]

2 – start scheduling nodes ready
3- for each node scheduled find new nodes ready
4- if all nodes ready considered for scheduling
\[ \text{cs} \leftarrow \text{cs} + 1; \]
5 – goto 1

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>f1</th>
<th>f2</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>A[1]</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- \( f_1 \) requires 2 clock cycles (ccs)
- \( f_2 \) requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

\[
\begin{align*}
\text{stage (cs)} & \quad f_1 & \quad f_2 & \quad \text{load/store} \\
1 & \quad & \quad & \quad & \quad A[1] \\
2 & \quad & \quad & \quad & \quad B[1] \\
3 & \quad & \quad & \quad & \quad \text{---} \\
4 & \quad & \quad & \quad & \quad \text{---} \\
5 & \quad & \quad & \quad & \quad \text{---} \\
6 & \quad & \quad & \quad & \quad \text{---} \\
7 & \quad & \quad & \quad & \quad \text{---} \\
8 & \quad & \quad & \quad & \quad \text{---} \\
\end{align*}
\]

\[
\begin{align*}
\text{cs} & \leftarrow 1; \\
1 - \text{DFG nodes ready Sorted by priorities (e.g., ALAP):} & \\
\text{A[2]} & \quad \text{B[2]} \\
\text{2} - \text{start scheduling nodes ready} & \\
3 \text{- for each node scheduled find new nodes ready} & \\
4 \text{- if all nodes ready considered for scheduling} & \\
\text{cs} & \leftarrow \text{cs+1;} \\
5 - \text{goto 1} & \\
\end{align*}
\]
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- f1 requires 2 clock cycles (ccs)
- f2 requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

cs ← 1;
1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

2 – start scheduling nodes ready
3- for each node scheduled find new nodes ready
4- if all nodes ready considered for scheduling

cs ← cs+1;
5 – goto 1

---

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>f1</th>
<th>f2</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>A[1]</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>B[1]</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Compilation

• (List) Scheduling

1. DFG nodes ready Sorted by priorities (e.g., ALAP):

   cs ← 1;
   1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

   2. Start scheduling nodes ready
   3. For each node scheduled, find new nodes ready
   4. If all nodes ready considered for scheduling
   5. Goto 1

Perform Scheduling considering:
- f1 requires 2 clock cycles (ccs)
- f2 requires 1 ccs
- load/stores require 1 ccs
- one load/store each time
Hardware Compilation

• (List) Scheduling

1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

2 – start scheduling nodes ready
3 - for each node scheduled find new nodes ready
4- if all nodes ready considered for scheduling

Perform Scheduling considering:
f1 requires 2 clock cycles (ccs)
f2 requires 1 ccs
load/stores require 1 ccs
one load/store each time

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>f1</th>
<th>f2</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>A[1]</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>B[1]</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td>1</td>
<td>A[2]</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Compilation

• (List) Scheduling

 Perform Scheduling considering:
  - f1 requires 2 clock cycles (ccs)
  - f2 requires 1 ccs
  - load/stores require 1 ccs
  - one load/store each time

\[\begin{align*}
\text{cs} & \leftarrow 1; \\
1 & \text{ - DFG nodes ready Sorted by priorities (e.g., ALAP):}
\end{align*}\]

\[\begin{align*}
2 & \text{ – start scheduling nodes ready}
3 & \text{ - for each node scheduled find new nodes ready}
4 & \text{ - if all nodes ready considered for scheduling}
\text{cs} & \leftarrow \text{cs+1;}
5 & \text{ – goto 1}
\end{align*}\]
Hardware Compilation

• (List) Scheduling

1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

\[
\text{cs} \leftarrow 1;
\]

2 - start scheduling nodes ready

3 - for each node scheduled find new nodes ready

4 - if all nodes ready considered for scheduling

\[
\text{cs} \leftarrow \text{cs}+1;
\]

5 - goto 1

Perform Scheduling considering:

- \( f_1 \) requires 2 clock cycles (ccs)
- \( f_2 \) requires 1 ccs
- load/stores require 1 ccs
- one load/store each time
Hardware Compilation

• (List) Scheduling

**Diagram:**


**Perform Scheduling considering:**

- $f_1$ requires 2 clock cycles (ccs)
- $f_2$ requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

**Steps:**

1. $\text{cs} \leftarrow 1$;
2. DFG nodes ready Sorted by priorities (e.g., ALAP):
3. Start scheduling nodes ready
4. For each node scheduled find new nodes ready
5. If all nodes ready considered for scheduling $\text{cs} \leftarrow \text{cs} + 1$;
6. Goto 1

**Table:**

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>f1</th>
<th>f2</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>A[1]</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>B[1]</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td>A[2]</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td></td>
<td>B[2]</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- $f_1$ requires 2 clock cycles (ccs)
- $f_2$ requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

2 - start scheduling nodes ready
3 - for each node scheduled find new nodes ready
4 - if all nodes ready considered for scheduling

\[
\text{cs} \leftarrow 1;
\]

\[
1, 2, 3, 4, 5, 6, 7, 8
\]

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>$f_1$</th>
<th>$f_2$</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td>B[2]</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

29
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- \( f_1 \) requires 2 clock cycles (ccs)
- \( f_2 \) requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

\[
\text{cs} \leftarrow 1;
\]

2 – start scheduling nodes ready
3- for each node scheduled find new nodes ready
4- if all nodes ready considered for scheduling
\[
\text{cs} \leftarrow \text{cs}+1;
\]
5 – goto 1

<table>
<thead>
<tr>
<th>stage (cs)</th>
<th>f1</th>
<th>f2</th>
<th>load/store</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td>A[1]</td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td>B[1]</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td>A[2]</td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>2</td>
<td>B[2]</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- \( f_1 \) requires 2 clock cycles (ccs)
- \( f_2 \) requires 1 ccs
- load/stores require 1 ccs
- one load/store each time

\[
\begin{align*}
cs & \leftarrow 1; \\
1 & - \text{DFG nodes ready Sorted by priorities (e.g., ALAP):} \\
2 & - \text{start scheduling nodes ready} \\
3 & - \text{for each node scheduled find new nodes ready} \\
4 & - \text{if all nodes ready considered for scheduling} \\
& \quad cs \leftarrow cs + 1; \\
5 & - \text{goto 1}
\end{align*}
\]
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
  f1 requires 2 clock cycles (ccs)
  f2 requires 1 ccs
  load/stores require 1 ccs
  one load/store each time

\[
\begin{align*}
\text{cs} &\leftarrow 1; \\
1 & - \text{DFG nodes ready Sorted by priorities (e.g., ALAP)}:
\]

\[
\begin{align*}
\text{C[1]} &\leftarrow \text{CS+1; for each node scheduled find new nodes ready} \\
2 & - \text{start scheduling nodes ready} \\
3 & - \text{if all nodes ready considered for scheduling}
\end{align*}
\]

\[
\begin{align*}
\text{f1} &\text{ requires 2 clock cycles (ccs)} \\
\text{f2} &\text{ requires 1 ccs} \\
\text{load/stores} &\text{ require 1 ccs}
\end{align*}
\]

\[
\begin{align*}
\text{stage (cs)} &\text{ f1 f2 load/store} \\
1 &\text{ A[1]} \\
2 &\text{ B[1]} \\
3 &\text{ A[2]} \\
4 &\text{ B[2]} \\
5 &\text{ C[1]} \\
6 &\text{ C[1]} \\
7 &\text{ C[1]} \\
8 &\text{ C[1]}
\end{align*}
\]
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
  f1 requires 2 clock cycles (ccs)
  f2 requires 1 ccs
  load/stores require 1 ccs
  one load/store each time

\[ \text{cs } \leftarrow 1; \]
1 - DFG nodes ready Sorted by priorities (e.g., ALAP):

2 – start scheduling nodes ready
3 - for each node scheduled find new nodes ready
4- if all nodes ready considered for scheduling
   cs \leftarrow cs+1;
5 – goto 1

8 stages ⇒ 8 clock cycles
Hardware Compilation

• (List) Scheduling

Perform Scheduling considering:
- \(f_1\) requires 2 clock cycles (ccs)
- \(f_2\) requires 1 ccs
- load/stores require 1 ccs
  - one load/store each time

• naïve scheduling!
Hardware Compilation

• After scheduling it is common to perform binding
  – binding specifically assigns a component of the library to each DFG node
Hardware Compilation

• Scheduling: second example

Perform Scheduling considering:
• \( f_1 \) and \( f_2 \) require 2 clock cycles (\#ccs)
• load/stores require 1 ccs
• one load/store each time

best and worst scheduling?
Hardware Compilation (C2Gates)

BASED ON CFG
• imperative code

int maxmin(int A[], int N) {
    int max = MIN_INT;
    for(int i=0; i<N; i++) {
        if(A[i] > max) max = A[i];
    }
    return max;
}

• Step to machine

int maxmin(int A[], int N) {
    int i=0;
    loop:
        if(i>=N) goto end;
        if(A[i] <= max) goto skip_then;
        max = A[i];
        skip_then:
            i++;
            goto loop;
    end:
        return max;
}
C2Gates

• Step to machine

```c
int maxmin(int A[], int N) {
    int max = MIN_INT;
    int i=0;
    loop:
        if(i>=N) goto end;
        if(A[i] <= max) goto skip_then;
        max = A[i];
    skip_then:
        i++;
        goto loop;
    end:
        return max;
}
```

• Step to machine (after scalar replacement)

```c
int maxmin(int A[], int N) {
    int max = MIN_INT;
    int i=0;
    loop:
        if(i>=N) goto end;
        int ai = A[i];
        if(ai <= max) goto skip_then;
        max = ai;
    skip_then:
        i++;
        goto loop;
    end:
        return max;
}
```
```c
int maxmin(int A[], int N) {
    int max = MIN_INT;
    int i=0;
    loop:
        if(i>=N) goto end;
        int ai = A[i];
        if(ai <= max) goto skip_then;
        max = ai;
        skip_then:
            i++;
        goto loop;
    end:
    return max;
}
```
max = MIN_INT; int i=0;

if(i>=N) goto end else loop;

max = MIN_INT; int i=0;
i>=N

int ai = A[i]; if(ai <= max) left else right;

max = ai;

i++;

return max;

return max;

---

C2Gates

From the CFG to the FSMD (FSM with data-path)
C2Gates

• Step to RTL

max = MIN_INT; int i=0;

i>=N

int ai = A[i]; ai <= max

max = ai;

return max;

i++;
C2Gates

- if-conversion

```
max = MIN_INT;
int i=0;

while i<N:
    int ai = A[i];
    max = ai: if ai>max; i++;

return max;
```

Latency:

- min\(\equiv\)max
- \(2N+3\)
C2Gates

- Merging

```c
max = MIN_INT;
int i=0;

int ai = A[i];
max = ai; if
ai>max;
i++;
i>=N

return max;
```

Latency

- \( \text{min} \equiv \text{max} \)
- \( N + 3 \)
• Back to the CFG
  – operations requiring more than one clock cycle, span over multiple states
  – giving results only in the end of the current clock cycle, cannot be chained with operations requiring that result
C2Gates

- Back to the CFG
  - operations requiring more than one clock cycle or
  - giving results only in the end of the current clock cycle
- **Split basic blocks** or start from a CFG of instructions and merge instructions as long as possible
C2Gates

- load data (A[i]) only available in the next clock cycle
- **Split basic blocks** or start from a CFG of instructions and merge instructions as long as possible
C2Gates

- load data (A[i]) only available in the next clock cycle
- **Split basic blocks** or start from a CFG of instructions and merge instructions as long as possible

```java
max = MIN_INT;
int i=0;

i>=N

int ai = A[i];

ai <= max

max = ai;

return max;

i++;
```
C2Gates

• Image of CFG
  – Easy way to generate application-specific architectures
  – Register Transfer Level (RTL) synthesis deal automatically with the FSMD generated
C2Gates

• Image of CFG
  – Operations performed within each clock cycle
    • difficult model when we have multicycle operations
    • maximum clock frequency limited to the slowest operation
    • many compilers use assignments to drive each new control step (achieved by a CFG of instructions instead of basic blocks)
    • Unnecessary control flow with most if-then and if-then-else constructs (if-conversion promotes control-data-flow)
C2Gates

• More advanced schemes
  – Use CFG of basic blocks but don’t use a direct correspondence between basic blocks and control steps
  – Basic blocks are represented as DAGs
    • DAGs explicitly represent parallelism, common subexpressions, etc.
int maxmin(int A[], int N) {
    int max = MIN_INT;
    for(int i=0; i<N; i++)
        if(A[i] > max) max = A[i];
    return max;
}

VHDL Code

-- VHDL Code

type state_type is (s0, s1, s2, s5);
signal current_state, next_state: state_type;
signal i, max_aux: std_logic_vector(31 downto 0);
begin
    process(reset, clk)
    begin
        if reset = '1' then
            current_state <= s0;
        elsif clk'event AND clk='1' then
            current_state <= next_state;
        end if;
    end process;
    process(current_state, start, A, N)
    variable ai: std_logic_vector(31 downto 0);
    begin
        done <= '0'; -- default values
        case current_state is
            when s0 =>  ... -- start state i <= (others => '0'); ...
            when s1 =>
                if(i >= N) then
                    next_state <= s5;
                else
                    next_state <= s2;
                end if;
            when s2 =>
                ai := ...; -- get A[i], A(CONV_INTEGER(i));
                if(ai > max_aux) then
                    max_aux <= ai;
                end if;
                i <= i+1;
                next_state <= s1;
            when s5 => ...
        end case;
    end process;
end process;
...
Hardware Compilation

BACK-END AFTER VHDL GENERATION
Back-End

- FPGA vendors have their own set of tools to perform final stages
- E.g., Xilinx has the ISE, an integrated system environment
A GUIDED TOUR TO GENERIC OPTIMIZATIONS
Optimizations

• Instruction-based: reduce operation counts/cost
  – Constant Propagation
  – *Strength Reduction*
  – Constant Folding
  – Common Subexpression Elimination
  – dead-code elimination

• Memory Operation: eliminate memory accesses
  – Scalar Replacement
  – Data reuse

• Loop-based: expose concurrent execution opportunities
  – *Loop Unrolling, fusion, fission, tiling, unroll-and-jam, etc.*

• Execution techniques: reduce execution time
  – pipelining
  – if-conversion
void smooth(short[] IN, short[] OUT) {
    int DIM = 350;
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < DIM-3+1; row++) {
        for (int col = 0; col< DIM-3+1; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval += IN[(row +wrow)*DIM+(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * DIM + col] = (short) sumval;
        }
    }
}
Optimizing

Constant Propagation

```c
void smooth(short[] IN, short[] OUT) {
    int DIM = 350;
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 350-3+1; row++) {
        for (int col = 0; col< 350-3+1; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval += IN[(row +wrow)*350+(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
```
Optimizing

Constant folding (Constant-Expression Evaluation):

```c
void smooth(short[] IN, short[] OUT) {
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 350-3+1; row++) {
        for (int col = 0; col < 350-3+1; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol < 3; wcol++) {
                    sumval += IN[(row + wrow)*350+(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
```
void smooth(short[] IN, short[] OUT) {
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                for (int wcol = 0; wcol<3; wcol++) {
                    sumval += IN[(row +wrow)*350+(col+wcol)]*K[wrow*3+wcol];
                }
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
Optimizing

**Algebraic simplification:**

```c
void smooth (short[] IN, short[] OUT) {
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                sumval+= IN[(row +wrow)*350+(col+0)]*K[wrow*3+0];
                sumval+= IN[(row +wrow)*350+(col+1)]*K[wrow*3+1];
                sumval+= IN[(row +wrow)*350+(col+2)]*K[wrow*3+2];
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
```
void smooth(short[] IN, short[] OUT) {
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 348; row++) {
        for (int col = 0; col < 348; col++) {
            int sumval = 0;
            for (int wrow=0; wrow < 3; wrow++) {
                sumval+= IN[(row + wrow)*350 + col]*K[wrow*3];
                sumval+= IN[(row + wrow)*350 + (col+1)]*K[wrow*3+1];
                sumval+= IN[(row + wrow)*350 + (col+2)]*K[wrow*3+2];
            }
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
Optimizing

Algebraic simplifications + constant folding:

```c
void smooth(short[] IN, short[] OUT) {
    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = IN[row*350+col]*K[0];
            sumval += IN[(row+1)*350+(col+1)]*K[1];
            sumval += IN[(row+2)*350+(col+2)]*K[2];
            sumval += IN[(row+1)*350+col]*K[3];
            sumval += IN[(row+1)*350+(col+1)]*K[4];
            sumval += IN[(row+2)*350+(col+2)]*K[5];
            sumval += IN[(row+2)*350+col]*K[6];
            sumval += IN[(row+2)*350+(col+1)]*K[7];
            sumval += IN[(row+2)*350+(col+2)]*K[8];
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
```
void smooth(short[] IN, short[] OUT) {
  short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
  for (int row=0; row < 348; row++) {
    for (int col = 0; col < 348; col++) {
      int sumval = IN[row*350+col]*K[0];
      sumval += IN[row*350+(col+1)]*K[1];
      sumval += IN[row*350+(col+2)]*K[2];
      sumval += IN[(row +1)*350+col]*K[3];
      sumval += IN[(row +1)*350+(col+1)]*K[4];
      sumval += IN[(row +1)*350+(col+2)]*K[5];
      sumval += IN[(row +2)*350+col]*K[6];
      sumval += IN[(row +2)*350+(col+1)]*K[7];
      sumval += IN[(row +2)*350+(col+2)]*K[8];
      sumval = sumval / 16;
      OUT[row * 350 + col] = (short) sumval;
    }
  }
}
Optimizing

*Code Elimination of declarations and initializations not used:*

```c
void smooth(short[] IN, short[] OUT) {
  short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
  for (int row=0; row < 348; row++) {
    for (int col = 0; col< 348; col++) {
      int sumval = IN[row*350+col];
      sumval += IN[row*350+col+1]*2;
      sumval += IN[row*350+col+2];
      sumval += IN[(row +1)*350+col]*2;
      sumval += IN[(row +1)*350+col+1]*4;
      sumval += IN[(row +1)*350+col+2]*2;
      sumval += IN[(row +2)*350+col];
      sumval += IN[(row +2)*350+col+1]*2;
      sumval += IN[(row +2)*350+col+2];
      sumval = sumval / 16;
      OUT[row * 350 + col] = (short) sumval;
    }
  }
}
```
void smooth(short[] IN, short[] OUT) {

    int row = 0;
    while (row < 348) {
        int col = 0;
        while (col < 348) {
            int sumval =
            IN[row*350+col];
            sumval +=
            IN[(row +1)*350+col]<<<1;
            sumval +=
            IN[(row +1)*350+col+1]<<<2;
            sumval +=
            IN[(row +1)*350+col+2]<<<1;
            sumval +=
            IN[(row +2)*350+col];
            sumval +=
            IN[(row +2)*350+col+1]<<<1;
            sumval +=
            IN[(row +2)*350+col+2];
            sumval = sumval >> 4;
            OUT[row * 350 + col] = (short) sumval;
            col++;
        }
        row++;
    }
}

void smooth(short[] IN, short[] OUT) {

    short[] K = {1, 2, 1, 2, 4, 2, 1, 2, 1};
    for (int row=0; row < 348; row++) {
        for (int col = 0; col< 348; col++) {
            int sumval = IN[row*350+col];
            sumval += IN[row*350+col+1]*2;
            sumval += IN[row*350+col+2];
            sumval += IN[(row +1)*350+col]*2;
            sumval += IN[(row +1)*350+col+1]*4;
            sumval += IN[(row +1)*350+col+2]*2;
            sumval += IN[(row +2)*350+col];
            sumval += IN[(row +2)*350+col+1]*2;
            sumval += IN[(row +2)*350+col+2];
            sumval = sumval / 16;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
After algebraic optimizations and reassociation:

```c
void smooth(short[] IN, short[] OUT) {
    for (int row=0; row < 348; row++) {  
        for (int col = 0; col< 348; col++) { 
            int sumval = IN[row*350+col];
            sumval += IN[row*350+col+1] << 1;
            sumval += IN[row*350+col+2];
            sumval += IN[row*350+col+351+col] << 1;
            sumval += IN[row*350+col+352+col] << 1;
            sumval += IN[row*350+col+700+col];
            sumval += IN[row*350+col+701+col] << 1;
            sumval += IN[row*350+col+702+col];
            sumval = sumval >> 4;
            OUT[row * 350 + col] = (short) sumval;
        }
    }
}
```
void smooth(short[] IN, short[] OUT) {
    for (int row=0; row < 348; row++) {
        for (int col = 0; col < 348; col++) {
            int row_350_col = row*350 + col;
            int sumval = IN[row_350_col];
            sumval += IN[row_350_col + 1] << 1;
            sumval += IN[row_350_col + 2];
            sumval += IN[row_350_col + 350] << 1;
            sumval += IN[row_350_col + 351] << 2;
            sumval += IN[row_350_col + 352] << 1;
            sumval += IN[row_350_col + 700];
            sumval += IN[row_350_col + 701] << 1;
            sumval += IN[row_350_col + 702];
            sumval = sumval >> 4;
            OUT[row_350_col] = (short) sumval;
        }
    }
}
Optimizations

• Application of a transformation may enable others
• Hard to figure out the sequence of optimizations that produce the best results
• Human intervention is needed!
• Recent projects try to automatically explore the applications of optimizations (design space exploration problem)
  – some of them using machine learning techniques (e.g., MILEPOST EU funded project)
DATA REUSE
Data Reuse

- Eliminate repeated accesses to the same data
  - Save data in internal registers on first access
  - Reuse data in register in subsequent accesses
  - Careful if data have been modified (dirty data)

- Benefits:
  - Increase data availability reducing bandwidth
  - Can be managed explicitly in RAM Blocks (FPGA) or using custom memory organization (ASIC)

- Costs:
  - Registers (increase pressure on resources)
  - Precise compiler knowledge of data access patterns
for (int i = 2; i < N; i++) {
    y[i] = x[i] + x[i-1] + x[i-2];
}
Data Reuse

```java
for (int i = 2; i< N; i++) {
    y[i] = x[i] + x[i-1] + x[i-2];
}
```
Data Reuse

```c
for (int i = 2; i < N; i++) {
    y[i] = x[i] + x[i-1] + x[i-2];
}
```

```c
int x_2 = x[0];
int x_1 = x[1];
int x_0;
for (int i = 2; i < N; i++) {
    x_0 = x[i];
    y[i] = x_0 + x_1 + x_2;
    x_2 = x_1;
    x_1 = x_0;
}
```
Data Reuse Example Design

- At each Clock Cycle:
  - Compute new y value
  - Access M0 for x
  - Shift Values of x in registers

- Each Iteration
  - 1 read and 1 write

- Naïve Solution:
  - requires 3 registers also
  - to save intermediate results
Data Reuse Example Design

- At each Clock Cycle:
  - Compute new y value
  - Access M0 for x
  - Shift Values of x in registers

- Each Iteration
  - 1 read and 1 write

- Naïve Solution:
  - requires 3 registers also
  - to save intermediate results
Optimizations

LOOP AND DATA LAYOUT TRANSFORMATIONS
Custom Data Layout

- Technique: Layout data (mostly arrays) in memories to suite array access patterns in loops

- Benefits:
  - Increase data availability reducing bandwidth
  - Can be managed explicitly in RAM Blocks (FPGA) or using custom memory organization (ASIC)

- Costs:
  - Registers (increase pressure on resources)
  - Precise compiler knowledge of data access patterns
Custom Data Layout Example

```c
int A[32][16]; int B[32][16];
for (i = 0; i < 32; i++)
    for (j = 0; j < 16; j++)
        A[i][j] = B[i][j] + 1;
```

(a) Original code
Custom Data Layout Example

(a) Original code

```c
int A[32][16]; int B[32][16];
for (i = 0; i < 32; i++)
  for (j = 0; j < 16; j++)
    A[i][j] = B[i][j] + 1;
```

(b) After unroll-and-jam

```c
int A[32][16]; int B[32][16];
for (i = 0; i < 32; i+=2)
  for (j = 0; j < 16; j +=2) {
    A[i][j] = B[i][j] + 1;
    A[i][j+1] = B[i][j+1] + 1;
    A[i+1][j] = B[i+1][j] + 1;
    A[i+1][j+1] = B[i+1][j+1] + 1;
  }
```
Custom Data Layout Example

int A00[16][8]; int A01[16][8]; int A10[16][8]; int A11[16][8];
int B00[16][8]; int B01[16][8];
int B10[16][8]; int B11[16][8];

// code to distribute A and B

for (i = 0; i < 16; i++)
    for (j = 0; j < 8; j++) {
        A00[i][j] = B00[i][j] + 1;
        A01[i][j] = B01[i][j] + 1;
        A10[i][j] = B10[i][j] + 1;
        A11[i][j] = B11[i][j] + 1;
    }
(c) Transformed code
for(int i=0; i<8;i++)
for(int j=0;j<8;j++)
    CosTrans[j+8*i] = CosBlock[i+8*j];

for(int i=0; i<8;i++)
for(int j=0;j<8;j++) {
    TempBlock[i+j*8] = 0;
    for(int k=0;k<8;k++)
        TempBlock[i+j*8] += InIm[i+k*8] * CosTrans[k+j*8];
}

TEMPORAL PARTITIONING

Optimizations
Temporal Partitioning

• Mapping a complex application into a single device by time-multiplexing its resources

```c
for(int i=0; i<8; i++)
    for(int j=0; j<8; j++)
        CosTrans[j+8*i] = CosBlock[i+8*j];

for(int i=0; i<8; i++)
    for(int j=0; j<8; j++) {
        TempBlock[i+j*8] = 0;
        for(int k=0; k<8; k++)
            TempBlock[i+j*8] += InIm[i+k*8] * CosTrans[k+j*8];
    }
```
Temporal Partitioning
Temporal Partitioning

- List-Scheduling based
  - Allocation
    - function \( f_1 = \{1,3\} \)
    - function \( f_2 = \{2\} \)
    - memory \( M_1 = \{A,B\} \), single port

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f_1 )</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>( f_2 )</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

\[ \text{tp} \leftarrow 1 \]

1. Ready nodes sorted by priority (e.g., ALAP)

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9
Temporal Partitioning

• List-Scheduling based

1. Ready node sorted by priority (e.g., ALAP)

2. try to place ready nodes in tp (for each node placed find update ready nodes)

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

- List-Scheduling based

1. Ready nodes sorted by priority (e.g., ALAP)
2. Try to place ready nodes in tp (for each node placed find update ready nodes)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>C[1]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

1. Ready nodes sorted by priority (e.g., ALAP)

2. Try to place ready nodes in tp (for each node placed find update ready nodes)

Max FPG Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

$$tp ← 1$$

1. Ready nodes sorted by priority (e.g., ALAP)

2. Try to place ready nodes in tp (for each node placed find update ready nodes)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>7</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

2

3
Temporal Partitioning

- List-Scheduling based

\[ \text{tp} \leftarrow 1 \]

1. Ready nodes sorted by priority (e.g., ALAP)
2. Try to place ready nodes in \( \text{tp} \) (for each node placed find update ready nodes)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

1. Nodes ready sorted by priority (e.g., ALAP)

2. Try to place ready nodes in tp (for each node placed find update ready nodes)

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>8</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

1. Nodes ready sorted by priority (e.g., ALAP)
2. try to place ready nodes in tp (for each node placed find update ready nodes)
3. if none node fits, step to next temporal partition (tp ← tp+1)

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

- List-Scheduling based

\[ \text{tp} \rightarrow 1 \]

1. Ready nodes sorted by priority (e.g., ALAP)

2. try to place ready nodes in tp (for each node placed find update ready nodes)

3. if none node fits, step to next temporal partition (tp \rightarrow tp+1)

### Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

- List-Scheduling based

1. Ready nodes sorted by priority (e.g., ALAP)
2. try to place ready nodes in tp (for each node placed find update ready nodes)
3. if none node fits, step to next temporal partition (tp ← tp+1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Temporal Partitioning

- List-Scheduling based

\[ \text{tp} \leftarrow 1 \]
1. Ready nodes sorted by priority (e.g., ALAP)
2. Try to place nodes ready in \( \text{tp} \) (for each node placed find update ready nodes)
3. If none node fits, step to next temporal partition (\( \text{tp} \leftarrow \text{tp} + 1 \))

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1], B[1]</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>A[2], B[2]</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

1. Ready nodes sorted by priority (e.g., ALAP)
2. try to place ready nodes in tp (for each node placed find update ready nodes)
3. if none node fits, step to next temporal partition (tp ← tp+1)

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

1. Ready nodes sorted by priority (e.g., ALAP)
   - C[1]
2. try to place ready nodes in tp (for each node placed find update ready nodes)
3. if none node fits, step to next temporal partition (tp ← tp+1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Maximum FPGA Area = 9

<table>
<thead>
<tr>
<th>Temporal Partition (tp)</th>
<th>Operations</th>
<th>Area used</th>
<th>#cc</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A[1]</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>B[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>C[1]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td>A[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>B[2]</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Temporal Partitioning

• List-Scheduling based

1. Ready nodes sorted by priority (e.g., ALAP)
2. Try to place ready nodes in tp (for each node placed find update ready nodes)
3. If none node fits, step to next temporal partition (tp ← tp+1)

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>f1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Temporal Partitioning

- Best temporal partitioning depends on the target architecture

<table>
<thead>
<tr>
<th>Operation</th>
<th>Latency</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f_1$</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>$f_2$</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>load/store</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

2 tps $\Rightarrow$ 8 clock cycles +3 operands to communicate

3 tps $\Rightarrow$ 10 clock cycles + 2 operands to communicate
Temporal Partitioning

• By partitioning computations in time, each of the implementations is simpler,
  – which may lead to better performance/power/energy results;

• Amortize some of the configuration time,
  – by overlapping configuring and computing stages of subsequent configurations;

• Permit to use smaller area/devices (with lower-cost) to implement complex applications.
Optimizations

LOOP PIPELINING TECHNIQUES
Loop Pipelining (Software Pipelining)

- Overlap loop iterations
  - Subsequent iterations start before previous finished

- Performance increase

- Register pressure (not a main problem in array architectures with pipeline stages)

- Conservative
  - to reduce complexity
Loop Pipelining (Software Pipelining)

- Explicit Prologue, Kernel, and Epilogue

```c
for(int i=0; i<10; i++) { 
    C[i] = A[i] * B[i]; 
}

a_tmp = A[0];
b_tmp = B[0];
for(int i=0; i<9; i++) {
    C[i] = a_tmp * b_tmp;
    a_tmp = A[i+1];
b_tmp = B[i+1];
}
C[9] = a_tmp * b_tmp;
```

- Predicates (implicit existence of Prologue, Kernel, and Epilogue)

```c
for(int i=0; i<11; i++) {
    C[i-1] = a_tmp * b_tmp; if i!=0
    a_tmp = A[i]; if i!=10;
b_tmp = B[i]; if i!=10;
}
```
Loop Pipelining (Software Pipelining)

... for(i=0; i<10; i++) {
  a = A[i]; b = B[i];
  ab = a * b;
  C[i]= ab;
}

a = A[0], b = B[0];
ab = a * b, a = A[1], b = B[1];
for(int i=0; i<8; i++) {
  C[i]=ab, ab=a * b, a=A[i+1], b=B[i+1];
}
C[8] = ab, ab = a * b;
C[9] = ab;

Prologue: fill the pipe
Kernel: steady state
Epilogue: drain the pipe
Dependences in a Loop

• Two types of dependences:
  – Intra-iteration (inside the same iteration)
  – Inter-iteration (across distinct iterations)
• Recurrence manifests itself as a cycle in the dependence graph
• The *loop-carry* delay \(<d,p>\) from an instruction \(i\) to another instruction \(j\) implies that:
  – \(j\) depends on a value computed by instruction \(i\) \(p\) iterations ago, and
  – at least \(d\) cycles — denoting pipeline delays — must elapse after the appropriate instance of \(i\) has been executed before \(j\) can start
Loop Pipelining (Software Pipelining)

- **Initiation Interval (II)**
  - number of cycles between the start of successive iterations

- Each iteration can be divided into **stages** consisting of II cycles each

Only the kernel involves executing full width of operations

Prologue and epilogue execute a subset (ramp-up and ramp-down)
Loop Pipelining (Software Pipelining)

• **Minimum Initiation Interval (MII)** is a lower bound on the II
  - $\text{MII} = \text{Max}(\text{ResMII}, \text{RecMII})$
  - $\text{ResMII} =$ resource constrained MII
    - Resource usage requirements of 1 iteration
  - $\text{RecMII} =$ recurrence constrained MII
    - Latencies imposed by cycles in DDG

• **Iterative Modulo Scheduling (IMS)**
  - candidate II is initially set equal to the MII and increased (by 1) until a modulo schedule is obtained

Optimizations

TASK-LEVEL PIPELINING
Objectives

- **To speed-up** applications with multiple and **data-dependent stages**
  - each stage seen as a set of nested loops
- **How?**
  - **Pipelining** those sequences of data-dependent stages using fine-grain synchronization schemes
  - Taking advantage of field-custom computing structures (FPGAs)
Motivation

```c
for(i=0; i<N; i++) {
    ... tmp = ...;
    A[i] = tmp;
    ...
}
```

Diagram:
- Task A
- Task B

```
for(i=0; i<N; i++) {
    ...
    tmp2 = A[i];
    ...
}
```
Motivation

```c
for(i=0; i<N; i++) {
    ...
    tmp = ...;
    A[i] = tmp;
    ...
}
```

```
for(i=0; i<N; i++) {
    ...
    tmp = ...;
    send(tmp);
    ...
}
```

```
for(i=0; i<N; i++) {
    ...
    tmp2 = A[i];
    ...
}
```

```
for(i=0; i<N; i++) {
    ...
    tmp2 = receive();
    ...
}
```
k=0;
for(i=0; i<N; i++) {
    ...
    tmp = ...;
    if(i % 4) k++;
    A[k] = tmp;
    ...
}
for(i=0; i<N; i++) {
    ...
    tmp2 = A[i];
    ...
}

Task A

Task B

Motivation
Computing Stages

- Concurrently
  - Ordered producer/consumer pairs
    - Send/receive


FIFO with N stages


Diagram:
- FIFO with N stages
Computing Stages

• Concurrently
  – Unordered producer/consumer pairs
    • Empty/Full table

![Diagram of newline](image)
Main Idea

• FDCT
  – Out-of-order producer/consumer pairs
  – How to overlap computing stages?
Main Idea

- Pipelined
  - Data input
  - Loop 1
  - Loop 2
  - FSM 1
  - Intermediate data (dual-port RAM)
  - Dual-port 1-bit table (empty/full)
  - Loop 2
  - FSM 2
  - Loop 3
  - Execution of Loops 1, 2
  - Execution of Loop 3
  - Data output
  - Intermediate data array

Time

0 1 2 3 4 5 6 7
8 16 24 32 40 48 56

117
Main Idea

Task A

Task B

Memory

Memory

Memory
Compilation Steps

1. Identify computing stages
   - Sequences of loops are candidates
2. Determine inter-stage buffer access index function based on access patterns (producer and consumer)
3. Determine size of inter-stage buffer

Stages 2 and 3 are a difficult topic and only limited solutions are available
Experimental Results

- Speed-up over 1-core

- 2-core w/ PSL vs. 1-core
- ASA w/ PSL vs. 1-core
- ASA vs. 1-core
- Smooth + Sobel (d)
- Smooth + Sobel (c)
- RGB2gray vs. 1-core
- Task A
- Task B
- Task C
- Task M
- Task L
- Task N
- Task M

- Address: hit/miss
- Data: in/out

- Task A
- Task B
HARDWARE/SOFTWARE PARTITIONING
Hardware/Software Partitioning

• Given a software application
  – identify regions of code whose migration to software is required to accomplish certain goals (performance, energy consumption, N, etc.)

```java
// Number of array elements N
int N=4;

MAXTIME c1 = new MAXTIME();
c1.start();

int L2NORM = 0;
for(int i=0; i<N;i++) {
    short Aux = X[i] - Y[i];
    L2NORM += Aux*Aux;
}
c1.end();
```

```java
// Number of array elements N
int N=4;

// create a reconfigware object
Reconfigware RW1 = new Reconfigware("Vect_dist.cal");
// set the FPGA clock frequency to 20MHz
RW1.setClock(20);
// Give control of the memory of the board to the Host
RW1.setBankControl(0);

// send the arrays to the memory of the board
RW1.writeRAM(X, AddrX, N);
RW1.writeRAM(Y, AddrY, N);

// give control of the memory of the board to the FPGA
RW1.setBankControl(3);

RW1.setMapLow32(0xfffffffe); // reset the FSM
RW1.setColumn(RESET, 0); // start the FSM
RW1.setColumn(START, 1);

L2NORM = RW1.getColumn(RESULT); // return the control of the memory of the board to the host
RW1.setBankControl(0);
```
THE ROLE OF THE PROGRAMMING MODEL
The Role of the Programming Model

• As the programming model may help the programmer, it can also exacerbate the compilation difficulties
  — Especially when a large gap between the programming model and the computational model of the target architecture does exist!
The Role of the Programming Model

\[ y[k] = \sum_{n=0}^{N-1} x[k - n] \times c[n] \]

- Load/store
- Streaming

```c
#include "io_ports.h"
#define c0 2
#define c1 4
#define c2 4
#define c3 2
#define M 256
#define PORT_A 0x1
#define PORT_B 0x2

int c[N] = {c0, c1, c2, c3};

main() {
  int x_0, x_1, x_2, x_3, y;
  x_2 = receive(PORT_A);
  x_1 = receive(PORT_A);
  x_0 = receive(PORT_A);
  for(int j=0; j<M; j++) // while(1) {
    x_3 = x_2;
    x_2 = x_1;
    x_1 = x_0;
    x_0 = receive(PORT_A);
    y = c0 * x_0 + c1 * x_1 + c2 * x_2 + c3 * x_3;
    send(PORT_B, y);
  }
}
```
The Role of the Computation Model

• By implementing a computation model that reflects the programming model:
  – Mapping is more natural
  – It can be more efficient, but

• Not always:
  – See the example of compiling an imperative programming model to a von-Neumann architecture
    • Compilation is easier
    • Efficiency can be worser
The Role of Domain-Specific Languages (DSLs)

• More efficient optimizations as the computing descriptions embody well-kown behaviors
• Specific and fully optimized machines can be generated
• Example:
  – $a*b+|a+bc^*|ef$
  (a regular expression)
The Role of Domain-Specific Languages (DSLs)

- Example: $a^*b+|a+bc^*|ef$
- Using a common software language

```c
#define S0 0 // begin state
#define S1 1
#define ERR 6
#define NOT_ACCEPT 0
#define ACCEPT 1

// can be implemented as a table
int next_state[][7] = {
    {S2, S1, ERR, S4, S5, ERR, ERR},
    {ERR, ERR, ERR, ERR, ERR, ERR, ERR}};
int action[7] = {NOT_ACCEPT, ACCEPT, NOT_ACCEPT, ACCEPT, NOT_ACCEPT, ACCEPT, NOT_ACCEPT};

// MAIN
int current_state = S0;
while(s=get_char()) {
    int index = char_map_to_col(s);
    current_state = next_state[current_state][index];
}
int is_match = action[current_state];
```
The Role of Domain-Specific Languages (DSLs)

- Example: $a^*b^+|a+bc^*|ef$
- Recognize directly the behavior and map it to an optimized specific architecture for regular expressions

Previous collaboration with: João Bispo, Yiannis Sourdis, Stamatis Vassiliadis (TUD, The Netherlands)
The Role of Domain-Specific Languages (DSLs)

- Example: \(a^*b+|a+bc^*|ef\), and \(a+bc^*\)
TRENDS
Current Status on Compilation: Limited Success

- **Hard** to achieve results comparable to manually designed solutions

Large gap!

Huge gap!

Diagram:

1. Application
2. Software to HDL Compiler
3. HDL
4. Mapping Techniques
5. Input representation for low-level targeting tools
6. Low-level-vendor Specific Tools
7. Reconfigurable Fabric

Automatic

Input representation for low-level targeting tools

Repository of Hardware Components

Huge gap!
Our Vision

Successive refinement
Refactoring, Code Transformations, Rewriting
Machine Learning

Help to diminish the gap

1. Application
2. Complementary Specification (Aspects and User-Knowledge)
3. Transformation Engine
4. Mapping Techniques
   - Input representation for low-level targeting tools
   - Low-level-vendor Specific Tools
5. Repository of Best Practices
6. Repository of Templates (patterns)
7. Reconfigurable Fabric
8. DSL

Automatic
CONCLUSIONS
Conclusions

• There is still plenty of opportunities to improve reconfigurable architectures and tools (including compilers)

• But, we need ASAP an easier way to program Reconfigurable Computing Architectures
  — The difficulties in programming these architectures
    • are preventing progress and innovation
    • are preventing a more widespread acceptance

• Future applications will continue to give interesting opportunities for reconfigurable computing
João M. P. Cardoso
jmpc@acm.org
http://www.fe.up.pt/~jmpc

THANK YOU!