Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays

Frank Hannig

hannig@cs.fau.de
Overview

• Motivation

• Architecture

• Mapping methodology

• Current and future work
Motivation

System-on-a-Chip (SoC)
Parallel Architectures’ Trade-offs

- Coarse-grained reconfiguration data is one to two orders of magnitude smaller than fine-grained
Architecture

Tightly-coupled processor arrays

• Coarse-grained and weakly-programmable
• Highly parameterizable architecture template
• Reconfigurable interconnect
Multicast Reconfiguration Scheme

- **Vertical Mask Register (V):**
  - Bit #2: 0
  - Bit #1: 1
  - Bit #0: 0
  - Columns: #0, #1, #2

- **Horizontal Mask Register (H):**
  - Bit #2: 0
  - Bit #1: 0
  - Bit #0: 0
  - Rows: #0, #1, #2

- **Multicast Signatures:**
  - V: 0 1 1
  - H: 1 1 0

- **Configuration Bit:**
  - Depending on the configuration, different multicast signatures can be used for reconfiguration.
Design flow, based on the PARO HLS tool

Algorithm (PAULA)

High-Level Transformations
- Localization
- Loop Perfectization
- Output Normal Form
- Loop Unrolling
- Partitioning
- Expression Splitting
- Affine Transformations
- ...

Space-Time Mapping
- Allocation
- Scheduling
- Resource Binding

Code Generation
- VLIW Code for each PE
- Configuration of Interconnect
- Code of Controller

Hardware Synthesis
- Processor Element
- Processor Array
- Controller
- I/O Interface
- HDL Generation

Test Bench Generation

WPPA Configuration

Architecture Model

Simulation

Hardware Description (VHDL)

FPGA

Simulation
Design flow, based on the PARO HLS tool

1. **Algorithm (PAULA)**
   - High-Level Transformations
     - Localization
     - Loop Perfectization
     - Output Normal Form
     - Loop Unrolling
     - Partitioning
     - Expression Splitting
     - Affine Transformations
     - ...  

2. **Space-Time Mapping**
   - Allocation
   - Scheduling
   - Resource Binding

3. **Code Generation**
   - VLIW Code for each PE
   - Configuration of Interconnect
   - Code of Controller

4. **Architecture Model**

5. **Hardware Synthesis**
   - Processor Element
   - Processor Array
   - Controller
   - I/O Interface
   - HDL Generation

6. **Hardware Description (VHDL)**
   - FPGA Simulation

7. **Test Bench Generation**

8. **Simulation**
Why not starting from C-code?

- Most existing high-level synthesis tools start from C/C++ code

- Limitation: Semantics of input language (statement order, loop order) define execution order \(\Rightarrow\) limited parallelism

Example:

```c
int s = 0;
for (i=0; i<=7; i++) { s += a[i]; }
```

- Tools have only few high-level transformations to parallelize code
Design entry: PAULA language

Key features:

• Functional programming language
• Full static single assignment (SSA) form, also for multidimensional arrays
• Powerful expressions for the specification of polyhedral and lattice iteration domains
• Convenient usage of reductions like $\sum$
• Architectural modeling capabilities
PAULA and intermediate representation

Extended reduced dependence graph (RDG):

Example, 2-D Gauss window filter:

\[
\begin{align*}
\text{par} & \ (x \geq 0 \ \text{and} \ x < 1280 \ \text{and} \ y \geq 0 \ \text{and} \ y < 1024) \\
& \{ \ w[0,0]=1; \ w[0,1]=2; \ w[0,2]=1; \\
& \quad \ w[1,0]=2; \ w[1,1]=4; \ w[1,2]=2; \\
& \quad \ w[2,0]=1; \ w[2,1]=2; \ w[2,2]=1; \\
& \quad \ h[x,y]=\text{SUM}[i=0 \ \text{and} \ i=2 \ \text{and} \ j=0 \ \text{and} \ j=2] \\
& \quad \quad \quad (\text{pic}\_\text{in}[x+i,y+j] \ * \ w[i,j]); \\
& \quad \text{pic}\_\text{out}[x,y]=h[x,y] >> 4; \ // \text{divided by 16}
\}
\end{align*}
\]
Design flow

Algorithm (PAULA)

High-Level Transformations
- Localization
- Loop Perfectization
- Output Normal Form
- Loop Unrolling
- Partitioning
- Expression Splitting
- Affine Transformations
- ...

Simulation

Space-Time Mapping
- Allocation
- Scheduling
- Resource Binding

Code Generation
- VLIW Code for each PE
- Configuration of Interconnect
- Code of Controller

Architecture Model

Hardware Synthesis
- Processor Element
- Processor Array
- Controller
- I/O Interface
- HDL Generation

Test Bench Generation

Simulation

FPGA
Localization

• Example of one-dimensional localization

```c
for (i=0; i <= N; i++)
{ b[i] = a[0];
}
```

• Example of two-dimensional localization

```c
for (i=0; i <= N; i++)
{ if (i > 0) a[i] = a[i-1];
    b[i] = a[i];
}
```
Design flow

**Algorithm (PAULA)**

**High-Level Transformations**
- Localization
- Loop Perfectization
- Output Normal Form
- Loop Unrolling
- Expression Splitting
- Partitioning
- Affine Transformations
- ...

**Simulation**

**Space-Time Mapping**
- Allocation
- Scheduling
- Resource Binding

**Code Generation**
- VLIW Code for each PE
- Configuration of Interconnect
- Code of Controller

**WPPA Configuration**

**Hardware Synthesis**
- Processor Element
- Processor Array
- Controller
- I/O Interface
- HDL Generation

**Hardware Description (VHDL)**

**Simulation**

**Front End**

**Back End**

**Test Bench Generation**

**Simulation**
Space mapping / partitioning

- LSGP partitioning

Main advantage:
Place & route for free! Since the space mapping directly defines the placement.
Space mapping / partitioning

- LPGS partitioning
Space mapping / partitioning

- Hierarchical partitioning

dependence graph

- Balancing of: Communication, I/O and different levels of (local) memory
- Adaptation of bandwidth, computational power, and memory constraints
Scheduling

• Features:
  – Resource constraints (number of functional units)
  – Module selection (different binding possibilities of operations)
  – Functional and software pipelining
  – Some part can be concurrently executed
    other have to be serialized
  – Exact approach based on mixed integer linear programming (MIP)

• Goal, simultaneous optimization of:
  – Local schedule (execution order within PEs) and
  – Global schedule (execution between all PEs)
Example, Median Filter

- **PAULA program of horizontal median filter**
  
  ```c
  par (x >= 0 and x < 1921 and y >= 0 and y < 1080) {
    p[x,y] = pi[x,y] if (x<1920);
    m[x,y] = 0 if (x==0);
    m[x,y] = p[x-1,y] if (x==1);
    m[x,y] = median(p[x,y],p[x-1,y],p[x-2,y]) if (x>1 and x<1920);
    m[x,y] = p[x-1,y] if (x==1920);
    po[x-1,y] = m[x,y] if (x>=1);
  }
  ```

- **Partitioned algorithm (4 stripes / processors)**
  
  ```c
  par (x>=0 and x<481 and y>=0 and y<1080 and z>=0 and z<=3) {
    p[x,y,z] = pi[x,y,z] if (x<480);
    m[x,y,z] = 0 if (x==0);
    m[x,y,z] = p[x-1,y,z] if (x==1 and z==0);
    m[x,y,z] = p[x+479,y,z-1] if (x==1 and z>0);
    m[x,y,z] = median(p[x,y,z],p[x-1,y,z],p[x-2,y,z]) if (x>1 and x<480);
    m[x,y,z] = p[x-1,y,z] if (x==480);
    po[x-1,y,z] = m[x,y,z] if (x>=1);
  }
  ```
Code Generation, Median Filter

- Substitution of median-function by explicit comparisons

```c
par (x>=0 and x<481 and y>=0 and y<1080 and z>=0 and z<=3)
{
  S0: p[x,y,z] = pi[x,y,z] if (x<480);
  S1: m[x,y,z] = 0 if (x==0);
  S2: m[x,y,z] = p[x-1,y,z] if (x==1 and z==0);
  S3: m[x,y,z] = p[x+479,y,z-1] if (x==1 and z>0);
  S4: C1[x,y,z] = (p[x,y,z]>p[x-1,y,z]) if (x>1 and x<480);
  S5: C2[x,y,z] = (p[x-2,y,z]>d[x,y,z]) if (x>1 and x<480);
  S6: C3[x,y,z] = (f[x,y,z]>e[x,y,z]) if (x>1 and x<480);
  S7: d[x,y,z] = ifrt(C1[x,y,z],p[x,y,z],p[x-1,y,z])
                  if (x>1 and x<480);
  S8: e[x,y,z] = ifrt(C1[x,y,z],p[x-1,y,z],p[x,y,z])
                  if (x>1 and x<480);
  S9: f[x,y,z] = ifrt(C2[x,y,z],d[x,y,z],p[x-2,y,z])
                  if (x>1 and x<480);
  S10: m[x,y,z] = ifrt(C3[x,y,z],f[x,y,z],e[x,y,z])
                     if (x>1 and x<480);
  S11: m[x,y,z] = p[x-1,y,z] if (x==480);
  S12: po[x-1,y,z] = m[x,y,z] if (x>=1);
}
```
Code Generation, Median Filter

- VLIW code fragment for one processor

<table>
<thead>
<tr>
<th>Line</th>
<th>ADD0</th>
<th>ADD1</th>
<th>BRANCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NOP</td>
<td>NOP</td>
<td>IF IC0,IC1 JMP 1, 2, 3, -</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>1</td>
<td>MOV RD2,IN1</td>
<td>MOV RD3,RF1</td>
<td>IF IC0 JMP 4, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>2</td>
<td>MOV RD2,IN1</td>
<td>NOP</td>
<td>IF IC0 JMP 4, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>3</td>
<td>NOP</td>
<td>MOV RD3,RF1</td>
<td>IF IC0 JMP 4, 5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>4</td>
<td>CMP RC1,RD2,RD3</td>
<td>MOV OUT1,RD5</td>
<td>IF IC0,RC2 JMP 6, 7, 8, 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>5</td>
<td>CMP RC1,RD2,RD3</td>
<td>NOP</td>
<td>IF IC0,RC2 JMP 6, 7, 8, 9</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td>6</td>
<td>MOV RF2,RD2</td>
<td>MOV RD1,RD6</td>
<td>IF IC0,RC1 JMP 10,11,12,13</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>--</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Zero loop overhead: Run control flow completely in parallel with data flow
Current and future work

• Currently:
  – Static mapping, compilation for one fixed architecture allocation (no. of PEs, etc.)
  – Static partitioning and scheduling
  – If other resource constraints are given or different requirements are needed, program has to compiled again

• Idea: *Symbolic loop parallelization and mapping*
  – Symbolic partitioning
  – Symbolic scheduling
  – Symbolic control generation

  \[ \Rightarrow \]
  * Replace at run-time only the symbols in the configuration data according to the available resources
  * Algorithms can be adapted quickly at run-time without recompilation
  * Self-adaption enables fast reaction on QoS parameters, system load, failures, etc.
Retargetable Mapping of Loop Programs on Coarse-grained Reconfigurable Arrays

Frank Hannig
Hardware/Software Co-Design
Department of Computer Science
University of Erlangen-Nuremberg
Am Weichselgarten 3
91058 Erlangen, Germany

Phone: + 49 9131 85-25153
Fax: + 49 9131 85-25149
Email: hannig@cs.fau.de
URL: http://www12.cs.fau.de

Acknowledgements
Hritam Dutta, Dmitrij Kissler, Alexey Kupriyanov,
Vahid Lari, Holger Ruckdeschel, Jürgen Teich

This work was partially supported by the German Research Foundation (DFG)
in projects under contracts TE 163 /3-1, TE 163 /3-2, and SFB TRR 89.