High-Level Programming of Reconfigurable Systems

By: Zain-ul-Abdin, Ph.D.
Outline

• High-Performance Embedded Computing Challenges
• Coarse-Grained Reconfigurable Architectures
  – Ambric
  – PACT XPP
• Occam-pi language & Compiler
• Case Studies
  – SAR Autofocus Criterion Calculation
  – Implementation Results
• Summary & Future Work
High-Performance Embedded Computing Challenges

Requirements:
- High Performance
- Energy Efficiency
- Adaptivity

Evolution of Embedded Computer Architectures

- **Microprocessors (µP)**
- **Arrays of Functional Units (FUs)**
- **Programmable Logic Arrays (PLA)**

**Adaptability to Application**

**Programmability**

**Granularity**

**UniProc**

**Multicore**

**Reconfigurability**

µP → Microprocessors
FUs → Functional Units
PLA → Programmable Logic Array

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Coarse-Grained Reconfigurable Architectures

• Classification
  – Hybrid architectures
  – Array of functional units
  – Array of processors
  – Array of soft processors
Coarse-Grained Reconfigurable Architectures

- Classification
  - Hybrid architectures
  - Array of functional units
  - Array of processors
  - Array of soft processors
Ambric Platform

- 45 brics (360 processors) @333 Mhz
- Peak performance > 1 TOPS
- 117 million transistors
- 11 W max power
Coarse-Grained Reconfigurable Architectures

- Classification
  - Hybrid architectures
  - Array of functional units
  - Array of processors
  - Array of soft processors
XPP Architecture

- 8x8 ALU-PAEs @ 64 Mhz
- 16 RAM-PAEs
- 24/32-bit data bus

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
High-Performance Embedded Computing Challenges

• Parallel Programming Models
  – Handling of concurrency and synchronization
  – Expose dynamic parallelism
  – Runtime-reconfigurability management
Traditional Approaches

• Sequential languages (C, Pascal)
  – Rely on sequential control flow
  – Intended for algorithm specification
  – Use annotations to adapt to target architecture
  – Sequential programs obscure too much information

• Traditional methods
  – Automatic parallelization by compilers
  – Parallelizing compilers have achieved close to technical perfection, but are not enough

• Lack of support for expressing dynamic reconfiguration
Our Approach

- Use of DSLs for Application development
- Occam-pi as an Intermediate language
  - CSP dataflow
  - Mobility features of pi-calculus
  - Expression of Reconfigurability
  - Supported by a compiler for allowing portability
Processes

• The simplest form of interaction is *synchronised* message-passing along *channels*.

• The simplest forms of channel are *zero-buffered* and *point-to-point* (i.e. *wires*).
Structured Processes - SEQ and PAR

SEQ

in ? sum
in ? x
sum := sum + x
out ! Sum

PAR

in0 ? a
in1 ? b
out ! a + b
c := a + (2*b)

Processes can run in any order, No data race hazards

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Structured Processes-
PROC instance

PROC octople (CHAN INT in?, out!)
CHAN INT a, b:
PAR
double (in?, a!)
double (a?, b!)
double (b?, out!)

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Copy Semantics

• **Classical occam:** data is *copied* from the workspace of $A$ into the workspace of $B$. Subsequent work by $A$ on its $x$ variable and $B$ on its $y$ variable causes no mutual interference.
Mobile Data & Channels

- Mobiles are safe references
- Assignment and communication with \textit{Movement} semantics
- Only one process may hold a given mobile
Movement Semantics

Before

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Movement Semantics

A \xrightarrow{c} B

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Dynamic Process Invocation

\[
\text{PROC A} () \\
\quad \ldots \text{local state} \\
\text{SEQ} \\
\quad \ldots \\
\text{FORKING} \\
\quad \text{SEQ} \\
\quad \ldots \\
\text{FORK} \ p(n, \text{svr, cli}) \\
\quad \ldots \\
\quad : \\
\]

VAL data is \textit{copied} into a FORKed process

MOBILE data and channel-ends are \textit{moved} into a FORKed process
Occam-pi to CGRA Compilation

![Diagram showing the compilation process from Occam-pi to CGRA.]

- **Occam-pi Code**
- **Occam Frontend**
  - ParseOccam
- **AST**
- **Transformations**
  - SimplifyTypes
  - SimplifyExpr
  - SimplifyProcs
  - Unnest
- **Ambric Backend**
  - GenerateSOPM
  - Ambric aStruct, aJava, assembly
- **XPP Backend**
  - GenerateXPP
  - NML Code
- **NML module Generator**
- **Interconnection code Generator**
- **Application section code Generator**

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Outline

• High-Performance Embedded Computing Challenges
• Coarse-Grained Reconfigurable Architectures
  – Ambric
  – PACT XPP
• Occam-pi language & Compiler
• Case Studies
  – SAR Autofocus Criterion Calculation
  – Implementation Results
• Concluding Remarks
• Future Work
Autofocus Criterion Calculation

\[
|f(r, \phi)|^2 \ast |f_\ast(r, \phi)|^2
\]

(incl. interpolations, e.g. NN or cubic)

Focus criterion \sim \sum |f(r, \phi)|^2 \ast |f_\ast(r, \phi)|^2

calculate for different shifts
Cubic Interpolation in Autofocus Criterion Calculation

- samples in contributing data set
- range-interpolated data
- final interpolated, contributing data

a) one interpolation

b) several interpolations

“High-Level Programming of Reconfigurable Systems”, Zain-ul-Abdin
Autofocus Criterion Calculation - Dataflow diagrams

Design-I

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Auto-focus Criterion Calculation

Dataflow diagrams

Design I

No. of SRDs = 71/168

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Autofocus Criterion Calculation - Dataflow diagrams

Design-II

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Autofocus Criterion Calculation - Dataflow diagrams

Design-II

No. of SRDs = 141/168

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
## Implementation Results-Ambric

### Resource Consumption Results

<table>
<thead>
<tr>
<th></th>
<th>SRDs</th>
<th>SRs</th>
<th>RU Banks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Am2045 Full Capacity</td>
<td>168</td>
<td>168</td>
<td>336</td>
</tr>
<tr>
<td>Design-I on Ambric</td>
<td>70 (42%)</td>
<td>24 (14%)</td>
<td>113 (34%)</td>
</tr>
<tr>
<td>Design-II on Ambric</td>
<td>141 (84%)</td>
<td>28 (17%)</td>
<td>208 (62%)</td>
</tr>
</tbody>
</table>

### Performance & Estimated Power Results

<table>
<thead>
<tr>
<th>Implementations</th>
<th>Latency (cycles)</th>
<th>Throughput (pixels/sec.)</th>
<th>Speedup Throughput</th>
<th>Power (Watts)</th>
<th>Throughput/W (pixels/W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sequential on Intel i7</td>
<td>-</td>
<td>21,600</td>
<td>1</td>
<td>17.5</td>
<td>1234</td>
</tr>
<tr>
<td>Design-I on Ambric</td>
<td>16,497</td>
<td>236,386</td>
<td>11</td>
<td>6.52</td>
<td>36,255</td>
</tr>
<tr>
<td>Design-II on Ambric</td>
<td>12,793</td>
<td>486,224</td>
<td>23</td>
<td>9.8</td>
<td>49,614</td>
</tr>
</tbody>
</table>
Implementation Results-XPP
16-tap Integer FIR

<table>
<thead>
<tr>
<th>Resource Usage (No. of ALUs)</th>
<th>Throughput (million samples/sec.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>XPP-VC</td>
<td></td>
</tr>
<tr>
<td>NML</td>
<td></td>
</tr>
<tr>
<td>Occam-pi II</td>
<td></td>
</tr>
<tr>
<td>Occam-pi I</td>
<td></td>
</tr>
</tbody>
</table>

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Implementation Results-XPP
1D-DCT

Resource Usage
(No. of ALUs)

Throughput
(million samples/sec.)

NML
Occam-pi ★
(Reconfigurable)

Occam-pi
XPP-VC

"High-Level Programming of Reconfigurable Systems", Zain-ul-Abdin
Summary

• Identified the significance of Coarse-Grained Reconfigurable Arrays
  – Based on GALS principle to overcome long interconnect delay
  – Energy efficient
• Simplified development based on concurrent programming model of Occam-pi
  – Raises the abstraction level while not compromising the performance
  – Source-to-source transformations enables target-specific optimizations
  – Able to express dynamic reconfiguration with a formal basis
Future Work

- Developing more large-scale case-studies for proof of the concept
- Extend the compiler framework to target other parallel architectures such as GPGPUs and heterogeneous systems
- Development of frontend for DSLs to complete the design flow
Thank you for your attention!