KAHRISMA: A Multi-grained Reconfigurable Multicore Architecture

Lars Bauer, Ralf König, Muhammad Shafique, Timo Stripf, Waheed Ahmed, Juergen Becker, Jörg Henkel
Talk Outline

- Motivation
- KAHRISMA Architecture Template
- Compile-Time Software Framework
- Run-Time System
- Case Study
- Summary & Conclusion
Motivation

- Increasing diversity of embedded applications
- More complex functionalities integrated in (mobile) devices
- Applications’ characteristics
  - Hardly predictable
    - Characteristics of upcoming applications
    - Combinations of applications running in parallel
  - Diverse processing behavior
    - Control flow vs. ILP, DLP

- Crucial design decisions can no longer be determined or even fixed at design time
- Runtime adaptive architectures show to be good candidates to provide the required flexibility
### State of the Art SoC – Limitations

<table>
<thead>
<tr>
<th>Software</th>
<th>Hardware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application I - Domain 1</td>
<td>Heterogeneous MPSoC</td>
</tr>
<tr>
<td><strong>RISC</strong>₁</td>
<td><strong>IP 1</strong></td>
</tr>
<tr>
<td><strong>CI ₂₁</strong></td>
<td><strong>RISC</strong></td>
</tr>
<tr>
<td><strong>CI ₁₁</strong></td>
<td><strong>Mem</strong></td>
</tr>
<tr>
<td>Application II - Domain 1</td>
<td><strong>IP 2</strong></td>
</tr>
<tr>
<td><strong>RISC</strong>₂</td>
<td></td>
</tr>
<tr>
<td><strong>CI ₂₂</strong></td>
<td><strong>IP n</strong></td>
</tr>
</tbody>
</table>

#### Potential Problem:
- If scenario has not been considered at compile time, both threads could claim same resources (IP 1) due to static mapping
  - either one of the threads might be executed
State of the Art SoC – Limitations

Software

Hardware

Problem:
- IP cores can not be reconfigured to realize additional RISC instances, VLIW instances, or ‘Cl m’ resources
- Applications might either not be executed at all or might not be able to fulfill QoS requirements
Reconfigurable Architectures – Classification

**Fine-grained**
- Based on bit-level Lookup Tables

**Coarse-Grained**
- Based on word-level ALUs

src: Xilinx

src: ADRES, IMEC
Reconfigurable Accelerators – Examples

Fine-grained
- Based on bit-level Lookup Tables

Coarse-Grained
- Based on word-level ALUs
Reconfigurable Architectures – Short Overview

- There is a large variety of reconfigurable processors
  - **FG:** PRISC, DISC, OneChip, Chimaera, XiRISC, RISPP, …
  - **CG:** ADRES, CCA, Montium, PACT XPP, PipeRench, …
- Typically: Domain specific component selection (FG or CG)
  - Less efficient for applications of different computational domains
- Combining both types (e.g. Morpheus or 4s)
  - Up to now: only loose coupling of architecture components

- No approach provides bit-level as well as word-level acceleration in a tightly-coupled adaptive architecture
  - Additionally, not sufficient adaptivity to provide RISC and/or VLIW ISAs on demand
  - Important when targeting ILP/DLP rich as well as TLP-based applications
Talk Outline

- Motivation
- KAHRISMA Architecture Template
- Compile-Time Software Framework
- Run-Time System
- Case Study
- Summary & Conclusion
KAHRISMA – Novel Contributions

- Architectural concept
  - Variety of reconfigurable modules realizing basic functionalities that dynamically can be combined
  - Tightly integrated coarse- and fine-grained reconfigurable data processing fabrics
    - Coarse-Gained and Fine-Grained Encapsulated Datapath Element (CG-, FG-EDPE)
  - On demand realization of different Instructions Set Architectures (ISA) as well as hardware-accelerators

- Compile-time software framework
  - ADL-based, retargetable

- Run-Time System
  - exploiting the high degree of parallelism
The KAHRISMA Architecture – Overview

1. Instruction Fetch & Align: Cache access, extraction of the actual instruction packets
   - Dispatching of instr. to EDPEs
   - Flow-Control, handling of Interrupts, Exceptions etc.

2. Instr. Analyze & Dispatch: Extraction of the individual Operations out of an instruction packet
   - Dispatching of instr. to EDPEs
   - Flow-Control, handling of Interrupts, Exceptions etc.

3. EDPE Array:
   - Realization of typical ALU operations with DSP optimizations

4. Memory Subsystem:
   - Load/Store Units
   - Multibanked cache
EDPE Array: Implementing ISAs and hardware accelerators

- FG-EDPEs are FPGA-like reconfigurable fabrics, optimized for bit/byte level operations, state machines etc.
- CG-EDPEs are ALU-like reconfigurable fabrics, optimized for word/sub-word level operations.
The KAHRISMA Architecture – Appl. Scenario
The KAHRISMA Architecture – Appl. Scenario (cont’d)
Hypermorphism:
Dynamically combining the reconfigurable modules to realize different ISAs as well as Custom Instructions (CIs) upon application requirements

KArlsruhe’s Hypermorphic Reconfigurable Instruction-Set Multigrained Array
Talk Outline

- Motivation
- KAHRISMA Architecture Template
- Compile-Time Software Framework
- Run-Time System
- Case Study
- Summary & Conclusion
Design/Compile-Time Tools

- **Integrated tool chain**
  - to simplify programming
  - to optimally exploit a flexible, heterogeneous architecture

- **Substantial features**
  - ADL based
    - Specification of Coarse-Grained (CG) & Fine-Grained (FG)-EDPEs and interconnect network
  - Automated application partitioning on code level:
    - Processor Mode vs. Array-Mode
      - Processor Mode: selection of appropriate ISA
      - Array Mode: Custom Instruction (CI) detection and mapping
  - Simulator
    - Supporting application development
    - Design-Space exploration
Compile-Time Toolflow

- **Tool input**
  - architecture description, code
- **Machine independent optimizations**
  - Dead code elimination, constant propagation, inlining etc.
- **CI Identification**
  - Array Mode implementation of computational hot-spots
- **Compilation Backend**
  - Code partitioning on ISA level
- **Binary Utilities**
  - ISA assembler, coarse-, fine- and mixed-grained linking
CI Identification, Selection, and Mapping

- **Automatic CI Identification**
  - Identification of possible CIs for mapping to FG- and/or CG-EDPEs

- **Hierarchical Composition**
  - Decomposition of the CI graph
  - Selection of beneficial CIs
  - Creation of multiple implementation versions

- **Mapping of CIs**
  - Architecture dependent
  - Configuration emission

- **Library construction**
  - Faster identification and selection of created CIs for other applications

---

For a detailed view of the diagram, please refer to the slide content.
Compilation Backend

- Extension of classical backend
  - Each pass relies on information from ADL
  - Identification of control- vs. data-flow rich application sections
  - Selection of appropriate ISA
    - RISC for control-flow rich application sections
    - \( n \)-issue VLIW for data-flow rich application sections

- Emission of implementation alternatives
  - Fastest execution
  - Lowest hardware utilization
  - Variants in-between
Talk Outline

- Motivation
- KAHRISMA Architecture Template
- Compile-Time Software Framework
- Run-Time System
- Case Study
- Summary & Conclusion
Run-Time System – Tasks

- Selection of most suited candidate from (compile-time prepared) CI implementation alternatives
  - Ranging from a software implementation up to a fast hardware implementation
  - Regarding the system status (battery level, user-defined performance constraints, current EDPE utilization, …)

- Binding to physical resources
  - Establish configurations on the architecture
Run-Time System – Overview

1. Request of new thread/application
2. Selection of implementation alternatives (ISAs and/or CIs)
3. Binding of required instances to physical resources

- Steps 2 and 3 are assisted by online monitoring
- Persistent online optimization
Run-Time System – Overview (cont’d)

- Online Monitoring
  - Reconfiguration requires time
  - Prediction of execution frequencies, threads deadlines, and system states
  - Look ahead reconfiguration of processor resources to start computation as early as possible (prefetching)
  - Persistent online optimization, e.g. array defragmentation

- Selection
  - Considering priorities and deadlines of the currently executing threads
  - Considering communication requirements – spatially adjacent EDPEs, locality to memory
  - Considering reconfiguration time of different modules

- Binding
  - Based on a hierarchical approach: determine a region before determining EDPEs within that region
Talk Outline

- Motivation
- KAHRISMA Architecture Template
- Compile-Time Software Framework
- Run-Time System
- Case Study
- Summary & Conclusion
Case Study

- Security domain: AES
- Multimedia domain: Deblocking Filter, CABAC
- Mapping of fine-/coarse-/multi-grained CIs has been done manually
  - Will serve as later comparison for the tool chain
- First implementation results for CG-EDPE
  - 90nm TSMC - tcbn90gwc
  - Speed: 500 MHz
  - Area: 536,007 \( \mu \text{m}^2 \)
- FG-EDPE clock derived from CG-EDPE clock
  - Clock ratio CG-EDPE/FG-EDPE: 5/1
  - FG-EDPE Frequency: 100 MHz
Advanced Encryption Standard (AES)

- Symmetrical Block Cipher
- Fixed Data Block Size: 128 Bit
- Variable Key Block Size: 128, 192, 256 Bit
- Depending on the Key Block Size, the inner loop will be executed
  - 9 x – 128 Bit
  - 11 x – 192 Bit
  - 13 x – 256 Bit

AddRoundKey (ARK)

SubBytes (SB)

ShiftRows (SR)

MixColumns (MC)

KeyExpansion (KE)

AddRoundKey

SubBytes

ShiftRows

AddRoundKey

loop iterations - key size (bit): 9-128, 10-192, 13-256

final round

initialization round
AES – Single EDPE CI Implementations

- **Processing time**
  - 1 FG-EDPE: 550 Cycles
  - 1 CG-EDPE: 332 Cycles
  - Note: All execution times are given for the 500 MHz clock for the CG-EDPEs

**Ressource Requirements:**

<table>
<thead>
<tr>
<th>1 CG-EDGE</th>
<th>1 FG-EDGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARK</td>
<td>ARK</td>
</tr>
<tr>
<td>SB</td>
<td>SB</td>
</tr>
<tr>
<td>SR</td>
<td>SR</td>
</tr>
<tr>
<td>MC</td>
<td>MC</td>
</tr>
<tr>
<td>KE</td>
<td>KE</td>
</tr>
<tr>
<td>ARK</td>
<td>ARK</td>
</tr>
</tbody>
</table>

- **Stage 1:**
  - AND: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF
  - OR: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF

- **Stage 2:**
  - AND: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF
  - OR: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF

- **Stage 3:**
  - AND: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF
  - OR: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF

- **Stage 4:**
  - AND: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF
  - OR: 0xFF000000, 0x00FF0000, 0x0000FF00, 0x000000FF

KP: XOR
RESULT: XOR
AES – Multi EDPE CI Implementations

- Processing time
  - 2 CG-EDPEs: 208 Cycles
  - Multi-Grained (2 CG, 1 FG): 163 Cycles
AES – Area-Performance Design Space

- RISC vs. multiple CI implementations

**Performance [Cycles]**

- **Fastest CI Version** is with 1 FG-EDPE and 2 CG-EDPE
- **RISC execution is the slowest**

KAHRISMA Run-Time System selects most appropriate alternative from the Area-Performance Design Space at run-time
Summary & Conclusion

- Presented a novel reconfigurable MPSoC concept
- Different reconfigurable modules realize basic functionalities that are combined dynamically
- On demand realization of different ISAs as well as CIs
- Tightly integrated coarse- and fine-grained reconfigurable fabrics

- Constituting KAHRISMA’s Hypermorphism
- Provides different levels of parallelism, from ILP and DLP up to TLP
- Allows to match the applications’ requirements much better
- Provides very high adaptivity and thus allows optimizing the working point at run time according to the application requirements
KAHRISMA: A Multi-grained Reconfigurable Multicore Architecture

Lars Bauer, Ralf König, Muhammad Shafique, Timo Stripf, Waheed Ahmed, Juergen Becker, Jörg Henkel

Thank you for your attention!