

Reconfigurable Architectures and Instruction Sets

Programmability, Code Generation, and Program Execution

Rainer Buchty

Universität Karlsruhe (TH) – Forschungsuniversität
Institut für Technische Informatik (ITEC)
Lehrstuhl für Rechnerarchitektur und Parallelverarbeitung

4. April 2006

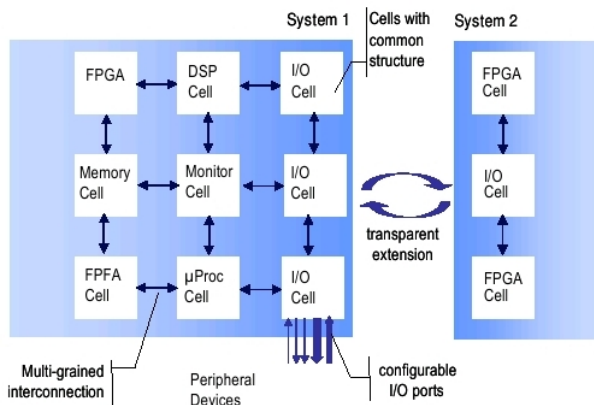
Promises of Reconfiguration

- Computation Efficiency
 - On-the-fly Hardware Acceleration
 - Dedicated Instructions
 - Building Blocks
- Economic Efficiency
 - Re-Use of Silicon Resources
 - On-the-fly adoption
 - No more “one size fits it all”
- Power Efficiency
 - Silicon Reuse
 - Hardware Acceleration
 - Power vs. Performance
 - Adoption to current needs

- Early architectures as proof of concept
- Reconfigurable hardware exists
 - FPGAs support total, partial, and dynamic partial reconfiguration
 - FPGA-vendor tool support still weak, but getting better
- So far, reconfiguration works fine for special, manually adopted applications, but...
- ...there are problems to solve
 - Reconfigurable Instruction Sets vs. Program Generation
 - Exchanging Software and Hardware
 - System Reconfiguration vs. Application Demands
 - Exploitation of inherent parallelism
 - Getting a uniform application description
 - Communication within the architecture

Digital on-demand Organism (DodOrg)

- Funded by DFG through SPP1183 “Organic Computing”
- Apply biological concepts to Computer Architecture
- HW: Tiled Architecture



- 1 Reconfigurable Instruction Sets
 - Arising Problems
 - Achieving compatibility
- 2 Granting Application Requirements
 - Reconfiguration might violate restrictions
 - Define and adhere to requirements
- 3 Example Implementation
- 4 Programming and Implementation
 - Application Programming
 - A possible solution?
 - Discussion: Reconfigurable Instruction Sets
- 5 Conclusion

- Instruction Set Reconfiguration
 - Re-use silicon area
 - Tuning to application needs
 - SW vs. HW alternatives

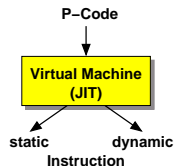


- Once the Instruction Set changes...
 - ...adopt code generation (compiler, binutils)
 - ...recompile the application
 - ...stop and reload the running application
 - ...resume operation.

- **Nearly impossible in a production environment.**
- Can we achieve a more lightweight process?

Getting a universal binary...

- Solution: Universal Binary
 - “One binary to fit them all”
 - On-the-fly translation to current HW configuration
- *“But JIT is slow and costly, think about Java!”*
- Java *bytecode* translation can be done in HW (FPL05)
- Even easier when bytecode is really close to target architecture
- Static (basic) instruction set enhanced by Dynamic (application-specific) instruction set
 - CARPE: Meta-RISC (Steil 2004, Heßmann 2004)
 - Xilinx Virtex-4 (APU, FCM)

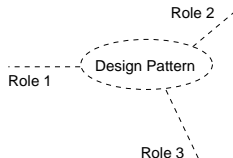


- Run-time mapping only for dynamic instruction set
- In HW or SW (Interpreter/VM)
- Map application-specific code to
 - Function in hardware: currently assigned opcode
 - Function in software: call to library function
- Once the system changes, only change mapping table!
- Not necessary to...
 - ...adjust code generation
 - ...recompile the application
 - ...interrupt execution (stop, load, resume)

- Applications have **restrictions** (timing, throughput, ...)
- Breaking restrictions means breaking the applications
- Reconfiguration likely to **break** restrictions
 - Exchange of HW accelerator with software implementation
 - Different implementations have different attributes!
- Reconfiguration must follow guidelines
- Guidelines dictated by applications
- Method required to express guidelines and grant requirements.
- Reconfiguration might change monitoring requirements!

Application Attributes

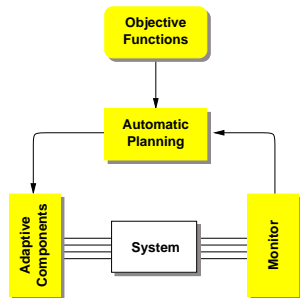
- Application description enriched by attributes
- Attributes define **requirements**
 - Throughput, Latency
 - Accuracy, Resolution, Datatypes
 - ...
- Various scenarios possible
 - Absolute Must (not less/not more than)
 - Ranges (min/max)
 - Quality Degradation



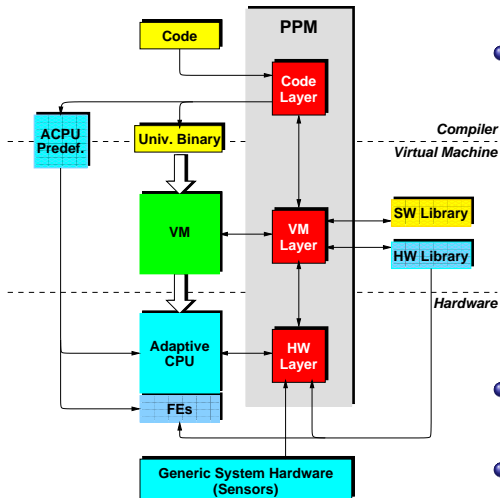
- Alternative implementations of application-specific functions enriched by attributes
 - HW-Accelerator
 - Various SW Implementations
- Attributes define **capabilities**
 - What can be achieved using this implementation?
 - What are its limitations / drawbacks?
 - Power vs. Performance
 - Performance vs. Quality
- Weighing against application requirements possible!

Evaluation Process: Adaptive Planning

- Application Requirements
- Implementation Capabilities
- Additional Directives
 - Sensor Input (Temperature, Power Supply...)
 - User Interaction
- Directives define **Objective Functions** (weighing of individual parameters)
 - Evaluation of System Configuration
 - Computation of required Reconfiguration
 - Choosing alternative representations (HW, SW) based on attributes



Example Implementation



- Application: Portable “Always-on Communication Device”
 - SdR Application
 - Chose most appropriate communication method from currently available ones
 - Cooperation with G. Acher, TU Munich
- Problem 1: Power vs. Performance
- Problem 2: Stay online

- Problem 1: Achieving uniform application description
 - Application in sequential programming language
 - SW modules in sequential programming language
 - HW modules in HW description language
- Problem 2: Extract Parallelism!
 - Currently: Manual parallelization and optimization by programmer
 - Even worse: How to program something like DodOrg?
- Idea: Use HW description languages (VHDL, Verilog, SystemC)

HDLs – A possible solution?

- Implicit definition of parallelism
 - Sequentiality must be explicitly programmed!
- Explicit definition of inter-module communication
 - Entity / Component
 - Component Instantiation
 - Process Invocation
- HW and SW implementations can be generated from the same description
 - Eases HW/SW codesign
 - Eases on-demand replacement of HW with SW implementations (and vice versa)
- Tool reuse possible: map HW description to existing tools / programming paradigms

Reconfigurable Instruction Set: Required?

- Typical low-level functions for computing-intense application already covered by GPP's static instruction set (MMX, SSE, ...)
- DSP algorithms rely on certain high-level functions (FFT, DCT, Huffman, ...) – further enhancement of the static instruction set?
- Similar situation for e.g. cryptographic algorithms (Multi-input XOR, parallel lookup & table indexing)
- Further Hardware-acceleration very algorithm/application-centric and not shareable
 - Monolithic building blocks
 - Does it belong into the ISA?
- Reconfiguration time and power
 - Interruption of execution
 - Overlapping execution and reconfiguration – “Shadowing” techniques costly

- Two major problems with reconfigurable architectures
 - Code generation and execution (Reconfigurable Instruction Set)
 - Grant application requirements
- Achieve universal binary using JIT
- Enrich application description by attributes
- Provide implementation alternatives as attributed libraries
- Weigh application demands against implementation capabilities
- Enrich run-time system by new functions

Thank you for your attention!