Adaptive Embedded Systems
Challenges of Run-Time Resource Management

Jan Madsen

Adaptivity

- An embedded system is adaptive, if it can modify its behavior and/or architecture to changes in requirements, objectives, and/or external conditions
- Adaptivity is increasingly important as the complexity and autonomy of embedded systems increases
- Improve performance and resource utilisation
- Improve reliability and fault-tolerance
Issues

• Hardware architecture
• Application mapping
• Run-time resource management
Hardware architecture

Issues
- Size of RU?
- Size of logic?
- Granularity of logic?
- Number of contexts?

\[ A = \#RU \times (A_{\text{logic}} + \#\text{context} \times A_{\text{context}}) + A_{\text{NoC}} \]

Application mapping

Task transformation
- RU = 50 CLB

• Task may be mapped to any RU
Application mapping

Task mapping

Run-time resource management
Run-time resource management

Issues
• Task dependency monitoring
• Task scheduling
• Task allocation and free-context management
• Task reallocation

→ Time critical!
Run-time resource management

- **C-node**
  - Collect resource distribution information
  - Decides on M-nodes to run application
- **M-node**
  - Allocate application to S-nodes
  - Monitor synchronization
- **S-node**
  - Run tasks of application
  - Multiple contexts requires local scheduling
Experiment: MP3 case

- **Assumptions**
  - Area = 10,000 CLBs
- **FPGA implementation**
  - Area = 2,408 CLBs (24% of chip)
  - Execution time: 61,793 cc

Experiments: RU size?

![Graph showing execution time and chip utilization for different RU sizes.

- Execution time (1000cc)
- Chip utilization %

- For 3x3, 4x4, and 5x5 RU sizes, the graphs illustrate the execution time and chip utilization percentages for FPGA, slow communication, and fast communication contexts.]
Experiments: number of contexts?

![Graph showing execution time and chip utilization for different number of contexts.]

Experiments: reallocation?

![Graphs showing execution time for different contexts and with reallocation.]

DTU Informatics
MPSoC'08
Allocation/reallocation strategies

• Basic
  – Form cluster around M-node to reduce communication

• Critical-path
  – Prioritize allocation of tasks on critical path

• Helix
  – Form local cluster for each application

• Dynamic-priority
  – Prioritize tasks that are close to finish

• Proactive
  – “Run-time clean-up”

Allocation/reallocation strategies

• Running 100 applications (of 5 different types)
Summary

• Understanding run-time behavior of dynamically reconfigurable systems is \textit{hard}
• Being able to do run-time resource management is \textit{even harder}
  – Analysis has to be fast to be feasible
  – What are the key issues to analyse?
  – Reallocation issues are complicated to analyse at run-time
• Essential to understand interplay between the different aspects of the dynamic behavior
• \textit{Need tools to support this!}

Acknowledgements

• Kehuai Wu
• Esben Rosenlund Hansen
• Michael Reibel Boesen
