Challenges of Mapping Real-Time Streaming Applications to General Purpose Manycores

Jonas Diemer, Rolf Ernst
diemer@ida.ing.tu-bs.de
Map2MPSoC Workshop 2010, 29 June 2010
Outline

- Motivation and Introduction
- Resource Management Approach
- QoS Enforcement and Analysis for the NoC
- Conclusion
Motivation

- **Goal:** Combine
  - Real-time, e.g., augmented reality, SDR
  - Best-effort, e.g., office, games,
  - On general-purpose many-cores
    - Consumer devices (phones, PCs)

- **Vision:** "App Store" for real-time applications
  - Provide guaranteed performance on a multitude of devices

- **System-level challenges:**
  - Resolve resource conflicts (predictability)
  - Application diversity (throughput vs. guarantees)
  - Applications change at run time
Characteristics of Application Classes

- **Best-effort applications**
  - Most existing applications, major role in user experience → “first-class citizen”
  - Unpredictable and bursty resource usage
  - **Latency-sensitive**: Application performance degrades with higher latency

- **Real-time streaming applications**
  - Require resource and timing guarantees
    - Resource sharing must be under control for efficient co-execution
  - Regular access patterns → **Latency-tolerant**: Performance does not degrade with higher latency (up to a certain latency bound)
General-Purpose Many-Cores = all shared resources

- Cores
- Packet switched Network-on-Chip interconnect
- Multi-level caches
  - Private L1 (+L2)
  - Distributed shared last-level cache (accessible via NoC)
- Multi-channel off-chip memory

Currently, resource sharing is managed by first-come first-serve strategies → Infeasible for guarantees!

Need predictable resource sharing mechanisms = Platform QoS

Question:
How can we provide end-to-end guarantees using individual resource sharing mechanisms?
1. Applications request resources from resource manager by providing an application model with timing / resource constraints

2. Resource manager performs mapping of application model

3. Application constraints and platform limitations are validated
   - Go back to mapping if constraints are not met

4. Lightweight platform QoS mechanisms for predictability

cf. e.g. [terBraak2010], [Shankar1999]
Resource Management Infrastructure – Enforcement

- Individual mechanisms
  - Cores: Scheduling, SMT policy
  - Cache: Address mapping [Cho2007], locking [Vera2003] and/or partitioning [Kim2004]
  - NoC: Lightweight Throughput Guarantees [Diemer2010a,b]
  - Memory: Priorities, rate limits [Heithecker2005]
  - Controlled by registers, config. messages
  - No compromises of BE throughput!

Application → Mapping → Constraint validation → Enforcement
Example: BE-Optimized QoS for NoCs

- Existing mechanisms put BE in background (low priority, idle slots)
- Idea: Exploit latency tolerance of RT streaming applications to improve BE latency
- Approach: Prioritize BE as long as guaranteed throughput (GT) traffic makes sufficient progress → “Back Suction” [Diemer2010b]
  - Progress measured by buffer occupancy (similar to Back Pressure)
  - Prioritize GT only if downstream buffer occupancy low

30% latency improvement over standard prioritization scheme

Improve application performance by ~ 10%

![Graph showing latency improvement and application runtime comparison](image)
Back Suction Architecture

- Reserve one set of VC (source → sink) per GT stream at run-time
- **Limit rate** (to guaranteed rate) at which sink may assert back suction
- **Threshold Module** at every VC
  - Forward back suction signal on low occupancy towards upstream
  - Threshold determines how early prioritization of GT propagates towards sink

![Back Suction Architecture Diagram]
Prioritize BE: Selective-Priority Arbiter

Separate arbiters
- BE: Winner-takes-all
- GT: Round-robin

Priority selection logic
- Select BE or GT based on
  - Signal $a_N$
  - Presence of BE/GT
Resource Management Infrastructure – Validation

- Validate timing constraints
  - Overlapping GT streams require scheduling analysis to guarantee individual throughputs
  - Throughput guarantees depend on selection of suction threshold
    → Analysis to determine minimum threshold

- Validate resource availability
  - Number of overlapping GT connections limited by available virtual channels
  - Available VC buffer space must be larger than suction threshold
  - Granularity of guarantees (rate limiter, threshold)
Real-Time Analysis of Back Suction (1)

- Overlapping GT streams share a router output port
- Scheduling analysis (similar to Network Calculus)
  - Stream = task
  - Output port = resource
  - Back Suction = task activation
  - Rate limit at sink = worst case arrival function
- Round-robin analysis at every router:
  - Worst-case service
  - Worst-case backlog
  - Threshold & Worst-case response time
  - Output event model

![Graph showing maximum number of back suction events over time](image)
Real-Time Analysis (2)

- Analysis performed on-line as part of the resource management process
  - Analyze at sink first (where we already have an activation model)
  - Propagate models from sinks towards sources
- Analysis time for system ~ 10-100ms (non-optimized python code!)

Analysis result: Feasibility, Suction thresholds
Resource Management Infrastructure – Mapping

- Map RT applications
  - Tasks → Processing core
  - Communication → GT/GL NoC Links
  - Buffers → Locked/partitioned cache
- Optimization (heuristic)
- Feedback from validation phase
Resource Management Infrastructure – Application Model

- Request specification: abstract extended DFG model for real-time applications
- Characterization of best-effort applications
  - Obtained from monitoring
  - Optional, to guide mapping heuristics
Conclusion

- Mixing real-time and best-effort applications efficiently is challenging
  - Worst-case predictability vs. best-effort throughput

- Platform with light-weight QoS
  - Predictable sharing mechanisms for individual resources
  - Low overhead and little negative effect on best-effort throughput (e.g. Back Suction)

- Need system-level resource management to
  - Give end-to-end guarantees based on individual mechanisms
  - Overcome resource dependencies
  - Perform run-time mapping
  - Handle limitations of QoS mechanisms
References


- **[Vera2003]**: X. Vera and B. Lisper and J. Xue, “Data cache locking for higher program predictability”, SIGMETRICS 2003


Thank You for Your Attention!
Questions?

Jonas Diemer, diemer@ida-ing.tu-bs.de