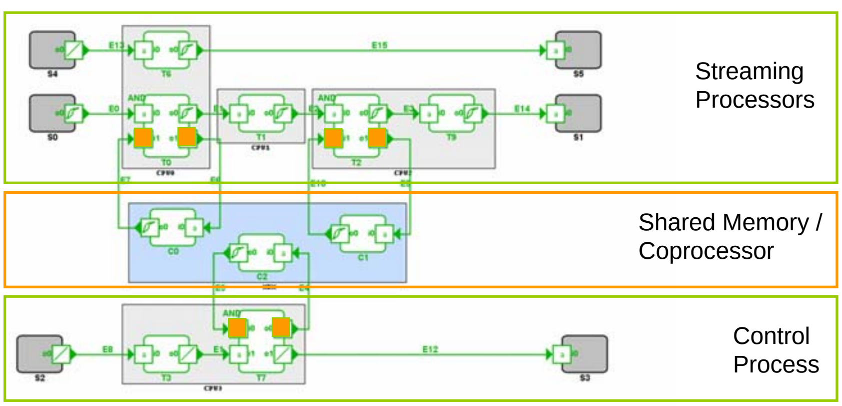
Analysis of Shared Coprocessor Accesses in MPSoCs

Overview

Simon Schliecker
Matthias Ivers
Rolf Ernst

Bologna, 22.05.2006

System Setup



New Task Model

Classical Task-model

C_i Core Execution Time

Read all data when activated

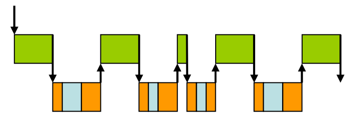
output all data when finished

“Communicating Task”-model

C_i Core Execution Time

$\mathbf{Q}_i$ Set of incurred *transactions*

Read *some* data when activated

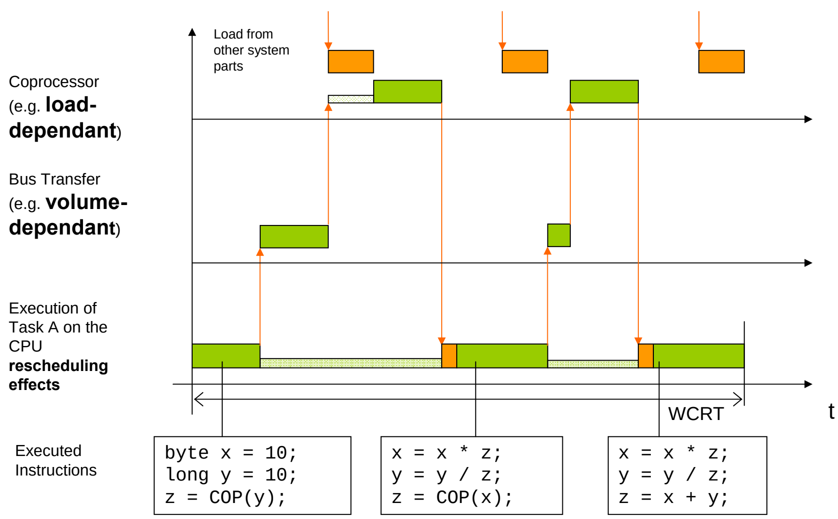


Output *some* data when finished

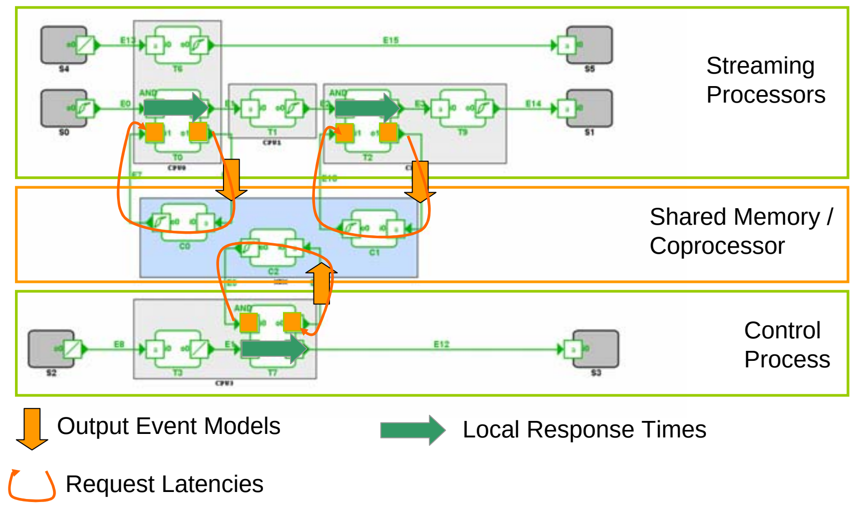
Access shared resource multiple times during execution.

Synchronization still only at task activation.

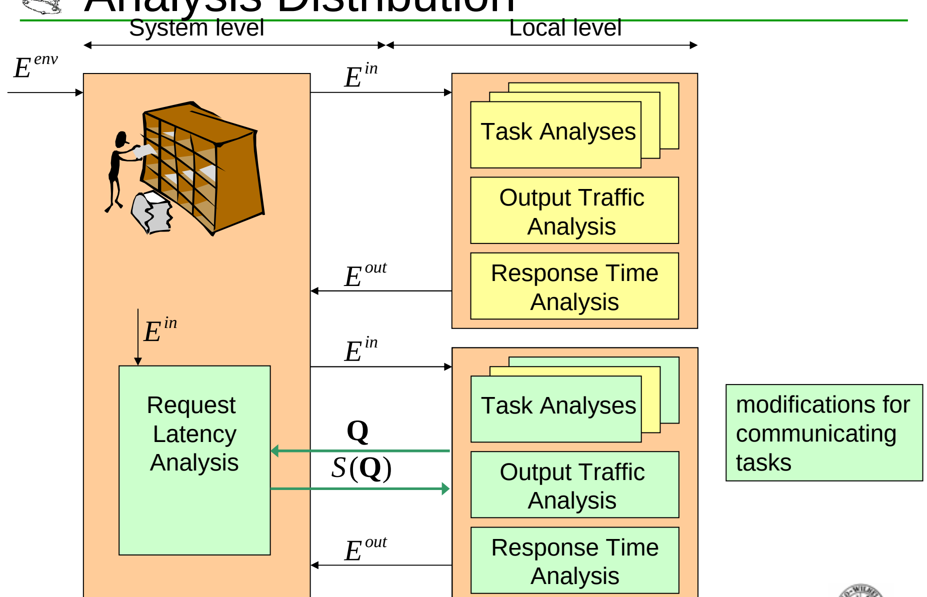
Effects on Coprocessor Accesses



Analysis Steps



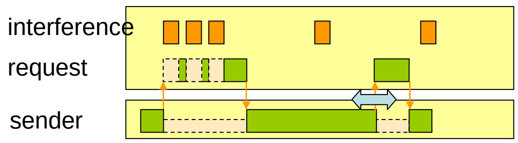
Analysis Distribution



Correlated Event Total Busy Times

- Not every request experiences the worst case in dynamic systems!

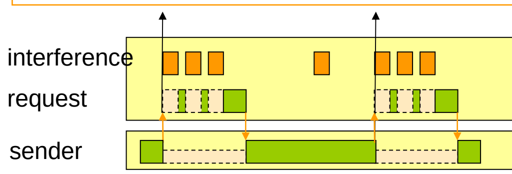
Problem: Can not predict exact request times!



variable times of interference
+ variable times of request

highly *variable* and *sensitive* request latencies!

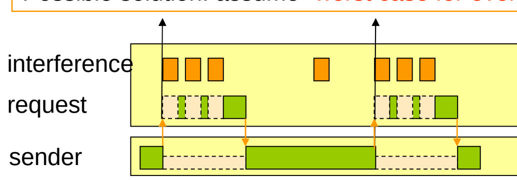
Possible solution: assume "worst case for every request"



Correlated Event Total Busy Times

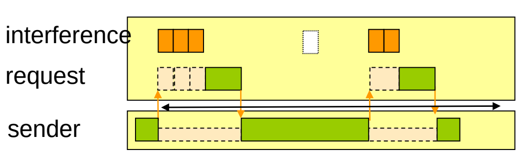
- Deriving individual bounds is difficult
- Using individual bounds for analysis is difficult
- but ignoring it is easy!

Possible solution: assume "worst case for every request"

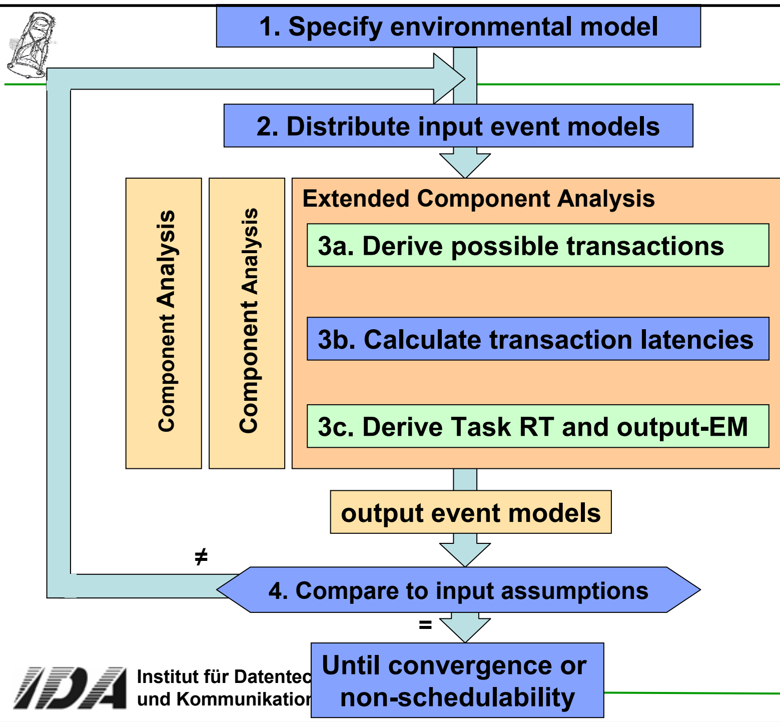


no remaining overestimation given interference with large jitter!

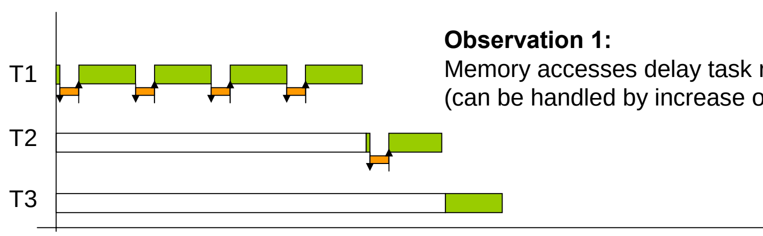
Better solution: approximate "maximum *total* busy time"



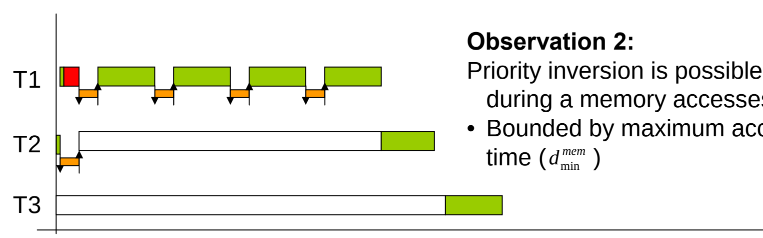
total interference
and all requests
in total time window
⇒"total busy time"



Response Time (if processor stalls)



Observation 1:
Memory accesses delay task response (can be handled by increase of CET)



Observation 2:
Priority inversion is possible during a memory accesses
- Bounded by maximum access time ($d_{\min}^{mem}$)

Scheduling Analysis

- If processor stalls during Memory requests:
  - Processor is NOT released, this extends CET.
  - Higher priority tasks can be *blocked* by maximum memory access time.
  - Buffer is always empty, because previous requests finished.

$$R_i = B_i + C_i + S(Q_i) + \sum_{j=1}^{i+1} \eta_j(R_i) \cdot (C_j + S(Q_j))$$

$$B_i = d_{\min}^{mem}$$

Total blocking time

CET and. CoP times

activations of hp task

CET and CoP times

Scheduling Analysis (2)

- The more requests are considered together, the smaller the overestimation!
  - Collect all requests that can lead to delay and add maximum total busy time
  - Perfect match for improved path latencies

$$R_i = (B_i + C_i + \sum_{j \in hp(i)} \eta_j(R_i) \cdot C_j) + S(\bigcup_{j \in hp(i)} Q_j))$$

$$B_i = d_{\min}^{mem}$$

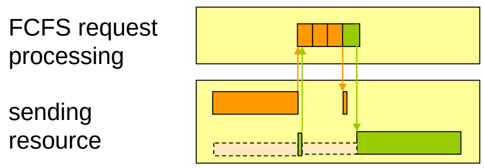
Total CoP times that can lead to delay of task i

Multithreading from real time perspective

Generally:
- stalling decreases and
- processor utilization increases

But: Additional interference for requests
- interference from previous requests can completely compensate the gain of reduced stalling!
- A task can be fully delayed by higher priority execution ***and*** requests
- FCFS ordering along request chain counters priorities on sending resource

⇒ **no gain for response time** under given task assumptions



Additional Critical Sections

- How much blocking time to take into account?
  - Blocking Memory Accesses can be “nested” into Critical Sections (not the other way around)
  - Assume a virtual semaphore “memory”:
    - All tasks require “memory” to be free to start executing
    - Some tasks spend no time accessing “memory”, but still must wait until it is free
    - Other tasks access “memory”, and may enter the critical section multiple times
  - Memory Accesses are “automatically” protected with highest priority!
  - Problem mapped to “nested critical sections problem” (Sha, Rajkumar)
    - Depends on utilized protocol PIP, PCP