How Unthoughtful Resource Sharing Counteracts WCET-Analysis

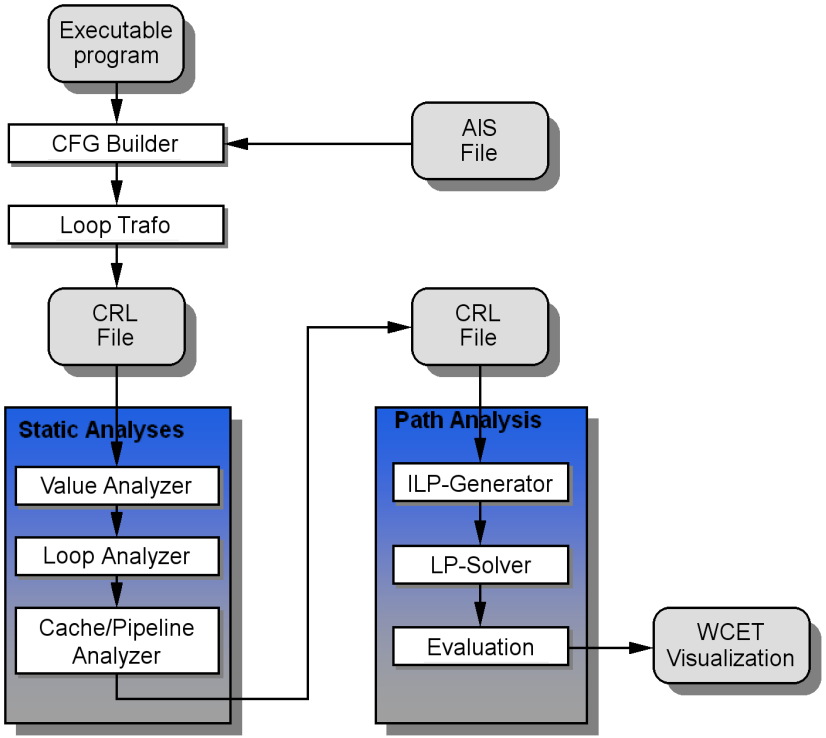
Dipl.-Inf. Christoph Cullmann
AbsInt Angewandte Informatik GmbH

Motivation

- Static analysis is widely accepted for calculating worst-case execution times (WCET) for safety-critical hard-realtime systems
- Tool support is available: aiT, e.g.
- The demand for more performance leads to the introduction of multi-core architectures, not designed for safety-critical applications

Single-Core WCET Analysis

- Analysis of „complex“ processors possible because the actual computations and accesses are fully determined by the control flow graph
- Therefore even possible to analyse architectures with timing anomalies and domino effects

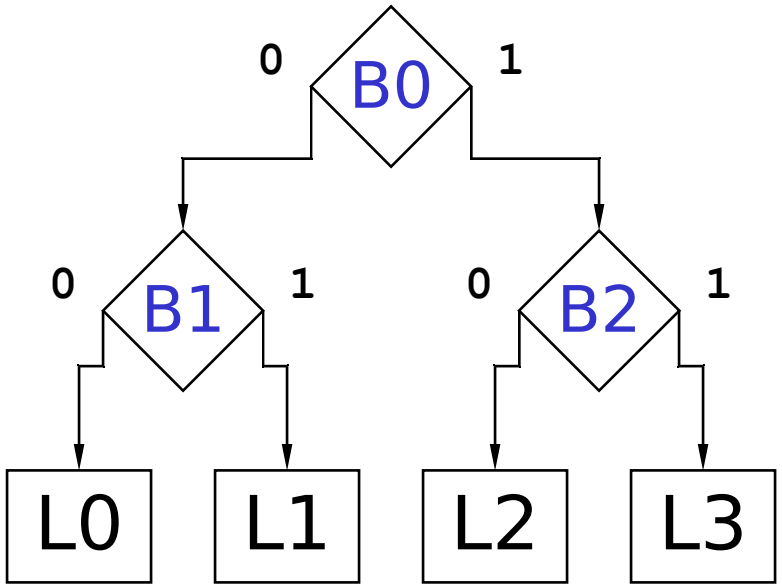


Domino Effect

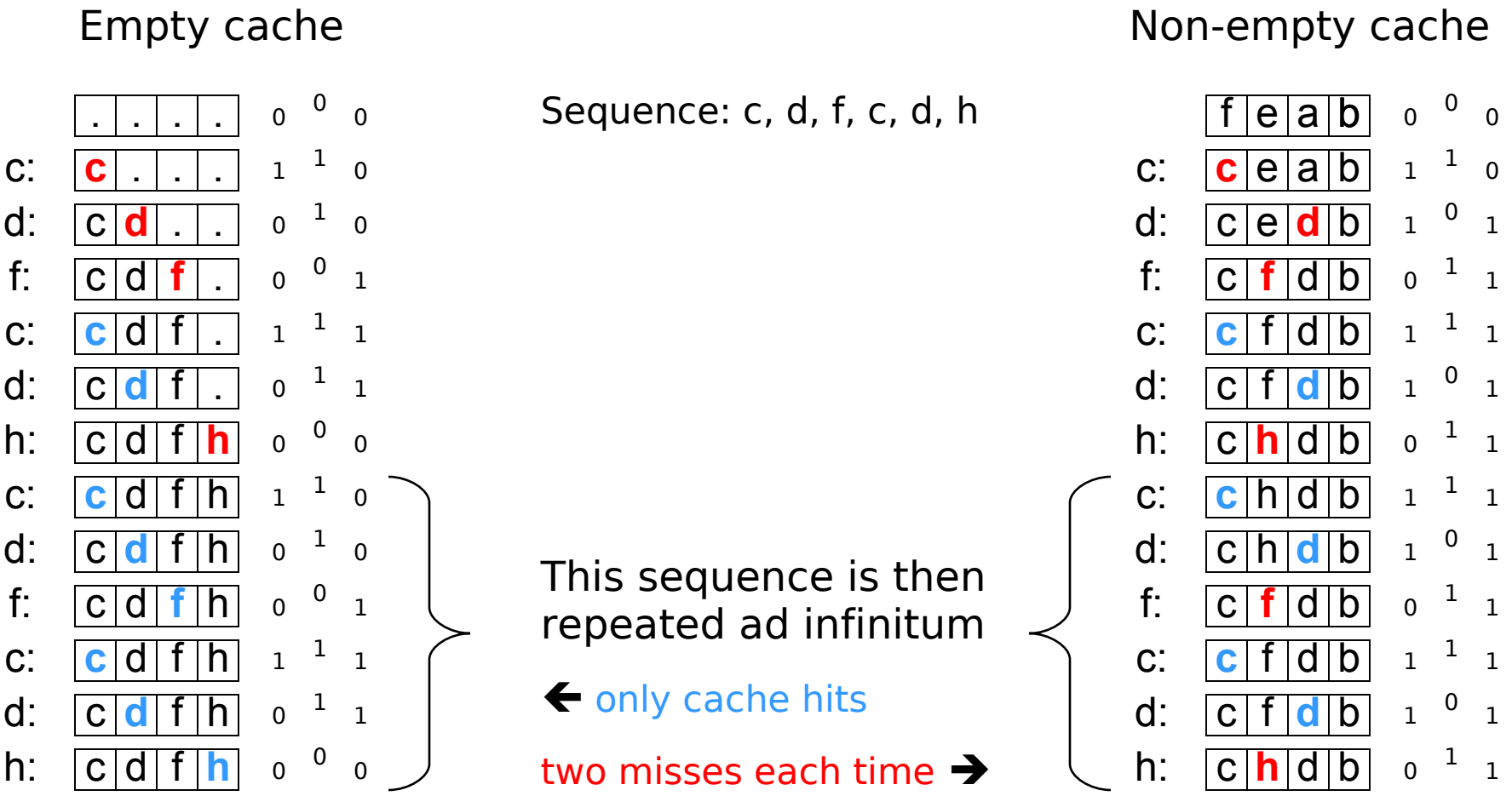
- Timing anomaly
- Execution time increase is not bounded by hardware determined constants
- Certain instruction sequences e.g. in loop bodies can trigger this effect and increase latencies in further iterations

Pseudo-LRU Replacement

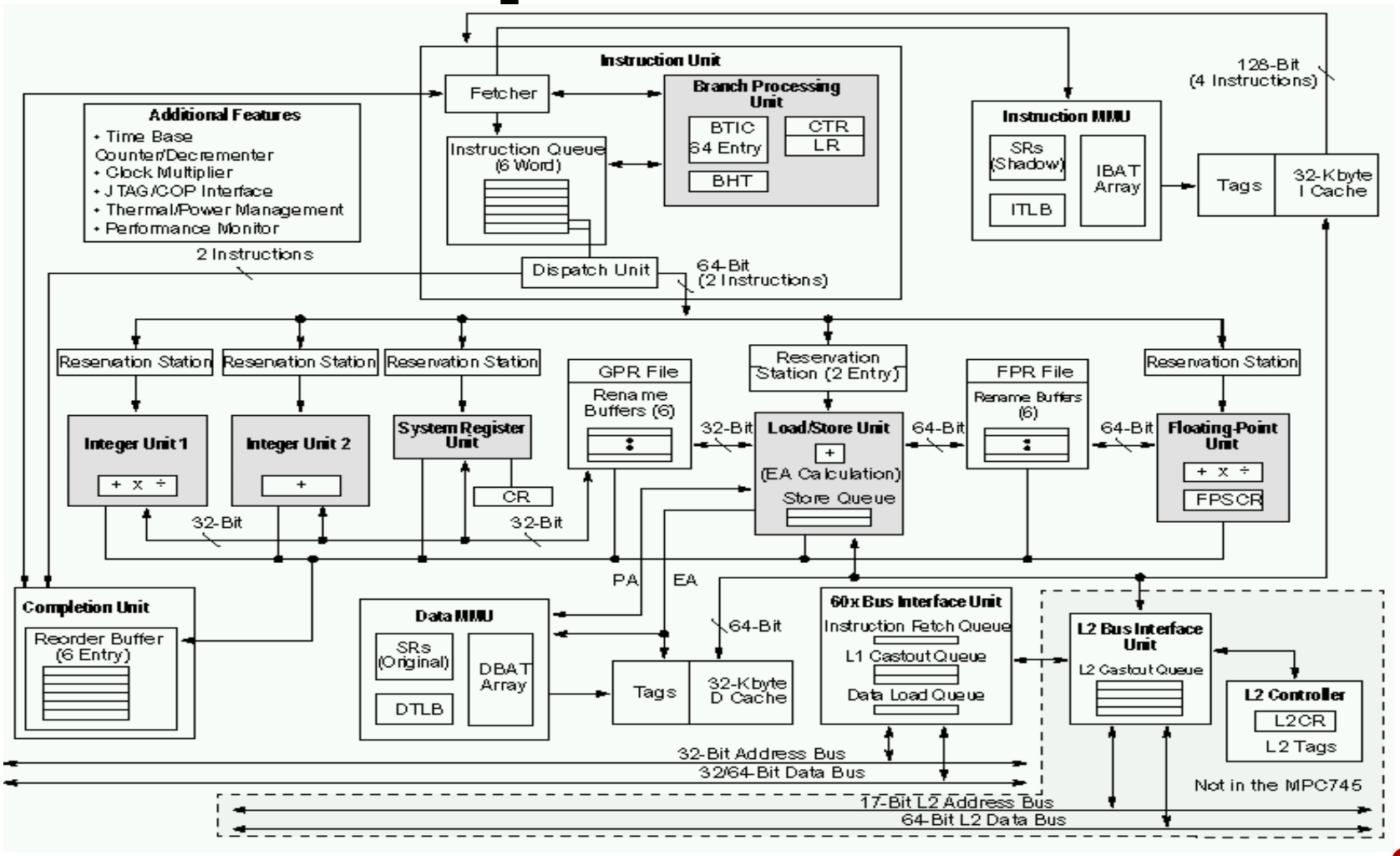
- Used for example in PPC G3
- Each setting of B[0..2] points to a specific line:



4-way PLRU Domino Effect



PPC755 Pipeline



Instruction Sequence S1

A lwz r20, 0(r2)

B addi r21, r20, 4

C mullw r19, r14, r29

D lwz r23, 0(r20)

E addi r24, r23, 4

F addi r25, r14, 4

G lwz r26, 0(r19)

H mullw r27, r14, r29

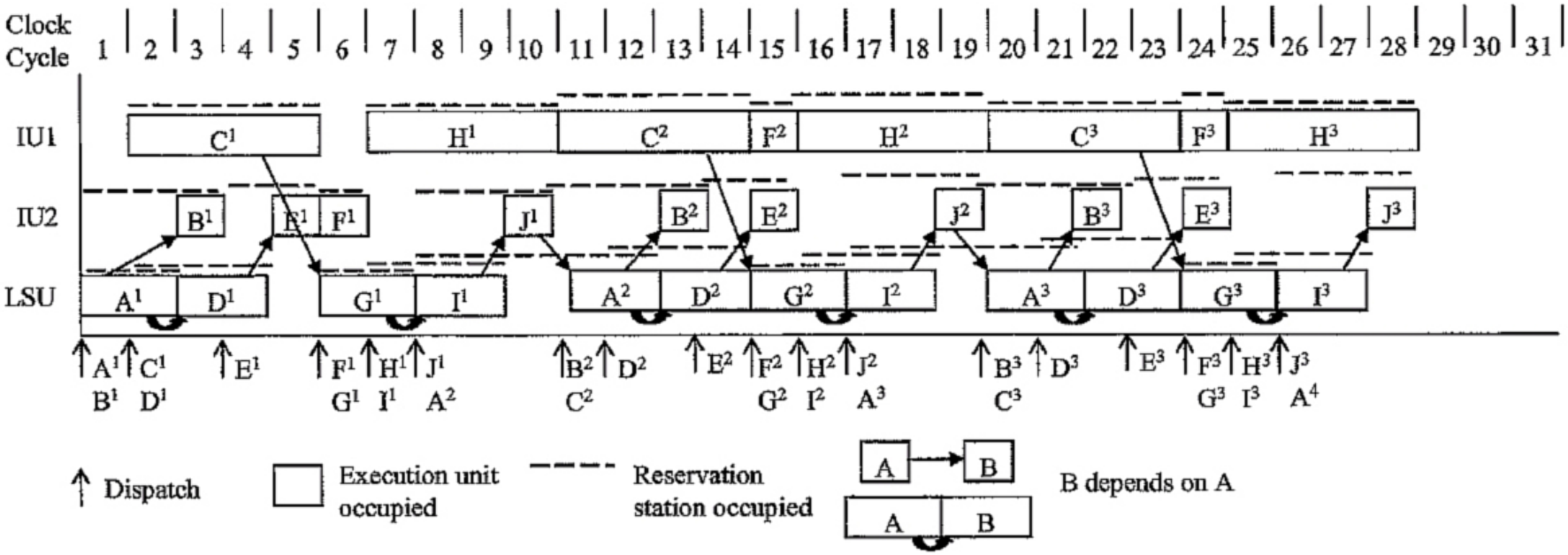
I lwz r28, 0(r26)

J addi r22, r28, 0

- mullw can only be executed by integer unit IU1
- lwz can only be executed by the load/ store unit LSU
- S1 must be repeated at least 3 times

Execution Units Overview

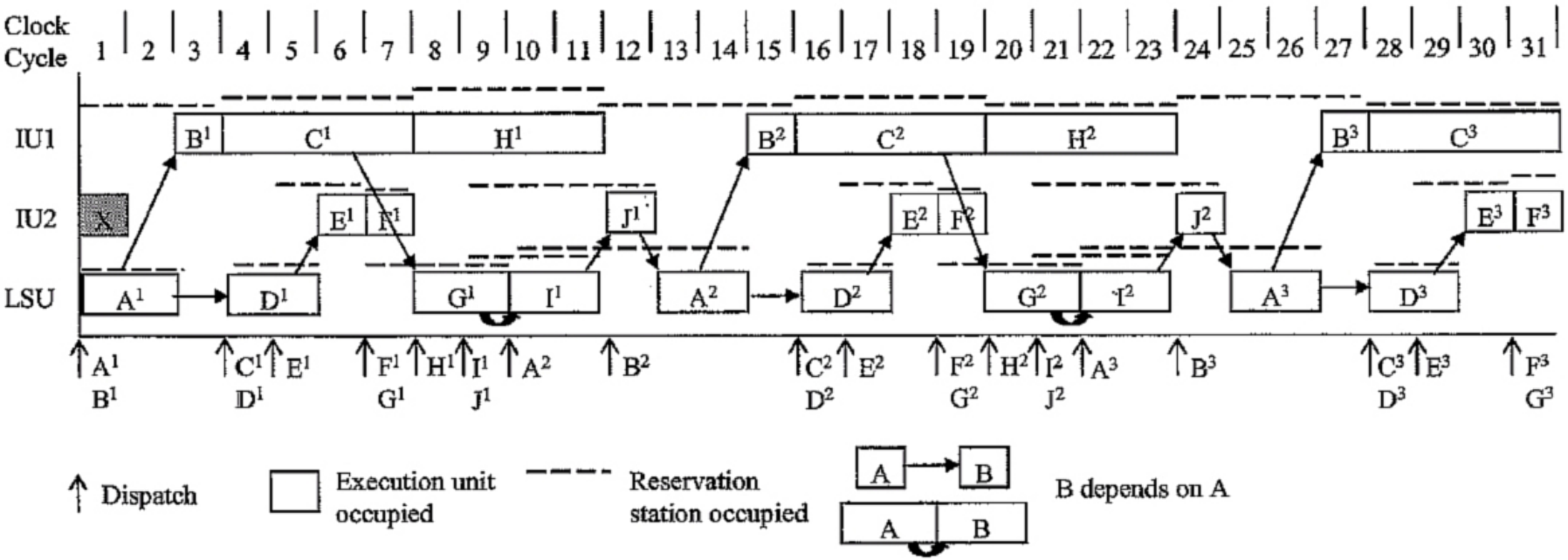
Distribution of instruction sequence S1 on the execution units IU1, IU2 and LSU.



- In cycle 1 instructions A and B are dispatched to LSU and IU2. So C can be dispatched to IU1 in cycle 1.
- 10 + 9(n-1) cycles are needed with n being the number of iterations

Example: Domino Effect

Distribution of instruction sequence S1 on the execution units IU1, IU2 and LSU with an additional leading instruction X. **Domino effect**!

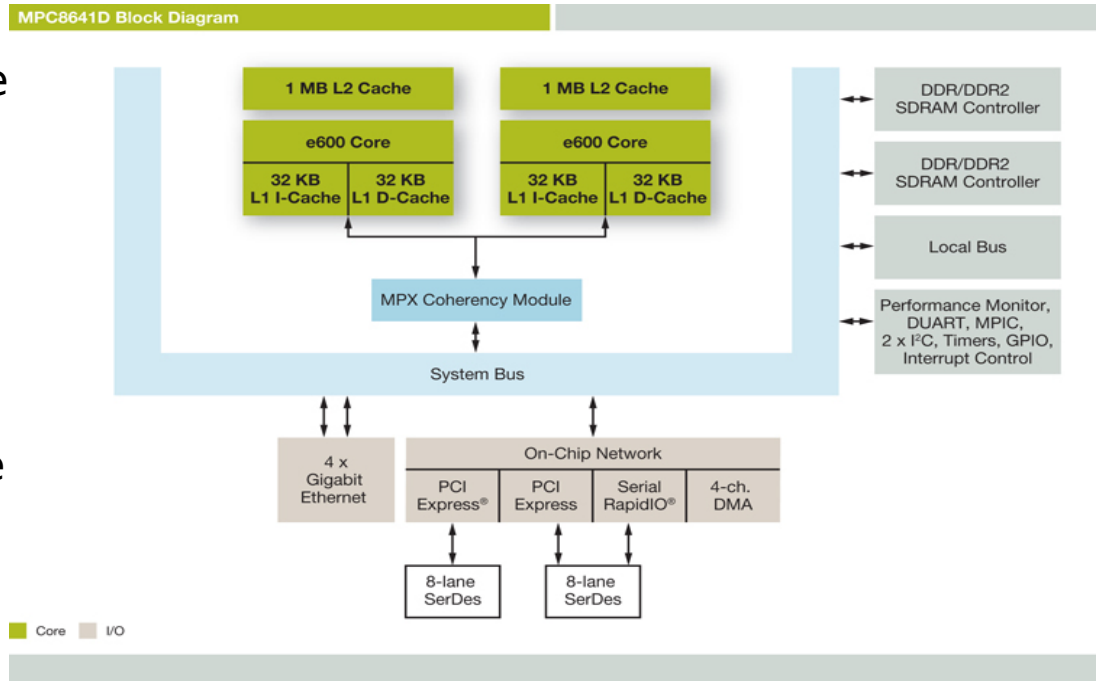


- With the insertion of instruction X, B is dispatched to IU1 in cycle 1.
- C can only be executed by IU1 and so has to wait for B to finish. B has to wait for the results of A.
- While J is executing B can be already dispatched to IU1 and the stream is again delayed
- **3 more cycles per iteration (33%)!**

Problem of Sharing

- On typical multi-core architectures memories are shared
- Simplistic WCET analysis would assume conflicts possible on each memory access

  => in the best case same performance as a single-core
- Idea:
  - Reduce conflicts
  - Count number of conflicting accesses



Classification of Architectures

Memory Hierarchies, Pipelines, and Buses for Future Architectures in Time-critical Embedded Systems

Reinhard Wilhelm, Daniel Grund, Jan Reineke, Marc Schlickling, Markus Pister, Christian Ferdinand

- Fully timing compositional architectures

  No timing anomalies (ARM7)

- Compositional architectures with constant-bounded effects

  Timing anomalies but no domino effects (Infineon TriCore)

- Non-compositional architectures:

  Domino effects and timing anomalies (PowerPC 755)

Reduce use of shared resources!

- Use compositional pipelines
- Use predictable caches
  - LRU-Replacement policy and/or scratchpad memories
  - Seperate instruction and data caches
- Non-shared memory where there is little sharing in the application

The PROMPT Design for Predictable Multi-core Architectures

The PREDATOR Consortium

PREDATOR Reconciling Predictability with Performance

PREDATOR is an ICT project in the 7th Framework Program of the EU

The PROMPT Design Process

1. Hierarchical privatization
   - decomposition of the set of applications according to the sharing relation on the global state
   - allocation of private resources for non-shared code and state
   - sound (and precise) determination of delays for accesses to the shared global state
2. Controlled socialization
   - introduction of sharing to reduce costs
   - controlling loss of predictability
3. Sharing of lonely resources – seldom accessed resources, e.g. I/O devices

Principles for the PROMPT Architecture and Design Process

- No **interference on shared resources** where not needed for performance
- **Harmonious integration of applications**, i.e. without introducing interferences on shared resources not existing in the applications