Addressing End-to-End Memory Access Latency in NoC Based Multicores

Akbar Sharifi, **Emre Kultursay,** Mahmut Kandemir and Chita Das
The Pennsylvania State University

MICRO 2012
Vancouver, BC

Abstract

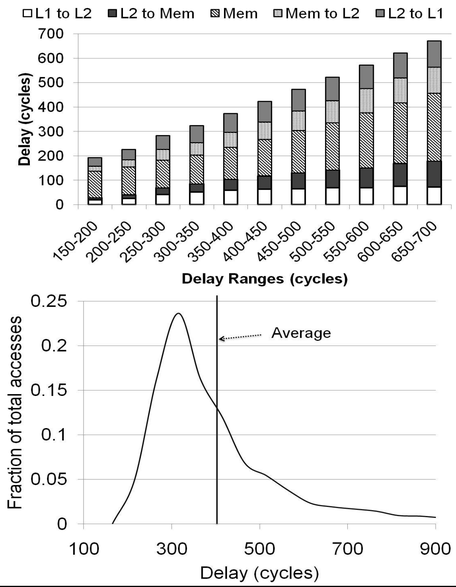
Targeting Network-on-Chip (NoC) based multicores, we propose two network prioritization schemes that can cooperatively improve performance by reducing end-to-end off-chip memory access latencies:

- Scheme 1: Prioritizes memory *response* messages such that, in a given period of time, messages of an application that experience higher latencies than the average message latency for that application are expedited and a more uniform memory latency pattern is achieved.
- Scheme 2: Prioritizes the *request* messages that are destined for idle memory banks over others with the goal of improving bank utilization and preventing long queues from being built in front of the memory banks.

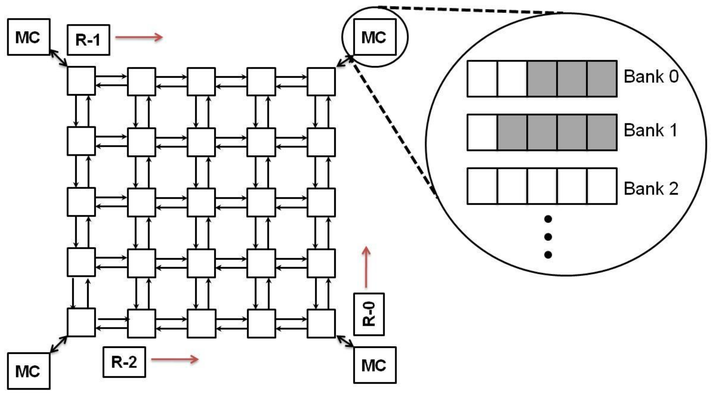
These two network prioritization-based optimizations together lead to uniform memory access latencies with a low average value. Our experiments with a 4x8 mesh network-based multicore show that, when applied together, our schemes can achieve 15%, 10% and 13% performance improvement on memory intensive, memory non-intensive, and mixed multi-programmed workloads, respectively.

Motivations

Motivation 1: Some memory accesses experience much higher delays than the others. For these messages, the contribution from the NoC is very significant.

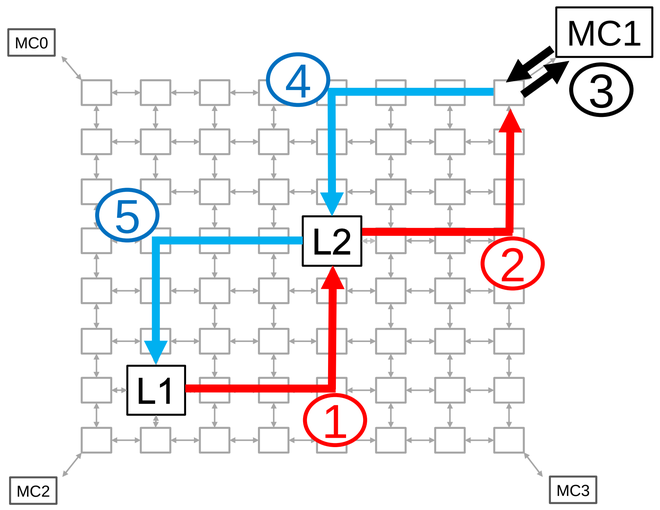


Motivation 2: Bank loads are non-uniform. At any instant of time, while some banks are busy, other banks are underutilized.



A snapshot of the states of three banks controlled by one of the memory controllers. As can be seen, Banks 0 and 1 have requests to be serviced, whereas Bank-2 is idle.

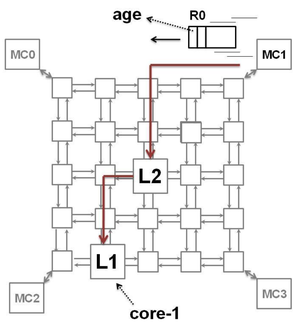
Memory Access On the Target System

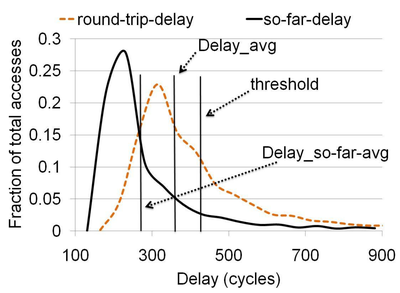


In order to look-up the corresponding L2 bank (determined by S-NUCA), a request message is sent from L1 to L2 (path1). If the data is a hit in L2 cache, then the response data is sent back to the L1 cache (path5). In case of an L2 miss, the request is forwarded to the memory controller (path 2). The memory controller schedules this request (path 3) and the response is sent first to the L2 cache (path 4) and then to the L1 cache (path 5).

Proposed Schemes

Scheme-1: Reducing variations across memory accesses.





MC1 has received a memory request from core-1, and R0 is the corresponding response message for that request after it is serviced by MC1. If R0 is detected to be "late", then it is prioritized over other messages on the network on its way back to the core.

Scheme-2: Balancing memory bank loads.

An off-chip request message is given a higher priority in the network if no message has been sent to the same bank recently. Decision is made based on the bank history table data kept in the routers.

Implementation

Scheme-1: At each router/memory-controller, once the message is ready to be sent out, the age field value is updated as follows:

$$age = age + \frac{(cycles_{current} - cycles_{message\_entry}) \times FREQ\_MULT}{local\_frequency}$$

Scheme-2: Each node exploits "local information" to estimate the pressure imposed by the requests on the memory banks. Specifically, each router keeps and updates a table that records the number of off-chip memory requests sent from this router to each bank in the last T cycles.

Summary

- Identified
  - Some memory accesses suffer long delays and block the core
  - Banks mostly idle, bank utilization varies
- Proposed two schemes
  1. Network prioritization and pipeline bypassing of "late" memory *response* messages to expedite them
  2. Network prioritization of memory *request* messages to improve bank utilization
- Demonstrated
  - Scheme 1 achieves 9% average speedup
  - Scheme 1+2 achieves 13% average speedup

Results

32-core system