Assignment 3: R-NUCA and Adaptive Odd-Even Routing

1 Part A: Odd-Even Routing

Implement the Odd-Even routing scheme as described in the paper [1].

Tejas-specific Details

- In the config file,
  - set <NumCores> to 16.
  - set <Interconnect> to NOC.
  - save the attached NocConfig.txt file in your system. Set <NocConfigFile> to the absolute path of the saved file.
  - set <NocTopology> to TORUS.
  - the <NocRoutingAlgorithm> tag is used to indicate the routing algorithm. Currently, some algorithms like simpleXY are recognized. Have it recognize Odd-Even also.
- Every simulated component in Tejas, such as cores, structures within cores like ROBs and LSQs, caches, TLBs etc, are derived from a base class `SimulationElement`. Each `SimulationElement` object has a field `CommunicationInterface` through which it communicates with other `SimulationElement` objects. Each `CommunicationInterface` essentially encapsulates a router in the NOC. If `SimulationElement` A wishes to communicate with `SimulationElement` B, it first sends the message to its own `CommunicationInterface`, which sends it to B's `CommunicationInterface`, which sends it to B. If two elements share the same `CommunicationInterface` (the ROB and LSQ of the same core have the same `CommunicationInterface`), then the communication between them is *direct* – it does not involve the `CommunicationInterface`.

  `CommunicationInterface` can be of two types: `BusInterface` and `NocInterface`. Concern yourself with `NocInterface` only.
- The NOC in Tejas is a packet-switched network. To understand how routing is done in Tejas, begin with the function `sendEvent()` in the file `SimulationElement.java`. Go through the call tree that arises from this function and understand all the concerned files.

2 Part B: Reactive-NUCA (R-NUCA)

Implement the R-NUCA scheme as described in the paper [2]. Compare the performance with (i) uniformly accessed shared cache, (ii) S-NUCA (already implemented in Tejas). Use the Odd-Even routing scheme implemented in Part A.

Tejas-specific Details

- The file `NocConfig.txt` describes the layout of the chip. The given file shows that there are 16 cores, and 16 L3 cache slices. These 16 slices are shared among the 16 cores in a NUCA arrangement. The paper [2] says each tile has a slice of the shared cache. Assume that, in `NocConfig.txt`, a core and the L3 slice to the right of the core are on the same tile.
- In the config file,
  - Currently, each core has private L1 and L2 caches, and shares an L3 cache with the other cores. The L3 cache is of type `L3Cache_1M_8`. See how cache types are defined and used in the config file. Create a new cache type `L3cache_4M_8` with <Size> as 4194304, and <Latency> as 22 cycles. Set the L3's type to `L3Cache_4M_8`.
  - Set <Nuca> of `L3Cache_4M_8` to R_NUCA. Currently, only S_NUCA and D_NUCA are recognized. Have it recognize R_NUCA also.
  - Set <Coherence> of `L2Cache_256K_8` to D1.
- In Tejas, the class `Cache` is used to represent all private caches and shared, uniformly accessed caches. Non-uniformly accessed shared caches are represented by the `NucaCache` class. Each slice that makes up a NUCA cache is of the type `SNucaBank` for S-NUCA, and `DNucaBank` for D-NUCA. All caches have a structure called the Miss Status Holding Register (MSHR) – it holds all requests to the cache that have yet not been responded to. In case of a NUCA cache in Tejas, all the slices share a single MSHR. This is a simulator approximation. You may do the same.
- The function `createElementsOfNOC()` in the file `ArchitecturalComponents.java` shows how the different components of the chip are created and initialized. You may follow this, and create the R-NUCA cache accordingly.
- Read the files `Cache.java`, `NucaCache.java`, `SNucaBank.java` (and any related files) to get an understanding of how the memory system works. When any cache gets a request from the upper level in the heirarchy (upper level may be another cache, or the pipeline), the request type is `Cache_Read` for a read request and `Cache_Write` for a write request. Follow the sequence of calls from the `handleEvent()` function in these files to understand the work flow.

- You need not concern yourself with coherence related functionalities. Similarly, you need not do the line invalidation that is suggested in the paper at the time of the *poison* state setting.

Simulate the given the suite of benchmarks (to be announced soon). Compare the performance with

1. Uniform Cache Access: set <Nuca> of `L3Cache_4M_8` to "None".
2. S-NUCA: set <Nuca> of `L3Cache_4M_8` to "S_NUCA".

3 Part C: Adaptive Odd-Even Routing

Implement an adaptive odd-even routing scheme. Basic odd-even routing can return more than one possible next hop. An adaptive scheme decides which next hop to use by taking into account the congestion in the outgoing links. Use the `NocSelScheme` tag in the config file to toggle between STATIC and ADAPTIVE. Compare with the performance of part B.

4 Part D: Bonus Question: Dynamic R-NUCA

Implement a dynamic version of R-NUCA. Compare with the results of part C.

References

[1] Ge-Ming Chiu. 2000. The Odd-Even Turn Model for Adaptive Routing. IEEE Trans. Parallel Distrib. Syst. 11, 7 (July 2000), 729-738. DOI=http://dx.doi.org/10.1109/71.877831

[2] Hardavellas, Nikos, et al. "Reactive NUCA: near-optimal block placement and replication in distributed caches." ACM SIGARCH Computer Architecture News. Vol. 37. No. 3. ACM, 2009.