Disclaimer: "The contents of this document are scribe notes for The University of Texas at Austin EE382V Spring 2007, Computer Architecture: User System Interplay. The notes capture the class discussion and may contain erroneous and unverified information and comments.*

The Stanford Dash Multiprocessor

Lecture #17: Monday, 26 March 2007
Lecturer: Mattan Erez
Scribe: Carl Olson
Reviewer: Mattan Erez, Min Kyu Jeong

The Computer Systems Laboratory at Stanford University developed a shared-memory multiprocessor called Dash (an abbreviation for Directory Architecture for Shared Memory). The fundamental premise behind the architecture is that it is possible to build a scalable high-performance machine with a single address space and coherent caches [1]. Dash accomplishes this by using a distributed directory at clusters of processors with a hierarchical bus structure. Although the Dash Multiprocessor was a research investigation of highly parallel architectures, a working prototype with 64 processors was built and tested at Stanford before the publishing of the paper in March 1992.

Reference:
[1] Lenoski et al., "The Stanford Dash Multiprocessor", IEEE Computer, Volume 25, Issue 3 (March 1992).

1 What is the problem being solved?

- Leverage commodity microprocessors and build a scalable high-performance machine with a single address space and coherent caches.
- Provide scalability that achieves linear or near-linear performance with tens of processors to thousands of processors while maintaining the price/performance advantage of an individual processor.
- Investigate parallel architectures without imposing excessive programming difficulty. Do not want to complicate the programming model.

- Cache-coherent shared address machines do not scale effectively. Snooping schemes cause common bus and individual processor caches to saturate.
- Problems of data partitioning and dynamic load distribution in parallel machines.

2 Who are the intended users?

- Application Programmers porting sequential applications to parallel machines.
- Users of current and future parallel applications.
- General Purpose Processing requiring high total system performance.
- Users of message passing machines because Dash can emulate message passing machines.

2.1 Intended Readers

- Operating System developers.
- Automatically parallelizing compiler writers.
- Computer architects defining scalable parallel architectures.
- Researchers developing interconnection networks. Dash claims to be independent, but high point-to-point interconnection, that can't broadcast, increases latency and bandwidth, making it bad for Dash. Interconnection is not explicit in paper, but Dash is not really independent of interconnect type.
- Parallel language developers to exploit underlying multicore hardware.
- Researchers interested in evaluation techniques for their parallel architectures.
- Readers interested in a brief overview of cache coherency and consistency concepts.

3 What is unique about the suggested solution?

- First operational machine to include scalable cache-coherence.
- Coherence protocol exploiting partitioned and distributed directories rather than a centralized one.
- Heirarchical (cluster based) cache and bus model. Combination of point-to-point and bus.

- Distributed memory to exploit locality. Private data can be made local to the cluster to avoid long latency remote reference.

- Coherence traffic is directory based point-to-point messaging. Directories allow you to send invalidates out to only the caches that need them.

- Prefetching in directory, update-write and deliver primitives, for handling blocking read-misses. The update-write operation sends the new data directly to all processors that have cached the data, while the deliver operation sends the data to specified clusters [1].

- Synchronization techiques: queue-based locks and fetch-and-increment operations. The queue-based locks in Dash use the directory to indicate which processors are spinning on the lock. Fetch-and-increment operations have low serialization and are useful for implementing several synchronization primitives such as barriers, distributed loops, and work queues [1].

- Creation of split-transactional bus protocol through masking of remote requests.

- Request and Reply Meshes eliminate request-reply deadlocks.

- Does not guarantee deadlock-free operation but deadlock avoidance through remote access cache buffering and NACKs.

- Shared address space allows users to assume uniform memory model.

- Supports multiple write invalidation overlaps due to non-blocking writes.

4 How is the idea evaluated?

- Simulated and built the Dash system. Three applications used to measure performance.

- Performance Monitoring Modules were used.

- In order to have lesser memory for the directory, invalidations per shared write is analyzed in just two applications. The results show that the bit-vector based tracking can be replaced with a few pointers and limited broadcasts to keep track of nodes and their accesses.

- Unloaded read miss latency in no contention is shown. What about number of each hierarchical misses in applications?

5 Was the evaluation in line with the stated user requirements?

- Not a very good job of evaluation considering the claims. Too many claims with limited results (at least the ones shown).
- Would like to see number of local misses, remote misses in applications not just duration of one unloaded read miss.
- Would like to see how Dash scales with different interconnection networks because the claim is that Dash is independent of interconnection.
- Would like to see how Dash OS performed on the Dash hardware.
- Does not really show contention of applications.
- Some general purpose applications, like MP3D, do not scale well with more processors.
- Evaluation of programming difficulty not done.
- Comparison of a message passing machine could have been done to see how it scales against Dash.
- Evaluation of prefetch mechanism to measure effectiveness and corresponding software level speculation overheads not done.
- Only three applications, of which MP3D does not scale very linearly (interprocessor communication, remote miss services, queueing delays). Mincut falls off after 64-nodes. Claims are that architecture is designed for scalability, but the conclusion is that it works well for 64 nodes.
- Could have given memory usage of applications, the size of the memory footprint.
- Dash was not compared to other machines. The speedup does not have a baseline comparison.
- Could have given the storage overhead of the whole directory (all distributed directories combined).
- Evaluation uses harmonic mean. Is this okay or is a weighted harmonic mean needed? Note: when MP3D is left out it increases harmonic mean to 12.8 versus 10.5 with MP3D.
- Would like to see how Dash scales with a few thousands processors (perhaps with a matrix benchmark).

- Because a real Dash system was built, it is feasible, but this does not prove that it will really scale to thousands of processors.

6 Was technology a factor in the problem or solution?

- Claim is solution is independent of interconnection network.
- Though not directly stated, better technology means lesser network delays.
- More processors can fit with better technology, and more processors can do more work.

7 Were new tools of software techniques introduced?

- Started work on "Jade" new parallel language for programmers to easily express parallelism.
- Dash OS: Modified version of Irix kernel with support for prefetch, update-write, queue-based locks, as opposed to a primitive kernel.
- Supports a parallel macro library for structured access to underlying hardware and O/S.
- A suite of high-level and low-level performance monitoring and analysis tools.
- Parallelizing compiler that does static data partitioning to improve locality, that exploits prefetching.
- Fetch-Increment, Update-write and Deliver Primitives.
- Performance monitoring tools can identify portions of code where the concurrency is smallest or where the most execution time is spent.
- Note: software was not the focus of the paper.

8 How may users with other requirements be affected?

- Relaxed consistency model is difficult to program. This implies users are given the responsibility of ensuring correctness of implementation.

- Queue-based lock chooses requester at random. Guarantees fairness but no priority can be enforced.
- Low contention applications may suffer software overheads due to queueing and prefetching, affecting performance.
- Directory memory does not scale very well due to bit-vector based management. Restricts machines with very large number of processors because of the cost produced and area taken by the directories.
- Since directory and interconnect costs are amortized over larger number of processors, programs with limited parallelism will have to pay these costs.
- Highly contended synchronization objects and heavily shared data objects create a hotspot problem that reduces network and memory throughput.
- Prefetch operations not very intuitive for inclusion by users in all applications, reducing programmability.
- Modification of existing code to include primitives to exploit hardware, limits backward compatibility.
- Applications exhibiting poor cache behavior or extensive read/write sharing could have significant delays during remote cache misses.
- Nonuniform memory accesses affect programming.
- With shared memory, programmers do not need to keep track of data. In message passing, programmers have to track data through programs.

9 How can the discussion of this paper be generalized in the context of the class?

- Scalable cache coherence is exhibited by the Dash parallel architecture. This is another example of supercomputing.
- Paper that addresses key aspect, scalability of parallel systems. Earlier papers were focussed on network delays and communication.
- Dash took something not usually scalable and made it scalable.
- A paper that identifies problems in its own architecture and addresses mechanisms to handle all of them.
- Paper claims to be enormously scalable but evaluation tells that it is good for 64 nodes.

- A paper that explains problems in designing for scalability, network scalability, software and hardware techniques to overcome the problems.

10 Summary and Additional Notes

The Dash multiprocesor parallel system showed a method for scalable cache coherence through the use of distributed directories. It also maintained a single shared address space to make the system easier to program. Several techiques were introduced for synchronization and bus architecture. At the time the paper was written, 64 processors was a very large system. However, the evaluation of Dash in this paper provided no evidence that Dash could scale to thousands of processors, as it claimed. The evaluation also failed to compare Dash to other systems, despite the performance evaluations done on Dash.

11 Further Discussion

- Intel and AMD multiprocessors are single address and cache coherent, but they will only scale to 16 or maybe 32 processor cores.
- The Dash paper did not evaluate a thousand or more processor case, but do we even need these "manycore" systems.
- Buses do not scale up to a thousand processors. Also, directories will get so large that we will not be able to put them in main memory because the many processors will be trying to get to directories. This means separate structures for directories will be needed.
- A hierarchy of network is a good idea that is feasible with packaging technology. Packaging technology influences architecture because the closer things are together, the faster they are, generally. So the way things are packaged affects the network.
- Large databases like machines with shared memory because database applications do not know where a large number of the data elements are which they are working on.