DISCLAIMER

The information contained herein is for informational purposes only, and is subject to change without notice. While every precaution has been taken in the preparation of this document, it may contain technical inaccuracies, omissions and typographical errors, and AMD is under no obligation to update or otherwise correct this information. Advanced Micro Devices, Inc. makes no representations or warranties with respect to the accuracy or completeness of the contents of this document, and assumes no liability of any kind, including the implied warranties of noninfringement, merchantability or fitness for particular purposes, with respect to the operation or use of AMD hardware, software or other products described herein. No license, including implied or arising by estoppel, to any intellectual property rights is granted by this document. Terms and limitations applicable to the purchase or use of AMD’s products are as set forth in a signed agreement between the parties or in AMD’s Standard Terms and Conditions of Sale.

AMD, the AMD Arrow logo, and combinations thereof are trademarks of Advanced Micro Devices, Inc. Other product names used in this publication are for identification purposes only and may be trademarks of their respective companies.

© 2014 Advanced Micro Devices, Inc. All rights reserved.
Abstract

Recently proposed 3D-stacked memory devices include multi-chip networking as well as memory access capability. By integrating a logic layer with the memories, efficient memory controllers and other useful logic may be coupled tightly to save memory access energy and provide enhanced functions for control, reliability, and computation. Memory system capacity may be expanded by creating a network of memory stacks that communicate with each other via a number of network hops. AMD believes such memory networks have the potential to do much more than merely forwarding memory requests and data. This paper outlines some of these use cases and advocates more research in this promising area.

1) Introduction

Future memory systems are expected to have a number of smart memory devices or nodes capable of accessing a local memory or forwarding requests to other memories. An example is a system with multiple 3D-stacked memories, each with their own logic die and each containing finite resources such as command and data buffers, memory controllers, and spare memory blocks for repair. The demands on each individual memory in the system may be vastly different, which could affect resources, latency, and bandwidth availability. Significant load imbalances may arise from multiple host processors sharing a network of memories in which pages of data with differing access frequencies are mapped into nearby memories.

We believe the memory architecture proposed in this report could ease bottlenecks by sharing resources across multiple controllers. At the same time, it could improve scheduling decisions by using global metadata flow to keep all stacks informed of each other’s utilization. Because there could be different demands on different memories (each with their own controller for their local address space), bottlenecks may occur due to demands on the memory-controller resources. Such bottlenecks include limited buffer space, bank contention, power budget, limited repair resources, and spare ECC and repair capacity. The system proposed in this report could be particularly useful for heterogeneous memory systems in which slow memory stacks (e.g., for non-volatile memories, or NVM) might run out of resources and become bottlenecks sooner than others.

The proposed system also contemplates the issue of memory controller load-balancing and enable resource-sharing in a multi-memory environment (e.g., a network of processors in memory, or PIMs).

In summary, this report discusses:

1. Continuous metadata flow and status-sharing among separate memory controllers.
2. Mechanisms to allow optimization based on the shared status information from other memories, and a means of requesting resources from other memories.
3. Autonomous local decision-making for memory command routing among controllers to reduce resource contention.
2) Proposed System

Figure 1: An example of the proposed memory network architecture. Several alternative architectures have been proposed in related work (e.g. [Kim13]). The system is composed of a number of host processors and memories, each with their own memory controllers and, optionally, PIM logic. The red lines show inter-memory communication paths. The memories might use a dedicated communication network as pictured, or pass messages through an existing interconnect.

Allowing smart memory devices to share data and resources with each other may improve system performance and efficiency. Figure 1 shows one possible implementation: Four host processors communicate with each other via one interconnect, and there might be an optional dedicated inter-memory communication channel as well. This channel allows each memory to report its status and resource availability, and to send requests to and receive requests from other memories to access shared resources to change configurations. Alternatively, the host interconnect may be used for inter-memory communication at the cost of some bandwidth.

Our intention is to provide each memory with a more global view of the state of other memories, with the information shared between neighboring memories or broadcast globally among all memories. The shared information would facilitate performance improvement by avoiding bottlenecks, whether they arise from interconnect links or busy memory controllers. In the proposed system, memories would share buffer and repair resources if they begin to run out on a stack. We describe possible usage scenarios in detail in the following sections.
3) Continuous State-sharing among Memories

The proposed memories (or their local control logic) would transmit their current states periodically, or on demand, to the control logic of other memory devices. This metadata could enable memories to request assistance from other controllers that have resources available to share. Possible examples of state that can be shared include:

- Per-bank average read and write latency.
- Average number of free buffer slots.
- Amount of remaining repair resources, including known-good RAM capacity, ECC check-bit capacity used, SRAM repair buffers, and spare through-silicon via (TSV) counts.
- Current link I/O bandwidth.

The metadata could be used locally in each memory controller to make routing and resource decisions, leading to a scalable system.

4) Mechanisms for Inter-memory Communication

Host-to-memory and memory-to-memory communication could coordinate to achieve the goals of sharing metadata between memory stacks. This may be achieved either via special messages among the hosts and memories, or transparently between the memories themselves. Here we outline some possibilities for each type of communication.

**Host-to-Memory messages** – Assuming a shared address space, any host may address any connected memory to the extent that there is an I/O path among them. The addresses generated by a host could be specified in full or by using a STACK SELECT command. This command implies that unless a new destination is selected, all future requests go to the addressed memory device. This could reduce the size of addresses that need to be sent across the bus with each memory request, reducing latency and bandwidth.

**Memory-to-Memory messages** – Metadata could be shared periodically among memory controllers via broadcast or on demand. The previously observed state history may be stored locally by each controller for future reference. Any requests to remote memories must traverse the network. The path taken by commands and data could be configured dynamically as a function of a number of parameters, known to the transmitting node. These decisions may be altered on the fly, depending on the global state seen by the observing memory node. Routing could take into account the following parameters:

- Total expected latency and available link bandwidth along the candidate access path.
- Expected availability of the target memory bank and buffers. For example, even if the target channel’s memory controller queue is almost full, if the sending node knows that the specific bank being addressed is not contended, it could increase the priority of the transaction to “high” and send it anyway, knowing that it will be serviced quickly and will not be subject to a long queuing delay.
5) **Autonomous Local Decision-making for Scalability**

Besides network routing decisions, various other command allocation and resource-sharing decisions may be made based on a local memory controller, benefitting from knowledge outside of its own memory device. The following points provide some examples of autonomous activities that might be coordinated between memories while remaining transparent to the host processors for ease of programming.

- The memory controllers buffer the requests and make decisions based on whether the transaction was targeted for local or remote memory, the state of the target, and intervening memory nodes for a remote access. This could make use of the memory-to-memory messages outlined in the previous section.
- If the request is for local memory, or the request reaches the target remote memory, it may be moved to a local-memory scheduler queue if enough space is available or if it has high enough priority. If scheduler resources are not available, commands and data could be buffered in another memory controller’s buffer space depending on availability. Because the local view of resource availability could be out of sync with the true state when a transaction arrives at a remote node, a denial response might be returned to the requester. If the request is buffered remotely, it becomes eligible to be returned to the target node when resources become free or a maximum age time-out occurs (when the command has to be handled). Use of other controllers’ resources may ease bottlenecks temporarily and allow forward progress of programs, particularly for write bursts when buffers fill up. This ought to be considered in conjunction with the additional latency incurred by performing remote accesses.
- In a similar fashion, repair resources may be shared among memory devices, allowing continued operation when all repair resources of one memory have been used. However, such sharing may have a performance impact.
- Memory scheduling buffer slots may be reserved by sending a special RESERVE command from one host or memory scheduler to another. RESERVE could be used when incoming high-priority command addresses that target a remote memory are decoded, or it could be sent by the host when it knows that the target memory is remote. The RESERVE command arriving at the destination node is intended to reserve a resource (e.g., command queue slot) for a period of time in anticipation of the arriving high-priority command. When the command arrives, the reserved slot is replaced with the command. This could reduce latency for the command.
6) Coherence Issues

Some of the concepts described in this work, such as remote command buffering, may lead to coherence problems. For example, if a write transaction has been moved temporarily to another memory’s input queue when the target has run out of space, data in the target memory location is invalid. To maintain coherence, we propose a few alternatives:

- The memory controller buffers may be treated as cache memories with a coherent interconnect across controllers. This could be supported directly by PIM networks, in which part of their caches can be reserved to function as memory-controller resources (buffers, repair regions, etc.).
- A broadcast mechanism (e.g., emulating a shared bus) or directory scheme may be used, in which all read requests have to check all memory-controller write buffers for a copy of the data before issue.
- Offload only data, and not commands, to other nodes. Any read command for which the write has offloaded its data somewhere else that reaches its destination would first retrieve the remote data by following a pointer to the resource where data actually is held.

7) Related Work

The Hybrid Memory Cube [HMC] specification suggests that dedicated I/O links might be assigned as pass-through, to access other networked memory stacks. Instead, we propose a different addressing mechanism for other stack nodes that does not need dedicated links. HMC also suggests using buffer tokens for flow control, indicating that a sender can send packets only when there are free buffer resources at the target controller. We add to this concept by reporting available buffer space, repair resources, spare area, known-good address capacity, and average latency, to name a few state variables. We also propose allowing any memory controller to reserve another memory controller’s resources.

In [Kim13], they proposed memory-centric system interconnects. They analyzed several different interconnect designs from a performance perspective, and proposed a pass-through architecture to reduce latency for accesses to remote memory nodes. In [Ham13], they proposed disintegrated memory controllers to support scalability for heterogeneous memories. The protocol-specific features of the controller are separated and associated with each memory type independently. They also proposed using controller-to-controller links for data migration. This report suggests additional capabilities for interconnected memory devices in an effort to improve performance and/or efficiency.

8) Conclusions

Designers may leverage the techniques outlined in this paper to increase performance, save energy, and/or increase system lifetime in the presence of errors. It is apparent that more effective optimization may depend on pooling resources and providing a higher-level view of global state to independent devices. However, information sharing may add to system complexity and should be managed carefully so that it does not affect performance or energy negatively.
9) References

