Volume 11 Issue 03 Published, August 22, 2007 ISSN 1535-864X DOI: 10.1535/itj.1103.01

Intel® Technology Journal

Tera-scale Computing

Integration Challenges and Tradeoffs for Tera-scale Architectures

More information, including current and past issues of Intel Technology Journal, can be found at:
http://developer.intel.com/technology/itj/index.htm

Integration Challenges and Tradeoffs for Tera-scale Architectures

Mani Azimi, Corporate Technology Group, Intel Corporation
Naveen Cherukuri, Corporate Technology Group, Intel Corporation
D. N. Jayasimha, Corporate Technology Group, Intel Corporation
Akhilesh Kumar, Corporate Technology Group, Intel Corporation
Partha Kundu, Corporate Technology Group, Intel Corporation
Seungjoon Park, Corporate Technology Group, Intel Corporation
Ioannis Schoinas, Corporate Technology Group, Intel Corporation
Aniruddha S. Vaidya, Corporate Technology Group, Intel Corporation

Index words: tiled architecture, on-die interconnect, cache hierarchy, communication protocol

ABSTRACT

Tera-scale processors promise to offer an unprecedented concentration of computing power and enable novel usages and applications. The computing power may be provided by a combination of general-purpose cores and special-purpose (fixed or programmable) computing engines. Further, Moore's law enables the integration of additional system resources to the processor die. However, the realization of tera-scale architecture is challenged by on-die power dissipation, wire delays, off-chip memory bandwidth, process variations, and higher failure rates. These challenges create opportunities for architectural innovation. One of the ways to address these challenges is through the use of a "tiled" architecture: the die is divided into a large number of identical or close-to-identical, tiles that are interconnected using a scalable and energy-efficient interconnect. This modular approach enables ease of layout and rapid integration of different blocks. Limited off-chip memory bandwidth requires innovations in the cache hierarchy, memory subsystem, and coherence protocol. We present an architectural vision for the tera-scale processors and discuss the performance, scalability, and manufacturability aspects of the uncore. We articulate key challenges and point to candidate solutions for these challenges.

INTRODUCTION

Over the last few years, dual-core processors have become mainstream in desktop, mobile, and server platforms due to their ability to deliver higher system performance more efficiently than single-core processors. The trend towards higher core counts is continuing strong with quad-core processors establishing an increasing presence across all market segments.

Industry experience with small-scale shared memory multiprocessors enabled a relatively effortless integration of a small number of processors into a single die. Moving beyond a small number to tens or hundreds of processor cores at the same time as other platform ingredients such as memory controllers, I/O bridges, and graphics engines find their way to the processor die, introduces significant challenges to the infrastructure that ties all these together. This infrastructure includes the on-die interconnect, the cache hierarchy, the memory, the I/O, and system interfaces. In this paper we use the term *uncore* to collectively refer to all the elements in the processor die that are not computing engines.

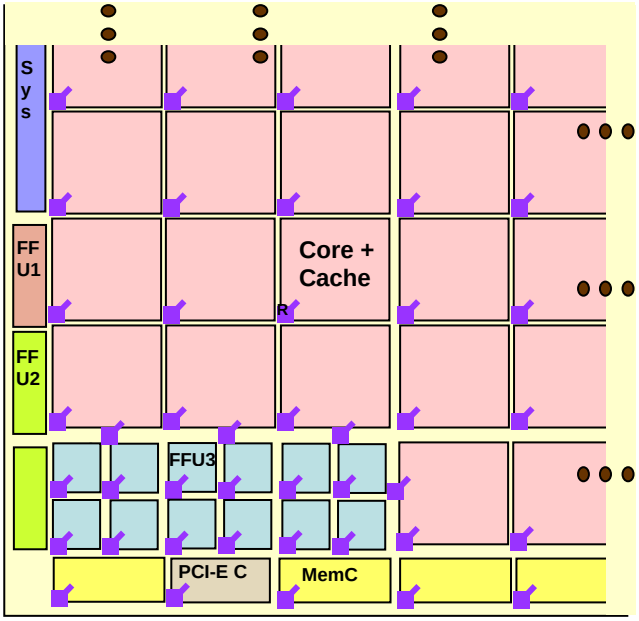
The tera-scale architecture uncore must be capable of satisfying the communication requirements of a large number of cores, fixed function computing engines, and the external memory and I/O system. In order to scale effectively, the uncore must find ways to keep the off-die bandwidth manageable and within the constraints of cost, power, and high-speed signaling technology. The uncore must be able to offer significant flexibility to assign computing resources to concurrently solve different problems. It must include mechanisms to enable high-volume manufacturing by enhancing reliability in the presence of increasing architectural complexity and

decreasing silicon geometries. Moreover, it must perform its functions within a constrained power envelope.

This paper is organized as follows. First, we describe the architectural vision for tera-scale processors. Second, we focus on the challenges and opportunities imposed by the tera-scale architecture in the key uncore elements such as the on-die interconnect, cache hierarchy, and memory architecture. We conclude with a summary of the key challenges, opportunities, and directions outlined in this paper.

ARCHITECTURAL VISION

The tera-scale architectural vision, as shown in Figure 1, takes the integration trend to its logical progression by consolidating not only a large number of general-purpose computing cores but also special-purpose computing engines (e.g., texture units, shader units, fixed function units), and platform elements, such as memory and I/O controllers, in a single die. A tera-scale processor may also include a system interface to allow multiple such processors to connect with each other and with other system peripherals.



FFU: Fixed Function Unit, **Mem C:** Memory Controller, **PCI-E C:** PCI-based Controller, **R:** Router, **Sys I/F:** System Interface

Figure 1: Tera-scale architecture: high-level block diagram

The tera-scale architecture uncore consists of the following key elements:

- A scalable high-bandwidth, low-latency, and power-efficient interconnect to connect the computing and platform elements together and allow them to exchange information with each other, access memory, and communicate with the rest of the system.
- A cache hierarchy that allows the multiple computing elements to effectively utilize and share the on-die memory resources.
- A scalable, high-bandwidth memory architecture that can effectively feed the large number of computing elements.

We expect tera-scale processors to be highly optimized for specific market segments through variations in the number of computing engines, by having different types of fixed function blocks, and having a different type and number of memory and I/O resources. Not all building blocks require the high bandwidth and low latency offered by the scalable interconnect. We expect blocks that are not candidates for integration into the main interconnect and cache hierarchy to be attached to auxiliary interconnects suitable for specific needs.

ON-DIE INTERCONNECT

The on-die interconnect is the primary "meeting ground" for various elements of the tiled architecture in Figure 1. Given its central nature, there are certain basic requirements for the on-die interconnect:

- **Scalability**: Given the requirements of a large number of nodes (agents) on the interconnect (high tens to low hundreds), we realistically desire a) a sub-linear growth in average distance with number of nodes, b) a relatively low per-hop latency through each switch under no-load conditions, and c) manageable growth in latency under loaded conditions.
- **Partitionability**: The topology, with appropriate routing support, should enable the tera-scale architecture to be dynamically partitioned to achieve both performance and fault isolation.
- **Fault tolerance**: The tera-scale architecture with its tiled structure has the potential for a graceful degradation under faults. Further, with the expected impact of variations on process technology, there is a greater susceptibility to "performance" faults (discussed in the next section). Hence, the topology, with appropriate support, should support routing around faults.
- **Validation and testing**: The interconnect should provide support for testing and validation, which is critical for high-volume manufacturing. For example, an interconnect that uses a deadlock-free routing approach is easier to test and validate compared to one using deadlock-recovery based routing.

- **Regularity**: In order to make the design of the tera-scale chip tractable, it is imperative that the layout, planning, and design of each tile be done in such a way as to make the tile physically symmetric. Thus integration may be achieved largely through abutment of tiles. To that end, each "tile" needs to plan its global wiring tracks.
- **Flexibility and design friendliness**:
  - Designs should facilitate "choppability" so that with minimal redesign effort, a range of market segments can be satisfied.
  - Furthermore, the basic router design should not change as the underlying parameter, for example, as the number of processors, changes with each process generation.
  - The tiled architecture will have different-sized tiles arising possibly from the need for heterogeneous cores (e.g., some suited for throughput and others suited for single-thread performance), specialized engines, fixed function units, etc. The on-die interconnect design needs to incorporate the needs of each by, for example, clustering multiple low-bandwidth engines into a single routing agent.

Candidate Topologies

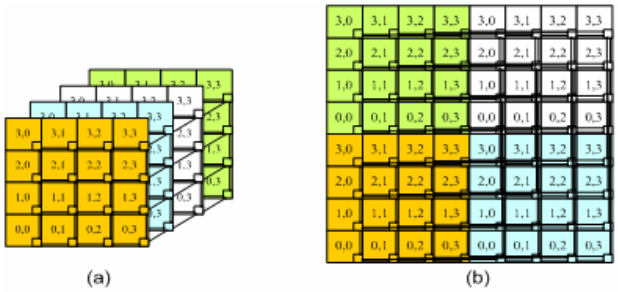


Figure 2: 2D embedding of a 64-node 3D-mesh network

In off-chip networks, both the number of links (which has a bearing on the topology) and the link widths are determined by pin-out limitations. In on-chip networks, this limitation is absent. The topology choices (apart from the number of routing agents that need to be supported) are determined by the wiring density and router complexity in terms of area and power. The wiring density is in turn determined by the number of metal layers available and the directionality constraints (uniform availability of horizontal and vertical metal layers), as well as the need for the topology to be embedded in 2D space. The latter point implies that for higher (greater than 2D) dimensional networks, *topological adjacency does not lead to spatial adjacency* [14]. This has significant implications both on the wire delay and on the wiring density. Consider the embedding of the 3D mesh in Figure 2. For the longest hop, the topological distance is 9, but three of these hops span half the length of the die. Hence, the distance in tile span units is 18!

Considering wiring density, router complexity, and design friendliness, tera-scale architecture topologies will be fixed-degree and will have a low dimension (1–2) in the foreseeable future. Thus, ring and 2D torus/mesh networks and their many variants [2] will be candidate topologies.

In the rest of this section, we use the 2D mesh as an example topology for illustrative purposes only.

Interconnect Microarchitecture

The main challenge in on-die networks is to achieve the required bandwidth and latency under the constraints of power and area. While topological choices, as mentioned above, help with bandwidth scaling and keep latencies manageable, they come at increasing power costs.

Wang et al. [28] show that the router power is almost 2x the power of the wires in the MIT-RAW [25] chip. Further decomposition shows that the power is roughly spent as much in the switch (crossbar) as in the buffers.

Wang et al. also propose segmented and cut-through crossbars as possible solutions to reduce crossbar power. Meanwhile Nicopoulos et al. [21] reduce buffer power through careful microarchitectural techniques to minimize the number of buffers required for the same network throughput.

Kumar et al. [16] observe that certain paths in a 2D mesh are common for a number of flows (between different source/destination pairs), and thus traffic traveling on these trunks could be aggregated and switched together—thus avoiding the need for packets to stop and be buffered at intermediate nodes. This in turn saves buffer power and reduces contention on network resources—the latter helping to improve the throughput and thus eventually the energy characteristics of the network.

Traffic Classes

Emerging workloads [10] may see different classes of traffic overlaid on the on-die network. Taylor et al. [25] and Gratz et al. [11] use a network fabric to route operands between different clusters of functional units. It is conceivable that the different cores of the tera-scale processor may be used to realize a virtual superscalar microarchitecture [29], thus necessitating fine-grained communication of operands and control, in addition to the cache coherent and message-based communication in the cache-memory subsystem.

Furthermore, as media applications become a dominant consumer of compute capacity on a chip, they will place hard real-time constraints on different shared resources such as cache and interconnect. In addition, running disparate applications on the same multi-core is likely to result in bandwidth over-subscription by some applications at the cost of starvation of some others. Careful rationing of bandwidth while providing latency guarantees to the necessary applications will require careful architecture definition and design of the fabric.

Resiliency

The need for fault tolerance arises from both an increased susceptibility to faults and the opportunity to gracefully degrade in a tera-scale environment.

Future process technology trends are likely to adversely affect the resilience of a tera-scale processor chip. Such trends might include process variations becoming a more significant determinant of overall performance and insufficient burn-in time to weed out infant mortality. Consequently, there is a higher probability of in-field failures and accelerated degradation potentially shortening the expected lifetime of the product [6].

Interconnect Fault-tolerance Approaches

The following mechanisms can be adopted for addressing resilience in a tera-scale processor interconnect. These approaches can be used either to address true faults or performance faults (i.e., when underperforming or "out-of-spec" tiles are treated as failed tiles).

Sparing: Spare processor tiles paired with network interfaces and switches can potentially solve a multiplicity of fault scenarios including increased possibility of in-field failures. Upon detection of failures in some tile components, spare tiles are activated after the interconnection network is reconfigured as shown in Figure 3.

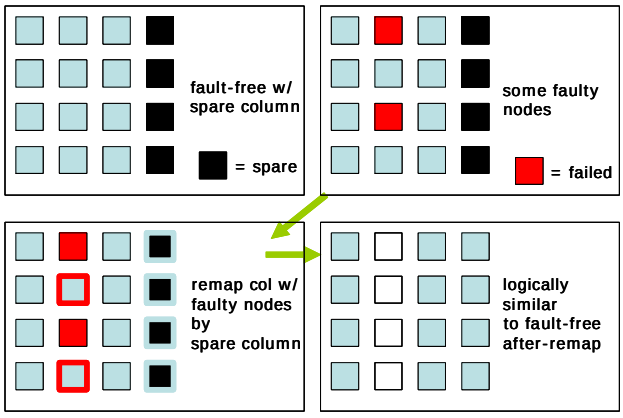


Figure 3: Illustrating use of spare tiles that maintain original topology

Fault-tolerant Routing

Fault-tolerant routing support is required in the interconnect to enable reconfiguration of the system components in the presences of failed tiles and routers. Upon system reset/initialization, a fault and topology discovery algorithm is run to determine the location/identity of failed components and to mark them in the interconnect. Other regions also need to be marked safe or unsafe from a deadlock-free routing perspective. A fault-tolerant routing algorithm is then configured to route around faulty and unsafe regions. Figure 4 shows faulty (dead) nodes in the interconnect. A few additional nodes are marked unsafe so as to form rectangular fault regions. After the fault-tolerant routing algorithm (such as in [5]) is configured, all working (and spare nodes, if sparing is used) tiles in the fabric can communicate with each other.

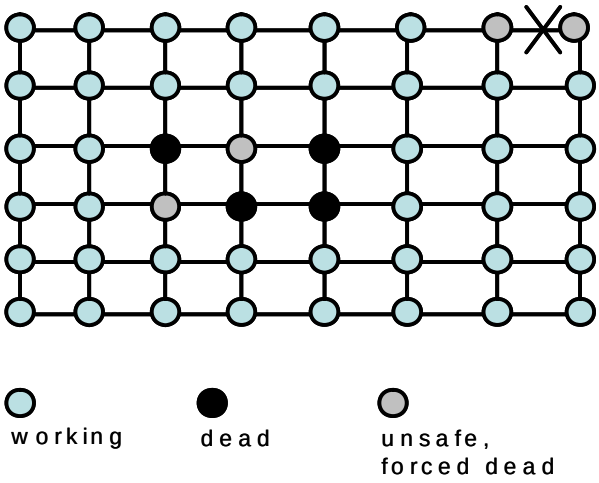


Figure 4: Illustrating need for fault-tolerant routing

The fault-tolerant routing algorithm should be simple to implement, deadlock free, and be able to handle a wide variety of faults. It is also desirable for the routing algorithm to adaptively respond to congestion that may occur in the network due to the additional effort needed to route around fault regions.

Partitioning for Performance Isolation

We expect several partitions to be supported on a tera-scale processor—each partition with a fraction of the total number of processing units, special-purpose units, and other platform elements. There may be several different usage models for a partitioned tera-scale processor including multiple server partitions in a consolidated "server on chip" or, for example, multiple virtual appliances on a home server.

It is desirable that the performance of each of the multiple partitions on a tera-scale processor be unaffected by the performance of other partitions. Some partitions may be more sensitive to performance perturbations from

other partitions or may have stricter Quality of Service (QoS) requirements.

Performance isolation: Performance isolation relies on confining intra-partition communication of a given QoS sensitive partition to physically distinct components of the on-die interconnect from that of other partitions. Figure 5 shows a configuration with three isolated partitions where traffic generated in one partition does not interfere with traffic from another partition.

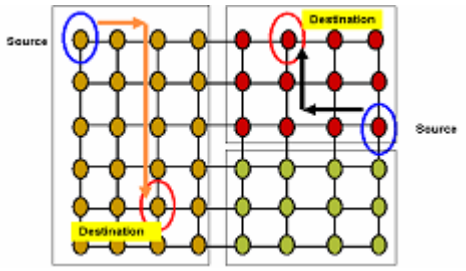


Figure 5: Performance isolation in a 2D mesh with rectangular partitions

Virtualization of Network Interfaces

In order to better realize a uniform interface that can comprehend the diverse needs of accelerators, fixed function units, general-purpose processor cores, cache blocks, etc., it is desirable to formalize and possibly even export the interconnect as an abstraction to the application programmer and/or the run-time system. Thus, one could envision a programmer fine-tuning an application's inter-processor communication requirements, based specifically on that application. For example, a media application requiring little support for cache coherence may be better served in power and performance through a direct send/receive interface. Similarly, the programmer may want to commandeer different levels of resources of the interconnects, i.e., number of buffers, switching priority, or bandwidth, and leave the rest of the hardware for use by another application.

A network interface that can allow this level of control and flexibility, yet can achieve good performance would be powerful. In addition, such a network makes for easy and rapid integration of multiple IP blocks that conform to the same interface.

CACHE HIERARCHY AND COHERENCE PROTOCOL

Diversity of workloads and concentration of compute resources in the tera-scale architecture put tremendous demands on the cache hierarchy and coherency protocol. This requires a flexible cache organization that can adapt to workload demands and puts minimal restrictions on the software to fully realize the performance potential. The associated coherency protocol needs to be efficient and scalable. It should also be flexible in terms of the requirements it imposes on the building blocks of the tera-scale architecture. In this section we highlight the challenges and tradeoffs associated with the cache hierarchy and protocol and point out potential directions for tera-scale architecture.

Developing parallel applications to harness and effectively use the massively parallel tera-scale processors is likely to be the key challenge for tera-scale computing. Many parallel programming models and languages have been deployed in different contexts over the last few decades and in fact, parallel programming remains an area of active research. A clear lesson, however, that we can draw from the history of parallel computing to date, is that hardware shared memory has proven to be a particularly successful programming model for general-purpose systems. Accordingly, tera-scale architecture should include first-class hardware support for shared memory. Industry and academic experience with coherence protocols for large-scale, hardware-shared memory machines has demonstrated that shared memory machines scaling to hundreds of processors can be successfully built. In fact, implementing a message-passing library such as the Message Passing Interface (MPI) over hardware-shared memory often results in higher bandwidth and lower latency than equivalent implementations using specialized low-latency cluster networks [19]. In addition, hardware support for shared memory will allow tera-scale processors to support common operating systems assuming that such operating systems overcome any existing scalability bottlenecks to harness the capabilities of tera-scale architecture.

A cache hierarchy should efficiently support a wide range of programming models and workloads. These are some important classes:

- Multiprogrammed workloads where there is no communication and data sharing among the processes running in different cores.
- Workloads with a mix of scalar and parallel sections. The performance of these workloads on tera-scale architecture is limited by the performance of the scalar section as indicated by Amdahl's law.
- Highly parallel workloads, where most of the computations can be parallelized. These workloads may exhibit one or more of the types of parralelism as described below:
    - Thread parallelism: Each thread may be similar or very different from each other and may or may not share data with other threads. Threads are created based on the granularities exposed by the application and then scheduled on available hardware contexts through task queues or other

constructs. Examples of this programming model can be found in transaction processing and Web applications.

- Data parallelism: A similar task is performed on different data sets, where some data may be shared between tasks. Applications are more structured, and algorithms are typically modified to fit the underlying cache organization. The number of threads used in this model is typically the same or less than the number of hardware contexts available. Examples of this programming model can be found in media, numerical analysis, and data-mining workloads.
- Streams: Programs are structured as kernels where input data are processed and output data are fed into other kernels. In this model threads (or kernels) are statically scheduled to hardware contexts. Within each kernel, thread- or data-level parallelism constructs can be applied to break tasks into ever finer sizes. Examples of this programming model can be found in media and graphics applications.

Cache Organization

A combination of different workloads and different types of parallelisms within these workloads presents unique architectural and design challenges for the cache hierarchy of tera-scale architecture. Architectural challenges center on the organization and policies associated with the cache hierarchy to meet performance, scalability, and energy-efficiency goals. Cache organization deals with the number of levels in the cache hierarchy, and with the size, associativity, latency, and bandwidth parameters at each level. Cache policies determine accessibility, allocation, and eviction policies to effectively utilize on-chip cache resources.

The objective of a cache hierarchy is to minimize the latency to frequently accessed data. In a traditional uniprocessor cache hierarchy, we move cache blocks closer and closer to the core through the levels in the cache hierarchy, based on access frequency. The same principle applies to multi-core cache hierarchies, but we have to take into account whether cores have to share a given level in the cache hierarchy or whether a level is implemented as a single physical block or as multiple physically distributed banks with non-uniform access latency to each bank.

In multi-core processors released over the last few years, the first one or two levels in the cache hierarchy are private to each core. However, different designs have pursued a range of options in sharing the last-level cache. In some designs such as those described in [20], the last-level cache is private to a core. In others, such as those described in [18, 23], the last-level cache is shared among multiple cores.

In CMPs with only a few cores, the last-level cache is being implemented as a single physical block with uniform access latency to the entirety of the cache by all the cores sharing it. As the number of cores and cache banks increase, physically distributed caches become attractive from a physical design perspective [15]. Moreover, by collocating a portion of the cache with a subset of the cores, there is an opportunity to reduce access latency to a portion of the cache, instead of offering equally high latency to all the cache. Figure 6 summarizes different multi-core cache organizations according to their suitability for the types of workloads, assuming a distributed multibank last-level cache.

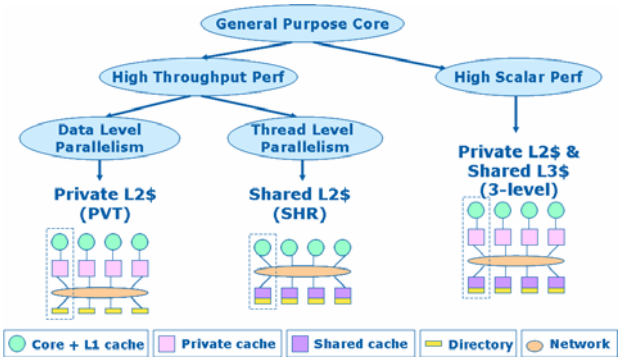


Figure 6: Cache organization options for multi-core architectures

In a tera-scale processor with a last-level cache physically distributed across multiple tiles, private and shared caches introduce distinct tradeoffs. A shared cache design increases effective cache capacity because only a single copy of a block shared by multiple cores resides in the cache. The downside is that any given block, whether private or shared, may be placed in a tile arbitrarily and be far away from the core(s) using it. In contrast, a private cache design will have all blocks used by a specific core on its local tile. However, since read-shared blocks will be replicated in multiple tiles, the effective cache capacity is reduced, and off-die traffic may increase.

Recent work suggests that other hybrid alternatives are possible: these combine the advantages of private and shared caches while avoiding their shortcomings. The key observation is that in a physically distributed cache design where some cache banks are closer to a specific core than others, one can optimize cache performance by optimizing the placement of blocks in the cache banks so that they are closer to the point of use. A number of approaches in the literature have been proposed to achieve this [4, 9, 30, 31]. Such approaches are beneficial in any multi-core processor with differential access

latency to a given portion of a shared cache, but are particularly effective in a tera-scale processor where there is large variation in the latency to access the cache in different tiles.

Fundamentally, all approaches have the following key policies to set: initial placement, read-shared block replication, block migration, and eviction. The initial placement policy defines where a block is placed in the cache hierarchy when it is fetched from memory. The replication policy determines whether multiple copies of a read-shared block can coexist in different cache banks. The block migration policy determines whether a block will move between tiles in response to processor accesses. Finally, the eviction policy determines what happens to a block evicted from a cache bank. Private and shared cache designs represent the end points in the design space with regard to these specific policies. For example, in a private design, a block is initially placed in the cache of the requesting core, while in a shared design, a block is placed in a cache bank determined by the physical address of the block (home tile). Hybrid approaches combine policies from private and shared design or introduce new policies to perform better than either private or shared designs, or they even dynamically switch between competing policies based on application demands. For example, the Adaptive Selective Replication (ASR) [4] determines the replication level within the context of a private cache design based on program behavior.

The enormous computing power available in tera-scale design implies that many applications (or applications consisting of many concurrent functions with distinct caching behavior) will be running concurrently (e.g., games physics with game AI and graphics rendering). Accordingly, when a level in the cache hierarchy is shared among multiple cores in the presence of diverse per-core access patterns and working sets, destructive interference can occur. One of the causes of destructive interference is the suboptimal behavior of the least recently used (LRU) replacement policy, typically implemented in processor caches, when the application workload exceeds the cache capacity. Sharing a cache level among multiple threads can further exacerbate the problem. This is a well known issue for any shared cache, including page disk and file system caches. Recent work in this area, however, shows some promise of success [22].

In its generalized version, the tera-scale architecture is a collection of modular and heterogeneous building blocks with well defined interfaces. Such a heterogeneous collection of elements puts its own unique requirements on the cache hierarchy, and meeting these with a single set of caching policies and a single cache hierarchy is quite challenging. For example, if an incarnation of tera-scale architecture is a collection of several general-purpose processors, some graphics coprocessors, a few network accelerators, a security coprocessor and so on, each of these processors exhibit very different data footprints and locality characteristics. Satisfying their needs through a unified cache hierarchy is challenging and requires further exploration.

Cache Coherency

The cache coherency protocol for tera-scale architecture must be scalable to a large number of caching agents and must enable efficient utilization of on-chip resources. The choice of a coherency protocol is closely linked to the cache organization and the interconnect. For example, a protocol designed for cache organization without any shared caches may be designed to keep precise information about the lines present in private caches, such that off-chip reads and writes are minimal. A snoop broadcast protocol is suitable when there is a broadcast interconnect, but it cannot be scaled.

On-chip interconnects are capable of providing an order of magnitude smaller latency and an order of magnitude higher bandwidth than off-chip socket-to-socket interconnects. Therefore, latency and bandwidth optimizations may not seem to be the primary goals for an on-chip coherency protocol. However, since tera-scale processors are expected to have a concentrated density of computing throughput, they do impose tremendously high bandwidth demands on the interconnect. Since the power delivery, cooling, and off-chip bandwidth available to each chip is not scaling with process technology, the protocol must enable improved utilization of on-chip cache structures, and the interconnect overhead, because of the protocol, must be kept to a minimum to gain the maximum performance under these limits.

Directory-based protocols have been widely used in large-scale, multichip multiprocessors [7, 8, 17, 24], where a directory is used to keep track of copies of blocks in different caches. The same concept can be applied to on-chip cache coherence protocols in tera-scale architecture with some modification. As illustrated in Figure 7, a directory consists of entries corresponding to lines in caches where each entry has a state field and a field to store the identities (indicated as pointers in the directory structure in Figure 7) of the caches with a copy of the block. The state field indicates if a block may be present in one of the caches and the possible states the cached copies could be in. For a directory that is inclusive of all the on-chip caches, a directory miss or a state of I (invalid) in the directory indicates that none of the caches have a copy of the block; a state of S (shared) indicates that some caches may have copies of the block in Shared state; and a state of X (exclusive) indicates that one of the caches may have a copy in either Modified, Exclusive, or

Shared state. When a directory entry is in S or X state, the identity field identifies the cache(s) with a copy of the memory block. The identity information can be stored in various ways, either as a set of bits with 1 bit for each cache (called full bit map), 1 bit for a group of caches (called coarse bit map) or a limited set of explicit cache identities with a mechanism to handle overflows. The cost of a full bit map directory may be acceptable for the first few generations of tera-scale architecture; however, a more compact representation may be desirable for further scalability.

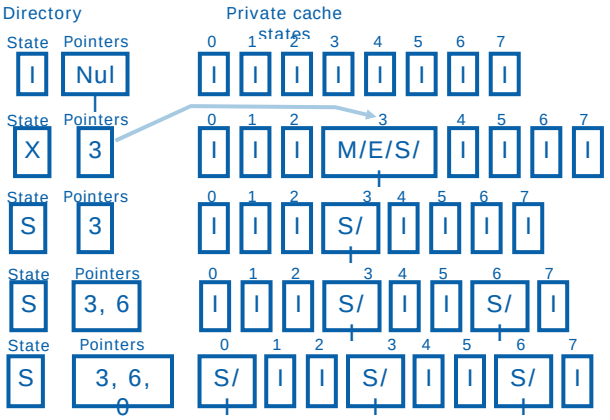


Figure 7: Directory structure to track cache lines

Since the purpose of the directory is to keep track of copies of a cache line in different private caches, the size, associativity, and replacement policy of the directory needs to provide adequate coverage for the total capacity of the private caches. Some designs combine the directory information in the same structure as the cache at the higher level in the hierarchy (if there is one), which may reduce complexity and area at the expense of some performance disadvantage due to conflicting policy requirements on the directory and cache. The cost and scalability or directory structures may start becoming a problem when the number of entities being tracked becomes very large. At that point, mechanisms to reduce directory size [8] or distributed directory [12] implementations may have to be considered.

The enormous amount of computing resources in a tera-scale platform enable a richer set of interactions between the computer and the end user than previously was possible. These include speech, motion and gesture recognition, enhanced visual effects, etc. often within virtual worlds where multiple users directly interact with each other. Interactions with the physical world introduce real-time considerations, and the tera-scale architecture must properly address them. Caches, however, interact in unpredictable ways with real-time applications. For this reason, processors targeted to interactive applications often include hardware mechanisms to allow applications to control the caching behavior to the point where one can reason about their expected performance [1]. Accordingly, the tera-scale cache hierarchy should include support in the form of locking primitives or similar mechanisms to allow applications to keep critical data in the caches. The exact form of such support is an area of active research.

Tera-scale architecture may also require much tighter integration of off-chip memory and I/O interfaces to take full advantage of its compute capabilities. Therefore, the on-chip protocol must enable optimizations for efficiently accessing local memory and for interacting with other auxiliary engines such as special-purpose co-processors and I/O controllers.

FEEDING THE BEAST: MEMORY ARCHITECTURE

With substantial increases in the computation power on a single die, one faces the challenge of feeding it with enough data bandwidth. For a small class of applications where the memory footprint is small, the memory accesses will mainly be exercising the on-die caches. For the majority of applications, a major increase in off-chip memory bandwidth is required. This manifests itself in two ways: (1) providing power-efficient high-speed off-die I/O; (2) providing power-efficient high bandwidth DRAM access. The former has seen steady progress in the past decade, but not at the required pace. The latter may require a new look at DRAM core and I/O design.

The first step to addressing the memory bandwidth challenge can be more efficient storage or improved management of the on-die storage. For example, embedded DRAM [3] helps to increase the density of on-die storage compared to SRAM. Efficient management of on-die storage by avoiding duplication of data in the cache hierarchy, as discussed in the previous section, is another step in increasing the effective capacity of on-die storage.

Integration of DRAM, e.g., GDDR memory, inside the processor package can offer more control over the I/O channel and thus allow a higher bandwidth, compared to crossing of package to motherboard-connector-DIMM path. Recent works have demonstrated methods of using 3D stacked SRAM to offer a low capacity high-bandwidth option, e.g., Intel's tera-scale prototype [13] and IBM's work on 3-D integrated circuits [26]. The freedom in the footprint design of such SRAM devices enables power-efficient solutions; however, limited capacity of such devices limit their application. 3D stacking of multiple DRAM dies can improve the memory capacity, but requires dense Through Silicon Vias (TSVs) to allow the required concurrency of accesses to independent DRAM banks.

Tera-scale Architecture Prototype

The Intel® Teraflop processor [27] is a prototype of some of the elements of tera-scale architecture. The Teraflop processor realizes an 80-core prototype with a 2D-mesh interconnect architecture that reaches more than 1Tflops of performance dissipating less than 100W of power. This illustrates the potential of the tera-scale architecture and validates the efficacy of some of the architectural building blocks.

SUMMARY

Tera-scale architecture presents tremendous challenges and opportunities to take advantage of Moore's Law. As discussed in this paper, the architectural and design tradeoffs for tera-scale architecture are unique to this architecture. A very high level of integration and the presence of heterogeneous building blocks necessitate a modular and scalable on-chip interconnect. Based on the organization, architectural building blocks, and physical design constraints, we expect ring, 2D-mesh, or similar topologies to be an attractive option. Interconnects with switches, such as 2D-mesh, though better in utilizing wiring tracks, bring their own challenges in terms of achieving aggressive latency targets within an acceptable power budget.

With shrinking device geometries and resulting increases in process variability and device failure rates, careful consideration needs to be given to get maximum performance without excessive cost through overly conservative designs. A flexible on-chip interconnect can play a role in dealing with variability and in-field failures by adapting to an optimal operating configuration through provisioning for fault-tolerant routing. A flexible interconnect can also be used to provide additional functionality, such as improved quality of service and performance isolation, to make tera-scale architectures more useful.

Providing adequate memory and I/O bandwidth to satisfy the needs of large numbers of compute engines in tera-scale architecture is a major challenge. Some of these can be addressed through using on-chip caches more effectively such that the needs of off-chip memory bandwidth are reduced. A higher integration of system components on tera-scale architecture also reduces pressure on memory bandwidth by avoiding the need for I/O controllers and compute engines to exchange data through the caches rather than memory. Technological approaches to improve the available memory bandwidth are also an active area of exploration, ranging from 3D stacked memory to higher speed memory interfaces, but they do have their own challenges, such as limited memory capacity and higher power consumptions, respectively.

In conclusion, tera-scale architecture is definitely in the not-too-distant future of mainstream computer architecture. Its realization, however, poses some challenges and a rich set of problems for researchers both in academia and industry. Problems and some solution strategies related to the "uncore" have been presented in this paper.

ACKNOWLEDGMENTS

We thank Yatin Hoskote, Dennis Brzezinski, David James, Mani Ayyar, Ching-Tsun Chou, Rama Menon, Saikat (Roy) Saharoy, Theodore Tabe, Hari Thantry, and Jianping (Jane) Xu for their contributions to material presented in this paper.

REFERENCES

[1] J. Andrews and N. Baker, "XBOX 360 System Architecture," *IEEE Micro*, March–April 2006.

[2] J. Balfour and W. J. Dally, "Design Tradeoffs for Tiled CMP On-Chip Networks," *International Conference on Supercomputing*, June 2006.

[3] J. Barth et al., "A 500MHz Random Cycle 1.5ns-Latency, SOI Embedded DRAM Macro Featuring a 3T Micro Sense Amplifier," *IEEE International Solid-State Circuits Conference*, Feb. 2007.

[4] B. M. Beckman, M. R. Marty and D. A. Wood, "ASR: Adaptive Selective Replication for CMP Caches," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture* (Micro), Orlando, FL, December 2006.

[5] R.V. Bopanna and S. Chalasani, "Fault-Tolerant Wormhole Routing Algorithms for Mesh Networks," *IEEE Trans. Computers*, vol. 44, no. 7, pp. 848–864, July 1995.

[6] S. Borkar, "Challenges in Reliable System Design in the Presence of Transistor Variability and Degradation," *IEEE Micro*, vol. 25, n. 6, pp. 10–16 Nov.–Dec. 2005.

[7] F. Briggs et. al., "Intel 870: A Building Block for Cost-Effective Scalable Servers," *IEEE Micro*, March–April 2002, pp. 36–47.

[8] D. Chaiken, C. Fields, K. Kurihara, A. Agarwal, "Directory-based cache coherence in large-scale multiprocessors," *IEEE Computer*, June 1990, pp. 49–58.

[9] J. Chang and G. S. Sohi, "Cooperative Caching for Chip Multiprocessors," in *Proceedings of the 33rd International Symposium on Computer Architecture*, Boston, MA, June 2006.

[10] "Compute-Intensive, Highly Parallel Applications and Uses," *Intel Technology Journal*, Volume 09 Issue 02, May 2005.

[11] P. Gratz, K. Sankaralingam, H. Hanson, P. Shivakumar, R. McDonald, S. Keckler, D. Burger, "Implementation and Evaluation of a Dynamically Routed Processor Operand Network," *IEEE/ACM International Symposium on Networks-on-Chips* (NOCS), May 2007.

[12] "IEEE standard for Scalable Coherent Interface (SCI)," *IEEE P1596*, August 1993.

[13] Intel News Release, "Intel Develops Tera-Scale Research Chips," Sept 26, 2006, at http://www.intel.com/pressroom/archive/releases/20060926corp_b.htm.

[14] D. N. Jayasimha, B. Zafar, Y. Hoskote, "On-die Interconnection Networks: Why They are Different and How to Compare Them," *Technical Report*, Microprocessor Technology Lab, Corporate Technology Group, Intel Corp.

[15] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating Systems*, Oct. 2002.

[16] A. Kumar, L-S. Peh, P. Kundu, N. Jha, "Express Virtual Channels: Towards the Ideal Interconnection Fabric," in *Proceedings 34th Annual International Symposium on Computer Architecture* (ISCA'07), pp. 150–161, June 2007.

[17] D. Lenoski, J. Laudon, T. Joe, D Nakahira, L Stevens, A. Gupta, and J. Hennessy, "The DASH Prototype: Implementation and Performance," In *Proceedings19th International Symposium on Computer Architecture*, pp. 92–103, Gold Coast, Australia, May 1992.

[18] A. S. Leon, et al., "A power-efficient high-throughput 32-thread SPARC processor," *IEEE International Solid-State Circuits Conference*, Feb. 2006.

[19] "MPI Performance Measurements" at http://www.llnl.gov/computing/mpi/mpi_benchmarks.html.*

[20] C. McNairy and R. Bhatia, "Montecito: A dual-core, dual-threaded Itanium® processor," *IEEE Micro*, March–April, 2005.

[21] C.A. Nicopoulos, D. Park, J. Kim, N. Vijaykrishnan, M. S. Yousif, C.R. Das, "ViChaR: A Dynamic Virtual Channel Regulator for Network-on-Chip Routers," *International Symposium On Microarchitecture* (MICRO'06) pp. 333–346, Dec. 2006.

[22] M. K. Qureshi, A. Jaleel, Y. N. Patt, S. C. Steely Jr., J. Emer, "Adaptive Insertion Policies for High Performance Caching," *International Symposium on Computer Arachitecture*, June 2007.

[23] N. Sakran, et al., "The Implementation of the 65nm Dual-Core 64b Merom Processor," *IEEE International Solid-State Circuits Conference*, Feb. 2007.

[24] P. Stenstrom, "A survey of cache coherence schemes for multiprocessors," *IEEE Computer*, pp. 12–24, June 1990.

[25] M. B. Taylor, W. Lee, S. Amarasinghe, A. Agarwal, "Scalar Operand Networks: On-chip Interconnect for ILP in Partitioned Architectures," *International Symposium on High Performance Computer Architecture*, February 2003.

[26] A. W. Topol et al., "Three-dimensional integrated circuits," IBM Journal of Research and Development, vol. 50, no. 4/5, 2006, pp. 491–506.

[27] S. Vangal et al., "An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS," *IEEE International Solid-State Circuits Conference*, Feb. 2007.

[28] H-S. Wang, L-S. Peh, N. Jha, "Power-driven design of router microarchitectures in on-chip networks," *International Symposium On Microarchitecture* (MICRO'03), pp. 105–116, Nov. 2003.

[29] P. Wu, A. E. Eichenberger, A. Wang, P. Zhao, "An integrated simdization framework using virtual vectors," *International Conference on Supercomputing*, pp. 169–178, June 2005.

[30] M. Zhang and K. Asanovic, "Victim Migration: Dynamically Adapting Between Private and Shared CMP Caches," *MIT CSAIL Technical Report*, MIT-CSAIL-TR-2005-064, Cambridge, MA, October 2005.

[31] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," in *Proceedings 32nd International Symposium on Computer Architecture*, Madison, WI, June 2005.

AUTHORS' BIOGRAPHIES

Mani Azimi is a Senior Principal Engineer and Director of the Platform Architecture Research team in the Microprocessor Technology Laboratory in Intel's Corporate Technology Group. He received his Ph.D. degree from Purdue University. He joined Intel in 1990 and has worked on a wide range of platform architecture topics including system protocol, processor interface, MP cache controller architecture, and performance modeling/analysis. He is currently focusing on tera-scale computer architecture challenges. His e-mail is mani.azimi at intel.com.

Naveen Cherukuri is a Research Scientist within Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. At Intel, he has worked on the Itanium® processor design and performance analysis. His current research focuses on the cache organization for tera-scale architecture. He has an MSEE degree from the University of Arizona, Tucson. His e-mail is naveen.cherukuri at intel.com.

D. N. Jayasimha is a Principal Engineer in the Corporate Technology Group at Intel Corporation with research interests in multiprocessor architectures, interconnects, and performance analysis. Prior to joining Intel he was a faculty member in Computer Science at the Ohio State University. He received his Ph.D. degree from the University of Illinois at Urbana Champaign. His e-mail is jay.jayasimha at intel.com

Akhilesh Kumar is a Principal Engineer in Intel's Corporate Technology Group and leads the definition of protocols for on-chip and off-chip system interconnects. His research interests include cache organization, on-chip and off-chip interconnects, and interface protocols. He received his Ph.D. degree in Computer Science from Texas A&M University. His e-mail is akhilesh.kumar at intel.com.

Partha Kundu is a Senior Staff Research Scientist within Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. He was an architect of the Intel® Itanium® architecture and a Principal Architect on a DEC/Alpha microprocessor. His research interests include on-chip networks, memory system design, transactional memory, and performance simulation. He holds an M.S. degree from the State University of New York, Stony Brook. His e-mail is partha.kundu at intel.com.

Seungjoon Park is a Research Scientist within Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. At Intel, he has contributed to the definition and formal verification of off-die and on-die cache coherence and system interface protocols. Prior to Intel, he worked at NASA Ames Research Center with the High-Assurance Software Design Research team on Java PathFinder, a system to verify executable Java bytecode programs. He received his Ph.D. degree in Electrical Engineering with a Minor in Computer Science from Stanford University, where he investigated the cache coherence protocol of the Stanford FLASH multiprocessor and developed operational memory models of SPARC V9 architecture. His e-mail is seungjoon.park at intel.com.

Ioannis (Yannis) Schoinas is a Principal Engineer in Intel's Corporate Technology Group. He received his B.S. and M.S. degrees from the University of Crete-Heraclion and his Ph.D. degree from the University of Wisconsin-Madison. At Intel he has worked on a wide range of platform architecture topics including coherence protocol, memory RAS, system partitioning, configuration management, system security, and virtualization. He is currently focusing on tera-scale computer architecture challenges. His e-mail is ioannis.t.schoinas at intel.com.

Aniruddha S. Vaidya. is a Research Scientist at Intel's Microprocessor Technology Labs (MTL) in Santa Clara, California. His contributions at Intel include workload characterization, performance analysis, and architecture of server platforms. His current focus is on router and interconnection network architecture for Intel's tera-scale computing initiative. Ani has B.Tech and M.Sc. (Engg.) degrees from Banaras Hindu University and the Indian Institute of Science, and a Ph.D. degree in Computer Science and Engineering from the Pennsylvania State University. His e-mail is aniruddha.vaidya at intel.com.

For further information visit:

developer.intel.com/technology/itj/index.htm