



Intel[®] Technology Journal

Tera-scale Computing

Datacenter-on-Chip Architectures: Tera-scale Opportunities and Challenges in Intel's Manufacturing Environment

Datacenter-on-Chip Architectures: Tera-scale Opportunities and Challenges

Ravi Iyer, Corporate Technology Group, Intel Corporation
Ramesh Illikkal, Corporate Technology Group, Intel Corporation
Li Zhao, Corporate Technology Group, Intel Corporation
Srihari Makineni, Corporate Technology Group, Intel Corporation
Don Newell, Corporate Technology Group, Intel Corporation
Jaideep Moses, Corporate Technology Group, Intel Corporation
Padma Apparao, Corporate Technology Group, Intel Corporation

Index words: chip multiprocessors, datacenters, tera-scale, QoS, cache, memory, platforms

ABSTRACT

We have entered an era of chip multiprocessor (CMP) platforms, where performance is delivered with the integration of more and more cores on a die. Tera-scale CMP architectures, consisting of several tens of physical cores and hundreds of hardware threads, are highly suitable for throughput computing especially in the server market place. In this paper, we start by highlighting tera-scale potential in datacenter environments. We show how a multi-tier datacenter workload that required tens (to hundreds) of platforms in the past can potentially map on to one (or a few) single-socket tera-scale CMP platforms running Virtual Machines (VMs) and thereby creating Datacenter-on-Chip (DoC) architectures.

Having introduced tera-scale DoC architectures, we then describe key challenges involved in providing high degrees of performance, scalability, and adaptability. Performance and scalability challenges point to the need for efficient handling of cache/memory/I/O requirements when a large number of cores are actively running many workloads. Adaptability challenges highlight the need for dynamically allocating cache, memory, and I/O resources amongst the simultaneously running VMs in order to enable Quality of Service (QoS). To address scalability and adaptability challenges, we then propose and evaluate important tera-scale architectural features: (a) hierarchy of shared caches and large DRAM caches for better cache/memory scalability and performance, and (b) cache/memory QoS techniques to form Virtual Platform Architectures (VPAs). Based on a detailed

evaluation, we show that these architectural features are highly beneficial for DoC tera-scale architectures.

INTRODUCTION

We have entered the era of CMP platforms with Intel's dual-core and quad-core processors [5, 8] flourishing in the mobile, desktop, and server marketplace. Within a decade, we expect to integrate more and more cores on-die and create tera-scale architectures consisting of several tens of physical cores and hundreds of hardware threads. Such tera-scale architectures are highly suitable for high-performance throughput computing especially in the server marketplace.

A decade ago, datacenters employed tens to hundreds of dual-processor and quad-processor server platforms (each running a single application) on an Ethernet fabric. However, recent trends show that most datacenters have started employing virtualization [21, 23, 31, 32] to consolidate multiple applications onto the same platform in order to improve efficiency, manageability, and overall cost [6]. With tera-scale architectures [7] comes the potential to accelerate the consolidation trend and potentially even enable small datacenters to run on a single (or a few) platforms, thus the term "Datacenter-on-Chip" (or DoC) architectures. In this paper, we use an e-commerce benchmark, TPC-W [29], to illustrate this by showing how an earlier configuration (with 60+ server platforms) can now potentially run on a single tera-scale DoC platform with 32 cores and 128 threads (4 threads per core).

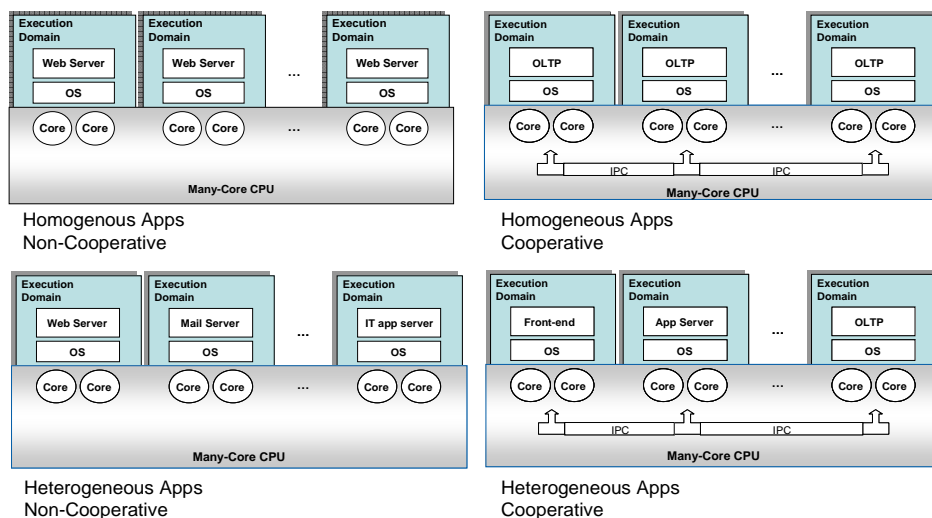


Figure 1: Datacenter-on-chip usage models: classification and examples

With tera-scale DoC architectures comes the challenge of designing a balanced platform with sufficient resources to sustain the large number of cores actively running VMs. In this paper, we evaluate the cache, memory, and I/O requirements as well as the behavior of the DoC architecture. We accomplish this by analyzing the TPC-W configuration as well as running detailed platform simulations that mimic multiple VMs running Online Transaction Processing (OLTP) workloads, Java* application server workloads, and even enterprise resource planning workloads simultaneously. To address the cache/memory scalability requirements, we show that (a) a hierarchy of shared caches is most suitable for DoC architectures since it maximizes performance and area efficiency when running virtualized server workloads and (b) integrating a large-capacity DRAM cache can significantly reduce the memory bandwidth requirements and thereby improve performance and scalability.

Another critical challenge in DoC architectures is that the performance of each VM can be highly non-deterministic since it depends heavily on the other VMs running simultaneously. Since an abundant number of cores is provided in tera-scale DoC architectures, the source of this non-determinism comes from interference in shared platform resources such as cache and memory. Through detailed simulations of simultaneously running VMs, we quantify the impact of this interference and the lack of QoS provided to each individual workload. Since datacenters typically provide service-level agreements, it is important to incorporate QoS hooks in the platform resources such as cache and memory. In this paper, we describe potential platform QoS mechanisms and evaluate the effectiveness of these mechanisms in improving the performance isolation provided to each VM.

DATACENTER-ON-CHIP USAGE MODELS AND TERA-SCALE ARCHITECTURES

In this section, we start by describing four classes of DoC usage models and then focus on one of them to highlight the potential of tera-scale architecture and describe the key challenges.

Datacenter-on-Chip Usage Models

Virtualization techniques make it possible to consolidate multiple server applications onto a single system. This usage model has been gaining momentum in enterprise datacenters because it improves resource sharing and usage, improves manageability, and reduces cost. We expect this trend to continue growing significantly in the coming years especially with the integration of more and more cores on the die. DoC essentially refers to the potential of multiple datacenter applications running simultaneously on a single-chip server platform. DoC usage scenarios can be classified into four broad categories based on (a) the types of applications being consolidated and (b) the level of communication and cooperation between the applications. Figure 1 illustrates the four DoC usage models that are explained further below.

- **Homogeneous/Non-Cooperating:** In this type of consolidation, multiple server applications of the same type are consolidated onto a single platform. However, these applications are independent in nature and no significant communication is required between the applications. A good example is the consolidation of a farm of Web servers that are serving Web pages and are load balanced. For the most part, these different

Web servers run on their own without having to communicate with each other.

- **Heterogeneous/Non-cooperating:** In this type of server consolidation, multiple different server applications are consolidated onto a single system. It is often the case in enterprise datacenters that servers are under utilized for a significant portion of the time when running a single application. The main motivation for this type of consolidation is to achieve maximum resource usage by consolidating multiple applications onto the same platform. In this type of consolidation, the applications being consolidated are still quite independent and do not need to communicate with each other. One example is that of consolidating an email server, file and print server, and user authentication server.
- **Homogeneous/Cooperating:** This type of consolidation occurs when a clustered application (e.g., database cluster) is consolidated onto a single system. Clusters use some sort of message passing either on a regular network fabric like Ethernet or on a more specialized fabric to communicate with each other. This communication turns into inter-VM communication once consolidated onto the same platform.
- **Heterogeneous/Cooperating:** This type of usage model occurs when multiple heterogeneous workloads that need to communicate with each other are consolidated. A good example of this type is where a multitiered application, like in TPC-W, is consolidated onto a single system. Here various tiers need to communicate with each other while servicing user requests. Hence inter-VM communication can be a significant factor, and handling this can be a challenge in virtualized environments, as we will see in the later part of this paper.

Mapping to Tera-scale Architectures

The DoC usage models described above can take advantage of more and more cores on-die since they have many applications (potentially multi-threaded) running simultaneously on a single platform. As a result, a tera-scale architecture with several tens of physical cores and hundreds of hardware threads integrated on the die is highly suitable for DoC usage. To illustrate the potential of tera-scale and highlight the challenges, we now focus on a case study of an e-commerce environment based on the TPC-W benchmark.

TPC-W [29] is a benchmark representative of an e-Commerce datacenter environment defined by the Transaction Processing Performance Council (TPC). The performance metric reported by TPC-W is the number of Web interactions processed per second (WIPS). Multiple Web interactions are used to simulate the activity of a retail store, and each interaction is subject to a response time constraint. The TPC-W benchmark is now obsolete;

however, the e-Commerce workload that it represents is very relevant and important. A typical TPC-W setup contains several different application components (as shown in Figure 2):

- **Web Servers** process incoming HTTP requests and prepare responses to be sent to the clients.
- **Web Cache Servers** cache static and dynamic content for fast access to data.
- **Image Servers** serve static images that are part of the response Web pages.
- **Application Servers** provide the e-Commerce functionality and are responsible for processing customer orders and payments for goods, among other things.
- **Database Servers** hold inventory of product, description, availability, pricing and other information.
- **Load Balancer and other Infrastructure Servers** distribute processing load among different Web and image servers equally by directing incoming HTTP requests to the server with the least load.

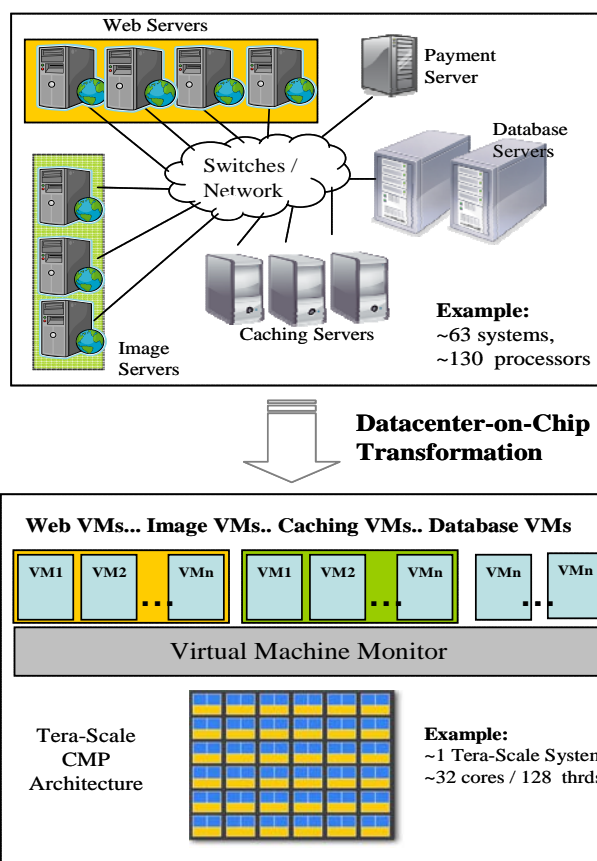


Figure 2: Mapping datacenter workloads to tera-scale

Table 1: Compute/cache/memory capacity data from TPC-W setup (example based on TPC-W publication [30])

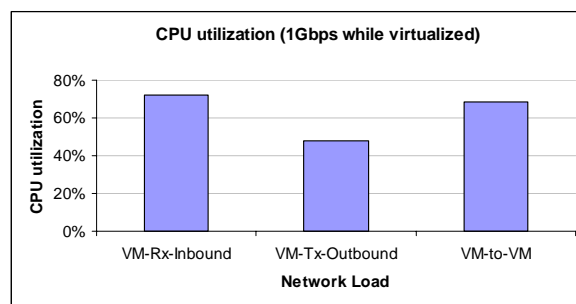
Server Type	Os/es	Num servers	Total Procs	Frequency (GHz)	Memory (MB)	Total Cache (l	CPU Util
Web server	Win 2K Server	26	52	1.26	19968	26624	80%
Web server/Other software	Win 2K Server	1	2	1.26	768	1024	80%
Image server	Win 2K Server	20	40	1.26	10240	20480	
Image server/Load Balancer	Win 2K Server	1	2	1.26	512	1024	
Database server	MS .NET EE	1	8	1.6	8192	8192	45%
Web cache	Win 2K Server	9	18	1.26	4608	9216	70%
Web cache	Win 2K Server	3	6	1.26	2304	3072	70%
Web cache	Volera	2	2	1.26	4096	1024	55%
Total		63	130		50688	70656	

Figure 2 shows how these different components are interconnected in a typical TPC-W setup of the past. TPC-W is a perfect example of understanding the requirements and behavior of consolidating multiple tiers (heterogeneous/cooperating) of a datacenter on a tera-scale CMP platform (bottom of Figure 2). Table 2 summarizes the number of systems, compute cores, cache and memory in an example configuration roughly based on a high-performing 2002 TPC-W publication [30]. As shown in the figure as well as the table, there are 63 systems employed in the example TPC-W configuration. Except for the database server, which employed four processors, all other systems consisted of two processors (without multi-threading). As a result, the total number of processors in the configuration was about 130. In a tera-scale CMP platform, we expect that a single processor socket could contain 32 cores each with 4 threads (SMT). As a result, the entire TPC-W example configuration can be potentially consolidated onto such a 32-core, 128-thread single-socket platform.

However, it is also critical that we take into account the amount of platform resources that are needed to support the execution of simultaneously running VMs of this nature. For example, Table 2 shows that the total cache capacity available in the TPC-W configuration adds up to 70MB in size. Given the area constraints and the fact that 32 cores will be integrated onto the die, our previous work [36] has shown that the amount of cache space available is likely to be less than 32MB. As a result, architectural techniques that enhance cache/memory scalability and performance need to be incorporated into the platform. We discuss these further in the next section.

Another key challenge in running heterogeneous VMs of this nature on the same platform is that they will contend for platform resources and interfere with each other. Given that these VMs are likely to get very different utility benefits from platform resources, and that the VMs are likely to be different in importance to the overall performance of the datacenter, it is important that we incorporate adaptability techniques in the platform so that resource usage can be dynamically controlled to provide performance isolation or QoS for DoC platforms. In the following section, we describe adaptability challenges and solutions to address these in tera-scale architectures.

Last but not least, it is also important to consider the overheads of virtualization on the DoC performance. In addition to the basic overhead of handling system calls, context switches, and interrupts for VMs, one primary concern in virtualized platforms is the overhead of I/O virtualization. For example, Figure 3 shows the overheads of virtualization for (a) transmitting network data to external platforms, (b) receiving network data from external platforms, and (c) inter-VM communication between VMs. The data shown in Figure 3 are based on measurements done on a recent Intel[®] Xeon[®] dual-core processor (3GHz) dual-socket platform [8] running the Xen hypervisor [3, 33]. The measurements show that (a) receiving network data at 1Gbps and processing requires 75% of CPU utilization under virtualization, (b) transmitting 1Gbps externally requires about 50% of CPU processing, and (c) communicating 1Gbps between VMs on the same platform requires about 70% of CPU utilization. Further, it should be noted that these compute cores are large out-of-order cores without multiple threads sharing the pipeline. As we design tera-scale processors, the use of smaller in-order cores with multiple threads sharing the pipeline may increase the associated processing overhead. However, since most of the cores in the example TPC-W configuration were underutilized (last column in Table 1), there is likely some headroom available to accommodate this extra I/O processing overhead. Extensions to techniques (such as Intel's I/O Acceleration Technology [14, 22]) are needed to minimize this overhead for a virtualized DoC environment. However, this is not covered in this paper.

**Figure 3: CPU overheads for network I/O virtualization**

SCALABILITY CHALLENGES AND SOLUTIONS

As described in the previous section, the tera-scale architecture offers a high compute density (large number of cores and threads) that is attractive for DoC usage models. However, in order to provide high performance and scalability, it is important to carefully design a balanced platform with sufficient resources (cache, memory, I/O, etc.). In this section, we present the DoC scalability considerations and discuss potential solutions that address the key challenges.

The first challenge is that of providing sufficient cache space in order to reduce memory stalls and minimize memory bandwidth bottlenecks. Previous work [36] has shown that die area and cost will significantly restrict the amount of cache space that can be provided in tera-scale processors. In DoC usage models, the fact that several multi-threaded server applications will run simultaneously poses two potential considerations for cache hierarchy design: (a) since the threads within each server application tend to share code as well as data, cache space efficiency can be improved if these threads are allowed to share cache space, (b) since the cache space usage of each of the server applications can be quite different at different times in the execution, better utilization can be achieved if the cache space is shared. To take advantage of both of these sharing properties, we propose and evaluate a hierarchy of shared caches for tera-scale DoC platforms.

Figure 4 illustrates a three-level hierarchy of shared caches in a tera-scale platform. The hierarchy of shared caches starts an L1 (16K to 64K) that is private to the core but shared between the multiple threads within the core. The L2 (256K to 1M, mid-level) cache is also shared by multiple cores within a “node.” The node forms the basic building block for the architecture. The L3 (8 to 32M, last-level) cache is logically shared by all of the nodes in the socket. However, since the L3 cache is quite large, it is physically distributed around the die in smaller “slices.” A scalable interconnect connects all the L3 cache slices and the nodes. The benefits of sharing at each level is best explained with an example. Figure 5 compares the cache performance of private and shared L2 caches for an OLTP workload (based on the TPC-C [28]). As shown in the figure, a shared cache organization (e.g., 512K shared by four cores) is equivalent in cache performance to a private cache organization (four cores each with a 256K private L2 cache). This essentially shows a potential of 2X space efficiency with a shared cache organization. Similar benefits were found for other server workloads as well as for other levels of the hierarchy.

Having defined a cache hierarchy, the next major challenge is that of providing sufficient memory

bandwidth to sustain the misses from the last-level cache. Figure 6 shows the cache scaling behavior of a consolidated server workload running on a last-level cache. These data were obtained from trace-driven simulations of four (8-threaded) workloads based on TPC-C [28], SPECjbb2005 [26], SPECjappserver2004 [25], and SAP SD/2T [24] running simultaneously on 32 single-threaded cores. The data show that consolidation workloads have good cache scaling behavior from 4MB all the way to 128MB of cache shared between the 32 cores.

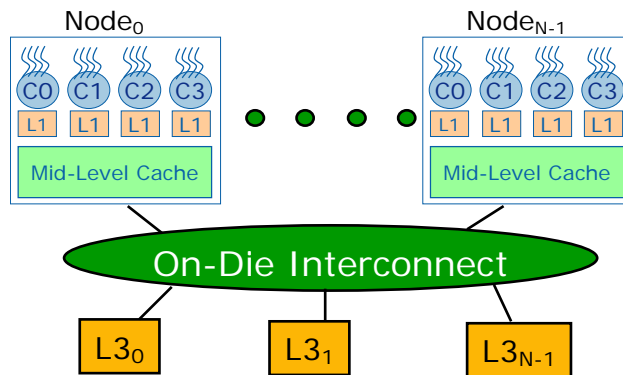


Figure 4: Tera-scale DoC hierarchy of shared caches

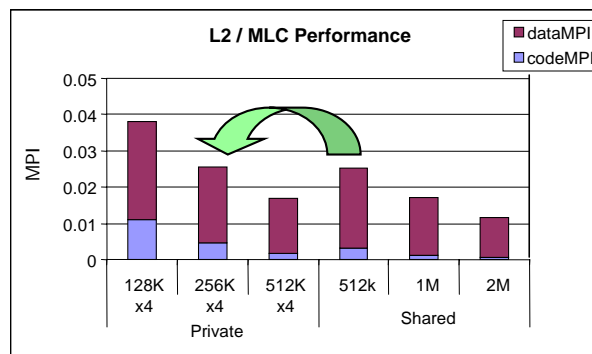


Figure 5: Tera-scale shared L2 cache benefits

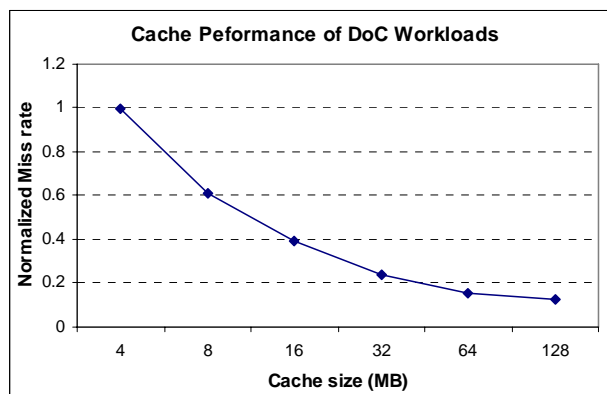


Figure 6: Tera-scale DoC L3 cache scaling behavior

To understand the memory bandwidth requirements of tera-scale DoC platforms, let us now consider the simulation configuration with 8MB L3 cache and 32 cores. In this configuration, we estimated that the bandwidth requirements can be as high as 20GB/s. Given that tera-scale processors may contain as many as 128 threads, the overall bandwidth requirements can be 100GB/s or higher. This in turn requires that a proportional number of memory channels be supported on the socket. Alternate solutions to solving the memory bandwidth bottleneck for tera-scale platforms could be the use of large capacity L4 caches. As shown in Figure 7, large capacity L4 caches can be implemented either as an additional package on the package (in a multi-chip package) or stacked (using 3D stacking technologies [1]). To understand the potential of large capacity L4 DRAM caches that can provide as much as twice the bandwidth at as little as one-third of the memory latency, we conducted simulations of a 32-core, 8MB L3 cache configuration with and without a 32MB or 64MB L4 cache. We found that significant performance benefits (from 10 to 40%) can be achieved depending on the organization of the DRAM cache, the exact bandwidth capability, and the latency benefits as compared to main memory latency. However, the key benefit is that of providing sufficient headroom in external memory so that the number of channels that is implemented can be reduced without affecting the performance.



Figure 7: Tera-scale DoC L3 cache scaling behavior

Having addressed the cache/memory scalability challenges for tera-scale DoC architectures (using a hierarchy of shared caches and large L4 caches), we next turn our attention to adaptability concerns and solutions.

ADAPTABILITY CHALLENGES AND SOLUTIONS

Flexible and dynamic management of platform resources is important in DoC tera-scale architectures since multiple VMs will be running simultaneously. Traditionally, the execution environment (a virtual machine monitor (VMM) or hypervisor in DoC) attempts to control the visible resources (number of cores and memory capacity for instance). However, this alone will not suffice for CMP platforms where more cores might be available to run the virtualized applications simultaneously, but they end up contending for other (invisible) shared resources

that have first-order performance impact [2, 4, 10, 16, 18, 34]. Key among these invisible resources are cache space and memory bandwidth. In addition to cache and memory, other resources that are shared include interconnects, micro-architectural resources in the core (shared between hardware threads), and power.

While sharing resources is generally the most efficient approach to maximize resource utilization, having no control over management of these resources can lead to loss of determinism, lack of performance isolation, and an overall lack of the notion of QoS provided to an individual application running on the platform. This has a very direct impact on the datacenter consolidation environments where more and more heterogeneous workloads are consolidated into a single platform contending for the shared hardware resources. Another important aspect to consider when managing shared resources is the relative importance of each of the consolidated applications. Not all applications consolidated may be of equal importance. The difference in priority could be based purely on the service level agreement provided to the customer or could be based on the relative throughput requirements of each of the consolidated applications. It could also be decided by the VMM layer based on the workload behavior (cache friendly, IOVM, etc.).

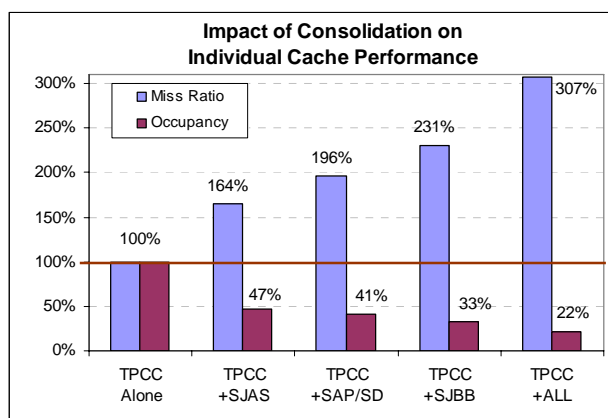


Figure 8: OLTP cache performance under consolidation

We start by studying the extent to which contention for shared cache space affects an individual OLTP application when running with one or more other consolidated applications. We performed a trace-driven simulation [9] of a 32-core processor with 8MB of last-level cache running (a) an 8-threaded OLTP application (based on TPC-C) running alone, (b) OLTP consolidated with an 8-threaded J2EE application server workload (based on SPECjappserver2004), (c) OLTP running consolidated with an 8-threaded ERP application (based on SAP SD/2T), (d) OLTP consolidated with a

8-threaded Java workload (based on SPECjbb2005), and (e) OLTP consolidated with all of the three above applications. Figure 8 shows the impact of consolidation on OLTP cache occupancy as well as OLTP miss rate. As the occupancy is reduced from 100% when running alone to as low as 20% when running with all other workloads, the miss rate goes up significantly (by as much as 3X). It should be noted that even though the compute resources available to the OLTP application remain the same when running alone and running in consolidated mode, the performance will be significantly affected due to the increase in miss rate.

QoS and Virtual Platform Architectures

Managing the allocation of shared resources in the platform is key to addressing the contention effects shown above and to providing performance differentiation, performance isolation, and the overall notion of QoS. Today, Intel and other processor manufacturers, support hardware virtualization features. While these features support functionally isolated VMs, they do not offer the ability to provide performance isolation. Our goal is to define mechanisms that allow VMs to transform into Virtual Platform Architectures (VPAs). A VPA is defined as a collection of virtual resources (i.e., some fraction of each of the physical shared resources) that a VM is provided. In this section we introduce our Platform QoS research that enables QoS-aware platforms and VPAs.

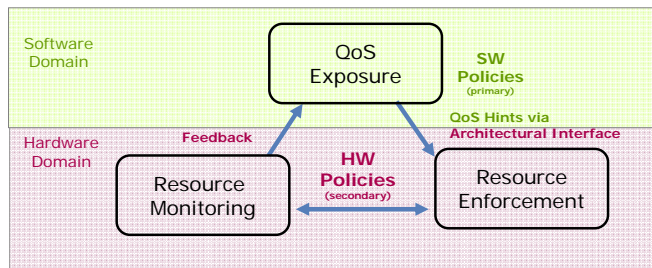


Figure 9: Platform QoS approach

Platform QoS

For DoC tera-scale architectures, there are three key questions that the Platform QoS research attempts to answer: (a) how much of each shared resource is an application or VM using (b) how can the resource allocation be modified to improve individual QoS or overall performance, and (c) what are the most appropriate interfaces and mechanisms needed between hardware and software to achieve QoS and VPAs?

The Platform QoS approach attempts to address these questions by enabling three major components: (a) monitoring, (b) enforcement, and (c) exposure. Figure 9 shows the components and their relationship in terms of information flow. The monitoring and enforcement components are implemented in hardware, whereas the

QoS policies and exposure can either be guided by software or by hardware. The monitoring component keeps track of shared resource usage on a per-application or per-VM basis. The resource monitoring ability allows the platform to pass back information to the execution environment (VMM or hypervisor in DoC) to determine the VPA that each VM ends up with in a platform. In addition, providing this information back to the software domain allows the VMM to optimize scheduling decisions or pass down hints for resource enforcement. The monitoring ability may also be useful to the system administrator to determine (a) whether a VM should be migrated to a different platform (if it is getting too few resources consistently), (b) what QoS hints should be passed down to the platform to modify resource allocation, or (c) what the end-customer should be charged based on resource usage. Alternatively, the administrator may be able to set up a QoS policy that performs one or more of the above dynamically, based on monitoring data.

The resource enforcement component implements shared resource partitioning based on software guidance. This requires an architectural interface to be exposed to the execution environment that allows the specification of resource allocation requirements on a per-VM basis. While we expect QoS policies to be determined primarily by software, it is also important to allow a path for future platform optimizations that dynamically manage resources entirely in hardware. The resource enforcement component enables the VMM to create VPAs with a user-specified amount of resources. To achieve a scalable low-cost QoS solution, we propose resource partitioning and QoS exposure on a class of service basis instead of a per-VM basis. This is sufficient since it is unlikely that all of the VMs running on the platform need performance isolation simultaneously. Instead, one or more VMs can be mapped to a single class of service as specified by the VMM, and a smaller number of classes of service can be supported by the platform. In this paper, we use the terms “priority class,” “priority level” and “class of service” interchangeably.

To help clearly describe the Platform QoS approach and mechanisms required, we now present a case study using the shared cache as the platform resource.

QoS Case Study Using Shared Caches

Since contention to shared cache (e.g., last-level) is a key issue, we now describe the implementation considerations for shared cache monitoring, enforcement, and exposure (highlighted in Figure 10).

In the case of cache monitoring, the goal is to keep track of cache space consumed on a per-application or per-VM basis. In order to do so, the VMM needs to pass down a

unique identity (ID) to the platform for each running VM. This can be easily done by writing the ID to a new register, a Platform QoS Register (PQR), that is part of the processor architectural state. Since the ID is finite, it should be noted that the ID might have to be recycled amongst VMs (when the number of VMs is larger than the number of IDs). Once the ID is passed down, each load/store generated by the CPU is tagged with the ID so that it is passed down to the last-level cache. In the last-level cache, each cache line is tagged with the ID, and a global cache occupancy counter is also maintained per ID. When a line is evicted from the core, the appropriate cache occupancy counter is decremented. When a new line is allocated into the cache, the appropriate cache occupancy counter is incremented. The implementation can be optimized for area by employing set sampling techniques [37] if so desired.

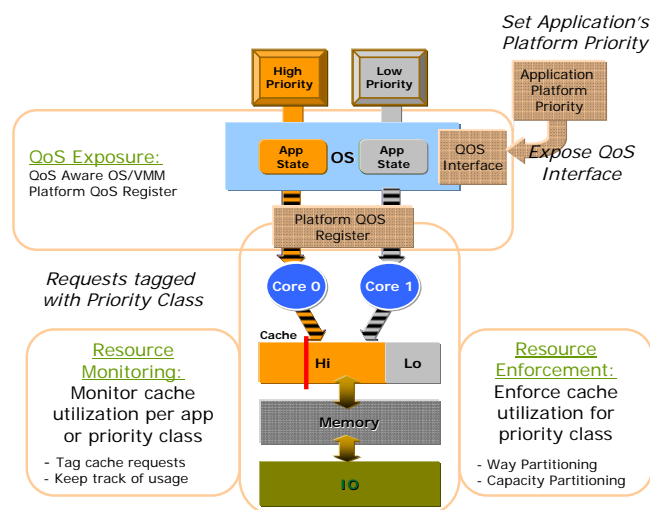


Figure 10: Cache QoS architecture and techniques

For cache enforcement, we are investigating the use of several forms of capacity partitioning. One potential approach attempts to limit the number of cache lines in the entire cache used by a certain class of service. Since the class of service is also associated with each cache line, this enforcement is accomplished by modifying the cache replacement policy to pick the next victim from a class that is currently exceeding its assigned cache quota.

For cache QoS exposure, we introduce the PQR. The PQR allows software to specify (a) the VM ID, (b) the class of service (also referred to as priority level or priority class) that this VM should be mapped to, and (c) an optional resource allocation target for that class of service. As described above, the VM ID is used by the platform to monitor cache occupancy per application. The class of service is used to guide the QoS-aware replacement decision.

To study the potential benefits of cache QoS enforcement we extended our trace-based cache simulations to implement cache enforcement. We conducted performance simulations of various consolidation scenarios where we limited the amount of cache space available to the low priority VM, but allowed high-priority VMs to allocate anywhere in the cache. In our example, we chose the OLTP application as the high-priority VM (with access to 100% of shared cache) and the three other consolidated applications as the low-priority VMs (limited to X% of the cache space). Figure 11 shows the OLTP miss rate as a function of X% (on the x-axis). As expected, reducing X from 100% to 12.5% improves the cache performance of the high-priority OLTP application significantly. It may be noted that this will negatively impact the performance of the low priority VMs, but that is expected as an outcome of performance differentiation and QoS.

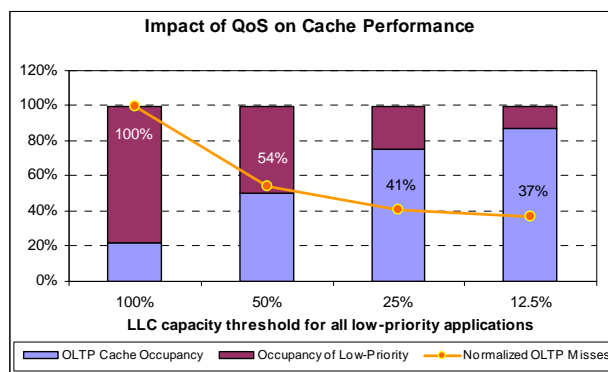


Figure 11: Case study of cache QoS benefits

In the previous sections, we focused on the cache-sharing impact and cache QoS implementations. However, the implications are similar for other shared platform resources. For example, memory bandwidth is another resource that has a direct impact on application performance. Memory QoS [11] can be achieved by implementing priority queues in the controller or enabling rate control of the request stream. Once all shared resources are enabled with QoS support, we could provide differentiated service to the individual VMs running on top of these resources. This combined with hardware-supported virtualization provides a complete VPA where functional and performance isolation is provided to VMs in a DoC architecture.

CONCLUSION

In this paper, we introduced DoC architectures and showed the potential of tera-scale platforms for DoC environments. The opportunity for more and more applications currently running on dedicated platforms to run on a tera-scale platform is tremendous, but it also

introduces some significant scalability and adaptability challenges that we need to address.

In this paper, we presented the scalability challenges for DoC in tera-scale platforms and described two important potential architectural features: (a) hierarchy of shared caches and (b) large-capacity L4 caches. We showed that enabling sharing at each level of the hierarchy can significantly maximize the space efficiency (e.g., sharing the mid-level L2 cache between multiple cores within a node provided a 2X better area efficiency as compared to private L2 caches). In addition, we also showed that large-capacity L4 caches (enabled either by 3D-stacking or a multi-chip package) can mitigate the memory bandwidth challenges for tera-scale platforms.

Last, but not least, we presented the adaptability challenges for DoC tera-scale environments. DoC environments suffer from the lack of performance isolation and performance differentiation since multiple simultaneously running VMs are contending for critical shared platform resources. We described our Platform QoS research that is investigating QoS techniques for resource monitoring and enforcement to enable performance isolation and differentiation. We showed how these QoS techniques allow us to transform VMs into VPAs. The end goal is to provide better QoS in tera-scale platforms for DoC environments.

Future work in this area is as follows. Research work along the lines of scalable cache/memory hierarchies [12, 27, 35, 19] and adaptable QoS techniques [4, 10, 11, 13, 15, 16, 17, 18, 20, 37] is a great start, but more and more emphasis on DoC usage models will be needed in the future.

REFERENCES

- [1] B. Black et al., "Die Stacking (3D) Microarchitecture," *39th Annual IEEE/ACM International Symposium on Microarchitecture, MICRO*, 2006.
- [2] D. Chandra, F. Guo, S. Kim, and Y. Solihin, "Predicting inter-thread cache contention on a chip multiprocessor architecture," *11th International Symposium on High Performance Computer Architecture (HPCA)*, Feb. 2005.
- [3] "Xen Virtualization Technology," *Xen Source*, at http://www.xensource.com/xen/xen/*
- [4] L. Hsu, S. Reinhardt, R. Iyer and S. Makineni, "Communist, Utilitarian, and Capitalist Policies on CMPs: Caches as a Shared Resource," *Int'l Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006.
- [5] Intel Corporation, "Intel Dual-Core Processors," at <http://www.intel.com/technology/computing/dual-core/>.
- [6] Intel Corporation, "Multiply Virtualization Maximize Server Harmony," at http://www.intel.com/business/technologies/virtualization.htm?iid=servproc+marquee_virtualization
- [7] Intel Corporation, "Tera-scale Computing," at <http://techresearch.intel.com/articles/Tera-Scale/1421.htm>
- [8] Intel Corporation, "Intel[®] Xeon[®] Processor 5000 Sequence," at http://www.intel.com/products/processor/xeon5000/index.htm?iid=servproc+body_xeon5000
- [9] R. Iyer, "On Modeling and Analyzing Cache Performance using CASPER," *Int'l Symposium on Modeling, Analysis and Simulation of Computer & Telecom Systems*, Oct. 2003.
- [10] R. Iyer, "CQoS: A Framework for Enabling QoS in Shared Caches of CMP Platforms," *18th Annual International Conference on Supercomputing (ICS'04)*, July 2004.
- [11] R. Iyer, L. Zhao, et al., "QoS Policies and Architecture for Cache/Memory in CMP Platforms," *SIGMETRICS*, 2007.
- [12] C. Kim, D. Burger, S. W. Keckler, "Nonuniform Cache Architectures for Wire-Delay Dominated On-Chip Caches," *IEEE Micro* 23(6), pp. 99–107, 2003.
- [13] S. Kim, D. Chandra, and Y. Solihin, "Fair Cache Sharing and Partitioning in a Chip Multiprocessor Architecture," *13th Int'l Conf. on Parallel Arch. & Compilation Techniques (PACT)*, Sept. 2004.
- [14] K. Lauritzen, T. Sawicki, et al., "Intel[®] I/O Acceleration Technology Improves Network Performance, Reliability and Efficiently," *Technology@Intel Magazine*, at <http://www.intel.com/technology/magazine/communications/intel-ioat-0305.htm>
- [15] C. Liu, A. Sivasubramaniam, and M. Kandemir, "Organizing the Last Line of Defense before Hitting the Memory Wall for CMPs," *10th IEEE Symposium on High-Performance Computer Architecture*, Feb. 2004.
- [16] K. Nesbit, et al., "Fair Queuing Memory Systems," in *Proceedings of Annual International Symposium on Microarchitecture (MICRO)*, June 2006.

- [17] K. Nesbit, et al., "Virtual Private Caches," *International Symposium on Computer Architecture (ISCA)*, June 2007.
- [18] M. K. Qureshi and Y. N. Patt, "Utility-Based Cache Partitioning: A Low-Overhead, High-Performance, Runtime Mechanism to Partition Shared Caches," *Int'l Symposium on Microarchitecture (MICRO)*, June 2006.
- [19] M. Qureshi, A. Jaleel, et al., "Adaptive Insertion Policies for High Performance Caching," *International Symposium on Computer Architecture (ISCA)*, June 2007.
- [20] N. Rafique, W.T. Lim and M. Thottethodi, "Architectural Support for Operating System-Driven CMP Cache Management," *Int'l Conference on Parallel Architectures and Compilation Technology (PACT 2006)*, Sept. 2006.
- [21] P. Ranganathan and N. Jouppi, "Enterprise IT Trends and Implications on Architecture Research," *11th Int'l Symp. on High Performance Computer Architecture (HPCA)*, 2005.
- [22] G. Regnier, S. Makineni, R. Illikkal, R. Iyer, et al., "TCP Onloading for Datacenter Servers," *IEEE Computer*, 2004.
- [23] M. Rosenblum and T. Garfinkel, "Virtual Machine Monitors: Current Technology and Future Trends," *IEEE Transactions on Computers*, 2005.
- [24] Sap America Inc., "SAP Standard Benchmarks," at http://www.sap.com/solutions/benchmark/index.epx*
- [25] SPECjAppServer2004, at http://www.spec.org/jAppServer/*
- [26] SPECjbb2005, at http://www.spec.org/jbb2005/*
- [27] E. Speight, H. Shafi, L. Zhang, R. Rajamony, "Adaptive Mechanisms and Policies for Managing Cache Hierarchies in Chip Multiprocessors," *32nd International Symposium on Computer Architecture (ISCA)*, June 2005.
- [28] TPC-C Benchmark, at www.tpc.org/tpcc/
- [29] TPC-W Benchmark, at www.tpc.org/tpcw/*
- [30] TPC-W Publication, at http://www.tpc.org/results/FDR/tpcw/ibm.x440.w.dr.02091201.pdf*
- [31] VMware Corporation, Server Consolidation and Containment with VMware Virtual Infrastructure, at http://www.vmware.com/pdf/server_consolidation.pdf*
- [32] R. Uhlig, et al., "Intel Virtualization Technology," *IEEE Transactions on Computers*, 2005.
- [33] T. Deshane, D. Dimatos, et al., "Performance Isolation of a Misbehaving Virtual Machine with Xen, VMware and Solaris," at <http://people.clarkson.edu/~jnm/publications/isolationOfMisbehavingVMs.pdf>
- [34] T. Y. Yeh and G. Reinman, "Fast and Fair: Data-stream Quality of Service," *Int'l Conf. of Compilers, Architecture and System For Embedded Systems (CASES)*, July 2004.
- [35] M. Zhang and K. Asanovic, "Victim Replication: Maximizing Capacity while Hiding Wire Delay in Tiled Chip Multiprocessors," *32nd International Symposium on Computer Architecture (ISCA-32)*, Madison, 2005.
- [36] L. Zhao, R. Iyer, et al., "Performance, Area and Bandwidth Implications on Large-Scale CMP Cache Design," *Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, Feb. 2007.
- [37] L. Zhao, R. Iyer, et al., "CacheScouts: Fine-Grain Monitoring of Shared Caches in CMP Platforms," to appear in *16th International Conference on Parallel Architectures and Compilation Techniques (PACT)*, Sept. 2007.

AUTHORS' BIOGRAPHIES

Ravi Iyer is a Principal Engineer with the Systems Technology Lab in Intel's Corporate Technology Group. His current research focus is on large-scale CMP architectures and technologies. Before joining STL, he held positions in the Communications Technology Lab (working on IO acceleration research) and in the Enterprise Products Group (working on server architecture and performance). He received his Ph.D. degree in Computer Science from Texas A&M University. He has filed 20+ patent applications and published 70+ papers in the areas of computer architecture, server design, network protocols/acceleration, workload characterization, and performance evaluation. He has held program committee member positions in various conferences and workshops (HPCA 2006, PACT 2007, IISWC 2007, etc.). He is also an Associate Editor for *IEEE Transactions on Parallel and Distributed Systems* (IEEE TPDS) and is currently guest co-editor for a special issue on CMP architectures. His e-mail is ravishankar.iyer@intel.com.

Ramesh Illikkal is a Senior Researcher in the Systems Technology Lab at Intel. His research interests are in CMP, server architectures, virtualization, and memory

hierarchies. He received his Masters degree in Electronics from Cochin University of Science and Technology. His e-mail is ramesh.g.illikkal at intel.com.

Li Zhao received her Ph.D. degree in Computer Science from the University of California, Riverside. She is currently a Senior Engineer in the Systems Technology Laboratory at Intel. Her research interests include computer architecture, network computing, and performance evaluation. She is a member of the IEEE. Her e-mail is li.zhao at intel.com.

Srihari Makineni is a Senior Researcher in the Systems Technology Lab at Intel. He has been working at Intel for more than 11 years. His research interests include cache/memory subsystems, interconnects, networking, and large-scale CMP architectures. Makineni received an M.S. degree in Electrical and Computer Engineering from Lamar University, Texas. His e-mail is srihari.makineni at intel.com.

Don Newell is a Senior Principal Engineer in the Systems Technology Lab at Intel. His research interests include server architecture, networking, and I/O acceleration. Newell received a B.S. degree in Computer Science from the University of Oregon. His e-mail is donald.newell at intel.com.

Jaideep Moses is a Senior Engineer in the Systems Technology Lab at Intel. Prior to this, Jaideep worked in the Communication Technology Lab on I/O acceleration research. He also worked in the former Enterprise Products Group focusing on modeling, simulation, and analysis of platform architecture and design including simulation-based verification of a coherence protocol. His current research focus is on large-scale CMP platform architecture analysis and design. Jaideep received his M.S. degree in Computer Science from the University of Texas at El Paso. His e-mail is jaideep.moses at intel.com.

Padma Apparao is a Senior Researcher in the Systems Technology Lab at Intel. Padma received her Ph.D. degree from the University of Florida and has been working on performance analysis of server workloads. Her current research interest is in the areas of virtualization and large-scale CMP architecture analysis. Her e-mail is padmashree.k.apparao at intel.com.

Note: The use of SPEC or TPC benchmark configurations and traces in this paper is purely for analysis and illustration. They are not intended to provide any indication of performance of a specific platform.

BunnyPeople, Celeron, Celeron Inside, Centrino, Centrino logo, Core Inside, FlashFile, i960, InstantIP, Intel, Intel logo, Intel386, Intel486, Intel740, IntelDX2, IntelDX4, IntelSX2, Intel Core, Intel Inside, Intel Inside logo, Intel. Leap ahead., Intel. Leap ahead. logo, Intel NetBurst, Intel NetMerge, Intel NetStructure, Intel SingleDriver, Intel SpeedStep, Intel StrataFlash, Intel Viiv, Intel vPro, Intel XScale, IPLink, Itanium, Itanium Inside, MCS, MMX, Oplus, OverDrive, PDCharm, Pentium, Pentium Inside, skool, Sound Mark, The Journey Inside, VTune, Xeon, and Xeon Inside are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Intel's trademarks may be used publicly with permission only from Intel. Fair use of Intel's trademarks in advertising and promotion of Intel products requires proper acknowledgement.

*Other names and brands may be claimed as the property of others.

Microsoft, Windows, and the Windows logo are trademarks, or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Bluetooth is a trademark owned by its proprietor and used by Intel Corporation under license.

Intel Corporation uses the Palm OS[®] Ready mark under license from Palm, Inc.

Copyright © 2007 Intel Corporation. All rights reserved.

This publication was downloaded from <http://www.intel.com>.

Additional legal notices at: <http://www.intel.com/sites/corporate/tradmarx.htm>.

THIS PAGE INTENTIONALLY LEFT BLANK

For further information visit:

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