New *Compaq AlphaServer* GS Series

Architecture White Paper

Contents

COMPAQ NonStop™

Compaq AlphaServer GS Series Architecture White Paper

An Architectural Overview of the *Compaq AlphaServer* Series

The *AlphaServer* GS Series computer family is the next generation enterprise class server family from Compaq Computer Corporation. This family of server systems provides a variety of server solutions ranging from 8 processor capable systems to 32 processor capable systems. These systems introduce an innovative new switch-based interconnect topology and distributed shared memory (DSM) architecture. This new architecture is combined with world class availability and a host of customer focussed features to make the *AlphaServer* GS Series the ideal solution for a variety of enterprise class problems, ranging from the largest commercial database applications to the most demanding high performance technical computing applications.

This paper presents an architectural overview of the *AlphaServer* GS Series system design. It consists of three sections that address the "why", the "what" and the "how" of the system architecture. The first section describes why the system is designed as it is. Particularly, this first section describes the specific customer problems that were addressed by the basic design and architecture of the *AlphaServer* GS Series. The second section outlines briefly what the system design and architecture look like. The third section, focuses in more detail on the architecture of the system compute kernel and analyzes how the *AlphaServer* GS Series architecture delivers the outstanding performance for enterprise applications.

Customer Focussed Features

There has historically been a common misconception that enterprise class servers are developed with only one goal in mind—maximum performance. That, in fact, is not the reality. Today's sophisticated enterprise customers need much more from their computer systems in terms of reliability, availability, manageability and reduced cost of ownership. To that end, the *AlphaServer* GS Series family's design is built on a foundation of six key system parameters:

- Capacity
- Performance-Enabling Infrastructure
- Economic Scalability
- Flexible and Forgiving Software Model
- Robust Data Center Features
- Performance

Each of these parameters will be explored briefly in the following sections.

Capacity

The *AlphaServer* GS Series features industry leading system capacity, particularly in terms of main memory storage capacity and IO connectivity. The combination of industry leading storage capacity and connectivity, when combined with the power of the *Alpha* processor, provides substantial advantage to the enterprise class customer. Consider:

- Thirty-two processors may not represent the industry's highest processor count; but thirty-two 731MHz *Alpha* 21264 processors certainly represents industry leading compute capacity.
- The memory capacity of the *AlphaServer* GS Series further extends Compaq's lead into the Very Large Memory (VLM) computing domain. The VLM paradigm was established with the *AlphaServer* 8400 Series' 16GB of memory capacity. With the *AlphaServer* 8400 series, performance gains of between 10x and 100x were demonstrated on large commercial workloads. These gains were realized by the machine's ability to provide large, in-memory caches for commercial databases. The *AlphaServer* GS Series not only continues the VLM theme, but it pushes it forward an order of magnitude with 256GB of memory capacity. This capacity provides the *AlphaServer* GS Series with the ability to handle the largest, most demanding, enterprise applications.
- The *AlphaServer* GS Series' 224 PCI adapter capability sets a new standard for enterprise class connectivity. This magnitude of connectivity provides the resources necessary to support the massive data base storage farms and network requirements common in the enterprise today. It allows for a wide, bottleneck free system configuration and a flexible data center layout.

Performance-Enabling Infrastructure

The *AlphaServer* GS Series resource capacities, by themselves, represent little more than potential performance to the customer. To bring all of these resources to bear on the customer's application, the system provides significant "performance-enabling infrastructure". This refers to the interconnect resources and supporting logic that tie the system's resources together. Some of the key measures of the *AlphaServer* GS Series infrastructure capabilities are as follows:

- An advanced memory system design that supports up to 256-way interleave and up to 51.2GB/s of aggregate memory bandwidth
- A system topology that, as it grows, maintains a constant, 1.6GB/s of per-cpu bandwidth
- A distributed IO subsystem that features an aggregate of 12.8GB/s of IO system bandwidth

Economic Scalability

While the massive capacities and support outlined herein provide wonderful resources for the largest enterprise customers, the *AlphaServer* GS Series also directly addresses the needs of customers with smaller requirements, or even small **initial** requirements and future growth plans. In particular, the system design addresses these varying capacity needs in a most economical manner, wherein the customers really do get what they pay for.

Specifically, the *AlphaServer* GS Series has been designed in a modular manner, wherein each module delivers not only incremental capacity, in terms of processors, memory storage, or IO connectivity, but delivers incremental performance-enabling infrastructure as well.

With this system family, for example, the purchase of an 8 processor system does not require the purchase of 32 or 64 processors worth of supporting infrastructure. It requires only the purchase of 8 processors worth of infrastructure. This way the customer need not pay for any unused resources or infrastructure.

Flexible and Forgiving Software Model

It is important to provide the customer with a software model that allows for straight forward porting of applications as well as some measure of flexibility with regard to how applications or operating domains are applied to the system.

Near-UMA

Although the *AlphaServer* GS Series architecture is technically a Non-Uniform Memory Access (NUMA) architecture, the latency profile of the system, particularly the profile under system load, makes it perform very much like a Uniform Memory Access (UMA) system. The *AlphaServer* GS Series features a basic latency profile wherein remote reference latencies differ from local reference latencies by only a factor of 3x, and the latency profile exhibits very little growth under system load.

Bus-based UMA systems on the other hand feature only a single latency value, but tend to exhibit latency growth of 3-4x their nominal latency when under load. As a consequence, applications that run on the simple bus-based UMA systems should port directly to the *AlphaServer* GS Series in a very straight forward manner.

That said, it is still worth noting that the best system performance is achieved by embracing the NUMA capabilities of the *AlphaServer* GS Series. This allows an application to truly take advantage of the potential of the system design. For many applications, the advanced capabilities of *Tru64 UNIX* will provide the required hooks into the NUMA model.

Partitions

In terms of software flexibility, the *AlphaServer* GS Series supports both soft system partitions, through the *OpenVMS* Galaxy operating system, and hard partitions, through hardware and firmware support.

The *OpenVMS* soft partitions allow compute resources to be apportioned and then dynamically reapportioned under the control of the *OpenVMS* Galaxy kernel. Security between soft partitions is provided within the *OpenVMS* kernel. This type of partitioning is used primarily for workload management. It provides the ability to allocate specific resources to specific applications. It is also able to provide a very light weight and low latency method for transferring resources between applications. This is made possible by virtue of the fact that all resources and partitions are controlled by a single operating system kernel.

Hard partitions allow for multiple operating systems or multiple versions of the same operating system to operate within discrete partitions of the same multiprocessor box. Dynamic repartitioning, like soft partitioning, allows for the partitioned resources in a running system to be migrated between multiple hard partitions, although at a coarser granularity than with soft partitions. Security between the partitions in the hard partition model is implemented in hardware, providing an additional level of security. With these characteristics, hard partitions provide a variety of useful features for the customer. With robust hardware based security and support for multi-operating system environments, hard partitions are ideal for server consolidation. With the addition of support for dynamic repartitioning, hard partitions can also address workload management needs and support rolling operating system upgrades.

Robust Data Center Features—Reliability, Availability, Serviceability (RAS)

Reliability and availability are critical characteristics for all computer users. They are particularly important for the enterprise customer employing computer systems in business critical applications. The *AlphaServer* GS Series meets the needs of these customers with a design that emphasizes not only reliability and availability, but also serviceability and data center layout issues.

Reliability and availability begin with the fundamental design. The fundamental design of the *AlphaServer* GS Series includes ECC protected RAM storage and interconnect. This includes protection on memory data as well as system coherency data. The system also features N+1 redundant main power supplies, redundant power converters and ultra-reliable cooling fans.

The system design further bolsters system availability and serviceability with an abundance of hot-swap capabilities. These include the support for the direct hot-swap of processors, IO subsystems, power supplies and even multiprocessor modular units. The hot-swap capabilities also include support for the indirect hot-swap of memories, Directories and Global Port modules, through the use of the modular unit hot-swap procedure.

Many of the key data center features of the *AlphaServer* GS Series, including hot-swap, are managed by a robust "back door" network of micro-controllers. This network provides basic services for powering, initializing and partitioning the system. It also provides services for tracking the state of the system configuration, as well as monitoring and modulating the system environment. In addition, this network, which operates on auxiliary power source, provides key access to system error state even in the presence of insidious system failures such as processor or power supply failures. The system micro-controller network provides both direct and remote dial-in access to the system and features a redundant failure over capability.

A final feature of the *AlphaServer* GS Series that merits mention here is its remote IO support. Remote IO refers to the practice of including a system's IO busses (e.g., PCI busses) in a separate rack or enclosure from the system kernel. In the case of the *AlphaServer* GS Series, IO busses can reside as far as 10 meters away from the kernel of the system. This provides clear advantages from the data center layout perspective. Customers with enterprise class IO requirements are able to spread out their IO subsystem, easing cable congestion problems. Every bit as important, however, is the fact that remote IO support enhances system serviceability. Moving IO cabling away from the kernel of the system allows the system to be serviced without contending with the disassembly, and associated destabilization, of the machine's IO subsystem. This is critical given the huge investment made in the storage and network infrastructure of today's enterprise applications.

Each of the RAS and/or data center features of the *AlphaServer* GS Series adds value for the enterprise customer. Together, they serve to reduce the Total Cost of Ownership.

Performance

While all of the factors discussed in this section are critical to the enterprise customer, performance is still critical in allowing the customer to execute their business efficiently and cost effectively. Since the third section of this paper will deal with the performance of the *AlphaServer* GS Series in great detail, it will suffice to note here that the system is designed to provide outstanding performance in a variety of commercial and high performance technical applications.

AlphaServer GS Series System Description

The following sections provide a brief overview of the modular components of the *AlphaServer* GS Series and the manner in which the components are combined to build up the various members of the *AlphaServer* GS Series family of computer systems. In the course of this description, the manner in which some of the foundational factors of the system effected the system structure will become evident. This is particularly true of the system's capacity, infrastructure and economic purchasing model.

The Quad Building Block

The *AlphaServer* GS 32-way Series is a modular system design. The fundamental modular unit of the system is the "Quad Building Block" or "QBB". The QBB, shown in figure 1, is comprised of the following resource capacities:

- Up to four 731Mhz *Alpha* 21264 Microprocessor modules
- Up to 32GB of memory storage across up to 4 carriers
- Support for up to 8 PCI busses, which in turn support up to 28 PCI adapters.

The aforementioned resource capacities are supported by the following performance enabling infrastructure capabilities:

- 17.6GB/s of raw interconnect bandwidth
- 6.4 GB/s of maximum, realizable memory bandwidth
- 1.6 GB/s of IO bandwidth
- 1.6 GB/s of per processor bandwidth

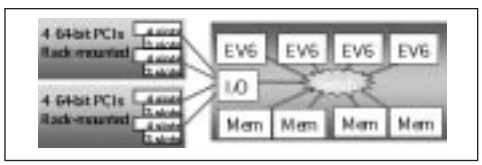

Figure 1: Quad Building Block (QBB) Block Diagram

The *AlphaServer* GS 8-way Series

The systems that comprise the *AlphaServer* GS 8-way Series family are formed by aggregating QBBs. The *AlphaServer* GS 8-way, for example, is formed by abutting two QBBs. The resultant system, shown in figure 2, scales in both its resource capacities and performance-enabling infrastructure. Its resource capacities now consist of:

- Up to eight 731Mhz *Alpha* 21264 Microprocessor modules
- Up to 64GB of memory storage across up to 8 carriers
- Support for up to 16 PCI busses, which in turn support up to 56 PCI adapters

These scaled resource capacities are supported by similarly scaled performance enabling infrastructure capabilities consisting of:

- 32GB/s of raw interconnect bandwidth
- 12.8 GB/s of maximum, realizable memory bandwidth
- 3.2 GB/s of IO bandwidth
- 1.6 GB/s of per processor bandwidth

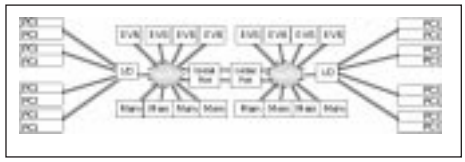

Figure 2: AlphaServer GS 8-way Block Diagram

The *AlphaServer* GS 16-way and the *AlphaServer* GS 32-way

The *AlphaServer* GS16-way and *AlphaServer* GS 32-way systems are formed by attaching QBBs to a "Global" or "Hierarchical" switch. The *AlphaServer* GS 16-way is formed by attaching four QBBs to the switch, while the *AlphaServer* GS 32-way, shown if figure 3, is formed by attaching 8 QBBs to the switch. As in the case of the *AlphaServer* GS 8-way, the *AlphaServer* GS 16-way and *AlphaServer* GS 32-way capacities and infrastructure scale together. The *AlphaServer* GS 32-way, for example, provides the following resource capacities:

- Up to thirty-two 731Mhz *Alpha* 21264 Microprocessor modules
- Up to 256GB of memory storage across up to 32 carriers
- Support for up to 64 PCI busses, which in turn support up to 224 PCI adapters

To support these capacities, the GS 32-way system provides the following performance enabling infrastructure capabilities:

- 140.8GB/s of raw interconnect bandwidth
- 51.2 GB/s of maximum, realizable memory bandwidth
- 12.8 GB/s of IO bandwidth
- 1.6 GB/s of per processor bandwidth

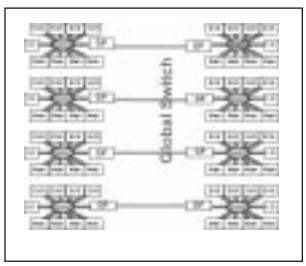

Figure 3: AlphaServer GS 32-way Block Diagram

AlphaServer GS 32-way System Performance

To understand the *AlphaServer* GS Series performance characteristics, it is instructive to take a three-step approach to analyzing performance in general. The first step involves analyzing what it takes to realize the best performance when executing a single processor thread in isolation. In this case the processor

will experience no sharing of data or resources. The second step involves analyzing what it takes to realize the best performance for each of a number of multiple independent threads executing in the same system. In this case, the processor will experience the sharing of resources but not the direct sharing of data. The third and final step involves analyzing what it takes to continue to realize the best performance on mutually exclusive threads, but wherein those threads are communicating and sharing data and system resources. The completion of the analysis in this third step will effectively answer the question of what it takes to realize the best overall performance in a multiprocessor server system.

Step 1—Single Processor Thread Performance

Idle system latency, examples of which are shown in figure 4, is a common number given as an indicator of system performance. It is, however, just one piece of a larger puzzle. To understand the best metrics for evaluating system performance effects, and to better understand how idle system latency fits into the overall performance puzzle, we should begin back at the source of the performance—the processor.

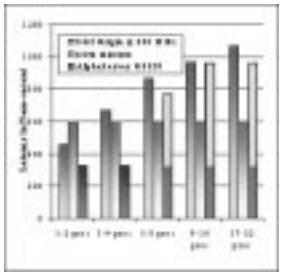

Figure 4: Idle System Latency

Processor performance is governed largely by "demand latency" the time it takes a processor to access an instruction or data item. Processor designs have, for many years, been using schemes like caching and pre-fetching to try to minimize demand latency.

While caching is a technique used to minimize a processor's system references, pre-fetching, out of order issue and speculative execution are schemes that processors now employ to get a head start on bringing system data into the processor's cache system. These methods allow a processor to issue multiple references to the system, often in anticipation of the need for data, instead of forcing a processor's operation to degenerate to the point where it must issue individual demand references. The latency measure associated with these multiple, anticipatory references is referred to as "perceived latency". It is equal to the latency, as perceived by the issuing processor, when the latencies of system references are amortized over the number of references issued by the processor. In modern systems that employ advanced processor techniques, the perceived latency is the best measure of a system's impact on a processor's demand latency.

Perceived latency is a function of three key system parameters: the forementioned idle system latency, as well as bandwidth and the number of outstanding references. Figure 5 illustrates perceived latency as a function of these parameters.

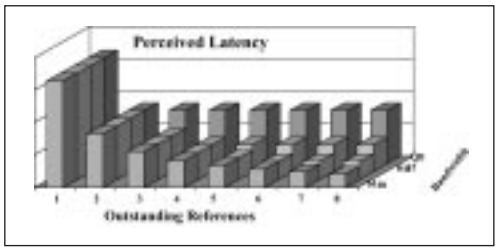

Figure 5: Perceived Latency

The front-most curve in figure 5 illustrates the effect of both the idle system latency and the number of outstanding references on perceived latency. The left most bar illustrates the case of a single outstanding reference. Its value is therefore equal to the idle system latency. Idle system latency is therefore a starting point value from which all other values are derived. As the number of outstanding references increases, and the curve moves from the left of the chart to the right, the perceived latency per reference decreases proportionally. Thus, the more outstanding references that can be supported, the smaller the perceived latency and the better the system performance.

The right-most curve in the chart illustrates the effects of bandwidth on perceived latency. The latency values for the front bars are derived by assuming that the bandwidth that is required to get the best result is made available. As available bandwidth is diminished, and the curves move from the front to the back of the chart, the system is unable to support as many outstanding references. As a result, it cannot achieve the best possible perceived latencies. This, in turn, limits the system's ability to achieve the best performance.

This chart allows a number of conclusions to be drawn with regard to achieving the best system performance for a single processor in isolation. Assuming the assertion that the best system performance is a achieved by a system design that achieves the best perceived latency, it can be concluded, in turn, that to achieve the best system performance, a system design must provide:

- Low idle system latency
- Support for as many outstanding references as possible
- Sufficient system bandwidth to support the maximum number of outstanding references that a processor may issue

Step 2—Multiple Independent Threads

To get the best performance result from multiple, independent threads, a system must be capable of providing the best perceived latency result for each processor in the system. To achieve this, all the requirements from the single thread case must be applied to each processor in the multi-thread case. Specifically:

- Idle system latency must still be kept to a minimum, even as the system grows to support more processors and memory.
- Bandwidth must still be maximized, but in a bifurcated manner. Bandwidth in the memory system must grow, so as not to limit the number of supported references for all of the processors combined. Bandwidth at each processor interface must, at the very least, be maintained so that as processors are added to the system, no individual processor's outstanding reference potential is limited.
- The total number of supported outstanding references must now grow, so that each processor can achieve its best perceived latency through the highest possible level of amortization.

To understand how these goals may be met, it is instructive to examine some lower level computer system characteristics. A computer system's memory subsystem design, the overall system interconnect topology and bandwidth profile and the system's "occupancy" profile are of particular interest.

Memory system design
There are two primary design goals for the memory system of an enterprise class shared memory server. The memory system must first provide sufficient storage capacity to support enterprise class applications. The memory capacity design goal for the *AlphaServer* GS 32-way series was 256GB of storage. The memory system must also provide the bandwidth support for a maximum number of references from all of a multiplicity of processors. The first goal primarily dictates form factor restrictions on the system design. The second goal, however, plays directly into the bandwidth requirements for attaining minimum perceived latency, and maximum performance, for a system's processors.

It is important to recognize that the bandwidth delivered by any multiprocessor shared memory subsystem is highly dependent upon the reference patterns issued to that subsystem. Some system designs may work well for well defined pathological reference patterns, but may not work well for more general, random reference patterns. In modern systems it is important to note that the memory reference patterns generated by processors that feature out of order issue and speculative execution tend to be irregular. In a multiprocessing machine with many such processors, the reference patterns as seen by the memory system tend to be even more irregular. Therefore, in most modern shared memory multiprocessors, a memory system designed around only regular, pathological reference patterns will not work very well.

The *AlphaServer* GS Series memory system development involved substantial simulation, including the simulation of both random and pathological reference patterns. The simulation results indicated that the two most critical factors in extracting bandwidth from a memory system are the memory system's interleave structure and the memory system's interconnect topology.

The study of system interleave indicated that the interleave and bandwidth properties of a system interact such that, if interleave is provided behind a bandwidth bottle neck of some sort, the effectiveness of that interleave is diminished in proportion to the severity of the bandwidth bottleneck. So, for example, interleave across the switch ports of a system, where data movement is free of bottlenecks, is the best kind of interleave. Behind these switch ports, interleave between independently interconnected arrays of SDRAMs is the next best kind of interleave. It is only limited by the bandwidth bottleneck of the switch port. Finally, interleave within an SDRAM part is the least effective type of interleave. It is severely limited by the bandwidth bottle-neck at the pins of the SDRAM part.

A system that makes heavy use of SDRAM interleave will therefore work well on rare, well behaved reference patterns, but will work poorly on random reference patterns.

The *AlphaServer* GS Series memory system design is biased toward the best types of interleave. It includes 32 interleaved switch ports and 64 independently interconnected memory arrays, and it relies on a relatively small 4-way interleaving within its SDRAMs to round out its full 256 interleave units.

To deliver the bandwidth made available by this interleave system to the system's processors, the *AlphaServer* GS Series includes massive, cross-bar switch based memory system interconnect bandwidth. Each memory port provides 1.6GB/s of bandwidth, while each QBB provides 6.4GB/s of bandwidth. With eight QBBs, this provides for a system aggregate of 51.2GB/s of bandwidth. In the *AlphaServer* GS 32-way and *AlphaServer* GS 16-way systems, the QBBs are interconnected by a Global Switch that provides a maximum 12.8GB/s of cross section bandwidth.

Of course the RAMs and wires of a memory system represent little more than memory system bandwidth potential. To deliver that potential to the application, the *AlphaServer* GS Series design supplements the powerful infrastructure elements of interleave and interconnect with two additional performance enabling strategies: "aggressive memory resource scheduling" and "aggressive data link bandwidth management".

The first of these strategies, "aggressive memory resource scheduling", refers to the manner in which the *AlphaServer* GS Series issues references to the memory system. Specifically, the system issues references to memory by means of a scheduling circuit in each QBB's local switch that can choose from, and expeditiously reorder, up to 28 pending references. It issues these references to the 32 interleaved memory units within its QBB in a manner that best utilizes the memory resources.

Most systems employ simple round-robin type arbitration schemes to guarantee fairness between processors. This, as one might imagine, does not yield the best reference patterns with respect to memory utilization. The *AlphaServer* GS Series still maintains the same processor fairness as round-robin schemes, but only on the architecturally required memory block basis. This frees the system to issue reference patterns that take best advantage of the memory system.

The second strategy, "aggressive data link bandwidth management", refers to optimizing the utilization of every interconnect link in the system. The practice of aggressive data link bandwidth management in the *AlphaServer* GS Series is made possible by means of specially designed buffered switch ASICs in both the QBB's local switch and the system's Global Switch. Unlike the simple, unbuffered switches found in systems like HP 9000 V2600 or the Sun E10000, these switches allow an input reference packet to be driven into a switch's buffer regardless of output conflicts with other incoming references. This allows each link to be associated with its own independent link controller, the sole job of which is to maximize the bandwidth on the associated link. The cumulative effect of all of these independent controllers is a higher overall system interconnect utilization.

In summary, the *AlphaServer* GS Series interleave strategy and memory interconnect topology, combined with the aggressive memory resource scheduling and aggressive data link bandwidth management, result in a huge memory system bandwidth, capable of supporting hundreds of outstanding system references

Processor port bandwidth
To achieve the best perceived latency result for multiple mutually exclusive threads, a system must continue to provide enough bandwidth to each processor to support that processor's maximum number of outstanding references, even as the system grows.

Many system designs on the market today aggregate processors behind common switch ports as they grow. This severely limits the bandwidth available per processor and therefore limits the number of outstanding references that each processor can support. As shown in figure 5, this limits perceived latency and overall system performance.

Systems like the HP 9000 V2600 and the Sun E10000 are designed such that the bandwidth available per processor diminishes as processors are added to the system. A fully loaded Sun UE10000, for example, requires four processors to share 1.6GB/s of bandwidth. This yields only 400MB/s for each processor. Similarly, a fully loaded HP 9000 requires four processors and an IO port to share 1.92GB/s of bandwidth. This leaves only 420MB/s for each processor.

The *AlphaServer* GS Series topology does not aggregate processors behind switch ports. As processors are added to the *AlphaServer* GS Series system, so too is processor bandwidth. In that way, each processor in a 32 processor system still has access to the same 1.6GB/s of bandwidth to which a single processor in isolation had access. Each processor can, therefore, continue to support its maximum number of outstanding references, its minimum perceived latency and its best performance.

Occupancy and outstanding references
All of the bandwidth made available by the *AlphaServer* GS Series memory system, the system-wide interconnect, and the processor port interconnect is nothing more than potential bandwidth made available to outstanding references. The system design, in terms of logic design and protocols, must be specifically designed to support large numbers of outstanding references to take advantage of this potential.

The *AlphaServer* GS Series logic is designed according to a principle referred to as "low occupancy", where "occupancy" is defined as the amount of time that any given reference consumes in any given resource in the system. A low occupancy design is therefore a design that minimizes the time that any reference in the system consumes at any given resource.

The occupancy of the references in a computer system directly determines the number of outstanding references the system can support. To understand this intuitively, consider a juggler. A juggler has two resources—his hands. The longer a juggler holds any one ball in his hands, the lesser number of balls he can keep in flight. In other words, the higher the occupancy of any given ball, the lower the number of balls in flight.

With its low occupancy design, the *AlphaServer* GS Series is capable of supporting a huge number of outstanding references. The system is theoretically capable of supporting in excess of 500 outstanding references. In practice, 200-300 outstanding references at steady state would not be an unreasonable expectation. To put these numbers in perspective, consider that the *AlphaServer* GS 14-way system, still an outstanding machine at solving real world problems, can support, at most, 16 outstanding references.

The effect of the low occupancy nature of the *AlphaServer* GS Series, and its corresponding ability to support hundreds of outstanding references, allow the system to amortize the latencies of individual references to the lowest possible perceived latencies. This in turn yields the best overall performance result.

Idle system latency

With an understanding of the system bandwidth and the outstanding reference properties of the *AlphaServer* GS Series, the only variable of the perceived latency equation left to address is the idle system latency. The most noteworthy feature of the idle system latency in the *AlphaServer* GS Series is that, given the system's hierarchical design, there are actually two classes of system latencies. The first class, local latency, refers to references that address memory locations that map to the same QBB as the issuing processor. The *AlphaServer* GS Series' local latency, at approximately 330ns, is a best-in-class enterprise server latency. The second class of latency, global latency, refers to references that address memory locations that map to a QBB other than the QBB of the issuing processor. The *AlphaServer* GS Series' remote latency, at 960ns, is just under 3x its local latency.

Our understanding of perceived latency has, to this point, only involved a single idle system latency variable. With the introduction of two latencies to this model, we must introduce a new metric, "average idle system latency". Average idle system latency is defined to be equal to the local latency times the percentage of references that address a local memory location, plus the remote latency times the percentage of references that address a remote memory location. If 100% of the references are local references, then the average idle system latency will be equal to the local latency. If 100% of the references are remote references, then the average idle system latency will be equal to the remote latency. Otherwise, the resulting average idle system latency will fall somewhere between the local and remote latencies.

When the distribution of references is weighted in favor of local references, the processors perceive a better average idle system latency and as a result, achieve better performance. This means that the memory location of data structures, relative to the processor on which their associated process is executing, can have a significant effect on system performance. This, in turn, means that software applications, and operating systems in particular, can have a direct effect on average idle system latency. Specifically, process specific pages can be mapped to the same QBB as the processor on which the process is running, system data structures can be distributed across all QBBs, and text pages can be replicated in all relevant QBBs. All of these techniques allow software to minimize average idle system latency, and as a result, improve system performance.

The concepts of average idle system latency and manipulation of latency via software not withstanding, the *AlphaServer* GS Series' variable average system latency needs to be reconciled with respect to the fixed latencies of competing uniform memory access (UMA) computer systems. Consider the chart in figure 6, which contrasts the average idle system latency of the *AlphaServer* GS 32-way system with the fixed idle system latency of Sun E10000. This chart seems to indicate that if more than 55% of the references in a *AlphaServer* GS 32-way system are local references then the *AlphaServer* GS will perform better than the E10000. Conversely, if more that 45% of references in the GS 32-way system are remote references then the E10000 will perform better then the GS system. What is important to remember is that the latency values illustrated in this chart are simply idle system latency values. They are simply one variable in the perceived latency equation. As a result, it is important to explore how these values function within the perceived latency framework before trying to draw relative performance conclusions.

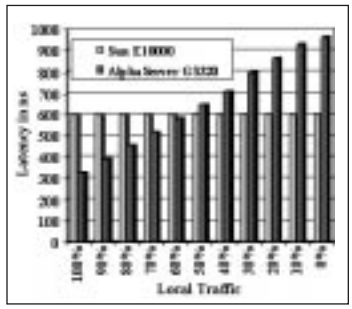

Figure 6: Comparison of Average Idle System Latency

To formulate an effective perceived latency comparison between systems with vastly different interconnect topologies, it is instructive to analyze a number of basic system interconnects. In doing so it is possible to characterize a variety of systems' perceived latency profiles as functions of the systems' combined latency, bandwidth and occupancy properties.

A small, switch-based system, similar to the *AlphaServer* GS 23-way's QBB, is a good starting point for a topological analysis. Such a system is illustrated in figure 7. With a geographically small switch and only one processor or memory unit per switch port, this topology has ideal bandwidth and latency properties. As such, the characteristics of this topology can best be described by the front, ideal curve in the perceived latency chart, shown again in figure 8.

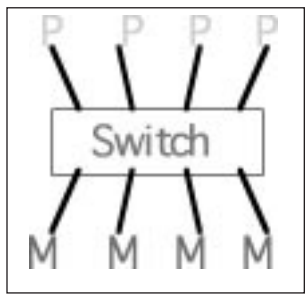

Figure 7: Small, Ideal Switch

This topology, however, is too limiting in terms of resources to be considered for an enterprise class system. To find a topology for an enterprise class system, this small topology must be grown.

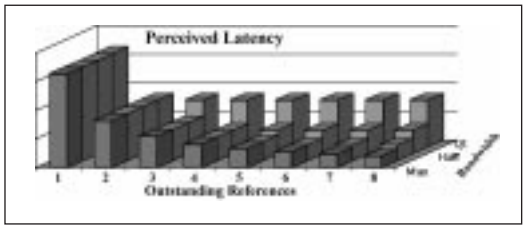

Figure 8: Perceived Latency for Small, Ideal Switch

The first manner in which this small switch can be grown is by literally stretching the switch and adding many more processor and memory ports. This type of topology, shown in figure 9, preserves the ideal bandwidth properties of the small switch. The massive geographical size of the switch, however, pushes the limits of physics and results in a much larger idle system latency. This, in turn, produces a perceived latency curve, shown in figure 10, similar to our ideal curve. This new curve, labeled "stretched" in figure 10, has a similar shape as the ideal curve, but at each point on the curve, the larger topology exhibits a substantially larger latency.

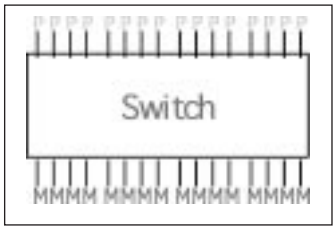

Figure 9: Stretched Switch

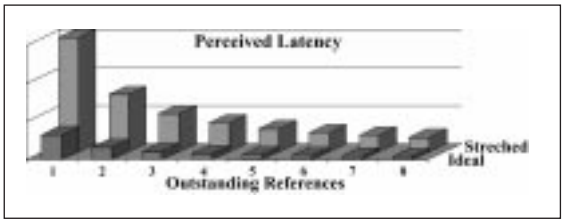

Figure 10: Perceived Latency for Stretched Switch

Another manner in which the small switch can be grown is by stretching the switch in a more moderate manner, and then aggregating processors and memory units behind the reduced number of switch ports. This type of topology, shown in figure 11, is employed in the Sun E10000, the HP 9000 V2600 and a number of other competing enterprise servers.

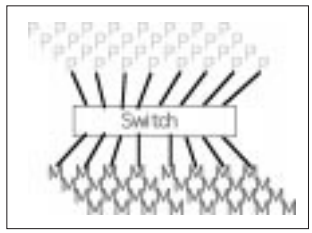

Figure 11: Aggregate Switch

This topology provides an idle system latency that is necessarily larger than that of the small switch, but it is substantially better than that of the stretched switch. The aggregating of processors and memory resources behind a smaller number of switch ports, however, creates substantial bandwidth bottlenecks in the system. This limits the number of outstanding transactions that the system can support and, as shown in the "aggregate" curve in figure 12, enforces a limit on the perceived latency that the system can achieve. This in turn limits the system's performance.

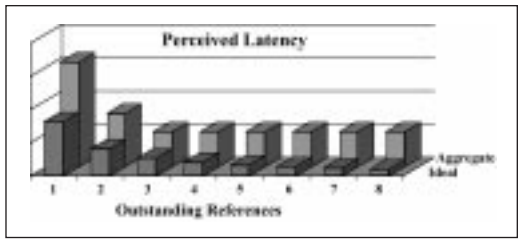

Figure 12: Perceived Latency for Aggregate Switch

A third manner in which the system can be grown is by interconnecting multiple copies of our small switch through a hierarchical or global switch. This topology, shown in figure 13, is the *AlphaServer* GS 32-way model.

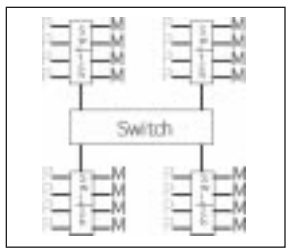

Figure 13: Hierarchical Switch

This topology maintains the ideal processor and memory port bandwidth properties of the small switch. At the same time, it introduces a cross-section bandwidth that can become a bottleneck when operating with a high percentage of remote references. This topology also introduces the local/remote idle system latency profile. These elements together produce a set of amortized latency curves, as opposed to a single curve. The set of curves, illustrated in figure 14, range from the unencumbered ideal latency curve, when all system references are to local memory locations, to a bandwidth limited, higher latency curve when all references are to remote memory locations.

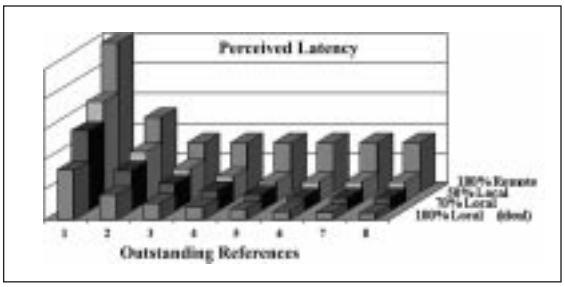

Figure 14: Perceived Latency for Hierarchical Switch

All of the topologies enumerated here can be evaluated with respect to each other by putting all of their perceived latency curves on the same chart. This chart, shown in figure 15, demonstrates that the hierarchical topology exhibits a far greater potential for minimizing perceived latency and maximizing performance.

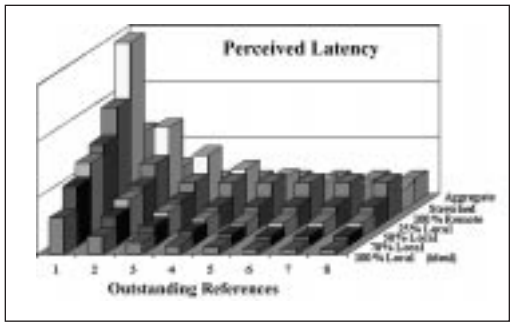

Figure 15: Comparison of Perceived Latencies

Even in the 75% remote traffic case of the hierarchical topology, where the idle system latency of the hierarchical topology is worse that the idle system latency of the aggregate topology, the hierarchical system needs to get only four outstanding references per processor to achieve a better perceived latency than the comparable latency in a aggregate system. At 50% remote traffic, a much more reasonable, if still pessimistic, operating point, the hierarchical system need only get 2 outstanding references per processor to achieve a better perceived latency than any other topology.

The overall conclusion from this chart, therefore, is that even though that hierarchical system may not, in the strictest sense, exhibit the smallest idle system latency, it does exhibit the best perceived, or amortized, latency profile and can therefore deliver the best overall performance of all of the system topologies.

Performance summary—Multiple mutually exclusive threads
The chart in figure 15 summarizes most of the elements required to achieve the best performance in a system with multiple mutually exclusive threads. It is worth, however, calling these elements out explicitly.

In summary, minimum perceived latency per processor thread is the primary system requirement for achieving the best performance for multiple, mutually exclusive processor threads. To achieve this, a superior memory system design is required. This design should include a superior memory interleave strategy, massive interconnect bandwidth, aggressive resource scheduling and aggressive data link bandwidth management. The system should also be designed according to the principles of low occupancy, with system references consuming as little time as possible in any given system resource. Finally, the system's topology should be optimized for minimum perceived latency, not idle system latency. Based on the preceding topological analysis, this implies that the system should provide multiple system latencies, including an extremely low "local" latency. It implies that the topology should preserve per processor bandwidth, and it implies that the topology should include any bandwidth bottlenecks in such a manner as software can mitigate their effect.

Step 3—Multiple Threads Sharing Data and Communicating

The analysis of multiple, mutually exclusive threads established specific requirements regarding system bandwidth, latency and occupancy, toward the goal of establishing the best possible perceived latency and associated system performance. When layering data sharing and processor communication on top of these mutually exclusive threads, the goal is to do so in such a way that the aforementioned bandwidth, latency and occupancy requirements are impacted as little as possible.

The new overhead introduced with data sharing and inter-processor communication, is comprised of two fundamental components: a data (or cache) coherency scheme and a data (or memory) consistency scheme.

The first component, a cache coherency scheme, is required to support the sharing of data. Cache coherency schemes typically consist of a coherency state storage system and a protocol that runs on the stored coherency state. To preserve performance, the coherency storage system must be implemented such that it does not limit system bandwidth as processor and memory bandwidths grow, or as processor and memory capacities grow. The coherency protocol, on the other hand, must be implemented such that it does not substantially increase either the average system latency or the occupancy of any given system reference. If any of these values are affected substantially, perceived latency will increase and system performance will suffer.

The second component, a memory consistency scheme, is required to support inter-processor communication. This is normally done through some notion of ordering events in the system. The maintenance of ordering events in a system represents a new system wide overhead. The key to minimizing the effects of this new overhead is to try to include it in such a way that only the relatively rare inter-processor communication event suffers any additional penalty associated with ordering; all normal computational references should, ideally, be able to proceed with no additional overhead. In addition, the entire consistency model should be implemented such that it does not increase the occupancy or latency of any system references.

Coherency Storage Topologies

The foundation for any cache coherency scheme is the coherency state storage topology – the collection of directory or tag stores that track the state of memory blocks. For best system performance it is critical that this system not limit system bandwidth and not increase system occupancy. All of the common coherency storage topologies either fail to meet these performance goals, or are simply impractical.

Snoop based coherency schemes are the most common schemes employed in the industry. They are employed in many small-scale multiprocessors and even some enterprise class servers such as the Sun E10000. On the positive side, these schemes tend to allow for low system occupancy. On the negative side, they also tend to limit system bandwidth. Adding duplicate tag stores, or even interleaved duplicate tag stores in the case of the Sun E10000, can mitigate the bandwidth limitations of this scheme somewhat. However, they are not adequate to support a memory system with the 51.2GB/s of bandwidth required in a system like the *AlphaServer* GS 32-way system.

Traditional, full directory schemes, wherein there is a comprehensive directory entry, with some number of bits for every processor in the system, for each memory block in the system, tend to meet both the bandwidth and occupancy requirements for best performance. However, their size and cost make them impractical for most systems.

In lieu of full directory schemes, system designs that choose directory based coherency often employ managed or cached directories. These provide the same bandwidth as the full directory scheme without the size and expense. Cached directory schemes, however, require cache replacement policies. When applied to directory stores, replacement policies entail the blocking or retry of traffic while all copies of memory blocks associated with a directory entry are returned to memory. This necessarily increases the occupancy and lowers the performance of the system as a whole.

The *AlphaServer* GS Series employs a coherency storage scheme based on a full but abbreviated directory. This directory includes one entry for every data block in memory, but limits the size of each directory entry. Since it is a full directory scheme, the *AlphaServer* GS Series coherency storage scheme exhibits ideal bandwidth and occupancy properties. Unlike the traditional full directory scheme, it is abbreviated in size, so it is practical in terms of implementation and cost. It adds a paltry 2.5% cost overhead to its associated memory system.

To enable the *AlphaServer* GS 32-way system to run a meaningful and performance optimized cache coherency protocol based on an abbreviated directory store, the system also includes distributed duplicate tag stores. One duplicate tag store is implemented in each of the system's eight possible QBBs. Each of these tag stores is responsible for directly supporting only the 6.4GB/s of memory bandwidth associated with its QBB. As a result, the duplicate tags, which inherently exhibit excellent occupancy characteristics, can be included in the system without limiting overall system bandwidth.

Cache Coherency Protocol

To complete a cache coherency system, a protocol must be layered on top of the coherency storage elements. For the overall system to achieve the best performance, this protocol must neither cause the occupancy of references in the system to increase, nor cause the average system latency to grow in any substantial way.

When data sharing is added to the system, it is inevitable that average latency will grow to a certain extent. The need to occasionally fetch data from other caches instead of memory will certainly cause latency to grow. Similarly, delays as a result of data dependencies between simultaneously pending references will also cause latencies to grow. Fortunately, the effect of this inevitable type of latency growth can be limited such that it penalizes only specific dependent references.

Many common cache coherency schemes resolve data and coherency dependencies by means of retrying references. In such schemes, if a reference is issued to a memory location to which a previous reference is still pending, the system will not process the new reference. Instead the system will send a message back to the issuing processor causing the processor to retry the reference. In systems that employ a dependency retry mechanism, latency growth as a result of dependencies can be quite unpredictable. For heavily contended blocks, the latency for a retried reference as seen by a given processor can grow quite large. In fact, it can actually grow to the point of being infinite, in which case a processor will be starved to a halt. In addition, the regular retry of references diminishes the amount of real usable bandwidth made available for other work.

Other common cache coherency schemes resolve data and coherency dependencies by means of blocking references. In these schemes, if a reference is issued to a memory location to which a previous reference is still pending, the system will cause the new reference to wait at the coherency storage element associated with the addressed memory block until all previous references are resolved. To support such schemes, system designs must include some type of wait station at each of the coherency storage elements. These wait stations, unfortunately, are inherently high occupancy structures, with significant impact on overall system occupancy. As outlined in the mutually exclusive threads section, high occupancy limits the number of outstanding references that a system can support. This limits the ability to amortize perceived latency down to its lowest values. The high occupancy of a coherency protocol, in particular, also directly impacts latencies due to the substantial queuing delays it inserts into the system. The unfortunate characteristic of these queuing delays is that, given that they build up around the common coherency storage elements, they tend to not only increase the latencies of specific dependent references, but all references that require access to a given general coherency storage resource. This brings down the performance of the whole machine.

The *AlphaServer* GS Series implements a new, starvation-free, low occupancy, cache coherency protocol. This protocol neither blocks nor retries references when they experience dependency issues. All dependencies are resolved in such a way that only dependent references experience latency penalties. To this end, all dependency resolution occurs at the periphery of the system, so that independent references from one set of processors need not suffer any penalty as a result of dependent references from some other set of processors. In other words, there is no coherency based queuing penalty at the common resources in the system. The resulting protocol effectively eliminates any occupancy penalty due to cache coherency and minimizes the latency penalty associated with data dependencies to approximately equal the aforementioned inevitable latency penalties.

While the *AlphaServer* GS Series' coherency protocol was designed for straight forward implementation and verification, it is a theoretically complex protocol. Since its completion as a formal protocol, it has been rigorously simulated and subjected to a full formal proof of correctness. While these efforts discovered some minor implementation problems, no errors were found in the protocol itself.

Memory Consistency

An ordering or consistency scheme is required to support inter-processor communication. As previously noted, the keys to minimizing the effects of the overhead of a consistency model consist of: limiting the effects of the consistency overhead to references explicitly involved in the communication and by minimizing the impact of the consistency model on system occupancy and latency.

System designs today use a variety of memory consistency models ranging from "sequential consistency", to "processor consistency", to more relaxed consistency models. Many of these models are very restrictive in terms of the optimizations that the systems are allowed to make. Sequential consistency, for example, even disallows read-to-read reordering, forcing a given read to literally complete and retire before a subsequent read can retire. Some of the more relaxed models, such as the Sun SPARC, IBM PowerPC and the Compaq *Alpha* models, support significant reference reordering and optimization. To provide functional guarantees to software, each of these models includes fence or barrier instructions. At the time of a given barrier instruction's completion, it is guaranteed that the executing processor has a view of the system memory state that is consistent with all processors that have previously executed barriers. It is also guaranteed that all processors that subsequently issue barriers will have a view of memory that is consistent with the issuing processor.

Consistency in all models requires "inheritance of ordering". This means that, for every reference **n** to block **X**, the issuing processor is consistent, with respect to the rest of the system, not only when it has a consistent view of the references that occurred between reference **n** and reference **n-1** to block **X**, but also when it has a consistent view of all references that preceded reference **n-1**. Simply put, when determining what happened before reference **n**, you must "inherit" everything that happened before reference **n-1**. This may seem obvious and trivial, but in distributed shared memory servers, it is a complicated problem to solve. Most system designs typically leverage the blocking or retry mechanisms used in their cache coherency schemes to guarantee proper inheritance of ordering. Reference **n** would be blocked at the directory, or forced to retry at the directory, until the entire system was guaranteed a consistent view of memory with respect to reference **n-1**. This type of scheme, however, not only burdens all references with ordering overhead, it also does so in a way that both increases the average latency per transaction and increases the occupancy per transaction. Both, of course, degrade system performance.

The *AlphaServer* GS Series' consistency model has two key properties that allow it to avoid the pitfalls of the more common protocols. These properties are what we refer to as "Eager Data Replies" and "Active Inheritance".

The term "Eager Data Replies" refers to the practice of divorcing the data movement, or coherency change portion of a memory reference, from the ordering or consistency portion of a reference. The data portion of the reference is returned to the issuing processor as expeditiously as possible, while the consistency portion is returned, in an independent pipelined manner, as expeditiously as possible, but only after all consistency requirements have been met. Eager Replies allow processors to execute upon the data returned from a system reference as soon as the data portion of that reference is "eagerly" returned to the processor. Only the relatively rare barrier instructions need to wait for the consistency portions of system references. More generally this means that only instructions explicitly involved in inter-processor communication—the aforementioned barrier instructions—are burdened with the added overhead of system ordering.

"Active Inheritance" refers to the practice of resolving memory consistency without resorting to the high occupancy and high latency tactics of blocking and retrying references. This technique allows a reference **n** to inherit consistency information from reference **n-1**, even if reference **n-1**'s consistency is not yet completely resolved. Active Inheritance is, as a result of the low occupancy cache coherency protocol in the *AlphaServer* GS 32-way, a necessity in this system. The *AlphaServer* GS Series' support for active inheritance is made possible by the unique division of labor the system employs in the management of its ordering messages and the "bus-like" nature of certain channels of transactions through its Global Switch. Unlike many common systems, the *AlphaServer* GS Series does not employ "Invalidate Acks" or "Write Acks" for the purpose of maintaining ordering. Instead, the *AlphaServer* GS 32-way system associates ordering messages with both read and write references and uses what is effectively a single ordering point in the Global Switch as a central point of consistency. This arrangement allows the system design to simply maintain hints regarding the state, relative to the central point of consistency, of previously issued references.

The combination of Eager Data Replies and Active Inheritance allows the GS Series consistency model to meet all performance requirements—normal operations are not burdened with ordering overhead, and consistency (including inheritance) is maintained without increasing average latency or occupancy. In addition to its operational advantages, this model is advantageous in that, although it is theoretically complex, it is very simple in practice and implementation. Even with regard to its theoretical complexity, it is worth noting that its correctness, like the correctness of the *AlphaServer* GS Series' coherency protocol, has not only been rigorously simulated but formally proved as well.

Performance Summary

With our analysis of cache coherency and system consistency complete, we have finished our three step analysis of system performance. We can now answer the question of what it takes to deliver the best performance in an SMP server system. The general answer is that it takes a system design that delivers the best perceived latency. The more specific answer is that it takes:

- A system topology that is optimized for perceived latency, not idle system latency
- A system topology that preserves per processor bandwidth
- A superior, high bandwidth memory system design
- A low occupancy system design that can take advantage of the provided bandwidth
- A cache coherency protocol that preserves available bandwidth, low occupancy and low latency
- A consistency model that that also preserves available bandwidth, low occupancy and low latency

Conclusion

The architecture of the *AlphaServer* GS Series family of systems integrates aggressive switch-based interconnect topology with an innovative new cache coherency and memory consistency architecture. The combination of these technologies makes the *AlphaServer* GS Series ready to grow past the limitations of conventional snoop and bus based systems. These performance advantages, combined with customer focussed data center, availability and maintainability features, make the *AlphaServer* GS Series the ideal choice for a wide range of enterprise applications.

Prepared by the Compaq High Performance Server Business Unit, Business Critical Servers Division.

Part # 12FT-0400A-WWEN

COMPAQ

AlphaPowered