Extending High-end System Performance: from SMP to NUMA

A Data General Technical White Paper

I. Background

Impressive as uniprocessor performance increases have been over the past decade, they have been insufficient to keep pace with the burgeoning performance needs of modern applications. As a result, most of today's high-end servers are increasingly designed as multiprocessor systems. The modular nature of many multiprocessor systems offers additional benefits:

- existing systems can be easily upgraded through the addition of processors to accommodate application or user base growth
- system availability can be enhanced through the elimination of single points of failure as processing power is spread among a number of CPUs
-

The vast majority of these commercial multiprocessor systems use an architecture called Symmetrical Multiprocessing or SMP. This multiprocessing model is implemented through a number of hardware and operating system features which maximize the efficient processing of jobs and I/O by each CPU. SMP provides many advantages, including ease of application software development and a shared resource computing model which is very efficient for a wide range of commercial applications.

As applications demand more and more throughput from high-end systems, however, it is becoming apparent that SMP has limitations as the architecture of choice for the highest level of performance (while remaining the preferred solution for the midrange and low-end systems). These limitations arise from physical and cost constraints imposed by the need to balance CPU and memory systems across a monolithic system bus.

These limitations have generated a certain interest in Massively Parallel Processors (MPP). However, this architecture, which has its roots in technical computing, has many limitations of its own for the commercial market – most notably the need for special application software development.

This White Paper examines another architecture - Non-Uniform Memory Access (NUMA) - one better suited to meeting the needs of commercial users and applications and better matched to industry trends such as platform-independent software and commodity hardware components. The benefits of the NUMA architecture include (further details on benefits can be found in section 4):

- Improved price/performance through the use of high-performance commodity components
- High availability by minimizing single points of failure
- Easy upgradability through the use of modular components
- Low-effort software development since the programming model is a familiar one
- Scalable performance as additional processing components are added to systems

This paper will define NUMA in the broader context of computer architectures and will cover, at a high level, the phased approach which Data General will take to NUMA – first as a performance-enhancing extension to a bus-based system and then as a means of leveraging high-performance, high-volume industry-standard server components. It will not attempt to cover detailed implementation specifics such as interconnection architectures relative to future Data General products. Rather this paper will discuss the basic components of Data General's systems strategy, both with respect to the hardware and to DG/UX (Data General's UNIX system-based operating system), and will describe how NUMA, as an architecture, fits with that strategy.

At this point, NUMA represents Data General's **strategic** direction for **high-end** systems based on Intel's **P6** architecture and beyond. "Commodity" P6 motherboards (currently expected to be available in up to quad-processor configurations), in addition to all P5 systems, will continue to be based on SMP technology. As this White Paper explains, NUMA extends the cost-effective range of SMP. It does not replace it.

II. NUMA Overview

Multiprocessor systems based on NUMA represent an extension to the SMP model of computing. NUMA brings unique characteristics that address the difficulties in creating balanced systems which hardware vendors increasingly encounter in the design of large SMP systems. At the same time, however, NUMA keeps the same simple programming model offered by SMP designs.

NUMA architectures were, until recently, largely seen almost exclusively as subjects of study in university research environments. Recent announcements have started to change this. For example, Convex's Exemplar SPP systems use NUMA concepts in their architecture.

Industry watchers are also starting to pay attention to NUMA as a logical extension to the SMP architecture which can bring high processor counts to bear on commercial data processing problems, without the programming and other software issues associated with MPP systems and other large-scale distributed architectures.

Models for Multiprocessing

- **Shared Memory**
- **UMA (Uniform Memory Access)**
- Multiprocessors with full shared memory and nearly uniform memory access times. This type encompasses traditional SMP architectures.
- **NUMA (Non-Uniform Memory Access)** Multiprocessors with shared memory in which an individual CPU can access near memory rapidly and far memory more slowly.
-
- **Distributed Memory**
- **NORMA (No Remote Memory Access)** Message-based multiprocessors with no memory shared among processors. Massively Parallel Processing (MPP) architectures and clustered systems fall under this model.

A short background on computer architectures provides a context for NUMA. The sidebar at the right gives a high-level view of the computer architectures of interest to commercial computing. Technically, these are all Multiple Instruction, Multiple Data (MIMD) computers.

There are two important points regarding this sidebar. The first is that NUMA is a basic computer **architecture**. NUMA, in itself, says nothing about processor types, interconnection technologies and topologies, or anything else related to a specific implementation.

The second is that NUMA is a shared memory architecture just like SMP and in contrast to MPP[1]. Shared memory architectures allow all processors in a system to directly access all global memory in a system. No message-passing is required to gain access to memory. A single operating system image controls all resource allocation and contention for the system. The only difference between NUMA and more traditional shared memory architectures is that in the latter case, memory access speeds are approximately

[1] MPP is used in this context to refer to "Share Nothing" systems (see Figure 2) which can be thought of as networks in a box, such as IBM's SP2.

uniform while in NUMA, they are not. Of course this “small” architectural difference has important implications for computer design.

Before moving on, it’s worth also taking a look at the view of computer architectures presented in Figure 1. This figure shows general-purpose computer architectures as expanding in two directions from the simple uniprocessor represented in the lower left-hand corner.

Distributed Memory: Clusters and MPP

As we move up the vertical axis in Figure 1 beyond simple failover configurations, we encounter two basic types of "small" (up to about 8 processor nodes) cluster configurations - those which operate against a shared disk and those which operate against largely independent disks.

A Shared Disk Cluster, such as one based on the Oracle Parallel Server Option, allows multiple processor nodes to share a single database through a distributed lock manager mechanism. An Oracle Parallel Server Option cluster can a "single system image" present to users and system administrators. In other words, the multiple systems making up the cluster will appear as if they are a single system in most contexts, both to users and system managers. Since the cluster will keep running even after the failure of a constituent system, the cluster has excellent high availability. Furthermore, since under normal circumstances all clustered systems are active, processing power is used more effectively than with a simple failover configuration in which one system normally acts as a hot backup.

The limitation in a shared disk cluster lies primarily in contention at the distributed lock manager and, ultimately, at the shared disk or database itself. For this reason, as clusters grow and blend into the Massively Parallel Processing model, we move back to the more traditional mode of distributed computing in which disks and databases or database segments are allocated to individual processors rather than shared across all nodes.

This fully distributed or networked style of multiprocessing has been most commonly found applied to solving certain types of complex technical problems rather than in the commercial arena. This is because technical applications, such as finite element analysis and other array-oriented problems, are generally more suitable to fine partitioning than are typical commercial tasks. While all styles of multiprocessing benefit from a task construction which allows threads or processes to proceed with a minimum of synchronization, such partitioning is an explicit requirement with MPP systems.

MPP’s attraction has been the ability to easily string together commodity **hardware** components as a cost-effective alternative to large, expensive mainframes or other monolithic, and expensive, computer systems. The largely independent processor-to-disk streams in MPP can also potentially allow for very large systems without performance limits imposed by resource contention.

MPP application developers must deal with a variety of load sharing, failover, and synchronization issues in their software and system configuration in spite of the efforts of some database software to hide these significant software complexities. In contrast, these software issues are dealt with in a largely transparent manner by the hardware, operating system, and platform software in SMP systems. **The bottom line is that extensive software modification is required to use an MPP system effectively.**

In addition, MPP or Share Nothing systems, such as IBM’s SP2, force strict requirements on database administrators and users: data must be distributed in the same way that it is queried. The database administrator must know how users will view the data in detail before building and loading the database. In oder to optimize reponse times, users also must understand the data distribution before constructing queries.

In advising users on the choice between MPP and SMP architectures (in particular SP2), the Meta Group recommends that “Users should avoid MPP architectures when SMP will do (500 GB today, 2TB+ 1996/97). Through 1997/98, IBM will continue to lag the market as Shared Memory Clusters, not MPP, becomes the prevalent high-end architecture.”[2]

[2] “IBM SP2: Habeas Corpus Redux.” META Fax, The Meta Group, July 3, 1995.

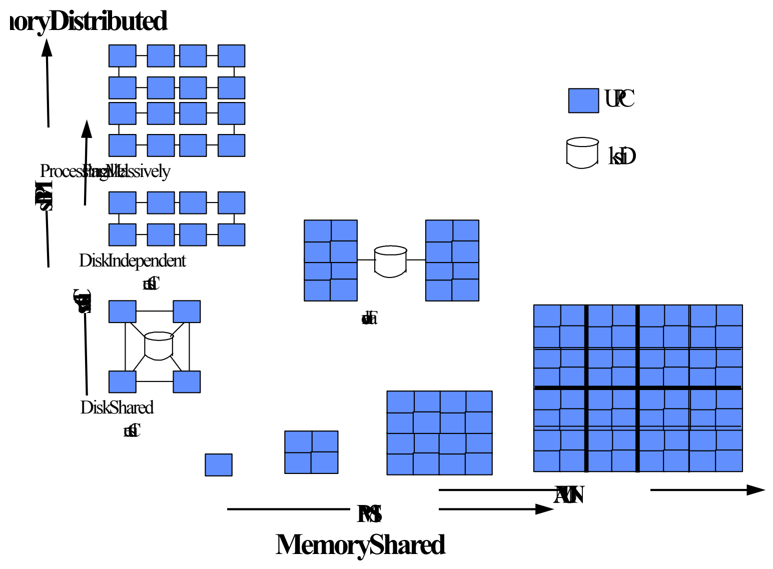

Figure 1. Extending Parallelism (adapted from D. H. Brown).

For this reason, commercial computing continues to evolve generally along the lines shown in Figure 2. Like all evolution, the lines are not always clean – SMP is often used together with clusters to achieve higher performance and increased availability, and MPP will likely continue to have a niche for certain specific high-end applications. Nevertheless, the industry prefers a dominant computer model which imposes minimum software changes at the application level.

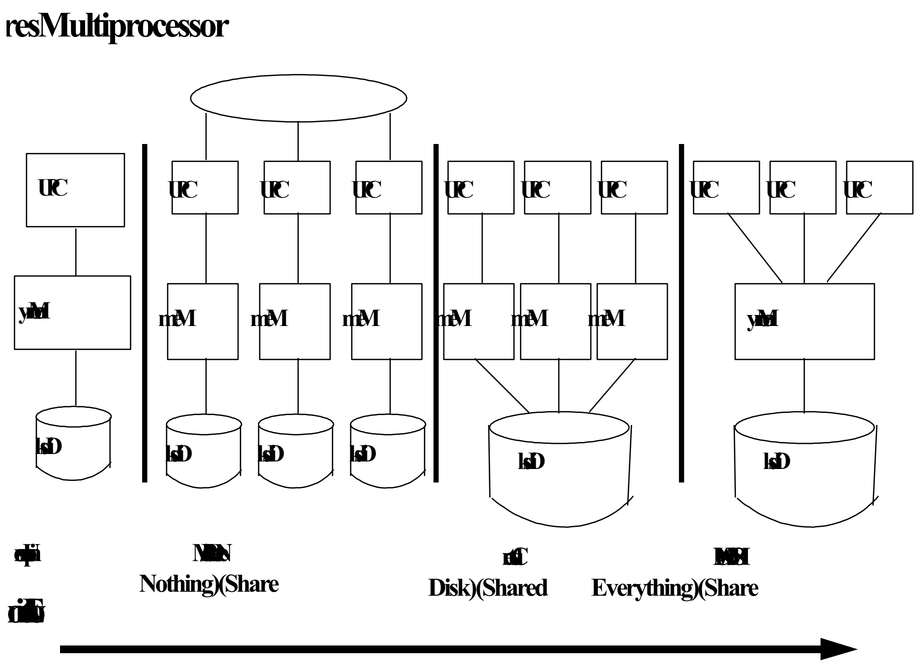

Figure 2. The evolution of multiprocessor architectural styles.

Shared Memory: SMP and NUMA

SMP continues to be the dominant multiprocessing architecture in commercial environments in large part because the operating system and hardware present a programming model to application developers which appears little different from that presented by a uniprocessor system. A shared memory image, as a result, is superior for programming and code portability[3]. As processors become faster and faster, however, some practical limitations of the SMP architecture are driving research and implementation of hybrid designs, including NUMA.

SMP systems work well with commercial applications because the shared memory model allows for efficient resource sharing. Shared memory systems operate under the concept of a set of global resources, the access to which is managed by the operating system. Data General's SMP model implemented in the DG/UX operating system kernel is fully re-entrant, multi-threaded and pre-emptible. As shown in Figure 3, DG/UX's two level of schedulers and the virtual processor concept ensure that:

- Any kernel process can execute on any kernel
- Any user process can execute on any processor
- More than one user can execute in the kernel simultaneously
- All I/O can be initiated and completed on any processor including starting an I/O operation on one processor and completing it on another

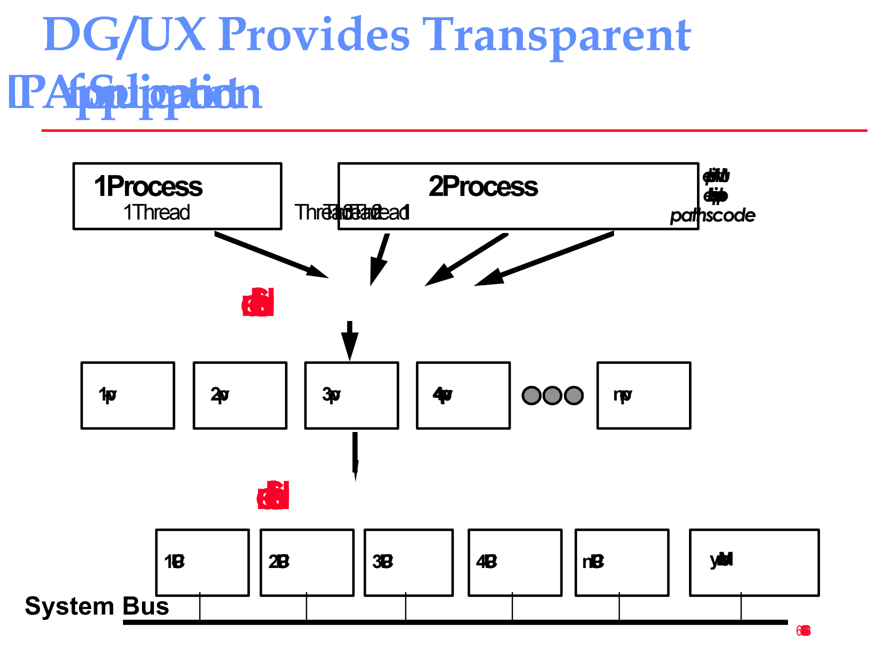

Figure 3. Data General's implementation of SMP in DG/UX.

As a result, AViiON servers running the DG/UX operating system provide near linear scalability even when resources such as memory and the system bus must be extensively shared among processes and threads. A good example of this is Data General's AV 9500 which efficiently scales up to 16 processors. The limitation comes when contention develops for global resources, such as when there are not enough cycles in the system bus to handle required memory read and write operations.

Resource capacity constraints can be, and have been, dealt with in SMP systems through a wide range of increasingly sophisticated methods in both hardware and the operating system. The system bus has been a major focus of activity since, as the central "hub" of a system, it touches most subsystems and, therefore,

[3] D. Feinberg of Gartner Group, "Trends and Directions of Parallel RDBMS Vendors," Gartner Group Fourth Annual Symposium on the Future of Information Technology, October 3-7, 1994.

has a global effect on system performance. Many SMP optimizations which, at first glance, may appear to be aimed at another area of the system such as memory are, in fact, really aimed at minimizing system bus traffic and thereby increasing its effective capacity.

Keeping system capacity ahead of the processing power curve is a difficult and increasingly expensive proposition. The only relief to the essentially monolithic nature of the system bus is to add complex features such as dual data busses. Running up the cycle time and widening the bus can also help but, again, only by adding considerable complexity and development cost. These costs are then passed on to customers in terms of higher system prices. Most importantly, once all these improvements are made, the bus could still be behind the power curve introduced by the next doubling of commodity microprocessor technology which requires twice the bus bandwidth to be used effectively.

III. NUMA - A Logical Progression for Data General

While NUMA is just starting to be talked about in the general computer literature, some of the fundamental concepts behind NUMA have already been introduced and shipped by Data General. Specifically, the Hierarchical Affinity mechanism introduced in DG/UX 5.4 R3.10 together with the large (32 MB) third-level cache on the AV 9500 Plus quad-processor board is a start at solving a subset of the problems which NUMA addresses. This combined hardware/software architecture represents the first step in breaking away from the monolithic system bus model in which all memory traffic must traverse this main system bus.

This is **not** to say that the AV 9500 Plus is a NUMA machine. The 32 MB of memory on each quad-processor board is treated logically as a cache and is not part of global memory. Nevertheless, the concept of a portion of memory being physically close to a group of CPUs (and off the system bus) delivers **some** of the advantages of a NUMA system.

Key Components of NUMA

NUMA introduces the concept of each CPU (or group of CPUs) having near and far memory. Near memory is memory accessed by a CPU (or group of CPUs) through some sort of local bus. Such memory can be accessed with low latency. Far memory (i.e., another CPU's near memory) is accessed over a system bus or other coherent memory interconnect mechanism and that access will have a longer latency (and potentially lower bandwidth).

Several points here are key:

- Far memory accesses are still direct memory accesses within the SMP model. There is no inter-CPU message-passing required and, most importantly, the programmatic interface is still just the same as for a conventional SMP system.
- Since many or most memory accesses will be to near memory, helped along by invisible operating system extensions deriving from the hierarchical affinity mechanism described earlier, CPU interconnect traffic is **much** reduced. This allows for interconnecting not only more processors but also for simpler connections to potentially substitute for very complex and expensive system bus designs. The result is better price/performance for the customer.
- Finally, by decoupling the interconnect from the CPU hardware, possible because a large monolithic backplane is no longer a requirement as bus traffic decreases, CPU components can increasingly be built from commodity parts. This capability matches one of the promises of MPP: commodity building-blocks. **NUMA allows for the hardware economies of commodity building-blocks with the software economies of the SMP programming model**.
-

Data General's NUMA Hardware Directions

As noted earlier, NUMA is "only" an architecture and, as such, does not speak to any implementation details. NUMA can be implemented in many different ways. CPU cards with local memory on a system bus can comprise a NUMA system. So can multiple server boxes based on Standard High Volume (SHV) commodity boards interconnected with a high-bandwidth external bus.

Our first implementation of NUMA will be a follow-on to the existing AViiON 88K-based 16-way system. Using a single board complex which incorporates processors, memory, and I/O (to reduce system bus/backplane traffic), this system requires NUMA since memory can be either on the same board complex as the processor (near memory) or on a different board complex (far memory). This will allow effective scaling of the AViiON 88K family up to 32 processors.

Data General will also use NUMA for high-end AViiON Intel-based systems in the future. When system performance requirements exceed those which can be met by "off-the-shelf" SMP in SHVs (initially quad-processor P6), an external coherent memory interconnect will be used to combine multiple SHVs into a single shared memory system with a NUMA architecture. A high-level logical block diagram of such a system is shown at the bottom of Figure 4. The SHVs shown in the figure are basically full quad-processor P6 systems with the addition of interconnect logic and support hardware. Physically, the system looks like multiple rackmounted P6 chassis interconnected by an external bus.

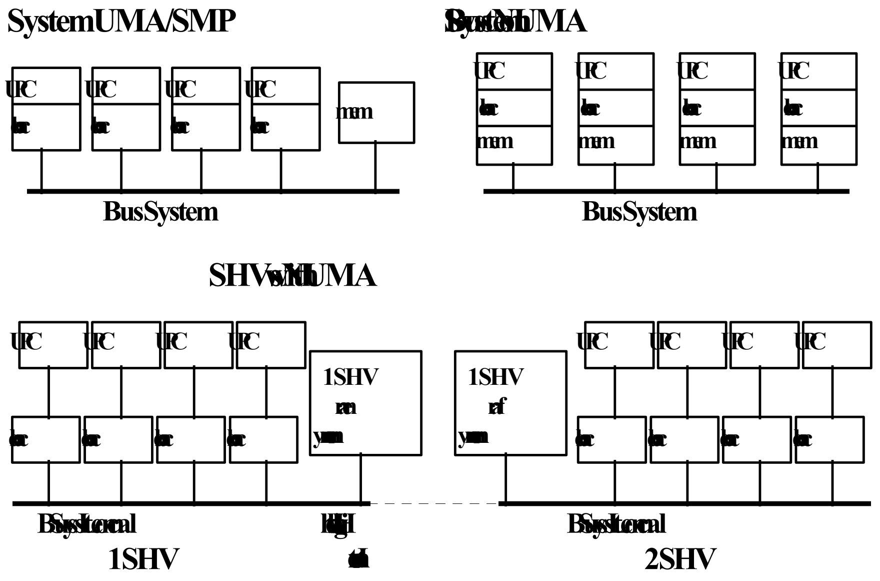

Figure 4. Block diagram for interconnected SHVs in a NUMA architecture.

Figure 4 also points out an interesting observation about NUMA performance potential. NUMA systems at first may be thought slower than SMP systems since, after all, they have "Non-Uniform Memory Access" rather than the "Uniform Memory Access" of SMP machines. In an SMP system, **all** memory is connected to CPUs over a high-bandwidth interconnect - the system bus. However, in an SMP system, all memory is far; there is no near memory. For example, all memory access on an AV 9500 is done over the system bus (i.e., all far memory in NUMA terms). Now, of course, this says nothing about the relative performance of specific SMP and NUMA architectures which use different interconnect technologies, CPU cache designs, etc., but it does say that there is nothing in the fundamental architecture to keep NUMA from delivering very high-end performance.

Compared to the alternatives, this general approach offers a variety of benefits to both Data General and our customers by **providing the hardware economies of MPP with the software economies of SMP**. Leveraging commodity processor components (SHVs) from the PC server space, eliminates the need for the sort of expensive custom components required by large SMP designs. These components, such as large monolithic backplanes and maximum performance memory designs, result in longer development cycles and can greatly increase system cost. Importantly, these savings do **not** come at the expense of software complexity for the end user or developer. With the exception of the operating system itself (and tuning of some platform software such as databases) applications have no need to be "NUMA-aware." The programming model is no different from that of SMP systems for which software is already available.

These future NUMA architecture systems based on SHVs can be expected to offer significant price/performance and high-availability advantages relative to traditional SMP products. It will be technically feasible to cover this performance space using SMP systems designed with custom high-speed subsystems, but the cost of such subsystems will be high. In the longer term, increasing processor speeds will render traditional interconnect technologies impractical, not only due to higher development costs and design complexity (with its time to market and reliability implications), but also because there are practical upper limits to backplane throughput imposed by cycle times and noise.

Data General's NUMA Software Directions

As stated earlier, applications and users are not expected to be affected by the NUMA architecture. **Software developed for SMP systems (or even, in many cases, uniprocessor systems) can be expected to run on NUMA systems without changes.** "Knowledge" of NUMA is largely localized at the operating system level.

DG/UX, however, will take steps to fully exploit the scalable hardware provided by SHVs interconnected into a NUMA system. These NUMA adaptations represent an evolution of DG/UX techniques and capabilities such as the hierarchical abstraction of system resources, developed to take advantage of current systems with large caches as well as distributed memory/bus capability (e.g., the quad-processor AV 9500 Plus board described previously).

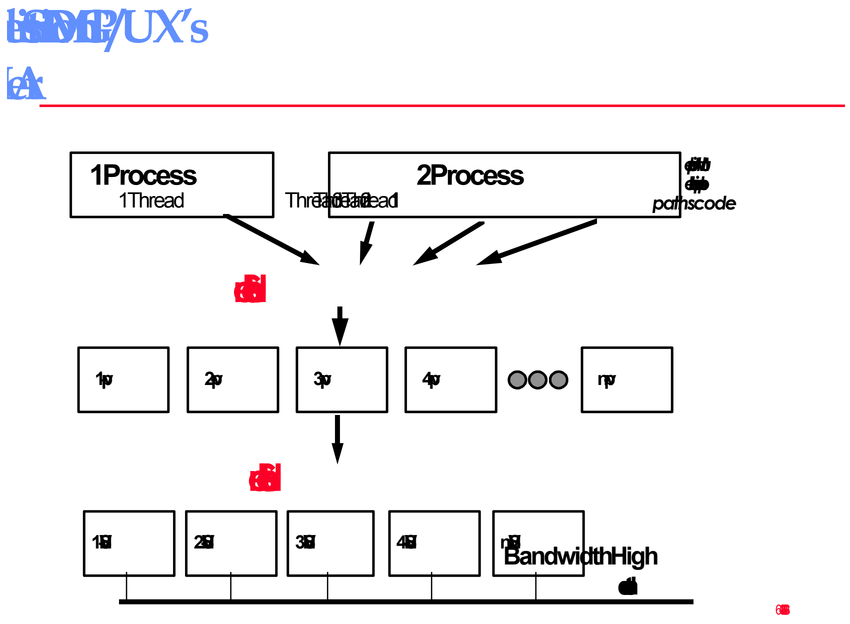

Figure 5. DG/UX's SMP techniques carry over to the NUMA architecture.

As shown in Figure 5, scaling techniques already implemented in DG/UX to support SMP carry over to the NUMA architecture by simply substituting SHV components for individual processors. These techniques include fine-grained locking, exploitation of localities wherever possible, and multi-threaded

- **Processor Affinity**
  - Objective: Service memory references from cache or “near” memory rather than “far” memory. Optimize processor efficiency and bus loading
  - Associate thread groups on their asigned processors whenever possible
  - Schedule thread groups on their attached processors whenever possible
  - Enhance policies to recognize the cost of migrating tasks across the global interconnect
- **Memory Affinity**
  - Same Objectives as for Processor Affinity
  - Associate processes with memory resources
  - Duplicate kernel code in near memory and allocate kernel data locally as well
  - Migrate key private pages with processes
  - Allocate physical memory frames local to requesting process
- **Module Load Balancing**
  - Objective: Avoid local resource bottlenecks
  - Distribute application load across available memory, bus, and CPU resources
  - Redesign key memory allocator data structures and algorithms based on the hierarchical system model
- **Configurable NUMA policies**
  - Provide tuning options for optimizing at cache/memory/bus hierarchy levels
- **Other optimizations**
  - Tie user memory affinity primarily to processes rather than threads
  - Kernel memory pools
  - Frame management
  - Hierarchical placement policies

throughput. Other specific scaling techniques (processor affinity, memory affinity, module load balancing, and configurable NUMA policies) are discussed in the chart below.

-

IV. NUMA Benefits

The previous sections have touched on a number of the benefits that systems based on SHV servers and the NUMA architecture can offer. In this section, these concepts are expanded with particular emphasis on the benefits which users see directly. These benefits fall primarily into five classes:

- Price/Performance
- High availability
- Field upgradability
- Low-effort software development
- Scalable performance

Price/Performance

As described in some detail earlier, NUMA can provide an architectural framework for large shared memory multiprocessors built around either lower cost commodity components or traditional SMP systems of the same performance level. Over time, the most important of these components become high-performance SHVs processing blocks. These SHVs can then be interconnected into a NUMA system at a much lower cost than using a high-end custom backplane as an SMP system would require. Since "NUMA-aware" operating systems and platform software can minimize the performance delta between SMP and NUMA implementations with the same number of processors (and the cost can be significantly lower in the NUMA implementation), the result is better price/performance in the NUMA system.

High Availability

A NUMA system consisting of SHVs using an appropriate interconnection is inherently modular and therefore minimizes single points of failure and allows for the easy replacement of failed components. In addition to whatever high availability features may be built into individual component SHV modules, this architectural model also allows for the deconfiguration and replacement of entire modules. The appropriate distribution and failover configuration of I/O among modules can minimize downtime associated with any sort of failure within a module. The net result is a scenario which can look very much like the military model of "black box" module repair upon failure – repairs are made quickly with short out-of-service times, and actual repairs are made off-line.

Field Upgradability

The same modular architecture which provides high availability also allows for easy field upgradability. The simple addition of standard modules can be used to either upgrade performance or to increase the level of redundancy in a system. Since existing modules are undisturbed by this sort of modular upgrade process, downtime in minimized. The only changes required to the existing system are external and relate to adding the new module to the existing interconnect.

Low Effort Software Development

NUMA architectures allow customers to remain close to the hardware-independent, uniprocessor model for applications development which they prefer. NUMA avoids the mandatory use of proprietary, vendor-specific extensions to languages and database development systems which are typically needed with MPP systems, as well as manual load balancing and partitioning across system elements.

Scalable Performance

Most of the discussion of NUMA architectures in this White Paper has focused on the relatively near- to mid-term future of NUMA as an alternative to traditional SMP designs and to MPP systems. In this timeframe, NUMA holds significant promise as a way to offer better hardware price/performance than larger SMP systems while preserving their simple programming model. MPP, while perhaps offering similar price/performance benefits for certain application classes, does so only at the cost of considerable software complexity.

... and the Future

In the longer term, however, NUMA systems have the potential to exceed the top performance levels that traditional SMP systems can achieve. NUMA allows the building of cost-effective systems with hundreds of CPUs. The high-end performance of NUMA systems is not limited by the problems of implementing extremely high speed system buses and memory subsystems, implementations which are constrained not only by cost but also by absolute physical limits. Rather the scalability barriers of NUMA are imposed more by software scalability and total application needs.

The availability of high-end scalable architectures such as NUMA built around SHVs will allow for new classes of applications which are not constrained by available processing power. Some of these applications will grow around the increasing centralization of key interdepartmental corporate data into data warehouses. Others will grow from the spreading use of new types of data such as video. All require CPU power and corresponding I/O capabilities which easily exceed achieved system capacities today.