Karen L. Noel
Nitin Y. Karkhanis

OpenVMS Alpha 64-bit Very Large Memory Design

The OpenVMS Alpha version 7.1 operating system provides memory management features that extend the 64-bit VLM capabilities introduced in version 7.0. The new OpenVMS Alpha APIs and mechanisms allow 64-bit VLM applications to map and access very large shared memory objects (global sections). Design areas include shared memory objects without disk file backing storage (memory-resident global sections), shared page tables, and a new physical memory and system fluid page reservation system.

Database products and other applications impose heavy demands on physical memory. The newest version of DIGITAL's OpenVMS Alpha operating system extends its very large memory (VLM) support and allows large caches to remain memory resident. OpenVMS Alpha version 7.1 enables applications to take advantage of both 64-bit virtual addressing and very large memories consistent with the OpenVMS shared memory model. In this paper, we describe the new 64-bit VLM capabilities designed for the OpenVMS Alpha version 7.1 operating system. We explain application flexibility and the system management issues addressed in the design and discuss the performance improvements realized by 64-bit VLM applications.

Overview

A VLM system is a computer with more than 4 gigabytes (GB) of main memory. A flat, 64-bit address space is commonly used by VLM applications to address more than 4 GB of data.

A VLM system allows large amounts of data to remain resident in main memory, thereby reducing the time required to access that data. For example, database cache designers implement large-scale caches on VLM systems in an effort to improve the access times for database records. Similarly, VLM database applications support more server processes than ever before. The combination of large, in-memory caches and an increased number of server processes significantly reduces the overall time database clients wait to receive the data requested.[1]

The OpenVMS Alpha version 7.0 operating system took the first steps in accommodating the virtual address space requirements of VLM applications by introducing 64-bit virtual addressing support. Prior to version 7.0, large applications—as well as the OpenVMS operating system itself—were becoming constrained by the limits imposed by a 32-bit address space.

Although version 7.0 eased address space restrictions, the existing OpenVMS physical memory management model did not scale well enough to accommodate VLM systems. OpenVMS imposes specific limits on the amount of physical memory a process can occupy. As a result, applications lacked the ability to keep a very large object in physical memory. In systems on which the physical memory is not plentiful, the mechanisms that limit per-process memory utilization serve to ensure fair-and-equal access to a potentially scarce resource. However, on systems rich with memory whose intent is to service applications creating VLM objects, the limitations placed on per-process memory utilization inhibit the overall performance of those applications. As a result, the benefits of a VLM system may not be completely realized.

Applications that require very large amounts of physical memory need additional VLM support. The goals of the OpenVMS Alpha VLM project were the following:

- Maximize the operating system's 64-bit capabilities
- Take full advantage of the Alpha Architecture
- Not require excessive application change
- Simplify the system management of a VLM system
- Allow for the creation of VLM objects that exhibit the same basic characteristics, from the programmer's perspective, as other virtual memory objects created with the OpenVMS system service programming interface

These goals became the foundation for the following VLM technology implemented in the OpenVMS Alpha version 7.1 operating system:

- Memory-resident global sections—shared memory objects that do not page to disk
- Shared page tables—page tables mapped by multiple processes, which in turn map to memory-resident global sections
- The reserved memory registry—a memory reservation system that supports memory-resident global sections and shared page tables

The remainder of this paper describes the major design areas of VLM support for OpenVMS and discusses the problems addressed by the design team, the alternatives considered, and the benefits of the extended VLM support in OpenVMS Alpha version 7.1.

Memory-resident Global Sections

We designed memory-resident global sections to resolve the scaling problems experienced by VLM applications on OpenVMS. We focused our design on the existing shared memory model, using the 64-bit addressing support. Our project goals included simplifying system management and harnessing the speed of the Alpha microprocessor. Before describing memory-resident global sections, we provide a brief explanation of shared memory, process working sets, and a page fault handler.

Global Sections

An OpenVMS global section is a shared memory object. The memory within the global section is shared among different processes in the system. Once a process has created a global section, others may map to the section to share the data. Several types of global sections can be created and mapped by calling OpenVMS system services.

Global Section Data Structures Internally, a global section consists of several basic data structures that are stored in system address space and are accessible to all processes from kernel mode. When a global section is created, OpenVMS allocates and initializes a set of these data structures. The relationship between the structures is illustrated in Figure 1. The sample global section is named "SHROBJ" and is 2,048 Alpha pages or 16 megabytes (MB) in size. Two processes have mapped to the global section by referring to the global

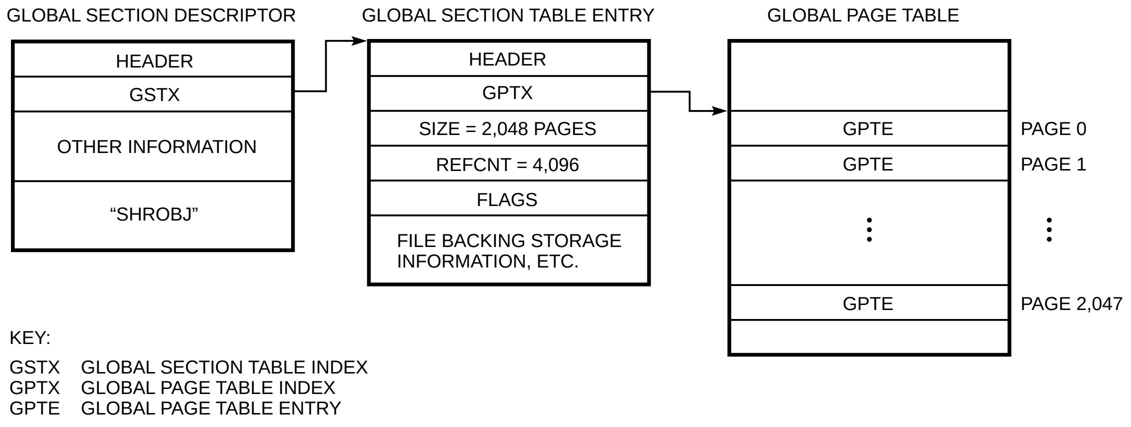

GLOBAL SECTION DESCRIPTOR
HEADER
GSTX
OTHER INFORMATION
"SHROBJ"
GLOBAL SECTION TABLE ENTRY
HEADER
GPTX
SIZE = 2,048 PAGES
REFCNT = 4,096
FLAGS
FILE BACKING STORAGE INFORMATION, ETC.
GLOBAL PAGE TABLE
GPTE
GPTE
GPTE
PAGE 0
PAGE 1
PAGE 2,047
KEY:
GSTX GLOBAL SECTION TABLE INDEX
GPTX GLOBAL PAGE TABLE INDEX
GPTE GLOBAL PAGE TABLE ENTRY

Figure 1
Global Section Data Structures

section data structures in their process page table entries (PTEs).

Process PTEs Mapping to Global Sections When a process maps to a global section, its process PTEs refer to global section pages in a one-to-one fashion. A page of physical memory is allocated when a process accesses a global section page for the first time. This results in both the process PTE and the global section page becoming valid. The page frame number (PFN) of the physical page allocated is stored in the process PTE. Figure 2 illustrates two processes that have mapped to the global section where the first process has accessed the first page of the global section.

When the second process accesses the same page as the first process, the same global section page is read from the global section data structures and stored in the process PTE of the second process. Thus the two processes map to the same physical page of memory.

The operating system supports two types of global sections: a global section whose original contents are zero or a global section whose original contents are read from a file. The zeroed page option is referred to as demand zero.

Backing Storage for Global Sections Global section pages require backing storage on disk so that more frequently referenced code or data pages can occupy physical memory. The paging of least recently used pages is typical of a virtual memory system. The backing storage for a global section can be the system page files, a file opened by OpenVMS, or a file opened by the application. A global section backed by system page files is referred to as a page-file-backed global section. A global section backed by a specified file is referred to as a file-backed global section.

When a global section page is invalid in all process PTEs, the page is eligible to be written to an on-disk backing storage file. The physical page may remain in memory on a list of modified or free pages. OpenVMS algorithms and system dynamics, however, determine which page is written to disk.

Process Working Sets

On OpenVMS, a process' valid memory is tracked within its working set lists. The working set of a process reflects the amount of physical memory a process is consuming at one particular point in time. Each valid working set list entry represents one page of virtual memory whose corresponding process PTE is valid. A process' working set list includes global section pages, process private section pages, process private code pages, stack pages, and page table pages.

A process' working set quota is limited to 512 MB and sets the upper limit on the number of pages that can be swapped to disk. The limit on working set quota matches the size of a swap I/O request.[2] The effects on swapping would have to be examined to increase working set quotas above 512 MB.

Process working set lists are kept in 32-bit system address space. When free memory is plentiful in the system, process working set lists can increase to an extended quota specified in the system's account file for the user. The system parameter, WSMAX, specifies the maximum size to which a process working set can be extended. OpenVMS specifies an absolute maximum value of 4 GB for the WSMAX system parameter. An inverse relationship exists between the size specified for WSMAX and the number of resident processes OpenVMS can support, since both are maintained in the 32-bit addressable portion of system space. For example, specifying the maximum value for WSMAX sharply decreases the number of resident processes that can be specified.

Should OpenVMS be required to support larger working sets in the future, the working set lists would have to be moved out of 32-bit system space.

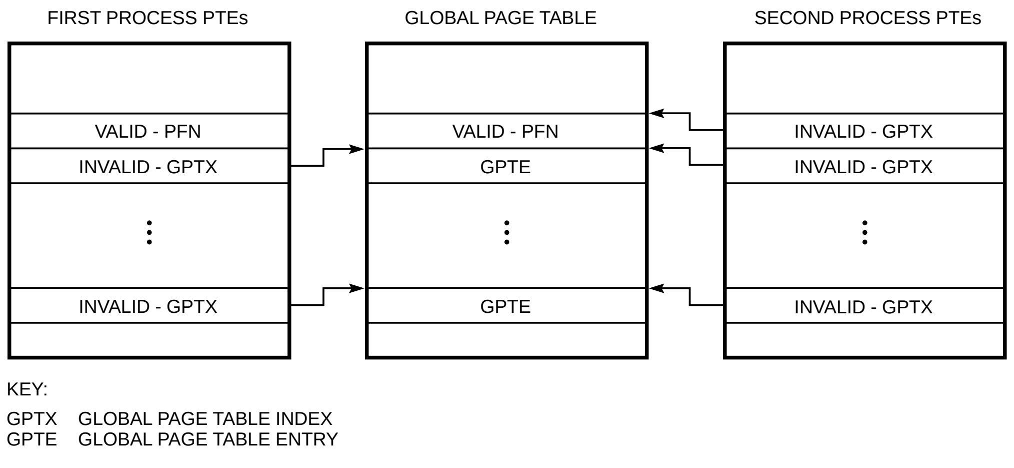


Figure 2
Process and Global PTEs

Page Fault Handling for Global Section Pages

The data within a global section may be heavily accessed by the many processes that are sharing the data. Therefore, the access time to the global section pages may influence the overall performance of the application.

Many hardware and software factors can influence the speed at which a page within a global section is accessed by a process. The factors relevant to this discussion are the following:

- Is the process PTE valid or invalid?
- If the process PTE is invalid, is the global section page valid or invalid?
- If the global section page is invalid, is the page on the modified list, free page list, or on disk within the backing storage file?

If the process PTE is invalid at the time the page is accessed, a translation invalid fault, or page fault, is generated by the hardware. The OpenVMS page fault handler determines the steps necessary to make the process PTE valid.

If the global section page is valid, the PFN of the data is read from the global section data structures. This is called a global valid fault. This type of fault is corrected quickly because the data that handles this fault is readily accessible from the data structures in memory.

If the global section page is invalid, the data may still be within a physical page on the modified or free page list maintained by OpenVMS. To correct this type of fault, the PFN that holds the data must be removed from the modified or free page list, and the global section page must be made valid. Then the fault can be handled as if it were a global valid fault.

If the page is on disk within the backing storage file, an I/O operation must be performed to read the data from the disk into memory before the global section page and process PTE can be made valid. This is the slowest type of global page fault, because performing a read I/O operation is much slower than manipulating data structures in memory.

For an application to experience the most efficient access to its shared pages, its process PTEs should be kept valid. An application may use system services to lock pages in the working set or in memory, but typically the approach taken by applications to reduce page fault overhead is to increase the user account's working set quota. This approach does not work when the size of the global section data exceeds the size of the working set quota limit of 512 MB.

Database Caches as File-backed Global Sections

Quick access to a database application's shared memory is critical for an application to handle transactions quickly.

Global sections implement shared memory on OpenVMS, so that many database processes can share the cached database records. Since global sections must have backing storage on disk, database caches are either backed by the system's page files or by a file created by the database application.

For best performance, the database application should keep all its global section pages valid in the process PTEs to avoid page fault and I/O overhead. Database processes write modified buffers from the cache to the database files on an as-needed basis. Therefore, the backing storage file required by OpenVMS is redundant storage.

Very Large Global Sections

The OpenVMS VLM project focused on VLM database cache design. An additional goal was to design the VLM features so that other types of VLM applications could benefit as well.

Consider a database cache that is 6 GB in size. Global sections of this magnitude are supported on OpenVMS Alpha with 64-bit addressing support. If the system page files are not used, the application must create and open a 6-GB file to be used as backing storage for the global section.

With the maximum quota of 512 MB for a process working set and with the maximum of a 4-GB working set size, no process could keep the entire 6-GB database cache valid in its working set at once. When an OpenVMS global section is used to implement the database cache, page faults are inevitable. Page fault activity severely impacts the performance of the VLM database cache by causing unnecessary I/O to and from the disk while managing these pages.

Since all global sections are pageable, a 6-GB file needs to be created for backing storage purposes. In the ideal case, the backing storage file is never used. The backing storage file is actually redundant with the database files themselves.

VLM Design Areas

The VLM design team targeted very large global sections (4 GB or larger) to share data among many processes. Furthermore, we assumed that the global section's contents would consist of zeroed memory instead of originating from a file. The team explored whether this focus was too narrow. We were concerned that implementing just one type of VLM global section would preclude support for certain types of VLM applications.

We considered that VLM applications might use very large amounts of memory whose contents originate from a data file. One type of read-only data from a file contains program instructions (or code). Code sections are currently not pushing the limits of 32-bit address space. Another type of read-only data from a file contains scientific data to be analyzed by the VLM application. To accommodate very large read-only data of this type, a large zeroed global section can be created, the data from the file can be read into memory, and then the data can be processed in memory.

If writable pages are initially read from a file instead of zeroed, the data will most likely need to be written back to the original file. In this case, the file can be used as the backing storage for the data. This type of VLM global section is supported on OpenVMS Alpha as a file-backed global section. The operating system's algorithm for working set page replacement keeps the most recently accessed pages in memory. Working set quotas greater than 512 MB and working set sizes greater than 4 GB help this type of VLM application scale to higher memory sizes.

We also considered very large demand-zero private pages, "malloc" or "heap" memory. The system page files are the backing storage for demand-zero private pages. Currently, processes can have a page file quota as large as 32 GB. A VLM application, however, may not want these private data pages to be written to a page file since the pages are used in a similar fashion as in-memory caches. Larger working set quotas also help this type of VLM application accommodate ever-increasing memory sizes.

Backing Storage Issues

For many years, database cache designers and database performance experts had requested that the OpenVMS operating system support memory with no backing storage files. The backing storage was not only redundant but also wasteful of disk space. The waste issue is made worse as the sizes of the database caches approach the 4-GB range. As a result, the OpenVMS Alpha VLM design had to allow for non-file-backed global sections.

The support of 64-bit addressing and VLM has always been viewed as a two-phased approach, so that functionality could be delivered in a timely fashion.[3] OpenVMS Alpha version 7.0 provided the essentials of 64-bit addressing support. The VLM support was viewed as an extension to the memory management model and was deferred to OpenVMS Alpha version 7.1.

Working Set List Issues. Entries in the process working set list are not required for pages that can never be written to a backing storage file. The fundamental concept of the OpenVMS working set algorithms is to support the paging of data from memory to disk and back into memory when it is needed again. Since the focus of the VLM design was on memory that would not be backed by disk storage, the VLM design team realized that these pages, although valid in the process PTEs, did not need to be in the process' working set list.

VLM Programming Interface

The OpenVMS Alpha VLM design provides a new programming interface for VLM applications to create, map to, and delete demand-zero, memory-resident global sections. The existing programming interfaces did not easily accommodate the new VLM features.

To justify a new programming interface, we looked at the applications that would be calling the new system service routines. To address more than 4 GB of memory in the flat OpenVMS 64-bit address space, a 32-bit application must be recompiled to use 64-bit pointers and often requires source code changes as well. Database applications were already modifying their source code to use 64-bit pointers and to scale their algorithms to handle VLM systems.[1] Therefore, calling a new set of system service routines was considered acceptable to the programmers of VLM applications.

Options for Memory-resident Global Sections

To initialize a very large memory-resident global section mapped by several processes, the overhead of hardware faults, allocating zeroed pages, setting process PTEs valid, and setting global section pages valid is eliminated by preallocating the physical pages for the memory-resident global section. Preallocation is performed by the reserved memory registry, and is discussed later in this paper. Here we talk about options for how the reserved memory is used.

Two options, ALLOC and FLUID, are available for creating a demand-zero, memory-resident global section.

ALLOC Option The ALLOC option uses preallocated, zeroed pages of memory for the global section. When the ALLOC option is used, pages are set aside during system start-up specifically for the memory-resident global section. Preallocation of contiguous groups of pages is discussed in the section Reserving Memory during System Start-up. Preallocated memory-resident global sections are faster to initialize than memory-resident global sections that use the FLUID option.

Run-time performance is improved by using the Alpha Architecture's granularity hint, a mechanism we discuss later in this paper. To use the ALLOC option, the system must be rebooted for large ranges of physically contiguous memory to be allocated.

FLUID Option The FLUID option allows pages not yet accessed within the global section to remain fluid within the system. This is also referred to as the fault option because the page fault algorithm is used to allocate the pages. When the FLUID (or fault) option is used, processes or the system can use the physical pages until they are accessed within the memory-resident global section. The pages remain within the system's fluid memory until they are needed. This type of memory-resident global section is more flexible than one that uses the ALLOC option. If an application that uses a memory-resident global section is run on a system that cannot be rebooted due to system availability concerns, it can still use the FLUID option. The system will not allow this application to run unless enough pages of memory are available in the system for the memory-resident global section.

The system service internals code checks the reserved memory registry to determine the range of pages preallocated for the memory-resident global section or to determine if the FLUID option will be used. Therefore the decision to use the ALLOC or the FLUID option is not made within the system services routine interface. The system manager can determine which option is used by specifying preferences in the reserved memory registry. An application can be switched from using the ALLOC option to using the FLUID option without requiring a system reboot.

Design Internals

The internals of the design choices underscore the modularity of the shared memory model using global sections. A new global section type was easily added to the OpenVMS system. Those aspects of memory-resident global sections that are identical to pageable global sections required no code modifications to support.

To support memory-resident global sections, the MRES and ALLOC flags were added to the existing global section data structures. The MRES flag indicates that the global section is memory resident, and the ALLOC flag indicates that contiguous pages were preallocated for the global section.

The file-backing storage information within global section data structures is set to zero for memory-resident global sections to indicate that no backing storage file is used. Other than the new flags and the lack of backing storage file information, a demand-zero, memory-resident global section looks to OpenVMS Alpha memory management like a demand-zero, file-backed global section. Figure 3 shows the updates to the global section data structures.

One important difference with memory-resident global sections is that once a global section page becomes valid, it remains valid for the life of the global section. Global section pages by definition can never become invalid for a memory-resident global section.

When a process maps to a memory-resident global section, the process PTE can be either valid for the ALLOC option or invalid for the FLUID option. When the ALLOC option is used, no page faulting occurs for the global section pages.

When a process first accesses an invalid memory-resident global section page, a page fault occurs just as with traditional file-backed global sections. Because the same data structures are present, the page fault code initially executes the code for a demand-zero, file-backed global section page. A zeroed page is allocated and placed in the global section data structures, and the process PTE is set valid.

The working set list manipulation steps are skipped when the MRES flag is encountered in the global section data structures. Because these global section pages are not placed in the process working set list, they are not considered in its page-replacement algorithm. As such, the OpenVMS Alpha working set manipulation code paths remained unchanged.

System Management and Memory-resident Global Sections

When a memory-resident global section is used instead of a traditional, pageable global section for a database cache, there is no longer any wasted page file storage required by OpenVMS to back up the global section.

The other system management issue alleviated by the implementation of memory-resident global sections concerns working set sizes and quotas. When a file-backed global section is used for the database cache, the database processes require elevated working

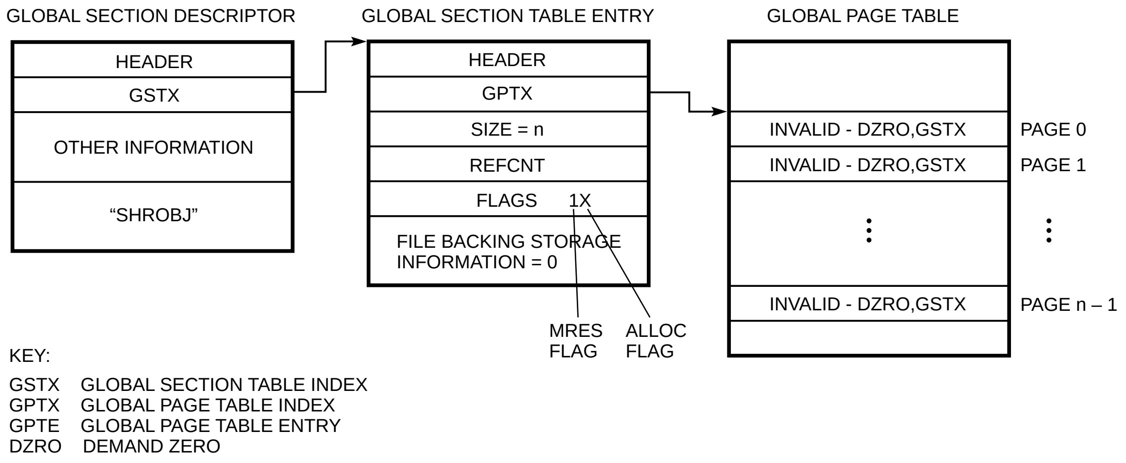


Figure 3
Memory-resident Global Section Data Structures

set quotas to accommodate the size of the database cache. This is no longer a concern because memory-resident global section pages are not placed into the process working set list.

With the use of memory-resident global sections, system managers may reduce the value for the WSMAX system parameter such that more processes can remain resident within the system. Recall that a process working set list is in 32-bit system address space, which is limited to 2 GB.

Shared Page Tables

VLM applications typically consume large amounts of physical memory in an attempt to minimize disk I/O and enhance overall application performance. As the physical memory requirements of VLM applications increase, the following second-order effects are observed due to the overhead of mapping to very large global sections:

- Noticeably long application start-up and shut-down times
- Additional need for physical memory as the number of concurrent sharers of a large global section increases
- Unanticipated exhaustion of the working set quota and page file quota
- A reduction in the number of processes resident in memory, resulting in increased process swapping

The first two effects are related to page table mapping overhead and size. The second two effects, as they relate to page table quota accounting, were also resolved by a shared page tables implementation. The following sections address the first two issues since they uniquely pertain to the page table overhead.

Application Start-up and Shut-down Times

Users of VLM applications can observe long application start-up and shut-down times as a result of creating and deleting very large amounts of virtual memory. A single process mapping to a very large virtual memory object does not impact overall system performance. However, a great number of processes that simultaneously map to a very large virtual memory object have a noticeable impact on the system's responsiveness. The primary cause of the performance degradation is the accelerated contention for internal operating system locks. This observation has been witnessed on OpenVMS systems and on DIGITAL UNIX systems (prior to the addition of VLM support.)

On OpenVMS, the memory management spinlock (a synchronization mechanism) serializes access to privileged, memory-management data structures. We have observed increased spinlock contention as the result of hundreds of processes simultaneously mapping to large global sections. Similar lock contention and system unresponsiveness occur when multiple processes attempt to delete their address space simultaneously.

Additional Need for Physical Memory

For pages of virtual memory to be valid and resident, the page table pages that map the data pages must also be valid and resident. If the page table pages are not in memory, successful address translation cannot occur.

Consider an 8-GB, memory-resident global section on an OpenVMS Alpha system (with an 8-kilobyte page size and 8-byte PTE size). Each process that maps the entire 8-GB, memory-resident global section requires 8 MB for the associated page table structures. If 100 processes are mapping the memory-resident global section, an additional 800 MB of physical memory must be available to accommodate all processes' page table structures. This further requires that working set list sizes, process page file quotas, and system page files be large enough to accommodate the page tables.

When 100 processes are mapping to the same memory-resident global section, the same PTE data is replicated into the page tables of the 100 processes. If each process could share the page table data, only 8 MB of physical memory would be required to map an 8-GB, memory-resident global section; 792 MB of physical memory would be available for other system purposes.

Figure 4 shows the amount of memory used for process page tables mapping global sections ranging in size from 2 to 8 GB. Note that as the number of processes that map an 8-GB global section exceeds

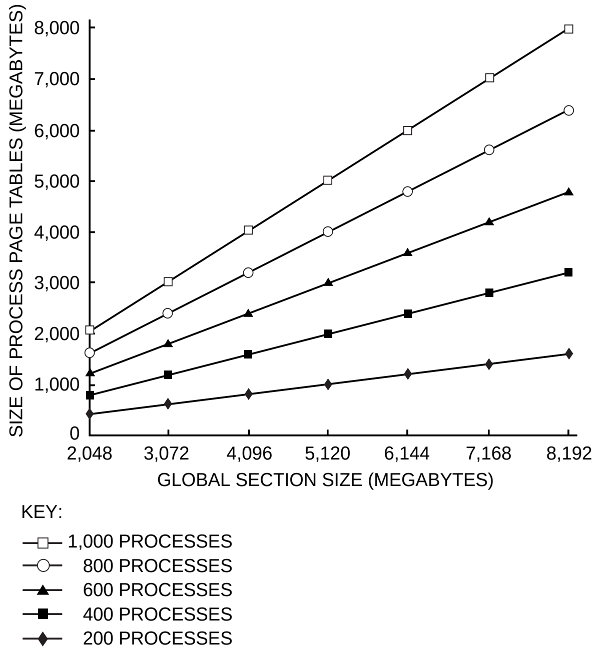


Figure 4
Process Page Table Sizes

1,000, the amount of memory used by process page tables is larger than the global section itself.

Shared Memory Models

We sought a solution to sharing process page tables that would alleviate the performance problems and memory utilization overhead yet stay within the shared memory framework provided by the operating system and the architecture. Two shared memory models are implemented on OpenVMS, shared system address space and global sections.

The OpenVMS operating system supports an address space layout that includes a shared system address space, page table space, and private process address space. Shared system address space is created by placing the physical address of the shared system space page tables into every process' top-level page table. Thus, every process has the same lower-level page tables in its virtual-to-physical address translation path. In turn, the same operating system code and data are found in all processes' address spaces at the same virtual address ranges. A similar means could be used to create a shared page table space that is used to map one or more memory-resident global sections.

An alternative for sharing the page tables is to create a global section that describes the page table structure. The operating system could maintain the association between the memory-resident global section and the global section for its shared page table pages. The shared page table global section could be mapped at the upper levels of the table structure such that each process that maps to it has the same lower-level page tables in its virtual-to-physical address translation path. This in turn would cause the data to be mapped by all the processes.

Figure 5 provides a conceptual representation of the shared memory model. Figure 6 extends the shared memory model by demonstrating that the page tables become a part of the shared memory object.

The benefits and drawbacks of both sharing models are highlighted in Table 1 and Table 2.

Model Chosen for Sharing Page Tables

After examining the existing memory-sharing models on OpenVMS and taking careful note of the composition and characteristics of shared page tables, the design team chose to implement shared page tables as a global section. In addition to the benefits listed in Table 2, the

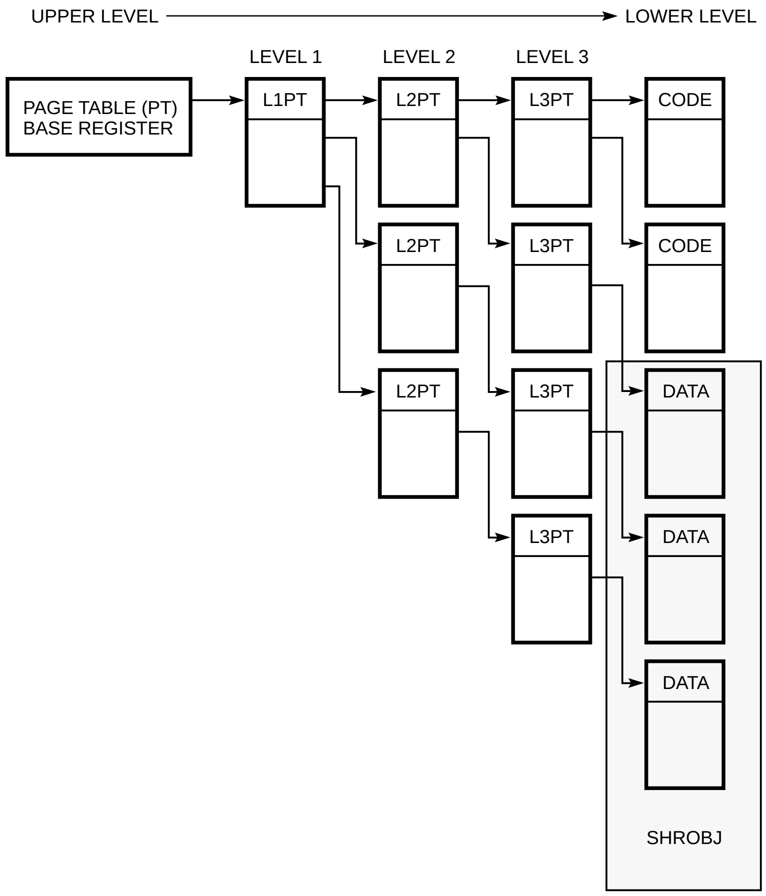


Figure 5
Shared Memory Object

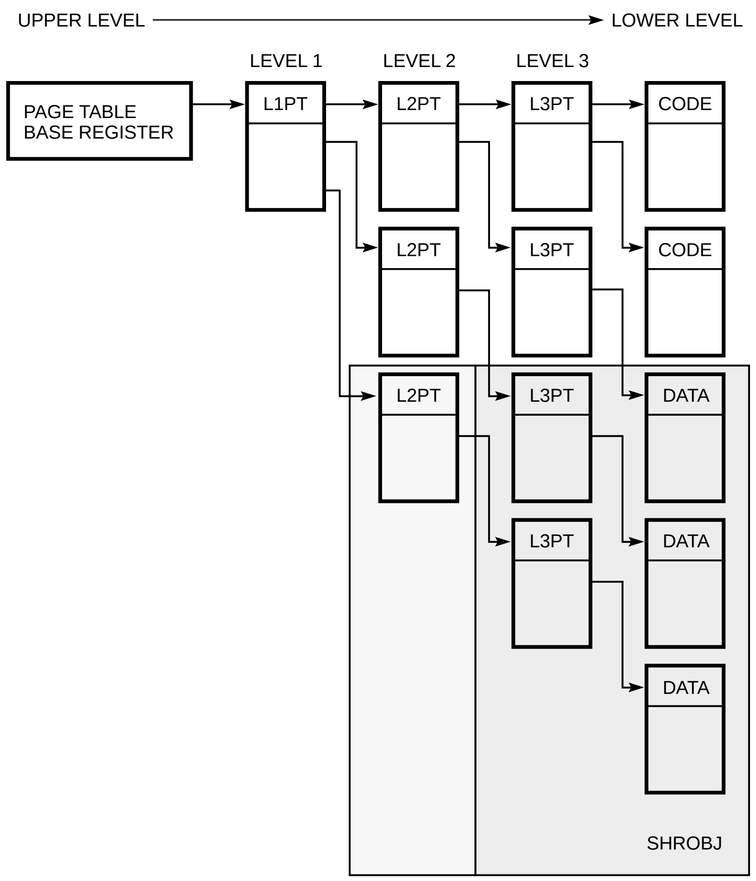


Figure 6
Shared Memory Objects Using Shared Page Tables

Table 1
Shared Page Table Space—Benefits and Drawbacks

Benefits	Drawbacks
Shared page table space begins at the same virtual address for all processes.	The virtual address space is reserved for every process. Processes not using shared page tables are penalized by a loss in available address space. Shared page table space is at least 8 GB in size, regardless of whether the entire space is used. A significant amount of new code would need to be added to the kernel since shared system space is managed separately from process address space.

Table 2
Global Sections for Page Tables—Benefits and Drawbacks

Benefits	Drawbacks
The same virtual addresses can be used by all processes, but this is not required. The amount of virtual address space mapped by shared page tables is determined by application need. Shared page tables are available only to those processes that need them. Shared page tables allow for significant reuse of existing global section data structures and process address space management code.	Shared page tables are mapped at different virtual addresses per process unless additional steps are taken.

design team noticed that shared page table pages bear great resemblance to the memory-resident pages they map. Specifically, for a data or code page to be valid and resident, its page table page must also be valid and resident. The ability to reuse a significant amount of the global section management code reduced the debugging and testing phases of the project.

In the initial implementation, shared page table global sections map to memory-resident global sections only. This decision was made because the design focused on the demands of VLM applications that use memory-resident global sections. Should significant demand exist, the implementation can be expanded to allow the mapping of pageable global sections.

Shared page tables can never map process private data. The design team had to ensure that the shared page table implementation kept process private data from entering a virtual address range mapped by a shared page table page. If this were to happen, it would compromise the security of data access between processes.

Shared Page Tables Design

The goals for the design of shared page tables included the following:

- Reduce the time required for multiple users to map the same memory-resident global section
- Reduce the physical memory cost of maintaining private page tables for multiple mappers of the same memory-resident global section
- Do not require the use of a backing storage file for shared page table pages
- Eliminate the working set list accounting for these page table pages
- Implement a design that allows upper levels of the page table hierarchy to be shared at a later time

Figure 6 demonstrates the shared page table global section model. The dark gray portion of the figure highlights the level of sharing supplied in OpenVMS Alpha version 7.1. The light gray portion highlights possible levels of sharing allowed by creating a shared page table global section consisting of upper-level page table pages.

Modifications to Global Section Data Structure Table 2 noted as a benefit the ability to reuse existing data structures and code. Minor modifications were exacted to the global section data structures so that they could be used to represent a shared page table global section. A new flag, SHARED_PTS, was added to the global section data structures. Coupled with this change was the requirement that a memory-resident global section and its shared page table global section be uniquely linked together. The correspondence between the two sets of global sections is managed by the operating system and is used to locate the data structures for one global section when the structures for the other global section are in hand. Figure 7 highlights the changes made to the data structures.

Creating Shared Page Tables To create a memory-resident global section, an application calls a system service routine. No flags or extra arguments are required to enable the creation of an associated shared page table global section.

The design team also provided a means to disable the creation of the shared page tables in the event that a user might find shared page tables to be undesirable. To disable the creation of shared page tables, the reserved memory registry entry associated with the memory-resident global section can specify that page tables are not to be used. Within the system service routine that creates a memory-resident global section,

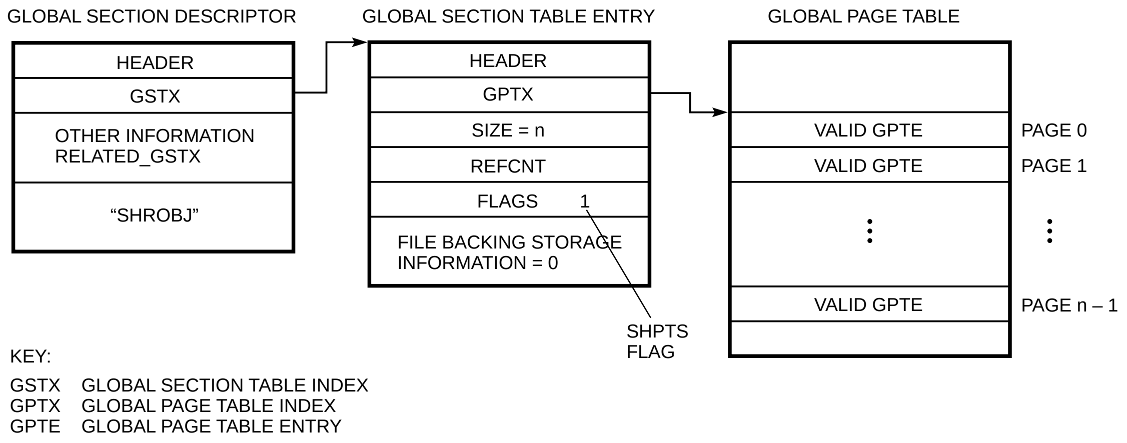

GLOBAL SECTION DESCRIPTOR
HEADER
GSTX
OTHER INFORMATION
RELATED_GSTX
"SHROBJ"
GLOBAL SECTION TABLE ENTRY
HEADER
GPTX
SIZE = n
REFCNT
FLAGS 1
FILE BACKING STORAGE INFORMATION = 0
SHPTS FLAG
GLOBAL PAGE TABLE
VALID GPTE
PAGE 0
VALID GPTE
PAGE 1
VALID GPTE
PAGE n – 1
KEY:
GSTX GLOBAL SECTION TABLE INDEX
GPTX GLOBAL PAGE TABLE INDEX
GPTE GLOBAL PAGE TABLE ENTRY

Figure 7
Data Structure Modifications

the reserved memory registry is examined for an entry associated with the named global section. If an entry exists and it specifies shared page tables, shared page tables are created. If the entry does not specify shared page tables, shared page tables are not created.

If no entry exists for the global section at all, shared page tables are created. Thus, shared page tables are created by default if no action is taken to disable their creation. We believed that most applications would benefit from shared page tables and thus should be created transparently by default.

Once the decision is made to create shared page tables for the global section, the system service routine allocates a set of global section data structures for the shared page table global section. These structures are initialized in the same manner as their memory-resident counterparts, and in many cases the fields in both sets of structures contain identical data.

Note that on current Alpha platforms, there is one shared page table page for every 1,024 global section pages or 8 MB. (The number of shared page table pages is rounded up to accommodate global sections that are not even multiples of 8 MB in size.)

Shared PTEs represent the data within a shared page table global section and are initialized by the operating system. Since page table pages are not accessible through page table space[4] until the process maps to the data, the initialization of the shared page table pages presented some design issues. To initialize the shared page table pages, they must be mapped, yet they are not mapped at the time that the global section is created.

A simple solution to the problem was chosen. Each shared page table page is temporarily mapped to a system space virtual page solely for the purposes of initializing the shared PTEs. Temporarily mapping each page allows the shared page table global section to be fully initialized at the time it is created.

An interesting alternative for initializing the pages would have been to set the upper-level PTEs invalid, referencing the shared page table global section. The page fault handler could initialize a shared page table page when a process accesses a global section page, thus referencing an invalid page table page. The shared page table page could then be initialized through its mapping in page table space. Once the page is initialized and made valid, other processes referencing the same data would incur a global valid fault for the shared page table page. This design was rejected due to the additional overhead of faulting during execution of the application, especially when the ALLOC option is used for the memory-resident global section.

Mapping to a Shared Page Table Global Section Mapping to a memory-resident global section that has shared page tables presented new challenges and constraints on the mapping criteria normally imposed by the virtual address space creation routines. The mapping service routines require more stringent mapping criteria when mapping to a memory-resident global section that has shared page tables. These requirements serve two purposes:

1. Prevent process private data from being mapped onto shared page tables. If part of a shared page table page is unused because the memory-resident global section is not an even multiple of 8 MB, the process would normally be allowed to create private data there.
2. Accommodate the virtual addressing alignments required when mapping page tables into a process' address space.

For applications that cannot be changed to conform to these mapping restrictions, a memory-resident global section with shared page tables can be mapped using the process' private page tables. This capability is also useful when the memory-resident global section is mapped read-only. This mapping cannot share page tables with a writable mapping because the access protection is stored within the shared PTEs.

Shared Page Table Virtual Regions The virtual region support added in OpenVMS Alpha version 7.0 was extended to aid in prohibiting process private pages from being mapped by PTEs within shared page tables. Virtual regions are lightweight objects a process can use to reserve portions of its process virtual address space. Reserving address space prevents other threads in the process from creating address space in the reserved area, unless they specify the handle of that reserved area to the address space creation routines.

To control which portion of the address space is mapped with shared page tables, the shared page table attribute was added to virtual regions. To map a memory-resident global section with shared page tables, the user must supply the mapping routine with the name of the appropriate global section and the region handle of a shared page table virtual region.

There are two constraints on the size and alignment of shared page table virtual regions.

1. The size of a shared page table virtual region must be an even multiple of bytes mapped by a page table page. For an 8-KB page system, the size of any shared page table virtual region is an even multiple of 8 MB.
2. The caller can specify a particular starting virtual address for a virtual region. For shared page table virtual regions, the starting virtual address must be aligned to an 8-MB boundary. If the operating system chooses the virtual address for the region, it ensures the virtual address is properly aligned.

If either the size or the alignment requirement for a shared page table virtual region is not met, the service fails to create the region.

The size and alignment constraints placed on shared page table virtual regions keep page table pages from spanning two different virtual regions. This allows the operating system to restrict process private mappings in shared page table regions and shared page table mappings in other regions by checking the shared page table's attribute of the region before starting the mapping operation.

Mapping within Shared Page Table Regions The address space mapped within a shared page table virtual region also must be page table page aligned. This ensures that mappings to multiple memory-resident global sections that have unique sets of shared page tables do not encroach upon each other.

The map length is the only argument to the mapping system service routines that need not be an even multiple of bytes mapped by a page table page. This is allowed because it is possible for the size of the memory-resident global section to not be an even multiple of bytes mapped by a page table page. A memory-resident global section that fits this length description will have a portion of its last shared page table page unused.

The Reserved Memory Registry

OpenVMS Alpha VLM support provides a physical memory reservation system that can be exploited by VLM applications. The main purpose of this system is to provide portions of the system's physical memory to multiple consumers. When necessary, a consumer can reserve a quantity of physical addresses in an attempt to make the most efficient use of system components, namely the translation buffer. More efficient use of the CPU and its peripheral components leads to increased application performance.

Alpha Granularity Hint Regions

A translation buffer (TB) is a CPU component that caches recent virtual-to-physical address translations of valid pages. The TB is a small amount of very fast memory and therefore is only capable of caching a limited number of translations. Each entry in the TB represents a single successful virtual-to-physical address translation. TB entries are purged either when a request is made by software or when the TB is full and a more recent translation needs to be cached.

The Alpha Architecture coupled with software can help make more effective use of the TB by allowing several contiguous pages (groups of 8, 64, or 512) to act as a single huge page. This single huge page is called a granularity hint region and is composed of contiguous virtual and physical pages whose respective first pages are exactly aligned according to the number of pages in the region. When the conditions for a granularity hint region prevail, the single huge page is allowed to consume a single TB entry instead of several. Minimizing the number of entries consumed for contiguous pages greatly reduces turnover within the TB, leading to higher chances of a TB hit. Increasing the likelihood of a TB hit in turn minimizes the number of virtual-to-physical translations performed by the CPU.[5]

Since memory-resident global sections are nonpageable, mappings to memory-resident global sections greatly benefit by exploiting granularity hint regions. Unfortunately, there is no guarantee that a contiguous set of physical pages (let alone pages that meet the alignment criteria) can be located once the system is initialized and ready for steady-state operations.

Limiting Physical Memory

One technique to locate a contiguous set of PFNs on OpenVMS (previously used on Alpha and VAX platforms) is to limit the actual number of physical pages used by the operating system. This is accomplished by setting the PHYSICAL_MEMORY system parameter to a value smaller than the actual amount of physical memory available in the system. The system is then rebooted, and the PFNs that represent higher physical addresses than that specified by the parameter are allocated by the application.

This technique works well for a single application that wishes to allocate or use a range of PFNs not used by the operating system. Unfortunately, it suffers from the following problems:

- It requires the application to determine the first page not used by the operating system.
- It requires a process running this application to be highly privileged since the operating system does not check which PFNs are being mapped.
- Since the operating system does not arbitrate access to the isolated physical addresses, only one application can safely use them.
- The Alpha Architecture allows for implementations to support discontiguous physical memory or physical memory holes. This means that there is no guarantee that the isolated physical addresses are successively adjacent.
- The PFNs above the limit set are not managed by the operating system (physical memory data structures do not describe these PFNs). Therefore, the pages above the limit cannot be reclaimed by the operating system once the application is finished using them unless the system is rebooted.

The Reserved Memory Solution

The OpenVMS reserved memory registry was created to provide contiguous physical memory for the purposes of further improving the performance of VLM applications. The reserved memory registry allows the system manager to specify multiple memory reservations based on the needs of various VLM applications.

The reserved memory registry has the ability to reserve a preallocated set of PFNs. This allows a set of contiguous pages to be preallocated with the appropriate alignment to allow an Alpha granularity hint region to be created with the pages. It can also reserve physical memory that is not preallocated. Effectively, the application creating such a reservation can allocate the pages as required. The reservation ensures that the system is tuned to exclude these pages.

The reserved memory registry can specify a reservation consisting of prezeroed PFNs. It can also specify that a reservation account for any associated page tables. The reservation system allows the system manager to free a reservation when the corresponding consumer no longer needs that physical memory.

The memory reserved by the reserved memory registry is communicated to OpenVMS system tuning facilities such that the deduction in fluid memory is noted when computing system parameters that rely on the amount of physical memory in the system.

SYSMAN User Interface The OpenVMS Alpha SYSMAN utility supports the RESERVED_MEMORY command for manipulating entries in the reserved memory registry. A unique character string is specified as the entry's handle when the entry is added, modified, or removed. A size in megabytes is specified for each entry added.

Each reserved memory registry entry can have the following options: preallocated PFNs (ALLOC), zeroed PFNs, and an allotment for page tables. VLM applications enter their unique requirements for reserved memory. For memory-resident global sections, zeroed PFNs and page tables are usually specified.

Reserving Memory during System Start-up To ensure that the contiguous pages can be allocated and that run-time physical memory allocation routines can be used, reserved memory allocations occur soon after the operating system's physical memory data structures have been initialized.

The reserved memory registry data file is read to begin the reservation process. Information about each entry is stored in a data structure. Multiple entries result in multiple structures being linked together in a descending-order linked list. The list is intentionally ordered in this manner, so that the largest reservations are honored first and contiguous memory is not fragmented with smaller requests.

For entries with the ALLOC characteristic, an attempt is made to locate pages that will satisfy the largest granularity hint region that fits within the request. For example, reservation requests that are larger than 4 MB result in the first page allocated to be aligned to meet the requirements of a 512-page granularity hint region.

The system's fluid page counter is reduced to account for the amount of reserved memory specified in each entry. This counter tracks the number of physical pages that can be reclaimed from processes or the system through paging and swapping. Another system-defined value, minimum fluid page count, is calculated during system initialization and represents the absolute minimum number of fluid pages the system needs to function. Deductions from the fluid page count are always checked against the minimum fluid page count to prevent the system from becoming starved for pages.

Running AUTOGEN, the OpenVMS system tuning utility, after modifying the reserved memory registry allows for proper initialization of the fluid page counter, the minimum fluid page count, and other system parameters, thereby accommodating the change in reserved memory. AUTOGEN considers entries in the reserved memory registry before selecting values for system parameters that are based on the system's memory size. Failing to retune the system can lead to unbootable system configurations as well as poorly tuned systems.

Page Tables Characteristic The page table reserved memory registry characteristic specifies that the reserved memory allotment for a particular entry should include enough pages for its page table requirements. The reserved memory registry reserves enough memory to account for lower-level page table pages, although the overall design can accommodate allotments for page tables at any level.

The page table characteristic can be omitted if shared page tables are not desired for a particular memory-resident global section or if the reserved memory will be used for another purpose. For example, a privileged application such as a driver could call the kernel-mode reserved memory registry routines directly to use its reservation from the registry. In this case, page tables are already provided by the operating system since the reserved pages will be mapped in shared system address space.

Using Reserved Memory Entries are used and returned to the reserved memory registry using a set of kernel-mode routines. These routines can be called by applications running in kernel mode such as the system service routines that create memory-resident global sections. For an application to create a memory-resident global section and use reserved memory, the global section name must exactly match the name of the reserved memory registry entry.

After the system service routine has obtained the reserved memory for the memory-resident global section, it calls a reserved memory registry routine again for the associated shared page table global section. If page tables were not specified for the entry, the system service routine does not create a shared page table global section.

A side benefit of using the ALLOC option for the memory-resident global section is that the shared page tables can be mapped into page table space using granularity hint regions as well.

Returning Reserved Memory The memory used by a memory-resident global section and its associated shared page table global section is returned to the reserved memory registry (by calling a kernel-mode routine) when the global section is deleted. Reserved memory is only returned when a memory-resident global section has no more outstanding references. Preallocated pages are not returned to the system's free page list.

Freeing Reserved Memory Preallocated reserved memory that is unused or partially used can be freed to the system's free page list and added to the system's fluid page count. Reserved fluid memory is returned to the system's fluid page count only.

Once an entry's reserved memory has been freed, subsequent attempts to use reserved memory with the same name may be able to use only the FLUID option, because a preallocated set of pages is no longer set aside for the memory-resident global section. (If the system's fluid page count is large enough to accommodate the request, it will be honored.)

The ability to free unused or partially used reserved memory registry entries adds flexibility to the management of the system. If applications need more memory, the registry can still be run with the FLUID option until the system can be rebooted with a larger amount of reserved memory. A pool of reserved memory can be freed at system start-up so that multiple applications can use memory-resident global sections to a limit specified by the system manager in the reserved memory registry.

Reserved Memory Registry and Other Applications
Other OpenVMS system components and applications may also be able to take advantage of the reserved memory registry.

Applications that relied upon modifications to the PHYSICAL_MEMORY system parameter as a means of gaining exclusive access to physical memory can enter kernel mode and call the reserved memory registry kernel-mode routines directly as an alternative. Once a contiguous range of PFNs is obtained, the application can map the pages as before.

Using and returning reserved memory registry entries requires kernel-mode access. This is not viewed as a problem because applications using the former method (of modifying the PHYSICAL_MEMORY system parameter) were already privileged. Using the reserved memory registry solves the problems associated with the previous approach and requires few code changes.

Performance Results

In a paper describing the 64-bit option for the Oracle7 Relational Database System,[1] the author underscores the benefits realized on a VLM system running the DIGITAL UNIX operating system. The test results described in that paper highlight the benefits of being able to cache large amounts of data instead of resorting to disk I/O. Although the OpenVMS design team was not able to execute similar kinds of product tests, we expected to realize similar performance improvements for the following reasons:

- More of a VLM application's hot data is kept resident instead of paging between memory and secondary storage.
- Application start-up and shut-down times are significantly reduced since the page table structures for the large shared memory object are also shared. The result is that many fewer page tables need to be managed and manipulated per process.
- Reducing the amount of PTE manipulations results in reduced lock contention when hundreds of processes map the large shared memory object.

As an alternative to product testing, the design team devised experiments that simulate the simultaneous start-up of many database server processes. The experiments were specifically designed to measure the scaling effects of a VLM system during application start-up, not during steady-state operation.

We performed two basic tests. In the first, we used a 7.5-GB, memory-resident global section to measure the time required for an increasing number of server processes to start up. All server processes mapped to the same memory-resident global section using shared page tables. The results shown in Figure 8 indicate that the system easily accommodated 300 processes. Higher numbers of processes run simultaneously caused increasingly large amounts of system stress due to the paging of other process data.

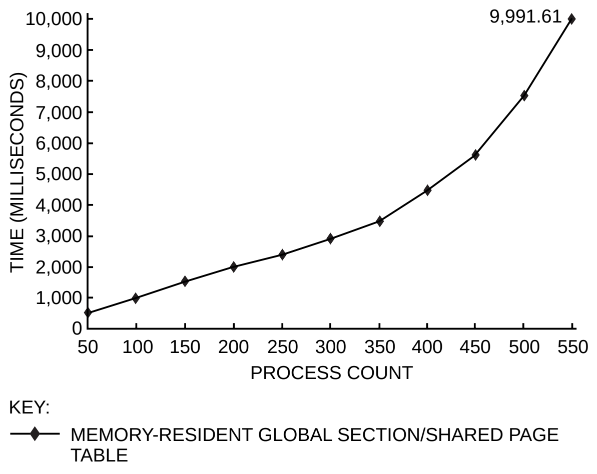


Figure 8
Server Start-up Time versus Process Count

In another test, we used 300 processes to measure the time required to map a memory-resident global section with and without shared page tables. In this test, the size of global section was varied. Note that the average time required to start up the server processes rises at nearly a constant rate when not using shared page tables. When the global section sizes were 5 GB and greater, the side effect of paging activity caused the start-up times to rise more sharply as shown in Figure 9.

The same was not true when using shared page tables. The time required to map the increasing section sizes remained constant at just under three seconds. The same experiment on an AlphaServer 8400 system with 28 GB of memory showed identical constant start-up times as the size of the memory-resident global section was increased to 27 GB.

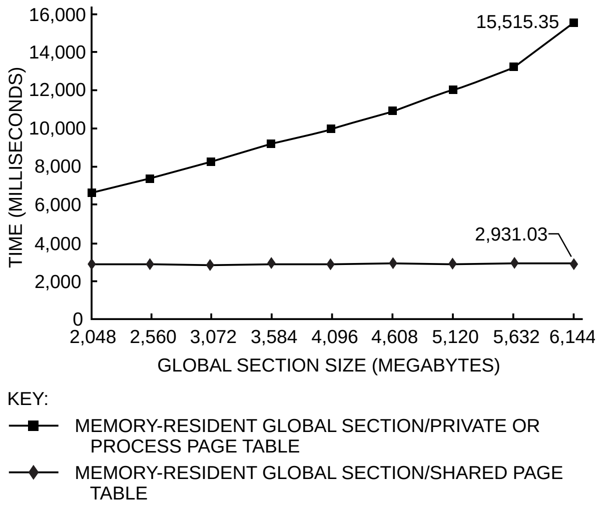


Figure 9
Server Start-up Time on an 8-GB System

Conclusion

The OpenVMS Alpha VLM support available in version 7.1 is a natural extension to the 64-bit virtual addressing support included in version 7.0. The 64-bit virtual addressing support removed the 4-GB virtual address space limit and allowed applications to make the most of the address space provided by Alpha systems. The VLM support enables database products or other applications that make significant demands on physical memory to make the most of large memory systems by allowing large caches to remain memory resident. The programming support provided as part of the VLM enhancements enables applications to take advantage of both 64-bit virtual addressing and very large memories in a modular fashion consistent with the OpenVMS shared memory model. This combination enables applications to realize the full power of Alpha VLM systems.

The Oracle7 Relational Database Management System for OpenVMS Alpha was modified by Oracle Corporation to exploit the VLM support described in this paper. The combination of memory-resident global sections, shared page tables, and the reserved memory registry has not only improved application start-up and run-time performance, but it has also simplified the management of OpenVMS Alpha VLM systems.

Acknowledgments

The authors wish to thank other members of the OpenVMS Alpha Kernel Group who contributed to the design and implementation of VLM support on OpenVMS Alpha: Tom Benson, Richard Bishop, Clair Grant, Andy Kuehnel, and Dave Wall. Jef Kennedy and Tony Lekas of Oracle Corporation provided great assistance as well. We would also like to thank Ron Higgins for his management support and Mike Harvey for his long-standing mentorship. Margie Sherlock and Tom Benson were especially helpful in assisting us with the writing aspects of this paper. We are also thankful to Gaitan D'Antoni who found the time in current project schedules to accommodate the writing of this paper.

References

1. V. Gokhale, "Design of the 64-bit Option for the Oracle7 Relational Database Management System," *Digital Technical Journal,* vol. 8, no. 4 (1996): 76–82.

2. R. Goldenberg and S. Saravanan, *OpenVMS AXP Internals and Data Structures, Version 1.5* (Newton, Mass.: Digital Press, 1994).

3. T. Benson, K. Noel, and R. Peterson, "The OpenVMS Mixed Pointer Sized Environment," *Digital Technical Journal,* vol. 8, no. 2 (1996): 72–82.

4. M. Harvey and L. Szubowicz, "Extending OpenVMS for 64-bit Addressable Virtual Memory," *Digital Technical Journal,* vol. 8, no. 2 (1996): 57–71.

5. R. Sites and R. Witek, *Alpha AXP Architecture Reference Manual,* 2d ed. (Newton, Mass.: Digital Press, 1995).

General References

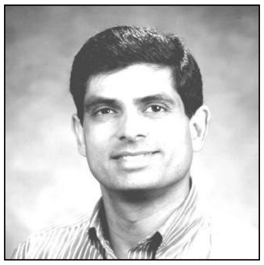
OpenVMS Alpha Guide to 64-bit Addressing and VLM Features (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-QSBCB-TE).

OpenVMS System Services Reference Manual: A-GETMSG (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-QSBMG-TE) and *OpenVMS System Services Reference Manual: GETQUI-Z* (Maynard, Mass.: Digital Equipment Corporation, Order No. AA-QSBNB-TE).

For a more complete description of shared memory creation on DIGITAL UNIX, see the *DIGITAL UNIX Programmer's Guide.*

Biographies

Karen L. Noel
A consulting engineer in the OpenVMS Engineering Group, Karen Noel was the technical leader for the VLM project. Currently, as a member of the OpenVMS Galaxy design team, Karen is contributing to the overall Galaxy architecture and focusing on the design of memory partitioning and shared memory support. After receiving a B.S. in computer science from Cornell University in 1985, Karen joined DIGITAL's RSX Development Group. In 1990, she joined the VMS Group and ported several parts of the VMS kernel from the VAX platform to the Alpha platform. As one of the principal designers of OpenVMS Alpha 64-bit addressing and VLM support, she has applied for nine software patents.



Nitin Y. Karkhanis
Nitin Karkhanis joined DIGITAL in 1987. As a member of the OpenVMS Alpha Executive Group, he was one of the developers of OpenVMS Alpha 64-bit addressing support. He was also the primary developer for the physical memory hole (discontiguous physical memory) support for OpenVMS Alpha. He is a coapplicant for two patents on OpenVMS VLM. Currently, Nitin is a principal software engineer in the OpenVMS NT Infrastructure Engineering Group, where he is a member of the OpenVMS DCOM porting team. Nitin received a B.S. in computer science from the University of Vermont in 1987.