Using a Distributed Single Address Space Operating System to Support Modern Cluster Computing

Alan Skousen Donald Miller
Computer Science and Engineering Department
Arizona State University, Tempe, Arizona 85287, USA
E-mail: {alan.skousen,donald.miller}@asu.edu

Abstract

Recent 64-bit microprocessors have made a huge 18.4 quintillion byte address space potentially available to programs. This has led to the design of Operating Systems that provide a single virtual address space in which all code and data reside in and across all levels of storage and all nodes of a distributed system. These operating systems, called SASOSs, have characteristics that can be used to support modern cluster computing in a distributed system in ways that provide an improved program development environment and higher performance than available from conventional operating systems. Sombrero, our SASOS design, makes use of its hardware support for object-grained protection, separate thread related protection domains and implicit protection domain crossing to provide support for modern cluster computing not available in SASOSs built on stock processors. Its design, which provides direct system level support for object oriented programming, includes a number of features targeted specifically at modern cluster computing.

1. Introduction

Single address space operating systems (SASOSs) have characteristics that can be used to support modern cluster computing in a distributed system in ways that provide an improved program development environment and higher performance than available from conventional operating systems. SASOSs provide permanent unique names for all resources, threads that can migrate throughout a distributed system without changing the pointer context in which they run, a basic substrate in which distributed applications can communicate using shared memory without additional setup, and persistence without use of a separate file system. When provided with changes to processor protection and memory management architecture to support its abstractions, a SASOS can provide an even better communications substrate for cluster computing. This paper presents a hardware-supported distributed SASOS, called Sombrero, with improved control transfer and resource access mechanisms that support sharing and protection in a clustered system of homogenous machines.

Sombrero makes use of its hardware support for object-grained protection, separate thread related protection domains and implicit protection domain crossing to provide support for modern cluster computing not available in SASOSs built on stock processors. Its design, which provides direct system level support for object oriented programming, includes a number of features targeted specifically at modern cluster computing. It contains resource access and control transfer mechanisms and additional support for the use of passive servers in client/server and other clustered computing systems. The use of passive servers, a concept available only to a SASOS, has a far-reaching affect on performance because it facilitates caching and migration that reduce the likelihood of a server forming a bottleneck. Passive servers migrate to the point of service via the caching effect, which provides improved user response time. With its large distributed single address space Sombrero has optimized access to neighboring computers to provide a scalable cluster targeted at close knit operations too large for a single node such as combined CAD/CAM and Process Management systems and DDBMSs.

In the next section, we briefly present an overview to single address spaces and SASOSs with and without additional hardware support as background for the cluster computing operations described in subsequent sections. Section 3 presents an overview of the Sombrero hardware and software design. Section 4 examines and compares protection, resource access and control transfer in process-oriented systems and SASOSs with and without additional protection hardware. Section 5 focuses on how these mechanisms are used as basic tools and for scaleable resource management in a clustered computer system. Section 6 reports on the status of the Sombrero prototype and a simulation of Sombrero hardware and one of Sombrero software. Section 7 discusses related hardware-supported SASOS work and Section 8 contains a brief

summary. The Appendix contains additional details of the Sombrero design.

2. Single address spaces and SASOSs

The recent advent of 64-bit RISC processors such as the DEC Alpha 21164, MIPS R10000 and HP PA8500 have made much greater virtual address spaces available to applications and the operating system though none of them as yet provide a full 64 bit address space. This has led to a reexamination of the premises of the conventional process-oriented operating system model. In this model every process has its own copy of a virtual address space supported by page-oriented protection and there is a separate file system namespace for persistent data. The fundamental question is whether another model might supply an improved program development environment and increased performance. This paper presents the case for a Single Address Space Operating System (SASOS) model in which the operating system provides one very large address space for all programs and data. The SASOS is supported by protection hardware specifically designed for single address space usage. In addition, as a 64-bit virtual address space is large enough to provide addresses for all nodes of a modest network for "a long time" [1], the design includes extensions for network operation in a computing cluster.

2.1. Characteristics of a single address space

In a single address space there is one persistent network-wide set of virtual addresses that can be used to uniquely name every data object in a computer system. Its characteristics are:

1. Virtual addresses can be permanently and uniquely bound to all data and code objects. This means virtual addresses can serve as unique names and the virtual address space can serve as the only namespace.
2. The virtual address namespace spans all levels of the storage hierarchy on every node. This means all physical storage can be viewed as a hierarchy of caches for the contents of virtual addresses.
3. The single large namespace is manipulated directly by the CPU and access to it is controlled directly by memory and protection management hardware. This means the CPU can directly enforce access policy on all objects defined in the system as it manipulates virtual addresses. This is especially helpful for the frequent case when policy is cached in the CPU.

For a more complete treatment of SASOS advantages and the paradigm shift see [2].

2.2. Process-oriented operating systems

In process-oriented operating systems such as UNIX, NT, MVS and Mach, each process runs in it's own VA space making VAs context dependent. Protection is provided by the underlying memory management hardware. Process boundaries encapsulate and bind temporary VA spaces, execution state, objects, protection domains and principals. Persistent storage is in a separate namespace. This leads to frequent use of message passing for control transfer, data exchange and synchronization and forces servers to be active in the sense that they must be executed by their own threads. IPC using shared memory requires prior setup and mapping. Data migration is difficult, necessitating creation of a new namespace for distributed data (DSM). Distinct node namespaces for code make task migration difficult.

2.3. SASOSs with and without changed protection hardware support

Most contemporary SASOSs have been designed to run on stock RISC 64-bit processors. These include Opal [1][3], Mungi [4], Arius [5] and Grasshopper [6]. This means additional software communication protocols are required for protection. Typically capabilities are used requiring operations in one or more additional namespaces. SASOSs with protection and memory management hardware targeted at single address space operation including AS/400 [7], Monads [8] and Sombrero [9][2] can take fuller advantage of, and make a more natural fit to, single address space characteristics. For example, crossing of protection domain boundaries can be made implicit at all levels and more efficient. These changes to protection hardware require changes to current processor architectures and system software and pre-existing software may not be able to take full advantage of the SASOS architecture.

2.4. Motivation for building a SASOS

Processors with a 64-bit architecture are now widely available although current models provide pin-level support for less than 64 bits of VA space (ex: DEC Alpha 43 bits, MIPS R10000 44 bits). Another development has been the increasing size and complexity of modern software, leading to increased software engineering efforts to reduce the costs of program development. Simpler mechanisms that support sharing and communication between large programs and databases are needed. These include common naming mechanisms and system level support for Object Oriented Design and Programming. It is generally agreed that, for a variety of reasons, within about 10 years performance increases derived from reducing chip feature sizes will begin to decline [10]. Increased performance will have to come from architectural changes such as increased parallelism [11] and shifting to single address space operation. New chip designs such as Intel's Merced [12] can include both a TLB and an RPLB and a facility for selecting the active protection hardware allowing process-oriented and SASOS designs to coexist. The major reason for building a SASOS is that the very large common namespace has the potential to provide a

simpler program development environment and increased performance in single node and homogenous distributed system clusters. The tradeoffs are the costs associated with shifting to new hardware[1] and to a new design paradigm.

3. Sombrero design overview

3.1. Model

The basic abstractions in Sombrero are memory object, protection domain, thread and principal. The basic storage abstraction is the ***Memory Object***. All resources are represented and named by memory objects. A memory object is characterized by its range. *Memory ranges* are distinct, variable sized, contiguous allocations of the virtual address space. Memory ranges contain *memory regions*, which are possibly noncontiguous sets of virtual addresses within a range with one or more non-overlapping protection attributes from the set {rwx}. The basic protection abstraction is the ***Protection Domain***. It is an abstract entity describing the set of memory objects and protection domains that are reachable by a thread operating within it and their associated access permissions. There are two kinds of protection domains, *General Protection Domains* (GPDs) and *Carrier Protection Domains* (CPDs). GPDs contain the code, data and other GPDs accessible to all of the threads executing within them and CPDs contain the data and the GPDs privately accessible to an individual thread. At any instant the effective protection domain is the union of a GPD and a CPD. A ***thread*** is the unit of execution represented by the current or most recent relevant register state of the CPU including the contents of the instruction pointer, stack pointer, general registers, reachable CPD memory and some additional status information. A ***principal*** is a unique protected identity that represents a user or a system service. It binds the resources, activities and access privileges carried out on behalf of programs to an owning user and is used to manage authorization of system resources.

3.2. Hardware design functional requirements

To evaluate whether every memory access can be allowed, protection hardware must be CPU-resident. Sombrero includes a design of this hardware called a Range Protection Lookaside Buffer (RPLB) which adds to and extends the functionality described by the PLB suggested by Koldinger[14]. The design is targeted at single nodes and distributed systems consisting of homogenous workstations

Ideally, CPU hardware to support protection in a single address space should allow every CPU instruction that accesses memory from an authorized protection domain to be executed while preventing unauthorized access. In addition, no special instructions should be required for a thread to traverse protection domains or to extend access to previously unaccessed memory objects; rather access should be implicit reflecting higher-level policy decisions.

To allow this implicit protected access it's necessary to cache memory access and domain switching authorizations in a way that the protection checks can be integrated into the instruction pipeline. To do this requires a cache for protection information and a pair of registers containing virtual addresses identifying the current general protection domain and a carrier protection domain associated with the currently executing thread. The cache needs to store triplets of the form:

<protection domain tag, resource accessed, access rights>

where the protection domain tag and resource accessed, which can be either a memory region or another protection domain, are represented by virtual addresses and the access rights are {*rwxs*} where *s* means a domain switch is allowed. A protection cache with these features, such as the Sombrero RPLB, can provide this functionality. Results of a VLSI synthesis of the RPLB are given in section 6.2. A brief logical design of the Sombrero RPLB is given in Appendix 1.

3.3. Software protection data structures design

Kernel-resident data structures, cache the data needed to convert between user-level policy information in the form of resources, access rights and users, e.g., files, {rwx} privileges, user ids and resource ownership, and the CPU-level representation of these things in terms of protection domains, virtual addresses and access privileges. These data structures called Protection and Resource Access Lists (PRALs) logically contain two types of simple lists interlinked to provide fast resource access and efficient rights revocation. One type, called Protection Domain Access Lists, define a protection domain. Each list is headed by a control block representing a GPD or a CPD and contains Access Descriptors that indicate the resources and protection domains accessible to the GPD or CPD. These lists contain the same fundamental protection information held by classical capability lists. The second type, called Resource Access Control Lists, contain the list of valid accessors for a resource or a protection domain. A control block representing a memory object or a GPD heads each list. These lists are classical ACLs. In fact, because PRAL lists could be quite long and because the memory objects represented vary in size they are implemented as binary search tables or hash tables. PRAL contents are updated implicitly on a protection fault or on user request via Sombrero Systems Service routines. The attributes of a single address space facilitate the co-location of protection

[1] It has been pointed out that only a two-fold or greater improvement in performance makes a switch to incompatible hardware worthwhile [13].

triples. The RPLB and PRAL designs make use of this providing basic support for the operating system structure described in the next section. More detailed descriptions of the PRAL design can be found in [15] and [16]. A software simulation of the PRALs is summarized in Section 6.3.

3.4. Operating system structure

Sombrero has a modular OS structure rather than the traditional hierarchical structure. Executive modules are instantiations of object classes that are implemented as Sombrero protection domains. Their methods can be extended and overridden by user level modules. The Sombrero executive supports explicit definition of user protection policy via service requests and implicit implementation of protection via protection faults.

Hardware caching of protection domain object regions, hardware caching of allowable domain crossings and hardware-supported private thread protection domains are crucial to this structure. They allow Sombrero service requests by applications and upcalls by executive services to be implemented by ordinary procedure calls. There is a flat rather than hierarchical and layered structure for all modules, including those within and those outside of the executive services. In-calls and out-calls replace system calls and up-calls. Communication between the executive and user modules is the same as that between user programs. System calls implemented with trap instructions and the context switches associated with kernel intervention as occur in process-oriented operating systems are not required except to access privileged instructions. Privilege level (the one controlled by the kernel/user bit) is orthogonal to protection domain definition, although typically only executive modules may need to execute in kernel mode. Executive services run at user privilege level except when kernel mode is necessary to execute privileged instructions or access sensitive memory areas. The architecture allows all but basic services to be provided entirely by user level servers. User level services include authorization, user identification and ownership establishment, and resource naming. Also included is a stub/gateway function mechanism used to transfer control to user method overrides such as those in authorizing agents on the occurrence of events such as a PRAL miss during resource access following an RPLB fault. SSI calls are sufficiently complete to implement the functionality provided by Solaris 2.4 [17] system calls. The details are contained in [18].

In this architecture, the executive can be viewed as a set of peer-level persistent objects. Depending on the implementation, the traditional kernel is tiny – a pico-kernel or essentially non-existent. The very small kernel handles redirections for exceptions and traps and is also entered to allow access to privileged instructions for such purposes as context switching. Alternatively, with a few additional CPU control registers, the executive can be implemented in a *completely* non-hierarchical manner.

4.Communication in Sombrero and process-oriented architectures

Protection, resource access and control transfer are the underlying issues that must be dealt with in the design of a communication mechanism. This section examines these issues, contrasting approaches in process-oriented systems, SASOSs built on stock RISC processors and Sombrero, which contains changed protection hardware. For a more complete treatment of communications see [2].

4.1. Protection

Process-oriented systems bind protection to pages with page tables whose entries can be cached in a TLB. The page tables and TLB define the per-process page translation/protection map in effect at a given time. Stock processors are designed to support this operation. Every SASOS built on a conventional processor is forced to use these protection mechanisms. Protection hardware specifically designed for use by a SASOS can bind protection directly to data objects by protecting the VAs that comprise them. It can also define a second protection domain associated with a computation's thread and distinct from the environment it is executing in. The next two sections present in greater detail the resource access and control transfer mechanisms available in Sombrero to support communication in a cluster of computers. Where appropriate, we compare these against mechanisms that perform the same functions in Opal [3] and Mungi [4], which run on conventional 64-bit RISC processors. The key point is that changed CPU protection hardware provides resource access and control transfer directly without requiring additional software protocols. These protocols complicate programming and reduce the transparency of distributed system communication.

4.2. Resource access

In Sombrero, resources are named and access is controlled by co-located protection triples contained in the PRAL and cached in the RPLB. In SASOSs without additional protection hardware resources are named and password capabilities ultimately backed by page tables and a TLB provide access control. Capabilities are used to explicitly attach segments or to perform specific operations on objects. The capabilities can be presented implicitly or via a mechanism using published capabilities that enables segments to be attached transparently on address faults reducing the need for most applications to deal with them. However, presenting capabilities requires a software protocol at runtime. This protocol always involves another namespace, in which the resource owner can validate the capabilities presented by the accessor.

This is what the passwords are used for. They are essentially a protected sparse namespace used to corroborate protection triples. No matter how they are implemented, these software protocols add program complexity and overhead at one or more levels. Co-located CPU-resident protection triples that cache higher level policy checking, such as those used in Sombrero, can be used directly as capabilities improving performance and simplifying programming. Software protocols always involve serial operations and additional instructions. Access control using an RPLB can be done in "parallel", i.e., the required runtime protection check is made during the execution of the instruction performing the memory access. No additional instructions are required. In a distributed single address space, programming and performance benefits in resource access extend across the entire cluster, simplifying remote data sharing and data migration.

4.3. Control transfer

Software protocols are also used for control transfer in SASOSs built on stock RISC processors. To protect a calling client's data on a domain crossing, the capability lists used for resource access are grouped in a different way to change a client's protection domain so that it includes objects accessible via a capability list in the server domain plus a selected subset of the objects accessible in the client's domain. These are called protection domain extensions (PDXs). PortalIDs are also used to control transfers into another protection domain with capability presentation and protection operations performed serially by objects called proxies on the calling side and guards on the called side. By caching allowed protection domain crossings in hardware as is done in Sombrero it is possible to remove almost all the run time overhead associated with protection domain crossings. The 45-121 machine cycles on a *RISC* platform for well-designed one-way IPC [19], [20] can be reduced to close to the net *one* machine cycle required for a procedure call. CPU hardware defining a separate CPD allows data copying based on trust relationships to be performed independently of domain switches. A new CPD associated with the thread containing copies of arguments and data sets needed by the called program that require protection can be created, followed by an ordinary procedure call to effect the general protection domain switch. Copies need not be made if protection is not required. If they are required, given that the carrier protection domain can be reused later, the number of setups is sharply reduced.[2] This achieves in hardware what a PDX achieves in software. The RPLB removes the need for portalIDs (an additional namespace) used in Opal because the cache of VAs at which domain crossing is permitted provides protection for individual entry point addresses. When protection is page-based, only the address of the page can be used this way. A separate hardware-supported protection domain map for the private data of a thread of execution, as provided in Sombrero, replaces the simplifying assumption of orthogonality of execution and protection that underlies the proxy/guard/portalID mechanism, and argument passing at domain crossings can be faster and simpler. Capability-based systems essentially build workarounds on top of the pre-existing substrate in order to make it suitable for use in a single address space. These factors always reduce performance and complicate the programming environment, increasing program development costs. CPU protection hardware oriented towards operation in a single address space *changes* the underlying substrate to make it more suitable for use in this environment. Simplification of the programming environment and performance improvements for control transfer under a distributed SASOS extend across a network of clustered computers simplifying task migration and service access.

5. Support for modern clustered computing

All SASOSs provide the potential for increased utilization of network resources. Most of these advantages derive simply from distributing the advantages of a persistent global namespace over a LAN. These include client/server performance increases from using passive servers, more efficient DSM from reduced mapping operations and improvements to distributed parallel computation that derive from increased thread mobility. Supporting a SASOS with changed CPU protection hardware provides network-wide advantages to cluster computing beyond those available to SASOSs that run on stock processors. This section describes the distributed system design of Sombrero and the improvements to program development environment and performance made possible by, or greatly facilitated by, its additional CPU-based protection hardware.

5.1. Distributed system design overview

In Sombrero the global VA space is initially partitioned among 'owning' nodes with subsequent refinements as nodes enter and leave a cluster via a fissioning and merging technique [21]. Each node allocates memory from its own exclusive portion of the VA space. This means that there is a 'home' node for every virtual address range. Network routers use this unique address range to route network packets based on the destination memory address of the message instead of using IP numbers (another namespace). When memory objects, protection domains and threads are first created, the control blocks used to manage them are given virtual addresses (names) from the home node partition. If the memory object, protection

[2] It is roughly proportional to the number of traveling threads rather than the product of traveling threads and modules entered.

domain or thread is migrated or replicated to another node, a surrogate control block is created at that node with a virtual address from the new node's VA partition. This surrogate contains the virtual address name of the original control block. Every copy of a control block has three token bits that indicate whether read, write and execute privileges are enabled for the memory object, protection domain or thread on its node. These can be used, for example, to indicate whether the read or write privileges for a given node's copy of the memory object are in effect, whether a protection domain on a node can be entered or whether a thread or its clone is active on the node. In order to provide consistency management to support the caching and replication present in a clustered computing system, every control block also has a pointer to an associated 'token tracking' data structure. This structure contains pointers to token tracking structures on other nodes for the same or related objects that are used for copy-set and address space management. Finally, every node has one or more user authorization agents and methods associated with memory object control blocks (MOCBs) that implement distributed system consistency policy for memory objects. Exactly when a token is set to enable or disable a privilege is a policy decision controlled by the methods specified by the MOCB. Sombrero MOCB default methods enforce sequential consistency. Other consistency model policies, such as causal or release, can be implemented by user extensions overriding the MOCB base class methods. The initial privilege to access an object, control a thread or enter a protection domain is obtained from an authorizing agent by the corresponding Sombrero executive service as indicated in Section 3.4. The migration and replication of kernel objects using surrogates make the network's multiple computers appear as a single virtual multiprocessor. Additional information on Sombrero distributed system design can be found in [22] and [9].

5.2. Distributed object copy-set management

Object grained protection hardware directly supports object regions rather than pages as the base unit of network distribution. Per-object copy set management replaces the per-page copy set management in process-oriented systems and SASOSs built on stock hardware because TLB per-page access rights are no longer required to determine protection. Network data transmission granularity is therefore determined by access policy rather than by operating system virtual address translation mechanisms. Even more important, because the hardware enforced protection triples (PD, VA, access rights) are invariant and available throughout the network-wide virtual address space, every virtual address, and every memory object containing it, is tied to a specific access policy. Once established, access policy, i.e., who can access what and when, is implicit and transparently distributed across the network by the protection hardware resident on every node. Local and remote protection faults support object access policy in much the same way as local and remote page faults support memory sharing in DSM systems. And as in DSM, the demand-paged virtual memory management already present on every node automatically generates the object region address faults needed. Object-region granularity enables simple scaleable, per-object data structures and pointers to be used. Token tracking structures contain the list pointers of simple network-wide graphs used to locate the most recent copy of an object's modified page, and control block tokens are used to enable access. Only the data structures in use at a node need be stored by the node.

5.3. Distributed parallel computing

Applications that perform course-grained parallel computation over a client/server system benefit from SASOSs with additional CPU protection hardware and the above-mentioned support for user-extensible semantics policy managers. One application of this is support for distributed parallel processing environments such as that provided in the Calypso research project [23]. Separate thread carrier protection domains directly support the kind of thread cloning needed in Concurrent Read Exclusive Write (CREW) semantics. These semantics, essentially a form of weak consistency semantics with a single synchronizing variable, are very useful in an environment where multiple nodes act as CPU servers for a single client node. CREW semantics allow concurrent multiple writes to the same and to different parts of the same memory objects on different nodes, controlled by global synchronization primitives and 'smart' application programmers [24].

5.4. Client/Server and other clustered workstation applications

The common global address space allows true thread migration in a SASOS. The Thread stack, thread private static data, thread ID, and other thread state, e.g., run status and priority, and general register contents can all be copied into the same virtual addresses and CPU general registers on the destination node. The thread can be immediately restarted and any missing data can be migrated to the thread. The performance of Distributed Shared Memory is considerably improved in all SASOSs because mappings between multiple VA namespaces are unnecessary. The ability of threads to migrate in a SASOS enables servers to be constructed as passive entities. They can exist in a separate protection domain on another node and, in a hardware supported SASOS, be entered and activated by remote clients' threads via ordinary procedure calls. The existence of passive servers has a far-reaching affect on client/server performance because it makes possible service migration and replication via caching that

reduces the likelihood of a server becoming a bottleneck and also provides improved performance from local operations. Hardware support for a separate CPD and GPD adds flexibility to server design. Servers can be designed to either retain client information within their own GPD or to operate on this information when it is introduced as part of the calling client's CPD. In the former case, a copy of the server GPD would migrate to the client node. In the latter case, the thread and its messages and other initialization data contained in a carrier protection domain can migrate to the server's node or server code can be cached on the client node for efficiency.

5.5. Distributed system fault tolerance

In a single address space every level of storage on every node can be viewed as a cache for the virtual address space. Sombrero takes advantage of this cache hierarchy to place an archival fault tolerant layer at the bottom of the hierarchy. This layer is used for checkpointing portions of the database that are in a consistent state. Fault tolerance in a single address space is no harder than (and no easier than) that in process-oriented systems with a separate file system namespace for persistent storage. Similar logging and private workspace mechanisms apply.

6. Prototype status and simulation results

6.1. Prototype

Sombrero is currently being prototyped on two Alpha 21164 boxes with sufficient resources to compile and run Windows NT source code. The reachable address space has been extended by specializing an NT process to include the full range available to the processor. This has been done by modifying the NT Alpha Privileged Architecture Library Code (PALCode) to recognize a Sombrero process and to forward TLB misses outside the normal NT address range to a Sombrero extension of the PALCode. A pager in NT user space forms the correct TLB entry and handles the miss. With this structure available the PALCode has been extended to emulate an RPLB that is triggered by misses in the TLB. Using this emulated RPLB hardware, individual threads are assigned to protection domains within the specialized process. This provides a functional protected single address space that supports our proposed SASOS. The NT operating system running in Kernel and Executive modes in the NT kernel and NT user space respectively provide the basic I/O facilities to Sombrero, which runs in Supervisor mode. We are currently testing Sombrero for correctness and proof of concept. Performance will be extrapolated by measuring the overhead caused by hardware emulation of the RPLB.

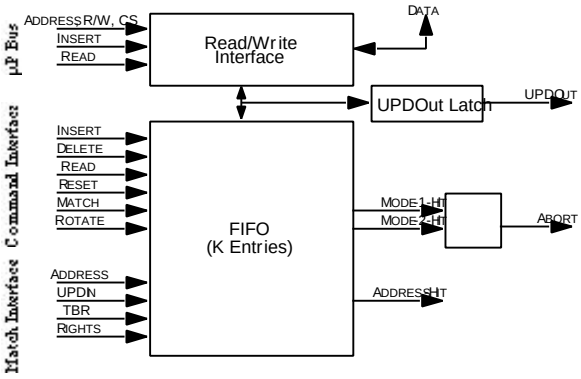


Figure 1. RPLB architecture synthesized with Synopsis Design Compiler using 0.35-micron library.

6.2. RPLB VLSI synthesis

The RPLB design was described and proof of concept verified in VHDL (VHSIC Hardware Description Language), using the Mentor design tools available in the ASU ULSI Design Laboratory. Subsequently, synthesis, the process of converting a language description (in this case VHDL) into a schematic or netlist description, was performed using Mentor's Autologic II synthesis tool and Synopsys Design Compiler and the Cascade Design Automation static timing analyzer, Tactic, performed timing analysis [25]. The objective was to design and synthesize a block able to provide memory protection within the time constraints of the DEC Alpha pipeline, and to do so in a 500 MHz, 3.3/2.5 V, 0.35 µm circuit. The synthesized RPLB (Figure 1) consists of two major blocks: the Read/Write interface through which entries are written and a K-entry (K=8 in this synthesis) FIFO buffer which also performs the matching function and implements a least-recently-used (LRU) replacement algorithm. Additionally, there is a latch UPDOut that stores the output of the FIFO when a valid domain switch occurs.

Eighteen different synthesis recipes were run on a single FIFO location to determine worst case single location size and timing and a number of runs were made to determine timing for the entire RPLB. Worst FIFO area observed was approximately 1.27 mm2 as compared to 208 mm2 for an Alpha 21164 manufactured to a 0.35 micron process indicating that an RPLB for k=64 would probably not increase Alpha chip size significantly. Timing analysis indicates a worst case speed of operation of the synthesized RPLB of 120 MHz for the entire FIFO and 250 MHz for a single FIFO location. Although this does not meet the timing requirements of a state-of-the-art 500 MHz Alpha microprocessor, there are architectural and technological reasons to believe that the synthesis indicates inclusion of an RPLB within a RISC CPU is well within the capabilities of today's silicon fabrication

technology. The most important of these were that the synthesis used an LRU replacement algorithm as contrasted to the Not Last Used algorithm used by DEC in the Alpha TLB [26]; a generic RPLB was synthesized rather than three separate RPLBs designed specifically for protection of instruction; data and domain switching accesses, and CAD tools, library elements and custom techniques available to DEC [27] were not available to the synthesis. More details can be found in [25].

6.3. PRAL simulation

Performance of the Sombrero protection model has been evaluated by simulating the average cost of memory access in Sombrero and in a typical, process-oriented operating system, Alpha-NT. Cache access traces from the Internet were used to drive the simulated hardware caches and software data structures, the Sombrero RPLB and PRALs and the Alpha TLB and Alpha-NT page tables. Performance statistics including cache miss rates and miss rate penalties versus the number of protection domains were collected. By necessity, due to the lack of an actual system, the numbers obtained were approximate but are useful to observe and characterize behavior differences between the two systems. Simulation results showed that while the RPLB miss penalty is likely to be twice as high as the TLB miss penalty, the RPLB miss rate is likely to be lower than the TLB miss rate under most common scenarios. This is true, for instance, when the memory objects accessed are likely to be larger than a page. In these situations performance, as measured by the product of miss rate and miss penalty appears to be roughly comparable with the RPLB performing significantly better as the average accessed memory object becomes much larger than a page. This offers a solution for the page table problems associated with larger address spaces. Details on the PRAL simulation can be found in [16].

7. Related hardware-supported SASOS work

Two SASOSs that include CPU hardware to provide protection, AS/400 and MONADS have been implemented in recent times.

7.1. AS/400

The IBM AS/400 [7] is the only SASOS in widespread use. It has a Single Address Space Operating System comprised of a low level piece, the System Licensed Internal Code (SLIC) and a higher level piece OS/400 and has a well defined interface, called the Machine Interface (MI) between them. It runs on Power PC 64-bit RISCs. There are probably over 100,000 AS/400 systems in commercial use. The system architecture is targeted at database management-type usage. The AS/400 has a great deal of special purpose hardware. All of memory has additional tag bits to differentiate between protected pointers and ordinary data. Secondary store has tag fields for the same purpose. The Power PC architecture used in the AS/400 has additional instructions for tag management and other purposes. OS/400 supports a single large address space on each node but this does not appear to span a network of machines, even though a principal class of users are client/server systems. The internal design is object-oriented, as is the MI available to the outside world. Protected pointers provide most of the protection for user level programs and data. Capabilities are used internally. The AS/400 MI is a technology independent interface that converts the underlying 64-bit single address space to an externally available high performance object-oriented single 64-bit address space especially suited for database management systems. The full set of single address space properties is not available to applications. The AS/400 is a very successful industrial strength product, demonstrating the feasibility of a SASOS in its targeted application area.

7.2. MONADS

MONADS [28] [8] is a research SASOS now only being actively pursued at Ulm Universitaat in Germany. Prior to that it was developed at three universities in Australia, most recently, the University of Sydney. It has extensive special purpose hardware support including a segmentation mechanism that supports capabilities. As callers must present a capability to cross domains that needs to be validated for the type of access and converted to an entry point, it does not appear that a single instruction suffices for a domain switch. MONADS' hardware, however, is streamlined to support fast implicit capability-based domain switching. MONADS served as a substrate for persistent object management system design. MONADS supports a network wide single level store and a flat 60-bit paged single virtual address. The MONADS hardware predates modern 64-bit RISC technology. Researchers at the University of Sydney are currently working on the Grasshopper orthogonal persistent object operating system, which is targeted to run on stock 64-bit RISC processors [6].

8. Summary

The SASOS concept and in particular Sombrero provide a low level substrate that substantially reduces the problems to clustered computing associated with process oriented architectures. In particular communication between cooperating nodes in a cluster becomes transparent allowing distributed program design to be handled as though it operated entirely within a single process. The resulting program simplification and reduction in the number of instructions needed for program execution reduces the cost of programming and makes better use of available hardware.

Appendix RPLB logical design

Figures A.1 and A.2 give an overview of RPLB intra-domain operation. The unit of protected resource access in Sombrero is a memory region. The virtual addresses of a region can be defined by a Match Mask and a Don't Care Mask as shown in Figure A.1. The current values of the CPD and GPD, actually the virtual addresses of the control blocks that store data about them, are stored in two high speed base registers, CPDBR and GPDBR as shown at the bottom of Figure A.2.

An RPLB entry for part or all of a region is shown at the top of Figure A.2. The top set of fields is loaded into the RPLB on a miss from a software data structure, called a region control block that holds this information. When the RPLB is loaded, combinational logic generates a Range Mask that defines the set of valid addresses stored in the RPLB. CPU emitted data is compared against the RPLB stored logical information on every memory reference to determine whether the access is allowed.

Figure A.3 gives an overview of RPLB inter-domain operation. To determine whether a new domain can be entered the RPLB needs to store the virtual address that is being accessed, the protection domain attempting to make the access and a flag {s} indicating that this entry specifies an attempted cross domain switch rather than intra-domain operation. In addition, it must provide the virtual address that is the name of the new General Protection Domain for storage into the GPDBR on a valid access.

The top set of fields is loaded into the RPLB on a miss. The CPU emitted data is compared against the RPLB stored information to determine whether the domain crossing is allowed. The RPLB associatively compares each entry against the CPU emitted data. For allowed accesses the new GPD name is stored in the GPDBR and this is followed by an intra-domain access attempt. More information on RPLB operation can be found in [Sko94] [SkMi96a].

CPU protection hardware targeted at SASOS functionality separates address translation and protection functions allowing the use of a single large page table. Although it's not essential to the Sombrero design, address translation can be moved from the CPU to the memory bus. This simplifies translation coherency on multiprocessor nodes and permits all on-chip code and data caches to be fast virtually indexed and tagged devices. It also enables the use of a single CAM-based inverted page table, as was done in Monads [8], for all of a node's virtual-physical address translation eliminating the need for a TLB.

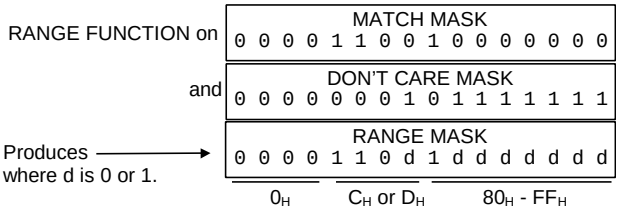


Figure A.1. RPLB Range Mask defining a region consisting of virtual addresses from $0C80_H$-$0CFF_H$ and $0D80_H$-$0DFF_H$.

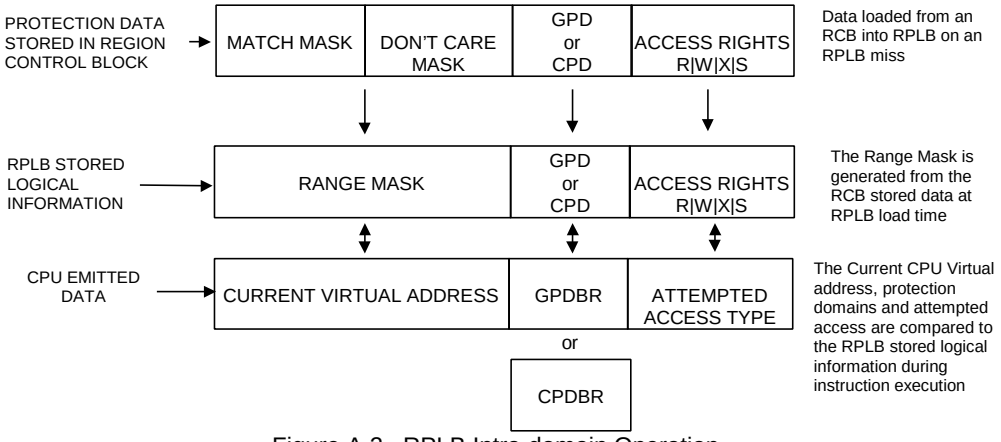


Figure A.2. RPLB Intra-domain Operation.

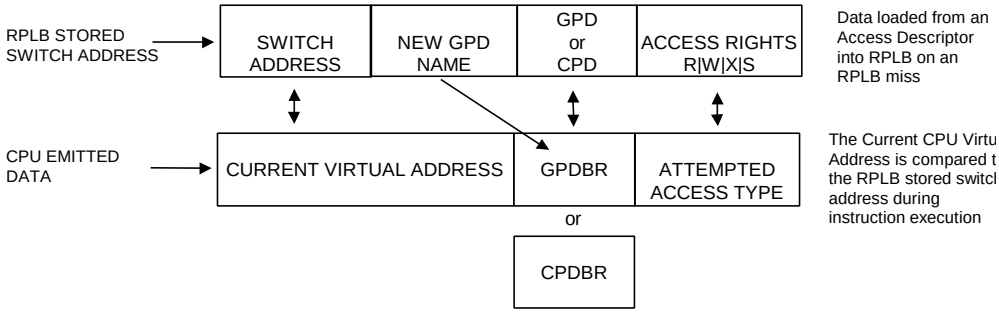


Figure A.3. RPLB Cross-domain Operation.

References

[1] Chase, J. S., Levy, H. M, Feeley, M. J., and Lazowska, E. D., "Sharing and Protection in a Single-Address-Space Operating System", *ACM Transactions on Computer Systems*, Vol. 12, No. 4, November 1994, pp. 271-307.

[2] Skousen, A. and Miller, D., "Operating System Structure and Processor Architecture for a Large Distributed Single Address Space", Proceedings of PDCS'98: 10th International Conference on Parallel and Distributed Computing and Systems. October 28-30, 1998.

[3] Chase, J. "An Operating System Structure for Wide-Address Architectures", *Tech. Report 95-08-06*, Department of Computer Science, University of Washington, August 1995, Ph.D. Dissertation.

[4] Heiser, G., Elphinstone, K., Vochteloo, J., Russell, S., and Liedtke, J., "Implementation and Performance of the Mungi Single-Address Space Operating System", *UNSW-CSE-TR-9704*, Department of Computer Systems, The University of New South Wales, June 1997.

[5] Wilkinson, T. and Murray, K., "Evaluation of a Distributed Single Address Space Operating System", *IEEE Proceedings of the 16th ICDCS*, 1996.

[6] Dearle, A., di Bona, R., Farrow, J., Henskens, F., Lindström, A., Rosenberg, J. and Vaughn, F., "Grasshopper: An Orthogonally Persistent Operating System", *Computing Systems*, 7(3), pp. 289-312, Summer 1994.

[7] Soltis, F. G., *Inside the AS/400*, Duke Press, 1995.

[8] Rosenberg, J., and. Abramson, D., "MONADS-PC - A Capability-Based Workstation to Support Software Engineering", *Proceedings of the Eighteenth Annual Hawaii International Conference on System Sciences*, pages 515-522, IEEE, 1985.

[9] Skousen, A. C., "Sombrero: A Very Large Single Address Space Distributed Operating System", Department of Computer Science and Engineering, Arizona State University, MS Thesis, December 1994.

[10] Pountain, D., "Amending Moore's Law", *Byte*, March 1998.

[11] Lo, J., Eggers, S., Emer, J., Levy, H., Stamm, R., and Tullsen, D., "Converting Thread-Level Parallelism to Instruction-Level Parallelism via Simultaneous MultiThreading", *ACM Transactions on Computing Systems*, August 1997.

[12] Halfhill, T., "Beyond Pentium II", *Byte*, December 1997.

[13] Yu, A., "The Future of Microprocessors", *IEEE Micro*, December 1996.

[14] Koldinger, E. J., Chase, J. S., and Eggers, S. J., "Architectural Support for Single Address Space Operating Systems", *Proceedings of 5th International Conference on Architectural Support for Programming Languages and Operating Systems*, *ACM SIGOPS Operating Systems Review*, October 1992.

[15] Skousen, A. C. and Miller, D. S., "Resource Access and Protection in Sombrero: Protection Model, Software Protection Data Structures and Hardware Range Protection Lookaside Buffer, ASU 64-bit OS Group WP 2", Computer Science and Engineering Department, Arizona State University, *TR-96-013*, May 1996.

[16] Khatri, S., "Protection Structures in the Sombrero Operating System", MS Thesis, December 1997.

[17] Sun Microsystems Inc., *Solaris 2.4 Manual Pages*, 1994.

[18] Carnes, M., "Sombrero System Interface", MCS Project, Computer Science and Engineering Department, Arizona State University, December 1996.

[19] Liedtke, J., "On Micro-Kernel Construction", *Proceedings of the 15th SOSP*, *Operating Systems Review*, Vol. 29, No. 5, December 1995, pp. 191-200.

[20] Liedtke, J., Elphinstone, K., Schönberg, S., Härtig, H., Heiser, G., Islam, N. and Jaeger, T., "Achieved IPC Performance (Still the Foundation for Extensibility)", *Sixth Workshop on Hot Topics in Operating Systems*, May 1997.

[21] Skousen, A. C. and Miller, D. S., "Implementing a Single Very Large Address Space across Multiple Nodes: Memory Partitioning, Protection Domain Migration, Kernel Replication, Consistency and Fault Tolerance, ASU 64-bit OS Group WP 10", Computer Science and Engineering Department, Arizona State University, *TR-96-021*, May 1996.

[22] Skousen, A. and Miller, D., "Using a Single Address Space Operating System to Support Client/Server Distributed Computing", Computer Science and Engineering Department, Arizona State University, *TR-97-019*, July 1997,

[23] Dasgupta, P., "Parallel Processing with Windows NT Networks", *Proceedings of the USENIX Windows NT Workshop*, August 1997, pp. 113-122.

[24] Mclaughlin, D., Sardesai, S. and Dasgupta, P., "Calypso NT: Reliable Efficient Parallel Processing on Windows NT Networks, *Technical Report, TR-97-001*, Department of Computer Science and Engineering, Arizona State University, 1997.

[25] Torla, M. J., "RPLB Design Project Report", MCS Project, Computer Science and Engineering Department, Arizona State University, July 1997.

[26] Digital Equipment Corporation, *Digital Semiconductor 21164 Alpha Microprocessor Hardware Reference Manual*, Order Number: EC-QP99B-TE, January 1997.

[27] Bowhill, W., Bell, S., Benschneider, J., et al, Circuit Implementation of a 300-MHz 64-bit Second-generation, CMOS Alpha CPU", *Digital Technical Journal*, Volume 7, Number 1, Special Edition, 1995.

[28] Rosenberg, J., "Architectural and Operating System Support for Orthogonal Persistence", *Computing Systems*, Vol. 5, No. 3, summer 1992, pp. 305-335.