♈-03 April 13, 2000

Multistriped Addressing

J.P. Grossman, Jeremy Brown, Andrew Huang, Tom Knight

Abstract

Data placement is an extremely important issue in parallel machines. In order to obtain good performance, it is often necessary to distribute a single object (such as a large array) across many nodes in a system. Traditional mechanisms for achieving this distribution are expensive as they require the mapping from virtual addresses to physical locations to be globally visible and coherent.

This paper presents ***Multistriped Addressing****, a novel technique for mapping virtual addresses to physical nodes which allows contiguous portions of the virtual address space to be striped across the system at varying granularities. Multistriped addressing has low hardware overhead and eliminates the need for global translation tables. It can be used to improve the efficiency of both cache-coherent and cacheless shared memory systems.*

1 Introduction

Parallel processing can be used to speed up many applications by several orders of magnitude. Massively parallel shared memory machines with hundreds or thousands of processor/memory nodes have been built (e.g. [Dally98], [Laudon97], [SGI]); in the future we will see machines with millions ([IBM00]) or even billions of nodes. In such large-scale systems, the layout of data in physical memory is crucial to achieving the maximum possible speedup. In particular, many good parallel algorithms require data structures to be distributed across multiple nodes in the system.

Typically, a fixed set of bits in the physical address (usually the highest bits) is used to determine the “home node” on which a piece of data resides. This scheme has the advantages of being simple and allowing data to be arbitrarily placed in the system on a page granularity. However, it has the significant disadvantage of requiring that every node be able to make arbitrary virtual to physical address translations. The address translation table for such a system is, in effect, a massive global data structure shared by all nodes. Translation lookaside buffers are caches specific to this data structure which must be kept coherent. This approach therefore has significant implied hardware complexity, and is inherently limited in its scalability.

From a hardware designer’s point of view, it is clearly much more appealing to use bits in the virtual address to locate home nodes. Address translation is then a local rather than a global mechanism, and no longer affects the scalability of the system. A minor objection is that this requires remote nodes to perform “extra work” to service remote memory requests. In reality, however, the same amount of work is being performed; it is simply being performed in a different (and in our opinion more natural) location. A more serious objection is that distributing an object across multiple nodes requires allocating many non-contiguous blocks of the virtual address space for the single object, complicating both memory management and object references. It is this problem on which we focus our attention.

In this paper we introduce **Multistriped Addressing**, a novel mapping from virtual addresses to physical nodes which enables data from a contiguous chunk of the virtual address space to be distributed in a flexible manner without the use of global address translation tables. The node ID is directly embedded into the virtual address at a controllable offset. This has the effect of partitioning virtual memory into a number of equally sized spaces, each of which is striped across the nodes of the system using a different granularity. This method has minimal hardware overhead and provides a convenient and efficient mechanism for distributing parallel data structures.

The following section describes Multistriped Addressing in detail including hardware requirements. In section 3 we discuss the advantages and disadvantages of Multistriped Addressing in comparison to other strategies. Finally, in section 4 we conclude and outline the context in which Multistriped Addressing was developed.

2 Technical Description

The simplest way to divide a large virtual address space among the nodes of a distributed-memory

Project Aries Technical Memo ARIES-TM-03
Artificial Intelligence Laboratory
Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology
Cambridge, MA

system is to allocate a fixed set of bits within the address to indicate the node ID (Figure 1a). This has the advantage of being extremely easy to implement in hardware (the node ID can be extracted using wires alone), but has the disadvantage of forcing distributed objects to occupy multiple non-contiguous portions of the virtual address space. This presents several difficulties:

1. Allocation, deallocation, and garbage collection (if applicable) of objects is much more expensive when the objects are non-contiguous in memory.
2. For systems implementing capabilities ([Fabry74], [Carter94]), the description and implementation of a capability is complicated by the need to provide access to non-continguous addresses while simultaneously preventing access to in-between addresses.
3. Indexing arbitrary data within the object is more expensive as the bits of the index need to be rearranged before they are added to the base address of the object.

Multistriped Addressing is an alternate approach which provides a flexible means for striping contiguous chunks of the virtual address space across physical nodes in the system. The top five bits (S) of a sixty four bit virtual address are used as an index into the address itself; they specify the location of the node ID within the address (Figure 1b).

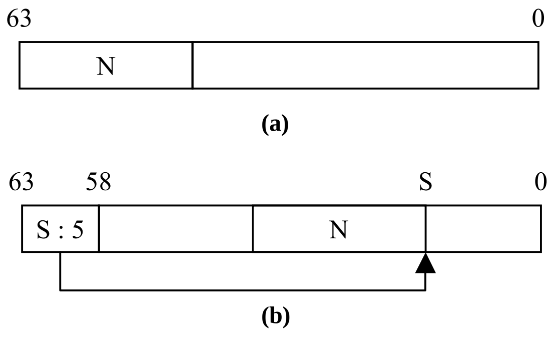

Figure 1: (a) Node ID (N) defined by a fixed set of virtual address bits. **(b)** Location of N within virtual address determined by striping granularity S.

The described format accomplishes two goals:

1. By providing a fixed mapping from virtual addresses to nodes, the need for system-wide global address translation tables is obviated.
2. Placing the node index within the address itself has the effect of striping data in a contiguous virtual address space across the nodes with a granularity of 2^S bytes.

Since S ranges from 0 to 31, data may be striped with any power-of-two granularity ranging from 1 byte to 2 Gigabytes. For systems with up to 256 million nodes, the node index N will have at most 28 bits, hence the fields S and N will never overlap. Note that the five bit field S is part of the address. The presence of this field does not reduce the size of the address space; rather it divides the 64 bit address space into 32 equally sized portions, each of which is striped across the processing nodes with a different granularity.

The only hardware required to extract node indices from addresses is a 59 bit right shifter, as shown in figure 2. Note that 59 is an upper bound on the size of the shifter; if the system supports up to 2^n nodes then a 31 + *n* bit shifter is required.

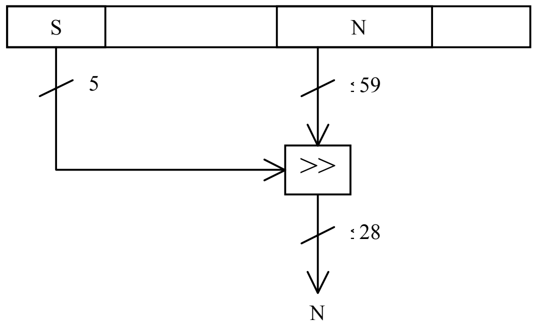

Figure 2: A 59 bit right shifter extracts the node index from the address.

2.1 Local Virtual Address

Once an address reaches a node, it is no longer desirable to have the node index in the middle of the address, as this can lead to sparsely populated pages. For example, suppose a one megabyte data structure is striped across a 256 node system at a granularity of 16 bytes. If the page size is 4096 bytes, then the addresses of this data structure which are mapped to a node N are of the form shown in Figure 3.

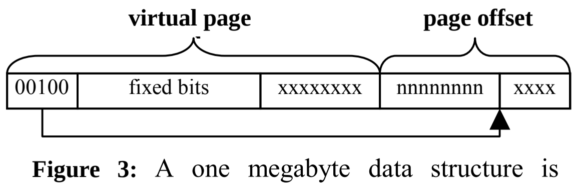

Figure 3: A one megabyte data structure is striped across 256 nodes at a granularity of 16 bytes (S = 4). The addresses mapped to a given node consist of upper bits which are constant across the data structure, eight node index bits (nnn…) embedded in the address, and twelve bits (xxx…) that vary across the data structure.

Note that the upper eight bits of the page offset are fixed (they form the node ID N), while the lower four bits of the page offset and the lower eight bits of the virtual page number vary across the data structure. As a result, the 4096 bytes of the structure which are mapped to a given node occupy exactly 16 bytes in each of 256 different pages. This leads to a rather inefficient use of physical memory.

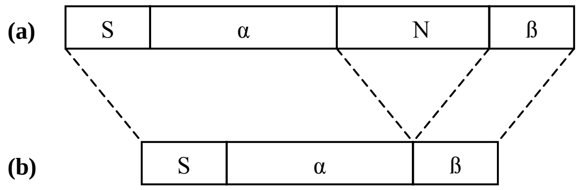

Figure 4: A local virtual address **(b)** is formed from the virtual address **(a)** by excising the node index N.

To fix this problem, we excise the node index to form a **local virtual address** as shown in Figure 4. Note that the original virtual address can be recovered using S and N, so this is a one to one mapping from virtual addresses to local virtual addresses. Continuing with the previous example, the data structure now occupies exactly one page on each node.

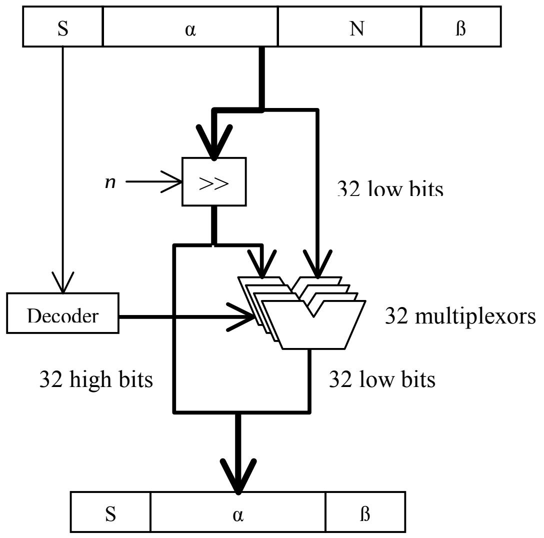

Figure 5: Hardware required to excise N. The virtual address is right shifted *n* bits. The high 32 bits of the shifted address are copied directly into the local virtual address. The low 32 bits of the shifted address are combined with the low 32 bits of the unshifted address using 32 two-input multiplexors. The multiplexors are controlled by a decoder which outputs 1's in bits 0, 1, …, S-1 and 0's in bits S, S+1, …, 31.

Performing this transformation involves selecting the appropriate bits from a shifted and unshifted version of the virtual address (in Figure 4, the bits in S and α are selected from the shifted address and the bits in ß are selected from the unshifted address). If N is *n* bits long then we need to compute an *n* bit right shift of the virtual address. If the exact size of the system (and therefore *n*) is known in advance then this can be done with wires, but it is usually desirable to be able to use the same hardware for systems with various numbers of nodes. Thus, the hardware required is a 64 bit right shifter (with only five bits of control as $n < 32$), a 32 bit decoder which generates 1's below a given index and 0's elsewhere, and 32 two-input multiplexors, as shown in Figure 5.

3 Discussion

An alternate approach to data distribution is to use the simple mapping of figure 1a and rely on coherent virtually-addressed data caches to move data to where it is needed. This approach combines the flexible demand-based data migration of a cache coherent shared memory multiprocessor (such as DASH [Lenoski92] or FLASH [Kuskin94]) with the advantages of a fixed virtual address to physical node mapping, namely the elimination of global translation tables. However, if a large shared object resides on a single home node, this node can become a system bottleneck as it must handle all cache coherence protocol messages related to the object. Thus, even cache-coherent systems can benefit from multistriped addressing as it allows such bottlenecks to be avoided.

Other striping mechanisms have been implemented in existing architectures. The M-Machine provides a global translation mechanism which allows large portions of the virtual address space to be mapped over rectilinear subsets of the system's three dimensional array of nodes [Dally94]. Translation table entries contain 37 bits which specify the location and power of two extents of the subset in all three directions, as well as the amount of data per node. The Tera Computer System [Alverson90] essentially employs a one dimensional version of this scheme, using segment tables to distribute consecutive virtual addresses in a segment among any power of two number of memory units. Both of these methods rely on a global translation mechanism and therefore incur a much greater hardware cost than multistriped addressing.

A limitation of any scheme which makes use of a fixed virtual address to physical node mapping is the inability to change a piece of data's home node. For cache coherent systems this is not so much of an issue as data migration is automatic and the role of multistriped addressing is simply to avoid bottlenecks in the coherency protocol. For systems with local cache only or even no cache at all (such as Tera [Alverson90]), inter-node data migration is more difficult as it necessitates a change in the object's virtual address. One possible solution to this problem is the use of *memory forwarding* [Luk99].

4 Conclusion

Multistriped addressing is an efficient mechanism for controlling the layout of data in large scale shared memory systems. The fixed mapping of virtual addresses to physical nodes eliminates the need for expensive global address translation mechanisms. Embedding home node IDs at controllable offsets within virtual addresses incurs small hardware costs and allows data to be striped across the system in a flexible manner.

Multistriped addressing was developed as part of an effort to find area efficient alternatives to traditional architectural mechanisms. In contrast to today's scalar architectures which make use of the entire area on a silicon die for a single monolithic processor, tomorrow's parallel architectures will place multiple processors on a single die in an attempt to maximize the overall parallelism of the system. In this domain area efficiency translates directly to performance by allowing greater parallelism at the same cost. Such a parallel architecture, which makes use of multistriped addressing, is currently under development.

References

[Alverson90] Robert Alverson, David Callahan, Daniel cummings, Brian Koblenz, Allan Perterfield, Burton Smith, "The Tera Computer System", Proc. 1990 ACM International Conference on Supercomputing, June 1990.

[Carter94] Nicholas P. Carter, Stephen W. Keckler, William J. Dally, "Hardware Support for Fast Capability-based Addressing", Proc. 6th International Conference on Architectural Support for Programming Languages and Operating Systems, 1994.

[Dally94] William J. Dally, Stephen W. Keckler, Nick Carter, Andrew Chang, Marco Fillo, Whay S. Lee, "M-Machine Architecture v1.0", MIT Concurrent VLSI Architecture Memo 58, Dept. of EECS, MIT, August 24, 1994.

[Dally98] William J. Dally, Andrew Chang, Andrew Chien, Stuart Fiske, Waldemar Horwat, John Keen, Richard Lethin, Michael Noakes, Peter Nuth, Ellen Spertus, Deborah Wallach, D. Scott Wills, "The J-Machine: A Retrospective", 25 Years of the International Symposia on Computer Architecture - Selected Papers, pp. 54-58

[Fabry74] R.S. Fabry, "Capability-Based Addressing", Communications of the ACM, Volume 17, Number 7, pp. 403-412, July 1974.

[IBM00] IBM, "Blue Gene Architecture", IBM press release, available at http://www.research.ibm.com/news/detail/architecture_fact.html

[Kuskin94] Jeffrey Kuskin, David Ofelt, Mark Heinrich, John Heinlein, Richard Simoni, Kourosh Gharachorloo, John Chapin, David Nakahira, Joel Baxter, Mark Horowitz, Anoop Gupta, Mendel Rosenblum, John Hennessy, "The Stanford FLASH Multiprocessor", Proc. ISCA '94, April 1994, pp. 302-313

[Laudon97] James Laudon, Daniel Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server", Proc. ISCA '97, pp. 241-251

[Lenoski92] Daniel Lenoski, James Laudon, Truman Joe, David Nakahira, Luis Stevens, Anoop Gupta, John Hennessy, "The DASH Prototype: Implementation and Performance"

[Luk99] Chi-Keung Luk, Todd C. Mowry, "Memory Forwarding: Enabling Aggressive Layout Optimizations by Guaranteeing the Safety of Data Relocation", Proc. ISCA '99, pp. 88-99.

[SGI] SGI, "Performance of the Cray T3ETM Multiprocessor", SGI technical white paper, http://www.sgi.com/t3e/performance.html