Design of an Interleaved Massive Memory Machine

Arvin Park
Department Computer Science
Princeton University
Princeton, New Jersey 08544

Technical Report No. 17
November 25, 1985

Abstract

This document presents a high level description of an architecture for a massive memory machine (a computer with several gigabytes of main memory). The architecture provides a logical extension of the standard three level (cache, main memory, disk) memory hierarchy to a four level scheme (massive register set, cache, main memory, disk). The design combines: the compiler optimized data motion of registers; the temporal locality advantages of caching; and the high throughput benefits of interleaving into a design which suggests a very natural physical implementation using present day packaging technologies.

This research was supported by the Defense Advanced Research Projects Agency of the Department of Defense and by the Office of Naval Research under Contracts Nos. N00014-85-C-0446 and N00014-85-K-0465, and by the National Science Foundation under Cooperative Agreement No. DCR-8420948. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

Introduction

This paper is divided into three sections. The first section describes a comprehensive design for a large memory computing system. Several features of this design involve innovative ideas that will provide interesting topics for continuing research. The second section describes a prototype two board implementation of the architecture which uses mostly off the shelf components and can be realistically implemented in within the next two years. The third section presents a simple design for a massive memory system using strictly standard components which can be implemented immediately (as soon as the PO's are sent out).

Overview of the Memory Hierarchy

The memory hierarchy for this Interleaved Massive Memory Machine (IMMM) is perhaps best described in comparison to hierarchies of existing machines. The table below contrasts the memory hierarchy of a VAX-11/780 to the new machine [1].

IMMM	VAX
registers 1K bytes 4 byte blocks	registers 64 bytes 4 byte blocks
massive registers 64 K bytes 16 byte blocks	cache 8 K bytes 8byte blocks
cache 16 M bytes 128 byte blocks	main memory 8 M bytes 512 byte blocks
main memory 4 G bytes 64 K blocks	disk 500 M bytes
disk 16 G bytes	

The registers for both machines look similar. Of course the IMMM has many more of them, but this is keeping in line with recently demonstrated performance benefits of register windowing techniques [2].

At the second level the IMMM has a set of massive registers that differs dramatically from caching on the VAX. Data motion at this level is compiler optimized while on the VAX, blocks are fetched on demand. Details of this will be discussed further.

At the third level, the 16 M byte IMMM cache doesn't differ dramatically from the main memory system of the VAX (which can be thought of as a cache for disk pages). The IMMM system is highly interleaved to support high throughput rates to the processor while the VAX only supports two way interleaving.

Fourth level memory for the VAX is 500 M bytes of disk drive while the IMMM has a 4 G byte semiconductor memory. On problems requiring random access to large data sets the VAX will thrash, and while the IMMM proceeds at full CPU speeds! Note that the disk drive for the IMMM is only

several times larger than its core memory space. Most data will permanently reside in core on the IMMM, and the disk will primarily be used as a backup device.

Massive Register Sets

The massive register set is an innovative component of the IMMM memory hierarchy. Development of this subsystem is motivated by demonstrated performance gains through the use of large numbers of registers [2][3]. The trend toward larger register sets is outlined below.

PDP 11/70	**->**	**VAX 11/780**	**->**	**RISC**	**->**	**IMMM**
32 bytes		**64 bytes**		**512 bytes**		**64 K bytes**

Present day computing systems typically maintain a cache as the 64 K byte store in the memory hierarchy while the IMMM maintains a massive register set. These two alternatives differ in one critical aspect. This is the notion of memory fetches **"On Demand"** versus **"By Command"**. In a caching system, memory blocks are pulled in as they are referenced "By Demand". In a register organization, memory blocks are pulled in "By Command". The register organization offers several important advantages.

1. **Simple Hardware**
2. **Code Density**
3. **Better Performance**

The massive register set will not require all of the complicated associative mapping hardware of a cache. So not only will the hardware be cheaper, but the memory references will be faster by eliminating associative tag lookups during each fetch. Code density will be increased by not using long pointers for most CPU operations. Only sixteen bits will be required to reference data items in the massive register set, instead of the 32 + bits of virtual address available on the massive memory system.

Perhaps the most important advantage of the massive register set is the performance gains that can be realized by using compiler optimized data motion commands. On many applications programs data motion patterns are known in advance. This knowledge can be utilized by an intelligent compiler to make efficient data motion decisions.

Register Implementation Issues

Several important issues have to be resolved before a massive register system is implemented.

1. **What do we store in the registers?**
2. **Compiler Intelligence**
3. **How are process/context switches implemented?**
4. **High Speed Block Transfers**

A judicious choice of objects to store in the registers is the most critical factor in massive register set performance. Architects of the CRISP project at Bell Labs have shown advantages in storing the top of the system stack with its associated local variables in registers [3]. The RISC project has demonstrated the advantages of storing function call state in registers [2]. These both fit into a more general theme of explicitly storing important state in registers "on command" versus retrieving this state in "on demand". The merits of storing various types of state information will have to be investigated further.

The task of identifying the critical state information is intimately tied to development of a sophisticated compiler for the system. Compilers already do this type of allocation for smaller register sets, but allocating thousands of registers may introduce incredible compiler complexity. How far can compilers be pushed?

Process and context switches usually require the operating system to save and restore the entire register set. This will simply not be practical for thousands of registers. Swapping at some level will probably be required, but not for each individual process switch. Perhaps sets of processes can be swapped in and out of the register set. The proper granularity of state to be swapped will have to be ascertained.

Because of the large amounts of state that will have to be moved during process/context switches, high speed block transfers will be a requirement to extract performance from a massive register system. Interleaving and separate I/O processors to perform high speed block transfers are being investigated.

Massive Cache

The next level of the memory hierarchy is the 16 M byte massive cache. This looks very much like a VAX's 8 M byte main memory. The VAX's main memory essentially acts as a cache for disk pages. A massive cache performs the same function for blocks brought in from the 4 G byte main memory. The basic difference between these two systems is the interleaved nature of the IMMM massive cache which allows for high rates of data throughput. The "by command" data fetch policy was not extended to this level of hierarchy because allocation is presently not well understood for the 64 K register set, and extensions of these concepts to megabytes are premature at this point.

The caching scheme maintains all of the temporal locality benefits of caching while allowing for high rates of interleaving to facilitate the following.

1. **State Swapping**
2. **Block Moves**
3. **Cold Start**

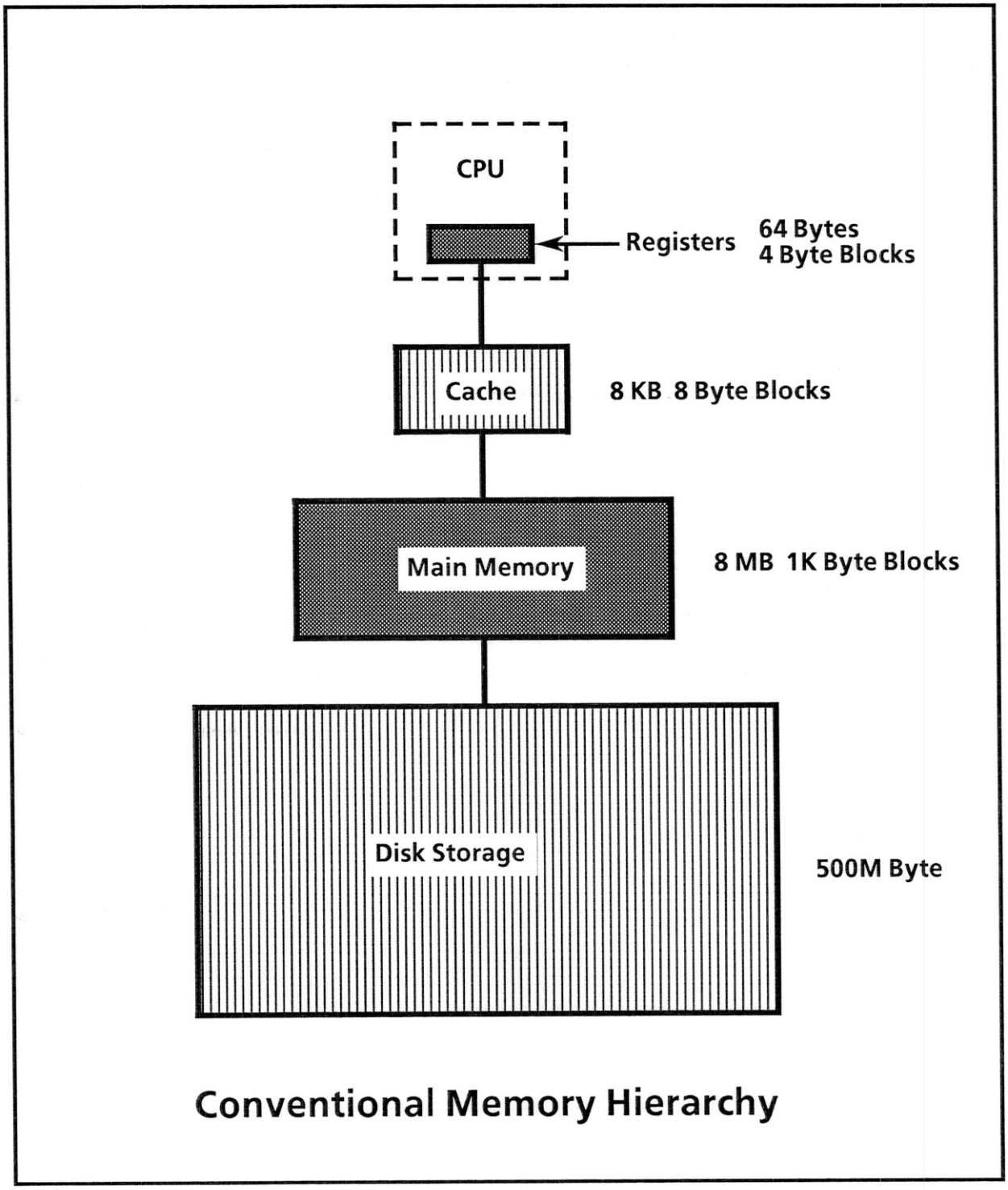
High throughput will allow fast state swapping into and out of the massive register set. This will be essential for efficient performance of the massive register system.

The interleaved organization will make possible a block move facility which bypasses the CPU. These block moves will allow rapid data transfers between buffers and files without burdening the CPU.

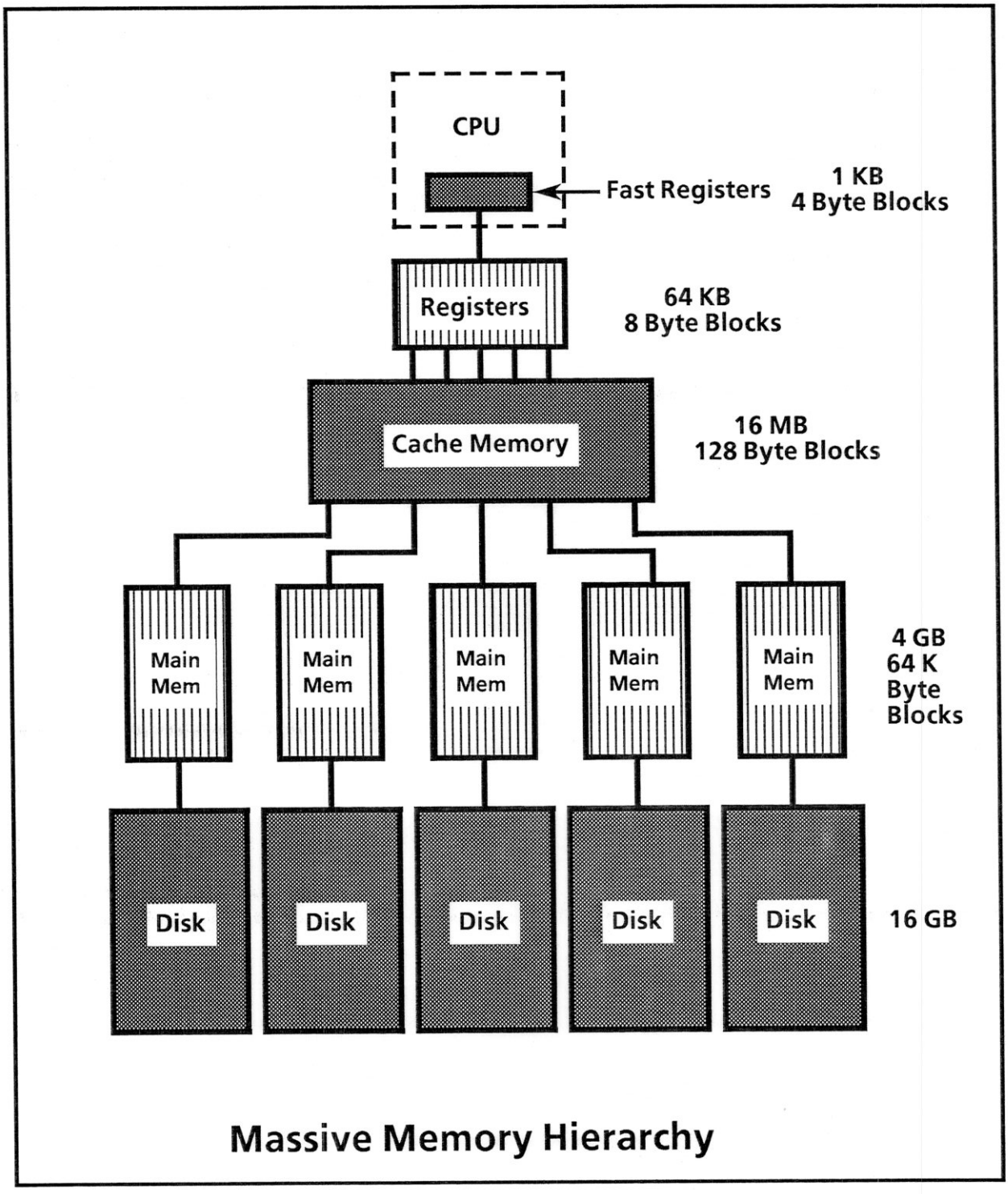
Cold start is a problem which hampers a large memory hierarchy. To start a new program/process, it will be necessary to move a lot of state from the lower levels of the memory hierarchy into the cache and massive register set. This can incur a large program initiation overhead if the hierarchy is deep enough. A high throughput interleaved organization will increase the speed of this program initiation task.

Operating System Issues

The architecture of the IMMM won't impact significantly on current operating system technologies. The most complicated part of an IMMM operating system will be the process/context state management for the massive register set. This will require non-trivial but manageable extensions of existing operating system components. Aside from process management, disk control for this interleaved system will differ greatly from UNIX control structures. Operating systems for such interleaved architectures already exist so this can be considered a solved problem.



Conventional Memory Hierarchy



Massive Memory Hierarchy

Two Board Implementation

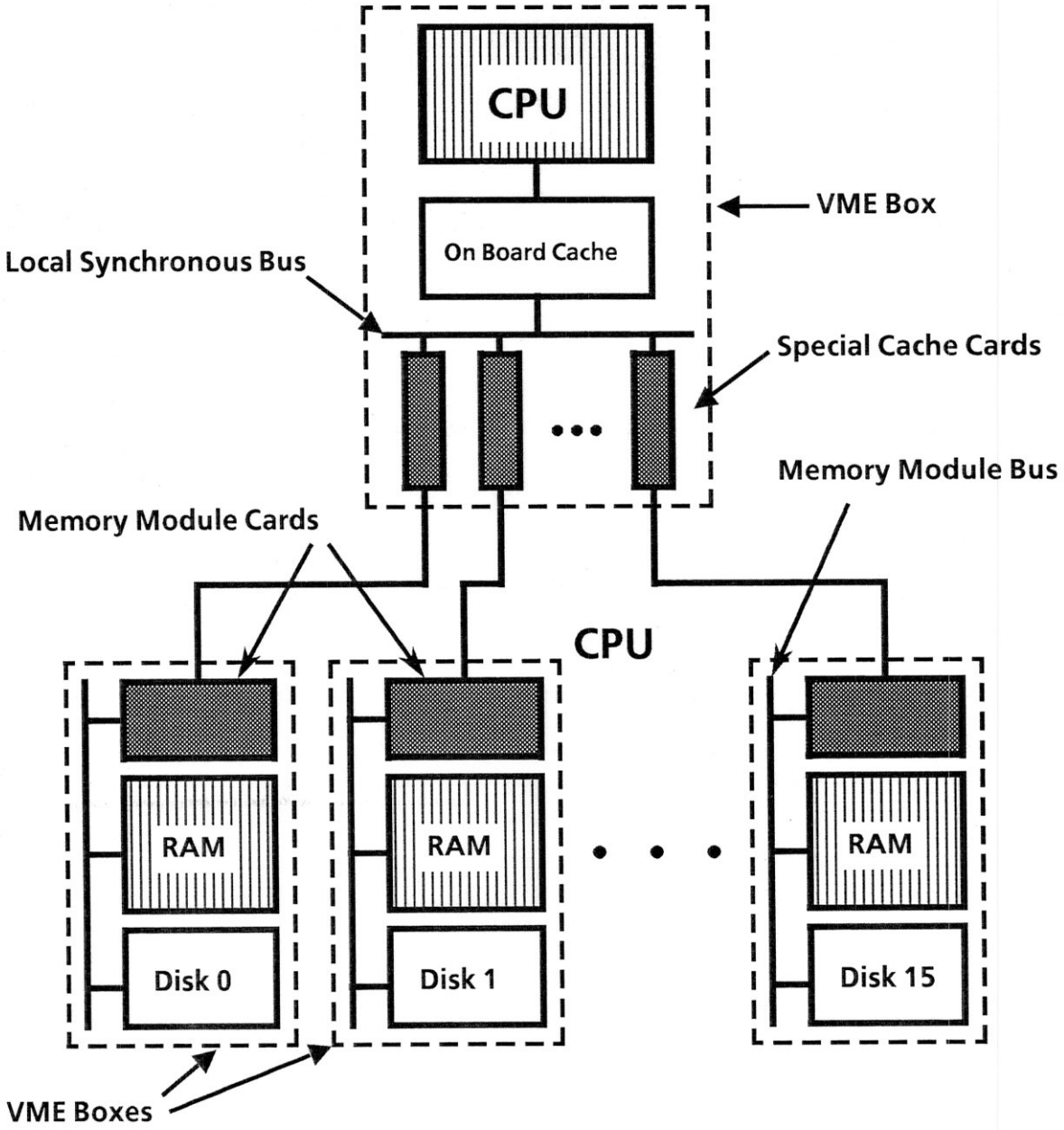
This section of the paper describes a two board implementation for the IMMM which does not include all of the features of the comprehensive design, but it can be realistically built in the next two years with manageable amounts of hardware and software development.

The design uses: VME card cages; standard CPU and memory cards; along with two custom interface boards to implement a high throughput massive memory system. The first board contains a special one M byte cache with a separate data channel to main memory. Sixteen of these boards fit in the same VME box to form a single 16 M byte cache. Each cache board connects to a custom interface board in a separate VME memory box, These separate VME boxes each contain sixteen 4 M byte Memory boards and a disk drive/DMA controller. The total memory in the sixteen secondary memory boxes is 1 G byte.

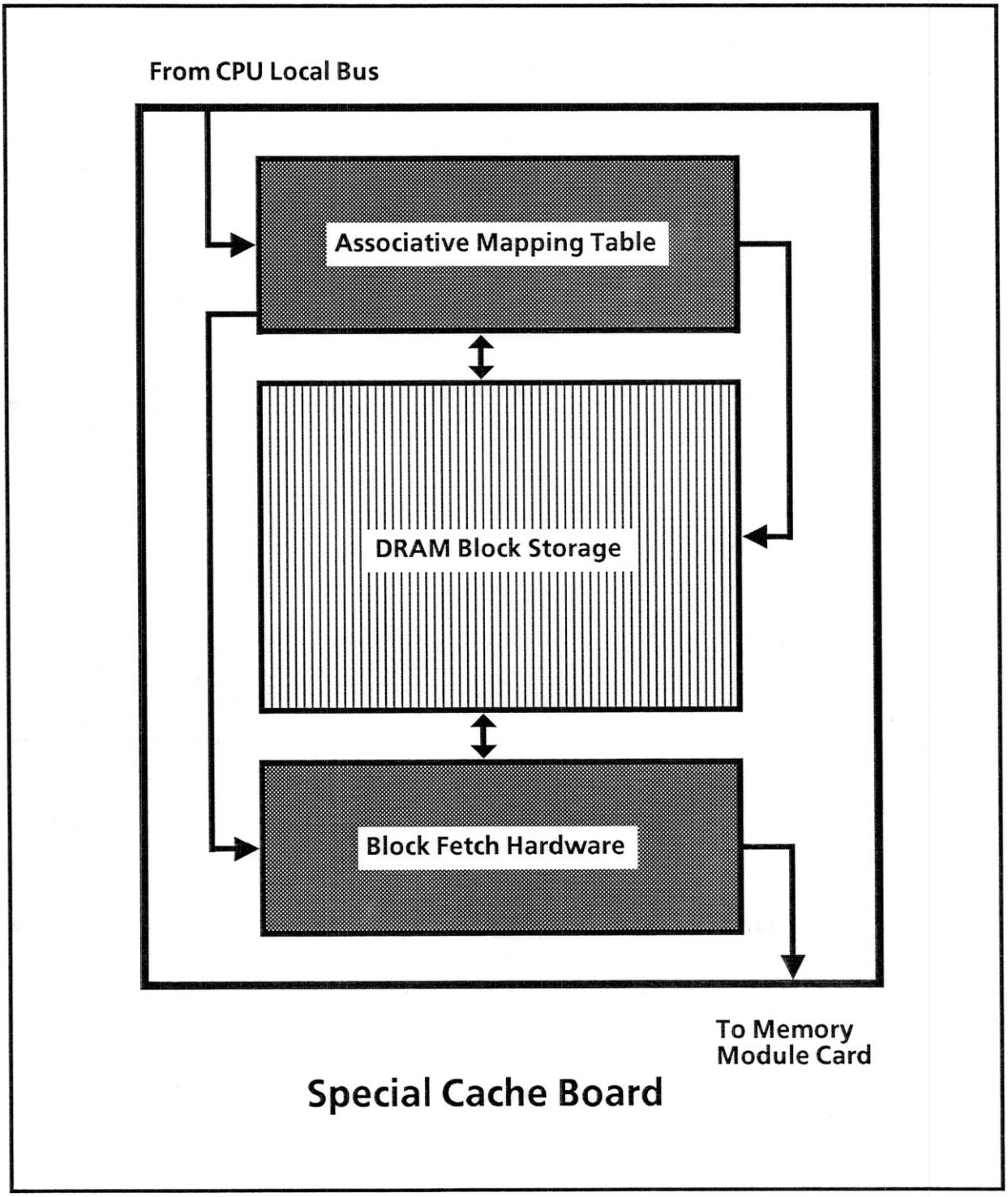
Software

Minimal operating system alterations will be necessary to port UNIX to this system. Disk drivers will have to be rewritten to support interleaved disks, and data spaces along with system buffers will have to be enlarged.

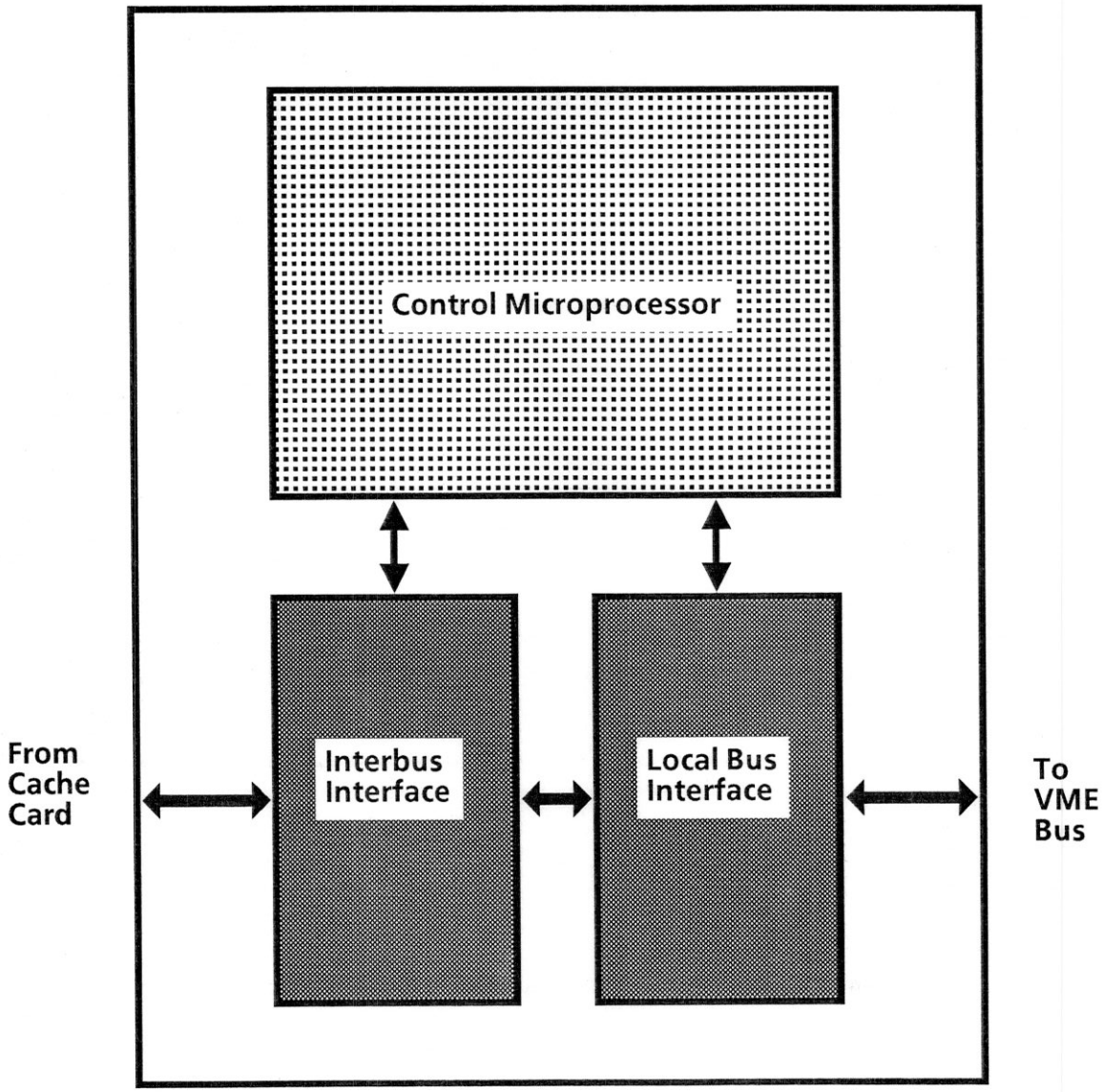
The basic organization and high level features of the Two Board implementation are illustrated in the following pages.



Interleaved Memory System



Special Cache Board



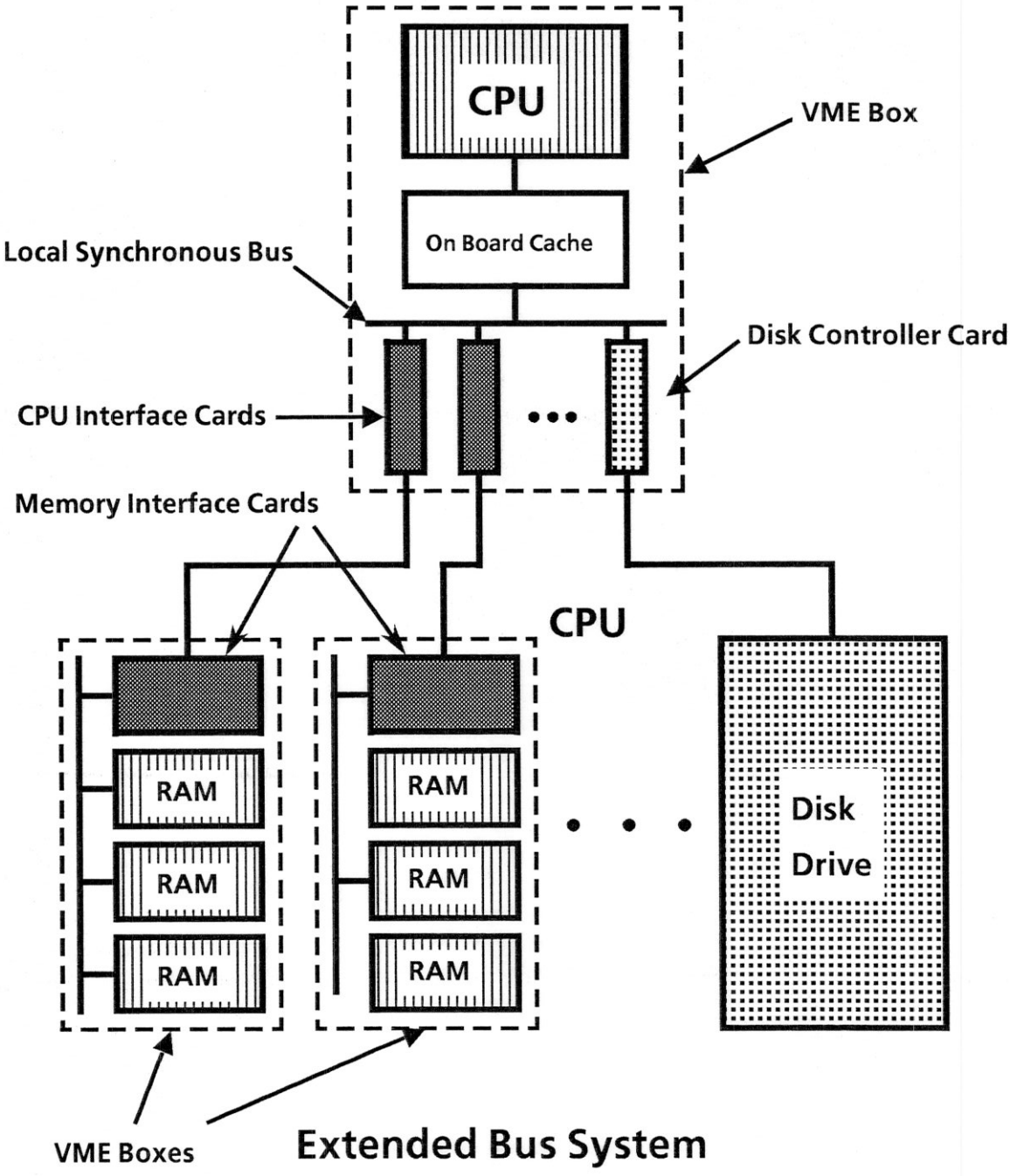
Memory Module Card

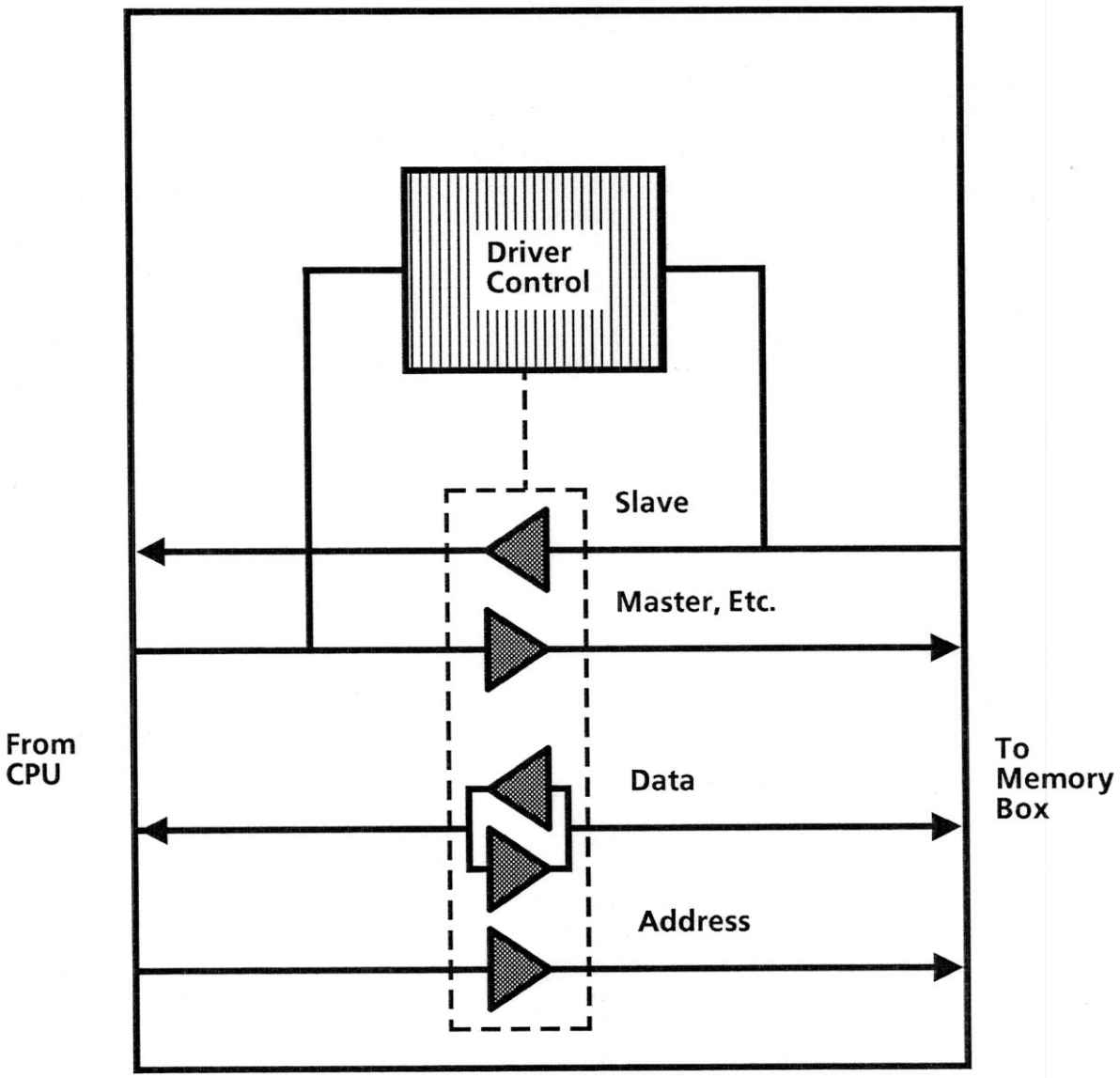
Extended Bus Design

This section of the paper describes a simple extended bus design for a massive memory system. It supports no intermediate level caching or interleaving, but is composed of strictly off the shelf components and can be implemented immediately.

The design also uses standard VME cages and memory boards. The cages are connected together with electrically buffered extender cards to form one large logical bus. No operating system modifications will be required, but a processor board with thirty two bits of address has yet to be found.

The system is illustrated in the following pages.


CPU
VME Box
On Board Cache
Local Synchronous Bus
Disk Controller Card
CPU Interface Cards
Memory Interface Cards
CPU
RAM
RAM
RAM
RAM
RAM
RAM
Disk
Drive
VME Boxes
Extended Bus System



CPU Interface Card

Conclusion

The architecture described in this paper incorporates the advantages of caching, interleaving, and large register sets into a design that can be implemented during the next two years with realistic amounts of hardware and operating system development.

References

[1] Digital Equipment Corporation, "VAX Hardware Handbook", Part IV The VAX-11/780, pp. 173-247.

[2] D. A. Patterson and R. S. Piepho, "RISC Assessment: A High-Level Language Experiment", Proceedings of the Ninth International Symposium on Computer Architecture, Austin, Texas, April 26-29, 1982.

[3] D. Ditzel, Personal communications, November, 1985