MEMORY HIERARCHY ASPECTS OF A MULTIPROCESSOR RISC: CACHE AND BUS ANALYSES

R. H. Katz, S. J. Eggers, G. A. Gibson, P. M. Hansen, M. D. Hill,
J. M. Pendleton, S. A. Ritchie, G. S. Taylor, D. A. Wood, and D. A. Patterson

Computer Science Division
Electrical Engineering and Computer Sciences Department
University of California, Berkeley
Berkeley, CA 94720

ABSTRACT: We describe the memory system design of a tightly-coupled multiprocessor. Each processor node consists of a VLSI RISC processor, a VLSI cache controller, cache data RAMs, and a standard bus. We show that adequate performance can be achieved only if the processor has an on-chip instruction buffer and a large (64KB - 256KB) local instruction and data cache. Ways of reducing the number of cache tags and the effects of various implementation alternatives for where to perform virtual memory translation are also described.

KEY WORDS AND PHRASES: RISC Architecture, Tightly Coupled Multiprocessor, Memory Hierarchy, Cache Design.

1. Introduction

VLSI Reduced Instruction Set Computers (RISCs) achieve high performance in two ways. First, they dedicate a relatively small portion of hardware resources to implementing a simple yet complete instruction set, judiciously chosen for a straightforward pipelined implementation [PATT85]. Second, they dedicate the resources thus saved to increased memory in the processor, in the form of large register files and caches.

Because of its potential for high uniprocessor performance, a VLSI RISC is an attractive component for building a multiprocessor system. We have been investigating the design issues and performance implications of a tightly coupled multiprocessor system consisting of 6 to 12 VLSI RISC processors. Because the performance of such a system is highly dependent on the interactions between the processors and the memory system, we focus on these issues. Our analyses are based on trace-driven uniprocessor simulations extrapolated to a multiprocessor system.

[1] Research supported by the SPUR Contract under the Defense Advanced Research Projects Agency's Strategic Computing Initiative Infrastructure Program.

Cache memories improve system performance by taking advantage of the principle of *locality* [DENN72]. *Temporal locality* is the property that information currently in use is also likely to be used in the near future. An example is the instructions in a program loop. *Spatial locality* is the property that items referenced closely in time are likely to be located near each other in the address space. Examples are the sequential execution of instructions and references to array elements. The design issues for a conventional processor cache revolve around (1) minimizing the cache access time, (2) maximizing the hit ratio, (3) minimizing the miss latency, and (4) minimizing the cost to make main memory consistent with an updated cache [SMIT82].

However, we must consider additional factors for a shared-memory, single-bus multiprocessor. First, each processor's cache must significantly reduce the *number* of system bus cycles needed to complete a program execution [GOOD83]. Otherwise, bus contention will limit the effective number of processors to just a few. To reduce bus traffic, the cache must be large; our results show that the cache is too large to reside on the processor chip with near-term technology. Thus we consider various multilevel caching schemes, including a small on-chip instruction buffer. Finally, the system must maintain a consistent view of memory across the distributed caches.

Figure 1.1 shows our model of a VLSI RISC multiprocessor. Each processor consists of an execution unit (E-UNIT) and an instruction unit (I-UNIT), with the latter including an instruction buffer (IB). The IB is likely to be organized as an on-chip instruction cache. Associated with each processor is a cache controller that participates in the implementation of a multiprocessor cache consistency protocol (described in a companion paper [KATZ85]), and a large cache memory. A dual-ported cache tag[2] memory has been placed on the cache controller chip. Implementing a dual-ported memory on the printed circuit board would require twice the number of conventional RAM chips. While we cannot integrate the processor, cache controller, and cache data memory on the same chip, it should be possible to implement the first two as companion chips, and to use

[2] Cache tags contain the high order address bits of the cached block, and are used to determine if an address matches an entry in the cache. Whether these are physical or virtual addresses will be deferred until Section 7.

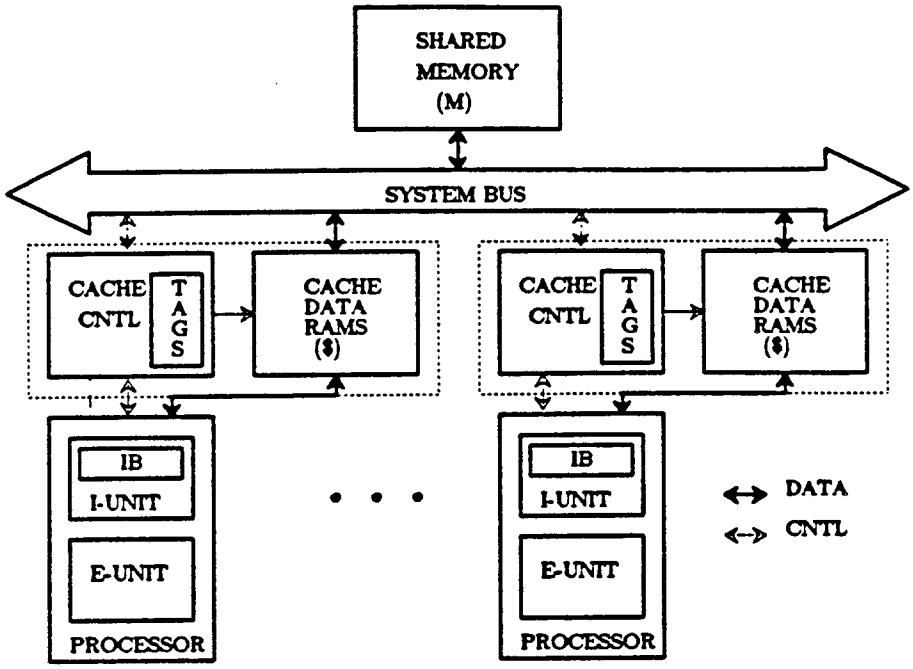

Figure 1.1 – Tightly Coupled RISC Multiprocessor

The processor is a single chip with an on-chip instruction buffer (IB). The cache controller and tag memory are integrated on another chip, with off-chip data RAMs. The caches work together to implement a cache consistency protocol, described elsewhere [KATZ85]. Shared memory and I/O devices are accessible through the system bus.

off-chip RAMs for the cache data. A system bus provides the gateway between the processor caches and the global shared memory.

The next section describes the programs we used to evaluate cache and bus behavior. Section 3 justifies the need for an on-chip instruction buffer. Section 4 describes the cache design from the processor's viewpoint. Comparisons are drawn between the cache behavior of a RISC and a VAX. Section 5 again looks at the cache design parameters, this time from the viewpoint of a realistic model of the system bus and memory system, based on the Texas Instruments 32-bit NuBus and memory boards. An overview of the evaluation of the Synapse N+1[3] Multiprocessor system is presented in section 6, as a counterpoint to the simulation studies of the previous

[3] Trademark of Synapse Computer Corporation.

sections. In section 7, we give the design alternatives and implementation implications of the location of virtual address translation. Section 8 contains our conclusions and gives our future directions.

2. Baseline Design

The experiments described in the next three sections are directing the design of the system components, in particular, the cache memory system. To perform the experiments, we must make a few assumptions. The envisioned processor will employ a load/store, register-oriented, reduced instruction set architecture. Since its detailed instruction set is still under development,

Program Name	(Language)	*RISC II* Size (KBytes) Instr	Data	Refs (Million) Instr	Data	*VAX* Size (KBytes) Instr	Data	Refs (Million) Instr	Data
QSORT	(C)	3	17	1.93	0.38	3	30	1.05	1.58
SEDX	(C)	22	48	13.98	3.08	16	60	5.32	5.07
RCOM of grep.c (RRCOMG/VRCOMG)	(C)	118	107	7.69	0.80	69	121	3.56	3.84
RCOM of puzzle+.c (RRCOMP/VRCOMP)	(C)	118	107	5.56	0.61	69	121	2.50	3.18
LISZT	(LISP)	-	-	-	-	115	922	3.21	3.89
SPICE	(F77)	-	-	-	-	308	1652	1.79	1.95
VAXIMA	(LISP)	-	-	-	-	2147	1156	7.94	9.69

Table 2.1 – RISC II and VAX Address Traces

This table lists the programs used in the following sections to evaluate cache and bus performance. QSORT is a quicksort of 2600 random numbers. SEDX is the execution of the SED stream editor replacing text in a 2007 line document. RCOM of grep.c is the execution of the RISC C-compiler compiling the grep program. The RISC execution trace is called RRCOMG; the VAX execution trace is called VRCOMG. We use that notation to label the plots in the figures below. RCOM of puzzle+.c is is the execution of the RISC C-compiler compiling the puzzle program. The RISC and VAX traces are called RRCOMP and VRCOMP, respectively. The remaining three traces have no RISC counterparts. These are the execution of SPICE (written in FORTRAN), the well-known circuit simulator, on a standard circuit benchmark, the Franz LISP compiler compiling itself, and VAXIMA (written in LISP) executing a standard symbolic manipulation benchmark. VAXIMA is a version of the MACSYMA expert system that runs on the VAX. The size figures for the VAX include statically and dynamically allocated data and stack. The size of RISC II programs include statically allocated data only. The average ratio of RISC II instructions to VAX instructions for the first four programs is 2.2:1.

Parameter	Value
Hierarchy	On-chip Instruction Buffer Off-chip Mixed Cache Shared Main Memory
Instruction Buffer Size	256 to 1K bytes
Cache Data Size	64K to 256K bytes
Cache Tag Size	1K Tags
Associativity	Direct Mapped
Information Cached	Mixed Instruction & Data
Block Size	64 to 256 bytes
Sub-block Size	32 bytes
Fetch Algorithm	Demand
Word Size	32 Bits
Address Size	32 Bits

Table 2.2 – Multiprocessor Cache Memory Parameters.
This table shows our final cache parameters based on the studies shown in the following sections.

we have assumed that its address behavior is similar to that of C programs on RISC II [KATE83] for C-like languages. To predict RISC behavior for programs written in other languages, we extrapolate from the VAX traces based on the comparative behavior of RISC and the VAX on C programs. (At present, there are no other programming systems such as FORTRAN or LISP for RISC II.)

Table 2.1 summarizes the size and program reference behavior (for RISC and VAX) for the programs used in our evaluations. Although there is one very small program (qsort), most programs are large (compilers) to very large (an algebraic manipulation system). Our cache decisions are summarized in Table 2.2. The following sections show how these parameters were derived.

3. Instruction Buffer and Cache Design

The instruction repertoire of a RISC processor is purposely kept small and simple. The cost is a larger number of executed instructions to complete a task. An analysis of the instruction

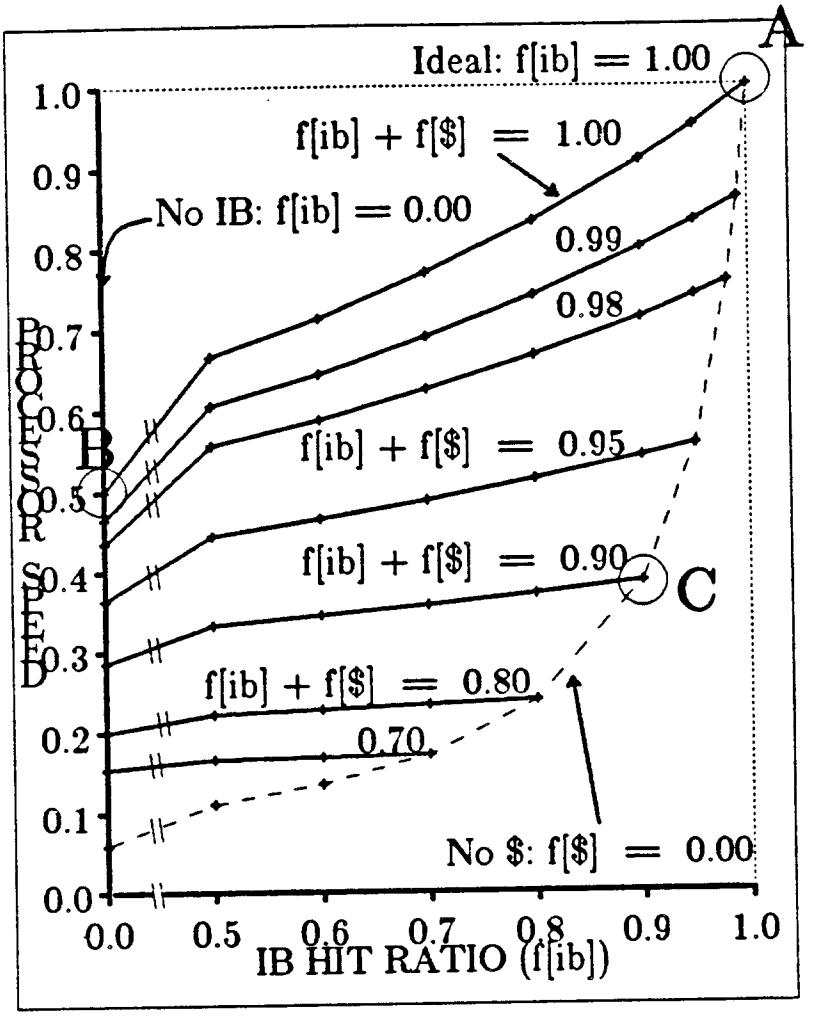

Figure 3.1 – Effective Processor Speed versus Instruction Buffer Hit Ratio

Uniprocessor speed is plotted relative to the "ideal" speed that could be achieved with all instruction references hitting in the IB (effective access time of 1.0 cycle). This point is labelled A. For each plotted line, f[IB] + f[$] = 1 - f[MM]. We assume that processor stalls are caused only by waiting for new instructions. Since other stalls (e.g., pipeline interlocks) will slow all configurations, these results indicate the *worst-case* speed degradation due to waiting for instructions. Point B shows the speed without an IB and point C shows performance without a $. This plot assumes that t[$] = 2 $\times$t t[IB] and t[MM] = 17 $\times$t t[IB].

traces of Table 2.1 indicates that on the average 2.2 RISC instructions are executed for every VAX instruction. Since it is imperative to access RISC instructions rapidly, the most effective kind of on-chip memory, in addition to the register file, is likely to be an instruction buffer.

Our memory hierarchy consists of an instruction buffer (IB), a mixed data and instruction cache ($), and a mixed data and instruction main memory (MM). *Effective access time,* t[eff], is the sum of access times (t[..]) to each level of the memory hierarchy weighted by the fraction (f[..]) of references serviced by each level:

$$t[eff] = f[IB] \times t[IB] + f[\$] \times t[\$] + f[MM] \times t[MM].$$

Since all references are serviced by one and only one level, the fractions must sum to unity:

$$f[IB] + f[\$] + f[MM] = 1.0.$$

Figure 3.1 shows relative processor speed as a function of the instruction buffer hit ratio. Scaling to processor cycles, we assume that the access time to the IB is 1 cycle and to the $ is 2 cycles. The time to access shared memory, based on the timing from the TI NuBus, is *17 cycles:* 2 cycles for a cache miss, 1 cycle to set-up the bus transaction, 1 cycle to transfer the address, 5 cycles for latency to the first data word, 7 cycles for the remaining 7 words of a 32-byte transfer block, and 1 cycle to deliver the word to the processor. Each plotted line represents a fixed miss ratio, i.e., the fraction of accesses serviced by the shared memory (f[MM]). Given an IB hit ratio (f[IB]) along the X-axis and that f[IB] + f[$] = miss ratio, then f[$] can be readily determined.

The "ideal" performance, located at the upper righthand corner of the plot (point A), is achieved when the IB hit ratio is 1.0, implying that the processor never waits for instructions. Without an instruction buffer, even if all references hit in the $, the processor speed is at best 50% of this ideal (Point B, the intersection of the f[IB] + f[$] = 1.0 curve with the y-axis, when the IB hit rate is 0).

The dashed line gives the relative processor speed for a system without a $ (i.e., f[$] = 0). Under these circumstances, effective processor speed is extremely sensitive to the IB hit ratio. Since the IB hit ratio is unlikely to exceed 0.90, processor speed without a $ is less than 40% of ideal (Point C in Figure 3.1). Thus, given our assumptions, *both an on-chip IB and local $ are required* to obtain reasonable uniprocessor performance.

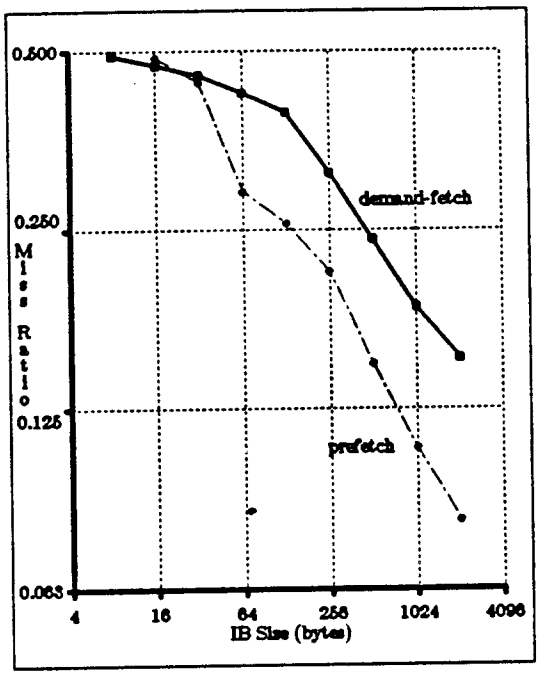

Figure 3.2 – IB Miss Ratio vs. IB Size and Prefetch Algorithm

The logarithm of the average miss ratios of simulations of four programs (VAXIMA, LISZT, SPICE, RRCOMP) is plotted vs. Instruction Buffer size. 8 byte blocks, fully-associative placement, and LRU replacement are assumed. "Demand Fetch" loads blocks only when directly referenced by the processor, and is similar in behavior to a conventional cache. "Prefetch" indicates the best-case performance of an intelligent prefetch strategy called "Tagged Prefetch" [SMIT78].

Figure 3.2 shows IB miss ratio as a function of IB size and prefetch algorithm. With an on-chip instruction buffer of between 256 and 1K bytes, we can expect miss ratios in the range of 12% – 25%, depending on the sophistication of the prefetch algorithm. Note that a significant amount of chip real estate would be necessary for the 1K byte buffer.

Referring again to Figure 3.1, if an IB hit ratio of 75% – 88% and a cache hit ratio of 98% – 99% can be achieved, then we can expect an effective processor speed of 65% – 80% of ideal.

Our conclusion from the results of this section is that we need both an on-chip instruction buffer and an off-chip data and instruction cache. The next two sections show that the cache can be designed to meet these hit ratio objectives for the buffer and cache.

4. Cache Memory Design: Single Processor Viewpoint

In this section, we examine the cache design parameters independently of the system bus. First, we look at the effects of RISC address reference behavior on the choice of cache size. Then we present a way to reduce the amount of tag memory on the VLSI cache controller without seriously affecting performance.

We have considered cache sizes of up to 256K bytes, with block sizes of 8, 16, 32, and 64 bytes, with and without sub-block placement (a cache with sub-blocks allows portions of a block to be loaded independently [HILL84]). Figure 4.1 verifies that miss ratios generally drop as cache size increases. A 64K mixed data and instruction cache yields miss ratios below 0.02 for all traces except VAXIMA. Further simulations (not shown here) confirmed that mixed caches are a better way to use the cache resource than separate instruction and data caches [SMIT82].

Figure 4.1 also reveals the importance of large caches, and especially instruction memories, for good RISC performance. The VAX makes 46% of its references to instructions and 54% to data, while RISC makes 87% instruction references and 13% data references. This is due to: (1) while the average instructions are approximately the same length (VAX: 3.8 bytes [EMER84], RISC: 4.0 bytes), RISC II executes 2.2 instructions for every VAX instruction, as already mentioned,[4] and (2) RISC's large on-chip register file results in a significant reduction in memory data accesses.

To obtain the same *number* of instruction misses for a RISC and VAX execution of the same program, the RISC cache must be larger. For example, the trace of the compiler running on the VAX (VRCOMP) has a 0.061 miss ratio in an 8K cache while the trace of the compiler running on RISC II (RRCOMP) requires approximately a 16K cache to achieve the same miss ratio. To ensure that the cache brings in enough bytes on each miss to obtain this many hits, the transfer block must be doubled along with the blocksize. For example, if the VAX cache is 16K with 16 byte blocks, there are 1024 possible blocks to miss on. The corresponding RISC cache would need

[4] Note that since the VAX I-stream referencing behavior is implementation specific [EMER84], our address trace statistics are computed as "one reference per instruction" for the VAX. All miss and traffic ratios reflect this assumption.

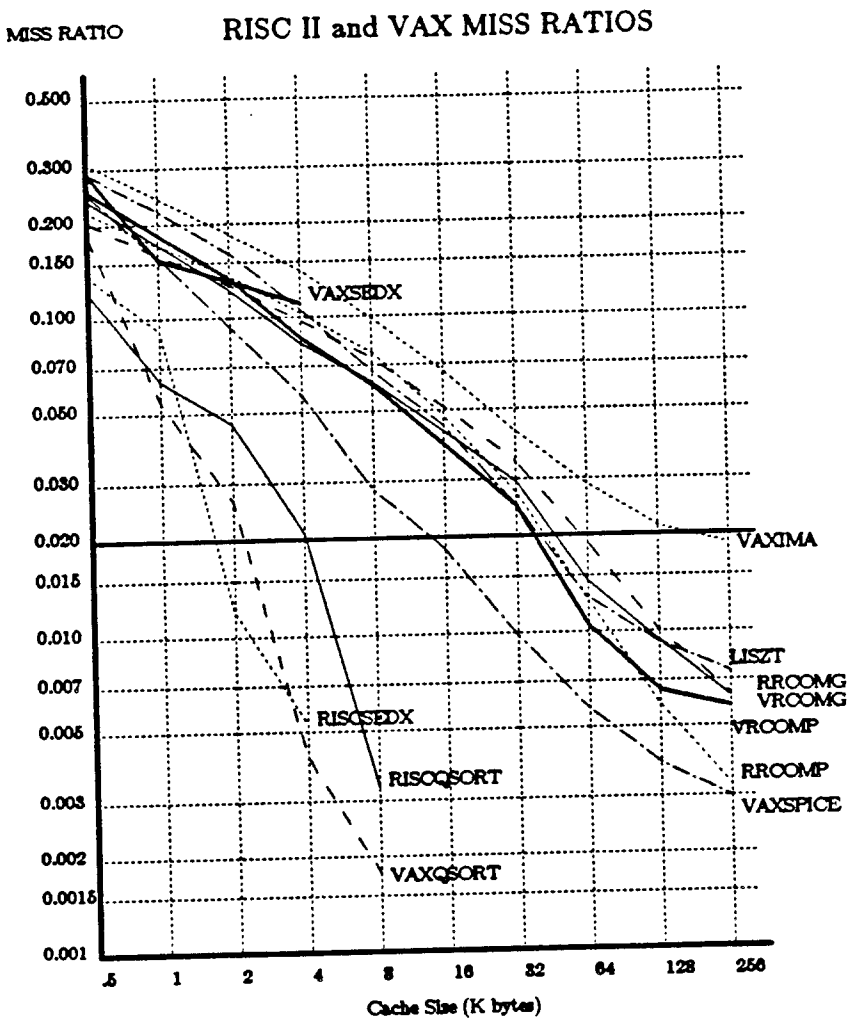

Figure 4.1 – Cache Miss Ratio as a Function of Cache Size (RISC & VAX)

Miss ratios decrease for increasing cache sizes. For all traces except VAXIMA, a 64K or larger cache results in miss ratios smaller than 0.02 (the solid horizontal line). To obtain the same miss ratios for a RISC as for a VAX, the RISC cache must be larger. For a cache size larger than 8K bytes for QSORT and larger than 4K bytes for SEDX, the traces result in miss ratios less than 0.1%, and consequently are not shown. Note the danger of basing one's analysis on small programs! SEDX and QSORT can completely fit into most of the cache sizes.

to be 32K with 32 byte blocks to obtain the same number of misses. This was verified by comparing absolute numbers of misses for the programs of Table 2.1.

Because our caches are large, the size of the tag memory on the VLSI cache controller is an important design issue. The number of address tags is equal to the cache size divided by the

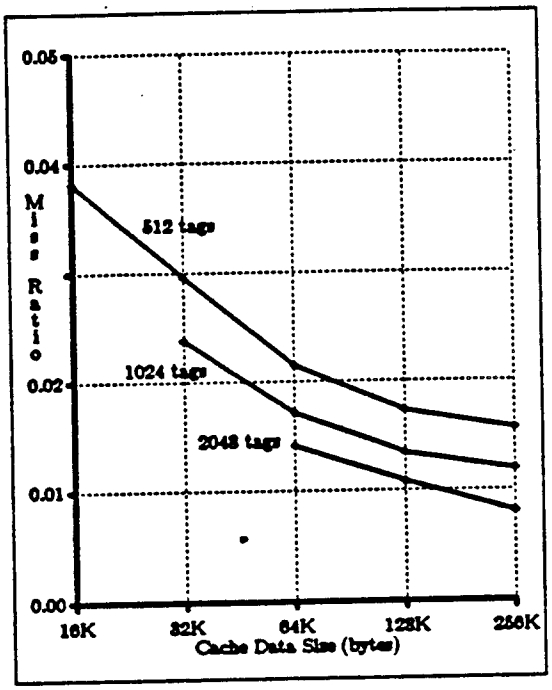

Figure 4.2 – Miss Ratios versus Cache Size and Number of Tags

The average miss ratio of five large programs is plotted as a function of cache data size for various numbers of address tags. Reducing the number of tags increases the miss ratio by a small amount, indicating that sub-block placement is worthwhile if tag memory is at a premium. The sub-block size for all caches shown is 32 bytes.

block size. In a fixed size cache, the number of tags can be reduced by increasing the block size. This is fortunate, since RISC address behavior dictates a large block size anyway. However, large block sizes increase bus latency. Both large block sizes *and* reduced latency can be achieved through *sub-block placement* [HILL84]. The sub-block size is often chosen to correspond to the original block that would have been used had the tag area not been so expensive.

Figure 4.2 plots miss ratios as a function of cache size and number of tags. Consider, for example, the 64K-byte cache with 2K address tags that has a miss ratio of 0.014. This cache has 2048 blocks of 32 bytes (64K/2K), and the block and sub-block sizes are identical. Halving the number of tags to 1024 increases the block size to 64 bytes, but only increases the miss ratio by 0.003. The figure indicates that sub-block placement can reduce the size of tag memory at a

small increase in the miss ratio.

Figures 4.3 and 4.4 further substantiate that sub-block placement is advantageous. One objective of the cache is to reduce the amount of traffic on the system bus, i.e., the number of bytes passing between the cache and memory. We quantify this traffic in terms of the *traffic ratio*, i.e., the ratio of generated bus traffic for a processor with a cache to one without a cache, executing the same workload [HILL84]. The plotted ratios have been scaled to account for the memory system's ability to perform block transfers with reduced latencies (i.e., five units of delay for the first word, one unit for each subsequent word).

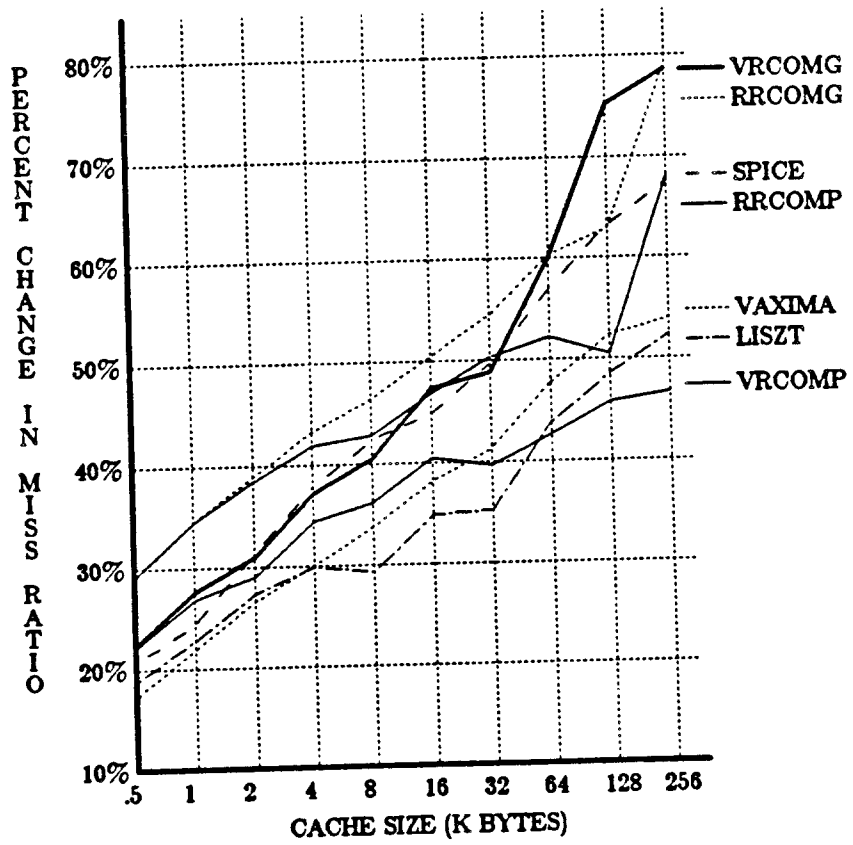

Figure 4.3 -- Miss Ratios with and without Sub-block Placement

For a 64K byte cache (direct-mapped with mixed instructions and data), halving the number of tags increases the miss ratio. However, the affect is not too significant. The miss ratios begin small (see Figure 4.1), and increasing them by less than 100% is still small.

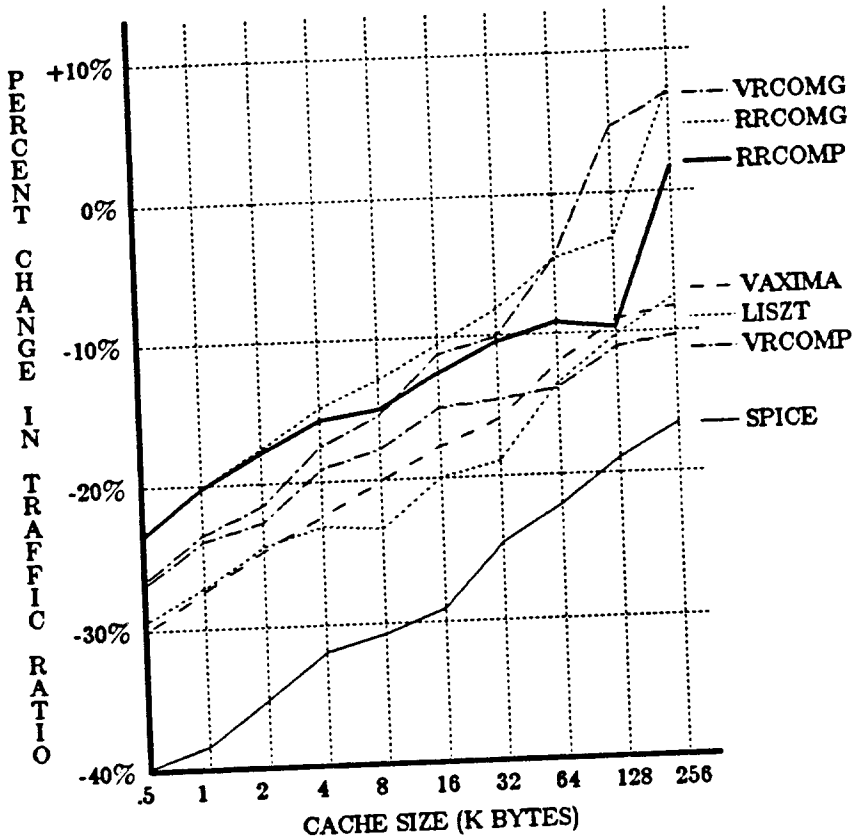

Figure 4.4 – Traffic Ratios with and without Sub-block Placement

The traffic ratio is advantageously reduced for most configurations. Once again, the magnitudes are small. For example, RRCOMG running with a 64K cache reduces the ratio from 5% (5 bytes transferred in a system with a cache for each 100 bytes transfered for one without) to 4.8% when sub-block placement is used. This is an improvement of 4%, as the plot shows.

Figure 4.3 shows the miss ratio for the largest programs listed in Table 2.1. The % increase in miss ratio is calculated by dividing the miss ratio for a cache of a given size using sub-block placement by the miss ratio for the same cache WITHOUT sub-block placement. Figure 4.4 shows the corresponding changes in traffic ratios for both RISC and VAX architectures. Although sub-block placement increases the miss ratio for a given cache size, the traffic ratio decreases, indicating that fewer bytes flow across the cache–memory interface. However, the absolute magnitudes are relatively small. By choosing two 32 bytes sub-blocks per 64 byte block, sub-block placement halves the number of tags in the VLSI cache controller while slightly decreasing

bus traffic and slightly increasing the miss ratio.

Our conclusion from the evaluations of this section is that a 64K byte cache will usually have hit rates at 98% or higher, even taking into account the larger number of instructions required for a RISC. We also plan to use sub-block placement to reduce the VLSI area for tags in the cache chip.

5. Cache Memory Design: Multiprocessor Viewpoint

In this section, an accurate simulation of a bus is used to determine the cache design. We assume a 10MHz/10 MIPS processor and the parameters of the Texas Instrument's NuBus: a 10Mhz 32-bit synchronous bus, with a fully interlocked bus protocol (bus held by the requester until the end of memory transfer), and multiplexed addresses and data. Arbitration is resolved in

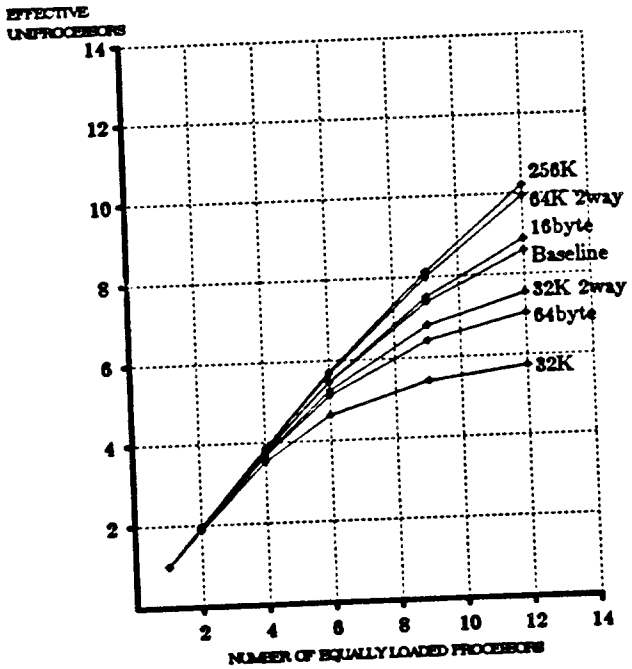

Figure 5.1 – Cache Design Tradeoffs

The number of effective uniprocessors are plotted vs. the number of equally loaded processors for a representative trace (LISZT). Baseline = 64K byte, direct mapped, mixed I & D, 32 byte transfer block.

parallel by fixed priority. The memory cycle time is 500ns (address valid to data valid) for the first word of a contiguous block, with an additional 100ns for each subsequent word.

Figure 5.1 shows the performance of the overall system for the baseline cache (64K byte cache size, 32 byte transfer blocks, direct mapped, mixed instructions and data) versus variations on the baseline as a function of equally loaded processors. The x-axis shows the number of processors executing identical workloads staggered in time. The y-axis shows the effective speed-up: the actual throughput of the multiprocessor divided by the throughput of a single processor for the same workload.

Cache size and associativity are the dominant parameters. Larger cache sizes do better, and 2-way set associativity further improves performance. We have chosen a direct mapped organization because of its implementation simplicity and potential for a reduced chip count. We compensate for the poorer performance of the direct mapped organization by increasing the size of the cache. For example, a 64K direct mapped cache achieves better performance than a 32K 2-way set associative cache.

While Figure 5.1 shows that a 16-byte block represents a small improvement over a 32-byte block (the baseline), it is hardly worth the cost of the additional sub-block validity bits that would need to be maintained on the cache controller chip. This is further substantiated in Figure 5.2. For the cache sizes of interest, the performance is relatively flat for 16- or 32-byte sub-block sizes. For a 64K cache, 32-byte sub-blocks obtains comparable performance, with graceful improvement should we later increase the cache size to 256K.

As the figures indicate, the baseline cache appears to achieve good performance for our benchmarks. Figures 5.3 and 5.4 show that for all traces except VAXIMA, the multiprocessor performance improves with added processors *even as the bus utilization approaches 80%*. This represents something of a surprise, since we did not expect to be able to drive the bus beyond 50% utilization, and still obtain reasonable performance increases as more processors are added to the system.

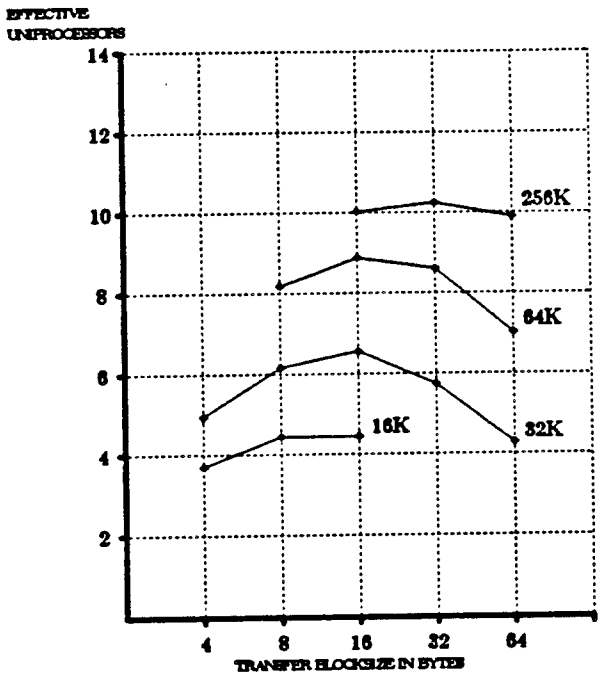

Figure 5.2 – Effect of Sub-block Size on Performance

Performance is plotted vs. fixed cache sizes and different sub-block sizes for a 12 processor system. All caches are direct mapped with mixed instruction and data. Larger caches are better. For the cache sizes of interest (64K, 256K), the performance is relatively insensitive to sub-block size. This is the LISZT trace extrapolated from the VAX to a RISC.

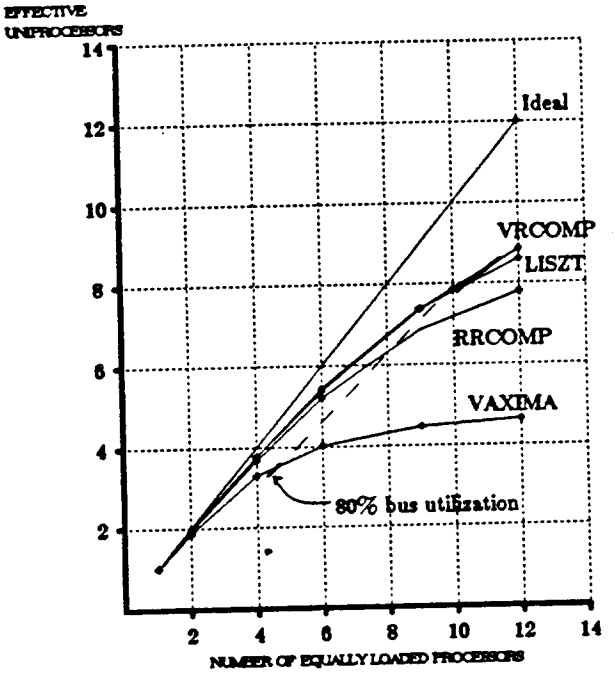

Figure 5.3 – Multiprocessor Performance of Various Traces with the Baseline Cache

The performance of various workloads are plotted. While VAXIMA is more poorly behaved, the other traces show good speed-ups as more processors are added to the system. The dashed line corresponds to the number of processors needed to load the system bus to 80% of its maximum utilization. We were surprised that the system appears to work well at this level of bus traffic.

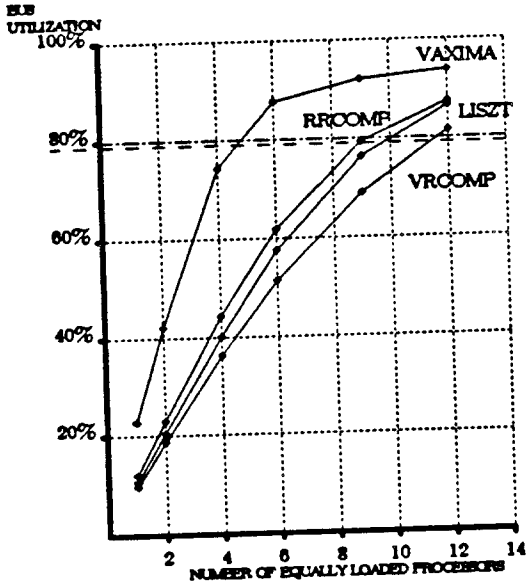

Figure 5.4 -- Bus Utilization of Various Traces with the Baseline Cache

Bus utilization is plotted for various traces and the number of equally loaded processors. While we had expected that system performance would saturate above a bus utilization of 50%, the figure indicates that work capacity continues to increase even at higher utilizations.

Our conclusion from this section is that multiprocessors with large caches and a shared bus will yield performance gains for as many as 6 to 12 processors.

6. Evaluation of an Existing Multiprocessor System

6.1. Motivation for the Experiment

The simulations of sections 4 and 5 focus on the contention for the system bus. They give no insight into the problem of contention for other resources, such as the I/O subsystem, main memory interleaving, and internal operating system data structures. In addition, contention may exist between communicating processes at the applications level. Because of the difficulty of reliably simulating these effects, we have evaluated an existing system to uncover the complexities. The results may not generalize, but they are representative of a real system.

The Synapse N+1 is a multiprocessor system developed for on-line transaction processing applications [FRAN84]. It uses a shared bus architecture, shown in Figure 6.1, that is similar to the structure of Figure 1.1. The general purpose processors (GPP) are M68000 based, with a 16K byte cache for each processor. The caches and memory controller implement an "ownership-based" cache consistency protocol, described in [FRAN84].

6.2. Methodology

The benchmark is the compilation of a large Pascal program, a task reasonably representative of a timesharing environment. Multiple copies of this program were compiled simultaneously to simulate a multiprocessing workload.

We examined memory contention and internal operating system contention as a function of the number of processors. The following were varied in the study:

(1) **Number of Processors.** The processors execute the benchmark program and most of the operating system code. Data was collected for a system configured with 1, 2, 4, 6, 8, 10, and

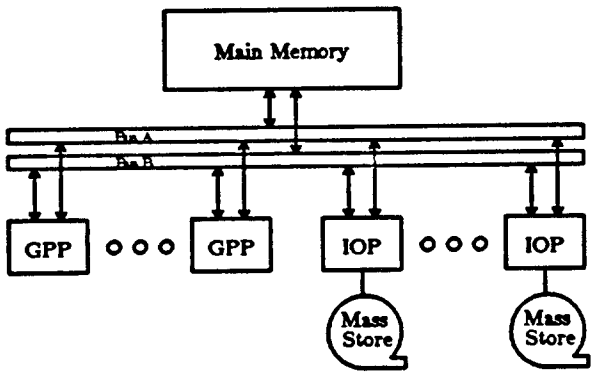

Figure 6.1 – Organization of the Synapse N+1

The Synapse system bus is a duplexed 32-bit 10 Mhz packet switched bus. The heart of the general purpose processor (GPP) is a Motorola 68000 with 16 Kbyte set-associative caches. The system supports up to 16 Mbytes of main memory on four memory controllers. I/O processors (IOP) are also 68000 based, but have 512 Kbytes of local memory in which part of the operating system resides.

12 processors.

(2) **Main Memory Size.** Memory was varied from 8M to 16M bytes in increments of 2M bytes. The degree of memory subsystem interleaving was kept constant throughout the experiments. The operating system itself requires nearly 3M bytes and is always resident. Also, the system is swap-based rather than page-based, so all code and process data (not file system pages) are either completely resident or swapped to disk.

(3) **Concurrent Processes.** The number of concurrently running processes was varied to maintain uniform queuing behavior. To hold idle time approximately constant, there were two processes for each configured processor in all benchmarks. Disks were added to the configuration as additional processors were added, to prevent increases in I/O contention. In addition, each disk had its own controller, and each disk controller its own I/O processor, to prevent unwanted contention at these levels.

(4) **Bus Bandwidth.** The Synapse N+1 has two busses to increase bandwidth and provide redundancy. It is possible to configure the system to use only one or both independently. The bandwidth of each bus is 32M bytes/second, for an aggregate bandwidth of 64M bytes/second. Data points were collected to determine the sensitivity of the system to bus bandwidth.

6.3. Results

Figure 6.2 plots the observed data against the hypothetical "linear growth" line. We have observed 12% degradation at twelve processors for both busses. Since the observations are invariant to the available bus bandwidth, it appears that bus contention is a minor factor in this configuration. These result indicate that *software contention* can reduce performance. Our benchmark has little applications level sharing, thus a true multiprocessing application would expect to have even more contention than that shown here.

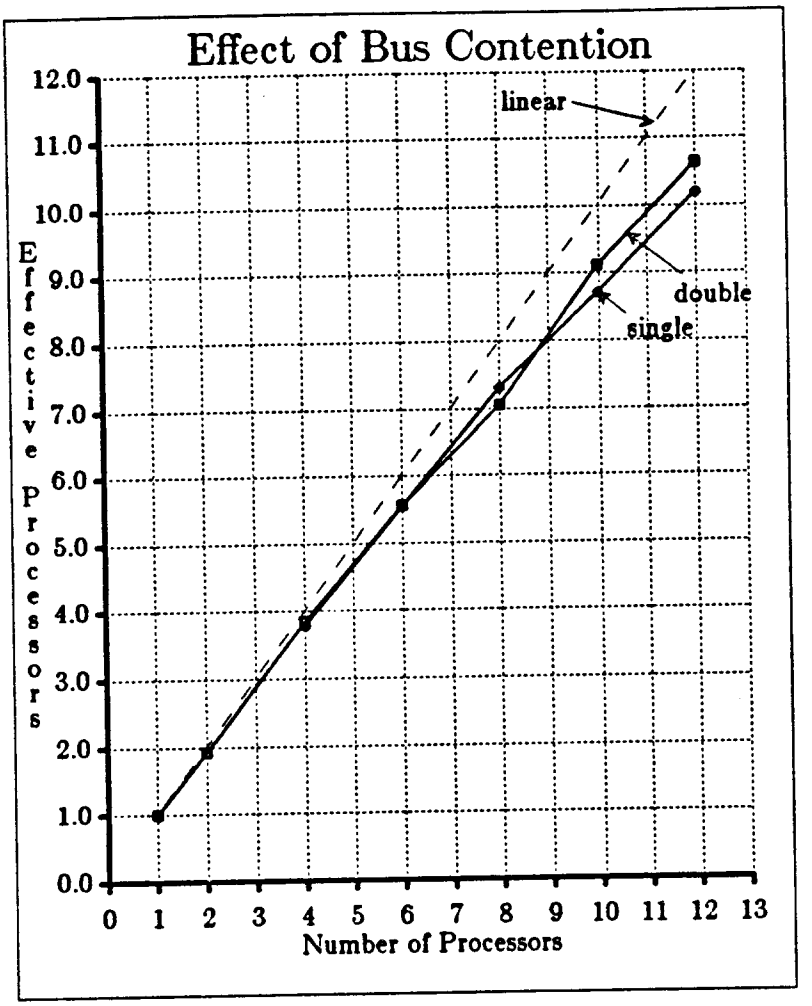

Figure 6.2 -- Effects of Bus Contention

Effective processors are plotted for a single and double bus system. The small difference between measurements for single and double bus configurations support the claim that other sources of contention are responsible.

The GPPs are relatively slow processors (.5 - 1.0 MIP), but the small caches (16K bytes) and correspondingly high miss ratios (8%) place a higher load on the bus. Synapse uses a fast bus, so almost linear speed-up is achieved nonetheless. In contrast, we estimate that each RISC processor will execute in the range of 5 - 10 MIPS, but that the system bus will have approximately 25% of the capacity of the Synapse bus (because it is neither duplexed nor uses a packet switched

protocol[5]). Thus, system performance will be more critically dependent on miss ratios, again supporting our plans for larger caches.

The second set of results, summarized in Figure 6.3, show the effects of memory contention. Performance increases with only slight degradation as long as there is sufficient memory to keep all processes running. However, when memory becomes over-subscribed and the system begins swapping, performance degrades rapidly. The point at which the rapid degradation of

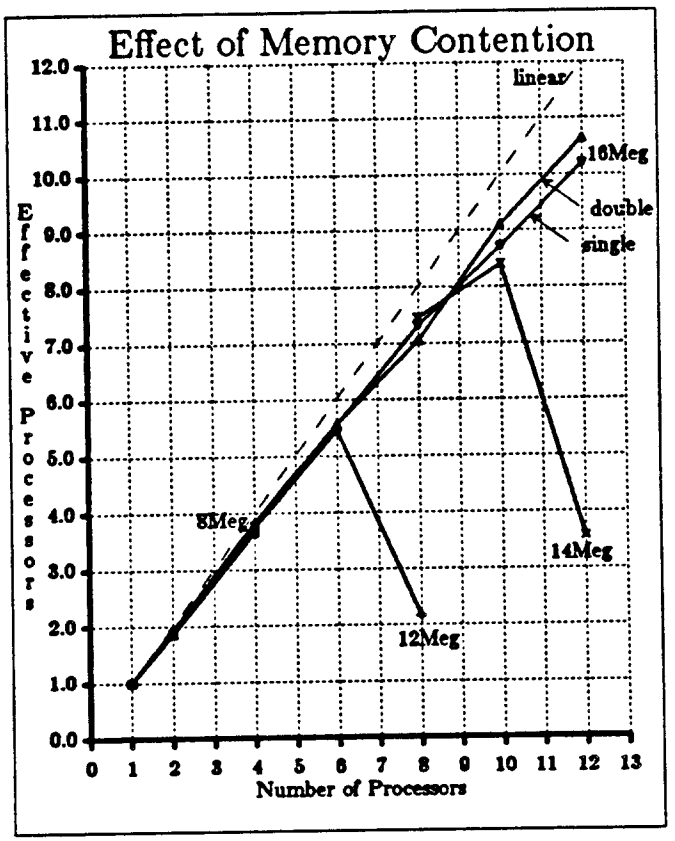

Figure 6.3 – Effects of Memory Contention

Performance grows nearly linearly until memory becomes over-subscribed. The rapidity of the degradation is due to the swapping discipline employed by the operating system. (NOTE TO REFEREES: Further data points are being collected to complete the plots and should be available by the time of paper acceptance.)

[5] In packet switched protocols, the bus is not held by the requester throughout the transfer. The server explicitly notifies the requester when the transfer is complete. Thus, more than one bus transaction can be in-progress simultaneously.

performance occurs is a function of program size and swapping discipline. We believe that a paging system would have similar behavior, but with different values and slower degradation.

Our conclusion from measuring a real multiprocessor is that linear speed-ups are possible for a shared memory multiprocessor of 6 to 12 processors. The Synapse system uses slower processors and smaller caches than we envision, and we believe that our faster RISC processor will require larger caches. We also conclude that we must work with the designers of the operating system to ensure that software contention does not become a bottleneck.

7. On The Placement Of Address Translation

Orthogonal to the decision of how to organize the cache (size, associativity, block size, number of sub-blocks, and so on) is the issue of where to perform virtual address translation. Figure 7.1 identifies five potential locations:

(1) Between the Instruction Unit and the Instruction Buffer
(2) Between the Instruction Buffer and the cache controller

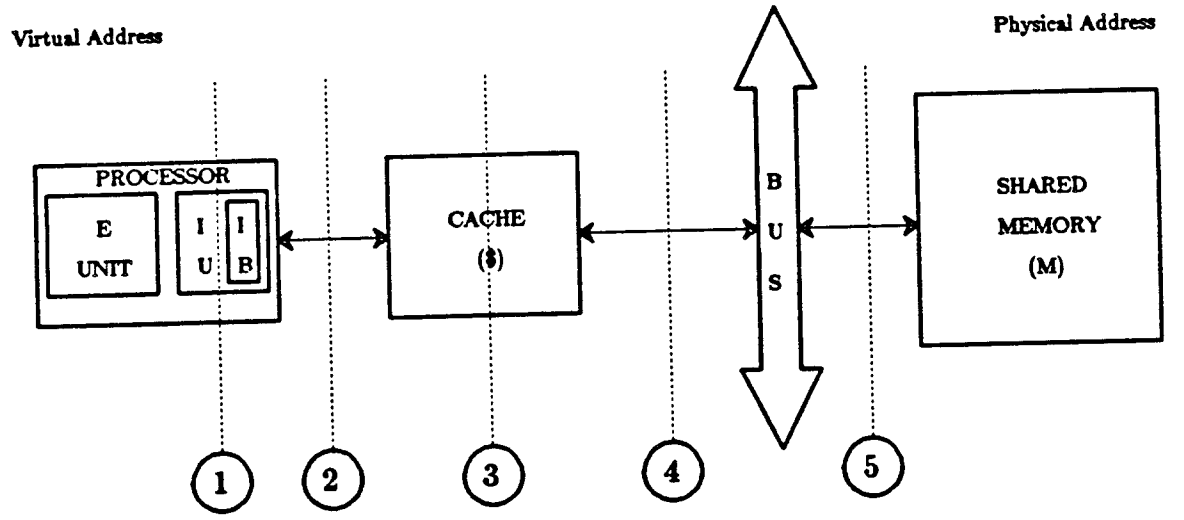

Figure 7.1 – Potential Locations for Address Translation

The potential locations for address translation are shown: (1) before the IB, (2) between the processor and the cache, (3) in parallel with the cache, (4) between the cache and the bus, and (5) between the bus and memory.

(3) In parallel with the cache controller
(4) Between the cache controller and the system bus
(5) Between the system bus and memory

Placing the translation between the Instruction Unit (IU) and the Instruction Buffer (IB) (option (1)) has significant disadvantages. An address translation is required for every reference, reducing the advantage of having the instruction memory on-chip. The IB is likely to be small enough so that the low-order bits of the virtual address can be used to directly access the buffer. Since physical addresses are not needed at this point, we cannot justify dedicating processor chip area to translation hardware.

Alternatively, the translation can be performed between the IB and the cache (option (2)). (This is the approach taken in the VAX-11/780 and 750 [DEC 79]). It allows the IB to be accessed without translation; however, data references and IB misses still incur the overhead of translation. Both this and the preceding option require doing the translation in series with the cache access. To keep the time for a cache reference to a minimum, we reject both alternatives.

Rather than doing the translation in series with a cache access, the third alternative is to do it in parallel (option (3)). Such a scheme reduces the cost of a cache reference, potentially to a single RAM cycle. The tag memory and the translation buffer are accessed with the virtual address in parallel. The resulting tag and physical page number are then compared to determine if the cache has a hit. This is the approach taken on many of the IBM 370 mainframes [SMIT82].

Since the cache is now referenced from the processor with *virtual* addresses, two complications are introduced: synonyms and the need for reverse translations. *Synonyms* arise when more than one virtual address refers to the same physical address. This complicates cache consistency because there is no longer a straightforward mapping between virtual and physical addresses. For example, it is possible to have multiple copies of the same memory block stored in a cache with different virtual addresses. *Reverse translation* occurs when a physical address on the bus must be translated back into a virtual address to access the cache from the memory side. Special mechanisms are required to perform this mapping, such as reverse translation buffers or a

fully-associative organization for the cache tags.

A way to simplify reverse translation is to restrict the placement of virtual pages in physical page frames (see Figure 7.2). The placement is chosen so that virtual and physical addresses coincide on a sufficiently large number of the low order bits to allow direct access to the cache[6]. The cache may now be referenced from either the virtual or physical "side" with no need for a reverse mapping.

Reverse translations can be avoided entirely by placing both virtual and physical addresses on the bus. The caches are addressed from the system bus side by virtual addresses; physical addresses are used to access main memory. This requires either a wider address bus or time multiplexing the addresses.

In the preceding three options, the translation must be performed on every cache reference. For high performance, this requires that the translation mechanism be very fast. A good deal of hardware and design effort must be spent to keep the mapping time down to one RAM access.

The fourth alternative performs translation only on a cache miss (option (4)). This is attractive since misses constitute a small percentage of all references. Thus, a slower mechanism implemented with less hardware can achieve the same effective access time as the previous, more costly mechanisms. The need for reverse translations is still present and requires the same mechanisms as discussed for option (3).

The Xerox Dragon [MCCR84], a VLSI-based multiprocessor system with a similar architecture as that described here, does its address translations only in the event of a cache miss. Reverse translation is handled by storing both physical and virtual page numbers for each block in the tag memory, and by implementing a fully associative lookup from the system bus side. Even on a miss, a translation is not necessarily required: if the referenced word is on the same virtual page as some other word already in the cache, translation is avoided by using the physical page number stored with that word's block. This is an elegant solution, but requires a fully associative

[6] IBM mainframes obtain the same effect by increasing cache associativity so that the page mapping is not restricted.

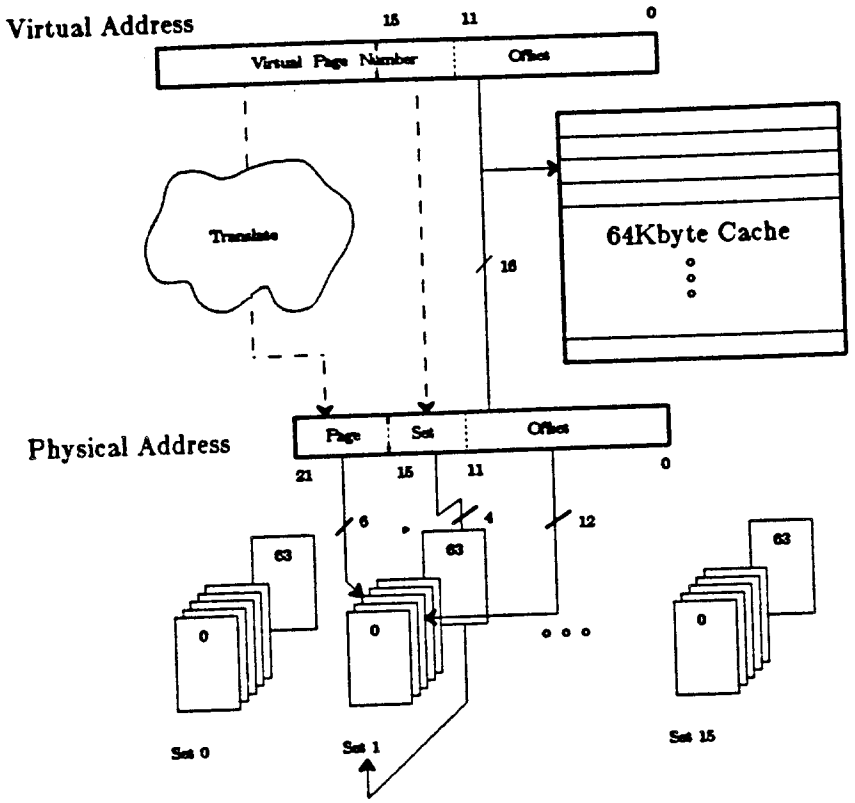

Figure 7.2 – Set-Associative Page Placement in Main Memory

If virtual and physical addresses are constrained to match on enough of the low-order bits, it becomes possible to map either address into the same cache locations. However, this restricts a physical page frame to hold only virtual pages whose addresses coincide on these low order bits. For example, a 64K byte (16 address bits) direct-mapped cache requires that the low order 16 bits of the physical and virtual address match. Assuming a 4MB main memory (22 address bits) with 4K byte pages, virtual pages can be placed into one of 16 sets (selected by address<15:12>) of 64 physical pages (selected by address<21:16>). This means that there are only 64 possible page frames for each virtual page, rather than 1024 (the total number of physical page frames).

lookup and roughly twice the memory for cache tags. Both of these costs severely limit the amount of cache that can be provided on a single VLSI chip.

The final option is to do the address translation in the main memory system (option (5)); the system bus would then use only virtual addresses. This has the advantage of centralizing the mapping hardware. The contention for this hardware would be no worse than that of main memory itself. There are, however, several disadvantages. First, the bus must be wider than a

strictly physical bus in order to accommodate the larger virtual address. Second, latency to memory must increase to allow for translation. For protocols in which the bus is "held," the bus will be busy for a longer period of time per reference. Since the bus is a critical resource in a tightly coupled multiprocessor, this is likely to have a serious effect on performance[7]. Reverse translations are eliminated, but to simplify the implemention of a cache consistency protocol, writable synonyms would have to be disallowed. This approach also requires the design of custom memory and I/O controllers.

At the time of this writing, we have not made a final decision of where to do the translation, other than to rule out options (1) and (2). A virtual bus is perhaps the most elegant, but involves substantial additional hardware design for memory and I/O controllers. The combined physical/virtual address bus has many of the advantages and probably requires less hardware design. We are also considering doing translation in the cache, with restrictive page placement to simplify reverse translation. We are conducting a study to determine the performance implications of restricting page placement in this way.

8. Summary, Conclusions, and Status

We have shown that a tightly coupled multiprocessor built from RISC-like processors with a relatively slow bus requires a large local cache and an on-chip instruction buffer to obtain reasonable performance. With properly designed caches, the simulations indicate that almost linear speed-up can be achieved for 6 to 12 processors using standard busses. As the evaluation of the Synapse Multiprocessor shows, this can actually be obtained in a real multiprocessor system.

Sub-block placement appears to be a promising mechanism for reducing the size of tag memory at a slight penalty in miss ratios. If the criterion is to reduce the amount of data that is transmitted on the bus, then sub-block placement is highly desirable. This is important for RISC architectures, since our studies indicate that relatively large block sizes are needed to obtain good

[7] It might be possible to design the translation mechanism to work largely in parallel with the memory RAM access. However, we lose the advantage of being able to do it in the "leisurely" fashion discussed in option (3).

miss ratios for a RISC.

We are currently undertaking the design and VLSI implementation of the multiprocessor architecture described here. We intend to build a RISC II style pipelined processor with on-chip instruction buffer, and a companion cache controller with on-chip tag memory, in a 2 micron CMOS technology.

Thomas Caudell, Doug Clark, Manolis Katevenis, Don Oxley, Ken Pier, Alan Jay Smith, Chuck Thacker, and Vincent Wong read drafts of this paper and provided many constructive comments.

9. References

[DEC 79] Digital Equipment Corporation, *VAX-780 Hardware Handbook,* Sales Support Literature, 1979.

[DENN72] Denning, P. J., "On Modeling Program Behavior," Proc. Spring Joint Computer Conference, AFIPS Press, Arlington, Va, V 40, 1972.

[EMER84] Emer, J. S., D. W. Clark, "A Characterization of Processor Performance in the VAX-11/780," 11th Annual Symposium on Computer Architecture, Ann Arbor, MI, (June 1984).

[FRAN84] Frank, S., "Tightly Coupled Multiprocessor System Speeds Memory–Access Times," *Electronics,* (Jan 12, 1984).

[GOOD83] Goodman, J., "Using Cache Memories to Reduce Processor–Memory Traffic," Proc. 10th Annual Symposium on Computer Architecture, Stockholm, Sweden, (June 1983).

[HILL84] Hill, M. D., A. J. Smith, "Experimental Evaluation of On-chip Microprocessor Cache Memories," 11th Annual Symposium on Computer Architecture, Ann Arbor, MI, (June 1984).

[KATE83] Katevenis, M. G. H., "Reduced Instruction Set Computer Architectures for VLSI," Ph.D. Dissertation, University of California, Berkeley, (October 1983).

[KATZ85] Katz, R. H., S. Eggers, D. Wood, C. Perkins, R. Sheldon, "Implementing a Cache Consistency Protocol," Proceedings 12th Annual Symposium on Computer Architecture, Boston, MA, (June 1985).

[MCCR84] McCreight, E. M., "The DRAGON Computer System: An Early Overview," NATO Advanced Study Institute on Microarchitecture of VLSI Computers, Urbino, Italy, (July 1984).

[PATT85] Patterson, D., "Reduced Instruction Set Computers," *Communications of the ACM,* V 28, N 1, (January 1985).

[SMIT78] Smith, A. J., "Sequential Program Prefetching in Memory Hierarchies," *IEEE Computer Magazine,* V 11, N 12, (December 1978).

[SMIT82] Smith, A. J., "Cache Memories," *ACM Computing Surveys,* V 14, N 3, (September 1982).