Proceedings of the21st Annual International
Symposium on Computer Architecture,
April18-21, 1994, Chicago, IL
Computer Architecture News, Vol. 22 - April 1994

Optimal Allocation of On-chip Memory for Multiple-API Operating Systems

David Nagle, Richard Uhlig, Trevor Mudge & Stuart Sechrest

Department of Electrical Engineering and Computer Science
University of Michigan
e-mail: bassoon@eecs.umich.edu, uhlig@eecs.umich.edu

Abstract

The allocation of die area to different processor components is a central issue in the design of single-chip microprocessors. Chip area is occupied by both core execution logic, such as ALU and FPU datapaths, and memory structures, such as caches, TLBs, and write buffers. This work focuses on the allocation of die area to memory structures through a cost/benefit analysis. The cost of memory structures with different sizes and associativities is estimated by using an established area model for on-chip memory. The performance benefits of selecting a given structure are measured through a collection of methods including on-the-fly hardware monitoring, trace-driven simulation and kernel-based analysis. Special consideration is given to operating systems that support multiple application programming interfaces (APIs), a software trend that substantially affects on-chip memory allocation decisions.

Results: *Small adjustments in cache and TLB design parameters can significantly impact overall performance. Operating systems that support multiple APIs, such as Mach 3.0, increase the relative importance of on-chip instruction caches and TLBs when compared against single-API systems such as Ultrix.*

Keywords: *On-chip Memory, Cache, TLB, Multiple-API Operating System, Mach*

1 Introduction

Improvements in integrated-circuit processing technology over the past decade have enabled the inclusion of on-chip memory structures in microprocessor designs. Current microprocessors typically dedicate about half of their chip area to memory structures such as TLBs, write buffers, and caches [MReport92, MReport93].

While numerous studies have demonstrated the effectiveness of caches and TLBs in improving performance, these structures are costly because they quickly consume the scarce resource of chip area. As a result, on-chip caches tend to be rather small (4- to 16-Kbytes). Deciding how much die area to allocate to on-chip memory structures is a complex problem because there are many competing approaches to improving system performance, including multiple-issue CPU cores, floating-point units, sophisticated instruction- and data-prefetching units, etc. Additionally, because on-chip memories are relatively small, minor adjustments to design parameters can have a major impact on overall performance.

Although designers consider on-chip memory to be important enough that they will allocate as much as half of a chip's area to these structures, there seems to be little consensus as to how the area should be divided between I-caches, D-caches and TLBs (See Table 1). While some designs have equal-sized I- and D-caches, others allocate more space to either the I-cache or the D-cache and still others implement a unified cache. Similarly, TLB sizes range from 32 to 384 entries. Cache line size and associativity can also affect the size (and speed) of on-chip memory structures, but again, there is great variation among existing microprocessor designs in the selection of these parameters.

Memory allocation decisions should consider expected workloads for the processor as well as recent trends in operating systems. For example, many new operating systems, such as Mach 3.0 and Windows NT, support multiple application programming interfaces (APIs).[1] These systems, which typically implement API servers in separate address spaces and have a long execution path from the service invocation point to the actual service routines, are likely to depend more on TLBs and I-caches [Anderson91].

This paper explores the problem of on-chip memory allocation through a cost/benefit analysis. We use an area model for on-chip memories developed by Mulder, Quach and Flynn (MQF) to estimate the cost (area) of memory structures with various sizes and associativities [Mulder91]. To determine the performance benefit of different memory structures, we employ a collection of analysis tools including hardware-based monitors, trace-driven simulators and kernel-based simulators. We consider two different operating systems, Mach 3.0 and Ultrix, and a collection of applications that rely heavily on operating system services. Once the costs and benefits of different memory structures are determined, we suggest guidelines for the allocation of on-chip memory depending on the type of applications and operating systems that a processor is expected to support.

The remainder of this paper is organized as follows: Section 2 examines related work. Section 3 briefly describes our analysis tools. A discussion of the impact of the operating system on chip-area allocation decisions is included in Section 4. Cost and performance analyses, along with recommendations for chip-area allo-

This work was supported by Defense Advanced Research Projects Agency under DARPA/ARO Contract Number DAAL03-90-C-0028 and a National Science Foundation Graduate Fellowship.

1. 4.3 BSD, POSIX, VMS, MS-DOS, Windows or Macintosh OS, etc.

Processor	Die Size (mm^2)	I-cache (size, assoc, line)			D-cache (size, assoc, line)			TLB (size, assoc)		
Intel i486DX	81	8-KB	4-way		(unified)			32-U		4-way
Cyrix 486DX	148	8-KB	4-way	4-word	(unified)			32-U		4-way
Intel Pentium	296	8-KB	2-way	8-word	8KB	2-way	8-word	32-I	64-D	4-way
DEC 21064 (Alpha)	234	8-KB	1-way	8-word	8-KB	1-way	8-word	32-I	12-D	full
Hitachi HARP-1 (PA-RISC)	264	8-KB	1-way	8-word	16-KB	1-way	8-word	128-I	128-D	1-way
PowerPC 601	121	32-KB	8-way	16-word	(unified)			256-U		2-way
MIPS R4000	184	8-KB	1-way	8-word	8-KB	1-way	8-word	96-U		full
MIPS R4200	81	16-KB	1-way	8-word	8-KB	1-way	4-word	64-U		full
MIPS R4400	184	16-KB	1-way	8-word	16-KB	1-way	8-word	96-U		full
MIPS TFP	298	16-KB	1-way	8-word	16-KB	1-way	8-word	384-U		3-way
SuperSPARC (Viking)	——	20-KB	5-way	16-word	16-KB	4-way	8-word	64-U		full
MicroSPARC	225	4-KB	1-way	8-word	2-KB	1-way	4-word	32-U		full
TeraSPARC	——	4-KB	1-way	8-word	4-KB	1-way	8-word	——	——	——

Table 1: On-chip Memory in Current-generation Microprocessors

Typical parameters for current on-chip memory structures. These data were collected from a variety of processor data books and issues of Microprocessor Report during the past two years [MReport 92, MReport93]. Throughout this paper we report line sizes in 4-byte words.

cation are in Section 5. Section 6 presents conclusions and suggests future work.

2 Related Work

The idea of building microprocessors with on-chip cache memories dates back to at least a decade ago, when VLSI advances began to enable such designs [Patterson80]. Subsequently, in two separate studies, Goodman and Hill & Smith demonstrated the effectiveness of small on-chip cache memories and argued that future processor designs should allocate some portion of die area to these structures in favor of developing more complex CPU datapaths [Goodman83, Hill84]. This work has been followed by numerous studies of on-chip memory structures, sometimes in conjunction with off-chip, second-level caches [Goodman86, Eickemeyer88, Alpert88, Short88, Przybylski89, Olukotun91, Farrens89]. The common conclusion of these papers is that on-chip memory structures are essential to minimizing off-chip memory accesses which, in turn, enables the low cycle times of modern processors.

Early studies used simple models of on-chip memory area costs, or neglect this factor entirely. The MQF model is more accurate. It takes into account data bits, tag bits and overhead logic (drivers, comparators and control), as well as different types of memory cells (dynamic versus static). This model has been shown to predict area within 10%[1] by comparing it against several existing on-chip memory implementations. We use the MQF model in this paper.

Many of the studies cited above fail to consider operating system references. Other work has shown that this can lead to over-optimistic predictions of miss and traffic ratios [Clark85a, Clark85b, Agarwal88, Torrellas92]. At the same time, operating systems are changing. OS researchers have argued that trends, such as support for multiple APIs, places additional pressure on existing hardware structures [Anderson91, Ousterhout89]. These claims are supported by recent work reporting that cache and TLB misses are substantially higher for new-generation operating systems like Mach 3.0 [Huck93, Chen93, Nagle93].

This paper extends previous work in several ways. We revise previous studies of on-chip memories in the context of present-day VLSI technology and through the use of the improved MQF chip-area model. We consider I-caches, D-caches and TLBs simultaneously under the same area constraints to develop specific recommendations for the amount of chip space that should be allocated to each structure. Finally, we consider the impact of operating systems and give special attention to the support of multiple APIs.

3 Experimental Methodology

Because different architectural analysis tools each have their strengths and weaknesses, we employ three complementary approaches:

- On-the-fly Hardware Monitoring
- Trace-driven Simulation
- Kernel-based Simulation

We use hardware monitoring to identify general trends of behavior among the operating systems and workloads that we considered. A DAS 9200 logic analyzer was connected to an R2000-based DECstation 3100 running the Ultrix and Mach operating systems. The logic analyzer was programmed to detect and count the different causes for processor stalls, such as I- and D-cache misses, TLB misses, write-buffer stalls, floating-point unit stalls, etc. The advantage of this method is that it non-invasively measures actual system activity without the error that usually accompanies simulation or analytical modeling due to incomplete assumptions about system operation. This system, which we call *Monster*, is described more completely in [Nagle92].

1. Earlier models were shown to be inaccurate by a factor of 2 or more.

Because hardware monitoring is unable to explore alternative design parameters, we also use trace-driven architectural simulation. We use trace sampling, a method that other researchers have shown to be accurate for cache simulation if two conditions are met:

- A sufficient number of samples, representative of the entire workload, must be collected. Each sample is used to obtain an estimator of the miss ratio during a given segment of the workload. Laha et al. report that 35 samples are usually sufficient to characterize a workload [Laha88], although other researchers report that some workloads (especially those with low miss ratios) may require as many as 100 samples to bring relative error to under 10% [Martonosi93].
- Samples must be long enough to prime the cache so that references will be known to hit or miss. This problem, which is most severe with simulations of large caches, is commonly referred to as *cold-start bias* and can introduce error in miss ratio estimators [Kessler91].

Our method satisfies both requirements. Monster was used to collect trace samples over random intervals of workload execution.[1] Fifty samples of 120- to 200-thousand references apiece were collected for each workload under both operating systems. The sample traces include multiprogramming and operating system references and total about 200 million references in all. Because we only consider small on-chip caches, cold-start bias is quickly removed by the length of samples used. Simulation results[2] using these trace samples were validated against on-the-fly hardware monitoring measurements by using cache parameters identical to those of the DECstation 3100 and by simulation using complete address traces. Error was found to be under 10%.

We also use a third approach, called kernel-based simulation, where architectural simulators are compiled into the kernel of an operating system and are active during system operation. For example, in a system with software-managed TLBs, all TLB misses trap into the operating system kernel. By modifying the kernel to pass these TLB miss events to a TLB simulator, it is possible to simulate alternative TLB configurations. Our kernel-based simulator, called *Tapeworm*, is described more completely in [Uhlig93]. Extensions to this basic technique are used to simulate instruction and data caches [Uhlig94]. Kernel-based simulation is an attractive compliment to trace-driven simulation because it is significantly faster, though less flexible.[3] Tapeworm's speed makes it possible to obtain performance figures without resorting to sampling. Kernel-based simulation results were compared against those obtained through trace-driven simulation to give an added measure of confidence to our results.

Throughout the paper we use the benchmarks listed in Table 2. The same benchmark binaries were run under both the Ultrix and Mach operating systems and all idle loop activity was removed from the experiments. This workload suite was chosen because it relies significantly on operating system services and emphasizes digital media applications.

Benchmark	Description
IOzone	A sequential file I/O benchmark that writes and then reads a 10 Megabyte file. Written by Bill Norcott.
jpeg_play	The xloadimage program written by Jim Frost. Displays four JPEG images.
mab	John Ousterhout's Modified Andrew Benchmark [Ousterhout89].
mpeg_play	mpeg_play V2.0 from the Berkeley Plateau Research Group. Displays 610 frames from a compressed video file [Patel92].
ousterhout	John Ousterhout's benchmark suite from [Ousterhout89].
video_play	A modified version of mpeg_play that displays 610 frames from an uncompressed video file.

Operating System	Description
Ultrix	Version 3.1 from Digital Equipment Corporation.
Mach	Carnegie Mellon University's version mk77 of the kernel and version uk38 of the 4.3 BSD UNIX server.

Table 2: Benchmarks and Operating Systems

Benchmarks were compiled with the Ultrix MIPS C compiler version 2.1 (level 2 optimization). Inputs were tuned so that each benchmark takes approximately the same amount of time to run (100-200 seconds under Mach).

4 The Effect of Multiple-APIs on Memory Allocation Decisions

Many recent operating systems (e.g., Mach [Accetta86], V [Cheriton84], Chorus [Rozier92], KeyKOS [Bomberger92], Windows NT [Custer93]) have been designed to support multiple APIs. For example, there currently exist servers for 4.3 BSD UNIX, MS-DOS, Macintosh OS and VMS that run on the Mach 3.0 microkernel [Black92, Malan91, Wiecek92]. These API services are typically implemented in one or more user-level programs and invoked through a remote procedure call (RPC) interface. Figure 1 depicts this situation for Mach, alongside the more traditional structure of a single-API system like Ultrix in which OS services reside mostly in the kernel.

To see why these OS trends might affect on-chip memory allocation decisions, consider Table 3 which shows three collections of cycle-per-instruction (CPI) measurements for mpeg_play on a machine with *off-chip* caches. The measurements in the first row do not include OS references, but the last two give CPI and stall breakdowns when Ultrix or Mach system code is included in the measurement.

The results of Row 1 in Table 3 would lead a chip designer to focus primarily on the write buffer and execution units, because the simulation indicates that these CPU components contribute to over 70% of all stall cycles. However, when OS effects are considered, the situation changes. An Ultrix-based system exhibits higher overall CPI and the relative importance of the I-cache, and especially the D-cache increases. A Mach-based system has an

1. Monster connects to the system at the CPU package pins. Because caches are implemented off-chip in the R2000, this method captures all memory references, not just those that miss the cache.
2. We use the cache2000 and Cheetah cache simulators [MIPS88, Sugumar93].
3. Our kernel-based TLB simulator can process memory references at a rate of over 6 million references / sec. A comparable trace-driven simulator processes at a rate of 20 to 150 thousand references / sec. On the other hand, our kernel-based cache simulator design restricts selection of line sizes and write policies.

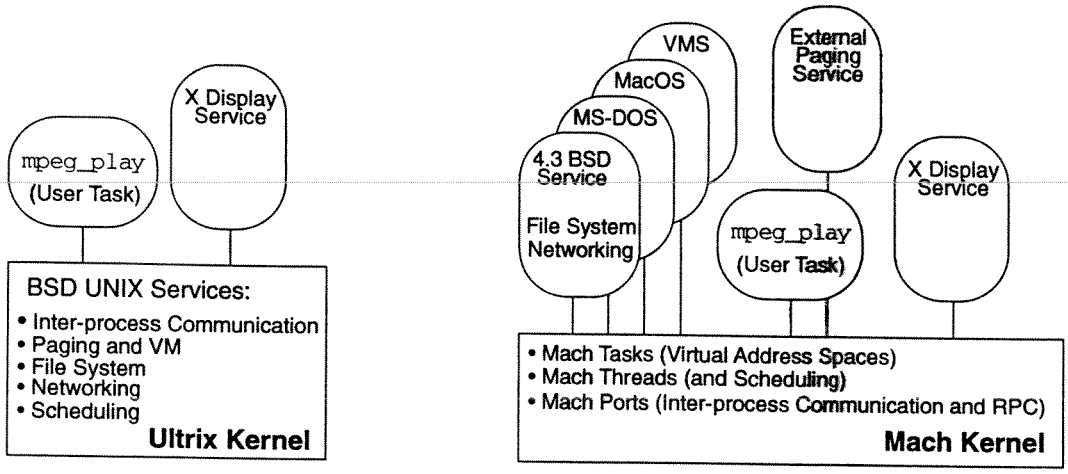

Figure 1: Structure of Ultrix and Mach

This Figure shows differences in the structure of Ultrix and Mach. In Ultrix, most system services reside in the kernel and are accessed through a system-call interface. In Mach, much of the system code runs at the user level and interacts via Mach messages or RPCs. Each of the API servers depicted above (BSD, MS-DOS, MacOS and VMS) exist in actual implementations [Black92, Malan91, Wiecek92], though not currently all on the same processor architecture.

Operating System	Measurement Method	CPI	TLB		I-cache		D-cache		Write Buffer		Other	
None	pixie + cache2000	1.43	0.01	(1%)	0.06	(14%)	0.05	(13%)	0.18	(41%)	0.14	(32%)
Ultrix	Monster	1.66	0.01	(2%)	0.10	(15%)	0.26	(39%)	0.14	(21%)	0.15	(23%)
Mach	Monster	2.06	0.15	(14%)	0.32	(30%)	0.30	(28%)	0.21	(20%)	0.08	(8%)

Table 3: The Effect of Operating Systems on CPU Stall Behavior

This table shows three collections of CPU stall measurements for the `mpeg_play` workload running on a DECstation 3100 with 64-Kbyte off-chip, direct-mapped instruction and data caches and a 64-entry, fully-associative TLB. The cache line size is one word. The different columns show the total CPI and the contributions of different system components to CPI increases above 1.0 (this is a single-instruction issue machine). Numbers in parenthesis give the relative contribution of each stall type. *Other* stands for non-memory related stalls, such as integer and floating-point interlock cycles.

In the first row, CPI was determined by a `cache2000` simulation driven by `pixie`-generated traces [MIPS88]. `cache2000` was configured to simulate a memory system with the same parameters as the DECstation 3100. Because `pixie` generates user-only references, this measurement does not consider operating system references. The last two rows give CPI under both Ultrix and Mach as measured by direct monitoring of the DECstation hardware, using Monster.

even higher CPI with a substantial increase in TLB and I-cache misses.

Why do measurements of the same workload running on the same hardware yield such different relative stall breakdowns? The error in the first row is due to the omission of OS activity in the simulation. The measurements are in error because only 40% of the total workload activity is considered[1] and interference effects between the different processes that participate in the workload are not taken into account.

The difference between the last two measurements is not a form of error; rather, it is due to differences in the structure of the Ultrix and Mach operating systems. Consider Figure 2 which illustrates the different ways that OS services are invoked under Ultrix and Mach. The `mpeg_play` process reads a compressed MPEG video stream from the file system, decompresses the frames, and sends these individual frames to the X display server to be viewed. File services are accessed through the BSD 4.3 file system interface, while X display services are implemented through the `Xlib` library which uses the BSD socket interface. In Ultrix, the file system and socket interfaces are implemented in the kernel, but in Mach they are implemented in the user-level BSD server[2]. This results in a dramatically different execution path for the invocation of OS services. In Ultrix, services are invoked through a single kernel system call trap to the code that performs the service (step (a) in Figure 2). The kernel returns by copying results directly back into the user address space and switching control from the kernel stack back to the user stack (step (b)). In Mach, UNIX system call traps must be converted into RPCs to the BSD server. This conversion is performed by an *emulation library* that is dynamically mapped into the address space of each UNIX process. The Mach kernel detects system calls that require emulation (1) and bounces them back to the emulation library (2) which marshals arguments into the form of an RPC (3) and sends this message through the kernel to the BSD

1. For `mpeg_play`, the remaining 60% of workload time is spent in the kernel (25%), in the user-level BSD server (30%) and in the user-level X display server (5%).

2. The X11 windowing system has been rewritten to use Mach IPC and VM sharing facilities instead of the BSD socket interface [Ginsberg93].

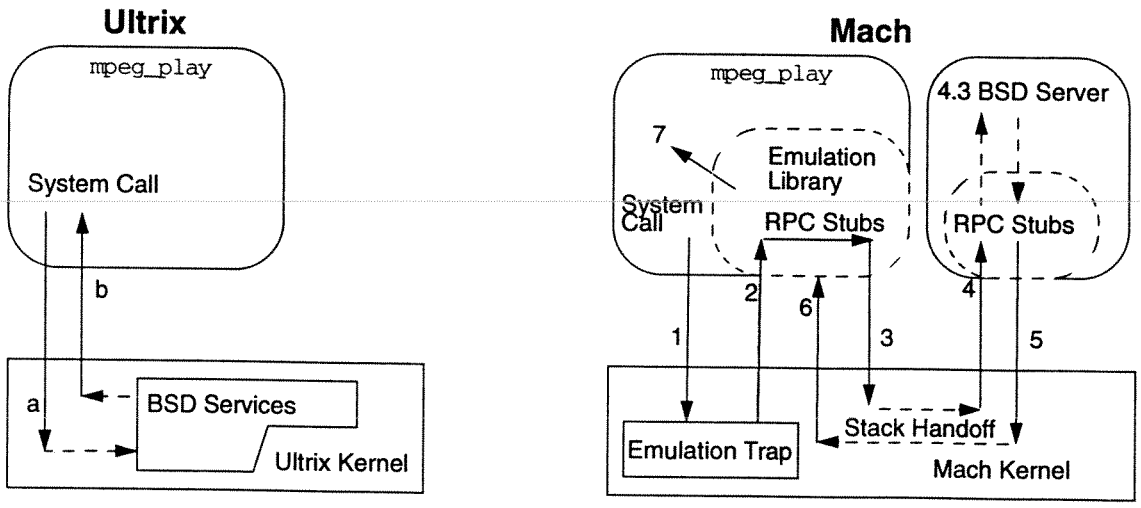

Figure 2: Service Invocation Paths in Ultrix and Mach

In Ultrix, BSD UNIX services reside in the kernel and are accessed through a single system call trap. In Mach, services reside in a user-level BSD server accessed via a remote procedure call (RPC) mechanism that passes through the Mach kernel.

server (4) which unpacks the arguments using its own RPC stub code. These steps (1-4) are equivalent to the single kernel trap (a) in Ultrix. After the BSD server performs the service, the results are sent in a reply message through the kernel (5) back to the emulation library (6) which then returns to the `mpeg_play` code. These steps (5-7) are equivalent to step (b) in Ultrix.

4.1 Reasons for Increased I-cache Misses

The longer invocation path to services in Mach is largely responsible for the relative increase in I-cache misses when compared with Ultrix. Our measurements indicate that the round-trip call and return path to services in Ultrix (steps (a) and (b) in Figure 2) is less than 100 instructions. In contrast, the call path (1-4) under Mach consists of approximately 1000 instructions and the return path (5-7) uses about 850 instructions. Note that these are just *service invocation* paths. The actual OS code that provides the service must still execute in both Ultrix and Mach, but differences with respect to this service code are minor because both systems are derived from the same 4.2 BSD code [Chen93]. Using the instruction path lengths above, and assuming 4 bytes per instruction, we see that Mach increases the code path to OS services by approximately 4 K-bytes of instruction memory and the return path by about 3 K-bytes. These are long paths relative to the typical size of on-chip instruction caches in today's microprocessors. For example, a single system call under Mach will completely overrun a 4-Kbyte on-chip I-cache on the path to the BSD server, which will then have to warm the I-cache to run well. The return path overruns the I-cache as well, leaving the calling user task with a cold I-cache. Our measurements show that roughly one third of the time spent in the kernel during `mpeg_play` is due to the send and receive messages that compose RPCs, so this case happens frequently enough to have a substantial impact on overall I-cache performance.

Other structural differences are responsible for increased I-cache misses under Mach. For example, in Ultrix, paging is implemented in the kernel, but Mach supports an external pager, running in user mode, that is responsible for locating pages in a backing store after a page fault [Draves91]. Similarly, recent versions of Mach have migrated I/O device drivers from the kernel to the user level [Forin91]. As a third example, Black et al. have described efforts to further decompose monolithic API servers (like the BSD server shown in Figure 2) into multiple, small-granularity servers (e.g., for naming, authentication, and file access) [Black92]. Each of these restructuring trends spreads-out system code and further increases instruction path lengths between software modules.

4.2 Reasons for Increased TLB Misses

The increase in TLB misses is also related to structural differences between Mach and Ultrix. The migration of OS services from kernel space (where they can run unmapped) to the user-level (where they must run mapped) increases the overall number of page table entries (PTEs) that must be held by the TLB and thus increases the TLB miss ratio. Decomposing OS services so that they reside in separate address spaces also increases the number of page tables and thus the total number of kernel PTEs held by the TLB. Mach's aggressive use of virtual memory sharing is also responsible for increased pressure on the TLB. These and other explanations for increased TLB miss ratios under Mach are documented more completely in [Nagle93] and [Anderson91].

4.3 Common Trends

Table 4 and Figure 3 show that higher CPIs and the increased reliance on the I-cache and TLB under Mach is common to all of the workloads that we considered. Recent work by Chen and Bershad corroborates this observation, although their data show a less pronounced shift to TLB stalls [Chen93]. This discrepancy may be due to differences between their workload suite and the benchmarks considered in this study.

It should be noted that the long path to system services under Mach is *not* a case of poor coding. This service invocation mechanism is common to other modular, object-oriented software systems (e.g. [Khalidi92]) and simply represents a cost for the advantages that they offer over traditional, single-service systems.

Workload	Operating System	CPI	TLB		I-cache		D-cache		Write Buffer		Other	
mpeg_play	Ultrix	1.66	0.01	(2%)	0.10	(15%)	0.26	(39%)	0.14	(21%)	0.15	(23%)
	Mach	2.06	0.15	(14%)	0.32	(30%)	0.30	(28%)	0.21	(20%)	0.08	(8%)
mab	Ultrix	1.88	0.02	(2%)	0.18	(21%)	0.38	(43%)	0.26	(29%)	0.04	(5%)
	Mach	2.13	0.12	(11%)	0.48	(42%)	0.28	(25%)	0.21	(19%)	0.04	(4%)
jpeg_play	Ultrix	1.31	0.00	(0%)	0.02	(7%)	0.13	(42%)	0.06	(19%)	0.10	(32%)
	Mach	1.51	0.05	(10%)	0.08	(16%)	0.17	(33%)	0.10	(19%)	0.11	(22%)
ousterhout	Ultrix	2.19	0.00	(0%)	0.11	(9%)	0.80	(67%)	0.24	(20%)	0.04	(3%)
	Mach	2.26	0.21	(17%)	0.44	(35%)	0.27	(21%)	0.31	(24%)	0.03	(3%)
IOzone	Ultrix	2.09	0.01	(1%)	0.10	(9%)	0.71	(65%)	0.18	(17%)	0.09	(8%)
	Mach	2.25	0.17	(14%)	0.34	(27%)	0.39	(31%)	0.31	(25%)	0.04	(3%)
video_play	Ultrix	2.48	0.05	(3%)	0.35	(24%)	0.82	(56%)	0.23	(15%)	0.03	(2%)
	Mach	2.51	0.28	(19%)	0.49	(33%)	0.43	(28%)	0.27	(18%)	0.04	(2%)
Average	Ultrix	1.94	0.02	(2%)	0.14	(15%)	0.52	(55%)	0.18	(19%)	0.08	(9%)
	Mach	2.12	0.16	(14%)	0.36	(32%)	0.31	(28%)	0.23	(21%)	0.06	(5%)

Table 4: CPI Stall Components for All Workloads Considered

In fact, the Mach implementation of RPC has been highly optimized through the use of techniques such as stack-handoff scheduling and continuations [Draves91] for the common case of small messages and out-of-line (virtual memory) transfers for the expensive case of large messages [Dean91].

Bershad has suggested other ways to avoid the costs of RPC, such as pre-allocating buffers between client and server address spaces for small messages using virtual memory primitives, or migrating more OS services into the client's address space as is currently done to a limited degree with the Mach emulation library [Bershad92]. Avoiding RPCs through more aggressive virtual memory sharing, however, is likely to shift misses from the I-cache to the TLB.

Though they have their performance penalties, these OS structuring approaches offer important advantages. Dynamically-loaded emulation libraries enable binary compatibility.[1] External pagers can simplify the implementation of distributed database or network-shared-memory applications. User-level device drivers are easier to debug, port and install, and small-granularity OS servers enhance opportunities for code reuse.

4.4 Summary of Effects

In summary, the functional advantages of modular, object-oriented software systems have been extensively documented in the literature. They include enhanced code re-usability, increased fault tolerance and ease of service distribution. We accept these trends as given, but note that they shift utilization of hardware components and thus prompt a re-evaluation of hardware architectures. In particular, the importance of I-caches and TLBs seems to increase relative to D-caches according to direct hardware measurements of a machine that implements its caches *off-chip*. We now shift our focus to the implementation of these memory structures *on* a microprocessor chip.

1. Ultrix executables need not be recompiled to run under Mach.

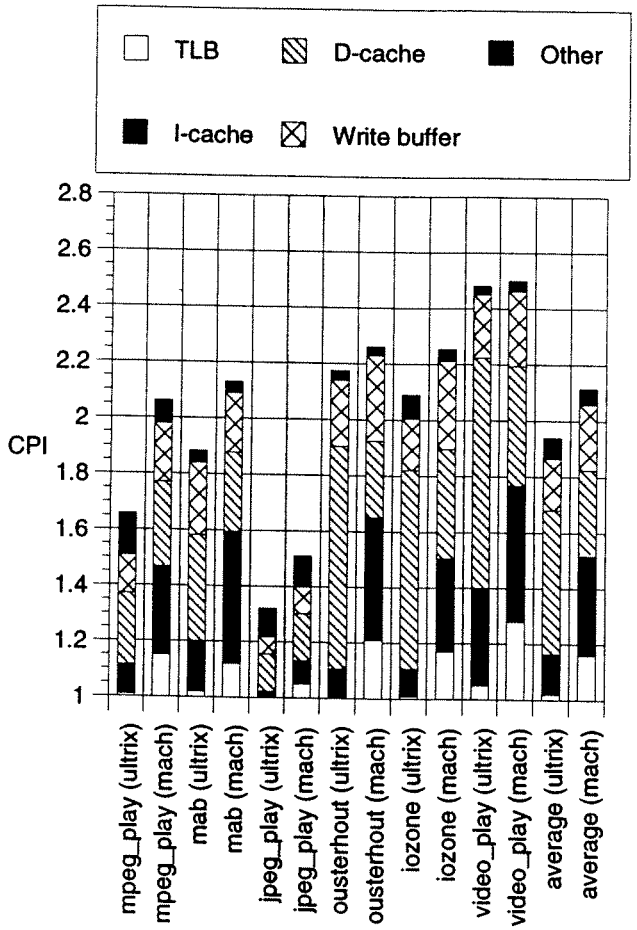

Figure 3: Components of CPI above 1.0

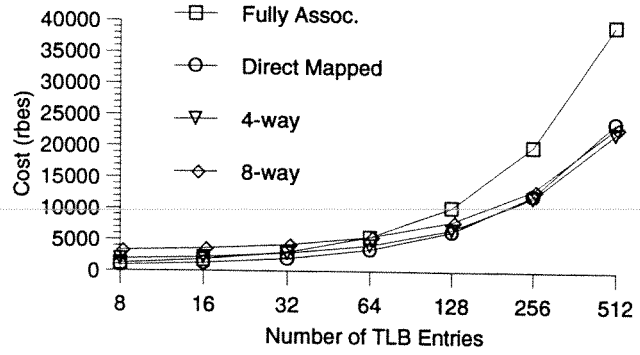

Figure 4: Area Cost for TLBs of Different Sizes and Associativities

This figure shows the size of various TLB configurations. Fully-associative TLBs require significantly more area than set-associative TLBs. For large TLBs, component sharing (sense amps, drivers, etc.) may actually make some associative structures smaller in area than direct-mapped TLBs.

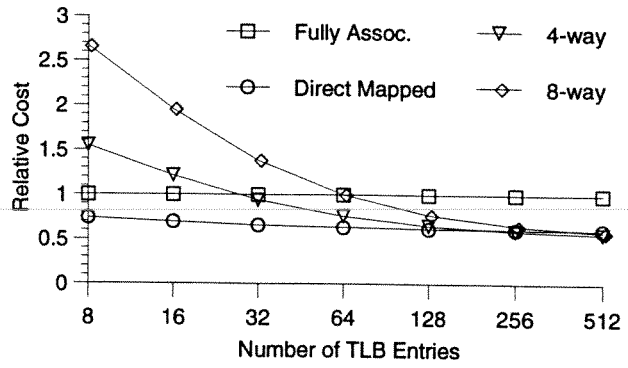

Figure 5: Area Cost of Set-associative TLBs Relative to Fully-associative TLBs

This graph plots the cost of 1-, 4- and 8-way, set-associative TLBs relative to the cost of fully-associative TLBs of the same size. For small TLBs, it is cheaper to build a fully-associative TLB than to implement 4- or 8-way associative TLBs. For larger TLBs, the trade-offs change; full associativity can cost twice as much as set associativity.

5 Cost and Benefit Analysis

5.1 Cost Analysis

To explore architectural trade-offs within the constraints of an area budget, several cost models have been developed to estimate the die area required for a given memory structure (e.g. register file, cache, TLB, write buffer) [Mulder91, Hill84, Alpert88]. This study uses the MQF model mentioned earlier [Mulder91]. The MQF model considers the memory cell type (dynamic or static), tag and data bits, organization (fully-associative, set-associative or direct-mapped), drivers and comparators to estimate die area using a technology-independent unit, the register-bit equivalent[1] (rbe). This section summarizes the costs of TLBs and caches as predicted by the MFQ model using the default parameters defined by the authors of the model.

Figure 4 graphs the area cost of TLBs with varying degrees of size and associativity. For small, set-associative TLBs (< 64 entries, 1- to 8-way set-associative), increasing the degree of associativity increases the relative die area required. A 16-entry, 8-way set-associative TLB requires 3 times the area of a 16-entry, direct-mapped TLB. For larger TLBs (> 64 entries), associativity has a much smaller impact on die area. For example, with the largest TLB size (512 entries), there is little difference in cost between a direct-mapped TLB and an 8-way, set-associative TLB.

Cost trade-offs also exist between set-associative and fully-associative TLBs (see Figure 5). Direct-mapped TLBs are always smaller than fully-associative TLBs. However, for small TLBs (< 64 entries), fully-associativity costs less than 4- or 8-way set-associativity. For TLBs with 64 or more entries, the opposite is true. In this range, a fully-associative TLB requires twice as much area as a 4- or 8-way, set-associative TLB. For example, for approximately the same cost, designers can choose either a 256-entry, fully-associative TLB or a 512-entry, 8-way TLB.

1. The rbe is defined to be the area equal to a one-bit storage cell in a register. For SRAMs and DRAMs, the storage cell is usually a fraction of an rbe.

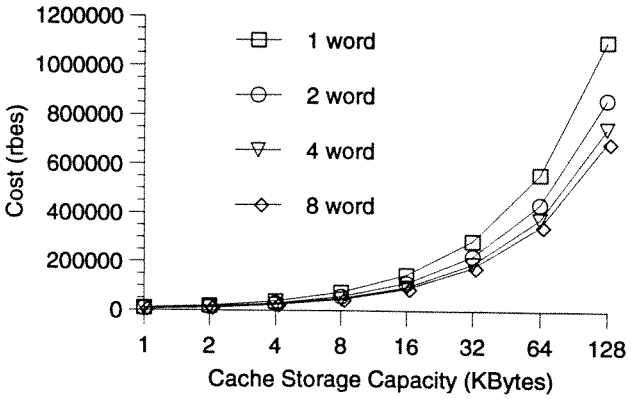

Figure 6: Area Cost for Caches of Different Capacity and Line Size

This graph plots the cost of various caches with 1-, 2-, 4- and 8-word lines. Larger line sizes reduce the cost by amortizing the cost of tag and status bits over more data bits.

For larger memory structures, such as caches, there is a different set of trade-offs. Figure 6 plots the relationship between area cost and cache organization. Larger line sizes reduce the cost of a cache by as much as 37% when moving from a 1-word line to an 8-word line size. Associativity (not pictured) has a much smaller impact on die area.

In general, the MQF model gives a good approximation of the total cost for a given memory structure. Mulder at al. compared the model's area predictions against 12 actual processor designs and found typical errors of under 10% and a maximum error of 20.1%. The authors note several limitations of their model. First, it does not consider the relationship between access time and area cost. Second, optimal layout geometry for different memory sizes cannot be completely modeled. Third, changing the aspect ratio can cause the model to underestimate actual area costs. To achieve a higher degree of accuracy, designers should use their own model to determine cost trade-offs within their specific processor technology. For the purposes of this paper, the MQF model is accurate

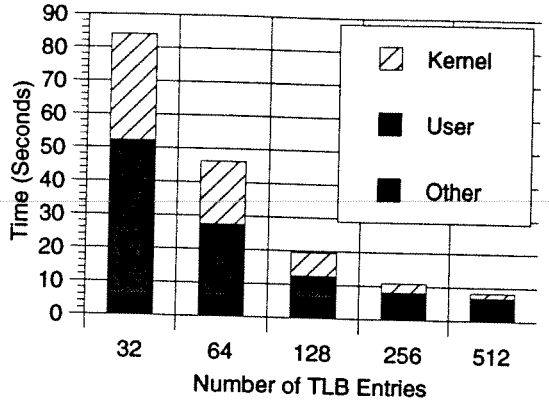

Figure 7: Total TLB Service Time vs. TLB size

This figure plots total TLB service time for fully-associative TLBs running under Mach. The 256- and 512-entry TLBs remove almost all misses except for compulsory misses. The "Other" category represents misses due to page faults and access protection violations.

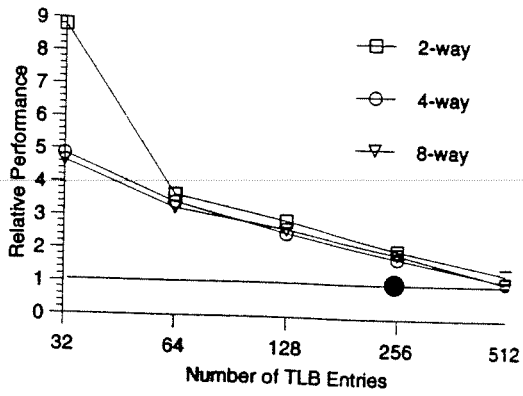

Figure 8: Set-Associative TLB Performance Relative to Fully-Associative TLBs

Performance of set-associative TLBs relative to a 256-entry, fully-associative TLB (represented as the straight line at 1). Direct-mapped TLBs exhibit very poor performance and are therefore not considered in this plot. These results are for the `video_play` benchmark running under Mach.

enough to allow us to estimate first-order cost trade-offs in on-chip memory allocation.

5.2 TLB Performance

As noted in Section 4.2, the TLB plays an important role in the overall performance of multiple-API operating systems. To examine this issue, we used Tapeworm to explore the performance of TLBs with various sizes and set-associativities. Figure 7 plots the total TLB service time for various fully-associative TLB configurations running our benchmark suite under Mach. A 64-entry, fully-associative TLB (as implemented in the MIPS R2000) requires over 46 seconds of service time. This time can be reduced to approximately 10 seconds with a 256- or 512-entry TLB. There is little to be gained beyond this size of TLB because the remaining misses are due to page faults and other compulsory (cold-start) misses.

Unfortunately, large fully-associative TLBs are difficult to build and can have excessively long access times.[1] Therefore, we examined the performance of various set-associative TLBs relative to a 256-entry, fully-associative TLB (Figure 8). For TLBs with 64 or more entries, there is little difference in performance between 2-, 4-, and 8-way set-associativity. More importantly, the 512-entry, set-associative TLBs will achieve about the same performance as a 256-entry, fully-associative TLB with little area penalty.

5.3 Cache Performance

As processor speeds continue to outstrip memory speeds, low miss ratios for on-chip caches become more critical. However, because of implementation constraints such as die size and access time, most on-chip primary caches will remain relatively small (< 64-Kbytes, see Table 1). In this section we search for cache organizations that deliver the most performance for the least cost when supporting a multiple-API operating system like Mach. We report cache performance for our entire benchmark suite under both Ultrix and Mach in terms of miss ratios and contribution to CPI.[2]

The top left graph in Figure 9 shows that under Ultrix, small on-chip I-caches have fairly low miss ratios. For example, the miss ratio for an 8K-byte I-cache with a 4-word (16-byte) line is

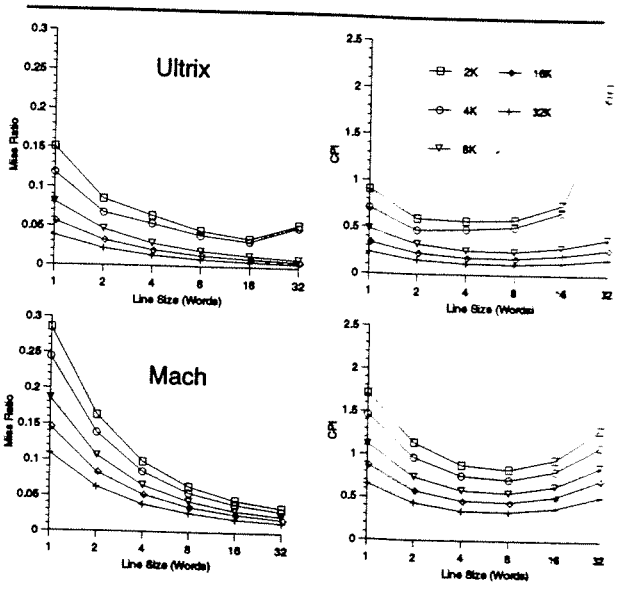

Figure 9: Instruction Cache Performance

This figure plots average I-cache miss ratios and I-cache contribution to CPI for the entire benchmark suite for various direct-mapped I-cache organizations. Notice that cache pollution due to large line sizes is more of an issue for Ultrix than for Mach.

1. To our knowledge, the largest associative, on-chip microprocessor TLB is in the HP 7100. This design has a 136 entry (120+16), fully-associative TLB.

2. The cache contribution to CPI was determined by estimating a primary cache miss penalty of 6 cycles for the first word in a line and 1 cycle for each additional word.

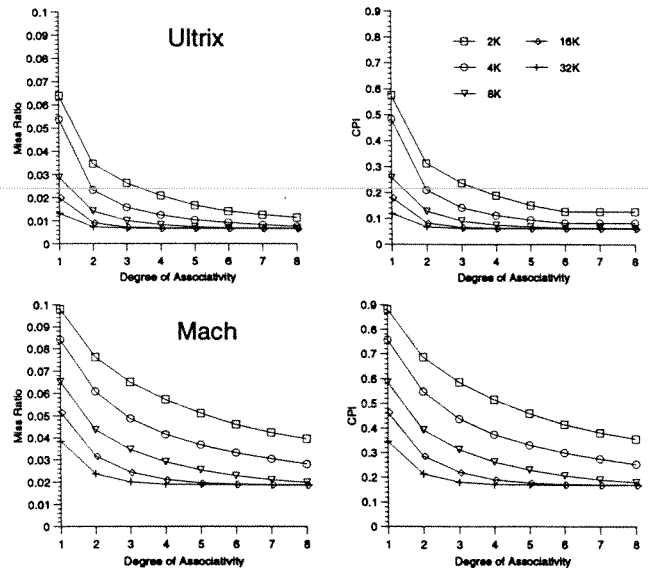

Figure 10: Performance of Set-Associative Instruction Caches

These plots show increased I-cache performance for various cache sizes and associativities. The line size for both graphs was fixed at 4 words. Increased associativity benefits Mach more than Ultrix over a broader range of cache configurations. The CPI graphs plot the I-cache's contribution to CPI.

0.028 and a 32K-byte I-cache with a 4-word line size has a miss ratio of 0.013. Note that cache pollution due to large line sizes is an issue when Ultrix runs with small caches.

The bottom left graph in Figure 9 shows that Mach generally exhibits higher I-cache miss ratios than Ultrix. For example, an 8K-byte, 4-word-line I-cache configuration under Mach has a miss ratio of 0.065, which is more than double that of Ultrix. The plot indicates that I-cache performance under Mach improves in larger increments (relative to Ultrix) as line size is increased. Further, cache pollution does not occur even for the largest line sizes of 32 words. This suggests that large I-cache line sizes are an effective means for lowering miss ratios under Mach. In fact, doubling the line size is more effective in reducing the miss ratio than doubling the cache size. To see why this is so, recall that Mach has a long instruction path to its user-level API servers (Section 4.1). The instructions along this path typically execute only once per invocation of an OS service and have a high probability of being displaced in a small cache before they are used again. Because of this, large line sizes are effective in reducing the average cost to load these instructions into the cache for each use. However, as shown in the CPI plots of Figure 9, the degree to which large line sizes can be used without increasing overall CPI is limited due to the additional cycles required to load a large line. For the miss penalties that we selected, I-cache line sizes of 16 words mark an upturn in the CPI plot.

Mach also experiences greater benefits from increased I-cache associativity than does Ultrix. The left side of Figure 10 shows that Ultrix exhibits the largest reductions in miss ratio for small caches and primarily when moving from a direct-mapped to a 2-way set-associative I-cache. On the other hand, increased associativity yields benefits over a broader range of cache configurations under Mach. For Mach, highly-associative I-caches can reduce the miss ratio, but cannot completely overcome the problems created by Mach's long code paths. An 8-way, 4K-byte I-cache still has a miss ratio of over 0.03.

For small caches, Mach's D-cache miss ratios are also higher than those of Ultrix (not shown). However, unlike the I-cache where longer line sizes and associativity significantly improved miss ratios, D-caches see a more modest improvement. Further, in contrast with I-caches, line sizes greater than 8 words begin to result in D-cache pollution under both operating systems. For our miss penalties, D-cache line sizes above 4 words begin to increase CPI.

5.4 Performance-driven Area Allocation

Section 5.2 and Section 5.3 showed that TLBs and I-caches are crucial to Mach's performance. Hypothetically, the performance problem could be overcome by building larger I-caches and TLBs. However, cost constraints limit the amount of chip area that can be dedicated to these structures. Therefore, designers must make careful trade-offs between TLB, I-cache and D-cache capacity, line sizes and associativity in order to optimize the performance under the constraints of limited on-chip memory budgets.

To determine which on-chip memory system configurations have the best performance within the constraints of a fixed, on-chip memory budget, we performed a cost/benefit analysis by combining the cost estimates from the MQF model with our TLB and cache performance data.

We selected a maximum die-area budget by examining various current generation microprocessors (Table 1). The data show that most TLBs are between 32 and 96 entries (fully-associative) while on-chip caches do not exceed 32K Bytes (for both instruction and data). The MQF model predicts that the total of these memory structures should cost less than 250,000 rbes. This relatively small amount of on-chip memory reflects technology constraints that will continue for the next several years [MReport93]. High-end systems will provide more on-chip memory, but access times will probably require that this be in a second-level cache. Moreover, trends towards inexpensive, scaled-back processors such as the MIPS R4200 and the DEC 21064 will keep many processor's on-chip primary caches small.

To illustrate our optimization process, we need CPI values rather than miss ratios or service times. Therefore, we assumed TLB miss penalties to be the same as with an R2000 processor. Because this is a software-managed TLB, miss penalties range from about 20 cycles for misses on user pages to over 400 cycles for kernel-space misses. As noted previously, cache miss penalties

	Total Storage Capacity	Degree of Associativity	Line Size (words)
TLB	64 entries up to 512 entries	1-, 2-, 4-, 8-way and fully-associative (up to 64 entries)	———
I- and D-cache	2K-bytes to 32K-bytes	1-, 2-, 4-, and 8-way	1, 2, 4, 8, 16, 32

Table 5: TLB and Cache Configurations Considered

The size for each possible TLB and cache configuration was computed using the MQF model. Combinations of I-cache, D-cache and TLB configurations that required fewer than 250,000 total rbes represent feasible allocations of on-chip memory that fit within the design budget.

TLB (size, assoc)		I-cache (size, line, assoc)			D-cache (size, line, assoc)			Total Cost (rbes)	Total CPI
512	8-way	16-KB	8-word	8-way	8-KB	8-word	8-way	163,438	1.333
512	4-way	16-KB	8-word	8-way	8-KB	8-word	8-way	162,497	1.334
512	2-way	16-KB	8-word	8-way	8-KB	8-word	8-way	162,579	1.335
512	8-way	32-KB	16-word	8-way	8-KB	8-word	8-way	249,089	1.335
512	4-way	32-KB	16-word	8-way	8-KB	8-word	8-way	248,148	1.336
512	8-way	32-KB	8-word	4-way	8-KB	8-word	8-way	243,502	1.336
512	2-way	32-KB	16-word	8-way	8-KB	8-word	8-way	248,230	1.337
512	4-way	32-KB	8-word	4-way	8-KB	8-word	8-way	242,561	1.337
512	2-way	32-KB	8-word	4-way	8-KB	8-word	8-way	242,643	1.338
512	8-way	16-KB	16-word	8-way	8-KB	8-word	8-way	167,815	1.339

Table 6: The Ten Best Area Allocations

The ten best allocations of die area given a budget of 250,000 rbes. Note that the CPI of these configurations only differ in the third decimal place, making them essentially equivalent in terms of experimental uncertainty.

were estimated to be 6 cycles for the first word in a line and 1 cycle for each additional word. Of course, different miss penalties will lead to different optimal configurations.

With 250,000 rbe's as our maximum amount of die area for on-chip memory structures, we used the MQF model to determine which combinations of TLB and cache configurations would fit on-chip. The configurations explored are listed in Table 5. Next, we used our TLB measurements and cache miss ratios from Mach to compute the contribution to CPI for each configuration, and then sorted the possible combinations by total CPI. The resulting list was very large.

Table 6 lists the 10 configurations with the lowest total CPI under Mach. All of the best configurations include a 512-entry TLB. This is due to two factors. First, large TLBs substantially reduce the TLBs contribution to CPI. Second, large TLBs do not cost very much relative to on-chip caches. For example, a 512-entry, 8-way set-associative TLB costs just 19,000 rbes while an 8K-byte, direct-mapped, 4-word-line cache costs over 74,000 rbes. Providing a large, set-associative TLB improves performance without significantly adding to total die size.

With respect to caches, the top allocation increases the I-cache size at the expense of the D-cache. Further, this allocation actually requires only 163,438 rbe's, 35% less than the maximum number of rbe's and has only 16-Kbytes of I-cache. Costlier configurations, like row four, supply more I-cache capacity, but their line size/associativity trade-offs lower the system's overall performance.

Most of the best performing configurations include a significant amount of cache associativity. However, access-time requirements may prohibit 4- or 8-way set-associative caches. Therefore, we computed another table by restricting cache configurations to set-associativities of 1 or 2. Table 7 shows that this restriction increases the best possible CPI to 1.428. Again, in these configurations, TLBs are large and I-caches are typically 2 to 4 times bigger than D-caches.

6 Conclusions and Future Work

OS trends are shifting the way that components of existing computer architectures are utilized. In particular, multiple-API operating systems require more TLB and I-cache support than single-API operating systems. At the same time, IC process limitations and cycle-time goals are forcing on-chip microprocessor memories to fit into tight design constraints. This paper focused on the particular problem of optimizing the performance of a multiple-API operating system (Mach 3.0) under die-area constraints. Because die yield (and hence manufacturing cost) is related to $(\text{die area})^3$, die area is an important constraint.

Our cost/benefit analysis found a number of die-area allocations that provide good support for Mach while staying within a given area budget. In general, the best configurations include large, set-associative TLBs because they eliminate a substantial component of CPI for relatively little cost. Good configurations also tend to make the I-cache 2 to 4 times larger than the D-cache. Further, large I-cache line sizes were found to be very effective in reducing both CPI and die area without leading to cache pollution under Mach. If timing constraints allow, larger cache associativities can be effective in reducing CPI under Mach, but are less effective under Ultrix.

Designers who perform this style of cost/benefit analysis and consider their own technology will probably find slightly different optimal points. Different workloads and less emphasis on the operating system are also likely to lead to other optimal configurations. However, if the trend to multiple-API systems continues, the importance of careful on-chip TLB and I-cache design will persist.

This work could be extended in two ways. First, we did not consider the impact of size and associativity on memory access times in a rigorous fashion. An accurate access-time model, such as that developed by Wada et al., could be used to add another dimension to this style of cost/benefit analysis [Wada92]. Second, we only considered the I-cache, D-cache and TLB in die area allocations. A more ambitious study could model the die-area cost and performance benefits of other architectural structures, such as write buffers, pre-fetching units, streaming buffers, branch-prediction units and floating-point units to see if, or to what extent they should be allocated space under a given microprocessor die budget.

Rank	TLB (size, assoc)	I-cache (size, line, assoc)	D-cache (size, line, assoc)	Total Cost (rbes)	Total CPI
1	512 8-way	32-KB 8-word 2-way	8-KB 4-word 2-way	239,259	1.428
5	512 8-way	32-KB 4-word 2-way	8-KB 8-word 2-way	248,628	1.447
13	512 8-way	32-KB 16-word 2-way	8-KB 8-word 2-way	232,040	1.462
21	512 8-way	32-KB 16-word 2-way	8-KB 2-word 2-way	241,256	1.473
24	512 8-way	32-KB 4-word 2-way	4-KB 4-word 2-way	228,214	1.475
27	256 8-way	32-KB 4-word 2-way	8-KB 2-word 2-way	249,684	1.477
59	64 Full	32-KB 8-word 2-way	8-KB 4-word 2-way	225,438	1.497
61	128 8-way	32-KB 8-word 2-way	8-KB 4-word 2-way	226,971	1.498
73	512 8-way	32-KB 16-word 2-way	8-KB 16-word 2-way	232,117	1.503
77	512 8-way	16-KB 8-word 2-way	16-KB 2-word 2-way	212,442	1.504
92	512 8-way	16-KB 4-word 2-way	16-KB 2-word 2-way	219,138	1.511
99	512 8-way	16-KB 8-word 2-way	8-KB 8-word 2-way	151,875	1.512
113	64 Full	32-KB 4-word 2-way	8-KB 8-word 2-way	234,807	1.516
1529	64 4-way	8-KB 1-word 1-way	16-KB 2-word 1-way	176,909	2.529

Table 7: Configurations that Cost Under 250,000 rbes

Memory configurations restricted to caches that are 1- or 2-way set associative. This list represents some of the first 113 best configurations and one of the poorer performing configurations (#1529). Configurations with similar features were removed from the list.

Acknowledgments

We would like to thank Madhusudhan Talluri and Mark Hill for their helpful comments on earlier versions of this paper.

References

[Accetta86] M. Accetta, et al. *Mach: A new kernel foundation for UNIX development*, In Summer 1986 USENIX Conference, USENIX, 1986.

[Agarwal88] A. Agarwal, et al. *Cache performance of operating system and multiprogramming workloads. ACM Transactions on Computer Systems* 6 (Number 4): 393-431, 1988.

[Alpert88] D. Alpert, et al. *Performance trade-offs for microprocessor cache memories. IEEE Micro* (Aug): 44-54, 1988.

[Anderson91] T. E. Anderson, et al. *The interaction of architecture and operating system design*, In Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, Santa Clara, California, ACM, 108-119, 1991.

[Bershad92] B. N. Bershad. *The increasing irrelevance of IPC performance for microkernel-based operating systems*, In Micro-kernels and Other Kernel Architectures, Seattle, Washington, USENIX, 205-211, 1992.

[Black92] D. L. Black, et al. *Microkernel operating system architecture and Mach*, In Micro-kernels and Other Kernel Architectures, Seattle, Washington, USENIX, 11-30, 1992.

[Bomberger92] A. Bomberger, et al. *The KeyKOS nanokernel architecture*, In USENIX Micro-Kernels and Other Kernel Architectures, Seattle, Washington, USENIX, 95-112, 1992.

[Chen93] B. Chen, et al. *The impact of operating system structure on memory system performance*, In Proc. 14th Symposium on Operating System Principles, 1993.

[Cheriton84] D. R. Cheriton. *The V kernel: A software base for distributed systems. IEEE Software* 1 (2): 19-42, 1984.

[Clark85a] D. W. Clark, et al. *Measuring VAX 8800 performance with a histogram hardware monitor*, In The 15th Annual International Symposium on Computer Architecture, Honolulu, Hawaii, IEEE, 176-185, 1985.

[Clark85b] D. W. Clark, et al. *Performance of the VAX-11/780 translation buffer: Simulation and measurement. ACM Transactions on Computer Systems* 3 (1): 31-62, 1985.

[Custer93] H. Custer. *Inside Windows NT.* Redmond, Washington, Microsoft Press, 1993.

[Dean91] R. W. Dean, et al. *Data movement in kernelized systems*, In Micro-kernels and Other Kernel Architectures, Seattle, Washington, USENIX, 243-261, 1991.

[Draves91] R. P. Draves, et al. *Using continuations to implement thread management and communication in operating systems.* In Proceedings of the 13th ACM Symposium on Operating Systems Principles, 122-136, 1991.

[Eickemeyer88] R. Eickemeyer, et al. *Performance evaluation of on-chip register and cache organizations*, In Proc. 15th Annual Symposium on Computer Architecture, Honolulu, Hawaii, 64-72, 1988.

[Farrens89] M. Farrens, et al. *Improving performance of small on-chip instruction caches*, In The 16th Annual International Symposium on Computer Architecture, ACM, 234-241, 1989.

[Forin91] A. Forin, et al. *An I/O system for Mach 3.0*, In USENIX Mach Symposium, Monterey, CA, USENIX, 163-176, 1991.

[Ginsberg93] M. Ginsberg, et al. *Using the Mach communication primitives in X11*. Carnegie-Mellon Technical Report, 1993.

[Golub91] D. Golub, et al. *Moving the default memory manager out of the Mach kernel*, In USENIX Mach Symposium, Monterey, CA, USENIX, 177-188, 1991.

[Goodman83] J. Goodman. *Using cache memory to reduce processor memory traffic*, In 10th International Symposium on Computer Architecture, Stockholm, Sweden, 124-131, 1983.

[Goodman86] J. Goodman, et al. *On the use of registers vs. cache to minimize memory traffic*, In Proc. 13th International Symposium on Computer Architecture, 375-383, 1986.

[Hill84] M. Hill, et al. *Experimental evaluation of on-chip microprocessor cache memories*, In 11th Annual International Symposium on Computer Architecture, Ann Arbor, Michigan, 158-166, 1984.

[Huck93] J. Huck, et al. *Architectural support for translation table management in large address space machines*, In The 20th Annual International Symposium on Computer Architecture, San Diego, California, IEEE, 39-50, 1993.

[Khalidi92] Y. A. Khalidi, et al. *An implementation of UNIX on an object-oriented operating system*. Sun Microsystems Laboratories. 1992.

[Kessler91] R. Kessler. *Analysis of multi-megabyte secondary CPU cache memories*. University of Wisconsin-Madison. 1991.

[Laha88] S. Laha, et al. *Accurate low-cost methods for performance evaluation of cache memory systems. IEEE Transactions on Computers* 37 (11): 1325-1336, 1988.

[Malan91] G. Malan, et al. *DOS as a Mach 3.0 application*. In USENIX Mach Symposium, USENIX, 27-40, 1991.

[Martonosi93] M. Martonosi, et al. *Effectiveness of trace sampling for performance debugging tools*, In SIGMETRICS, Santa Clara, California, ACM, 248-259, 1993.

[MReport92] Microprocessor Report. Sebastopol, CA, MicroDesign Resources, 1992.

[MReport93] Microprocessor Report. Sebastopol, CA, MicroDesign Resources, 1993.

[MIPS88] MIPS. *RISCompiler Languages Programmer's Guide*. MIPS, 1988.

[Mulder91] J. Mulder, et al. *An area model for on-chip memories and its application. IEEE Journal of Solid-State Circuits* 26 (2): 98-106, 1991.

[Nagle92] D. Nagle, et al. *Monster: a tool for analyzing the interaction between operating systems and architectures*. CSE-TR147-92. University of Michigan, 1992.

[Nagle93] D. Nagle, et al. *Design tradeoffs for software-managed TLBs*. In the 20th Annual International Symposium on Computer Architecture, San Diego, California, 27-38, May 1993.

[Olukotun91] O. A. Olukotun, et al. *Implementing a cache for a high-performance GaAs microprocessor*, In Proc. 18th Annual International Symposium on Computer Architecture, Toronto, Canada, 138-147, 1991.

[Ousterhout89] J. Ousterhout. *Why aren't operating systems getting faster as fast as hardware. WRL Technical Note* (TN-11): 1989.

[Patterson80] D. A. Patterson, et al. *Design considerations for single-chip computers of the future. IEEE Transactions on Computers* C-29 (2): 108-116, 1980.

[Patel92] K. Patel, et al. *Performance of a Software MPEG Video Decoder*. University of California, Berkeley. 1992.

[Przybylski89] S. Przybylski, et al. *Characteristics of performance-optimal multi-level cache hierarchies*, In Proc. 16th Annual International Symposium on Computer Architecture, Jerusalem, Israel, 114-121, 1989.

[Rozier92] M. Rozier, et al. *Overview of the Chorus distributed operating system*, In Micro-kernels and Other Kernel Architectures, Seattle, Washington, USENIX, 39-69, 1992.

[Short88] R. Short, et al. *A simulation study of two-level caches*, In Proc. 15th Annual International Symposium on Computer Architecture, Honolulu, Hawaii, 81-88, 1988.

[Sugumar93] R. Sugumar. *Multi-configuration simulation algorithms for the evaluation of computer designs*. University of Michigan. 1993.

[Torrellas92] J. Torrellas, et al. *Characterizing the caching and synchronization performance of multiprocessor operating system*, In Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, Massachusetts, ADM, 162-174, 1992.

[Uhlig93] R. Uhlig, et al. *Kernel-based memory simulation*. Technical Report CSE-TR-185-93. University of Michigan, 1993.

[Uhlig94] R. Uhlig, et al. *Kernel-based memory simulation (Extended Abstract)*. To appear in 1994 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems (Poster Session), Nashville, Tennessee, May 1994.

[Wada92] T. Wada, et al. *An analytical access time model for on-chip cache memories. IEEE Journal of Solid-State Circuits* 27 (8): 1147-1156, 1992.

[Wiecek92] C. A. Wiecek, et al. *A model and prototype of VMS using the Mach 3.0 kernel*, In USENIX Micro-kernels and Other Kernel Architectures Workshop, Seattle, Washington, USENIX, 187-203, 1992.