**The Impact of Operating System Structure on Memory System Performance**

**2.TRACE OVERWIEW:**

We estimate the behavior of Ultrix and Mach running the thirteen industry-standard workloads described in Table 1-1. Each program and operating system were adapted with epoxie [10, 37], which is a program that rewrites assembly code to record a complete address trace of instruction and memory references. Traces are accurately enclosed both -within a single context and across user and system contexts. Traced addresses are modified to reflect those of the original and not the traced instruction stream. The same traced binaries are run on both Ultrix and Mach.

We composed our traces on a DEC station 5000/200 system running Ultrix and Mach 3.0 with CMU’s UNIX server. We ran the program’s one at a time in single-user mode. The only activity was owed to the program itself, the kernel, and within the case of Mach, the user-level operating system server. Some of the experiments defined in this report required that we run an equivalent program several times against different simulated memory systems. To ensure reliability from run to run, the system was rebooted before every experiment.

The trace is fed into a reproduction of the DS5000/200 memory system using the parameters shown in Table 2-2. The simulator consumes user and system traces while they are produced, and does not write them to non-volatile storage [1O]

There are two main motives for simulating the memory system from the DS5000/200. First, the DS5000/200 memory system is fairly conventional, with no unusual properties like a particularly small TLB, virtually indexed cache, or untagged cache or TLB, that might reduce the generalization of our results. Second, Mach 3.0 and Ultrix both run on the DS5000/200, letting us to verify our simulation results against observed system behavior [11 ].

|  |  |
| --- | --- |
| **Assertion** | **Implication** |
| 1. The operating system has less instruction and data locality than user programs [14, 15]. | The operating system isn’t getting faster as fast as user programs. |
| 2. System execution is more dependent on instruction cache behavior than is user execution [35]. | A balanced cache system for user programs may not be balanced for the system. |
| 3. Collisions between user and system references lead to significant performance degradation in the memory system (cache and TLB) [30, 35, 36]. | A split user/system cache could improve performance. |
| 4. Self-interference is a problem in system instruction reference streams [28, 35]. | Increased cache associativity and/or the use of text placement tools could improve performance. |
| 5. System block memory operations are responsible for a large percentage of memory system reference costs [31, 35]. | Programs that incur many block memory operations will  run more slowly than expected. |
| 6. Write buffers are less effective for system (as opposed to user) reference streams [5, 18]. | A write buffer adequate for user code may not be adequate for system code. |
| 7. Virtual page mapping strategies can have significant impact on cache performance [25, 29]. | Systems should support a flexible page mapping interface, and should avoid default strategies that are prone to pathological behavior. |

**Table 1-1**: Seven assertions about the memory behavior of operating systems.

**The two operating systems:**

Both operating systems implement the UNIX system call interface, while their underlying implementations differ significantly.  Ultrix may be a monolithic system during which all OS code is implemented within the kernel.  A program running on Ultrix begs the operating system through a system call interface. In contrast, Mach 3.0 is a microkernel that spreads and implements a small number of orthogonal abstractions including Interprocess communication (IPC), threads, and virtual memory. Higher-level operating system services are implemented in a user-level process called the UNIX server. A program running on Mach 3.0 associates the UNIX server through the Mach kernel’s IPC interface [19], together with a user-level transparent emulation library, which is a shared library that is loaded into the address space of each process.

For our cross-system contrasts, the main UNIX components of Ultrix and CMU’s server are similar but not identical. Although both systems are derived from the same code base, they have matured in different environments. We have nevertheless attempted to remove noticeable superficial differences between the two systems. For example, both systems are compiled at the same optimization level with DEC’s Ultrix compiler from MIPS Computer Systems, and both systems use an outsized file buffer cache (12 MB).

|  |  |  |  |
| --- | --- | --- | --- |
| Workload | Description | Mach time | Ultrix time |
| **sed** | The UNIX stream editor run three times over the same 17K input file | 0.58 | 0 57 |
| **egrep** | The UNIX pattern search program run  three times over a 27K input file | 2 01 | 1.90 |
| **yacc** | The LR(1) parser-generator run on an 11K grammar | 1 75 | 1.82 |
| **gcc** | The GNU C compiler (gcc) translating a 17K (preprocessed) source file into optimized Sun-3 assembly code. | 3.70 | 4 20 |
| **compress** | Data compression using Lempel-Ziv encoding. A l00K file is compressed then uncompressed. | 1.32 | 1 32 |
| **ab** | The Andrew Benchmark with gcc The assembler was not traced | 112.18 | 98.96 |
| **espresso** | A program that minimizes Boolean function run on a 30K input file | 6.23 | 6.46 |
| **lisp** | The 8-queens problem solved in LISP | 56 46 | 54.97 |
| **eqntott** | A program that converts boolean equations to truth tables using a 1390 byte input file | 66 05 | 65.85 |
| **fpppp** | A program that does quantum chemistry analysis. This program is written in Fortran | 25 20 | 16 78 |
| **doduc** | Monte-Carlo simulation of the time evolution of a nuclear reactor component described by 8K input file. This program is written in Fortran. | 22.94 | 24.56 |
| **liv** | The Livermore Loops benchmark | 1.24 | 1 22 |
| **tomcatv** | A program that generates a vectorized mesh. This program is written in Fortran. | 139.42 | 155.44 |

**instruction cache:** 64 KB, direct-mapped, physical, 16-byte line, 15 cycle miss penalty.

**datacache:**64 KB, direct-mapped, physical. 4-byte line, write allocate, 15 cycle read miss penalty. read miss fetches 16 aligned bytes.

**write buffer:** six entries, page-mode writes complete in 1 cycle, non-page mode writes complete in 5 cycles; CPU reads have priority for memory access, but wait for writes that have already started. 4 KB page size for page-mode writes.

**Translation buffer:** 64 entries. 56 random/8 wired entries, trap to software on TLB miss. Each TLB entry maps a 4 KB page.

**page mapping** Deterministic. The physical page used to back a given virtual page is determined by the virtual page number and its address space identifier. The deterministic strategy prevents conflicts within any 64 KB (cache size) range of virtual addresses.

**kernel memory:** All kernel text and most kernel data are

in unmapped physical memory.

**Table 2-1:** Memory system simulation parameter

**Table 2-2:** Experimental workload with execution time for a DECStation 5000/200

**2.1. Sources of distortion:**

A drawn program is both larger (about a factor of two) and slower (about a factor of 15) than its untraced counterpart. The first effect, called memory dilation, can increase paging and TLB miss rates. We avoid perturbations due to paging by collecting our traces on a machine with a large physical memory in order that page-outs don’t occur. We contain TLB effects by simulating rather than tracing those TLB misses that could be affected by memory dilation. On the DEC station, there are two kinds of TLB misses. A TLB miss on a user virtual address (UTLB miss) is handled by a lightweight miss handler. This routine is not traced but is simulated using references from the traces. A TLB miss to mapped kernel memory (KTLB miss) is handled by a more general TLB miss handler.

The second effect, called time dilation, causes activity dependent on external events to appear to complete faster than in an un instrumented system. To pledge this, we have configured the system clock to interrupt at 1/15th the standard rate. We have not modified either system’s I/O behavior to account for time dilation, as this would require subtle system changes that might themselves introduce other distortions. Instead, we separate the instruction reference stream into non-idle and idle references. Idle references, which occur during I/O operations as part of a system’s idle loop in a unit programmed environment, are multiplied by the system code’s expansion factor.

An address trace contains virtual addresses, yet the actual and simulated cache is indexed by physical addresses. The third source of distortion is therefore due to the simulator’s model of the virtual memory system’s page mapping strategy, The Ultrix mapping strategy, similar to the one described in Table 2-1, is deterministic, in contrast, Mach’s strategy is random (a virtual page is bound to the first free physical page on the free list), This difference can have a measurable impact on system behavior (We isolate the consequences of the page-mapping strategies during a later section.) Our simulator uses the deterministic mapping strategy described in Table 2-1, which improves the repeatability of simulation results, and eliminates a source of variability between the two systems.

**2.2. Workloads and summary of results:**

A summary of the trace results for every program is shown in Table 2-3, with aggregate results shown in Table 2-4. For Ultrix, system behavior is constrained to the kernel. For Mach, system behavior includes the kernel, the UNIX server, and the emulation library.

For a given workload, Ultrix issues more disk requests than Mach, leading to greater idle instruction counts and delays. Ultrix is more conservative than Mach’s UNIX server in obliging meta-data updates to disk, Additionally, programs under Mach are demand-paged, whereas under Ultrix they are loaded completely at program startup, sometimes leading to unnecessary disk accesses. On average, we saw about 1.4 times more 1/0 requests for workloads under Ultrix than for Mach. Because this difference in I/O behavior is orthogonal to the difficulty of kernel architecture, we exclude idle references from our remaining discussion.

For workloads that rely heavily on UNIX services, the combined Mach system components (microkernel, UNIX server, and emulation library) execute more instructions and generally involve more data references than Ultrix.

**Memory cycles per instruction:**

We use our simulation results to calculate memory cycles per instruction (MCPI), which will be the number of CPU stall cycles due to the memory system divided by the number of instructions performed, MCPI is one of numerous components of cycles per instruction (CPI), which is a metric commonly used to estimate computer systems [22]. Other components of CPI, like one cycle per instruction for execution, interlocks during multiply, divide, and floating-point operations, and no-ops introduced by the compiler for load and branch delays, remain moderately constant even as processor cycle time decreases, in difference, MCPI is a function of the ratio of memory speed to processor speed, is less dependent on processor architecture, and can dominate overall CPI if current trends in processor and memory speed continue. As mentioned, we’ve excluded idle-loop activity from our MCPI calculations. The idle loop rarely misses in the cache, so a system could achieve an artificially low MCPI by executing an arbitrarily large number of idle instructions.

The MCPI components of the numerous programs reflect their internal behavior. The programs seal, egrep, yacc, gcc, and compress all have comparatively high system MCPI components due to their greater reliance on the operating system, especially the filing system. The gcc compiler, while run on a comparatively small input file, has a large program text and requires more system activity during program loading. The scientific workloads fpppp, liv, doduc, tomcatv) are dominated by user activity, as shown by their small system MCPI component.

The counts from Table 2-3 along with the MCPI’s can be used to estimate actual run times for Ultrix, which uses a page mapping strategy similar to the simulators.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **non -idle**  **instructions** | | | | **id le**  **instructions** | | **instruction**  **cache misses** | | | | **data cache reads** | | | |  | | | | |
| Ultrix | %s | Mach | %s | Ultrix | Mach | Ultrix | %s | Mach | %s | Ultrix | %s | Mach | %s | Ultrix | %s | Mach | %s |
| **se d** | 5704 | 24 | 7763 | 44 | 5876 | 1270 | 51 | 96 | 149 | 98 | 1515 | 17 | 2015 | 38 | 16 | 97 | 70 | 98 |
| **egrep** | 43277 | 4 | 45029 | 7 | 2495 | 914 | 43 | 93 | 140 | 98 | 8965 | 3 | 9425 | 8 | 32 | 91 | 71 | 97 |
| **yacc** | 32799 | 6 | 34539 | 10 | 13220 | 2809 | 69 | 89 | 166 | 96 | 6893 | 6 | 7283 | 11 | 48 | 50 | 93 | 71 |
| **gcc** | 29318 | 22 | 35939 | 36 | 63684 | 27027 | 485 | 42 | 999 | 71 | 8257 | 12 | 9941 | 27 | 120 | 44 | 318 | 71 |
| **compress** | 16896 | 19 | 19926 | 31 | 5555 | 2225 | 70 | 96 | 215 | 97 | 4778 | 12 | 5452 | 23 | 166 | 28 | 271 | 50 |
| **ab** | 869732 | 33 | 1198172 | 51 | 689324 | 247969 | 15612 | *51* | 28619 | 73 | 223588 | 26 | 295572 | 44 | 6658 | 79 | 11262 | 86 |
| **espresso** | 135385 | 2 | 137806 | 4 | 21601 | 8069 | 187 | 45 | 344 | 70 | 32034 | 2 | 32652 | 3 | 93 | 32 | 168 | 58 |
| **lisp** | 1288027 | 3 | 1276619 | 2 | 1005 | 0 | 222 | 61 | 2004 | 54 | 468287 | 2 | 467668 | 2 | 655 | 45 | 734 | 68 |
| **eqntott** | 1414369 | 1 | 1417868 | 1 | 10632 | 0 | 126 | 88 | 254 | 97 | 296691 | 1 | 297901 | 1 | 14328 | 3 | 14489 | 4 |
| **fpppp** | 265457 | 8 | 262998 | 7 | 17102 | 5667 | 4135 | 21 | 3735 | 19 | 139172 | 2 | 139036 | 2 | 131 | 27 | 177 | 47 |
| **doduc** | 321325 | 1 | 325351 | 2 | 18474 | 4983 | 6239 | 5 | 6292 | 7 | 122009 | 0 | 123020 | 1 | 550 | 9 | 612 | 21 |
| **liv** | 23008 | 3 | 23778 | 6 | 1585 | 639 | 21 | 93 | 72 | 98 | 7968 | 2 | 8176 | 4 | 17 | 88 | 30 | 96 |
| **tomcatv** | 2005703 | 1 | 2005590 | 1 | 10823 | 134 | 138 | 82 | 326 | 84 | 967474 | 0 | 968074 | 0 | 85451 | 0 | 85522 | 0 |

**Data cache read misses**

**Table 2-3:** Summary of trace results

**3. Comparative system behavior:**

As shown within the previous section, the foremost substantial difference between Mach and Ultrix is that the number and cost of non-idle instructions required to run an application. In this section, we discuss the influence that major system components have on system performance.

In Figure 3-1 we separate system overheads into 11 major components and compare Ultrix and Mach in terms of given components. The components are: trap (system call and exception handling), UTLB (user TLB miss), KTLB (kernel TLB miss), VM-MD (machine-dependent virtual memory), VM-mi (Mach’s machine independent virtual memory), Block Ops (block memory moves and zeroes), UNIX service (the remaining routines in the Ultrix kernel and Mach UNIX server), Microkernel (Mach’s microkernel, including device management and scheduling), IPC (the Mach kernel’s message system), Emulator (Mach’s transparent emulation library), and S-MCPI (system memory cycles per system instruction). The primary ten categories reflect relative overheads in terms of non-idle system instructions executed. The last category, S-MCPI, reflects relative memory system overhead for system references. Four of the activities (Microkernel, Emulator, IPC, VM-mi) occur only in Mach. Block Ops for Mach includes operations from both the Mach kernel and therefore the UNIX Server.

The number at the top of each and every bar is system cycles as a percentage of total cycles. The Ultrix instruction counts are normalized to a minimum of one for all workloads. The heights of the bars reflect system, but not total execution, overheads. For workloads where system activity is small relative to total activity (doduc, for example), system contribution to performance is minor.

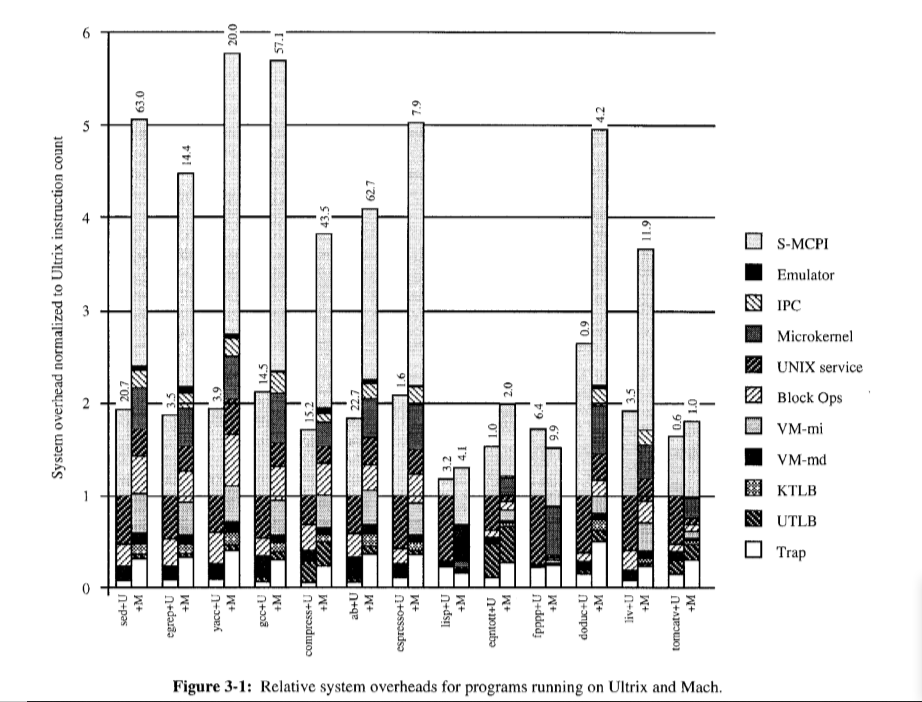
Some characteristics of system behavior are worth seeing from Figure 3-1. First, the memory penalty for system instructions is from one to three times greater for Mach than for Ultrix.

Second, Mach’s virtual memory storage system performs more instructions than the one implemented in Ultrix, which has been flattened into one machine-dependent layer. Mach has an additional machine-independent layer that is more costly than either system’s machine-dependent layer.

This comparison should not be taken to mean that one system is better or worse than the other since Mach’s virtual memory interface provides substantially more functionality and portability than Ultrix’s.

The third point of comparison is the relative instruction cost of UNIX service in Ultrix, which is larger than that under Mach, For Ultrix, the UNIX service category includes many machine-dependent services such as device management that are counted as part of Mach’s Microkernel category. On the other hand, the Microkernel category indicates that there’s a measurable cost for providing those services through a separate set of kernel interfaces. The UNIX service category also includes some services that are implemented in Mach’s transparent emulation library. For example, lisp features a relatively high UNIX service component under Ultrix, but almost none under Mach. This is because lisp frequently modifies UNIX signal state to support garbage collection, and signal state can be manipulated from within Mach’s transparent emulation library.

Lastly, Figure 3-1 shows that the overhead of Mach’s IPC, in terms of instructions executed, is liable for a little portion of overall system overhead. This suggests that microkernel optimizations focusing exclusively on IPC [8, 18, 20, 26, 34], without considering other sources of system overhead like MCPI, will have a limited impact on overall system performance [7].



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | i-cache cycles | d -cache cycles | tlb cycles | wbuffer cycles |
| Ultrix user | 0.07 | 0 08 | 0 00 | 0.02 |
| Mach u ser | 0.07 | 0.08 | 0.00 | 0.02 |
| Ultrix system | 0 43 | 0 23 | 0.00 | 0.05 |
| Mach system | 0 57 | 0 29 | 0.01 | 0 07 |

**Table 2-4:** Summary penalty cycles (per instruction).

These figures, which are the average over all the workloads, characterize system execution (non-idle system cycles / non- idle system instructions) vs. user execution (user cycles / user instructions). As such, they do not reflect the impact of system execution on overall performance.

**4. Seven assertions:**

In this section, we estimate the strength of the seven assertions enumerated in the introduction. Our basic approach is to address each assertion in the context of our traces. In several cases, we present the results of additional simulations in which we vary the base architecture to determine the sensitivity of system performance to the assertion in question.

**4.1. System and user locality:**

As cache behavior is a sign of locality, Table 2-3 supports the primary assertion***: the operating system has less instruction and data locality than user programs***. The system can contribute up to 51% of non-idle instruction cache references, but in most cases (17 of 26) the system contribution is a smaller amount than 10%, Given this, a disproportionately greater number of instruction cache misses are due to the system (greater than 70% for two-thirds of the workload/system pairs).

In terms of knowledge references, the system contributes a bigger percentage of misses than references, again supporting the assertion that the system’s data locality is worse than the users. Even so, in just five of the workload/system combinations does the system contribute more than 90% of data misses, and only twelve if the threshold is lowered to 50%. Although the system’s contribution of instruction and data references are comparable, the percentage of misses is not. Instruction references miss more often than data references for both Mach and Ultrix. From this, we conclude that instruction locality is worse than data locality during system execution.

**4.2. System instruction locality:**

Percentages are convenient for comparing system and user behavior, but they mist overall performance effects. For example, while 97% of instruction cache misses for eqntott under Mach are due to the system, the system instruction cache miss rate is irrelevant.  
A good indicator of the performance effect of locality is the cache’s contribution to MCPI. In Table 4-1 we combine our baseline data from Table 2-3 with cache miss penalties for the simulated memory system to yield the MCPI contributions from the cache. The module of MCPI due to system instruction cache references leads that due to the user in 20 of 26 cases. In contrast, the system data cache component leads in only thirteen cases. Furthermore, the majority of the system (as opposed to the user) cache penalties are due to poor instruction cache behavior; only five runs show greater penalties for data than instructions, and in these runs the system’s contribution to MCPI is small. This behavior provisions the second assertion**: *system execution is more dependent on instruction cache behavior than is user execution****.* However, many of the programs in our workload have small working sets that fit entirely in the instruction cache. Larger programs such as gee, which do not fit in the cache, can have instruction cache penalties that competing for that of the system.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| workload | instruction cache | | data cache | |
| Ultrix  sys u ser | Mach  sys user | Ultrix  sys u ser | Mach  sys user |
| sed | 0.129 0.005 | 11.283 0.005 | 0.041 0.001 | 0.132 0.003 |
| egrep | 0.014 0.001 | 0.046 0.001 | 0.010 0.000 | 0.023 0.000 |
| yacc | 0.028 0.004 | 0.069 0.003 | 0.011 0.011 | 0.029 0.012 |
| gcc | 0 103 0 145 | 0.294 0.123 | 0 027 0 034 | 0.094 0.039 |
| compress | 0.060 0.002 | 0.157 0.005 | 0 042 0 106 | 0 101 0 102 |
| ab | 0.139 0.130 | 0.261 0.098 | 0.091 0.024 | 0.121 0.020 |
| espresso | 0 009 0 012 | 0.026 0.011 | 0 003 0 007 | 0.011 0.008 |
| lisp | 0.002 0.001 | 0.013 0.011 | 0 003 0 004 | 0.006 0.003 |
| eqntott | 0.001 0.000 | 0.003 0.000 | 0 005 0 147 | 0 006 0 147 |
| fpppp | 0.050 0 184 | 0 040 0 173 | 0 002 0 005 | 0 005 0 005 |
| doduc | 0 014 0 277 | 0 020 0 270 | 0.002 0 023 | 0 006 0 022 |
| liv | 0.013 0.000 | 0.045 0.000 | 0.010 0.001 | 0.018 0.000 |
| tomeatv | 0.000 0.000 | 0.002 0.000 | 0 005 0 634 | 0 005 0 634 |

**Table 4-1:** *MCPI* contributions from the cache.

Table 4-1 quantifies the difference in MCPI between Mach and Ultrix. Memory penalties due to system instruction and system data references are greater for Mach than for Ultrix, while user memory penalties are similar. Increased system activity in Mach, as is shown in Figure 3-1, leads to a bigger cache contribution to MCPI.

**4.3. Competition between the user and system:**

The increased cache activity in Mach suggests that user code running on Mach may run more slowly than on Ultrix due to increased cache competition. To evaluate this, we ran the workloads against a simulated memory system that had independent 64 KB system and user caches. Again, by “system” for Mach, we mean the Mach kernel, the UNIX server, and the emulator.

Although our separate user and system caches double the effective cache size, the general dominance of the two leftmost components indicates that they do not significantly reduce miss rates relative to a smaller unified cache. The largest interference effects (for example, lisp) occur when the cache miss rate is low, such a couple of interference misses may result during a large relative change. The absolute contribution of competition misses to MCPI is shown in Table 4-2. These points imply that the third assertion, ***collisions between user and system references lead to significant performance degradation in the memory system***, is not true for these workloads.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| workload | Ultrix | | | Mach | | |
| inst | data | total | inst | data | total |
| sed | 0.010 | -0.006 | 0.004 | 0.009 | 0.004 | 0.013 |
| egrep | 0.003 | 0.000 | 0.003 | 0.002 | 0.002 | 0.004 |
| yacc | 0.005 | 0.002 | 0.007 | 0.004 | 0.005 | 0.009 |
| gcc | 0.050 | 0.007 | 0.057 | 0.047 | 0.018 | 0.065 |
| compress | 0.004 | 0.018 | 0.022 | 0.010 | 0.034 | 0.044 |
| ab | 0.038 | 0.006 | 0.044 | 0.029 | 0.000 | 0.029 |
| espresso | 0.005 | 0.002 | 0.007 | 0.004 | 0.004 | 0.008 |
| lisp | 0.002 | 0.006 | 0.008 | 0.022 | 0.004 | 0.026 |
| eqntott | 0.000 | 0.004 | 0.005 | 0.000 | 0.005 | 0.005 |
| fpppp | 0.072 | 0.002 | 0.074 | 0.047 | 0.002 | 0.049 |
| doduc | 0.023 | 0.002 | 0.025 | 0.016 | 0.002 | 0.018 |
| liv | 0.001 | 0.004 | 0.005 | 0.000 | 0.001 | 0.002 |
| tomcatv | 0.000 | 0.005 | 0.006 | 0.000 | 0.005 | 0.006 |

**Table 4-2:** MCPI contributions from cache competition

**4.4. System self-interference:**

Self-interference occurs when insufficient cache associativity results in a cache miss. The impact of self-interference in user-code is well-understood [23]. To evaluate the impact of system self-interference, we simulated a two-way LRU set-associative cache of the same size as our direct-mapped cache. As in the previous section, user references are isolated from the system-only cache, although they continue to generate TLB misses and subsequent system activity.

The increased associativity eliminates a significant fraction of misses, and is more effective for instruction than data references. This confirms the fourth assertion: ***self-interference is a problem in system instruction reference streams.***

Self-interference has the largest relative impact when MCPI is low, and the smallest relative impact when MCPI is high. A high MCPI implies that the cache is full, which is a situation that cannot be helped by increased associativity. For example, seal, egrep, and liv have high MCPI's but gain relatively little from associativity. In contrast, associativity helps most with lisp and tomcatv, where MCPI is relatively low. Associativity is usually less beneficial for Mach than for Ultrix because applications on Mach tend to possess a better MCPI.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | TLB refs | | | UTLB | | misses | | | KTLB | misses | UTLB | MCPI | KTLB | MCPI |
| workload | user | Ultrix | Mach | U-user | M-user | | Ultrix | Mach | Ultrix | Mach | Ultrix | Mach | Ultrix | Mach |
| sed | 5596 | 423 | 1079 | 0.09 0.49 | | | 0.06 | 6.67 | 441 | 2132 | 0.000 | 0.021 | 0.011 | 0.063 |
| egrep | 50399 | 546 | 1116 | 0.07 0.39 | | | 0.10 | 6.41 | 472 | 1847 | 0.000 | 0.003 | 0.002 | 0.009 |
| yacc | 37460 | 571 | 1323 | 0.25 1.26 | | | 0.04 | 7.89 | 359 | 2280 | 0.000 | 0.003 | 0.003 | 0.015 |
| gcc | 30093 | 1582 | 2951 | 32.33 3J.87 | | | 0.10 | 17.87 | 1521 | 3305 | 0.000 | 0.016 | 0.006 | 0.022 |
| compress | 17892 | 986 | 2085 | 82.58 82.06 | | | 0.12 | 10.24 | 712 | 3982 | 0.000 | 0.012 | 0.005 | 0.045 |
| ab | 755092 | 90958 | 195492 | 1148.79 1208.04 | | | 12.83 | 1457.98 | 95058 | 575598 | 0.000 | 0.030 | 0.014 | 0.108 |
| espresso | 164313 | 660 | 1281 | 0.86 2.64 | | | 0.05 | 7.67 | 452 | 3111 | 0.000 | 0.001 | 0.000 | 0.005 |
| lisp | 1706833 | 12974 | 26783 | 0.10 12.69 | | | 0.04 | 15.68 | 376 | 8063 | 0.000 | 0.000 | 0.000 | 0.002 |
| eqntott | 1690678 | 3579 | 3697 | 692.01 692.57 | | | 0.11 | 24.03 | 1321 | 9760 | 0.000 | 0.000 | 0.000 | 0.002 |
| fpppp | 580307 | 3632 | 1169 | 4.72 13.54 | | | 0.34 | 9.02 | 366 | 2273 | 0.000 | 0.000 | 0.000 | 0.003 |
| doduc | 438563 | 899 | 2162 | 16.78 30.53 | | | 0.04 | 18.26 | 402 | 5811 | 0.000 | 0.000 | 0.000 | 0.005 |
| liv | 30123 | 232 | 417 | 0.03 0.11 | | | 0.04 | 2.62 | 197 | 701 | 0.000 | 0.003 | 0.001 | 0.007 |
| tomcatv | 2949614 | 4480 | 2684 | 317.34 321.79 | | | 014 | 2569 | 1608 | 8135 | 0.000 | 0.000 | 0.000 | 0001 |

**Table 4-3:** TLB activity

This table shows TLB references (x 1000), UTLB misses (x 1000), **KTLB** misses (x I), UTLB *MCPI,* and KTLB ***MCPI*** for system and user across the various workloads. The number of users UTLB references is the same for both systems, as the same user code is executed. UTLB miss counts depend on competition from the system, so the table shows separate numbers for Ultrix and Mach. **KTLB** misses do not occur in usercode.

**4.5. Block operations:**

Operating systems perform block memory operations to transfer data between I/O devices and memory, and to copy data between address spaces, Table 4-4 shows that block memory operations and their subsequent interference are often liable for a considerable fraction of total MCPI, especially for programs that perform significant I/O. Espresso, while not I/O intensive, pays the high relative penalty for block operations because program loading overheads dominate its cache behavior. From the measurements, we conclude that assertion five: ***system block memory operations are liable for a larger percentage of memory system reference costs is true***, and most vital in I/O intensive applications.

|  |  |  |
| --- | --- | --- |
| workload | Ultrix | Mach |
| MC PI %total | MC PI %total |
| se d | 0.066 29.2 | 0.131 26.6 |
| egrep | 0. 014 39.3 | 0.017 20.9 |
| yacc | 0.017 25.6 | 0.027 20.9 |
| gcc | 0. 116 26.8 | 0. 159 23.0 |
| compress | 0.055 22.1 | 0.071 17.0 |
| ab | 0. 100 23.4 | 0.057 10.7 |
| espresso | 0.009 21.3 | 0.013 19.9 |
| lisp | 0.000 0.3 | 0.000 0.0 |
| eqn tott | 0.000 0.4 | 0.001 0.6 |
| fpppp | 0.003 1.2 | 0.005 2.2 |
| dod uc | 0.003 0.9 | 0.006 1.9 |
| liv | 0.008 7.1 | 0.013 7.9 |
| tomcatv | 0.000 0 0 | 0 000 0 0 |

**Table 4-4:** MCPI from block memory operations.

For each system, this table shows the MCPI contribution of block moves (and subsequent interference), and also the per- centage of total MCPI due to block moves.

In terms of MCPI, Table 4-4 shows that block operations incur a much bigger absolute overhead for programs running on Mach than on Ultrix. Table 4-5 shows that Mach generally references more data than Ultrix in block operations which more of these references undergo to memory. Block operations in Mach occur within the kernel as part of the VM and IPC systems, and within the UNIX server as part of the file system. In contrast, Ultrix block operations, which occur entirely within the kernel, are due mostly to VM and file system operations.

**4.6. Streaming writes:**

Operating systems stream data to memory during block transfers, like for I/O and IPC, and through context switches and exception handling. Write buffers accelerate streaming writes by allowing the CPU to run before memory. The effect of streaming write operations on system performance can be measured by counting stall cycles due to writes. The number of write stall cycles per instruction for user and system code under Ultrix and Mach is shown in Table 4-6. In most cases system behavior is worse than user behavior, supporting the sixth assertion**: *write buffers are less effective for system references****.*

|  |  |  |  |
| --- | --- | --- | --- |
| workload | Ultrix | Mach | |
| systern u ser | system | u ser |
| se d | 0.061 0.000 | 0.076 | 0.000 |
| egrep | 0.050 0.002 | 0.065 | 0.002 |
| yacc | 0. 062 0.000 | 0.076 | 0.000 |
| gcc | 0.106 0.012 | 0.129 | 0.012 |
| com press | 0.043 0.01l | 0.063 | 0.013 |
| ab | 0.040 0.009 | 0.043 | 0.010 |
| espresso | 0.093 0.001 | 0.111 | 0.001 |
| lisp | 0.007 0.004 | 0.064 | 0.005 |
| eqn tott | 0.014 0.000 | 0.024 | 0.0110 |
| fpppp | 0.030 0. 017 | 0.037 | *0. 015* |
| doduc | 0.101 0.018 | 0.095 | 0.018 |
| liv | 0.052 0 090 | 0.075 | 0 090 |
| tomcat v | 0.023 0 033 | 0.044 | 0.033 |

**Table 4-6:** Write buffer stall cycles per instruction.

This table shows write buffer stall cycles per user instruction and write buffer stall cycles per system instruction. Runs in which system behavior is worse than user behavior is shown in bold face.

System write buffer stalls per instruction are usually higher for Mach than for Ultrix. Overall cache miss rates are higher with Mach, and the DEC station 5000/200 memory system gives CPU reads priority over outstanding writes. Consequently, fewer memory cycles are available for the write buffer to retire outstanding writes, resulting in a larger number of stalls. Additionally, the interleaved read misses decrease the frequency of low-latency page mode writes.

**4.7. Page mapping strategy:**

The system’s virtual page mapping strategy can affect the performance of a physically indexed cache because it determines the location and overlay of virtual pages within the cache. As an example, the OS can decrease self-interference misses for small applications by employing a virtual-to-physical mapping that uniformly distributes consecutive virtual pages throughout the cache. For localities smaller than the cache size, such a way prevents collisions within the cache. This approach also makes possible tools that reposition the layout of text and data in memory to improve cache performance [27, 17].

In our discussion so far, we have pretended a deterministic strategy for both the Ultrix and Mach reference streams. As earlier mentioned, Ultrix uses a deterministic strategy, even though Mach’s strategy is random (a virtual page is allocated to the next physical page on the free list). To isolate the consequence of the page mapping strategy, we modified our simulator to use random mappings, and to sustain page tables so that page mappings do not change during a given run.

In some cases, the deterministic strategy produces a page mapping with low user cache miss rates. Specific examples are sed and lisp under Ultrix, and egrep and liv for both systems. In these cases, the deterministic strategy results in good behavior, and therefore the random strategy can attain significantly worse. Our results, though, suggest that such cases are rare within the absence of program reordering.

Overall, these observations confirm the seventh assertion: ***virtual to physical page mapping strategy can have a big impact on cache performance***. Furthermore, a deterministic strategy can have a bad impact on performance for a direct-mapped cache when program reordering tools aren’t used. In such cases, a random strategy is less likely to bring constantly poor behavior.

**5. Conclusions:**

For the majority of workloads we ponder, the number and cost of non-idle instructions performed are substantially advanced for Mach than for Ultrix. Six of the assertions about operating systems and memory system behavior are true, while two have little or no impact on system performance. One is false. Numerous are sensitive to the operating system architecture. Specifically:

● **System and user locality**. System locality is measurably worse than user locality, and the performance effect can be significant, The Mach microkernel-based system has a poorer system locality than Ultrix.

● **System instruction locality**. Relative to user behavior, system test shows less locality than system data. However, user workloads like gcc with big text can have instruction cache penalties that opposing that of the operating system.

● **User/system competition**. User/system competition may be a measurable component of cache and TLB miss rates. For these workloads, however, system performance is not suffering from user/system competition. The impact of Mach’s microkernel structure on competition is not significant.

● **System self-interference**. Self-interference accounts for a substantial number of system misses, particularly in system text. However, the cases with the worst overall behavior are also those that advantage least from associativity. Compared to Ultrix, associativity rejects a lower percentage of Mach’s cache misses because of its greater request for cache resources.

● **Block operations**. Block operations are often in charge of a large component of overall MCPI, particularly for applications that perform I/O. Mach moves more data with block operations and has a higher MCPI due to block operations than Ultrix.

● **Streaming writes**. System code presents a higher load to the write buffer than user code. Mach’s increased cache MCPI results in a larger number of write buffer stalls due to competition between memory reads and writes.

● **Page mapping strategy**. Page mapping strategies can have a large effect on cache performance. The page mapping strategy is independent of operating system architecture.

The performance of the operating system, either monolithic or microkernel-based, is more sensitive to memory system latency than that of applications. The locality of system code and data is inherently deprived, and changes to memory systems that help application performance by taking advantage of the locality are unlikely to bring proportional improvements to the system.

**References:**

1. Michael J. Accetta, Robert V. Baron, William Bolosky, David B. Gohrb, Richard F, Rashid, Avadis Tevanian, Jr., and Michael W. Young Mach: A New Kernel Foundation for Unix Development. Proceedings of the Summer 1986 USENIX Conference, July. 1986, pp. 93-113

2. Anant Agarwal, Richard L, Sites, and Mark Horowitz. ATUM: A New Technique for Capturing Address Traces Using Microcode. The Proceedings of the 13th International Symposium on Computer Architecture, June, 1986. pp. 119-127.

3. Anant Agarwal, John Hennessy, and Mark Horowitz, “Cache Performance of Operating System and Multiprogramming Workloads”. ACM Transactions on Computer Systems 6, 4 (November 1988), pp 393-431.

4. Anant Agarwal. Analysis of Cache Performance for Operating Systems and Multiprogramming. Kluwer Academic Publishers, Boston, MA, 1989.

5. Thomas E. Anderson, Henry M. Levy. Brian N. Bershad, and Edward D Lazowska. The Interaction of Architecture and Operating System Design. The Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April, 1991, pp. 108-120.

6. Ozalp Babaoglu and William Joy. Converting a SwapBased System to do Paging in an Architecture Lacking Page Referenced Bits The Proceedings of the 8th ACM International Symposium on Operating System Principles, December, 1981, pp. 76-86.

7. Brian N. Bershad. The Increasing Irrelevance of IPC Performance for Microkernel-Based Operating Systems. The Proceedings of the First USENIX Microkernels and Other Kernels Workshop, April, 1992, pp 204-211.

8. Brian N, Bershad, Thomas E, Anderson, Edward D. Lazowska and Henry M. Levy. “Lightweight Remote Procedure Call”. ACM Transactions on Computer Systems 8, 1 (February 1990), pp. 37-55.

9. Brian N. Bershad, Richard P. Draves, and Alessandro Forin, Using Microbenchmarks to Evaluate System Performance. The Proceedings of the Third Workshop on Workstation Operating Systems, April, 1992, pp 148-153.

10. Anita Borg, R.E. Kessler, Georgia Lazana, and David Wall Long Address Traces from RISC Machines: Generation and Analysis. WRL Research Report 89/14, Digital Equipment Corporation Western Research Laboratory, 1989

11. J. Bradley Chen. Software Methods for System Address Tracing. The Proceedings of the Fourth Workshop on Workstation Operating Systems, October, 1993

12. J. Bradley Chen. Memory Behavior for an Xl 1 Window System. The Proceedings of the Winter 1994 USENIX Conference, January, 1994.

13. J. Bradley Chen, Anita Borg, and Norman P. Jouppi. A Simulation Based Study of TLB Performance. The Proceedings of the 19th Annual International Symposium on Computer Architecture, May, 1992, pp. 114-123.

14. Douglas W, Clark, “Cache Performance in the VAX-1 1/780”. ACM Transactions on Computer Systems 1, 1 (February 1983), pp. 24-37.

15. Douglas W. Clark and Joel S. Emer. “Performance of the VAX 11/780 Translation Buffer: Simulation and Measurement”. ACM Transactions on Computer Systems 3, 1 (February 1985), 270-301.

16. M. DeMoney, J. Moore, and J. Mashey. Operating System Support on a RISC. Proceedings of the 31st Computer Society International Conference (Spring Compcon ‘86), March, 1986, pp 138-143.

17. Digital Equipment Corporation. cord. Ultrix manual page.

18. Richard P. Draves, Brian N, Bershad, Richard F. Rashid and Randall W. Dean. Using Continuations to Implement Thread Management and Communications in Operating Systems. Proceedings of the 13th ACM Symposium on Operating Systems Principles, October, 1991, pp. 122-136.

19. Richard P Draves. A Revised IPC Interface. Proceedings of the First Mach USENIX Workshop, October, 1990, pp. 101-121.

20. Peter Druschel, Larry L. Peterson, and Norman C. Hutchinson. Beyond Microkernel Design: Decoupling Modularity and Protection in Lipto. The Proceedings of the 12th International Conference on Distributed Computing Systems, June, 1992.

21. David Golub, Randall Dean, Alessandro Forin and Richard Rashid. UNIX asan Application Program. Proceedings of the Summer 1990 USENIX Conference, June, 1990, pp. 87-95.

22. John L. Hennessy and David A. Patterson. Computer Architecture: A Quantitative Approach. Morgan Kaufmann, Palo Alto, CA, 1990.

23. Mark D. Hill. Aspects of Cache Memory and Instruction Rear- Performance. Ph.D. The University of California at Berkeley, Computer Sciences Division, November 1987. Number UCB/CSD 87/381.

24. Gerry Kane. MIPS RLSC Architecture. Prentice Hall, Englewood Cliffs, NJ, 1987.

25. R E. Kessler and Mark D. Hill. “Page Placement Algorithms for Large Real-Indexed Caches”. ACM Transactions on Computer Systems 10, 4 (November 1992), 338-359.

26. Jay Lepreau, Mike Hibler, Bryan Ford, Jeffrey Law, and Douglas Orr. In-Kernel Servers on Mach 3.0: Implementation and Performance. Proceedings of the Third USENIX Mach Symposium, April, 1993, pp. 39-56.

27. Scott McFarling. Program Optimization for Instruction Caches. The Proceedings of the Third International Conference on Architectural Support for Programming Languages and Operating Systems, April, 1989, pp. 183-191.

28. Jeffrey C. Mogul and Anita Borg. The Effect of Context Switches on Cache Performance. The Proceedings of the Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, April, 1991, pp. 75-84.

29. David Nagle, Richard Uhlig, and Trevor Mudge. Monster: A Tool for Analyzing the Interaction Between Operating Systems and Computer Architectures. University of Michigan, November, 1992. CSE-TR-147-92.

30. David Nagle, Richard Uhlig, Tim Stanley, Stuart Sechrest, Trevor Mudge and Richard Brown. Design Tradeoffs for Software-Managed TLBs. Proceedings of the 20th Annual International Symposium on Computer Architecture, May, 1993, pp. 27-38.

31. John K. Ousterhout. Why Operating Systems Aren’t Getting Faster As Fast As Hardware. Proceedings of the Summer 1991 USENIX Conference, June, 1991, pp. 247-256.

32. Steven A. Przybylski. Cache Design: A PeornranceDwected Approach. Morgan-Kaufmann, San Mateo, CA, 1990.

33. Richard Rashid, Avadis Tevanian, Jr., Michael Young, David Gohrb, Robert Baron, David Black, William Bolosky and Jonathan Chew. Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures. Proceedings of the Second International Conference on Architectural Support for Programming Languages and Operating Systems,April, 1987, pp. 31-39.

34. M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Giend, M. Guillemot, F. Herrmann, P. Leonard, S. Langlois, and W. Neuhauser. “Chorus Distributed Operating Systems”. Computing Systems 1,4 (1988), pp. 305-370.

35. JosepTorellas, Anoop Gupta, and John Hennessy. Characterizing the Caching and Synchronization Performance of a Multiprocessor Operating System. The Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, October, 1992, pp. 162-174.

36. Bart C. Vashaw. Address Trace Collection and Trace Driven Simulation of Bus Based, Shared Memory Multiprocessors. Ph.D. Th., Carnegie Mellon University, 1992. Department of Electrical and Computer Engineering.

37. David W. Wall. Systemsfor Late Code Modification. In Code Generation --- Concepts, Tools, Techniques, SpringerVerlag, 1992, pp. 275-293.