# CRITIQUE OF MICROKERNEL ARCHITECTURES

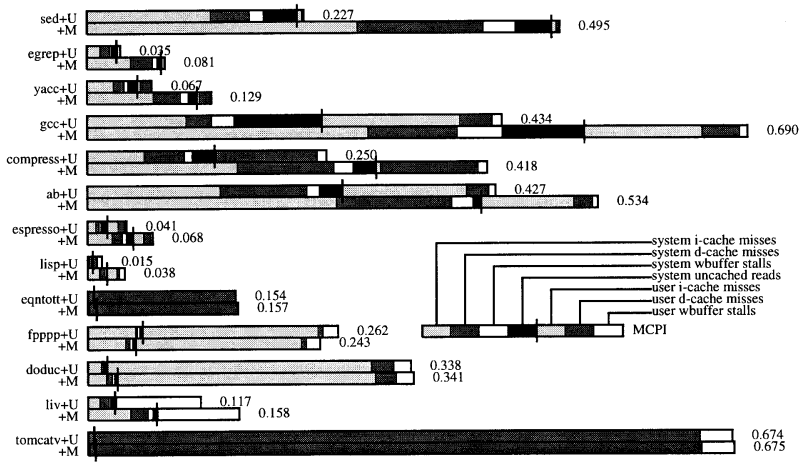
- First generation $\mu$-kernel systems exhibited poor performance when compared to monolithic UNIX implementations.
- Reasons are investigated by comparing benchmark performance of Ultrix and Mach on same hardware (DECstations) in:

  B. Chen and B.N. Bershad: The impact of operating system structure on memory system performance. In *Proc. 14th ACM Symp. OS Princip.*, pages 120–133, Asheville, NC, USA, December 1993.

  ➜ Collect execution traces from instrumented user and system code.
  
  ➜ Run on DECstation 5000/200 (25MHz R3000).
  
  ➜ Traces fed to memory system simulator.
  
  ➜ Analyse MCPI (memory cycles per instruction).



## ULTRIX VS. MACH MCPI






## INTERPRETATION

**Observations:**

- Memory penalty (i.e. cache missess or write stalls) higher in Mach.
- Mach VM system executes more instructions than Ultrix (but has more functionality).

**Claim:**

- Degraded performance is (intrinsic?) result of OS structure.
- IPC cost (known to be high in Mach) is not a major factor [Ber92].



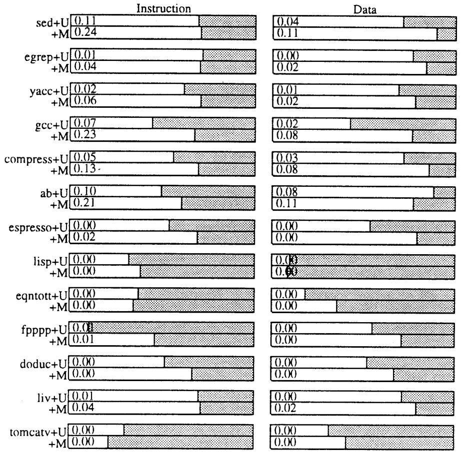
## ASSERTIONS

1. **OS has less instruction and data locality than user code.**
   ➜ System code has higher cache and TLB miss rates.
   
   ➜ Particularly bad for instructions.
2. **System execution is more dependent on instruction cache behaviour than is user execution**
   ➜ MCPIs dominated by system i-cache misses.

   Note: most benchmarks were small, i.e. user code fits in cache.
3. **Competition between user and system code is not a problem**
   ➜ Few conflicts between user and system caching.
   
   ➜ TLB misses are not a relevant factor

   Note: the hardware used has direct-mapped physical caches.
   $\Rightarrow$ Split system/user caches wouldn't help.

## Self interference




- ➜ Only system cache misses.
- ➜ Shaded: System cache misses removed by associativity.
- ➜ MCPI for system-only, using R3000 direct-mapped cache.
- ➜ Reductions due to associativity were obtained by running system on a simulator and using a two-way associative cache of the same size.



## Assertions...

4 **Self-interference is a problem in system instruction reference streams.**
- ➜ High internal conflicts in system code.
- ➜ System would benefit from higher cache associativity.

5 **System block memory operations are responsible for a large percentage of memory system reference costs.**
- ➜ Particularly true for I/O system calls.

6 **Write buffers are less effective for system references.**
- ➜ Write buffer allows limited asynchronous writes on cache misses

7 **Virtual to physical mapping strategy can have significant impact on cache performance**
- ➜ Unfortunate mapping may increase conflict misses.
- ➜ “Random” mappings are to be avoided.



## Other experience with $\mu$-kernel performance

- System call costs are (inherently?) high.
  - ➜ Typically hundreds of cycles, 900 for Mach/i486.
- Context (address-space) switching costs are (inherently?) high.
  - ➜ Getting worse (in terms of cycles) with increasing CPU/memory speed ratios [Ous90].
  - ➜ IPC (involving system calls and context switches) is inherently expensive.



## So, what's wrong?

$\mu$-kernels heavily depend on IPC

- Is the $\mu$-kernel idea flawed?
- Should some code never leave the kernel?
- Do we have to buy flexibility with performance?

## A Critique of the Critique

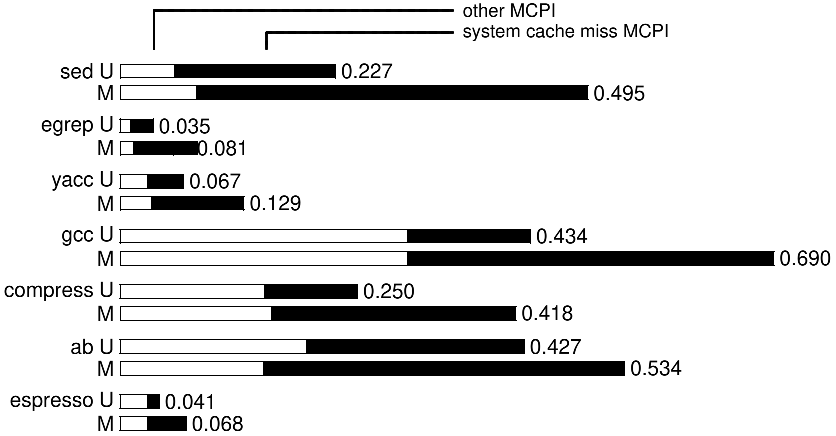
Data presented earlier:

- ➜ are specific to one (or a few) system,
- ➜ results cannot be generalised without thorough analysis,
- ➜ no such analysis has been done.

⇒ Cannot trust the conclusions [Lie95].



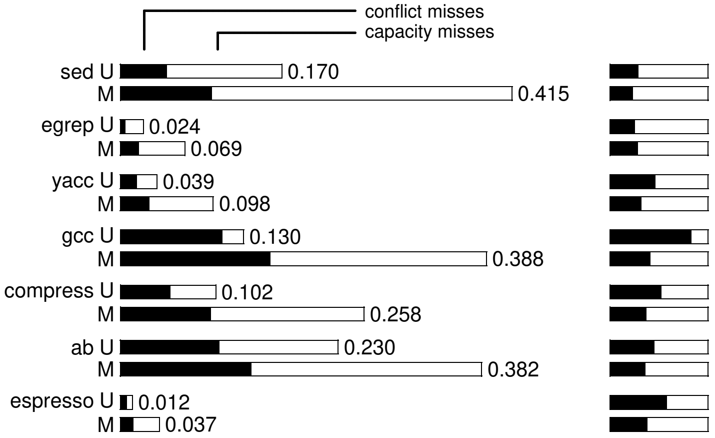
## Re-analysis of Chen & Bershad's Data




MCPI for Ultrix and Mach



## Re-analysis of Chen & Bershad's Data...




MCPI caused by cache misses: conflict (black) vs capacity (white)



## Conclusion

- Mach system (kernel + UNIX server + emulation library) is too big!
- UNIX server is essentially same.
- Emulation library is irrelevant (according to Chan & Bershad).

⇒ Mach $\mu$-kernel working set is too big

**Can $\mu$-kernels be build which avoid these problems?**

## REQUIREMENTS FOR $\mu$-KERNELS:

- Fast (system call costs, IPC costs)
- Small (big $\Rightarrow$ slow)

$\Rightarrow$ Must be well designed, providing a minimal set of operations.

Can this be done?

## ARE HIGH SYSTEM CALL & CONTEXT SWITCHING COSTS ESSENTIAL?

Example: kernel call cost on i486

- Mach kernel call: 900 cycles
- Inherent (hardware-dictated cost): 107 cycles.
  $\Rightarrow$ 800 cycles kernel overhead.
- L4 kernel call: 123–180 cycles (15–73 cycles overhead).

$\Rightarrow$ Mach's performance is a result of Mach's design and implementation not the $\mu$-kernel concept!

## $\mu$-KERNEL DESIGN PRINCIPLES

$\mu$-kernels must be [Lie96]:

**Minimal:** If it doesn't *have to be* in the kernel, it *shouldn't* be in the kernel

**Providing the right abstractions** which can be made fast **and** allow efficient implementation of services

**Well written:** It pays to shave a few cycles off a TLB refill handler or the IPC path

**Unportable:** $\mu$-kernel must be targeted to specific hardware
no problem if it's small, and higher layers are portable
Example: Liedtke reports significant rewrite of memory management when porting from 486 to Pentium
$\Rightarrow$ "abstract hardware layer" is too costly

## WHAT MUST A $\mu$-KERNEL PROVIDE?

- Virtual memory/address spaces,
- threads,
- *fast* IPC,
- unique identifiers (for IPC addressing).

## $\mu$-KERNEL DOESN'T NEED TO PROVIDE:

- file system
  use user-level server (as in Mach);
- device drivers
  user-level driver invoked via interrupt (= IPC);
- page-fault handler.



## L4 IMPLEMENTATION TECHNIQUES

- Appropriate system calls (*reply & receive next* $\Rightarrow$ fewer syscalls/RPC).
- Rich message structure (value and reference parameters in message).
- Copy message only once (i.e. **not** user$\rightarrow$kernel$\rightarrow$user).
- One kernel stack (for interrupt handling) per thread (in TCB).
- TCBs in VM
- Thread UIDs (containing thread ID).
- Short messages in registers.
- As many syscall parameters in registers as possible.
- "Hottest" kernel code is shortest.
- Kernel IPC code on single page, critical data on single page.
- Many H/W specific optimisations .



## PERFORMANCE

| *System* | *CPU* | *MHz* | RPC $\mu$s | cyc/IPC | semantics |
|---|---|---|---|---|---|
| L4 | R4600 | 100 | 1.7 $\mu$s | 100 | full |
| L4 | Alpha | 433 | 0.2 $\mu$s | 45 | full |
| L4 | Pentium | 166 | 1.5 $\mu$s | 121 | full |
| L4 | 486 | 50 | 10 $\mu$s | 250 | full |
| QNX | 486 | 33 | 76 $\mu$s | 1254 | full |
| Mach | R2000 | 16.7 | 190 $\mu$s | 1584 | full |
| SCR RPC | CVAX | 12.5 | 464 $\mu$s | 2900 | full |
| Mach | 486 | 50 | 230 $\mu$s | 5750 | full |
| Amoeba | 68020 | 15 | 800 $\mu$s | 6000 | full |
| Spin | Alpha 21064 | 133 | 102 $\mu$s | 6783 | full |
| Mach | Alpha 21064 | 133 | 104 $\mu$s | 6916 | full |
| Exo-tlrpc | R2000 | 116.7 | 6 $\mu$s | 53 | restricted |
| Spring | SparcV8 | 40 | 11 $\mu$s | 220 | restricted |
| DP-Mach | 486 | 66 | 16 $\mu$s | 528 | restricted |
| LRPC | CVAX | 12.5 | 157 $\mu$s | 981 | restricted |



## REFERENCES

[Ber92] Brian N. Bershad. The increasing irrelevance of IPC performance for microkernel-based operating systems. In *Proc. USENIX W. Microkernels & other Kernel Arch.*, pages 205–211, Seattle, WA, USA, April 1992.

[CB93] J. Bradley Chen and Brian N. Bershad. The impact of operating system structure on memory system performance. In *Proc. 14th ACM Symp. OS Princip.*, pages 120–133, Asheville, NC, USA, December 1993.

[Lie95] Jochen Liedtke. On $\mu$-kernel construction. In *Proc. 15th ACM Symp. OS Princip.*, pages 237–250, Copper Mountain, CO, USA, December 1995.

[Lie96] Jochen Liedtke. Towards real microkernels. *Comm. ACM*, 39(9):70–77, September 1996.

[Ous90] J.K. Ousterhout. Why aren't operating systems getting faster as fast as hardware? In *Proc. 1990 Summer USENIX Techn. Conf.*, pages 247–56, June 1990.