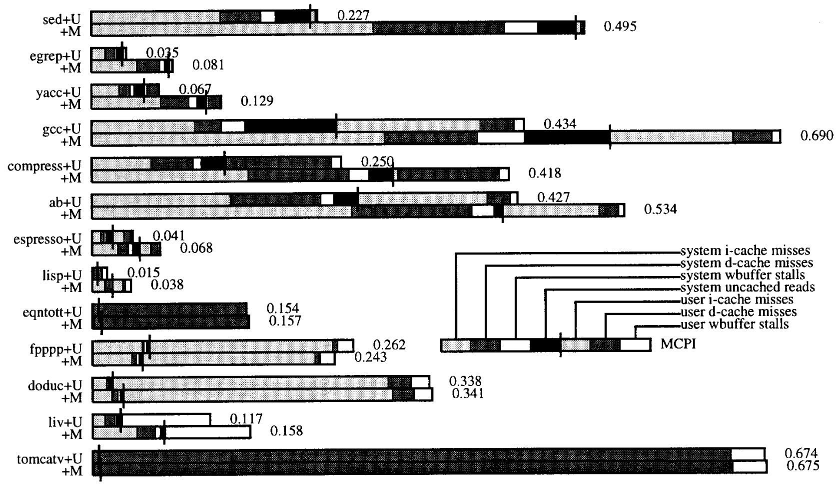
Critique of Microkernel Architectures

- First generation μ-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.

  Analyse MCPI (memory cycles per instruction):

Ultrix vs. Mach MCPI



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

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
⇒ split system/user caches wouldn't help

Diversion: Origin of cache misses

- data or instructions are associated with cache locations according to some algorithm (hash)
- many addresses hash to same cache location
- if several items map to same cache location, may be able to store
  - only one (*direct-mapped cache*)
  - 2, 4, any (*2-way, 4-way, fully associative cache*)

⇒ cache misses can have two reasons:
  - capacity misses: cache too small to hold all data
  - conflict misses: space in cache, but too much gets mapped to same cache location

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 μ-kernel performance

- system call costs are (inherently?) high
  (typically hundreds of cycles, 900 for Mach/i486)
- context (address-space) switching costs are (inherently?) high

Things have been getting worse (in terms of cycles) with increasing CPU/memory speed ratios
⇒ IPC (involving system calls and context switches) is inherently expensive

For a study of such effects see e.g.:
J.K. Ousterhout: Why aren't operating systems getting faster as fast as hardware? In *Proc. 1990 Summer USENIX Conf.*, pages 247–56, June 1990.

So, what's wrong?

μ-kernels heavily depend on IPC

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

A Critique of the Critique

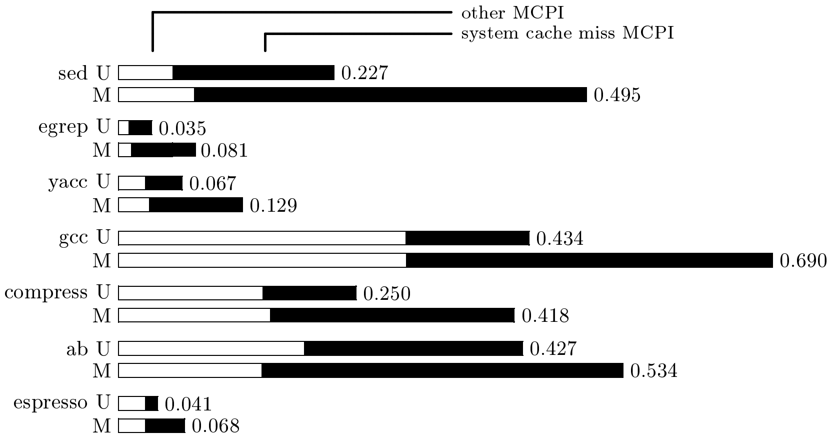
Data presented earlier

- 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

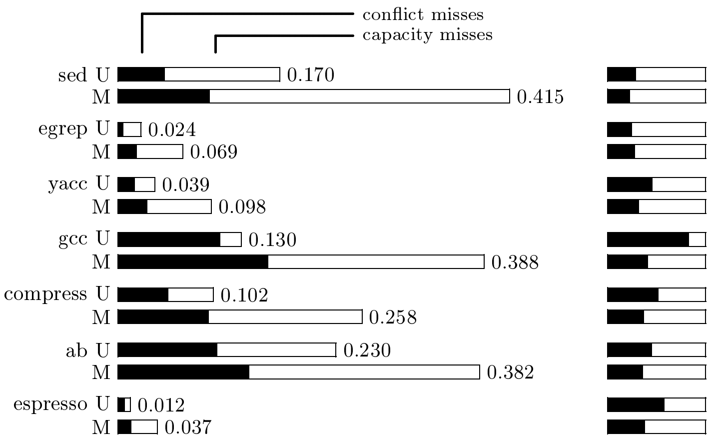
J. Liedtke: On μ-kernel construction. In *Proc. 15th ACM Symp. OS Princip.*, pages 237–250, Copper Mountain, CO, USA, December 1995.

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 vs capacity

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 μ-kernel working set is too big

Can μ-kernels be build which avoid these problems?

Requirements for μ-kernels

- fast (system call costs, IPC costs)
- small (big ⇒ slow)

⇒ must be well designed
providing a minimal set of operations

Can this be done?

0

Are high system call & context switching costs essential?

Example: kernel call cost on i486

0Mach kernel call: 900 cycles 1 inherent (hardware-dictated cost): 107 cycles
⇒ 800 cycles kernel overhead 2 L4 kernel call: 123–180 cycles (15–73 cycles overhead) 3 Mach's performance is a result of Mach's design and implementation
not the μ-kernel concept!

⇒ μ-kernel Design Principles

μ-kernels must be:

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: μ-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
⇒ "abstract hardware layer" is too costly

What must a μ-kernel provide?

- virtual memory/address spaces
- threads
- *fast* IPC
- unique identifiers (for IPC addressing)

μ-kernel doesn't need to provide

- file system
  use user-level server
- device drivers
  user-level driver invoked via interrupt (= IPC)

L4 implementation techniques

- appropriate system calls (*reply & receive next* ⇒ fewer syscalls/RPC)
- rich message structure (value and reference parameters in message)
- copy message only once (i.e. **not** user→kernel→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 μs	cyc/IPC	semantics
L4	R4600	100	1.7 μs	98	full
L4	Alpha	433	0.2 μs	45	full
L4	Pentium	166	1.5 μs	121	full
L4	486	50	10 μs	250	full
QNX	486	33	76 μs	1254	full
Mach	R2000	16.7	190 μs	1584	full
SCR RPC	CVAX	12.5	464 μs	2900	full
Mach	486	50	230 μs	5750	full
Amoeba	68020	15	800 μs	6000	full
Spin	Alpha 21064	133	102 μs	6783	full
Mach	Alpha 21064	133	104 μs	6916	full
Exo-tlrpc	R2000	116.7	6 μs	53	restricted
Spring	SparcV8	40	11 μs	220	restricted
DP-Mach	486	66	16 μs	528	restricted
LRPC	CVAX	12.5	157 μs	981	restricted

[1] 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.

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

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

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