Page Tables for 64-bit Address Spaces

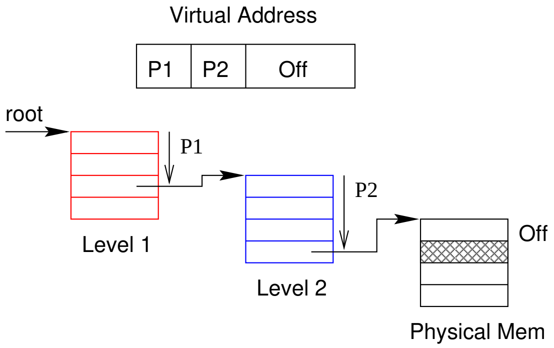
- Classical page table structures were designed for
  - ➜ 32-bit address spaces, and
  - ➜ the Unix two-segment model.
- How do they perform in large (64-bit) address spaces?
- How do they deal with sparse address-space use?
- What are the alternatives?

For a discussion see [JM98b, JM98a, HH93, Elp99].

Page Table Issues

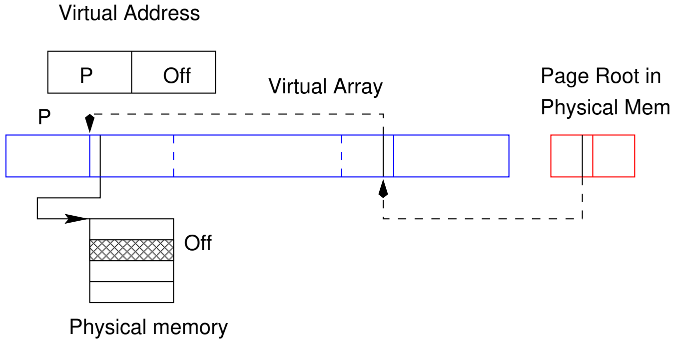
- Access time (best/worst), number of memory accesses
- Cache friendliness
- Insertion/deletion costs
- Size
- Support for sharing/aliasing
- Support for operations on page clusters

Multi-level Page Table



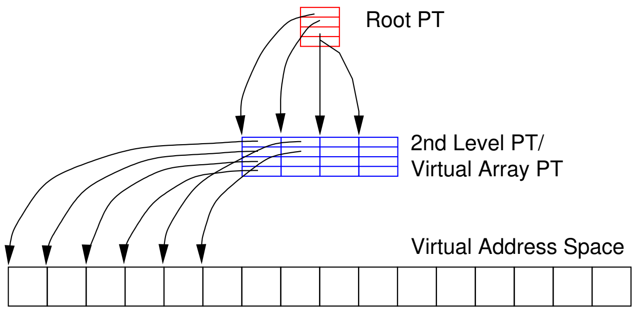
Motorola, Sparc, ... Also:
- ➜ hierarchical page table
- ➜ forward-mapped page table
- ➜ top-down page table

Virtual Linear Array Page Table

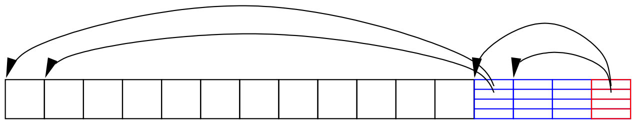


E.g. VAX page table [LL82, CE85], also used on MIPS [KH92]. Also called:
- bottom-up page table

MPT vs VLAPT

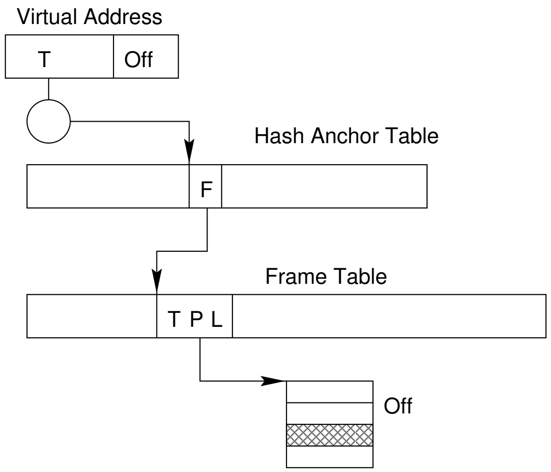


Putting the page table into the mapped address space:



 © Gernot Heiser 2000

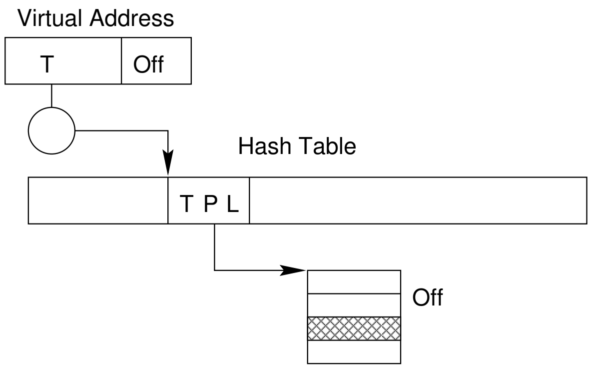
Inverted Page Table



IBM [Coc81, CM88]

 © Gernot Heiser 2000

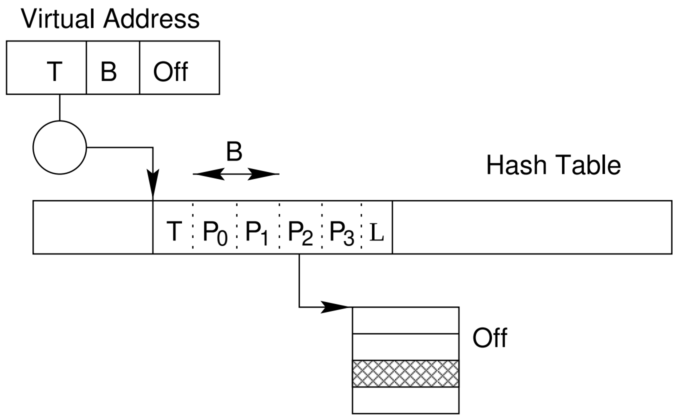
Hashed Page Table



Monads [RA85], HP-PA [Lee89], PowerPC

 © Gernot Heiser 2000

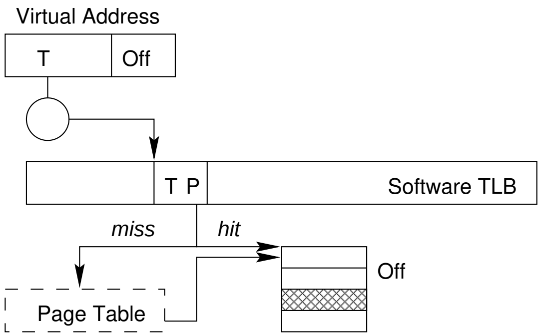
Clustered Page Table



Talluri [THK95, Tal95], also MIPS R4k [Hei91].

 © Gernot Heiser 2000

Software TLB

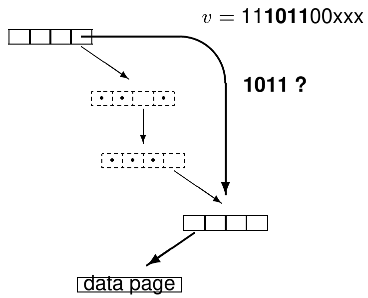


Also called:
- secondary TLB
- TLB cache
- translation storage buffer

Lookup in H/W [HH93], S/W [BKW94], H/W supp. [Sun97].

 © Gernot Heiser 2000

Guarded Page Tables



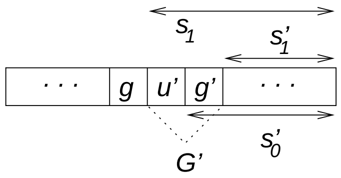
Idea: strip *guard* (**1011**) and store with pointer [Lie93, Lie96].

Called *path compression*, results in *Patricia tree* [GBY90].

 © Gernot Heiser 2000

GPT Node Implementation: Liedtke & Elphinstone

G	s_1	s'_0	child pointer
...	...	...	...
...	...	...	...
...	...	...	...



s_1**:** # untranslated address bits (guard & node index stripped)

s_0**:** # address bits to be translated at node (s'_0 for child node)

$s_1 - s'_0$**:** $\log$ of child node size (# child index bits)

$s_0 - s_1$**:** # guard bits

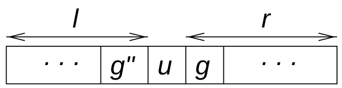
G**:** extended guard (index bits + guard)

u**:** node index

 © Gernot Heiser 2000

Alternative Implementation: Szmajda

l	r	child pointer
...	...	...
...	...	...
...	...	...



- Guard value is ignored during node traversal
- Leaf nodes store complete VPN tag (incl. guards)
- Guard match checked after traversal

Implementation in progress on MIPS R4700 [AN95, Szm99]

 © Gernot Heiser 2000

How does one compare page tables?

- TLB refill performance.
- Memory overhead.
- Manipulation performance ((un)mapping)
- Setup and tear-down costs.

 © Gernot Heiser 2000

Testbed

- Hardware:
  - U4600 research/teaching platform
  - 100MHz MIPS R4700 CPU
  - 64MB RAM
- Software: L4/MIPS microkernel
  - implemented various PT structures in kernel
  - instrumented kernel for analysis of overheads
  - GPT implementation [LE96] uses fixed node size (varied between runs)

 © Gernot Heiser 2000

Conventional Applications Used

source	*name*	*size (Mb)*	*type*	*remarks*
SPEC	go	0.8	I	game of go
	swim	14.2	F	PDE solver
	gcc	9.3	I	GNU C compiler
	compress	34.9	I	file (un)compression
	apsi	2.2	F	PDE solver
	wave5	40.4	F	PDE solver
Alburto	c4	5.1	I	game of connect four
	nsieve	4.9	I	prime number generator
	heapsort	4.0	I	sorting large arrays
	mm	7.7	F	matrix multiply
	tfftdp	4.0	F	fast fourier transform

 © Gernot Heiser 2000

Sparse Benchmarks Used

uniform:

- uniformly distributed pages in 1TB address space
- uses from 64 (256kB) to 8,192 (32MB) pages

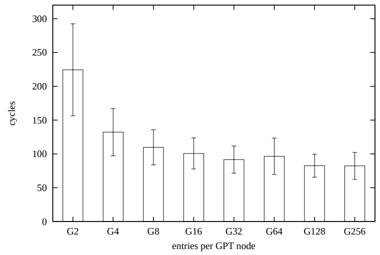
file:

- uniformly distributed multi-page objects
- object size follows empirical file-size distribution [Elp93]
- uses from 64 (2,020kB) to 15,826 (62MB) objects

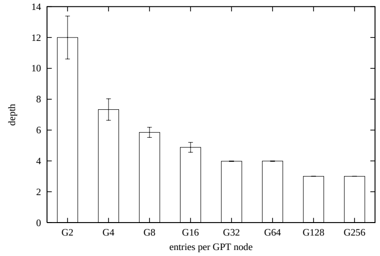
 © Gernot Heiser 2000

GPT TLB refill performance

Refill Time



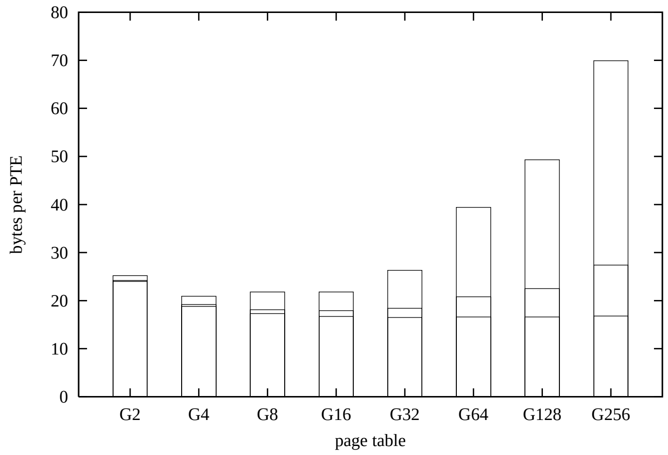
Tree Depth



Conventional benchmarks. Fastest possible refill: 19 cycles.

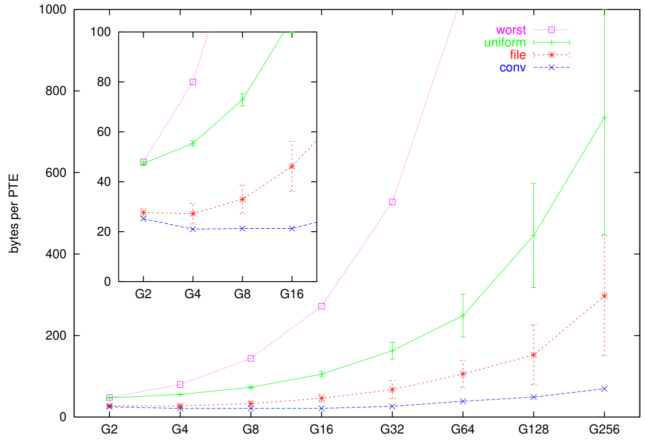
 © Gernot Heiser 2000

GPT size behaviour under conventional Apps



 © Gernot Heiser 2000

GPT size behaviour under sparse conditions



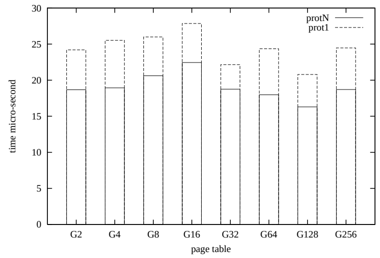
“Worst”: Theoretical worst case

“Conv”: Worst result of conventional BM.

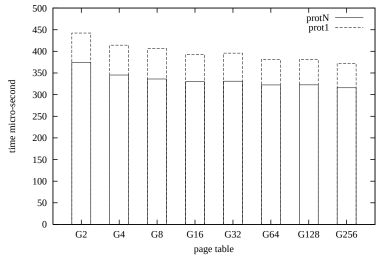
 © Gernot Heiser 2000

(Un)Maping Performance

Cached

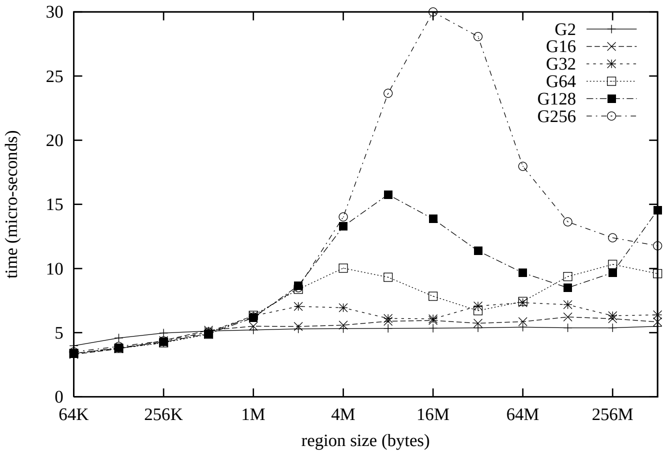


Uncached

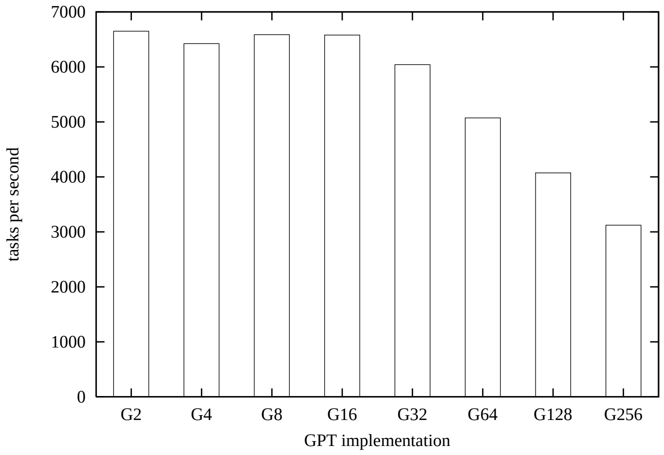


 © Gernot Heiser 2000

SPARSE UNMAPPING PERFORMANCE



TASK CREATION PERFORMANCE



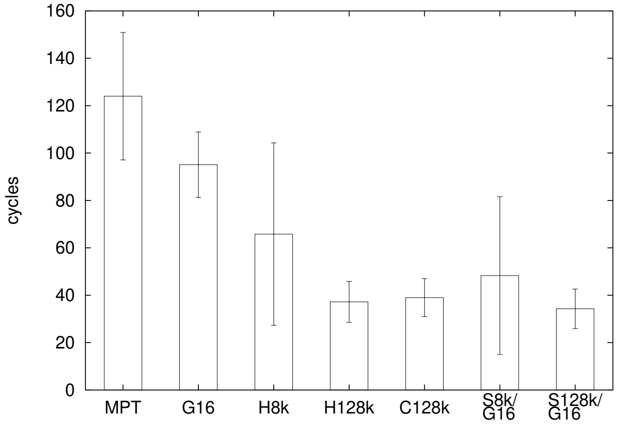
GPT SUMMARY

- TLB refill slow (tree deep for small, medium node sizes).
- Overhead low for smaller node sizes, even for sparse case.
- Sparse manipulation costs constant for smaller node sizes.
- Setup and tear-down costs "scale".
- 16 seems to be good compromise for *fixed* size GPT nodes

ADDING A SOFTWARE TLB

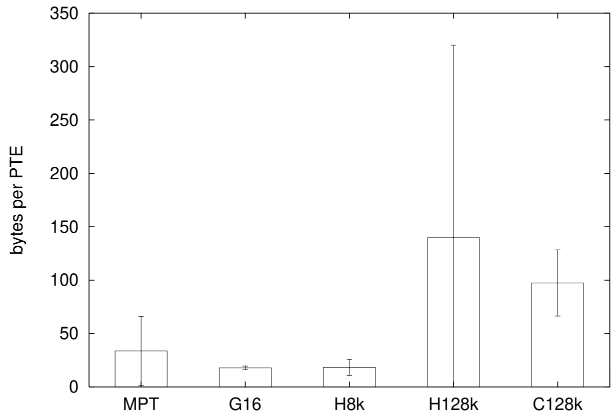
- Should improve refill performance.
- Low memory overhead if shared between all tasks.
- Desirable properties of GPTs preserved.
- Compare 16-entry GPTs to:
  - GPTs with a STLB,
  - a small and large hashed PT,
  - a small and large clustered PT,
  - and a multi-level page table.

Refill Performance



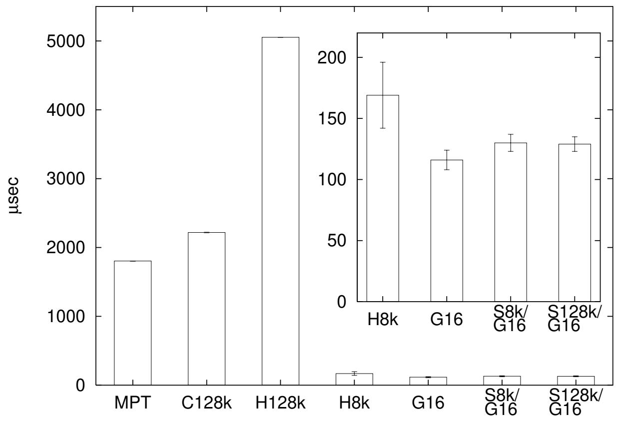
Average TLB refill cost for conventional benchmarks

Page Table Size

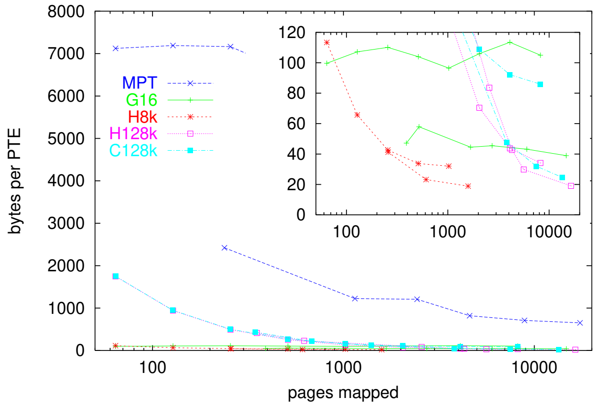


Average size for conventional benchmarks (ignoring STLB size).

Task creation/deletion cost

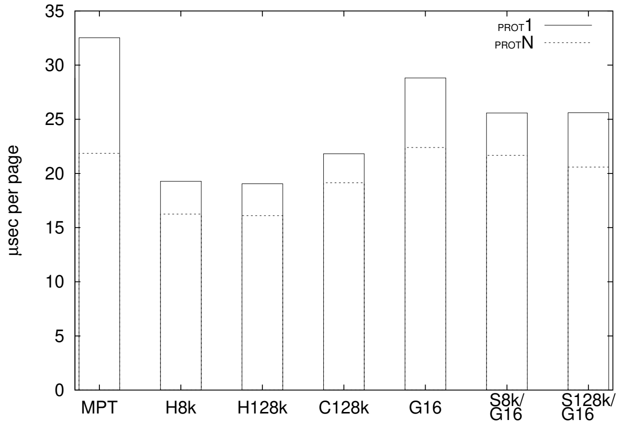


Size for Sparse Address-Space Use



Page table size for sparse benchmarks: “uniform” (top), “file”.

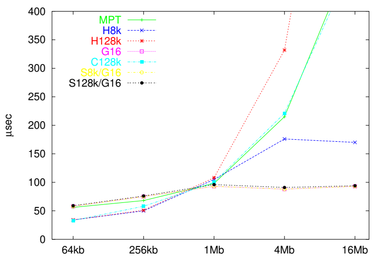
(Un)Map Costs (Conventional)



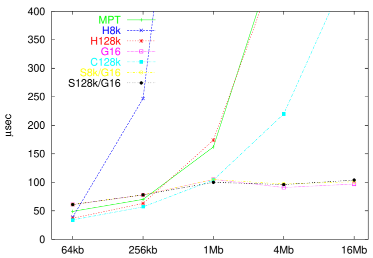
© Gernot Heiser 2000

Sparse Unmapping Comparison

Minimal Size



Additional 64Mb region



© Gernot Heiser 2000

Conclusion

- GPTs adapt to sparsity, but inherent cost due to tree depth
- refill from hashed table is always fastest
- GPT with STLB generally performs as good as other PTs
- GPT with STLB outperform others for:
  - operations on sparse regions.
  - setup and tear-down costs.
- GPTs with STLB cope with large class of applications
- Potential improvements by adaptable node size

For details see [EHL99, Elp99]

© Gernot Heiser 2000

References

[AN95] Arne Andersson and Stefan Nilsson. Efficient implementation of suffix trees. *Softw: Pract. & Exp.*, 25:129–141, 1995.

[BKW94] Kavita Bala, M. Frans Kaashoek, and William E. Weihl. Software prefetching and caching for translation lookaside buffers. In *Proc. 1st Symp. Operat. Syst. Design & Implem.*, pages 243–253, Monterey, CA, USA, 1994. USENIX/ACM/IEEE.

[CE85] Douglas W. Clark and Joel S. Emer. Performance of the VAX-11/780 translation buffer: Simulation and measurement. *ACM Trans. Comp. Syst.*, 3:31–62, 1985.

[CM88] Albert Chang and Mark F. Mergen. 801 Storage: Architecture and programming. *ACM Trans. Comp. Syst.*, 6:28–50, 1988.

[Coc81] John Cocke. Virtual to real address translation using hashing. *IBM Techn. Disclosure Bull.*, 24(6), November 1981.

© Gernot Heiser 2000

[EHL99] Kevin Elphinstone, Gernot Heiser, and Jochen Liedtke. Page tables for 64-bit computer systems. In *Proc. 4th Australasian Comp. Arch. Conf.*, pages 211–226, Auckland, New Zealand, January 1999. Springer Verlag. Available from URL http://www.cse.unsw.edu.au/~disy/papers/.

[Elp93] Kevin Elphinstone. Address space management issues in the Mungi operating system. Technical Report UNSW-CSE-TR-9312, School Comp. Science & Engin., University NSW, Sydney 2052, Australia, November 1993.

[Elp99] Kevin Elphinstone. *Virtual Memory in a 64-bit Microkernel*. PhD thesis, School Comp. Science & Engin., University NSW, Sydney 2052, Australia, March 1999. Available from http://www.cse.unsw.edu.au/~disy/papers/.

[GBY90] G.H. Gonnet and R. Baeza-Yates. *Handbook of Algorithms and Data Structures*. Addison-Wesley, 2^{nd} edition, 1990.

 © Gernot Heiser 2000

[Hei91] Joe Heinrich. *MIPS R4000 Microprocessor User's Manual*. MIPS Computer Systems, Inc., Sunnyvale, CA, USA, 1991.

[HH93] Jerry Huck and Jim Hays. Architectural support for translation table management in large address space machines. In *Proc. 20th Int. Symp. Comp. Arch.*, pages 39–50. ACM, 1993.

[JM98a] Bruce Jacob and Trevor Mudge. Virtual memory in contemporary microprocessors. *IEEE Micro*, pages 60–75, July–August 1998.

[JM98b] Bruce Jacob and Trevor Mudge. Virtual memory: Issues of implementation. *IEEE Comp.*, 31:33–43, June 1998.

[KH92] Gerry Kane and Joe Heinrich. *MIPS RISC Architecture*. Prentice Hall, 1992.

[LE96] Jochen Liedtke and Kevin Elphinstone. Guarded page tables on MIPS R4600 or an exercise in architecture-dependent micro optimization. *Operat. Syst. Rev.*, 30(1):4–15, January 1996.

 © Gernot Heiser 2000

[Lee89] Ruby B. Lee. Precision architecture. *IEEE Comp.*, 22(1):78–91, January 1989.

[Lie93] Jochen Liedtke. A basis for huge fine-grained address spaces and user level mapping. In *Proc. 7th European Conf. Obj. Orient. Progr. (ECOOP) Workshop on Granularity of Objects in Distributed Systems (GODS'93)*, Kaiserslautern, Germany, July 1993.

[Lie96] Jochen Liedtke. *On the Realization Of Huge Sparsely-Occupied and Fine-Grained Address Spaces*. Oldenbourg, Munich, Germany, 1996.

[LL82] Henk M. Levy and P. H. Lipman. Virtual memory management in the VAX/VMS operating system. *IEEE Comp.*, 15(3):35–41, March 1982.

[RA85] John Rosenberg and David Abramson. MONADS-PC—a capability-based workstation to support software engineering. In

 © Gernot Heiser 2000

Proc. 18th Hawaii Int. Conf. System Sciences, volume 1, pages 222–31. IEEE, 1985.

[Sun97] Sun Microsystems, Palo Alto, CA, USA. *UltraSPARC User's Manual*, July 1997. URL http://www.sun.com/microelectronics/-manuals/ultrasparc/802-7220-02.pdf.

[Szm99] Christian Szmajda. A new virtual memory implementation for L4/MIPS. BE thesis, School Comp. Science & Engin., University NSW, Sydney 2052, Australia, November 1999.

[Tal95] Madhusudhan Talluri. *Use of Superpages and Subblocking in the Address Translation Hierarchy*. Phd thesis, University of Wisconsin-Madison Computer Sciences, 1995. Technical Report #1277.

[THK95] Madhusudha Talluri, Mark D. Hill, and Yousef A. Khalid. A new page table for 64-bit address spaces. In *Proc. 15th ACM Symp. OS Princip.*, pages 184–200, Copper Mountain Resort, Co, USA,

 © Gernot Heiser 2000

December 1995. ACM.

© Gernot Heiser 2000