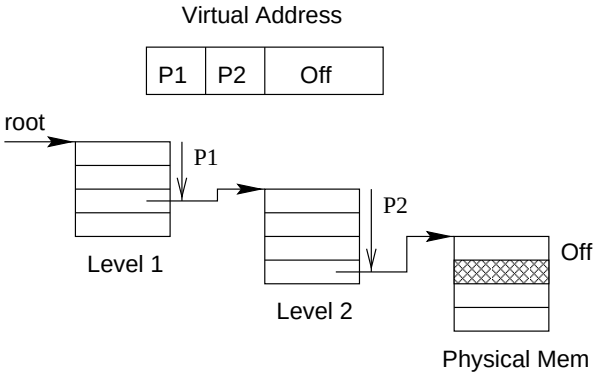
Page Tables for 64-bit Address Spaces

- ☞ 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?

 © Gernot Heiser 1998

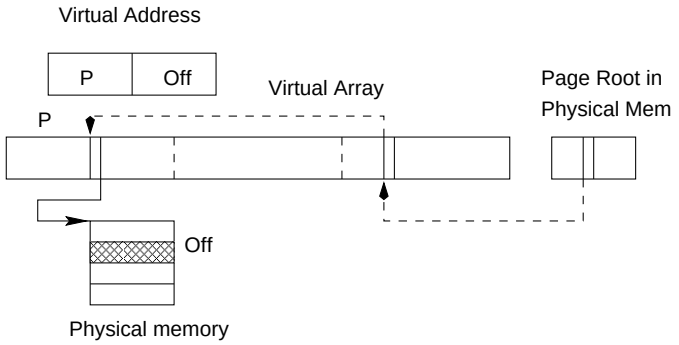
Multi-level Page Table



Also called:
- hierarchical page table
- forward-mapped page table
- top-down page table

 © Gernot Heiser 1998

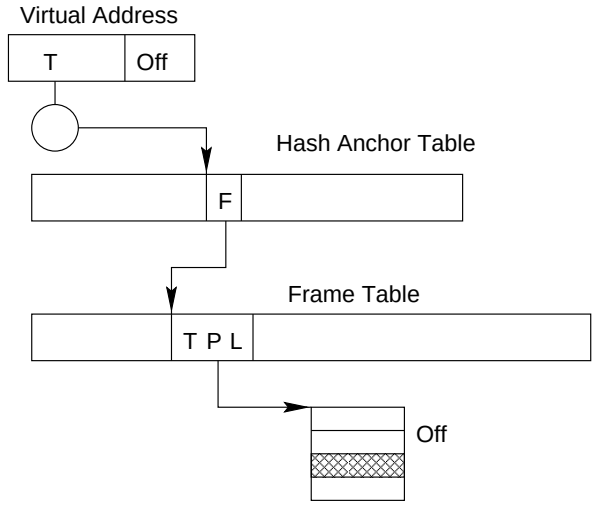
Virtual Linear Array Page Table



Also called:
- bottom-up page table

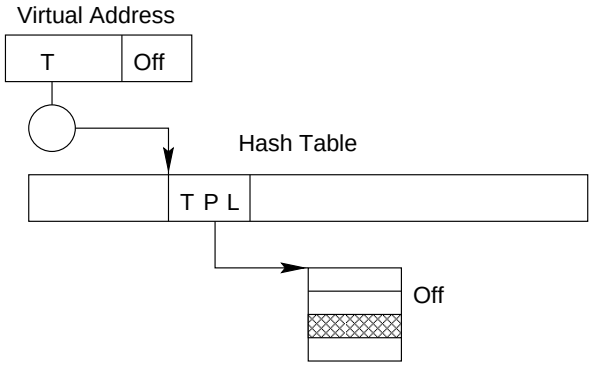
 © Gernot Heiser 1998

Inverted Page Table



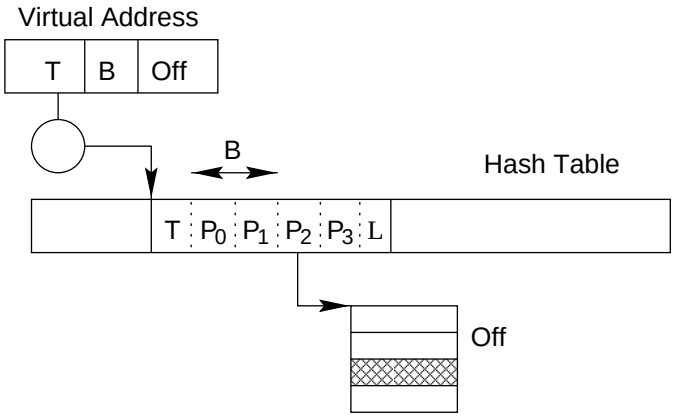
 © Gernot Heiser 1998

Hashed Page Table



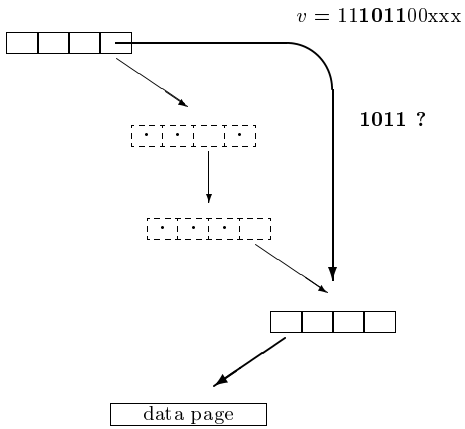
 © Gernot Heiser 1998

Clustered Page Table



 © Gernot Heiser 1998

Guarded Page Tables



Idea: strip *guard* (**1011**)

 © Gernot Heiser 1998

How does one compare page tables?

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

 © Gernot Heiser 1998

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

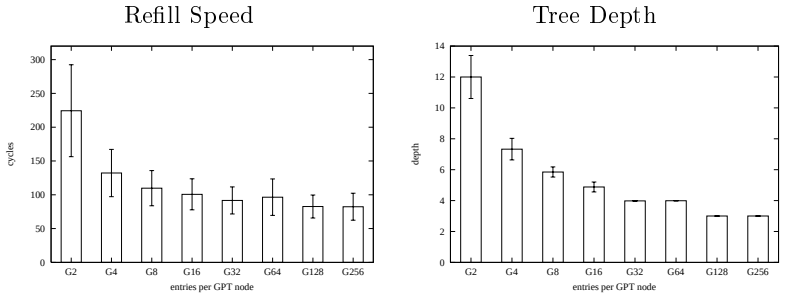
 © Gernot Heiser 1998

Conventional Applications Used

	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

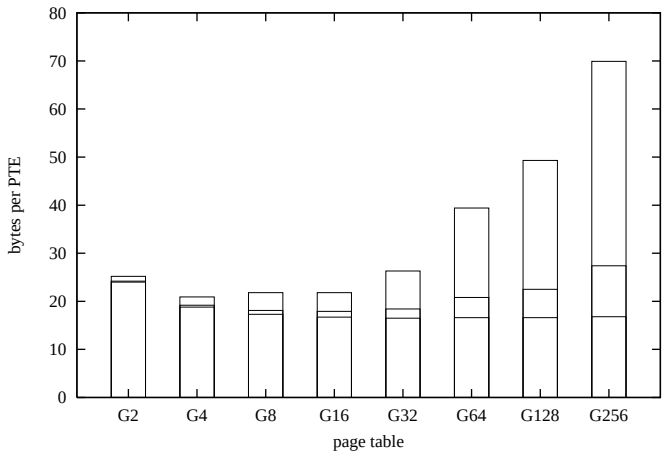
 © Gernot Heiser 1998

GPT TLB refill performance



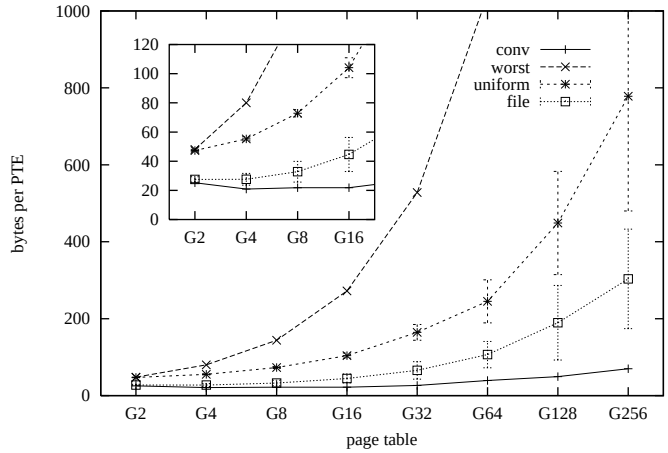
 © Gernot Heiser 1998

GPT size behaviour under conventional Apps



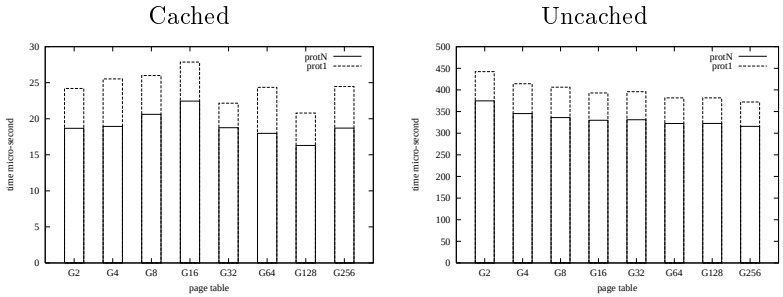
 © Gernot Heiser 1998

GPT size behaviour under sparse conditions



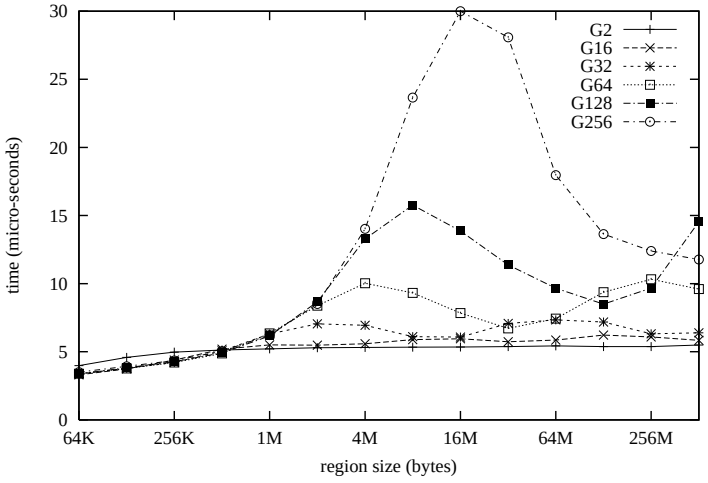
 © Gernot Heiser 1998

(Un)Maping Performance



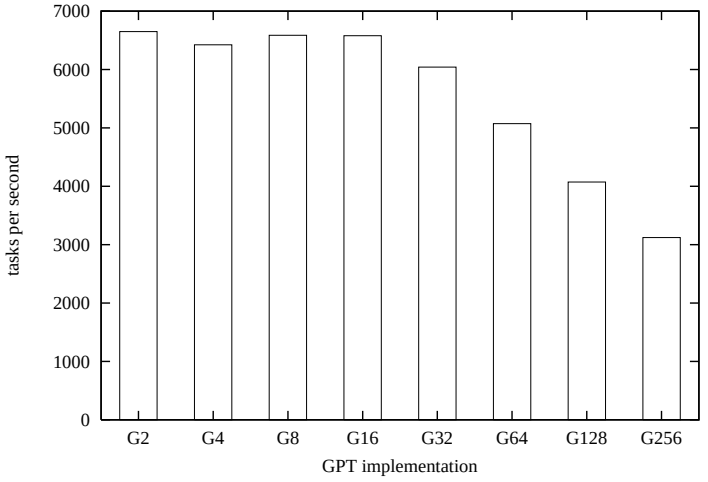
 © Gernot Heiser 1998

Sparse Unmapping Performance



 © Gernot Heiser 1998

Task Rate Performance



 © Gernot Heiser 1998

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".

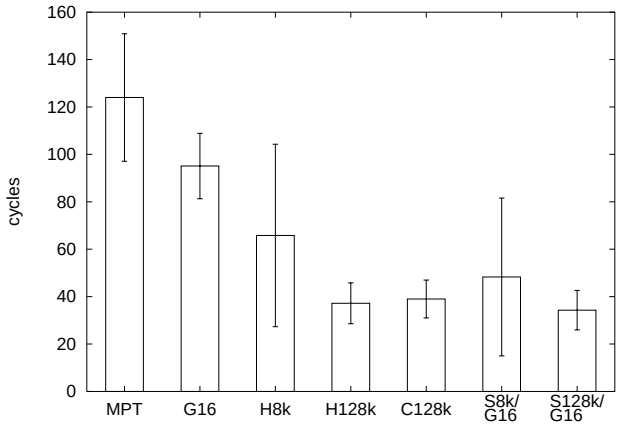
 © Gernot Heiser 1998

Adding a software TLB cache (STLB)

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

 © Gernot Heiser 1998

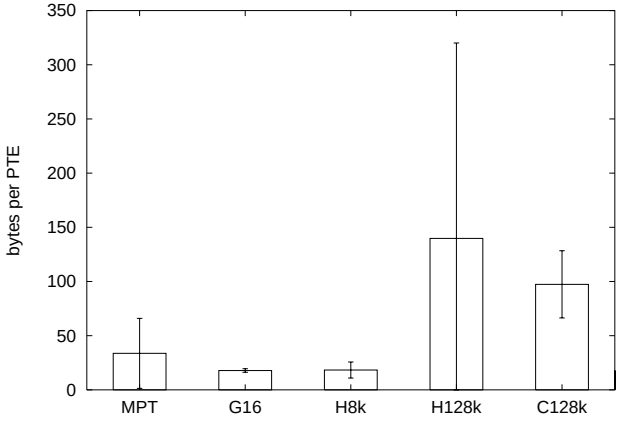
Refill Performance



Average TLB refill cost for conventional benchmarks

 © Gernot Heiser 1998

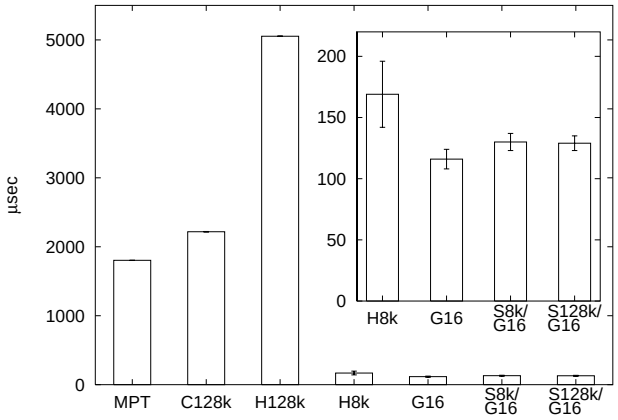
Page Table Size



Average size for conventional benchmarks

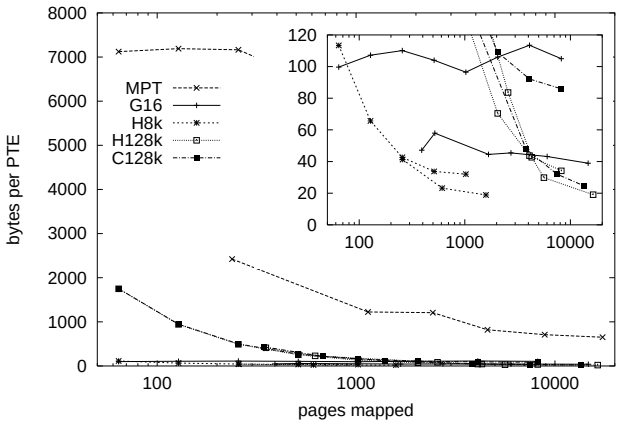
 © Gernot Heiser 1998

Task creation/deletion cost



 © Gernot Heiser 1998

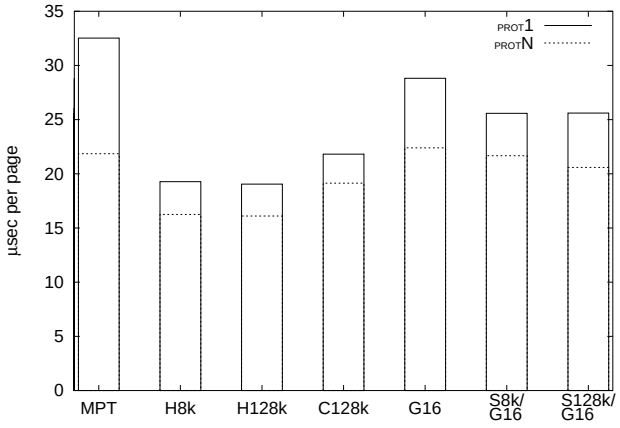
Size for Sparse Address-Space Use



Page table size for sparse benchmarks ("uniform" and "file")

 © Gernot Heiser 1998

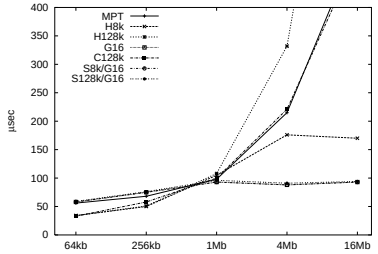
(Un)Map Costs (Conventional)



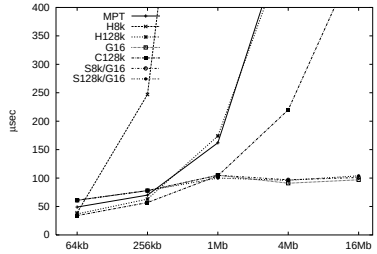
 © Gernot Heiser 1998

Sparse Unmapping Comparison

Minimal Size



Additional 64Mb region



 © Gernot Heiser 1998

Conclusion

- GPTs adapt well to sparsity
- inherent cost high due to depth of tree
- refill from hashed table is always fastest
- backing STLB with a GPT performs at least as well as other page table schemes in most situations, and clearly outperforms others in:
  - operations on sparse regions.
  - setup and tear-down costs.
- GPTs with an STLB adapt, ie the don't need tuning to expected application behaviour.

 © Gernot Heiser 1998

References

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

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

[3] Jerry Huck and Jim Hays. Architectural support for translation table management in large address space machines. In *Proceedings of the 20th International Symposium on Computer Architecture*, pages 39–50. ACM, 1993.

[4] Douglas W. Clark and Joel S. Emer. Performance of the VAX-11/780 translation buffer: Simulation and measurement. *ACM Transactions on Computer Systems*, 3:31–62, 1985.

[5] John Cocke. Virtual to real address translation using hashing. *IBM Technical Disclosure Bulletin*, 24(6), November 1981.

 © Gernot Heiser 1998

[6] Albert Chang and Mark F. Mergen. 801 Storage: Architecture and programming. *ACM Transactions on Computer Systems*, 6:28–50, 1988.

[7] John Rosenberg and David Abramson. MONADS-PC—a capability-based workstation to support software engineering. In *Proceedings of the 18th Hawaii International Conference on System Sciences*, volume 1, pages 222–31. IEEE, 1985.

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

[9] Madhusudha Talluri, Mark D. Hill, and Yousef A. Khalid. A new page table for 64-bit address spaces. In *Proceedings of the 15th ACM Symposium on OS Principles*, pages 184–200, Copper Mountain Resort, Co, USA, December 1995. ACM.

[10] Jochen Liedtke. A basis for huge fine-grained address spaces and user level mapping. In *Proceedings of the 7th European Conference on Object Oriented Programming (ECOOP) Workshop on Granularity of*

 © Gernot Heiser 1998

Objects in Distributed Systems (GODS'93), Kaiserslautern, Germany, July 1993.

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

[12] Kevin Elphinstone, Gernot Heiser, and Jochen Liedtke. Page tables ofr 64-bit computer systems. In *Proceedings of the 4th Australasian Computer Architecture Conference*, Auckland, New Zealand, January 1999. Springer Verlag. Available as UNSW-CSE-TR-9804 from `http://www.cse.unsw.EDU.AU/~disy/papers.html`.

 © Gernot Heiser 1998