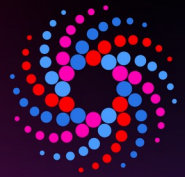


SC22
Dallas, TX | hpc accelerates.

Porting the Kitten Lightweight Kernel Operating System to RISC-V

Nicholas Gordon (me)*, Kevin Pedretti†, Jack Lange‡

*University of Pittsburgh, †Sandia National Laboratories, ‡Oak Ridge National Laboratory

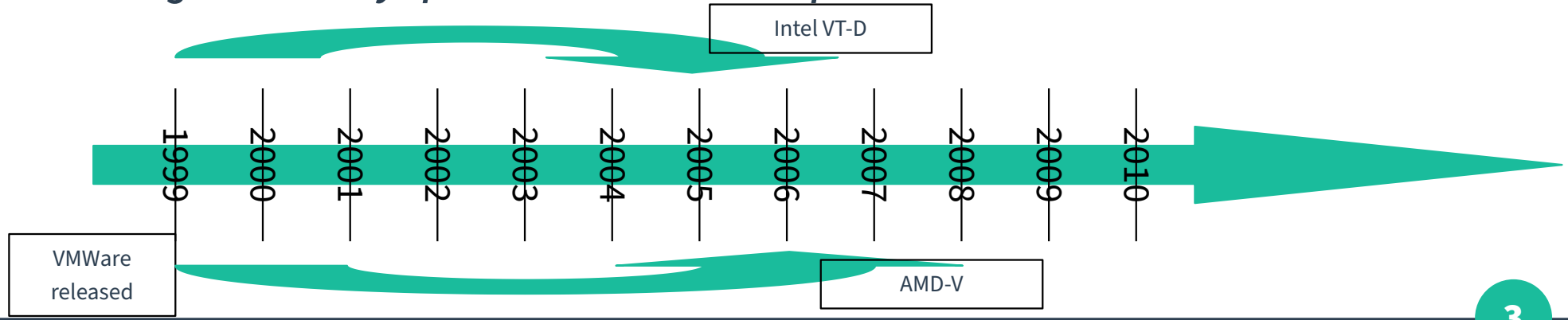
Co-Design: a pipe dream or nearly there?

- **X86 has ~40 years of legacy stapled to it by this point**
  - Still starts in real mode!
- **Linux has adapted to x86 and thus *inherited* and *integrated* that legacy.**

- **RISC-V is the new kid on the block and making waves in hardware research**
- **We need corresponding flexible, extensible system software: Kitten**

Software and Hardware: A strained relationship

- **Hardware gets designed by hardware guys, software gets designed by software guys**
  - Hardware constraints: feature size, power draw, heat dissipation, complexity
  - Systems software has constraints, too: hardware and its capabilities
- **But as each changes, catchup can take a long time!**
- ***Co-design has always promised to solve this problem***



Why is it like this? What can we do?

- **Chips are very hard to make. Circuit design, simulations, tapeouts, fabrication delays, costs, etc.**
- **Hardware features may not get used anyway:**
  - hardware task-switching, x86's rings 1 and 2, etc.
- **Co-design is becoming more practical:**
  - Reduce time/cost to prototype hardware: FPGAs
  - Reduce difficulty to extend ISAs: RISC-V
  - Reduce difficulty to test new hardware: *extensible, simple software*

RISC-V provides open, flexible, and extensible hardware

- **RISC-V *democratizes* hardware design**
  - Try asking Intel, AMD, etc. to try an experimental change
  - Xtensa controllers are an option, but not targeted at research/academia
- **RISC-V has a good supply of research hardware implementations, exploring a wide variety of architectural ideas (just a few examples):**
  - Pipelining: SHAKTI
  - Superscalar: Berkeley's RISC-V BOOM
  - VLIW: Packet Manipulator
  - Manycores: Hammerblade, Manticore
- **RISC-V has chip design tools:**
  - Chipyard, FireSim

What is the software complement?

- **The obvious choice is Linux.**
  - However, Linux's enormous complexity makes modification hard, to say the least.
  - The task struct on Linux is ~800 lines long:

```
727   struct task_struct {

1524          /*
1525           * WARNING: on x86, 'thread_struct' contains a variable-sized
1526           * structure.  It *MUST* be at the end of 'task_struct'.
1527           *
1528           * Do not put anything below here!
1529           */
1530  };
```

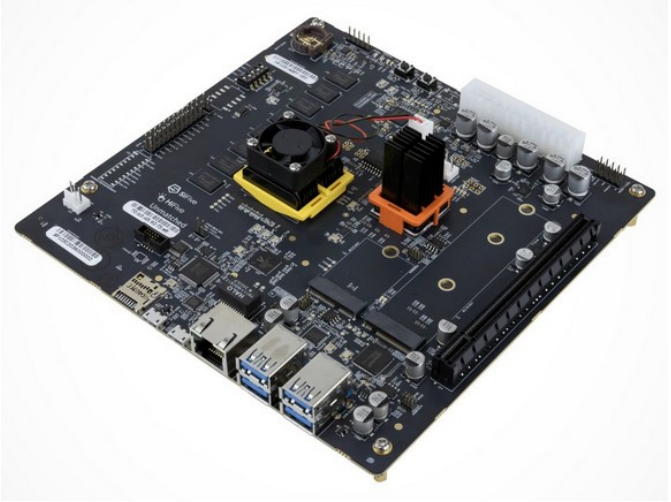
The software complement is Kitten

- **Kitten is simple but capable**
  - Fewer and simpler hardware abstractions than Linux
  - Less complexity and cruft than Linux
- **This means extending and modifying Kitten is a lot easier.**
  - No legacy support built in
  - Bootstrapping on new hardware is easier
- **We also already ported it to ARM64, in just a few months by a few people.**

- **Kitten + RISC-V is an enabling combination for co-design**

Objectives of the Port

- **Platforms: QEMU, SiFive Unmatched**
- **Benchmarks: HPCG, STREAM, RandomAccess**
- **OS Features:**
  - Single-process support
  - FPU
  - Timers
  - Interrupts
  - × Multi-process
  - × Multi-core
- **These are on the way!**
- **This took about one summer by one person.**



Some interesting lessons learned

- **Some interesting challenges faced along the way:**
  - Bootstrap
  - Paging structure
  - Linker relaxations
  - Register quantity vs ARM and thread-local storage
  - L2 cache prefetchers and performance
  - L2 cache or scratchpad memory
- **How did Kitten make this easier? How hard was this port?**

Bootstrap

- **Need an assembly stub to initialize various bits of the processor and OS (head.S usually)**
- **A complete, working head.S embodies a lot of knowledge, assumptions about hardware**
  - Don’t want to rewrite it
- **So don’t ☺. Kitten can nearly directly re-use Linux’s head.S stub**
- **Just have to “work the ends together”**
- **Estimated work time: a week**

Paging Structure

- Borrow heavily from Linux when possible, can't borrow here. *Write from scratch.*
- RISC-V paging structure is very simple. Pick a bit-width (39-bits here) and here's your format. From the Unmatched manual:

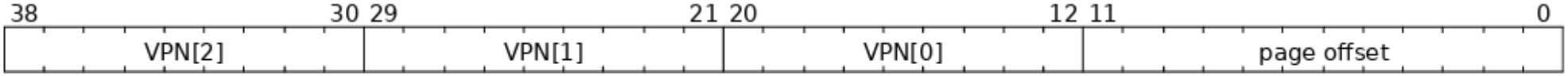


Figure 11: *Sv39 Virtual Address*

Figure 13: *Sv39 PTE Format*

Paging, continued

- **Leafs are found by checking the RWX permission bits**
  - No collapsing, inherited permissions like on x86
- **Accessed/Dirty are controlled *by the kernel*, not the hardware**
  - Though an exception will be generated if they are wrong!
- **A toggle to prevent supervisor read/write to user pages.**
- **Less C structs to manage, simpler memory code**
- **Estimated work time: 2 weeks**

Linker relaxations and maybe a gotcha?

- **RISC-V is a register-rich, but addressing mode-poor ISA.**
- **Addressing any 32-bit address requires two instructions**
- **But RISC-V specifies a *global pointer (gp)*, which elides one of the those instructions.**
  - Points to an extra ELF section, the *small* BSS (.sbss).
  - Kitten has to load this to the right place
  - Have to juggle it when switching from physical to virtual memory
- **Estimated work time: a few days**

Architectural Register Quantity and Thread-Local Storage

- **RISC-V has a many general-purpose registers, but few system registers.**
- **When switching tasks, need to store kernel stack, task struct pointer, etc.**
  - ARM64 has “banked registers” containing context at each level: stack pointer, thread pointer
- **RISC-V has one architectural “scratch” register per level**
  - We emulate Linux: put the thread info struct at offset 0 inside the task struct, and point tp to both of them
  - Fetch the kernel stack from there
  - But tp points to thread-local storage, to it has to be juggled when in kernel-mode
- **Kitten can borrow from Linux when needed, but *avoid its complexity otherwise***
- **Estimated work time: one or two days**
-

Performance and cache prefetchers

- **To assess port completeness and correctness as well as characterize hardware, we ran typical benchmarks on:**
  - QEMU and the Unmatched
  - Linux and Kitten
- **The Unmatched *lacks an L2 prefetcher.***
  - This has some consequences for memory performance.
- **Benchmarks: HPCG, Stream, RandomAccess**
  - Want to also run NAS suite in future

Performance: HPCG

- HPCG: A typical benchmark, a decent mix of memory and compute intensiveness.
- Kitten on QEMU on the workstation outperforms the Unmatched by a factor of about 2.

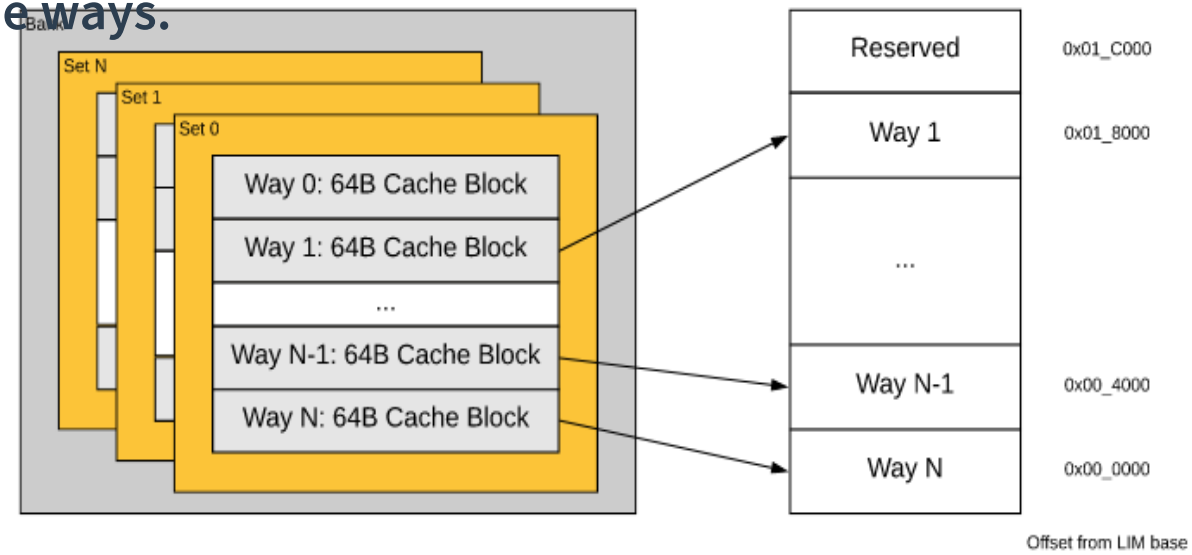
Memory Performance: STREAM and RandomAccess

- Two common benchmarks to characterize memory performance
- QEMU shows a substantial advantage in STREAM, but not in RandomAccess.

RandomAccess

SiFive Unmatched and its L2 Cache

- The SiFive Unmatched has a very flexible L2 cache that can serve as L2 cache, scratchpad, direct memory, or some mix of all three.
- At boot time, must enable cache ways.
- Enabled ways can be written to as a scratchpad (where evictions might occur at any time)
- Disabled ways can be *directly addressed* and used as ordinary RAM



Conclusion

- **Although the hardware needs improvement, early results here are promising**
- **Porting illustrates Kitten's flexibility and suitability to new, experimental hardware**

Co-design targets in the future:

- **Exploration of hardware-accelerated global address space**
- **Security extensions, including capabilities (CHERI)**
- **Other global pointers**
  - Would allow for more relaxation and thus faster access
  - Require toolchain support, like GCC on ARM64 can do
  - Easy to implement in Kitten, hard (?) to put into toolchain
- **L2 cache scratchpad/direct memory: accelerated locks, message-passing buffers, etc.**
  - A week or two to implement (?)

Thank you!

- Feel free to ask any questions you have.