NetBSD®

MIPS TECHNOLOGIES

# Improving NetBSD/mips

BSDCan 2012
The Technical BSD Conference
High value. Low cost. Something for everyone.

matt@netbsd.org

## Historical Overview

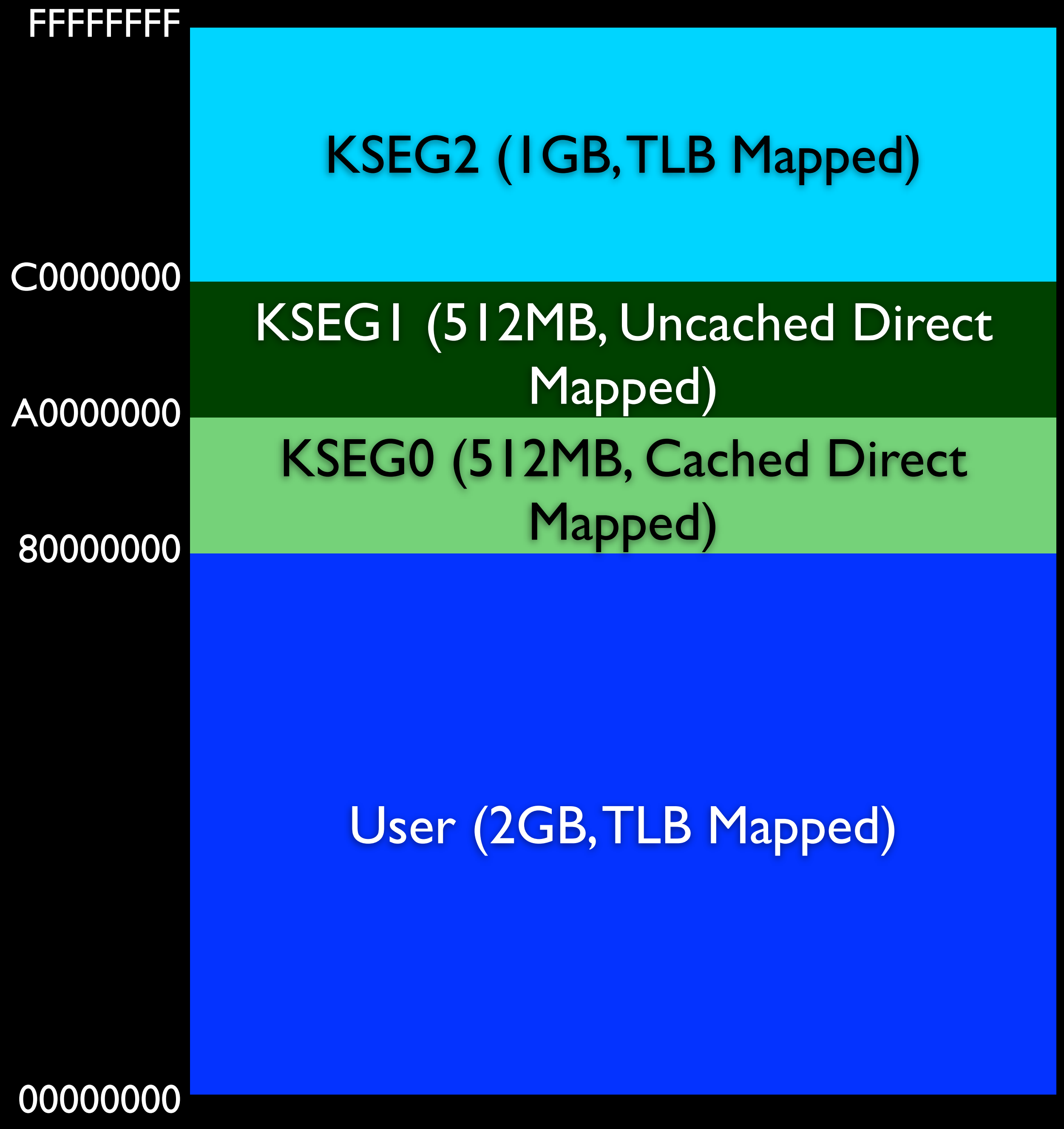
- My first MIPS experience was back in 1988 when I helped port ULTRIX to the DECstation 3100 (pmax) for Digital Equipment Corporation.
- NetBSD gained its initial MIPS support 6 years later when it imported the BSD 4.4-lite pmax code.
- Generic MIPS code was split off from pmax in 1998 when newmips was added. Most other mips ports were added between 1999 and 2003.

# The MIPS Architecture

- MIPS-1 (R3000/R2000), MIPS-2
  - 32-bit, no locking (LL/SC) instructions, simple TLB
- MIPS-3 (R4000+, 64-bit), MIPS64
  - 64-bit, locking (LL/SC) instructions, dual entry TLB
- MIPS32
  - 32-bit, locking instructions, dual entry TLB

# The MIPS Architecture Continued

- 32-bit address space is split into 4 regions (1 user, 3 kernel)
- Addresses are signed (positive addresses are user and negative address are for the kernel)
- KSEG0/KSEG1 map the first 512MB of physical memory



## The MIPS Architecture Continued

- 64-bit address space is split into equal sized 4 regions (user, supervisor, kernel direct mapped (XKPHYS), kernel TLB mapped (XKSEG))
- Addresses are signed (positive addresses are user and supervisor while negative addresses are for the kernel)
- KSEG0/KSEG1 map first 512MB of physical memory
- XKPHYS direct maps all of physical memory (using one of 8 CCAs)

## MIPS ABIs

- O32 came first and is still the only ABI used for 32-bit only CPUs.  Only the first 4 arguments are passed by register.
- O64 was the first 64-bit ABI and is simply O32 extended to use 64-bit registers.  O64 is not supported by NetBSD.
- N64 also uses 64-bit registers but simplifies stack usage and passes the first 8 arguments by register.
- N32 is N64 but addresses are 32-bit.

## RMI/NetLogic/Broadcom XLR/XLS/XLP

- MIPS64 multicore superscalar SoCs for “big” embedded systems.
- 2-8 cores, 4 threads per core; up to 32 threads (vCPU) per chip
- 40-bit physical address space, full 62-bit TLB mapped address spaces.
- Multiple PCI-X or PCIe buses
- Security, Compression, and Networking Accelerators

## NetBSD/mips 5.0 Status

- Uses O32 ABI.
- Kernel is 32-bit only.
- Can only use memory which is addressable via KSEG0. I/O registers may be located beyond the first 512MB of physical memory.
- Uniprocessor only.

## What Needed To Be Done?

- 64-bit address space.
- Be able to use lots (16+GB) of memory.
- Generic MP support
- Support new optional kernel features (preemption, fast soft interrupts)

# NetBSD/mips64 is NetBSD/mips

- Unlike the sparc64 or powerp64 approaches, I decided that mips64 support will be integrated with the 32-bit mips support; thus there is no `src/sys/arch/mips64.`
- Except for a few minor cases of coping with ABI differences, the same code is used for 64-bit mips and 32-bit mips as well as for O32, N32, and N64. Uses CPP macros to abstracted 32-bit and 64-bit instructions: PTR_L, LONG_L, REG_L, etc.
- Added builtin CPP macros for __mips_{o,n}{32,64} to make ABI testing simpler.

# N32: The NetBSD/mips64 ABI

- Most programs don’t need a 64-bit address space.
- The XLR/XLS don’t have a FPU. Emulating the new FP instructions semantics needed by N32 or N64 is a lot of work; use software emulation instead.
- N32 kernel grovelers don’t like running with a N64 kernel.
- N32 userland can run under either N32 or N64 compiled kernels.

## COMPAT_NETBSD32

### Running 32-bit programs on a 64-bit kernel.

- System needs to be able to able boot while using a completely 32-bit userland.
- Added compat code to deal with mount(2) and the routing socket.
- New interfaces need to be 64-bit clean. Replace old unclean interfaces with new 64-bit clean interface.
- Update routing socket protocol to use a 64-bit clean interface.

# 64-Bit Address Space Support

- Requires three levels of page tables:
  - 4KB page size supports a 40-bit (1TB) address space
  - 8KB page size supports a 44-bit (16TB) address space
- Optimize for 32-bit address space which only requires a two level lookup. Even when using 4KB pages, 2 level of page tables maps 2GB which is the size of the 32-bit user address space.
- XKPHYS is used to access memory without needed a TLB entry. Used for I/O address and dynamic kernel memory.

## 64-Bit Address Space Support Continued

Since many devices only support DMA operations to addresses in the first 4GB of physical address space, the system needs to track those pages at or above 4GB separately from those below that threshold. If 32-bit DMA is requested for a page that does not reside with the first 4GB, a bounce buffer (allocated from the first 4GB) is used to perform the DMA

# 64-bit Kernels

- mips64 kernels can be built using N32 or N64 ABIs but run with a 64-bit kernel address space regardless of ABI.
- Kernels load into KSEG0 and use 32-bit symbols (-msym32) so that only 2 instructions are needed to load symbol's value (instead of 5).
- N64 kernels are converted to N32 ELF format (possible since symbols are 32-bits) to allow bootloaders to load them without knowing ELF64.
- N32 kernels use KSEG2 (1GB) for TLB mapped addresses. N64 kernels use XKSEG (4EB) for TLB mapped addresses.

## Abstractions

sys/arch/mips contains the cpu-dependent code which is the machine-dependent abstraction. However there are many variants of MIPS cpus and each needs to look the same to the generic MIPS code.

- mips_locore_jumpvec_t - chip/isa dependent routines (tlb, etc.)
- mips_locore_atomicvec_t (LL/SC or RAS based routines)
- struct locoresw (idle, smp routines)
- struct splsw (interrupt and spl routines)

# Dynamic Fixups

Abstractions are nice but calling via indirect calls has a cost. It would be great to have the speed of direct calls. So the mips kernel will cause calls to "stub" routines to be automatically changed to direct calls to the routine the stub would have called indirectly.

The stubs are place in a special section at the end of .text segment. The kernel text is scanned looking for calls to addresses within that section. When one is found, the fixup code will "emulate" the instructions in the stub to determine what routine the stub would have called and then fixes the original instruction to call that routine directly.

## Dynamic Fixups continued

- A normal kernel typically has over 5,000 calls modified which is accomplished in about 3ms.
- This has worked so well that the PowerPC ports now use this technique.
- I plan on adding that capability to the ARM ports as well.
- Module symbol resolution to stub routines need to be fixed up to use real routines.
- Easiest to implement on architectures that have fixed sized instructions.

# SMP Changes

- pmap module is mostly lockless by using atomic sequences.
- LL/SC atomic routines only used on machines capable of running with multiple CPUs.
- Add a hook for sending IPIs (hardware dependent)
- Add a hook for spinup secondary processors
- cpu_want_resched make MP aware

# Changing the Page Size

- A MIPS kernel can use any page size it wants (specified by a configuration option in the kernel configuration file). The Lonsoong kernels use a 16KB page size due to virtual cache aliasing.
- 8KB or 32KB (“odd” page shift) require less complexity since there is always only one PTE per dual TLB entry. Page sizes with an even page size uses two independent PTEs per dual TLB entry and that requires code to deal with invalidating one of the two mappings in that TLB entries.

# Exception Code Flavors

| Flavor | Used by |
|---|---|
| MIPS1 | R2000, R3000 |
| MIPS3 | R4000, R4400, R5000, etc. |
| MIPS32 | MIPS 4K |
| MIPS64 | BCM1250, MIPS 5K |
| MIPS32R2 | MIPS 24K,74K |
| MIPS64_RMIXL | XLR/XLS |
| MIPS64R2 | XLP |
| MIPS64R2_RMIXL | XLP |

## Embedded MIPS

- MIPS32-based SoCs with 32MB to 256MB are used for home routers and access points. XLR/XLS/XLP based systems with 512MB to 16GB are used for networking or storage servers.
- The MIPS32-based SoCs typically use the MIPS 24K or 74K CPU cores which require dealing with instruction hazards differently. These cores also have the DSP Application-Specific Extension which is treated in a similar manner as a FPU.
- No storage except for a small amount of NOR or NAND flash.

# Using Memory Outside of KSEG0

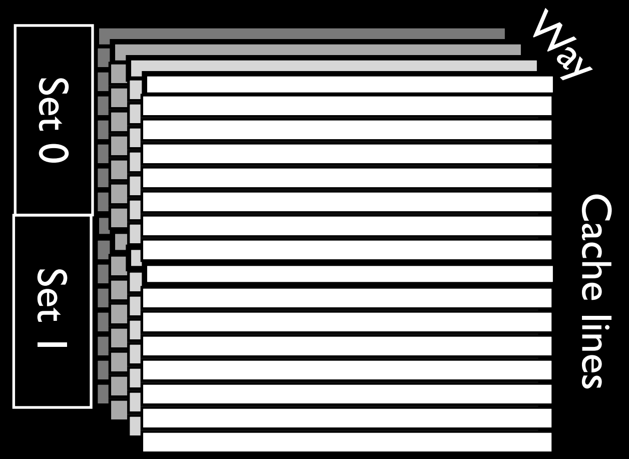
- On N64 kernels, this memory is accessed via the 62-bit XKPHYS direct mapped segment.
- O32 kernels do not support accessing memory outside of KSEG0 since the systems which use those kernels don’t have memory outside of KSEG0.
- N32 supports a “KSEGX” region which uses fixed TLB entries to map extra RAM to extend the amount of direct-mapped space available for kernel allocations. Since N32 can use more memory than it can actually map, avoiding TLB mapped memory improves performance.

## Users of KSEG0.

- First page contain exception vectors
- Kernel resides near the start of KSEG0.
- Pages in KSEG0 (and KSEGX) are held in reserve and are used for kernel memory allocations via the pool allocator. This requires the system to track these pages separately from those pages not in KSEG0 and KSEGX.

# Virtual Cache Aliasing

32KB 4-way
512B/line



- Newer MIPS implementations have cache implementations that have a “way” size greater the page size.
- Virtually indexes may cause a cache line to appear in multiple cache sets which can lead to memory corruption.
- The number of sets (way-size / page-size) is also referred as the number of page colors.

# Preventing Virtual Cache Aliasing

If it can be guaranteed that bits that select the set/color are always the same in both the virtual address and the physical address then aliasing can not happen.

Simplest way to achieve this is to make sure the page size is greater or equal to the way size (This is why the Lonsoong kernels use a 16KB page size). But if there is a large way-size and small amount of memory this can be prohibitively expensive.

Otherwise when selecting a page for a virtual address, always select a congruent physical page. When mapping a physical page, always select a congruent virtual address.

## Managing Page Colors

To achieve the selection of congruent pages, each color must have its own free list. Fortunately UVM in NetBSD supports this. But that's not enough, in addition, each freelist also needs to separately managed and that UVM does not do. If the system is short of pages of a particular color, it needs to reclaim pages of that specific color, not just any color.

Adding to this mess, the MIPS dependent code also needs to track 3 types of pages: those in KSEG0/KSEGX, all other pages in the first 4GB, and finally the rest of the pages (at or above 4GB).

# UVM Page Groups

So managing pages by color is not enough, they need to managed by type of free page as well as by color. Each collection of pages is treated as a page group. Each page group has its own set of active pages, inactive pages, free target, free count, etc. What UVM used to track globally for all pages is now done on a per page group basis.

For a MIPS64 system with 4GB, a 32KB way size, and a 4KB page size, this will result in 24 (3 types by 8 colors) page groups.

Since pages are reclaimed by page group, writing these to swap is more difficult since you can no longer map these page as virtually contiguous since that would violate the congruent page mapping rule. So for now the I/O is done page by page.

# UVM Color Matching

Whenever a page is allocated for a virtual address, UVM will pick a physical page that has the same color. If UVM needs to map a physical page, it will select a virtual address that has the same color. This required changing UVM to pass color hints almost all the time.

This feature will also help other architectures (like ARM which also has VIPT caches which prefer hard page coloring has well).

# Direct Mapped UAREAS

- Contains the Process Control Block (PCB) and kernel stack for a LWP.
- If located in TLB mapped memory, must be locked into the TLB at context switch.
- Uses valuable TLB mapped address space.
- First make UAREAS non-swappable (not much space to reclaimed and removing support simplifies the kernel).
- Add hooks allowing a port to allocate its own UAREA. First used by mips, but also used by powerpc, alpha, and amd64.

## Common pmap for TLB-based MMUs

The MIPS pmap was converted to a common pmap implementation to be used by other architectures which use a software updated TLB based MMU.  First non-MIPS port to use this pmap was the powerpc booke port.

MIPS still needs to convert to using the new common pmap...

# PCU - Per CPU Unit

- Almost every port has its own implementation of lazy FP switching using varying logic on how to do it.
- PCU merges this logic in a common machine independent framework integrated with the scheduler.
- Normally used for FPU, but can be also be used other units such as MIPS DSP or PowerPC AltiVec or SPE.

```
void pcu_state_save(lwp_t *lwp)
      save the current CPU's state
      into the given LWP's MD
      storage.

void pcu_state_load(lwp_t *lwp, bool
used)
      load PCU state from the given
      LWP's MD storage to the
      current CPU.  the 'used'
      argument is true if it isn't
      the first time the LWP uses
      the PCU.

void pcu_state_release(lwp_t *lwp)
      tell MD code detect the next
      use of the PCU on the LWP,
      and call pcu_load().
```

## Q&A?

matt@netbsd.org