

IPF paravirt_ops work effort estimation



Agenda

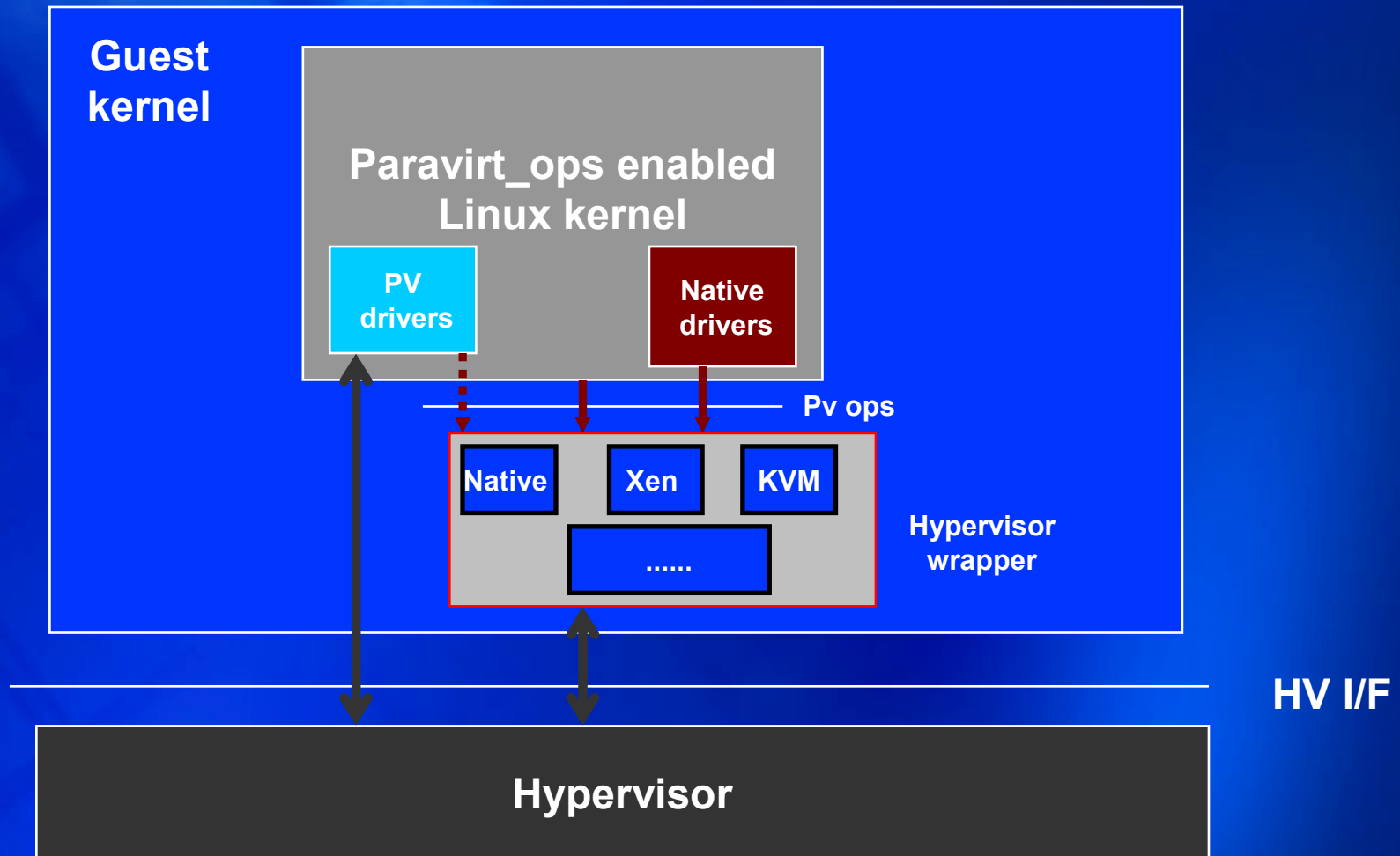
- **Overview**
- **X86 pv_ops background**
- **Challenge for IA64**
- **Cooperation & work items**

Goal

- **Call for community to go with pv_ops**
 - A little bit details for cpu side pv_ops for easy kick off

Here pv_ops has dedicated meaning i.e. Rusty Russell proposed paravirt_ops solution, which appears in X86/32 bits side

Paravirt_ops framework



I/Fs

- **Pv_ops (paravirt_ops)**
 - Primitive operation to privilege resources for para-virtualization
 - A function table initialized by hypervisor wrapper
 - Native is abstracted too
 - PV drivers may talk to HV directly
- **HV I/F**
 - Hypercall / Event channel / Share memory etc.
 - Provided by hypervisor
- **Hypervisor wrapper**
 - Hook for each hypervisor (include native)

Agenda

- Overview
- **X86 pv_ops background**
- Challenge for IA64
- Cooperation & work items

X86 pv_ops background

- **struct paravirt_patch_template{**
 - struct pv_init_ops pv_init_ops; IA64 too
 - /* dynamic patching & some initial setup */ Covered by CPU
 - struct pv_time_ops pv_time_ops;
 - struct pv_cpu_ops pv_cpu_ops;
 - /* sensitive instruction replacement, such as clts, cr0/cr4 r/w, tr/gdt/ldt/idt ops, cpuid, msr, wbinvd, pmc, tsc, iret */ Key for IA64
 - struct pv_irq_ops pv_irq_ops; Likely be in CPU
 - /* eflag save/resore, cli/sti, halt & init */
 - struct pv_apic_ops pv_apic_ops;
 - /* APIC register r/w, calibrate, startup_ipi_hook (VMI only) */
 - struct pv_mmu_ops pv_mmu_ops;
- **};** Xen irqchip used instead of virtual apic

Covered by CPU



X86 ASM code replacement

```

- #define PARA_PATCH(struct, off) ((PARAVIRT_PATCH_##struct + (off)) / 4)
- #define PARA_SITE(ptype, clobbers, ops) \
    - 771;; \
    - ops; \
    - 772;; \
    - .pushsection .parainstructions,"a"; \
    - .long 771b; \
    - .byte ptype; \
    - .byte 772b-771b; \
    - .short clobbers; \
    - .popsection \

- #define DISABLE_INTERRUPTS(clobbers) \
    - PARA_SITE(PARA_PATCH(pv_irq_ops, PV_IRQ_irq_disable), clobbers, \
    - pushl %eax; pushl %ecx; pushl %edx; \
    - call *%cs:pv_irq_ops+PV_IRQ_irq_disable; \
    - popl %edx; popl %ecx; popl %eax)

```

Original "cli"
instruction is replaced
with this MACRO

X86 C code replacement

```
– static inline void raw_local_irq_disable(void)
– {
–     asm volatile(paravirt_alt("pushl %%ecx; pushl %%edx;“
–                               PARAVIRT_CALL
–                               "popl %%edx; popl %%ecx")
–               :
–               : paravirt_type(pv_irq_ops.irq_disable),
–               paravirt_clobber(CLOBR_EAX)
–               : "memory", "eax", "cc");}
```

Original “cli”
instruction is replaced
with call to this
function

This one will call
pv_irq_ops.irq_disable,
but some special
techniques are used for
better patching later on

X86 pv_ops patching - 1

- **.parainstructions**
 - A special sections used to store all patchable code info
 - struct paravirt_patch_site {
 - u8 *instr; /* original instructions */
 - u8 instrtype; /* type of this instruction */
 - u8 len; /* length of original instruction */
 - u16 clobbers; /* what registers you may clobber */
 - };
- **C code pv_ops uses _paravirt_alt(ins, type, clobber) to generate some code, and mark it as patchable.**
 - Almost all pv_ops
- **ASM code uses PARA_SITE(ptype, clobbers, ops)**
 - Mainly for kernel/entry_32.S, covers
 - Cli, sti, iret & “sti; sysexit”

X86 pv_ops patching - 2

- **Patching in place (paravirt_patch_insns)**
 - For hot code (ASM)
 - Directly copy code into original code space
 - Benefit:
 - Avoid function call
- **Patching with direct call (paravirt_patch_default)**
 - For iret & "sti; sysexit", use jmp (op code: 0xe9)
 - Use IP relative call (op code: 0xe8) to convert indirect call to direct call
 - Benefit:
 - Avoid fetching function address, helps in CPU predication
- **Xen patch (xen_patch)**
 - In place: Cli / sti, save/restore eflags
 - Others: direct call/jmp
- **Lguest (lguest_patch)**
 - Same with Xen

Agenda

- Overview
- X86 pv_ops background
- **Challenge for IA64**
 - Components
 - IVT table
 - Calling convention
- Cooperation & work items

IA64 pv_ops

- **Replace sensitive instruction with pv_os**
 - Non-privilege sensitive instruction
 - Thash/Ttag/fc
 - Fc behavior differently when ring!=0
 - Cover when vPSR.ic != PSR.ic
 - Performance critical privilege instruction
- **CPU pv_ops**
 - Cover timer, mmu, irq
- **Platform pv_ops**
 - Pv_init_ops
 - Dma: Depend on X86
 - Irqchip: Similar with X86 xen irqchip
 - PMU/xenoprof etc.
- **ASM code issues**
 - IVT table & system call gate page
 - Other ASM code (such as entry.S, fsys.S)
 - No stack register access too for some of them such as ia64_leave_syscall/ia64_switch_to/ia64_leave_kernel

CPU pv_ops

- **Different table for C & ASM code**
 - X86 like function table for C
 - Pv_cpu_ops
 - Follow C convention
 - Jump only function table for ASM code
 - Pv_cpu_asm_ops
 - Return thru jump too
 - Follow static register convention
- **Challenge for pv_cpu_asm_ops**
 - IVT code doesn't have memory access ready --> need treasure fixed clobber registers

Proposal: Remove running_on_xen condition check thoroughly which conflict with pv_ops concept

Agenda

- Overview
- X86 pv_ops background
- **Challenge for IA64**
 - Components
 - **IVT table**
 - Calling convention
- Cooperation & work items

IVT table alternatives

- **Dual source**
 - Pros: Flexible for each hypervisor and native
 - Cons: Ugly and maintenance issue
- **Single source**
 - Replace sensitive instruction with pv MACRO.
 - Example code:

<ul style="list-style-type: none"> – @@ -102,7 +116,7 @@ * unimplemented address bits valid page table mapping – - mov r16=cr.ifa – + MOV_FROM_CR_IFA(r16, r24, r25) – #ifdef CONFIG_HUGETLB_PAGE – movl r18=PAGE_SHIFT 	<ul style="list-style-type: none"> - the faulting virtual address uses * - the faulting virtual address has no */ // get address that caused the TLB miss mov r25=cr.itir
--	--

IVT table alternatives - Single source

- **Dual compile, Dual IVT instance**
 - Generate code in place
 - Can use float clobber registers
 - Directly jump
 - Need fixed clobber registers
 - Indirectly jump
 - Need fixed clobber registers
- **Single instance**
 - Same with above indirectly jump

Proposal: Go with single source/dual compile/generate code in place approach to reduce impact to native IVT code

Example of generating code in place

- **Generate pv MACRO code in place**

- `#ifdef CONFIG_XEN`
- `#define MOV_FROM_CR_IFA(reg, clob1, clob2) \`
- `+ movl clob1=XSI_IFA;; \`
- `+ ld8 reg=[clob1];;`
- `#endif`

- `#ifdef CONFIG_NATIVE`
- `#define MOV_FROM_CR_IFA(reg, clob1, clob2) \`
- `+ mov reg=cr.ifa;`
- `#endif`

- **Misc**

- Recycle the redundant space (__cpuinitdata ?)
- System call gate page uses same mechanism.

Other ASM code

- **Head.S**
 - Add a new start entry point (for Xen)
- **Fsys.S, entry.S**
 - Dual compile?
 - Single instance with indirect function call?

Agenda

- Overview
- X86 pv_ops background
- **Challenge for IA64**
 - Components
 - IVT table
 - **Calling convention**
- Cooperation & work items

Calling convention

- **Static registers (Clobber registers)**
 - In: R8/R9, Out: R8 --- Current Xen
 - Need extra save/restore for R8/R9
 - In: R24/R25, Out: R24 – PAL like convention
 - Need HV I/F changes, could be future work
- **Stacked registers**
 - Follow C convention

Proposal: Hide the issue in hypervisor wrapper for now,
switch to PAL like convention in future



Agenda

- Overview
- X86 pv_ops background
- Challenge for IA64
- Cooperation & work items

How to cooperate?

- **How to avoid duplicate effort**
 - Partition the work ?
- **Repository for pv_ops**
 - Gitorious.org ?
- **Push patch out?**
 - DomU patch only for now?
 - Complete basic pv_ops feature first?
- **X86 dom0 pv_ops support status?**
 - dma

Current Status

Items	Status	Comments
Forward port	Done	
pv_info	Done	
pv_init_ops	Done	
pv_cpu_asm_ops	mostly Done	__cpuinitdata?
pv_cpu_ops	WIP	Indirect C function call
pv_iosapic_ops	Done	
pv_irq_ops	Done	
pv_time_ops	WIP	To share code with x86 if possible
Revice patches		To reduce modifications.



All patches need review by others²⁴

work items – phase 1

- **Basic cpu pv_ops**
 - Frame work
 - C code pv_cpu_ops
 - ASM code
 - Dual compile IVT.s: Mostly done by Isaku's patch
 - Dual compile Gate.s
 - Other ASM code changes
 - Recycle memories used for ivt & gate page
 - PAL/SAL
 - Virtual timer (need abstract?)
 - Port/reuse X86 xen Irqchip in upstream
 - Pv_ops for IOSAPIC



work items – phase 1: Cont.

- **Native wrapper for pv_ops**
 - Binary patching
- **Xen wrapper for pv ops**
 - Remove running_on_xen
- **Xen domain builder & startup code**

Pv_ops work – phase 2

- **Dma, i.e. p2m or swiotlb issue**
- **Kexec/Kdump: could be later**
- **PMU/PMC/PMD**
- **static registers calling convention**
 - **hypervisor modification to support better**
 - **Backward support need to be maintained**
- **mca**

IPF paravirt_ops work effort estimation



Agenda

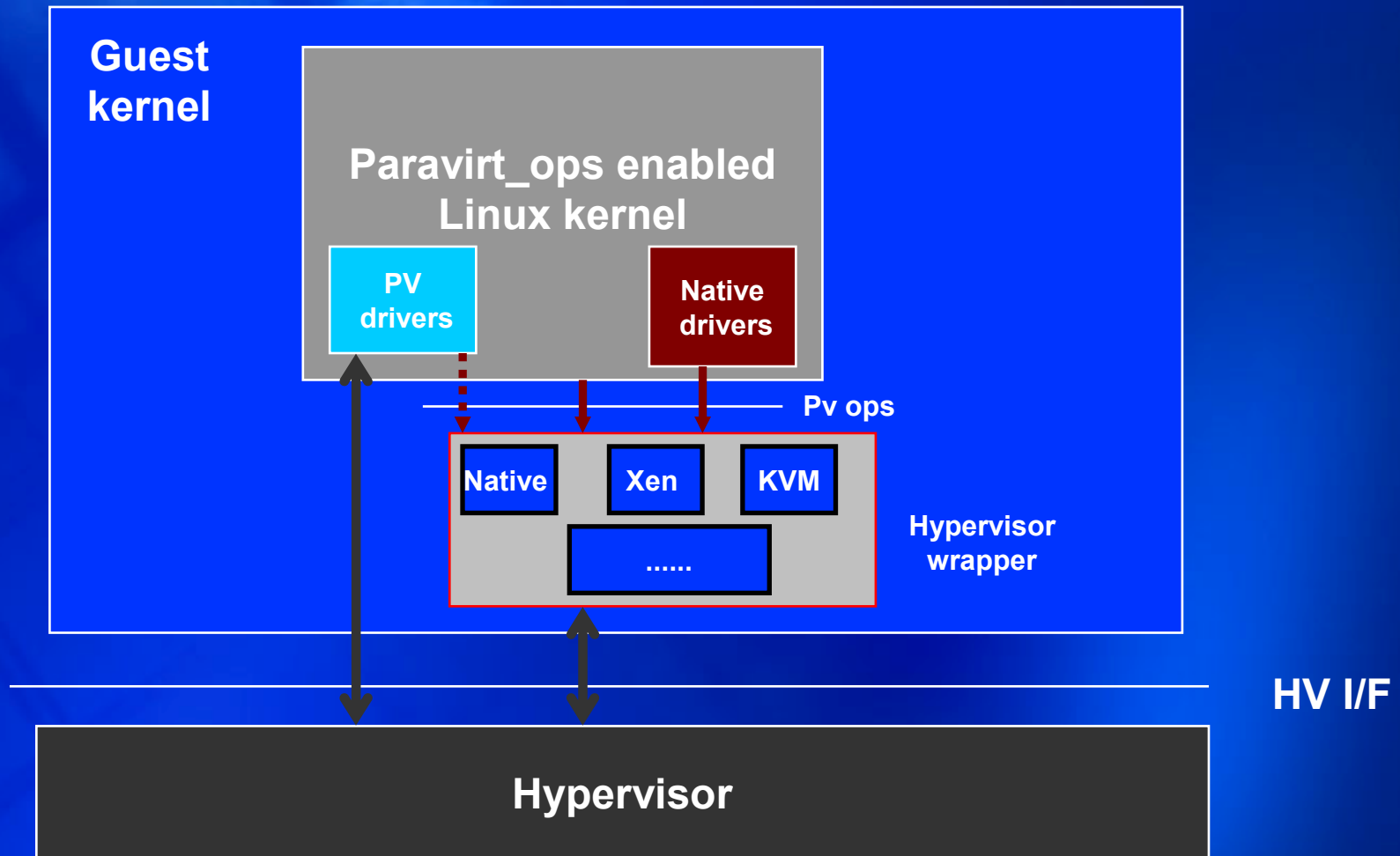
- **Overview**
- **X86 pv_ops background**
- **Challenge for IA64**
- **Cooperation & work items**

Goal

- **Call for community to go with pv_ops**
 - A little bit details for cpu side pv_ops for easy kick off

Here pv_ops has dedicated meaning i.e. Rusty Russell proposed paravirt_ops solution, which appears in X86/32 bits side

Paravirt_ops framework



I/Fs

- **Pv_ops (paravirt_ops)**
 - Primitive operation to privilege resources for para-virtualization
 - A function table initialized by hypervisor wrapper
 - Native is abstracted too
 - PV drivers may talk to HV directly
- **HV I/F**
 - Hypercall / Event channel / Share memory etc.
 - Provided by hypervisor
- **Hypervisor wrapper**
 - Hook for each hypervisor (include native)

Agenda

- Overview
- **X86 pv_ops background**
- Challenge for IA64
- Cooperation & work items

X86 pv_ops background

- **struct paravirt_patch_template{**
 - struct pv_init_ops pv_init_ops; IA64 too
 - /* dynamic patching & some initial setup */ Covered by CPU
 - struct pv_time_ops pv_time_ops;
 - struct pv_cpu_ops pv_cpu_ops;
 - /* sensitive instruction replacement, such as clts, cr0/cr4 r/w, tr/gdt/ldt/idt ops, cpuid, msr, wbinvd, pmc, tsc, iret */ Key for IA64
 - struct pv_irq_ops pv_irq_ops; Likely be in CPU
 - /* eflag save/resore, cli/sti, halt & init */
 - struct pv_apic_ops pv_apic_ops;
 - /* APIC register r/w, calibrate, startup_ipi_hook (VMI only) */
 - struct pv_mmu_ops pv_mmu_ops;
- **};** Xen irqchip used instead of virtual apic

Covered by CPU



X86 ASM code replacement

```

- #define PARA_PATCH(struct, off) ((PARAVIRT_PATCH_##struct + (off)) / 4)
- #define PARA_SITE(ptype, clobbers, ops) \
    - 771;; \
    - ops; \
    - 772;; \
    - .pushsection .parainstructions,"a"; \
    - .long 771b; \
    - .byte ptype; \
    - .byte 772b-771b; \
    - .short clobbers; \
    - .popsection \

- #define DISABLE_INTERRUPTS(clobbers) \
    - PARA_SITE(PARA_PATCH(pv_irq_ops, PV_IRQ_irq_disable), clobbers, \
    - pushl %eax; pushl %ecx; pushl %edx; \
    - call *%cs:pv_irq_ops+PV_IRQ_irq_disable; \
    - popl %edx; popl %ecx; popl %eax)

```

Original "cli"
instruction is replaced
with this MACRO

X86 C code replacement

```
- static inline void raw_local_irq_disable(void)
- {
-     asm volatile(paravirt_alt("pushl %%ecx; pushl %%edx;“
-                               PARAVIRT_CALL
-                               "popl %%edx; popl %%ecx")
-                 :
-                 : paravirt_type(pv_irq_ops.irq_disable),
-                   paravirt_clobber(CLOBR_EAX)
-                 : "memory", "eax", "cc");}
```

Original “cli”
instruction is replaced
with call to this
function

This one will call
pv_irq_ops.irq_disable,
but some special
techniques are used for
better patching later on

X86 pv_ops patching - 1

- **.parainstructions**
 - A special sections used to store all patchable code info
 - struct paravirt_patch_site {
 - u8 *instr; /* original instructions */
 - u8 instrtype; /* type of this instruction */
 - u8 len; /* length of original instruction */
 - u16 clobbers; /* what registers you may clobber */
 - };
- **C code pv_ops uses _paravirt_alt(ins, type, clobber) to generate some code, and mark it as patchable.**
 - Almost all pv_ops
- **ASM code uses PARA_SITE(ptype, clobbers, ops)**
 - Mainly for kernel/entry_32.S, covers
 - Cli, sti, iret & “sti; sysexit”

X86 pv_ops patching - 2

- **Patching in place (paravirt_patch_insns)**
 - For hot code (ASM)
 - Directly copy code into original code space
 - Benefit:
 - Avoid function call
- **Patching with direct call (paravirt_patch_default)**
 - For iret & "sti; sysexit", use jmp (op code: 0xe9)
 - Use IP relative call (op code: 0xe8) to convert indirect call to direct call
 - Benefit:
 - Avoid fetching function address, helps in CPU predication
- **Xen patch (xen_patch)**
 - In place: Cli / sti, save/restore eflags
 - Others: direct call/jmp
- **Lguest (lguest_patch)**
 - Same with Xen

Agenda

- Overview
- X86 pv_ops background
- **Challenge for IA64**
 - Components
 - IVT table
 - Calling convention
- Cooperation & work items

IA64 pv_ops

- **Replace sensitive instruction with pv_os**
 - Non-privilege sensitive instruction
 - Thash/Ttag/fc
 - Fc behavior differently when ring!=0
 - Cover when vPSR.ic != PSR.ic
 - Performance critical privilege instruction
- **CPU pv_ops**
 - Cover timer, mmu, irq
- **Platform pv_ops**
 - Pv_init_ops
 - Dma: Depend on X86
 - Irqchip: Similar with X86 xen irqchip
 - PMU/xenoprof etc.
- **ASM code issues**
 - IVT table & system call gate page
 - Other ASM code (such as entry.S, fsys.S)
 - No stack register access too for some of them such as ia64_leave_syscall/ia64_switch_to/ia64_leave_kernel

CPU pv_ops

- **Different table for C & ASM code**
 - X86 like function table for C
 - Pv_cpu_ops
 - Follow C convention
 - Jump only function table for ASM code
 - Pv_cpu_asm_ops
 - Return thru jump too
 - Follow static register convention
- **Challenge for pv_cpu_asm_ops**
 - IVT code doesn't have memory access ready --> need treasure fixed clobber registers

Proposal: Remove running_on_xen condition check thoroughly which conflict with pv_ops concept

Agenda

- Overview
- X86 pv_ops background
- **Challenge for IA64**
 - Components
 - **IVT table**
 - Calling convention
- Cooperation & work items

IVT table alternatives

- **Dual source**
 - Pros: Flexible for each hypervisor and native
 - Cons: Ugly and maintenance issue
- **Single source**
 - Replace sensitive instruction with pv MACRO.
 - Example code:

<ul style="list-style-type: none"> – @@ -102,7 +116,7 @@ * unimplemented address bits valid page table mapping – - mov r16=cr.ifa – + MOV_FROM_CR_IFA(r16, r24, r25) – #ifdef CONFIG_HUGETLB_PAGE – movl r18=PAGE_SHIFT 	<ul style="list-style-type: none"> - the faulting virtual address uses * - the faulting virtual address has no */ // get address that caused the TLB miss mov r25=cr.itir
--	--

IVT table alternatives - Single source

- **Dual compile, Dual IVT instance**
 - Generate code in place
 - Can use float clobber registers
 - Directly jump
 - Need fixed clobber registers
 - Indirectly jump
 - Need fixed clobber registers
- **Single instance**
 - Same with above indirectly jump

Proposal: Go with single source/dual compile/generate code in place approach to reduce impact to native IVT code

Example of generating code in place

- **Generate pv MACRO code in place**

- `+#ifdef CONFIG_XEN`
- `+#define MOV_FROM_CR_IFA(reg, clob1, clob2)\`
- `+ movl clob1=XSI_IFA;;`
- `+ ld8 reg=[clob1];;`
- `+#endif`

- `+#ifdef CONFIG_NATIVE`
- `+#define MOV_FROM_CR_IFA(reg, clob1, clob2)\`
- `+ mov reg=cr.ifa;`
- `+#endif`

- **Misc**

- Recycle the redundant space (__cpuinitdata ?)
- System call gate page uses same mechanism.

Other ASM code

- **Head.S**
 - Add a new start entry point (for Xen)
- **Fsys.S, entry.S**
 - Dual compile?
 - Single instance with indirect function call?

Agenda

- Overview
- X86 pv_ops background
- **Challenge for IA64**
 - Components
 - IVT table
 - **Calling convention**
- Cooperation & work items

Calling convention

- **Static registers (Clobber registers)**
 - In: R8/R9, Out: R8 --- Current Xen
 - Need extra save/restore for R8/R9
 - In: R24/R25, Out: R24 – PAL like convention
 - Need HV I/F changes, could be future work
- **Stacked registers**
 - Follow C convention

Proposal: Hide the issue in hypervisor wrapper for now,
switch to PAL like convention in future



Agenda

- Overview
- X86 pv_ops background
- Challenge for IA64
- Cooperation & work items

How to cooperate?

- **How to avoid duplicate effort**
 - Partition the work ?
- **Repository for pv_ops**
 - Gitorious.org ?
- **Push patch out?**
 - DomU patch only for now?
 - Complete basic pv_ops feature first?
- **X86 dom0 pv_ops support status?**
 - dma

Current Status

Items	Status	Comments
Forward port	Done	
pv_info	Done	
pv_init_ops	Done	
pv_cpu_asm_ops	mostly Done	__cpuinitdata?
pv_cpu_ops	WIP	Indirect C function call
pv_iosapic_ops	Done	
pv_irq_ops	Done	
pv_time_ops	WIP	To share code with x86 if possible
Revice patches		To reduce modifications.



All patches need review by others²⁴

work items – phase 1

- **Basic cpu pv_ops**
 - Frame work
 - C code pv_cpu_ops
 - ASM code
 - Dual compile IVT.s
 - Dual compile Gate.s
 - Other ASM code changes
 - Recycle memories used for ivt & gate page
 - PAL/SAL
 - Virtual timer (need abstract?)
 - Port/reuse X86 xen Irqchip in upstream
 - Pv_ops for IOSAPIC



work items – phase 1: Cont.

- **Native wrapper for pv_ops**
 - Binary patching
- **Xen wrapper for pv ops**
 - Remove running_on_xen
- **Xen domain builder & startup code**

Pv_ops work – phase 2

- **Dma, i.e. p2m or swiotlb issue**
- **Kexec/Kdump: could be later**
- **PMU/PMC/PMD**
- **static registers calling convention**
 - **hypervisor modification to support better**
 - **Backward support need to be maintained**
- **mca**