paravirt_ops/IA64

Isaku Yamahata <yamahata@valinux.co.jp>
VA Linux Systems Japan K.K.

Introduction

Goal

- Merge xenLinux/IA64 modification to Linux upstream. Both domU/dom0 support.

Old History

- There have already been several unsuccessful merge efforts
- So we are trying again...

30 Jan, 2005	Xen and the Art of Linux/IA64 Virtualization	Dan Magenheimer	2.6.12
28 Oct, 2005	virtualization hooks patch for IA64 2.6.15	Dan Magenheimer	2.6.15
03 Jun, 2006	Xen/IA64 kernel changes 2.6.17-rc5	Alex Williamson	2.6.17-rc5

What’s Different This Time?

- The x86 paravirt_ops (pv_ops for short) has been merged
  - After several approaches to virtualization support were proposed, finally the paravirt_ops approach won
  - Consensus on virtualization via source-code API
- (Minimal) Xen/x86 has been merged
  - The Xen common code is already there
  - Though, portability patches would be necessary

Merging Strategy

- The first merge is the most difficult
- Basically follow the x86 approach
  - Virtualization infrastructure via source code API, i.e. paravirt_ops.
  - Minimize modifications at first step
    - Make patch size and the number of patches as small as possible for code review.
  - Postpone optimization where possible
  - Dom0 support is also postponed

What is paravirt_ops

- The indirect layer for virutalization
  - Virtualization support via source-code API
  - It allows a single kernel binary to run on all supported execution environments including bare-metal hardware.
  - It is implemented as C function pointer.
  - But, it supports special optimization, binary patching.
    - Indirect calls can be transformed into a direct call or in-place execution.
  - A set of macros for assembly code
    - Hand-written assembly code also needs paravirtualization.

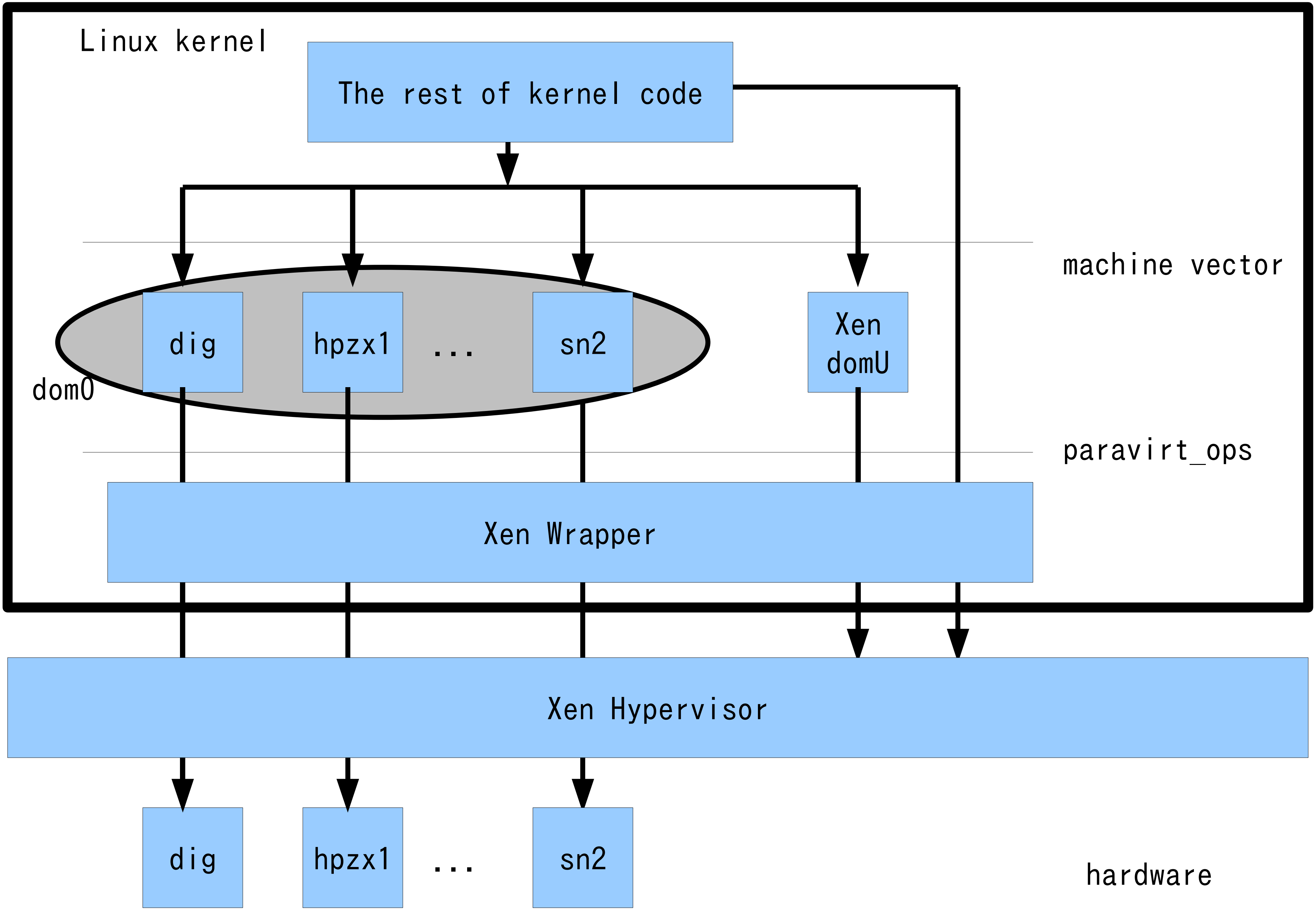
paravirt_ops/IA64

Challenges

- IA64 machine vectors
- Privileged instruction (para)virtualization
- Binary patching
- Hand-written assembly code paravirtualization

IA64 Machine Vector

- C function pointers for platform support.
  - Initialization, IPI, DMA API...
- One possible approach is to enhance machine vector.
- But paravirtualization needs more.
  - Initialization at very early stage of the boot sequence.
  - Binary patch optimization.
    - Paravirtualization overhead on bare metal hardware should be as small as possible.


Linux kernel
The rest of kernel code
machine vector
dig
hpzx1
...
sn2
Xen domU
dom0
paravirt_ops
Xen Wrapper
Xen Hypervisor
dig
hpzx1
...
sn2
hardware

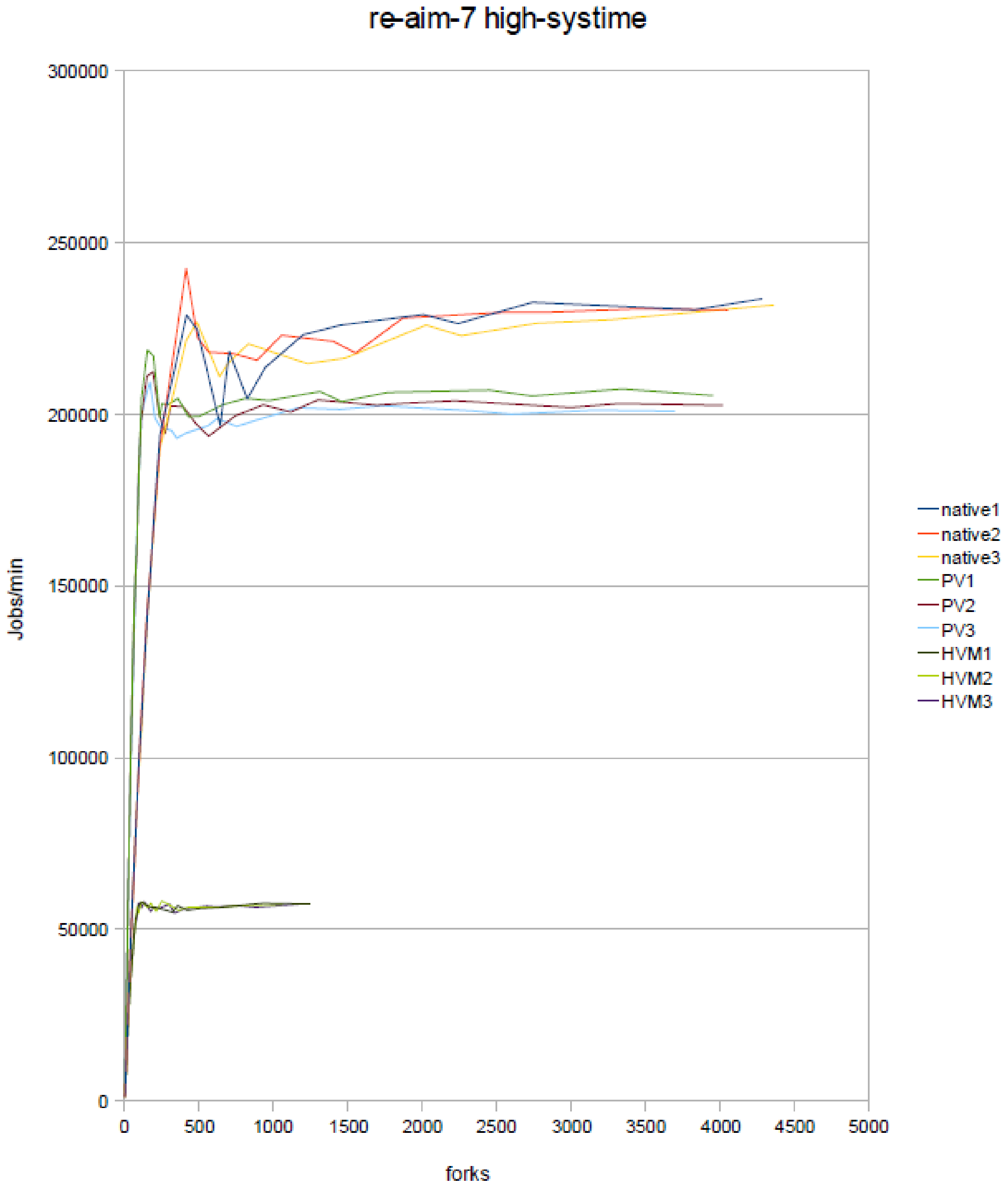
Instruction (Para)Virtualization

- The main issues for IA64 is privileged instruction paravirtualization.
- Some approaches were discussed.
  - Pre-Virtualization and after-burning
  - Hybrid Virtualization. Rely on hardware support(VT-i).
  - Privify. (vBlades approach)
  - paravirt_ops.

Instruction (Para)Virtualization(cont.)

	hyervisor change	Linux Change	Performance in theory	Single binary?
previrtualization	yes	No	Low	No
hybrid	yes	No	Low	Yes
privify	yes	No	Low	No
paravirt_ops	No	Yes	High	Yes

- With the preliminary benchmark results(the next slides), paravirt_ops approach was adopted.
- The re-aim-7 which was used is CPU intensive benchmark.



	Max JPM	% of native
Native	242347.19	100.00%
Para-virt	218694.95	90.24%
Full-virt	58289.32	24.05%

Preliminary benchmark done by Alex Williamson
http://lists.xensource.com/archives/html/
xen-ia64-devel/2008-01/msg00194.html

Binary Patch

- IA64 intrinsics(privileged instructions used by C code.) needs binary patch
    - They are performance critical.
    - e.g. mask/unmask interrupts.
- But not all the hooks needs binary patch
    - The hooks for high level functionalities accepts C indirect call overheads.
    - Unlike x86, we don’t support binary patch for all the hooks because...

Binary Patch(cont.)

- To allow binary patch, the call instruction needs to be annotated. But there is no way to write C calling conversion with GCC extended inline assembly code.
- So other calling convention has to be used.
  - Non-banked static registers(xen/ia64 like)
  - Banked Static registers.(PAL call like)
  - C function call like convention.
- Anyway those functions can’t be written in C.

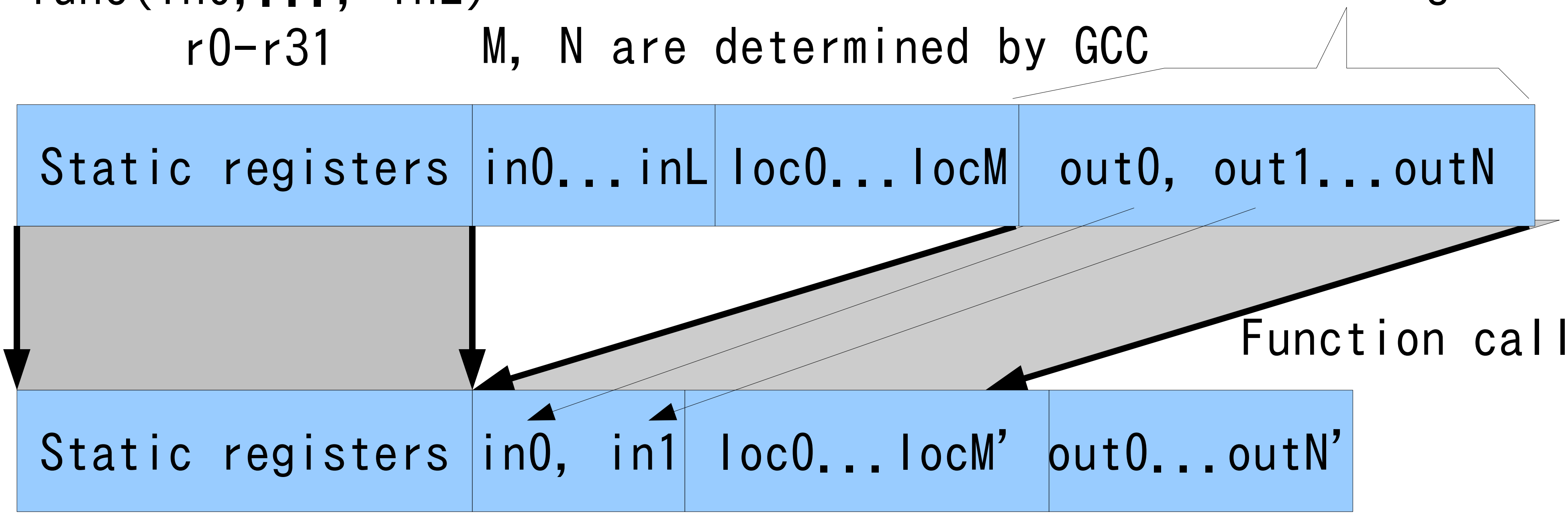
```
caller_func(in0, ... inL)
    loc0, ... locM
    out0, ... outN

    bp_func(out0, out1)
    other_func(out0, out1, ... outN)
```

asm(“... ::: “out0 , “out1) is bad.
Here out0, ... outN are clobbered.
No way to specify out0, ...outN
as clobbered registers where N is
unknown.

func(in0,..., inL)

r0-r31 M, N are determined by GCC

Clobbered registers



r0-r31 M’, N’ are determined by GCC

bp_func(in0, in1)

Hand-Written Assembly Code

- .S files
- Kernel entry(fault handler,system call)/exit and context switch.
- Stack is not always usable.
  - For example, only several registers can be used on fault.
- They are well tuned so that it’s difficult to touch them without performance degradation.

Hand Written Assembly Code(cont.)

- Single source file, compile multiple times using CPP macros and Makefile tricks.
- Most of those macros 1:1 correspond to single instruction

Example Patch

```
-(p8) mov cr.itir=r25
+     MOV_TO_ITIR(p8, r25, r24)
```

Native

```
#define MOV_TO_ITIR(pred, reg, clob)    \
(pred)  mov cr.itir = reg               \
        CLOBBER(clob)
```

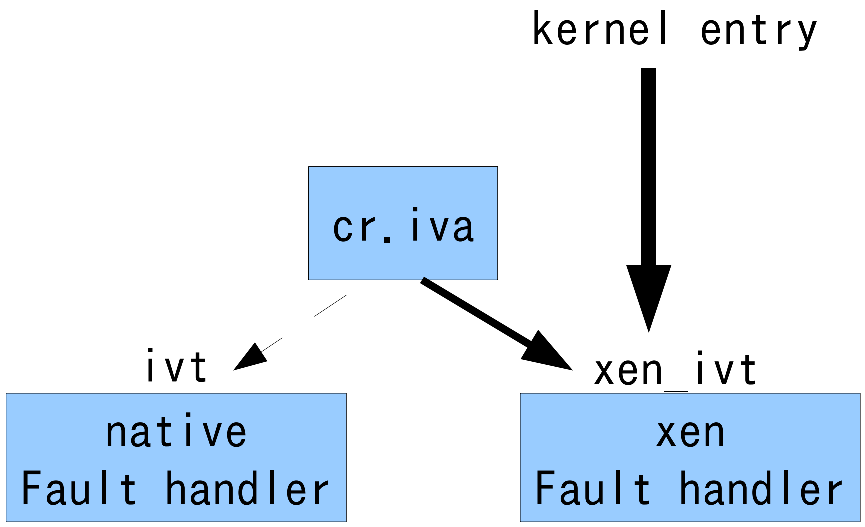
Xen

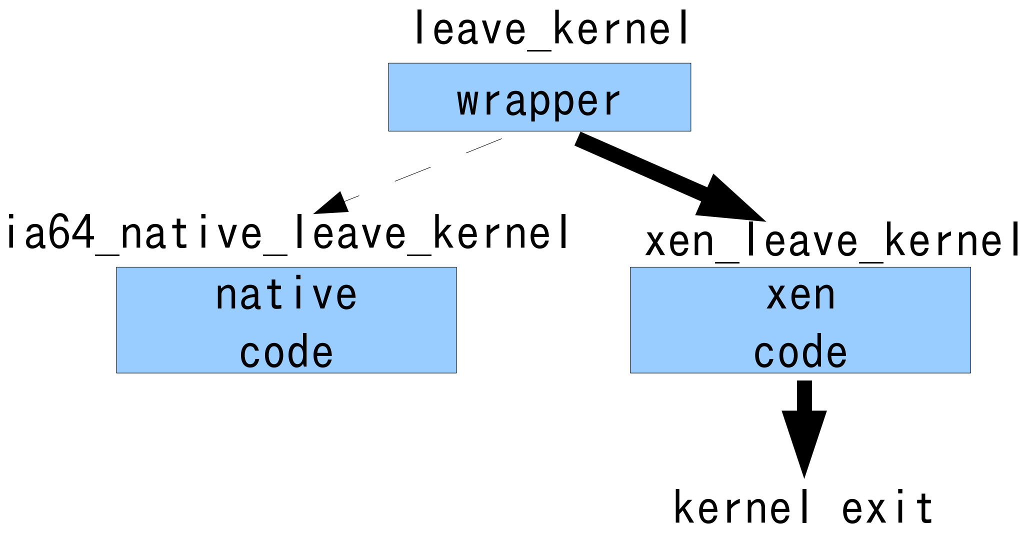
```
#define MOV_TO_ITIR(pred, reg, clob)    \
(pred)  movl clob = XSI_ITIR;           \
        ;;                              \
(pred)  st8 [clob] = reg
```

Extra clobberable registers are found by careful code reading.

Switching Hand-Written Assembly Code

- Fault handler is pointed by cr.iva register.
- Other path is switched by indirect call.
  - Clobberable registers for branch are found by careful code reading.





PV Checker

- .S paravirtualization is fragile.
  - It’s very easy to write raw instruction.
  - Enforce people to use paravirtualized instructions.
- Simple checker is implemented.
  - Doesn’t cover all the breakage.
  - most of easy breakage can be detected.

Example: cover instruction case

```
#define COVER   nop 0
#define cover   .error ”cover should not be used directly.”
```

Current Status and Future Plans

Current IA64 pv_ops

name	description	type
pv_info	general info	-
pv_init_ops	initialization	normal C function pointers
pv_cpu_ops	privileged instructions	Normal C function pointers Needs binary patch
pv_cpu_asm_ops	macros for hand-written assembly	asm macros
pv_iosapic_ops	iosapic related operations	normal C function pointers
pv_irq_ops	irq related operations	normal C function pointers
pv_time_ops	steal time accounting	normal C function pointers

- The resulting set of hooks is completely different from x86’s.
- That’s because of architecture difference between x86 and IA64.

Diffstat

	file changed	insersions	deletions
pv_cpu_asm_ops	9	517	177
pv_cpu_ops	10	550	49
others	16	507	35

- This figures shows paravirt_ops/IA64 issues are in cpu instruction paravirtualization.

Comparison with x86 pv_ops

items	X86_32(2.6.26-rc6)	IA64
pv_init_ops	5	6
pv_cpu_asm_ops	7 macros	32 macros/6 asm labels
pv_cpu_ops	32	14
pv_mmu_ops	29	n/a
others	13	18

- Again this figure shows that paravirt_ops/IA64 relies on hand-written assembly code paravirtualization.
- The number of IA64 pv_cpu_ops is smaller than x86's.
  - On IA64 privileged register operation is done by 'mov' instructions which is replaced by 2 function pointers.
  - On the other hand on x86 many function pointers are defined. e.g. load_cr3()
- Xen/IA64 fully virtualizes MMU so that pv_mmu_ops is unnecessary.

Activity

Date	subject	comment
16 Jan, 2008	Time for hybrid virtualization?	discussion
07 Feb, 2008	forward ported to linux ia64 upstream	single jumbo patch
17 Feb, 2008	paravirt_ops suport in IA64	paravirt_ops discussion
21 Feb, 2008	ia64/xen domU paravirtualization	split patches
24 Feb, 2008	ia64/xen: paravirtualization of hand written assembly code	discussion
26 Feb, 2008	RFC: ia64/xen TAKE 2: paravirtualization of hand written assembly code	
28 Feb, 2008	RFC: ia64/pv_ops: ia64 intrinsics paravirtualization	
05 Mar, 2008	ia64/xen take 3: ia64/xen domU paravirtualization	started pv_ops
09 Apr, 2008	RFC: ia64/pv_ops take 4: ia64/xen domU take 4	Fully converted into pv_ops
01 May, 2008	ia64/pv_ops take 5, ia64/xen domU take 5	
19 May, 2008	ia64/pv_ops take 6, ia64/xen domU take 6	pv_ops part was merged to linux ia64 test branch
10 Jun, 2008	ia64/xen domU take 7	preliminary for save/restore

Current status and future plan

Items		status
minimal domU	pv_ops	in linux IA64 test branch
	Xen/domU	W.I.P.(working patch, under review)
domU optimization	more .S paravirtualization	
	binary patch	Experimental patch existed
	balloon	
	save/restore	Preliminary support. Needs more paravirtualization
	free unused pages	
	revise boot protocol	
dom0	DMA API	
	kexec/kdump	
	PMU/PMD/PMC(xenoprof)	
	mca	
	and more?	

Acknowledgements

- Thanks to
  - Eddie Dong(some of slides are cited from his slides with modification)
  - Alex Williamson(The previous Xen/IA64 maintainer)
  - Akio Takebe
  - Qing He
  - Simon Horman
  - Jeremy Fitzhardinge(The Linux xen maintainer)
  - Tony Luck(The Linux IA64 maintainer)
  - Dan Magenheimer(The Xen/IA64 initiator)

Thank you

- http://wiki.xensource.com/xenwiki/XenIA64/UpstreamMerge