

Virtualisation Case Study: Itanium and vNUMA

Slide 1

Matthew Chapman
matthewc@cse.unsw.edu.au

ITANIUM

- High-performance processor architecture
- Also known as IA-64
- Joint venture between HP and Intel
- Not the same instruction set as IA-32/AMD64/EM64T
- Very good floating-point performance

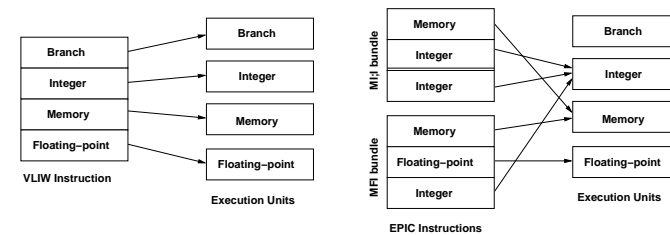
Slide 2

2000:	Itanium "Merced"	180 nm, up to 800Mhz, 4MB cache
2002:	Itanium II "McKinley"	180 nm, up to 1Ghz, 3MB cache
2003:	Itanium II "Madison"	130 nm, up to 1.6Ghz, 6MB cache
2004:	Itanium II "Madison 9M"	130 nm, up to 1.67Ghz, 9MB cache
2006:	Itanium II "Montecito"	90 nm, dual-core, up to 1.6Ghz, 24MB cache
2008:	"Tukwila"	65 nm, quad-core, up to 2.5 Ghz?

EXPLICITLY PARALLEL INSTRUCTION-SET COMPUTING (EPIC)

- Goal to increase *instructions per cycle*
 - Itanium can have similar performance to x86 at a lower clock speed
- Based on *Very Long Instruction Word (VLIW)*
- Explicit parallelism in instruction set
- Simplified instruction decode and issue
- Scheduling decisions made by compiler

Slide 3



EXAMPLE

Load and add three numbers in assembly code:

```
ld8  r4 = (r1)
ld8  r5 = (r2)
ld8  r6 = (r3)
;;
add  r7 = r4, r5
;;
add  r8 = r6, r7
;;
```

Slide 4

Slide 5

Resulting instructions:

MMI	ld8	r4 = (r1)
	ld8	r5 = (r2)
	nop.i	0
M;MI;	ld8	r6 = (r3)
	::	
	add	r7 = r4, r5 // Arithmetic on M too
	nop.i	0
	::	
M;MI	add	r8 = r6, r7
	::	
	nop.m	0
	nop.i	0

Slide 6

A better way:

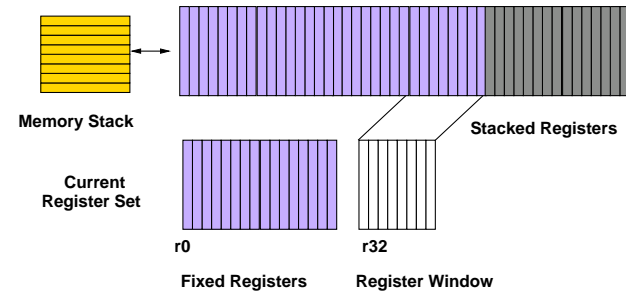
MMI;	ld8	r4 = (r1)
	ld8	r5 = (r2)
	nop.i	0
	::	
MI;l;	ld8	r6 = (r3)
	add	r7 = r4, r5
	::	
	add	r8 = r6, r7
	::	

Slide 7

ITANIUM REGISTER SET

Large general register file

→ partially managed by the *register stack engine* (RSE)

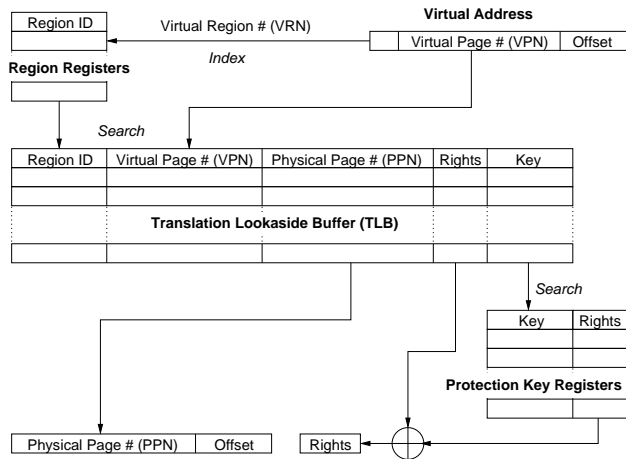


Slide 8

ITANIUM MMU

- 64-bit addressing
- Process address spaces are constructed from 8x61-bit *regions*
- Regions are 61-bit segments of a global 85-bit address space
- Page sizes from 4KB to 4GB
- Protection keys
- Choice of two hardware-walked page table formats

Slide 9



VIRTUALISING THE ITANIUM

Slide 10

- We demote the OS from privilege level 0 (most privileged) to 1
- The privileged instructions in the OS will now trap, and can be emulated with respect to a virtual machine
- Which instructions behave differently, and do not trap?
 - *sensitive instructions*

Slide 11

ITANIUM INSTRUCTION SET

Fixed point arithmetic

add	extr	pcmp	pshr	sxt
addp4	mix	pmax	pshradd	tbit
addl	movl	pmin	psub	tnat
and	mux	pmpy	shl	unpack
andcm	or	pmpyshr	shladd	xchg
cmp	pack	popcnt	shladdp4	xor
cmp4	padd	psad	shr	zxt
czx	pavg	pshl	shrp	
dep	pavgsub	pshladd	sub	

Slide 12

Floating point arithmetic

fabs	fcvt	fnma	fpms	fsetc
fadd	fma	fnmpy	fpneg	fsub
famax	fmax	fnorm	fpnegabs	fswap
famin	fmerge	for	fpnma	fsxt
fand	fmin	fpabs	fpnmpy	fxor
fandcm	fmix	fpack	fprcpa	getf
fchkf	fmpy	fpamax	fprsqrta	mov fr
fclass	fms	fpmerge	frcpa	self
fclrf	fneg	fpmin	frsqrta	xma
fcmp	fnegabs	fpmpy	fselect	xmpy

Slide 13

Branch

br brl brp chk mov.br

Register Stack Engine

alloc flushrs mov.ar.bspstore
clrrrb invala **mov.ar.rsc**^b
cover^a loadrs

^aside-effect at privilege level 0 when psr.ic=0
^bar.rsc contains real privilege level

Slide 14

Memory access

cmpxchg fwb ldfp probe sync
fc^a ld lfetch st
fetchadd ldf mf stf

Other unprivileged

break mov.ar mov.ip nop srlz
epc **mov.cpuid** mov.pr rum
hint ^b mov.um sum

^abypasses protection check at privilege level 0
^breturns cpuid of real processor

Slide 15

Memory management

itc mov.pkr ptc tak tpa
itr mov.rr ptr **thash**^a **ttag**^a

Other privileged

bsw mov.dbr mov.pmc mov.psr rsm
mov.cr mov.ibr **mov.pmd**^b rfi ssm

^anot privileged
^bunprivileged reads return 0 instead of trapping

Slide 16

OTHER ISSUES

Fixed set of privilege levels

- Guest PL 0..3 must be mapped onto 1..3
- Possible loss of protection

Exception handlers live in virtual memory

- Need to co-ordinate address with guest kernel

Complex translation modes

- Separate control over instruction/data/RSE translation
e.g. data physical, register stack virtual
- Difficult to emulate in virtual mode

Register Stack Engine

- Not completely virtualisable
- Partially loaded frames cannot exist outside PL0

DEALING WITH NON-VIRTUALISABLE ARCHITECTURES

Static translation (e.g. vBlades)

- Preprocess OS binary substituting sensitive instructions

Dynamic translation (e.g. VMware)

- Slide 17** → Scan and translate sensitive instructions at runtime

Para-virtualisation (e.g. Xen)

- Manually modify guest operating system

Hardware support (e.g. Intel VT-i)

- Modify the architecture to close virtualisation holes
-
-

INTEL VIRTUALISATION TECHNOLOGY FOR ITANIUM (VT-I)

First introduced in Montecito processor (aka Dual-Core Itanium 2)

- Slide 18**
- New `psr.vm` control bit
 - All sensitive instructions fault with `psr.vm=1` (even in PL0)
 - One bit of address space reserved for hypervisor (allows guest kernel and hypervisor to safely cohabit PL0)
 - `vmsw` instruction for entering/exiting hypervisor (can be used to invoke hypervisor services)
-

BUT...

Pure virtualisation is expensive!

- Slide 19**
- Trap for every privileged instruction
 - Need to read instruction from memory, decode and emulate
 - Some privileged instructions are very common (e.g. enabling/disabling interrupts)
-
-

CUTTING THE COST OF VIRTUALISATION

Virtualisation acceleration

- Slide 20**
- VT-i provides a framework that could support acceleration hardware in the processor
 - However, roadmap for this hardware is not clear
 - Multicore design trends may pressure architects to limit core complexity rather than increasing it
-

CUTTING THE COST OF VIRTUALISATION

Para-virtualisation (e.g. Xen/ia64)

- Guest kernel needs to be ported to hypervisor interface
- ✓ Good performance possible
- ✗ Porting is time-consuming and error-prone

Slide 21 Optimised para-virtualisation (e.g. vBlades)

- Only para-virtualise the performance-critical parts
- ✓ Good compromise, time-performance tradeoff can be chosen
- ✗ Still involves manual porting

Pre-virtualisation (automated para-virtualisation)

- Privileged instructions automatically substituted at assembly time
-
-

PRE-VIRTUALISATION

Step 1. Preprocessor transforms instructions to allow macro replacement:

Slide 22

```
(p6) mov r16=cr.iip
```

↓

```
emul_read_cr pr=p6,dst=r16,cr=19
```

PRE-VIRTUALISATION

Step 2. Macro definition file implements macros for a particular hypervisor:

Slide 23

```
.macro emul_read_cr pr=p0,dst,cr
(\pr)  mov \dst=__vnuma_cpu+CPU_CR_OFFSET+8*\cr
      ;;
(\pr)  ld8 \dst=[\dst]
.endm
```

LMBENCH RESULTS

Basic latencies (Xen, in μ s)

Slide 24

Benchmark	Native	Pure	Para	Pre
null call	0.04	0.96	0.50	0.04
null I/O	0.27	6.32	2.91	0.42
stat	1.10	10.69	4.14	1.43
open/close	1.99	20.43	7.71	2.60
install sighandler	0.33	7.34	2.89	0.50
handle signal	1.69	19.26	2.36	2.23
fork	56	513	164	152
exec	316	2084	578	566
fork+exec sh	1451	7790	2360	2231

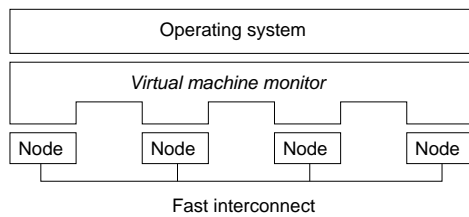
vNUMA

Slide 25

- Itanium virtual machine monitor developed at UNSW
- Native/standalone/Type-1 VMM
- Optimised for performance
 - Written mostly in C, but non-standard runtime conventions to minimise cost of entry to C
- Supports previrtualised guests for even better performance
- Distributed!

vNUMA DISTRIBUTION

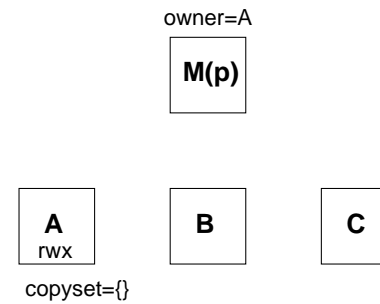
Slide 26



- Run vNUMA on multiple nodes of a cluster
- vNUMA locates and manages CPU/memory/I/O resources
- Presents illusion of a single large NUMA machine to guest OS
- "Physical memory" of virtual machine is distributed using *distributed shared memory* techniques

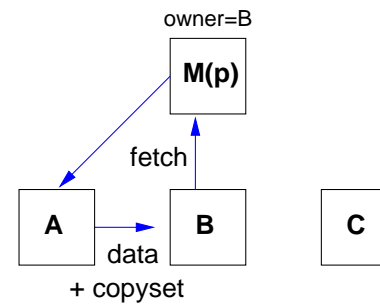
EXAMPLE

Slide 27



EXAMPLE

Slide 28



Why?

Slide 29

- Single machine easy to use and administer
- Allows distribution of legacy applications (e.g. Oracle)
- Cool application of virtualisation!
