Virtualisation Case Study:
Itanium and vNUMA

Matthew Chapman
matthewc@cse.unsw.edu.au

Slide 1

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: “Tukwilla” 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
":
```
Resulting instructions:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MMI</td>
<td>ld8</td>
<td>r4 = (r1)</td>
</tr>
<tr>
<td></td>
<td>ld8</td>
<td>r5 = (r2)</td>
</tr>
<tr>
<td></td>
<td>nop.i</td>
<td>0</td>
</tr>
<tr>
<td>M;MI;</td>
<td>ld8</td>
<td>r6 = (r3)</td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>r7 = r4, r5 // Arithmetic on M too</td>
</tr>
<tr>
<td></td>
<td>nop.i</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>;;</td>
<td></td>
</tr>
<tr>
<td>M;MI</td>
<td>add</td>
<td>r8 = r6, r7</td>
</tr>
<tr>
<td></td>
<td>;;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>nop.m</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>nop.i</td>
<td>0</td>
</tr>
</tbody>
</table>

A better way:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MMI;</td>
<td>ld8</td>
<td>r4 = (r1)</td>
</tr>
<tr>
<td></td>
<td>ld8</td>
<td>r5 = (r2)</td>
</tr>
<tr>
<td></td>
<td>nop.i</td>
<td>0</td>
</tr>
<tr>
<td>M;MI;</td>
<td>ld8</td>
<td>r6 = (r3)</td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>r7 = r4, r5</td>
</tr>
<tr>
<td></td>
<td>;;</td>
<td></td>
</tr>
<tr>
<td></td>
<td>add</td>
<td>r8 = r6, r7</td>
</tr>
<tr>
<td></td>
<td>;;</td>
<td></td>
</tr>
</tbody>
</table>

Itanium Register Set

Large general register file

- partially managed by the register stack engine (RSE)

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
Virtualising the Itanium

- 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

Itanium Instruction Set

### Fixed point arithmetic

- add
- addp4
- addl
- and
- andcm
- cmp
- cmp4
- czx
- dep
- extr
- mix
- movl
- or
- pack
- padd
- pavg
- pavgsub
- pcmp
- pmax
- pmin
- pmph
- pmphshr
- popcnt
- psad
- pshl
- pshladdr
- pshr
- shr
- shrp
- sub
- sxr
- xshl
- xshr
- xzr

### Floating point arithmetic

- fabs
- fadd
- fmax
- fmin
- fand
- fandcm
- fcld
- fclass
- fchkl
- fcmp
- fneg
- fnegabs
- fnma
- fnm
- fnmax
- fnmin
- fnms
- fnms
- fpack
- fro
- frms
- frmin
- fpr
- fquad
- fnegabs
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
- fcmp
-
**Slide 13**

**Branch**
- br
- brl
- brp
- chk
- mov br

**Register Stack Engine**
- alloc
- clrm
- cover
- flushrs
- invala
- loadrs
- mov ar,bspstore

*Side-effect at privilege level 0 when psr.ic=0*

*[arrsc contains real privilege level]*

---

**Slide 14**

**Memory access**
- cmpxchg
- fwb
- ldtp
- probe
- sync

*tc*

**Other unprivileged**
- break
- epc
- hint
- mov ar
- mov cpuid
- mov ip
- mov pr
- mov um
- nap
- rum
- srlz
- sum

*Side-effect at privilege level 0*

*compares protection check at privilege level 0*

*mov cpuid of real processor*

---

**Slide 15**

**Memory management**
- itc
- mov plk
- ptc
- tak
- tpa
- lfr
- mov rr
- ptr
- thash
- ttag

*unprivileged reads return 0 instead of trapping*

---

**Slide 16**

**Other unprivileged**
- bsw
- mov cr
- mov dbr
- mov ibr
- mov pmc
- mov pmd
- mov psr
- rfi
- rsm
- ssm

*Not privileged*

*Not privileged*

**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)
- 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)

- 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!
- 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**
- 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/lA-64)
- Guest kernel needs to be ported to hypervisor interface
- ✓ Good performance possible
- ✗ Porting is time-consuming and error-prone

**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:

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

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

**Pre-virtualisation**

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

```plaintext
.macro emul_read_cr pr=p0,dst,cr(pr)
mov dst=__vnuma_cpu+CPU_CR_OFFSET+8*cr

↓
ld8 dst=[dst].endm
```

**LMbench Results**

Basic latencies (Xen, in μs)

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

- 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

- Run vNUMA on multiple nodes of a cluster
- vNUMA locates and manages CPU/memory/IO 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 13

Slide 25

Slide 26

Slide 27

Slide 28
Why?

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