A case for Virtual Channel Processors

Rolf Neugebauer & Derek McAuley
Intel Research, Cambridge
firstname.lastname@intel.com
Motivation

- Heterogeneous multi-processors
  - Multiple processors (NP, GP, RAID, DSP, …)
  - Multi-core, SMP, SMT
- HW/SW trade-off changes (esp. for I/O)
  - No dedicated off-loading engines
- Traditional OS structure is too inflexible
  - Kernel: one protection domain, one resource domain
  - Difficult to follow HW/SW tradeoff
- Device driver bugs are already a major problem
- Can’t really change the entire OS
Virtual Channel Processors

• (Re-)introduce channel processors for I/O processing
  • Perform I/O on behalf of the OS
• Provide HW independent I/O abstraction
  • Clean and simple interfaces
  • Restructure OS to use channel processors
• Provide performance and fault isolation
  • Execute in different resource & protection domains
• Channel processors are *virtual*
  • May execute on the main CPU(s)
• Enable different, evolving implementations (HW/SW)
  support multiple, heterogeneous cores, novel I/O devices
More on VCPs

• Leverage Virtual Machine technology
  • Execute VCPs or part of VCPs in a VM
  • “clean slate”, software becomes simpler
  • Software can be optimised for the task at hand
  • Run device drivers within a VCP
  • OS independent device drivers

• Provide an idealised I/O interface to OS and Apps
  • Asynchronous, zero copy message queues
The big picture

- Application
- Guest OS VM
- Application
- OS Kernel
- VMM
- VCP
- VCP
- Hardware
Example: iSCSI

- Non-public APIs
- Interactions between sub-systems
  - e.g., buffer-cache vs socket buffer
  - HW offloading?
- Complex implementation

- Simple API
- Optimised network implementation
- HW offloading contained within VCP
- Better resource control
Implementation Plans

- Use the Xen Virtual Machine Monitor
  - Virtualisation with low overhead (cf SOSP’03)
  - Runs Linux (and others) as a guest operating system
- Use network processing/iSCSI as test app
  - Other examples: RAID, soft-modems
- Investigate use of SMT and SMP features
- Incorporate novel HW (cf TwinCities)
Challenges

- Interfaces between VCPs and OS & APPs
  - Mainly for the control path

- Changes to the OS kernel required

- Performance impact

- Scheduling

- Fault recovery
Related Work

- Original Channel Processors (IBM CP-67, VM/370)
  - Dedicated hardware, channel programs
- TCP Servers/Split-OS, ETA, Piglet/AsyMOS
  - Dedicate processors to I/O processing
  - VCPs provide a more flexible abstraction
- Micro Kernels
  - Execute device drivers etc in different processes
  - VCPs are less generic, focus on bulk data transfer
- I/O interfaces: FBufs, IO-Lite, RBufs, …
Conclusions

- Virtual Channel Processors (VCPs)
  - Encapsulate I/O processing
  - Provide performance and fault isolation
  - Provide flexibility for changing HW/SW tradeoffs

- Implementation plans
  - Based on the Xen VMM
  - iSCSI as sample application
Questions?

Rolf.Neugebauer@intel.com
Performance

Figure 3: Relative performance of native Linux (L), XenoLinux (X), VMware workstation 3.2 (V) and User-Mode Linux (U).
# Network Performance

<table>
<thead>
<tr>
<th></th>
<th>TCP MTU 1500</th>
<th></th>
<th>TCP MTU 500</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TX</td>
<td>RX</td>
<td>TX</td>
<td>RX</td>
</tr>
<tr>
<td>Linux</td>
<td>897</td>
<td>897</td>
<td>602</td>
<td>544</td>
</tr>
<tr>
<td>Xen</td>
<td>897 (-0%)</td>
<td>897 (-0%)</td>
<td>516 (-14%)</td>
<td>467 (-14%)</td>
</tr>
<tr>
<td>VMW</td>
<td>291 (-68%)</td>
<td>615 (-31%)</td>
<td>101 (-83%)</td>
<td>137 (-75%)</td>
</tr>
<tr>
<td>UML</td>
<td>165 (-82%)</td>
<td>203 (-77%)</td>
<td>61.1 (-90%)</td>
<td>91.4 (-83%)</td>
</tr>
</tbody>
</table>

Table 6: ttcp: Bandwidth in Mb/s
Network Performance

Figure 4: SPEC WEB99 for 1, 2, 4, 8 and 16 concurrent Apache servers: higher values are better.

Figure 5: Performance of multiple instances of PostgreSQL running OSDB in separate Xen domains. 8(dif) bars show performance variation with different scheduler weights.
Para-Virtualization

Retain ABI
Run guest OS at lower privilege (ring 1 on IA32)
Port guest OS to make calls to hypervisor rather than priv instrs
  - e.g. cli/sti, page table updates
Use virtual device drivers (timer, network, disk)
  - exports "ideal" device interface
  - greatly reduces overhead
Exposing real resources can be beneficial
  - time: virtual (scheduling) and real (SRT apps, TCP timers etc)
  - machine memory (page colouring)
  - physical disk number (for sw raid)
## Porting OSes

<table>
<thead>
<tr>
<th>Section</th>
<th>Linux</th>
<th>XP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arch indep</td>
<td>78</td>
<td>1299</td>
</tr>
<tr>
<td>Virtual network</td>
<td>484</td>
<td></td>
</tr>
<tr>
<td>Virtual block driver</td>
<td>1070</td>
<td></td>
</tr>
<tr>
<td>Xen specific (non-driver)</td>
<td>1363</td>
<td>3321</td>
</tr>
<tr>
<td>Total</td>
<td>2995</td>
<td>4620</td>
</tr>
<tr>
<td>Code base</td>
<td>220k</td>
<td>1200k</td>
</tr>
</tbody>
</table>

Linux and NetBSD almost all changes to arch-dependent code

XP MM HAL interface isn't ideal, requires “arch-indep” changes

Effort small relative to writing a new OS