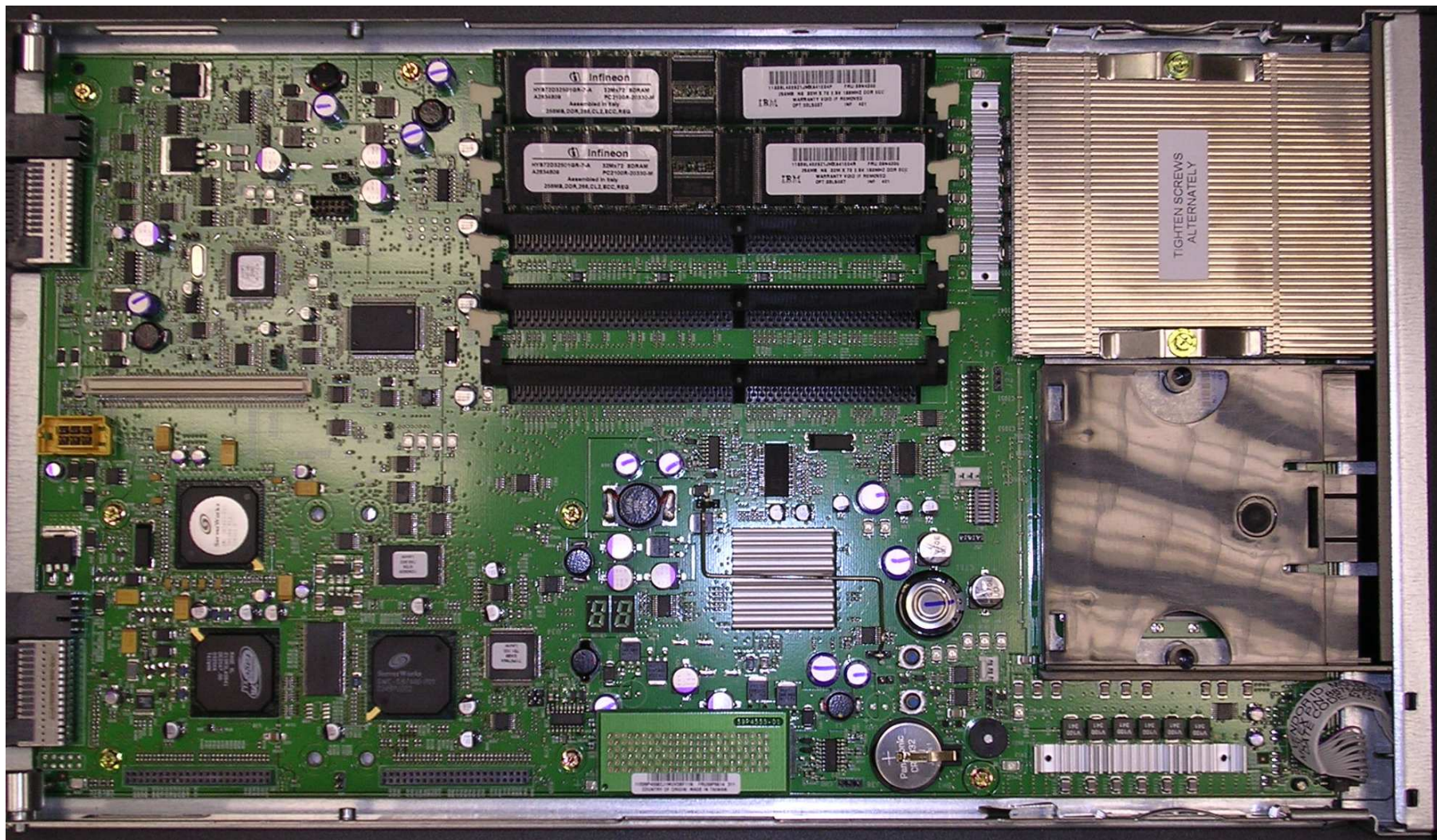


IP-Only Server

**Muli Ben-Yehuda, Oleg Goldshmidt, Hillel Kolodner,
Zorik Machulsky, Vadim Makhervaks, Julian Satran,
Marc Segal, Leah Shalev, Ilan Shimony**

IBM Haifa Research Laboratory

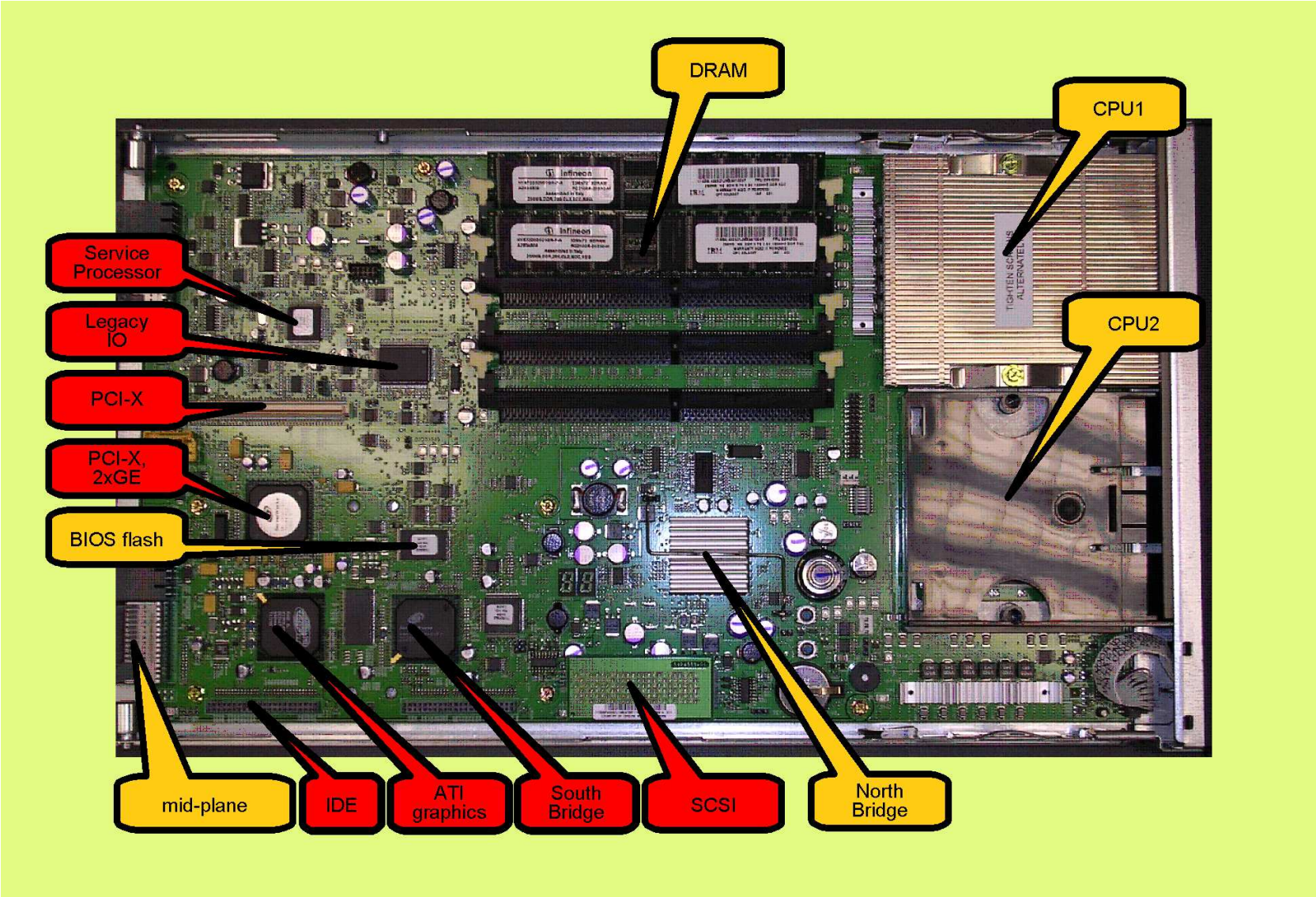
Do Servers Look Different from PCs?



Should Servers Look Different?

- probably yes:
 - disks can — and should — be remote: FC or iSCSI (including boot)
 - removable media (CD-ROM, floppy, etc.) are used only for installation, and are not used at all with modern installation methods
 - no users working directly on the console, remote administration
 - SSH/X
 - MS Windows Terminal Services
 - VNC
- KVM console is still needed for some operations, notably “before” and “after” OS — BIOS configuration, crash (BSOD, panic)

How Different Can Servers Be?



So Let's Remove The Legacy Stuff!

- universal I/O channel — IP/Ethernet
- iSCSI disk access (including boot)
- “remote” all legacy peripherals over network:
 - KVM
 - USB
 - removable media
 - etc.

Advantages of Removing Stuff

- less components on board
 - improved MTBF
 - real estate savings
 - better air flow
 - power savings
- management access from anywhere, all the time
 - no more walking around with a laptop and a serial cable
 - a remote user calls, says he cannot boot...
- simultaneous access to multiple machines
- multiplatform solution with no loss of functionality

Hasn't This Been Done Already?

- lots of software solutions to access OS and applications, **when the OS is operational**
 - SSH/X, MS WTS, VNC, SunRay, Citrix, USB/IP
- **add-on** HW solutions: KVM/IP (Cyclades, etc.)
- **add-on** HW/FW: SOL — IBM's RSA, Intel's AMT/vPro
- **partial**: IBM Power blades don't have a graphics adapter
- **specific to Linux/LinuxBIOS**: remote access to pre-OS and post-OS environments
 - LinuxBIOS, netconsole, netdump
- Super Dense Server (IBM) (**also specific**)
 - small board with CPU, DRAM, NIC, flash
 - LinuxBIOS, modified Linux, COE (text mode)

Hardware Emulation

- emulate legacy controllers in a single, simple, cheap piece of hardware
 - ASIC or FPGA
 - HW identifies itself as the legacy controllers
 - captures all the relevant bus transactions and sends over network, the actual I/O is performed on the other end
- completely transparent to all software and firmware
 - applications, OS, bootloader, BIOS
 - run the usual software without modifications
- essential for pre-OS and post-OS environments
 - the main CPU, NIC need not be operational

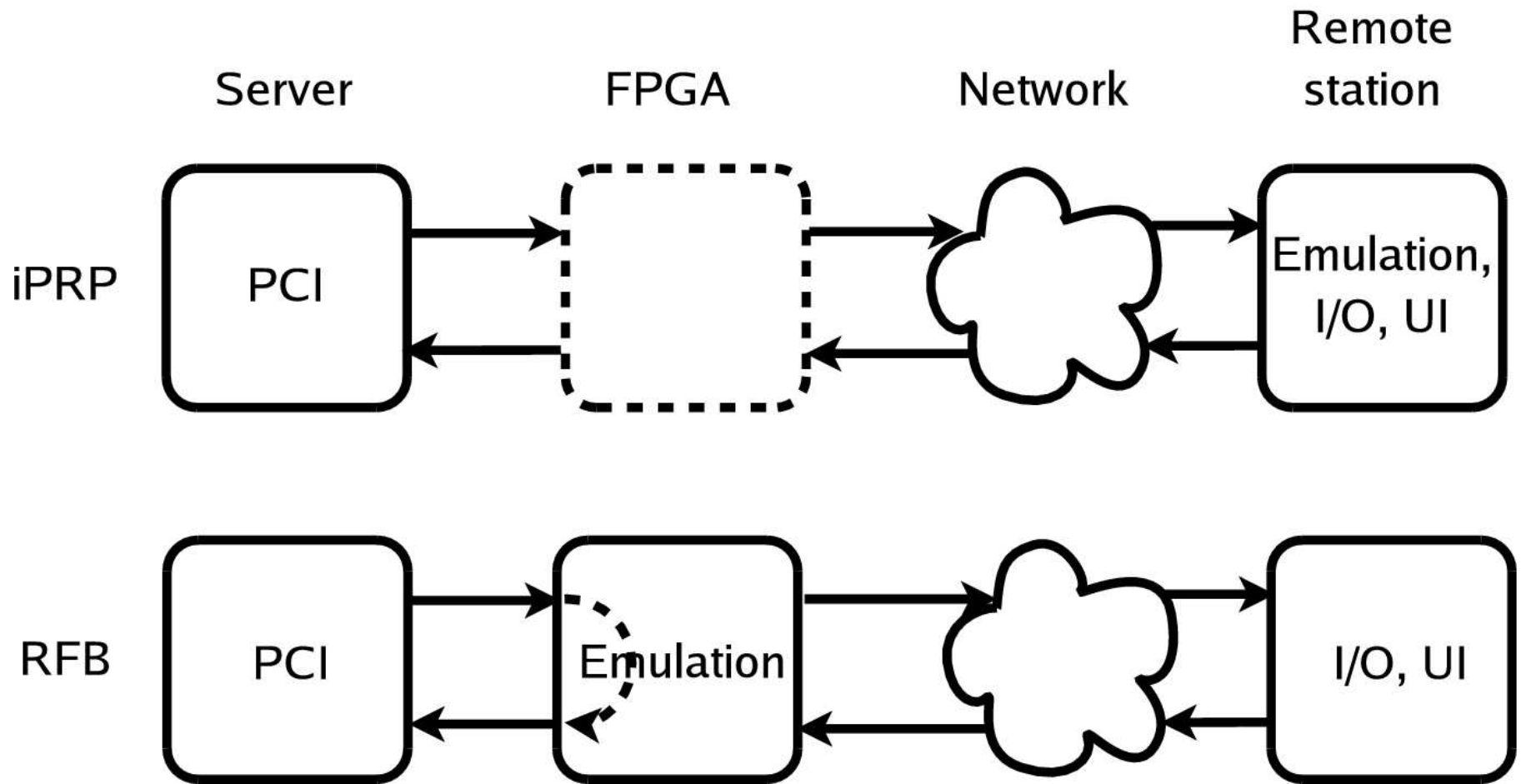
Design Guidelines

- run unmodified software/firmware stack
- remote access from power-on to power-off or crash
 - while OS is operational other methods may be more efficient
- minimal state for disconnected operation
 - boot over network, RTC initialized by remote device
- must operate disconnected (except storage)
- text and graphical (VGA) support
- open protocols, remote station not proprietary
- concurrent access to multiple servers

Approaches to I/O Emulation

- iPRP: PCI over IP
 - capture PCI transactions, send as IP packets
 - receive responses as IP packets, send to PCI bus
 - **server cannot work disconnected**
 - debugging tool, mainly to find all the places where BIOS and OS touch local devices
- RFB: Remote Framebuffer (protocol behind VNC)
 - translate KVM PCI transactions to RFB, send to remote station
 - the remote station is a normal VNC client
 - network is necessary only for display and user input, other PCI transactions can be processed locally
 - **can operate disconnected**

iPRP vs. RFB



Implementation

- Xilinx FPGA evaluation board with Ethernet plug-in
 - 50 MHz
 - 378 K of memory
 - 8400 logic slices ($\approx 10^6$ gates)
- major components:
 - expansion ROM
 - PCI interface
 - network interface
 - transaction processing

Expansion ROM

- Something needs to tell the host (the BIOS) that it is a video card
- Implemented a BIOS expansion ROM on the FPGA evaluation board
 - contains the VGA initialization routines
 - invoked by the main system BIOS during boot
 - based on the GPL VGA BIOS from the Bochs x86 emulator
 - with insignificant modifications

PCI Interface I

- responsible for correct presentation and behavior on the PCI bus
- answers to the following addresses on the bus:
 - I/O addresses 0x60, 0x64 — keyboard controller
 - I/O ranges 0x3B0 ÷ 0x3BB, 0x3C0 ÷ 0x3DF — VGA controller
 - memory range 0xA0000 ÷ 0xBFFFF — VGA memory

PCI Interface II

- VGA
 - BIOS discovers the device as an add-on VGA adapter
 - considers it the default (as opposed to the on-board one)
 - configures the chipset so that all VGA I/O and memory transactions are routed to the device
- KB and mouse require NB and I/O bridge configuration by BIOS
- transactions are captured and placed into a FIFO

Network Interface

- low-level interface to the Ethernet adapter
- DMA engine
- in general, does not have to be separate
 - overhead is small
 - separate link is useful, in case the main link is down or saturated
- in our implementation, iSCSI used the main link, and KVM used the FPGA's Ethernet interface

Transaction Processing

- a single firmware loop
 - get transactions from the FIFO (put there by the PCI interface module)
 - if possible, process locally
 - else wrap the transaction in an iPRP or RFB packet and send to the network
 - read responses from network forwarded to the PCI interface, sent to bus

iPRP Protocol I

- UDP-based
 - includes a simple recovery mechanism
 - UDP checksum for data correctness
 - go-back-N for delivery correctness
- emulates PCI memory and I/O transactions, configuration space accesses
- main design guideline: simplicity and ease of implementation
 - not a PCI implementation, cannot be used to talk to a PCI bridge
- makes FPGA transparent

iPRP Protocol II

- command: a PCI transaction, e.g., memory read, I/O write, ACK
- message: N commands in a UDP packet
 - only the last command may be a read
- send triggers: read, timeout, maximal message size

iPRP Protocol: Debugging

- remote station based on the PCI device code from the open source Bochs emulator
- fed it the PCI transactions received as iPRP payload
 - for host writes updated the VGA, keyboard, mouse state machines, displayed the Bochs screen to user
 - for host reads send response back
- exceptional debugging tool:
 - trivial to extract and log the actual server PCI transactions on the remote station
 - the log can drive a standalone or emulation version of the RFB-based firmware for debugging or verification
- remarkably robust — ran for days on end (light load)

RFB Protocol I

- based firmware on Bochs code (same as used for iPRP remote)
 - ported to C, eliminated the use of libraries and facilities FPGA does not have
 - reduced memory footprint (< 250 K), improved efficiency (50 MHz, no caches)
- implemented a VNC server in firmware
- open protocol, remote station can be a regular VNC client
- Bochs includes an RFB implementation (needed many modifications)
- removed VGA modes not used by Linux or Windows, 128 K framebuffer

RFB Protocol II

- custom embedded TCP stack
 - very low memory, minimal copying
 - implements a complete TCP state machine
- single loop, no context switching, no memory management
 - local processing, host reads answered immediately
 - heuristics to decide when to update the remote station

Dirty Tricks and Black Magic

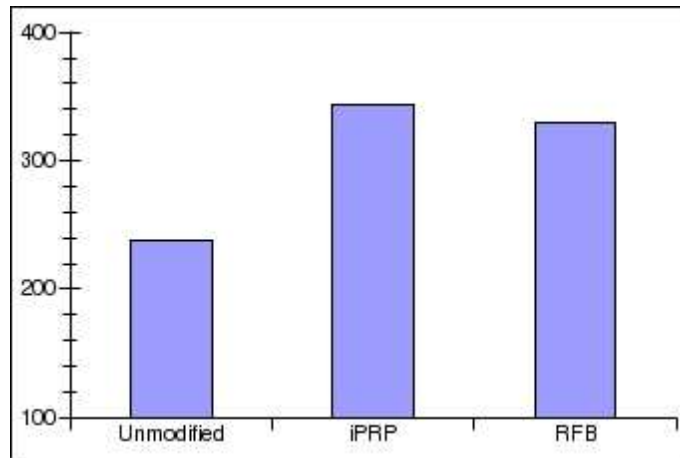
- coalesce sequential write transactions in larger packets
- RFB: handle read transaction locally
- iPRP: send reads over immediately, to avoid “bus parking”, host CPU stalling
- interrupt requests from network (e.g., keyboard)
 - need to generate IRQ1, but cannot do it with an add-on card
 - issue a write PCI transaction to the chipset’s OpenHCI USB register (legacy keyboard emulation), chipset raises IRQ1, host CPU accesses 0x60 I/O port
 - captured by FPGA, that responds with the 0x60 content

Experimental Setup

- 2 identical IBM x235 servers with PCI-based FPGA evaluation boards with 100 Mb/s network interfaces
- remote station: TP R40, 1.4 GHz Pentium M, 512 M RAM
- servers booting Windows 2003 Server or RHEL 3.0
- iSCSI targets on Intellistation MPro PCs (via main NIC)
 - used local disks for KVM performance measurements
- compared wall time of server boot
 - native, iPRP, RFB
 - from power-on to Linux console prompt or “Welcome to Windows”
- compared user experience after boot

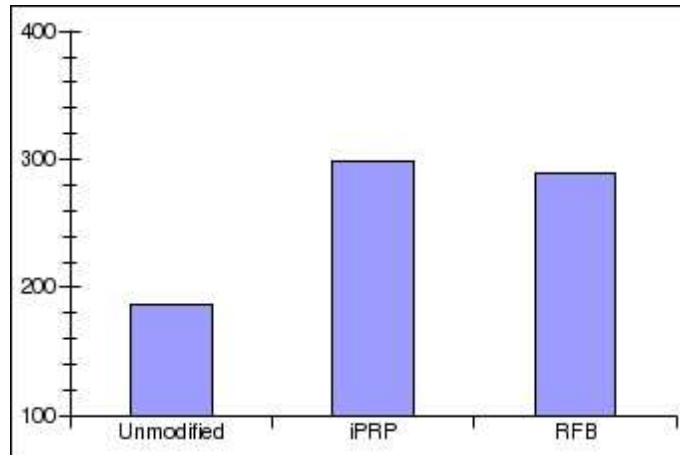
Boot Performance Evaluation

NB: 50 MHz CPU, no caches, **no tuning**



Linux:

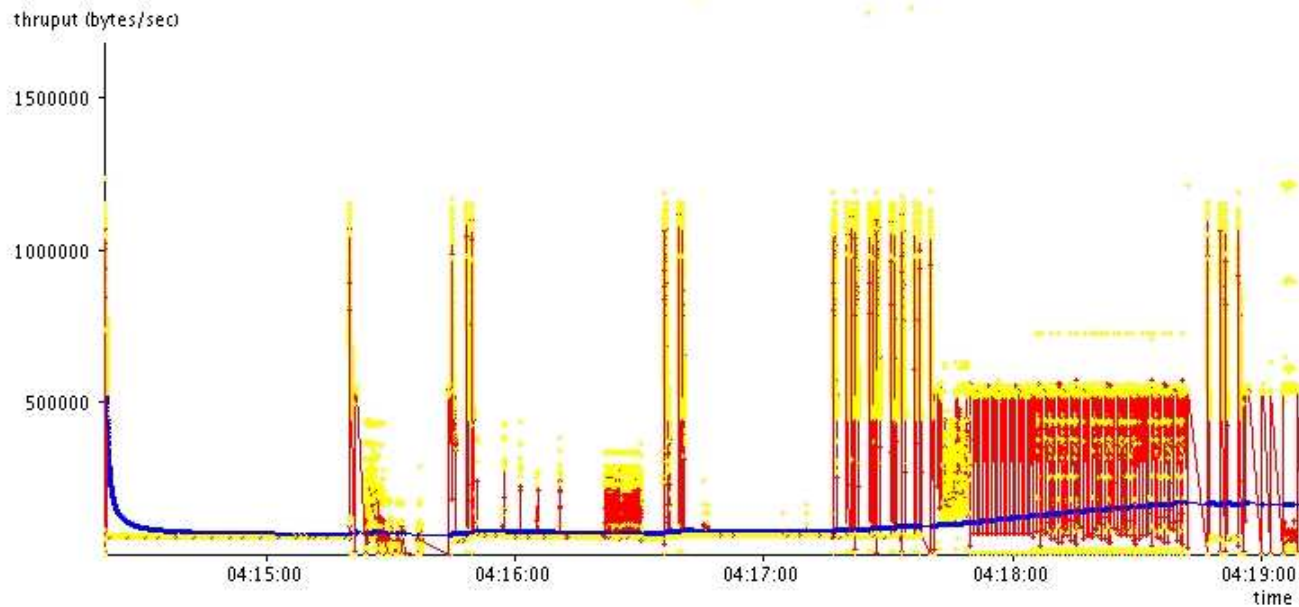
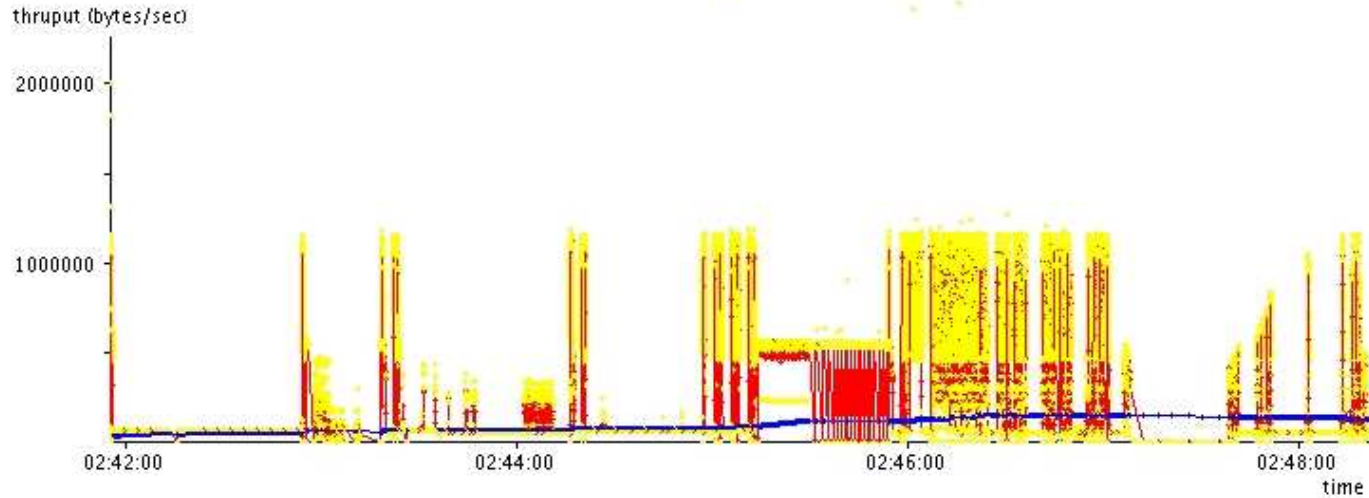
- native: 238 sec
- iPRP: 343 sec
- RFB: 330 sec (38% slowdown)



Windows:

- native: 186 sec
- iPRP: 299 sec
- RFB: 289 sec (55% slowdown)

Network Throughput



Summary

- we designed and implemented an “IP-Only Server” featuring:
 - a single I/O channel — IP/Ethernet network
 - iSCSI disk access (including boot)
 - I/O emulation HW provides remote KVM access from power-on to power-off or crash, concurrently to multiple servers
 - no modifications to software stack
 - OS-independent, platform-independent
 - remote station a VNC client, can be disconnected and reconnected at will, from anywhere
 - some slowdown (FPGA a bottleneck), acceptable user experience

Prospects

- support more peripherals
- port to other architectures (e.g., Power)
- virtualization:
 - give unmodified guests physical device access, while providing isolation and offering each guest its own view of I/O subsystem
 - transparent virtualization layer: separate sets of device control registers for different partitions
 - share a remote device between several servers
- numerous (network-related) security issues
- I'd like to have one at home (monitor in the living room, CPU, disks, fans in a closet)

MORE INFORMATION:

- USENIX'06 proceedings.
- IBM Research Technical Report
- Feel free to write to `olegg@.il.ibm.com`

QUESTIONS?

BACKUP