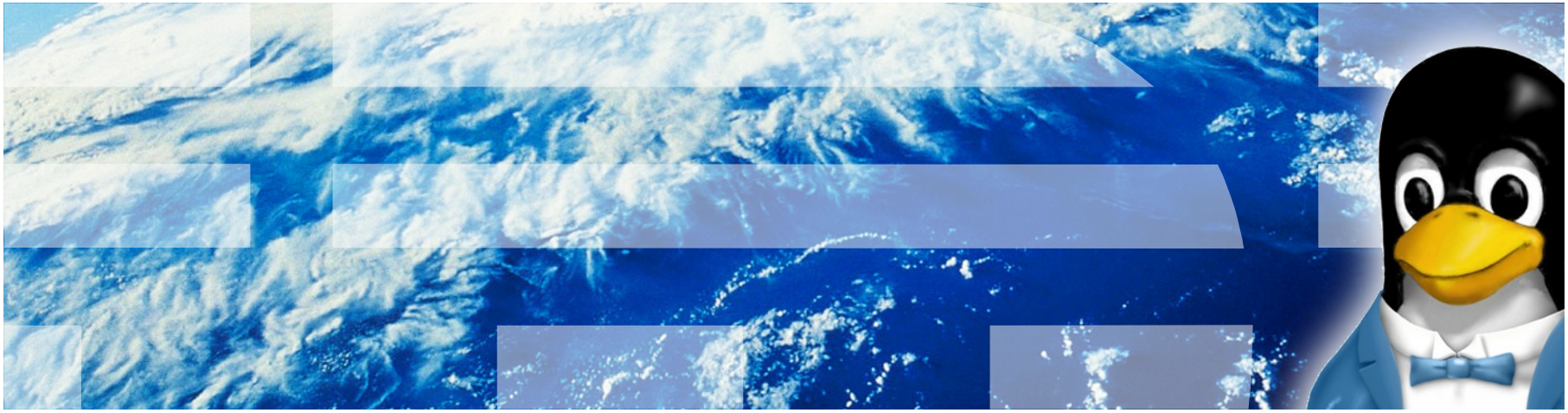
KVM on IBM POWER8 machines



Paul Mackerras, IBM LTC OzLabs

paulus@au1.ibm.com

Outline

- PowerKVM distribution
- POWER8™
- Thread modes
- Guest SMT modes
- Future work

New IBM Power Systems based on POWER8 – April 2014

- POWER8 roll-out is leading with scale-out (1-2S) systems
- Expanded Linux focus: Ubuntu, KVM, and Open Stack
- Scale-up POWER8 (>2S) systems will be rolled out over time
- PCI Gen3 right out of POWER8 processor
- OpenPOWER Innovations

Power Systems S812L

- 1-socket, 2U
- Linux Only

Power Systems S822L

- 2-socket, 2U
- Linux Only

Power Systems S822

- 2-socket, 2U
- All Operating Systems

Power Systems S814

- 1-socket, 4U
- All Operating Systems

Power Systems S824L

- 2-socket, 4U
- Linux Only
- SOD

Power Systems S824

- 2-socket, 4U
- All Operating Systems

1 & 2 Sockets

Linux

ubuntu

PowerVM

openstack

SUSE

PowerSC PowerVC

PowerKVM

redhat

PowerHA PowerVP

PowerKVM

- **Hypervisor / Host OS**
  - Available as optional preload on IBM POWER8-based Linux-only servers
- **Supports Linux on Power guests**
  - Both big-endian and little-endian
  - RHEL6.5, RHEL6.6, RHEL7 and later releases
  - SLES11 SP3 and later releases
  - Ubuntu 14.04 and later releases
  - Fedora >= 18 and recent OpenSUSE
- **Open source**
  - Based on Fedora 19
  - Kernel 3.10.23 + local modifications
  - QEMU 1.6 + local modifications
  - Libvirt 1.1.3 + local modifications
  - Kimchi
- **Open system**
  - Administrator can ssh in as root
  - Can install additional software, e.g. cloud management agents
  - Can be replaced by other distributions

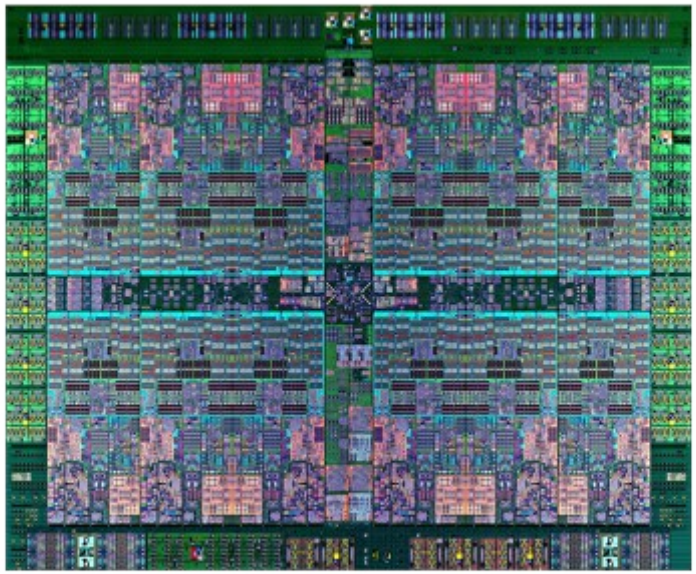
PowerKVM

- **Supported by IBM**
  - Has undergone extensive testing
  - Each version supported for 3 years from initial release
- **Management**
  - Virsh, kimchi
  - OpenStack and oVirt agents can be installed
- **Releases**
  - v2.1.0 (initial release) 10 June 2014
    - v2.1.0.1 (SP1) 30 June 2014
    - v2.1.0.2 (SP2) 29 August 2014
  - v2.1.1 October 2014
    - PCI pass-through supported
    - Kernel 3.10.53, QEMU 2.0, libvirt 1.2.5
  - v3.1 next year
  - Updates can be done with yum or by reinstalling
- **Very similar interfaces and usage patterns to x86 KVM-based hypervisors**
  - Some differences inside guest due to Linux on Power platform conventions vs. x86
    - e.g. use device tree instead of DMI data
  - Some differences in virtual to physical CPU mappings (see below)

POWER8 Processor – Built for Big Data

Technology

•22nm SOI, eDRAM, 15 ML 650mm2



Cores

- 12 cores (SMT8)
- 8 dispatch, 10 issue, 16 exec pipe
- 2X internal data flows/queues
- Enhanced prefetching
- 64K data cache, 32K instruction cache

Accelerators

- Crypto & memory expansion
- Transactional Memory
- VMM assist
- Data Move / VM Mobility

Caches

- 512 KB SRAM L2 / core
- 96 MB eDRAM shared L3
- Up to 128 MB eDRAM L4 (off-chip)

Energy Management

- On-chip Power Management Micro-controller
- Integrated Per-core VRM
- Critical Path Monitors

Memory

- Up to 230 GB/s sustained bandwidth

Bus Interfaces

- Durable open memory attach interface
- Integrated PCIe Gen3
- SMP Interconnect
- CAPI (Coherent Accelerator Processor Interface)

ComputerWorld: To make the chip faster, IBM has turned to a more advanced manufacturing process, increased the clock speed and added more cache memory, but perhaps the biggest change heralded by the Power8 cannot be found in the specifications. After years of restricting Power processors to its servers, IBM is throwing open the gates and will be licensing Power8 to third-party chip and component makers.

The Register: the Power8 is so clearly engineered for midrange and enterprise systems for running applications on a giant shared memory space, backed by lots of cores and threads. Power8 does not belong in a smartphone unless you want one the size of a shoebox that weighs 20 pounds. But it most certainly does belong in a badass server, and Power8 is by far one of the most elegant chips that Big Blue has ever created, based on the initial specs.

PCWorld: With Power8, IBM has more than doubled the sustained memory bandwidth from the Power7 and Power7+, to 230 GB/s, as well as I/O speed, to 48 GB/s. Put another way, Watson's ability to look up and respond to information has more than doubled as well.

Microprocessor report: Called Power8, the new chip delivers impressive numbers, doubling the performance of its already powerful predecessor, Power7+. Oracle currently leads in server-processor performance, but IBM's new chip will crush those records. The Power8 specs are mind boggling.

Source: Hotchips presentation

POWER8

- **Eight threads per core (SMT8)**
  - Up from 4 threads per core in POWER7
  - Core automatically switches between ST (single-threaded), SMT2, SMT4 and SMT8
  - Automatic rebalancing of threads between thread sets
  - All threads share MMU context, therefore must be in same partition
    - Hypervisor code can run in real mode (MMU off)
- **Core can be split into 4 subcores (“micro threading mode”)**
  - Two threads per subcore
  - Separate MMU context per subcore, therefore 4 guests can share one core
  - Core permanently in SMT8 mode
  - Each subcore gets 1 dispatch cycle in 4
    - Reduces impact of one subcore on another, for predicable performance
  - Currently implemented: single system-wide selection of mode
    - Can only be changed when no guests are running

POWER8

- **Fast IPI mechanism**
  - Message-send (msgsnd) instruction and new interrupt vector
  - Intra-core variant available to supervisor-mode code (OS level)
    - Destination thread number in register for msgsndp instruction
    - My thread number in new special-purpose register, TIR
    - Thread numbers are not virtualizable
- **Micro-partition prefetch (MPP)**
  - Mechanism for hardware to write L2 or L3 cache address tags to memory buffer
  - Mechanism for hardware to prefetch cache lines given addresses in buffer
  - Implemented in PowerKVM kernel on guest context switch
  - Reduces impact of moving VCPUs from one core to another
- **Transactional memory**
  - Allows instructions to be grouped into “transactions”
  - Loads and stores of a transaction appear to be performed atomically from the point of view of other CPUs
  - Transaction can be aborted at any time
    - CPU state rolled back to the beginning of the transaction
    - Two copies of all user-visible state: “transactional” and “checkpointed” state
  - Extra state to be restored/saved on VCPU entry/exit when TM is active

Thread modes

- **How do we accommodate the requirement for all threads in a subcore to be in the same partition?**
- **Current solution: Host runs single-threaded: only one thread per (sub)core online**
  - Whole core mode: CPUs 0, 8, 16, 24, … online
  - 4-way split-core mode: CPUs 0, 2, 4, 6, 8, … online
  - Other CPUs (SMT threads) offline and in “nap” mode (power-saving state)
    - Scheduler will not schedule any tasks on them
    - But they can be woken with an IPI and used to run guest VCPUs
- **Any VCPU task can enter the guest without coordinating with other CPU threads**
  - Guest entry is fast
- **Running more than one VCPU in a (sub)core:**
  - One VCPU tasks takes responsibility for all the VCPUs running on the (sub)core
    - Call this the “runner” task
  - Other VCPU tasks sleep waiting for their VCPU to need service in the kernel (e.g., execution in userspace or in-kernel page-fault or hypercall handling)
- **One out, all out:**
  - Whenever one VCPU needs to exit to the host kernel (MMU on), all guest VCPUs executing on the same core must exit

Thread modes

- **Disadvantage: execution of guest VCPU is disconnected from VCPU task**
  - Accounting on host side reflects (sub)core usage rather than VCPU usage
    - Total CPU time consumed seems lower on host than guest
    - Utilization seems higher on host than guest
  - nova-libvirt driver defines available VCPUs for a given host as the physical active CPUs
    - Thinks only 12 VCPUs can be run on a 12-core machine
  - Users get confused about how many VCPUs they can run without over-committing
    - Answer depends on the thread mode
  - Virsh vcpuinfo output is confusing for users: doesn't show guest SMT mode, shows some VCPUs as sleeping when they are actually active
  - Libvirt <placement='auto'> uses numad, which also thinks you need one online CPU per VCPU
  - CPU hotplug becomes confusing
    - Guests expect cores to be added/removed not individual threads
  - Pinning VCPU tasks to a single CPU can cause problems
  - Competition between execution in host and execution in guest

Thread modes

- **Possible alternative proposal: run host in SMT8 mode**
- **Need fast, efficient mechanism to get control of other threads in core**
  - KVM kernel thread per CPU
  - Task wakeup takes ~10 µs vs. ~1—2 µs for waking offline thread from nap
  - Subject to scheduler's decisions about what to run on those other threads
- **Dynamic split-core mode**
  - Use whole-core mode when all threads are in the same guest or all in the host
  - Use split-core mode otherwise
  - Minimize number of CPU threads which we need control of
- **Advantage: assumptions about one host CPU == 1 VCPU become true**
- **Disadvantages: complexity and possible inefficiency**

Guest SMT modes

- **How do we schedule guest VCPUs onto physical CPU cores?**
- **Current solution: allocate VCPUs to virtual cores (vcores) according to requested threading mode**
  - Run a vcore on a physical core whenever any VCPU task in the vcore wants to enter the guest
  - VCPUs in a vcore always run in the same core as each other
  - On POWER8, VCPUs 0—7 in vcore 0, 8—15 in vcore 1, etc.
  - ST (threads=1, default): number VCPUs 0, 8, 16, …
  - SMT8: number VCPUs 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, …
- **Advantage: guests can control their own SMT mode**
  - Virtual siblings are actual physical siblings
    - Cache-aware scheduling in the guest is meaningful
  - Guest can choose SMT mode by off-lining/on-lining CPUs
    - ppc64_cpu command makes this easy
- **Disadvantage: default of threads=1 uses more resources than naively expected**
  - Users expect one VCPU to be a thread not a whole core

Guest SMT modes

- **Alternate proposal: for threads=1, pack VCPUs into cores as densely as possible**
  - Whole-core mode: run any 8 VCPUs from the same guest together on a core
  - Split-core mode: run any 2 VCPUs from the same guest together on a core
- **For threads=2, 4 or 8, bind VCPUs into vcores**
  - Gives advantages of explicit SMT control and cache sharing control for knowledgeable users
  - VCPU binding needed to allow guests to use msgsndp instruction

Future work

- **Implement and assess alternate proposal for guest SMT modes**
- **Implement and assess alternate proposal for thread modes**
- **Reduce differences vs. x86**
  - Implement SPICE
  - Implement CPU/memory hotplug
- **Little-endian host**
- **Improve performance**
- **Support guest use of on-chip accelerator hardware**
- **Better/easier NUMA affinity management**

Legal Statement

This work represents the view of the author and does not necessarily represent the view of IBM.

Guest entry/exit