Introduction to UltraSPARC Architecture

June 2009

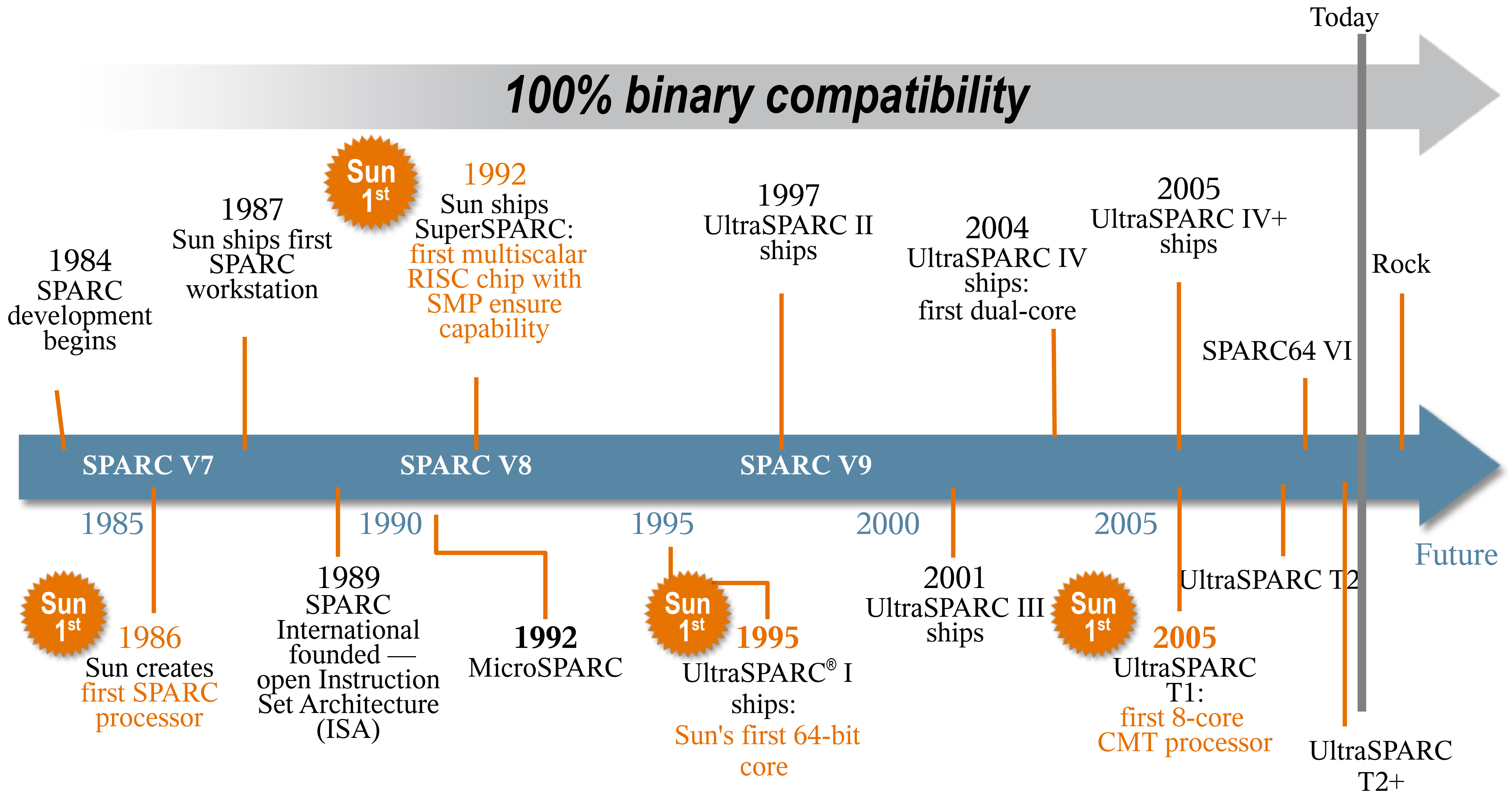
David Weaver
Principal Engineer, UltraSPARC Architecture
Principal Evangelist, OpenSPARC
Architecture Group
Sun Microelectronics

Agenda

1. SPARC Origins
2. RISC Principles
3. SPARC Basics & Architecture Generations
4. CMT Control in UltraSPARC Architecture
5. UltraSPARC Architecture Reference Material

The History of SPARC®

More than two decades of continuous technical innovation



SPARC Origins

Sun microsystems

Timeline: 1983-1984

- 68020, Z8000, and (to a lesser degree) 80286 were key processors in the workstation space
  - all of them required coprocessors and/or add-on boards to attain respectable floating-point performance
- Sun (like most Unix vendors) was using 68020
  (the main non-680x0 Unix player was DEC w/ VAX 11/7x0)
- PA-RISC and MIPS were also in development
- Berkeley RISC and Stanford MIPS were big in academia
- all GP processors were in-order, single-issue
- compilers were not particularly sophisticated
- Motorola was *not* keeping pace, esp in floating-point – what could Sun do?

University Roots

- SUN
  - > Founders three from Stanford (Business and Hardware) and one from Berkeley (Software, BSD-Unix)
  - > **S**tanford **U**niversity **N**etwork
- SPARC
  - > **S**calable **P**rocessor **ARC**hitecture
  - > Based on Berkeley's RISC projects
  - > Different than MIPS (Stanford's project) mainly in register windows and MIPS pipeline focus
  - > First successful commercial RISC processor design clearly demonstrated the power of RISC concept and helped ushered in the generation of RISC/modern processor design until now

“Sunrise” project, ~1983-1986

- Initially, planned as a gate-array-based *floating-point coprocessor* for Sun's 680x0 systems
- Observations:
  - extend it to be a GP processor?
  - even in a *gate-array* implementation, it:
    - could trounce 680x0 performance
    - have lots more future headroom than 680x0 appeared to have
- Decided to proceed with GP processor
  - two gate-array chips -- “IU” integer unit plus ”FPU”
  - MMU, I/O, memory controller, etc all off-chip
  - 2-chip implementation => separate f.p. regs & instructions, FQ

“Sunrise” architecture

- 32-bit RISC architecture
- Derived largely from Berkeley RISC design
  - Professor David Patterson hired as part-time consultant
- In mid-1987, initial Sun 4/260 workstation was *much* faster than contemporary 680x0 and i386 systems
- Renamed SPARC, “**S**calable **P**rocessor **Arc**hitecture”, before product release in 1987
  - wanted “Sun” out of the name, so would be more openly available/implemented in industry

RISC Principles

Sun microsystems

RISC Principles

- RISC = "Reduced Instruction Set Computing"
  - less-complex instructions meant short cycle times
  - build up complex operations from simpler ones
  - common cases execute faster => overall speedup
- Instructions
  - all fixed-length (32 bits)
  - no complex instructions
    - almost *all* instructions with single-cycle latency
    - longer latency: load, store, floating-point
    - no HW integer mul/div! (1-cycle multiply-step instead; later changed)
  - addressing modes: immediate, register, reg + imm, reg + reg

RISC Principles (2 of 2)

- Register-oriented architecture
  - No memory-to-memory operations. Period. (still true)
- Memory accesses
  - all *aligned*
  - flat virtual address space (e.g. no segments)
  - bi-endian (default = big-endian)
- RISC rule of thumb:

  If a new feature adds x% to the cycle time,
  it must add *more than* x% to the overall performance
  to be considered.

SPARC Basics and Architecture Generations

Sun microsystems

Architecture vs. Implementation

- Clear distinction!
- ***Architecture*** = ISA (instruction set architecture)
  - "what software sees"; performance/timing **not** considered
  - is specified in *UltraSPARC Architecture* ("UA") doc
  - indicates ISA which is:
    - common across current implementations *and*
    - is likely to *continue* into the future
- ***Implementation*** = an instantiation of that architecture
  - a hardware implementation (a.k.a. "microarchitecture")
  - or a combination of HW + emulation SW
  - or a software simulator (pure software)
  - is specified in *Implementation Supplement* to UA specs

SPARC V7 (1987)

- Followed RISC principles
- 32-bit instructions
- 32-bit data and 32-bit virtual addresses
- “word” = 32 bits (4 bytes)
  “doubleword” = 64 bits (8 bytes)
- Separate integer and floating-point register files
- Condition codes
  - one set integer condition codes
  - one set f.p. (“fcc”) condition codes
  - two versions of many integer ops exist --
    one that sets integer condition codes and one that does not

SPARC V7 (1987)

(2 of 3)

- F.p. registers of different precisions overlaid
  - For example,
    5th single-precision register = **f4**
    3rd double-precision register = **f4** + **f5**
    2nd quad-precision register = **f4** + **f5** + **f6** + **f7**
- Delayed Branches (helped hide branch latency)
- "Windowed" Register File
  (fast parameter passing/return -- fewer memory accesses, for most apps)
- misc "interesting" features, like Tagged Add/Subtract
  (for accelerating LISP execution)

SPARC V7 (1987)

- Address Space Identifiers (ASIs)
  - 8-bit value that identifies a particular address space
  - An ASI is (implicitly or explicitly) associated with **every** instruction access or data access
  - Access data vs. control register vs. I/O
  - Came to be used to modify a data access, e.g.
    - set endianness of the access
    - set effective privilege level of access (AS_IF_USER, AS_IF_PRIV)
  - Also, was later (mis-)used as extra opcode bits (starting in 1995)
    - e.g. short float loads, partial stores, block loads/stores

Reference: UltraSPARC Architecture spec, Chapter 10, “ASIs”

SPARC Assembly Language Hints

- *Always* reads from left to right,
  so leftmost operand(s) are source operand(s)
  and rightmost operand is the destination (if any)

- register names begin with “%”

- Indirect (e.g. memory) references are enclosed in square brackets
  - %r1 = contents of register o1
  - [%r1+8] = contents of the *memory location* at address %r1+8

- Windowed registers: **i** = in, **o** = out, **l** = local, **g** = global

Delayed Control Transfer Instr'ns (dCTIs)

- *Two* program counters – PC and nPC (*next* PC)
- At end of each normal instruction (incl. non-taken dCTI):
  - PC ⟵ nPC *and* nPC ⟵ nPC+4
- Taken delayed control-transfer instruction (dCTI), such as CALL, JMPL, or taken branch, causes:
  - PC ⟵ nPC *and* nPC ⟵ **target address of CTI**
- Actual control transfer ia “delayed” by one instruction (...hence the term **d**CTI)
- Instruction appearing in word physically after a dCTI (that is, at PC+4) is called a “delay slot” instruction
- variants exist (e.g., annulling branch)

DCTI Example #1

- ***Un***taken branch (including BN [branch-never])
- Execution sequence:
  - > *integer condition code Z is **off** (prev op result was non-zero)*
  - > 0040: ① bz label1 *! not taken*
  - > 0044: ② mov 0, %i5
  - > 0048: ③ mov 1, %i5
  - > 004A: ④ ld [%o3+8],%o7
  - > .....
  - > label1:
  - > 0100: add %i4, %i5, %i6
  - > 0102: cmp %i6,0
  - > .....

DCTI Example #2

- *Taken* branch (or CALL or JMPL, incl BA [branch-always])
- Execution sequence:
  - > *integer condition code Z is **on** (prev op result was zero)*
  - > 0040: ① bz label1 *! taken*
  - > 0044: ② mov 0, %i5
  - > 0048: mov 1, %i5
  - > 004A: ld [%o3+8],%o7
  - > .....
  - > label1:
  - > 0100: ③ add %i4, %i5, %i6
  - > 0102: ④ cmp %i6,0
  - > .....

Windowed Register File (1 of 3)

- ...a.k.a. “register windows”
- given:
  - 5-bit register specifier (only access 32 int registers at a time)
  - push/pop or save/restore args from memory is expensive
- most efficient means to pass function args = in registers
- ...but compilers not too sophisticated

- one solution: ...windowed registers

Windowed Register File

- ...as viewed by Software

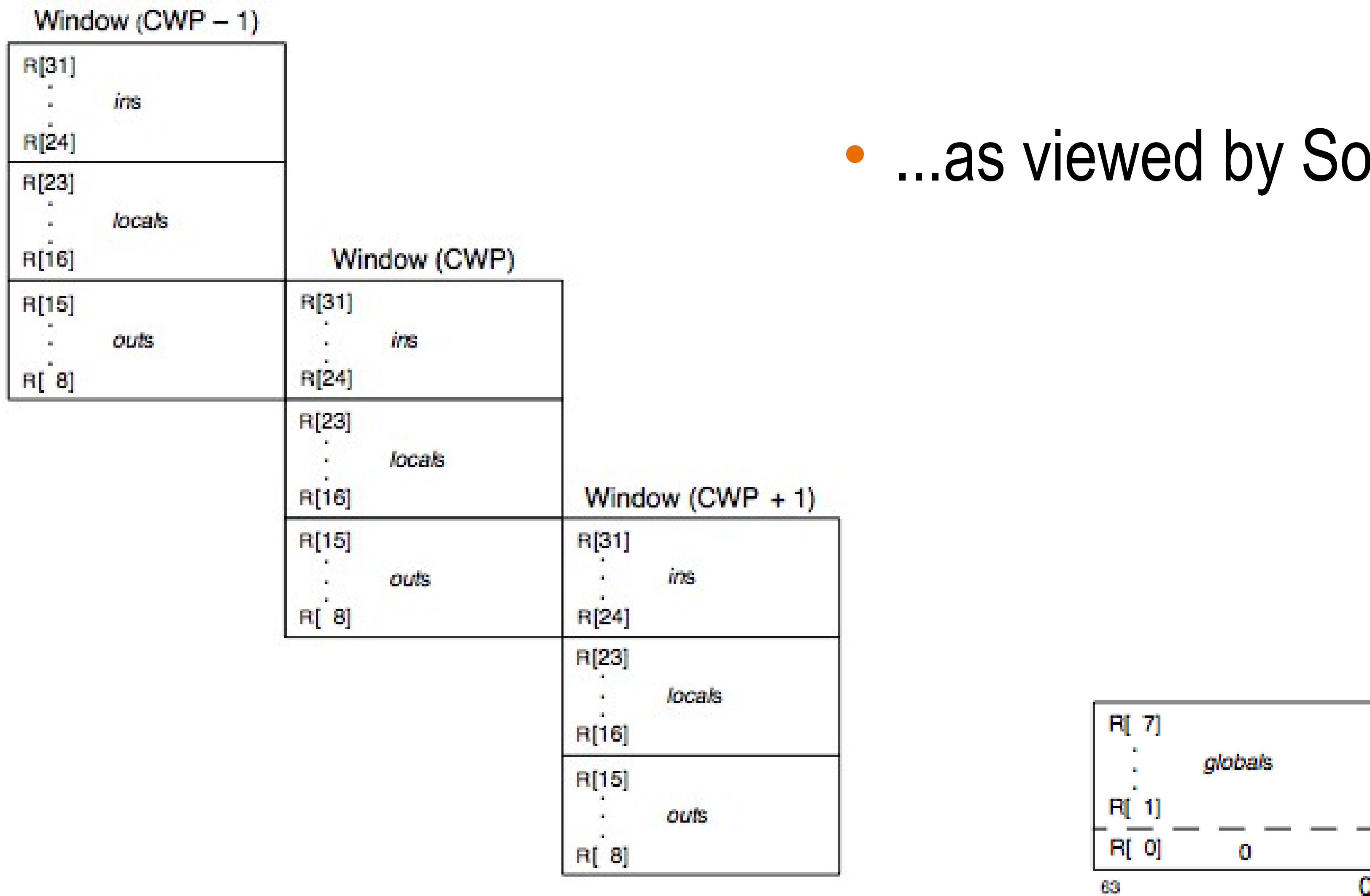
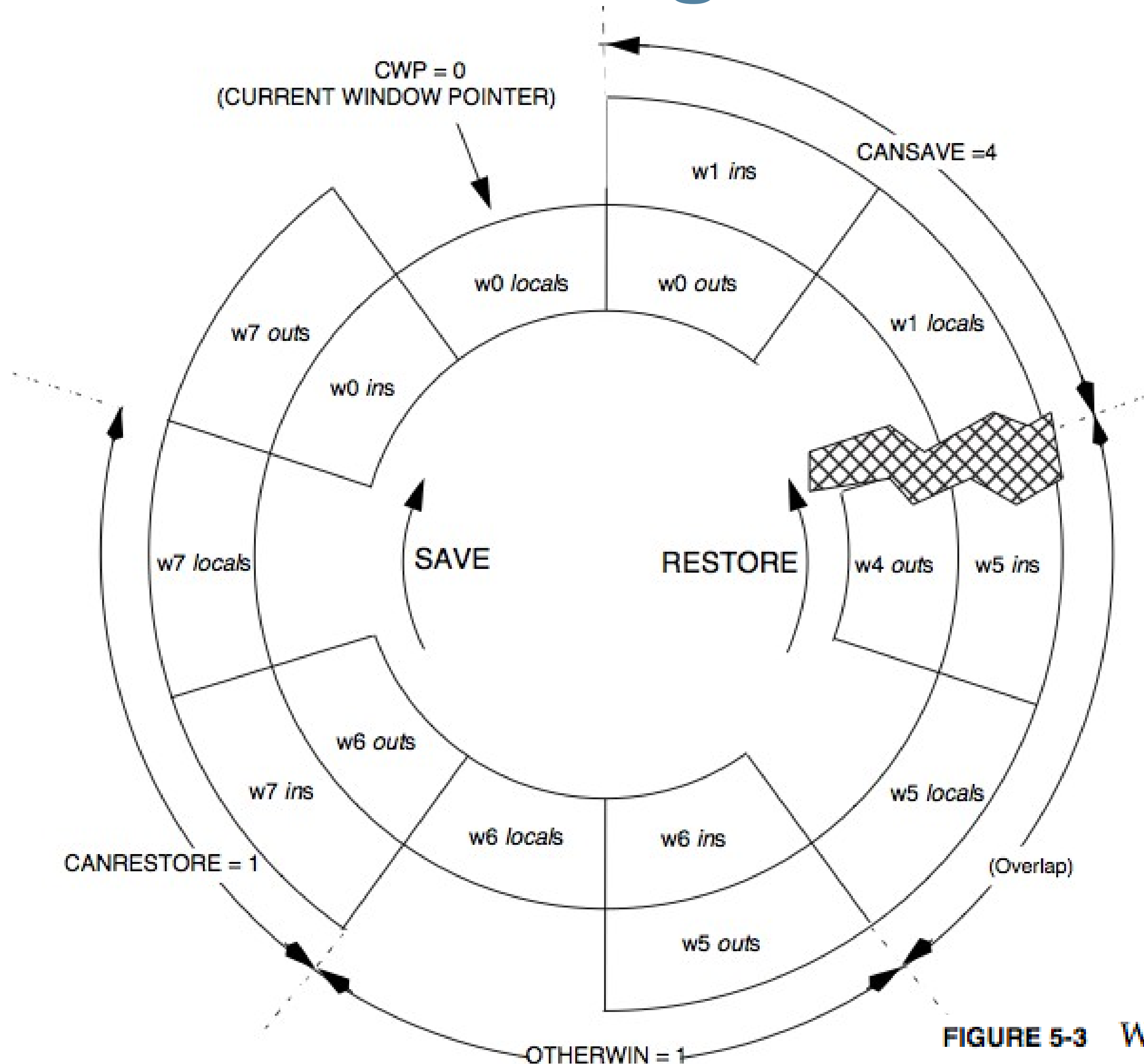


FIGURE 5-2 Three Overlapping Windows and Eight Global Registers

Windowed Register File



- ...as implemented in Hardware

FIGURE 5-3 Windowed R Registers for *N_REG_WINDOWS* = 8

SPARC V7 Implementations

- Original 16MHz Fujitsu gate-array implementation, 1987

  ...which later became SPARC V8 compliant with OS upgrade to emulate new instructions

SPARC V8 (~1988)

- added hardware integer Multiply and Divide instructions
- changed 80-bit extended precision to 128-bit Quad precision floating-point *using same opcodes*
  - change not an issue because extended-precision was never implemented in hardware, it was only trap-and-emulate (slow!), and/so had never been used by customers
- specification improved; published for first time in book form (SPARC V8 Architecture Manual)
- later (1994) was basis for IEEE Std. 1754, "A 32-bit Microprocessor Architecture"

SPARC V8 Implementations

- Many implementations, including:
  - Sun/Cypress 40 Mhz CMOS, ~1989
  - Sun microSPARC
  - Sun BiCMOS SuperSPARC (3-way superscalar), ~1992
  - Solbourne Computer SPARC
  - Weitek SPARC
  - Ross HyperSPARC

Sun microsystems

SPARC V9 (~1993)

- (still) 32-bit instructions
- **64**-bit data and **64**-bit virtual addresses
- Integer ops generate both *32*-bit (icc) and *64*-bit (xcc) condition codes
- 4 sets of floating-point condition codes (fcc0, fcc1, fcc2, fcc3)
  - > allowed better instruction scheduling, loop unrolling optimizations
- Branches & Moves based on register contents
- Static branch-prediction bits (no longer useful)
- Software-initiated data prefetch instructions
- *Completely* binary-compatible for SPARC V8 software (w/ appropriate OS & library support)

SPARC V9 Implementations

- Fujitsu SPARC64 I, 1995 – added FMA instructions
- Sun UltraSPARC I, 1995 – added VIS-1 instructions
- Sun UltraSPARC II, ~1997 – first implemented PREFETCH
- Fujitsu SPARC64 II and III (no IV)
- Sun UltraSPARC III, ~2000 – added VIS-2 instructions
- Fujitsu SPARC64 V, ~2000
- Sun UltraSPARC IV, ~2004 – added basic CMT
- Fujitsu SPARC64 VI
- Fujitsu SPARC64 VII
- Fujitsu SPARC64 VIIIfx (supercomputer)

Recent SPARC Generations (Families)

- **UltraSPARC Architecture 2005** (Sun, 2005):
  partial VIS1 + VIS2, full CMT, hyperprivileged mode
  - UltraSPARC T1 (Niagara), 2005 →OpenSPARC T1
- **UltraSPARC Architecture 2007** (Sun, 2007):
  minor architectural clean-up, added full VIS1 + VIS2
  - UltraSPARC T2 (Niagara-2), 2007 →OpenSPARC T2
  - UltraSPARC T2+ (Victoria Falls), 2008
  - (more to come)
- **UltraSPARC Architecture 2009** (Sun, 2009): *VIS3, HPC*
  - UltraSPARC __ (Rock)
  - (more to come)

Specification Differences, V9 → UA 2005

- Formatting improvements
- UA specs are more complete and more precise than SPARC V9 spec; for example:
  - include all known V9 errata
  - list the specific conditions under which each exception may be raised, for every instruction
  - clarify relative trap priorities
  - close many old implementation dependencies
  - specify Sun extensions to architecture
- Document Design:
  - Architecture Spec + Implementation Supplements structure

Architecture Extensions, V9 → UA 2005

- Sun's VIS1 and VIS2 instructions, incl. GSR register
- GL register (replacing old ag/ig/mg global regs)
- Privileged register-window management instructions ALLCLEAN, OTHERW, NORMALW, and INVALW
- "Deferred" traps split into two categories
  - SPARC V9 deferred traps are now "resumable deferred" traps

Architecture Changes, V9 → UA 2005

- **Hyperprivileged** mode has been added, including:
  - > several hyperprivileged registers
  - > a few hyperprivileged instructions
    - > notably RDHPR and WRHPR (hyperprivileged register access)
  - > effects on the Tcc instruction
  - > effects on the trap model
  - > SIR instruction is now hyperprivileged
  - > VER register is now the hyperprivileged (**H**VER)
  - > full control of Chip MultiThreading (CMT) features

Architecture Changes, Earlier UltraSPARCs → UA 2005

- For Block Store instructions, an intermediate "zero" state is allowed to be observed during execution

Feature Classification in UA 2005

- Architectural features are now classified and tagged
  - Software Class (letter)
  - Implementation Class (digit)
  - allows smooth long-term architectural evolution (addition and deprecation of features)

Why Hyperprivileged Mode?

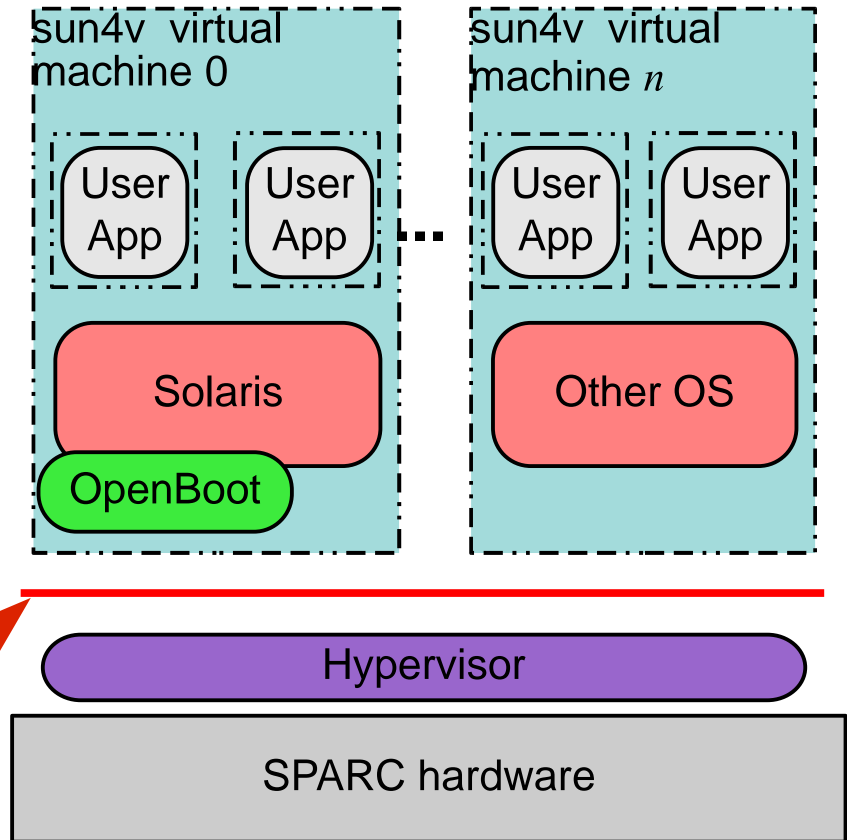
- Allows running multiple simultaneous guest OSs
  - (and/or multiple versions of the same OS)
- Allows running older OS (that uses hypervisor API) on newer hardware, without need to port the OS
- Simplifies OS ports (Linux in 2 months!)
- Allows implementation of logical domains
- Allows *virtualization*

Why Virtualization?

- Insulates higher levels of software from underlying hardware, by adding another software abstraction layer
  - > Protects customers' investment in application software from changes in underlying software (OS)
  - > Buying new, faster HW no longer requires running a new version of the OS
- Allows ability to "oversubscribe" resources (run multiple top-level software)

Virtualization

- Thin software layer between OS and platform hardware
- Para-virtualized OS
- Hypervisor + sun4v interface
  - Virtualizes machine HW and isolates OS from register-level
  - Delivered with *platform*, not with OS
  - Not itself an OS



stable interface "**sun4v**"

CMT Control in UltraSPARC Architecture

Sun microsystems

Terms

- **Thread**
  - software term, overused, ambiguous
- **Strand** or **Virtual Processor**
  - state HW must maintain to execute a software thread
  - addressable unit for interrupts
- **Pipeline** or **Microcore**
  - execution pipeline; may be used by one or more strands
- **Core** or **Physical Core**
  - Pipeline(s) plus other HW, e.g. caches, needed to execute one or more threads
- **Processor**
  - physical module (1+ cores) that plugs into a system

Terms

(2 of 2)

- **Strand Available** vs. **not-Available**:
  - > VP is functional, can be used if Enabled & Running
  - > flawed Strands are set to not-Available at factory, allowing use of partial-good die (e.g. 4-core or 6-core T1/T2 processors)
- Strand **Enabled** vs. **Disabled**:
  - > software can be executed on strand if set to Running
  - > can be used to disable a strand (or core) with uncorrectable HW errors at runtime
- Strand **Parked** vs. **Running**:
  - > Park = temporary suspension or “pause” in execution
  - > Park synonyms: **Suspended**, **not-Running**

Virtual Processor Control

- STRAND_AVAILABLE register
  - 64-bit mask of functional VPs (virtual processors / strands)
- STRAND_ID register
  - HW-assigned strand ID #
  - max strand ID# in STRAND_AVAILABLE
  - max # strands per core
- STRAND_INTR_ID register
  - SW-assignable 16-bit interrupt ID #

Virtual Processor Control

- Enabling/Disabling virtual processors
  - a *heavyweight* operation
  - expect all state to be lost when disabled
  - virtual processor “dead to the world”
    - no effect on other VPs
    - does not participate in coherency
- STRAND_ENABLE_STATUS register
  - each bit indicates if corresponding VP is Enabled
- STRAND_ENABLE register
  - enables/disables VPs
  - typically takes effect at next Warm Reset

Virtual Processor Control (3 of 4)

- Parking/UnParking virtual processors
  - a light*weight* operation
  - *temporary* suspension of execution
  - no state lost when unparked (RUNNING)
  - virtual processor “in a light sleep”
    - still participates in coherency!
    - interrupt to parked VP – as if VP was too busy to service it
- STRAND_RUNNING_STATUS register
  - each bit indicates if corresponding VP is Running (vs. Parked)
- STRAND_RUNNING register
  - parks/unparks VPs
  - takes effect after some latency ... no reset involved

Virtual Processor Control (4 of 4)

- XIR_STEERING register
  - mask indicating to which virtual processors an XIR (Externally-Initiated Reset) should be forwarded
- ERROR_STEERING register
  - encoded Strand ID #, to which errors should be sent for processing

CMT Reference

- UltraSPARC Architecture spec,
  Chapter 15, **C**hip-level **M**ulti**T**hreading

UltraSPARC Architecture Reference Materials

Sun microsystems

Reference Materials

- SPARC V9 Manual:
  - PDF: http:// www.sparc.com / specificationsDocuments.html
  - Hardcopy: http:// www.Amazon.com / dp / 0130992275
  - Errata list (88): http:// www.sparc.com / standards / v9-errata.html
- UltraSPARC Architecture (2005, 2007, 2009) *and* Implementation Supplement specfications (Sun-Internal versions):
  - http:// systemsweb.sfbay.sun.com / archperf / SPARC
- OpenSPARC versions of UA docs & Impl'n Supplements:
  - T1: http:// www.opensparc.net / opensparc-t1 /
  - T2: http:// www.opensparc.net / opensparc-t2 /

Introduction to UltraSPARC Architecture

June 2009

David Weaver
Principal Engineer, UltraSPARC Architecture
Principal Evangelist, OpenSPARC
Architecture Group
Sun Microelectronics