SPARC64™ VIII fx: Fujitsu’s New Generation Octo Core Processor for PETA Scale computing

August 25, 2009
Takumi Maruyama
LSI Development Division
Next Generation Technical Computing Unit
Fujitsu Limited
SPARC64™ VIIIfx Design Target

- Processor for Fujitsu’s supercomputer for the peta-scale computing age, which realizes **both high performance and low power**
  - Unachievable goal with conventional processor design
    - More GF (Giga Flops)
    - More Efficiency
    - No more high frequency
  - Large ISA (Instruction Set Architecture) extension is required
- High Integration: SoC (System On Chip)
- High Reliability
- Reuse SPARC64™ VII design when applicable
HPC-ACE
(High Performance Computing - Arithmetic Computational Extensions)

◆ ISA (Instruction Set Architecture) of SPARC64VIIIfx is:
  ■ Complies with
    • The SPARC-V9 standard
    • JPS (Joint Programmer’s Specification): Extension to SPARC-V9
  ■ HPC-ACE: Fujitsu’s unique ISA extension for HPC
    • Large register sets
    • SIMD (single instruction multiple data) instructions
    • etc

◆ You can download SPARC64™ VIIIfx ISA documents from
  http://jp.fujitsu.com/solutions/hpc/brochures/
  ■ The SPARC® Architecture Manual Version 9
  ■ SPARC® Joint Programming Specification (JPS1): Commonality
  ■ SPARC64 VIIIfx Extensions
Large register sets 1/2

Register enhancements from V9
- 160→192 INT registers
- 32→256 FP registers (DP)
  - Lower 32 registers are the same with SPARC-V9’s.
  - FP registers are “flat”. Extended FP registers can be accessed with non-SIMD instruction as well.

Why needed
- To extract more parallelism currently limited by the number of Arch registers.
- Reduce spill/fill overhead
Large register sets 2/2

Instruction format for 256 FP registers

- 8 bit x 4 (3 read +1 write) register number fields are necessary for FMA (Floating-point Multiply and Add) instruction.
- But SPARC-V9 instruction length is limited (32bits – fixed)

- Defined a new prefix instruction (SXAR) to specify upper-3bit of register numbers of the following two instructions.

SXAR (Set XAR) instruction

- XAR: Extended Arithmetic Register
  - Set by the SXAR instruction
  - Valid bit is cleared once the corresponding subsequent instruction gets executed.

- SXAR1: set XAR for subsequent one instruction.
- SXAR2: set XAR for subsequent two instructions.
SIMD

- SIMD FP operation
  - 1 SIMD FP instruction executes two SP or DP floating-point operations

- SIMD Load/Store
  - 1 SIMD load/store instruction accesses two contiguous SP or DP data in memory
  - Data alignment requirement for SIMD-load (DP) is 8 byte rather than usual 16 byte.
  - Combine two independent FP operation into one with the Flat FP registers

- SXAR instruction
  - Specifies SIMD operation of the subsequent two instructions.
  - No new opcode is required for SIMD operation

-> More SW optimization possibility with SIMD
FP Trigonometric Functions

◆ Instructions for fast Trigonometric Function calculation

◆ Taylor series approximation of \( \sin(x) \)

\[
\approx x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \frac{1}{9!}x^9 - \frac{1}{11!}x^{11} + \frac{1}{13!}x^{13} - \frac{1}{15!}x^{15}
\]

\[
= x ((((((((0 - \frac{1}{15!})x^2 + \frac{1}{13!})x^2 - \frac{1}{11!})x^2 + \frac{1}{9!})x^2 - \frac{1}{7!})x^2 + \frac{1}{5!})x^2 - \frac{1}{3!})x^2 + 1)
\]

◆ New instruction - ftrimadd

- Function: \( rs1 \times \text{abs}(rs2) + \text{T[index]} \rightarrow rd \)

- Usage:
  - \# S = 0
  - ftrimadd X, 7, S  \# S * X^2 - 1/15! \rightarrow S
  - ftrimadd X, 6, S  \# S * X^2 + 1/13! \rightarrow S
  - ftrimadd X, 5, S  \# S * X^2 - 1/11! \rightarrow S
  - ftrimadd X, 4, S  \# S * X^2 + 1/9! \rightarrow S
  - ftrimadd X, 3, S  \# S * X^2 - 1/7! \rightarrow S
  - ftrimadd X, 2, S  \# S * X^2 + 1/5! \rightarrow S
  - ftrimadd X, 1, S  \# S * X^2 - 1/3! \rightarrow S
  - ftrimadd X, 0, S  \# S * X^2 + 1/1! \rightarrow S
  - Fmuld X, S  \# S * X \rightarrow S
Software controlled Cache

- Give SW ability to control cache to optimize performance while keeping cache coherency
- Divide Cache into 2 groups (sectors)
  - SXAR instruction specifies the sector
    - sector 0: Instruction fetch / normal operand access (default)
    - sector 1: operand access explicitly specified by SXAR
  - Sector cache configuration register
    Specifies the ratio of sector 0 and 1 at the same index

- HW Implementation
  - Keep sector info of each cache line.
  - Select the way to be replaced to meet the sector ratio on cache misses
  - SW specifies a sector depending on temporal and spatial locality of data

L2 Cache structure

Ex) Sector0:Sector1=30%:70%

Way0
Way9
Other HPC-ACE features

◆ Conditional operation
  ■ For efficient execution of Loop with ‘if’
  ■ Conditional branch can be removed with the following conditional instructions.
    ● FP Conditional Compare to Register
    ● Move Selected FP-register on FP-register’s condition
    ● Store FP-register on FP-register’s condition
  ➞ Compiler can optimize the loop with SW pipeline

◆ FP Reciprocal Approximation of Divide/Square-root
  ■ Calculate reciprocal approximation with rounding error < 1/256
  ■ To achieve higher divide/square-root performance with pipelined operation

◆ FP Minimum and Maximum

```c
for (i=0; i<00; i++) {
  if (A(i)>0)
    FPop X(i)
  else
    FPop Y(i)
}
```

Usage:

- `%f2 / %f10 -> %f0`
- `frcpad %f10, %f6`
- `fmuld %f2, %f6, %f2`
- `fnmsubd %f6, %f10, 1.0, %f6`
- `fmuld %f6, %f6, %f0`
- `fmadd %f6, %f6, %f6, %f4`
- `fmaddc %f0, %f0, %f6, %f0`
- `fmadd %f4, %f2, %f2, %f4`
- `fmaddc %f0, %f4, %f2, %f0`
Integrated Multicore Parallel Architecture

- **SPARC64™ VIIIfx HW**
  - HPC-ACE
  - Shared L2 cache to avoid false sharing
  - Hardware barrier for fast inter-core synchronization

- **Fujitsu’s compiler technology**
  - Automatic parallelization

**Vector**

```fortran
DO J=1,N
  DO I=1,M
    A(J)=A(J)+A(I,J+1)*B(I,J)
  END
END
```

**Conventional Scalar**

```fortran
DO J=1,N
  DO I=1,M
    A(J)=A(J)+A(I,J+1)*B(I,J)
  END
END
```

More possibility of optimization:
Parallelize the innermost loop

- **SPARC64™ VIIIfx**

  - HPC-ACE
  - Shared L2 cache to avoid false sharing
  - Hardware barrier for fast inter-core synchronization

- **Fujitsu’s compiler technology**
  - Automatic parallelization

- **Vector**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **Conventional Scalar**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **More possibility of optimization:**
  Parallelize the innermost loop

- **SPARC64™ VIIIfx**

  - HPC-ACE
  - Shared L2 cache to avoid false sharing
  - Hardware barrier for fast inter-core synchronization

- **Fujitsu’s compiler technology**
  - Automatic parallelization

- **Vector**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **Conventional Scalar**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **More possibility of optimization:**
  Parallelize the innermost loop

- **SPARC64™ VIIIfx**

  - HPC-ACE
  - Shared L2 cache to avoid false sharing
  - Hardware barrier for fast inter-core synchronization

- **Fujitsu’s compiler technology**
  - Automatic parallelization

- **Vector**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **Conventional Scalar**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **More possibility of optimization:**
  Parallelize the innermost loop

- **SPARC64™ VIIIfx**

  - HPC-ACE
  - Shared L2 cache to avoid false sharing
  - Hardware barrier for fast inter-core synchronization

- **Fujitsu’s compiler technology**
  - Automatic parallelization

- **Vector**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **Conventional Scalar**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **More possibility of optimization:**
  Parallelize the innermost loop

- **SPARC64™ VIIIfx**

  - HPC-ACE
  - Shared L2 cache to avoid false sharing
  - Hardware barrier for fast inter-core synchronization

- **Fujitsu’s compiler technology**
  - Automatic parallelization

- **Vector**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **Conventional Scalar**

  ```fortran
  DO J=1,N
    DO I=1,M
      A(J)=A(J)+A(I,J+1)*B(I,J)
    END
  END
  ```

- **More possibility of optimization:**
  Parallelize the innermost loop
SPARC64™ VIII fx Chip Overview

- **Architecture Features**
  - 8 cores
  - Shared 5 MB L2$
  - Embedded Memory Controller
  - 2 GHz

- **Fujitsu 45nm CMOS**
  - 22.7mm x 22.6mm
  - 760M transistors
  - 1271 signal pins

- **Performance (peak)**
  - 128GFlops
  - 64GB/s memory throughput

- **Power**
  - 58W (TYP, 30°C)
  - Water Cooling – Low leakage power and High reliability
### SPARC64™ VIIIfx Core spec

<table>
<thead>
<tr>
<th>Execution Unit</th>
<th>Instruction control</th>
<th>L1D$</th>
</tr>
</thead>
<tbody>
<tr>
<td>FPR + FUB</td>
<td>L1$ control</td>
<td></td>
</tr>
</tbody>
</table>

### Instruction Set

- **Architecture**: SPARC-V9/JPS + HPC-ACE
- **#FP-operations per clock**: 8 (= 4 Floating Multiply and Add)
- **Execution units**:
  - **<integer>**
    - ALU x2
    - SHIFT x2
    - MULT x1
    - DIVIDE x1
    - AGEN x2
  - **<floating-point>**
    - FMA x4 (2SIMD)
    - COMPARE x2
    - DIVIDE x2
    - VIS x1
- **#Registers**
  - 188 (GPR)
  - 32 (GUB)
  - 256 (FPR)
  - 48 x2 (FUB)
- **L1$**
  - L1I$ 32KB/2way
  - L1D$ 32KB/2way
SPARC64™ VIIIfx Pipeline

Fetch (4stages) → Issue (3stages) → Dispatch (4(int)/5(fp) stages) → Reg.-Read Execute (L1$: 3(int)/4(fp)stages) → Memory (2stages) → Commit

- **L1 I$ 32KB 2Way**
- **L1 D$ 32KB 2Way**
- **CSE 48Entry**
- **Fetch Port 20Entry**
- **Store Port 8Entry**
- **Write Buffer 5Entry**
- **L2$ 5MB 10Way**
- **Memory Controller**
- **DIMM**
- **PC**
- **Control Registers**
- **Fetch (4stages)**
- **Issue (3stages)**
- **Dispatch (4(int)/5(fp) stages)**
- **Reg.-Read Execute (L1$: 3(int)/4(fp)stages)**
- **Commit (2stages)**

**SPARC64™ VIIIfx Pipeline Diagram**
Changes in the pipeline

◆ Issue stage
  ■ Added “PD” stage to handle SXAR
  ■ 6-instructions packed into 4 at “PD”
  ■ An CSE entry is assigned at “D”
  ■ The packed instructions have
    ● Extended register fields
    ● SIMD operation attributes
  ■ 4 packed (=6 unpacked) instructions can be committed at the same time.

■ Execution Stage
  ■ “B” stage has been split into two for flat Floating-point Register access
SPARC64™ VIIIfx realizes much higher core performance than SPARC64™ VII thanks to HPC-ACE despite its lower frequency.

Benchmark: Euroben + in-house

Performance Results – Core
SPARC64™ VIIIfx@2.0GHz

Hardware measured results

- $/system
- Execution unit
- 1 commit
- 2-3 commit
- 4 commit

Performance:
- Loop6: $y(i) = x_1(i) / x_2(i)$
  - Performance = $x 4.3$

- Loop14: 9th Degr Polyn
  - Performance = $x 6.8$

- Loop14: 9th Degr Polyn
  - Performance = $x 1.4$
SPARC64™ VIIIfx shows about x 2.5 performance of SPARC64™ VII

- Stall time due to the execution unit busy has been reduced dramatically.

→ Expect x 3 performance with further compiler optimization
SPARC64™ History and Future

Evolution rather than revolution

- **SPARC64™ V (1 core)**
  - RAS
  - Single thread performance

- **SPARC64™ VI (2 core x 2 VMT)**
  - Throughput

- **SPARC64™ VII (4 core x 2 SMT)**
  - More Throughput
  - High Performance Computing

- **SPARC64™ VIIIfx (8 core)**
  - High Performance Computing
  - Low Power
  - SoC

**Peak Performance & Power**

- Performance (GF) = x 3
- Power (W) = x 0.5
SPARC64™ VIIIfx Summary

◆ SPARC64™ VIIIfx has been designed to be used for Fujitsu’s supercomputer for the PETA-scale computing age.

◆ HPC-ACE instruction sets overcomes the limitation of conventional SPARC-V9 architecture.

◆ SPARC64™ VIIIfx has combined high performance and low power.

◆ SPARC64™ VIIIfx chip is up and running in the lab.

◆ Fujitsu will continue to develop SPARC64™ series to meet the needs of a new era.
Abbreviations

• **SPARC64™ VIIIfx**
  - **IB**: Instruction Buffer
  - **RSA**: Reservation Station for Address generation
  - **RSE**: Reservation Station for Execution
  - **RSF**: Reservation Station for Floating-point
  - **RSBR**: Reservation Station for Branch
  - **GUB**: General Update Buffer
  - **FUB**: Floating point Update Buffer
  - **GPR**: General Purpose Register
  - **FPR**: Floating Point Register
  - **CSE**: Commit Stack Entry