SH4 RISC Microprocessor for Multimedia

Fumio Arakawa, Osamu Nishii, Kunio Uchiyama, Norio Nakagawa
Hitachi, Ltd.

Outline

1. SH4 Overview
2. New Floating-point Architecture
3. Length-4 Vector Instructions
4. 3DCG Performance
5. Double Precision Support
6. Conclusions

3DCG: Three Dimension Computer Graphics
SH4 Overview

- Hitachi’s SuperH Series Family
- For Consumer Multimedia Systems
  Home Video Game, Handheld PC
- Excellent Performance with Consumer Price
  300 VAX MIPS
- Excellent 3DCG-Performance with Consumer Price
  5.0 M Polygons/sec*
- IEEE 754 Standard Floating-point Architecture
  Double-precision with Hardware Emulations

* measured with an original simple geometry benchmark

SH4 Specifications

<table>
<thead>
<tr>
<th>Technology</th>
<th>0.25 µm CMOS, 5 Layer Metals</th>
</tr>
</thead>
<tbody>
<tr>
<td>Voltage</td>
<td>1.8 V (I/O: 3.3 V)</td>
</tr>
<tr>
<td>Frequency</td>
<td>167 MHz (internal) / 83,55, etc. MHz (I/O)</td>
</tr>
<tr>
<td>Performance</td>
<td>300 MIPS (Dhrystone), 1.17 GFLOPS (peak)</td>
</tr>
<tr>
<td>Cache</td>
<td>8/16 KB (Inst./Data) Direct-mapped</td>
</tr>
<tr>
<td>TLB</td>
<td>4/64-entry (Inst./Unified) Fully-associative</td>
</tr>
<tr>
<td>Interfaces</td>
<td>SRAM, DRAM, SDRAM, burst ROM, PCMCIA</td>
</tr>
<tr>
<td>Peripherals</td>
<td>DMAC, SCI, RTC, Timer</td>
</tr>
</tbody>
</table>

HITACHI
Pipeline Stages

- Simple Five-stage Pipelines
- Two-way Superscalar

<table>
<thead>
<tr>
<th>Pipeline Stages</th>
<th>Inst. Fetch</th>
<th>Integer</th>
<th>Floating Point</th>
<th>Load / Store</th>
<th>Branch</th>
</tr>
</thead>
</table>

Superscalar Issue Combinations

<table>
<thead>
<tr>
<th></th>
<th>INT</th>
<th>FP</th>
<th>LS</th>
<th>BR</th>
<th>BO</th>
<th>NS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integer (INT)</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Floating Point (FP)</td>
<td>O</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Load / Store (LS)</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Branch (BR)</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Both INT &amp; LS (BO)</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
</tr>
<tr>
<td>Not Superscalar (NS)</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

INT: Add, Subtract, Shift, etc.
FP: Floating-point Add, Subtract, Multiply, Divide, etc.
LS: Load/Store/Transfer from/to Integer/Floating-point Register, etc.
BR: Branch Always/Conditionally, etc.
BO: Move between Integer Registers, Integer Compare, etc.
NS: Load to Control Register, etc.
Floating-point Arch. Enhancement

- Two Sets of 16 Single Precision Registers
  - The extra set fits 4 by 4 matrix storage
- Length-4 Vector Instructions
  - Inner Product
  - Transform Vector
- Register Pair Load/Store/Transfer Instructions
  - Enough bandwidth for vector operations
- Double Precision Format Mode

Floating-point Instructions

- Common
  - FADD (add)
  - FSUB (subtract)
  - FMUL (multiply)
  - FDIV (divide)
  - FSQRT (Square Root)
  - FCMP (Compare)
  - FNEG (Negate)
  - FABS (Absolute Value)
  - FLOAT (Convert Integer to float)
  - FTRC (Convert float to Integer)
  - FMOV (Move from/to Register)

- Single Precision Mode Only
  - FMAC (multiply-Accumulate)
  - FIPR (Inner Product)
  - FTRV (Transform Vector)

- Double Precision Mode Only
  - FCNVDS (Convert Double to Single)
  - FCNVSD (Convert Single to Double)
**Inner Product Instruction**

- 16 Registers = 4 (Length-4) Vector Registers
  
  \[
  \begin{align*}
  f_{v0} &= (f_{r0}, f_{r1}, f_{r2}, f_{r3}) \\
  f_{v4} &= (f_{r4}, f_{r5}, f_{r6}, f_{r7}) \\
  f_{v8} &= (f_{r8}, f_{r9}, f_{r10}, f_{r11}) \\
  f_{v12} &= (f_{r12}, f_{r13}, f_{r14}, f_{r15})
  \end{align*}
  \]

- Operation
  
  \[
  f_{rn'} = (f_{vm}, f_{vn})
  \]

  \[
  m, n: 0, 4, 8, 12 \quad n' = n + 3
  \]

- 1 cycle Pitch

- 4 cycle Latency

- Vector Normalization, Intensity Calculation, Surface Judgment

---

**Inner Product Hardware (Mantissa)**

- Calculating \( f_{r7} = (f_{v0}, f_{v4}) \)

---

**Floating-point Register File**

- \( f_{r0} \), \( f_{r4} \), \( f_{r1} \), \( f_{r5} \), \( f_{r2} \), \( f_{r6} \), \( f_{r3} \), \( f_{r7} \)

**Multiplier**

- \( f_{r0} \times f_{r4} \)
- \( f_{r1} \times f_{r5} \)
- \( f_{r2} \times f_{r6} \)
- \( f_{r3} \times f_{r7} \)

**Alignment Shifter**

**4-input Adder**

\[
fr0 \times fr4 + fr1 \times fr5 + fr2 \times fr6 + fr3 \times fr7
\]

**4-to-2 Compressor & 2-input Adder**

**Normalizer**

**Rounnder**
Inner Product Hardware (Exponent)

- Calculating \( fr7 = (fv0, fv4) \)

**Floating-point Register File**

- \( fr0 \) + \( fr4 \)
- \( fr1 \) + \( fr5 \)
- \( fr2 \) + \( fr6 \)
- \( fr3 \) + \( fr7 \)

**Adder**

- Maximum
- Exponent
- Selector

**Subtractor**

- Alignment
- Shift Count

**Normalizer-Rounder**

**Inner Product Accuracy**

- Inner Product Inst. is an Approximate Inst.
- No Accurate Intermediate Value
  (the width is too wide to implement)
- More Accurate than the Worst Order Multiply and Add Inst. Combinations

- Maximum Error:
  \((\text{Maximum Product } \times 2^{-25}) + (\text{Result } \times 2^{-23})\)
- If source operands are rounded values, this is enough accuracy.
- For example: 4 Products are \( 2^{26}, -2^{26}, 1, 0 \).
  Accurate Result = 1. Inner Product Inst. Result = 0.
**Inner Product v.s. SIMD Multiply-Add**

<table>
<thead>
<tr>
<th></th>
<th>Inner Product</th>
<th>4 Multiply-Add</th>
</tr>
</thead>
<tbody>
<tr>
<td>Peak Performance</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Latency</td>
<td>4</td>
<td>12</td>
</tr>
<tr>
<td>Register Port (Read/Write)</td>
<td>8/1</td>
<td>12/4</td>
</tr>
<tr>
<td>Normalizer &amp; Rounder</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>Floating-point Hardware</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

1) Inner Product Inst. and 4 Multiply-Add achieve the same peak performance, which is one inner product per cycle.
2) SH4 takes 3 cycles for Multiply-Add. $4 \times 3 = 12$.
3) Eight more cycles must be filled with independent insts. for the peak performance with SIMD architecture.
4) Twice more hardware is necessary for SIMD architecture.

---

**Elastic Pipeline**

- Pipeline stages become 4 cycles for vector Inst.
- In-Order Issue, Out-of-Order Completion

<table>
<thead>
<tr>
<th>ID</th>
<th>RR</th>
<th>FX</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>RR</td>
<td>AG</td>
<td>MA</td>
<td>WB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Floating-point Pipeline (Vector)

Load/Store Pipeline

- Only Floating-point Non-Vector Arithmetic Inst. right after Vector Inst. is interlocked.

ID: Instruction Decode, RR: Register Read, FX,F1,F2,F3: Floating-point Execution, AG: Address Generation, MA: Memory Access, WB: Register Write Back
Transform Vector Instruction

- Extra 16 Registers = 4 by 4 Matrix

\[
\begin{pmatrix}
xf0 & xf4 & xf8 & xf12 \\
xf1 & xf5 & xf9 & xf13 \\
xf2 & xf6 & xf10 & xf14 \\
xf3 & xf7 & xf11 & xf15
\end{pmatrix}
\]

xf: extra floating-point register

- Operation

\[ fvn = \text{matrix} \cdot fvn \quad n: 0,4,8,12 \]

- 4 cycle Pitch, 7 cycle Latency
- Coordinate Transformation,
  Coordinate Transformation Matrix Generation
- No Work Registers

Why Transform Vector Instruction ?

- Transform Vector Operation = 4 Inner Product Insts. ?
  NO !!
- Modification for Transform Vector: 

\[
\begin{align*}
vx0 &= (xf0, xf4, xf8, xf12) \\
vx1 &= (xf1, xf5, xf9, xf13) \\
vx2 &= (xf2, xf6, xf10, xf14) \\
vx3 &= (xf3, xf7, xf11, xf15)
\end{align*}
\]

"fv0 = matrix \cdot fv0" is divided into 4 Inner Products: 

\[
\begin{align*}
fr8 &= (vx0,fv0) \\
fr9 &= (vx1,fv0) \\
fr10 &= (vx2,fv0) \\
fr11 &= (vx3,fv0)
\end{align*}
\]

- 4 More Work Registers
- Complicated and More Operands
- No Generality (Just for Transform Vector)
- Transform Vector Inst. is Better.
**Transform Vector Implementation**

\[fv0 = \text{Matrix} \cdot fv0\]

<table>
<thead>
<tr>
<th>ID</th>
<th>RR</th>
<th>FX</th>
<th>F1</th>
<th>F2</th>
<th>F3</th>
<th>WB</th>
</tr>
</thead>
<tbody>
<tr>
<td>RR</td>
<td>FX</td>
<td>F1</td>
<td>F2</td>
<td>F3</td>
<td>WB</td>
<td></td>
</tr>
</tbody>
</table>

- \(fr0 = (xv0, fv0)\)
- \(fr1 = (xv1, fv0)\)
- \(fr2 = (xv2, fv0)\)
- \(fr3 = (xv3, fv0)\)

- All reg. reads complete before first reg. write. No work regs. are necessary.

ID: Instruction Decode, RR: Register Read, WB: Register Write Back
FX, F1, F2, F3: Floating-point Execution

---

**Pair Load/Store/Transfer Mode**

- Normal Mode:
  - 4-bit Reg. Field represents 16 Regs. of one set.
  - Set specifier must be changed for another set access.

- Pair Mode:
  - 4-bit Reg. Field represents 16 Pair Regs. of all sets.
  - All Regs. can be accessed.

- Transform Vector Throughput: 1 vector / 4 cycles
- Load/Store Throughput: 2 vectors (4 pairs) / 4 cycles
  - Enough for Storing Previous Result Vector and Loading Next Vector during Transform Vector
Simple 3DCG Geometry Benchmark

(1) Coordinate Transformation
(2) Perspective Transformation
(3) Intensity Calculation

3DCG Geometry Performance

M polygons/sec

<table>
<thead>
<tr>
<th></th>
<th>without New Arch.</th>
<th>with New Arch.</th>
</tr>
</thead>
<tbody>
<tr>
<td>M</td>
<td>1.2 M</td>
<td>5.0 M</td>
</tr>
</tbody>
</table>
Double Precision Support

- Floating-point Libraries for WindowsCE
  - Double Precision
  - ANSI/IEEE 754 Standard
- Emulation with Single Precision Hardware
  - Best cost-performance way
  - Peak performance is 27.8 MFLOPS.
  - Software emulation is 20 times slower.
  - Double precision hardware is 6 times faster but 2.5 times more.
- New Double Precision Mode
  - Single's code becomes double's code.

Double Precision Hardware

- Mantissa Part
  - Add/Subtract/Multiply/Convert: Add Feedback Path

```
Addend (32b/64b) --|-----------------|-----------------|------------------|--|-----------------|-----------------|-----------------|
| Addend Aligner | Multiply-Add Block | | | | Normalizer Rounder |
| [Aligned Addend (73b)] | Pre-normalizing Result (73b) | | | | Result (32b) |
```

- Divide/Square Root: Extend from 24 to 53 bits
- Exponent Part: Extend from 8 to 11 bits
Double Precision Multiply
- Four multiply-adds generate product
- Sticky bit is generated from partial products
- required width: 106b --> 55b

```
Mantissa (53b)  Higher (21b)  Lower (32b)
```

```
+ Lower x Higher (53b)
+ Higher x Lower (53b)
+ Higher x Higher (42b)
= Product (53b + guard/round)

HOT Chips IX  in August, '97  SH4 RISC Microprocessor for Multimedia

Conclusions
- Excellent Performance with Consumer Price
  - 300 VAX MIPS
- Excellent 3DCG-Performance with Consumer Price
  - New Inner Product & Vector Transformation Insts.
  - 5.0 M Polygons/sec
  - 1.17 GFLOPS (peak with the new insts.)
- IEEE 754 Standard Floating-point Architecture
  - Double-precision with Hardware Emulations
  - 27.8 MFLOPS (peak)