Embedded Software Optimization for MP3 Decoder Implemented on RISC Core

Yingbiao Yao, Qingdong Yao, Peng Liu, Zhibin Xiao
Zhejiang University
Information Science & Electronic Engineering Department, P.R. China

IEEE Transaction on Consumer Electronics
November 2004
Abstract

- This paper proposes general software optimization techniques for embedded systems based on processors, which mainly include general optimization methods in high language and software & hardware co-optimization in assembly language. Then these techniques are applied to optimize our MP3 decoder, which is based on RISC32, a RISC core compatible with MIPS instruction set. The last optimization decoder requires 48 MIPS and 49Kbytes memory space to decode 128Kbps, 44.1KHz joint stereo MP3 in real time with CPI 1.15, and we have achieved performance increase of 46.7% and memory space decrease of 38.8% over the original decoding software.
Outline

- What’s the problem?
- Related work
- Introduction to RISC32
- Embedded software optimization techniques
- MP3 decoding software optimization
- Experiment result
- Conclusion
What’s the problem?

- Low cost with fast time to market is the top requirement in embedded system
  - Develop based on *microprocessor architecture* by software and hardware co-design
- How to make software run efficiently on selected processor had become a main problem
Many portable MP3 players adopt DSP core or dual core that are comprised by DSP and RISC.
Introduction to RISC32

- 6 pipeline stages
  - Instruction fetch (IF), instruction decoding (ID), execute (EX), data memory access (DM), data tag comparing (TC) and write back (WB)
- Compatible with MIPS-I ISA
- Separate 16Kbytes ICache and 16Kbytes DCache by Harvard architecture
- Direct mapping, write-through cache strategy
- 16Kbytes on-chip RAM
- 200 MHz clock frequency
- 1mW power dissipation per MHz
The embedded software optimization can be divided into two parts:

- **Algorithm optimization in high level language**
  - Tradeoff among computation load, the complexity of data conveying and the size of coefficients

- **Code optimization in assembly level**
  - Take the features of instruction set, micro-architecture and pipeline
Embedded software optimization techniques (2)

- System functions and constraint conditions analysis
- The hardware features analysis
- Software optimization
- Software and hardware co-validation
  - In C language
  - In assembler
  - Success
    - N
    - Y
      - Object software output

- Software module partition performance estimation
- Module algorithm optimization
- General optimization
  - Resource statistics
  - Memory optimization
  - Assembly program structure optimization
Embedded software optimization techniques (3)

- Phase 1
  - Analysis all kinds of constraint in the embedded system

- Phase 2
  - Analysis the feature of specific processor

- Phase 3
  - Software optimization in C and in assembly code
  - There have not ever existed a good complier for the embedded processor to meet the requirement suggested from phase 1
  - Embedded software must be close link to target processor

- Phase 4
  - Software and hardware co-validation to make certain to reach the goal
Software optimization in high level language

- Software module partition and performance estimation
  - Divide application software into small module
  - Run and get the profile information

- Module algorithm optimization
  - Modify algorithm to match with the used processor

- General optimization
  - Implement general optimization
Software optimization in assembly level

- Resource statistics
  - Memory sizes of objective code
  - Execution MIPS

- Memory size optimization

- Assembly program structure optimization
  - Take the features of instruction set, micro-architecture and pipeline into account
MP3 decode flow

1. Bitstream
   - Sync seek
     - Head info decoding
       - Error checking
         - Huffman Code Bits
         - Huffman Information
         - Scalefactor Information
   - Huffman Decoding
   - Requantization
   - Reordering
   - Joint stereo Decoding
   - IMDCT
   - Frequency Inversion
   - Alias Reduction
   - Subband Synthesis
   - PCM
Profiling of ISO reference decoder

- Employing **floating-point** computation

<table>
<thead>
<tr>
<th>Module</th>
<th>CPU Time(%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huffman decoding</td>
<td>4.1</td>
</tr>
<tr>
<td>Requantization</td>
<td>16.9</td>
</tr>
<tr>
<td>Stereo processing</td>
<td>1.2</td>
</tr>
<tr>
<td>IMDCT</td>
<td>17.9</td>
</tr>
<tr>
<td>Subband synthesis</td>
<td>58.2</td>
</tr>
<tr>
<td>Other</td>
<td>1.7</td>
</tr>
</tbody>
</table>
## Performance analysis

- **Integer computation-based**

### Performance of our integer decoder without optimization

<table>
<thead>
<tr>
<th>Module</th>
<th>MIPS ON ISA simulator</th>
<th>CPU Time (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Huffman decoding</td>
<td>4.1</td>
<td>6.1</td>
</tr>
<tr>
<td>Requantization</td>
<td>3.2</td>
<td>4.8</td>
</tr>
<tr>
<td>Stereo processing</td>
<td>1.5</td>
<td>2.2</td>
</tr>
<tr>
<td>IMDCT</td>
<td>19.2</td>
<td>28.7</td>
</tr>
<tr>
<td>Subband synthesis</td>
<td>33.4</td>
<td>50.0</td>
</tr>
<tr>
<td>other</td>
<td>5.4</td>
<td>8.1</td>
</tr>
<tr>
<td>total</td>
<td>66.8</td>
<td>100.0</td>
</tr>
</tbody>
</table>
Algorithm Optimization in C-code level
-- IMDCT

- Using the Britanak & Rao’s algorithm

<table>
<thead>
<tr>
<th>Multiplication &amp; Addition in IMDCT</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>36 points long block</td>
</tr>
<tr>
<td>ISO reference</td>
</tr>
<tr>
<td>Britank Rao’s</td>
</tr>
<tr>
<td>12 points short block</td>
</tr>
<tr>
<td>ISO reference</td>
</tr>
<tr>
<td>Britank Rao’s</td>
</tr>
</tbody>
</table>
Algorithm Optimization in C-code level
- subband synthesis

- Two complexity
  - Matrix operation
    - Can be computed by a 32-point DCT and some data copy operation
  - Window filter

- Using Lee’s DCT algorithm

<table>
<thead>
<tr>
<th>MULTIPLICATION &amp; ADDITION IN (1)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mul</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>ISO reference</td>
</tr>
<tr>
<td>Lee’s</td>
</tr>
</tbody>
</table>
Algorithm Optimization in C-code level
- zero value optimization

- Main idea
  - The result is always zero during zero value of input
- Set tags to indicate zero position
- Ex:

Without zero value optimization

\[ \text{For } (i=0 ; i<576 ; i++) \]
\[ \text{Output}[i] = \text{function}[\text{hufi}] \]

With zero value optimization

\[ \text{For } (i=0 ; i<\text{Position}_\text{zero} ; i++) \]
\[ \text{Output}[i] = \text{function}[\text{hufi}] \]
\[ \text{For } (i=\text{Position}_\text{zero} ; i<576 ; i++) \]
\[ \text{Output}[i] = 0 \]
Memory Size Optimization (1)

- The memory size of MP3 decoder related to
  - Coefficient sizes, bytes per coefficient and data spacing of decoding

\[ x_{r_i} = \text{sign}(h_{uf_i}) \cdot |h_{uf_i}|^{\frac{4}{3}} \cdot 2^{\frac{1}{4}(\text{global}_\text{gain}[\text{gr}]-210)} \cdot 2^{-(\text{scale}_\text{mul} \cdot (\text{scale}_l[sfb][ch][gr]+\text{preflag}[\text{gr}] \cdot \text{pretab}[sfb]))} \]

- It needs \(|h_{uf_i}|^{\frac{4}{3}}\) and \(2^x\) (x is not a integer number)
Memory Size Optimization (2)

- The value of $huf_i$ varies from 0 to 8206
  - Need 16Kbytes to store these coefficients if a coefficient take 2 bytes

- Modify the computation
  - $\left| huf_i \right|^{4/3} = 16 \quad \Box \quad \left| huf_i / 8 \right|^{4/3}$
  - Setting upper bound
    - $(huf_i > 256) \ ? 255 : huf_i$

- It needs only 256 coefficients with unaware of distortion
Memory Size Optimization (3)

- Another problem: \(2^{1/4}(\text{global_gain}[gr] - 210)\),
  \[0 \leq \text{global_gain}[gr] < 255\]
- It can be decomposed as \(2^N2^{i/4}\)
  - \(2^N\) can be realized by shift
- Need only 4 coefficients
CPI Optimization

- CPI = CPI\(_{idea}\) + CPI\(_{stall}\)
  - eliminate the CPI\(_{stall}\)
- The main reasons causing pipeline stall
  - JBU (Jump Branch Unit) is placed at EX stage
  - Data cache access is separate into DM and TC stage
  - Cache miss penalty
CPI Optimization (cont.)

- **Solutions**
  - Modify the order of assembly program to eliminate control hazards and load stalls by delay slot techniques
  - Reduce program and data space
  - Rearrange data space
## The Optimization Results

<table>
<thead>
<tr>
<th>Item</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without optimization</td>
<td>90MIPS with CPI=1.35 and 80Kbytes memory</td>
</tr>
<tr>
<td>Algorithm optimization in C</td>
<td>Reduce 13.6MIPS 15.1%</td>
</tr>
<tr>
<td>General optimization in C</td>
<td>Reduce 7.2MIPS 8.0%</td>
</tr>
<tr>
<td>Zero value optimization</td>
<td>Reduce 9.0MIPS 10.0%</td>
</tr>
<tr>
<td>Code optimization in assembler</td>
<td>Reduce 12.2MIPS 13.6%</td>
</tr>
<tr>
<td>Programs after optimization</td>
<td>48MIPS with CPI=1.15 and 49Kbytes memory</td>
</tr>
</tbody>
</table>
Conclusion

- Proposed embedded software code optimization techniques
  - optimized the MP3 decoding program on RISC32 according to these techniques
- Algorithm optimization in C language for achieving smaller computation complexity
- Optimize target code for achieving low CPI value and small memory sizes on RISC32