Complex Library Mapping for Embedded Software Using Symbolic Algebra

Armita Peymandoust  Tajana Simunic*
Giovanni De Micheli
Stanford University
*Hewlett-Packard Labs

Outline

- Motivation
- MP3 decoder on SmartBadge IV
- Methodology for library mapping
- Library characterization
- Automatic recognition of target code
- Polynomial representation generation
- Automatic mapping to complex library elements
- Results
**Motivation**

- High demand for embedded multimedia applications
  - Low cost with aggressive time to market
  - Power consumption is limited
- More flexibility in software design
  - Hardware changes are less frequent and costly
- Automate steps in optimizing algorithmic-level specification into embedded software
  - Improve productivity and reliability
- Intelligent use of available pre-optimized libraries
  - Avoid manual translation

**Embedded System: SmartBadge IV**

Application: MP3 decoder

- MP3 code and compliance test obtained from ISO

Previous work: Run real time MP3 decode on SmartBadge
**MP3 Variations**

<table>
<thead>
<tr>
<th>Comparison</th>
<th>Original</th>
<th>Manual</th>
<th>IPP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Energy (mWhr)</td>
<td>509.55</td>
<td>4.461</td>
<td>0.41</td>
</tr>
<tr>
<td>Improvement</td>
<td>na</td>
<td>114.22</td>
<td>1254.69</td>
</tr>
<tr>
<td>Performance (s)</td>
<td>503.92</td>
<td>5.47</td>
<td>0.31</td>
</tr>
<tr>
<td>Improvement</td>
<td>na</td>
<td>92.11</td>
<td>1608.42</td>
</tr>
</tbody>
</table>

- Faster than real time, i.e. additional energy savings

**Goal:** Use IPP library for MP3 decoder automatically

---

**Intel® Integrated Performance Primitives**

- A library of general and multimedia signal processing kernels optimized for SA-1100
- Enables migration to the SA-1100 from DSP
- Reduces cost and the need for hand optimization
- Compatible with popular embedded operating systems
  - Linux OS and Windows CE
- Supports application porting across certain Intel platforms
Ideal Software Design Flow

Algorithmic-level C Code

Pre-optimized Embedded Library

Compiler-like tool

Optimized C Code using Embedded Library

Reality of Embedded Software

Algorithmic-level C Code

Pre-optimized Embedded Library

Optimized C Code using Embedded Library
Contribution

A library mapping methodology:

- Finds a characterization function for library elements
- Helps identify potential code sections
- Generates polynomial representations for target code
- Uses symbolic algebra to efficiently map the identified code to pre-optimized embedded library

Related Work

- Tree covering code generation [Aho]
  - Map to simple processor instruction
- Retargetable compiler [Goossens, Paulin, Marwedel]
  - Instruction mapping of ASIPs
- Power aware compiling
  - Memory access optimization [Catthoor, Kandemir]
  - Instruction reordering [Tiwari]
- Symbolic algebra and complex instruction mapping [DATE2002]
Our Methodology

Algorithmic-level C Code

Profiling

Critical Code

Polynomial Formulation

Target Code

Symbolic Library Mapping

Pre-optimized Embedded Library

Library Characterization

Optimized C Code using Embedded Library

Library characterization

- Target library:
  - Commercial library available for the particular processor
    - Example: Intel’s integrated performance primitives library
  - A set of in-house pre-optimized routines
  - IEEE floating-point math library for Linux OS
- Each library element is labeled with:
  - Type of inputs and outputs
  - Accuracy
  - Performance and energy consumption
    - Cycle-accurate simulator
  - Functionality by its polynomial representation
    - Obtained from the documentation
Example: Library characterization

<table>
<thead>
<tr>
<th>Library Element</th>
<th>Perf (s)</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>float SubBandSyn</td>
<td>0.94813</td>
<td>1</td>
</tr>
<tr>
<td>fixed SubBandSyn</td>
<td>0.01026</td>
<td>92</td>
</tr>
<tr>
<td>IPP SubBandSyn</td>
<td>0.00198</td>
<td>479</td>
</tr>
<tr>
<td>float IMDCT</td>
<td>0.3872</td>
<td>1</td>
</tr>
<tr>
<td>fixed IMDCT</td>
<td>0.0144</td>
<td>27</td>
</tr>
<tr>
<td>IPP IMDCT</td>
<td>0.0002</td>
<td>1898</td>
</tr>
</tbody>
</table>

\[x_j = \sum_{k=0}^{\frac{n-1}{2}} y_k \cos(\frac{\pi}{2n} (2i+1+\frac{n}{2})(2k+1))\]

Target code identification

- Profiling by cycle accurate simulator
  - Find critical code segments
- Formulate as large polynomials as possible
  - Likelihood of finding a more complex library element increases
  - Achieved by transformation such as loop unrolling, constant and variable propagation, inlining, ...
Energy Profiler Architecture

Source Code

```c
for (i=0; i<30; i++)
{
    x[i] = y[i] + 2 * x[i + 1];
    y[i] = x[i] + z[i];
    LD     R21, #30;
    ADD  R21, R23,R27;
    ... 
}
```

Software Profile

```
fun energy {
    getD 15%
    sort 10%
    init 2%
    ...
}
```

Profiler Operation

- **Profiler supporting tools:**
  - Cycle-accurate energy consumption simulator
  - Compiler
- **Profiler input:**
  - Source code
- **Profiler output:**
  - Cumulative energy
  - Energy consumed by each procedure

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Cumulative</th>
<th>Self</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>(mWhr)</td>
<td>(mWhr)</td>
</tr>
<tr>
<td>main</td>
<td>3.20E-01</td>
<td>5.20E-03</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>III_hybrid</td>
<td>6.71E-02</td>
<td></td>
</tr>
<tr>
<td>SubBandSynthesis</td>
<td>3.72E-02</td>
<td></td>
</tr>
<tr>
<td>III_stereo</td>
<td>2.75E-02</td>
<td></td>
</tr>
<tr>
<td>III_reorder</td>
<td>2.02E-02</td>
<td></td>
</tr>
<tr>
<td>III_antialias</td>
<td>1.45E-02</td>
<td></td>
</tr>
<tr>
<td>III_dequantize_sample</td>
<td>1.40E-02</td>
<td></td>
</tr>
<tr>
<td>III_hufman_decode</td>
<td>3.74E-03</td>
<td></td>
</tr>
<tr>
<td>III_get_scale_factor</td>
<td>1.28E-04</td>
<td></td>
</tr>
<tr>
<td>decode_info</td>
<td>3.20E-05</td>
<td></td>
</tr>
<tr>
<td>III_hybrid</td>
<td>6.71E-02</td>
<td></td>
</tr>
<tr>
<td>inv mdct</td>
<td>6.36E-03</td>
<td></td>
</tr>
<tr>
<td>SubBandSynthesis</td>
<td>3.72E-02</td>
<td></td>
</tr>
<tr>
<td>chendct2_scaled</td>
<td>1.77E-02</td>
<td></td>
</tr>
<tr>
<td>III_stereo</td>
<td>2.75E-02</td>
<td></td>
</tr>
<tr>
<td>III_reorder</td>
<td>2.02E-02</td>
<td></td>
</tr>
<tr>
<td>III_antialias</td>
<td>1.45E-02</td>
<td></td>
</tr>
<tr>
<td>III_dequantize_sample</td>
<td>1.40E-02</td>
<td></td>
</tr>
<tr>
<td>III_hufman_decode</td>
<td>3.74E-03</td>
<td></td>
</tr>
<tr>
<td>huffman_decoder</td>
<td>2.17E-03</td>
<td></td>
</tr>
<tr>
<td>initialize_huffman</td>
<td>1.03E-05</td>
<td></td>
</tr>
<tr>
<td>hasstef</td>
<td>3.20E-06</td>
<td></td>
</tr>
</tbody>
</table>
Polynomial Formulation

Calculates a polynomial representation for the critical code segments:

- Linear functions
  - Extract from the C code
- Bit manipulations or Boolean functions
  - Use interpolation-based algorithms [Smith01]
- Nonlinear functions
  - Approximate by a polynomial
  - Taylor or Chebyshev series expansion
  - Verify approximation to ensure accuracy

Our Methodology

- Algorithmic-level C Code
- Profiling
- Critical Code
- Polynomial Formulation
- Target Code
- Symbolic Library Mapping
- Pre-optimized Embedded Library
- Library Characterization
- Optimized C Code using Embedded Library
Symbolic Library Mapping

**Given:** Polynomial rep. of critical code sections $S$
- A characterized library of complex elements
- A routine for accuracy and cost feedback

**Goal:** Decompose $S$ into available complex library

- Optimize power consumption and performance
- Use symbolic computer algebra routines & algorithms
  - Higher level optimizations possible
  - Example: factorization and simplification
- Output: library element(s) implementing $S$ optimally

Symbolic Computer Algebra

- Algorithms for algebraic manipulation on expressions containing undetermined quantities
- Exact rational arithmetic, arbitrary-precision floating-point arithmetic
- Includes a set of multivariate polynomial manipulation algorithms based on Gröbner basis [Buchberger]
  - Implemented in mathematical tools [Maple, Mathematica]
  - Apply to algorithmic design and library mapping

**Mapping:** Simplification modulo set of polynomials [DAC01]

**Preprocessing:** Horner Form, Factor, Expand, THR, and Substitution [ICCAD01]
Example

- Phase shift keying modulation
  - Map a code segment of PSK to Library

\[ S = 1 - 0.5x_0^2 - x_0x_1 - 0.5x_1^2 + 0.041667x_0^4 + 0.166668x_0^3x_1 + 0.250002x_0^2x_1^2 + 0.166668x_0x_1^3 + 0.041667x_1^4; \]

\[ \text{siderel} := \{y = x_0 + x_1\}; \]
\[ \text{simplify}(S, \text{siderel}, [x_0, x_1, y]); \]
\[ y := x_0 + x_1; \]
\[ s := \cos(x); \]
**Mapping Algorithm**

- Polynomial Representation of Critical Code
  - THR
  - Factor
  - Expand
  - Horner

- Polynomial Rep. of Library Elements
  - Select Side Relation Set
  - Simplify
  - Add to Side Relation Set

- Mapped?
  - Yes
  - No

Choose Best Solution

**Implementation**

- An automatic library mapping tool
  - Minimal execution time mapping
  - Implemented in C with calls to Maple V
  - Enhancement to general compilers

- Inputs:
  - Polynomial representation of critical code segments
  - Polynomial representation of complex library elements

- Output:
  - Sequence of library calls and processor instructions
Experimental Setup

- System power measurements
- Per element power measurements
- Read timer for execution time

- Fine grain power measurements by data acquisition board

Original Code

- One frame is decode in 2.6 seconds
- Profiler results show three critical functions
- Generate as large as possible polynomials for the critical functions
Mapped to In-house Library

- One frame is decode in 0.03 seconds
- Profiler results show two critical functions

Mapped to IPP Library

- One frame is decode in 0.005 seconds
- More evenly distributed execution time
MP3 Final Results

<table>
<thead>
<tr>
<th>Code version</th>
<th>Perf (s)</th>
<th>Factor</th>
<th>Energy (mWhr)</th>
<th>Factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Original</td>
<td>503.92</td>
<td>1.0</td>
<td>509.553</td>
<td>1.0</td>
</tr>
<tr>
<td>IPP SubBand</td>
<td>301.43</td>
<td>1.7</td>
<td>292.516</td>
<td>1.7</td>
</tr>
<tr>
<td>IPP SubBand &amp; IMDCT</td>
<td>211.27</td>
<td>2.4</td>
<td>199.123</td>
<td>2.6</td>
</tr>
<tr>
<td>In-House (IH) Library</td>
<td>5.47</td>
<td>92.1</td>
<td>4.461</td>
<td>114.2</td>
</tr>
<tr>
<td>In-House + IPP SubBand</td>
<td>3.33</td>
<td>151.4</td>
<td>2.7955</td>
<td>182.3</td>
</tr>
<tr>
<td>IH+IPP SubBand &amp; IMDCT</td>
<td>1.43</td>
<td>352.4</td>
<td>1.1708</td>
<td>435.2</td>
</tr>
<tr>
<td>IPP MP3</td>
<td>0.41</td>
<td>1240.8</td>
<td>0.3133</td>
<td>1626.4</td>
</tr>
</tbody>
</table>

- Runs a factor of four faster than real-time
- Additional energy savings are possible by using frequency and voltage scaling

Conclusions

- Bridge the gap between algorithmic design and optimized embedded software
- A software library mapping methodology
  - Library element characterization
  - Highlights potential target code
  - Polynomial representation
  - Maps to pre-optimized library available
- Using a new symbolic mapping method
- Significant productivity improvement