DSP Code Generation with Optimized Data Word-Length Selection

D. MENARD, O. SENTIEYS

R2D2 Team
IRISA / INRIA
ENSSAT / University of Rennes 1
name@irisa.fr
Outline of the presentation

• Motivations
• Floating-point to Fixed-point Conversion
• Data Word-Length Selection
• Experiments and Results
• Conclusion
Motivations

• Embedded digital signal processing systems
  • Specification with floating-point data types
  • Implementation in fixed-point architectures

• Methodologies for the automatic conversion of a floating-point description into a fixed-point specification
  • Goals for Digital Signal Processors (DSP)
    ⇨ Maximize the accuracy
    ⇨ Minimize the size and the code execution time

⇨ New methodology for the floating-point to fixed-point conversion
Motivations

• Fixed-point data specification

  Integer part : \( m \) bits
  Fractional part : \( n \) bits

  Fixed-point format : \((b,m,n)\)

• Goals and constraints of fixed-point coding
  • Obtain a valid fixed-point specification
    ➞ Avoid overflows
    ➞ Respect fixed-point arithmetic rules

  • Maximize the accuracy
Motivations

• Existing methodologies:
  • FRIDGE [2], CoCentric Fixed Point Designer (Synopsys)
  • Autoscaler for C (University of Seoul) [1]
    ⇒ Transformation of floating-point C code into Fixed-point C code

• Goals of our methodology
  • Implementation of floating-point algorithms in fixed-point DSP under accuracy constraint
    ⇒ Code execution time minimized under accuracy constraint
    ⇒ The DSP architecture is taken into account for fixed-point data coding
  • Data types supported by the DSP


Part 2

Methodology for the Floating-point to Fixed-point Conversion
Methodology structure

![Methodology Diagram]

- **Compilation Infrastructure**: 
  - **Front-end SUIF**
  - **SUIF To C**
  - **IR SUIF-CALIFE**
  - **Code generation**
  - **CALIFE**
  - **Assembly code**

- **Assembly code**
- **C source code**

- **Precision Evaluation**

- **Floating-point to Fixed-point Conversion**

- **SQNR\(_{\text{min}}\)**

- **$SQNR(b, \bar{m})$**
Floating-to-Fixed Point Conversion

- **Goal**
  - Determination of the data dynamic range

- **Method**
  - Technique based on an analytical approach: interval arithmetic, L1 norm

- **Goals**
  - Determination of the data binary-point position
  - Insertion of the scaling operations

- **Method**
  - Propagation of the binary-point position through the Data Flow Graph
    - A rule is defined for each kind of operator
Floating-to-Fixed Point Conversion

- **Goals**
  - Optimization of the scaling operation location
    - The scaling operations are moved to reduce the code execution time
  - Minimize the code execution time as long as the accuracy constraint is fulfilled

- **Method**
  - DSP without Instruction Level Parallelism (ILP) (conventional DSP)
    - The optimization is achieved before the code generation
  - DSP with ILP (VLIW)
    - The optimization is achieved during the scheduling stage
Part 3

Data Word-Length Selection
Data word-length in DSP

• DSP native data word-length : \( N \)
  • \( N \) is equal to 16 bits for most of the DSP
  • \( N \) can be customized for ASIP or some DSP cores

• Multi-precision instructions
  • The data are stored in memory with a greater precision
  • The word-length of a multi-precision data is a multiple of the DSP natural word-length (\( N \))
    \[ \Rightarrow \text{Increase the computation accuracy} \]
    \[ \Rightarrow \text{Increase the code execution time} \]

<table>
<thead>
<tr>
<th>Double precision operations</th>
<th>Classical operations</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Addition</td>
</tr>
<tr>
<td>Addition</td>
<td>2</td>
</tr>
<tr>
<td>Multiplication</td>
<td>3</td>
</tr>
</tbody>
</table>
Data word-length in DSP

• SWP instructions (Sub-Word Parallelism)
  • Several operations executed in parallel on a same operator
    ⇒ Decrease the code execution time
    ⇒ Decrease the computation accuracy
  • DSP with SWP capabilities can manipulate a wide range of data types

<table>
<thead>
<tr>
<th>Processor</th>
<th>Data type (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TMS320C64x (T.I.)</td>
<td>8, 16, 32, 40, 64</td>
</tr>
<tr>
<td>TigerSHARC (A.D.)</td>
<td>8, 16, 32, 64</td>
</tr>
<tr>
<td>SP5, UniPhy (3DSP)</td>
<td>8, 16, 24, 32, 48</td>
</tr>
<tr>
<td>CEVA-X1620 (CEVA)</td>
<td>8, 16, 32, 40</td>
</tr>
<tr>
<td>ZSP500 (LSI Logic)</td>
<td>16, 32, 40, 64</td>
</tr>
<tr>
<td>OneDSP (Siroyan)</td>
<td>8, 16, 32, 44, 88</td>
</tr>
</tbody>
</table>
Data word-length optimization

- Minimize the code execution time under an accuracy constraint
- Select the set of instructions which minimizes the code execution time $T(b)$ and fulfill the precision constraint $SQNR_{min}$
Accuracy evaluation

- Technique based on an analytical approach [3]
  - Reduce the evaluation time compared to the simulation based approaches
    - Simulation based approach drawbacks
      - Long simulation time
      - Fixed-point format optimization process requires multiple simulations

- Accuracy evaluation metric
  - Signal to Quantization Noise Ratio (SQNR)
    - Computation of the SQNR expression \( SQNR(\tilde{b}) \) according to the data word-length \( \tilde{b} \)

Code execution time estimation

• Processor model
  • Data flow instruction set which includes the SWP, the classical and the multi-precision instructions

<table>
<thead>
<tr>
<th>Instruction $j_k$</th>
<th>Function $\gamma_k$</th>
<th>Execution time $t_k$</th>
<th>I/O operand word-length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>$b^{in1}$</td>
</tr>
<tr>
<td>$j_1$</td>
<td>MULT</td>
<td>0.25</td>
<td>8</td>
</tr>
<tr>
<td>$j_2$</td>
<td>MULT</td>
<td>0.5</td>
<td>16</td>
</tr>
<tr>
<td>$j_3$</td>
<td>MULT</td>
<td>1</td>
<td>32</td>
</tr>
<tr>
<td>$j_4$</td>
<td>ADD</td>
<td>0.25</td>
<td>16</td>
</tr>
<tr>
<td>$j_5$</td>
<td>ADD</td>
<td>0.5</td>
<td>32</td>
</tr>
<tr>
<td>$j_6$</td>
<td>ADD</td>
<td>1</td>
<td>64</td>
</tr>
</tbody>
</table>

• Code execution time estimation
  • Goals
    ⇝ Compare and select two instruction series
    ⇝ Simple model to obtain a small evaluation time
Code execution time estimation

• Execution time $T$ estimation technique

\[ T = \sum_{i} t_i \cdot n_i \]

- Operation $o_i$
- Execution time
- Number of time that the operation $o_i$ is executed

• DSP without Instruction Level Parallelism (ILP)
  ➞ Accurate results

• DSP with ILP
  ➞ Estimation of the vertical code execution time
  ➞ The gains due to code parallelization are closed for two instruction series
    ➞ Classical and SWP instructions use the same functional unit at the same clock cycle
Solution modelization

- Application data flow graph example (FIR filter)

\[
x[i] \quad h[i] \\
\rightarrow \quad o_0 \quad \times \\
\rightarrow \quad o_1 \\
\rightarrow \quad ACC
\]

\[I_0 = \{j_1, j_2, j_3\} \]

\[I_1 = \{j_4, j_5, j_6\} \]

Solution modelization with a tree

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Function</th>
<th>Execution time ( t_i )</th>
<th>I/O operand word-length</th>
</tr>
</thead>
<tbody>
<tr>
<td>( j_1 )</td>
<td>MULT</td>
<td>0.25</td>
<td>( 8 ) ( 8 ) ( 16 )</td>
</tr>
<tr>
<td>( j_2 )</td>
<td>MULT</td>
<td>0.5</td>
<td>( 16 ) ( 16 ) ( 32 )</td>
</tr>
<tr>
<td>( j_3 )</td>
<td>MULT</td>
<td>1</td>
<td>( 32 ) ( 32 ) ( 64 )</td>
</tr>
<tr>
<td>( j_4 )</td>
<td>ADD</td>
<td>0.25</td>
<td>( 16 ) ( 16 ) ( 16 )</td>
</tr>
<tr>
<td>( j_5 )</td>
<td>ADD</td>
<td>0.5</td>
<td>( 32 ) ( 32 ) ( 32 )</td>
</tr>
<tr>
<td>( j_6 )</td>
<td>ADD</td>
<td>1</td>
<td>( 64 ) ( 64 ) ( 64 )</td>
</tr>
</tbody>
</table>

\( o_0 \) (MULT)
\( j_1 \)
\( 8 \times 8 \rightarrow 16 \)
\( j_2 \)
\( 16 \times 16 \rightarrow 32 \)
\( j_3 \)
\( 32 \times 32 \rightarrow 64 \)
\( o_1 \) (ADD)
\( j_4 \)
\( 16 \)
\( j_5 \)
\( 32 \)
\( j_6 \)
\( 64 \)

\( \xrightarrow{\text{Branch \\ & Bound algorithm}} \)

Selected Instruction

Operand data word-length \( b_i \)
Search space limitation

- Instruction combination restriction
- Fractional part word-length restriction \( n_{\text{output}} \geq n_{\text{input}} \)

- FIR filter example (same binary-point position for the data)
Search space limitation

• Partial solution evaluation: the exploration is stop if
  → The minimal execution time which can be obtained is greater than the minimal execution time already obtained
  → The maximal SQNR which can be obtained is lower than the SQNR constraint

• Node evaluation order
  • The variables are proceeded according to their influence on the execution time and the SQNR

• Reduction of the number of values per variable
  • Two step optimization
    → 1. Integer optimization leading to the solution $\tilde{b}_i$
    → 2. Branch & Bound with 2 values per variable

\[ b_i^{\text{min}} \leq \tilde{b}_i \leq b_i^{\max} \quad \text{with} \quad b_i^{\text{min}}, b_i^{\max} \in B \]
Part 4

Experiments and Results
Complex correlator

• Complex correlator characteristic

<table>
<thead>
<tr>
<th>Solutions</th>
<th>Operation and data types (bits)</th>
<th>mult</th>
<th>sous</th>
<th>add</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8×8 → 16</td>
<td>16+16→16</td>
<td>16+16→16</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>16×16 → 32</td>
<td>32+32→32</td>
<td>32+32→32</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>32×16 → 32</td>
<td>32+32→32</td>
<td>32+32→32</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>32×16 → 64</td>
<td>64+64→64</td>
<td>64+64→64</td>
<td>32</td>
<td></td>
</tr>
</tbody>
</table>

Precision constraint: $\text{SQNR}_{\text{min}}$ (dB)

Normalised Execution time

Data Flow Graph
Rake receiver

- Used in the WCDMA receiver for UMTS
- Application description

```
Finger 0
Finger 1 y_j(k)
Finger 2 y_z(k)
+
y(k)  F̂

s_x(n)  +

y(k)  ŷ(k)

Finger (path l)

x_i(n)

x^4  ↓ 4
DPDCH/DPCCH decoding
DPDCH

dll late

x^2  ↓ 4
DLL on-time

x^0  ↓ 4
DLL early

Phase removing  δ̂
Channel estimation

DPDCH

DLL

• Symbol decoding subsystem
  • 10 MULT
  • 8 ADD

• Synchronization subsystem
  • 3*10 MULT
  • 3*6 ADD
```
Rake Receiver

• The SQNR constraint is defined according to the receiver performance (Bit Error Rate)

• Target processor C64x (Texas Instrument)

• SWP improvement factor:
  
  \[
  F = \frac{T_{\text{exec-NoSWP}}}{T_{\text{exec-SWP}}}
  \]

<table>
<thead>
<tr>
<th>Number of fingers grouped in the same loop</th>
<th>SWP improvement factor $F$</th>
<th>Symbol decoding subsystem</th>
<th>Synchronization subsystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2.83</td>
<td>1.91</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2.79</td>
<td>2.79</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.51</td>
<td>3.18</td>
<td></td>
</tr>
</tbody>
</table>

Optimization time: 180s

• The code execution time can be reduced dramatically
Conclusion

• New methodology for the Floating-point to Fixed-point conversion
  • The architecture is taken into account

• Data word-length selection
  • Minimize the code execution time under accuracy constraint
    ⇨ This approach allows to
      • Analyze the trade-off between the code execution time and the accuracy
      • Reduce the code execution time

• Perspective: methodology for the case of ASIC / FPGA
  • Several phases of this methodology can be reused