Domain Specific Optimizations

Saman Amarasinghe and William Thies
Massachusetts Institute of Technology

PACT 2003
September 27, 2003
Schedule

1:30-1:40  Overview (Saman)
1:40-2:20  Stream Architectures (Saman)
2:20-3:00  Stream Languages (Bill)
3:00-3:30  Break
3:30-3:55  Stream Compilers (Saman)
3:55-4:20  Domain-specific Optimizations (Saman)
4:20-5:00  Scheduling Algorithms (Bill)
Conventional DSP Design Flow

1. Spec. (data-flow diagram)
2. Design the Datapaths (no control flow)
3. DSP Optimizations
4. Coefficient Tables
5. Rewrite the program
6. Architecture-specific Optimizations (performance, power, code size)
7. C/Assembly Code

- Signal Processing Expert in Matlab
- Software Engineer in C and Assembly

Architectures, Languages, and Compilers for the Streaming Domain
PACT 2003 Tutorial - Saman Amarasinghe, William Thies - MIT CSAIL

3
Design Flow with StreamIt

Application-Level Design

StreamIt Program (dataflow + control)

DSP Optimizations

Architecture-Specific Optimizations

C/Assembly Code

Application Programmer

StreamIt compiler
Design Flow with StreamIt

- Benefits of programming in a single, high-level abstraction
  - Modular
  - Composable
  - Portable
  - Malleable

- The Challenge: Maintaining Performance
  - Replacing Expert DSP Engineer
  - Replacing Expert Assembly Hacker
Our Focus: Linear Filters

- Most common target of DSP optimizations
  - FIR filters
  - Compressors
  - Expanders
  - DFT/DCT

\[
\text{Output is weighted sum of inputs}
\]

- Example optimizations:
  - Combining Adjacent Nodes
  - Translating to Frequency Domain
Representing Linear Filters

- A linear filter is a tuple $\langle \mathbf{A}, \mathbf{\hat{b}}, o \rangle$
  - $\mathbf{A}$: matrix of coefficients
  - $\mathbf{\hat{b}}$: vector of constants
  - $o$: number of items popped

Example

$$y = x \mathbf{A} + \mathbf{\hat{b}}$$
Representing Linear Filters

- A linear filter is a tuple \( \langle A, \hat{b}, o \rangle \)
  - **A**: matrix of coefficients
  - **\( \hat{b} \)**: vector of constants
  - **o**: number of items popped

**Example**

\[
A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}, \quad \hat{b} = \begin{bmatrix} 1 \\ 1 \end{bmatrix}, \quad o = 1
\]
Extracting Linear Representation

work peek N pop 1 push 1 {
    float sum = 0;
    for (int i=0; i<N; i++) {
        sum += h[i]*peek(i);
    }
    push(sum);
    pop();
}

- Resembles constant propagation
- Maintains linear form $\langle \tilde{v}, \tilde{b} \rangle$ for each variable
  - Peek expression: generate fresh $\tilde{v}$
  - Push expression: copy $\tilde{v}$ into $A$
  - Pop expression: increment $o$
Optimizations using Linear Analysis

1) Combining adjacent linear structures

2) Shifting from time to the frequency domain

3) Selection of ‘optimal’ set of transformations
1) Combining Linear Filters

- Pipelines and splitjoins can be collapsed
- Example: pipeline

\[ y = x A \]
\[ z = y B \]
\[ z = x C \]
Combination Example

\[
A = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}
\]

\[
B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}
\]

\[
C = \begin{bmatrix} 32 \end{bmatrix}
\]
AB for any A and B??

- Linear Expansion

Original

\[ [A] \]

E

E

pop = \sigma

Expanded

\[
\begin{bmatrix}
\end{bmatrix}
\]
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:

\[ A \]
\[ \text{pop} = 1 \]

\[ B \]
\[ \text{pop} = 2 \]

\[ A' \]

\[ B' \]
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:

\[ \text{pop} = 1 \]

\[ \text{pop} = 2 \]
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:

A

\[ pop = 1 \]

B

\[ pop = 2 \]

A’

\[ \times \times \]

B’
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:

$A$

$B$

$A'$

$B'$
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:

\[ \text{pop} = 1 \]

\[ \text{pop} = 2 \]
AB for any A and B??

Need to “expand” matrices to a steady-state cycle:
AB for any A and B??

Expanded dimensions match

A

pop = 1

B

pop = 2

A'

pop = 2

B'

pop = 6
Floating-Point Operations Reduction

- FIR
- RateConvert
- TargetDetect
- FMRadio
- Radar
- FilterBank
- Vocoder
- Oversample
- DTOA

Flops Removed (%)

Benchmark
2) From Time to Frequency Domain

- Convolutions can be done cheaply in the Frequency Domain

\[ \sum X_i^* W_{n-i} \]

- Painful to do by hand
  - Blocking
  - Coefficient calculations
  - Startup
  - Multiple outputs
  - Interfacing with FFT library

\[ X \leftarrow \mathcal{F}(x) \]
\[ Y \leftarrow X \cdot^* H \]
\[ y \leftarrow \mathcal{F}^{-1}(Y) \]

FFT

VVM

IFFT
Generic Freq. Implementation

```c
float → float pipeline optimizedFreq (A, b, e, o, u) {
    add float → float filter {
        N ← 2[^lg(2e)]
        m ← N - 2e + 1
        partials ← new array[0...e - 2, 0...u - 1]
        r ← m + e - 1

        init {
            for j = 0 to u - 1
                H[*], j ← FFT(N, A[*], u - 1 - j)
        }

        prework peek r pop r push u * m {
            x ← pop(0...m + e - 2)
            X ← FFT(N, x)
            for j = 0 to u - 1 {
                Y[*], j ← X * H[*], j
                y[*], j ← IFFT(N, Y[*], j)
                partials[*], j ← y[m + e - 1...m + 2e - 3, j]
            }
            for i = 0 to m - 1
                for j = 0 to u - 1
                    push(y[i + e - 1, j] + b[j])
        }

        work pop r push u * r {
            x ← pop(0...m + e - 2)
            X ← FFT(N, x)
            for j = 0 to u - 1 {
                Y[*], j ← X * H[*], j
                y[*], j ← IFFT(N, Y[*], j)
                partials[*], j ← y[m + e - 1 + i, j]
            }
            for i = 0 to e - 1
                for j = 0 to u - 1
                    push(y[i, j] + partials[i, j])
                    partials[i, j] ← y[m + e - 1 + i, j]
            }
            for i = 0 to m - 1
                for j = 0 to u - 1
                    push(y[i + e - 1, j] + b[j])
            }
        }

        add Decimator(o, u)
    }
}
```
Floating-Point Operations Reduction

![Bar chart showing flops removed in various benchmarks.](chart.png)
3) Transformation Selection

- When to apply what transformations?
  - Linear filter combination can increase the computation cost
  - Shifting to the Frequency domain is expensive for filters with pop > 1
    - Compute all outputs, then decimate by pop rate
  - Some expensive transformations may later enable other transformations, reducing the overall cost
Selection Algorithm

- Estimate minimal cost for each structure:
  - Linear combination
  - Frequency translation
  - No transformation
    - If hierarchical, consider all possible groupings of children

  Cost function based on profiler feedback

- Overlapping sub-problems allows efficient dynamic programming search
Radar (Transformation Selection)

First compute cost of individual filters:
Radar (Transformation Selection)

First compute cost of individual filters:

- Linear Combination
- Frequency
- No Transform

(low)
Radar (Transformation Selection)

First compute cost of individual filters:

Linear Combination

Frequency

No Transform

1x1
Radar (Transformation Selection)

Then, compute cost of 1x2 nodes:

- Linear Combination
- Frequency
- No Transform

1x1
Radar (Transformation Selection)

Then, compute cost of 1x2 nodes:

<table>
<thead>
<tr>
<th>Linear Combination</th>
<th>Frequency</th>
<th>No Transform</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Linear Combination" /></td>
<td><img src="image2" alt="Frequency" /></td>
<td><img src="image3" alt="No Transform" /></td>
</tr>
</tbody>
</table>

1x1

low  high
Radar (Transformation Selection)

Then, compute cost of 1x2 nodes:

1x2

Linear Combination

Frequency

No Transform

1x1
Radar (Transformation Selection)

Continue with 1x3 2x1 3x1 4x1
1x4 2x2 3x2 4x2
2x3 3x3 4x3
2x4 3x4 4x4

Overall solution
Radar (Transformation Selection)
Radar (Transformation Selection)
Radar (Transformation Selection)
Radar

Maximal Combination and Shifting to Frequency Domain

unakanbikil

Sink

Using Transformation Selection

2.4 times as many FLOPS

half as many FLOPS
Floating-Point Operations Reduction

![Bar chart showing flops removed for different benchmarks.]

- **FIR RateConvert**
- **TargetDetect**
- **FM Radio**
- **Radar**
- **Filter Bank**
- **Vocoder**
- **Oversample**
- **DToA**

- **linear**
- **freq**
- **autosel**

**Benchmarks**

- **Flops Removed (%)**
  - FIR: 0.3%
  - RateConvert: -140%
  - TargetDetect: 0%
  - FM Radio: 0%
  - Radar: 0%
  - Filter Bank: 0%
  - Vocoder: 0%
  - Oversample: 0%
  - DToA: 0%

Architectures, Languages, and Compilers for the Streaming Domain
PACT 2003 Tutorial - Saman Amarasinghe, William Thies - MIT CSAIL
Experimental Results

- Fully automatic implementation
  - StreamIt compiler

- StreamIt to C compilation
  - FFTW for shifting to the frequency domain

- Benchmarks all written in StreamIt

- Measurements
  - Dynamic floating-point instruction counting
  - Speedups on a general purpose processor
Execution Speedup

On a Pentium IV
Related Work

- SPIRAL/SPL (Püschel et. al)
  - Automatic derivation of DSP transforms
- FFTW (Friego et. al)
  - Wicked fast FFT
- Affine Analysis (Karr, Acta Informatica, 1976)
  - Affine relationships among variables of a program
- Linear Analysis (Cousot, Halbwachs, POPL, 1978)
  - Automatic discovery of linear restraints among variables of a program
Conclusions

- A DSP Program Representation: *Linear Filters*
  - A dataflow analysis that recognizes linear filters

- Three Optimizations using Linear Information
  - Adjacent Linear Structure Combination
  - Time Domain to Frequency Domain Transformation
  - Automatic Transformation Selection

- First Step in Replacing the DSP Engineer from the Design Flow
  - On the average 90% of the FLOPs eliminated
  - Average performance speedup of 450%

- StreamIt: A Unified High-level Abstraction for DSP Programming
  - Increased abstraction does not have to sacrifice performance

http://cag.lcs.mit.edu/linear/