Linear Analysis and Optimization of Stream Programs

Andrew A. Lamb
William Thies
Saman Amarasinghe

The New Laboratory for Computer Science and Artificial Intelligence
Massachusetts Institute of Technology
Streaming Application Domain

- Based on audio, video, or data stream
- Increasingly prevalent and important
  - Embedded systems
    - Cell phones, handheld computers
  - Desktop applications
    - Streaming media
    - Software radio
      - Real-time encryption
      - Graphics packages
  - High-performance servers
    - Software routers (Example: Click)
    - Cell phone base stations
    - HDTV editing consoles
Properties of Stream Programs

- A large (possibly infinite) amount of data
  - Limited lifetime of each data item
  - Little processing of each data item
- Computation: apply multiple filters to data
  - Each filter takes an input stream, does some processing, and produces an output stream
  - Filters are independent and self-contained
- A regular, static computation pattern
  - Filter graph is relatively constant
  - A lot of opportunities for compiler optimizations
The StreamIt Language

Goals:
- Provide a High-Level Programming Paradigm
- Improve Programmer Productivity
- Match Performance of Hand-Hacked Assembly

Contributions
- Language Design, Structured Streams, Buffer Management (*CC 2002*)
- Exploiting Wire-Exposed Architectures (*ASPLOS 2002*)
- Scheduling of Static Dataflow Graphs (*LCTES 2003*)
- Domain Specific Optimizations (*PLDI 2003*)
Example: Freq band detection

- Used in...
  - metal detector
  - garage door opener
  - spectrum analyzer

Source:
Application Report SPRA414
Texas Instruments, 1999
void->void pipeline FrequencyBand {
    float sFreq = 4000;
    float cFreq = 500/(sFreq*2*pi);
    float wFreq = 100/(sFreq*2*pi);
    add D2ASource(sFreq);
    add BandPassFilter(1, cFreq-wFreq, cFreq+wFreq, 100);
    add splitjoin {
        split duplicate;
        for (int i=0; i<4; i++) {
            add Detector(i/4);
            add LEDOutput(i);
        }
        join roundrobin(0);
    }
}
Freq band detection in StreamIt

void->void pipeline FrequencyBand {
    float sFreq = 4000;
    float cFreq = 500/(sFreq*2*pi);
    float wFreq = 100/(sFreq*2*pi);
    add D2ASource(sFreq);

    float->float pipeline BandPassFilter(float gain, float ws, float wp, int num) {
        add LowPassFilter(1, wp, num);
        add HighPassFilter(gain, ws, num);
    }

    add splitjoin {
        split duplicate;
        for (int i=0; i<4; i++) {
            add Detector(i/4);
            add LEDOutput(i);
        }
        join roundrobin(0);
    }
}

declare float->float pipeline BandPassFilter(1, 200, 500, 8);
Freq band detection in StreamIt

float->float filter LowPassFilter(float g, float cFreq, int N)
{
  float[N] h;
  init {
    int OFF = N/2;
    for (int i=0; i<N; i++) {
      h[i] = g*sin(...);
    }
  }
  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();
  }
}
void->void pipeline FrequencyBand {
    float sFreq = 4000;
    float cFreq = 500/(sFreq*2*pi);
    float wFreq = 100/(sFreq*2*pi);
    add D2ASource(sFreq);
    add BandPassFilter(1, cFreq-wFreq, cFreq+wFreq, 100);
    add splitjoin {
        split duplicate;
        for (int i=0; i<4; i++) {
            add Detector(i/4);
            add LEDOutput(i);
        }
        join roundrobin(0);
    }
}

float->float pipeline BandPassFilter(float gain, float ws, float wp, int num) {
    add LowPassFilter(1, wp, num);
    add HighPassFilter(gain, ws, num);
}

float->float filter LowPassFilter(float g, float cFreq, int N) {
    float[N] h;

    init {
        int OFF = N/2;
        for (int i=0; i<N; i++) {
            h[i] = g*sin(...);
        }
    }

    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();
    }
}
Freq band detection on a TI DSP

A/D

Low pass
High pass

Band pass

Duplicate

Detect
Detect
Detect
Detect

LED
LED
LED
LED
DSP Implementation

Source: Application Report SPRA414, Texas Instruments, 1999
Cont.

Set up DAC Module

Bits 15-14 (00) FREE, SOFT - Emulation Control Bits
Stop immediately on emulation suspend

Bits 13-11 (010) TMODE2-TMODE0 - Count Mode Selection

SPLK #0011100000001111b, OCRA

Bits 10-8 (000) TPS2-TPS0 - Input Clock Prescaler
OCRA - Output Control Register A
Divide by 1

Bit 15 (0) CRA.15 - IOPB7
Bit 7 (0) Reserved

Date Space, 0WS
Bit 14 (0) CRA.14 - IOPB6
Bit 6 (0) TENABLE - Timer Enable

I/O Space, 1WS
Bit 13 (1) CRA.13 - T3PWM/T3CMP
Disable timer operations

Bit 12 (1) CRA.12 - T2PWM/T2CMP
Internal Clock Source

Bit 10 (0) CRA.10 - IOPB2
Bits 3-2 (00) TCLD1, TCLD0 - Timer Compare Register Reload

Bit 9 (0) CRA.9 - IOPB1
Condition

XVALUE .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Bit 8 (0) CRA.8 - IOPB0
When counter is 0

.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Bit 1 (1) TECMPR - Timer compare enable
Bit 3 (1) CRA.3 - ADCIN8
Enable timer compare operation

.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Bit 2 (1) CRA.2 - ADCIN9
Bit 1 (1) CRA.1 - ADCIN1

5432109876543210

.sect ".blk1"
VALUEIN .word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

GP Timer 2 - Not Used

.word 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0

Set up ADC Module

T2CON - GP Timer 2 Control Register

Bits 15-14 (00) FREE, SOFT - Emulation Control Bits
Stop immediately on emulation suspend

Bits 13-11 (000) TMODE2-TMODE0 - Count Mode Selection

Bit 15 (0) Suspend-SOFT - Complete Conversion before halting emulator
Bit 7 (0) TSWT1 - GP Timer 1 timer enable bit

Coefficient for 500Hz Bandpass filter for 4kHz Sampling Frequency

BCOEFF .word 0000h, 0002h, 0002h, 0001h
.data
Coefficients for 500Hz Highpass filter for 6kHz Sampling Frequency

.word 0000h, 0003h, 0005h, 0007h

Internal Clock Source

.word 0000h, 0013h, 0025h, 0021h

Condition

.word 0011h, 00A4h, 00FDh, 00BFh

Enable ADC
When counter is 0

Bit 10 (0) ADCCONRUN - ADC Continuous Conversion Mode
Bit 1 (0) TECMPR - Timer compare enable

Disable Continuous Conversion
Disable timer compare operation

.word 0033h, 0206h, 02F1h, 0220h

.mask ADCINTFLAG

.word 0FFC2h, 0FD19h, 0FBD6h, 0FD05h

.mask ADCEVENT

.word 003Ch, 03B8h, 054Ah, 03C5h

Mask ADCEVENT

.word 0FFD4h, 0FBB4h, 0F9EAh, 0FBADh
Bit 8 (1) ADCINTFLAG - ADC Interrupt Flag
Bit 1 (0) TECMPR - Timer compare enable

.bit 0 (0) SELT1PR - Period Register select (7ffh)

.Bit 2-0 (101) ADCPSCALE - ADC Input Clock Prescaler (7ffh)

Threshold values for LEDs

THRESHOLD1,1 ;Threshold value for 1st LED
THRESHOLD2,1 ;Threshold value for 2nd LED
THRESHOLD4,1 ;Threshold value for 4th LED
THRESHOLD7,1 ;Threshold value for 7th LED
THRESHOLD8,1 ;Threshold value for 8th LED
RESET_MAX,1 ;Counter to determine when to reset MAXIN

0.1 usec x 16 x 6 = 9.6 usec >= 6 usec

Source: Application Report SPRA414, Texas Instruments, 1999
The New Laboratory for Computer Science and Artificial Intelligence
Massachusetts Institute of Technology

Source: Application Report SPRA414, Texas Instruments, 1999
; Output is below THRESHOLD4, THRESHOLD2, & THRESHOLD1. Turn off LEDS
; BELOW1 SPLK #0000h,LEDSOUT
; Output is below THRESHOLD4, but above THRESHOLD1. Turn on DS1
; ABOVE1 SPLK #0001h,LEDSOUT
; Output is below THRESHOLD4, but above THRESHOLD2. Check if above
; THRESHOLD3
; ABOVE2 LT DIFFIN
; TH3 MPY THRESHOLD3
; PAC
; SACH TEMP,1
; LACC TEMP
; SUB DIFFOUT
; BCND ABOVE3,LT
; Output is below THRESHOLD4 and THRESHOLD3, but above THRESHOLD2. Turn on DS1-DS2
; BELOW3 SPLK #0003h,LEDSOUT
; Output is below THRESHOLD4, but above THRESHOLD3 and THRESHOLD2. Turn on DS1-DS3
; ABOVE3 SPLK #0007h,LEDSOUT
; Output is above THRESHOLD4. Check if above THRESHOLD6
; ABOVE4 LT DIFFIN
; TH6 MPY THRESHOLD6
; PAC
; SACH TEMP,1
; LACC TEMP
; SUB DIFFOUT
; BCND ABOVE6,LT
; Output is above THRESHOLD4, but below THRESHOLD6. Check if above
; THRESHOLD5.
; BELOW6 LT DIFFIN
; TH5 MPY THRESHOLD5
; PAC
; SACL * ;ACC = ADCTRL1
; ADD #1h ;Set bit to restart the ADC
; SACL * ;Start converting next value
; LDP #232
; LACC EVIFRA ;Clear the flag register of Event Manager
; SACL EVIFRA
; CLRC INTM ;ENABLE INTERRUPTS
; RET ;Return to main line
;===================================================================
; ISR - PHANTOM
; Description: Dummy ISR, used to trap spurious interrupts.
; Revision:
; Last Update: 16-06-95
;===================================================================

Source: Application Report SPRA414, Texas Instruments, 1999
Conventional DSP Design Flow

Spec. (data-flow diagram) →

Design the Datapaths (no control flow) →

DSP Optimizations →

Coefficient Tables →

Rewrite the program →

Architecture-specific Optimizations (performance, power, code size) →

C/Assembly Code

Signal Processing Expert in Matlab

Software Engineer in C and Assembly
Any Design Modifications?

- Center frequency from 500 Hz to 1200 Hz?
  - According to TI, in the conventional design-flow:
    - Redesign filter in MATLAB
    - Cut-and-paste values to EXCEL
    - Recalculate the coefficients
    - Update assembly
  - If using StreamIt
    - Change one constant
    - Recompile
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

Output is weighted sum of inputs

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

- A linear filter is a tuple \(<A, b, o>\)
  - **A**: matrix of coefficients
  - **b**: vector of constants
  - **o**: number of items popped

- Example

\[
x \rightarrow \begin{pmatrix} A & b & o \end{pmatrix} \downarrow
\]

\[
y = xA + b
\]
Representing Linear Filters

- A linear filter is a tuple \(<A, b, o>\)
  - \(A\): matrix of coefficients
  - \(b\): vector of constants
  - \(o\): number of items popped

- Example

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

- Resembles constant propagation
- Maintains linear form \( <v, b> \) for each variable
  - Peek expression: generate fresh \( v \)
  - Push expression: copy \( v \) 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

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

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

\[ C = [32] \]
AB for any A and B??

- Linear Expansion

Original

$[A]$

$pop = \sigma$

Expanded

$\begin{bmatrix}
[A] \\
[A] \\
[A] \\
\end{bmatrix}$
Floating-Point Operations Reduction

Flops Removed (%)

Benchmark

-40%
-20%
0%
20%
40%
60%
80%
100%

FIR
RateConvert
TargetDetect
FMRadio
Radar
FilterBank
Vocoder
Oversample
DToA

0.3%

The New Laboratory for Computer Science and Artificial Intelligence
Massachusetts Institute of Technology
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 etc.

\[ X \leftarrow \mathcal{F}(x) \] \hspace{1cm} \text{FFT}

\[ Y \leftarrow X.* H \] \hspace{1cm} \text{VWM}

\[ y \leftarrow \mathcal{F}^{-1}(Y) \] \hspace{1cm} \text{IFFT}
Floating-Point Operations Reduction

-40%  -20%  0%  20%  40%  60%  80%  100%

FIR  RateConvert  TargetDetect  FMRadio  Radar  FilterBank  Vocoder  Oversample  DTOA

Flops Removed (%)

linear  freq

-140%  0.3%
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

- Overlapping sub-problems allows efficient dynamic programming search

Cost function based on profiler feedback
Radar (Transformation Selection)

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

First compute cost of individual filters:

- Linear Combination
- Frequency
- No Transform
Radar (Transformation Selection)

First compute cost of individual filters:

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

![1x1](image4)
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="Image" alt="Matrix Diagram" /></td>
<td><img src="Image" alt="Matrix Diagram" /></td>
<td><img src="Image" alt="Matrix Diagram" /></td>
</tr>
</tbody>
</table>

min=$\min=$

min$\neq$
Radar (Transformation Selection)

Then, compute cost of 1x2 nodes:

- Linear Combination
- Frequency
- No Transform
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

Maximal Combination and Shifting to Frequency Domain

Using Transformation Selection

2.4 times as many FLOPS

half as many FLOPS
Floating-Point Operations Reduction

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>linear</th>
<th>freq</th>
<th>autosel</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RateConvert</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TargetDetect</td>
<td></td>
<td></td>
<td>0.3%</td>
</tr>
<tr>
<td>FMRadio</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Radar</td>
<td></td>
<td></td>
<td>-140%</td>
</tr>
<tr>
<td>FilterBank</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vocoder</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Oversample</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DTOA</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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

- Effective 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/