Linear Filters in StreamIt

Andrew A. Lamb
MIT
Laboratory for Computer Science
Computer Architecture Group
8/29/2002
Outline

- Introduction
- Dataflow Analysis
- Hierarchical Matrix Combinations
- Performance Optimizations
Basic Idea

**Filter A**

```
a = pop();
b = pop();
c = (a + b) / 2 + 1;
push(c);
```

**Matrix Mult.**

\[
\begin{bmatrix}
1/2 \\
1/2
\end{bmatrix}
\begin{bmatrix}
x \\
y
\end{bmatrix}
= \begin{bmatrix}
1
\end{bmatrix}
\]

\[
y = xA + b
\]
What is a Linear Filter?

- Generic filters calculate some outputs (possibly) based on their inputs.
- Linear filters: outputs \(y_j\) are weighted sums of the inputs \(x_i\) plus a constant.

\[
y = \sum_{i \in [1,N]} w_i x_i + b \\
\text{for } b \text{ constant} \\
w_i \text{ constant for all } i \\
N \text{ is the number of inputs}
\]

\[
y = w_1 x_1 + w_2 x_2 + (...) + w_N x_N + b
\]
Linearity and Matrices

- Matrix multiply is exactly weighted sum
- We treat inputs \((x_i)\) and outputs \((y_j)\) as vectors of values \((x,\) and \(y\) respectively)\n- Filter is represented as a matrix of weights \(A\) and a vector of constants \(b\)
- Therefore, filter represents the equation \(y = xA + b\)
Equation Example, $y_1$

$$
\begin{bmatrix}
  x_1 & x_2 & x_3 & x_4 \\
\end{bmatrix}
\begin{bmatrix}
  a_{1,1} & a_{1,2} & a_{1,3} \\
  a_{2,1} & a_{2,2} & a_{2,3} \\
  a_{3,1} & a_{3,2} & a_{3,3} \\
  a_{4,1} & a_{4,2} & a_{4,3} \\
\end{bmatrix}
+ \begin{bmatrix}
  b_1 & b_2 & b_3 \\
\end{bmatrix}
= \begin{bmatrix}
  y_1 & y_2 & y_3 \\
\end{bmatrix}
$$

$$y_1 = (x_1 a_{1,1} + x_2 a_{2,1} + x_3 a_{3,1} + x_4 a_{4,1} + b_1)$$
Equation Example, $y_2$

$$
\begin{bmatrix}
x_1 & x_2 & x_3 & x_4
\end{bmatrix}
\begin{bmatrix}
a_{1,1} & a_{1,2} & a_{1,3} \\
a_{2,1} & a_{2,2} & a_{2,3} \\
a_{3,1} & a_{3,2} & a_{3,3} \\
a_{4,1} & a_{4,2} & a_{4,3}
\end{bmatrix}
\begin{bmatrix}
b_1 & b_2 & b_3
\end{bmatrix}
= \begin{bmatrix}
y_1 & y_2 & y_3
\end{bmatrix}
$$

$$
y_2 = (x_1 a_{1,2} + x a_{2,2} + x_3 a_{3,2} + x_4 a_{4,2} + b_2)
$$
Equation Example, $y_3$

$$\begin{bmatrix} x_1 & x_2 & x_3 & x_4 \end{bmatrix} \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \\ a_{4,1} & a_{4,2} & a_{4,3} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}$$

$$y_3 = (x_1 a_{1,3} + x a_{2,3} + x_3 a_{3,3} + x_4 a_{4,3} + b_3)$$
Usefulness of Linearity

- Not all filters compute linear functions
  - `push(pop() * pop());`

- Many fundamental DSP filters do
  - DFT/FFT
  - DCT
  - Convolution/FIR
  - Matrix Multiply
Example: DFT Matrix

DFT: \( X(m) \equiv \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N} \), \( m = 0,1,2,\ldots, N - 1 \)

\[
\begin{bmatrix}
W_N^{0\cdot0} & W_N^{0\cdot1} & \cdots & W_N^{0\cdot(N-1)} \\
W_N^{1\cdot0} & W_N^{1\cdot1} & \cdots & W_N^{1\cdot(N-1)} \\
\vdots & \vdots & \ddots & \vdots \\
W_N^{(N-1)\cdot0} & W_N^{(N-1)\cdot1} & \cdots & W_N^{(N-1)\cdot(N-1)}
\end{bmatrix}
\]

\[F_N = \begin{bmatrix}
W_N^{0\cdot0} & W_N^{0\cdot1} & \cdots & W_N^{0\cdot(N-1)} \\
W_N^{1\cdot0} & W_N^{1\cdot1} & \cdots & W_N^{1\cdot(N-1)} \\
\vdots & \vdots & \ddots & \vdots \\
W_N^{(N-1)\cdot0} & W_N^{(N-1)\cdot1} & \cdots & W_N^{(N-1)\cdot(N-1)}
\end{bmatrix}
\]

\[W_N = e^{-j2\pi / N}\]
Example: IDFT Matrix

\[ x(n) \equiv \frac{1}{N} \sum_{m=0}^{N-1} X(m) e^{j2\pi mn/N} , \quad n = 0,1,2,\ldots,N-1 \]

\[ F_{N}^{-1} = \begin{bmatrix}
  w_{N}^{0\cdot0} & w_{N}^{0\cdot1} & \cdots & w_{N}^{0\cdot(N-1)} \\
  w_{N}^{1\cdot0} & w_{N}^{1\cdot1} & \cdots & w_{N}^{1\cdot(N-1)} \\
  \vdots & \vdots & \ddots & \vdots \\
  w_{N}^{(N-1)\cdot0} & w_{N}^{(N-1)\cdot1} & \cdots & w_{N}^{(N-1)\cdot(N-1)}
\end{bmatrix} \]

row m  

column n

\[ w_{N} = \left( \frac{1}{N} \right) e^{j2\pi/N} \]
Usefullness, cont.

- Matrix representations
  - Are “embarrassingly parallel”
  - Expose redundant computation
  - Let us take advantage of existing work in DSP field
  - Well understood mathematics

1: Thank you, Bill Thies
Outline

- Introduction
- Dataflow Analysis
- Hierarchical Matrix Combinations
- Performance Optimizations
Dataflow Analysis

- Basic idea: convert the general code of a filter’s work function into an affine representation (eg $y = xA + b$)
  - The $A$ matrix represents the linear combination of inputs used to calculate each output.
  - The vector $b$ represents a constant offset that is added to the combination.
“Linear” Dataflow Analysis

- Much like standard constant prop.
- Goal: Have a vector of weights and a constant that represents the argument to each `push` statement which become a column in $A$ and an entry in $b$.
- Keep mappings from variables to their linear forms (e.g., vector $+$ constant).
“Linear” Dataflow Analysis

- Of course, we need the appropriate generating cases, eg
  - constants $\rightarrow \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} + c$
  - pop/peek(x) $\rightarrow \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} + 0$
“Linear” Dataflow Analysis

- Like const prop, confluence operator is set union.

- Need combination rules to handle things like multiplication and addition (vector add and constant scale)
Ridiculous Example

```plaintext
a=peek(2);
b=pop();
c=pop();
pop();
pop();
d=a+2b;
e=d+5;
```
Ridiculous Example

```
a=peek(2);
b=pop();
c=pop();
pop();
pop();
d=a+2b;
e=d+5;
```
Ridiculous Example

```c
a=peek(2);
b=pop();
c=pop();
pop();
d=a+2b;
e=d+5;
```
Ridiculous Example

```c
a=peek(2);
b=pop();
c=pop();
pop();
d=a+2b;
e=d+5;
```
a = peek(2);
b = pop();
c = pop();
pop();
d = a + 2b;
e = d + 5;
Ridiculous Example

\[
\begin{align*}
a &= \text{peek}(2); \\
b &= \text{pop}(); \\
c &= \text{pop}(); \\
\text{pop}(); \\
d &= a + 2b; \\
e &= d + 5;
\end{align*}
\]
Constructing matrix $A$

Filter A:

push(b);
push(a);

$A = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}$  $b = [0 \ 0]$

$a \rightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 4$

$b \rightarrow \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix} + 8$
Constructing matrix $A$

Filter A:

```
push(b);
push(a);
```

```
A = \begin{bmatrix}
0 & 5 \\
0 & 6 \\
0 & 7
\end{bmatrix} \quad b = \begin{bmatrix}
0 \\
8
\end{bmatrix}
```

```
\begin{array}{l}
a \rightarrow \begin{bmatrix}
1 \\
2 \\
3
\end{bmatrix} + 4 \\
\end{array}
```

```
\begin{array}{l}
b \rightarrow \begin{bmatrix}
5 \\
6 \\
7
\end{bmatrix} + 8 \\
\end{array}
```
Constructing matrix $A$

Filter $A$:

filter code

$\text{push}(b) ;$
$\text{push}(a) ;$

$$A = \begin{bmatrix} 1 & 5 \\ 2 & 6 \\ 3 & 7 \end{bmatrix} \quad b = \begin{bmatrix} 4 & 8 \end{bmatrix}$$

$a \rightarrow \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} + 4$

$b \rightarrow \begin{bmatrix} 5 \\ 6 \\ 7 \end{bmatrix} + 8$
weights1 = {1,2,3};
weights2 = {4,5,6};
float sum1 = 0; float sum2 = 0;
float mean1 = 5; mean2 = 17;
for (int i=0; i<3; i++) {
    sum1 += weights1[i]*peek(3-i-1);
    sum1 += weights1[i]*peek(3-i-1);
}
push(sum2 – mean2);
push(sum1 – mean1);
pop();

push = 2
pop = 1
peek = 3
Matrix Mult.

\[ A = \begin{bmatrix}
  3 & 6 \\
  2 & 5 \\
  1 & 4 \\
\end{bmatrix} \]

\[ b = \begin{bmatrix}
  5 \\
  17 \\
\end{bmatrix} \]

\[ y = xA + b \]

\[ \text{size}(x) = 3 \]
\[ \text{size}(y) = 2 \]
Outline

- Introduction
- Dataflow Analysis
- Hierarchical Matrix Combinations
- Performance Optimizations
Combining Filters

- Basic idea: combine pipelines, splitjoins (and possibly feedback loops) of linear filters together
- End up with a single large matrix representation
- The details are tricky in the general case (eg I am still working on them)
The matrix $C$ is calculated as $A' B'$ where $A'$ and $B'$ have been appropriately scaled and duplicated to make the dimensions work out.

In the case where $\text{peek}(B) \neq \text{pop}(B)$, we might have to use two stage filters or duplicate some work to get the dimensions to work out.
A split join reorders data, so the columns of \( \mathbf{C} \) are interleaved copies of the columns of \( \mathbf{A}_1 \) through \( \mathbf{A}_N \).

Matching the rates of \( \mathbf{A}_1 \) through \( \mathbf{A}_N \) is a challenge that I am still working out.
It is unclear if we can do anything of use with a FeedbackLoop construct. Eigen values might give information about stability, but it is not clear if that is useful... more thought is needed.
Outline

- Introduction
- Dataflow Analysis
- Hierarchical Matrix Combinations
- Performance Optimizations
Performance Optimizations

- Take advantage of our compile time knowledge of the matrix coefficients.
  - eg don’t waste computation on zeros
- Try and leverage existing DSP work on factoring matrices.
- Try to recognize parallel structures in our matrices.
- Use frequency analysis.
Factoring for Performance

\[ A = \begin{bmatrix}
    a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\
    a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\
    a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\
    a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4}
\end{bmatrix} \]

16 multiplies
12 adds

\[ BC = \begin{bmatrix}
    b_{1,1} & 0 & 0 & 0 & 0 \\
    0 & b_{2,2} & 0 & 0 \\
    0 & 0 & b_{3,3} & 0 \\
    0 & 0 & 0 & b_{4,4}
\end{bmatrix} \begin{bmatrix}
    c_{1,1} & 0 & 0 & 0 \\
    c_{2,1} & c_{2,2} & 0 & 0 \\
    c_{3,1} & c_{3,2} & c_{3,3} & 0 \\
    c_{4,1} & c_{4,2} & c_{4,3} & c_{4,4}
\end{bmatrix} \]

14 multiplies
6 adds
SPL/SPIRAL

- Software package that will attempt to find a fast implementation signal processing algorithms described as matrices.
- It attempts to find a sparse factorization of an arbitrary matrix.
- It can automatically derive FFT (eg the Cooley-Turkey algorithm) from DFT definition.
- Claim that their performance is $\approx$ FFTW\(^1\).

1. See [http://www.fftw.org](http://www.fftw.org)
Recognize Parallel Structure

- We can go from SplitJoin to matrix.
- Perhaps we can recognize the reverse transformation.
- Also, implement blocked matrix multiply to keep parallel resources busy.
Instead of computing the matrix product straight up, possibly go to frequency domain.

Rids us of offset vector (added to response at f=0).

 Might allow additional optimizations (because of possible symmetries exposed in frequency domain).
Work left to do

- Implementation of single filter analysis.
- Combining hierarchical constructs.
- Understand the math of automatic matrix factorizations (group theory).
- Analyze frequency analysis.
- Implement optimizations.
- Get results.
Questions for the Future

- Are there any other optimizations?
- Can we produce inverted matrices
  - programmer codes up transmitter and StreamIt automatically creates the receiver.\(^1\)
- How many cycles of a “real” DSP application are spent computing linear functions?
- Can we combine the linear description of what happens inside a filter with the SARE representation of what is happening between them? (POPL paper)

1. Thank you, BT