Linear Analysis and Optimization of Stream Programs

Masterworks Presentation
Andrew A. Lamb

4/30/2003
Professor Saman Amarasinghe
MIT Laboratory for Computer Science
Motivation

- Digital devices, massive computation pervade modern life (cell phones, MP3, HDTV, etc.)
- Devices complex, software more complicated
  - Performance constraints (real time, power consumption) dictate high level of optimization
  - Best performance → assembly (50% cell phone code is written in assembly)
  - Assembly is (very) hard to reuse
- Automatic optimization is critical
Outline

- Motivation
- StreamIt
- Linear Dataflow Analysis
- Performance Optimizations
- Results
The StreamIt Language

- Goals:
  - High performance
  - Improved programmer productivity (modularity)

- Contributions:
  - Structured model of streams
  - Compiler buffer management
  - Automated scheduling (Michal Karczmarek)
  - Target complex architecture (Mike Gordon)
  - Domain specific optimizations (Andrew Lamb)
Programs in StreamIt

- **Traditional:** stream programs are graphs
  - No simple textual representation
  - Difficult to analyze and optimize
- **Insight:** stream programs have structure

![Diagram showing unstructured and structured program graphs]
Why Structured Streams?

- Compared to structured control flow

  GOTO statements

  if / else / for statements

- PRO: more robust, more analyzable
- CON: “restricted” style of programming
Structured Streams

- Basic programmable unit:
  - Filter

- Hierarchical structures:
  - Pipeline
  - SplitJoin
  - Feedback Loop
Representing Filters

- Autonomous unit of computation
- No access to global resources
- One input, one output
- Communicates through FIFO channels
  - pop(), peek(index)
  - push(value)
- “Firing” is the atomic execution step
- A firing’s peek, pop, push rates must be constant
- Code within filter is general purpose – like C
Outline

- Motivation
- StreamIt
- Linear Dataflow Analysis
- Performance Optimizations
- Results
What is a Linear Filter?

- Generic filters generate outputs (possibly) based on their inputs

- Linear filters: standard sense of linearity
  - outputs \( y_j \) are weighted sums of the inputs \( x_i \) (possibly plus a constant)

\[
\bar{y} = \sum_{i=0}^{e-1} a_i x_i + b
\]

- \( b \) constant
- \( a_i \) constant for all \( i \)
- \( e \) is the number of inputs
Linearity and Matricies

- Represent multiple inputs, multiple outputs with matrix multiply
- We treat inputs \((x_i)\) and outputs \((y_j)\) as vectors of values \((x, \text{and } y\) respectively)
- Filter representation:
  - Matrix of weights \(A\)
  - Vector of constants \(b\)
  - peek, pop, push rates
- A filter firing computes:
  \[ y = xA + b \]
Extracting Linear Filters

- **Goal**: convert the filter’s imperative code into an equivalent linear node
  \[ y = \mathbf{x}A + b \]

- **Technique**: “Linear Dataflow Analysis”
  - Resembles standard constant propagation
  - “Linear form” is a vector and a constant
  - Keep mapping from each expr. to linear form

\[
\begin{align*}
\text{expr} & \quad \rightarrow \quad \begin{bmatrix}
    a_0 \\
    \vdots \\
    a_{e-1}
\end{bmatrix} + b \\
\end{align*}
\]

- Extract linear form for each value pushed
Outline

- Motivation
- StreamIt
- Linear Dataflow Analysis
- Performance Optimizations
- Results
1. Combining Linear Filters

- Pipelines and splitjoins containing only linear filters can be collapsed into a single node
- Example: pipeline with peek(B)=pop(B)

\[
\begin{align*}
x_2 &= x_1 A \\
y &= x_2 B
\end{align*}
\]

\[
y = x_1 A' B' = x_1 C
\]

\[C = A' B'\]

where \(A'\) and \(B'\) have been scaled and duplicated so that the dimensions match.
2. Frequency Replacement

- First, identify linear nodes with FIR filters from discrete time linear systems

\[ \lambda = (A, b, 3, M, N) \]

\[ A = \begin{bmatrix} 2 & \cdots & 1 \\ 1 & \cdots & 2 \\ 2 & \cdots & 1 \end{bmatrix} b = [0 \cdots 0] \]

Linear Node

\[ \lambda = (A, b, \text{peek}, \text{pop}, \text{push}) \]
3. Automatic Selection

- Applying optimizations blindly is not good
- Combination example:

\[ \lambda = (A, b, 3, 1, 1) \]

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

\[ \lambda = (A, b, 1, 1, 3) \]

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

\[ \lambda = (A, b, 3, 1, 3) \]

\[ A = \begin{bmatrix} 4 & 5 & 6 \\ 8 & 10 & 12 \\ 12 & 15 & 18 \end{bmatrix} \]

2 mults
output
originally

3 mults
output
after “optimization”
Outline

- Motivation
- StreamIt
- Linear Dataflow Analysis
- Performance Optimizations
- Results
Fmul Reduction

Bar chart showing the multiplications removed (%) for different benchmarks. The benchmarks include FIR, RateConvert, TargetDetect, FMRadio, Radar, FilterBank, Vocoder, Oversample, and DTOA. The chart compares three methods: freq (light blue), linear (dark blue), and autosel (light blue).
Execution Speedup

The chart above illustrates the execution speedup for various benchmarks. The benchmarks include FIR, RateConvert, TargetDetect, FMRadio, Radar, FilterBank, Vocoder, Oversample, and DTOA. The speedup is measured in percentage compared to a baseline performance.

- **FIR**: Speedup up to 800%.
- **RateConvert**: Speedup up to 600%.
- **TargetDetect**: Speedup up to 400%.
- **FMRadio**: Speedup up to 600%.
- **Radar**: Speedup up to 800%.
- **FilterBank**: Speedup up to 1000%.
- **Vocoder**: Speedup up to 600%.
- **Oversample**: Speedup up to 600%.
- **DTOA**: Speedup up to 600%.

The chart uses different colors to represent different speedup categories: freq, linear, and autosel.
Conclusion

- StreamIt is a new language for high performance DSP applications.
- Personal research contributions:
  - Dataflow analysis determines a linear node that represents input/output relationship.
  - Combination and optimization using linear nodes.
  - Average performance speedup of 450%.

Using StreamIt and domain specific optimizations, modularity does not sacrifice performance.