High-Productivity Stream Programming for High-Performance Systems

Rodric Rabbah, Bill Thies, Michael Gordon, Janis Sermulins, and Saman Amarasinghe
Massachusetts Institute of Technology

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

HPEC 2005, MIT LL
The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping

programmability

domain specific optimizations

architecture specific optimizations

- Simple and effective optimizations for streams
- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors
Why an Emphasis on Streaming?
Streaming in other Domains as well

- Cryptography
- Databases
- Face recognition
- Network processing and security
- Scientific codes
- ...

- Attractive programming model because of a simple mapping from specification to implementation
Properties of Stream Programs

- Mostly regular and repeating computation
- Parallel, independent computation with explicit communication

- Amenable to aggressive compiler optimizations
  [ASPLOS ’02, PLDI ’03, LCTES’03, LCTES ’05]
The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping

- Simple and effective optimizations for streams

- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors
Programming in StreamIt

```c
void->void pipeline FMRadio(int N, float freq1, float freq2) {
    add AtoD();
    add FMDemod();
    add splitjoin {
        split duplicate;
        for (int i=0; i<N; i++) {
            add pipeline {
                add LowPassFilter(freq1 + i*(freq2-freq1)/N);
                add HighPassFilter(freq2 + i*(freq2-freq1)/N);
            }
        }
        join roundrobin();
    }
    add Adder();
    add Speaker();
}
```
Programming in StreamIt

void->void pipeline FMRadio(int N, float freq1, float freq2) {
    add AtoD();  // Streams are easily composed
    add FMDemod();

    add splitjoin {
        split duplicate;
        for (int i=0; i<N; i++) {
            add pipeline {
                add LowPassFilter(freq1 + i*(freq2-freq1)/N);
                add HighPassFilter(freq2 + i*(freq2-freq1)/N);
            }
        }
    }
    join roundrobin();

    add Adder();
    add Speaker();
}
Programming in StreamIt

void->void pipeline FMRadio(int N, float freq1, float freq2) {

    add AtoD();
    add FMDemod();
    add splitjoin {
        split duplicate;
        for (int i=0; i<N; i++) {
            add pipeline {
                add LowPassFilter(freq1 + i*(freq2-freq1)/N);
                add HighPassFilter(freq2 + i*(freq2-freq1)/N);
            }
        }
        join roundrobin();
    }
    add Adder();
    add Speaker();

}
void->void pipeline FMRadio(int N, float freq1, float freq2) {
    add AtoD();   // Application is architecture independent (i.e., portable)
    add FMDemod();
    add splitjoin {
        split duplicate;
        for (int i=0; i<N; i++) {
            add pipeline {
                add LowPassFilter(freq1 + i*(freq2-freq1)/N);
                add HighPassFilter(freq2 + i*(freq2-freq1)/N);
            }
        }
        join roundrobin();
    }
    add Adder();
    add Speaker();
}
Filters as Computational Elements

float → float filter FIR (int N) {
    work push 1 pop 1 peek N {
        float result = 0;
        for (int i = 0; i < N; i++) {
            result += weights[i] * peek(i);
        }
        push(result);
        pop();
    }
}

input

0 1 2 3 4 5 6 7 8 9 10 11

output
Benefits of StreamIt

- Communication is exposed and pipeline parallelism is more readily discovered
- Flow of data provides a frame of reference for reasoning about “time” [PPoPP ’05]
  - Powerful advantage when debugging parallel programs

versus

- Multiple threads with independent program counters
- Non-deterministic execution
StreamIt Development Environment

General Debugging Information

StreamIt Graph Components
- expanded and collapsed views of basic programmable unit
- communication buffer with live data

Compiler and Output Consoles

not shown: the StreamIt On-Line Help Manual

StreamIt Graph Zoom Panel

StreamIt Text Editor

void*int filter intSource {
    init {
        work push 4 peek 4
        push(g++);
        push(g++);
    }
}

int*int filter intPrinter {
    init {
        work pop 4 peek 4
        print(g++);
        print(g++);
        print(g++);
    }
}

Compiler and Output Consoles

[PHEC ’05]
StreamIt Applications

• Software radio
• Frequency hopping radio
• Acoustic beam former
• Vocoder
• GMTI (ground moving target indicator)
• DES and Serpent blocked ciphers
• Sorting
• FFTs and DCTs
• JPEG
• ...
MPEG: Motion Video Codec

independently coded
forward/backward predicted
forward predicted

frames encoded using motion prediction
encoding ▼
decoding ▲

luminance and chrominance color data are separated
encoding ▼
decoding ▲

DCT and quantization of 8x8 image block

MPEG-2 decoder
MPEG: Motion Video Codec

- Implementation statistics
  - 4921 lines of code
    - 48 static streams
    - Compile to ~2150 filters
      - 352x240 resolution
    - Reference C implementation has 9832 lines of code
      - Supports interlacing and multi-layer streams
  - 8 weeks of development
  - 1 programmer with no prior MPEG-2 experience

MPEG-2 decoder
Specification in Section 7.4.1: $F''[0][0] = \text{intra\_dc\_mult} \times \text{QF}[0][0]$

Table 7-4 - Relation between intra\_dc\_precision and intra\_dc\_mult

<table>
<thead>
<tr>
<th>intra_dc_precision</th>
<th>bits_of_precision</th>
<th>intra_dc_mult</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

int->int filter `InverseQuantization()` {
    int[4] intra\_dc\_mult = {8, 4, 2, 1};
    int intra\_dc\_precision;

    work pop 1 push 1 {
        push(intra\_dc\_mult[intra\_dc\_precision] * pop());
    }
};
Specification in Section 7.4.1: \( F'[0][0] = \text{intra}\_\text{dc}\_\text{mult} \times \text{QF}[0][0] \)

Table 7-4 - Relation between \text{intra}\_\text{dc}\_\text{precision} and \text{intra}\_\text{dc}\_\text{mult}

<table>
<thead>
<tr>
<th>\text{intra}_\text{dc}_\text{precision}</th>
<th>\text{bits}_\text{of}_\text{precision}</th>
<th>\text{intra}_\text{dc}_\text{mult}</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>1</td>
</tr>
</tbody>
</table>

\[
\text{int[4] intra}\_\text{dc}\_\text{mult} = \{8, 4, 2, 1\};
\]

for (int m = 0; m < W*H/(16*16); m++)
   // six components for chrominance and luminance
   for (int comp = 0; comp < 6; comp++)
      if (macroblock[m].intra)
         macroblock[m].block[comp][0] *= intra\_dc\_mult[intra\_dc\_precision];
   
   // and many lines later
   if (cc == 0)
      val = (dc\_dct\_pred[0] += Get_Luma\_DC\_dct\_diff());
   else if (cc == 1)
      val = (dc\_dct\_pred[1] += Get_Chroma\_DC\_dct\_diff());
   else
      val = (dc\_dct\_pred[2] += Get_Chroma\_DC\_dct\_diff());
   if (Fault\_Flag) return;
   bp[0] = val << (3-intra\_dc\_precision);
The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping
- Simple and effective optimizations for streams
- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors
Conventional DSP Design Flow

1. Specification (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
Design Flow with StreamIt

- Application-Level Design
  - StreamIt Program (dataflow + control)
  - Domain-Specific Optimizations
  - Architecture-Specific Optimizations
  - C/Assembly Code

Application Programmer

StreamIt compiler
- Leverage analyzability of streams and filter code to enable novel stream transformations
- In this talk: linear optimizations
  [PLDI ’03, PLDI ’05, CASES ’05]
Linear Filter Example

- “Drop every third bit in the bit stream”

\[
\begin{align*}
\text{bit} \rightarrow \text{bit filter} \ & \ \text{DropThirdBit} \ \\
& \{ \ \\
& \quad \text{work push} \ 2 \ \text{pop} \ 3 \ \{ \ \\
& \quad \quad \text{push}(\text{pop}()); \ \\
& \quad \quad \text{push}(\text{pop}()); \ \\
& \quad \quad \text{pop}(); \ \\
& \quad \} \ \\
& \} 
\end{align*}
\]
In General

- A linear filter is a tuple $\langle A, \bar{b} \rangle$
  - A: matrix of coefficients
  - $\bar{b}$: vector of constants

Example

- Linear dataflow analysis resembles constant propagation
Opportunities for Linear Optimizations

- Occur frequently in streaming codes
  - FIR filters
  - Compressors
  - Expanders
  - DFT/DCT
  - Bit permutations in encryption algorithms
  - JPEG and MPEG codecs
  - ...

- Example optimizations
  - Combining adjacent nodes
  - Also, translating to frequency domain when profitable
Combining Linear Filters

Filter 1

\[ x \]

\[ y = x A \]

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

Filter 2

\[ y \]

\[ z = y B \]

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

Combined Filter

\[ z = x C \]

\[ C = [32] \]
Results from Linear Optimizations

Pentium 4 results compared to baseline StreamIt
The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping

- Simple and effective optimizations for streams

- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors
Core Compilation Technology

• Focused on a common challenges in modern and future architectures
  - MIT Raw fabric architecture
  - Clusters of workstations
  - ARM, x86, and IA-64

• Compiler’s role: map the computation and communication pattern to processors, memories, and communication substrates
Compiler Issues

- Load balancing
- Resource utilization
- Fault tolerance
- Dynamic reconfiguration
- ...

- In this talk: cache aware scheduling and partitioning [LCTES ’05]
Example Cache Optimization

<table>
<thead>
<tr>
<th></th>
<th>Baseline</th>
<th>Full Scaling</th>
<th>Cache Aware 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>for i = 1 to N</td>
<td>for i = 1 to N</td>
<td>for i = 1 to N</td>
</tr>
<tr>
<td></td>
<td>A();</td>
<td>A();</td>
<td>A();</td>
</tr>
<tr>
<td></td>
<td>B();</td>
<td>for i = 1 to N</td>
<td>B();</td>
</tr>
<tr>
<td></td>
<td>C();</td>
<td>for i = 1 to N</td>
<td>C();</td>
</tr>
<tr>
<td></td>
<td>end</td>
<td>end</td>
<td>end</td>
</tr>
</tbody>
</table>

Working Set Size

- **Baseline**
  - A
  - B
  - C

- **Full Scaling**
  - A
  - B
  - C

- **Cache Aware 1**
  - A
  - B
  - C

**cache size**
## Example Cache Optimization

### Code Snippet

```plaintext
for i = 1 to N
    A();
    B();
    C();
end
```

### Table

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Full Scaling</th>
<th>Cache Aware 2</th>
</tr>
</thead>
</table>
| for i = 1 to N
  A();
  B();
  C();
end | for i = 1 to N
  A();
  B();
  for i = 1 to N
  B();
  C(); | for i = 1 to 64
  A();
  B();
  end
  for i = 1 to 64
  C(); |

### Diagram

- **Working Set Size**
- **Cache Size**

The diagram illustrates the cache accesses for different scaling methods. The green arrows indicate cache hits, while the red arrows indicate cache misses.
Example Cache Optimization

### Baseline
```
for i = 1 to N
    A();
    B();
    C();
end
```

### Full Scaling
```
for i = 1 to N
    A();
    for i = 1 to N
        B();
        C();
    end
```

### Cache Aware
```
for j = 1 to N/64
    for i = 1 to 64
        A();
        B();
        C();
    end
end
```

---

**Working Set Size**

- **Baseline**
  - A
  - B
  - C

- **Full Scaling**
  - A
  - B
  - C

- **Cache Aware**
  - A
  - B
  - C

**Cache size**

- **Baseline**
  - A
  - B

- **Full Scaling**
  - A
  - B

- **Cache Aware**
  - A
  - B

---

**Inst**

- **Baseline**
  - A
  - B
  - C

- **Full Scaling**
  - A
  - B
  - C

- **Cache Aware**
  - A
  - B
  - C

**Data**

- **Baseline**
  - A
  - B
  - C

- **Full Scaling**
  - A
  - B
  - C

- **Cache Aware**
  - A
  - B
  - C
Evaluation Methodology

• **StreamIt compiler generates C code**
  - Baseline StreamIt optimizations
    • Unrolling, constant propagation
  - Compile C code with gcc-v3.4 with -O3 optimizations

• **StrongARM 1110 (XScale) embedded processor**
  - 370MHz, 16Kb I-Cache, 8Kb D-Cache
  - No L2 Cache (memory 100× slower than cache)
  - Median user time

• Also Pentium 3 and Itanium 2 processors

• **Suite of 11 StreamIt Benchmarks**
Cache Optimizations Results

ignoring cache constraints  cache aware

average execution time (normalized to baseline Streamlt)

StrongARM 1110  Pentium 3  Itanium 2
Concluding Remarks

• StreamIt improves programmer productivity without compromising performance
  - Easily identify pipeline and data parallelism
  - Expose information for domain specific and architecture specific optimizations

- Malleable, composable, analyzable, portable
- Linear analysis and optimizations
- Cache aware scheduling and partitioning
Broader Impact

• Integration into future HPCS languages
  - IBM: X10

• StreamIt for graphics applications
  - Programmable graphics pipeline [Graphics Hardware ‘05]

• StreamIt for emerging architectures

• Looking for users with interesting applications
High-Productivity Stream Programming for High-Performance Systems

Rodric Rabbah, Bill Thies, Michael Gordon, Janis Sermulins, and Saman Amarasinghe
Massachusetts Institute of Technology

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

HPEC 2005, MIT LL