Cache Aware Optimization of Stream Programs

Janis Sermulins, William Thies, Rodric Rabbah and Saman Amarasinghe

LCTES
Chicago, June 2005
Streaming Computing Is Everywhere!

- Prevalent computing domain with applications in embedded systems
  - As well as desktops and high-end servers
Properties of Stream Programs

- Regular and repeating computation
- Independent actors with explicit communication
- Data items have short lifetimes
# Application Characteristics: Implications on Caching

<table>
<thead>
<tr>
<th></th>
<th>Scientific</th>
<th>Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Control</strong></td>
<td>Inner loops</td>
<td>Single outer loop</td>
</tr>
<tr>
<td><strong>Data</strong></td>
<td>Persistent array processing</td>
<td>Limited lifetime producer-consumer</td>
</tr>
<tr>
<td><strong>Working set</strong></td>
<td>Small</td>
<td>Whole-program</td>
</tr>
<tr>
<td><strong>Implications</strong></td>
<td>Natural fit for cache hierarchy</td>
<td>Demands novel mapping</td>
</tr>
</tbody>
</table>
## Application Characteristics: Implications on Compiler

<table>
<thead>
<tr>
<th></th>
<th>Scientific</th>
<th>Streaming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parallelism</td>
<td>Fine-grained</td>
<td>Coarse-grained</td>
</tr>
<tr>
<td>Data access</td>
<td>Global</td>
<td>Local</td>
</tr>
<tr>
<td>Communication</td>
<td>Implicit random access</td>
<td>Explicit producer-consumer</td>
</tr>
<tr>
<td>Implications</td>
<td>Limited program transformations</td>
<td>Potential for global reordering</td>
</tr>
</tbody>
</table>
Motivating Example

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

Working Set Size

<table>
<thead>
<tr>
<th>inst</th>
<th>data</th>
<th>inst</th>
<th>data</th>
<th>inst</th>
<th>data</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
</tr>
<tr>
<td>+</td>
<td>+</td>
<td>C</td>
<td></td>
<td>C</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>C</td>
<td>B</td>
<td>C</td>
<td>B</td>
<td>C</td>
</tr>
<tr>
<td>C</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Motivating Example

<table>
<thead>
<tr>
<th>Baseline</th>
<th>Full Scaling</th>
<th>Full Scaling</th>
</tr>
</thead>
<tbody>
<tr>
<td>for i = 1 to N&lt;br&gt;A();&lt;br&gt;B();&lt;br&gt;C();&lt;br&gt;end</td>
<td>for i = 1 to N&lt;br&gt;A();&lt;br&gt;for i = 1 to N&lt;br&gt;B();&lt;br&gt;for i = 1 to N&lt;br&gt;C();&lt;br&gt;end</td>
<td>for i = 1 to 64&lt;br&gt;A();&lt;br&gt;B();&lt;br&gt;for i = 1 to 64&lt;br&gt;C();&lt;br&gt;end</td>
</tr>
</tbody>
</table>
### Motivating Example

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

**Working Set Size**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**cache size**

<table>
<thead>
<tr>
<th>A + B + C</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>A + B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>inst</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
</tbody>
</table>
Outline

- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion
Model of Computation

• Synchronous Dataflow [Lee 92]
  – Graph of autonomous filters
  – Communicate via FIFO channels
  – Static I/O rates

• Compiler decides on an order of execution (schedule)
  – Many legal schedules
  – Schedule affects locality
  – Lots of previous work on minimizing buffer requirements between filters
Example StreamIt Filter

```
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();
}
```
StreamIt Language Overview

• StreamIt is a novel language for streaming
  – Exposes parallelism and communication
  – Architecture independent
  – Modular and composable
    • Simple structures composed to create complex graphs
  – Malleable
    • Change program behavior with small modifications
Freq Band Detector in StreamIt

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

add D2ASource(sFreq);

add BandPassFilter(100, cFreq-wFreq, cFreq+wFreq);

add splitjoin {
  split duplicate;
  for (int i=0; i<4; i++) {
    add pipeline {
      add Detect (i/4);
      add LED (i);
    }
  }
  join roundrobin(0);
}
}
Outline

• StreamIt
• **Cache Aware Fusion**
• Cache Aware Scaling
• Buffer Management
• Related Work and Conclusion
Fusion

- Fusion combines adjacent filters into a single filter

- Reduces method call overhead
- Improves producer-consumer locality
- Allows optimizations across filter boundaries
  - Register allocation of intermediate values
  - More flexible instruction scheduling
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

- Suite of 11 StreamIt Benchmarks

- Evaluate two fusion strategies:
  - Full Fusion
  - Cache Aware Fusion
Results for Full Fusion

(StrongARM 1110)

Hazard: The instruction or data working set of the fused program may exceed cache size!
Cache Aware Fusion (CAF)

• Fuse filters so long as:
  – Fused instruction working set fits the I-cache
  – Fused data working set fits the D-cache

• Leave a fraction of D-cache for input and output to facilitate cache aware scaling

• Use a hierarchical fusion heuristic
Hierarchical Fusion Heuristic
Hierarchical Fusion Heuristic

Does splitjoin fit in cache? Yes!
Hierarchical Fusion Heuristic
Hierarchical Fusion Heuristic

Does splitjoin fit in cache? No!
Hierarchical Fusion Heuristic

Does pipeline segment fit in cache?  No!
Hierarchical Fusion Heuristic

- Identify highest bandwidth connection, fuse greedily
- Does pipeline segment fit in cache?  No!
Hierarchical Fusion Heuristic

Identify highest bandwidth connection, fuse greedily
Full Fusion vs. CAF

The graph compares the execution time of various applications using Full Fusion and CAF. The execution time is normalized to baseline StreamIt. The applications shown include bitonic, fir, fft-fine, fft-coarse, 3gpp, beamformer, matmult, fmradio, filterbank, filterbank2, ofdm, and average. The highest execution time for CAF is 2.7 times that of Full Fusion.
Outline

• StreamIt
• Cache Aware Fusion
• **Cache Aware Scaling**
• Buffer Management
• Related Work and Conclusion
Improving Instruction Locality

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

miss rate = 1 / N

Working Set Size

<table>
<thead>
<tr>
<th>cache size</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
</tr>
</tbody>
</table>

miss rate = 1
Impact of Scaling

Fast Fourier Transform
Impact of Scaling

Fast Fourier Transform
How Much To Scale?

Data Working Set Size

Our Scaling Heuristic:
- Scale as much as possible
- Ensure at least 90% of filters have data working sets that fit into cache
How Much To Scale?

Our Scaling Heuristic:

- Scale as much as possible
- Ensure at least 90% of filters have data working sets that fit into cache
Impact of Scaling

Heuristic choice is 4% from optimal

Fast Fourier Transform
Scaling Results

- Full Fusion
- CAF
- CAF + scaling

Execution time (normalized to baseline StreamIt)

- bitonic
- fir
- fft-fine
- fft-coarse
- 3gpp
- beamformer
- matmult
- fmradio
- filterbank
- filterbank2
- ofdm
- average
Outline

- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion
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();
    }
}

Sliding Window Computation
Buffer Management

Circular Buffer:

\[0 1 2 3 4 5 6 7\]
Buffer Management

Circular Buffer:

1 2 3 4 5 6 7

N
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]
Buffer Management

Circular Buffer:

\[ 8 \ 9 \ \times \ 4 \ 5 \ 6 \ 7 \]

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

\[ 0 \ 1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \]
Buffer Management

Circular Buffer:

Copy-Shift:

peek(i) → A[(head + i) mod 8]
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:
Buffer Management

Circular Buffer:

\[
\text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8]
\]

Copy-Shift:
Buffer Management

Circular Buffer:

\[8 9 \times 4 5 6 7\]

\[\text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8]\]

Copy-Shift:

\[\times 3 4 5 6 7 8 9\]
Buffer Management

Circular Buffer:

```
8 9 X X 4 5 6 7
```

\[
\text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8]
\]

Copy-Shift:

```
X X 4 5 6 7 8 9
```

\[
\text{N}
\]
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

Copy-Shift + Scaling:
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

Copy-Shift + Scaling:
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

Copy-Shift + Scaling:
Buffer Management

Circular Buffer:

\[ 8 \ 9 \ 4 \ 5 \ 6 \ 7 \]

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

\[ \overline{4 \ 5 \ 6 \ 7 \ 8 \ 9} \]

Copy-Shift + Scaling:

\[ \overline{11 \ 12 \ 13 \ 14 \ 15 \ 16} \]
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

Copy-Shift + Scaling:
Buffer Management

Circular Buffer:

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

Copy-Shift + Scaling:
Buffer Management

Circular Buffer:

\[ \begin{array}{cccccccc}
8 & 9 & \times & \times & 4 & 5 & 6 & 7 \\
\end{array} \]

\[ \text{N} \]

\[ \text{peek}(i) \rightarrow A[(\text{head} + i) \mod 8] \]

Copy-Shift:

\[ \begin{array}{cccccccc}
\times & \times & 4 & 5 & 6 & 7 & 8 & 9 \\
\end{array} \]

\[ \text{N} \]

Copy-Shift + Scaling:

\[ \begin{array}{cccccccccccccccccccccccccccc}
\end{array} \]

\[ \text{N} \]
Performance vs. Peek Rate

(StrongARM 1110)

- modulation
- copy-shift
- copy-shift + scaling

execution time (seconds)

N (peek rate)

FIR
Evaluation for Benchmarks

(StrongARM 1110)

- caf + scaling + modulation
- caf + scaling + copy-shift

<table>
<thead>
<tr>
<th></th>
<th>filterbank</th>
<th>filterbank2</th>
<th>fmradio</th>
<th>ofdm</th>
</tr>
</thead>
<tbody>
<tr>
<td>caf</td>
<td>1.0</td>
<td>0.8</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>scaling</td>
<td>1.0</td>
<td>0.8</td>
<td>0.9</td>
<td>1.0</td>
</tr>
<tr>
<td>modulation</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>copy-shift</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>execution time</td>
<td>0.8</td>
<td>0.8</td>
<td>0.9</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Results Summary

<table>
<thead>
<tr>
<th></th>
<th>StrongARM 1110</th>
<th>Pentium 3</th>
<th>Itanium 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Large L2 Cache</td>
<td></td>
<td>Large Reg. File</td>
<td>VLIW</td>
</tr>
<tr>
<td>full fusion</td>
<td>0.75</td>
<td>0.55</td>
<td>0.45</td>
</tr>
<tr>
<td>caf</td>
<td>0.55</td>
<td>0.5</td>
<td>0.4</td>
</tr>
<tr>
<td>caf + scaling</td>
<td></td>
<td>0.4</td>
<td>0.4</td>
</tr>
<tr>
<td>caf + scaling + buffer</td>
<td></td>
<td></td>
<td>0.4</td>
</tr>
</tbody>
</table>

average execution time (normalized to baseline Streamlt)
Outline

• StreamIt
• Cache Aware Fusion
• Cache Aware Scaling
• Buffer Management
• Related Work and Conclusion
Related work

• Minimizing buffer requirements
  – S.S. Bhattacharyya, P. Murthy, and E. Lee
    • Software Synthesis from Dataflow Graphs (1996)
    • AGPAN and RPMC: Complimentary Heuristics for Translating DSP Block Diagrams into Efficient Software Implementations (1997)
    • Synthesis of Embedded software from Synchronous Dataflow Specifications (1999)
  – P.K.Murthy, S.S. Bhattacharyya
    • A Buffer Merging Technique for Reducing Memory Requirements of Synchronous Dataflow Specifications (1999)
    • Buffer Merging – A Powerful Technique for Reducing Memory Requirements of Synchronous Dataflow Specifications (2000)
  – R. Govindarajan, G. Gao, and P. Desai
    • Minimizing Memory Requirements in Rate-Optimal Schedules (1994)

• Fusion

• Cache optimizations
Conclusions

• Streaming paradigm exposes parallelism and allows massive reordering to improve locality

• Must consider both data and instruction locality
  – Cache Aware Fusion enables local optimizations by judiciously increasing the instruction working set
  – Cache Aware Scaling improves instruction locality by judiciously increasing the buffer requirements

• Simple optimizations have high impact
  – Cache optimizations yield significant speedup over both baseline and full fusion on an embedded platform