Programming by Sketching for Bit-Streaming Programs

Armando Solar-Lezama    Rodric Rabbah    Rastislav Bodík
Kemal Ebcğlu

Presented by Michael Vrable

November 10, 2005
Bit-Streaming Programs?

- Programs that manipulate data at the bit-level
- Operate on large stream of data
- Key example: cryptographic algorithms (symmetric ciphers)
Bit-Streaming Programs?

- Programs that manipulate data at the bit-level
- Operate on large stream of data
- Key example: cryptographic algorithms (symmetric ciphers)

DES Initial Permutation
Key Concerns

- Code should be easy to write
  - ...and easy to verify
  - Important to get cipher implementations right!
- Code should execute efficiently
  - Operate at word level, not bit level
  - Akin to vectorization, but not identical
StreamIt: A Domain-Specific Language

(a) reference program

```
bit->bit filter DropThird {
    work push 2 pop 3 {
        for (int i = 0; i < 3; ++i) {
            bit x = peek(i);
            if (i < 2) push(x);
            pop();
        }
    }
}
```

(b) same task, word-aligned

```
```

Sketching for Bit-Streaming Programs
Compiling Bit-Streaming Programs

out &= 1101101101101101b;
out = (out & 1110011111111111b) | ((out & 0001100000000000b) << 1);
out = (out & 1111110011111111b) | ((out & 0000001100000000b) << 2);

// etc.

- Functional, but tedious to write
- Not very efficient
Optimizing Bit-Streaming Programs

Naïve

Optimized
Representing Bit-Streaming Programs in StreamBit

- **Filters**
  - Represent as affine transformation
  - \( f(x) = Mx + v \)
  - \( M_{\text{DropThird}} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \)

- **Pipelines**
  - \( f_1 \Rightarrow f_2 \Rightarrow f_3 \cdots \)

- **Splitjoins**
  - Split input (duplicate/round-robin)
  - Feed through multiple filters in parallel
  - Recombine output
Base Compilation Algorithm

(a)  

(b)  

(c)  

(4-bit architecture)
Instruction Decomposition

▶ Shifts: \[
\begin{pmatrix}
0 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
\end{pmatrix}
\]

▶ Masks: \[
\begin{pmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1 \\
\end{pmatrix}
\]

Instruction Decomposition Example:

\[
\begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 1 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
= \begin{pmatrix}
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
+ \begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\begin{pmatrix}
0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 \\
\end{pmatrix}
\]
Optimized code difficult to produce
  ▶ May require locally sub-optimal choices
  ▶ Large search space for compiler to explore
Solution: Allow user to guide compiler search via sketches
Compiler still responsible for correctness
Program Sketching

SketchDecomp[
    [shift(1:16 by 0 || 1)],
    [shift(1:16 by 0 || 2)],
    [shift(1:16 by 0 || 4)]
];

Sketching for Bit-Streaming Programs
Resolving Permutation Sketches

- Permutations an important class of bit-level operations
- Compiler may resolve sketches by solving series of constraints
- Represent permutation as vector

\[ \langle x_1, x_2, \ldots, x_N \rangle \]

where \( x_i \) is the number of positions by which bit \( i \) moves

- Represent \( k \)-stage decomposition as vector

\[ \langle y_1^1, \ldots, y_N^1, y_2^2, \ldots, y_N^2, \ldots y_1^k, \ldots y_N^k \rangle \]
Establishing Permutation Constraints

- $\text{shift}(b_1, \ldots, b_M \text{ by } j)$
  \[ y_{b_i}^k = j \]

- $\text{shift}(b_1, \ldots, b_M \text{ by } ?)$
  \[ y_{b_i}^k = y_{b_{i+1}}^k \]

- $\text{pos}(b_i, p)$
  \[ b_i + \sum_{j=1}^{k} y_{b_i}^j = p \]

- $\text{shift}(b_1, \ldots, b_M \text{ by } a || b)$
  \[ y_{b_i}^k \in \{a, b\} \]
Establishing Permutation Constraints

- \text{shift}(b_1, \ldots, b_M \text{ by } j)
  \begin{align*}
y_{b_i}^k &= j \\
y_{b_i}^k &= y_{b_{i+1}}^k
\end{align*}

- \text{Final positions are correct}

- \text{pos}(b_i, p)
  \begin{align*}
b_i + \sum_{j=1}^{k} y_{b_i}^j &= p \\
y_{b_i}^k &\in \{a, b\}
\end{align*}

- \text{Shift}(b_1, \ldots, b_M \text{ by } a \parallel b)
  \begin{align*}
y_{b_i}^k &\in \{a, b\}
\end{align*}

- Bits cannot collide
Solving Permutation Constraints

- Represent linear constraints as matrix equation
  \[ Sy = t \]

- Solve via Gaussian elimination
  \[ y = z + \sum_{i=1}^{m} \alpha_i v_i \]

- Equivalent to...
  \[ \forall \alpha = y - z \]
  with \( \alpha, y \) as unknowns

- \( \alpha \) arbitrary, \( y \) limited by non-linear constraints
Other Sketches

- Pipeline restructuring
- Permutation sketches to aid in affine computations
- Table conversions
User Productivity

StreamBit programs unoptimized (no sketching)

C programs manually optimized

Sketching for Bit-Streaming Programs
Code Optimization

![Graph showing performance improvement over time for different implementations. The y-axis represents words per microsecond, and the x-axis represents time in hours. The graph compares a C implementation tuned by an expert with a StreamBit implementation tuned with sketching by a StreamBit expert.](image-url)
StreamIt an effective domain-specific language
Base StreamBit compiler produces decent code
Sketching guides compiler in producing very high-quality code
Only reference code need be correct