Lecture 8

The StreamIt Language
Languages Have Not Kept Up

C ↔ von-Neumann machine

Modern architecture

Two choices:

- Develop cool architecture with complicated, ad-hoc language
- Bend over backwards to support old languages like C/C++
Why a New Language?

For uniprocessors, C was:
• Portable
• High Performance
• Composable
• Malleable
• Maintainable

Bill Thies, MIT.
Why a New Language?

Uniprocessors: C is the common machine language

Bill Thies, MIT.
Common Machine Languages

Uniprocessors: Common Properties
- Single flow of control
- Single memory image

Differences:
- Register File
- ISA
- Functional Units

von-Neumann languages represent the common properties and abstract away the differences

Multicores: Common Properties
- Multiple flows of control
- Multiple local memories

Differences:
- Number and capabilities of cores
- Communication Model
- Synchronization Model

Need common machine language(s) for multicores
Streaming as a Common Machine Language

- For programs based on streams of data
  - Audio, video, DSP, networking, and cryptographic processing kernels
  - Examples: HDTV editing, radar tracking, microphone arrays, cell phone base stations, graphics

- Several attractive properties
  - Regular and repeating computation
  - Independent filters with explicit communication
  - Task, data, and pipeline parallelism
Streaming Models of Computation

- Many different ways to represent streaming
  - Do senders/receivers block?
  - How much buffering is allowed on channels?
  - Is computation deterministic?
  - Can you avoid deadlock?

- Three common models:
  1. Kahn Process Networks
  2. Synchronous Dataflow
  3. Communicating Sequential Processes
# Streaming Models of Computation

<table>
<thead>
<tr>
<th></th>
<th>Communication Pattern</th>
<th>Buffering</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kahn process networks (KPN)</td>
<td>Data-dependent, but deterministic</td>
<td>Conceptually unbounded</td>
<td>- UNIX pipes&lt;br&gt;- Ambric (startup)</td>
</tr>
<tr>
<td>Synchronous dataflow (SDF)</td>
<td>Static</td>
<td>Fixed by compiler</td>
<td>- Static scheduling&lt;br&gt;- Deadlock freedom</td>
</tr>
<tr>
<td>Communicating Sequential Processes (CSP)</td>
<td>Data-dependent, allows non-determinism</td>
<td>None (Rendesvouz)</td>
<td>- Rich synchronization primitives&lt;br&gt;- Occam language</td>
</tr>
</tbody>
</table>

**Diagram:**
- \( SDF \)
- \( KPN \)
- \( CSP \)

*space of program behaviors*
The StreamIt Language

- A high-level, architecture-independent language for streaming applications
  - Improves programmer productivity (vs. Java, C)
  - Offers scalable performance on multicore

- Based on synchronous dataflow, with dynamic extensions
  - Compiler determines execution order of filters
  - Many aggressive optimizations possible
The StreamIt Project

- **Applications**
  - DES and Serpent [PLDI 05]
  - MPEG-2 [IPDPS 06]
  - SAR, DSP benchmarks, JPEG, …

- **Programmability**
  - StreamIt Language (CC 02)
  - Teleport Messaging (PPOPP 05)
  - Programming Environment in Eclipse (P-PHEC 05)

- **Domain Specific Optimizations**
  - Linear Analysis and Optimization (PLDI 03)
  - Optimizations for bit streaming (PLDI 05)
  - Linear State Space Analysis (CASES 05)

- **Architecture Specific Optimizations**
  - Compiling for Communication-Exposed Architectures (ASPLOS 02)
  - Phased Scheduling (LCTES 03)
  - Cache Aware Optimization (LCTES 05)
  - Load-Balanced Rendering (Graphics Hardware 05)
Example: A Simple Counter

```c
void->void pipeline Counter() {
    add IntSource();
    add IntPrinter();
}

void->int filter IntSource() {
    int x;
    init { x = 0; }
    work push 1 { push (x++); }
}

int->void filter IntPrinter() {
    work pop 1 { print(pop()); }
}
```

Counter

IntSource

IntPrinter

% strc Counter.str -o counter
% ./counter -i 4
0
1
2
3

Bill Thies, MIT.
Representing Streams

- Conventional wisdom: streams are graphs
  - Graphs have no simple textual representation
  - Graphs are difficult to analyze and optimize
- Insight: stream programs have structure

unstructured

structured
Structured Streams

- Each structure is single-input, single-output
- Hierarchical and composable

Bill Thies, MIT.
float->float filter LowPassFilter (int N, float freq) {
    float[N] weights;

    init {
        weights = calcWeights(freq);
    }

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

Bill Thies, MIT.
void FIR(
    int* src,
    int* dest,
    int* srcIndex,
    int* destIndex,
    int srcBufferSize,
    int destBufferSize,
    int N) {

    float result = 0.0;
    for (int i = 0; i < N; i++) {
        result += weights[i] * src[(*srcIndex + i) % srcBufferSize];
    }
    dest[(*destIndex) % destBufferSize] = result;
    *srcIndex = (*srcIndex + 1) % srcBufferSize;
    *destIndex = (*destIndex + 1) % destBufferSize;
}

● FIR functionality obscured by buffer management details

● Programmer must commit to a particular buffer implementation strategy
Pipeline Example: Band Pass Filter

float → float pipeline BandPassFilter (int N,
  float low,
  float high) {
    add LowPassFilter(N, low);
    add HighPassFilter(N, high);
}

Bill Thies, MIT.
float→float **pipeline** Equalizer (int N,  
    float lo,  
    float hi) {

    **add** splitjoin {

        split duplicate;
        for (int i=0; i<N; i++)
            **add** BandPassFilter(64, lo + i*(hi - lo)/N);
        **join** roundrobin(1);
    }

    **add** Adder(N);

}
void->void pipeline FMRadio(int N, float lo, float hi) {
    add AtoD();
    add FMDemod();

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

    add Adder();
    add Speaker();
}

Bill Thies, MIT.
The Beauty of Streaming

“Some programs are elegant, some are exquisite, some are sparkling. My claim is that it is possible to write grand programs, noble programs, truly magnificent ones!”

— Don Knuth, ACM Turing Award Lecture

Image removed due to copyright restrictions.
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(N)

join roundrobin(N)
SplitJoins are Beautiful

split duplicate  

split roundrobin(N)  

join roundrobin(N)
SplitJoins are Beautiful

split duplicate

split roundrobin(N)

join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(N)  join roundrobin(N)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate      split roundrobin(1)      join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

- split duplicate
- split roundrobin(1)
- join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate  

split roundrobin(1)  

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate  
split roundrobin(1)  
join roundrobin(1)
SplitJoins are Beautiful

split duplicate    split roundrobin(1)    join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(1)

join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate          split roundrobin(1)          join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(1)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1)
SplitJoins are Beautiful
SplitJoins are Beautiful

split duplicate  split roundrobin(2)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(2)  join roundrobin(1)
SplitJoins are Beautiful

split duplicate  split roundrobin(2)  join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1)
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1,2,3)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1,2,3)
SplitJoins are Beautiful

- split duplicate
- split roundrobin(2)
- join roundrobin(1,2,3)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1,2,3)

Bill Thies, MIT.
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1,2,3)
SplitJoins are Beautiful

split duplicate  

split roundrobin(2)

join roundrobin(1,2,3)
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1,2,3)
SplitJoins are Beautiful

split duplicate

split roundrobin(2)

join roundrobin(1, 2, 3)
Matrix Transpose

\[ \begin{array}{c}
\{ N \} \\
M \\
\downarrow \\
\{ \text{Transpose} \} \\
M \\
\downarrow \\
\{ N \}
\end{array} \]
Matrix Transpose

Bill Thies, MIT.
Matrix Transpose

Bill Thies, MIT.
Matrix Transpose

\[ N \times M \]

\[ \text{roundrobin}(M) \]

\[ \text{roundrobin}(1) \]

Bill Thies, MIT.
Matrix Transpose

roundrobin(1)

roundrobin(M)
Matrix Transpose

```
float->float splitjoin Transpose (int M, int N) {
    split roundrobin(1);
    for (int i = 0; i < N; i++) {
        add Identity<float>;
    }
    join roundrobin(M);
}
```
Bit-reversed ordering

- Many FFT algorithms require a bit-reversal stage
- If item is at index $n$ (with binary digits $b_0 \ b_1 \ldots \ b_k$), then it is transferred to reversed index $b_k \ldots b_1 b_0$
- For 3-digit binary numbers:

  00001111
  00110011
  01010101

Bill Thies, MIT.
Bit-reversed ordering

- Many FFT algorithms require a bit-reversal stage
- If item is at index $n$ (with binary digits $b_0 \ b_1 \ldots \ b_k$), then it is transferred to reversed index $b_k \ldots \ b_1 \ b_0$
- For 3-digit binary numbers:

```
00001111
00110011
01010101
```

```
00001111
00110011
01010101
```
Bit-reversed ordering

- Many FFT algorithms require a bit-reversal stage
- If item is at index $n$ (with binary digits $b_0 \, b_1 \ldots \, b_k$), then it is transferred to reversed index $b_k \ldots \, b_1 \, b_0$
- For 3-digit binary numbers:

```
00001111
00110011
01010101
```

![Diagram of bit-reversal process](image)
complex->complex pipeline BitReverse (int N) {
    if (N==2) {
        add Identity<complex>;
    } else {
        add splitjoin {
            split roundrobin(1);
            add BitReverse(N/2);
            add BitReverse(N/2);
            join roundrobin(N/2);
        }
    }
}
int->int pipeline MergeSort (int N) {
    if (N==2) {
        add Sort(N);
    } else {
        add splitjoin {
            split roundrobin(N/2);
            add MergeSort(N/2);
            add MergeSort(N/2);
            join roundrobin(N/2);
        }
    }
    add Merge(N);
}
N-Element Merge Sort (3-level)

Bill Thies, MIT.

6.189 IAP 2007 MIT
Bitonic Sort

Courtesy of William Thies.
Used with permission.
FFT

Courtesy of William Thies. Used with permission.
Filterbank

Courtesy of William Thies.
Used with permission.
FM Radio with Equalizer

Courtesy of William Thies.
Used with permission.
Radar-Array Front End

Courtesy of William Thies.
Used with permission.
Case Study: MPEG-2 Decoder in StreamIt
MPEG-2 Decoder in StreamIt

MPEG bit stream

VLD

quantization coefficients

macroblocks, motion vectors

split roundrobin(N*B, V);

add VLD(QC, PT1, PT2);

add splitjoin {

split roundrobin(N*B, V);

add pipeline {
    add ZigZag(B);
    add IQuantization(B) to QC;
    add IDCT(B);
    add Saturation(B);
}

add pipeline {
    add MotionVectorDecode();
    add Repeat(V, N);
}

add pipeline {
    add MotionCompensation(4*(B+V)) to PT1;
    for (int i = 0; i < 2; i++) {
        add pipeline {
            add MotionCompensation(B+V) to PT1;
            add ChannelUpsample(B);
        }
    }
}

join roundrobin(B, V);

add splitjoin {
    split roundrobin(4*(B+V), B+V, B+V);

add MotionCompensation(4*(B+V)) to PT1;
for (int i = 0; i < 2; i++) {
    add pipeline {
        add MotionCompensation(B+V) to PT1;
        add ChannelUpsample(B);
    }
}

join roundrobin(1, 1, 1);

add PictureReorder(3*W*H) to PT2;

add ColorSpaceConversion(3*W*H);
Teleport Messaging in MPEG-2

Bill Thies, MIT.
Messaging Equivalent in C

The MPEG Bitstream

File Parsing

Decode Picture

Decode Macroblock

Decode Block

ZigZagUnordering

Inverse Quantization

Motion Compensation

Saturate

IDCT

Motion Compensation For Single Channel

Frame Reordering

Output Video

Bill Thies, MIT.
MPEG-2 Implementation

- Fully-functional MPEG-2 decoder and encoder
- Developed by 1 programmer in 8 weeks
- 2257 lines of code
  - Vs. 3477 lines of C code in MPEG-2 reference
- 48 static streams, 643 instantiated filters
Conclusions

- **StreamIt language preserves program structure**
  - Natural for programmers

- **Parallelism and communication naturally exposed**
  - Compiler managed buffers, and portable parallelization technology

- **StreamIt increases programmer productivity, enables parallel performance**