6.189 IAP 2007

Lecture 8

The StreamIt Language
Languages Have Not Kept Up

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
Why a New Language?

Uniprocessors: C is the common machine language

# of cores

1 2 4 8 16 32 64 128 256 512


Bill Thies, MIT.
Why a New Language?

What is the common machine language for multicores?
# Common Machine Languages

## Uniprocessors:

<table>
<thead>
<tr>
<th>Common Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single flow of control</td>
</tr>
<tr>
<td>Single memory image</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Differences:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Register File</td>
</tr>
<tr>
<td>ISA</td>
</tr>
<tr>
<td>Functional Units</td>
</tr>
</tbody>
</table>

## Multicores:

<table>
<thead>
<tr>
<th>Common Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td>Multiple flows of control</td>
</tr>
<tr>
<td>Multiple local memories</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Differences:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number and capabilities of cores</td>
</tr>
<tr>
<td>Communication Model</td>
</tr>
<tr>
<td>Synchronization Model</td>
</tr>
</tbody>
</table>

Von-Neumann languages represent the common properties and abstract away the differences.

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

| Kahn process networks (KPN) | Data-dependent, but deterministic | Conceptually unbounded | - UNIX pipes  
- Ambric (startup) |
|---------------------------|----------------------------------|-----------------------|------------------|
| Synchronous dataflow (SDF) | Static                           | Fixed by compiler     | - Static scheduling  
- Deadlock freedom |
| Communicating Sequential Processes (CSP) | Data-dependent, allows non-determinism | None (Rendevouz) | - Rich synchronization primitives  
- Occam language |

**Notes**

- **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 multicores

- 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()); }
}

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

- filter
- pipeline: may be any StreamIt language construct
- splitjoin: parallel computation
- feedback loop
Filter Example: Low Pass Filter

float->float \textbf{filter} \texttt{LowPassFilter} (int \textit{N}, float \textit{freq}) {
  float[\textit{N}] \texttt{weights};

  \textbf{init} {
    \texttt{weights} = \texttt{calcWeights}(\textit{freq});
  }

  \textbf{work peek} \textit{N} \textbf{push} 1 \textbf{pop} 1 {
    float \texttt{result} = 0;
    \texttt{for} (int \texttt{i}=0; \texttt{i}<\texttt{weights.length}; \texttt{i}++) {
      \texttt{result} += \texttt{weights}[\texttt{i}] \ast \texttt{peek}(\texttt{i});
    }
    \texttt{push}(\texttt{result});
    \texttt{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] = result;
    *srcIndex = (*srcIndex + 1) % srcBufferSize;
    *destIndex = (*destIndex + 1) % destBufferSize;
}
Pipeline Example: Band Pass Filter

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

Transpose

Bill Thies, MIT.
Matrix Transpose
Matrix Transpose

roundrobin(?)

roundrobin(?)

Bill Thies, MIT.
Matrix Transpose

roundrobin(M)

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);
}

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
01100011
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:
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:
```
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);
        }
    }
}
```
N-Element Merge Sort

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)
Bitonic Sort
FFT
Block Matrix Multiply
Filterbank
FM Radio with Equalizer
Radar-Array Front End
MP3 Decoder
Case Study:
MPEG-2 Decoder in StreamIt
MPEG-2 Decoder in StreamIt

picture type

MPEG bit stream

quantization coefficients

macroblocks, motion vectors

split joiner

frequency encoded

macroblocks

differentially coded

motion vectors

split roundrobin

ZigZag

<QC>

IQuantization

IDCT

Saturation

motion vectors

join roundrobin

split roundrobin

split joiner

Y

Cb

Cr

Motion Compensation

<PT1>

Motion Compensation

<PT1>

Motion Compensation

<PT1>

Channel Upsample

Channel Upsample

joiner

recovered picture

<PT2>

Picture Reorder

Color Space Conversion

output to player

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);

}

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

picture type

splitter

Motion Compensation

Motion Compensation

Motion Compensation

Channel Upsample

Channel Upsample

joiner

recovered picture

IQ

VLD

Order

MC

MC

MC

MC

Y

Cb

Cr
The MPEG Bitstream

File Parsing

decode

picture

decode

macroblock

decode

block

motion

compensation

saturate

inverse

quantization

motion compensation

for single channel

IDCT

motion compensation

frame reordering

output video

Messaging Equivalent in C
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