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
Why a New Language?

What is the common machine language for multicores?

Bill Thies, MIT.
## 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.

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

*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
Structured Streams

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

Bill Thies, MIT.
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();
    }
}
Low Pass Filter in C

```c
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;
}
```

- FIR functionality obscured by buffer management details
- Programmer must commit to a particular buffer implementation strategy
Pipeline Example: Band Pass Filter

float → float \textbf{pipeline} \texttt{BandPassFilter} (int N, float low, float high) {
    \texttt{add} \texttt{LowPassFilter}(N, low);
    \texttt{add} \texttt{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();
}
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
SplitJoins are Beautiful

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

split duplicate

split roundrobin(N)

join roundrobin(N)
Split Joins 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)
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)**

Bill Thies, MIT.
SplitJoins are Beautiful

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

split duplicate  

split roundrobin(1)  join roundrobin(1)

Bill Thies, MIT.
SplitJoins are Beautiful
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)

Bill Thies, MIT.
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)

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)

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)

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)

Bill Thies, MIT.
SplitJoins are Beautiful

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

Bill Thies, MIT.
SplitJoins are Beautiful
SplitJoins are Beautiful

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

Bill Thies, MIT.
Split Joins are Beautiful

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

Bill Thies, MIT. 57 6.189 IAP 2007 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(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)

Bill Thies, MIT.
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)

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)

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)

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)
Matrix Transpose

![Diagram of matrix transpose]

Bill Thies, MIT.
Matrix Transpose
Matrix Transpose
Matrix Transpose

roundrobin(1)
roundrobin(M)

Bill Thies, MIT.
Matrix Transpose

$\begin{bmatrix} N \\ M \end{bmatrix}$

roundrobin(1)

$\begin{bmatrix} roundrobin(N) \\ roundrobin(M) \end{bmatrix}$

Bill Thies, MIT.
Matrix Transpose

```cpp
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
```

![Diagram showing bit-reversal process]
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:

  \[
  \begin{align*}
  &00001111 \\
  &00110011 \\
  &01010101 \\
  &\downarrow \quad \downarrow \\
  &00001111 \\
  &00110011 \\
  &01010101 \\
  &\downarrow \\
  &RR(w_2) \\
  &\leftarrow \quad \rightarrow \\
  \end{align*}
  \]
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
```

```
RR(1)
RR(1)
RR(2)
RR(2)
RR(4)
```
Bit-reversed ordering

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)

N/2

N/4

N/8

Sort

Sort

Sort

Merge

Merge

Merge

Merge

N/2

N/4

N/8

Sort

Sort

Sort

Merge

Merge

Merge

Merge

N

N/2

N/4

N/8

Sort

Sort

Sort

Merge

Merge

Merge

Merge

90

Bill Thies, MIT.

6.189 IAP 2007 MIT
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

MPEG bit stream

- VLD
  - macroblocks, motion vectors
  - quantization coefficients

- split
  - frequency encoded macroblocks
  - differentially coded motion vectors

- ZigZag
- IQuantization
- Motion Vector Decode
- IDCT
- Saturation

- join

- splitter
  - Y Cb Cr

- Motion Compensation
  - reference picture

- Channel Upsample
- join

- splitter
- recovered picture

- Picture Reorder

- Color Space Conversion

output to player

Bill Thies, MIT.
Teleport Messaging in MPEG-2

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