Stream Programming: Luring Programmers into the Multicore Era

Bill Thies

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Spring 2008
Multicores are Here

# of cores


1 2 4 8 16 32 64 128 256 512
Multicores are Here

Hardware was responsible for improving performance
Multicores are Here

Now, performance burden falls on programmers
Is Parallel Programming a New Problem?

- No! Decades of research targeting multiprocessors
  - Languages, compilers, architectures, tools...

- What is different today?
  1. **Multicores vs. multiprocessors.** Multicores have:
     - New interconnects with non-uniform communication costs
     - Faster on-chip communication than off-chip I/O, memory ops
     - Limited per-core memory availability
  2. **Non-expert programmers**
     - Supercomputers with >2048 processors today: 100 [top500.org]
     - Machines with >2048 cores in 2020: >100 million [ITU, Moore]
  3. **Application trends**
     - Embedded: 2.7 billion cell phones vs 850 million PCs [ITU 2006]
     - Data-centric: YouTube streams 200 TB of video daily
Streaming Application Domain

• 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
Streaming Application Domain

- **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

- **Properties of stream programs**
  - Regular and repeating computation
  - Independent filters with explicit communication
  - Data items have short lifetimes
Brief History of Streaming


Models of Computation
- Petri Nets
- Kahn Proc. Networks
- Comp. Graphs
- Communicating Sequential Processes
- Synchronous Dataflow

Modeling Environments
- Ptolemy
- Matlab/Simulink
- Gabriel
- Grape-II
- etc.

Languages / Compilers
- Lucid
- Id
- Sisal
- Erlang
- Occam
- LUSTRE
- pH
- C
- lazy
- VAL
- Occam
- LUSTRE
- pH
Brief History of Streaming

Models of Computation
- Petri Nets
- Kahn Proc. Networks
- Synchronous Dataflow
- Comp. Graphs
- Communicating Sequential Processes

Modeling Environments
- Ptolemy
- Matlab/Simulink
- Gabriel
- Grape-II
- etc.

Languages / Compilers
- Lucid
- Id
- Sisal
- Erlang
- Esterel
- C
- lazy
- VAL
- Occam
- LUSTRE
- pH

Strengths
- Elegance
- Generality

Weaknesses
- Unsuitable for static analysis
- Cannot leverage deep results from DSP / modeling community
# Brief History of Streaming

## Models of Computation
- Petri Nets
- Kahn Proc. Networks
- Synchronous Dataflow
- Comp. Graphs
- Communicating Sequential Processes

## Modeling Environments
- Ptolemy
- Matlab/Simulink
- Gabriel
- Grape-II
- etc.

## Languages / Compilers
- Lucid
- Id
- Sisal
- Erlang
- Esterel
- C
- lazy
- VAL
- Occam
- LUSTRE
- pH
- StreamIt
- Brook
- Cg
- StreamC

## Strengths
- Elegance
- Generality

## Weaknesses
- Unsuitable for static analysis
- Cannot leverage deep results from DSP / modeling community

---

**“Stream Programming”**
StreamIt: A Language and Compiler for Stream Programs

- **Key idea:** design language that enables static analysis

- **Goals:**
  1. Expose and exploit the parallelism in stream programs
  2. Improve programmer productivity in the streaming domain

- **Project contributions:**
  - Language design for streaming [CC'02, CAN'02, PPoPP'05, IJPP'05]
  - Automatic parallelization [ASPLOS'02, G.Hardware'05, ASPLOS'06]
  - Domain-specific optimizations [PLDI'03, CASES'05, TechRep'07]
  - Cache-aware scheduling [LCTES'03, LCTES'05]
  - Extracting streams from legacy code [MICRO'07]
  - User + application studies [PLDI'05, P-PHEC'05, IPDPS'06]

- 7 years, 25 people, 300 KLOC
- 700 external downloads, 5 external publications
StreamIt: A Language and Compiler for Stream Programs

- **Key idea**: design language that enables static analysis

- **Goals**:
  1. Expose and exploit the parallelism in stream programs
  2. Improve programmer productivity in the streaming domain

- **I contributed to**:
  - Language design for streaming [CC'02, CAN'02, PPoPP'05, IJPP'05]
  - Automatic parallelization [ASPLOS'02, G.Hardware'05, ASPLOS'06]
  - Domain-specific optimizations [PLDI'03, CASES'05, TechRep'07]
  - Cache-aware scheduling [LCTES'03, LCTES'05]
  - Extracting streams from legacy code [MICRO'07]
  - User + application studies [PLDI'05, P-PHEC'05, IPDPS'06]

- 7 years, 25 people, 300 KLOC
- 700 external downloads, 5 external publications
StreamIt: A Language and Compiler for Stream Programs

- Key idea: design language that enables static analysis

- Goals:
  1. Expose and exploit the parallelism in stream programs
  2. Improve programmer productivity in the streaming domain

- This talk:
  - Language design for streaming [CC'02, CAN'02, PPoPP'05, IJPP'05]
  - Automatic parallelization [ASPLOS'02, G.Hardware'05, ASPLOS'06]
  - Domain-specific optimizations [PLDI'03, CASES'05, TechRep'07]
  - Cache-aware scheduling [LCTES'03, LCTES'05]
  - Extracting streams from legacy code [MICRO'07]
  - User + application studies [PLDI'05, P-PHEC'05, IPDPS'06]

- 7 years, 25 people, 300 KLOC
- 700 external downloads, 5 external publications
Part 1: Language Design

*William Thies, Michal Karczmarek, Saman Amarasinghe (CC’02)*

*William Thies, Michal Karczmarek, Janis Sermulins, Rodric Rabbah, Saman Amarasinghe (PPoPP’05)*
StreamIt Language Basics

• High-level, architecture-independent language
  – Backend support for uniprocessors, multicores (Raw, SMP), cluster of workstations

• Model of computation: synchronous dataflow
  – Program is a graph of independent filters
  – Filters have an atomic execution step with known input / output rates
  – Compiler is responsible for scheduling and buffer management

• Extensions to synchronous dataflow
  – Dynamic I/O rates
  – Support for sliding window operations
  – Teleport messaging [PPoPP’05]
Representing Streams

• Conventional wisdom: stream programs 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
Radar-Array Front End
MP3 Decoder
Bitonic Sort
FM Radio with Equalizer
Ground Moving Target Indicator (GMTI)

99 filters
3566 filter instances
Example Syntax: FMRadio

```c
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();
}
```
StreamIt Application Suite

- Software radio
- Frequency hopping radio
- Acoustic beam former
- Vocoder
- FFTs and DCTs
- JPEG Encoder/Decoder
- MPEG-2 Encoder/Decoder
- MPEG-4 (fragments)
- Sorting algorithms
- GMTI (Ground Moving Target Indicator)
- DES and Serpent crypto algorithms
- SSCA#3 (HPCS scalable benchmark for synthetic aperture radar)
- Mosaic imaging using RANSAC algorithm

Total size: 60,000 lines of code
Control Messages

- Occasionally, low-bandwidth control messages are sent between actors.
- Often demands precise timing:
  - Communications: adjust protocol, amplification, compression
  - Network router: cancel invalid packet
  - Adaptive beamformer: track a target
  - Respond to user input, runtime errors
  - Frequency hopping radio
- Traditional techniques:
  - Direct method call (no timing guarantees)
  - Embed message in stream (opaque, slow)
Idea 2: Teleport Messaging

- Looks like method call, but timed relative to data in the stream

```java
TargetFilter x;
if newProtocol(p) {
    x.setProtocol(p) @ 2;
}
```

```java
void setProtocol(int p) {
    reconfig(p);
}
```

- Exposes dependences to compiler
- Simple and precise for user
  - Adjustable latency
  - Can send upstream or downstream
Part 2: Automatic Parallelization

Michael I. Gordon, William Thies, Saman Amarasinghe (ASPLOS’06)

Streaming is an Implicitly Parallel Model

- Programmer thinks about functionality, not parallelism
- More explicit models may…
  - Require knowledge of target [MPI] [cG]
  - Require parallelism annotations [OpenMP] [HPF] [Cilk] [Intel TBB]
- Novelty over other implicit models?
  [Erlang] [MapReduce] [Sequoia] [pH] [Occam] [Sisal] [Id] [VAL] [LUSTRE] [HAL] [THAL] [SALSA] [Rosette] [ABCL] [APL] [ZPL] [NESL] […]
  \[ Exploiting streaming structure for robust performance \]
Parallelism in Stream Programs

Task parallelism
- Analogous to thread (fork/join) parallelism
Parallelism in Stream Programs

Task parallelism
- Analogous to thread (fork/join) parallelism

Data parallelism
- Analogous to DOALL loops

Pipeline parallelism
- Analogous to ILP that is exploited in hardware
Evaluation: Fine-Grained Data Parallelism

Throughput Normalized to Single Core StreamIt

- BitonicSort
- Channel/Vocoder
- DCT
- DES
- FFT
- Filterbank
- FMRadio
- Serpent
- TDE
- MPEG2-subset
- Vocoder
- Radar
- Geometric Mean

Raw Microprocessor
- 16 inorder, single-issue cores with D$ and I$
- 16 memory banks, each bank with DMA
- Cycle accurate simulator
Evaluation: Fine-Grained Data Parallelism

Good Parallelism! Too Much Synchronization!

Throughput Normalized to Single Core StreamIt

Fine-Grained Data

BitonicSort  ChannelVocoder  DCT  DES  FFT  Fillerbank  FMRadio  Serpent  TDE  MPEG2-subset  Vocoder  Radar  Geometric Mean
Coarsening the Granularity
Coarsening the Granularity

```
Coarsening the Granularity

Splitter

BandPass Compress Process Expand

BandPass Compress Process Expand

BandStop

BandStop

Joiner

Adder
```
Coarsening the Granularity
Coarsening the Granularity
Evaluation: Coarse-Grained Data Parallelism

Good Parallelism! Low Synchronization!
Simplified Vocoder

Target a 4-core machine
Data Parallelize

Target a 4-core machine
Data + Task Parallel Execution

Target a 4-core machine
We Can Do Better

Target a 4-core machine
Coarse-Grained Software Pipelining
Evaluation: Coarse-Grained
Task + Data + Software Pipelining

Throughput Normalized to Single Core Streamit

- Fine-Grained Data
- Coarse-Grained Task + Data
- Coarse-Grained Task + Data + Software Pipeline

Evaluation:
- Coarse-Grained Task + Data + Software Pipelining
Evaluation: Coarse-Grained Task + Data + Software Pipelining

Throughput Normalized to Single Core Stream

Best Parallelism!
Lowest Synchronization!
Parallelism: Take Away

• Stream programs have abundant parallelism
  – However, parallelism is obfuscated in language like C

• Stream languages enable new & effective mapping
  – In C, analogous transformations impossibly complex
  – In StreamC or Brook, similar transformations possible
    [Khailany et al., IEEE Micro’01] [Buck et al., SIGGRAPH’04] [Das et al., PACT’06] […]

• Results should extend to other multicores
  – Parameters: local memory, comm.-to-comp. cost
  – Preliminary results on Cell are promising [Zhang, dasCMP’07]
Part 3: Domain-Specific Optimizations

Andrew Lamb, William Thies, Saman Amarasinghe (PLDI’03)
Sitij Agrawal, William Thies, Saman Amarasinghe (CASES’05)
DSP Optimization Process

- Given specification of algorithm, minimize the computation cost
DSP Optimization Process

• Given specification of algorithm, minimize the computation cost
  – Currently done by hand (MATLAB)
DSP Optimization Process

• Given specification of algorithm, minimize the computation cost
  – Currently done by hand (MATLAB)

• Can compiler replace DSP expert?
  – Library generators limited [Spiral][FFTW][ATLAS]
  – Enable unified development environment
Focus: Linear State Space Filters

• Properties:
  – Outputs are linear function of inputs and states
  – New states are linear function of inputs and states

• Most common target of DSP optimizations
  – FIR / IIR filters
  – Linear difference equations
  – Upsamplers / downsamplers
  – DCTs

\[
\begin{align*}
  x' &= Ax + Bu \\
  y &= Cx + Du
\end{align*}
\]
Focus: Linear State Space Filters

\[ u \]

\[ x' = Ax + Bu \]

\[ y = Cx + Du \]
**Focus: Linear Filters**

```
float->float filter Scale {
    work push 2 pop 1 {
        float u = pop();
        push(u);
        push(2*u);
    }
}
```
float->float filter Scale {
  work push 2 pop 1 {
    float u = pop();
    push(u);
    push(2*u);
  }
}
Combining Adjacent Filters

\[ y = Du \]
\[ z = Ey \]
\[ z = EDu \]
\[ z = Gu \]
Combination Example

6 mults output

Filter 1

Filter 2

\[ E = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \]

\[ D = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \]

Combined Filter

\[ G = \begin{bmatrix} 32 \end{bmatrix} \]

1 mults output
The General Case

• If matrix dimensions mis-match? Matrix expansion:

Original

\[
[D]
\]

\[\text{pop} = \sigma\]

Expanded

\[
\begin{bmatrix}
[D] & [D] \\
[D] & [D]
\end{bmatrix}
\]
The General Case

- If matrix dimensions mis-match? Matrix expansion:

\[
\begin{align*}
A^e &= A^n A_{pre} \\
B^e &= \begin{bmatrix}
A^n B_{pre} & A^{n-1}B & A^{n-2}B & \ldots & B \\
\end{bmatrix} \\
C^e &= \begin{bmatrix}
CA_{pre} \\
CA A_{pre} \\
\vdots \\
CA^{n-1}A_{pre}
\end{bmatrix} \\
D^e &= \begin{bmatrix}
CB_{pre} & D & 0 & 0 & \ldots & 0 & 0 \\
CAB_{pre} & CB & D & 0 & \ldots & 0 & 0 \\
CA^2B_{pre} & CAB & CB & D & \ldots & 0 & 0 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
CA^{n-1}B_{pre} & CA^{n-2}B & CA^{n-3}B & CA^{n-3}B & \ldots & CB & D
\end{bmatrix}
\end{align*}
\]
## The General Case

### Pipelines

\[
A = \begin{bmatrix}
A_1 & 0 \\
B_2C_1 & A_2
\end{bmatrix} \quad A_{\text{pre}} = \begin{bmatrix}
A_1^e & 0 \\
B_{\text{pre}2}C_1^e & A_{\text{pre}2}
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
B_1 \\
B_2D_1
\end{bmatrix} 
B_{\text{pre}} = \begin{bmatrix}
B_1^e \\
B_{\text{pre}2}D_1^e
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
D_2C_1 & C_2
\end{bmatrix}
\]

\[
D = D_2D_1
\]

\[
\text{initVec} = \begin{bmatrix}
\text{initVec}_1 \\
\text{initVec}_2
\end{bmatrix}
\]

### Feedback Loops

\[
x_1' = A_1x_1 + B_1u_1 = A_1x_1 + B_1y = A_1x_1 + B_1(C_2x_2 + D_{2,1}u + D_2C_3x_3)
\]
\[
= A_1x_1 + B_1C_2x_2 + B_1D_{2,1}u + B_1D_2C_3x_3
\]

\[
x_2' = A_2x_2 + B_2u_2 = A_2x_2 + B_2u + B_{2,2}y_3 = A_2x_2 + B_{2,1}u + B_2C_3x_3
\]

\[
y_2' = C_2x_2 + D_2u_2 = C_2x_2 + D_{2,1}u + D_{2,2}y_3 = C_2x_2 + D_{2,1}u + D_2C_3x_3
\]

\[
x_3' = A_3x_3 + B_3u_3 = A_3x_3 + B_3y_1 = A_3x_3 + B_3(C_1x_1 + D_1u_1)
\]
\[
= A_3x_3 + B_3(C_1x_1 + D_1y) = A_3x_3 + B_3(C_1x_1 + D_1(C_2x_2 + D_{2,1}u + D_{2,2}C_3x_3))
\]
\[
= A_3x_3 + B_3C_1x_1 + B_3D_1C_2x_2 + B_3D_1D_{2,1}u + B_3D_1D_{2,2}C_3x_3
\]
The General Case

**Splitjoins**

\[
A = \begin{bmatrix}
A_s & 0 & 0 & \ldots & 0 \\
A_{1rs} & A_{1rr} & 0 & \ldots & 0 \\
A_{2rs} & 0 & A_{2rr} & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
A_{krr} & 0 & 0 & \ldots & A_{krs}
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
B_s \\
B_{1r} \\
B_{2r} \\
\vdots \\
B_{kr}
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
C_{1s1} & C_{1r1} & 0 & \ldots & 0 \\
C_{2s1} & C_{2r1} & 0 & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{ks1} & 0 & 0 & \ldots & C_{kr1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{sk1} & 0 & 0 & \ldots & C_{krk}
\end{bmatrix}
\]

\[
D = \begin{bmatrix}
D_{11} \\
D_{21} \\
\vdots \\
D_{k1} \\
D_{1k} \\
D_{2k} \\
\vdots \\
D_{kk}
\end{bmatrix}
\]

\[
C_i = \begin{bmatrix}
C_{is1} & C_{ir1} \\
C_{is2} & C_{ir2} \\
\vdots & \vdots \\
C_{iexecutions} & C_{irexexecutions}
\end{bmatrix}
\]

\[
D_i = \begin{bmatrix}
D_{i1} \\
D_{i2} \\
\vdots \\
D_{iexecutions}
\end{bmatrix}
\]

\[
A_{pre} = \begin{bmatrix}
0 & 0 & 0 & \ldots & 0 \\
0 & A_{pre1rr} & 0 & \ldots & 0 \\
0 & 0 & A_{pre2rr} & \ldots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \ldots & A_{prekrr}
\end{bmatrix}
\]

\[
B_{pre} = \begin{bmatrix}
B_{pres} \\
B_{pre1r} \\
B_{pre2r} \\
\vdots \\
B_{prekr}
\end{bmatrix}
\]

\[
\text{initVec} = \begin{bmatrix}
\delta \\
\text{initVec}_{1r} \\
\text{initVec}_{2r} \\
\vdots \\
\text{initVec}_{kr}
\end{bmatrix}
\]
Floating-Point Operations Reduction

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Flops Removed (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIR</td>
<td>80%</td>
</tr>
<tr>
<td>RateConvert</td>
<td>80%</td>
</tr>
<tr>
<td>TargetDetect</td>
<td>80%</td>
</tr>
<tr>
<td>FMRadio</td>
<td>80%</td>
</tr>
<tr>
<td>Radar</td>
<td>80%</td>
</tr>
<tr>
<td>FilterBank</td>
<td>80%</td>
</tr>
<tr>
<td>Vocoder</td>
<td>80%</td>
</tr>
<tr>
<td>Oversample</td>
<td>80%</td>
</tr>
<tr>
<td>DTOA</td>
<td>0.3%</td>
</tr>
</tbody>
</table>

Floating-Point Operations Reduction
Floating-Point Operations Reduction

![Bar chart showing floating-point operations reduction for different benchmarks. The benchmarks include FIR, RateConvert, TargetDetect, FMRadio, Radar, FilterBank, Vocoder, Oversample, and DTOA. The reduction is shown as a percentage for both linear and frequency domains. The chart indicates a significant reduction in operations for most benchmarks, with some showing a 140% increase.]
Radar (Transformation Selection)
Radar (Transformation Selection)
Radar

Maximal Combination and Shifting to Frequency Domain

Using Transformation Selection

2.4 times as many FLOPS

half as many FLOPS
Execution Speedup

On a Pentium IV
Execution Speedup

Additional transformations:
1. Eliminating redundant states
2. Eliminating parameters
   (non-zero, non-unary coefficients)
3. Translation to the compressed domain

On a Pentium IV
StreamIt: Lessons Learned

- In practice, I/O rates of filters are often matched [LCTES’03]
  - Over 30 publications study an uncommon case (CD-DAT)

- Multi-phase filters complicate programs, compilers
  - Should maintain simplicity of only one atomic step per filter

- Programmers accidentally introduce mutable filter state

```c
void>int filter SquareWave() {
    work push 2 {
        push(0);
        push(1);
    }
}
```

```c
void>int filter SquareWave() {
    int x = 0;
    work push 1 {
        push(x);
        x = 1 - x;
    }
}
```
Future of StreamIt

- Goal: influence the next big language

**Origins of C++**

- Structural influence
- Feature influence
- Academic origin

Source: B. Stroustrup, *The Design and Evolution of C++*
Research Trajectory

• **Vision:** Make emerging computational substrates universally accessible and useful

1. **Languages, compilers, & tools for multicores**
   – I believe new language / compiler technology can enable scalable and robust performance
   – Next inroads: expose & exploit flexibility in programs

2. **Programmable microfluidics**
   – We have developed programming languages, tools, and flexible new devices for microfluidics
   – Potential to revolutionize biology experimentation

3. **Technologies for the developing world**
   – TEK: enable Internet experience over email account
   – Audio Wiki: publish content from a low-cost phone
   – uBox / uPhone: monitor & improve rural healthcare
Conclusions

• A parallel programming model will succeed only by luring programmers, making them do less, not more

• Stream programming lures programmers with:
  – Elegant programming primitives
  – Domain-specific optimizations

• Meanwhile, streaming is implicitly parallel
  – Robust performance via task, data, & pipeline parallelism

• We believe stream programming will play a key role in enabling a transition to multicore processors

Contributions

– Structured streams
– Teleport messaging
– Unified algorithm for task, data, pipeline parallelism
– Software pipelining of whole procedures
– Algebraic simplification of whole procedures
– Translation from time to frequency
– Selection of best DSP transforms
Acknowledgments

• Project supervisors
  – Prof. Saman Amarasinghe – Dr. Rodric Rabbah

• Contributors to this talk
  – Michael I. Gordon (Ph.D. Candidate) – leads StreamIt backend efforts
  – Andrew A. Lamb (M.Eng) – led linear optimizations
  – Sitij Agrawal (M.Eng) – led statespace optimizations

• Compiler developers
  – Kunal Agrawal – Jasper Lin – Janis Sermulins
  – Allyn Dimock – Michal Karczmarek – Phil Sung
  – Qiuyuan Jimmy Li – David Maze – David Zhang

• Application developers
  – Basier Aziz – Shirley Fung – Ali Meli
  – Matthew Brown – Hank Hoffmann – Satish Ramaswamy
  – Matthew Drake – Chris Leger – Jeremy Wong

• User interface developers
  – Kimberly Kuo – Juan Reyes