Language and Compiler Support for Stream Programs

Bill Thies

Computer Science and Artificial Intelligence Laboratory
Massachusetts Institute of Technology

Thesis Defense
September 11, 2008
Hi Bill,

I have a few UROP opportunities in the RAW project ...

Most of the projects can lead to an MENG thesis and beyond ...
Hi Bill,

I have a few UROP opportunities in the RAW project...

Most of the projects can lead to an MENG thesis and beyond...
Acknowledgments

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

• Contributors to this talk
  – Michael I. Gordon (Ph.D. student) – led development of Raw backend
  – Andrew A. Lamb (M.Eng) – led development of linear optimizations
  – Sitij Agrawal (M.Eng) – led development of statespace optimizations

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

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

• User interface developers
  – Kimberly Kuo
  – Juan Reyes
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
- 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
**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
- 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
- Estel
- 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
Filterbank
Block Matrix Multiply
MP3 Decoder
Bitonic Sort
FM Radio with Equalizer
Ground Moving Target Indicator (GMTI)

99 filters
3566 filter instances
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
void setProtocol(int p) {
    reconfig(p);
}
```

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

- 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
Baseline: Fine-Grained Data Parallelism
Evaluation:
Fine-Grained Data Parallelism

Throughput Normalized to Single Core StreamIt

- BitonicSort
- ChannelVocoder
- 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!
Coarsening the Granularity
Coarsening the Granularity

Splitter

BandPass Compress Process Expand

BandStop

BandPass Compress Process Expand

BandStop

Joiner

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

Good Parallelism! Low Synchronization!
Simplified Vocoder

Data Parallel

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

Cores

Time

Target a 4-core machine
Coarse-Grained Software Pipelining

Prologue

New Steady State
Evaluation: Coarse-Grained Task + Data + Software Pipelining

Throughput Normalized to Single Core StreamIt

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

\[
x' = Ax + Bu
\]

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

\[ x' = Ax + Bu \]

\[ y = Cx + Du \]
float->float filter Scale {
  work push 2 pop 1 {
    float u = pop();
    push(u);
    push(2*u);
  }
}

Linear dataflow analysis

\[ y = Du \]
float->float filter Scale {
    work push 2 pop 1 {
        float u = pop();
        push(u);
        push(2*u);
    }
}

Linear dataflow analysis

\[
\begin{bmatrix}
y_1 \\
y_2
\end{bmatrix} = \begin{bmatrix}
1 \\
2
\end{bmatrix} u
\]
Combining Adjacent Filters

Filter 1

\[ y = Du \]

Filter 2

\[ z = Ey \]

Combined Filter

\[ z = EDu \]

\[ z = Gu \]
Combination Example

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

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

6 mults output

1 mults output

Filter 1

Filter 2

Combined Filter

u → y → z

u → y → z

6 mults

output

output

1 mults

output

G = [32]

\( D = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \)

\( E = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix} \)
The General Case

• If matrix dimensions mis-match? Matrix expansion:

Original

\[
[D]
\]

\( pop = \sigma \)

Expanded

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

- If matrix dimensions mis-match? Matrix expansion:

\[
A^e = A^n A_{\text{pre}}
\]

\[
B^e = \begin{bmatrix}
A^n B_{\text{pre}} & A^{n-1} B & A^{n-2} B & \ldots & B \\
C A_{\text{pre}} & \ldots & \ldots & \ldots & \ldots \\
C A A_{\text{pre}} & \ldots & \ldots & \ldots & \ldots \\
\vdots & \ddots & \ddots & \ddots & \ddots \\
C A^{n-1} A_{\text{pre}} & \ldots & \ldots & \ldots & \ldots \\
\end{bmatrix}
\]

\[
C^e = \begin{bmatrix}
C B_{\text{pre}} & D & 0 & 0 & \ldots & 0 & 0 \\
C A B_{\text{pre}} & C B & D & 0 & \ldots & 0 & 0 \\
C A^2 B_{\text{pre}} & C A B & C B & D & \ldots & 0 & 0 \\
\vdots & \vdots & \ddots & \ddots & \ddots & \ddots & \ddots \\
C A^{n-1} B_{\text{pre}} & C A^{n-2} B & C A^{n-3} B & C A^{n-3} B & \ldots & C B & D \\
\end{bmatrix}
\]
## The General Case

### Pipelines

\[
A = \begin{bmatrix}
A_1 & 0 \\
B_2 C_1 & A_2 \\
\end{bmatrix} \quad A_{\text{pre}} = \begin{bmatrix}
A_1^e & 0 \\
B_{\text{pre}2} C_1^e & A_{\text{pre}2} \\
\end{bmatrix}
\]

\[
B = \begin{bmatrix}
B_1 \\
B_2 D_1 \\
\end{bmatrix} \quad B_{\text{pre}} = \begin{bmatrix}
B_1^e \\
B_{\text{pre}2} D_1^e \\
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
D_2 C_1 & C_2 \\
\end{bmatrix} \quad \text{initVec} = \begin{bmatrix}
\text{initVec}_1 \\
\text{initVec}_2 \\
\end{bmatrix}
\]

\[
D = D_2 D_1
\]

### Feedback Loops

\[
\begin{align*}
\tilde{x}_1 &= A_1 \tilde{x}_1 + B_1 \tilde{u}_1 = A_1 \tilde{x}_1 + B_1 \tilde{y} = A_1 \tilde{x}_1 + B_1 (C_2 \tilde{x}_2 + D_{2.1} \tilde{u} + D_{2.2} C_3 \tilde{x}_3) \\
&= A_1 \tilde{x}_1 + B_1 C_2 \tilde{x}_2 + B_1 D_{2.1} \tilde{u} + B_1 D_{2.2} C_3 \tilde{x}_3 \\
\tilde{x}_2 &= A_2 \tilde{x}_2 + B_2 \tilde{u}_2 = A_2 \tilde{x}_2 + B_{2.1} \tilde{u} + B_{2.2} \tilde{y}_3 = A_2 \tilde{x}_2 + B_{2.1} \tilde{u} + B_{2.2} C_3 \tilde{x}_3 \\
\tilde{y}_2 &= C_2 \tilde{x}_2 + D_2 \tilde{u}_2 = C_2 \tilde{x}_2 + D_{2.1} \tilde{u} + D_{2.2} \tilde{y}_3 = C_2 \tilde{x}_2 + D_{2.1} \tilde{u} + D_{2.2} C_3 \tilde{x}_3 \\
\tilde{x}_3 &= A_3 \tilde{x}_3 + B_3 \tilde{u}_3 = A_3 \tilde{x}_3 + B_3 \tilde{y}_1 = A_3 \tilde{x}_3 + B_3 (C_1 \tilde{x}_1 + D_1 \tilde{u}_1) \\
&= A_3 \tilde{x}_3 + B_3 (C_1 \tilde{x}_1 + D_1 \tilde{y}) = A_3 \tilde{x}_3 + B_3 (C_1 \tilde{x}_1 + D_1 (C_2 \tilde{x}_2 + D_{2.1} \tilde{u} + D_{2.2} C_3 \tilde{x}_3)) \\
&= A_3 \tilde{x}_3 + B_3 C_1 \tilde{x}_1 + B_3 D_1 C_2 \tilde{x}_2 + B_3 D_1 D_{2.1} \tilde{u} + B_3 D_1 D_{2.2} C_3 \tilde{x}_3
\end{align*}
\]
### The General Case

#### Splitjoins

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
</table>
| \[
A = \begin{bmatrix}
A_s & 0 & 0 & \cdots & 0 \\
A_{1rs} & A_{1rr} & 0 & \cdots & 0 \\
A_{2rs} & 0 & A_{2rr} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
A_{krs} & 0 & 0 & \cdots & A_{krr}
\end{bmatrix}
\] | \[
B = \begin{bmatrix}
B_s \\
B_{1r} \\
B_{2r} \\
\vdots \\
B_{kr}
\end{bmatrix}
\] | \[
C = \begin{bmatrix}
C_{1s1} & C_{1r1} & 0 & \cdots & 0 \\
C_{2s1} & C_{2r1} & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{ks1} & 0 & 0 & \cdots & C_{kr1} \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
C_{ksk} & 0 & 0 & \cdots & C_{krk}
\end{bmatrix}
\] | \[
D = \begin{bmatrix}
D_{11} \\
D_{21} \\
\vdots \\
D_{k1} \\
\vdots \\
D_{kk}
\end{bmatrix}
\] |

<table>
<thead>
<tr>
<th>C_i</th>
<th>D_i</th>
</tr>
</thead>
</table>
| \[
C_i = \begin{bmatrix}
C_{i1s1} & C_{i1r1} \\
C_{i2s2} & C_{i2r2} \\
\vdots & \vdots \\
C_{i1s1} & C_{i1r1} \\
\vdots & \vdots \\
C_{i1s1} & C_{i1r1}
\end{bmatrix}
\] | \[
D_i = \begin{bmatrix}
D_{i1} \\
D_{i2} \\
\vdots \\
D_{i1} \\
\vdots \\
D_{i1}
\end{bmatrix}
\] |

<table>
<thead>
<tr>
<th>A_\text{pre}</th>
<th>B_\text{pre}</th>
</tr>
</thead>
</table>
| \[
A_{\text{pre}} = \begin{bmatrix}
0 & 0 & 0 & \cdots & 0 \\
0 & A_{\text{pre1rr}} & 0 & \cdots & 0 \\
0 & 0 & A_{\text{pre2rr}} & \cdots & 0 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
0 & 0 & 0 & \cdots & A_{\text{prekrr}}
\end{bmatrix}
\] | \[
B_{\text{pre}} = \begin{bmatrix}
B_{\text{pre1s}} \\
B_{\text{pre1r}} \\
B_{\text{pre2r}} \\
\vdots \\
B_{\text{prekr}}
\end{bmatrix}
\] |

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

![Bar chart showing floating-point operations reduction across various benchmarks. The chart includes benchmarks such as FIR, RateConvert, TargetDetect, FMRadio, Radar, FilterBank, Vocoder, Oversample, and DTOA. The y-axis represents the percentage of flops removed, ranging from 100% to 0%. The x-axis lists the benchmarks. Most benchmarks show significant flops removed, with a few showing minimal or zero reduction.](image-url)
Floating-Point Operations Reduction

- FIR
- RateConvert
- TargetDetect
- FM Radio
- Radar
- Filter Bank
- Vocoder
- Oversample
- DTOA

Flops Removed (%)

Benchmarks:
- Linear
- Freq

Graph showing percentage of floating-point operations removed for various benchmarks.
Radar (Transformation Selection)
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
Floating Point Operations Reduction

FIR RateConvert TargetDetect FMRadio Radar FilterBank Vocoder Oversample DTOA

-140% 0.3% -20% -40%

Benchmark

Flops Removed (%)

linear freq autosel
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

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