Language, Compiler and Development Support for Stream Computing

Rodric M. Rabbah
Massachusetts Institute of Technology
Joint work with Michael Gordon, Michal Karczmarek, Andrew Lamb, Jasper Lin, William Thies, Kimberly Kuo, Juan C. Reyes, David Maze, and Saman Amarasinghe

http://cag.csail.mit.edu/streamit

HPL Workshop
May 2004

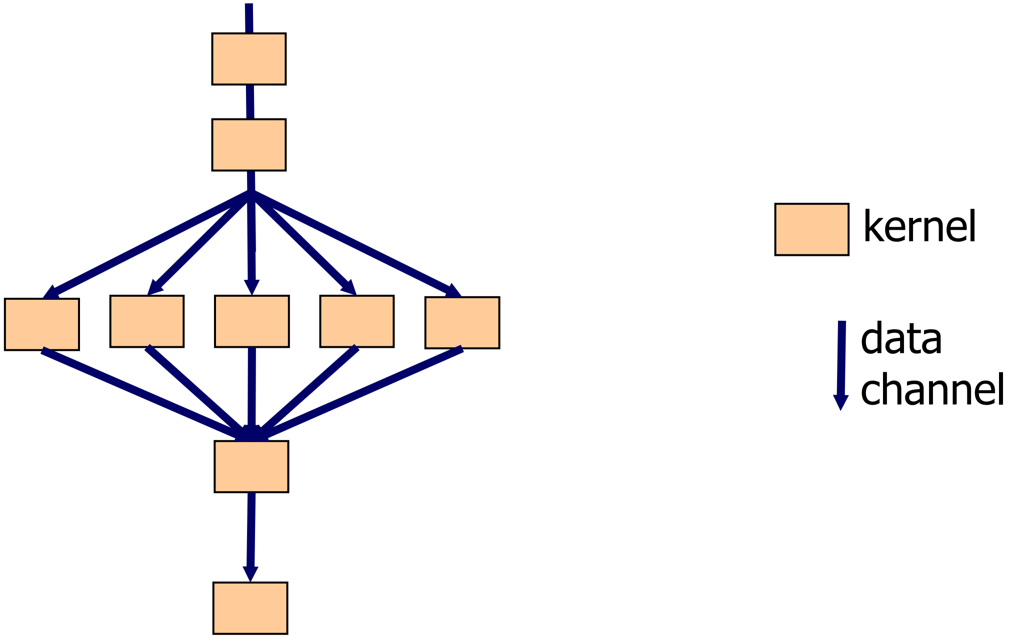
Domain of Streaming Computing

- Increasingly prevalent computing domain with applications in
  - Embedded systems
    - Cell phones, handheld computers, DSPs
  - Desktop workstations
    - Streaming media, real-time encryption, software radio, graphics
  - High-performance servers
    - Software routers, cell phone base stations, radar tracking, HDTV editing consoles, databases

- Predominant data streams include audio, video, and data

What is Stream Computing?

- "A model that uses sequences of data and computation kernels to expose and exploit concurrency and locality for efficiency."
  - Workshop on Streaming Systems, Summer 2003
    http://cag.csail.mit.edu/wss03



Properties of Streaming Programs

- Process large (possibly infinite) amounts of data
  - Data have limited lifetime and undergoes little processing
- Processing consists of a series of data transformations
  - Filter is the basic unit for data transformation
    - Input data  Output data
    - Filters are independent and self-contained
- "Regular" computation patterns
  - Flow of data between filters is mostly constant
  - Many opportunities for compiler optimizations
- Occasional changes in program control (messaging)

HPCS Goals in the Streaming Context

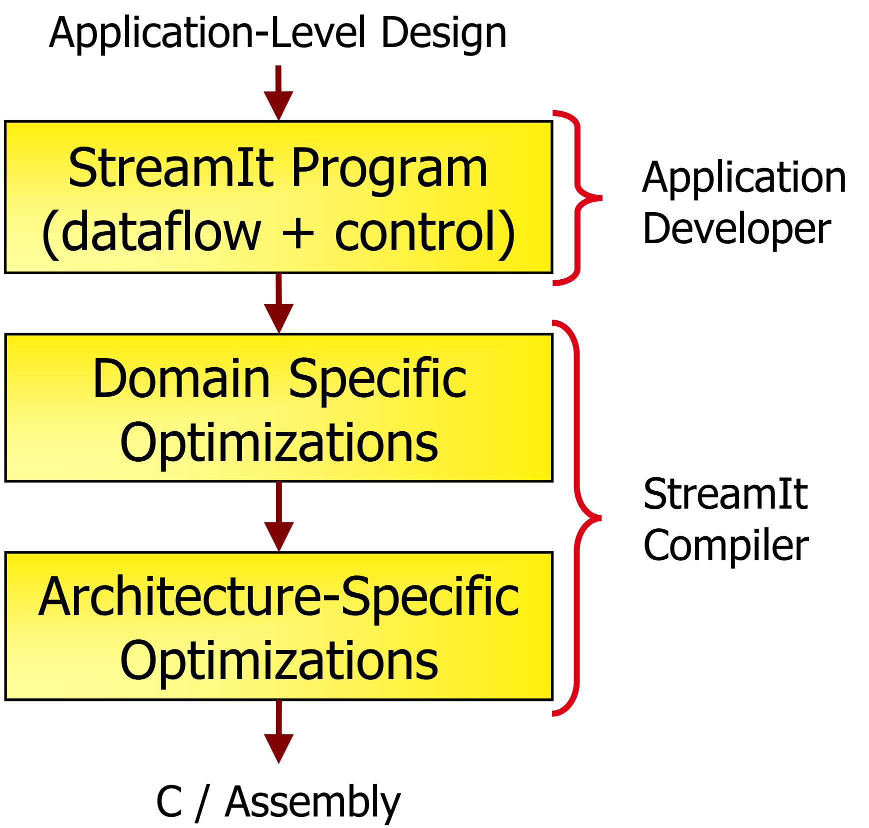
- Stream programming often requires special expertise
  - Conventional programming languages used in large applications make it difficult to extract parallelism
  - Low performance is not acceptable
  - System experts on the critical path

- Programming high-end machines should not be the "exclusive domain of experts"
  - Use high-level abstractions to naturally describe streaming computation
  - Make the language compiler-friendly
    - Expose parallelism and communication
    - Empower the compiler to perform novel optimization
    - Facilitate an efficient mapping of programs to (future) architectures

StreamIt Overview

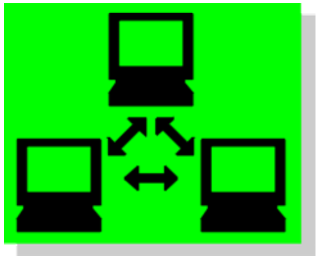
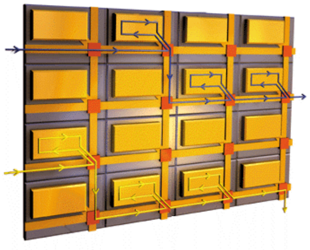
- StreamIt is a high-level, architecture-independent language for stream computing
  - Facilitate the rapid implementation of complex applications
  - Expert-programmer no longer on critical path to achieving high performance
  - StreamIt compiler applies novel domain specific optimizations

design flow with StreamIt



StreamIt in the Big Picture

- Unified programming model with single machine abstraction
  - Expose parallelism and communication
    - Uniprocessor, cluster of workstations, or tiled architecture

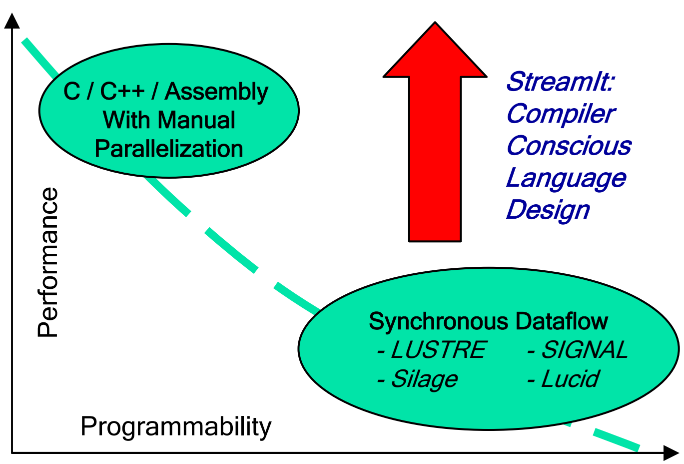
  - Hide granularity of execution, architecture details
  - Natural textual representation
  - Innovative compiler technology focuses on the core set of challenges
    - Versatility, load balancing, fault tolerance, …

provide “performance transparency”, portability, and support for “programming in the large”

Related Work

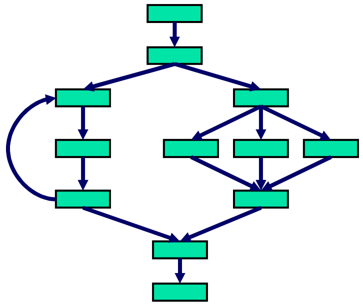
- "Stream languages"
  - KernelC/StreamC, Brook: augment C with data-parallel kernels
  - Cg: allow low-level programming of graphics processors
- Prototyping environments
  - Ptolemy, Simulink, etc.: provide graphical abstractions, but do not focus on compiling for performance or reliability
- Control languages for embedded systems
  - LUSTRE, Esterel, etc.: can verify safety properties, but do not expose high-bandwidth data flow for optimization

- In general, little features for scalable program development, or too abstract and unstructured
  - Compiler cannot perform enough analysis and optimizations
  - StreamIt exposes more task parallelism, and uses constructs that are easier to analyze



StreamIt Language Overview

- A StreamIt program is a structured graph of nodes
  - Nodes are autonomous units of computation
  - Edges are communication channels (FIFOs)

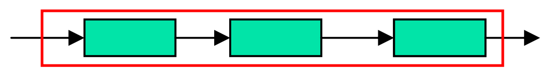


- Hierarchical structure
  - Single-Input to Single-Output language constructs
- Graph components are parameterizable
  - Short and natural recursive stream graph definitions

StreamIt Language Constructs

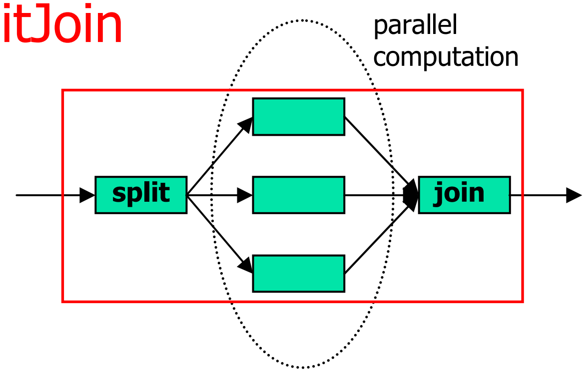
- Language constructs
  - Filter

  - Pipeline

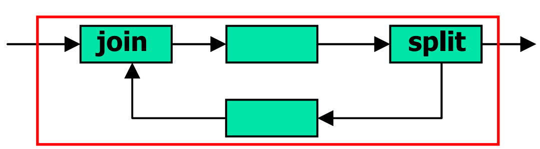


 may be any of the StreamIt language constructs

  - SplitJoin



  - FeedBack Loop

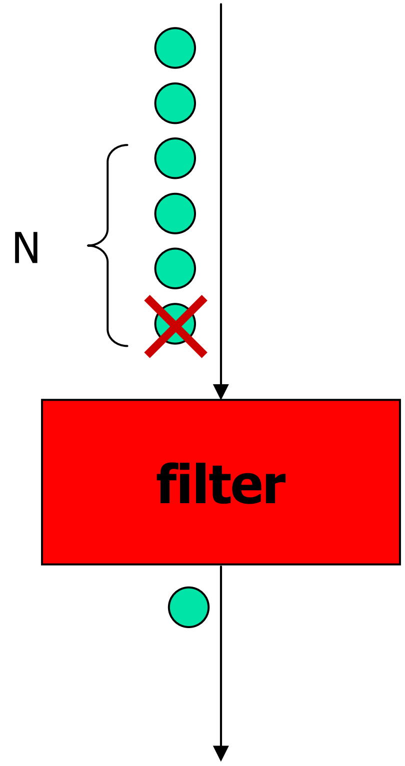


- Programming paradigm is
  - Modular
    - Important for large scale development
  - Malleable
    - Parameterized templates allow program to change behavior with small source code modifications
  - Composable
    - Composition of simple structures creates large and complex graphs
    - Enables inductive reasoning about correctness
  - Portable
    - Application is architecture independent

Filters as Computational Elements

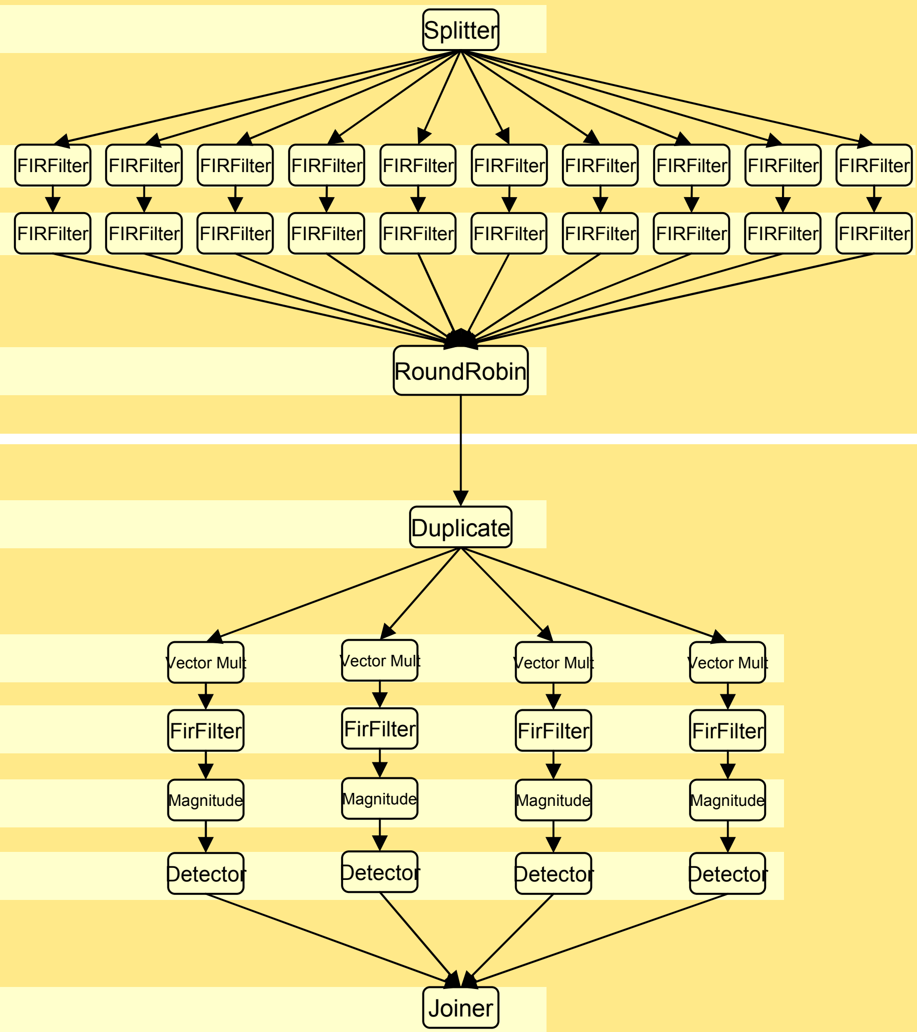
- Filters are the basic programmable units
  - An initialization function and a steady-state work function
  - Communicate via FIFOs: pop(), peek(index), push(value)

```
float→float filter FIR (int N) {
    float[N] weights;
    init {
        weights = calculate_weights(N);
    }
    work push 1 pop 1 peek N {
        float result = 0;
        for (int i = 0; i < N; i++) {
            result += weights[i] * peek(i);
        }
        push(result);
        pop();
    }
}
```



Example Application–Radar Front-End

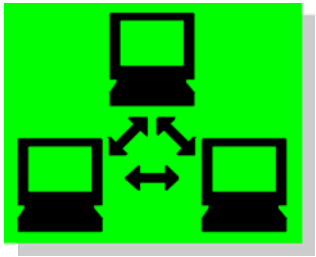
```
complex→ void pipeline BeamFormer(int numChannels, int numBeams) {
    add splitjoin {
        split duplicate;
        for (int i=0; i<numChannels; i++) {
            add pipeline {
                add FIR1(N1);

                add FIR2(N2);
            };
        };
        join roundrobin;
    };

    add splitjoin {
        split duplicate;
        for (int i=0; i<numBeams; i++) {
            add pipeline {
                add VectorMult();

                add FIR3(N3);

                add Magnitude();

                add Detect();
            };
        };
        join roundrobin(0);
    };
}
```



StreamIt Compiler

StreamIt Compiler Overview

- The StreamIt language hides granularity of execution and architecture details
  - Compiler backend supports
    Uniprocessor, cluster of workstations, and MIT Raw


- Innovative compiler technology focuses on the core set of challenges to deliver high performance in future architectures
  - Automating domain specific optimizations
    - Optimization of linear streams
    - Translation to the frequency domain
  - Partitioning, layout, routing, …

Linear Filter Optimizations

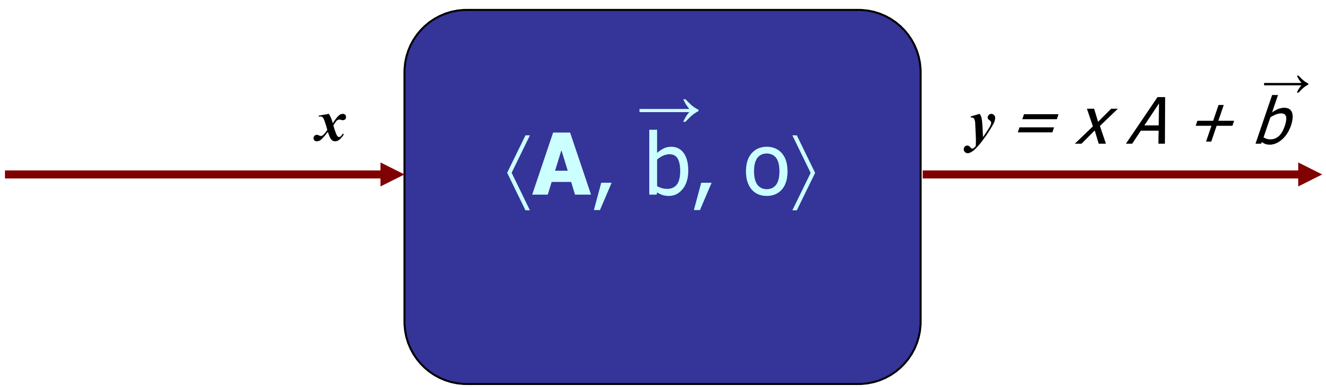
- Most common target of DSP optimizations
  - FIR filters
  - Compressors
  - Expanders
  - DFT/DCT

  Output is weighted sum of inputs

- Example optimizations:
  - Combining Adjacent Nodes
  - Translating to Frequency Domain

Representing Linear Filters

- A linear filter is a tuple $\langle \mathbf{A}, \vec{b}, o \rangle$
  - **A**: matrix of coefficients
  - $\vec{b}$: vector of constants
  - o: number of items popped
- Example



Extracting Linear Representation

```
work peek N pop 1 push 1 {
    float sum = 0;
    for (int i=0; i<N; i++) {
      sum += h[i]*peek(i);
    }
    push(sum);
    pop();
}
```

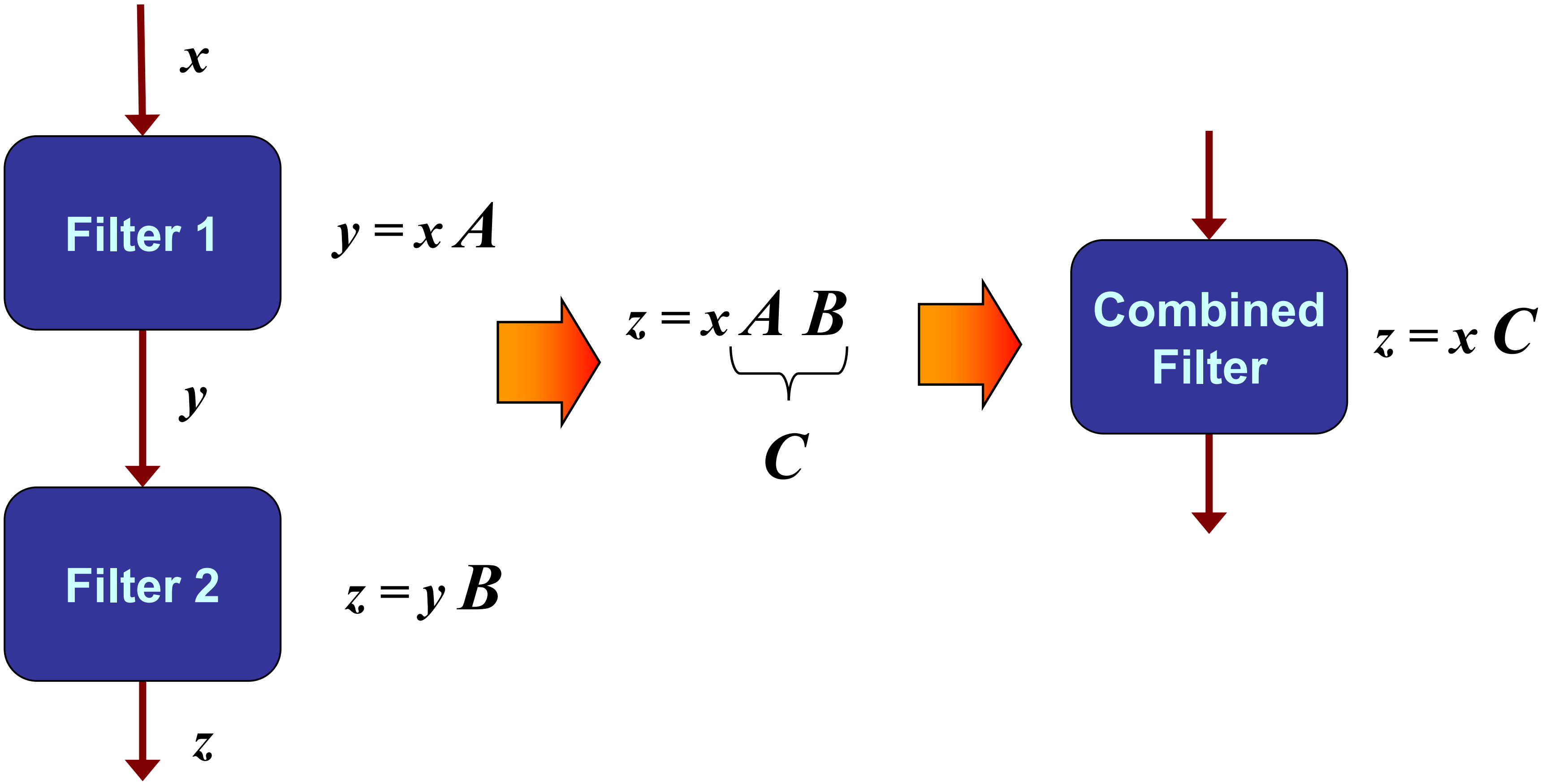
Linear
Dataflow
Analysis

$$\langle \mathbf{A}, \vec{b}, o \rangle$$

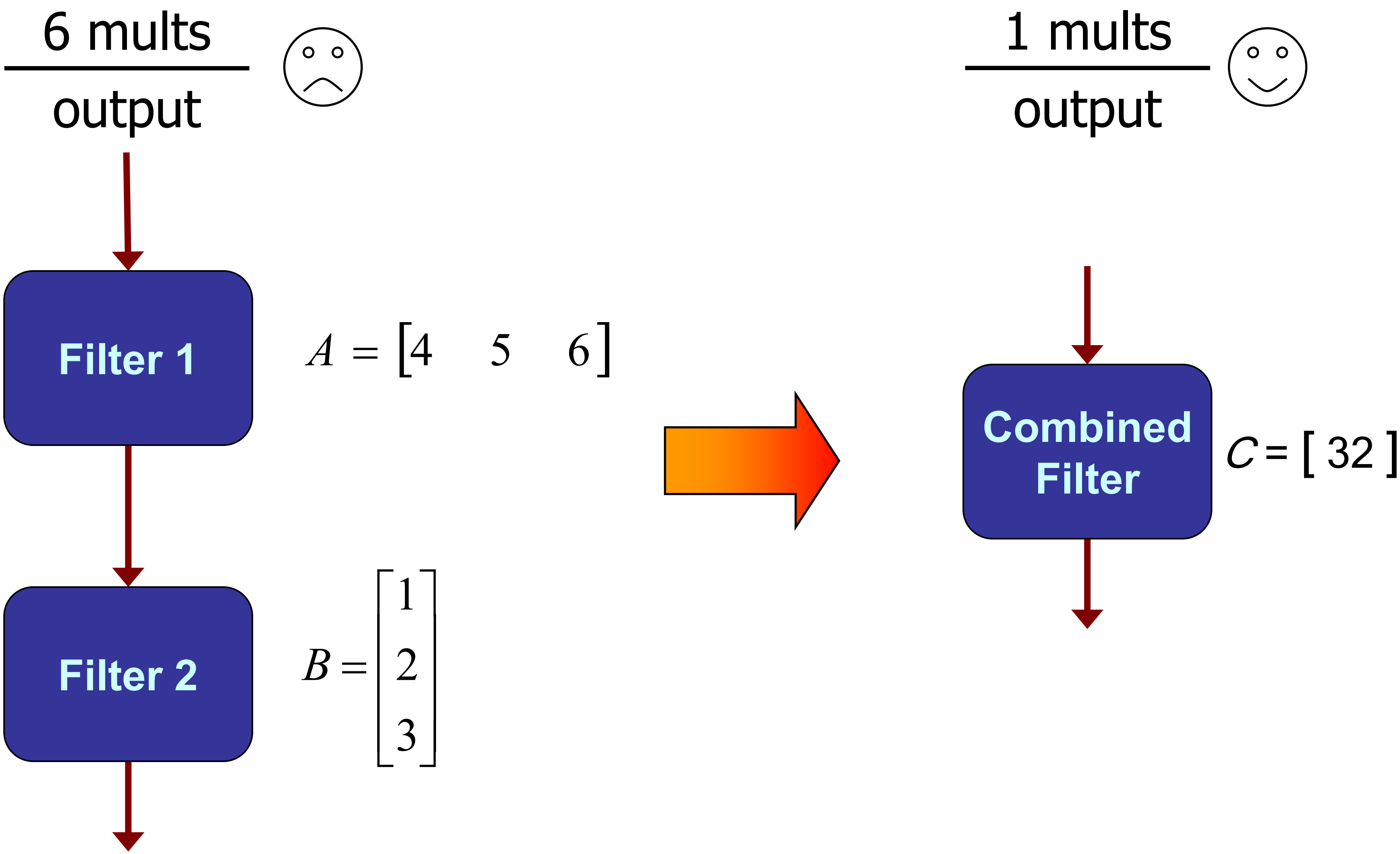
- Resembles constant propagation
- Maintains linear form $\langle \vec{v}, \vec{b} \rangle$ for each variable
  - Peek expression: generate fresh $\vec{v}$
  - Push expression: copy $\vec{v}$ into **A**
  - Pop expression: increment o

Combining Linear Filters

- Pipelines and splitjoins can be collapsed
- Example: pipeline

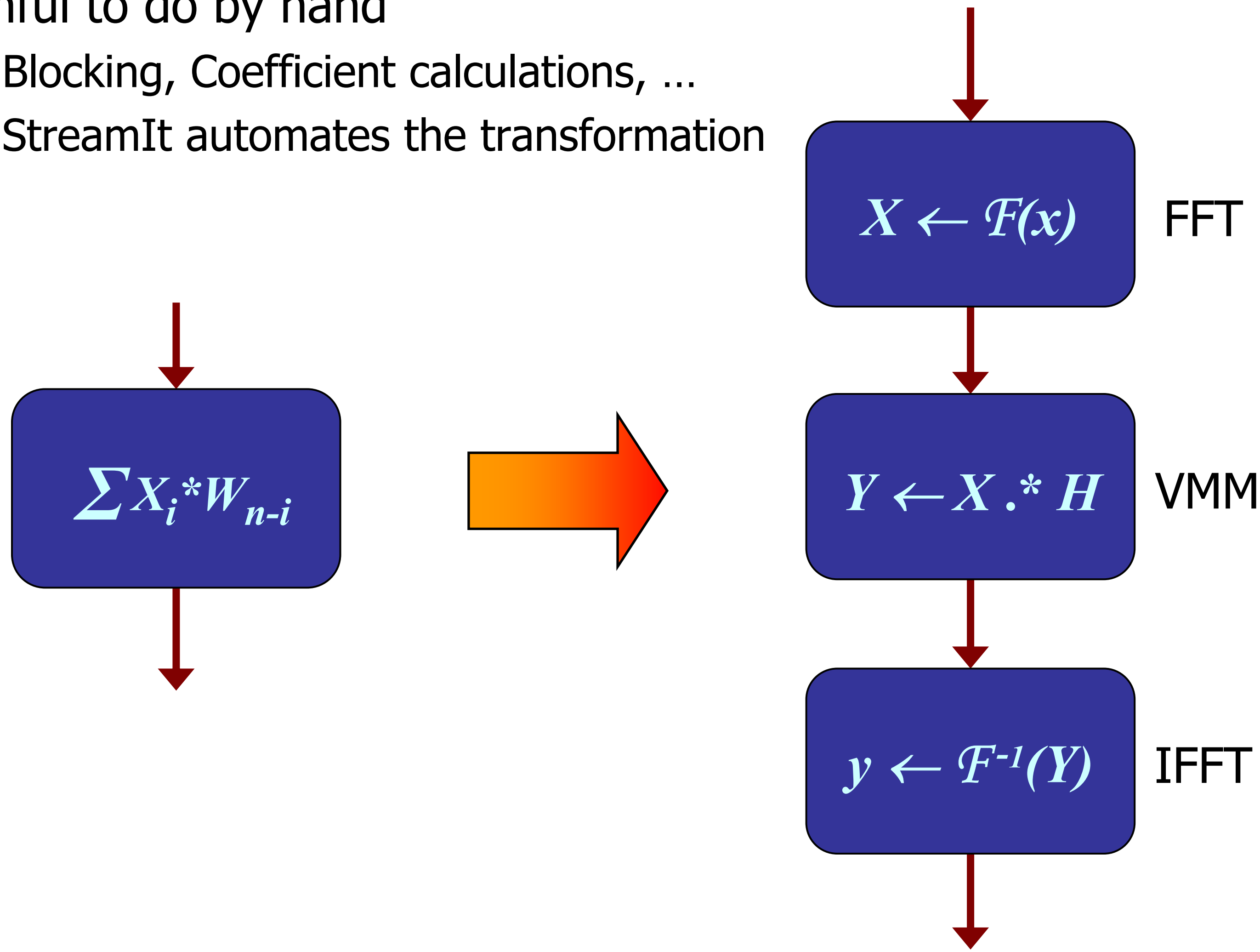


Combination Example

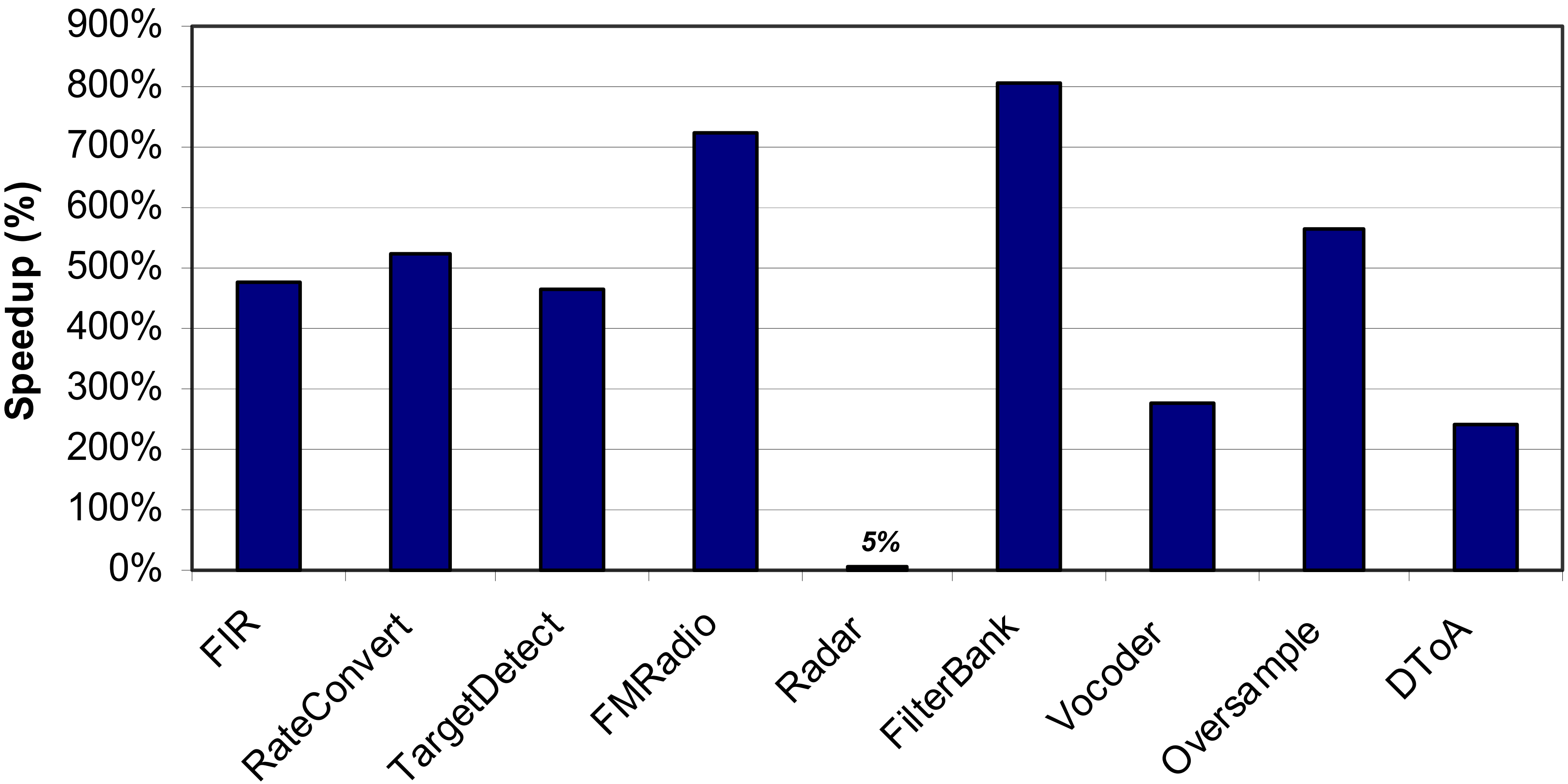


From Time to Frequency Domain

- Convolutions are cheap in the Frequency Domain
  - Painful to do by hand
    - Blocking, Coefficient calculations, …
    - StreamIt automates the transformation

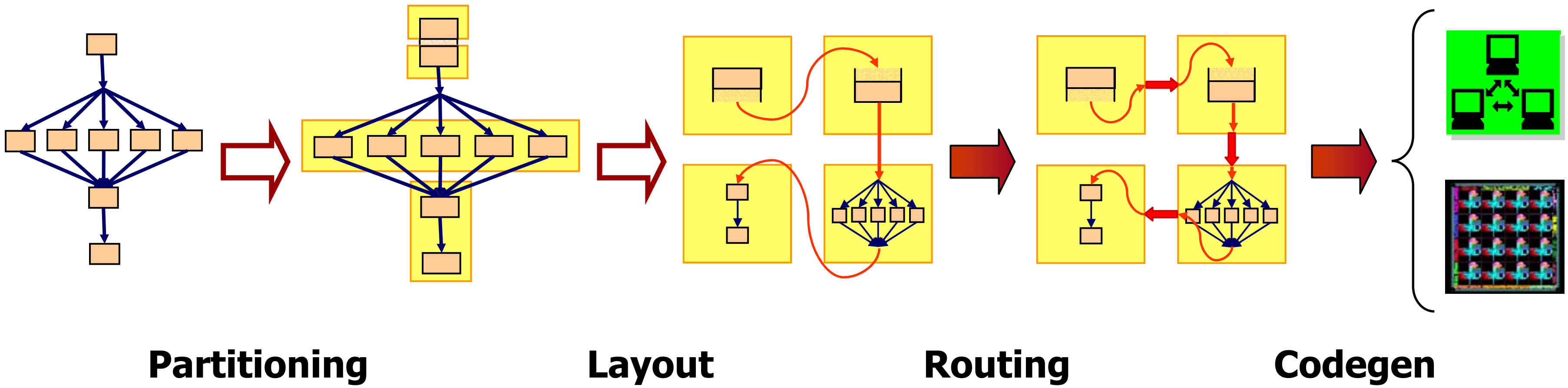


Linear Optimization of Stream Graph



On a Pentium IV

Backend for Parallel Platforms



- StreamIt exposes communication patterns
  - Automatic generation and optimization of routing code
  - Otherwise, may require extensive (assembly) programming

    FIR – Raw backend

    - 15 statements of StreamIt code achieve the same performance as 352 statements of manually-tuned C

    Frequency Hopping Radio – cluster backend

    - 50% higher throughput and 35% less communication, when using StreamIt's messaging construct

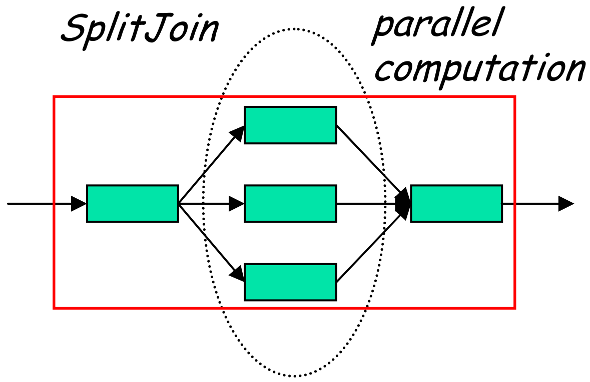
Development Support

StreamIt Development Tool (SDT)

- Graphical development environment
- Text editor
  - Key-word highlighting and indentation schemes
- Graphical editor for the rapid prototyping of stream applications
  - Fast composition of stream graphs
- Graphical debugger
  - Step by step execution
  - Inspection and modification of program variables
- Online help manuals
- Integrated with the IBM Eclipse Tool Platform
  - Available at http://cag.csail.mit.edu/streamit/html/eclipse-plugin.html

Debugging Parallel StreamIt Programs

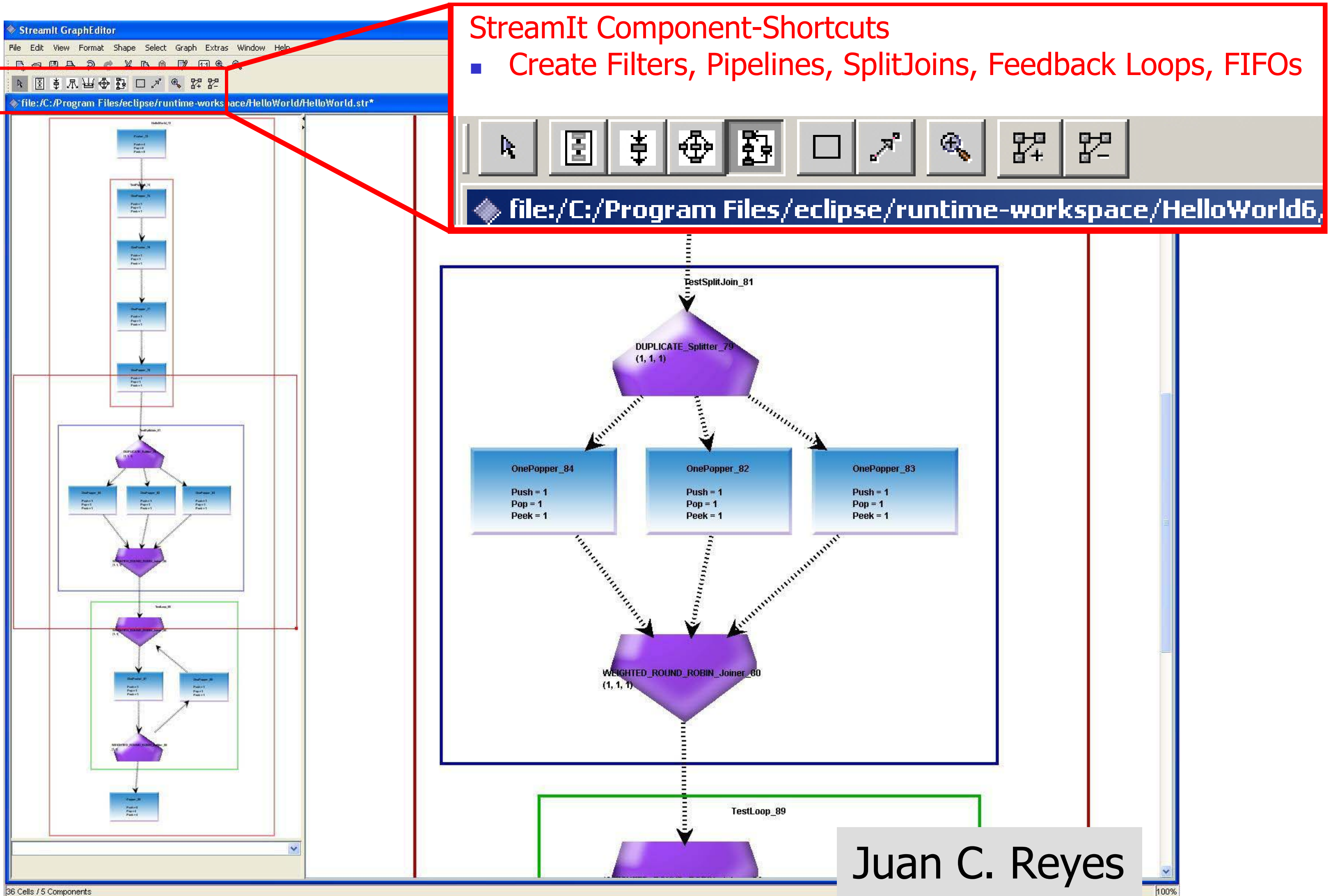
- Parallelism and communication are exposed
  - Tracking the flow of data in a stream graph affords a frame of reference for reasoning about "time"
  - Powerful advantage when debugging parallel programs



versus

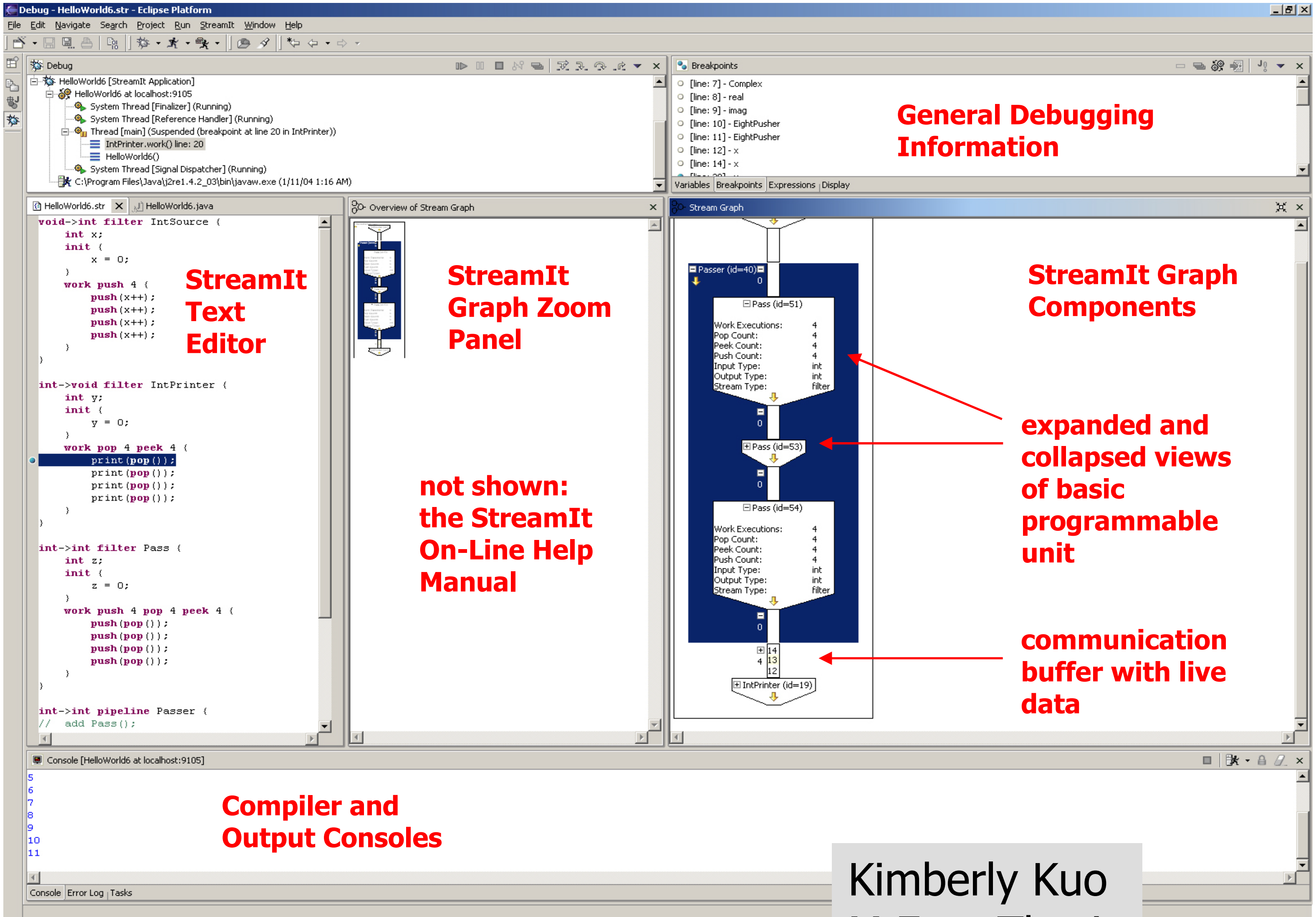
- Multiple threads with independent program counters
- Non-deterministic execution

StreamIt Graphical Editor



Juan C. Reyes
M.Eng. Thesis

StreamIt Debugging Environment



Kimberly Kuo
M.Eng. Thesis

Programmer Evaluation of the SDT

- A detailed study was held in April to evaluate the SDT from a user’s perspective
  - Monitored environment to track user progress on a core set of problems
    - Language and SDT tutorial
    - Pre-study and post-study questionnaires
    - Post-study interview
- Visualization capabilities in the SDT considered "invaluable" to some users
  - Especially when stream graphs were large (and generated recursively)
- The ability to track the flow of data was also considered extremely helpful
- Many lessons learned!

For More Information

StreamIt Homepage

http://cag.csail.mit.edu/streamit

download papers, benchmarks, compiler, SDT, ...