High-Productivity Stream Programming for High-Performance Systems

Rodric Rabbah, Bill Thies, Michael Gordon, Janis Sermulins, and Saman Amarasinghe

Massachusetts Institute of Technology

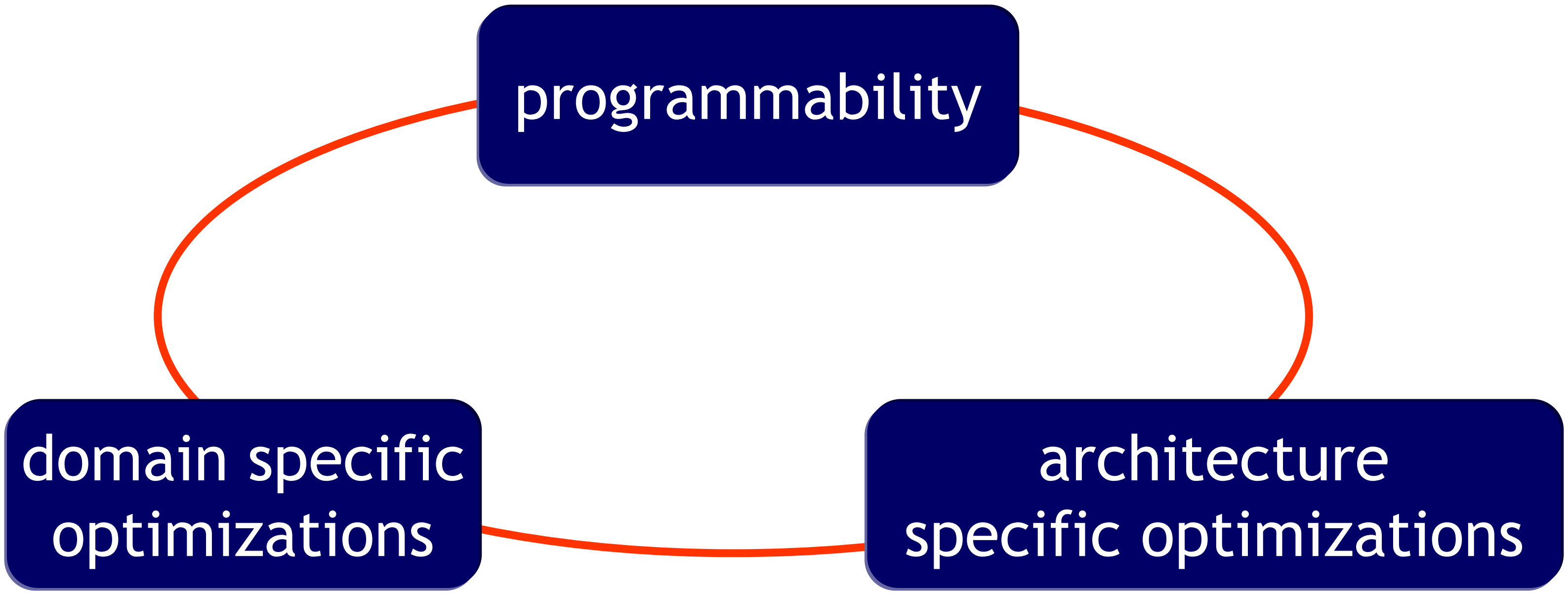
StreamIt

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

HPEC 2005, MIT LL

The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping



- Simple and effective optimizations for streams
- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors

Why an Emphasis on Streaming?

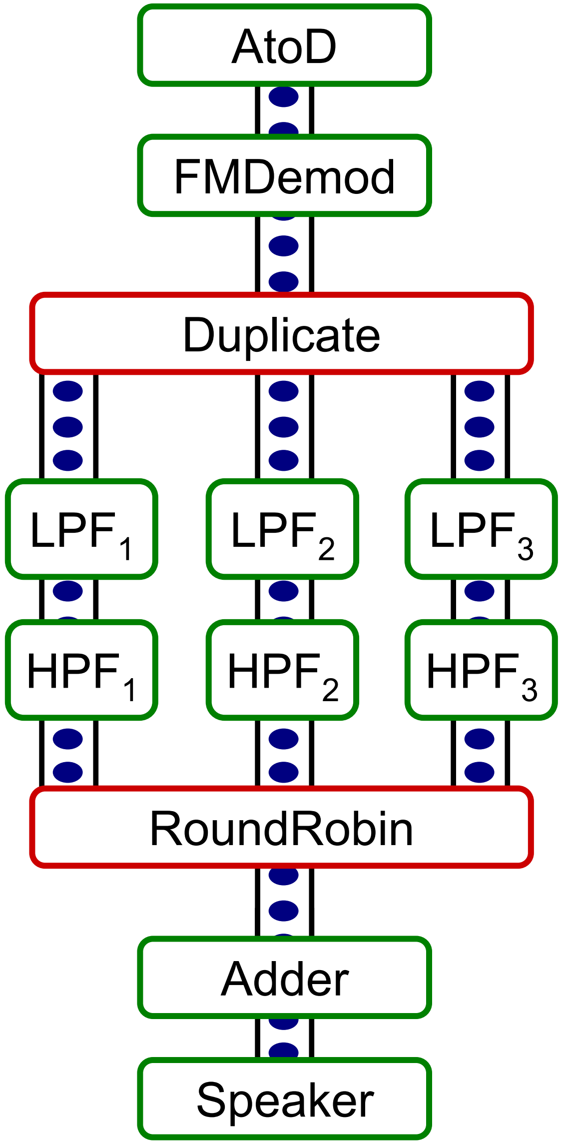
Streaming in other Domains as well

- Cryptography
- Databases
- Face recognition
- Network processing and security
- Scientific codes
- …

- Attractive programming model because of a simple mapping from specification to implementation

Properties of Stream Programs

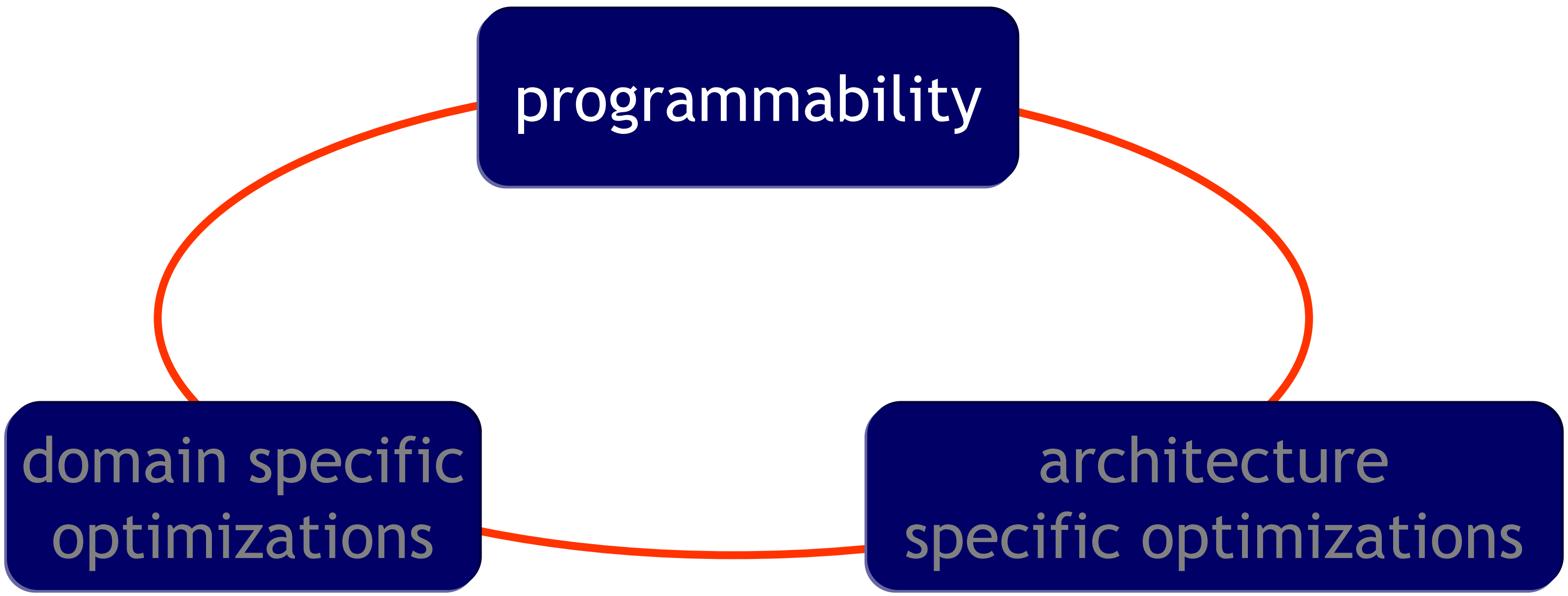
- Mostly regular and repeating computation
- Parallel, independent computation with explicit communication



- Amenable to aggressive compiler optimizations
  [ASPLOS '02, PLDI '03, LCTES'03, LCTES '05]

The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping



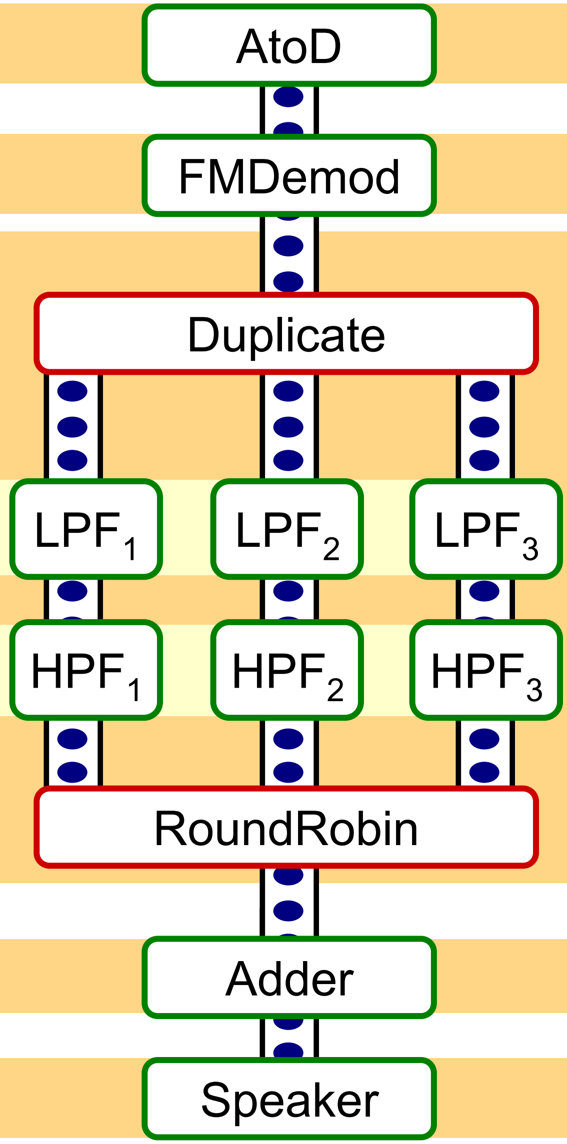
- Simple and effective optimizations for streams

- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors

Programming in StreamIt

```
void->void pipeline FMRadio(int N, float freq1, float freq2) {
  add AtoD();

  add FMDemod();

  add splitjoin {
    split duplicate;
    for (int i=0; i<N; i++) {
      add pipeline {
        add LowPassFilter(freq1 + i*(freq2-freq1)/N);

        add HighPassFilter(freq2 + i*(freq2-freq1)/N);
      }
    }
    join roundrobin();
  }
  add Adder();

  add Speaker();
}
```

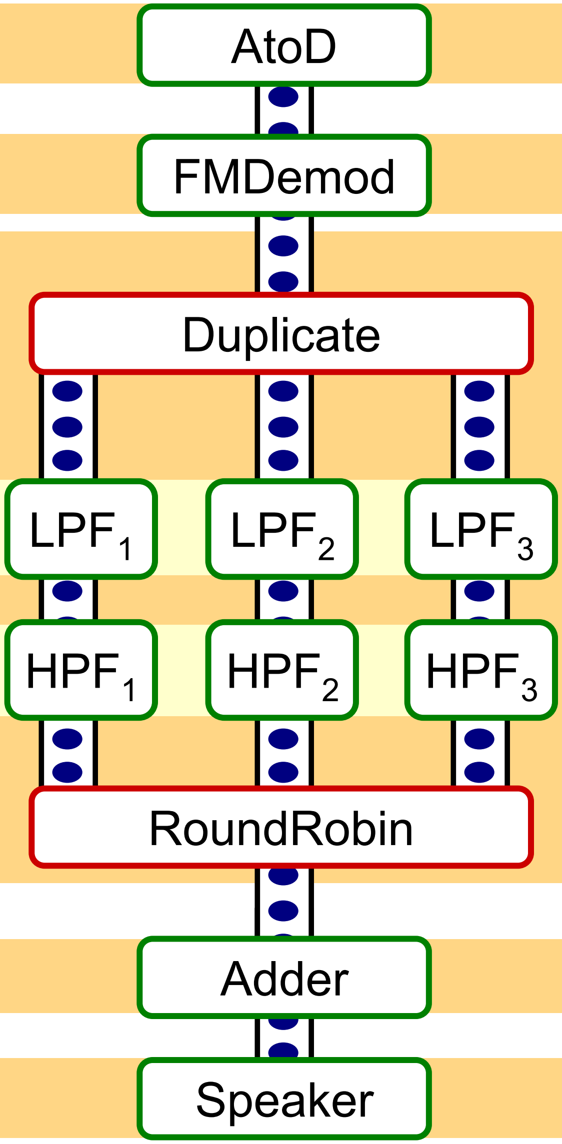
– Natural correspondence between text and application graph



Programming in StreamIt

```
void->void pipeline FMRadio(int N, float freq1, float freq2) {
  add AtoD();

  add FMDemod();

  add splitjoin {
    split duplicate;
    for (int i=0; i<N; i++) {
      add pipeline {
        add LowPassFilter(freq1 + i*(freq2-freq1)/N);

        add HighPassFilter(freq2 + i*(freq2-freq1)/N);
      }
    }
    join roundrobin();
  }
  add Adder();

  add Speaker();
}
```

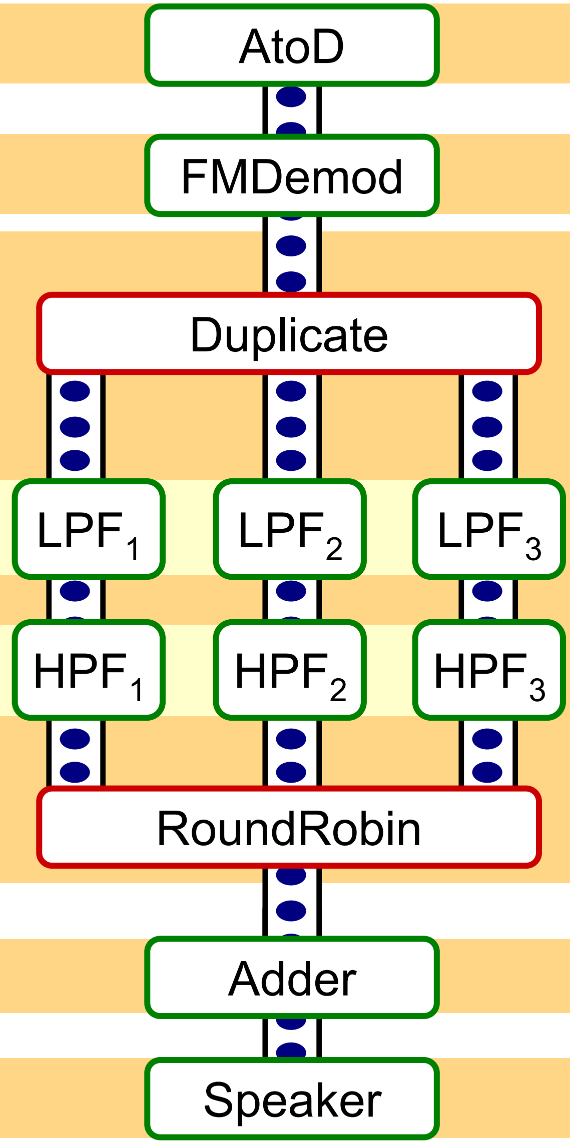
– Streams are easily composed



Programming in StreamIt

```
void->void pipeline FMRadio(int N, float freq1, float freq2) {
  add AtoD();

  add FMDemod();

  add splitjoin {
    split duplicate;
    for (int i=0; i<N; i++) {
      add pipeline {
        add LowPassFilter(freq1 + i*(freq2-freq1)/N);

        add HighPassFilter(freq2 + i*(freq2-freq1)/N);
      }
    }
    join roundrobin();
  }
  add Adder();

  add Speaker();
}
```

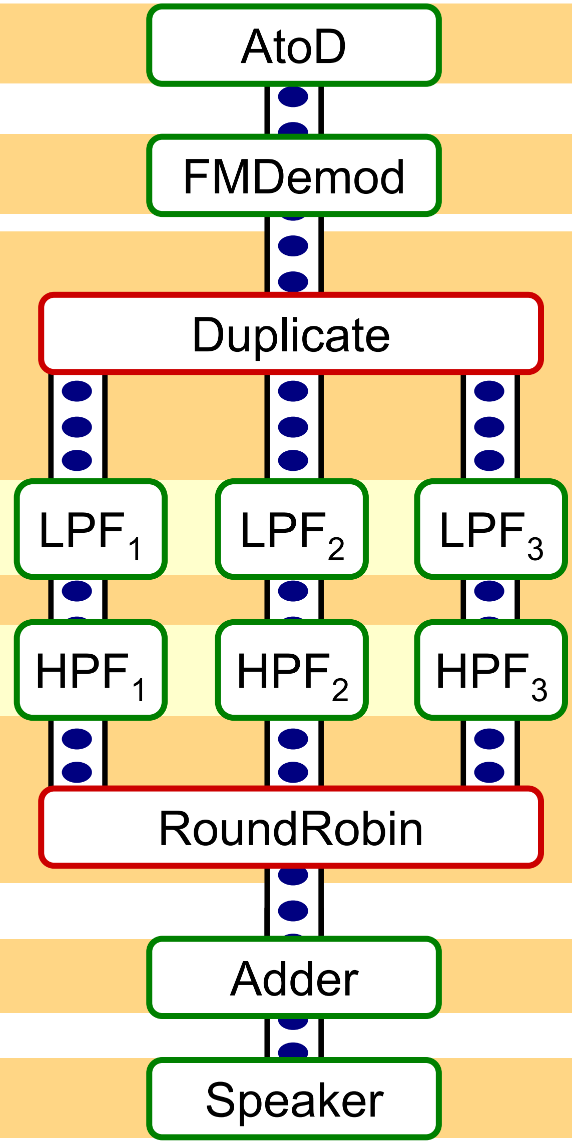
- Streams are parameterized, and malleable



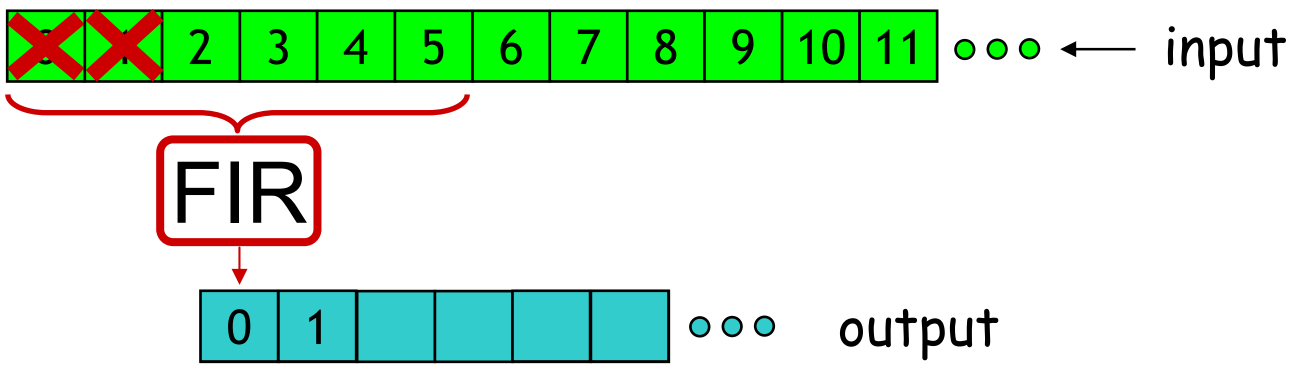
Programming in StreamIt

```
void->void pipeline FMRadio(int N, float freq1, float freq2) {
  add AtoD();

  add FMDemod();

  add splitjoin {
    split duplicate;
    for (int i=0; i<N; i++) {
      add pipeline {
        add LowPassFilter(freq1 + i*(freq2-freq1)/N);

        add HighPassFilter(freq2 + i*(freq2-freq1)/N);
      }
    }
    join roundrobin();
  }
  add Adder();

  add Speaker();
}
```

- Application is architecture independent (i.e., portable)



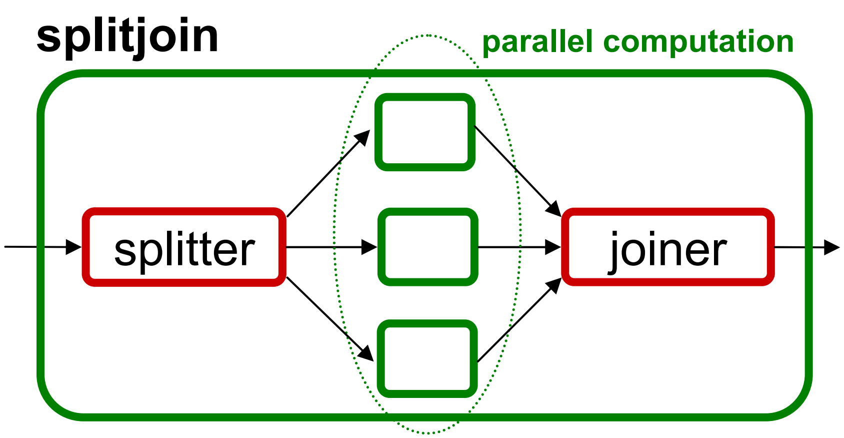
Filters as Computational Elements




```
float→float filter FIR (int 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();
    }
}
```

Benefits of StreamIt

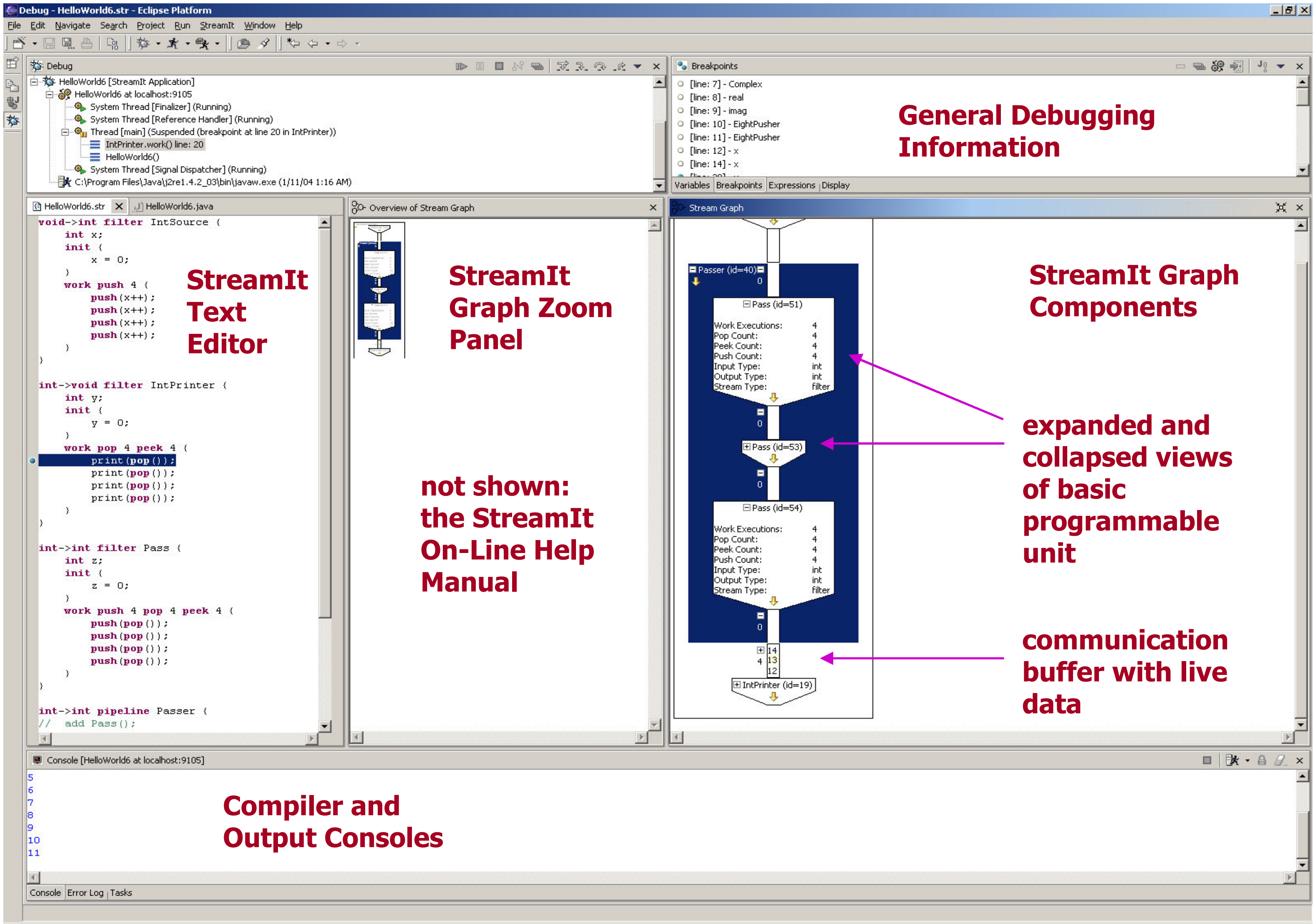
- Communication is exposed and pipeline parallelism is more readily discovered
- Flow of data provides a frame of reference for reasoning about "time" [PPoPP '05]
  - Powerful advantage when debugging parallel programs



versus

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

StreamIt Development Environment

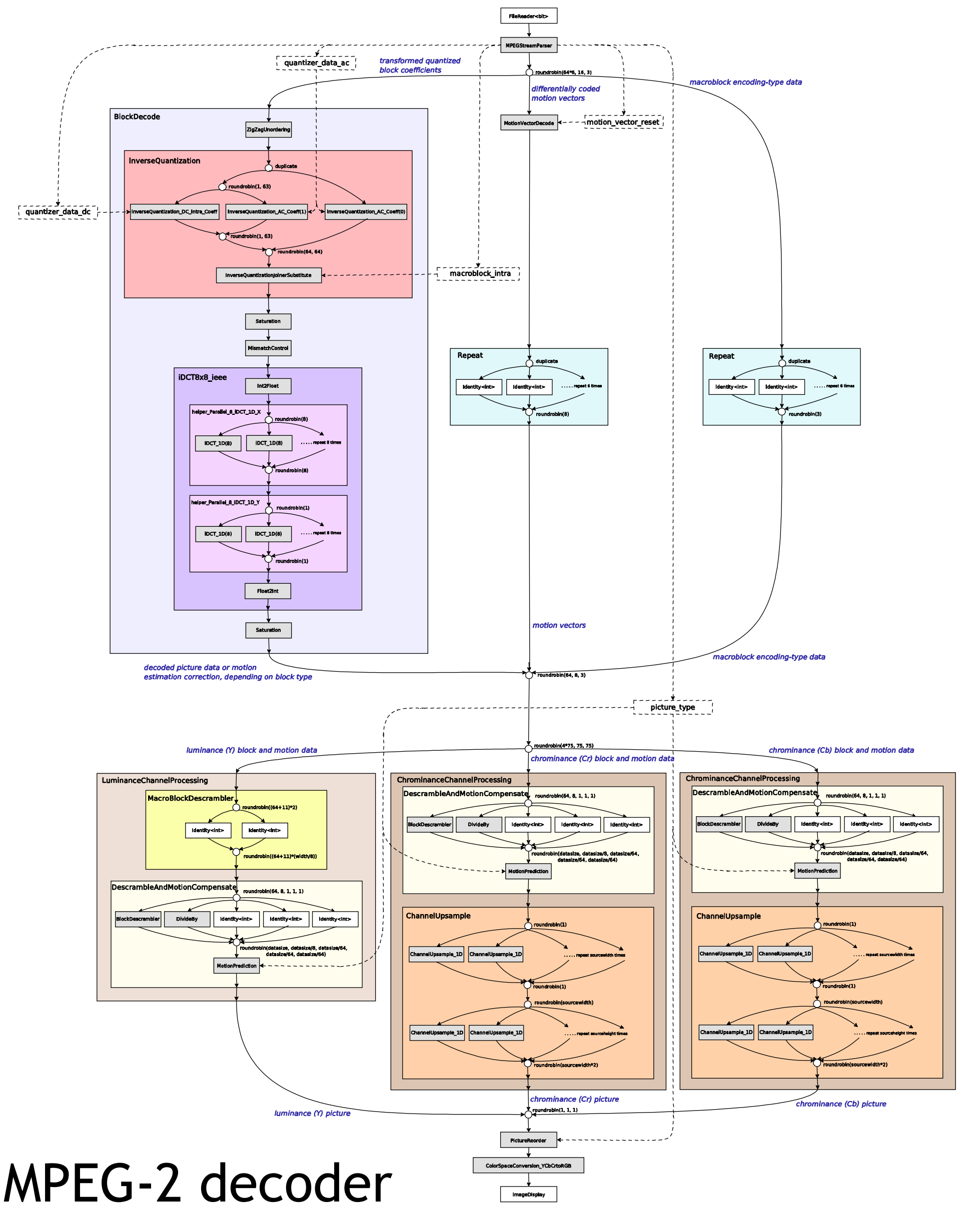


[PHEC ’05]

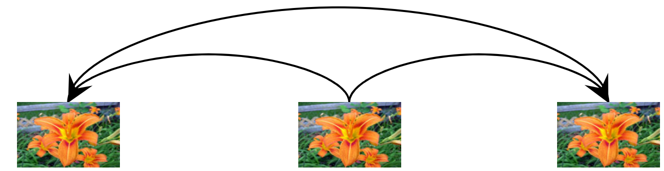
StreamIt Applications

- Software radio
- Frequency hopping radio
- Acoustic beam former
- Vocoder
- GMTI (ground moving target indicator)
- DES and Serpent blocked ciphers
- Sorting
- FFTs and DCTs
- JPEG
- …

MPEG: Motion Video Codec



MPEG-2 decoder



independently coded

forward/backward predicted

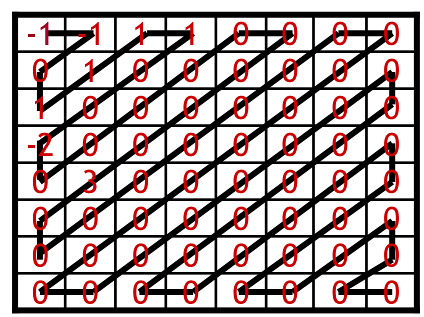
forward predicted

frames encoded using motion prediction

encoding ⇓ ⇑ decoding

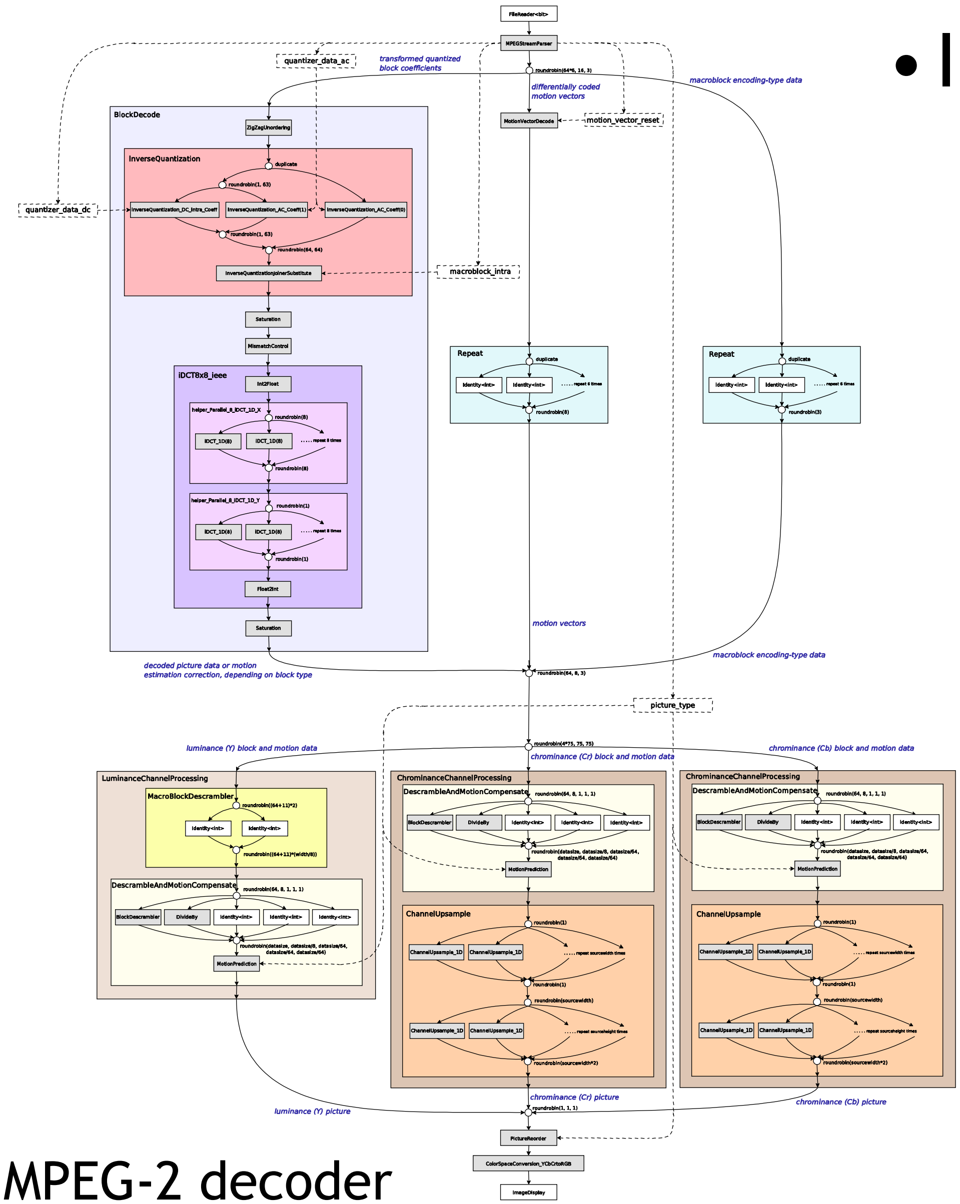
luminance and chrominance color data are separated

encoding ⇓ ⇑ decoding



DCT and quantization of 8x8 image block

MPEG: Motion Video Codec



MPEG-2 decoder

- Implementation statistics
  - 4921 lines of code
    - 48 static streams
    - Compile to ~2150 filters
      - 352x240 resolution
    - Reference C implementation has 9832 lines of code
      - Supports interlacing and multi-layer streams
  - 8 weeks of development
  - 1 programmer with no prior MPEG-2 experience

Excerpt from StreamIt Implementation

Specification in Section 7.4.1: **F”[0][0] = intra_dc_mult x QF[0][0]**

Table 7-4 - Relation between intra_dc_precision and intra_dc_mult

intra_dc_precision	bits_of_precision	intra_dc_mult
0	8	8
1	9	4
2	10	2
3	11	1

```
int->int filter InverseQuantization() {
  int[4] intra_dc_mult = {8, 4, 2, 1};
  int intra_dc_precision;

  work pop 1 push 1 {
    push(intra_dc_mult[intra_dc_precision] * pop());
  }
}
```

Excerpt from Reference Implementation

Specification in Section 7.4.1: **F”[0][0] = intra_dc_mult x QF[0][0]**

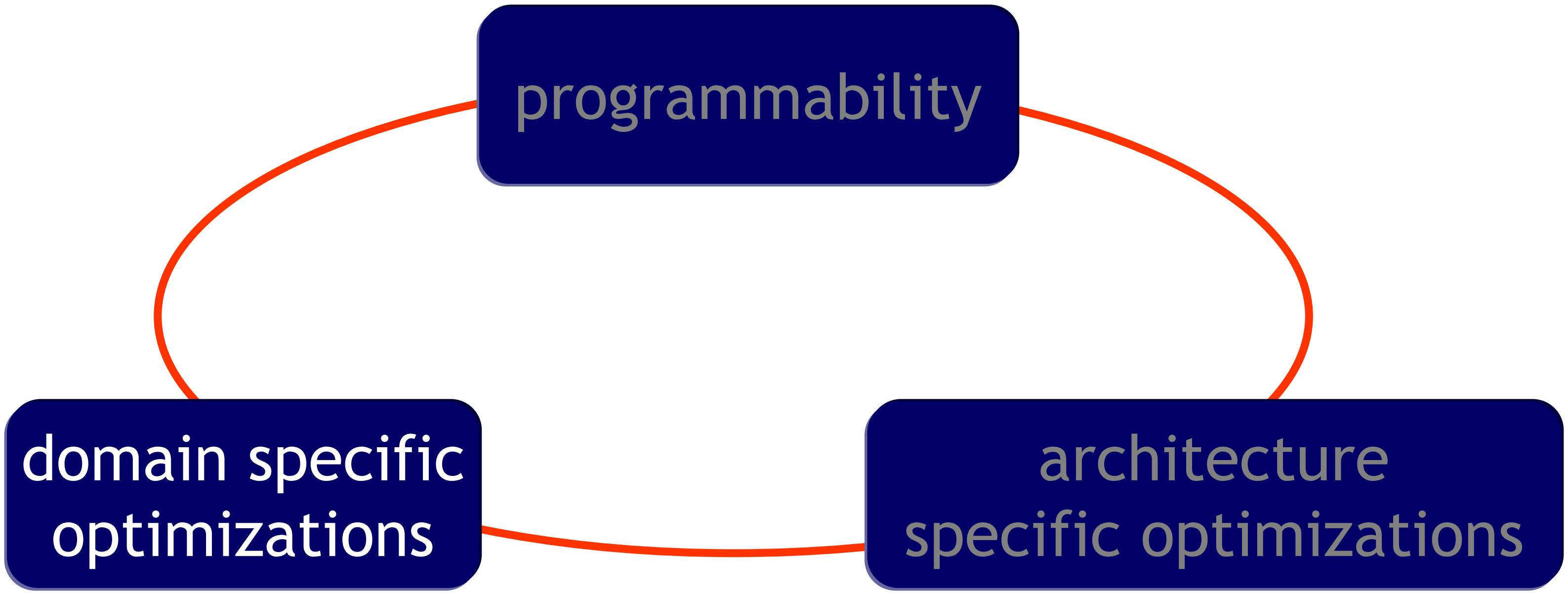
Table 7-4 - Relation between intra_dc_precision and intra_dc_mult

intra_dc_precision	bits_of_precision	intra_dc_mult
0	8	8
1	9	4
2	10	2
3	11	1

```
int[4] intra_dc_mult = {8, 4, 2, 1};

for (int m = 0; m < W*H/(16*16); m++)
 // six components for chrominance and luminance
 for (int comp = 0; comp < 6; comp++)
   if (macroblock[m].intra)
      macroblock[m].block[comp][0] *= intra_dc_mult[intra_dc_precision];
   // and many lines later
   if (cc == 0)
      val = (dc_dct_pred[0] += Get_Luma_DC_dct_diff());
   else if (cc == 1)
     val = (dc_dct_pred[1] +=  Get_Chroma_DC_dct_diff());
   else
      val = (dc_dct_pred[2] += Get_Chroma_DC_dct_diff());
   if (Fault_Flag) return;
   bp[0] = val << (3-intra_dc_precision);
```

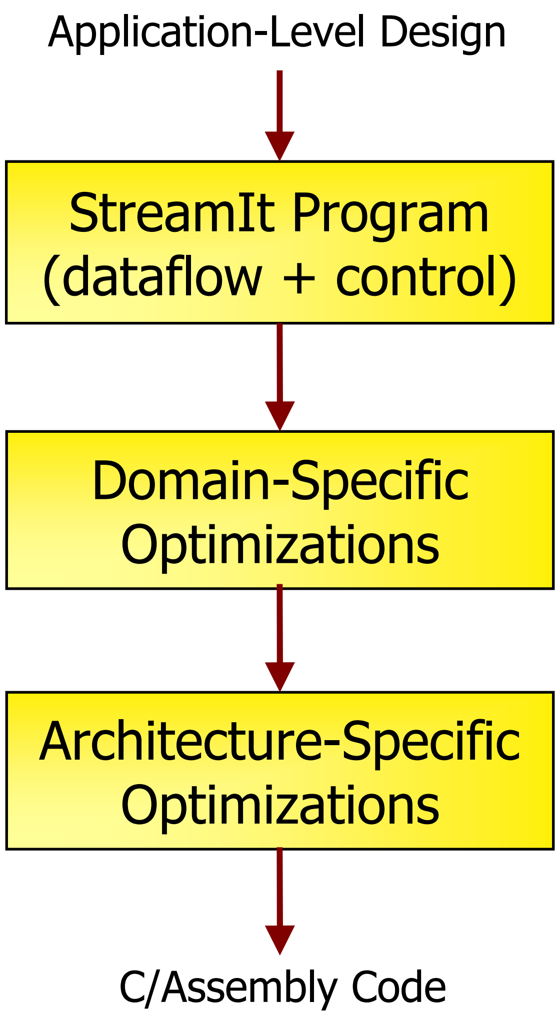
The StreamIt Vision

- Boost productivity, enable faster development and rapid prototyping



- Simple and effective optimizations for streams
- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors

Design Flow with StreamIt



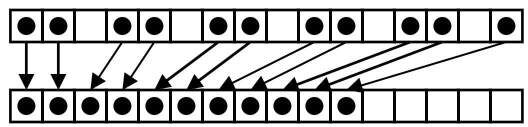
Application Programmer

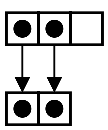
StreamIt compiler

- Leverage analyzability of streams and filter code to enable novel stream transformations
- In this talk: linear optimizations [PLDI ’03, PLDI ’05, CASES ’05]

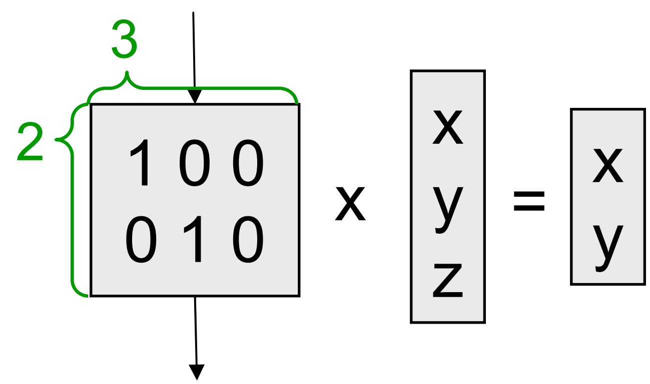
Linear Filter Example

- “Drop every third bit in the bit stream”





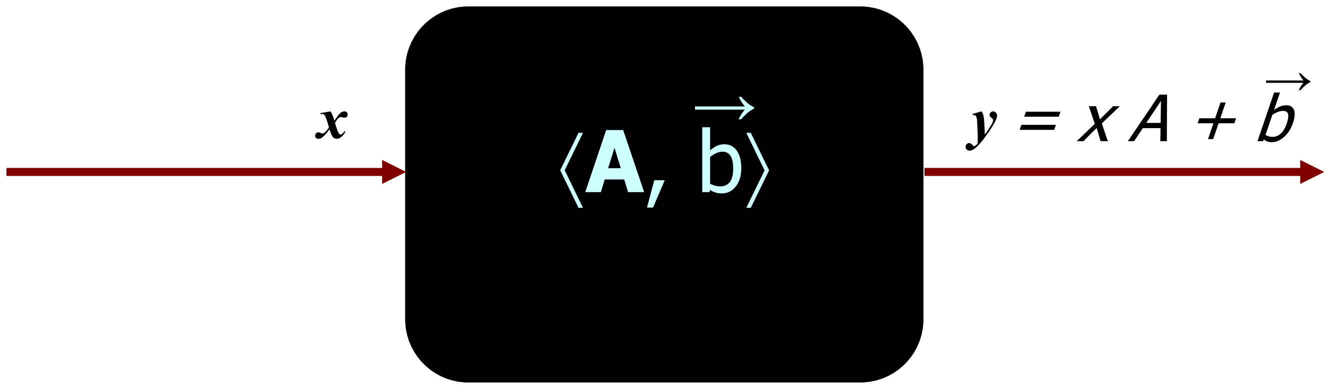
```
bit→ bit filter DropThirdBit {
    work push 2 pop 3 {
        push(pop());
        push(pop());
        pop();
    }
}
```



In General

- A linear filter is a tuple $\langle \mathbf{A}, \vec{b} \rangle$
  - $\mathbf{A}$: matrix of coefficients
  - $\vec{b}$: vector of constants

- Example



- Linear dataflow analysis resembles constant propagation

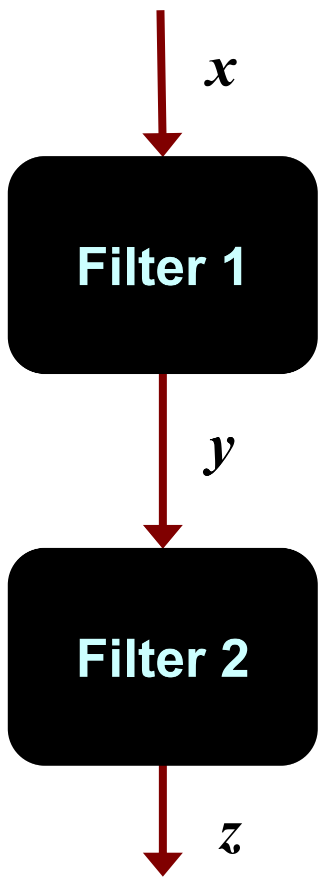
Opportunities for Linear Optimizations

- Occur frequently in streaming codes
  - FIR filters
  - Compressors
  - Expanders
  - DFT/DCT
  - Bit permutations in encryption algorithms
  - JPEG and MPEG codecs
  - …
- Example optimizations
  - Combining adjacent nodes
  - Also, translating to frequency domain when profitable

Combining Linear Filters

$\frac{\text{6 mults}}{\text{output}}$ ☹

$\frac{\text{1 mult}}{\text{output}}$ ☺



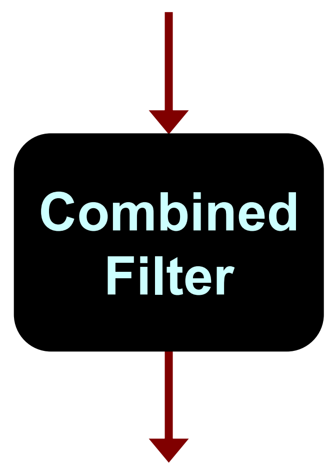
$$A = \begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$$

$$y = x\,A$$

$$z = y\,B$$

$$B = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$$

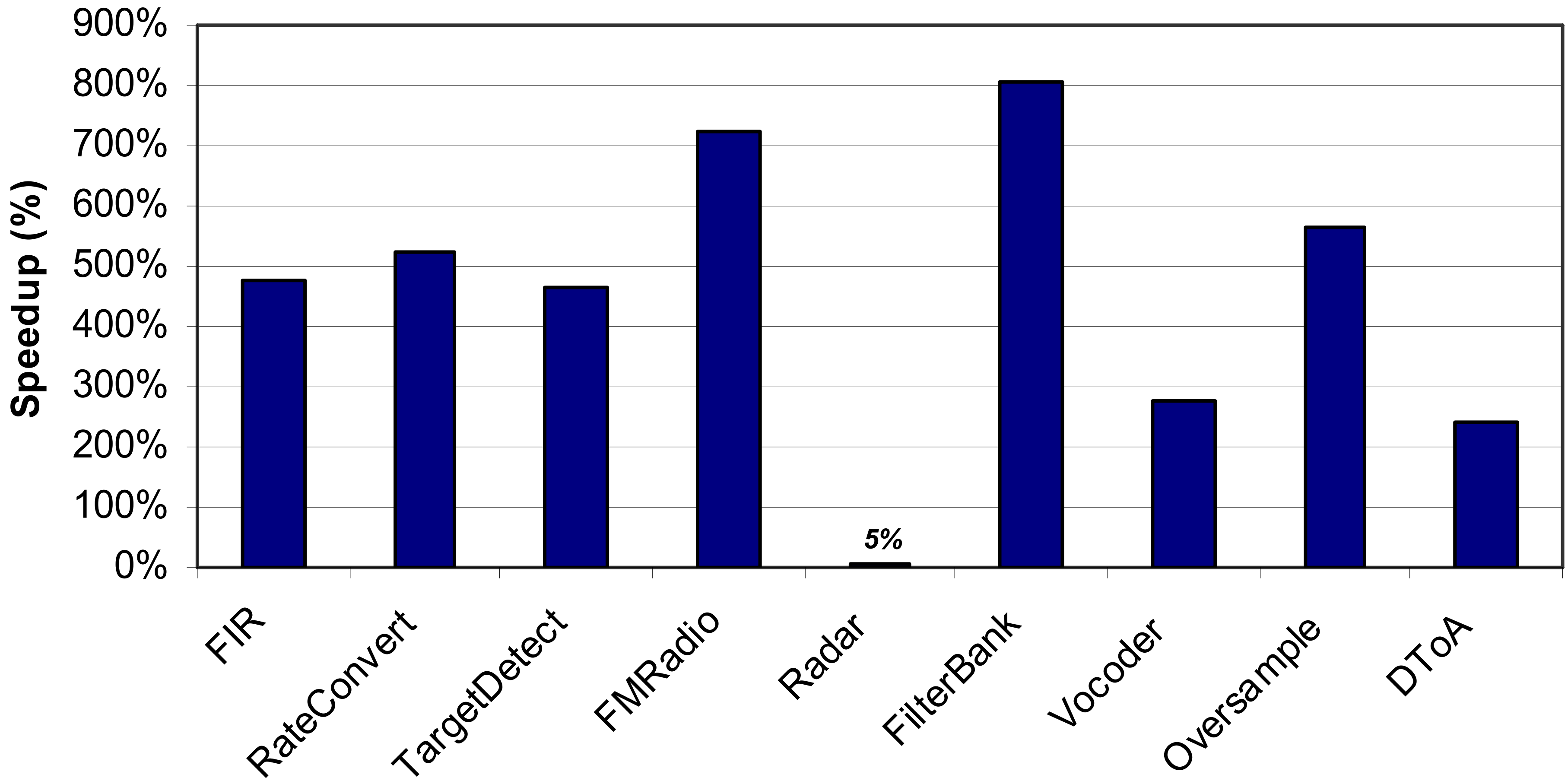
$$z = x\,\underbrace{A\ B}_{C}$$



$$z = x\,C$$

$$C = [\ 32\ ]$$

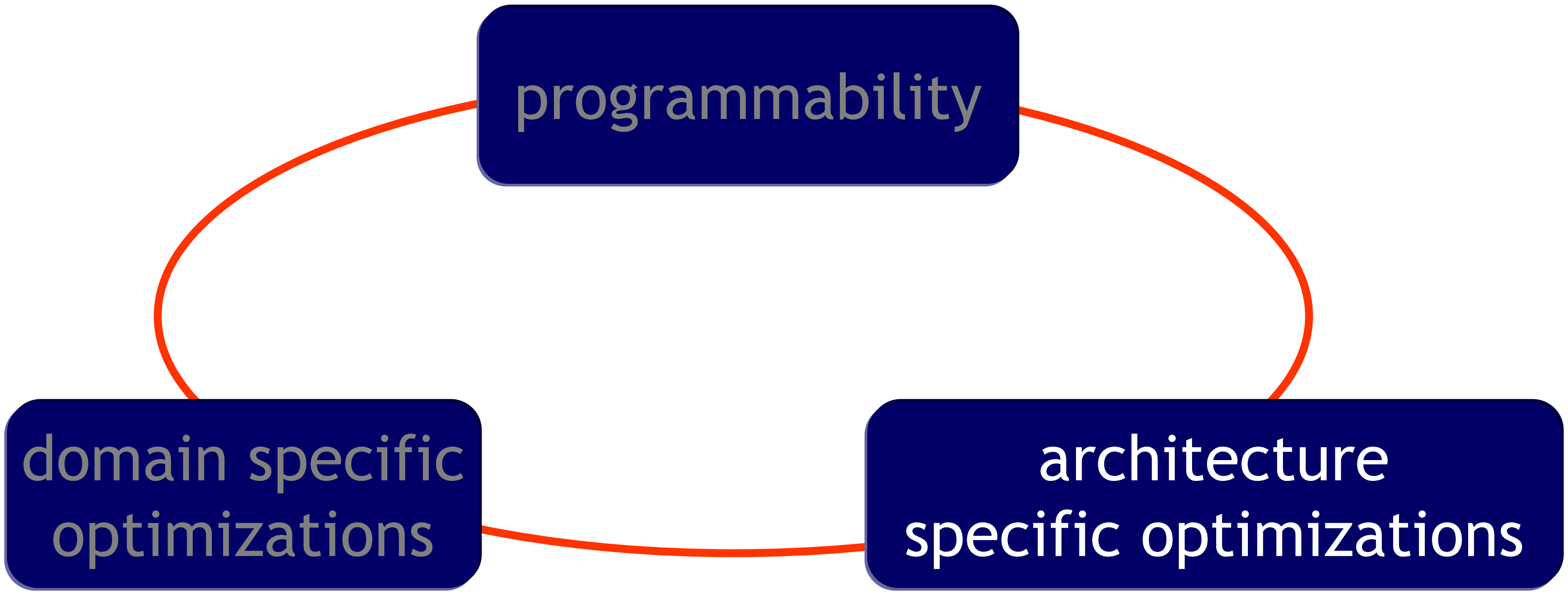
Results from Linear Optimizations



Pentium 4 results compared to baseline StreamIt

The StreamIt Vision

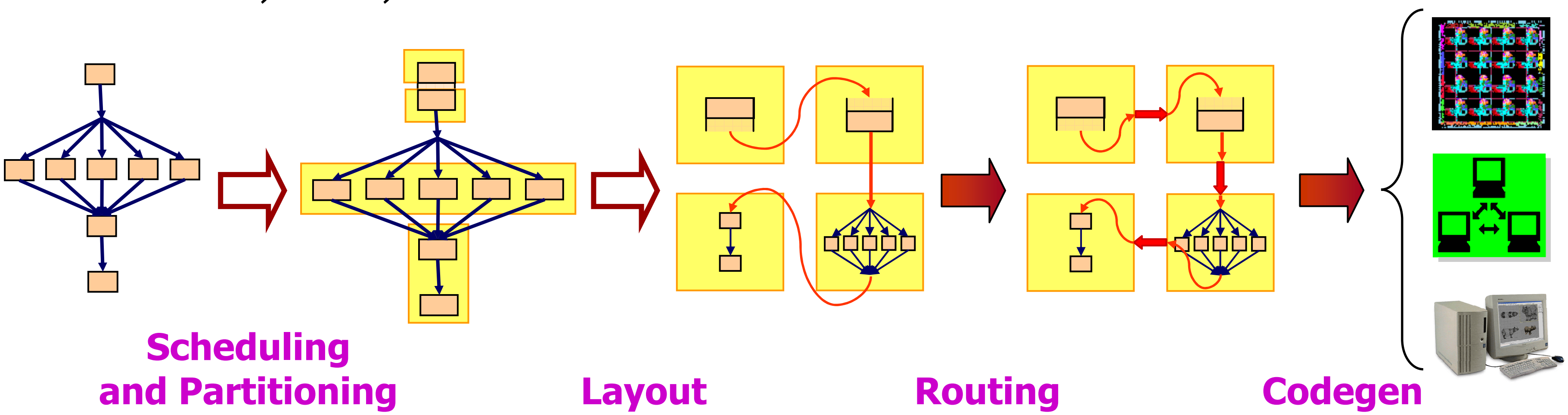
- Boost productivity, enable faster development and rapid prototyping



- Simple and effective optimizations for streams
- Targeting tiled architectures, clusters of workstations, DSPs, and traditional uniprocessors

Core Compilation Technology

- Focused on a common challenges in modern and future architectures
  - MIT Raw fabric architecture
  - Clusters of workstations
  - ARM, x86, and IA-64



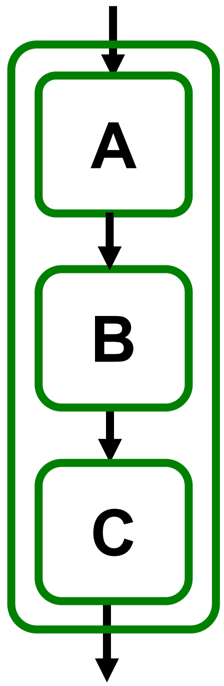
- Compiler’s role: map the computation and communication pattern to processors, memories, and communication substrates

Compiler Issues

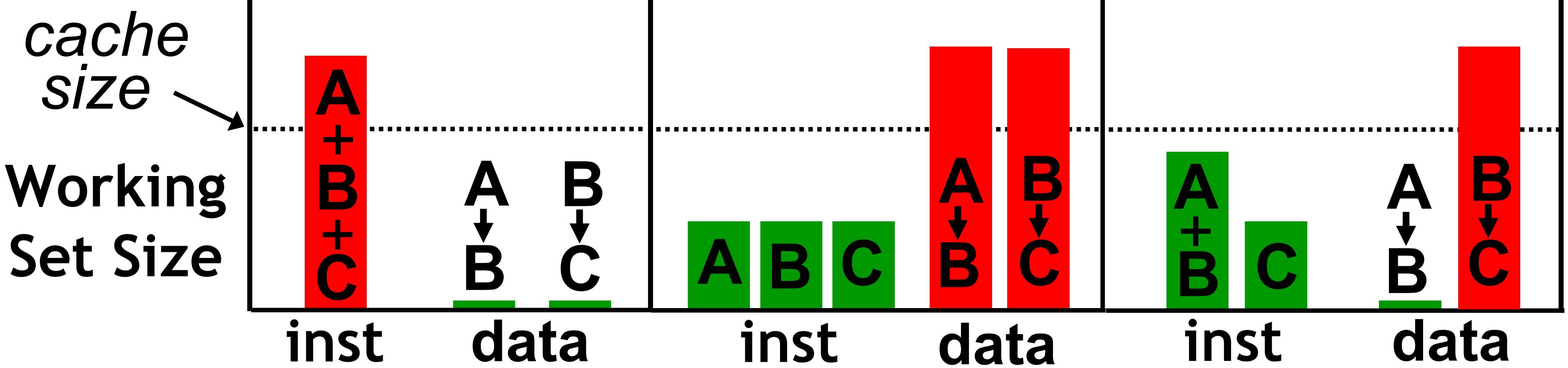
- Load balancing
- Resource utilization
- Fault tolerance
- Dynamic reconfiguration
- …

- In this talk: cache aware scheduling and partitioning [LCTES '05]

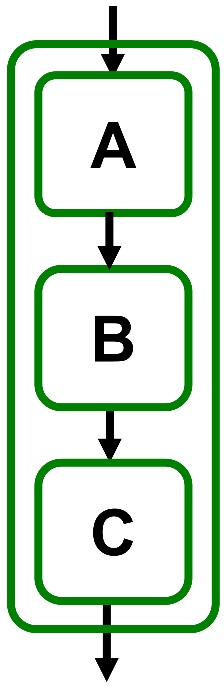
Example Cache Optimization



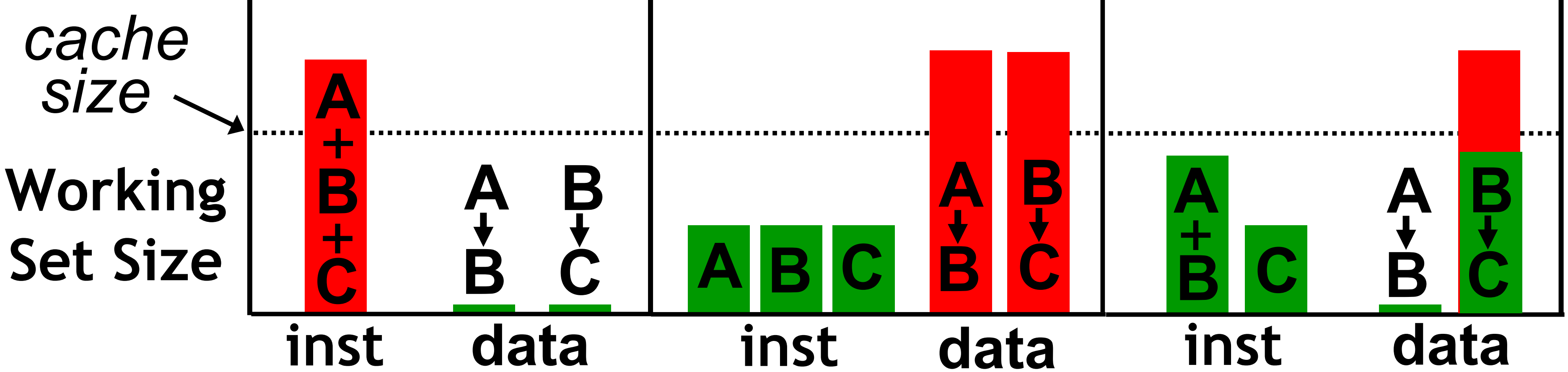
Baseline	Full Scaling	
for i = 1 to N A(); B(); C(); end	for i = 1 to N A(); for i = 1 to N B(); for i = 1 to N C();	for i = 1 to N A(); B(); end for i = 1 to N C();



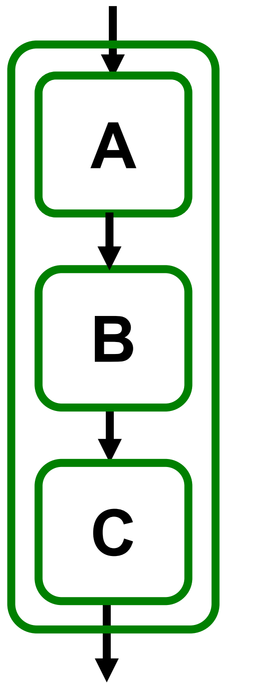
Example Cache Optimization



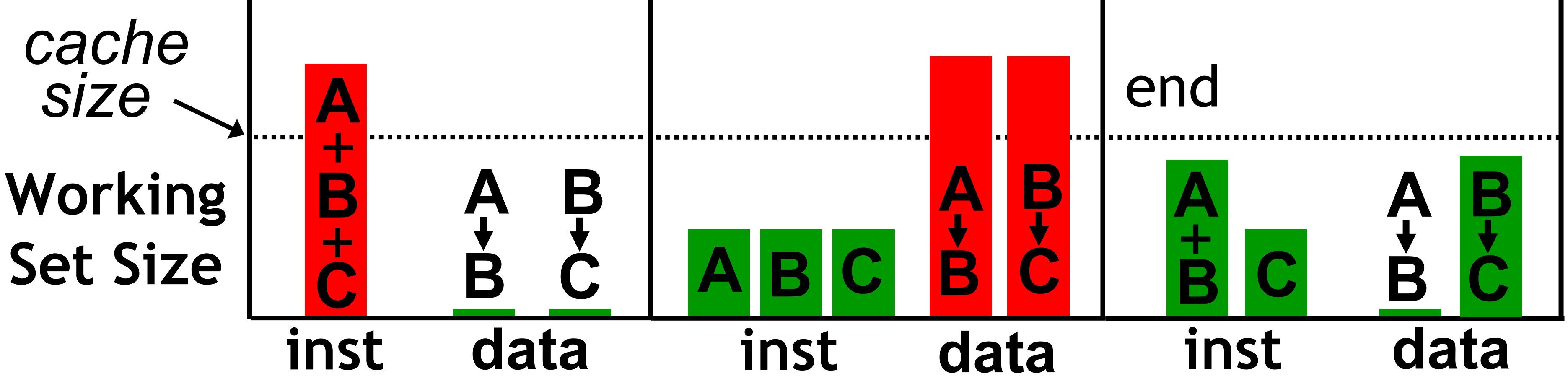
Baseline	Full Scaling	
for i = 1 to N A(); B(); C(); end	for i = 1 to N A(); for i = 1 to N B(); for i = 1 to N C();	for i = 1 to 64 A(); B(); end for i = 1 to 64 C();



Example Cache Optimization



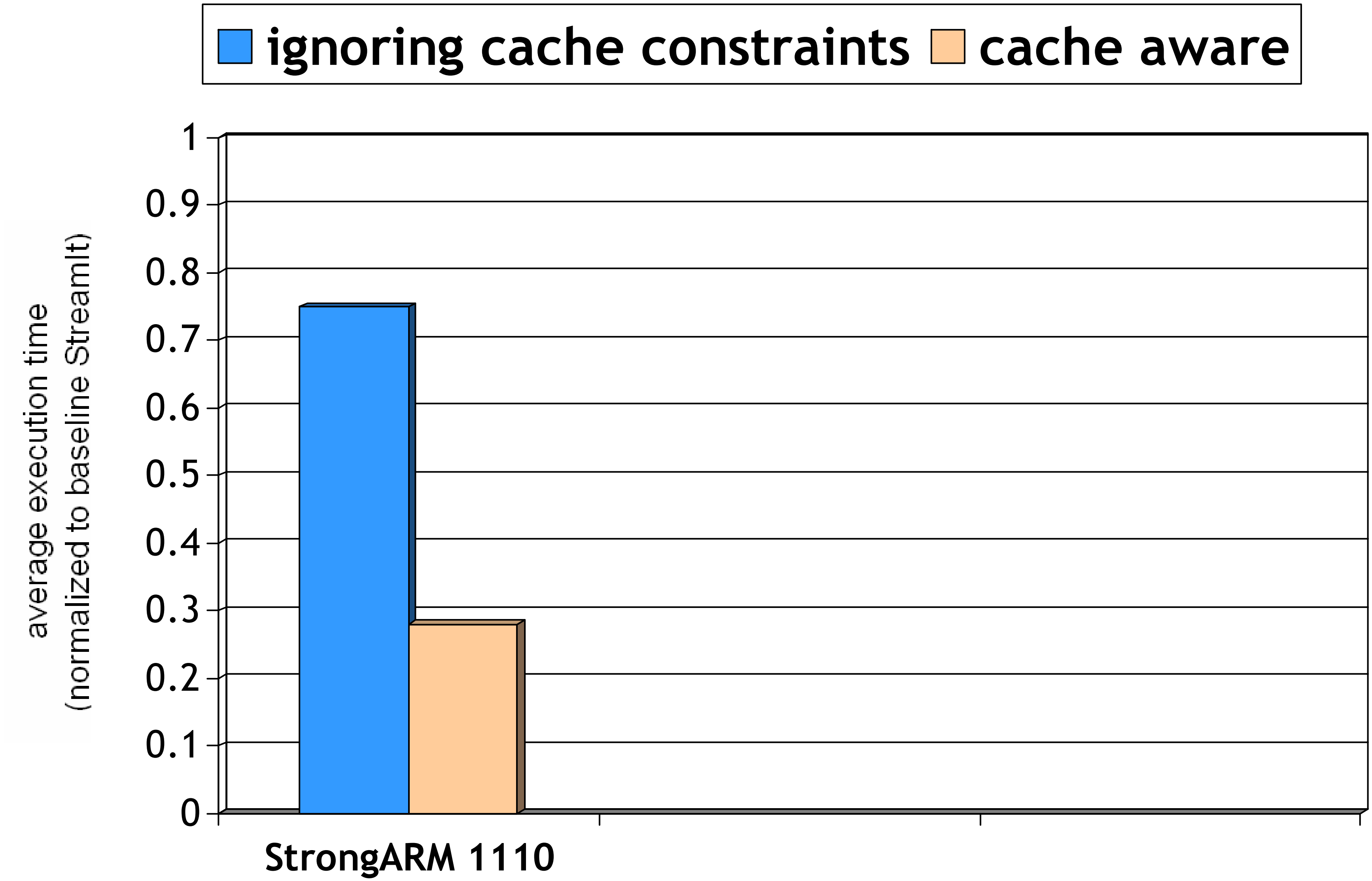
Baseline	Full Scaling	Cache Aware
for i = 1 to N A(); B(); C(); end	for i = 1 to N A(); for i = 1 to N B(); for i = 1 to N C();	for i = 1 to 64 A(); B(); end for i = 1 to 64 C(); end



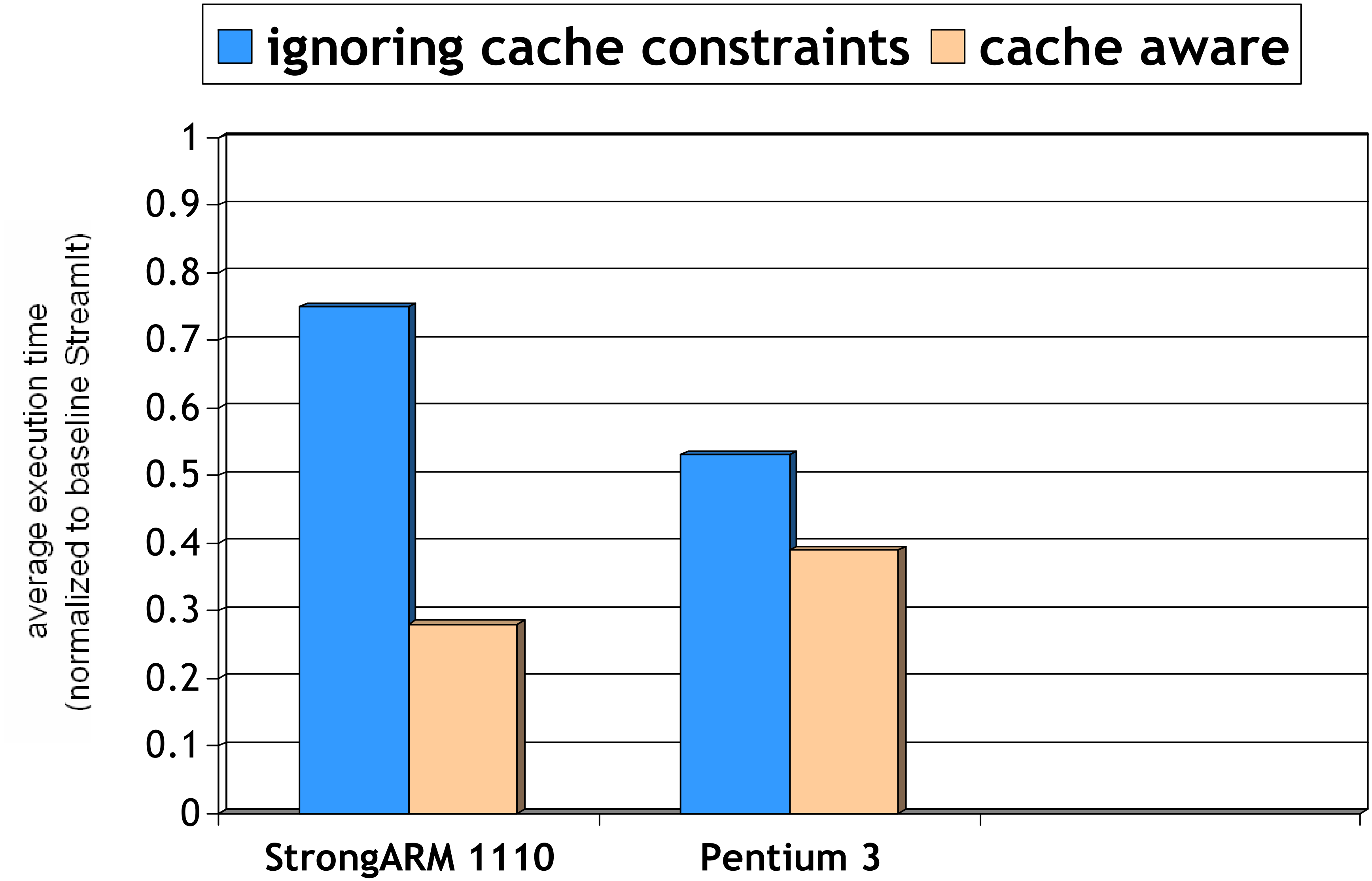
Evaluation Methodology

- StreamIt compiler generates C code
  - Baseline StreamIt optimizations
    - Unrolling, constant propagation
  - Compile C code with gcc-v3.4 with -O3 optimizations
- StrongARM 1110 (XScale) embedded processor
  - 370MHz, 16Kb I-Cache, 8Kb D-Cache
  - No L2 Cache (memory 100× slower than cache)
  - Median user time
- Also Pentium 3 and Itanium 2 processors
- Suite of 11 StreamIt Benchmarks

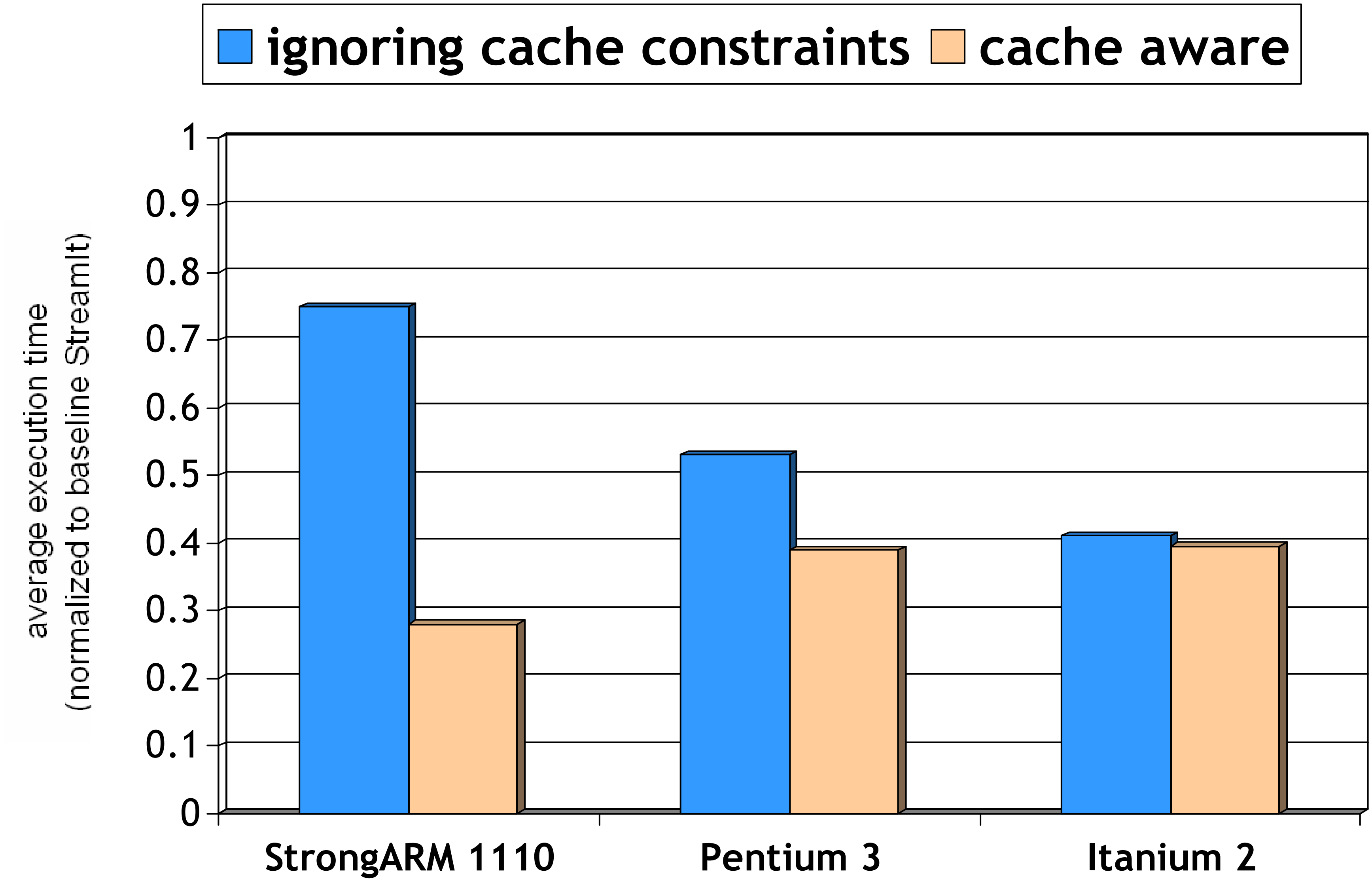
Cache Optimizations Results



Cache Optimizations Results

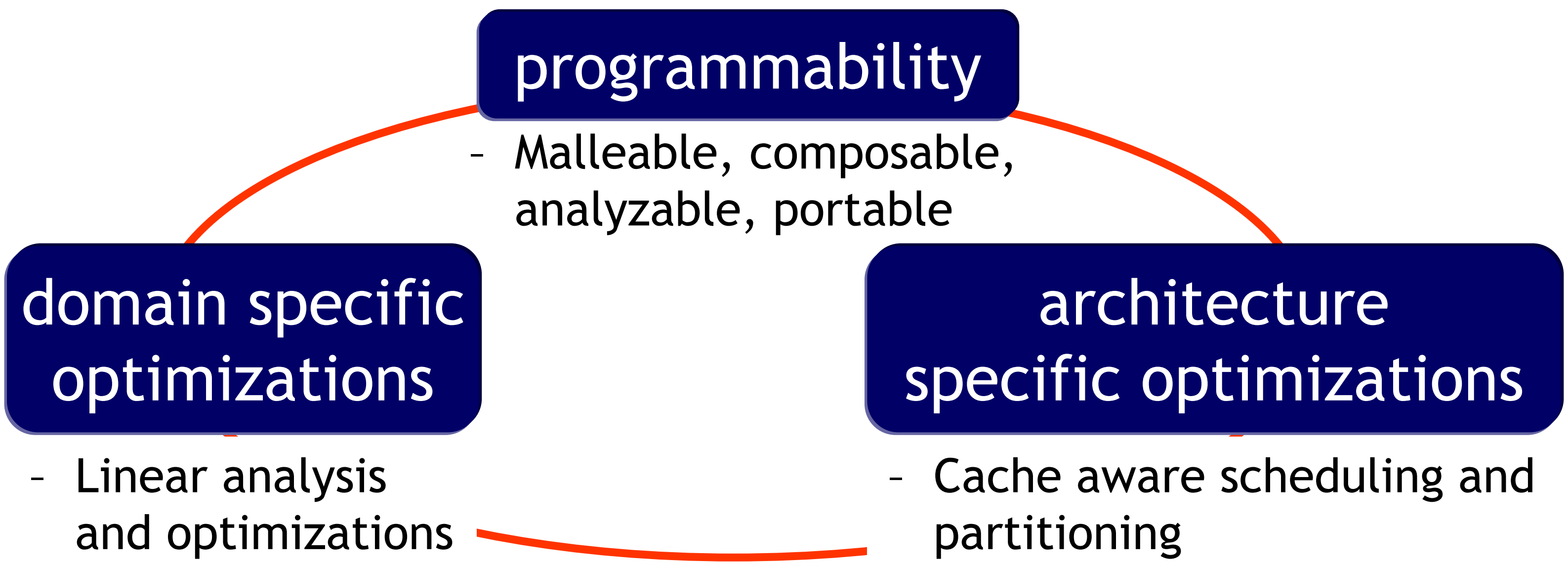


Cache Optimizations Results



Concluding Remarks

- StreamIt improves programmer productivity without compromising performance
  - Easily identify pipeline and data parallelism
  - Expose information for domain specific and architecture specific optimizations



Broader Impact

- Integration into future HPCS languages
  - IBM: X10
- StreamIt for graphics applications
  - Programmable graphics pipeline [Graphics Hardware '05]
- StreamIt for emerging architectures
- Looking for users with interesting applications

High-Productivity Stream Programming for High-Performance Systems

Rodric Rabbah, Bill Thies, Michael Gordon, Janis Sermulins, and Saman Amarasinghe
Massachusetts Institute of Technology

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

HPEC 2005, MIT LL