Cache Aware Optimization of Stream Programs

Janis Sermulins, William Thies, Rodric Rabbah and Saman Amarasinghe

LCTES
Chicago, June 2005

Streaming Computing Is Everywhere!

- Prevalent computing domain with applications in embedded systems
  - As well as desktops and high-end servers



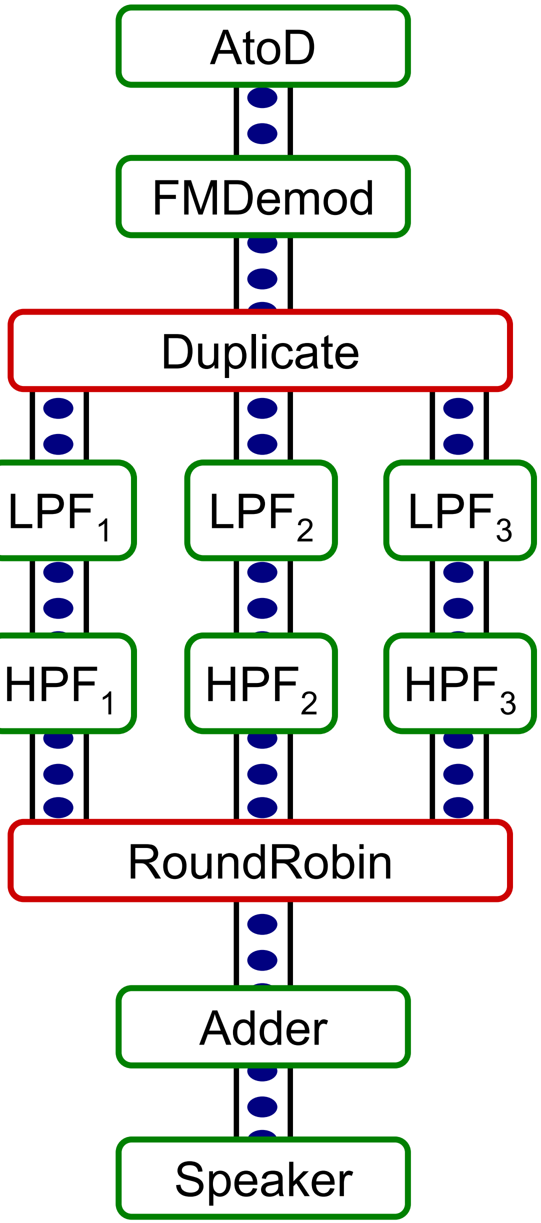







Properties of Stream Programs

- Regular and repeating computation
- Independent actors with explicit communication
- Data items have short lifetimes



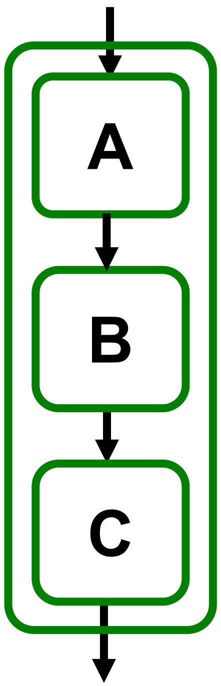
Application Characteristics: Implications on Caching

	Scientific	**Streaming**
Control	Inner loops	Single outer loop
Data	Persistent array processing	Limited lifetime producer-consumer
Working set	Small	Whole-program
Implications	Natural fit for cache hierarchy	Demands novel mapping

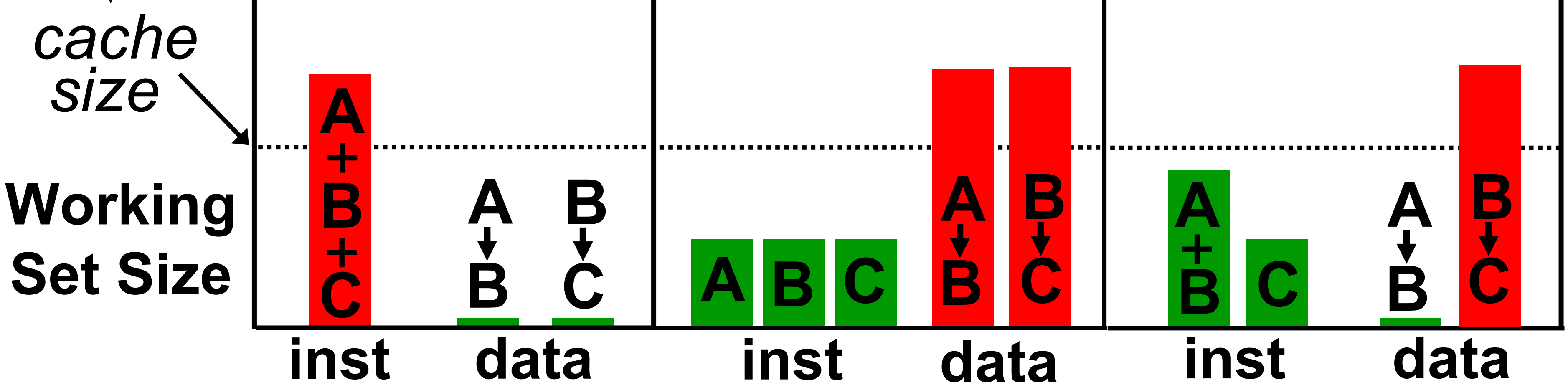
Application Characteristics: Implications on Compiler

	Scientific	**Streaming**
Parallelism	Fine-grained	Coarse-grained
Data access	Global	Local
Communication	Implicit random access	Explicit producer-consumer
Implications	Limited program transformations	Potential for global reordering

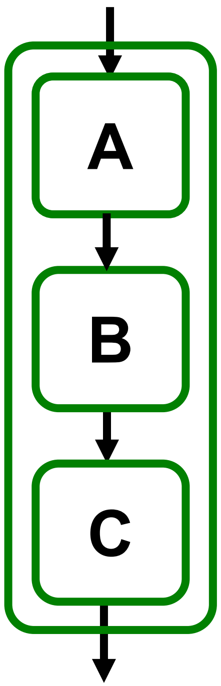
Motivating Example



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**();

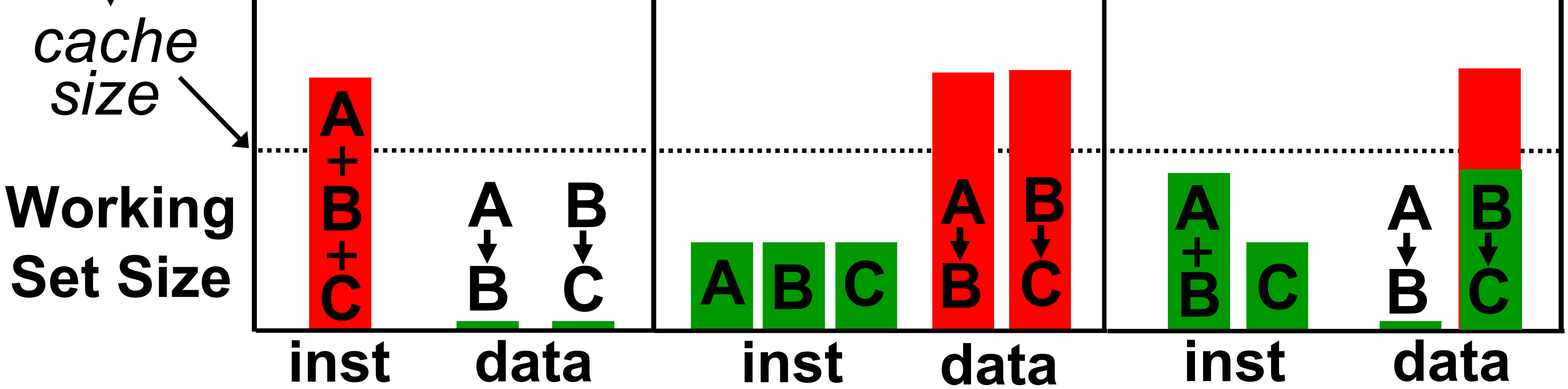


Motivating Example

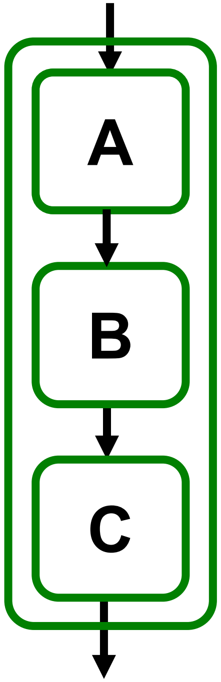


Baseline	Full Scaling	

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();

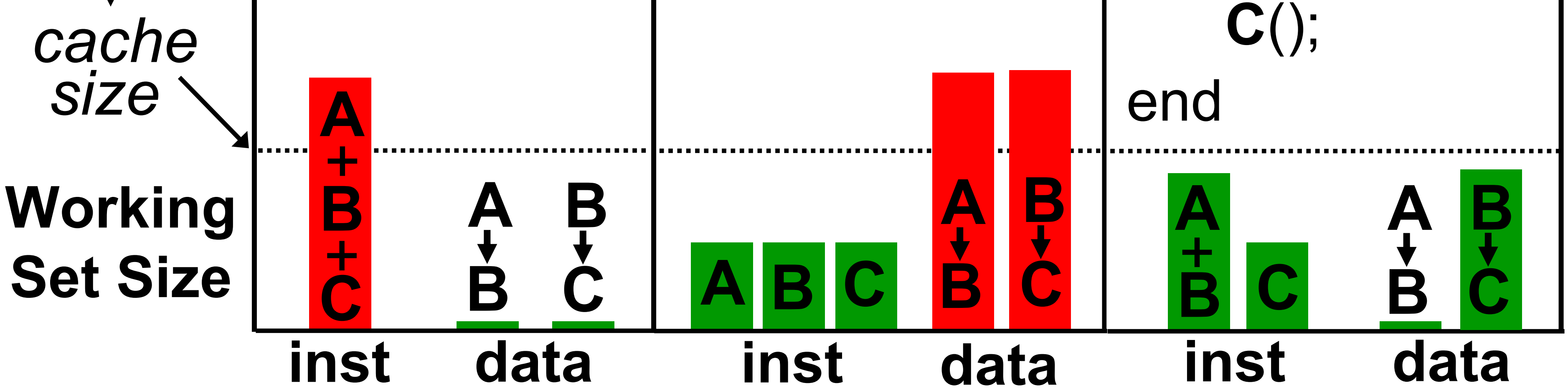


Motivating Example



Baseline	Full Scaling	Cache Opt

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/64
		  for i = 1 to 64
		    A();
		    B();
		  end
		  for i = 1 to 64
		    C();
		end

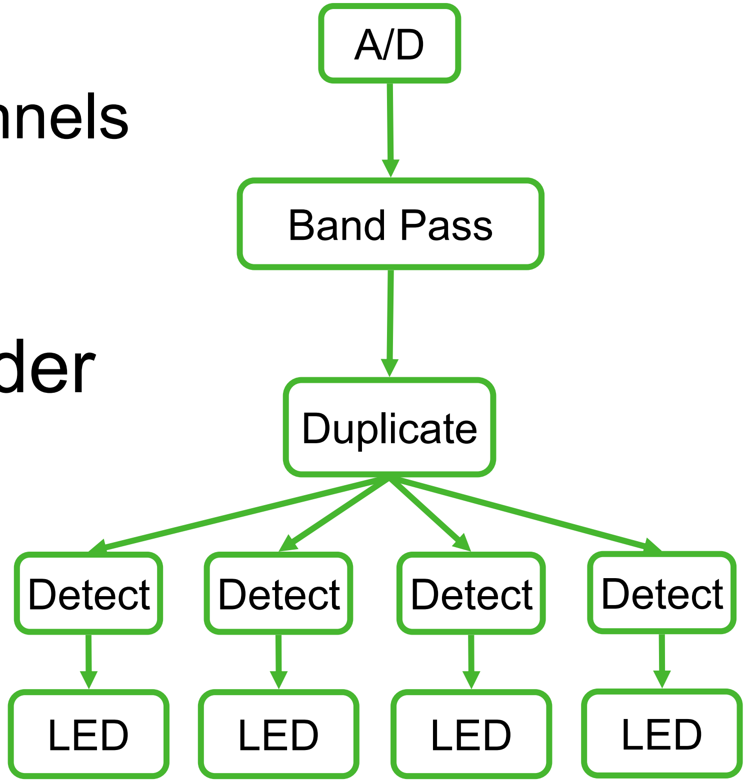


Outline

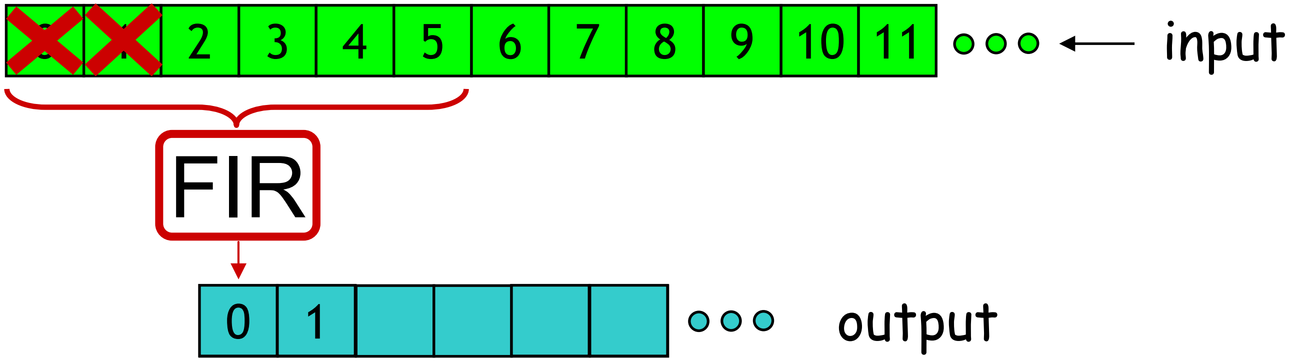
- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion

Model of Computation

- Synchronous Dataflow [Lee 92]
  - Graph of autonomous filters
  - Communicate via FIFO channels
  - Static I/O rates
- Compiler decides on an order of execution (schedule)
  - Many legal schedules
  - Schedule affects locality
  - Lots of previous work on minimizing buffer requirements between filters



Example StreamIt Filter

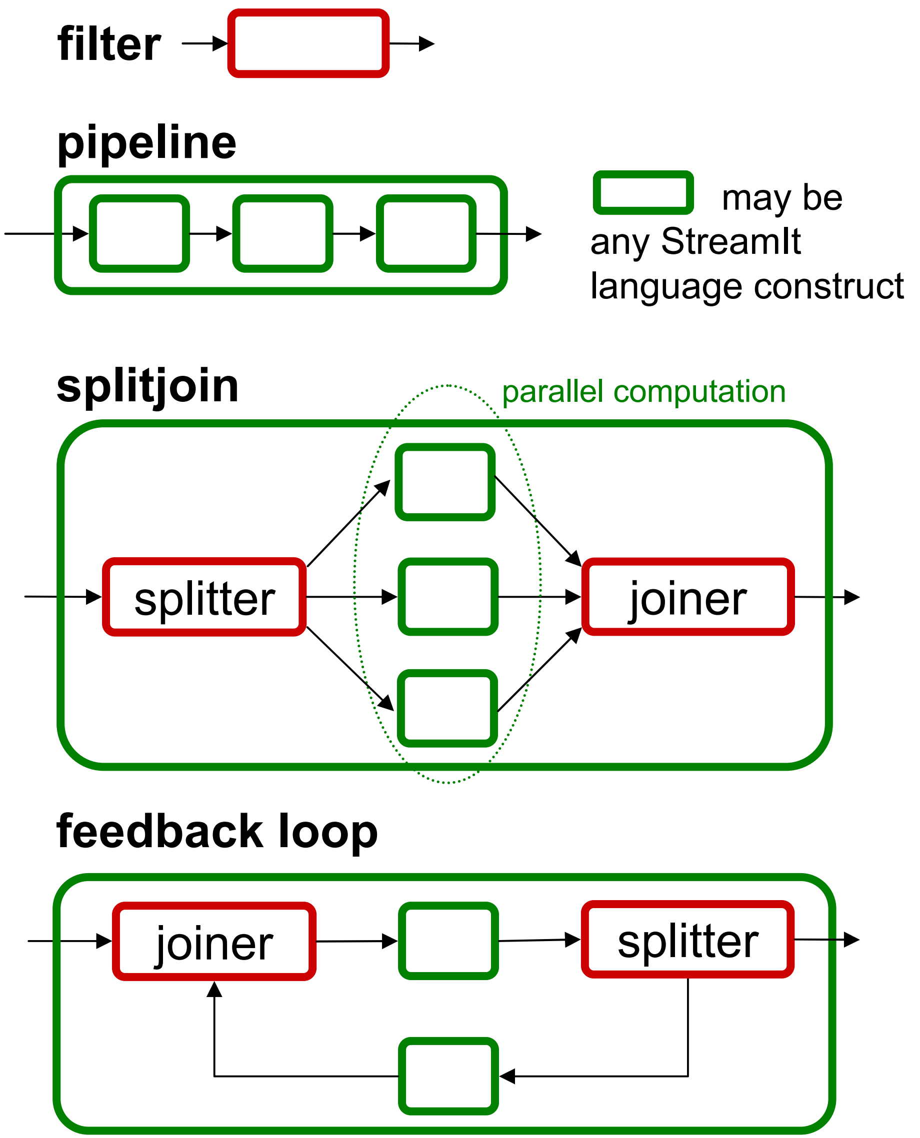



```
float→float filter FIR (int N) {

    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();
    }
}
```

StreamIt Language Overview

- StreamIt is a novel language for streaming
  - Exposes parallelism and communication
  - Architecture independent
  - Modular and composable
    - Simple structures composed to creates complex graphs
  - Malleable
    - Change program behavior with small modifications

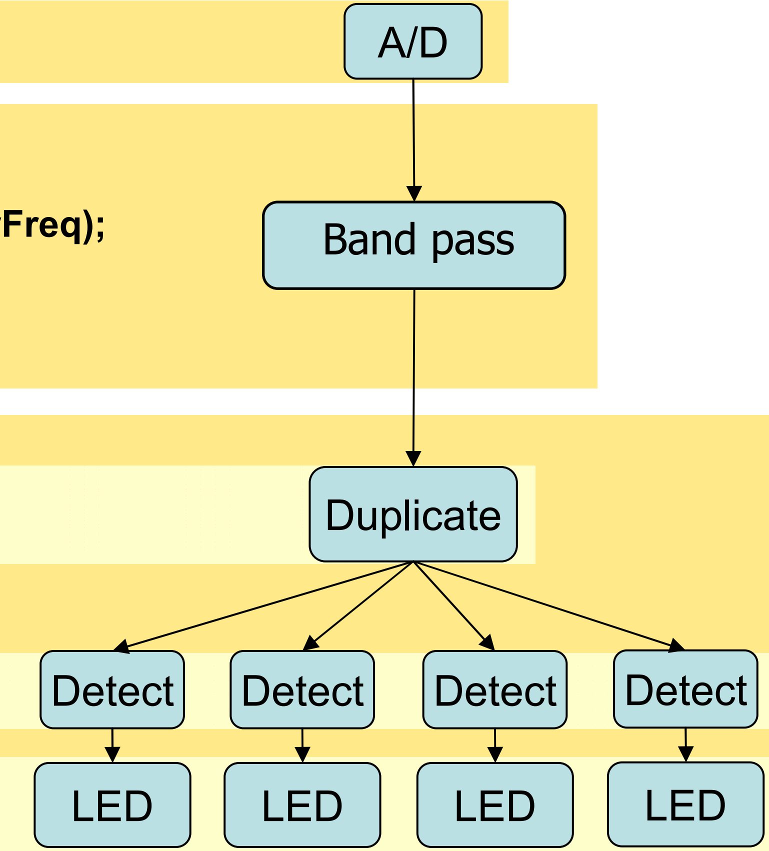


Freq Band Detector in StreamIt

```
void->void pipeline FrequencyBand {
  float sFreq = 4000;
  float cFreq = 500/(sFreq*2*pi);
  float  wFreq = 100/(sFreq*2*pi);
  add D2ASource(sFreq);

  add BandPassFilter(100, cFreq-wFreq, cFreq+wFreq);

  add splitjoin {
    split duplicate;
    for (int i=0; i<4; i++) {
      add pipeline {
        add Detect (i/4);
        add LED (i);
      }
    }
    join roundrobin(0);
  }
}
```

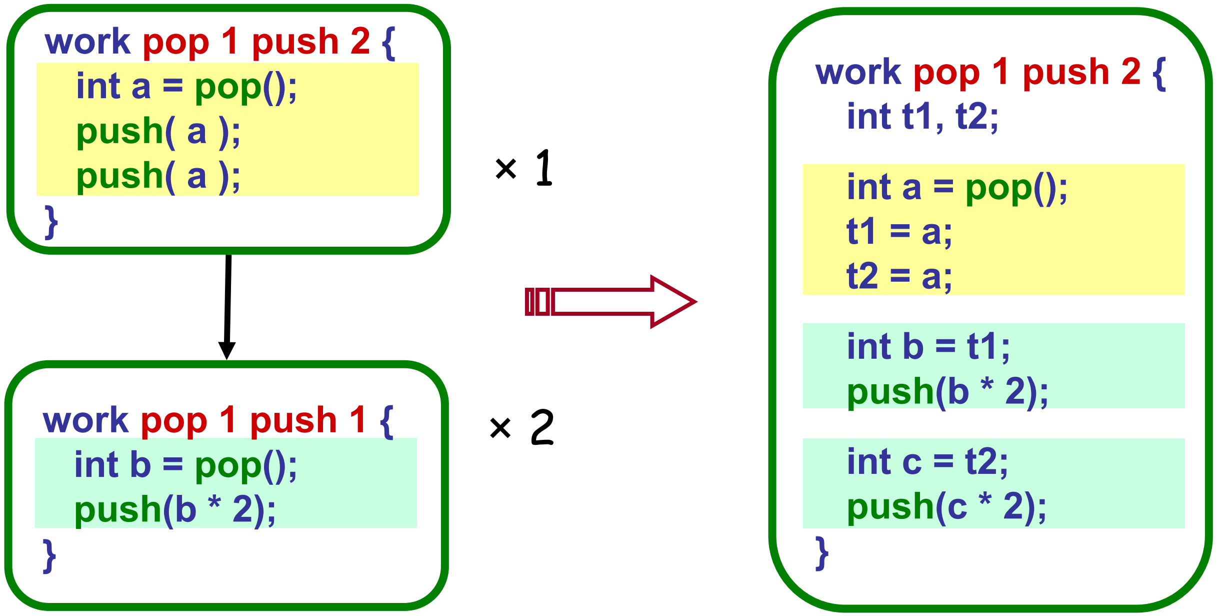


Outline

- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion

Fusion

- Fusion combines adjacent filters into a single filter



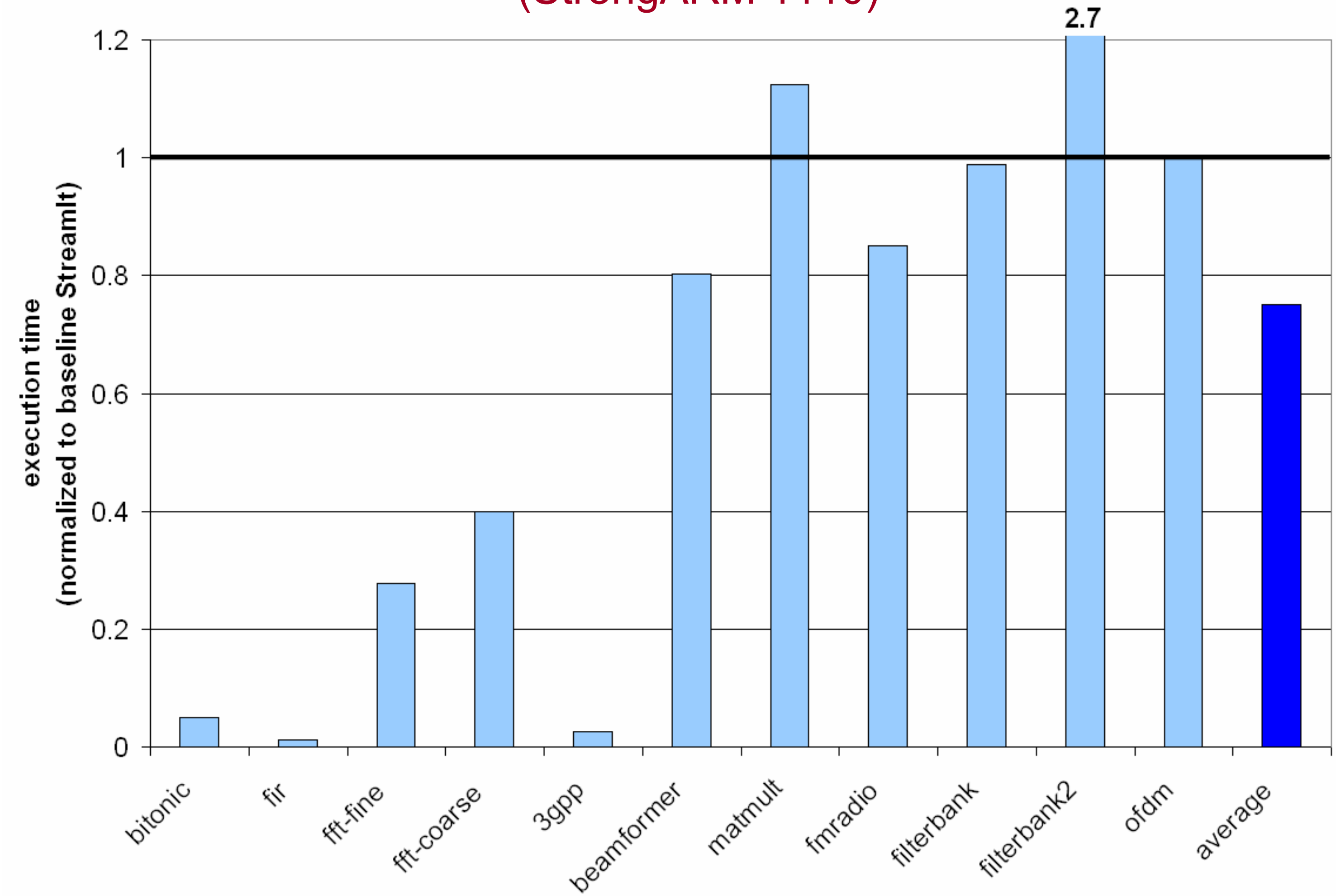
- Reduces method call overhead
- Improves producer-consumer locality
- Allows optimizations across filter boundaries
  - Register allocation of intermediate values
  - More flexible instruction scheduling

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
- Suite of 11 StreamIt Benchmarks
- Evaluate two fusion strategies:
  - Full Fusion
  - Cache Aware Fusion

Results for Full Fusion

(StrongARM 1110)

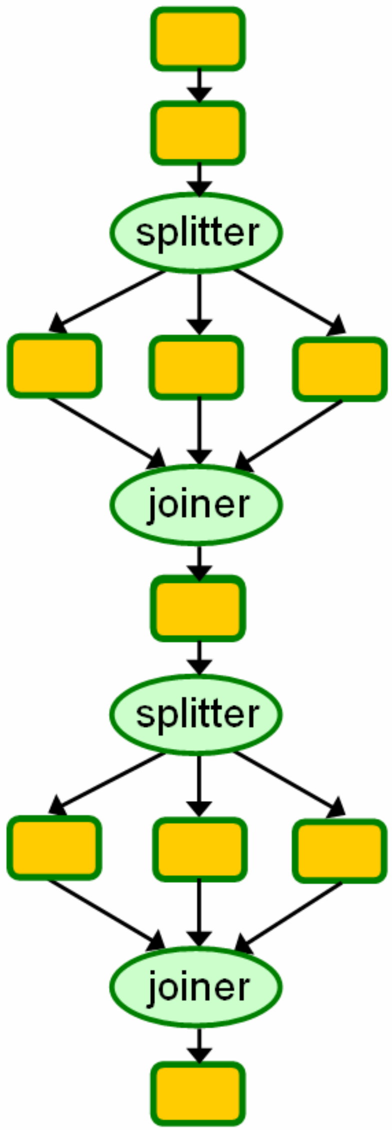


Hazard: The instruction or data working set of the fused program may exceed cache size!

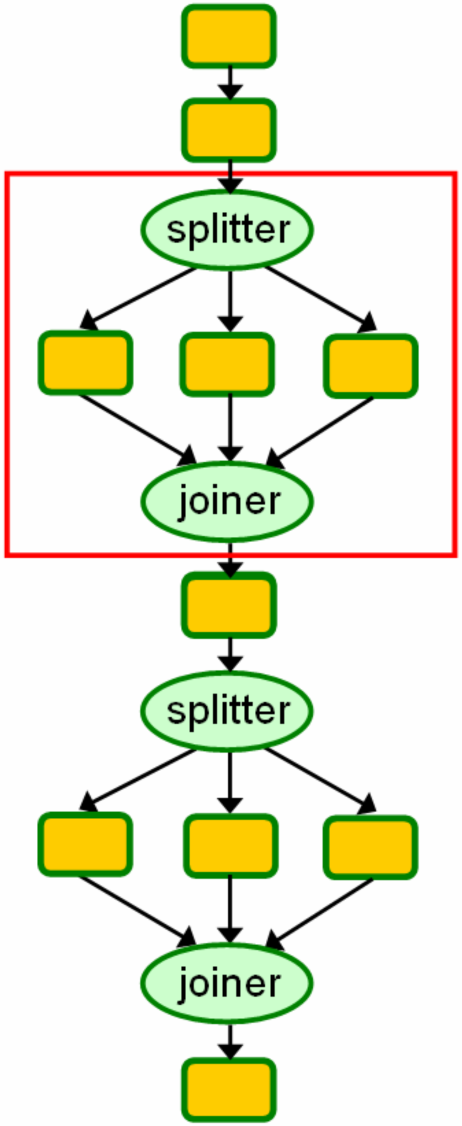
Cache Aware Fusion (CAF)

- Fuse filters so long as:
  - Fused instruction working set fits the I-cache
  - Fused data working set fits the D-cache
- Leave a fraction of D-cache for input and output to facilitate cache aware scaling
- Use a hierarchical fusion heuristic

Hierarchical Fusion Heuristic

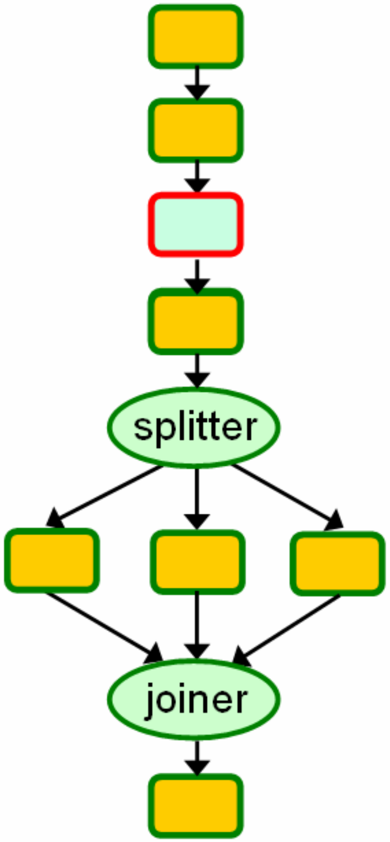


Hierarchical Fusion Heuristic

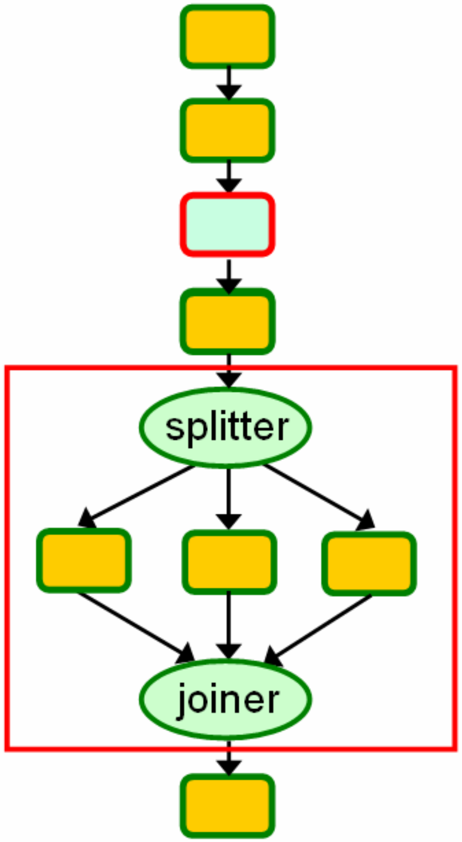


Does splitjoin fit in cache? Yes!

Hierarchical Fusion Heuristic

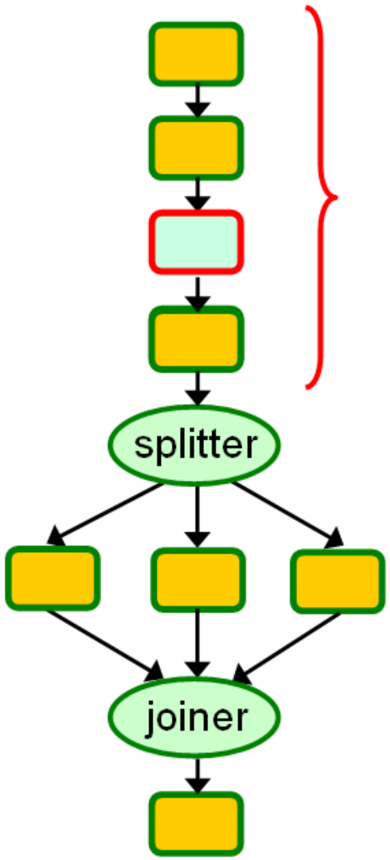


Hierarchical Fusion Heuristic



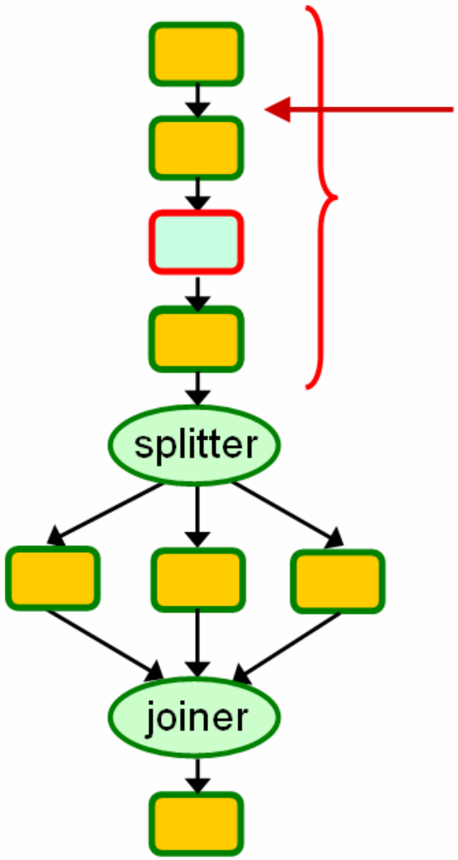
Does splitjoin fit in cache? No!

Hierarchical Fusion Heuristic



Does pipeline segment fit in cache? No!

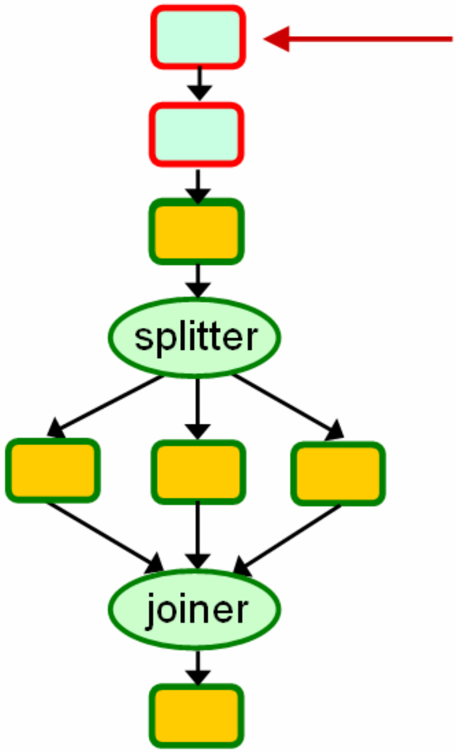
Hierarchical Fusion Heuristic



Identify highest bandwidth connection, fuse greedily

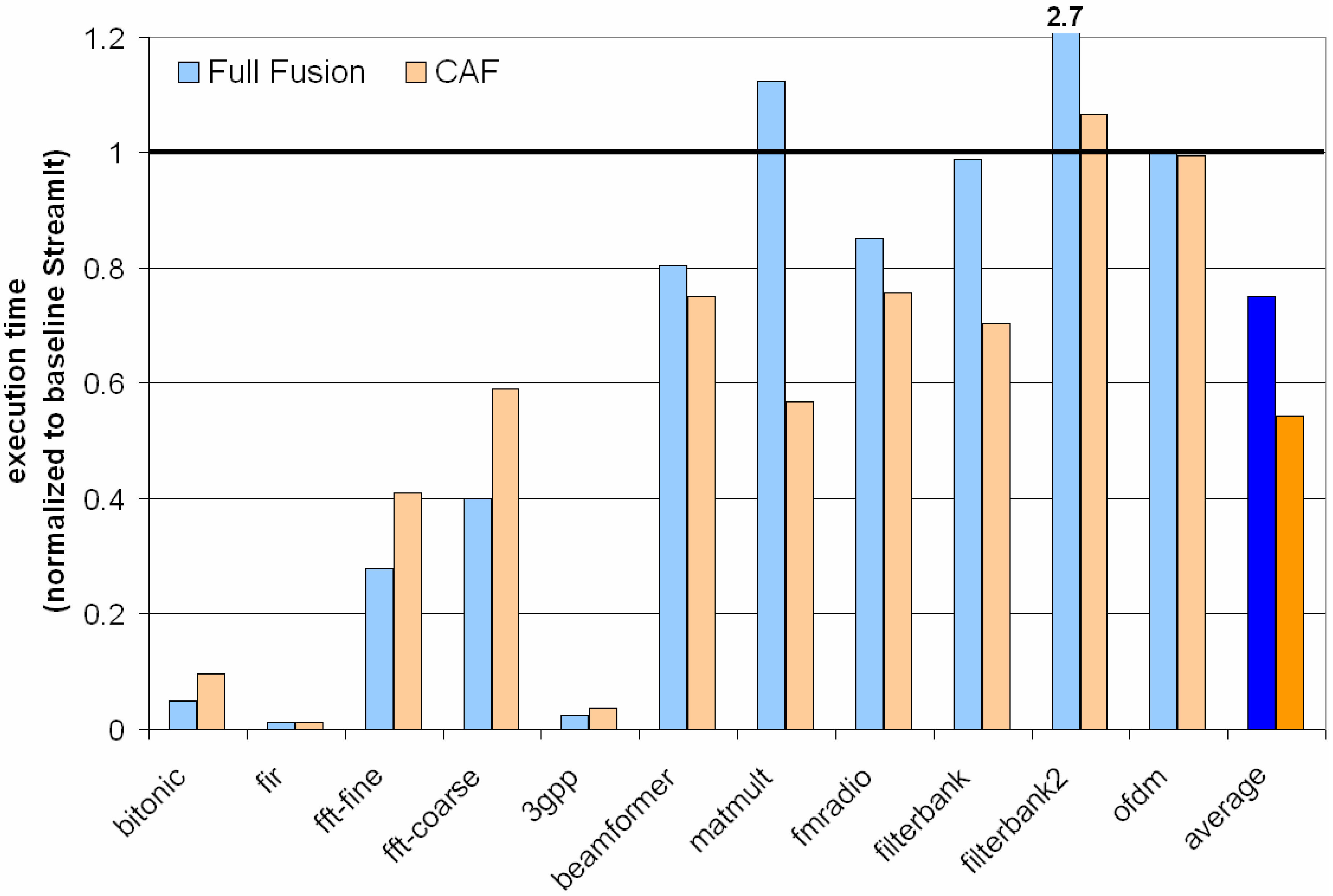
Does pipeline segment fit in cache? No!

Hierarchical Fusion Heuristic



Identify highest bandwidth connection, fuse greedily

Full Fusion vs. CAF

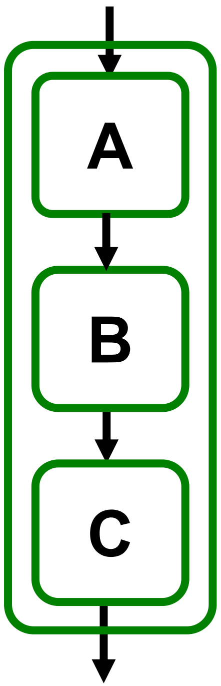


Outline

- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion

Improving Instruction Locality

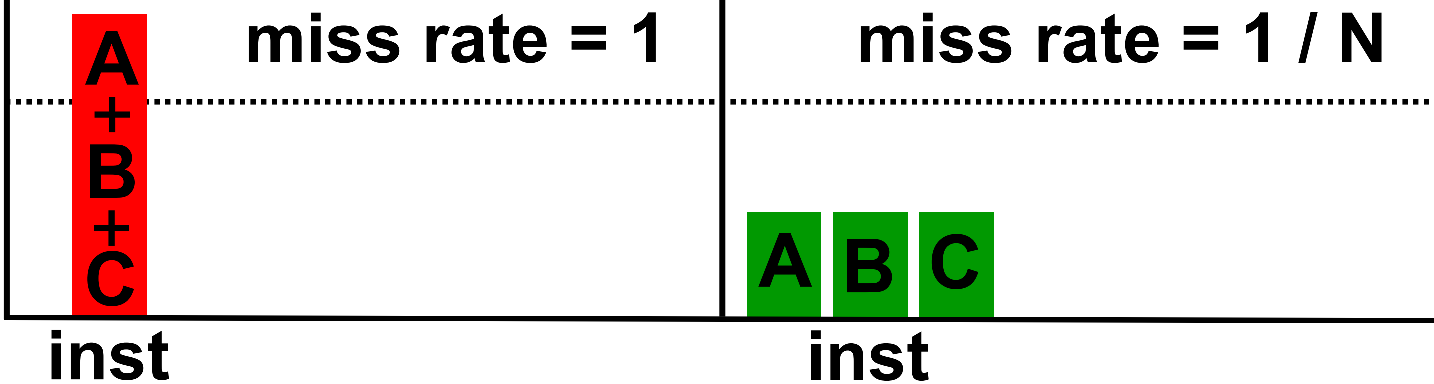
✗ cache miss ✓ cache hit



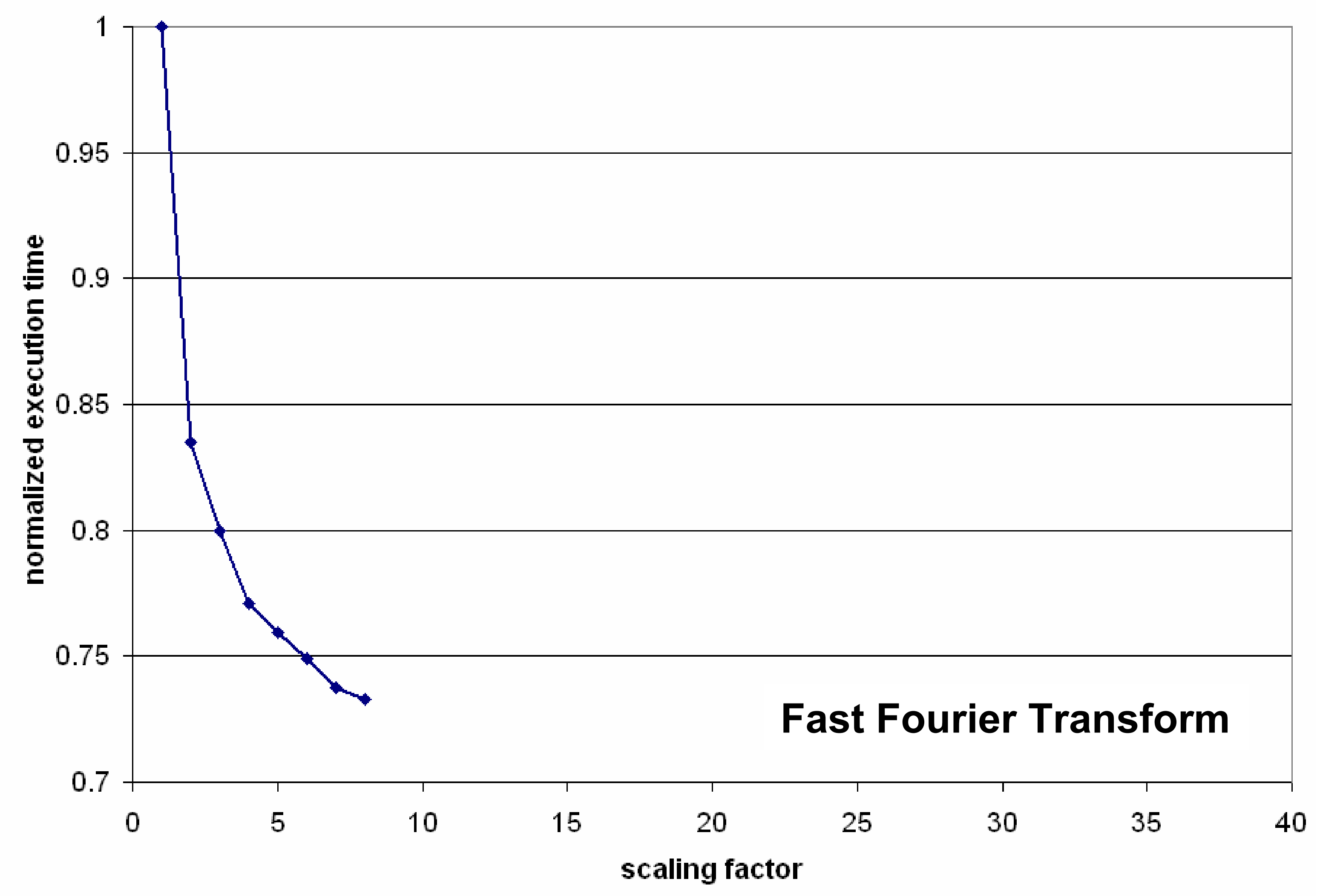
Baseline	Full Scaling
for i = 1 to N	for i = 1 to N
A(); ✗ ✗ ✗ ...	**A**(); ✗ ✓ ✓ ✓ ...
B(); ✗ ✗ ✗ ...	for i = 1 to N
C(); ✗ ✗ ✗ ...	**B**(); ✗ ✓ ✓ ✓ ...
end	for i = 1 to N
	C(); ✗ ✓ ✓ ✓ ...
miss rate = 1	**miss rate = 1 / N**

cache size

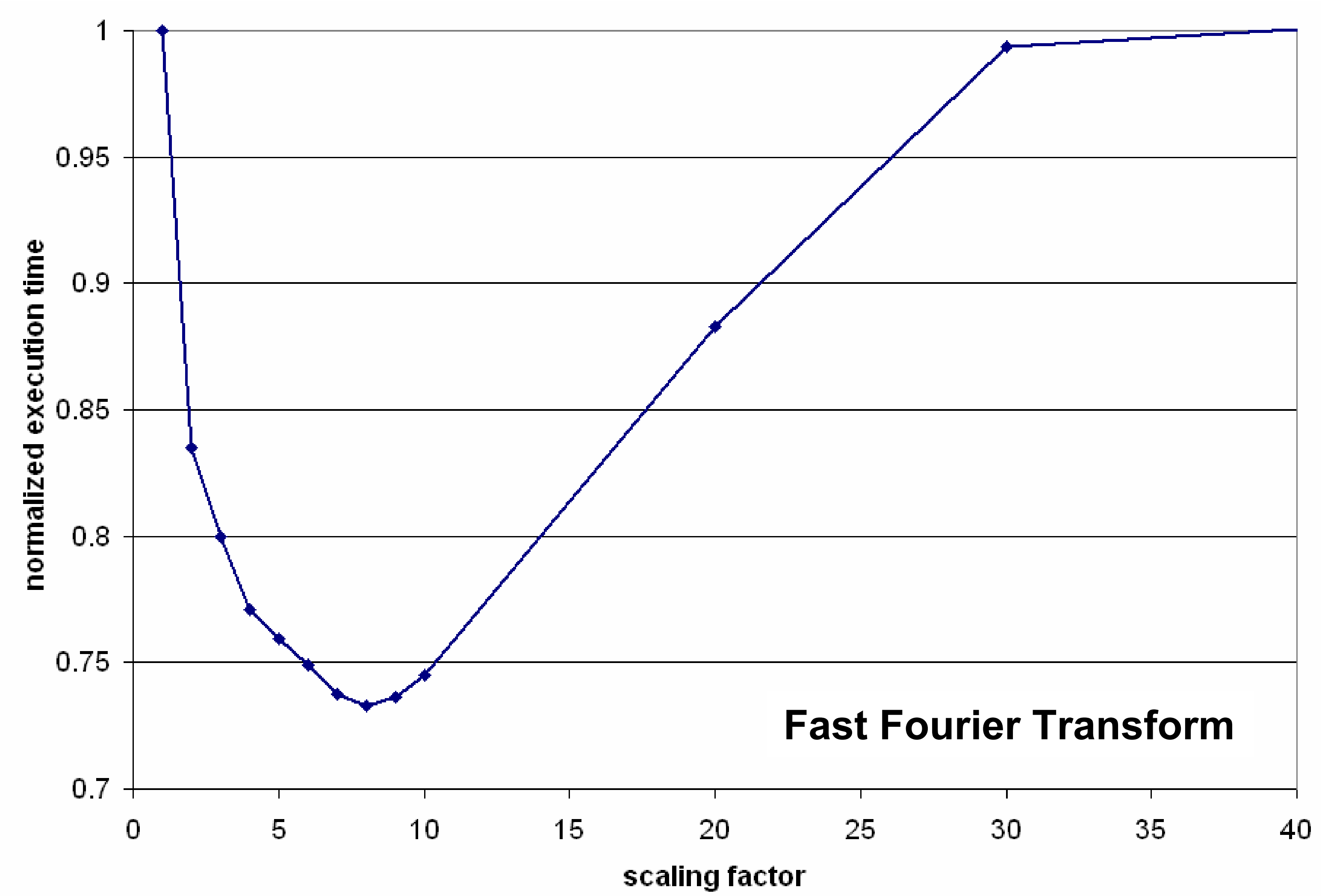
Working Set Size



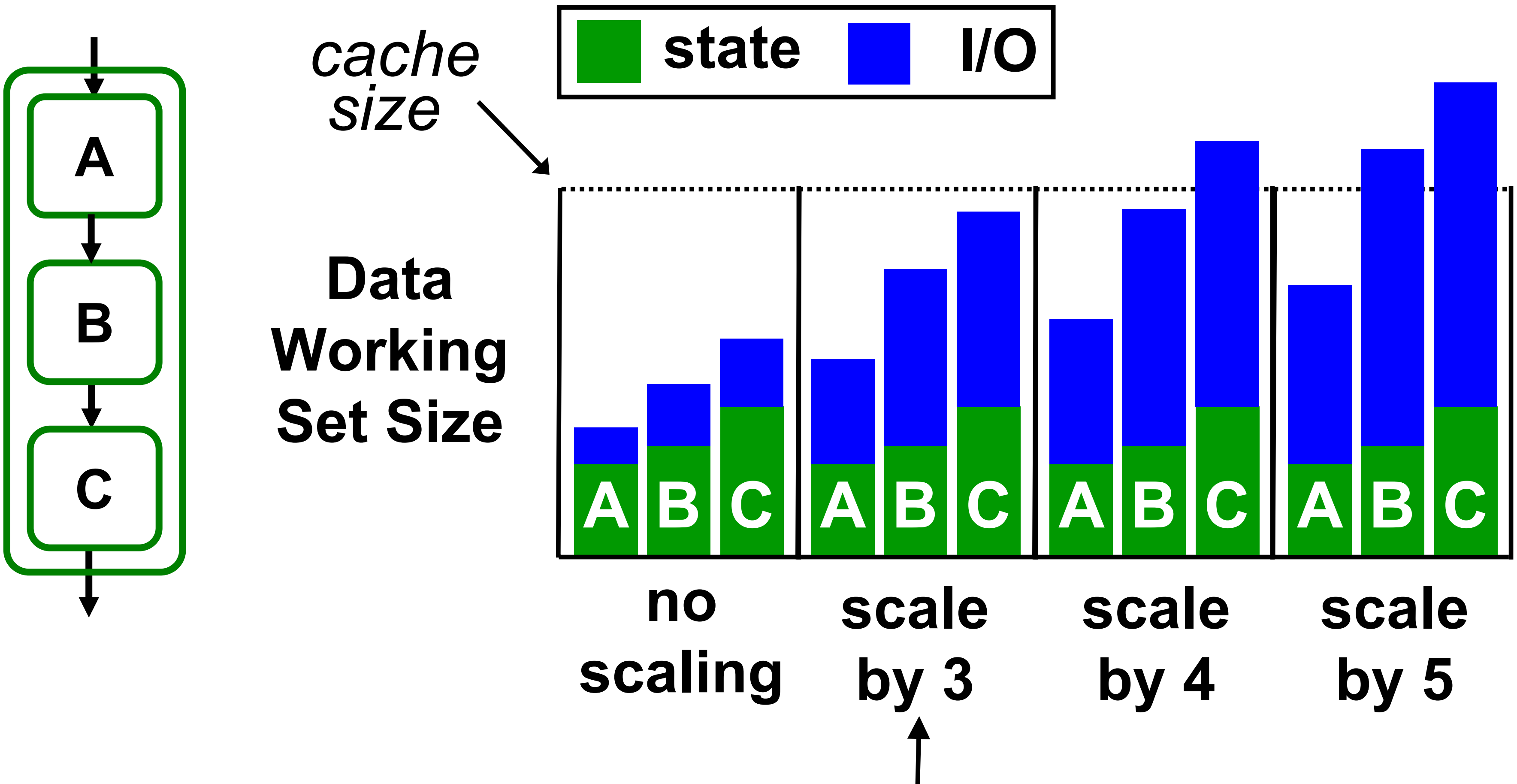
Impact of Scaling



Impact of Scaling



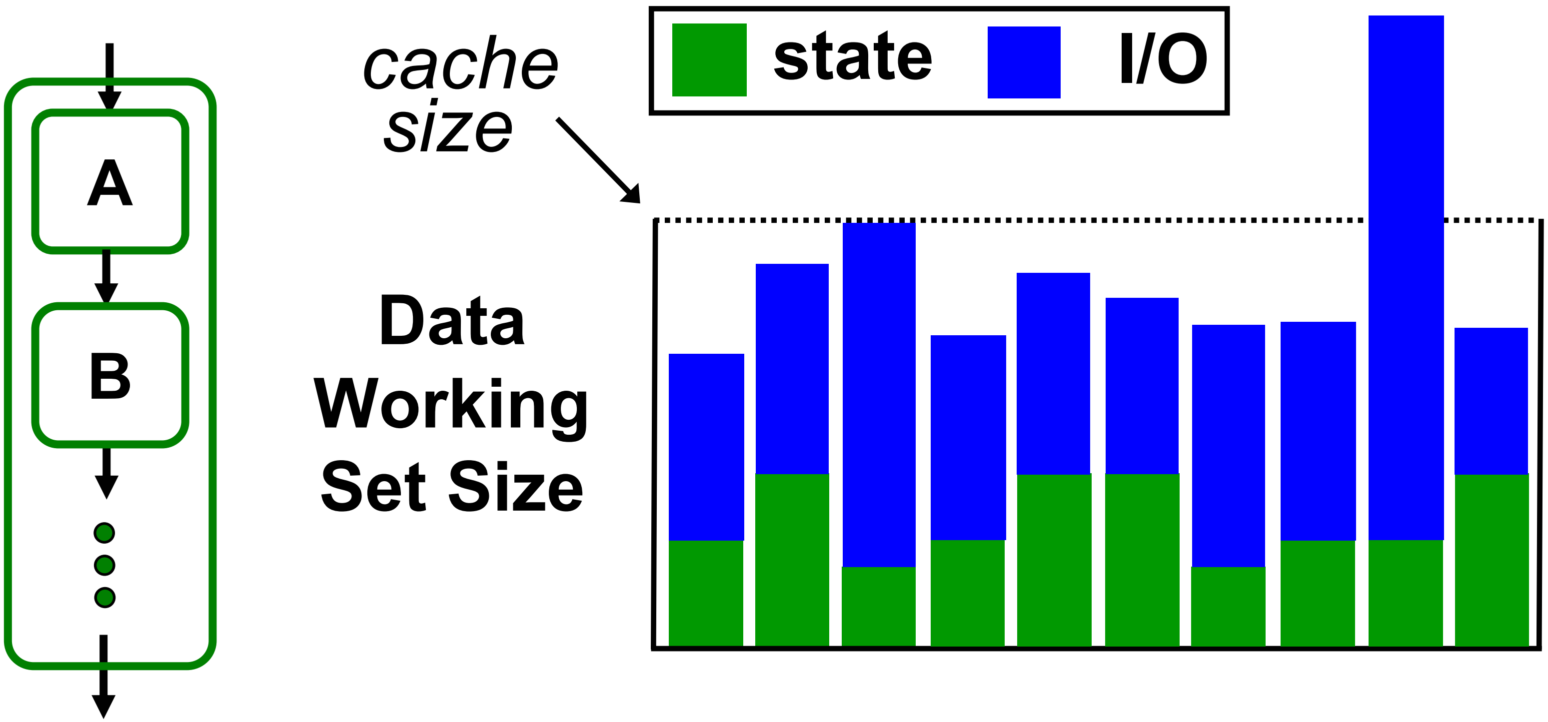
How Much To Scale?



Our Scaling Heuristic:

- Scale as much as possible
- Ensure at least 90% of filters have data working sets that fit into cache

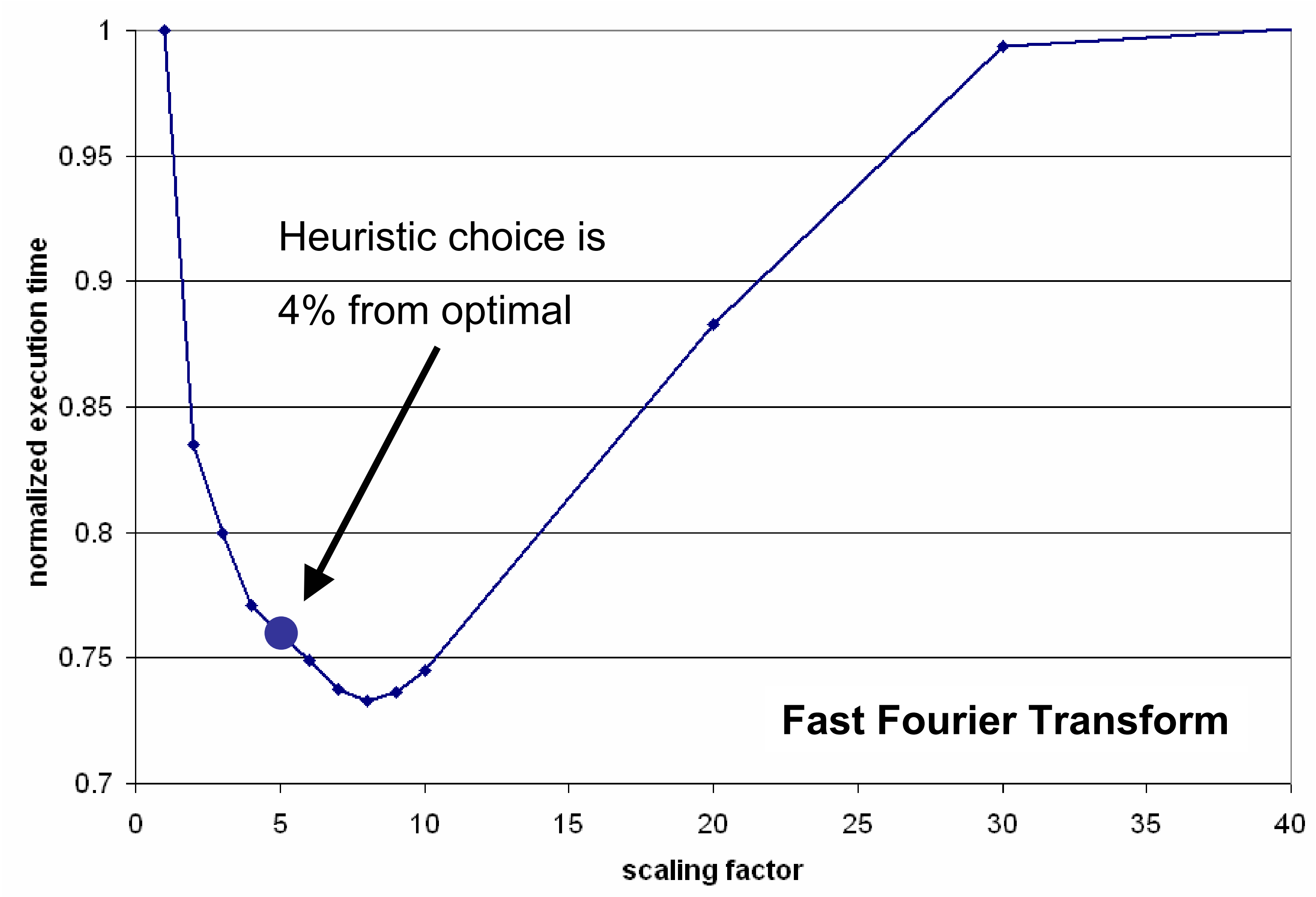
How Much To Scale?



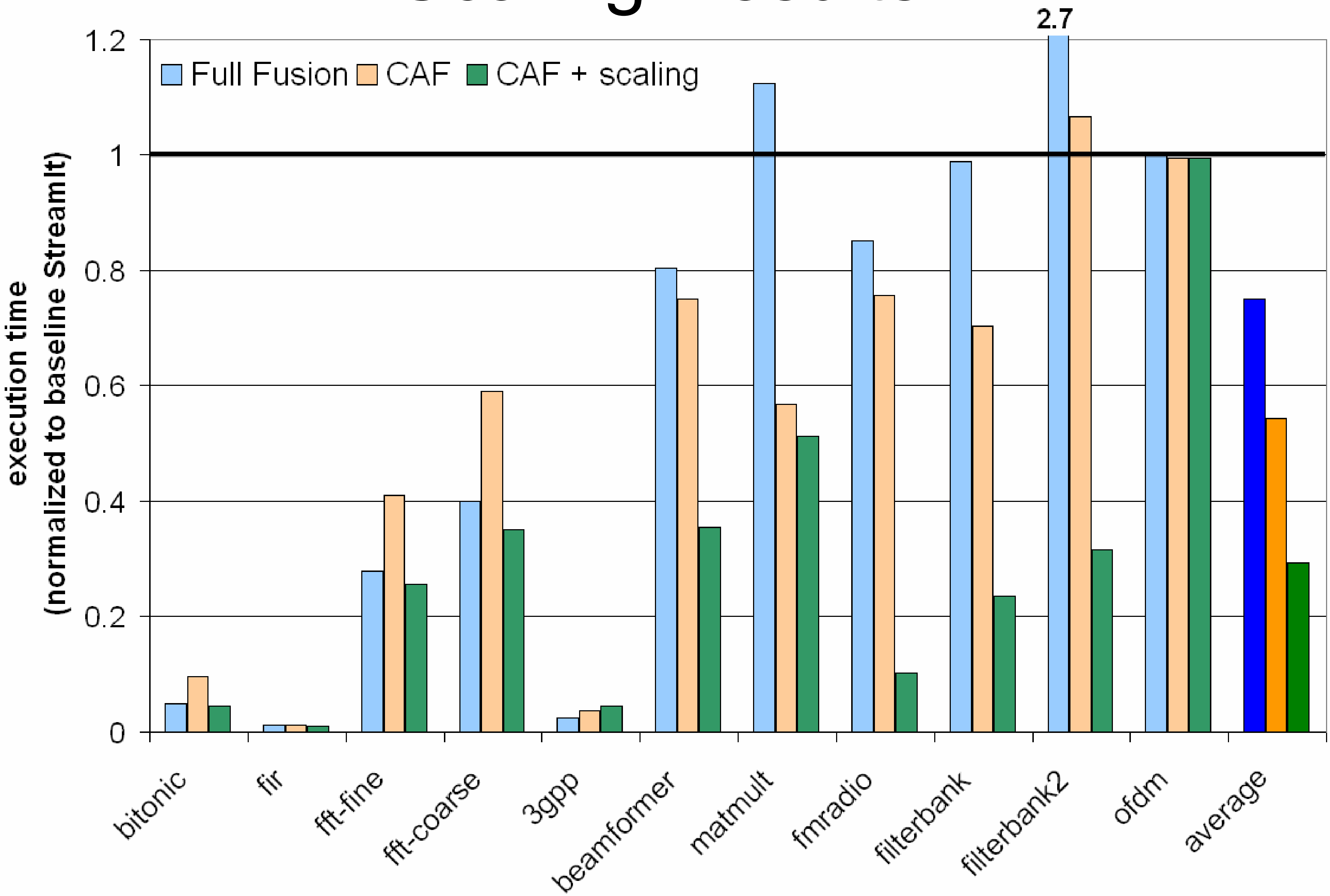
Our Scaling Heuristic:

- Scale as much as possible
- Ensure at least 90% of filters have data working sets that fit into cache

Impact of Scaling



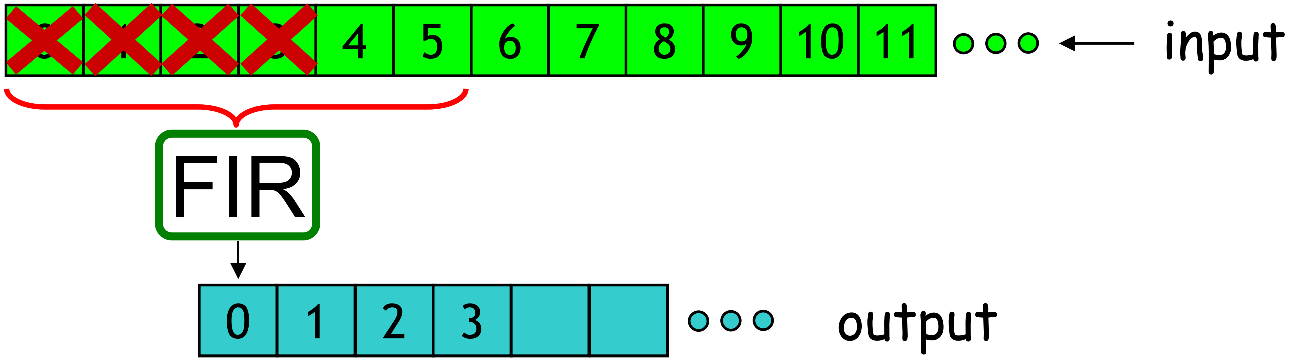
Scaling Results



Outline

- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion

Sliding Window Computation

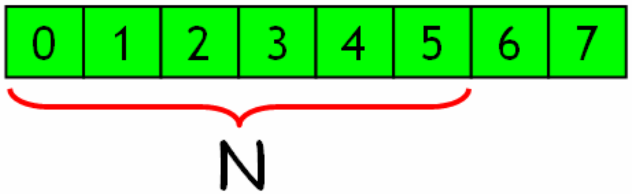



```
float→float filter FIR (int N) {

    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();
    }
}
```

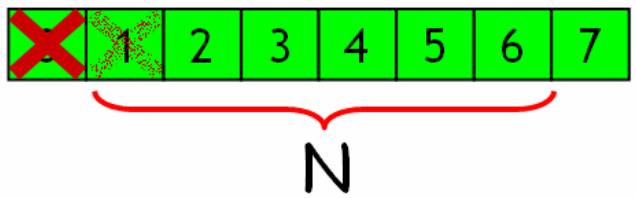
Buffer Management

Circular Buffer:



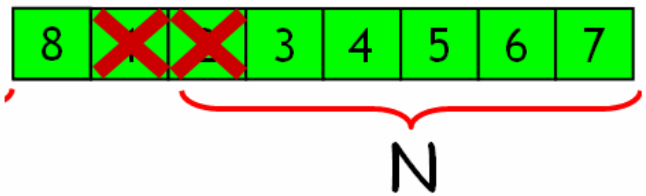
Buffer Management

Circular Buffer:



Buffer Management

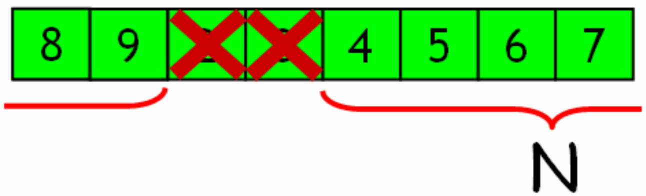
Circular Buffer:



peek(i) → A[(head + i) mod 8]

Buffer Management

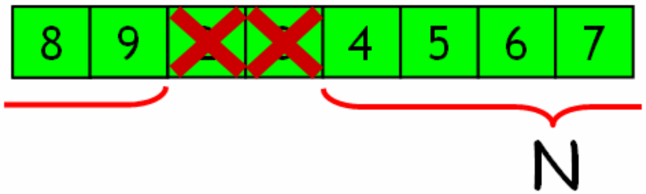
Circular Buffer:



peek(i) → A[(head + i) mod 8]

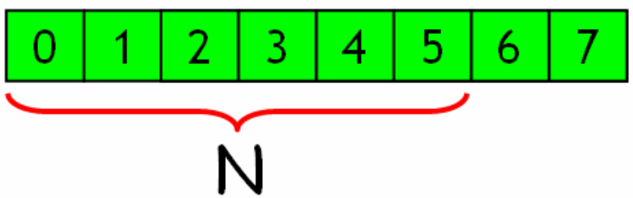
Buffer Management

Circular Buffer:



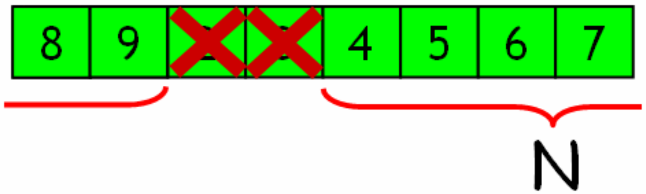
peek(i) → A[(head + i) mod 8]

Copy-Shift:



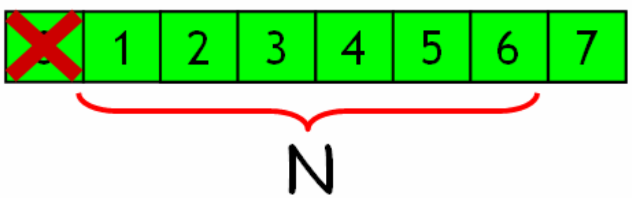
Buffer Management

Circular Buffer:



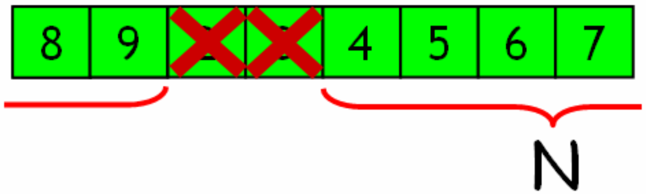
peek(i) → A[(head + i) mod 8]

Copy-Shift:



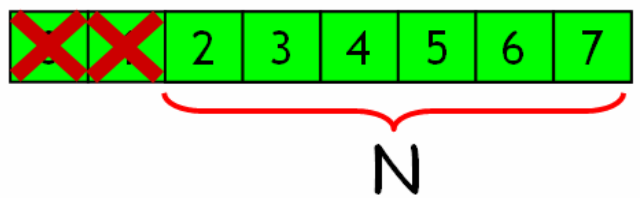
Buffer Management

Circular Buffer:



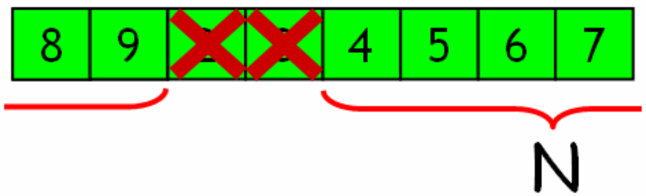
peek(i) → A[(head + i) mod 8]

Copy-Shift:



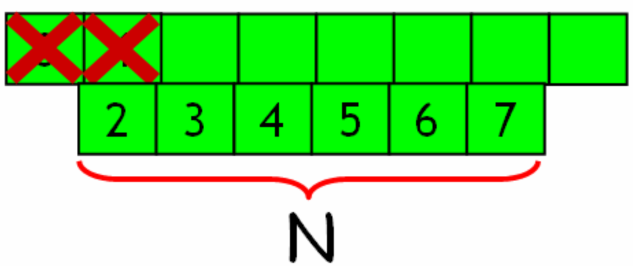
Buffer Management

Circular Buffer:



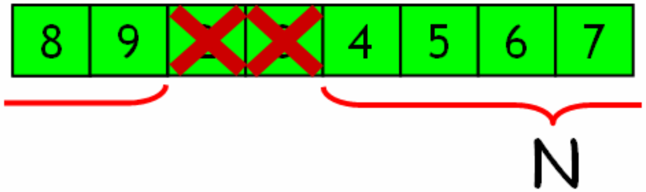
peek(i) → A[(head + i) mod 8]

Copy-Shift:



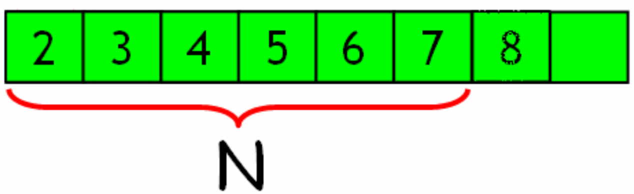
Buffer Management

Circular Buffer:



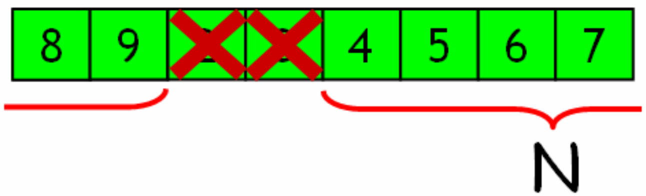
peek(i) → A[(head + i) mod 8]

Copy-Shift:



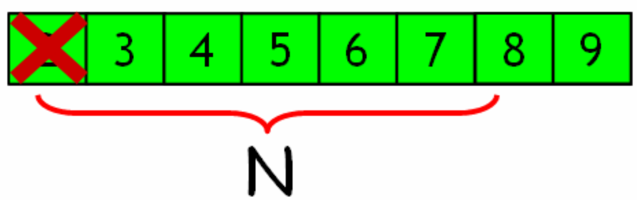
Buffer Management

Circular Buffer:



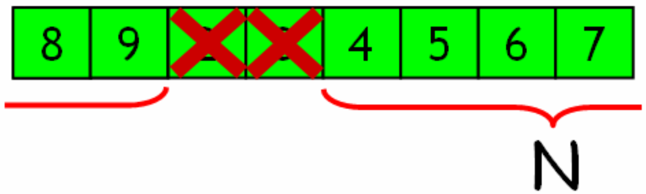
peek(i) → A[(head + i) mod 8]

Copy-Shift:



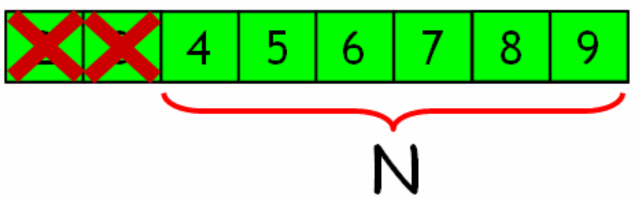
Buffer Management

Circular Buffer:



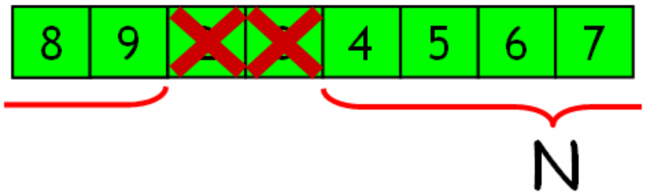
peek(i) → A[(head + i) mod 8]

Copy-Shift:



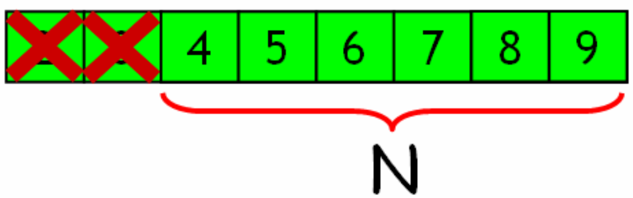
Buffer Management

Circular Buffer:

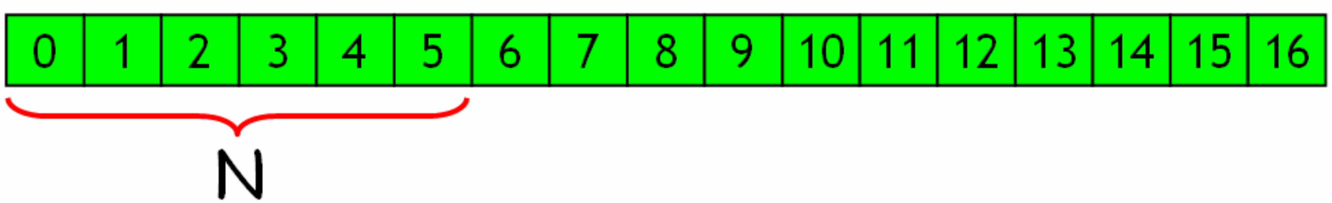


peek(i) → A[(head + i) mod 8]

Copy-Shift:

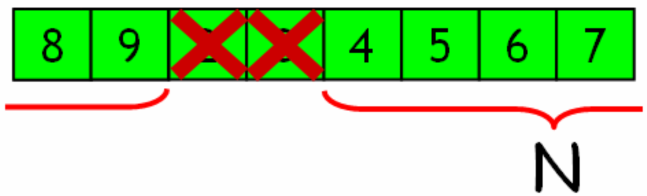


Copy-Shift + Scaling:



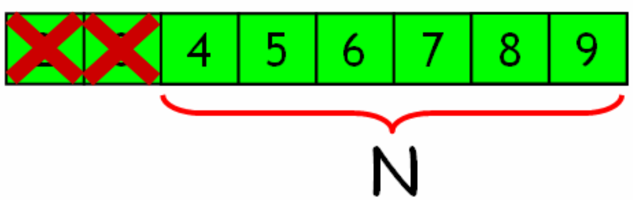
Buffer Management

Circular Buffer:

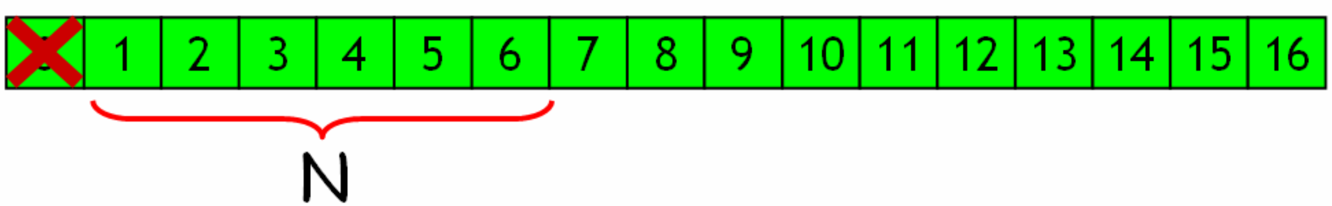


peek(i) → A[(head + i) mod 8]

Copy-Shift:

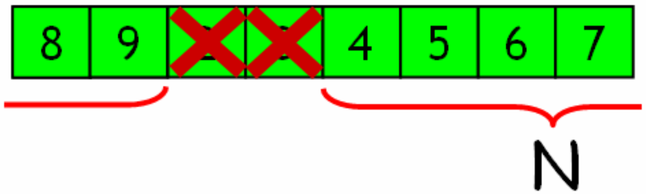


Copy-Shift + Scaling:



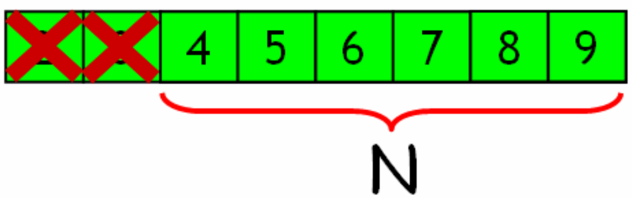
Buffer Management

Circular Buffer:

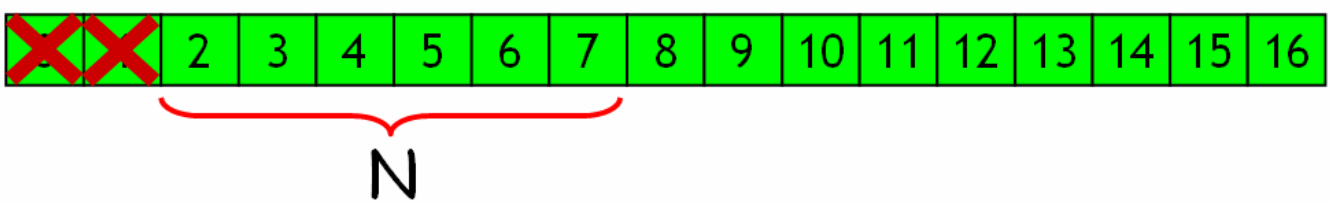


peek(i) → A[(head + i) mod 8]

Copy-Shift:

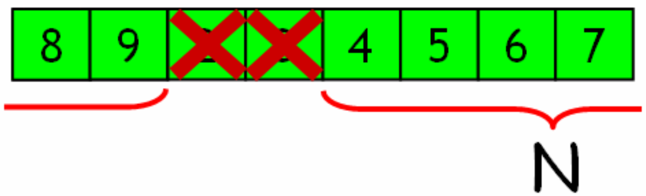


Copy-Shift + Scaling:



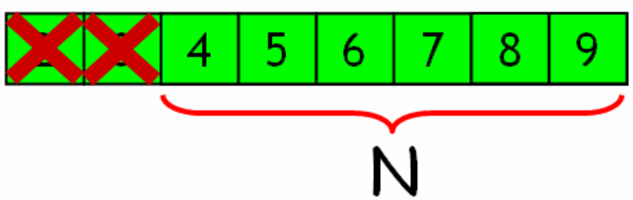
Buffer Management

Circular Buffer:



peek(i) → A[(head + i) mod 8]

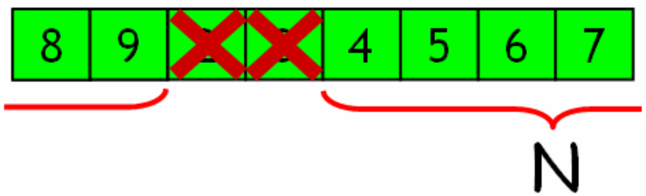
Copy-Shift:



Copy-Shift + Scaling:

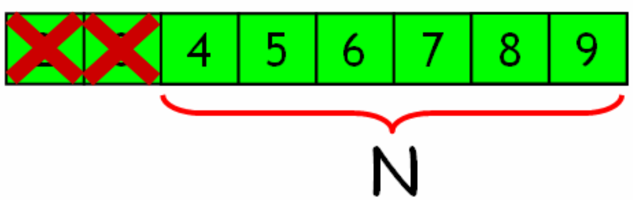
Buffer Management

Circular Buffer:

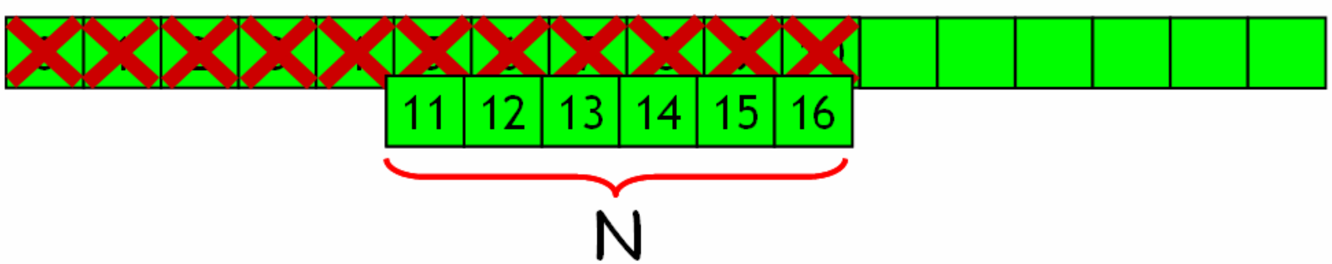


peek(i) → A[(head + i) mod 8]

Copy-Shift:

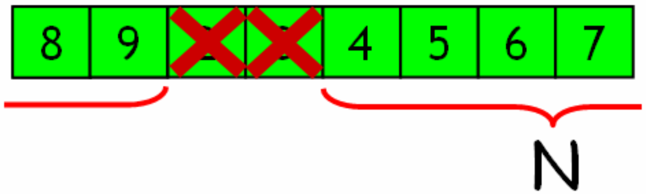


Copy-Shift + Scaling:



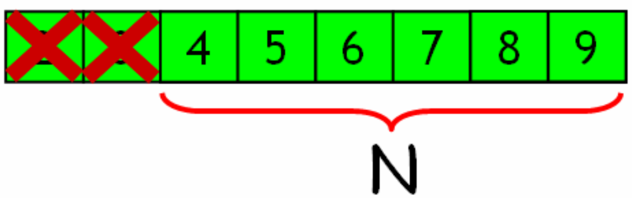
Buffer Management

Circular Buffer:

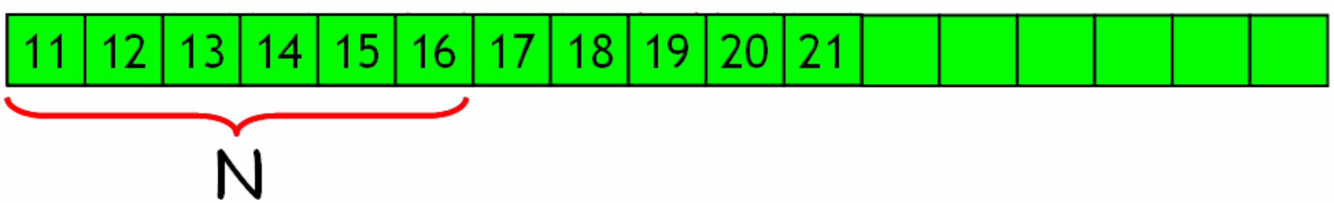


peek(i) → A[(head + i) mod 8]

Copy-Shift:

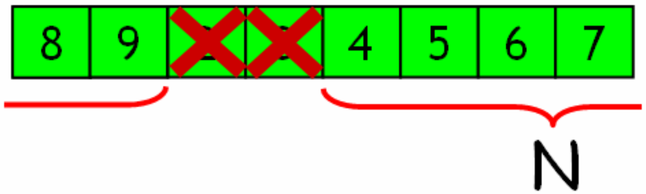


Copy-Shift + Scaling:



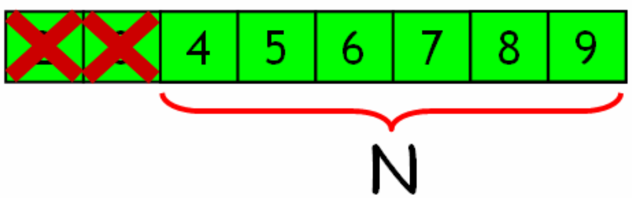
Buffer Management

Circular Buffer:

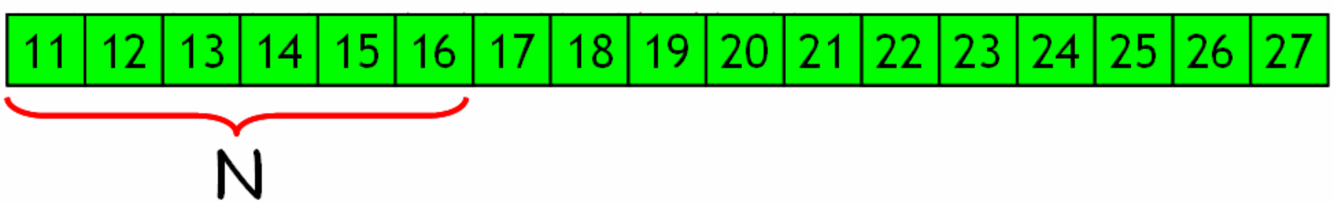


peek(i) → A[(head + i) mod 8]

Copy-Shift:

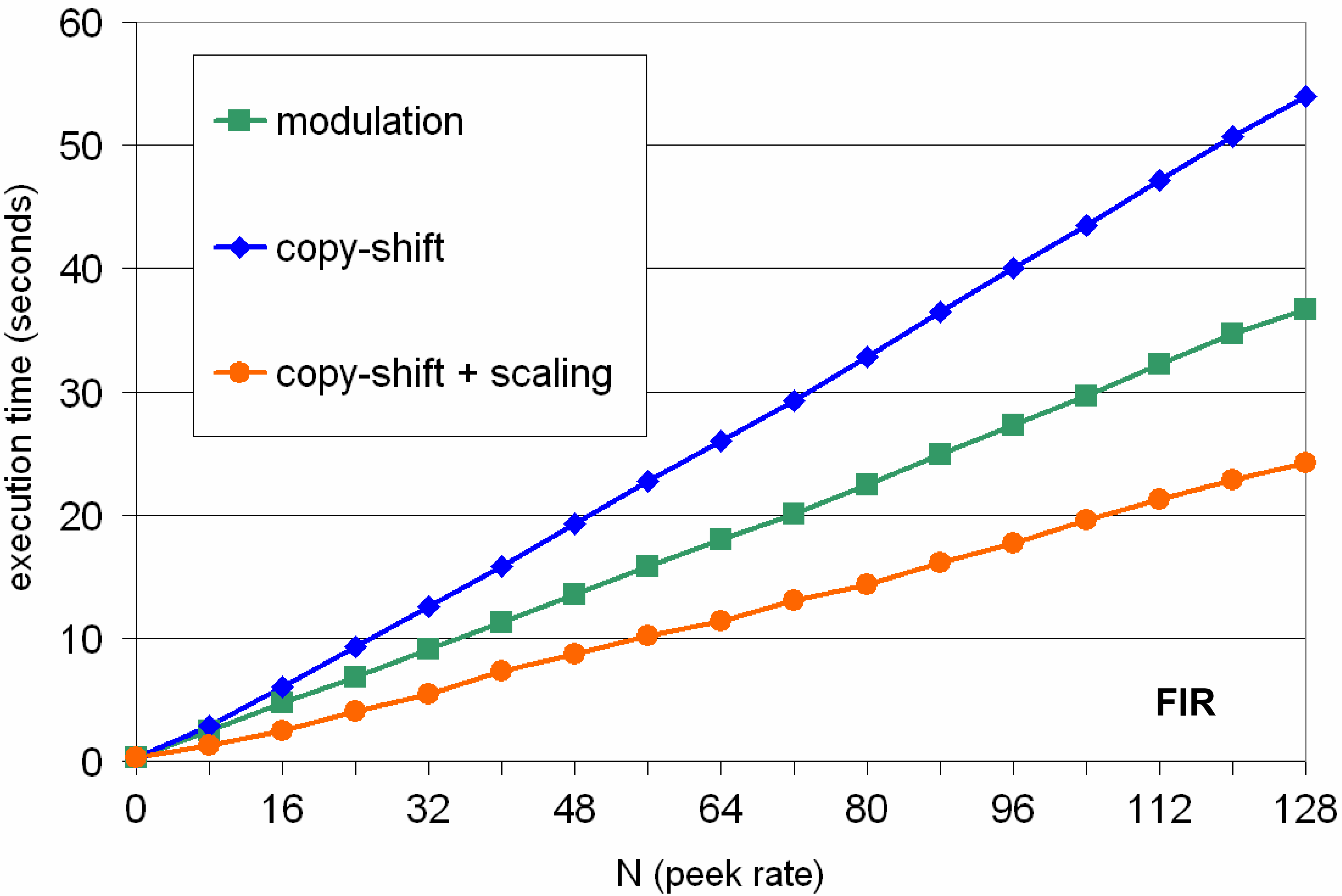


Copy-Shift + Scaling:



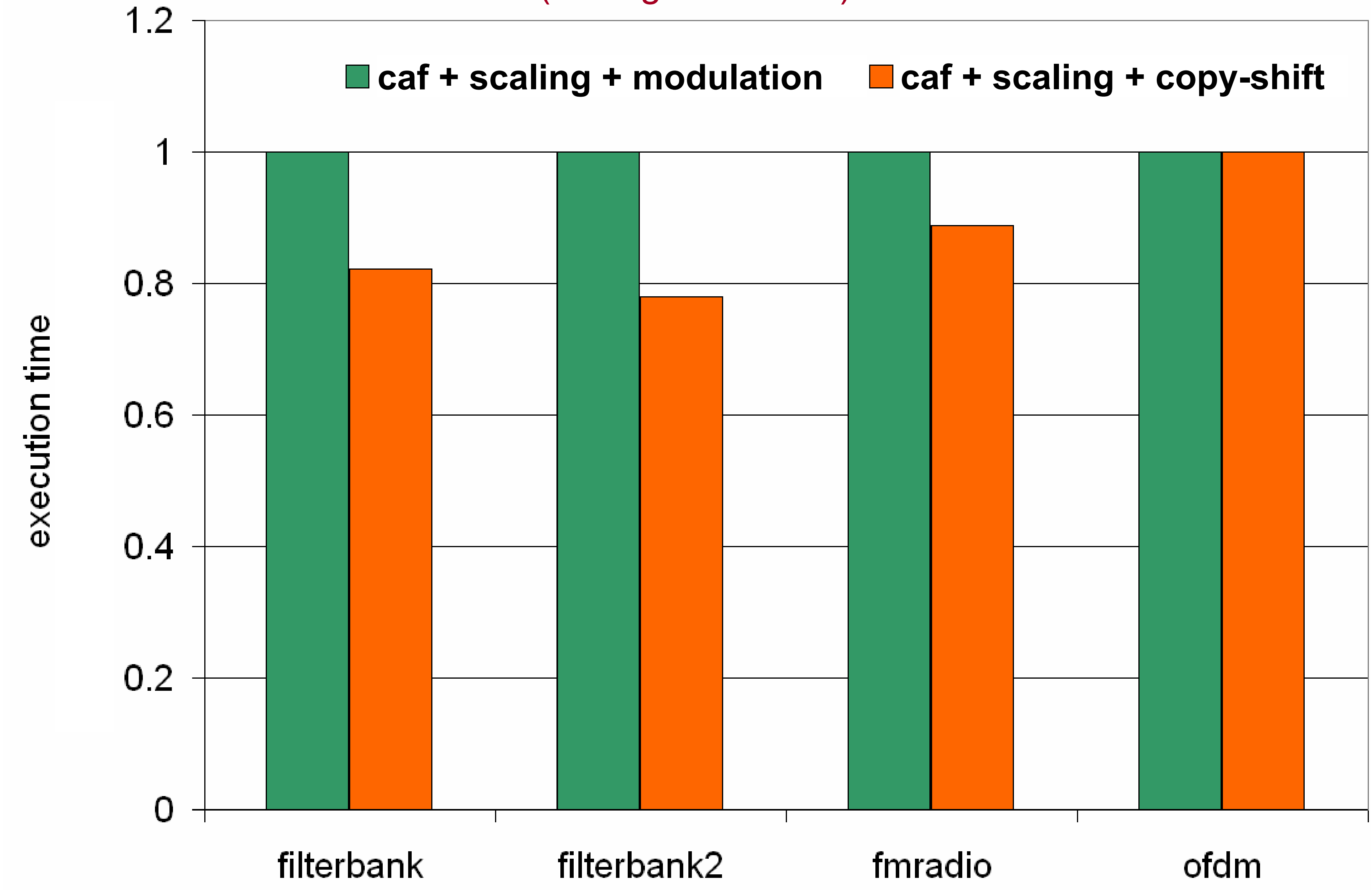
Performance vs. Peek Rate

(StrongARM 1110)

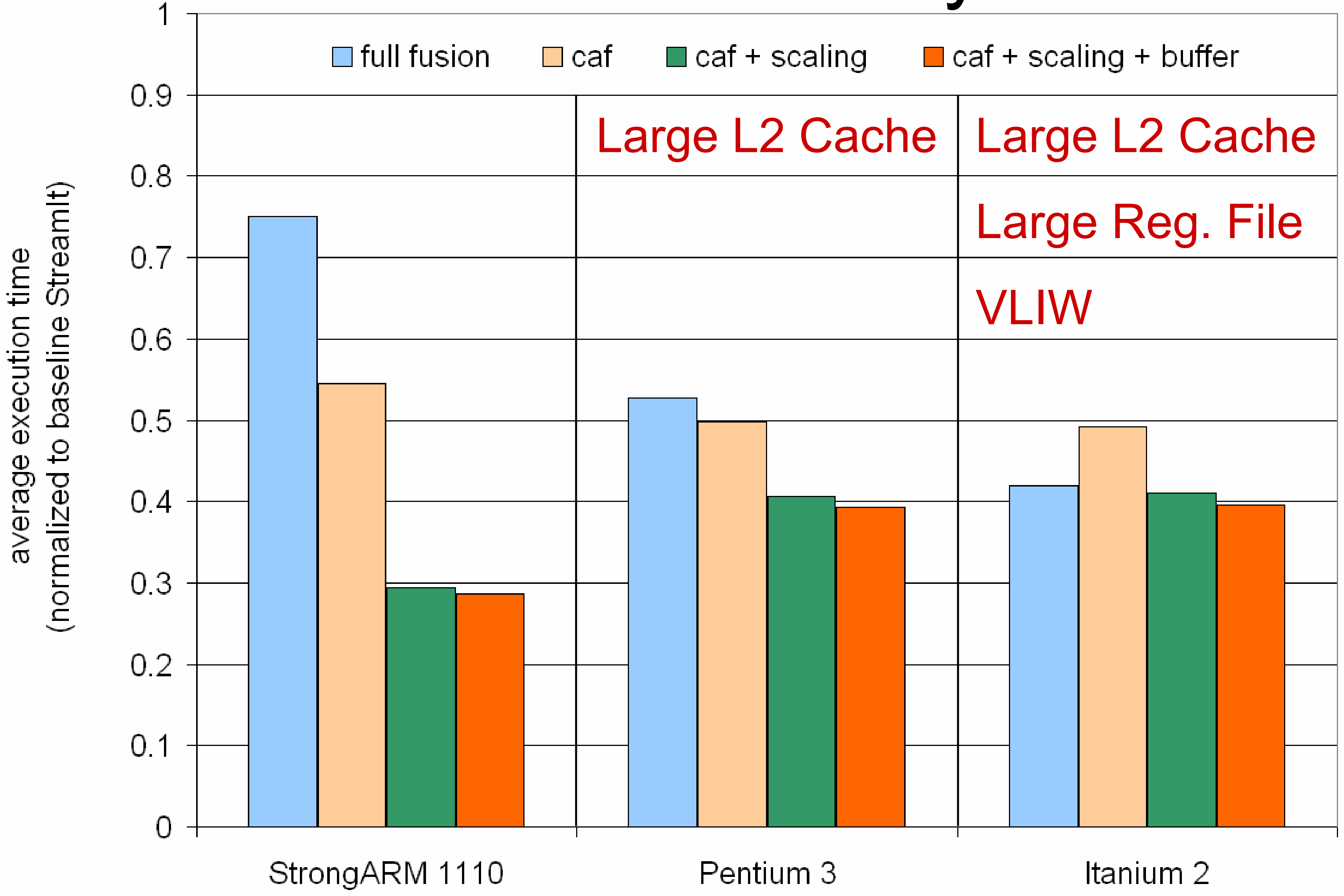


Evaluation for Benchmarks

(StrongARM 1110)



Results Summary



Outline

- StreamIt
- Cache Aware Fusion
- Cache Aware Scaling
- Buffer Management
- Related Work and Conclusion

Related work

- Minimizing buffer requirements
  - S.S. Bhattacharyya, P. Murthy, and E. Lee
    - Software Synthesis from Dataflow Graphs (1996)
    - AGPAN and RPMC: Complimentary Heuristics for Translating DSP Block Diagrams into Efficient Software Implementations (1997)
    - Synthesis of Embedded software from Synchronous Dataflow Specifications (1999)
  - P.K.Murthy, S.S. Bhattacharyya
    - A Buffer Merging Technique for Reducing Memory Requirements of Synchronous Dataflow Specifications (1999)
    - Buffer Merging – A Powerful Technique for Reducing Memory Requirements of Synchronous Dataflow Specifications (2000)
  - R. Govindarajan, G. Gao, and P. Desai
    - Minimizing Memory Requirements in Rate-Optimal Schedules (1994)
- Fusion
  - T. A. Proebsting and S. A. Watterson, Filter Fusion (1996)
- Cache optimizations
  - S. Kohli, Cache Aware Scheduling of Synchronous Dataflow Programs (2004)

Conclusions

- Streaming paradigm exposes parallelism and allows massive reordering to improve locality
- Must consider both data and instruction locality
  - Cache Aware Fusion enables local optimizations by judiciously increasing the instruction working set
  - Cache Aware Scaling improves instruction locality by judiciously increasing the buffer requirements
- **Simple optimizations have high impact**
  - Cache optimizations yield significant speedup over both baseline and full fusion on an embedded platform