Software Synthesis Trade-offs in Dataflow Representations of DSP Applications

Shuvra S. Bhattacharyya
Department of Electrical and Computer Engineering, and
Institute for Advanced Computer Studies
University of Maryland, College Park 20742
ssb@eng.umd.edu, http://www.ece.umd.edu/~ssb

DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

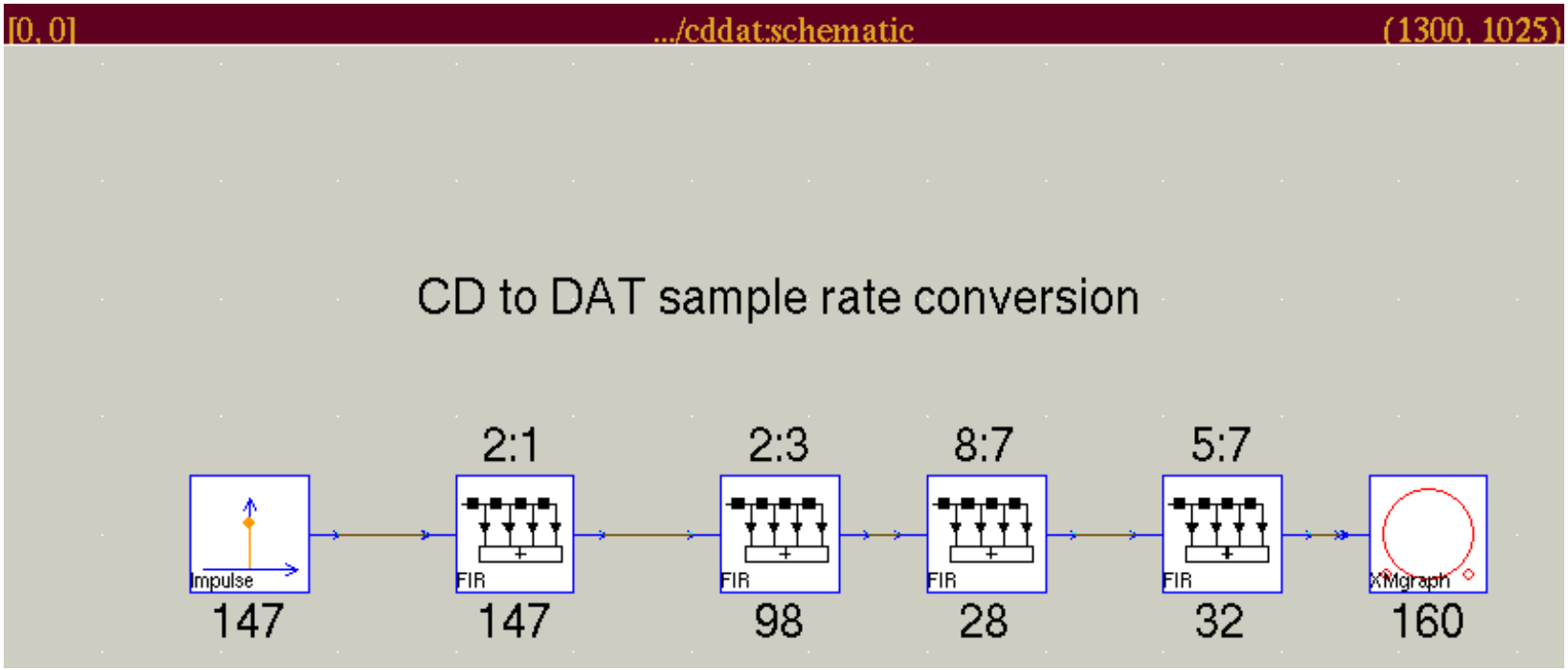
DEPARTMENT OF
ELECTRICAL &
COMPUTER ENGINEERING

DSP system implementation

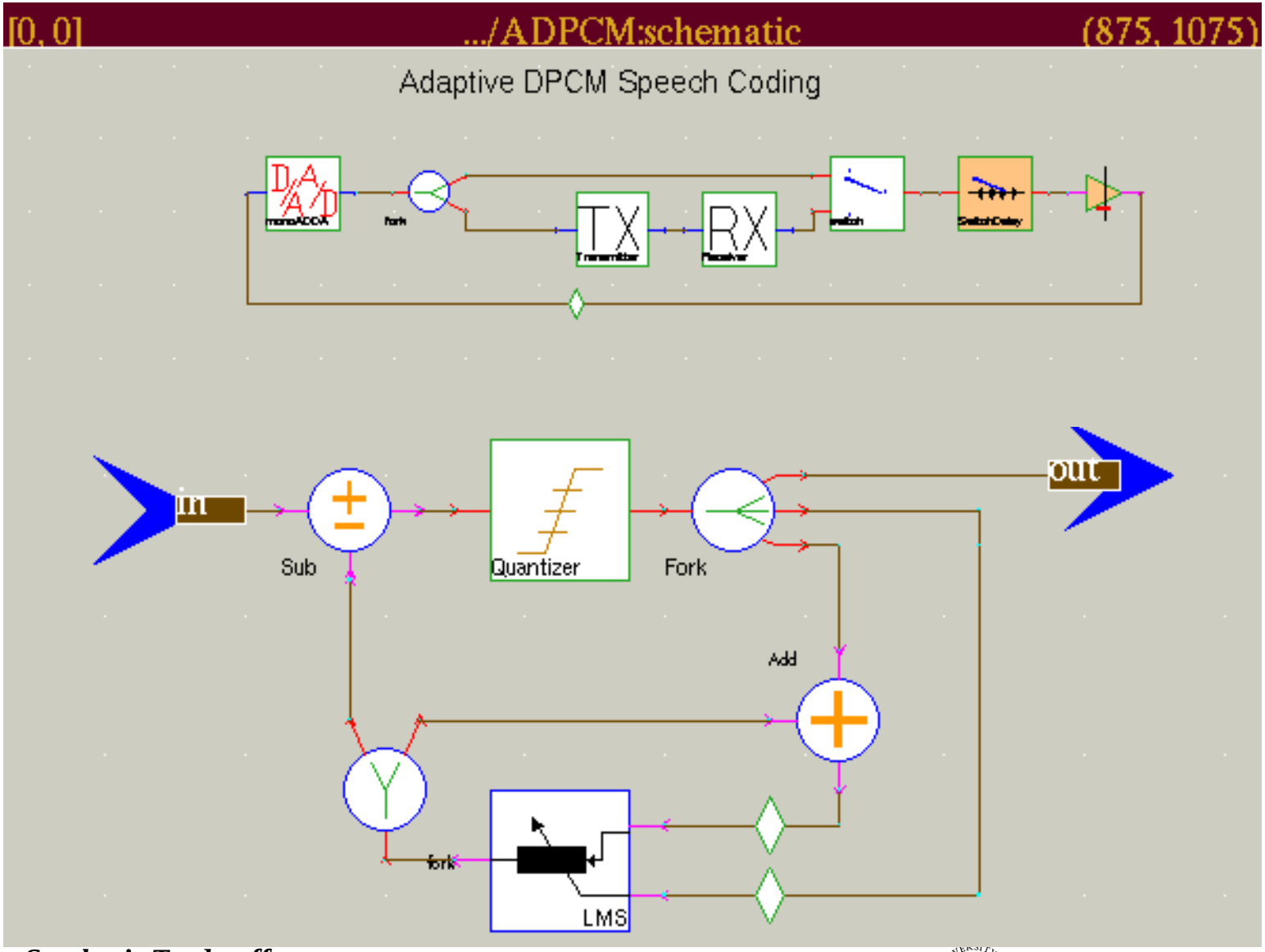
- Computational characteristics
  - Infinitely iterated
  - Possibly significant multirate components
  - Largely deterministic control flow
  - Deadline oriented
- Implementation objectives
  - Speed (latency and throughput)
  - Predictable satisfaction of real-time constraints
  - Memory requirements (code and data)
  - Memory partitioning (code/data, on-chip/off-chip)
  - Power consumption (Peak power, energy/function, …)
  - Cost
  - Time to market

Block diagram environments for DSP

application designed by members of the Ptolemy project, UC Berkeley



Speech coding example



application designed by members of the Ptolemy project, UC Berkeley

Block specification

```
        codeblock(std) {
        ; initialize address registers for coef and
 delayLineove     #$addr(coef)+$val(coefLen)-1,r3
; insert here
        move    $ref(delayLineStart),r5
; delayLine
        move    #$val(stepSize),x1
        move    $ref(error),x0
        mpyr    x0,x1,a
        move    a,x0
        move    x:(r3),b        y:(r5)+,y0
        }

        codeblock(loop) {
        do      #$val(loopVal),$label(endloop)
        macr    x0,y0,b
        move    b,x:(r3)-
        move    x:(r3),b        y:(r5)+,y0
$label(endloop)
        }

        codeblock(noloop) {
        macr    x0,y0,b
        move    b,x:(r3)-
        move    x:(r3),b        y:(r5)+,y0
        }
```

Advantages of block-diagram-based DSP design

- Exposes coarse-grain parallelism
- Exposes additional high-level structure that is helpful for analysis, verification, and optimization
- Encourages desirable software engineering practices
  - Modularity
  - Code re-use
- Intuitive to DSP algorithm designers
  - Block diagrams correspond to signal flow graphs
  - Easy to learn
  - Naturally supports a visual programming interface
- Avoids overspecification of the system

DSP actors

- *Fine-grain dataflow:* nodes are simple "atomic" functions, < 5 processor instructions
- *Coarse-grain dataflow:* nodes are complex functions of 10-100 instructions or more
- For application modeling, we are interested in *mixed-grain dataflow*
  - node functions can have arbitrary complexity
  - examples: *adders, FFT units, FIR filters, adaptive filters,...*
- Internal functionality of a dataflow node
  - Can be specified in an arbitrary programming language (usually C or assembly lang. for DSP software), or
  - Can be specified as a nested dataflow graph (*hierarchical dataflow*)

Evolution of dataflow models for DSP

- Synchronous dataflow, Lee/Messerschmitt, 1987 [16]
  - SPW (Cadence), ADS (Hewlett Packard), …
- Well behaved stream flow graphs, Gao/Govindarajan/Panangaden, 1992 [17]
- The token flow model, Buck 1993 [18]
- Multidimensional synchronous dataflow, Lee, 1992 [15]
- Scalable synchronous dataflow, Ritz/Pankert/Meyr, 1993 [7]
  - COSSAP (Synopsys)
- Cyclo-static and cyclo-dynamic dataflow, Bilsen et al. [13]
  - Virtuoso Synchro (Eonic Systems)
- Bounded dynamic dataflow, Pankert/Mauss/Ritz/Meyr, 1994 [14]
- The processing graph method, Kaplan/Stevens, 1995 [19]
  - U. S. Naval Research Lab
- Parameterized dataflow, Bhattacharya/Bhattacharyya, 2001 [22]

Dataflow modeling issues

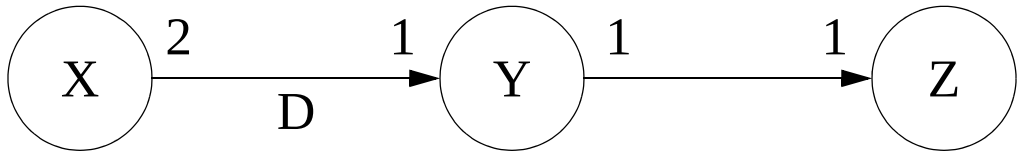
- A **consistent** dataflow specification avoids (for all possible input sets):
  - **U**: Unbounded buffer accumulation
  - **D**: Deadlock (buffer underflow)
- A **partially-consistent** specification avoids *U* and *D* for some inputs, and suffers from *U* or *D* for other inputs.
- A **decidable** dataflow MoC is one in which consistency can always be verified in finite time.
- A **binary-consistency** dataflow model is one that contains no partially-consistent specifications.
- A **statically-schedulable** model is one for which static, periodic schedules can be constructed for all consistent specifications.

Comparison of dataflow models

	Decidable	Binary consistency	Statically-schedulable
synchronous dataflow	YES	YES	YES
cyclo-static dataflow	YES	YES	YES
well-behaved dataflow	YES	YES	NO
token-flow model	NO	NO	NO
bdd.dynamic dataflow	NO	NO	NO
multidimensional SDF	YES	YES	YES
scalable SDF	YES	YES	YES

Synchronous dataflow

- The number of tokens produced and consumed by each actor is fixed
- Edges may have delays, implemented as initial tokens
- Periodic schedules
- Unique repetitions vector "q"

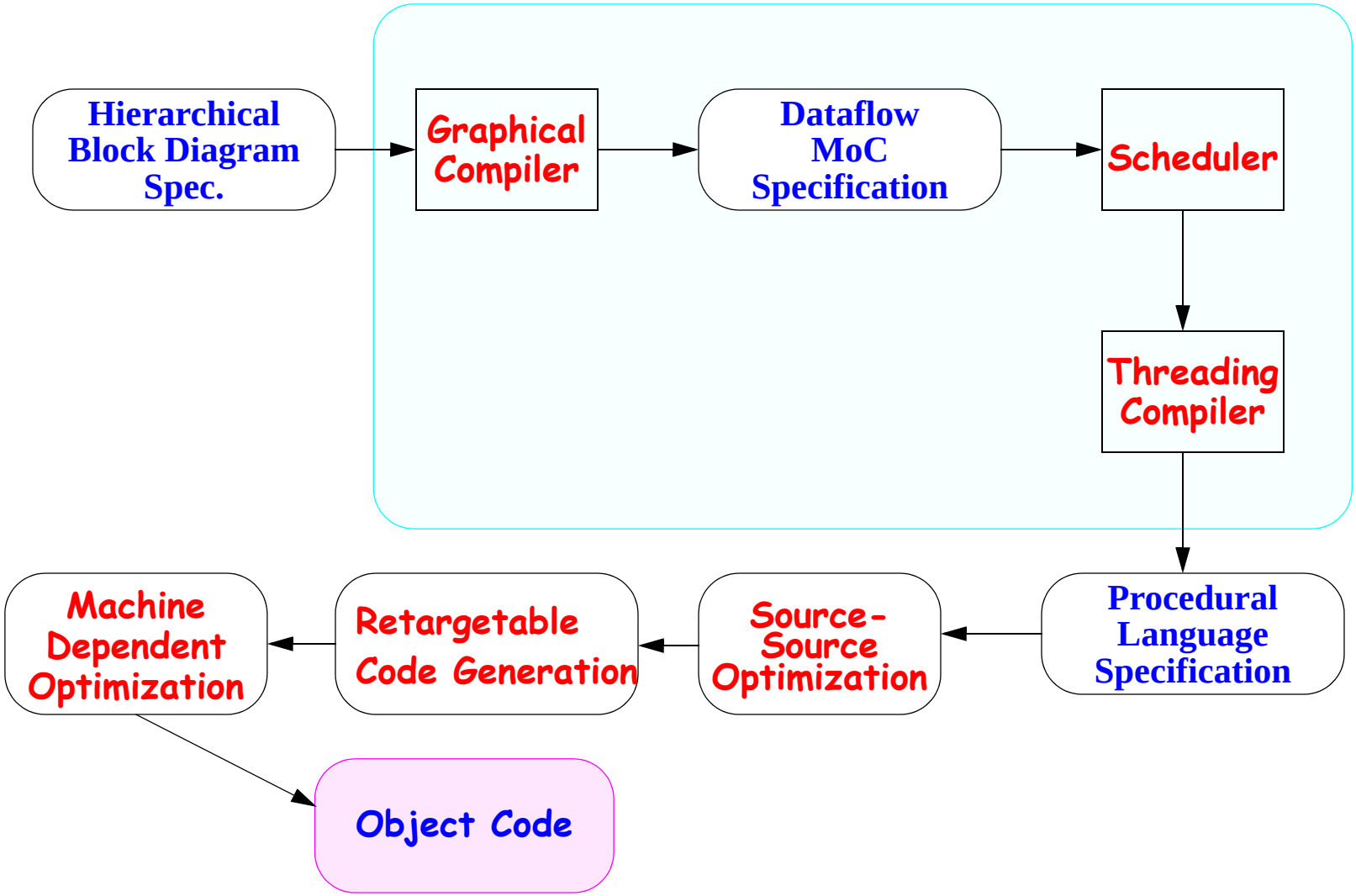


static, periodic schedules

$q_X = 1, \quad q_Y = 2, \quad q_Z = 2$

YXZYZ, XYZYZ, X(2 YZ)

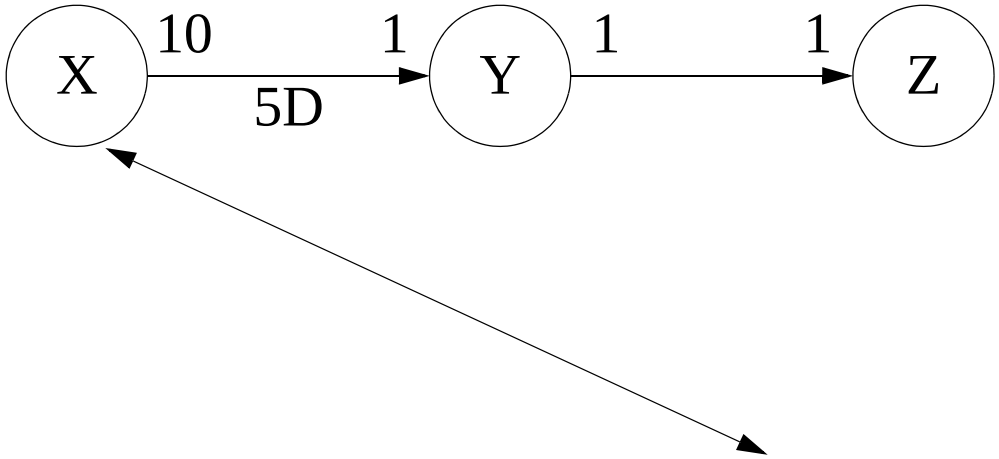
Synthesis flow for DSP Software



Related Work

- Loop transformations
  - Assume loop nests are given a-priori
  - Focus on individual nests in isolation
  - Complementary to SDF loop rolling
- Memory management and register allocation
  - Assume given loop nest / instruction schedule, or
  - Assume homogeneous data sizes, or
  - Assume arbitrary control flow
- Block processing of dataflow graphs
  - Ritz et al.: multirate block processing
  - Lalgudi et al., Zivojnovic et al.: retiming techniques
  - Hong et al.: minimizing context switching for single-rate graphs

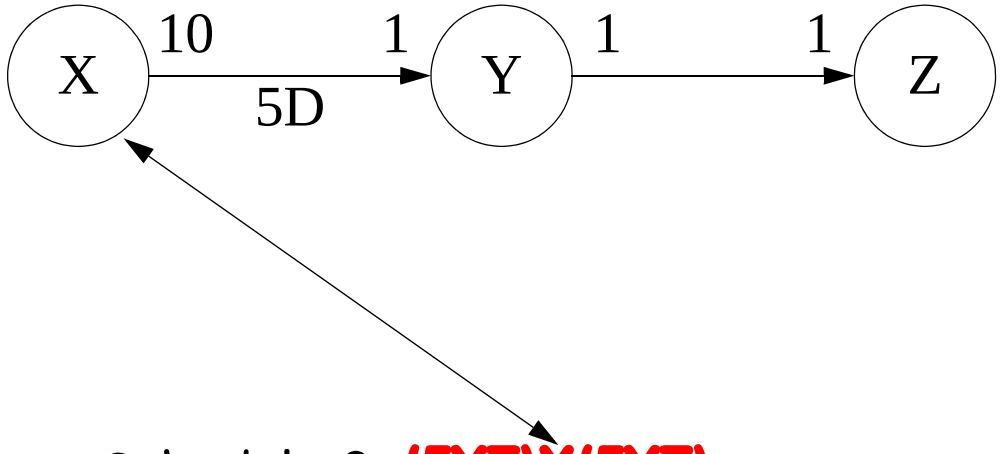
Scheduling trade-offs 1



Schedule 1: YZYZYZYZYZXYZYZYZYZYZ

Threading Mode	Code Size	Buff. Cost	Loop Overhead	Subroutine Overhead	Actor Activations
Inlined	$\kappa(X) + 10\kappa(Y) + 10\kappa(Z)$	11	0	0	21
Subr.	$\kappa(X) + \kappa(Y) + \kappa(Z) + 21\sigma_c$	11	0	$21\sigma_t$	21

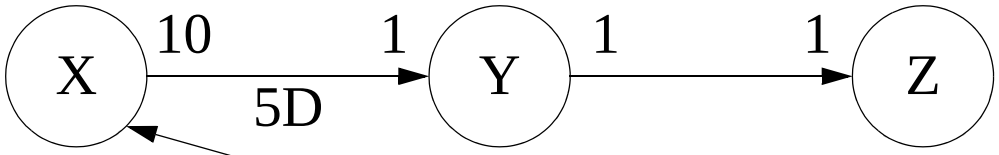
Scheduling trade-offs 2



Schedule 2: (5YZ)X(5YZ)

Threading Mode	Code Size	Buff. Cost	Loop Overhead	Subroutine Overhead	Actor Activations
Inlined	$\kappa(X) + 2\kappa(Y) + 2\kappa(Z) + 2L_c$	11	$2L_s + 10L_i$	0	21
Subr.	$\kappa(X) + \kappa(Y) + \kappa(Z) + 2L_c + 5\sigma_c$	11	$2L_s + 10L_i$	$21\sigma_t$	21

Scheduling trade-offs 3



Schedule 3: X(10Y)(10Z)

Schedule 4: X(10YZ)

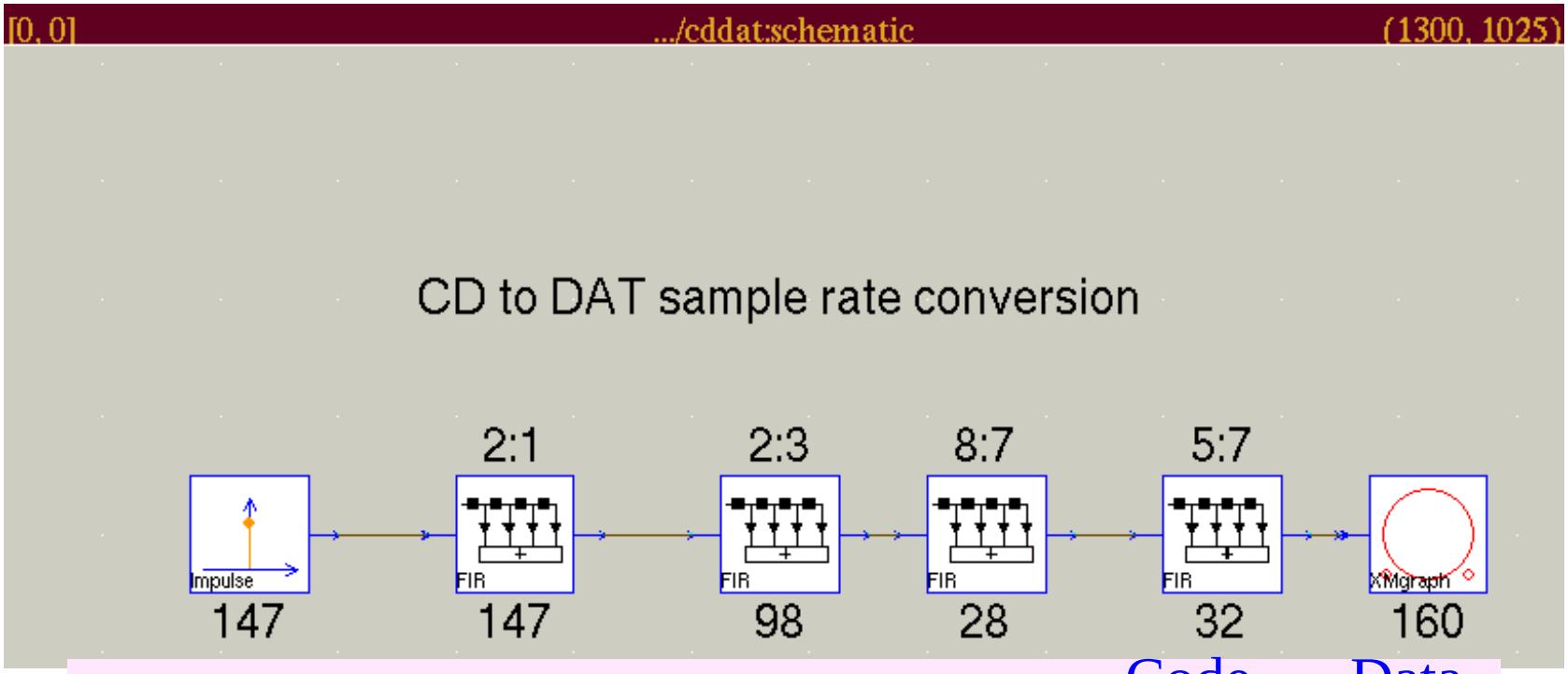
Threading Mode	Code Size	Buff. Cost	Loop Overhead	Subroutine Overhead	Actor Activations
Inlined	$\kappa(X)+\kappa(Y)$ $+\kappa(Z)+2L_c$	25	$2L_s+20L_i$	0	3
Inlined	$\kappa(X)+\kappa(Y)$ $+\kappa(Z)+L_c$	16	L_s+10L_i	0	21

vs. 11 for previous schedules

Scheduling example — summary

Schedule	Thr. Mode	Code Size	Buff. Cost	Loop Ovhd.	Subr. Ovhd	Actor Activs.
$YZYZYZYZYZX$ $YZYZYZYZYZ$	I	$\kappa(X) + 10\kappa(Y) + 10\kappa(Z)$	11	0	0	21
$YZYZYZYZYZX$ $YZYZYZYZYZ$	S	$\kappa(X) + \kappa(Y) + \kappa(Z)$ $+21\sigma_c$	11	0	$21\sigma_t$	21
$(5YZ)X(5YZ)$	I	$\kappa(X) + 2\kappa(Y) + 2\kappa(Z)$ $+2L_c$	11	$2L_s + 10L_i$	0	21
$(5YZ)X(5YZ)$	S	$\kappa(X) + \kappa(Y) + \kappa(Z)$ $+2L_c + 5\sigma_c$	11	$2L_s + 10L_i$	$21\sigma_t$	21
$X(10Y)(10Z)$	I	$\kappa(X) + \kappa(Y) + \kappa(Z)$ $+2L_c$	25	$2L_s + 20L_i$	0	3
$X(10YZ)$	I	$\kappa(X) + \kappa(Y) + \kappa(Z) + L_c$	16	$L_s + 10L_i$	0	21

Code/data design space example



	Code	Data
Minimum buffer schedule, no looping	**13735**	**32**
Minimum buffer schedule, with looping	**9400**	**32**
Worst minimum code size schedule	**170**	**1021**
Best minimum code size schedule	**170**	**264**

Buffer minimization

- Construction of minimum buffer, static, periodic schedules is intractable [12]
  - Cycles
  - Delays
  - Production/consumption mismatches on edges (**iteration**)
- Graph-theoretic upper bound analysis [8]
  - Exact for some useful subclasses of graphs
- Optimal scheduling technique for restricted graphs [9]
  - No directed cycles
  - No interacting undirected cycles
- "Demand-driven" scheduling approach [11]
  - Good general heuristic
- Buffer minimization for maximum-throughput (**rate-optimal**) schedules [10]

CD to DAT example

minimum buffer schedule:

A B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E
A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C
D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B A B C A B C A
B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F E B C A
F F F F F B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C
D E A F F F F F B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C A
B C D E A F F F F F B A B C A B C A B A B C D E A F F F F F E B C A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F F F F F B A
B C A B C A B A B C A B C D E A F F F F F B A B C A B C A B A B C D E A F F F
F F B C A B A B C A B C A B A B C D E A F F F F F B C A B A B C A B C D E A F
F F F F B A B C A B C A B A B C A B C D E A F F F F F E B A F F F F F B C A B C
A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F F F F F B C A
B A B C A B C D E A F F F F F B A B C A B C A B A B C A B C D E A F F F F F B
A B C A B C A B A B C D E A F F F F F B C A B A B C A B C A B A B C D E A F
F F F F B C A B A B C A B C D E F F F F F E F F F F F

Optimization Techniques

- Buffer minimization
  - Lifetime analysis
  - Buffer merging
- Joint code and data minimization
- Data partitioning
- Synchronization optimization
  - Redundant synchronization elimination
  - Resynchronization
- Transaction ordering

Summary

- Decidable dataflow models and synchronous dataflow
- Periodic schedules
- Implicit iteration
- Complex design space
- Optimization techniques

Bibliography 1

[1] V. Zivojnovic, S. Ritz, and H. Meyr, "Retiming of DSP programs for optimum vectorization," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1994.

[2] K. N. Lalgundi, M. C. Papaefthymiou, and M. Potkonjak, "Optimizing Systems for Effective Block-Processing: The k-Delay Problem," in *Proceedings of the Design Automation Conference*, 1996, pp. 714--719.

[3] S. Ritz, M. Willems, and H. Meyr, "Scheduling for Optimum Data Memory Compaction in Block Diagram Oriented Software Synthesis," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1995.

[4] P. K. Murthy and S. S. Bhattacharyya, "Shared Memory Implementations of Synchronous Dataflow Specifications using Lifetime Analysis Techniques," Institute for Advanced Computer Studies, University of Maryland at College Park UMIACS-TR-99-32, June 1999.

[5] S. S. Bhattacharyya and P. K. Murthy, "The CBP parameter — a useful annotation to aid SDF compilers," Institute for Advanced Computer Studies, University of Maryland at College Park UMIACS-TR-99-56, September 1999.

Bibliography 2

[6] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, "Optimal Parenthesization of Lexical Orderings for DSP Block Diagrams," in *Proceedings of the International Workshop on VLSI Signal Processing*: IEEE press, 1995.

[7] S. Ritz, M. Pankert, and H. Meyr, "Optimum Vectorization of Scalable Synchronous Dataflow Graphs," in *Proceedings of the International Conference on Application Specific Array Processors*, 1993.

[8] M. Ade, R. Lauwereins, and J. A. Peperstraete, "Data Memory Minimisation for Synchronous Data Flow Graphs Emulated on DSP-FPGA Targets," in *Proceedings of the Design Automation Conference*, 1994, pp. 64--69.

[9] M. Cubric and P. Panangaden, "Minimal Memory Schedules for Dataflow Networks," in *CONCUR '93*, 1993.

[10] R. Govindarajan, G. R. Gao, and P. Desai, "Minimizing Memory Requirements in Rate-Optimal Schedules," in *Proceedings of the International Conference on Application Specific Array Processors*, 1994.

[11] E. A. Lee, W. H. Ho, E. Goei, J. Bier, and S. S. Bhattacharyya, "Gabriel: A Design Environment for DSP," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 37, 1989.

Bibliography 3

[12] S. S. Bhattacharyya, P. K. Murthy, and E. A. Lee, *Software Synthesis from Dataflow Graphs*: Kluwer Academic Publishers, 1996.

[13] G. Bilsen, M. Engels, R. Lauwereins, and J. A. Peperstraete, "Cyclo-Static Dataflow," *IEEE Transactions on Signal Processing*, vol. 44, pp. 397--408, 1996.

[14] M. Pankert, O. Mauss, S. Ritz, and H. Meyr, "Dynamic Data Flow and Control Flow in High Level DSP Code Synthesis," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1994.

[15] E. A. Lee, "Representing and Exploiting Data Parallelism Using Multidimensional Dataflow Diagrams," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1993, pp. 453--456.

[16] E. A. Lee and D. G. Messerschmitt, "Static Scheduling of Synchronous Dataflow Programs for Digital Signal Processing," *IEEE Transactions on Computers*, 1987.

[17] G. R. Gao, R. Govindarajan, and P. Panangaden, "Well-Behaved Programs for DSP Computation," in *Proceedings of the International Conference on Acoustics, Speech, and Signal Processing*, 1992.

Bibliography 4

[18] J. T. Buck and E. A. Lee, "The Token Flow Model," in *Advanced Topics in Dataflow Computing and Multithreading*, L. Bic, G. Gao, and J.-L. Gaudiot, Eds.: IEEE Computer Society Press, 1993.

[19] R. S. Stevens, "The Processing Graph Method Tool (PGMT)," in *Proceedings of the International Conference on Application Specific Systems, Architectures, and Processors*, 1997, pp. 263--271.

[20] J. Teich, E. Zitzler, and S. S. Bhattacharyya, "Optimized Software Synthesis for Digital Signal Processing Algorithms — an Evolutionary Approach", in *Proceedings of the IEEE Workshop on Signal Processing Systems*, October, 1998.

[21] S. Sriram and S. S. Bhattacharyya. *Embedded Multiprocessors: Scheduling and Synchronization*. Marcel Dekker, Inc., 2000.

[22] B. Bhattacharya and S. S. Bhattacharyya. Parameterized dataflow modeling for DSP systems. *IEEE Transactions on Signal Processing*, 49(10):2408-2421, October 2001.