Converting Graphical DSP Programs into Memory-Constrained Software Prototypes

Shuvra S. Bhattacharyya, Hitachi America, Ltd.

Praveen K. Murthy, University of California, Berkeley

Edward A. Lee, University of California, Berkeley

International Workshop on Rapid Systems Prototyping,
Chapel Hill, North Carolina, June 7-9, 1995.

Application Specific Software Environments

- Provide <u>syntax</u> that is natural for the application domain
- Incorporate appropriate <u>computational models</u>
  - may be streamlined to enable powerful optimization
- Optimize for appropriate <u>implementation constraints</u>

Embedded DSP systems

- **Computational characteristics**
  - **Infinitely iterated**
  - **Possibly multirate**
  - **Mostly deterministic control flow**
- **Implementation objectives**
  - **Target throughput**
  - **Memory**
  - **Latency**
  - **Power**

Computational Models for DSP Software

- **Synchronous dataflow**
  — Lee/Messerschmitt, 1987
- **Well behaved stream flow graphs**
  — Gao/Govindarajan/Panangaden, 1992
- **The token flow model**
  — Buck/Lee 1992
- **Multidimensional synchronous dataflow**
  — Lee 1992
- **Scalable synchronous dataflow**
  — Ritz/Pankert/Meyr, 1993
- **Cyclo-static dataflow**
  — Bilsen/Engels/Lauwereins/Peperstraete, 1994

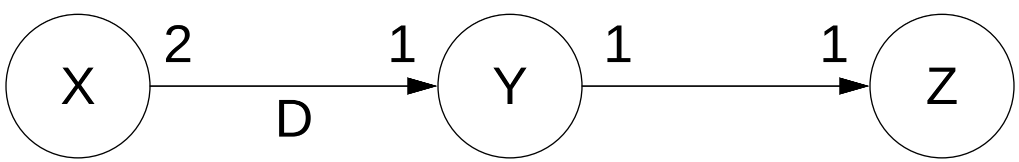
All are closely related to the synchronous dataflow model

Problem overview

- **Minimization of memory requirement (program *and* data) when synthesizing software from a synchronous dataflow program**
  - **Target throughput**
  - **Memory**
  - **Latency**
  - **Power**
- **May be critical to all other objectives**
  - **On-chip vs. off-chip memory**
  - **Limited on-chip memory on programmable DSPs**

Synchronous dataflow

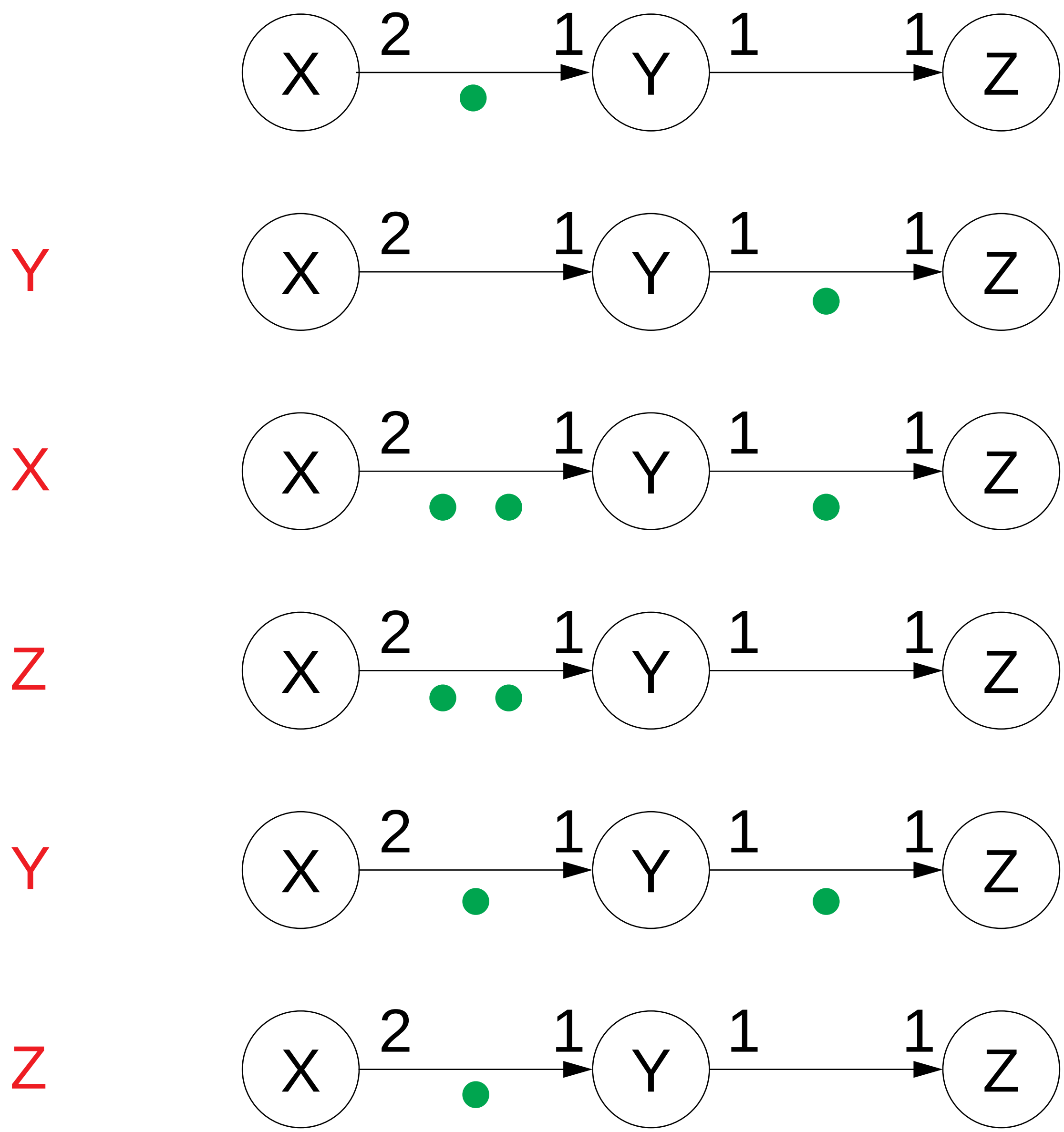
- **The number of tokens produced and consumed by each actor is fixed.**
- **Periodic schedules.**
- **Unique** *repetitions vector* **q.**



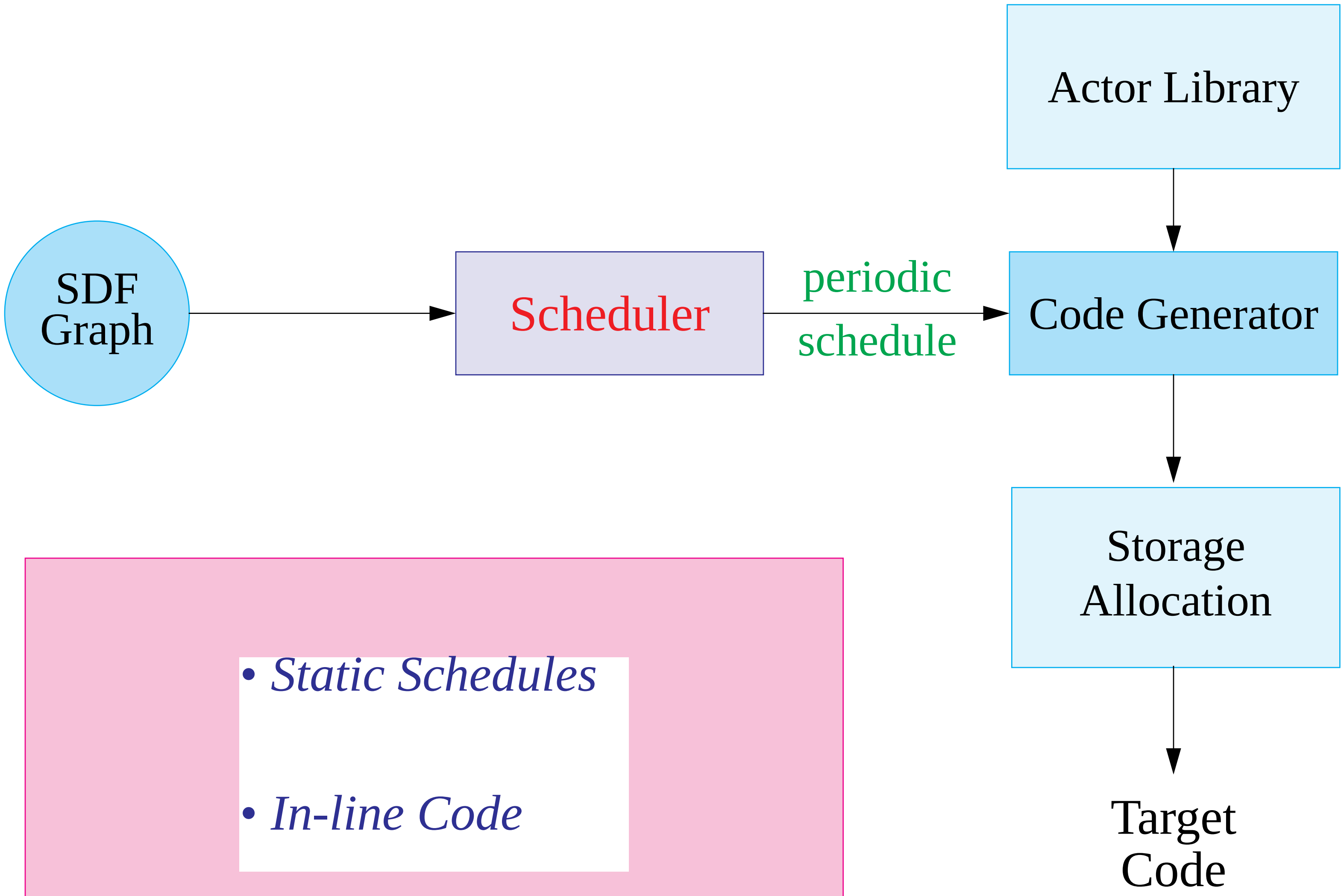
$$q_X = 1, \quad q_Y = 2, \quad q_Z = 2$$

YXZYZ, XYZYZ, X(2 YZ)

Periodic schedule example



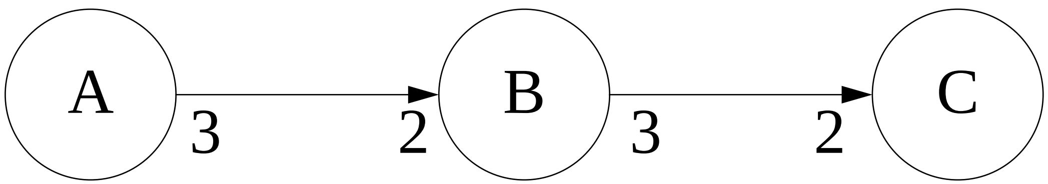
Code synthesis model



- *Static Schedules*
- *In-line Code*

Looped schedules

$$q_A = 4, \quad q_B = 6, \quad q_C = 9$$



Schedules

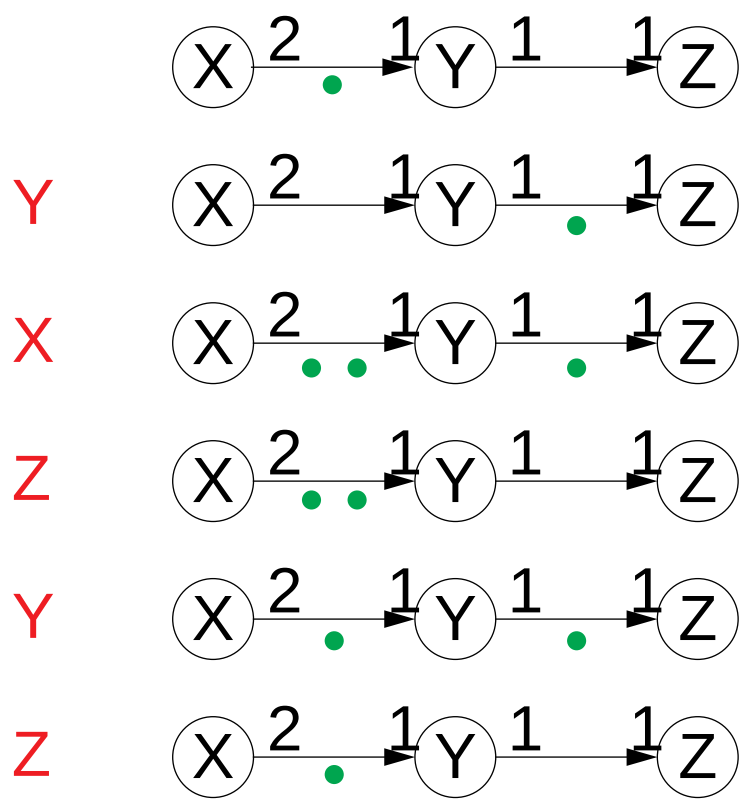
(2 ABC)CBCAB(2 C)A(2 BC)C

Single Appearance Schedules → **(4 A)(6 B)(9 C)**

→ **(4 A)(3 (2B)(3 C))**

Buffering model

- **Buffer on every arc in the graph.**
- **Size of a buffer is given by the maximum number of tokens queued on the arc in the schedule.**
- **Total buffering cost is sum of individual buffer sizes.**



buffering cost $= 2 + 1 = 3$

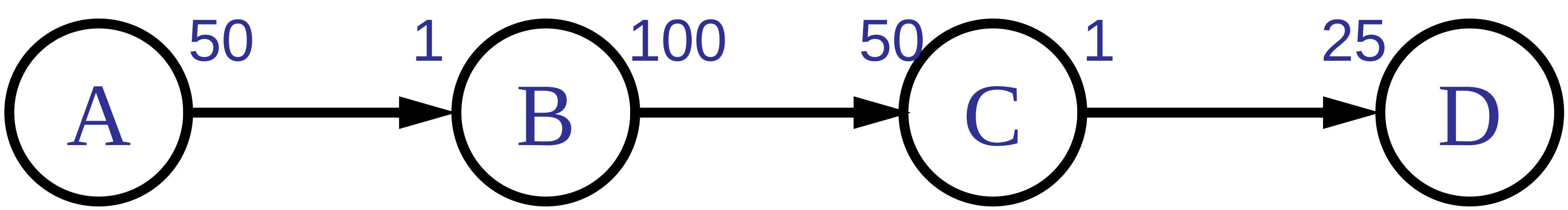
Advantages over alternative buffering models

Alternative #1: *Flat* single appearance schedules with shared buffers.

- Buffering requirement can be very bad for some graphs.
- Does not handle delays well.
- Latency is maximized.

Alternative #2: Use nested schedules with buffer sharing.

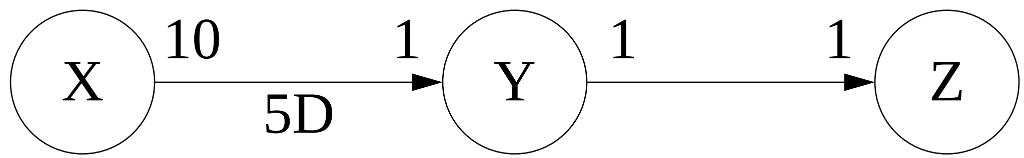
- More awkward to implement.
- Cost function is more complicated.



A (50 B) (100 C) (4 D): Cost = 5000

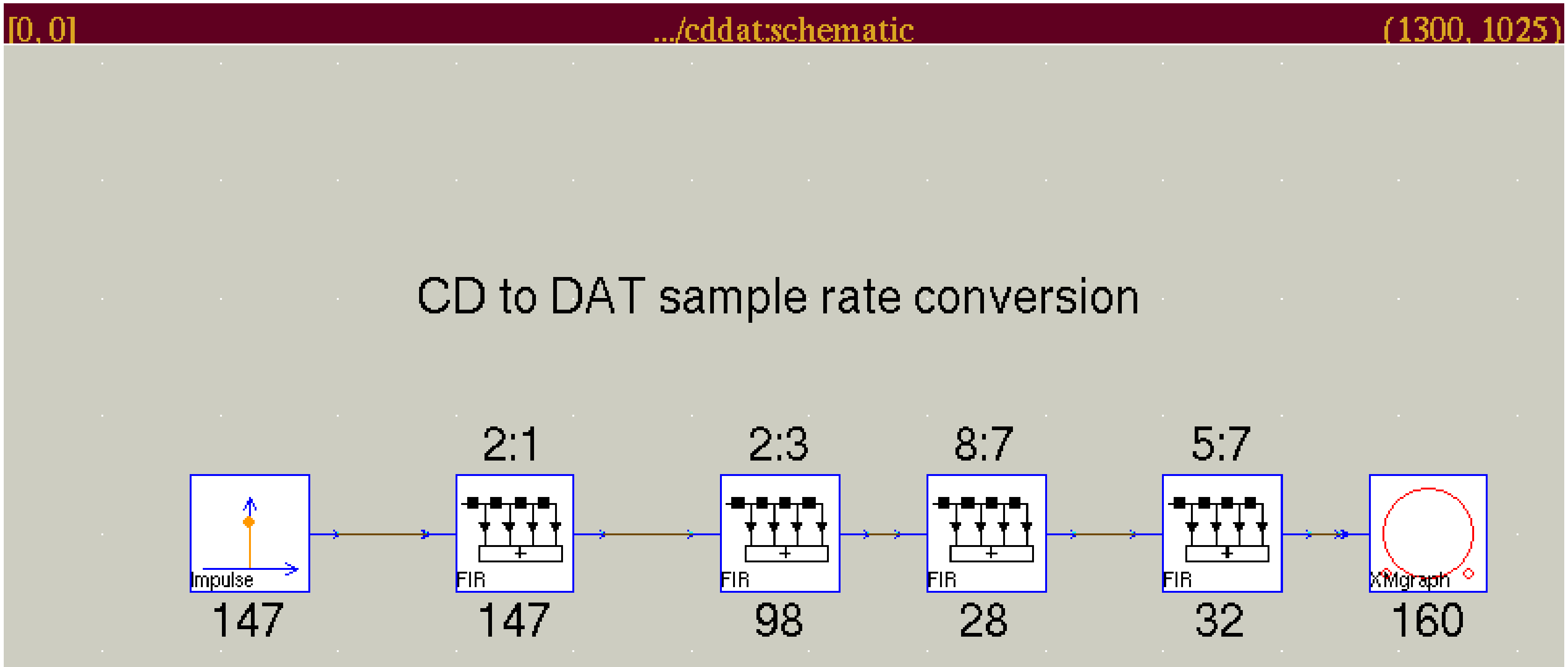
A (2 (25 (B (2 C))) (2 D)): Cost = 200

Code size vs. buffer memory trade-offs



single appearance schedules

(5 YZ) X (5 YZ)	X (10 Y) (10 Z)	X (10 Y Z)
buffering cost: $10 + 1 = 11$	buffering cost: $15 + 10 = 25$	buffering cost: $15 + 1 = 16$
code size: $S(X) + 2S(Y) + 2S(Z)$	code size: $S(X) + S(Y) + S(Z)$	code size: $S(X) + S(Y) + S(Z)$



	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**

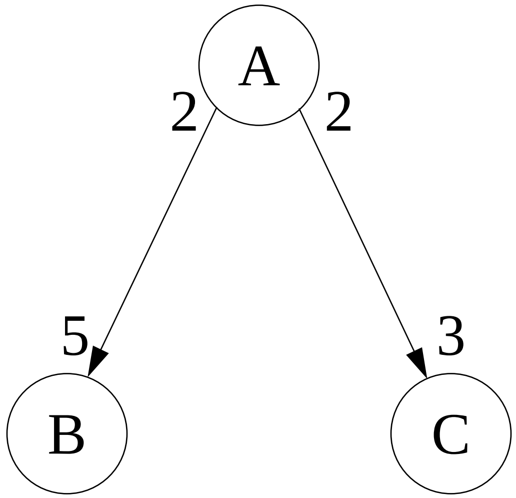
Minimum buffer single appearance schedules

Schedule	Buffering Cost
(9 A)(12 B)(12 C)(8 D)	36 + 12 + 24 = **72**
(3 (3 A)(4 B C))(8 D)	12 + 1 + 24 = **37**
(3 (3 A)(4 B))(4 (3 C)(2 D))	12 + 12 + 6 = **30**

- **Finding buffer-minimal single appearance schedules is NP-complete, even for acyclic, homogenous SDF graphs [Murthy, PhD thesis 1996].**
- **Thus, need to use heuristics.**

Minimum buffer single appearance schedules

single appearance schedule $\Leftrightarrow$ a parenthesization of a lexical ordering



q = (15, 6, 10)

- (3 (5 A) (2 B)) (10 C) buff. cost = 40
- (15 A) (2 (3 B) (5 C)) buff. cost = 60
- (15 A) (6 B) (10 C) buff. cost = 60
- (5 (3 A) (2 C)) (6 B) buff. cost = 36
- (15 A) (2 (5 C) (3 B)) buff. cost = 60

Dynamic programming post optimization (DPPO)

- **Given a lexical ordering, computes an optimal parenthesization.**
- **Time complexity $O(n^3)$.**

$$\ldots((x_i x_{i+1} \ldots x_k)(x_{k+1} \ldots x_{j-1} x_j))\ldots$$

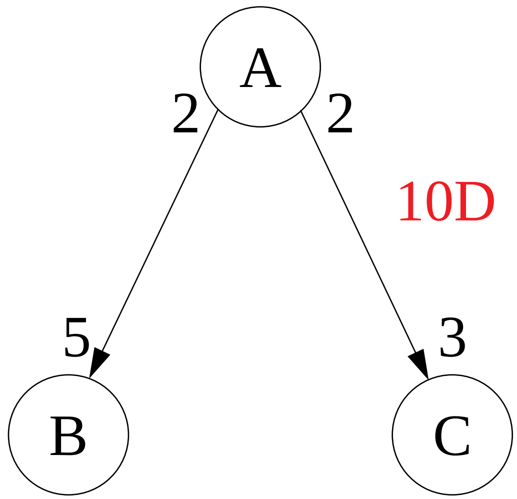
split cost

left subchain cost

right subchain cost

$$b[i, j] = MIN_{i \le k < j}\{b[i, k] + b[k+1, j] + c_{ij}[k]\}$$

Lexical orderings that are not topological sorts



- (5 (3 A) (2 C)) (6 B) buff. cost = 46
- (5 (2 C) (3A)) (6 B) buff. cost = 40

Two heuristics for constructing lexical orderings

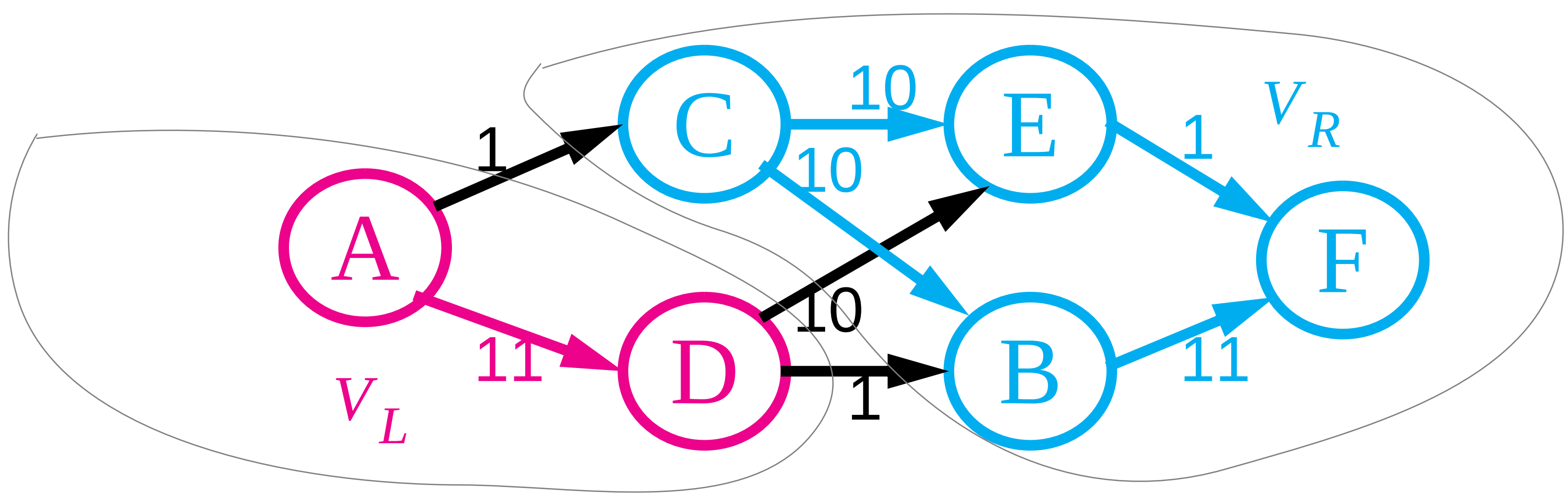
- **Pairwise grouping of adjacent nodes (PGAN)**
  - **Bottom-up algorithm**
  - **Effective for regular topologies**
  - **Optimal for a class of graphs**
- **Recursive partitioning by minimum cuts (RPMC)**
  - **Top-down algorithm**
  - **Effective for irregular topologies**

Complementary:
often, when one does poorly, the other does well

Recursive partitioning by minimum cuts

- **Idea: Find a *cut* of the graph such that**
  - **a) All arcs cross the cut in the forward direction.**
  - **b) The cut results in fairly even-sized sets.**
  - **c) Amount of data crossing the cut is minimized.**

Recursively schedule the nodes on the left side of the cut before nodes on the right side of the cut.



RPMC continued

- **Splitting the graph where the minimum amount of data is transferred is a *greedy* approach and is not optimal in general.**

- **Finding the minimum cut such that all of the conditions *a*, *b*, and *c* are satisfied is itself a difficult problem:**
  - **Methods based on max-flow-min-cut theorem do not work.**
  - **Graph partitioning when the size of the partition has to be bounded is NP-complete.**

- **Therefore, a heuristic solution is needed.**

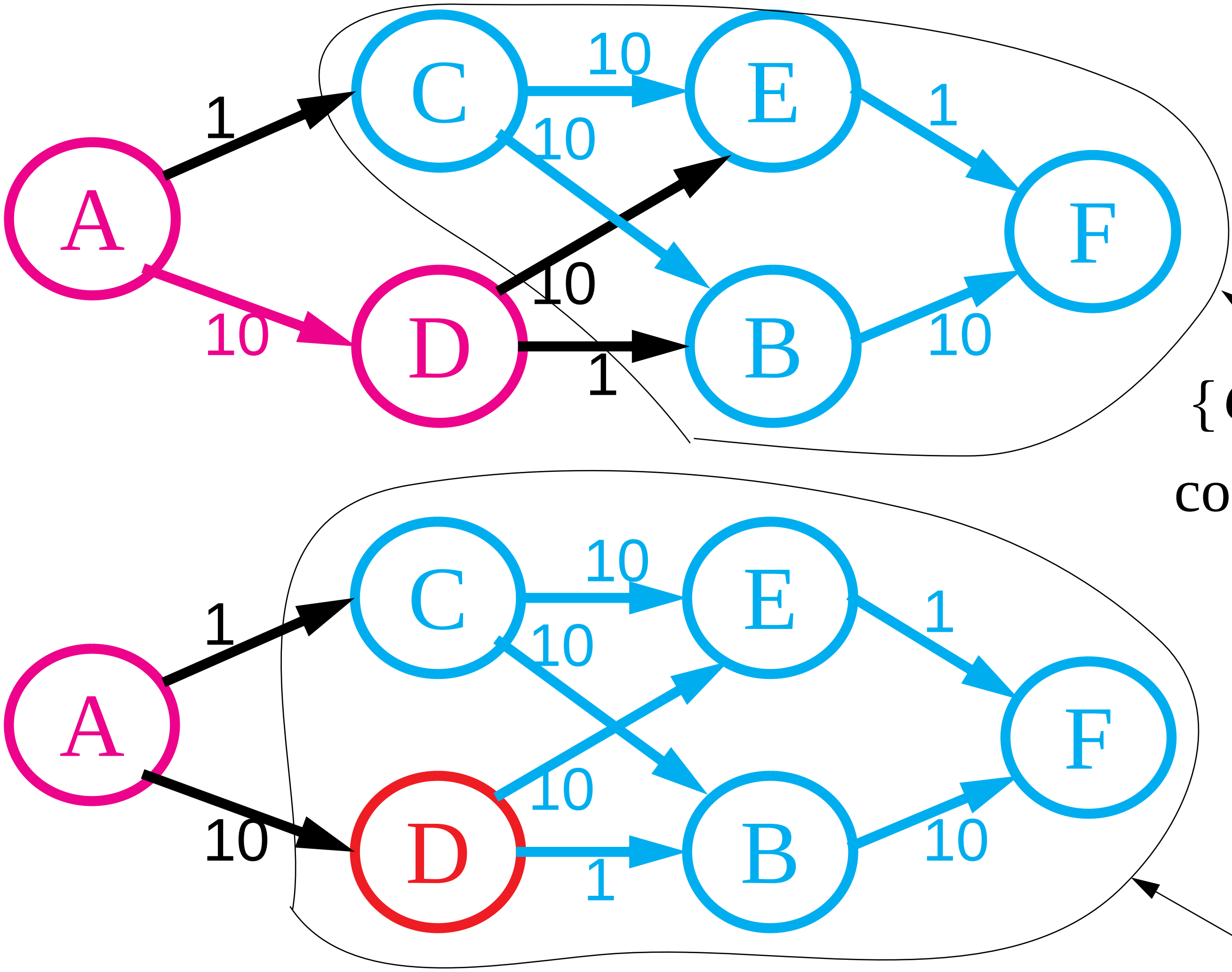
A heuristic for legal minimum cuts

- **Let $V_R(u)$ be the set of nodes consisting of u and its descendents. Let $V_L = V \setminus V_R(u)$.**
- **This forms a cut satisfying condition (*a*).**
- **Perform a local optimization by moving those nodes from V_L that reduce the cost into $V_R(u)$.**
- **Do this for all nodes u in the graph.**

- **Repeat above steps to generate cuts obtained by letting $V_L(u)$ be the set of nodes consisting of u and it ancestors, and letting $V_R = V \setminus V_L(u)$.**
- **Keep the cut with the lowest cost.**

- **Runs in time $O(|V||E| + |V|^2 \bullet \log(|V|))$.**

RPMC example



$\{C\} \cup desc(C)$,

$cost = 12$

$\{C\} \cup desc(C) \cup \{D\}$,

$cost = 11$

Acyclic pairwise grouping of adjacent nodes

Idea: Develop a loop hierarchy by clustering two adjacent nodes at each step.

Definition: *Clustering* means combining two or more nodes into one hierarchical node.

- The graph with the hierarchical node instead of the nodes that were clustered is called the *clustered graph*.

Definition: A *clusterizable* pair of nodes is a pair of nodes that, when clustered, does not cause *deadlock*.

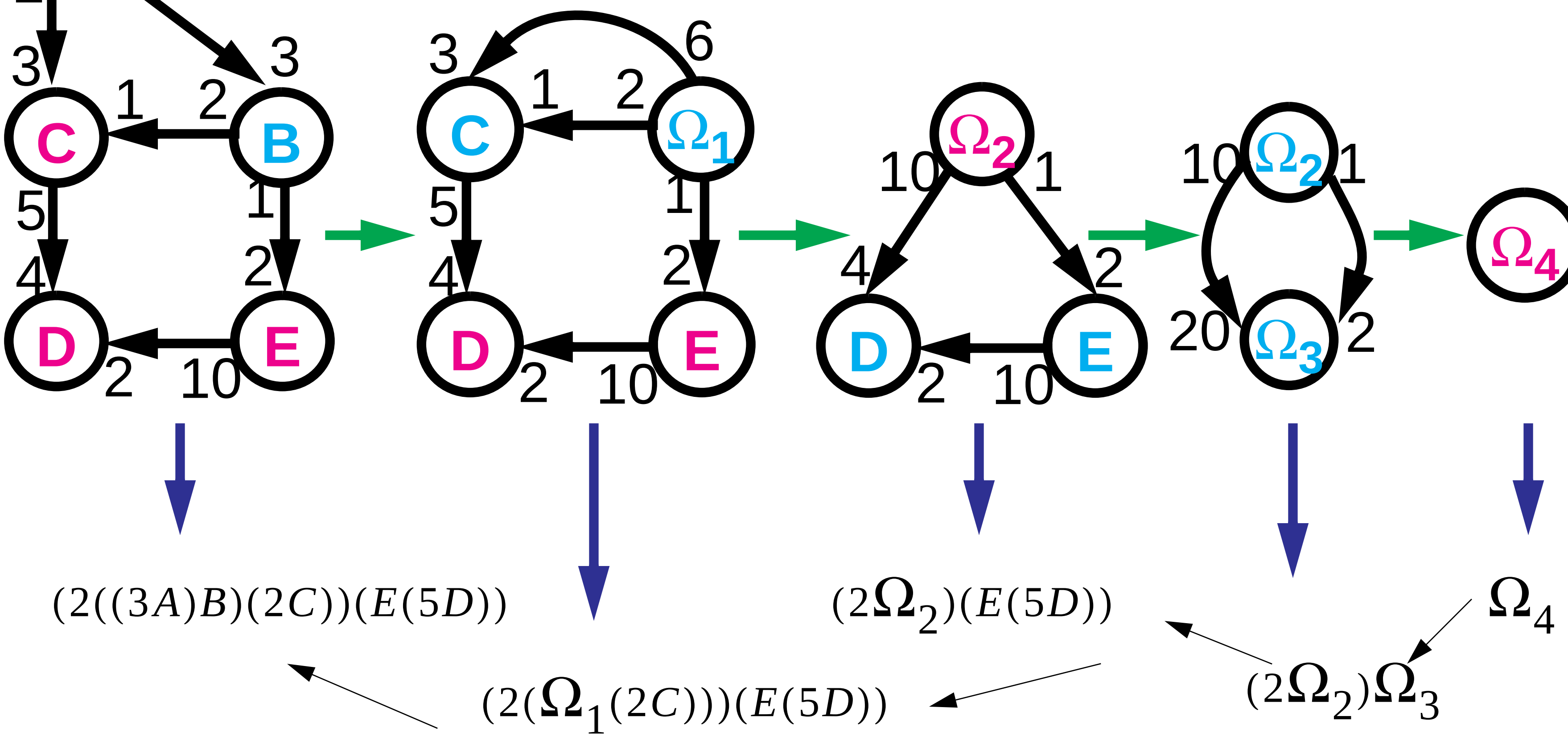
- A sufficient condition for clusterizability: Two nodes are clusterizable if clustering them does not introduce a cycle in the clustered graph.

APGAN algorithm

- **Cluster two nodes that maximize** $gcd\{r(A), r(B)\}$ **over all clusterizable pairs** $\{A, B\}$ **.**
- **Continue until only one node is left in the clustered graph**
- **This is similar to the Huffman coding algorithm.**
- **After constructing cluster hierarchy, retrace steps to determine the nested schedule.**
- **Post-process the schedule using dynamic programming to generate an optimal nesting for the lexical ordering generated by APGAN.**
- **Runs in time** $O(|V|^3)$ **for sparse graphs.**

APGAN example

Cyan nodes are clustered at each step.



Optimality of APGAN

Definition: The *buffer memory lower bound* for a (delayless) arc (e) is given by

$$BMLB(u, v) = \frac{cons(e)prod(e)}{gcd\{cons(e), prod(e)\}}$$

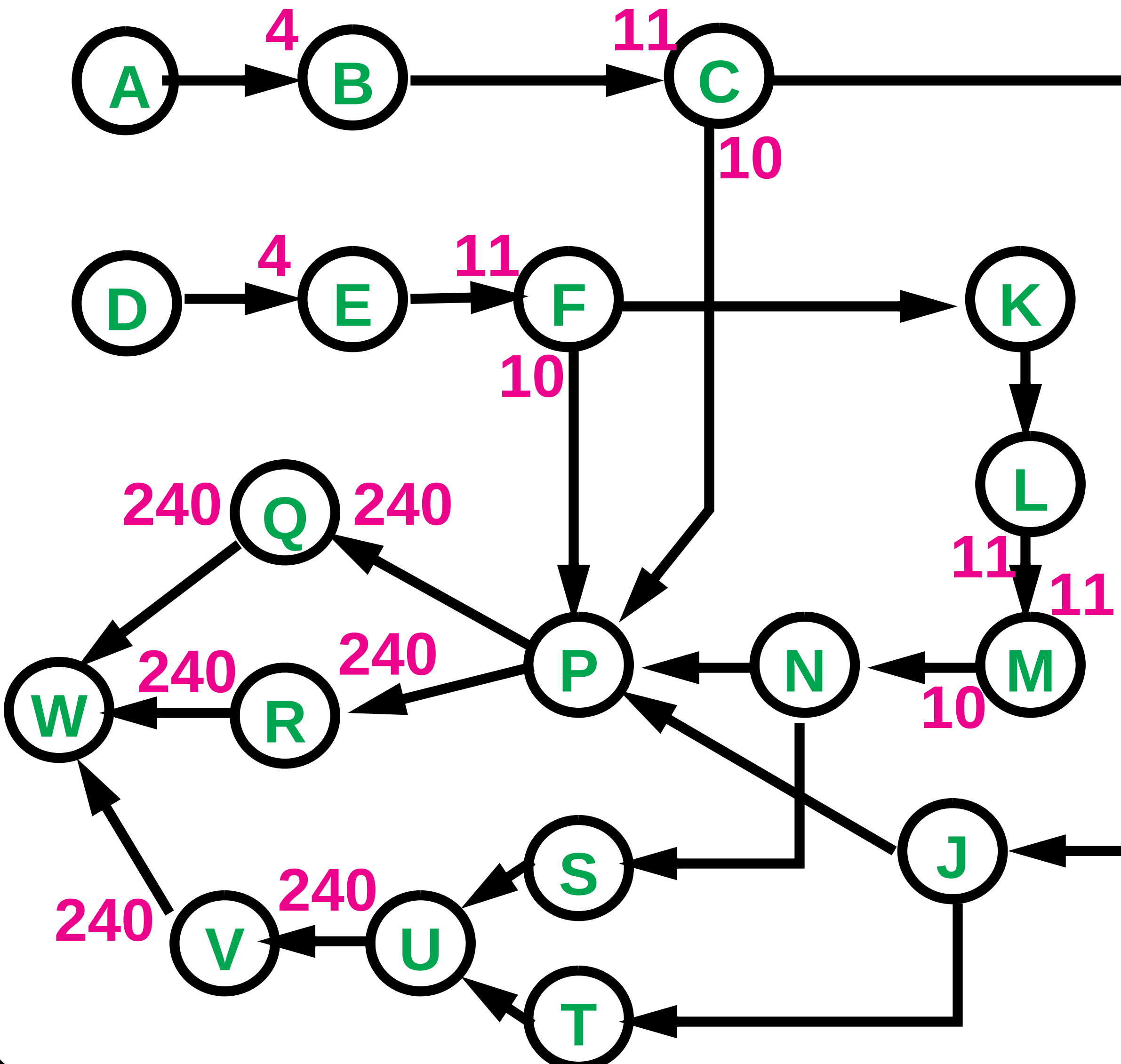
— This represents the least amount of buffering needed on this arc in any single appearance schedule.

Definition: A *BMLB schedule* for an acyclic SDF graph is a single appearance schedule whose buffering cost is equal to the sum of the BMLB costs for each arc.

Theorem: The APGAN algorithm will find a BMLB schedule whenever one exists if $delay(e) < \tau(e)$ for each edge e.

Example: mobile satellite receiver

This example is from [Ritz95]:



BMLB = 1540

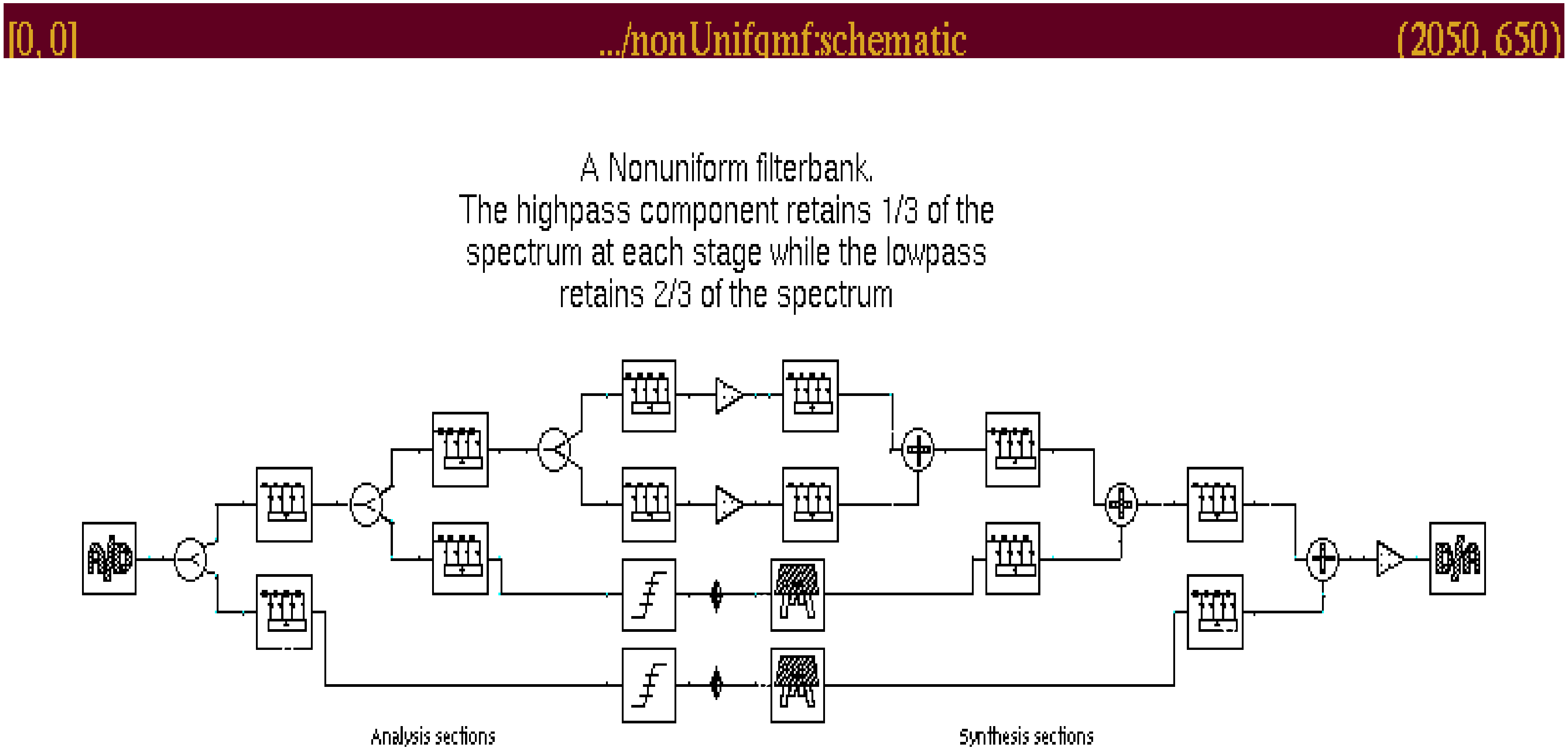
APGAN = 1540

RPMC = 2480

Ritz* = 2040

** Ritz generates a naive single appearance schedule and uses the shared buffer cost.*

Nonuniform filter bank



BMLB = 85

RPMC = 128

APGAN = 137

Performance on practical examples

Performance of the two heuristics on various acyclic graphs.

System	BMUB	BMLB	APGAN	RPMC	Average Random	Graph size(nodes/ arcs)
Fractional decimation	61	47	47	52	52	26/30
Laplacian pyramid	115	95	99	99	102	12/13
Nonuniform filterbank (1/3,2/3 splits, 4 channels)	466	85	137	128	172	27/29
Nonuniform filterbank (1/3,2/3 splits, 6 channels)	4853	224	756	589	1025	43/47
QMF nonuniform-tree filterbank	284	154	160	171	177	42/45
QMF filterbank (one-sided tree)	162	102	108	110	112	20/22
QMF analysis only	248	35	35	35	43	26/25
QMF Tree filterbank (4 channels)	84	46	46	55	53	32/34
QMF Tree filterbank (8 channels)	152	78	78	87	93	44/50
QMF Tree filterbank (16 channels)	400	166	166	200	227	92/106

Performance on random graphs

RPMC < APGAN	63%
APGAN < RPMC	37%
RPMC < min(2 random)	83%
APGAN < min(2 random)	68%
RPMC < min(4 random)	75%
APGAN < min(4 random)	61%
min(RPMC,APGAN) < min(4 random)	87%
RPMC < APGAN by more than 10%	45%
RPMC < APGAN by more than 20%	35%
APGAN < RPMC by more than 10%	23%
APGAN < RPMC by more than 20%	14%

Conclusion

- **Objective: minimizing buffer cost for a minimum code size schedule**
- **Problem is NP-complete, even for acyclic, HSDF graphs.**
- **DPPO: generates an optimum parenthesization for a given lexical ordering. For well-ordered graphs, where there is only one topological sort, DPPO thus generates buffer-optimal single appearance schedules.**
- **Two heuristics are used to generate lexical orderings for arbitrary acyclic SDF graphs:**
  - **RPMC: Does well on some practical examples with irregular topologies and on random graphs**
  - **APGAN: Does well on a lot of practical examples but not as well on random graphs. It is optimal for a class of graphs.**