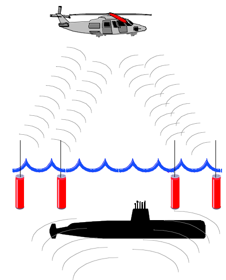
Managing Memory Requirements in the Synthesis of Real-Time Systems from Processing Graphs

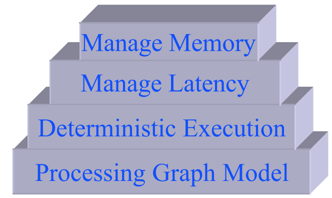
Steve Goddard
Kevin Jeffay

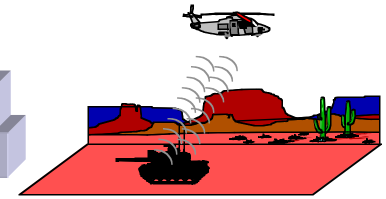
Department of Computer Science
University of North Carolina at Chapel Hill

{goddard,jeffay}@cs.unc.edu
http://www.cs.unc.edu/Research/dirt

Outline



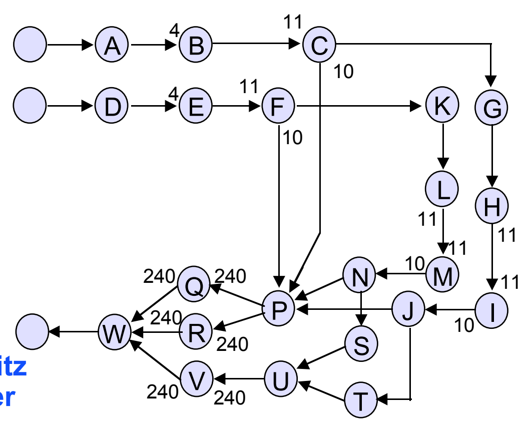




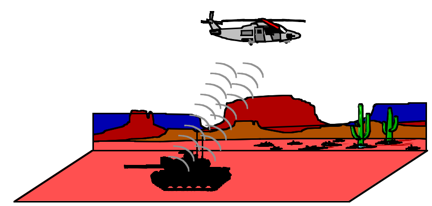
- Introduction
  - » Signal processing graphs
  - » Fundamental design issues
  - » Our approach
  - » Managing memory requirements
- Processing graph model
- Executing nodes
- Managing memory requirements
- Summary

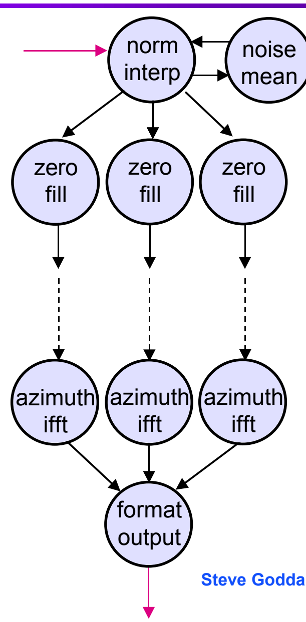
INSMART Satellite Receiver Application

- Our dynamic scheduling algorithm requires memory for
  - » 1,599 tokens for unique buffers
  - » 1,101 tokens for shared buffer
- The AGPAN static scheduling algorithm of [Bhattacharyya, Murthy, and Lee 1996] requires 332% more buffer space
- The scheduling algorithm of [Ritz 1995] requires 377% more buffer space



Fundamental Design Issues

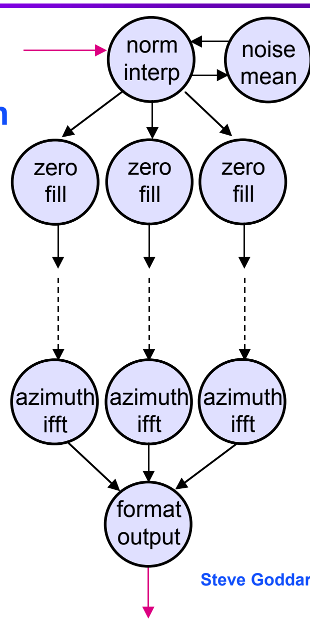




- Can the application meet its hard-real-time processing requirements?
- What is the latency bound?
- How much memory is required?

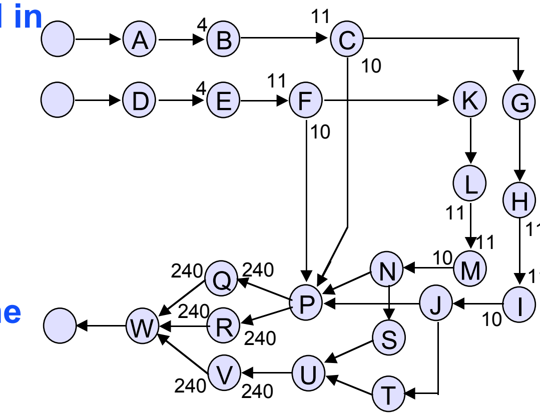
Our Approach

- Use real-time scheduling theory to provide deterministic node execution
  - » Derive node execution rates
  - » Map nodes to real-time tasks
- Derive latency
- Derive memory requirements
- Understand fundamental tradeoffs between latency, memory requirements, and schedulability



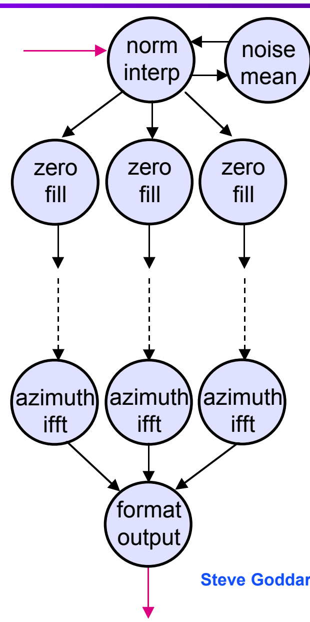
Related Work

- Processing graphs are a general paradigm used in several methodologies
  - » SDF
  - » LASM
  - » SARTOR
  - » RTP/C
  - » . . .
- PGM was created by the U.S. Navy for signal processing



Managing Memory Requirements

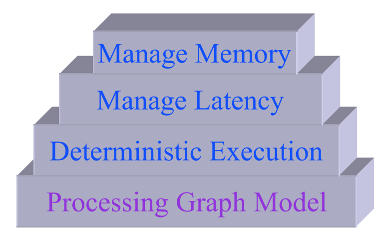
- Space for scheduling algorithm
  - » Code space
  - » State space
- Space for nodes
  - » Store code for nodes as a procedure
- Buffering on graph edges
  - » The amount of intermediate results stored on graph edges can be quite substantial
  - » Schedule to reduce data accumulation on graph edges
  - » Help signal processing engineers create memory efficient graphs



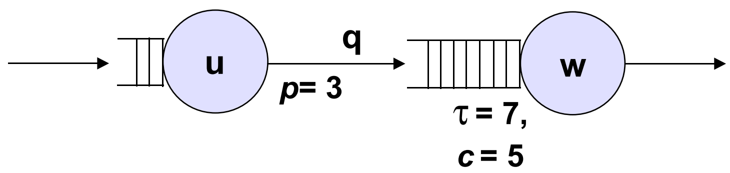
Outline

for the rest of the story

- Processing graph model
  - » PGM
- Executing nodes
  - » Example executions
  - » Derive execution rates
  - » Mapping to real-time tasks
- Managing memory requirements
  - » Focus on buffer requirements
- Summary

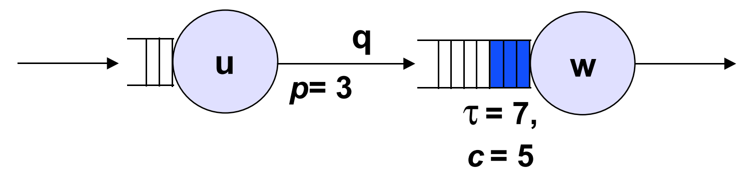


Introduction to the U.S. Navy's *Processing Graph Method* (PGM)



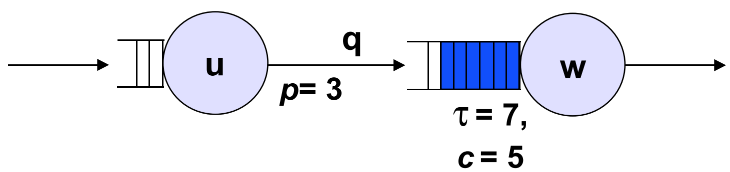
- Each queue has 3 dataflow attributes: *p*, τ, and *c*:
  - *p*: amount produced when a node executes – produce(q)
  - τ: minimum amount required on a queue for the node to execute – threshold(q)
  - *c*: amount consumed when a node executes – consume(q)

PGM



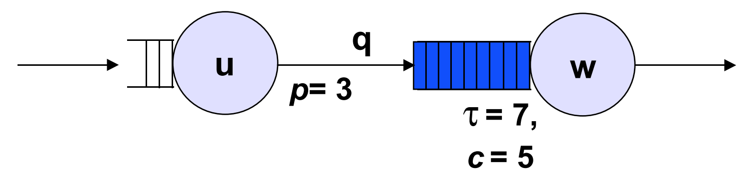
- Node u produces 3 tokens
  - But the input queue to w requires 7 tokens before it is *over threshold*

PGM



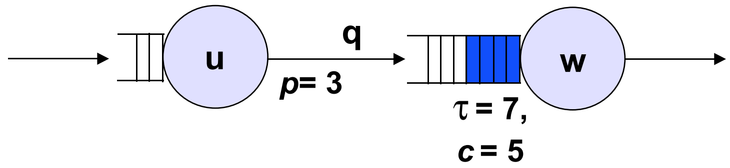
- Node u produces 3 more tokens for a total of 6
  - But the input queue to w requires 7 tokens before it is over threshold

PGM



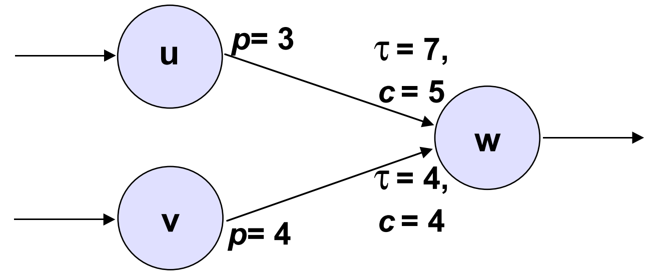
- Node u produces 3 more tokens for a total of 9
  - Now the input queue to w requires is over threshold
- Two types of latency
  - Inherent latency
    - Node w cannot execute until its input queue is over threshold
  - Imposed latency
    - The scheduling creates additional latency if it delays the execution of node w

PGM



- Node w executes and consumes 5 of the 9 tokens leaving 4

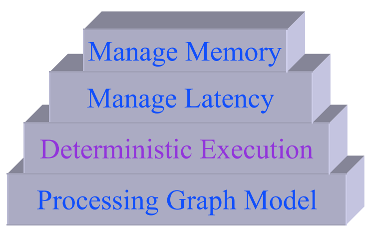
PGM



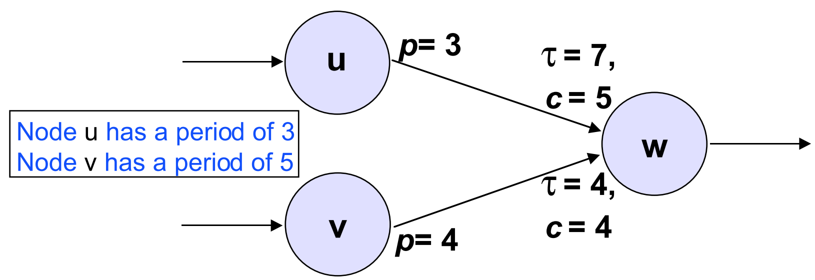
- A node executes when all of its input queues are over threshold
- Both latency and buffer requirements are affected
- May initialize queues with data to reduce initial latency

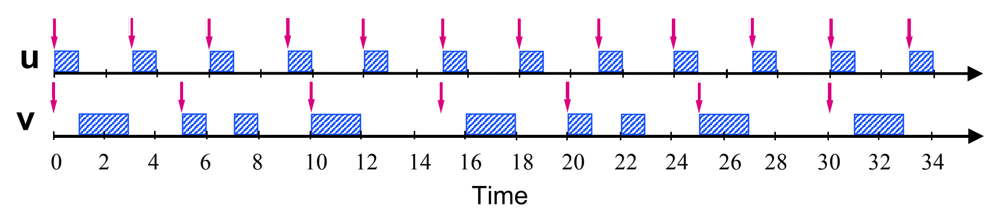
Outline
for the rest of the story

- Processing graph model
  - » PGM
- Executing nodes
  - » Example executions
  - » Derive execution rates
  - » Mapping to real-time tasks
- Managing memory requirements
  - » Focus on buffer requirements
- Summary

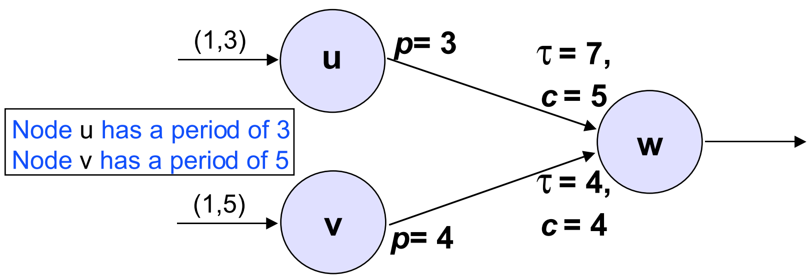


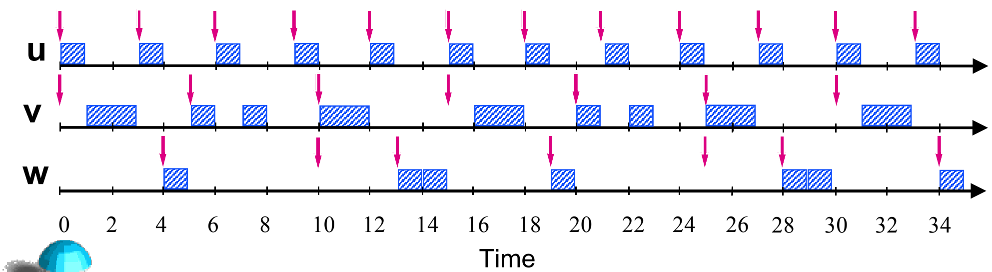
Node Execution Example



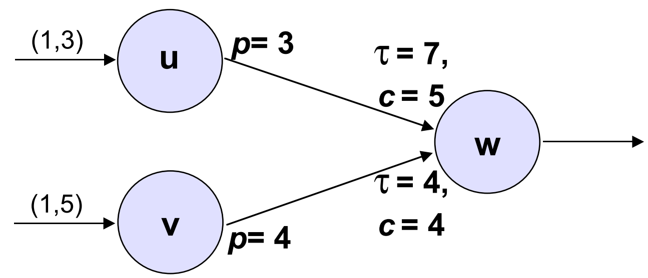


Node Execution Example





Deriving Node Execution Rates



Thm:

$$y(w) = \mathrm{lcm}\left\{ \frac{\mathrm{consume}(q)\cdot y(k)}{\gcd(\mathrm{produce}(q)\cdot x(k),\ \mathrm{consume}(q))} \;\middle|\; k = u,v \right\}$$

$$x(w) = y(w)\frac{\mathrm{produce}(q)\cdot x(k)}{\mathrm{consume}(q)\cdot y(k)}$$

Ex:

$$= \mathrm{lcm}\{15,5\} = 15$$

$$= 15 \times \frac{3\times 1}{5\times 3} = 3$$

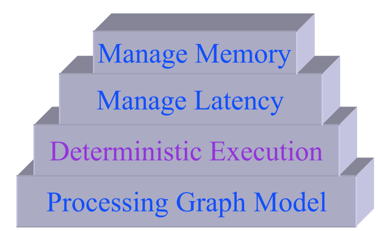
Real-Time Task Model

- **Rate-Based Execution (RBE) Task Model**
  - » ***T*** **= (x,y,d,e): x,y are rate specification, d is relative deadline, and e is worst case execution time**
  - » **No restriction on releases, but deadlines assigned such that no more than x deadlines expire in an interval of length y for task *T*.**
- **EDF Scheduling**
  - » **Of the eligible tasks, the task with the nearest (earliest) deadline is executed first**
- **Issues**
  - » **x,y, and e are constrained, d is free variable**
  - » **d is used to manage memory requirements**

Outline

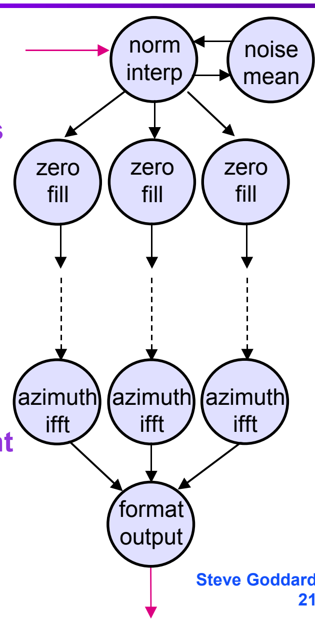
for the rest of the story

- **Processing graph model**
  - » **PGM**
- **Executing nodes**
  - » **Example executions**
  - » **Derive execution rates**
  - » **Mapping to real-time tasks**
- **Managing memory requirements**
  - » **Focus on buffer requirements**
- **Summary**

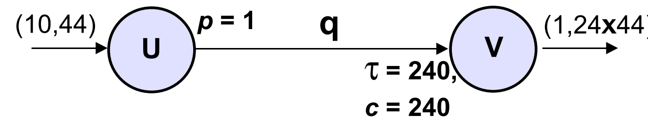


Managing Memory Requirements

- **Scheduling state space**
  - Only need **x**,**y**, and **d** parameters for nodes attached to input devices since other nodes cannot execute faster than their rate specification.
  - Therefore only need **d** parameter for other nodes
- **Space for nodes**
  - Store code for nodes as a procedure
- **Buffering on graph edges**
  - A lot of special cases are needed to get tight buffer bounds
  - Common special cases exist



Buffer Bounds



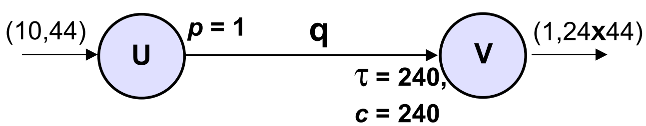
Thm:

$$Buf(q) \le \left\lceil \frac{\max(y(V), s(V)+d(V)-s(U))}{y(U)} \right\rceil x(U) \cdot produce(q) + threshold(q) - consume(q)$$

Solve for d(V):

$$y(V) - s(V) + s(U) \le d(V) \le s(U) - s(V) + \frac{Buf(q) \cdot y(U)}{x(U) \cdot produce(q)}$$

Buffer Bounds



$$y(V) - s(V) + s(U) \le d(V) \le s(U) - s(V) + \frac{Buf(q) \cdot y(U)}{x(U) \cdot produce(q)}$$

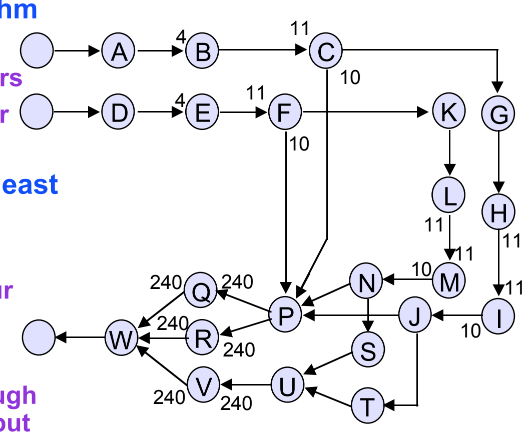
Buf(q)	d(V)
240	44
350	528
470	1056

Ex:

$$24 \times 44 - (24 \times 44 - 1) + 43 \le d(V) \le 43 - (24 \times 44 - 1) + \frac{Buf(q) \times 44}{10 \times 1}$$

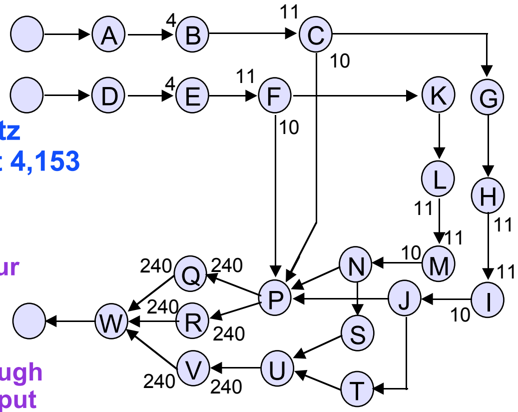
INSMART Satellite Receiver Application

- Our dynamic scheduling algorithm requires memory for
  - 1,599 tokens using unique buffers
  - 1,101 tokens using shared buffer
- The AGPAN static scheduling algorithm requires space for at least 3,655 tokens
  - 2,112 tokens on input queues
  - 332% more buffer space than our dynamic scheduling approach
- Why?
  - Schedule cannot start until enough data has accumulated on the input queues.



(24(11(4A)B)CGHI(11(4D)E)FKLM(10NSJTUP))QRV(240W)

INSMART
Satellite Receiver Application



- The scheduling algorithm of [Ritz 1995] requires space for at least 4,153 tokens
  - » 2,112 tokens on input queues
  - » 377% more buffer space than our dynamic scheduling approach
- Why?
  - » Schedule cannot start until enough data has accumulated on the input queues.

(1056A) (264B) (24C) (24G) (24H) (24I) (240J)
(1056D) (264E) (24F) (24K) (24L) (24M) (240N)
(240P) (240S) (240U) VQR(240W)

Summary

- Real-time scheduling theory is used to provide deterministic node execution so that latency and memory requirements can be managed using dynamic scheduling techniques
- Static scheduling may require significantly more memory than dynamic scheduling
- Software engineering tools that use our framework for evaluating and managing latency and memory requirements would help signal processing engineers
- Our analysis techniques are not limited to signal processing applications