*Proc. 2000 IEEE Conf. on Acoustics, Speech, and Signal Processing*

# Parallel Implementation of Multifilters

*Niranjan Damera-Venkata and Brian L. Evans*

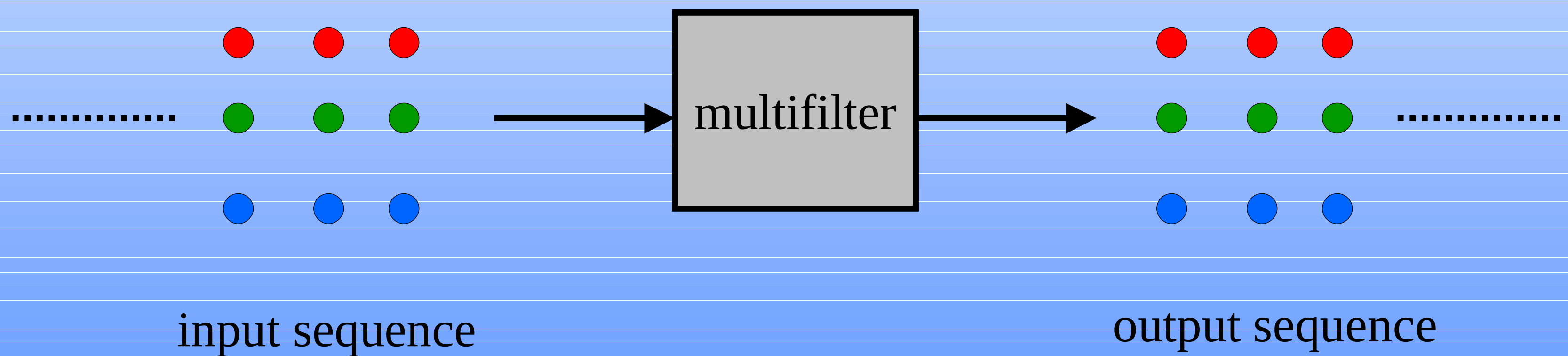
Dept. of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712-1084 USA

{damera-v,bevans}@ece.utexas.edu

http://www.ece.utexas.edu

# Multifilters

- Filters with matrix-valued coefficients
  - Process of vector-valued signals
  - Applications: color signals, multispectral images, and multichannel data




- Problem: Multifiltering is computationally intensive
  - Matrix-vector multiply accumulates
- Solution: Efficient parallel implementation

# Notation

- Signal and multifilter in time domain

$\vec{x}(m)$ *N*-element vector-valued signal

$\overset{\frown}{h}(m)$ *NxN*-matrix valued filter

- Signal and multifilter in frequency domain

$$\vec{X}(z) = \sum_m \vec{x}(m) z^{-m} \qquad \overset{\frown}{H}(z) = \sum_m \overset{\frown}{h}(m) z^{-m}$$

- Multifilter operation

$$\vec{y}(m) = \sum_{k=0}^{M-1} \overset{\frown}{h}(k) \vec{x}(m-k) \qquad \vec{Y}(z) = \overset{\frown}{H}(z) \vec{X}(z)$$

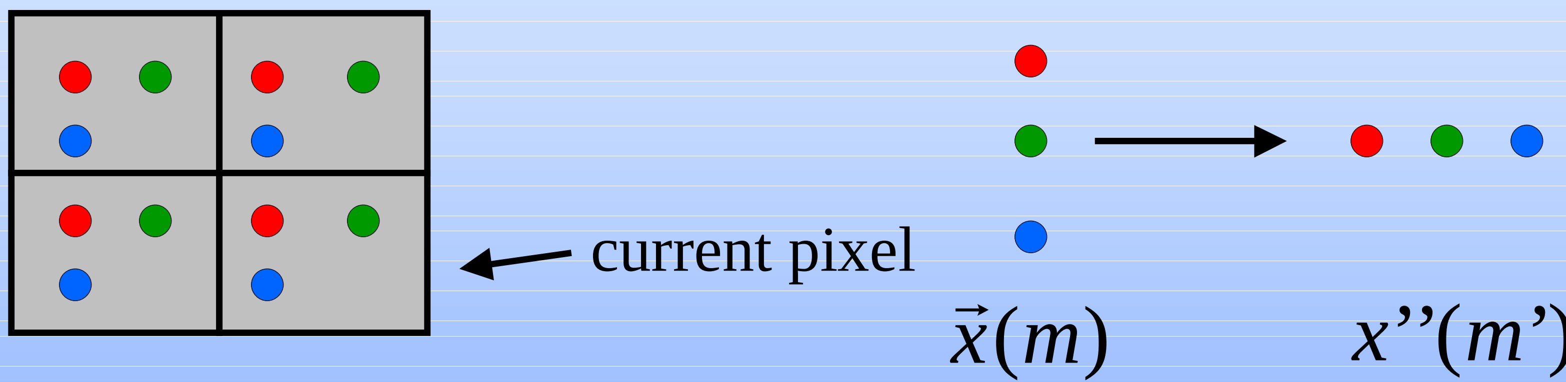
$$\overset{\frown}{\Gamma} = \left[ \overset{\frown}{h}(0) \,|\, \overset{\frown}{h}(1) \,|\, \ldots \,|\, \overset{\frown}{h}(M-1) \right]$$

# Implementation on Digital Signal Processors

- DSPs designed for efficient scalar filtering
- Direct implementation on a single processor
  - $M \times N^2$ multiply-accumulates
  - Inefficient loop code
- Direct implementation on multiple processors
  - Shared circular buffers (of size *M*)
  - Data duplication increases memory
- Solution: Implement multifilters using scalar filtering
  - Does not require buffer sharing
  - Single line loop code
  - Efficient use of hardware looping
  - Speedup of *N* over direct implementation

# Multifilters and Block Filtering

- Reorder vector samples spatially as blocks




- Filter the scalar sequence using scalar filters
  - *n*th sample of output vector sequence is obtained by

$$y_n(m) = \sum_{k=0}^{MN-1} h^{(n)}(k)\, x''(Nm + N - 1 - k)$$

$h^{(n)}(m)$ is formed by reversing nth row of $\widehat{\Gamma}$

# Polyphase Filtering

- Frequency domain representation

$$Y_n(z) = (\downarrow N)\left(H^{(n)}(z)X''(z)z^{(N-1)}\right)$$

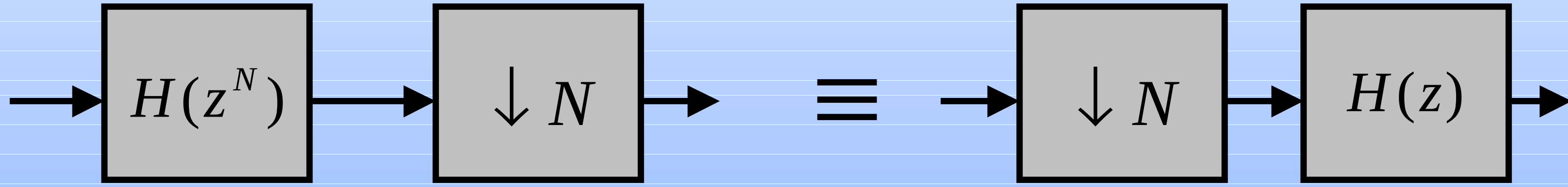
- Polyphase decomposition of the filter

$$H^{(n)}(z) = \sum_{i=0}^{N-1} H_i^{(n)}(z^N)z^{-(N-1-i)}$$

$$Y_n(z) = (\downarrow N)\left(\sum_{i=0}^{N-1} H_i^{(n)}(z^N)X''(z)z^{-(N-1-i)}z^{(N-1)}\right)$$

# Simplification of Polyphase Filtering

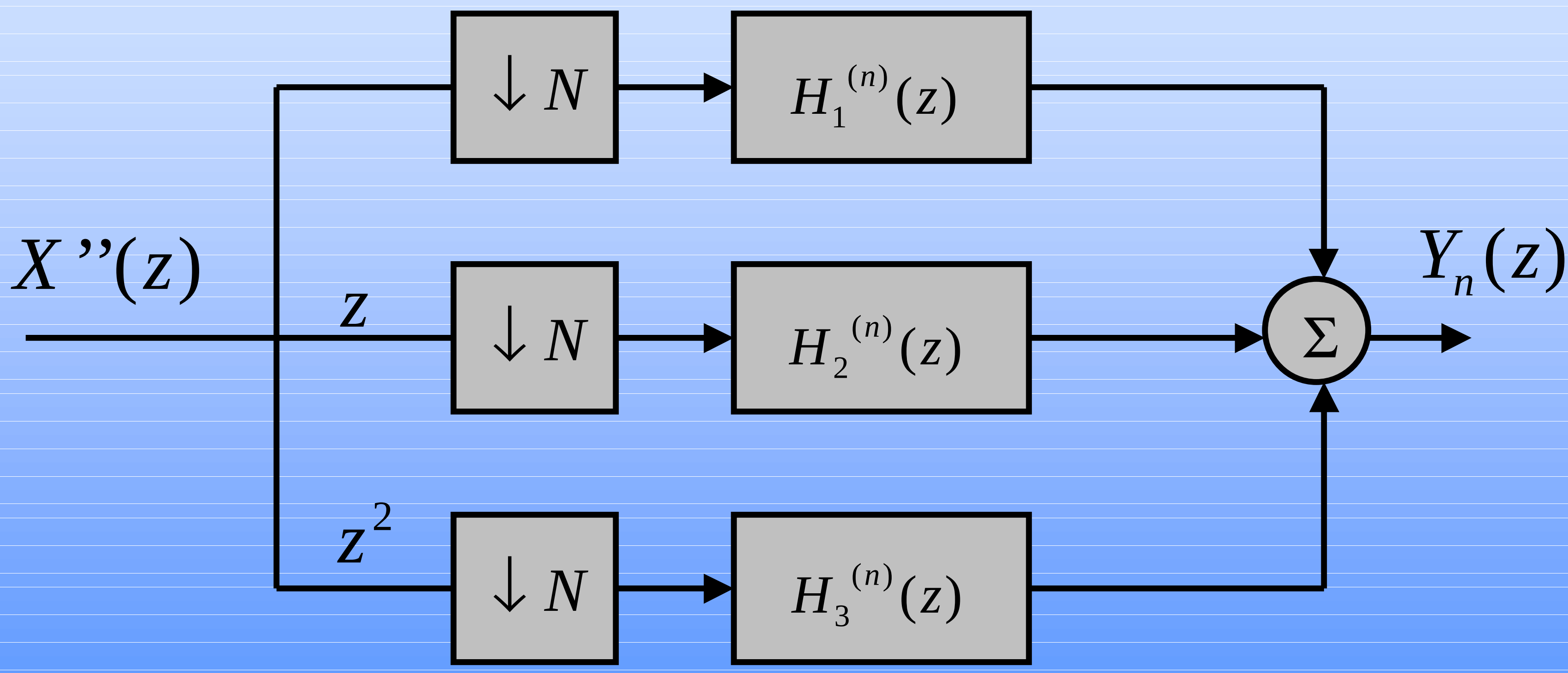
- Commute filtering and downsampling



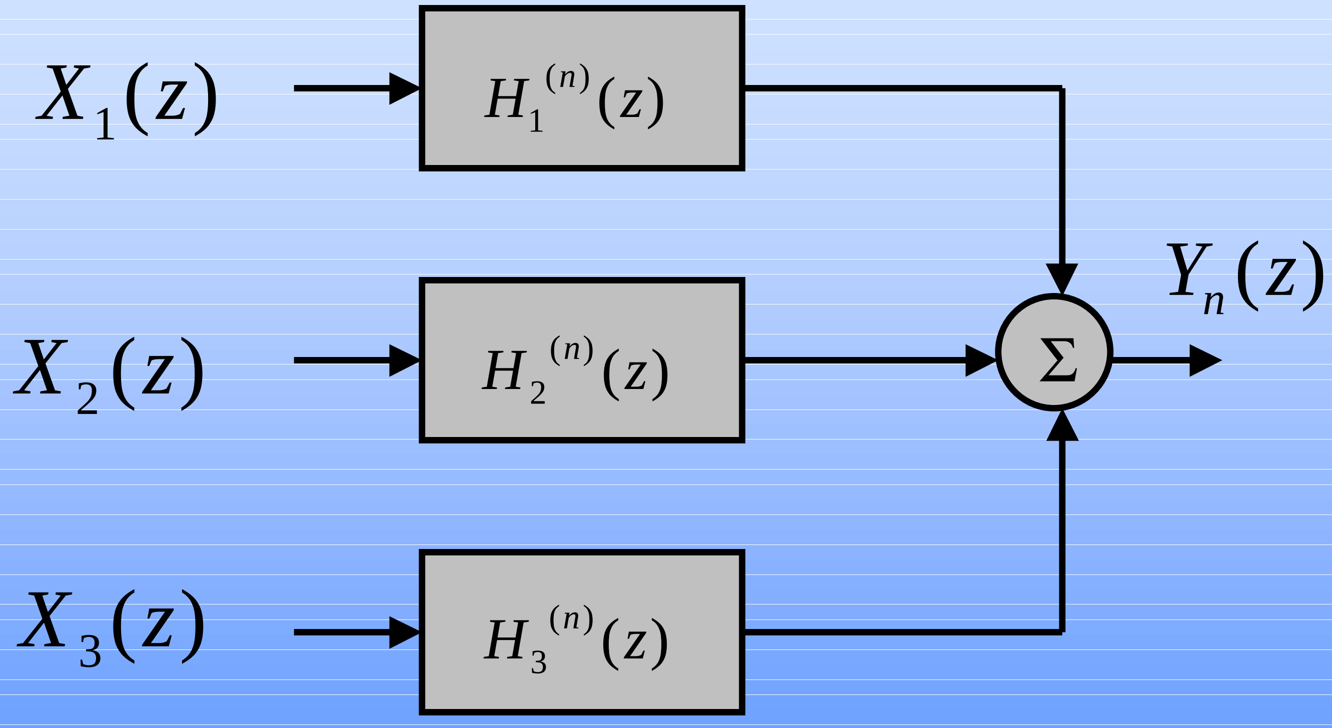

- Noble identity yields

$$Y_n(z) = \sum_{i=0}^{N-1} H_i^{(n)}(z)\left(\downarrow N\right)\left(X''(z) z^i\right)$$

# Parallel Block Filtering



# Parallel Implementation of Multifilters




$$X_i(z) = (\downarrow N)(X''(z) z^i)$$

# Conclusions

- Efficient parallel implementation of multifilters
  - Implement multifilters using scalar filtering operations
  - Each component is buffered and filtered independently
  - Single instruction hardware loops on DSP
  - No data duplication
- Speeds computation by a factor N
  - polyphase filter complexity gain is (*1/N*) for the same throughput
- Simple extension to multidimensional vector-valued sequences