Parallel Recursive Filtering of Infinite Input Extensions

Diego Nehab
IMPA

André Maximo
GE Global Research

GTC 2017
Linear time-invariant filters
Linear

• Invariant to scale
Linear

• Invariant to addition
Time invariant

• Invariant to shift
Convolution

• The convolution of sequences $b$ and $h$ is a sequence $a$

$$b, h, a : \mathbb{Z} \to \mathbb{R} \quad \quad a = b \ast h \quad \quad a_i = \sum_{j=-\infty}^{\infty} b_j h_{i-j}$$
Linear shift-invariant filters are convolutions.
Examples
Outline of talk

• Introduction
  • Recursive filters are very useful
  • Initialization at the boundaries is an important problem

• Exact recursive filtering of infinite input extensions
  • Closed-form formulas available for the first time
  • Enable simple and effective algorithms

• Parallelization
  • Fastest recursive filtering algorithms to date
  • First to filter infinite extensions exactly
General model for convolutions

• Linear difference equations

\[ Az = Bw \]

\[ a_r z_{i-r} + \cdots + a_1 z_{i-1} + a_0 z_i + a_{-1} z_{i+1} + \cdots + a_{-r} z_{i+r} = b_d w_{i-d} + \cdots + b_1 w_{i-1} + b_0 w_i + b_{-1} w_{i+1} + \cdots + b_{-d} w_{i+d} \]
General model for convolutions

- Decompose into direct and recursive parts

\[ \begin{align*}
\text{direct} & \quad \text{filter} & \quad \text{recursive} & \quad \text{z} \\
x_i & = b_s w_i - s + \cdots + b_1 w_{i-1} + b_0 w_i + b_{-1} w_{i+1} + \cdots + b_{-s} w_{i+s} \\
\text{recursive part is the inverse of a convolution} & \quad \text{it is like a linear system} & \quad Az = Bw \quad Az = x \\
\end{align*} \]

\[ \begin{bmatrix}
\vdots \\
x_{i-2} \\
x_{i-1} \\
x_i \\
x_{i+1} \\
x_{i+2} \\
\vdots 
\end{bmatrix}
= \begin{bmatrix}
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots \\
\vdots 
\end{bmatrix}
\begin{bmatrix}
b_s & b_1 & b_0 & b_{-1} & \cdots & b_{-s} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
b_s & b_1 & b_0 & b_{-1} \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots 
\end{bmatrix}
\begin{bmatrix}
w_{i-2} \\
w_{i-1} \\
w_i \\
w_{i+1} \\
w_{i+2} \\
\vdots 
\end{bmatrix} = \begin{bmatrix}
a_0 & a_{-1} & \cdots & a_{-r} \\
a_1 & a_0 & a_{-1} & \cdots & a_{-r} \\
\vdots & \vdots & \ddots & \ddots & \vdots \\
a_r & a_1 & a_0 & a_{-1} & \cdots \\
a_r & a_1 & a_0 & \cdots & \vdots \\
\vdots & \vdots & \vdots & \vdots & \ddots 
\end{bmatrix}
\begin{bmatrix}
z_{i-2} \\
z_{i-1} \\
z_i \\
z_{i+1} \\
z_{i+2} \\
\vdots 
\end{bmatrix}
= \begin{bmatrix}
x_{i-2} \\
x_{i-1} \\
x_i \\
x_{i+1} \\
x_{i+2} \\
\vdots 
\end{bmatrix} \]

finite impulse response support (FIR) \hspace{1cm} infinite impulse response support (IIR)
General model for convolutions

- Decompose recursive part into *causal* and *anticausal* passes

\[ w \xrightarrow{\text{direct}} x \xrightarrow{\text{causal}} \text{recursive} \xrightarrow{\text{anticausal}} z \]

\[
x = Bw \\
Dy = x \\
Az = x \\
Ez = y
\]

causal part is forward-substitution \( Dy = x \)

\( O(rn) \)

\[
y_i = \frac{1}{\sqrt{a_0}} x_i - d_1 y_{i-1} - \cdots - d_r y_{i-r}
\]

anticausal is back-substitution \( Ez = y \)

\( O(rn) \)

\[
z_i = \frac{1}{\sqrt{a_0}} y_i - e_1 z_{i+1} - \cdots - e_r z_{i+r}
\]

\[
\begin{bmatrix}
1 \\
d_1 \\
\vdots \\
d_r \\
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
\vdots \\
1 \\
\end{bmatrix}
\begin{bmatrix}
1 \\
e_1 \\
\vdots \\
e_r \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
\end{bmatrix}
\begin{bmatrix}
1 \\
1 \\
\vdots \\
1 \\
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
\end{bmatrix}
\]

\[
Dy = \sqrt{a_0}
\]

\[
Ez = \sqrt{a_0}
\]
Downsampling with cardinal cubic B-splines

- Input
- Prefiltered
- Cubic B-spline
- Post-processed output
- Cardinal cubic B-spline
- Recursive filter
- Infinite support

[Catmull-Rom]

[Nehab & Hoppe 2014]
Fast image blur

Blur with FIR filter (given $\sigma$)
2$[\sigma]n$ operations

- Direct convolution ($s = 2[\sigma]$)

Blur with FFT
$n \log n$ operations

Blur with IIR filter (any $\sigma$)
6$n$ operations

- Causal pass ($r = 3$)
- Anticausal pass ($r = 3$)

[van Vliet et al. 1998]
What do near input boundaries?
Infinite input extensions

repeat periodically

reflect periodically

constant padding

filter

filter

filter
Tileable textures

- Textures designed to be tiled in a certain way
- Filtering must respect the periodicity
Dealing with boundaries in practice

• In the frequency domain
  • DFT/DCT imply *infinite extensions*
  • Computations are *exact* even for IIR filters

• In the time domain
  • Direct convolution can decide out of bounds input arbitrarily
  • Recursive filters must define out of bounds *outputs*!
Approximation by input padding

- Even more wasted computation
- Filtered
- Even more wasted memory
- Wasted memory
- Filtered
- Wasted computation
- More wasted computation
- Even more wasted memory

Amount of padding depends on impulse response “support”
Outline of talk

• Introduction
  • Recursive filters are very useful
  • Initialization at the boundaries is an important problem

• Exact recursive filtering of infinite input extensions
  • Closed-form formulas available for the first time
  • Enable simple and effective algorithms

• Parallelization
  • Fastest recursive filtering algorithms to date
  • First to filter infinite extensions exactly
Exact recursive filtering of finite input
Exact recursive filtering of finite input

• Instead, first obtain the initial feedbacks
  • To initialize causal pass
  • To initialize subsequent anticausal pass
• How to obtain these feedbacks in closed form?
  • Depends on the choice of infinite input extension
Constant padding

\[ y_0 = M_1 x_0 \]
\[ z_7 = M_2 y_6 + M_3 x_7 \]

precomputed \( r \times r \) matrices

\[ M_1 = S_F \bar{A}_F \]
\[ M_2 = S_{RF} A_F^r \]
\[ M_3 = (S_R \bar{A}_R - S_{RF} A_F^r)M_1 \]

\[ S_{RF} - A_R^r S_{RF} A_F^r = \bar{A}_R \]
\[ S_R = (I - A_R^r)^{-1} \]

input extension

\[ \cdots x_0 \quad x_0 \quad x_0 \quad x_0 \quad x_0 \quad x_0 \quad x_0 \quad x_1 \quad x_2 \quad x_3 \quad x_4 \quad x_5 \quad x_6 \quad x_7 \quad x_7 \quad x_7 \quad x_7 \quad x_7 \quad x_7 \quad \cdots \]

finite input

\[ \cdots y_{-5} \quad y_{-4} \quad y_{-3} \quad y_{-2} \quad y_{-1} \quad y_0 \quad y_1 \quad y_2 \quad y_3 \quad y_4 \quad y_5 \quad y_6 \quad y_7 \quad y_8 \quad y_9 \quad y_{10} \quad y_{11} \quad y_{12} \quad \cdots \]

input extension

\[ \cdots \]

infinite series

\[ \text{initial causal feedback} \]

\[ \cdots \]

finite output

\[ z_1 \quad z_2 \quad z_3 \quad z_4 \quad z_5 \quad z_6 \quad z_7 \quad z_8 \quad z_9 \quad z_{10} \quad z_{11} \quad z_{12} \quad \cdots \]

initial anticausal feedback

\[ \text{causal} \]

\[ \text{anticausal} \]
Periodic repetition

\[ y_6 = M_4 \hat{y}_6 \]
\[ z_1 = M_5 \hat{z}_1 \]

\[ M_4 = (I - A_F^n)^{-1} \]
\[ M_5 = (I - A_R^n)^{-1} \]

\( y_6 = M_4 \hat{y}_6 \)
\( z_1 = M_5 \hat{z}_1 \)

\[ M_4 = (I - A_F^n)^{-1} \]
\[ M_5 = (I - A_R^n)^{-1} \]
Periodic reflection

\[ M_6 = (I - A_F^{2h})^{-1} \left( A_F^h - KA_R^h (A_F^{-1} A_F^r A_R) \right) \]

\[ M_7 = (I - A_F^{2h})^{-1} K (A_R^{-1}) (I - A_F^r A_R) \]

\[ M_8 = (K - A_R^{-1}) A_R \]

\[ M_9 = (K - A_R^{-1}) A_R A_F^h \]

\[ \delta x = M_8 \dot{y}_6 + M_9 y_{12} \]

\[ y_{12} = M_6 \dot{y}_6 + M_7 \ddot{z}_1 \]
Is this correct? Is this fast?

- All required $r \times r$ matrices $M_1$ to $M_9$ exist and can be precomputed
  - Proofs in the paper assume only filter stability
- Exact filtering for all infinite extensions takes $O(r \, n)$
  - May require twice as much computation
- Real advantage comes with parallelization
  - No additional cost
Outline of talk

• Introduction
  • Recursive filters are very useful
  • Initialization at the boundaries is an important problem

• Exact recursive filtering of infinite input extensions
  • Closed-form formulas available for the first time
  • Enable simple and effective algorithms

• Parallelization
  • Fastest recursive filtering algorithms to date
  • First to filter infinite extensions exactly
Modern GPU

- Many multiprocessors each supporting many hardware threads
  - 10k threads is typical
- On-chip shared memory within each multiprocessor
  - 48k to be shared by threads local to multiprocessor
- Global memory with high throughput but high latency
  - High throughput but high latency
- Challenge is hide latency and keeping all cores busy
Independent rows then columns

\[ \begin{array}{c}
\begin{array}{cccccccccccccccc}
x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 & x_{10} & x_{11} & x_{12} & x_{13} & x_{14} & x_{15} & x_{16} \\
\end{array}
\end{array} \]

input

\[ \begin{array}{c}
\begin{array}{cccccccccccccccc}
y_1 & y_2 & y_3 & y_4 & y_5 & y_6 & y_7 & y_8 & y_9 & y_{10} & y_{11} & y_{12} & y_{13} & y_{14} & y_{15} & y_{16} \\
\end{array}
\end{array} \]

causal

\[ \begin{array}{c}
\begin{array}{cccccccccccccccc}
z_1 & z_2 & z_3 & z_4 & z_5 & z_6 & z_7 & z_8 & z_9 & z_{10} & z_{11} & z_{12} & z_{13} & z_{14} & z_{15} & z_{16} \\
\end{array}
\end{array} \]

anticausal

output

[Ruijters and Thevenaz 2010]
## Summary of parallel algorithms

$r$ is the filter order

$p$ is the number of processors

$h$ is the image height

$w$ is the image width

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Step complexity</th>
<th>Threads</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent rows then columns</td>
<td>$4r \frac{hw}{p}$</td>
<td>$h, w$</td>
<td>$8hw$</td>
</tr>
</tbody>
</table>
Performance on Fermi (GF100)

Throughput (GiP/s)

1st order

-independent rows then columns

Input size (pixels)

[64^2, 128^2, 256^2, 512^2, 1024^2, 2048^2, 4096^2]

[Nehab, Maximo, Lima & Hoppe 2011]
Split rows and columns into blocks

\[ y_{bi} = A^b_R y_{b(i-1)} + \dot{y}_{bi} \]

\[ z_{bi} = A^b_R z_{b(i+1)} + \dot{z}_{bi} \]

[Sung & Mitra 1986; Nehab, Maximo, Lima & Hoppe 2011]
## Summary of parallel algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Step complexity</th>
<th>Threads</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent rows then columns</td>
<td>$4r \frac{hw}{p}$</td>
<td>$h, w$</td>
<td>$8hw$</td>
</tr>
<tr>
<td>+ split rows and columns</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$\frac{hw}{b}$</td>
<td>$\approx 9hw$</td>
</tr>
</tbody>
</table>

$r$ is the filter order  
$p$ is the number of processors  
$h$ is the image height  
$w$ is the image width
Performance on Fermi (GF100)

Throughput (GiP/s) vs. Input size (pixels)

- Independent rows then columns
- + Split rows and columns

[Sources: Nehab, Maximo, Lima & Hoppe 2011]
Overlap causal-anticausal processing

\[
y_{bi} = A^b_F y_{b(i-1)} + \dot{y}_{bi}
\]

\[
z_{bi} = A^b_R z_{b(i+1)} + H(A_{RB})A_{FP} y_{b(i-1)} + \ddot{z}_{bi}
\]

[Overlap causal-anticausal processing by Nehab, Maximo, Lima & Hoppe 2011]
## Summary of parallel algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Step complexity</th>
<th>Threads</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent rows then columns</td>
<td>$4r \frac{hw}{p}$</td>
<td>$h, w$</td>
<td>$8hw$</td>
</tr>
<tr>
<td>+ split rows and columns</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$hw/b$</td>
<td>$\approx 9hw$</td>
</tr>
<tr>
<td>+ overlap causal with anticausal</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$hw/b$</td>
<td>$\approx 5hw$</td>
</tr>
</tbody>
</table>

$r$ is the filter order  
$p$ is the number of processors  
$h$ is the image height  
$w$ is the image width
Performance on Fermi (GF100)

![Graph showing throughput vs. input size]

Throughput (GiP/s) vs. Input size (pixels)

- **1st order**
  - Independent rows then columns
  - + split rows and columns
  - + overlap causal and anticausal

[Source: Nehab, Maximo, Lima & Hoppe 2011]
Overlap causal, anticausal, rows, columns

[Nehab, Maximo, Lima & Hoppe 2011]
Stage 1
Stage 2

\[
\begin{align*}
  z_{m,n} &= A_R^t z_{m+1,n} + \sum (A_{RB} A_{FB}^t y_{m-1,n} A_{RE} \tilde{y}_{m,n} - y_{m+1,n} A_{RB} A_{FP} y_{m-1,n}) + (H A_{RB} A_{FB})^t \tilde{v}_{m,n}
  
  &+ (H A_{RB} A_{FB})^t (H A_{RB} A_{FB})^t \tilde{v}_{m,n}
\end{align*}
\]
Stage 3
Stage 3
## Summary of parallel algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Step complexity</th>
<th>Threads</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent rows then columns</td>
<td>4( r ) ( hw/p )</td>
<td>( h, w )</td>
<td>8( hw )</td>
</tr>
<tr>
<td>+ split rows and columns</td>
<td>( \approx 8r ) ( hw/p )</td>
<td>( hw/b )</td>
<td>( \approx 9hw )</td>
</tr>
<tr>
<td>+ overlap causal with anticausal</td>
<td>( \approx 8r ) ( hw/p )</td>
<td>( hw/b )</td>
<td>( \approx 5hw )</td>
</tr>
<tr>
<td>+ overlap rows with columns</td>
<td>( \approx 8r ) ( hw/p )</td>
<td>( hw/b )</td>
<td>( \approx 3hw )</td>
</tr>
</tbody>
</table>

\( r \) is the filter order
\( p \) is the number of processors
\( h \) is the image height
\( w \) is the image width
Performance on Fermi (GF100)

Throughput (GiP/s)

1st order

- independent rows then columns
- + split rows and columns
- + overlap causal and anticausal
- + overlap rows and columns

Input size (pixels)

[64^2, 128^2, 256^2, 512^2, 1024^2, 2048^2, 4096^2]

[Nehab, Maximo, Lima & Hoppe 2011]
Performance on Fermi (GF100)

Throughput (GiP/s)

Input size (pixels)

2nd order

overlap causal and anticausal

+ overlap rows and columns

[Nehab, Maximo, Lima & Hoppe 2011]
New trick for better performance

sequentially find per-block column feedbacks

\[ y_{m,n} = A_F^b y_{m-1,n} + \hat{y}_{m,n} \]

\[ z_{m,n} = A_R^b z_{m+1,n} + H(A_{RB})A_{FP} y_{m-1,n} + \hat{z}_{m,n} \]

\[ \tilde{u}_{m,n} = (A_{RE} z_{m+1,n} + A_{RB} A_{FB} y_{m-1,n}) T(A_{FB})^t + \tilde{u}_{m,n} \]

\[ \tilde{v}_{m,n} = (A_{RE} z_{m+1,n} + A_{RB} A_{FP} y_{m-1,n}) (H(A_{RB}) A_{FB})^t + \tilde{v}_{m,n} \]

\[ u_{m,n} = u_{m,n-1} (A_F^b)^t + \tilde{u}_{m,n} \cdot z_{m+1,n} + A_{RB} A_{FB} y_{m-1,n} ) T(A_{FB})^t + \tilde{u}_{m,n} \]

\[ v_{m,n} = v_{m,n+1} (A_R^b)^t + u_{m,n-1} (H(A_{RB}) A_{FP})^t + \tilde{v}_{m,n} \cdot z_{m+1,n} + A_{RB} A_{FP} y_{m-1,n} ) (H(A_{RB}) A_{FB})^t + \tilde{v}_{m,n} \]

sequentially find per-block row feedbacks

(exactly the same as column processing)
## Summary of parallel algorithms

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Step complexity</th>
<th>Threads</th>
<th>Bandwidth</th>
</tr>
</thead>
<tbody>
<tr>
<td>independent rows then columns</td>
<td>$4r \frac{hw}{p}$</td>
<td>$h, w$</td>
<td>$8hw$</td>
</tr>
<tr>
<td>+ split rows and columns</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$\frac{hw}{b}$</td>
<td>$\approx 9hw$</td>
</tr>
<tr>
<td>+ overlap causal with anticausal</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$\frac{hw}{b}$</td>
<td>$\approx 5hw$</td>
</tr>
<tr>
<td>+ overlap rows with columns</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$\frac{hw}{b}$</td>
<td>$\approx 3hw$</td>
</tr>
<tr>
<td>+ new trick</td>
<td>$\approx 8r \frac{hw}{p}$</td>
<td>$\frac{hw}{b}$</td>
<td>$\approx 3hw$</td>
</tr>
</tbody>
</table>

$r$ is the filter order  
$p$ is the number of processors  
$h$ is the image height  
$w$ is the image width
Performance on Kepler (GK110)

Throughput (GiP/s) vs. Input size (pixels)

3rd order

4th order

- overlap causal and anticausal
- + overlap rows and columns
- + new trick
Performance on Pascal (GP102)

5th order

- overlap causal and anticausal
- + overlap rows and columns
- + new trick

Throughput (GiP/s) vs Input size (pixels)
Back to infinite extensions

\[ M_6 = (I - A_F^{2h})^{-1} \left( A_F^h - KA_R^h (A_F)^{-1} A_F^r \bar{A}_R \right) \]

\[ y_{12} = M_6 \ddot{y}_6 + M_7 \ddot{z}_1 \]

\[ \delta x = M_8 \dot{y}_6 + M_9 y_{12} \]

\[ M_7 = (I - A_F^{2h})^{-1} K (A_R)^{-1} (I - A_F^r A_R^r) \]

\[ M_8 = (K - A_R^r)^{-1} \bar{A}_R \]

\[ M_9 = (K - A_R^r)^{-1} \bar{A}_R A_F^h \]
Back to infinite extensions

• Side effect of 2nd stage of parallel algorithm
• Just what we need to compute exact initial feedbacks!
Performance on Kepler (GK110)

1st order (cubic B-spline interpolation)

![Graph showing throughput vs. input size](image)

- Padding
- Exact clamp to border
- Exact repeat
- Exact reflect

[Chaurasia et al. 2015]

[Chaurasia et al. 2015]
Performance on Kepler (GK110)

3rd order (2D Gaussian blur with $\sigma = n/6$)

Throughput (GiP/s)

Input size (pixels)
Summary

• Introduction
  • Recursive filters are very useful
  • Initialization at the boundaries is an important problem

• Exact recursive filtering of infinite input extensions
  • Closed-form formulas available for the first time
  • Enable simple and effective algorithms

• Parallelization
  • Fastest recursive filtering algorithms to date
  • First to filter infinite extensions exactly
Thank you!

• Please download and use our code

https://github.com/andmax/gpufilter

• Questions?