6th International Conference Logic and Applications
LAP 2017, September 18-22, 2017, Dubrovnik, Croatia

On the Computational Complexity of the Discrete Pascal Transform

Dušan B. Gajić[1], Radomir S. Stanković[2]

[1]University of Novi Sad, Faculty of Technical Sciences, Dept. of Computing and Control
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia

[2]University of Niš, Faculty of Electronic Engineering, Dept. of Computer Science
Aleksandra Medvedeva 14, 18000 Niš, Serbia

E-mail: [1]dusan.gajic@uns.ac.rs, [2]radomir.stankovic@gmail.com

Outline

1. Spectral transforms, Pascal's triangle, and Pascal's matrix
2. The discrete Pascal transform (DPT) and the fast Pascal transform (FPT) algorithm
3. The algorithm for computing the DPT using the Toeplitz matrix and the FFT
4. Implementation of the considered algorithms
5. Experimental settings and results
6. Conclusions

Spectral Transforms

signal (function)

↓ apply

spectral transform

↓ achieve

redistribution of information content

↓ perform in spectral domain

1. easier observation of some properties of signals
2. more efficient computation of certain operations

Applications:
Digital logic design
(spectral transforms over GF(p) and ring of integers modulo p),
Digital signal processing, pattern recognition…

Spectral Transforms

Spectral transforms are mathematical operators in linear vector spaces which assign to a function *f* a corresponding spectrum S_f defined as

$$\mathbf{S}_f = \mathbf{T}^{-1}\mathbf{F},$$

$\mathbf{T}$ - Matrix with basis functions as columns

$\mathbf{F}$ - Functional vector for *f*

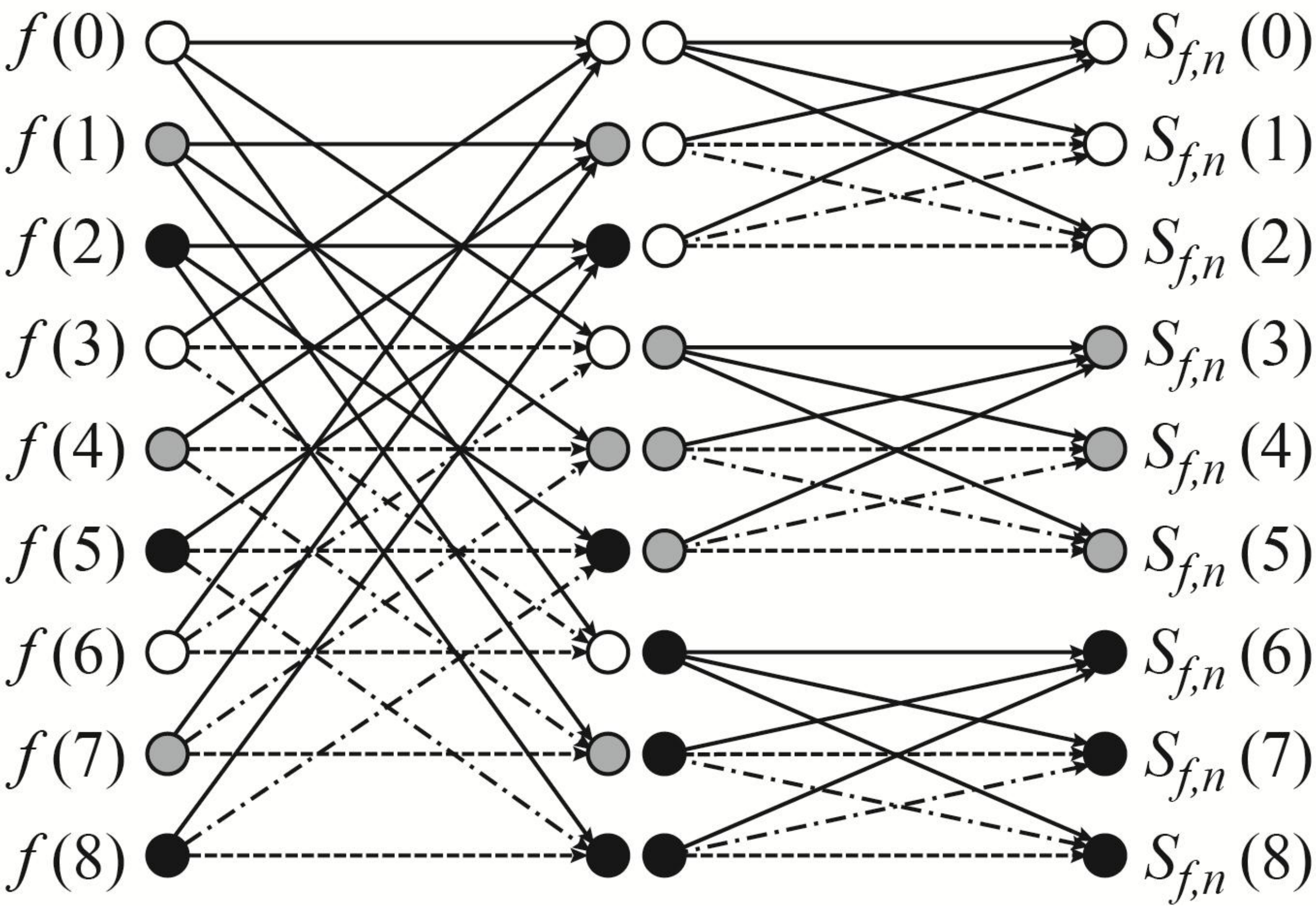
$$f:\{0,1,\ldots,p-1\}^n \rightarrow \{0,1,\ldots,p-1\} \Rightarrow \mathbf{F} = [f(0), f(1),\ldots, f(p^n-1)]^T$$

$$\mathbf{S}_f = [s_f(0), s_f(1),\ldots, s_f(p^n-1)]^T \quad \mathbf{S}_f = \begin{bmatrix} transform \\ matrix \end{bmatrix} \cdot \mathbf{F} \Rightarrow O(N^2)$$

Function is reconstructed from the spectrum as: $\mathbf{F} = \mathbf{T}\mathbf{S}_f$

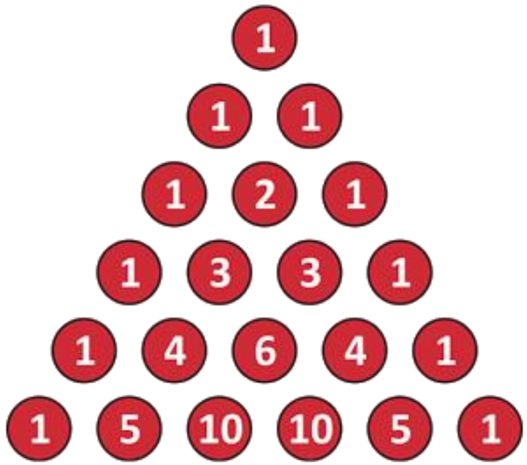
Fast algorithms are based on the **factorization** of the **transform matrix** into **sparse matrices** $\Rightarrow O(N \log N)$

Example: Vilenkin-Chrestenson Transform p=3, n=2



Pascal's Triangle and Pascal's Matrix

- **Pascal's triangle** is obtained through the arrangement of binomial coefficients, appearing in the expansion of $(x+y)^q$, where q is a non-negative integer, in staggered rows by the increasing values of q

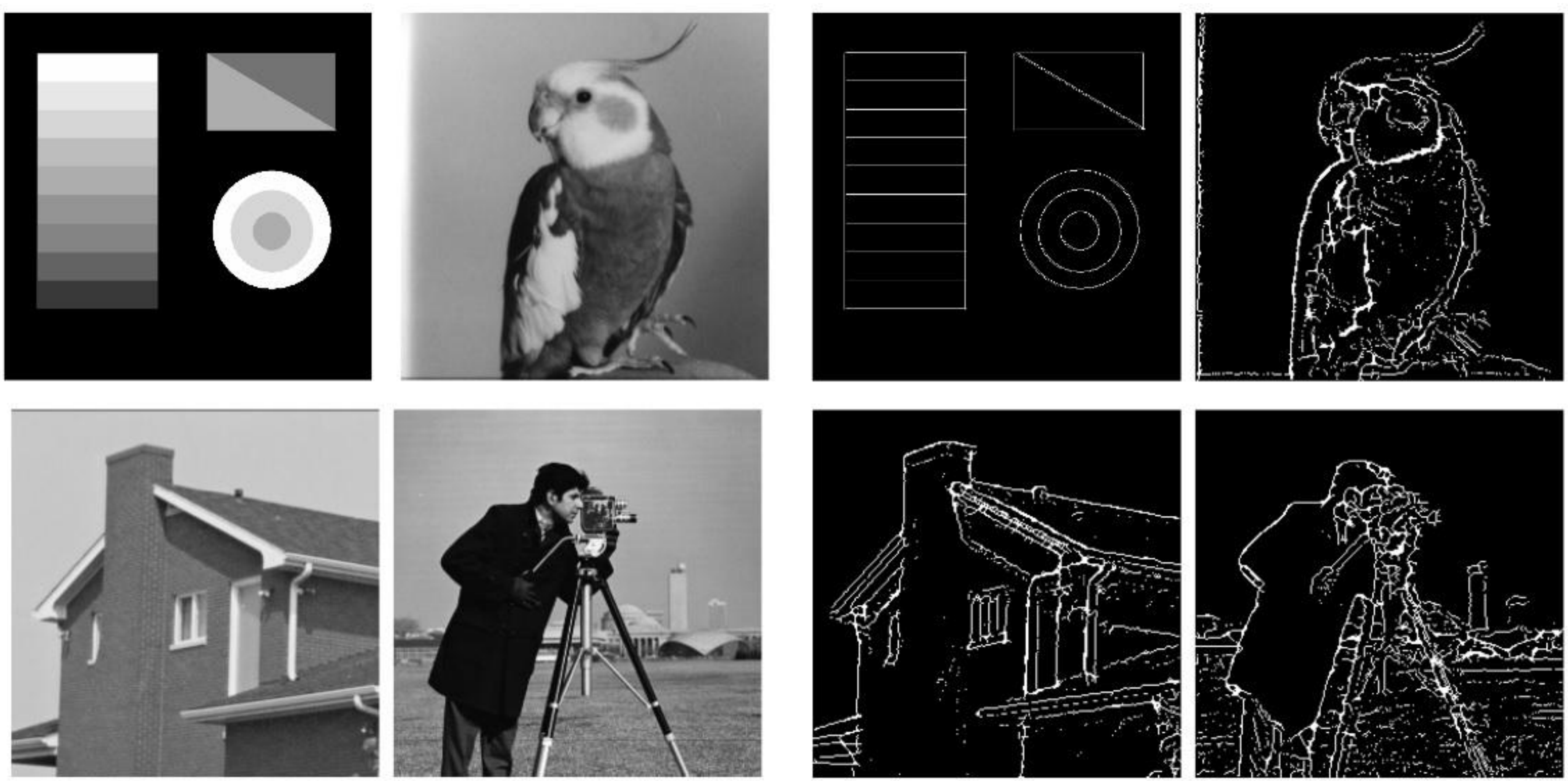


- When rows are aligned to the left margin and zeros are added to complete the square, we get the **lower triangular Pascal matrix**
- Three forms of Pascal's matrix: **lower triangular**, **upper triangular**, and **symmetric**

The Discrete Pascal Transform (DPT)

- Introduced by Aburdene and Goodman in 2005, by an **ad hoc multiplication with -1 of every other column of the Pascal matrix**
- Used in **digital image processing**, **pattern recognition**, **digital watermarking**, and related areas
- DPT computing time is a limiting factor in many of its applications
- DPT modulo p equals the Reed-Muller-Fourier (RMF) transform
- Since Pascal's matrix does not have the Kronecker product structure, the FFT-like algorithms cannot be devised directly
- **Current algorithms** for computing the DPT have $O(N^2)$ asymptotical time complexity
- Find **a method** for computing the DPT in $O(N \log N)$

Edge Detection by using the DPT



Source: Lin, R. S., ”A simple edge detection method by discrete Pascal transformation”, Research Report, Taiwan, 2006.

The Discrete Pascal Transform (DPT)

- The **discrete Pascal transform** of a function f, specified in N points and represented by its function vector $\mathbf{F}$ of length N, is defined as

$$\mathbf{S}_f = \mathbf{P} \cdot \mathbf{F}$$

  where $\mathbf{P}$ is the $(N \times N)$ **discrete Pascal transform matrix** and S_f is the **Pascal spectrum** of f
- The DPT transform matrices for N = 2, 3, 4:

$$\mathbf{P}_2 = \begin{bmatrix} 1 & 0 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{P}_3 = \begin{bmatrix} 1 & 0 & 0 \\ 1 & -1 & 0 \\ 1 & -2 & 1 \end{bmatrix} \qquad \mathbf{P}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix}$$

Computing the DPT by its Definition

- **Example 1**: The DPT of a function f, represented by its function vector $\mathbf{F} = [1, 2, 3, 6]^{\mathrm{T}}$, can be directly computed as

$$\mathbf{S}_f(4) = \mathbf{P}_4 \cdot \mathbf{F} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 6 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ 0 \\ 2 \end{bmatrix}$$

- This algorithm requires N^2 multiplications and $N(N-1)$ additions

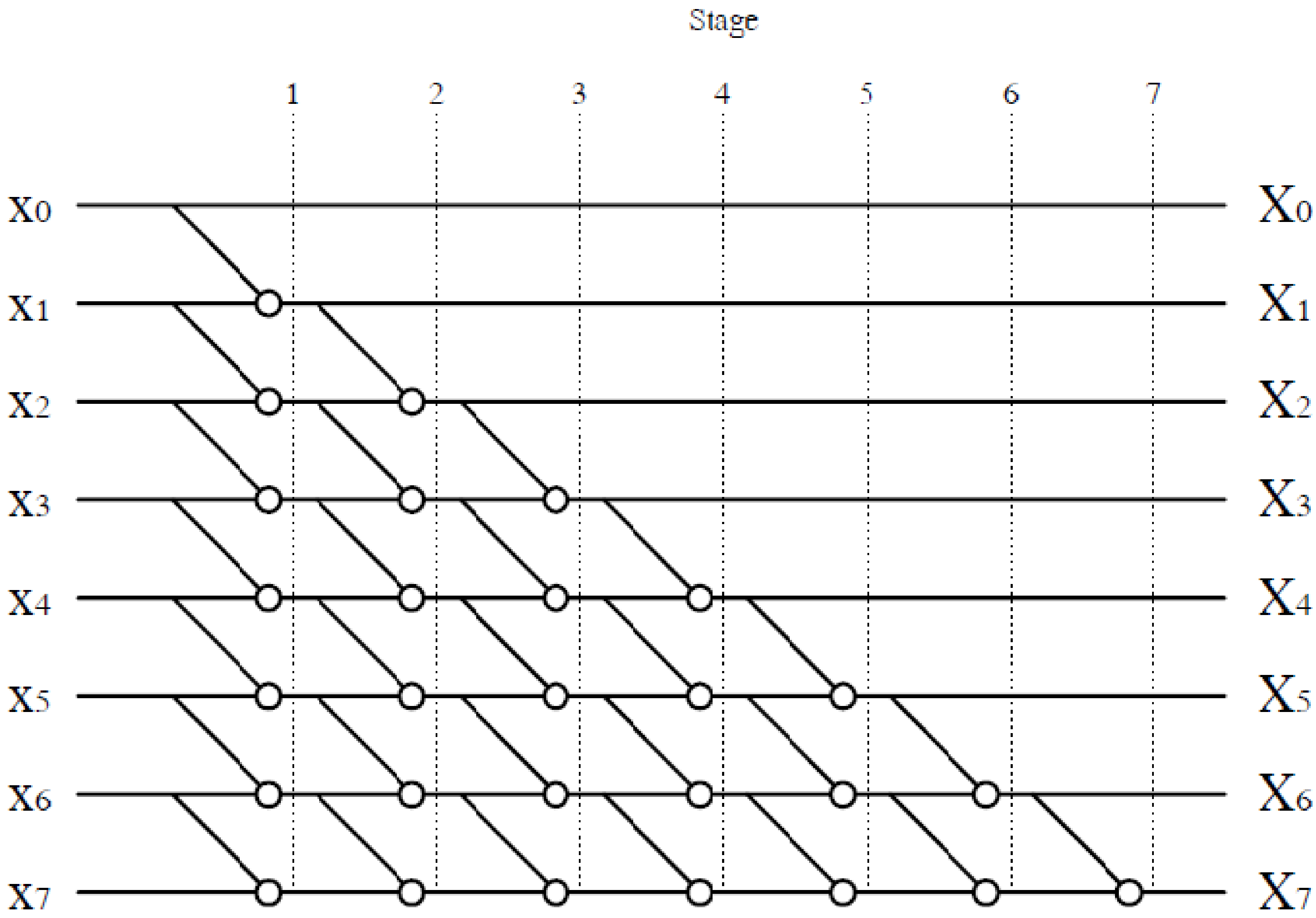
- Arithmetic complexity can be reduced by using the triangular matrix structure, thus halving the number of additions and multiplications – time and space complexity remain $O(N^2)$

The Fast Pascal Transform (FPT)

- The **FPT** algorithm requires $\frac{1}{2}N(N-1)$ additions and no multiplications
- Proposed by Skodras in 2006, has $N-1$ steps, output of one step is input for the next
- The FPT factorization for $\mathbf{P}_4$:

$$\mathbf{P}_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 1 & -2 & 1 & 0 \\ 1 & -3 & 3 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 1 & -1 & 0 \\ 0 & 0 & 1 & -1 \end{bmatrix}$$

The Fast Pascal Transform (FPT)



Source: Skodras, A., “Efficient Computation of the Discrete Pascal Transform”, EUSIPCO2006, Florence, Italy.

The Algorithm using the Toeplitz Matrix

- Based on the factorization of Pascal's matrices proposed in **[Kailath, Sayed, 1999]**
- We modify this factorization by using the **Hadamard product** with the **vector consisting of ±1 integers** to convert Pascal's matrix into the Pascal transform matrix
- Pascal's matrix is **decomposed** into a **product of three matrices with special structure - two diagonal** and **one Toeplitz matrix**
- The **Toeplitz matrix** is **embedded into a circulant matrix** of order $2N$, which **can be diagonalized** by the **Fourier matrix**
- This allows the use of the **FFT** and leads to an $O(N \log N)$ algorithm

The Algorithm using the Toeplitz Matrix

- Factorization: $\mathbf{P} = diag(\mathbf{V}_1) \cdot \mathbf{T} \cdot diag(\mathbf{V}_2)$
- Diagonal matrices:

$$\mathbf{V}_1 = \begin{bmatrix} 0! \\ 1! \\ 2! \\ 3! \\ \vdots \\ (N-1)! \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 6 \\ \vdots \\ (N-1)! \end{bmatrix}$$

$$\mathbf{V}_2 = \begin{bmatrix} \frac{1}{0!} \\ \frac{1}{1!} \\ \frac{1}{2!} \\ \frac{1}{3!} \\ \vdots \\ \frac{1}{(N-1)!} \end{bmatrix} \circ \begin{bmatrix} 1 \\ -1 \\ 1 \\ -1 \\ \vdots \\ -1 \end{bmatrix} = \begin{bmatrix} 1 \\ -1 \\ \frac{1}{2} \\ -\frac{1}{6} \\ \vdots \\ -\frac{1}{(N-1)!} \end{bmatrix}$$

The Algorithm using the Toeplitz Matrix

- The Toeplitz matrix:

$$\mathbf{T} = \begin{bmatrix} \frac{1}{0!} & 0 & 0 & \dots & 0 \\ \frac{1}{1!} & \frac{1}{0!} & 0 & \dots & 0 \\ \frac{1}{2!} & \frac{1}{1!} & \frac{1}{0!} & \dots & 0 \\ \frac{1}{3!} & \frac{1}{2!} & \frac{1}{1!} & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \frac{1}{(N-1)!} & \frac{1}{(N-2)!} & \frac{1}{(N-3)!} & \dots & 1 \end{bmatrix}$$

The Algorithm using the Toeplitz Matrix

- Since the entries in the Toeplitz matrix have very different magnitudes, numerical stability must be adressed
- Introduction of the factor $\alpha \approx \dfrac{N-1}{e}$ [Tang, Duraiswami, Gumerov, 2004]
- Factorization becomes: $\mathbf{P}(\alpha) = diag(\mathbf{V}_1(\alpha)) \cdot \mathbf{T}(\alpha) \cdot diag(\mathbf{V}_2(\alpha))$, where

$$\mathbf{V}_1(\alpha) = \begin{bmatrix} 1 \\ \frac{1}{\alpha} \\ \frac{2}{\alpha^2} \\ \frac{6}{\alpha^3} \\ \vdots \\ \frac{(N-1)!}{\alpha^{N-1}} \end{bmatrix} \qquad \mathbf{V}_2(\alpha) = \begin{bmatrix} 1 \\ -\frac{\alpha}{1} \\ \frac{\alpha^2}{2} \\ -\frac{\alpha^3}{6} \\ \vdots \\ \frac{\alpha^{N-1}}{(N-1)!} \end{bmatrix}$$

The Algorithm using the Toeplitz Matrix

1) Generate $\mathbf{V}_1(\alpha)$, $\mathbf{V}_2(\alpha)$, and the first column of $\mathbf{T}(\alpha)$
2) Compute $\mathbf{Y}(\alpha) = \mathbf{V}_2(\alpha) \circ \mathbf{F}$
3) Embed $\mathbf{T}(\alpha)$ into a circulant matrix of order $2N$. By concatenating the first column and the first transposed row of $\mathbf{T}(\alpha)$, vector $\mathbf{X}(\alpha)$ is formed. If we pad the vector $\mathbf{Y}(\alpha)$ of length N with zeros to reach the length of $2N$, the computation can proceed as follows:
   a) Compute the FFT of $\mathrm{Y}(\alpha)$
   b) Compute the FFT of $\mathrm{X}(\alpha)$
   c) Compute $\mathrm{Z}(\alpha) = \mathrm{X}(\alpha) \circ \mathrm{Y}(\alpha)$
   d) Compute the inverse FFT of $Z(\alpha)$
4) Compute the component-wise product of $\mathbf{V}_1(\alpha)$ and the first N components of $\mathbf{Z}(\alpha)$

Computational Complexity of the Algorithms

Operation	Definition DPT	Triangular DPT	Goodman-Aburdene	FPT	Toeplitz-FFT
Arithmetic Complexity					
multiplication	N^2	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)-(N-1)$	0	$5N$ (real), $2N+6N\log 2N$ (complex)
addition	$N(N\text{-}1)$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$	$\frac{1}{2}N(N-1)$	$6N\log 2N$ (complex)
division	0	0	0	0	$2N$ (real)
Asymptotic Complexity					
	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N^2)$	$O(N\log N)$

Example: $N = 2^{20}$ FPT – 549,755,289,600 integer additions
Toeplitz-FFT – 7,340,032 real and 266,338,304 complex operations

Implementation of the Algorithms

- Computation of the DPT by the definition practically feasible only for small functions (up to $N = 2^{16}$)
- The FPT algorithm:
  - $N - 1$ sequential steps - input for each step depends on the output of the previous step
  - requires out-of-place implementation, difficult to exploit parallelism on modern processors
- The algorithm using the Toeplitz matrix and the FFT:
  - $\log N$ steps
  - Nvidia cuFFT library offers efficient FFT computations on the GPU
  - component-wise real and complex vector multiplications performed efficiently on GPUs

Experimental Platform

Component	Specification
CPU microarchitecture clock (GHz) processing power (GFLOPS) cores/threads	Intel Xeon E5-1620 Haswell 3.6 122 4/8
RAM	32GB DDR4 ECC 2133 MHz
GPU microarchitecture processing power (GFLOPS) cores memory type bandwidth (GB/s)	Nvidia Quadro K620 Kepler 768 384 2 GB DDR3 28.8 GB/s
OS	Windows 10 64-bit
IDE	Microsoft Visual Studio 2015
GPU SDK	Nvidia CUDA Toolkit 8.0
GPU Profiler	Nvidia Nsight VS Edition 5.2

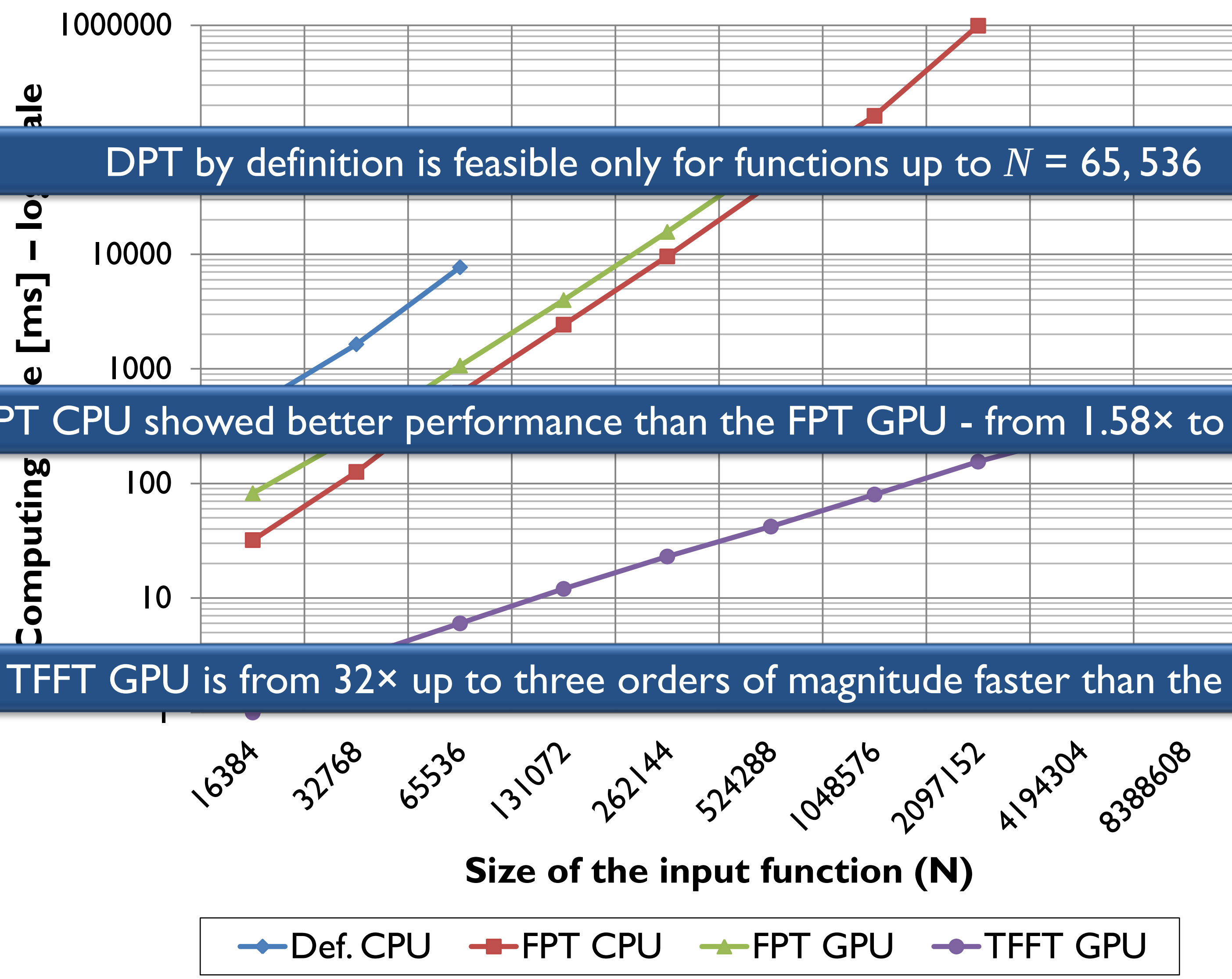
The Considered Implementations

- Randomly generated vectors as input data – elements generated in the range of pixel values in RGBA images [0, 255]
- **Implementations:**
  - **Def. DPT CPU -** the algorithm using the direct multiplication of the lower triangular DPT matrix with the function vector, implemented in C/C++ and processed on the CPU, used only as a performance baseline
  - **FPT CPU, FPT GPU -** the FPT algorithm implmentations for CPUs and GPUs. Since the CUDA GPU implementation of this algorithm we first developed resulted in low performance, we also implemented the algorithm in C/C++ for processing on the CPU
  - **TFFT GPU -** the proposed algorithm, implemented in CUDA C/C++ and processed on the GPU, uses the Nvidia cuFFT

Experimental Results - Computing Times [ms]

n	$N = 2^n$	CPU		GPU	
		Def. DPT CPU	FPT CPU	FPT GPU	TFFT GPU
14	16,384	466	32	82	1
15	32,768	1,639	126	268	3
16	65,536	7,705	618	1,067	6
17	131,072	-	2,432	3,983	12
18	262,144	-	9,589	15,689	23
19	524,288	-	41,243	65,845	42
20	1,048,576	-	161,831	-	80
21	2,097,152	-	990,056	-	155
22	4,194,304	-	> 1 h	-	270
23	8,388,608	-	> 1 h	-	497

Experimental Results - Computing Times [ms]



DPT by definition is feasible only for functions up to $N = 65,536$

FPT CPU showed better performance than the FPT GPU - from 1.58× to 2.56×

TFFT GPU is from 32× up to three orders of magnitude faster than the FPT

Conclusions

- An **efficient method** for the **computation** of the **discrete Pascal transform**, characterized by the $O(N \log N)$ asymptotical time complexity
- The algorithm is based on the **factorization** of the DPT matrix into **three matrices with special structure – two diagonal** and **one Toeplitz matrix** and **uses the FFT**
- The proposed algorithm is **very suitable for highly-parallel computation on the GPUs -** from 32× up to three orders of magnitude faster than the FPT
- The application of the proposed method can **significantly extend the set of problem instances** to which the DPT can be applied
- **Future work**: evalution of the performance of the proposed algorithm for **edge detection** in **high-resolution images**

6th International Conference Logic and Applications
LAP 2017, September 18-22, 2017, Dubrovnik, Croatia

On the Computational Complexity of the Discrete Pascal Transform

Dušan B. Gajić[1], Radomir S. Stanković[2]

[1]University of Novi Sad, Faculty of Technical Sciences, Dept. of Computing and Control
Trg Dositeja Obradovića 6, 21000 Novi Sad, Serbia

[2]University of Niš, Faculty of Electronic Engineering, Dept. of Computer Science
Aleksandra Medvedeva 14, 18000 Niš, Serbia

E-mail: [1]dusan.gajic@uns.ac.rs, [2]radomir.stankovic@gmail.com