Fast 3D Convolution algorithms (to be reviewed)

A. Kouzelis, Prof. Ioannis Pitas
Aristotle University of Thessaloniki
pitas@csd.auth.gr

Version 3.0.2

Date 30/01/2021

Artificial Intelligence & Information Analysis Lab

Fast 3D Convolution algorithms

- 3D linear and cyclic convolutions
- Fast 3D convolutions by using FFTs
- Block-based methods
- Optimal Winograd 3D convolutions

Introduction

- Convolution plays a very important role in image/video processing and analysis, machine learning etc.
- Convolutional neural networks (CNNs) are based on the convolution (they form the first layers).
- Computationally expensive, $O(N^6)$ in 3D.
- There is a need for efficient convolution algorithms.

3D Signals and Systems

- A 3D ***signal*** is a mapping of the form:

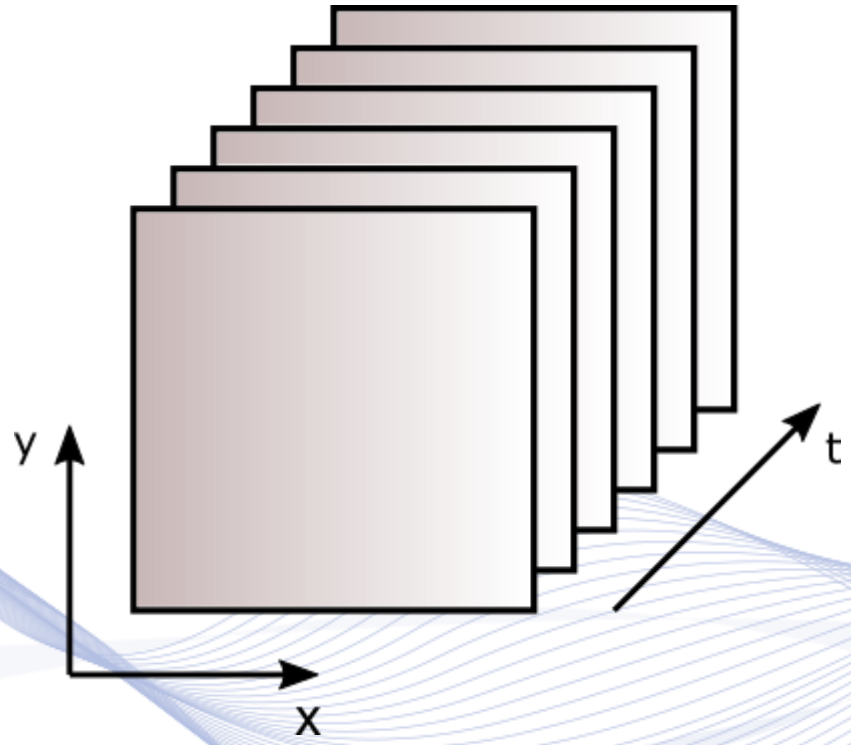
$$f_a: \mathbb{R}^3 \rightarrow \mathbb{R}.$$

- The discrete version is:

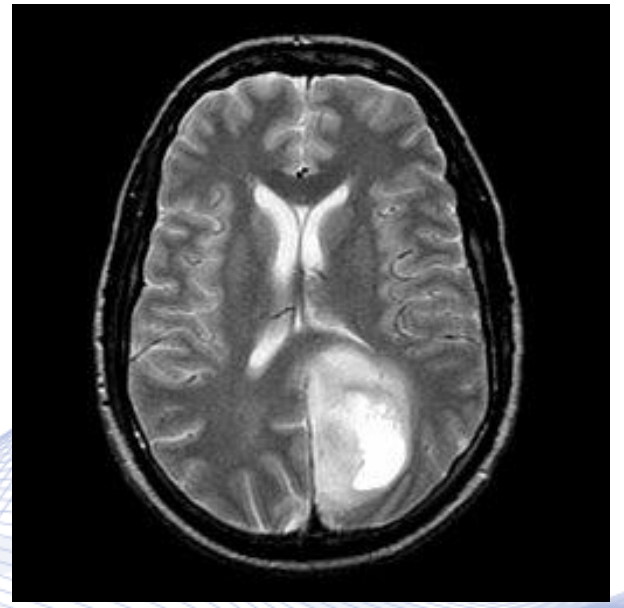
$$f: \mathbb{Z}^3 \rightarrow \mathbb{R}.$$

- For example:
  - Digital video signal: $f(n_1, n_2, n_3) = f_a(n_1\Delta x, n_2\Delta y, n_3\Delta t)$.
  - 3D volumetric image: $f(n_1, n_2, n_3) = f_a(n_1\Delta x, n_2\Delta y, n_3\Delta z)$.
  - $\Delta x, \Delta y, \Delta z$ are spatial sampling intervals and Δt is the temporal sampling interval.

3D Signals and Systems



Spatiotemporal video signal

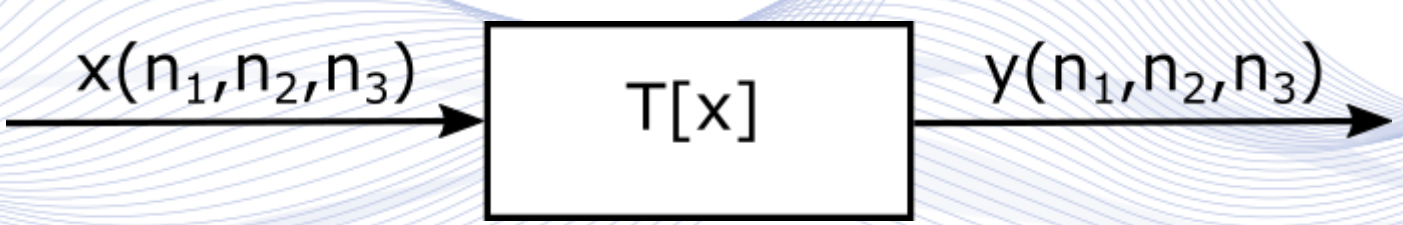


2D slice of a 3D MRI image [WIK-MRI]

3D Signals and Systems

A 3D ***discrete-time system*** is defined as a transformation T that maps an input signal $x(n_1, n_2, n_3)$ into an output signal $y(n_1, n_2, n_3)$:

$$y(n_1, n_2, n_3) = T[x(n_1, n_2, n_3)].$$



3D Signals and Systems

A 3D system is called ***Linear Shift Invariant*** (LSI) if it is:

- ***Linear***:

$$T[a\, x_1(n_1, n_2, n_3) + b\, x_2(n_1, n_2, n_3)]$$
$$= a\, T[x_1(n_1, n_2, n_3)] + b\, T[x_2(n_1, n_2, n_3)].$$

- ***Shift-invariant***:

$$y(n_1 - k_1, n_2 - k_2, n_3 - k_3) = T[x(n_1 - k_1, n_2 - k_2, n_3 - k_3)].$$

3D Signals and Systems

Any 3D discrete signal $x(n_1, n_2, n_3)$ can be decomposed into:

$$x(n_1, n_2, n_3)$$

$$= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \sum_{k_3=-\infty}^{\infty} x(k_1, k_2, k_3)\delta(n_1 - k_1, n_2 - k_2, n_3 - k_3),$$

where $\delta(n_1, n_2, n_3) = \begin{cases} 1, & \text{if } n_1 = n_2 = n_3 = 0 \\ 0, & \text{otherwise} \end{cases}$.

3D Signals and Systems

The system output y can be decomposed as:

$$y(n_1, n_2, n_3) = T[x(n_1, n_2, n_3)]$$

$$= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \sum_{k_3=-\infty}^{\infty} x(k_1, k_2, k_3) h(n_1 - k_1, n_2 - k_2, n_3 - k_3),$$

where $h(n_1, n_2, n_3) \equiv T[\delta(n_1, n_2, n_3)]$ is the system ***impulse response*** which completely characterizes the system.

3D Linear Convolution

For a 3D LSI system with impulse response h, the input x and output y are related by the ***3D linear convolution***:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \ast\ast\ast h(n_1, n_2, n_3)$$

$$= \sum_{k_1=-\infty}^{\infty} \sum_{k_2=-\infty}^{\infty} \sum_{k_3=-\infty}^{\infty} x(k_1, k_2, k_3) h(n_1 - k_1, n_2 - k_2, n_3 - k_3).$$

3D linear convolution is commutative: $x \ast\ast\ast h = h \ast\ast\ast x$.

3D Linear Convolution

If the system's impulse response h is of finite size $N_{h_1} \times N_{h_2} \times N_{h_3}$, the system is called ***Finite Impulse Response*** (FIR) system and is described by:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \ *** \ h(n_1, n_2, n_3)$$

$$= \sum_{k_1=0}^{N_{h_1}-1} \sum_{k_2=0}^{N_{h_2}-1} \sum_{k_3=0}^{N_{h_3}-1} h(k_1, k_2, k_3) x(n_1 - k_1, n_2 - k_2, n_3 - k_3).$$

3D Linear Convolution

If the input signal $x(n_1, n_2, n_3)$ is also finite $(N_{x_1} \times N_{x_2} \times N_{x_3})$, the resulting output signal $y = x \ast\ast\ast h$ has size:

$$(N_{x_1} + N_{h_1} - 1) \times (N_{x_2} + N_{h_2} - 1) \times (N_{x_3} + N_{h_3} - 1).$$

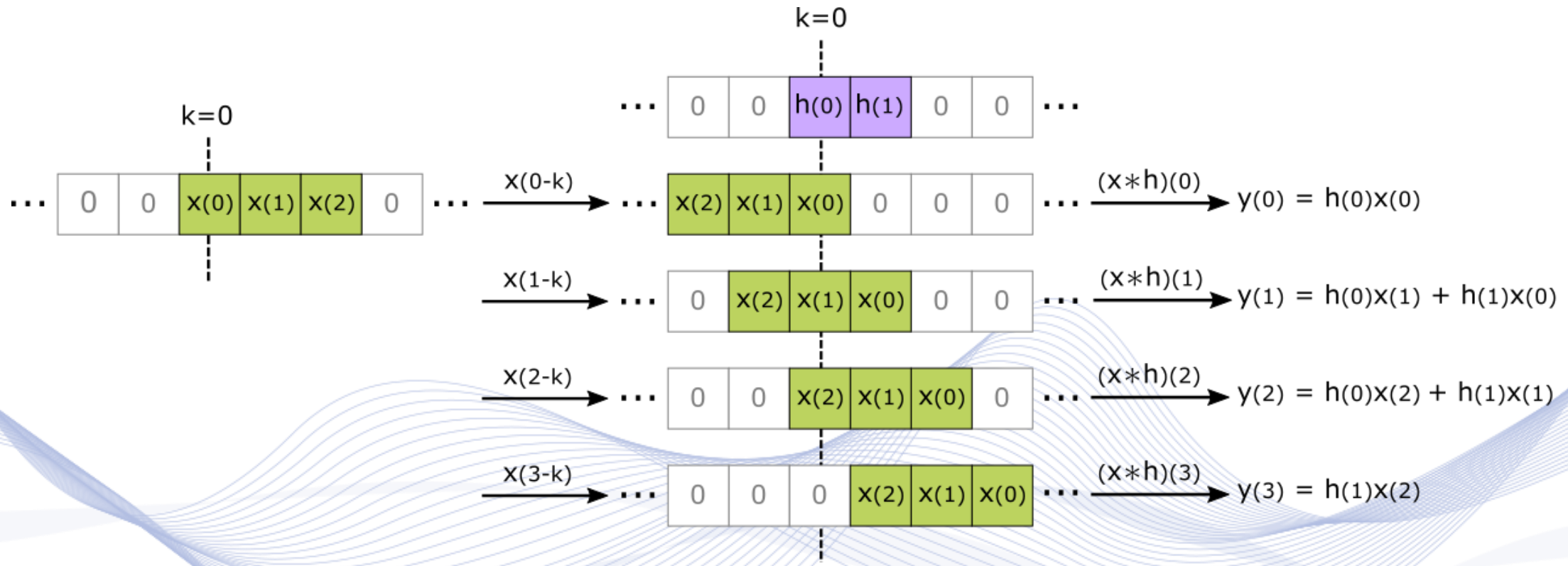
3D Linear Convolution

The 3D convolution has the following computational interpretation:

- The signal $x(k_1, k_2, k_3)$ is reflected about the axes k_1, k_2 and k_3 and then translated by n_1, n_2 and n_3 in each axis respectively, to form the $x(n_1 - k_1, n_2 - k_2, n_3 - k_3)$.

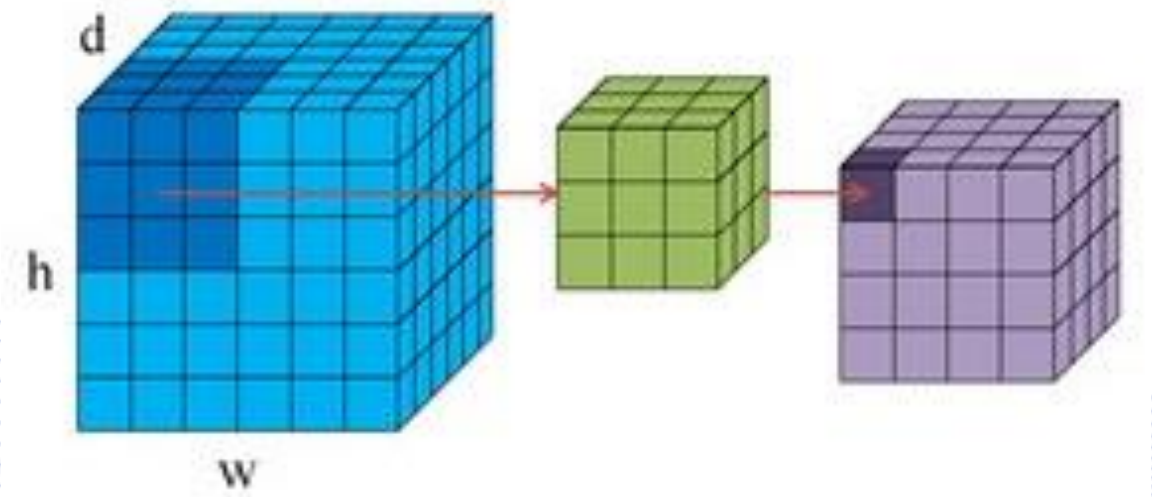
- For each n_1, n_2 and n_3 the products $h(k_1, k_2, k_3)x(n_1 - k_1, n_2 - k_2, n_3 - k_3)$ are summed over all k_1, k_2 and k_3.

3D Linear Convolution



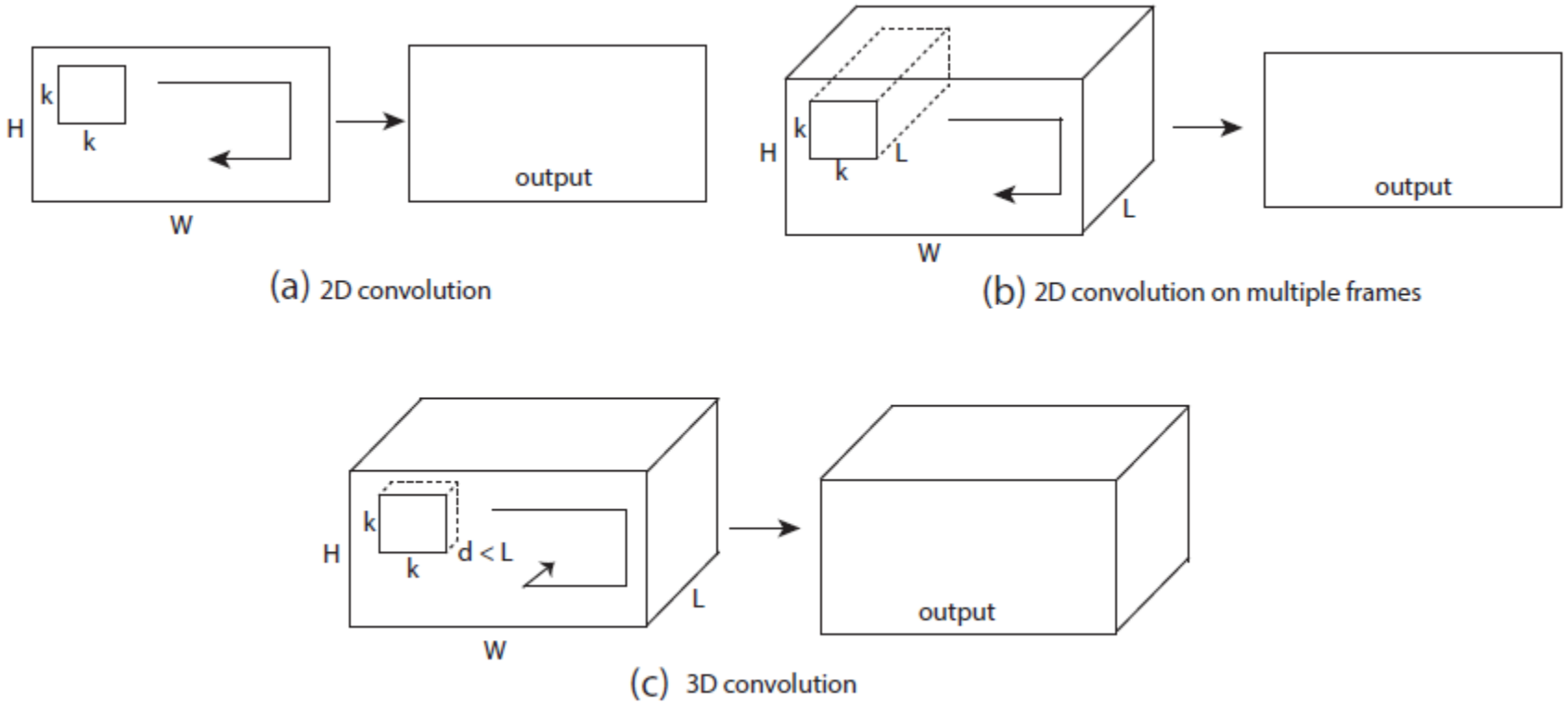
Example of 1D linear convolution: The signals $x(n)$ and $h(n)$ of finite size $N_x = 3$ and $N_h = 2$ respectively, and the output signal $y(n)$ is of size $N_x + N_h - 1 = 4$.

3D Linear Convolution



An illustration of 3D convolution with a kernel of size $3 \times 3 \times 3$ (from [DON2020]).

3D Linear Convolution



2D convolution vs 3D convolution (from [TRA2015]).

3D Cyclic Convolution

The ***3D cyclic convolution*** is defined as:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \circledast\circledast\circledast h(n_1, n_2, n_3)$$

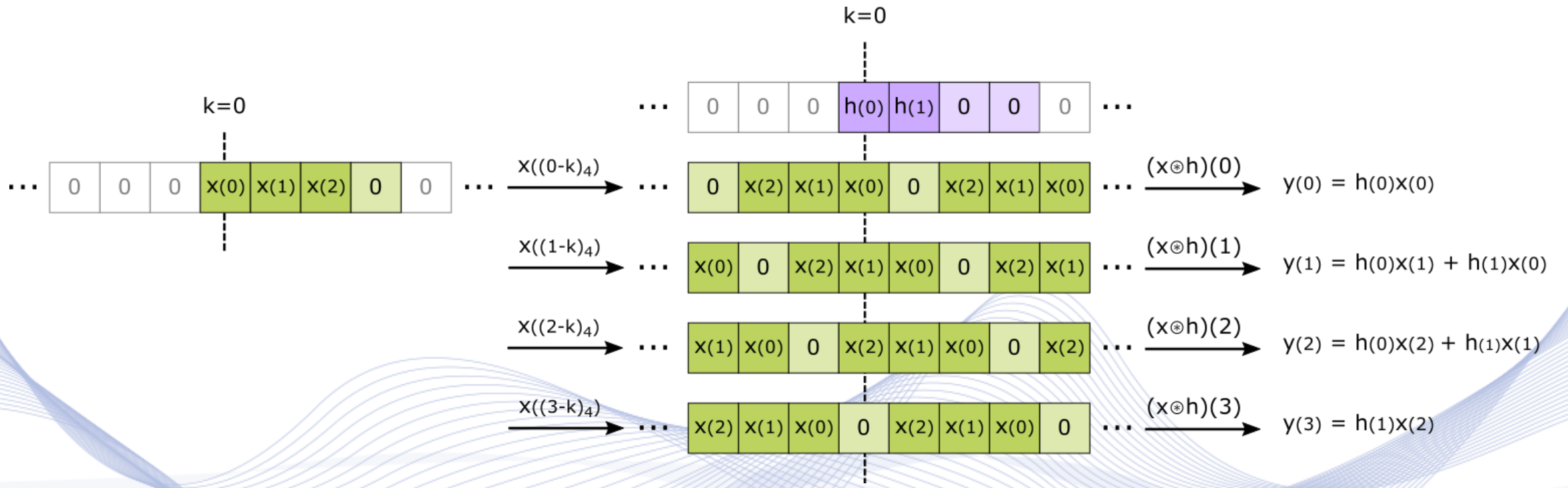
$$= \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \sum_{k_3=0}^{N_3-1} x(k_1, k_2, k_3) h\left((n_1 - k_1)_{N_1}, (n_2 - k_2)_{N_2}, (n_3 - k_3)_{N_3}\right),$$

where $(n)_N$ denotes $n \bmod N$ and is the ***cyclic shift***. We use the symbol $\circledast$ to distinguish it from the linear convolution.

3D Cyclic Convolution

- 3D linear convolution can be embedded in 3D cyclic convolution by zero-padding the $x(n_1, n_2, n_3)$ and $h(n_1, n_2, n_3)$ in each dimension.
- Performing cyclic convolution on these padded signals is equivalent to performing linear convolution on the original signals.
- Cyclic convolutions are useful because they can be computed using DFT (via FFT algorithms) and other fast algorithms such as Winograd convolution algorithms.

3D Cyclic Convolution



Example of 1D cyclic convolution which is equivalent to linear convolution. The original signals $x(n)$ and $h(n)$ are of size $N_x = 3$ and $N_h = 2$ respectively. By zero-padding them to the same size $N_x + N_h - 1 = 4$, the resulting output signal $y(n)$ (of size 4) is the same as $y(n)$ obtained from linear convolution of the original signals.

3D Z-transform

- The ***3D Z-transform*** of a $N_1 \times N_2 \times N_3$ signal x is defined as:

$$X(z_1, z_2, z_3) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1} x(n_1, n_2, n_3) z_1^{-n_1} z_2^{-n_2} z_3^{-n_3},$$

  where z_1, z_2, z_3 are complex variables.
- It can be considered as a polynomial of three variables z_1, z_2, z_3, by multiplying it by the monomial $z_1^{N_1-1} z_2^{N_2-1} z_3^{N_3-1}$.

3D Z-transform

- An important property is that the 3D linear convolution is equivalent to the polynomial products in the z-domain:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \ *** \ h(n_1, n_2, n_3)$$
$$\leftrightarrow Y(z_1, z_2, z_3) = X(z_1, z_2, z_3)H(z_1, z_2, z_3).$$

- Similarly for the 3D cyclic convolution:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \circledast\circledast\circledast h(n_1, n_2, n_3)$$
$$\leftrightarrow Y(z_1, z_2, z_3) = X(z_1, z_2, z_3)H(z_1, z_2, z_3) \bmod (z_1^{N_1} - 1), (z_2^{N_2} - 1), (z_3^{N_3} - 1).$$

3D Discrete Fourier Transform

The ***3D Discrete Fourier Transform (DFT)*** of a 3D $N_1 \times N_2 \times N_3$ signal x is defined as:

$$X(k_1, k_2, k_3) = \sum_{n_1=0}^{N_1-1} \sum_{n_2=0}^{N_2-1} \sum_{n_3=0}^{N_3-1} x(n_1, n_2, n_3) W_{N_1}^{n_1 k_1} W_{N_2}^{n_2 k_2} W_{N_3}^{n_3 k_3},$$

where $W_{N_i} \equiv e^{-j2\pi/N_i}$, $i = 1,2,3$, are N_i-th ***primitive roots of unity***.

3D Discrete Fourier Transform

The ***inverse 3D DFT (IDFT)*** is given by:

$$x(n_1, n_2, n_3)$$

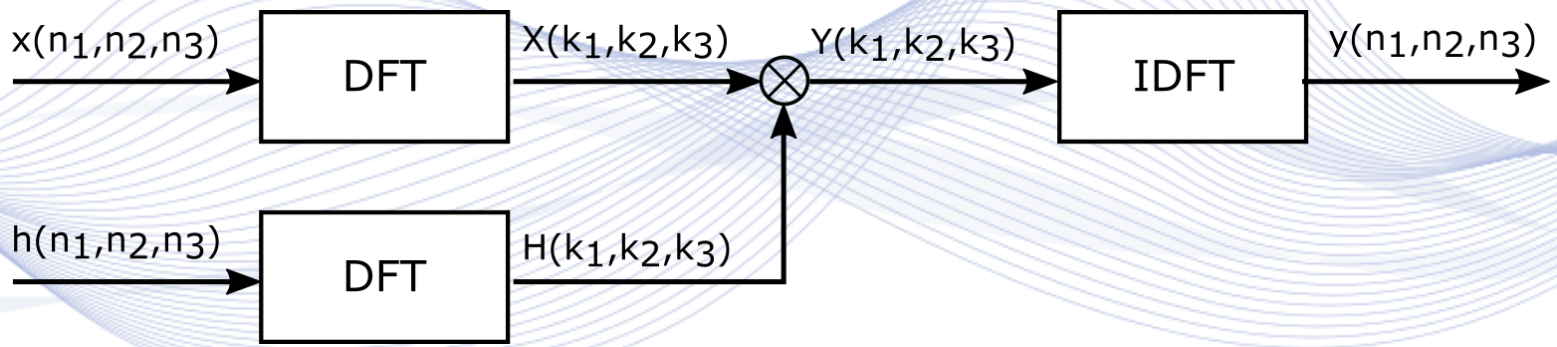
$$= \frac{1}{N_1 N_2 N_3} \sum_{k_1=0}^{N_1-1} \sum_{k_2=0}^{N_2-1} \sum_{k_3=0}^{N_3-1} X(k_1, k_2, k_3) W_{N_1}^{-n_1 k_1} W_{N_2}^{-n_2 k_2} W_{N_3}^{-n_3 k_3}.$$

3D Discrete Fourier Transform

- The ***convolution theorem*** states that the cyclic convolution in $\mathbb{Z}^3$ is equivalent to the multiplication in the DFT domain:

$$y(n_1, n_2, n_3) = x(n_1, n_2, n_3) \circledast\circledast\circledast h(n_1, n_2, n_3)$$
$$\leftrightarrow Y(k_1, k_2, k_3) = X(k_1, k_2, k_3)\, H(k_1, k_2, k_3).$$

- Thus, the 3D cyclic convolution can be computed by DFT:



3D Fast Fourier Transform

The 3D DFT can be rewritten as ([DUD1984]):

$$X(k_1, k_2, k_3) = \sum_{n_1=0}^{N_1-1} F(n_1, k_2, k_3) W_{N_1}^{n_1 k_1},$$

where:

$$F(n_1, k_2, k_3) = \sum_{n_2=0}^{N_2-1} G(n_1, n_2, k_3) W_{N_2}^{n_2 k_2},$$

$$G(n_1, n_2, k_3) = \sum_{n_3=0}^{N_3-1} x(n_1, n_2, n_3) W_{N_3}^{n_3 k_3}.$$

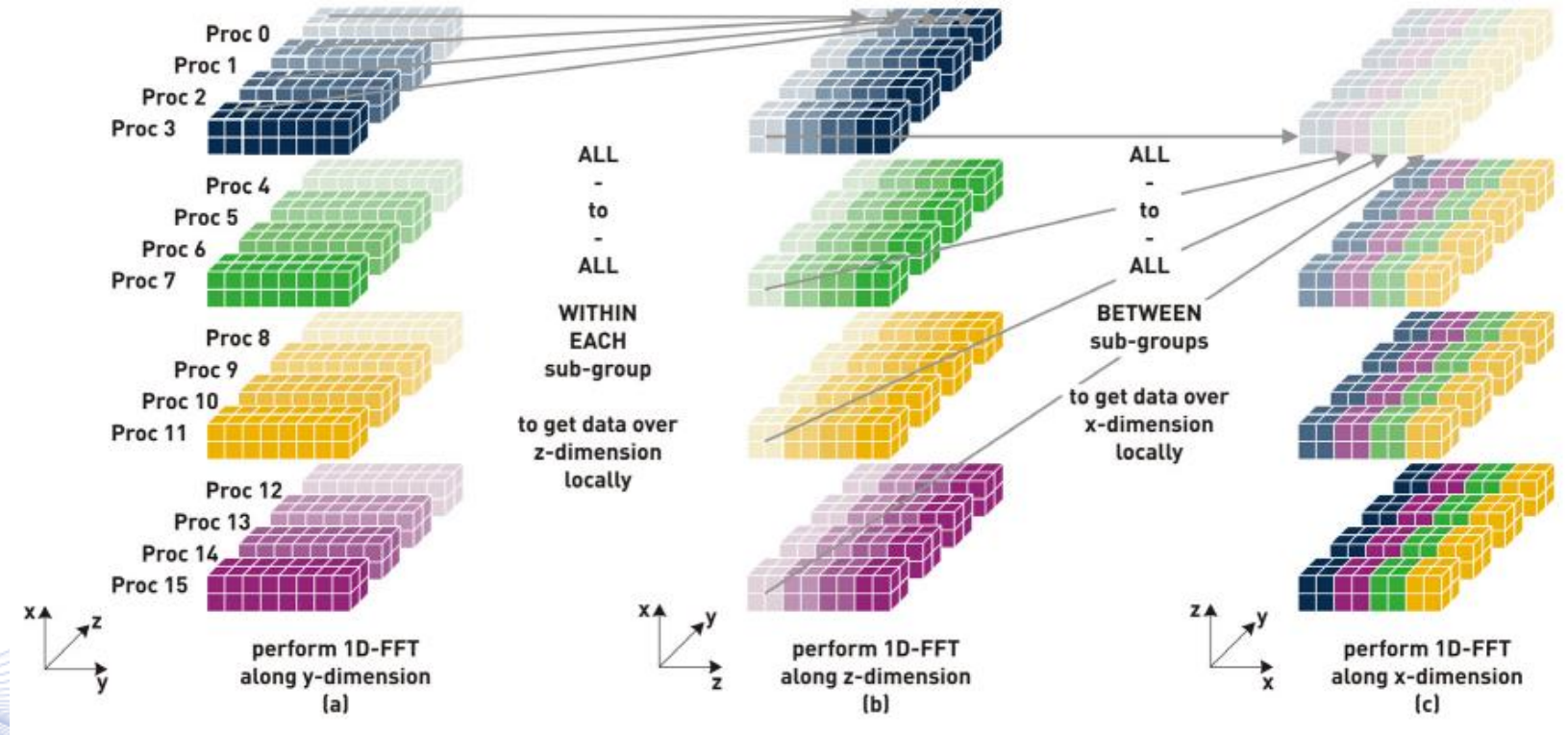
3D Fast Fourier Transform

In this way, the 3D DFT becomes decomposed into:

- $N_1 N_2$ 1D DFTs G of length N_3, along the axis n_3,
- $N_1 N_3$ 1D DFTs F of length N_2, along the axis n_2,
- $N_2 N_3$ 1D DFTs x of length N_1, along the axis n_1,

where each 1D DFT can be computed using 1D FFT algorithm.

3D Fast Fourier Transform



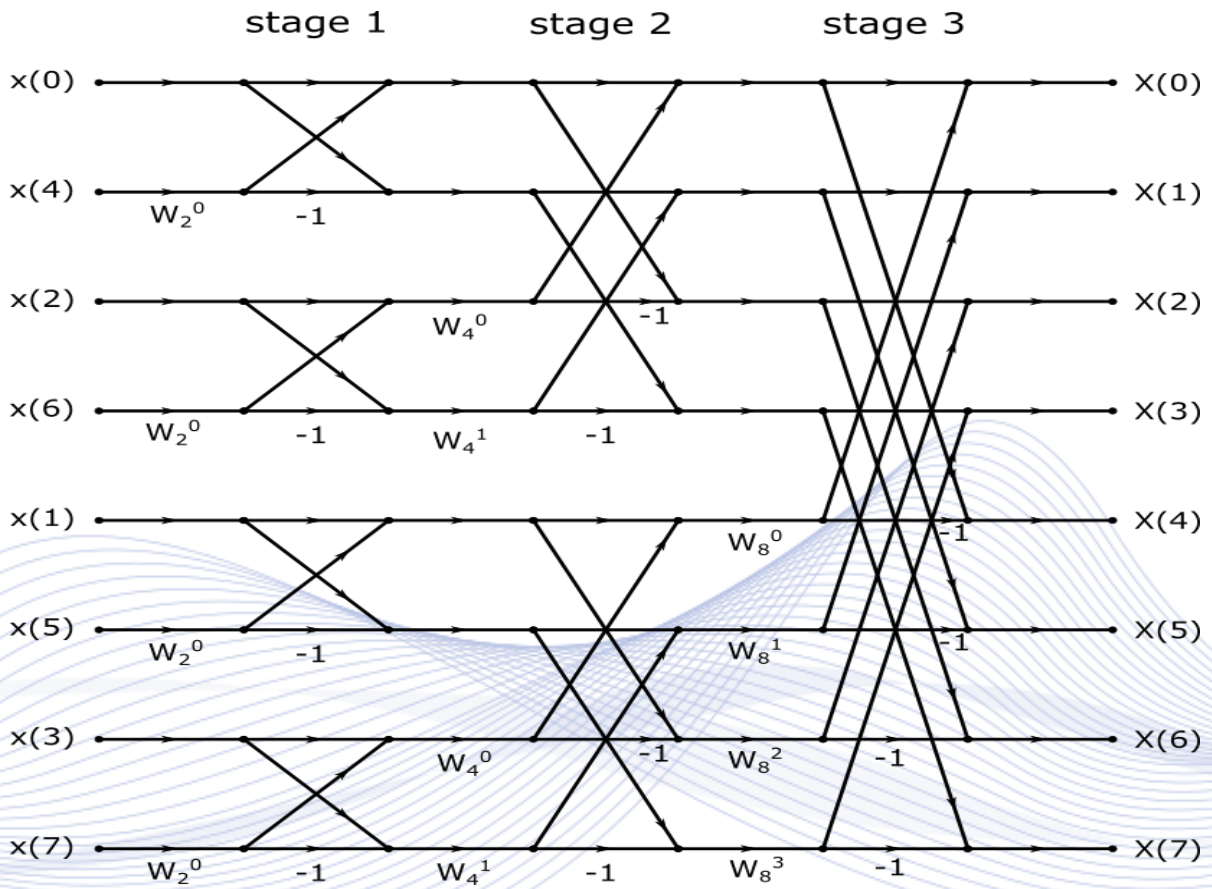
Decomposition of 3D FFT into 1D FFTs (from [HEI2005]).

3D Fast Fourier Transform

- There are many variants of 1D FFT algorithms. The best known is the Cooley-Tuckey ***radix-2 decimation in time*** (DIT) FFT algorithm.

- It uses the “divide and conquer” approach by recursively breaking down the 1D DFT of any composite size $N = N_1 N_2$ into N_1 smaller DFTs of sizes N_2.

3D Fast Fourier Transform



Data flow diagram of 1D radix-2 FFT algorithm for *N*=8.

3D Fast Fourier Transform

The radix-2 FFT algorithm rearranges the 1D DFT into a sum over even and odd indices:

$$X(k) = \sum_{n=0}^{N/2-1} x(2n) W_{N/2}^{nk} + W_N^k \sum_{n=0}^{N/2-1} x(2n+1) W_{N/2}^{nk}$$

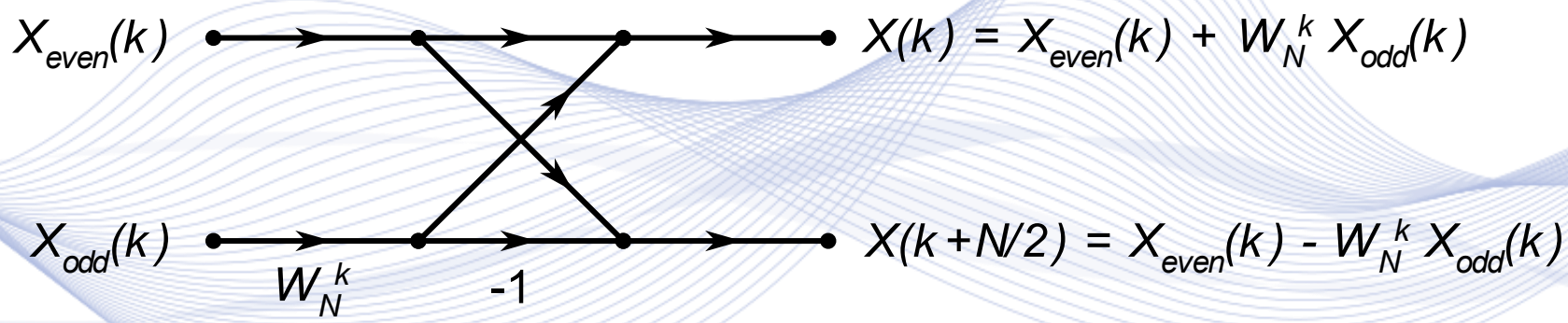
$$= X_{even}(k) + W_N^k X_{odd}(k).$$

By using the properties $W_N^{k+N} = W_N^k$ and $W_N^{k+N/2} = -W_N^k$, it can be shown that:

$$X(k + N/2) = X_{even}(k) - W_N^k X_{odd}(k).$$

3D Fast Fourier Transform

Therefore, we can express a N-point DFT in terms of two $N/2$-point DFTs $X_{even}(k)$ and $X_{odd}(k)$, $0 \leq k \leq \frac{N}{2} - 1$, which are then combined by ***butterfly operation*** (only one multiplication and two additions):



3D Fast Fourier Transform

The number of additions and multiplications required for computing the 3D FFT by using 1D radix-2 FFTs is [PIT2000]:

$$A = N_1 N_2 N_3 \log_2(N_1 N_2 N_3),$$

$$M = \frac{N_1 N_2 N_3}{2} \log_2(N_1 N_2 N_3).$$

This is much better as compared to $(N_1 N_2 N_3)^2$ multiplications required for the direct computation of 3D DFT.

Block convolutions

- Computation of convolution by DFT methods (using FFT algorithms) for signals of large sizes can be very memory consuming.
- To overcome this problem, block methods can be used.
- Limiting the size of blocks limits the amount of storage required while maintaining the efficiency of the procedure [DUD1984].
- There are two block-based methods: ***overlap-add*** and ***overlap-save***.

Overlap-add method

- The overlap-add method is based on the distributive property of the convolution.
- A 3D signal x of size $N_{x_1} \times N_{x_2} \times N_{x_3}$ can be partitioned into blocks of size $B_1 \times B_2 \times B_3$:

$$x_{ijk}(n_1, n_2, n_3) = \begin{cases} x(n_1, n_2, n_3), & \begin{array}{c} iB_1 \leq n_1 < (i+1)B_1, \\ jB_2 \leq n_2 < (j+1)B_2, \\ kB_3 \leq n_3 < (k+1)B_3, \end{array} \\ 0, & \text{otherwise.} \end{cases}$$

Overlap-add method

- The signal $x(n_1, n_2, n_3)$ can be re-constructed from these blocks:

$$x(n_1, n_2, n_3) = \sum_i \sum_j \sum_k x_{ijk}(n_1, n_2, n_3).$$

- The output $y(n_1, n_2, n_3)$ is the sum of convolutions of these blocks with the impulse response $h(n_1, n_2, n_3)$:

$$y(n_1, n_2, n_3) = \sum_i \sum_j \sum_k y_{ijk}(n_1, n_2, n_3),$$

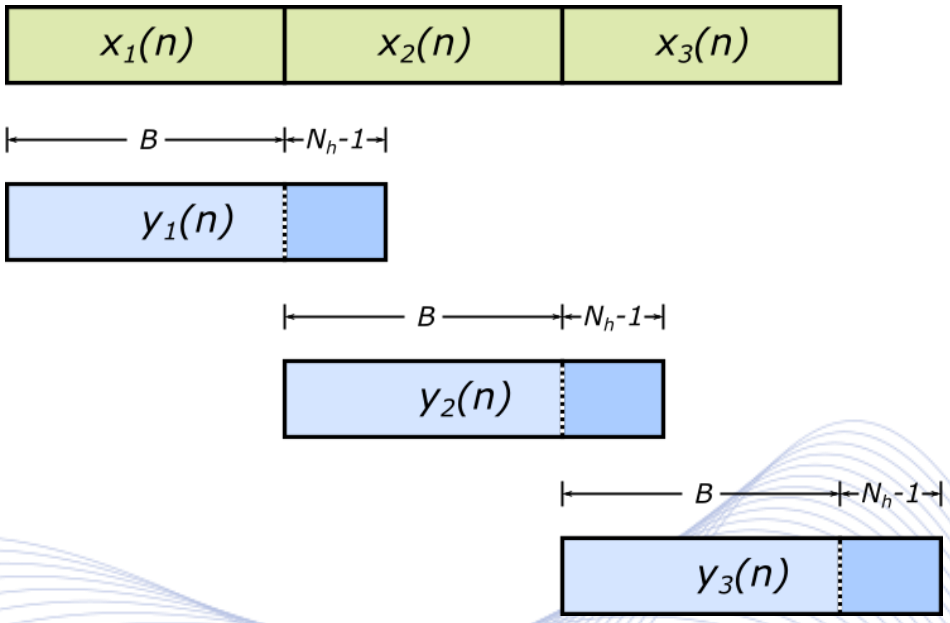
where $y_{ijk} = x_{ijk} *** h$ is the block output.

Overlap-add method

- The size of each output block y_{ijk} is $\left(B_1 + N_{h_1} - 1\right) \times \left(B_2 + N_{h_2} - 1\right) \times \left(B_3 + N_{h_3} - 1\right)$ which is greater than the size of the corresponding input block x_{ijk}.

- Therefore, each output block overlaps with its adjacent output blocks by some amount determined by the size of h.

Overlap-add method



Overlap-add method for convolution in 1D. The input signal $x(n)$ is partitioned into three blocks $x_1(n)$, $x_2(n)$ and $x_3(n)$, each of length B. The impulse response $h(n)$ is of length N_h and the output blocks $y_i(n) = (x_i * h)(n)$, $i = 1, 2, 3$, are of length $B + N_h - 1$ each. There are $N_h - 1$ overlapping points between output blocks $y_i(n)$ and $y_{i+1}(n)$. The output signal $y(n)$ is formed by adding all the overlapping output blocks $y_i(n)$.

Overlap-add method

- The convolutions of each x_{ijk} with h can be efficiently computed by FFT of size at least $\left(B_1 + N_{h_1} - 1\right) \times \left(B_2 + N_{h_2} - 1\right) \times \left(B_3 + N_{h_3} - 1\right)$ [PIT2000].

- Limiting the block size reduces the required memory but increases the number of computations.

Overlap-save method

- The overlap-save method is an alternative block method.
- The 3D output is partitioned into $B_1 \times B_2 \times B_3$ non-overlapping blocks:

$$y(n_1, n_2, n_3) = \sum_i \sum_j \sum_k y_{ijk}(n_1, n_2, n_3).$$

- The corresponding 3D input section $x_{ijk}(n_1, n_2, n_3)$ of size $B_1 \times B_2 \times B_3$ is extended to $x'_{ijk}(n_1, n_2, n_3)$ of size $(B_1 + N_{h_1} - 1) \times (B_2 + N_{h_2} - 1) \times (B_3 + N_{h_3} - 1)$.

Overlap-save method

- The 3D cyclic convolution $y'_{ijk} = x'_{ijk} \circledast\circledast\circledast h$ can be efficiently computed by FFT of size $(B_1 + N_{h_1} - 1) \times (B_2 + N_{h_2} - 1) \times (B_3 + N_{h_3} - 1)$.
- Each of the resulting blocks y'_{ijk} will contain a sub-block of size $B_1 \times B_2 \times B_3$ which is identical to the desired linear convolution $y_{ijk} = x_{ijk} *** h$ (which are added to form y).
- In both block methods the choice of block size affects the amount of storage needed and the number of computations.

Winograd convolution algorithm

- We saw that the 3D cyclic convolution can be efficiently computed by 1D FFTs.
- When the length of the convolution kernel is small, the best convolution algorithms, as measured by the number of required multiplications, are the Winograd convolution algorithms [BLA2010].
- The Winograd convolution algorithms are based on the ***Chinese Remainder Theorem (CRT)*** for polynomials.

Chinese Remainder Theorem

Let $M(z) = \prod_{i=1}^{k} m_i(z)$ be a product of k pair-wise coprime polynomials and let $M_i(z) = \frac{M(z)}{m_i(z)}$ for each i.

Since $M_i(z)$ and $m_i(z)$ are coprime for all i, the Bezout's identity holds:

$$M_i(z)N_i(z) + m_i(z)n_i(z) = 1, \qquad \forall i,$$

for some polynomials $N_i(z)$ and $n_i(z)$, which are not unique.

Chinese Remainder Theorem

Then, the system of congruences $c_i(z) = c(z) \bmod m_i(z),\ i = 1, \ldots, k$, has a unique solution modulo M:

$$c(z) = \sum_{i=1}^{k} c_i(z) M_i(z) N_i(z) \bmod M(z).$$

1D Winograd convolution

For simplicity, we first present the 1D Winograd convolution algorithm and later extend it to 3D.

The 1D cyclic convolution of length N can be expressed in terms of polynomials in z-domain as:

$$Y(z) = X(z)H(z) \bmod z^N - 1.$$

1D Winograd convolution

The polynomial $P(z) = z^N - 1$ can be factorized into ν irreducible ***cyclotomic polynomials*** $p_i(z) = \Phi_{d_i}(z)$ over the field of rational numbers Q:

$$P(z) = \prod_{i=1}^{\nu} p_i(z),$$

where ν is the number of all the divisors d_i of N.

1D Winograd convolution

- The cyclotomic polynomials are given by the relation:

$$\Phi_n(z) = \prod_{\gcd(k,n)=1} (z - w_n^k),$$

where $w_n = e^{j2\pi/n}$ is the primitive n-th root of unity.

- It follows that $\deg(\Phi_n) = \phi(n)$, where $\phi(n)$ is the Euler's totient function.
- The cyclotomic polynomials have integer coefficients. For example: $\Phi_1(z) = z - 1, \Phi_2(z) = z + 1, \Phi_3(z) = z^2 + z + 1$.

1D Winograd convolution

The Winograd convolution algorithm is based on the reduction of the polynomial product $Y(z) = X(z)H(z) \bmod P(z)$ into the products of smaller-degree polynomials modulo $p_i(z)$:

$$Y_i(z) = X_i(z)H_i(z) \bmod p_i(z),$$

where

$$X_i(z) = X(z) \bmod p_i(z),$$
$$H_i(z) = H(z) \bmod p_i(z).$$

1D Winograd convolution

- The polynomial $Y(z)$ can be reconstructed by using the CRT for polynomials:

$$Y(z) = \sum_{i=1}^{\nu} Y_i(z) R_i(z) \bmod p_i(z),$$

where $R_i(z) = \delta_{ij} \bmod p_j(z)$.

- In practice the polynomials $R_i(z)$ can be computed using the following relation [GAR1987]:

$$R_i(z) = \frac{P(z)}{p_i(z)} \left[\left(\frac{z}{N} \frac{\mathrm{d}p_i}{\mathrm{d}z} \right) \bmod p_i(z) \right].$$

1D Winograd convolution

- The Winograd convolution algorithm can be expressed compactly in the following matrix notation (bilinear form):

$$\mathbf{y} = \mathbf{C}(\mathbf{Ax} \otimes \mathbf{Bh}),$$

where $\otimes$ denotes element-wise product.

- Matrices $\mathbf{A}$ and $\mathbf{B}$ typically have elements $-1, 0, 1$. Therefore products $\mathbf{Ax}$ and $\mathbf{Bh}$ represent additions instead of multiplications.

1D Winograd convolution

- We can use the equivalent form:

$$\mathbf{y} = \mathbf{R}\mathbf{B}^{\mathrm{T}}\left(\mathbf{A}\mathbf{x} \otimes \mathbf{C}^{\mathrm{T}}\mathbf{R}\mathbf{h}\right),$$

  where $\mathbf{R}$ is a $N \times N$ permutation matrix. The $\mathbf{R}\mathbf{B}^{\mathrm{T}}$ and $\mathbf{C}^{\mathrm{T}}\mathbf{R}\mathbf{h}$ can be both precomputed.

- Winograd convolution algorithms are optimal, having the minimal number of multiplications $2N - \nu$ [WIN1980].

- GEneral Matrix Multiplication (GEMM) BLAS or CUBLAS routines can be used for fast matrix-vector multiplications.

3D Winograd convolution

- The Winograd convolution can be extended to three dimensions [PIT1987].
- The 3D cyclic convolution can be expressed as:

$$Y(z_1, z_2, z_3) = X(z_1, z_2, z_3)\, H(z_1, z_2, z_3) \bmod P_1(z_1), P_2(z_2), P_3(z_3),$$

where $P_i(z_i) = \left(z_i^{N_i} - 1\right),\ i = 1, 2, 3.$

- Each $P_i(z_i)$ can be factorized into ν_i cyclotomic polynomials:

$$P_i(z_i) = \prod_{j_i=1}^{\nu_i} p_{ij_i}(z_i), \qquad \deg\{p_{ij_i}\} = N_{ij_i},\ i = 1,2,3.$$

3D Winograd convolution

Therefore, the $Y(z_1, z_2, z_3)$ can be reduced to the products of smaller-degree polynomials:

$$Y_{j_1 j_2 j_3}(z_1, z_2, z_3) = X_{j_1 j_2 j_3}(z_1, z_2, z_3)\, H_{j_1 j_2 j_3}(z_1, z_2, z_3) \bmod p_{1 j_1}(z_1), p_{2 j_2}(z_2), p_{3 j_3}(z_3),$$

where:

$$X_{j_1 j_2 j_3}(z_1, z_2, z_3) = X(z_1, z_2, z_3) \bmod p_{1 j_1}(z_1), p_{2 j_2}(z_2), p_{3 j_3}(z_3)$$

$$H_{j_1 j_2 j_3}(z_1, z_2, z_3) = H(z_1, z_2, z_3) \bmod p_{1 j_1}(z_1), p_{2 j_2}(z_2), p_{3 j_3}(z_3)$$

for $1 \le j_i \le \nu_i,\ i = 1, 2, 3.$

3D Winograd convolution

Using the CRT, $Y(z_1, z_2, z_3)$ can be reconstructed as follows:

$$Y(z_1, z_2, z_3) = Y_{j_1 j_2 j_3}(z_1, z_2, z_3)\, R_{1 j_1}(z_1) R_{2 j_2}(z_2) R_{3 j_3}(z_3)$$

$$\bmod p_{1 j_1}(z_1), p_{2 j_2}(z_2), p_{3 j_3}(z_3),$$

where

$$R_{i j_i}(z_i) = \delta_{j_i k_i} \bmod p_{i k_i}(z_i), \qquad 1 \le k_i \le \nu_i.$$

3D Winograd convolution

- The number of multiplications of this 3D algorithm is:

$$(2N_1 - \nu_1)(2N_2 - \nu_2)(2N_3 - \nu_3),$$

i.e., the computational complexity is of order $O(N^3)$.

- However, this is not the minimal computational complexity, because there can exist further factorizations such as each $p_{1j_1}(z_1)$ over $Q[z_2]/p_{2j_2}(z_2)$ or $Q[z_3]/p_{3j_3}(z_3)$ etc.

3D Winograd convolution

It can be shown that the optimal algorithm for 3D cyclic convolution exists and requires the following minimum number of multiplications [PIT1987]:

$$M = \sum_{j_1=1}^{\nu_1} \sum_{j_2=1}^{\nu_2} \sum_{j_3=1}^{\nu_3} M_{j_1 j_2 j_3},$$

where

$$\begin{aligned} M_{j_1 j_2 j_3} = \min \{ & (2N_{1j_1} - 1)(2N_{2j_2} - k_{2j_2})(2N_{3j_3} - k_{3j_3}), \\ & (2N_{2j_2} - 1)(2N_{1j_1} - k_{1j_1})(2N_{3j_3} - k_{3j_3}), \\ & (2N_{3j_3} - 1)(2N_{1j_1} - k_{1j_1})(2N_{2j_2} - k_{2j_2}) \}. \end{aligned}$$

3D Winograd convolution

- Such optimal algorithms can be expressed in the matrix form that we saw for 1D Winograd convolution:

$$\mathbf{y} = \mathbf{R}\mathbf{B}^{\mathrm{T}}\left(\mathbf{A}\mathbf{x} \otimes \mathbf{C}^{\mathrm{T}}\mathbf{R}\mathbf{h}\right),$$

  which then can be computed by using linear algebra libraries such as BLAS and cuBLAS.

- However, finding the matrices A, B, C can be a tedious task and has to be done by hand for a desired convolution size.

Bibliography

[PIT2017] I. Pitas, “Digital video processing and analysis” , China Machine Press, 2017 (in Chinese).
[PIT2013] I. Pitas, “Digital Video and Television” , Createspace/Amazon, 2013.
[PIT2021] I. Pitas, “Computer vision”, Createspace/Amazon, in press.
[NIK2000] N. Nikolaidis and I. Pitas, “3D Image Processing Algorithms”, J. Wiley, 2000.
[PIT2000] I. Pitas, “Digital Image Processing Algorithms and Applications”, J. Wiley, 2000.
[PIT1987] I. Pitas, M. Strintzis, “Multidimensional cyclic convolution algorithms with minimal multiplicative complexity”, IEEE transactions on acoustics, speech, and signal processing, vol. 35, no. 3, pp. 384-390, 1987.

Bibliography

[WIN1980] S. Winograd, **“Arithmetic complexity of computations”**, vol. 33. Siam, 1980.

[WIN1980] S. Winograd, **“On Multiplication of Polynomials Modulo a Polynomial”**, SIAM J. Comput., 9(2), 225–229, 1980.

[LAV2016] A. Lavin, S. Gray, **“Fast algorithms for convolutional neural networks”**, Proceedings CVPR, pp. 4013-4021, 2016.

[NUS1981] H. J. Nussbaumer, **“Fast Fourier transform and convolution algorithms”**, Springer Series in Information Sciences 2, 1981.

Bibliography

[BLA2010] R. E. Blahut, **“Fast Algorithms for Signal Processing”**. Cambridge University Press, 2010.

[AGA1977] R. Agarwal and J. Cooley, **"New algorithms for digital convolution,"** in IEEE Transactions on Acoustics, Speech, and Signal Processing, vol. 25, no. 5, pp. 392-410, 1977

[DUH1990] P. Duhamel, M. Vetterli, **“Fast fourier transforms: A tutorial review and a state of the art”**. Signal Processing, 19(4), 259–299, 1990.

[GAR1987] H. K. Garg, **“Analysis of the Chinese remainder theorem and cyclotomic polynomials-based algorithms for cyclic convolution—Part I: Rational number system”**, Circuits, Systems and Signal Processing volume 16, pages595–610, 1997.

Bibliography

[DUD1984] D. E. Dudgeon, R. M. Mersereau, **“Multidimensional Digital Signal Processing”**, Prentice Hall, 1984.

[TRA2015] D. Tran, L. Bourdev, R. Fergus, L. Torresani and M. Paluri, **“Learning Spatiotemporal Features with 3D Convolutional Networks”**, IEEE International Conference on Computer Vision (ICCV), 2015.

[HEI2005] J. Heike, “**Fourier transforms for the bluegene communication network**”, Master’s thesis, The University of Edinburgh, 2005.

[DON2020] Dong H, Zhang L, Zou B. “**PolSAR Image Classification with Lightweight 3D Convolutional Networks**”. *Remote Sensing*. 2020.

[WIK-MRI] https://en.wikipedia.org/wiki/Magnetic_resonance_imaging

[WIK-FFT] https://en.wikipedia.org/wiki/Cooley–Tukey_FFT_algorithm

[WIK-CIRCCONV] https://en.wikipedia.org/wiki/Circular_convolution

Q & A

Thank you very much for your attention!

More material in
http://icarus.csd.auth.gr/cvml-web-lecture-series/

Contact: Prof. I. Pitas
pitas@csd.auth.gr