CS 598 EVS: Tensor Computations

Bilinear Algorithms

Edgar Solomonik

University of Illinois at Urbana-Champaign

Bilinear Problems

- A number of basic numerical problems can be thought of as bilinear functions associated with particular order 3 tensors

$$z = f^{(T)}(x, y)$$

defines the "problem"

$$z_i = \sum_{jk} t_{ijk} x_j y_k$$

- convolution
- matrix multiplication (tensor contraction)
- symmetry

- These problems admit nontrivial fast *bilinear algorithms*, which correspond to low-rank CP decompositions of the tensors

(A, B, C)

minimize # of products

$$z = C^T (Ax * By) = f^{(T)}(x, y)$$

"linear combinations of elements of x"

Bilinear Problems

- A bilinear problem for any inputs $\boldsymbol{a} \in \mathbb{R}^n$ and $\boldsymbol{b} \in \mathbb{R}^k$ computes $\boldsymbol{c} \in \mathbb{R}^m$ as defined by a tensor $\mathcal{T} \in \mathbb{R}^{m \times n \times k}$

$$c = f^{(\mathcal{T})}(a, b)$$

- Variants of discrete convolutions (linear convolution, correlation, cyclic convolution) provide simple examples of $\mathcal{T}$

transposition of $\mathcal{T}$

$$c_i = \sum_j a_j b_{i-j}$$

$$c_i = \sum_{j,k} t_{ijk} a_j b_k$$

$t_{ijk} = 1$ if $k = i-j \mod n$

$= 0$ otherwise

$$\bar{C}_{ij} = \sum_k \bar{a}_{ik} \bar{b}_{kj}$$

$\bar{A}, \bar{B}, \bar{C} \in \mathbb{R}^{n \times n}$

$T \in \mathbb{R}^{n^2 \times n^2 \times n^2}$

$$\downarrow$$

$$C_{\underbrace{i+(j-1)n}_{I}} = \sum_k a_{\underbrace{i+(k-1)n}_{J}} b_{\underbrace{k+(j-1)n}_{K}}$$

$$C_I = \sum_{JK} t_{IJK} a_J b_K$$

$t_{IJK} = 1$ if $\exists i, j, k$

$I = i+(j-1)n$, $J = i+(k-1)n$

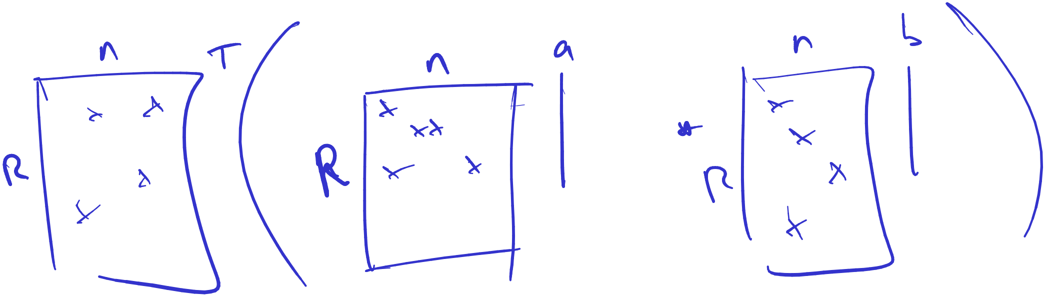
...

Bilinear Algorithms

A bilinear algorithm (V. Pan, 1984) $\Lambda = (\underline{\boldsymbol{F}^{(A)}}, \underline{\boldsymbol{F}^{(B)}}, \underline{\boldsymbol{F}^{(C)}})$ computes where $\boldsymbol{a}$

$$c = F^{(C)T}\left(F^{(A)} a * F^{(B)} b\right)$$

and $\boldsymbol{b}$ are inputs and $*$ is the Hadamard (pointwise) product.

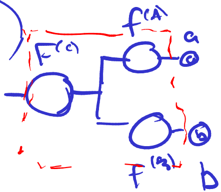


Bilinear Algorithms as Tensor Factorizations

- A bilinear algorithm corresponds to a CP tensor decomposition

$$c_i = \sum_{r=1}^{R} f^{(C)}_{ir} \left(\sum_{j=1}^{n} f^{(A)}_{jr} a_j\right) \cdot \left(\sum_{k=1}^{n} f^{(B)}_{kr} b_k\right)$$

$$= \sum_{j=1}^{n} \sum_{k=1}^{n} \underbrace{\sum_{r=1}^{R} f^{(C)}_{ir} f^{(A)}_{jr} f^{(B)}_{kr}}_{t_{ijk}} a_j b_k$$



- For multiplication of $n \times n$ matrices, we can define a *matrix multiplication tensor* and consider algorithms with various bilinear rank

$T \in \mathbb{R}^{n^2 \times n^2 \times n^2}$ $\quad$ T has n^3 nonzeros (1s)

$a_{ik} b_{kj}$

T has $\underset{CP}{\text{rank}} \le n^3$

$e_i \otimes e_j \otimes e_k$

$n=2$ $\quad$ $T \in \mathbb{R}^{4\times 4\times 4}$, CP rank of T is 7.

Strassen's Algorithm

Strassen's algorithm $\begin{bmatrix} \boldsymbol{C}_{11} & \boldsymbol{C}_{12} \\ \boldsymbol{C}_{21} & \boldsymbol{C}_{22} \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{12} \\ \boldsymbol{A}_{21} & \boldsymbol{A}_{22} \end{bmatrix} \cdot \begin{bmatrix} \boldsymbol{B}_{11} & \boldsymbol{B}_{12} \\ \boldsymbol{B}_{21} & \boldsymbol{B}_{22} \end{bmatrix}$

$$
\begin{aligned}
\boldsymbol{M}_1 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{22}) \\
\boldsymbol{M}_2 &= (\boldsymbol{A}_{21} + \boldsymbol{A}_{22}) \cdot \boldsymbol{B}_{11} \\
\boldsymbol{M}_3 &= \boldsymbol{A}_{11} \cdot (\boldsymbol{B}_{12} - \boldsymbol{B}_{22}) \\
\boldsymbol{M}_4 &= \boldsymbol{A}_{22} \cdot (\boldsymbol{B}_{21} - \boldsymbol{B}_{11}) \\
\boldsymbol{M}_5 &= (\boldsymbol{A}_{11} + \boldsymbol{A}_{12}) \cdot \boldsymbol{B}_{22} \\
\boldsymbol{M}_6 &= (\boldsymbol{A}_{21} - \boldsymbol{A}_{11}) \cdot (\boldsymbol{B}_{11} + \boldsymbol{B}_{12}) \\
\boldsymbol{M}_7 &= (\boldsymbol{A}_{12} - \boldsymbol{A}_{22}) \cdot (\boldsymbol{B}_{21} + \boldsymbol{B}_{22})
\end{aligned}
$$

$$
\begin{aligned}
\boldsymbol{C}_{11} &= \boldsymbol{M}_1 + \boldsymbol{M}_4 - \boldsymbol{M}_5 + \boldsymbol{M}_7 \\
\boldsymbol{C}_{21} &= \boldsymbol{M}_2 + \boldsymbol{M}_4 \\
\boldsymbol{C}_{12} &= \boldsymbol{M}_3 + \boldsymbol{M}_5 \\
\boldsymbol{C}_{22} &= \boldsymbol{M}_1 - \boldsymbol{M}_2 + \boldsymbol{M}_3 + \boldsymbol{M}_6
\end{aligned}
$$

$f^{(c)} = 4 \begin{bmatrix} 1 & 1\ -1 & 1 \end{bmatrix}$

By performing the nested calls recursively, Strassen's algorithm achieves cost,

$$T(n) = 7T(n/2) + O(n^2) = O(n^{\log_2 7})$$

$$T_{(n^2)} = T_{(n)} \otimes T_{(n)} = [A \otimes A,\ B \otimes B,\ C \otimes C]$$

Fast Bilinear Algorithms for Convolution

- Linear convolution corresponds to polynomial multiplication

degree n-1

$a, b \in \mathbb{R}^n$

$p(x) = a_0 + a_1 x + a_2 x^2 + \dots$

$q(x) = b_0 + b_1 x + b_2 x^2 + \dots$

x^2

$p \cdot q$ degree $2n-2$

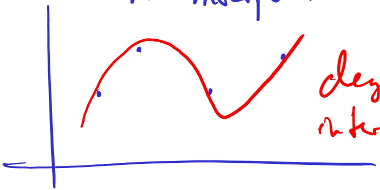
$$p \cdot q(x) = \sum_{i=1}^{2n-2} \sum_j \underbrace{a_j b_{i-j}} x^i$$

$$p \cdot q(x) = \sum_{i=1}^{2n-2} c_i x^i = c_0 + c_1 x + c_2 x^2 + \dots$$

$$c = a \circ b$$

- The *Toom-Cook* convolution algorithm computes the coefficients of $p \cdot q$ by computing $(p \cdot q)(x_i)$ for $i \in \{1, \dots, n + k - 1\}$ and interpolates

n interpolation points



degree n-1 poly. interpolant is unique

$p \cdot q(x_i) = p(x_i)\, q(x_i)$

Vandermonde matrix

$V_{ij}(x) = x_i^j$

$(V(x)a)_i = p_a(x_i) = a_0 + a_1 x_i + \dots$

$p_a(x) = y$

solve

$V(x)a = y$

need $2n-1$ nodes

for $p \cdot q$ interpolant to be specified fully

$x \in \mathbb{R}^{2n-1}$

$\sim \mathbb{C}^{2n-1}$

$p_a(x)$ $\quad$ $p_b(x)$

$p_c(x)$

$$c = \underline{V^{-1}_{(2n-1)}(x)} \left(\underline{V_{(n)}(x)}\, a * \underline{V_{(n)}(x)}\, b \right)$$

$V_{(2n-1)} \in \mathbb{R}^{2n-1 \times 2n-1}$ $\quad$ $V_{(n)}(x) \in \mathbb{R}^{2n-1 \times n}$


2n-1
2n-1
n
2n-1

$$\underline{V_{(2n-1)}^{-1}} \qquad \left(\boxed{\underline{V_{(2n-1)}}} \begin{bmatrix} a \\ 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} , \quad \begin{bmatrix} \; \\ \; \end{bmatrix} \right)$$

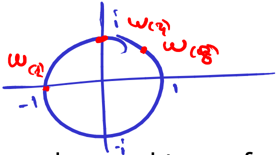
$$\underline{k = 2n-1}$$

Toom-Cook Convolution and the Fourier Transform

- Vandermonde matrices are ill-conditioned with real nodes, but can be perfectly conditioned with complex nodes

$\kappa(V_{(n)}) = \Theta(2^n)$ if $x \in \mathbb{R}^n$

instead $x \in \mathbb{C}^n$



$\omega_{(n)}^n = 1$

nth roots of unity

- The *fast Fourier transform (FFT)* can be used to perform products with the DFT matrix in $O(n \log n)$ time

$$D_{(n)} = V_{(n)}\left(\begin{bmatrix} \omega_{(n)}^0 \\ \vdots \\ \omega_{(n)}^{n-1} \end{bmatrix}\right) \qquad \left(\frac{1}{n} D_{(n)}\right)^{-1} = D_{(n)}^H$$

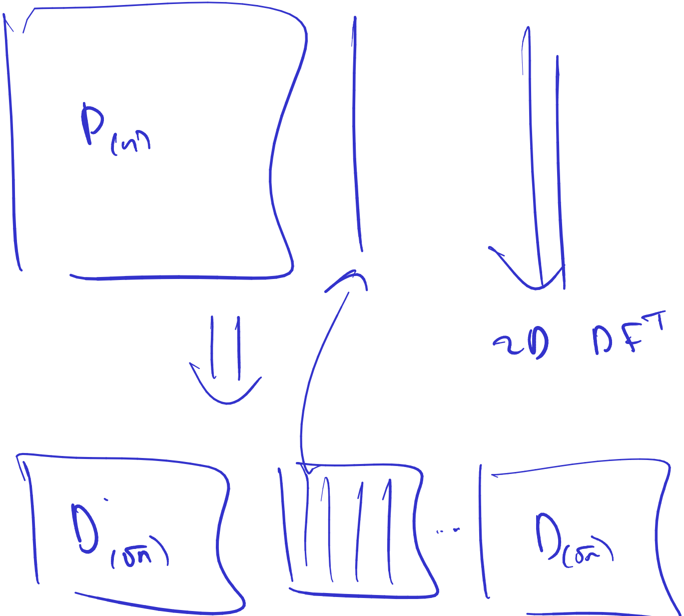
$$D_{(n)}[i,j] = \omega_{(n)}^{ij}$$

$D_{(n)}$ is symmetric but not Hermitian

1D DFT

$\Downarrow$

2D DFT



$$b = D_{(n)} a$$

$$b_i = \sum_j \omega_{(n)}^{ij} a_j$$

$$b_{i_1 + i_2(\sqrt{n})} = \sum_{j_1} \sum_{j_2} \omega_{(n)}^{(i_1 + i_2(\sqrt{n}))(j_1 + j_2(\sqrt{n}))} a_{j_1 + j_2(\sqrt{n})}$$

Cyclic Convolution via DFT

- For linear convolution $\boldsymbol{D}^{(n+k-1)}$ is used, for cyclic convolution $\boldsymbol{D}^{(n)}$ suffices

$$c_i = \sum_{j=0}^{n-1} a_j b_{(i-j) \bmod n}$$

$$\omega \left\{ \frac{1 - \cancel{\omega_{(n)}^{n(j+k-i)}}}{1 - \omega_{(n)}^{(j+k-i)}} = \sum_{r=0}^{n-1} \omega_{(n)}^{r(j+k-i)} = 0 \text{ if } j+k-i = 0 \right.$$

$$c_i = \frac{1}{n} \sum_{r=1}^{n} \omega_{(n)}^{-ir} \left(\sum_{j=1}^{n} \omega_{(n)}^{rj} a_j \right) \left(\sum_{k=1}^{n} \omega_{(n)}^{rk} b_k \right) = \frac{1}{n} \sum_{j,k,r} \omega_{(n)}^{-ir+rj+rk} a_j b_k$$

- The DFT also arises in the eigendecomposition of a circulant matrix

Toeplitz

$$T(a) = \begin{bmatrix} a_0 & a_{n+1} & \cdots & \\ a_1 & \ddots & \ddots & \\ \vdots & \ddots & \ddots & a_{n+1} \\ a_n & \cdots & a_1 & a_0 \end{bmatrix}$$

$$a \circ b = T\left(\begin{bmatrix} a \\ 0 \end{bmatrix}\right) b = \left(a^T T\left(\begin{bmatrix} b \\ 0 \end{bmatrix}\right)\right)^T$$

$$\begin{bmatrix} a_0 & & \\ a_1 & \ddots & \\ & \ddots a_1 & a_0 \end{bmatrix}$$

$$
\begin{bmatrix}
a_0 & & & \\
a_1 & \ddots & & \\
\vdots & \ddots & a_0 & \\
a_n & \cdots & a_1 \; \vdots & a_n
\end{bmatrix}
\begin{bmatrix}
b_0 \\ \vdots \\ b_n
\end{bmatrix}
=
\begin{bmatrix}
a_0 b_0 \\
a_1 b_0 + a_0 b_1 \\
a_2 b_0 + a_1 b_1 + a_0 b_2 \\
\vdots
\end{bmatrix}
$$

circulant matrix

$$C(x) = \begin{bmatrix} x_0 & x_{n-1} & & & x_2 & x_1 \\ x_1 & \ddots & & & & x_2 \\ x_2 & & \ddots & & & \\ & & & \ddots & & \\ & & & & \ddots & x_{n-1} \\ x_{n-1} & & & x_2 & x_1 & x_0 \end{bmatrix}$$

The eigenvectors of C are columns of $\underline{D_{(n)}}$

Winograd's Algorithm for Convolution

- The DFT/FFT requires complex arithmetic, motivating alternatives such as the more general Winograd family of algorithms

Algebraic Formulation of Winograd's Algorithm for Convolution

- Winograd's convolution algorithm can be written as a bilinear algorithm by defining appropriate linear transformations

Algebraic Formulation of Winograd's Algorithm for Convolution

- Given an operator $\boldsymbol{X}_{\langle m,d\rangle} \in \mathbb{C}^{\mathsf{deg}(m)\times(d+1)}$ to compute coefficients of $\rho = p \pmod{m}$, we can efficiently compute

Algebraic Formulation of Winograd's Algorithm for Convolution

- Winograd's convolution algorithm effectively merges smaller bilinear algorithms for linear convolution

Algebraic Formulation of Winograd's Algorithm for Convolution

- A missing piece of the above formulation is how to realize Bézout's identity to compute $N^{(i)}$ and $e^{(i)}$

Nested Bilinear Algorithms for Convolution

- 2D convolution is equivalent to nested 1D convolution
- 1D convolution can be reduced to 2D convolution with some work
- For more details on the above derivations and a broader survey of convolution algorithms, see `https://arxiv.org/abs/1910.13367`

Symmetric Tensor Contractions

- Bilinear algorithms can also be used to accelerate tensor contractions for tensors with symmetry
- Bilinear algorithms for symmetric tensor contractions exist with lower rank than their nonsymmetric counterparts

Symmetric Matrix Vector Product

- Consider computing $\boldsymbol{c} = \boldsymbol{A}\boldsymbol{b}$ with $\boldsymbol{A} = \boldsymbol{A}^T$

Partially-Symmetric Tensor Times Matrix (TTM)

- Can use symmetric mat-vec algorithm to accelerate TTM with partially symmetric tensor from $2n^4$ operations to $(3/2)n^4 + O(n^3)$

Computing Symmetric Matrices

- Output symmetry can also be used to reduced cost, for example when computing a symmetrized outer product $\boldsymbol{C} = \boldsymbol{a}\boldsymbol{b}^T + \boldsymbol{b}\boldsymbol{a}^T$

- To symmetrize product of two symmetric matrices, can compute anticommutator, $\boldsymbol{C} = \boldsymbol{A}\boldsymbol{B} + \boldsymbol{B}\boldsymbol{A}$

General Symmetric Tensor Contractions

- We can now consider the cost of a symmetrized contraction over v indices of symmetric tensors $\boldsymbol{\mathcal{A}}$ (of order $s+v$) and $\boldsymbol{\mathcal{B}}$ (of order $v+t$)
- Such tensor contractions can be done using $n^{s+t+v}/(s+t+v)! + O(n^{s+t+v-1})$ products