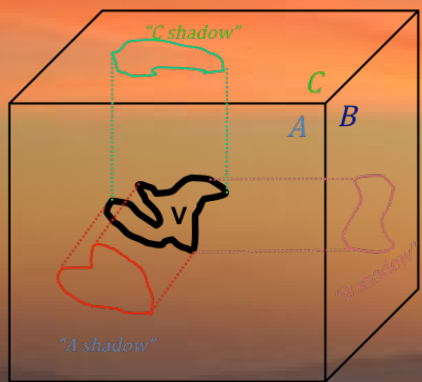
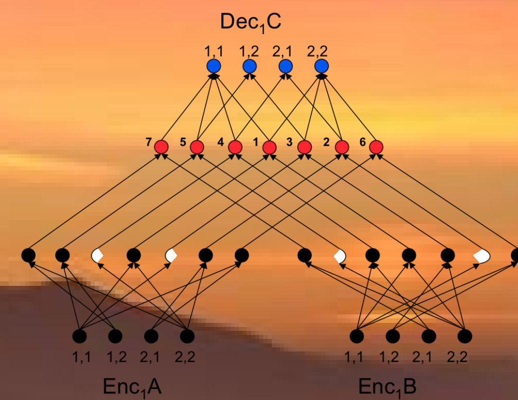
FAST MATRIX MULTIPLICATION

THEORY AND PRACTICE





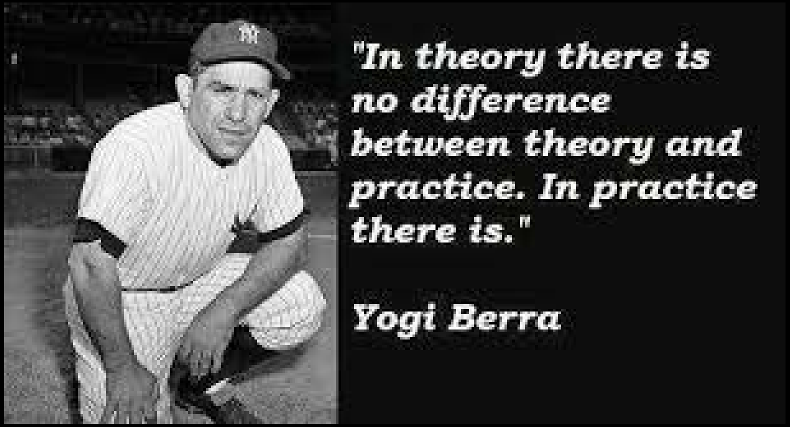
OLGA HOLTZ

UC Berkeley

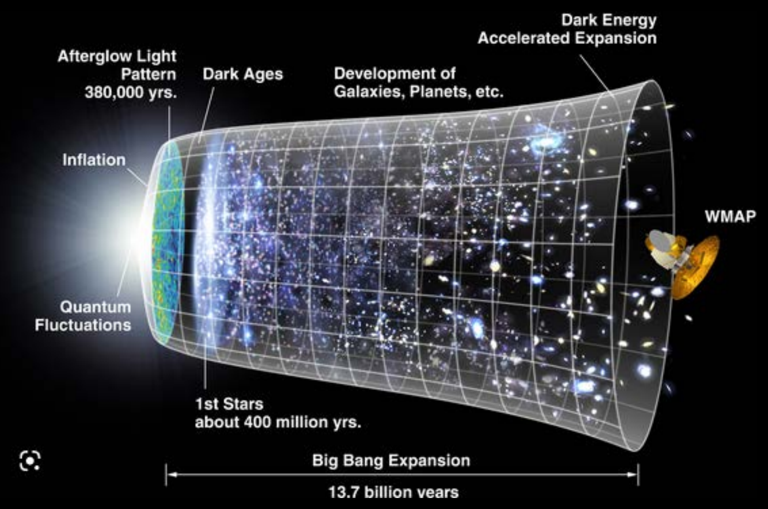
WACT 2025

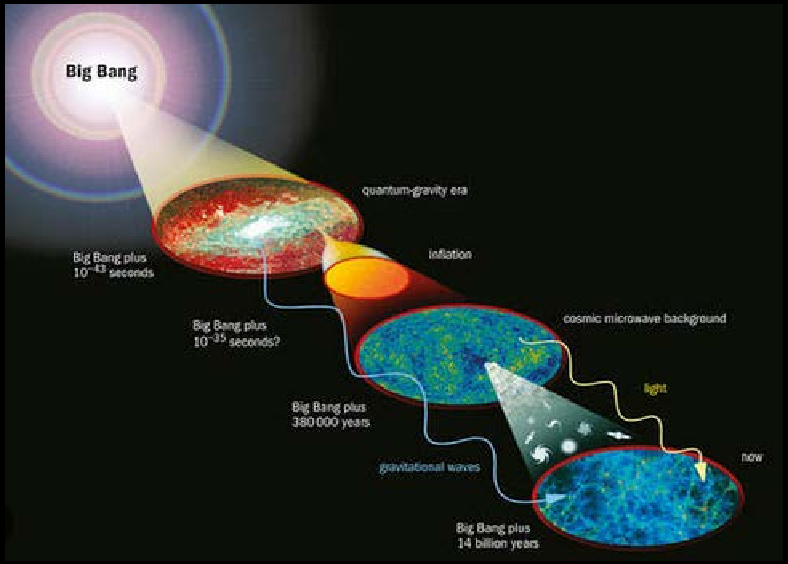
Bochum, Germany

ABSTRACT


"In theory there is no difference between theory and practice. In practice there is."
Yogi Berra

WHY MATRIX MULTIPLICATION?





Real-world problems of data science, physics, engineering are solved through **linearization.**

This requires manipulating huge amounts of data organized as rectangular arrays = **matrices.**

Matrix multiplication is the workhorse of numerical linear algebra.

Arithmetic complexity (= number of arithmetic operations performed) of matrix multiplication **determines** arithmetic complexity of **all direct** matrix algorithms.

STRASSEN'S FAST MATRIX MULTIPLICATION

[Strassen 1969]

Compute 2 x 2 matrix multiplication
using only 7 multiplications (instead of 8).
Apply recursively (block-wise)

$$\begin{array}{c} n/2 \\ n/2 \end{array} \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{bmatrix} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \bullet \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}$$

$$\begin{aligned}
M_1 &= (A_{11} + A_{22}) \cdot (B_{11} + B_{22}) \\
M_2 &= (A_{21} + A_{22}) \cdot B_{11} \\
M_3 &= A_{11} \cdot (B_{12} - B_{22}) \\
M_4 &= A_{22} \cdot (B_{21} - B_{11}) \\
M_5 &= (A_{11} + A_{12}) \cdot B_{22} \\
M_6 &= (A_{21} - A_{11}) \cdot (B_{11} + B_{12}) \\
M_7 &= (A_{12} - A_{22}) \cdot (B_{21} + B_{22})
\end{aligned}$$

$$\begin{aligned}
C_{11} &= M_1 + M_4 - M_5 + M_7 \\
C_{12} &= M_3 + M_5 \\
C_{21} &= M_2 + M_4 \\
C_{22} &= M_1 - M_2 + M_3 + M_6
\end{aligned}$$

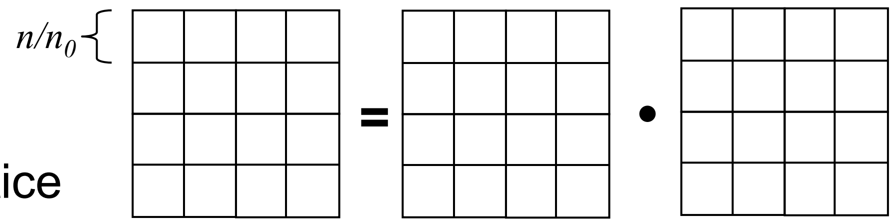
$$T(n) = 7 \cdot T(n/2) + O(n^2)$$

$$\Rightarrow T(n) = \Theta(n^{\omega})$$

$$\omega = \log_2 7 \approx 2.81 < 3$$

POST-STRASSEN IMPROVEMENTS

- Compute n_0 x n_0 matrix multiplication using only n_0^{ω} multiplications (instead of n_0^3).
- Apply recursively (block-wise)



$T(n) = n_0^{\omega} \cdot T(n/n_0) + O(n^2)$

$\Rightarrow T(n) = \Theta(n^{\omega})$

$\omega \approx$

2.81 [Strassen 1969] works fast in practice

2.79 [Pan 1978]

2.78 [Bini 1979]

2.55 [Schönhage 1981]

2.50 [Pan Romani, Coppersmith Winograd 1984]

2.48 [Strassen 1987]

2.376 [Coppersmith Winograd 1990]

2.373 [Vassilevska Williams 2011]

2.37287 [Le Gall 2014]

2.37286 [Alman Vassilevska Williams 2020]

2.37**155** [**Vassilevska Williams Xu Xu Zhou 2023**]

TENSORS AND MATRIX TRIPLES

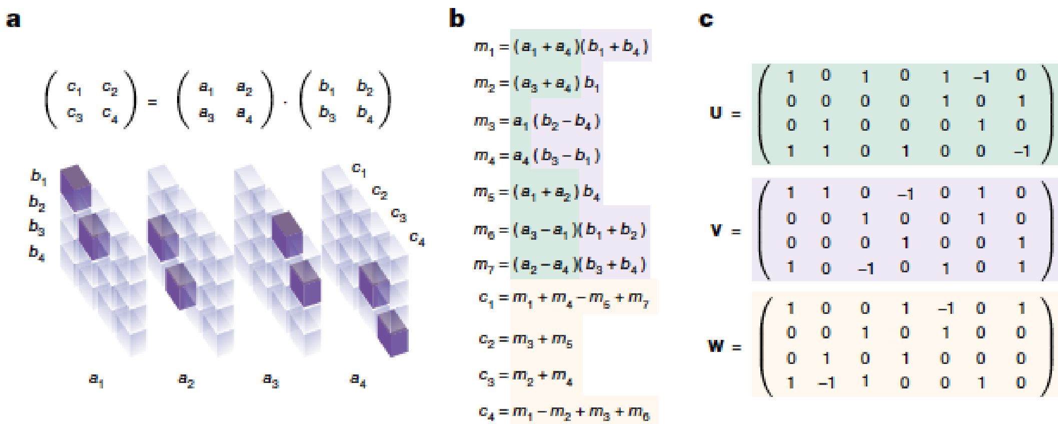


Fig. 1 | Matrix multiplication tensor and algorithms. a, Tensor $\mathcal{T}_2$ representing the multiplication of two 2×2 matrices. Tensor entries equal to 1 are depicted in purple, and 0 entries are semi-transparent. The tensor specifies which entries from the input matrices to read, and where to write the result. For example, as $c_1 = a_1b_1 + a_2b_3$, tensor entries located at (a_1, b_1, c_1) and (a_2, b_3, c_1) are set to 1. **b**, Strassen's algorithm[2] for multiplying 2×2 matrices using 7 multiplications. **c**, Strassen's algorithm in tensor factor representation. The stacked factors **U**, **V** and **W** (green, purple and yellow, respectively) provide a rank-7 decomposition of $\mathcal{T}_2$ (equation (1)). The correspondence between arithmetic operations (**b**) and factors (**c**) is shown by using the aforementioned colours.

The complexity of matrix multiplication is measured **by the rank**, or rather, **the border rank** of the **matrix multiplication tensor.**

Algorithm 1

A meta-algorithm parameterized by $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$ for computing the matrix product $\mathbf{C}=\mathbf{AB}$. It is noted that R controls the number of multiplications between input matrix entries.

Parameters: $\{\mathbf{u}^{(r)}, \mathbf{v}^{(r)}, \mathbf{w}^{(r)}\}_{r=1}^{R}$: length-$n^2$ vectors such that $\mathcal{T}_n = \sum_{r=1}^{R} \mathbf{u}^{(r)} \otimes \mathbf{v}^{(r)} \otimes \mathbf{w}^{(r)}$

Input: **A**, **B**: matrices of size $n \times n$

Output: $\mathbf{C}=\mathbf{AB}$

(1) **for** $r=1, \ldots, R$ **do**

(2) $\quad m_r \leftarrow (u_1^{(r)}a_1 + \cdots + u_{n^2}^{(r)}a_{n^2})\,(v_1^{(r)}b_1 + \cdots + v_{n^2}^{(r)}b_{n^2})$

(3) **for** $i=1, \ldots, n^2$ **do**

(4) $\quad c_i \leftarrow w_i^{(1)}m_1 + \cdots + w_i^{(R)}m_R$

return **C**

THE LASER METHOD

- **Strassen's laser method** starts with a tensor of near-minimal border rank and builds a large tensor from it, which admits a degeneration to a large matrix multiplication tensor

- All advances since 1987 so far were based on the **Coppersmith-Winograd tensor**

- **Barriers to upper bounds** were found and clarified over the past decade, including a geometric identification of the barriers inspired by quantum information theory

- New methods to overcome those barriers were just recently set forth by **algebraic geometry**

AI FINDS NEW FAST MATRIX MULTIPLICATION

Article

Discovering faster matrix multiplication algorithms with reinforcement learning

https://doi.org/10.1038/s41586-022-05172-4

Received: 2 October 2021

Accepted: 2 August 2022

Published online: 5 October 2022

Open access

Check for updates

Alhussein Fawzi[1,2,3], Matej Balog[1,2], Aja Huang[1,2], Thomas Hubert[1,2], Bernardino Romera-Paredes[1,2], Mohammadamin Barekatain[1], Alexander Novikov[1], Francisco J. R. Ruiz[1], Julian Schrittwieser[1], Grzegorz Swirszcz[1], David Silver[1], Demis Hassabis[1] & Pushmeet Kohli[1]

Improving the efficiency of algorithms for fundamental computations can have a widespread impact, as it can affect the overall speed of a large amount of computations. Matrix multiplication is one such primitive task, occurring in many systems—from neural networks to scientific computing routines. The automatic discovery of algorithms using machine learning offers the prospect of reaching beyond human intuition and outperforming the current best human-designed algorithms. However, automating the algorithm discovery procedure is intricate, as the space of possible algorithms is enormous. Here we report a deep reinforcement learning approach based on AlphaZero[1] for discovering efficient and provably correct algorithms for the multiplication of arbitrary matrices. Our agent, AlphaTensor, is trained to play a single-player game where the objective is finding tensor decompositions within a finite factor space. AlphaTensor discovered algorithms that outperform the state-of-the-art complexity for many matrix sizes. Particularly relevant is the case of 4×4 matrices in a finite field, where AlphaTensor's algorithm improves on Strassen's two-level algorithm for the first time, to our knowledge, since its discovery 50 years ago[2]. We further showcase the flexibility of AlphaTensor through different use-cases: algorithms with state-of-the-art complexity for structured matrix multiplication and improved practical efficiency by optimizing matrix multiplication for runtime on specific hardware. Our results highlight AlphaTensor's ability to accelerate the process of algorithmic discovery on a range of problems, and to optimize for different criteria.

MAIN IDEA:

Attack 3D tensor decomposition problem (which is NP hard) via deep reinforcement learning (DRL) instead of earlier strategies such as human search or continuous minimization.

NEW ALTERNATIVE METHODS

Karstadt and Schwartz [2020] developed a method to pre- and post-process the matrix triple U, V, W before using them. This does not change the exponent of the corresponding matrix multiplication method but improves (sometimes substantially) its leading cofficients. Further work is ongoing, jointly with many people.

Kauers and Moosbauer [2023] suggested random walks on the so-called flip graphs to find new fast matrix multiplication algorithms. These graphs, where vertices represent algorithms and edges local transformations or ”flips,” encode the landscape of possible MatMul algorithms.

FLIP GRAPH METHOD

arXiv:2212.01175v1 [cs.SC] 2 Dec 2022

FLIP GRAPHS FOR MATRIX MULTIPLICATION

MANUEL KAUERS* AND JAKOB MOOSBAUER†

ABSTRACT. We introduce a new method for discovering matrix multiplication schemes based on random walks in a certain graph, which we call the flip graph. Using this method, we were able to reduce the number of multiplications for the matrix formats (4, 4, 5) and (5, 5, 5), both in characteristic two and for arbitrary ground fields.

1. INTRODUCTION

Nobody knows the computational cost of computing the product of two matrices. Strassen's discovery [17] that two 2×2-matrices can be multiplied with only 7 multiplications in the ground field launched intensive research on the complexity of matrix multiplication during the past decades. One branch of this research aims at finding upper (or possibly also lower) bounds on the matrix multiplication exponent ω. The current world record $\omega < 2.37188$ is held by Duan, Wu and Zhou [7] and only slightly improves the previous record $\omega < 2.37286$ by Alman and Williams [1]. These results concern asymptotically large matrix sizes.

Another branch of research on matrix multiplication algorithms concerns specific small matrix sizes. For 2×2 matrices, it is known that there is no way to do the job with only 6 multiplications [19], and that Strassen's algorithm is essentially the only way to do it with 7 [6]. Also for multiplying a 2×2 matrix with a $2\times p$ matrix and for multiplying a 2×3 matrix with a 3×3 matrix, optimal algorithms are known [11]. For all other formats, the known upper and lower bounds do not match. For example, for the case 3×3 times 3×3, the best known upper bound is 23 [14] and the best known lower bound is 19 [2] unless we impose restrictions on the ground domain such as commutativity.

An upper bound for a specific matrix format can be obtained by stating an explicit matrix multiplication scheme with as few multiplications as possible. Such schemes can be discovered by various techniques, including hand calculation [17, 14], numerical methods [16, 18], SAT solving [5, 10, 9], or machine learning [8]. The latter approach, due to Fawzi et al., has received a lot of attention, even in the general public, because it led to an unexpected improvement of the upper bound for multiplying two 4×4 matrices from 49 to 47 multiplications in characteristic two. They also reduced the bound for multiplying two 5×5 matrices from 98 to 96 in characteristic two, and found improvements for some formats involving rectangular matrices.

In our quick response [12] to the paper of Fawzi et al., we announced that we can find further schemes for 4×4 matrices using 47 multiplications in characteristic two, and that we can reduce the number of multiplications required for 5×5 matrices in characteristic two to 95. In the present paper, we introduce the method by which we found these schemes.

We define a graph whose vertices are correct matrix multiplication schemes and where there is an edge from one scheme to another if the second can be obtained from the first by some kind of transformation. We consider two transformations. One is called a flip and turns a given scheme to a different one with the same number of multiplications, and the other is called a reduction and turns a given scheme to one with a smaller number of multiplications. The precise construction of this flip graph is given in Sect. 3. In Sect. 4, we illustrate (parts of) the flip graph for 2×2 and 3×3 matrices.

In order to find better upper bounds for a specific format, we start from a known scheme, e.g., the standard algorithm, and perform a random walk in the flip graph. Although reduction edges are much more rare than flip edges, it turned out that there are enough of them to reach interesting schemes with a reasonable amount of computation time. In particular, we were able to match the best known algorithms for all multiplication formats $n\times m$ times $m\times p$ with $n,m,p\le 5$, and found better bounds in four cases. These results are reported in Sect. 5.

Key words and phrases. Bilinear complexity; Strassen's algorithm; Tensor rank.
* Supported by the Austrian FWF grants P31571-N32 and I6130-N.
† Supported by the Land Oberösterreich through the LIT-AI Lab.

1

VERTICES:

MatMul algorithms

EDGES:

Connected by flips

FLIPS:

$$A\otimes B\otimes \Gamma \;+\; A\otimes B'\otimes \Gamma'$$
$$= A\otimes (B+B')\otimes \Gamma \;+\; A\otimes B'\otimes (\Gamma'-\Gamma).$$

WHAT TO DO:

Random walk on this graph!

ALTERNATIVE BASIS METHOD

Matrix Multiplication, a Little Faster

ELAYE KARSTADT and ODED SCHWARTZ, The Hebrew University of Jerusalem

Strassen's algorithm (1969) was the first sub-cubic matrix multiplication algorithm. Winograd (1971) improved the leading coefficient of its complexity from 6 to 7. There have been many subsequent asymptotic improvements. Unfortunately, most of these have the disadvantage of very large, often gigantic, hidden constants. Consequently, Strassen-Winograd's $O(n^{\log_2 7})$ algorithm often outperforms other fast matrix multiplication algorithms for all feasible matrix dimensions. The leading coefficient of Strassen-Winograd's algorithm has been generally believed to be optimal for matrix multiplication algorithms with a 2×2 base case, due to the lower bounds by Probert (1976) and Bshouty (1995).

Surprisingly, we obtain a faster matrix multiplication algorithm, with the same base case size and asymptotic complexity as Strassen-Winograd's algorithm, but with the leading coefficient reduced from 6 to 5. To this end, we extend Bodrato's (2010) method for matrix squaring, and transform matrices to an alternative basis. We also prove a generalization of Probert's and Bshouty's lower bounds that holds under change of basis, showing that for matrix multiplication algorithms with a 2×2 base case, the leading coefficient of our algorithm cannot be further reduced, and is therefore optimal. We apply our method to other fast matrix multiplication algorithms, improving their arithmetic and communication costs by significant constant factors.

CCS Concepts: • **Mathematics of computing → Computations on matrices**; • **Computing methodologies → Linear algebra algorithms**;

Additional Key Words and Phrases: Fast matrix multiplication, bilinear algorithms

ACM Reference format:
Elaye Karstadt and Oded Schwartz. 2020. Matrix Multiplication, a Little Faster. *J. ACM* 67, 1, Article 1 (January 2020), 31 pages.
https://doi.org/10.1145/3364504

A preliminary version of this paper appeared in Proceedings of the 29th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA'17) [41].
This research is supported by grants 1878/14, and 1901/14 from the Israel Science Foundation (founded by the Israel Academy of Sciences and Humanities) and grant 3-10891 from the Ministry of Science and Technology, Israel. This research was also supported by the Einstein Foundation and the Minerva Foundation; the PetaCloud industry-academia consortium; by a grant from the United States-Israel Bi-national Science Foundation, Jerusalem, Israel; and the HUJI Cyber Security Research Center in conjunction with the Israel National Cyber Bureau in the Prime Minister's Office. We acknowledge PRACE for awarding us access to Hazel Hen at GCS@HLRS, Germany. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 818252).
Authors' addresses: E. Karstadt and O. Schwartz, The Rachel and Selim Benin, School of Computer Science and Engineering, The Hebrew University of Jerusalem, Rothberg Family Buildings, The Edmond J. Safra Campus, 9190416 Jerusalem, Israel; emails: elaye.karstadt@mail.huji.ac.il, odedsc@cs.huji.ac.il.

0004-5411/2020/01-ART1 $15.00
https://doi.org/10.1145/3364504

I IMPROVEMENTS:

Table 1. 2×2 Fast Matrix Multiplication Algorithms[2]

Algorithm	Additions	Arithmetic Complexity	IO-Complexity
Strassen [58]	18	$7n^{\log_2 7} - 6n^2$	$12 \cdot M \left(\sqrt{3} \cdot \frac{n}{\sqrt{M}}\right)^{\log_2 7} - 18n^2$
Strassen-Winograd [61]	15	$6n^{\log_2 7} - 5n^2$	$10.5 \cdot M \left(\sqrt{3} \cdot \frac{n}{\sqrt{M}}\right)^{\log_2 7} - 15n^2$
Ours	12	$5n^{\log_2 7} - 4n^2 + 3n^2 \log_2 n$	$9 \cdot M \left(\sqrt{3} \cdot \frac{n}{\sqrt{M}}\right)^{\log_2 7} + 9n^2 \cdot \log_2 \left(\sqrt{2} \cdot \frac{n}{\sqrt{M}}\right)$

Table 2. Alternative Basis Algorithms

Algorithm	Linear Operations	Improved Linear Operations	Arithmetic Leading Coefficient	Improved Leading Coefficient	Computations Saved
$\langle 2, 2, 2; 7\rangle$ [61]	15	12	6	5	16.6%
$\langle 3, 2, 3; 15\rangle$ [56]	64	52	9.61	7.94	17.37%
$\langle 2, 3, 4; 20\rangle$ [56]	78	58	8.8	7.46	16.18%
$\langle 3, 3, 3; 23\rangle$ [56]	87	75	7.21	6.57	8.87%
$\langle 6, 3, 3; 40\rangle$ [56]	1246	198	55.63	9.36	83.17%

HOW?

$$W^T((U \cdot \vec{A}) \odot (V \cdot \vec{B})) = \overrightarrow{A \cdot B}.$$

$$v(\overrightarrow{A \cdot B}) = v(W^T(U \cdot \vec{A} \odot V \cdot \vec{B})) = (Wv^T)^T(U\phi^{-1} \cdot \phi(\vec{A}) \odot V\psi^{-1} \cdot \psi(\vec{B})).$$

SOME VERY RECENT RESULTS

MAIN IDEA:

Just sparsify U, V, and W!

A FEW HIGHLIGHTS:

Algorithms	Leading Monomial	Linear Operations		Leading Coefficients				
		Original	[Here]	Original	[Here]	Saving	Square [Here]	Saving
$\langle 2,6,6;56\rangle$ [30]	$n^{\log_{72} 56^3} \approx n^{2.824}$	1361	284	57.84	8.87	84.66%	8.61	85.11%
$\langle 4,4,5;62\rangle$ [29]	$n^{\log_{80} 62^3} \approx n^{2.835}$	1465	284	35.31	7.58	78.54%	7.58	78.54%
$\langle 3,4,5;47\rangle$ [18]	$n^{\log_{60} 47^3} \approx n^{2.831}$	298	228	10.52	8.26	21.48%	8.25	21.58%
$\langle 3,4,11;103\rangle$ [18]	$n^{\log_{132} 103^3} \approx n^{2.848}$	701	512	11.0	7.99	27.36%	7.96	27.64%
$\langle 3,5,9;105\rangle$ [18]	$n^{\log_{135} 105^3} \approx n^{2.846}$	670	560	10.04	8.43	16.04%	8.3	17.33%
$\langle 4,5,5;76\rangle$ [18]	$n^{\log_{100} 76^3} \approx n^{2.831}$	549	437	11.14	9.06	18.67%	8.97	19.48%

Table 1: Decomposition of new algorithms with recent improvements in exponents by flip graph [30, 29] and AlphaTensor [18]. Exponents were computed by the method in [22], calculated as $\log_{n\cdot m\cdot k} t^3$, defined in Section 2.1. See Table 3 for new algorithms with no exponent improvements. The column "Linear Operations" contains two columns representing the sum $q_U + q_V + q_W$ of the "Original" and our sparse matrix multiplication algorithm. The column "Leading Coefficients" contains five columns representing the "Original" leading coefficient, our leading coefficient, the percentage improved, our leading coefficient with the method in section 5, and the total percentage improvement with section 5.

Alternative Bases for New Fast Matrix Multiplication Algorithms

Olga Holtz* Abraham Hsu† Yoav Moran‡ Oded Schwartz§ Gal Wiernik¶

Abstract

Fast matrix multiplication algorithms are of practical use, provided that they apply to feasible input sizes and have small leading coefficients in their arithmetic and IO complexities. In recent years, many new sub-cubic time matrix multiplication algorithms that are applicable to feasible matrices have been introduced, including the recently discovered algorithms using AlphaTensor (Nature, 2022) and flip graphs (ISSAC 2023). However, their arithmetic and IO complexities have quite large leading coefficients, making them impractical.

We decrease these coefficients (by up to 89%), resulting in algorithms with more practical potential. To this end, we use the alternative basis method and provide a new way to compute the multiplications recursively. We provide an algorithm for finding optimal decompositions for the alternative basis method and use dynamic programming for the recursion reordering. Our new matrix multiplication algorithms retain the improved exponent while decreasing the leading coefficient of the arithmetic and communication costs. These result in the fastest existing algorithms for several base case dimensions. Combined with lower bounds on the arithmetic costs of bilinear algorithms, we conclude that some of our algorithms are optimal.

1 Introduction

Matrix multiplication is a fundamental computation used for many applications in computer science, physics, AI, and more. The arithmetic complexity of naive matrix multiplication algorithms is $\Theta(n^3)$, where n is the dimension of the matrix. In 1969, Strassen [51] discovered the first sub-cubic matrix multiplication algorithm, with arithmetic complexity of $\Theta(n^{\log_2 7})$. This discovery poses the question, what is the most efficient matrix multiplication algorithm?

Research on this topic is divided into two main categories. One category focuses on decreasing asymptotic complexity, often at the cost of huge minimal applicable input instances [1, 6, 14, 15, 17, 34, 44, 45, 50, 53, 57]. The other category includes algorithms that apply to feasible instances [3, 4, 5, 6, 23, 25, 26, 28, 32, 31, 38, 41, 44, 45, 48, 59, 20]. Fawzi et al. [18] and Kauers et al. [30, 29] recently discovered new matrix multiplication algorithms using AlphaTensor and flip graphs, respectively. However, the leading coefficients of the arithmetic cost of these algorithms are large, making them less feasible in practice.

1.1 Previous Work

1.1.1 Feasible Matrix Multiplication Algorithms. Strassen discovered the first fast matrix multiplication algorithm in 1969 [51]. Later, Pan [41, 40, 39, 38] used the trilinear aggregation technique to construct feasible asymptotically faster matrix multiplication algorithms. Specifically, Pan's [41] $\langle 44,44,44;36133\rangle$-algorithm[1] holds the current record for the fastest feasible algorithm. Since 2010, many algorithms have been discovered through computer-aided search. Smirnov [48] and later Tichavsky and Kovac [56] discovered matrix multiplication algorithms, such as the $\langle 3,3,6;40\rangle$-algorithm, that are asymptotically faster than Strassen's algorithm and have a smaller base case than Pan's [41] algorithms. Benson and Ballard [5] found additional algorithms through a search based on Johnson and Mcloughlin's [25] and Smirnov's [48] work. Fawzi et al. [18] discovered new algorithms using AlphaTensor. They encode the search for matrix multiplication algorithms as a game. More recently, Kauers et al. [29, 30] introduced a method to discover algorithms through random walks in graphs denoted as flip graphs. Arai et al. [2] improved this method by eliminating inefficiencies and reducing the multiplications required for a few algorithms.

1.1.2 Leading Coefficients Following Strassen's discovery, Winograd [59] decreased the leading coefficient of the arithmetic cost of Strassen's algorithm from $7n^{\log_2 7} - 6n^2$ to $6n^{\log_2 7} - 5n^2$. Bodrato [10] used the intermediate representation method and reduced the leading coefficient of repeated, and chain matrix multiplica-

* University of California at Berkeley
† University of California at Berkeley
‡ The Hebrew University of Jerusalem
§ The Hebrew University of Jerusalem
¶ The Hebrew University of Jerusalem

[1] See Section 2.1 for this notation.

MORE RESULTS

Algorithms	Leading Monomial	Linear Operations				Leading Coefficients					
		Original	[3]	[35]	[Here]	Original	[3]	[Here]	Saving	Square [Here]	Saving
(2, 2, 2; 7) [51]	$n^{\log_8 7^3} \approx n^{2.807}$	18	12	18	12	7.0	5.0	5.0	28.57%	5.0	28.57%
(3, 2, 2; 11) [5]	$n^{\log_{12} 11^3} \approx n^{2.89}$	22	18	21	18	5.06	4.26	4.26	19.01%	4.26	19.01%
(2, 3, 2; 11) [56]	$n^{\log_{12} 11^3} \approx n^{2.89}$	22	18	N/A	18	4.71	3.91	3.91	16.99%	3.91	16.99%
(4, 2, 2; 14) [5]	$n^{\log_{16} 14^3} \approx n^{2.856}$	48	28	37	28	8.33	5.27	5.27	36.73%	4.9	41.18%
(3, 2, 3; 15) [24]	$n^{\log_{18} 15^3} \approx n^{2.811}$	55	39	N/A	39	8.28	6.17	6.17	25.48%	6.12	26.09%
(3, 2, 3; 15) [5]	$n^{\log_{18} 15^3} \approx n^{2.811}$	64	39	44	39	9.61	6.17	6.17	35.80%	6.12	36.32%
(5, 2, 2; 18) [5]	$n^{\log_{20} 18^3} \approx n^{2.894}$	53	32	40	32	6.98	4.46	4.46	36.10%	4.37	37.39%
(4, 2, 3; 20) [49]	$n^{\log_{24} 20^3} \approx n^{2.828}$	78	51	N/A	51	8.9	5.88	5.88	33.93%	5.76	35.28%
(4, 2, 3; 20) [5]	$n^{\log_{24} 20^3} \approx n^{2.828}$	82	51	N/A	51	9.19	5.88	5.88	36.02%	5.76	37.32%
(4, 2, 3; 20) [5]	$n^{\log_{24} 20^3} \approx n^{2.828}$	86	54	N/A	54	9.38	6.12	6.12	34.75%	6.04	35.61%
(4, 2, 3; 20) [5]	$n^{\log_{24} 20^3} \approx n^{2.828}$	104	58	N/A	58	11.38	6.38	6.38	43.94%	6.18	45.69%
(2, 3, 4; 20) [5]	$n^{\log_{24} 20^3} \approx n^{2.828}$	96	58	62	58	9.96	6.12	6.12	38.55%	6.04	39.36%
(3, 3, 3; 23) [5]	$n^{\log_{27} 23^3} \approx n^{2.854}$	87	66	65	66	7.21	5.71	5.71	20.80%	5.71	20.80%
(3, 3, 3; 23) [5]	$n^{\log_{27} 23^3} \approx n^{2.854}$	88	65	64	65	7.29	5.64	5.64	22.63%	5.64	22.63%
(3, 3, 3; 23) [5]	$n^{\log_{27} 23^3} \approx n^{2.854}$	89	65	68	65	7.36	5.64	5.64	23.37%	5.64	23.37%
(3, 3, 3; 23) [5]	$n^{\log_{27} 23^3} \approx n^{2.854}$	97	61	63	61	7.93	5.36	5.36	32.41%	5.36	32.41%
(3, 3, 3; 23) [5]	$n^{\log_{27} 23^3} \approx n^{2.854}$	166	73	82	73	12.86	6.21	6.21	51.71%	6.21	51.71%
(3, 3, 3; 23) [31]	$n^{\log_{27} 23^3} \approx n^{2.854}$	98	74	62	74	8.0	6.29	6.29	21.38%	6.29	21.38%
(3, 3, 3; 23) [48]	$n^{\log_{27} 23^3} \approx n^{2.854}$	84	68	68	68	7.0	5.86	5.86	16.29%	5.86	16.29%
(4, 4, 2; 26) [5]	$n^{\log_{32} 26^3} \approx n^{2.82}$	235	105	N/A	96	18.1	7.81	7.08	60.88%	6.98	61.44%
(4, 3, 3; 29) [5]	$n^{\log_{36} 29^3} \approx n^{2.819}$	164	102	98	102	10.27	6.73	6.73	34.47%	6.71	34.66%
(3, 4, 3; 29) [5]	$n^{\log_{36} 29^3} \approx n^{2.819}$	137	109	N/A	109	8.54	6.96	6.96	18.50%	6.94	18.74%
(3, 4, 3; 29) [5]	$n^{\log_{36} 29^3} \approx n^{2.819}$	167	105	101	105	10.27	6.73	6.73	34.47%	6.71	34.66%
(3, 5, 3; 36) [49]	$n^{\log_{45} 36^3} \approx n^{2.824}$	199	139	N/A	139	9.62	6.87	6.87	28.59%	6.87	28.59%
(6, 3, 3; 40) [48]	$n^{\log_{54} 40^3} \approx n^{2.774}$	1246	190	N/A	190	55.63	8.9	8.9	84.00%	8.64	84.47%
(3, 3, 6; 40) [56]	$n^{\log_{54} 40^3} \approx n^{2.774}$	1822	190	N/A	190	79.28	8.9	8.9	88.77%	8.64	89.10%

COMMUNICATION (I/O) COMPLEXITY

Two kinds of costs:

Arithmetic (FLOPs)

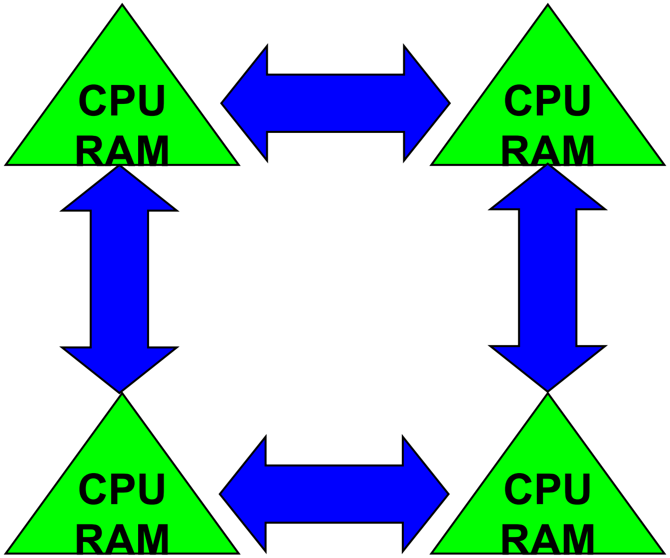
Communication: moving data between

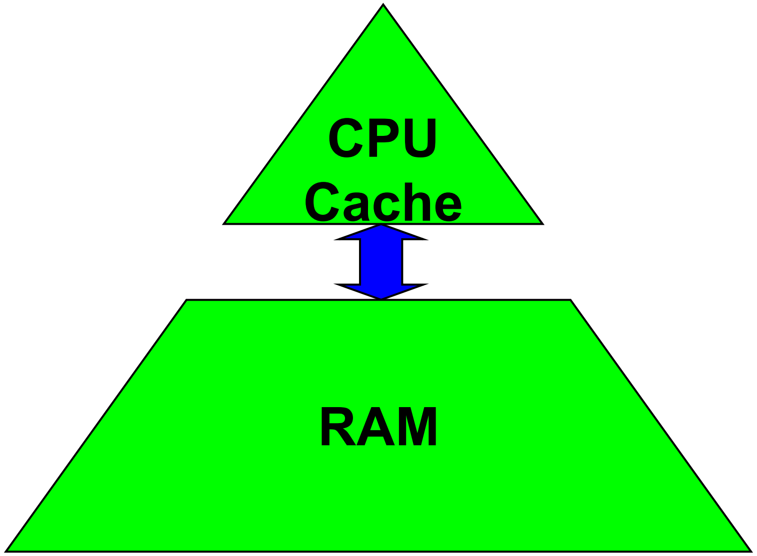
levels of a memory hierarchy (sequential case)

over a network connecting processors (parallel case)

Communication-minimizing algorithm:

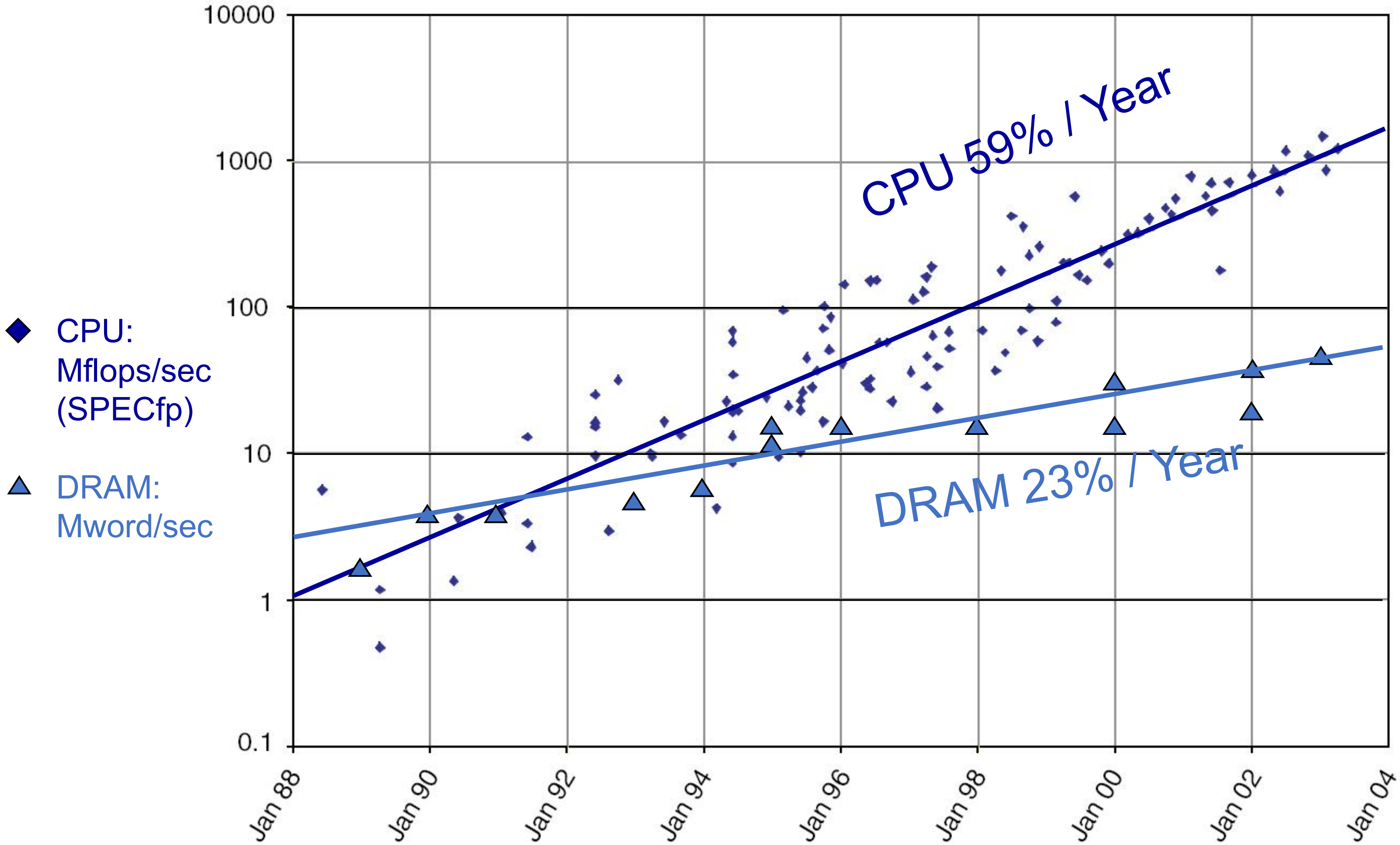
Save **time**, save **energy**.





MOORE'S LAW

Hardware trends: exponential growth with large gaps



Source: Graham, Snir and Patterson, eds.
Getting up to Speed: The Future of Supercomputing

PROGRESS ON I/O COMPLEXITY

Lower (and matching upper) bounds for:

BLAS, LU, Cholesky, LDL^T, and QR factorizations,
eigenvalues and singular values, i.e.,
essentially all direct methods of linear algebra.

Dense or sparse matrices
In sparse cases: BW a function of the actual FLOPs count.

Sequential, hierarchical, and parallel models

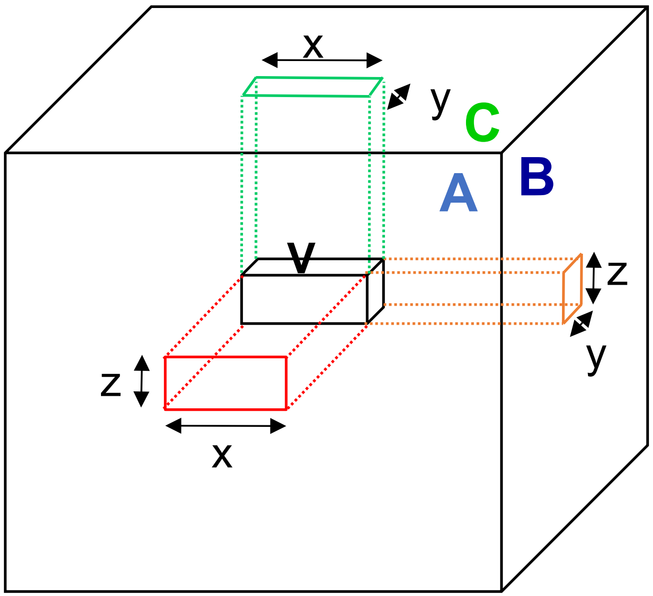
Bandwidth and latency

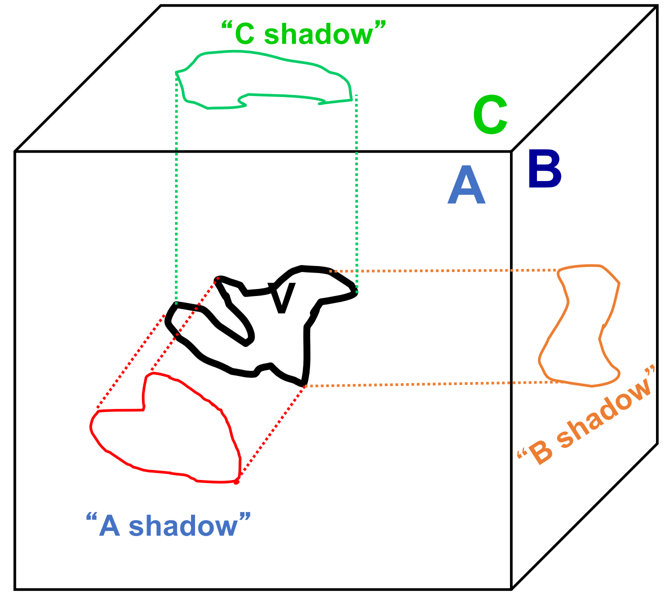
Compositions of linear algebra operations

Certain graph optimization problems

Fast matrix multiplication

GEOMETRIC INEQUALITIES USED





Volume of box

$V = x \cdot y \cdot z$

$= (xz \cdot zy \cdot yx)^{1/2}$

Thm: (**Loomis & Whitney**, 1949)

Volume of 3D set

$V \leq$ (area(A shadow)

$\cdot$ area(B shadow)

$\cdot$ area(C shadow)) $^{1/2}$

Geometric Embedding

Follows [Irony,Toledo,Tiskin 04], based on [Loomis & Whitney 49]

Matrix multiplication form:

$\forall (i,j) \in n \times n, \quad C(i,j) = \Sigma_k A(i,k)B(k,j),$

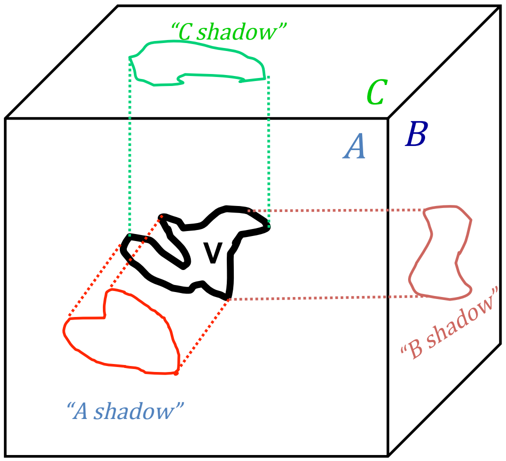
How many useful FLOPs can we perform with access to S inputs and outputs?

With O(M) inputs/outputs we can compute O($M^{3/2}$) FLOPs.

$\Rightarrow$ We have to perform $\Omega(M/M^{3/2})$ I/O per FLOP.

$\Rightarrow$

$$BW = \Omega\left(\frac{n^3}{M^{1/2}}\right)$$



Thm: (Loomis & Whitney, 1949)
Volume of 3D set

$V \leq (\text{area}(A\ \text{shadow}) \cdot \text{area}(B\ \text{shadow}) \cdot \text{area}(C\ \text{shadow}))^{1/2}$

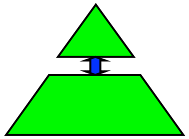
GEOMETRIC EMBEDDING

[Ballard, Demmel, H, Schwartz 2011a]
Follows [Irony,Toledo,Tiskin 04], based on [Loomis & Whitney 49]

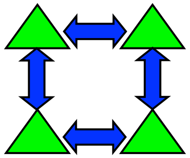
Generalized form:

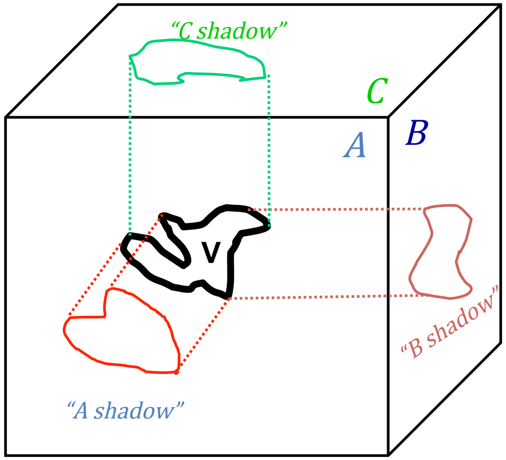
$$\forall (i,j) \in S, \quad C(i,j) = f_{i,j}(\quad g_{i,j,k_1}(A(i,k_1), B(k_1,j)),$$
$$g_{i,j,k_2}(A(i,k_2), B(k_2,j)),$$
$$\ldots, \qquad k_1, k_2, \ldots \in S_{i,j}$$
$$\text{other arguments})$$

$$\Omega\left(\frac{\#FLOPs}{M^{1/2}}\right)$$

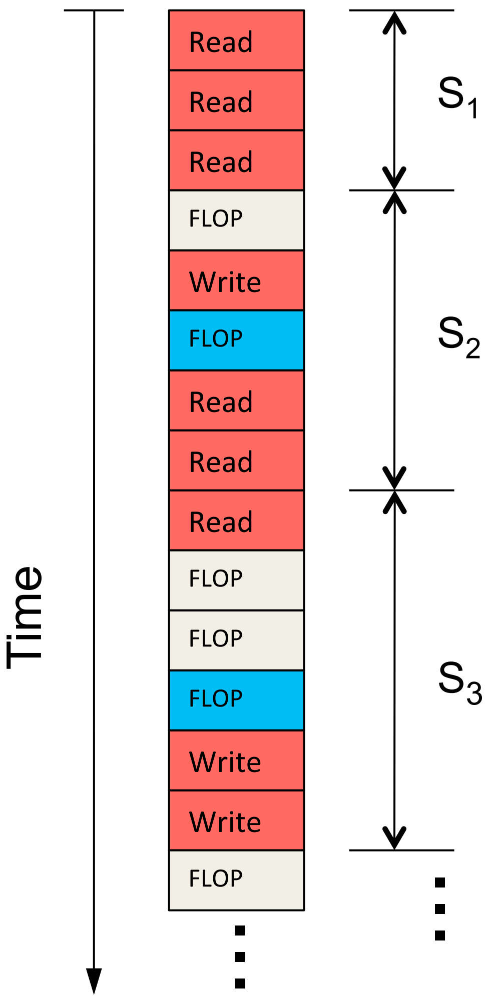


$$\Omega\left(\frac{\#FLOPs}{M^{1/2}} \cdot \frac{1}{P}\right)$$





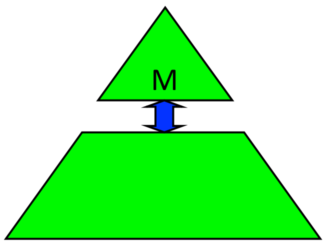
GEOMETRIC EMBEDDING



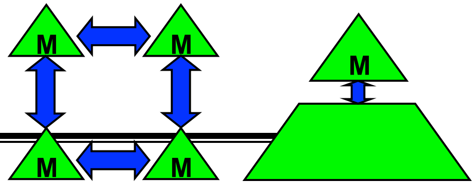
Example of a partition,
M = 3

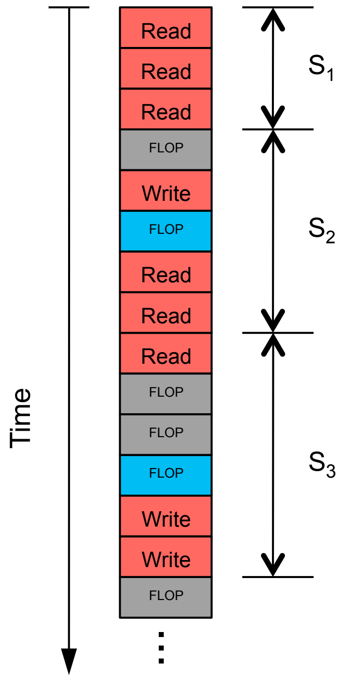
For a given run (algorithm, machine, input)

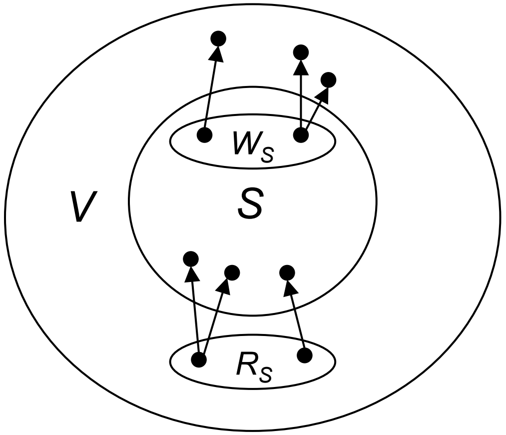
1. Partition computations into segments of M reads / writes
2. Any segment *S* has 3M inputs/outputs.
3. Show that #multiplications in S ≤ k
4. The total communication BW is
   BW = BW of one segment · #segments
   $\geq$ M · #mults / k = M · n^3 / k
5. By Loomis-Whitney:
   BW ≥ M · n^3 / $(3M)^{3/2}$



The partitioning argument







For a given run (Algorithm, Machine, Input)

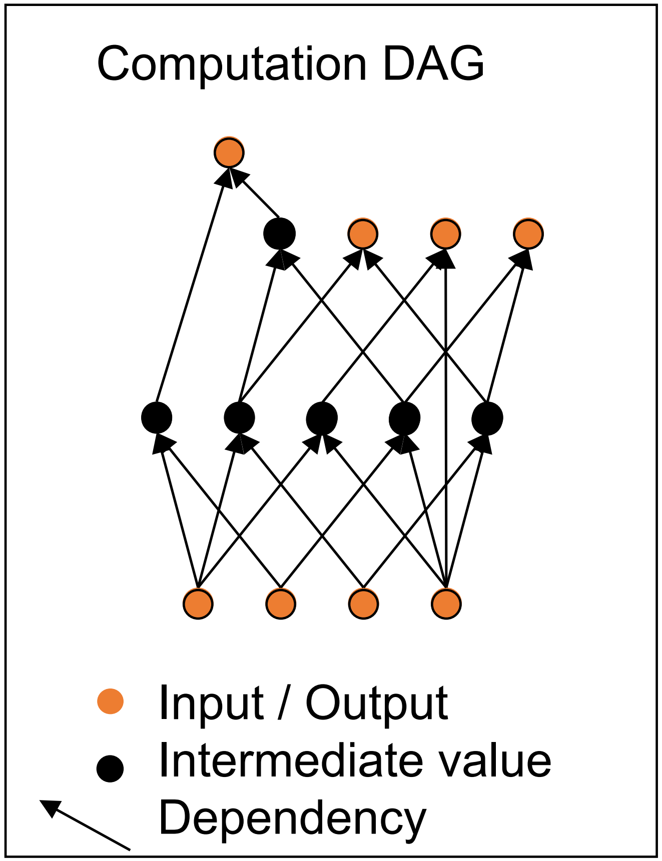
1. Consider the computation DAG: G = (V, E)
   V = set of computations and inputs
   E = dependencies

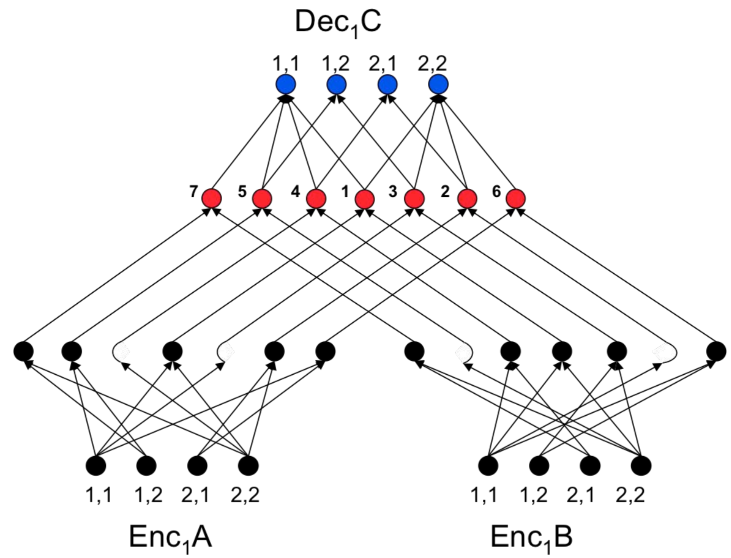
2. Partition G into segments S of $\Theta(M^{\omega/2})$ vertices
   (correspond to time / location adjacency)

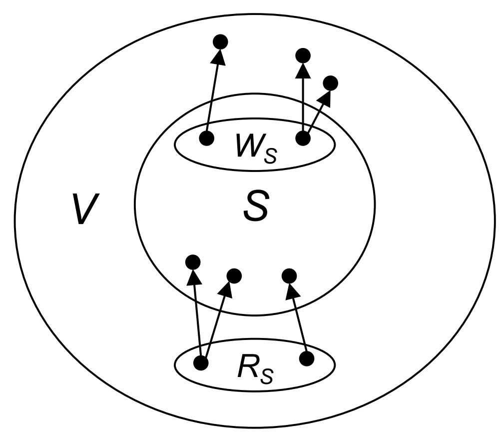
3. Show that every S has
   $\geq$ 3M vertices with incoming / outgoing edges
   $\Rightarrow$ perform $\geq$ M read/writes.

4. The total communication BW is
   BW = BW of one segment $\cdot$ #segments
   $= \Omega(M) \cdot \Theta(n^{\omega}) / \Theta(M^{\omega/2})$
   $= \Omega(n^{\omega} / M^{\omega/2-1})$

GRAPH EXPANSION



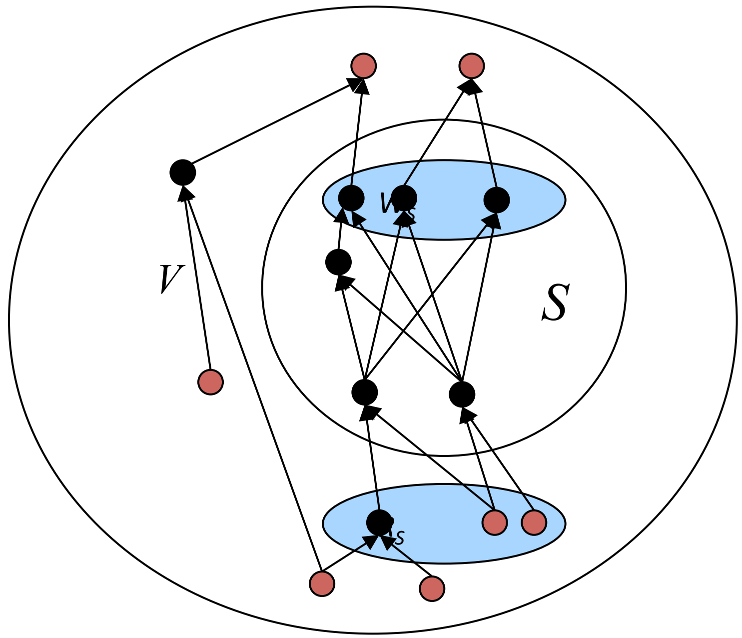


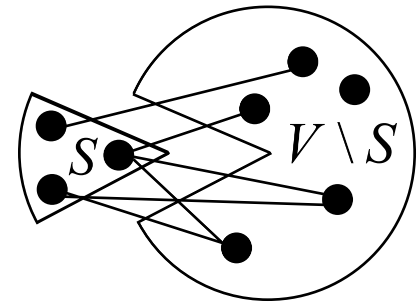


EXPANSION

The Computation Directed Acyclic Graph

- Input / Output
- Intermediate value
- Dependency



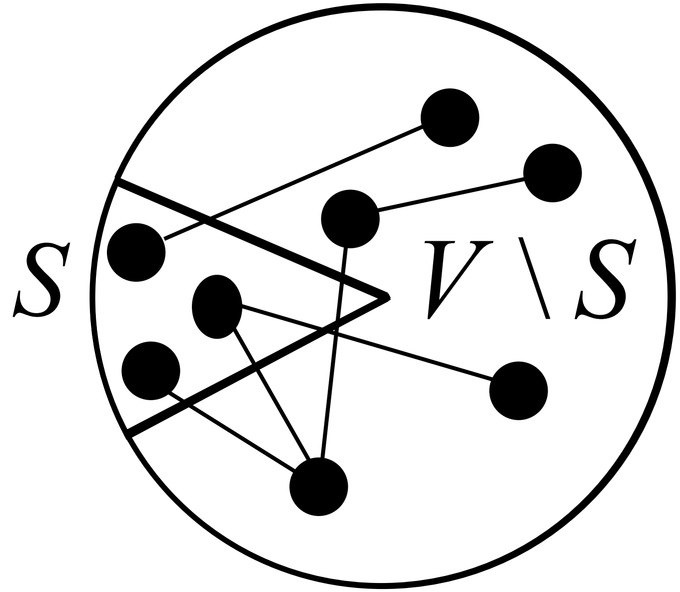


Communication Cost is

(Small-Sets) Graph Expansion

EXPANSION

[Ballard, Demmel, Holtz, S. 2011b], in the spirit of [Hong & Kung 81]



Let $G = (V,E)$ be a graph

$$h \equiv \min_{S, |S| \le \frac{|V|}{2}} \frac{\left|E\left(S, \overline{S}\right)\right|}{\left|E(S)\right|}$$

A is the normalized adjacency matrix

of a regular undirected graph, with eigenvalues:

$1 = \lambda_1 \ge \lambda_2 \ge \ldots \ge \lambda_n$

$\gamma \equiv 1 - max\,\{\lambda_{2,}\ |\lambda_n|\}$

Thm: [Alon-Milman84, Dodziuk84, Alon86]

$$\frac{1}{2}\gamma \le h \le \sqrt{2\gamma}$$

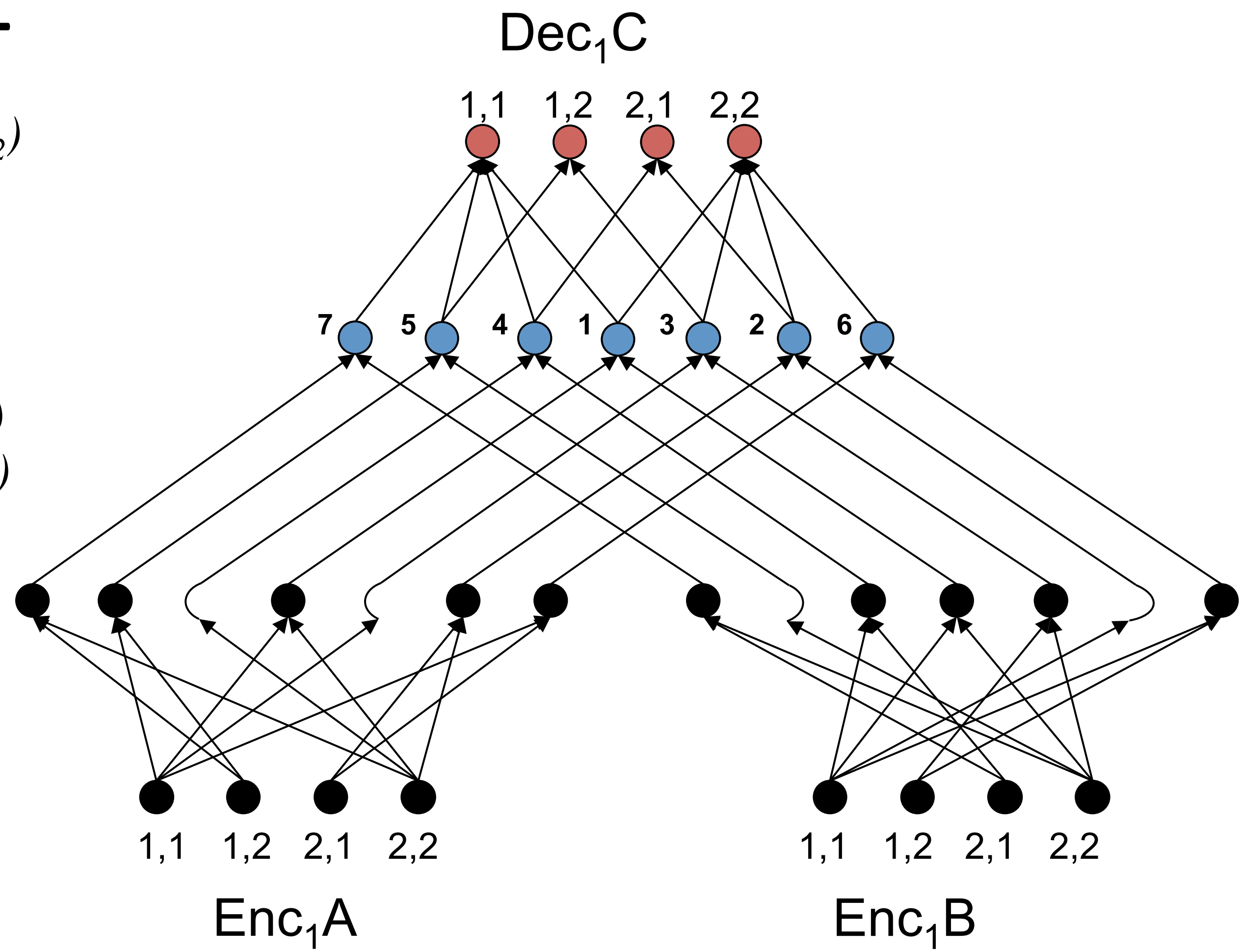
Small sets expansion:

$$h_s \equiv \min_{S, |S| \le s} \frac{\left|E\left(S, \overline{S}\right)\right|}{\left|E(S)\right|}$$

The DAG of Strassen, n = 2

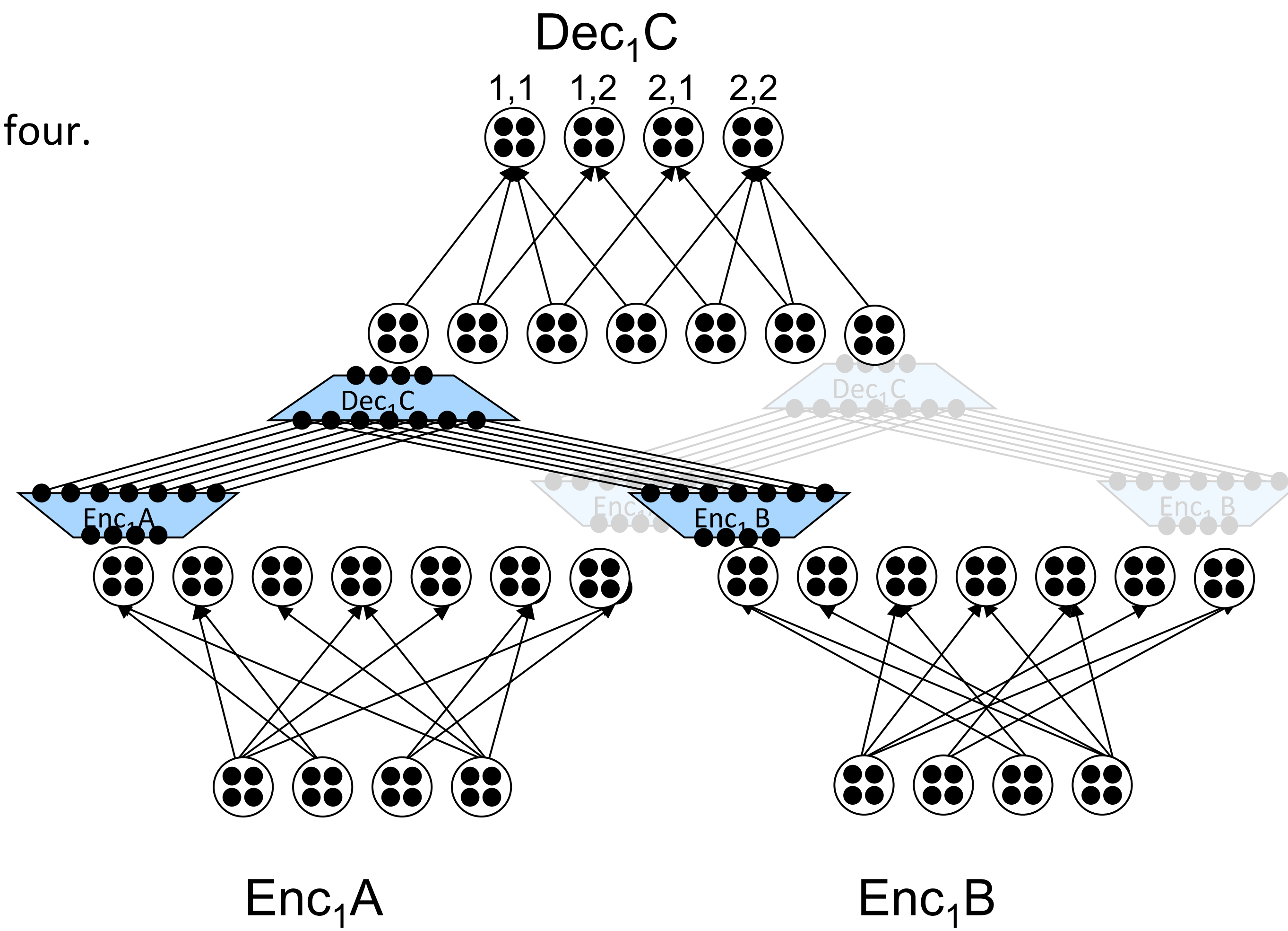
$M_1 = (A_{11} + A_{22}) \cdot (B_{11} + B_{22})$
$M_2 = (A_{21} + A_{22}) \cdot B_{11}$
$M_3 = A_{11} \cdot (B_{12} - B_{22})$
$M_4 = A_{22} \cdot (B_{21} - B_{11})$
$M_5 = (A_{11} + A_{12}) \cdot B_{22}$
$M_6 = (A_{21} - A_{11}) \cdot (B_{11} + B_{12})$
$M_7 = (A_{12} - A_{22}) \cdot (B_{21} + B_{22})$

$C_{11} = M_1 + M_4 - M_5 + M_7$
$C_{12} = M_3 + M_5$
$C_{21} = M_2 + M_4$
$C_{22} = M_1 - M_2 + M_3 + M_6$

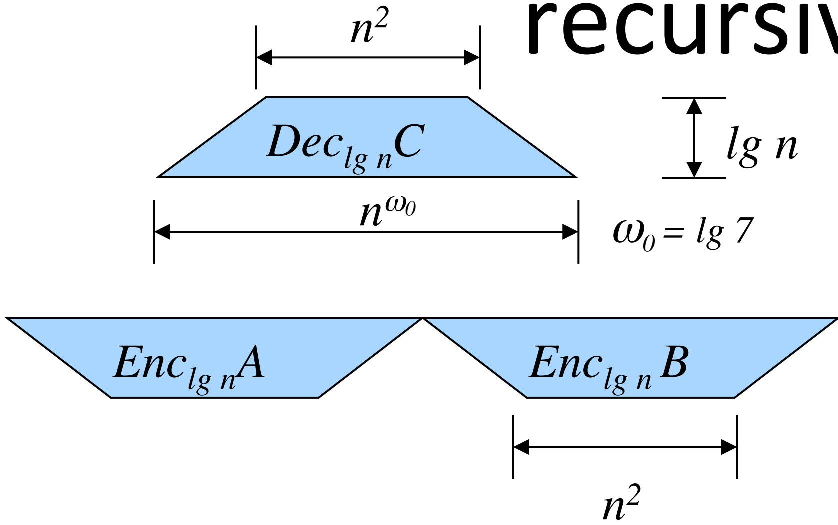


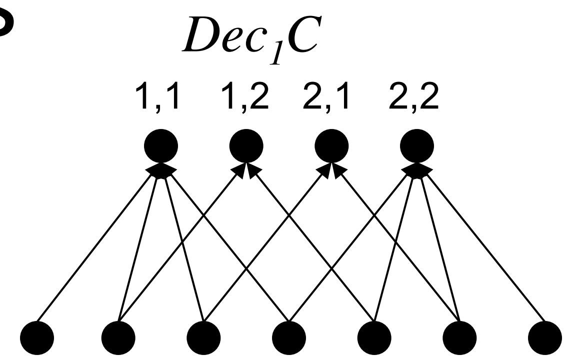
The DAG of Strassen, n=4

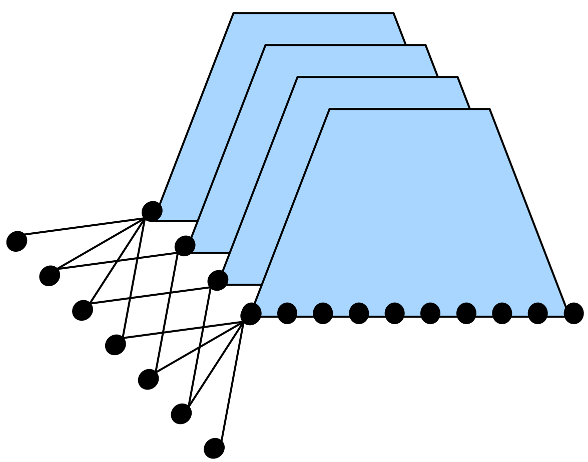
One recursive level:
- Each vertex splits into four.
- Multiply blocks



The DAG of Strassen: further recursive steps





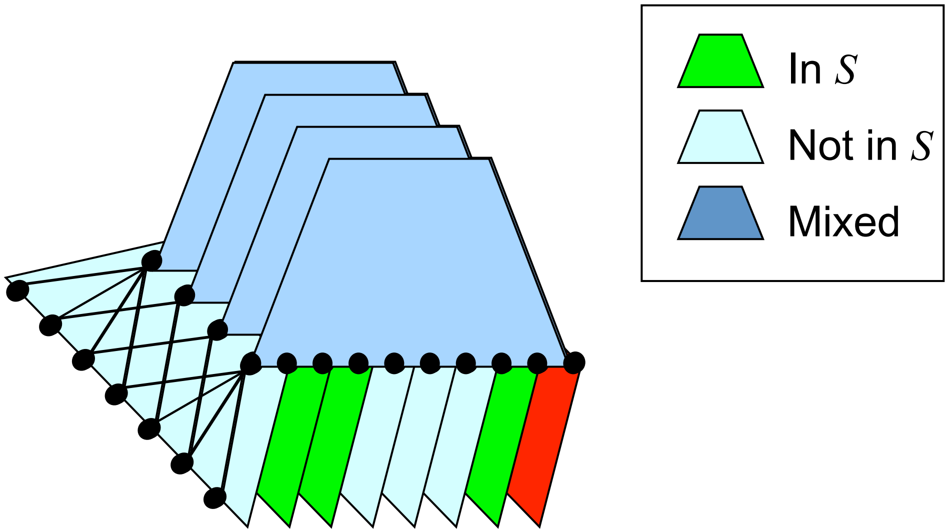


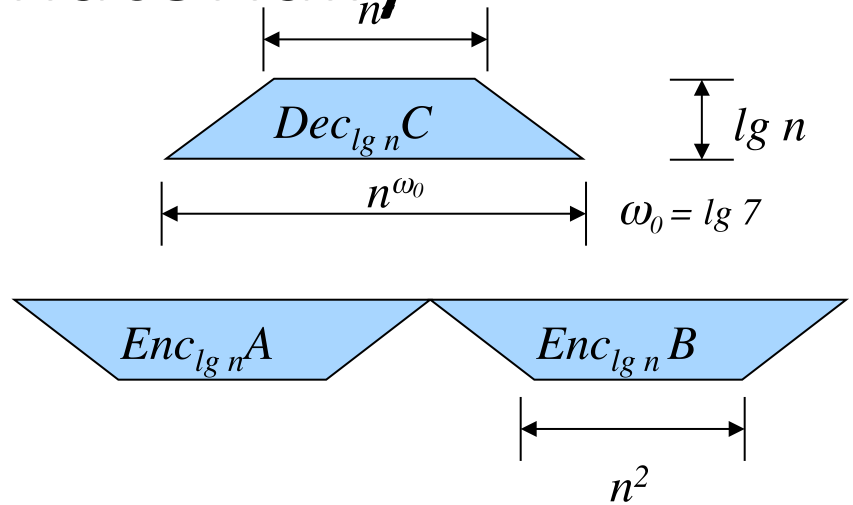
<u>Recursive construction</u>

Given Dec_iC, Construct $Dec_{i+1}C$:

1. Duplicate 4 times
2. Connect with a cross-layer of Dec_1C

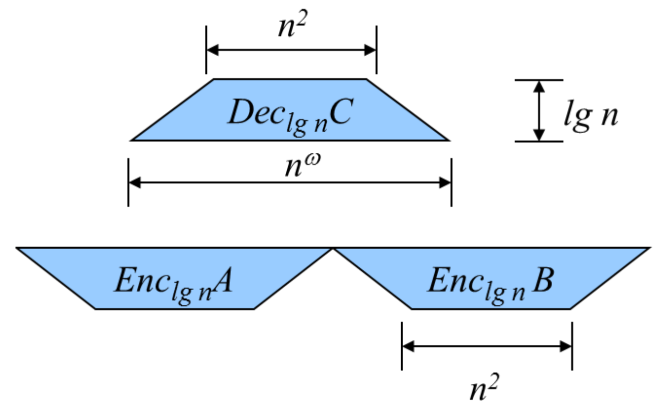
Estimating the edge expansion- Combinatorially





- Dec_1C is a consistency gadget: Mixed pays ≥ 1/12 of its edges.
- The fraction of S vertices is consistent between the 1st level and the four 2nd levels (deviations pay linearly).

Is Strassen's Graph a Good Expander?



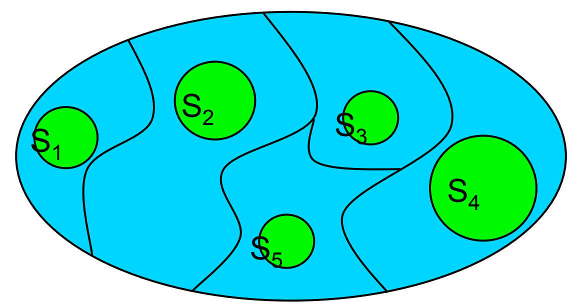
For n-by-n matrices:

$$h\left(Dec_{\lg n}C\right) = \Omega\left(\left(\frac{4}{7}\right)^{\lg n}\right) \qquad h \equiv \min_{S,|S|\le\frac{|V|}{2}} \frac{\left|E\left(S,\overline{S}\right)\right|}{\left|E\left(S\right)\right|}$$

For $M^{1/2}$-by-$M^{1/2}$ matrices:

$$h\left(Dec_{\lg\sqrt{M}}C\right) = \Omega\left(\left(\frac{4}{7}\right)^{\lg\sqrt{M}}\right) = \Omega\left(\frac{M}{M^{\omega_0/2}}\right), \qquad \omega_0 = \lg 7$$

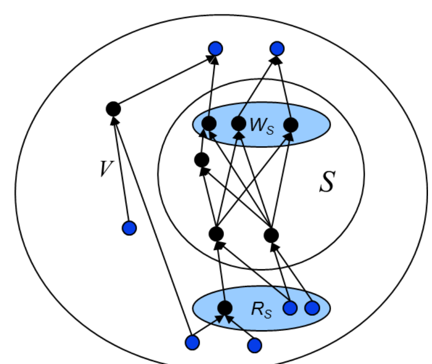
For $M^{1/2}$-by-$M^{1/2}$ sub-matrices
(or other small subsets):



$$h_{M^{\omega_0/2}}\left(Dec_{\lg n}C\right) = \Omega\left(\frac{M}{M^{\omega_0/2}}\right) \qquad h_s \equiv \min_{S,|S|\le s} \frac{\left|E\left(S,\overline{S}\right)\right|}{\left|E\left(S\right)\right|}$$

Summing up (the partition argument)

$$BW = \Omega\left(n^{\omega_0}\frac{M}{M^{\omega_0/2}}\right)$$



$\blacksquare$

ADVERTISEMENT

Organizers: Peter Bürgisser (TU Berlin), Olga Holtz (UC Berkeley), Daniel Kressner (EPFL), J.M. Landsberg (Texas A&M), Oded Schwartz (Hebrew U Jerusalem), Nikhil Srivastava (UC Berkeley).

SIMONS INSTITUTE
for the Theory of Computing

Home

Complexity and Linear Algebra

Tuesday, Sept. 2 – Friday, Dec. 12, 2025

THANK YOU!

QUESTIONS?