



中山大學
SUN YAT-SEN UNIVERSITY



多核程序设计与实践

应用举例：矩阵乘法

陶钧

taoj23@mail.sysu.edu.cn

中山大学 数据科学与计算机学院
国家超级计算广州中心

- 矩阵乘法意义
- 矩阵乘法计算回顾
- 矩阵乘法分析
- 分治法 (Strassen's method)
- 稀疏矩阵与向量相乘



- 矩阵是各种实际应用中十分常见的表现形式

- 邻接矩阵、拉普拉斯算子等
- 邻接矩阵相乘应用举例：

- 假设A, B, C, D, E, F六地之间的邻接关系如以下矩阵M所示，求从X地出发，经过k步到Y地的路线数目($X, Y \in \{A, B, C, D, E, F\}$) ?

M =

	A	B	C	D	E	F
A	0	0	1	0	1	0
B	0	0	0	1	1	1
C	1	0	0	1	0	1
D	0	1	1	0	0	0
E	1	1	0	0	0	1
F	0	1	1	0	1	0

- 矩阵是各种实际应用中十分常见的表现形式

 - X地出发， 经过k步到Y地的路线数目为 $M_{X,Y}^k$

 - 例如， k=2， X=C， Y=B时：

 - M中行C代表从C地出发到每地的路线数目，列B代表从每地出发到B地的路线数目
 - 行C乘列B代表从C地出发经每地到B地的路径数目（可能存在环，是否能消除？）

$M =$

	A	B	C	D	E	F
A	0	0	1	0	1	0
B	0	0	0	1	1	1
C	1	0	0	1	0	1
D	0	1	1	0	0	0
E	1	1	0	0	0	1
F	0	1	1	0	1	0

$M^2 =$

	A	B	C	D	E	F
A						
B						
C						
D						
E						
F						

$$\begin{matrix} 1 & 0 & 0 & 1 & 0 & 1 \end{matrix} \cdot \begin{matrix} 0 & 0 & 0 & 1 & 1 & 1 \end{matrix} = 2$$

● 矩阵是各种实际应用中十分常见的表现形式

– X地出发， 经过k步到Y地的路线数目为 $M_{X,Y}^k$

- 例如， $k=2$, $X=C$, $Y=B$ 时：

- M中行C代表从C地出发到每地的路线数目，列B代表从每地出发到B地的路线数目
- 行C乘列B代表从C地出发经每地到B地的路径数目（可能存在环，是否能消除？）

» 不能消除环（如果 $NP \neq P$ ）

» 考虑哈密顿回路问题（NP）：

- 是否存在路径， 经过所有顶点一次且仅一次回到出发顶点
- 假设能通过修改矩阵，而使得在路径中消除环
- 设 $k=n$, $X=Y=A$, 若矩阵 $M_{A,A}^n = 0$ 意味着路径不存在，否则路径存在
- 意味着 $O(n^4)$ 时间内能对哈密顿回路问题求解



- 矩阵乘法更常见的例子是代表一个线性变换

- 如，深度学习中常见 $y=Wx+b$ ，其中W为权重矩阵
- 例如，以下例子将直角坐标 (x, y, z) 变换到以 $(1, 0, 1)$ ， $(0, 1, 0)$ 及 $(0, 1, 1)$ 为轴的坐标系中
- 同理，矩阵A、B相乘可视作映射的复合
 - 将矩阵B的列向量空间的向量基分别置于矩阵A的列向量空间中

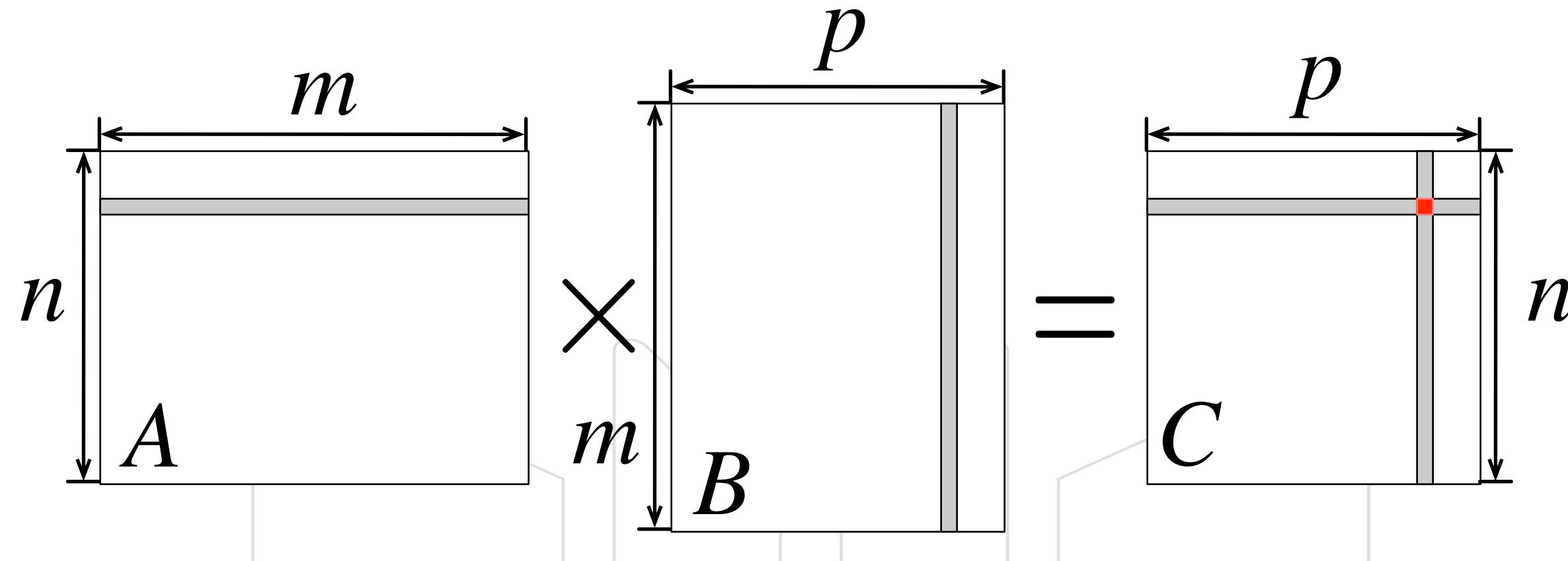
$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} 0 \\ 1 \\ 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} x \\ y+z \\ x+z \end{pmatrix}$$

- 矩阵乘法意义
- 矩阵乘法计算回顾
- 矩阵乘法分析
- 分治法 (Strassen's method)
- 稀疏矩阵与向量相乘



● 定义

- 对C中所有元素，计算 $C_{ij} = \sum_{k=1}^m A_{ik} \cdot B_{kj}$
 - 共有 $n \times p$ 次向量乘积
 - 每次向量乘积为 m 次乘法与 $m - 1$ 次加法



● CPU性能分析

- 访问连续内存的重要性
 - C/C++中，数组为row-major
- 交换循环顺序能获得大幅性能提升
 - 但此方法却很难直接应用于CUDA上（为什么？）

```
#define N 3000
int A[N][N], B[N][N], C[N][N];

for(int i=0; i<N; ++i)
    for(int j = 0; j<N; ++j)
        for(int k = 0; k < N; ++k)
            C[i][j] += A[i][k]*B[k][j];
```

运行时间：231.348秒

```
#define N 3000
int A[N][N], B[N][N], C[N][N];

for(int i=0; i<N; ++i)
    for(int k = 0; k<N; ++k)
        for(int j = 0; j<N; ++j)
            C[i][j] += A[i][k]*B[k][j];
```

运行时间：72.557秒

● 交换顺序后

- 假设每个线程处理最内层循环
 - 为什么不每个线程处理一次乘法?
 - 回顾Brent's theorem (讲义8)
 - 每个线程将处理N次乘法，并进行N次加法
 - N次加法作用于N个不同的数字！（需要原子操作！）
 - 而不交换顺序虽然对B的访问模式并非最优，但却不需要原子操作

```
#define N 3000
int A[N][N], B[N][N], C[N][N];

for(int i=0; i<N; ++i)
    for(int j = 0; j<N; ++j)
        for(int k = 0; k < N; ++k)
            C[i][j] += A[i][k]*B[k][j];
```

运行时间：231.348秒

```
#define N 3000
int A[N][N], B[N][N], C[N][N];

for(int i=0; i<N; ++i)
    for(int k = 0; k<N; ++k)
        for(int j = 0; j<N; ++j)
            C[i][j] += A[i][k]*B[k][j];
```

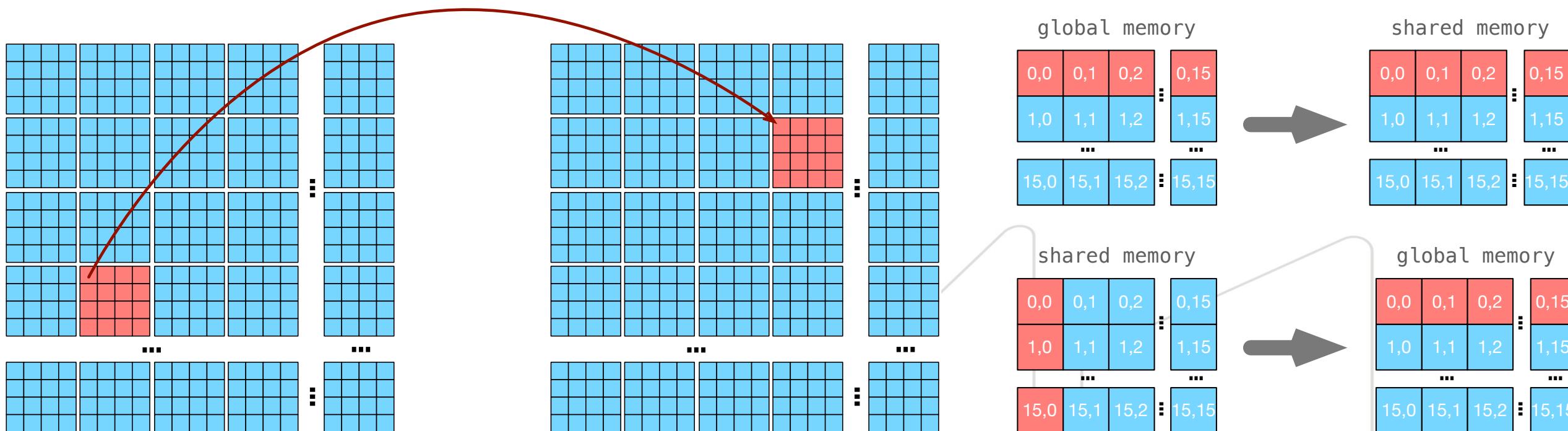
运行时间：72.557秒

- 矩阵乘法意义
- 矩阵乘法计算回顾
- 矩阵乘法分析
- 分治法 (Strassen's method)
- 稀疏矩阵与向量相乘



● 分块矩阵乘法

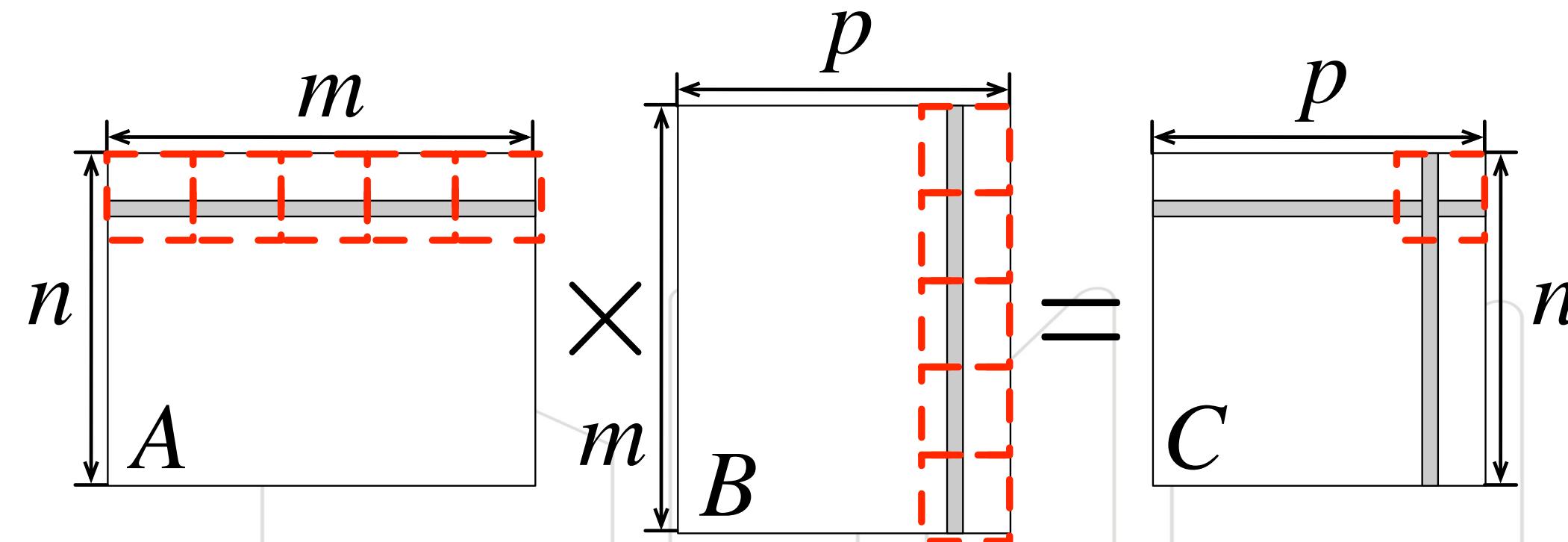
- 将矩阵分块后相乘
- 类比矩阵转置（讲义8）
 - 分块处理转置
 - 块内在共享内存中使用column-major的访问方式
 - 此时关键因素为消除存储体冲突而非合并访问



● 分块矩阵乘法

- 将矩阵分块后相乘
- 此时计算C中一个方形子矩阵的值为

$$C_{i,j} = \sum_{k=1}^{m/\text{BDIM}} A_{i,k} \times B_{k,j}$$



● 分块矩阵乘法

- 将矩阵分块后相乘
- 此时计算C中一个方形子矩阵的值为

- 例如，将 4×4 的矩阵A与B各分为四块进行矩阵乘法

$$A = \begin{pmatrix} 1 & 2 & 3 & 4 \\ 4 & 1 & 2 & 3 \\ 3 & 4 & 1 & 2 \\ 2 & 3 & 4 & 1 \end{pmatrix}$$

$$B = \frac{1}{40} \begin{pmatrix} -9 & 11 & 1 & 1 \\ 1 & -9 & 11 & 1 \\ 1 & 1 & -9 & 11 \\ 11 & 1 & 1 & -9 \end{pmatrix}$$

$$C = AB$$

$$A = \begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix} \quad A_{11} = \begin{pmatrix} 1 & 2 \\ 4 & 1 \end{pmatrix}, A_{12} = \begin{pmatrix} 3 & 4 \\ 2 & 3 \end{pmatrix}, A_{21} = \begin{pmatrix} 3 & 4 \\ 2 & 3 \end{pmatrix}, A_{22} = \begin{pmatrix} 1 & 2 \\ 4 & 1 \end{pmatrix}$$

$$B = \frac{1}{40} \begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} \quad B_{11} = \begin{pmatrix} -9 & 11 \\ 1 & -9 \end{pmatrix}, B_{12} = \begin{pmatrix} 1 & 1 \\ 11 & 1 \end{pmatrix}, B_{21} = \begin{pmatrix} 1 & 1 \\ 11 & 1 \end{pmatrix}, B_{22} = \begin{pmatrix} -9 & 11 \\ 1 & -9 \end{pmatrix}$$

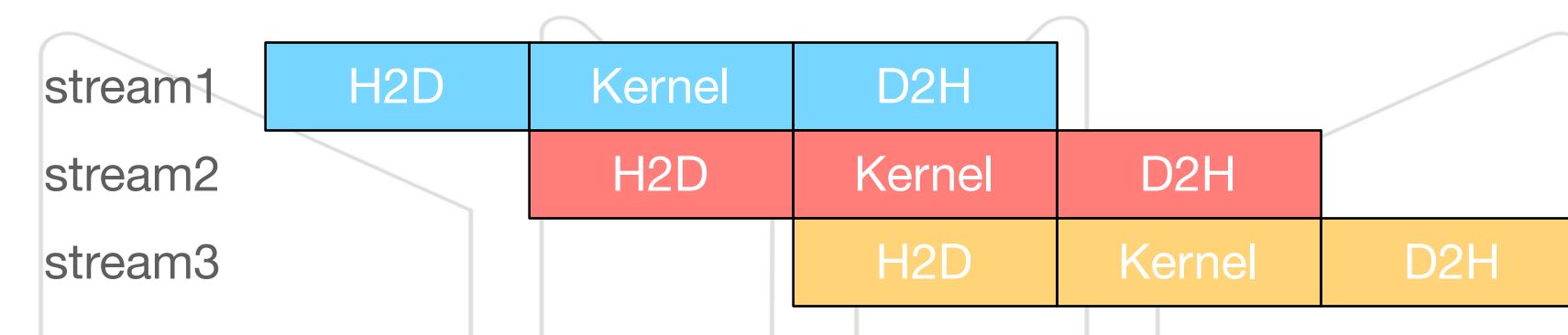
$$C = \begin{pmatrix} C_{11} & C_{12} \\ C_{21} & C_{22} \end{pmatrix}$$

$$C_{i,j} = \sum_{k=1}^{m/\text{BDIM}} A_{i,k} \times B_{k,j}$$

$$\begin{aligned} C_{11} &= A_{11}B_{11} + A_{12}B_{21} \\ &= \frac{1}{40} \begin{pmatrix} 1 & 2 \\ 4 & 1 \end{pmatrix} \begin{pmatrix} -9 & 11 \\ 1 & -9 \end{pmatrix} + \frac{1}{40} \begin{pmatrix} 3 & 4 \\ 2 & 3 \end{pmatrix} \begin{pmatrix} 1 & 1 \\ 11 & 1 \end{pmatrix} \\ &= \frac{1}{40} \begin{pmatrix} -9+2 & 11-18 \\ -36+1 & 44-9 \end{pmatrix} + \frac{1}{40} \begin{pmatrix} 3+44 & 3+4 \\ 2+33 & 2+3 \end{pmatrix} \\ &= \frac{1}{40} \begin{pmatrix} -7 & -7 \\ -35 & 35 \end{pmatrix} + \frac{1}{40} \begin{pmatrix} 47 & 7 \\ 35 & 5 \end{pmatrix} \\ &= \frac{1}{40} \begin{pmatrix} 40 & 0 \\ 0 & 40 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \end{aligned}$$

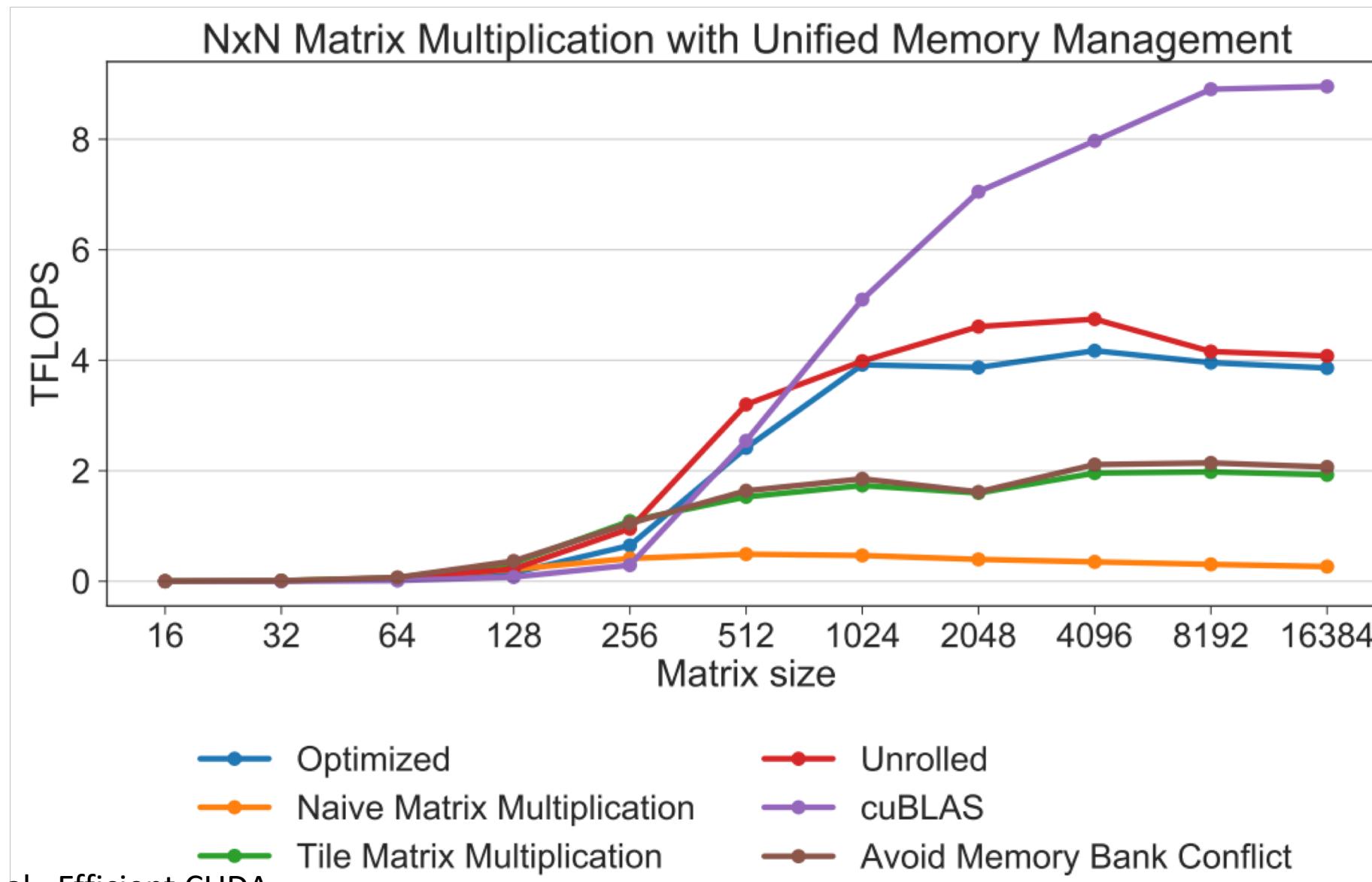
● 分块矩阵乘法

- 将矩阵分块后相乘
- 每个线程块可以用于两块矩阵的乘积
 - 每个线程计算一个C矩阵块中一个元素的值（块内无需使用原子操作）
 - 块间需要原子操作更新
- 可用于处理大规模矩阵
 - 多次被一次载入显存
- 便于使用CUDA流分多次处理数据
 - 更好地重叠计算与内存拷贝的时间



● 分块矩阵乘法

— 性能



图片来自C. Coleman et al., Efficient CUDA

- 矩阵乘法意义
- 矩阵乘法计算回顾
- 矩阵乘法分析
- 分治法 (Strassen's method)
- 稀疏矩阵与向量相乘



- Divide and conquer

- 直接思路：将每个矩阵分为4份
 - 在计算子矩阵乘法时，继续使用分治法将其分为4份
 - 如，计算ae时，将a和e各分为4份
 - 时间复杂度 $T(n) = 8T\left(\frac{n}{2}\right) + O(n^2) = O(n^3)$
 - 总计算量完全一致（参考分块计算）

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} ae+bg & af+bh \\ ce+dg & cf+dh \end{pmatrix}$$

A B C

● Strassen's method

– 仍然将矩阵A与B分别分为4块

- 如下计算p1, p2, ..., p7
- 使用p1, p2, ..., p7更新C中4块子矩阵的值
- **好处是?**

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} p_5+p_4-p_2+p_6 & p_1+p_2 \\ p_3+p_4 & p_1+p_5-p_3-p_7 \end{pmatrix}$$

A B C

$$p_1 = a(f-h)$$

$$p_2 = (a+b)h$$

$$p_3 = (c+d)e$$

$$p_4 = d(g-e)$$

$$p_5 = (a+d)(e+h)$$

$$p_6 = (b-d)(g+h)$$

$$p_7 = (a-c)(e+f)$$

● Strassen's method

– 仍然将矩阵A与B分别分为4块

- 验证: $p_5 + p_4 - p_2 + p_6 = (a+d)(e+h) + d(g-e) - (a+b)h + (b-d)(g+h)$
 $= ae + ah + de + dh + dg - de - ah - bh + bg + bh - dg - dh$
 $= ae + bg$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} e & f \\ g & h \end{pmatrix} = \begin{pmatrix} p_5 + p_4 - p_2 + p_6 & p_1 + p_2 \\ p_3 + p_4 & p_1 + p_5 - p_3 - p_7 \end{pmatrix}$$

A B C

$$p_1 = a(f-h)$$

$$p_2 = (a+b)h$$

$$p_3 = (c+d)e$$

$$p_4 = d(g-e)$$

$$p_5 = (a+d)(e+h)$$

$$p_6 = (b-d)(g+h)$$

$$p_7 = (a-c)(e+f)$$

● Strassen's method

– 效率分析：7次矩阵乘法与18次矩阵加法

- 加法效率比乘法更高

- 减少递归分支

- 时间复杂度： $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) = O(n^{\log_2 7}) = O(n^{2.8074})$

- 复杂度推导：master theorem

- $- T(n) = aT\left(\frac{n}{b}\right) + f(n)$

- $-$ 如果 $f(n) = O(n^{\log_b a - \varepsilon})$ 且 $\varepsilon > 0$, 则 $T(n) = O(n^{\log_b a})$



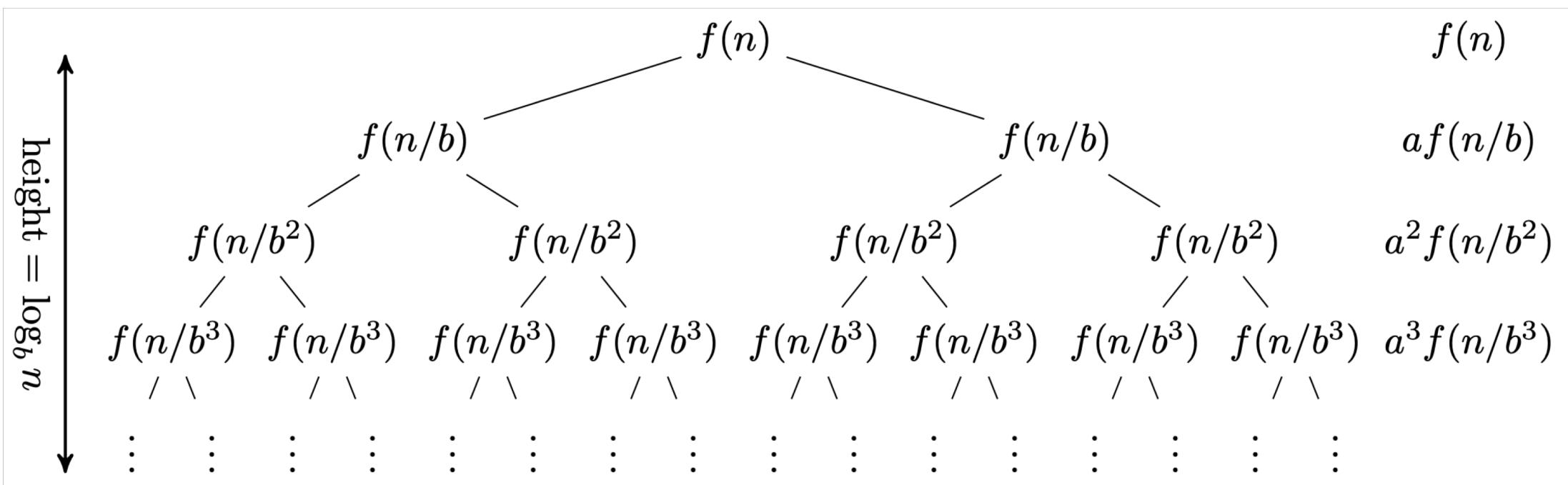
- Master theorem

- 将 $T(n) = aT\left(\frac{n}{b}\right) + f(n)$ 沿树状展开可得

- $T(n) = \sum_{i=0}^{\log_b a} a^i f\left(\frac{n}{b^i}\right) + O(n^{\log_b a})$

- $\sum_{i=0}^{\log_b a} a^i f\left(\frac{n}{b^i}\right)$ 为内部节点开销

- $O(n^{\log_b a})$ 为叶子节点开销: $a^{\log_b n} f(1) = n^{\log_b a} f(1)$



● Master Theorem

– 在此，只考慮 $\log_b a > \varepsilon$ ，有：

$$T(n) = \sum_{i=0}^{\log_b n} a^i f(n/b^i) + O(n^{\log_b a}) \leq \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} + O(n^{\log_b a})$$

$$\begin{aligned} \sum_{i=0}^{\log_b n} a^i (n/b^i)^{\log_b a - \varepsilon} &= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i b^{-i \log_b a} b^{i\varepsilon} = n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} a^i a^{-i} b^{i\varepsilon} \\ &= n^{\log_b a - \varepsilon} \sum_{i=0}^{\log_b n} b^{\varepsilon i} = n^{\log_b a - \varepsilon} \frac{b^{\varepsilon(\log_b n + 1)} - 1}{b^\varepsilon - 1} \\ &= n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon - 1}{b^\varepsilon - 1} \leq n^{\log_b a - \varepsilon} \frac{n^\varepsilon b^\varepsilon}{b^\varepsilon - 1} = n^{\log_b a} \frac{b^\varepsilon}{b^\varepsilon - 1} \\ &= O(n^{\log_b a}). \end{aligned}$$

● Strassen's method

– 根据Master theorem

- 使用Strassen's method: $T(n) = 7T\left(\frac{n}{2}\right) + O(n^2) = O(n^{2.8074})$

- 使用直接思路: $T(n) = 8T\left(\frac{n}{2}\right) + O(n^2) = O(n^3)$

– Strassen's method比直接思路复杂度更低

- 但常数更高，因此，当n较小时，仍然是直接思路更快

- CUDA实现更为复杂

- 逻辑复杂，中间变量更多

- 混合策略？

- 对于n较大时，使用Strassen's method

- 当递归至n较小时，直接计算

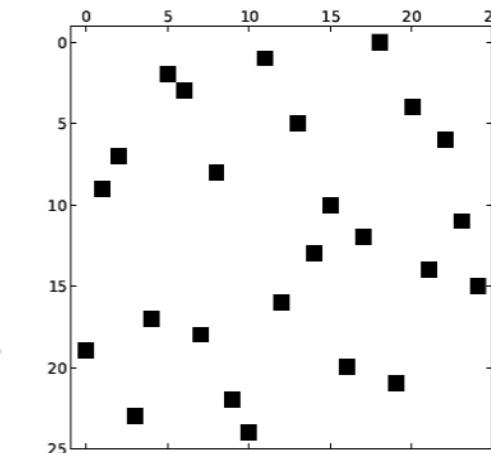
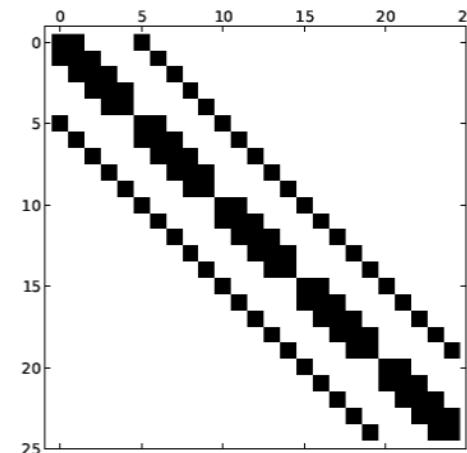
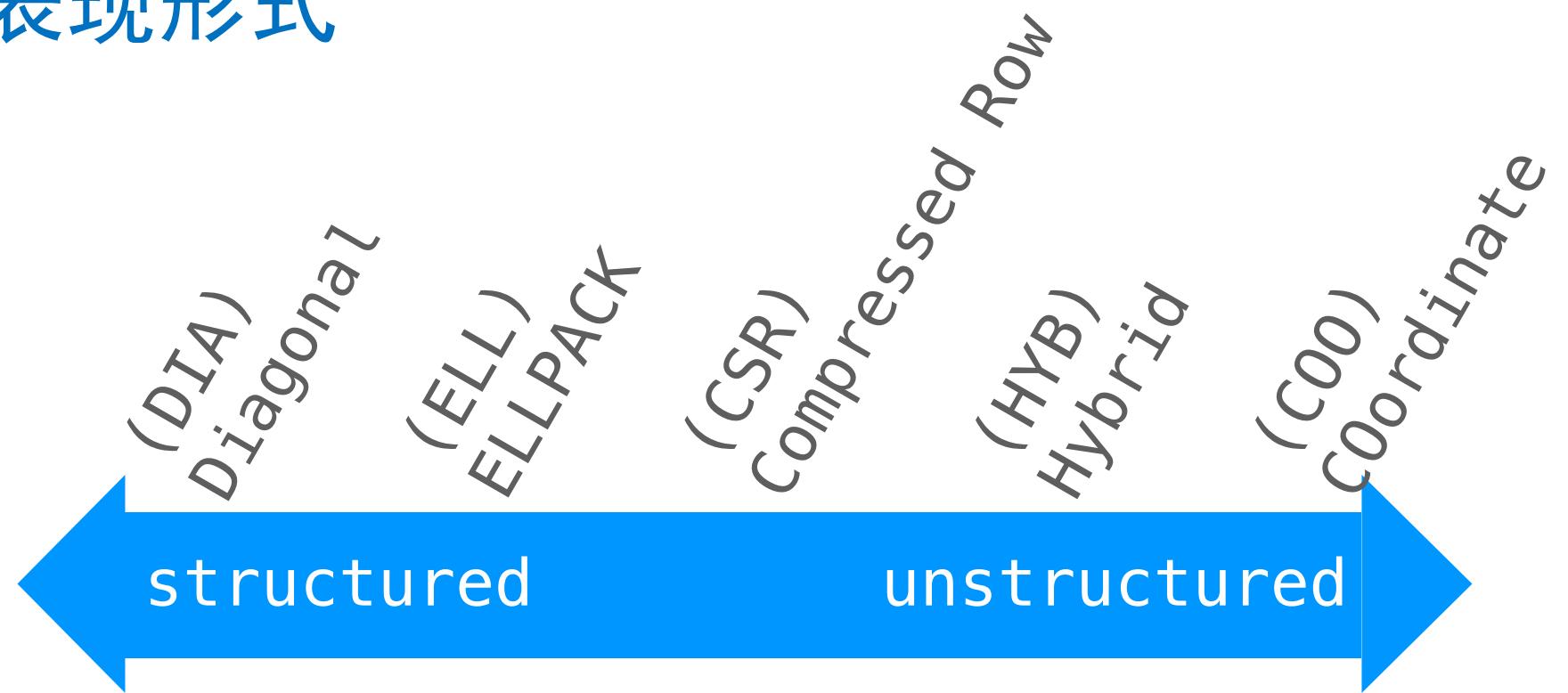
- 矩阵乘法意义
- 矩阵乘法计算回顾
- 矩阵乘法分析
- 分治法 (Strassen's method)
- 稀疏矩阵与向量相乘



- 以上讨论的都是稠密矩阵，而现实应用中，许多矩阵都是稀疏的
 - 稀疏：数值为0的元素远多于非0元素
 - 如，路网的邻接矩阵表示形式
 - 通常，任意路口仅与至多5条道路相连
 - 社交网络
 - 每个用户的好友数相对于网络规模而言很小
 - 通常是不规则及非结构化的 (irregular & unstructured)
 - 使用稠密矩阵乘法进行计算将引入大量无意义运算 (乘0)



- 稀疏矩阵表现形式



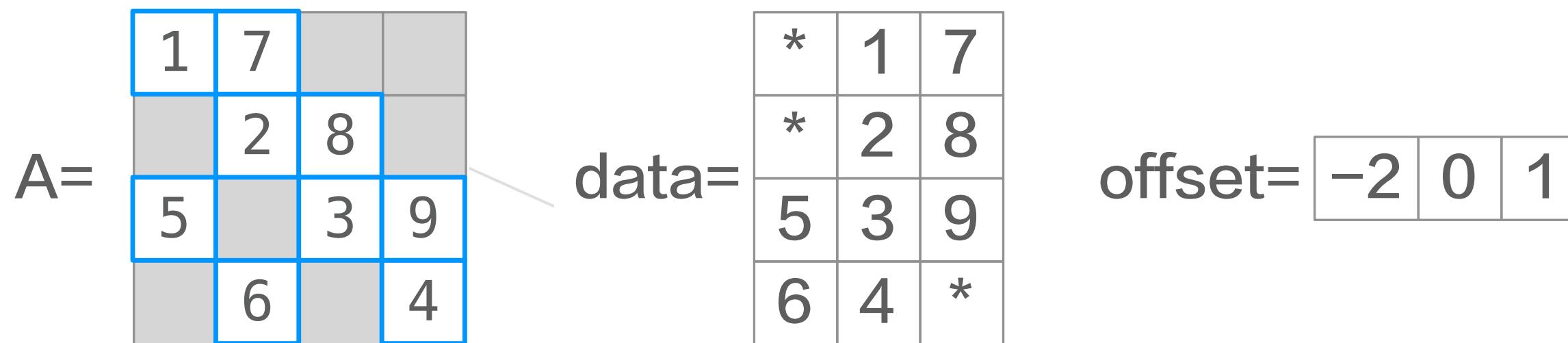
● 稀疏矩阵表现形式 (DIA)

- 基于对角线的表现形式

- offset代表data中每列元素距离对角线的偏移量
- 矩阵中位于同一行的元素仍然在同一行
- 需要矩阵满足沿对角线分布的形式

- 并行方案

- 每个线程处理data中的一行（按column-major方式存储为合并访问）



● 稀疏矩阵表现形式 (ELL)

- 假设每行最大非0元素个数较小

- data中每行为原矩阵中非0值（不足部分以*标记）
- indices中记录每个元素在原矩阵中的列信息
- 当某行非0元素远比其他行多时，有效数据比例低

- 并行方案

- 每个线程块处理data中的一行（按column-major方式存储为合并访问）

1	7		
	2	8	
5		3	9
	6		4

A=

data=

1	7	*
2	8	*
5	3	9
6	4	*

indices=

0	1	*
1	2	*
0	2	3
1	3	*

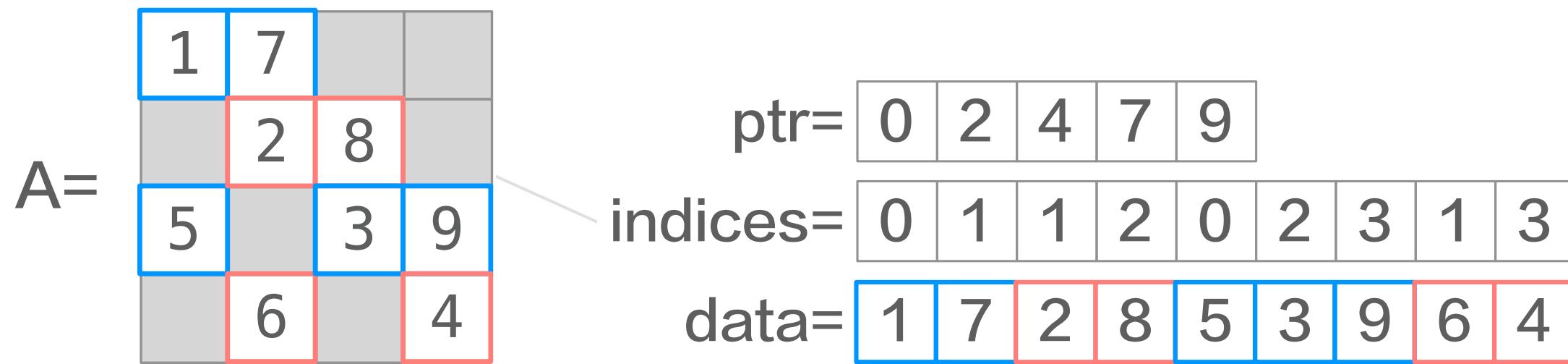
- 稀疏矩阵表现形式 (CSR)

- 行压缩

- 将非0元素置data之中
 - ptr记录每行在data中的开始位置
 - indices记录每个元素在原始矩阵中的列信息

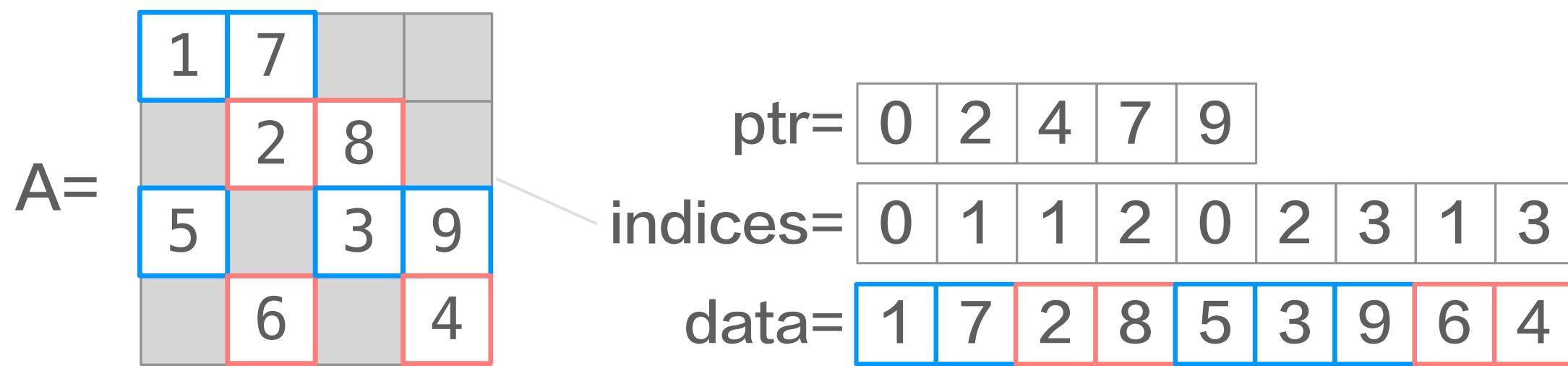
- 优点：对矩阵特性无要求

- 并行方案：每个线程处理一行数据（**问题？**）



● 稀疏矩阵表现形式 (CSR)

- 并行方案1：每个线程处理一行数据（**问题？**）
 - 线程工作量不均匀
 - 非合并非对齐访问
- 并行方案2：每个线程束处理一行数据（**问题？**）
 - 线程束工作量不均匀
 - 合并访问、但非对齐访问



● 稀疏矩阵表现形式 (COO)

- 记录所有非0元素坐标 (行、列)
 - row记录每个非0元素的行信息
 - indices与data与CSR一致
- 优点：对矩阵无要求（但占用内存可能较多）
- 并行策略
 - 每个线程处理一个非0元素

A=

1	7		
	2	8	
5		3	9
	6		4

row= 0 0 1 1 2 2 2 3 3

indices= 0 1 1 2 0 2 3 1 3

data= 1 7 2 8 5 3 9 6 4

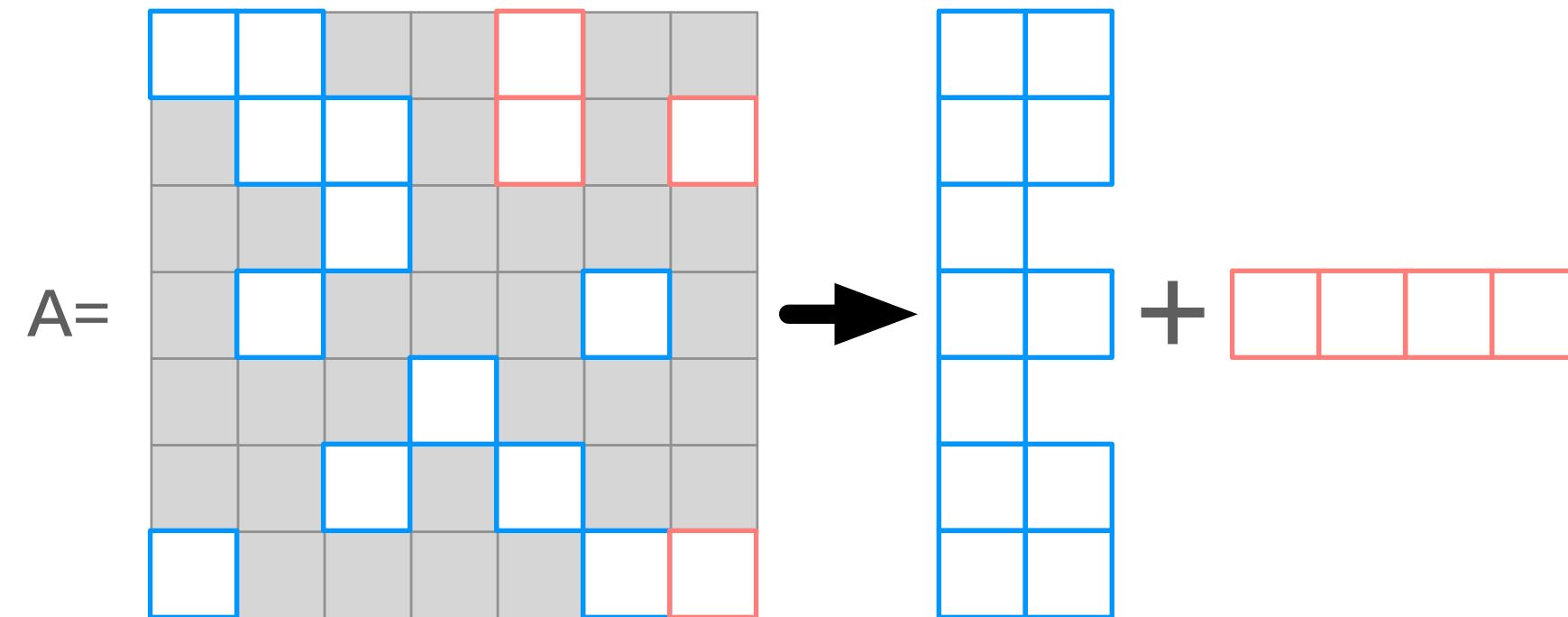
- 稀疏矩阵表现形式 (Hybrid)

- 混合形式

- 规则部分使用ELL
 - 超出部分使用COO

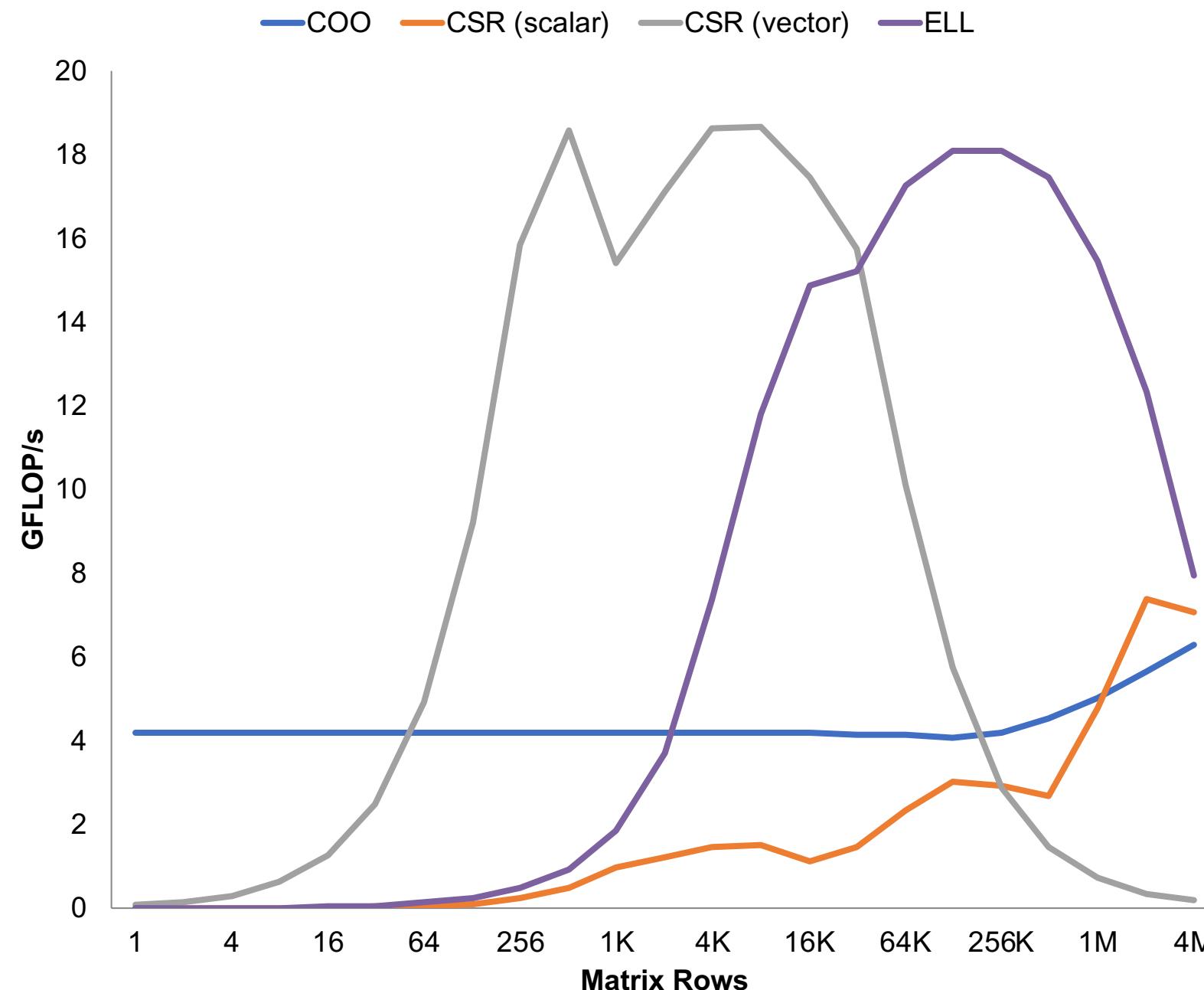
- 并行方案

- 分别根据ELL及COO方式处理



- 性能：4M个非0元素

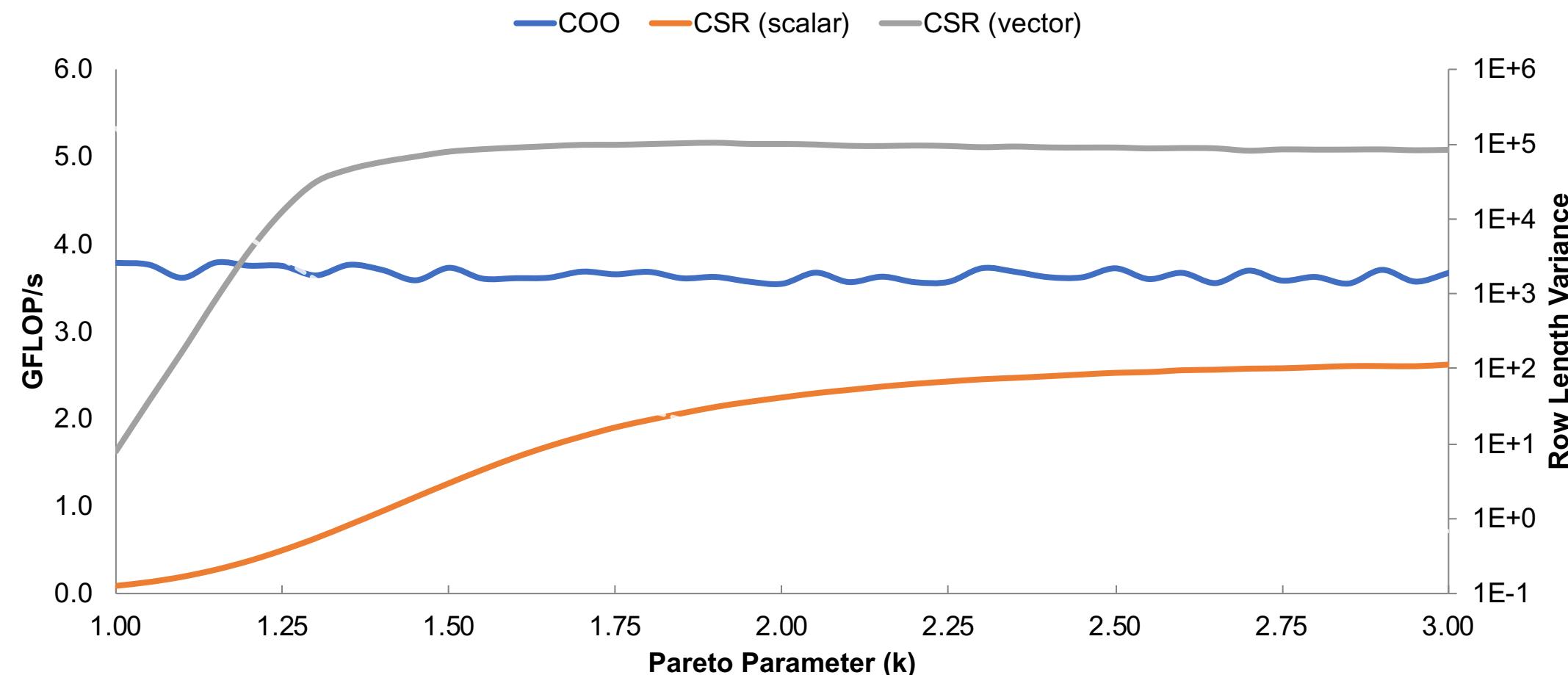
- 最大活跃线程束数：960
- 最大活跃线程数：30k
- 每行长度不一致
 - CSR (s) : 空闲线程
 - CSR (v) : 空闲流处理器
- COO时间相对稳定



图表来自NVIDIA

- 性能：每行中非0元素数量遵循帕雷托分布

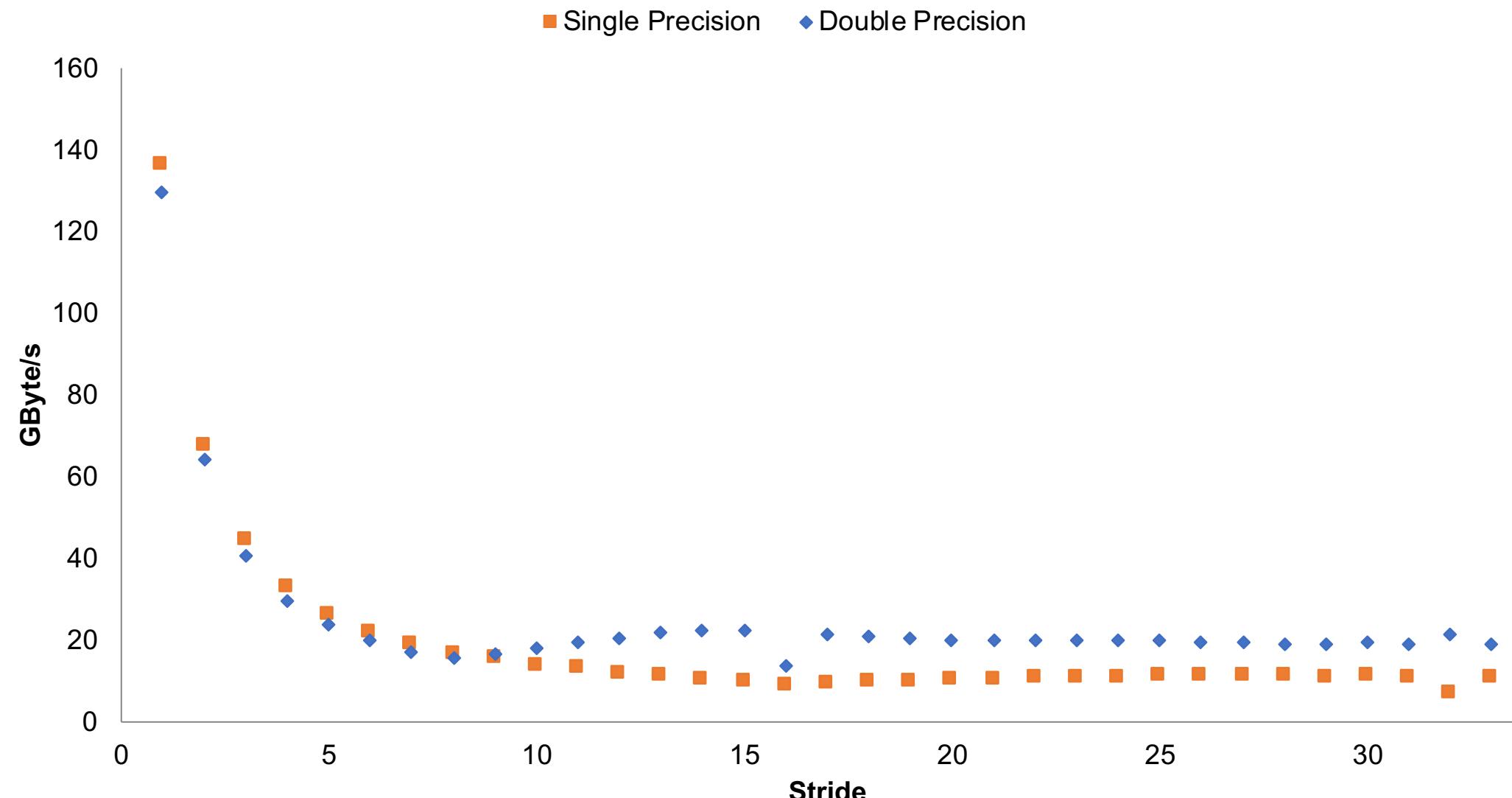
- 非合并访问开销极大
- 非对齐访问也会造成额外开销
- DIA, ELL与COO为合并访问，CSR(v)部分合并访问，CSR(s)极少合并访问



图表来自NVIDIA

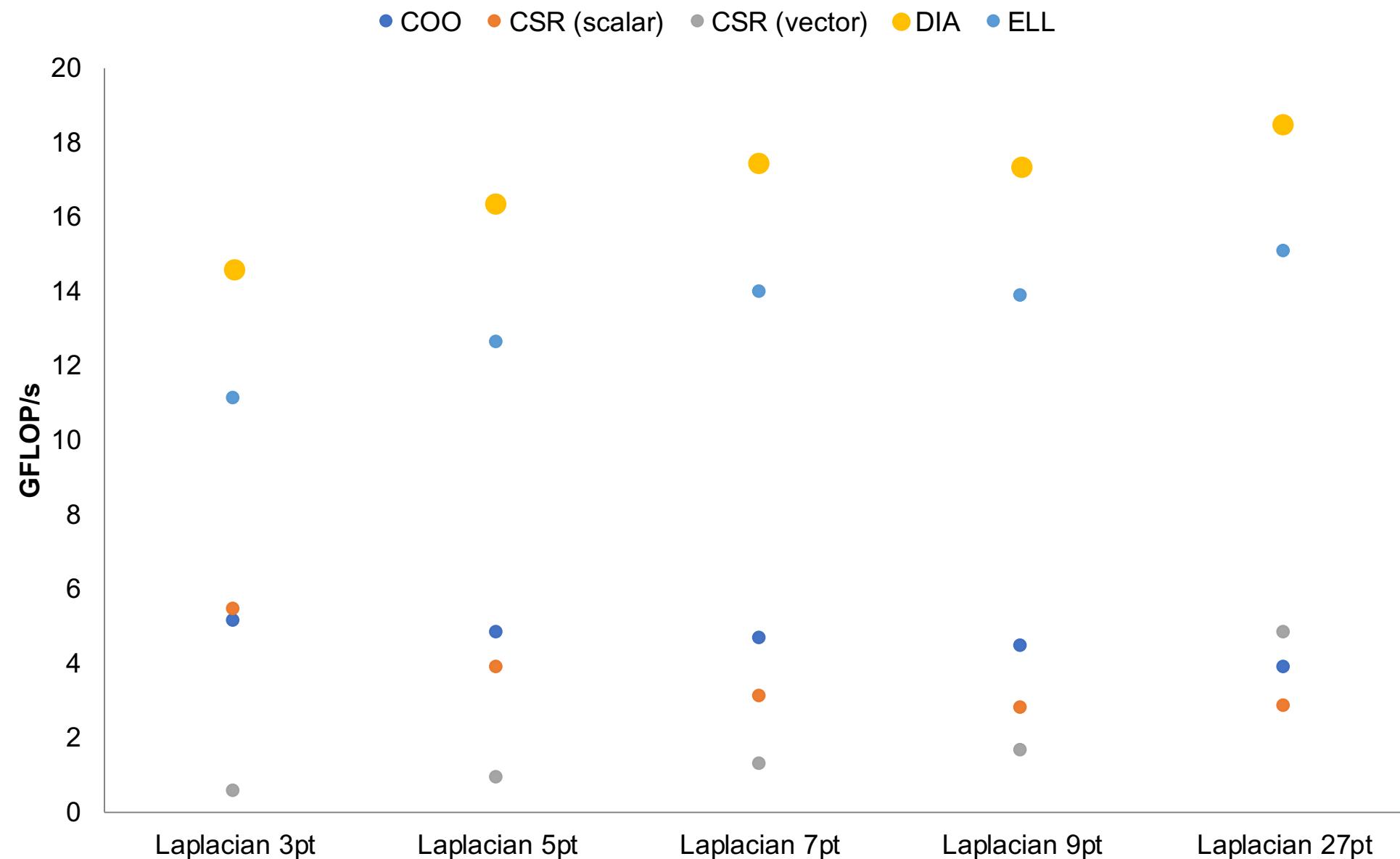
- 性能：带宽与访问步长

- 合并访问非常重要！



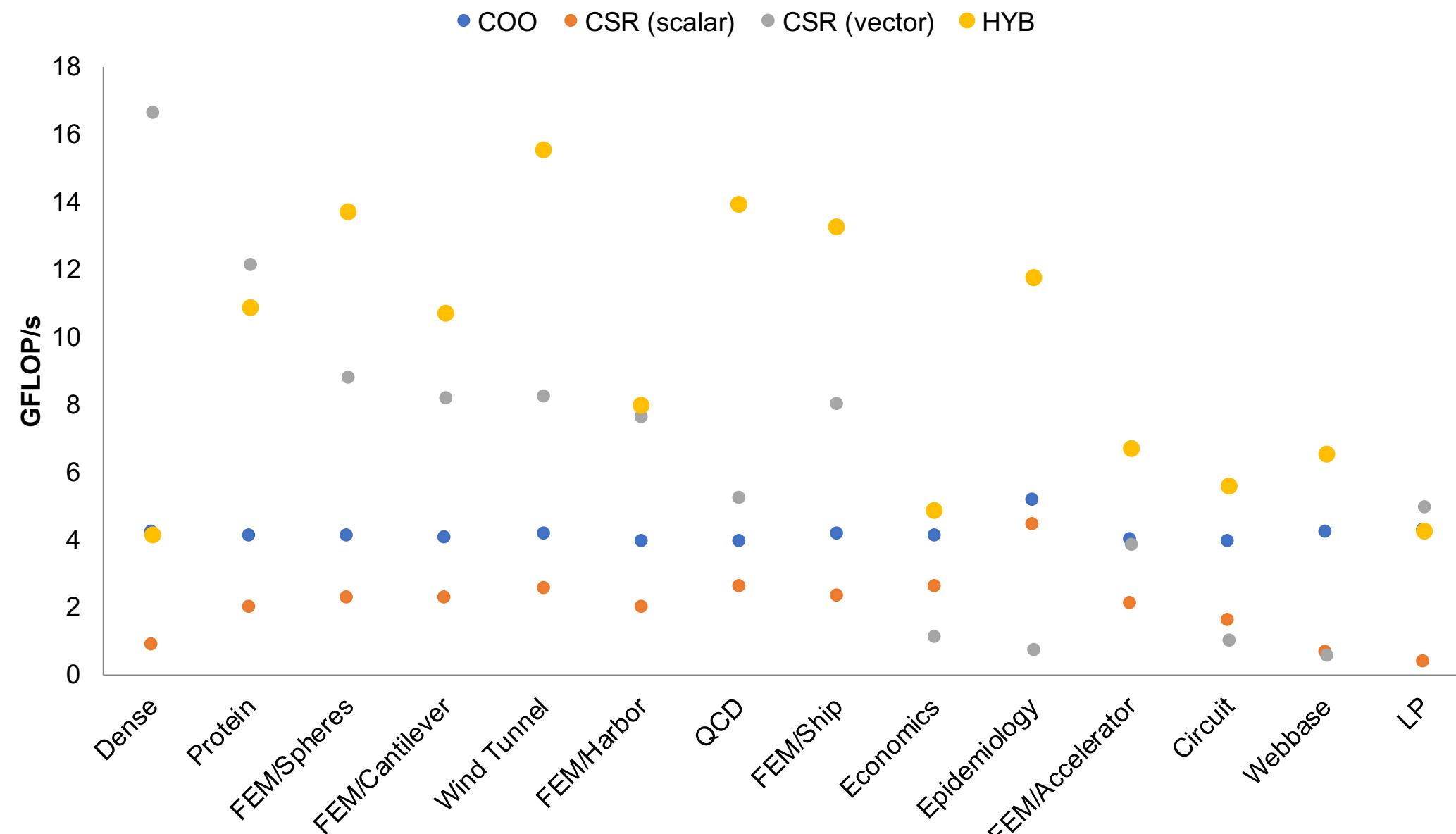
图表来自NVIDIA

● Structured Matrices



图表来自NVIDIA

● Unstructured Matrices



图表来自NVIDIA

C. Coleman et al., Efficient CUDA,
<http://cs242.stanford.edu/f17/assets/projects/2017/cod-ddkang.pdf>

GeeksforGeeks, Divide and Conquer | Set 5 (Strassen's Matrix Multiplication),
<https://www.geeksforgeeks.org/strassens-matrix-multiplication/>

N. Bell et al., Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors, <https://www.nvidia.com/docs/IO/77944/sc09-spmv-throughput.pdf>



Questions?

