Sparse matrices, Poisson reconstruction, LUTs

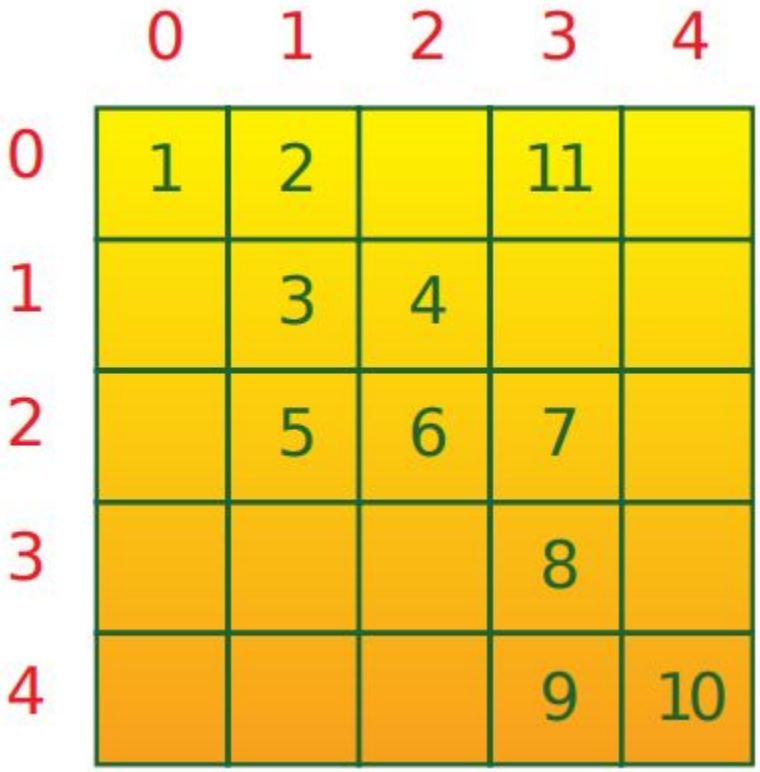
Вычисления на видеокартах. Лекция 8

Sparse matrices,
Poisson surface reconstruction on GPU,
Look Up Tables

Полярный Николай

polarnick239@gmail.com

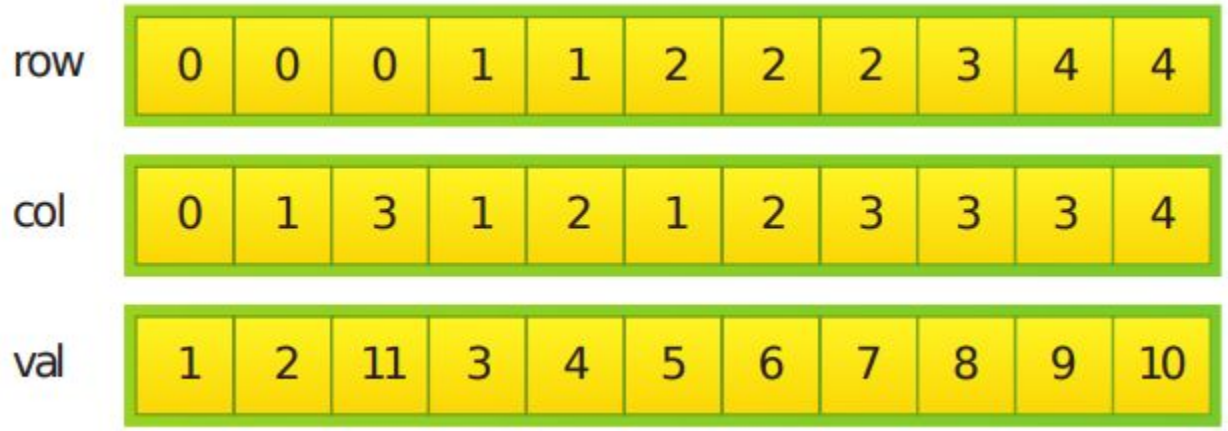
Представления разреженных матриц



Число ненулевых значений **NNZ=11**

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats

COOrdinate-wise - **COO**



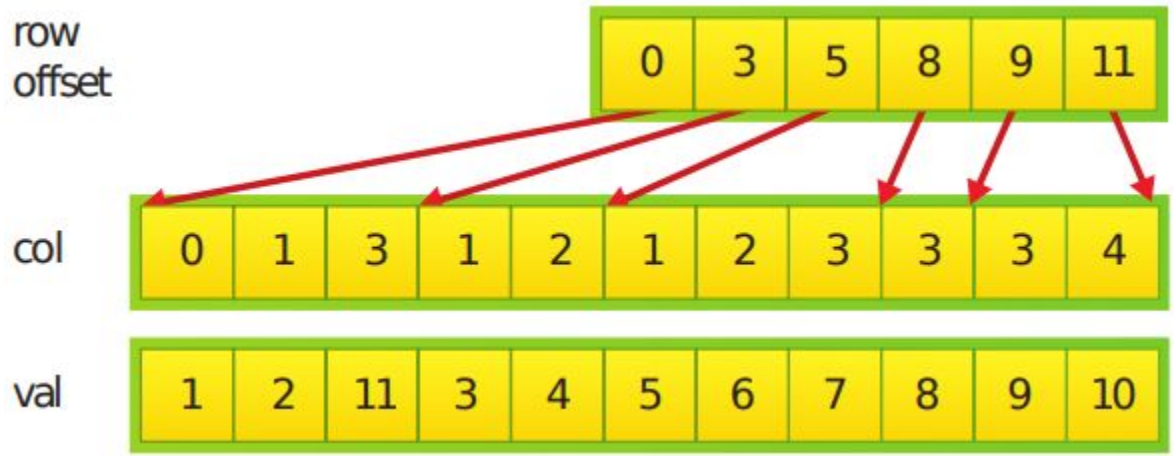
Row index - NNZ

Column index - NNZ

Values - NNZ

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats

Compressed **S**parse **R**ow - **CSR**



Row offsets - N+1

Column index - NNZ

Values - NNZ

NB: Сложно обратиться к колонке. Можно запускать **warp/wavefront** на ряд.

Аналогично с **C**ompressed **S**parse **C**olumn - **CSC**.

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats

ELLpack - **ELL**

col

0	1	3
1	2	*
1	2	3
3	*	*
3	4	*

val

1	2	11
3	4	*
5	6	7
8	*	*
9	10	*

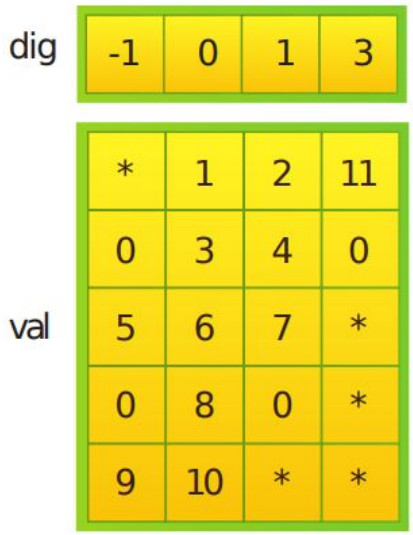
Column index - N*M

Values - N*M

Где **M** - максимальное число элементов в ряду.

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats

DIAgonal **- DIA**



Diagonal offsets - D

Values - N*D

Где **D** - число диагоналей с хотя бы одним ненулевым значением.

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats

Memory Footprint

Format	Structure	Values
Dense	–	$N \times N$
COO	$2 \times NNZ$	NNZ
CSR	$N + 1 + NNZ$	NNZ
ELL	$M \times N$	$M \times N$
DIA	D	$D \times N_D$

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats

HYB

Наблюдение:

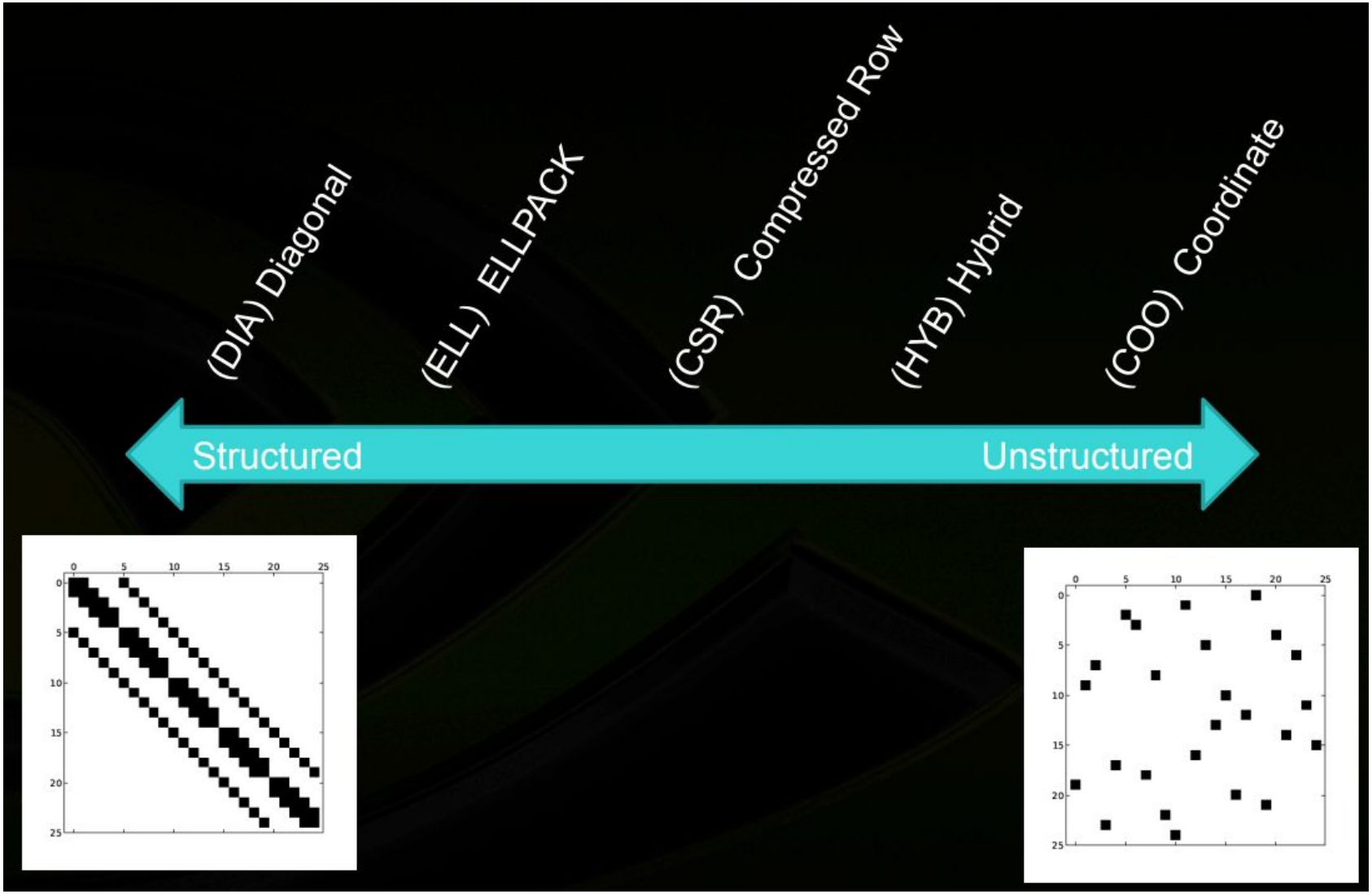
Большинство матриц почти подпадают под какой-то из паттернов, но содержат небольшое количество элементов лежащих вне этого паттерна.

$$A := B + C$$

Теперь **B** - идеально подходит под паттерн и представляется в соответствующем виде **ELL** или **DIA**.

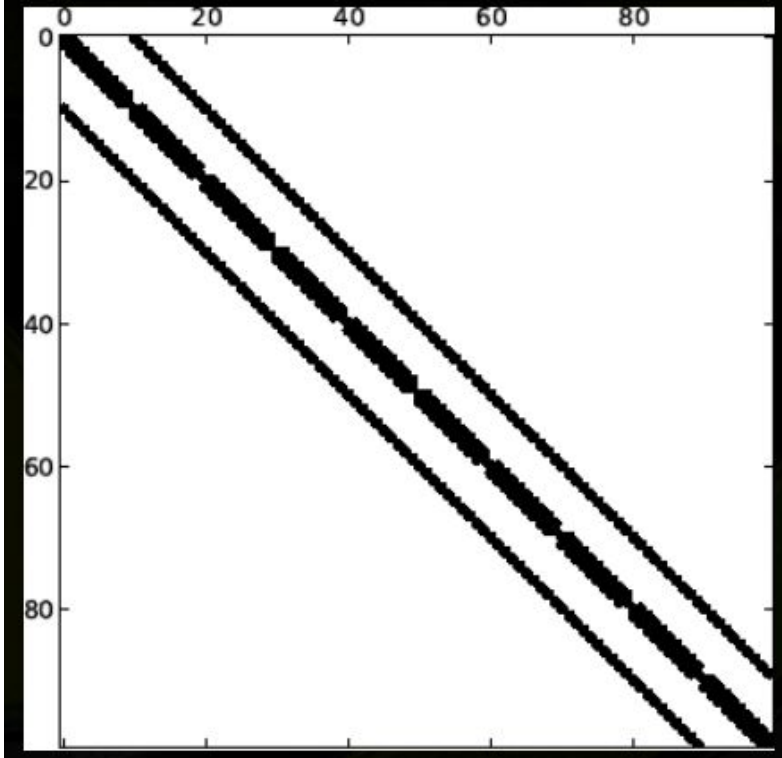
C - почти пустая матрица, соответственно для нее подходит **COO**.

Источник: Sparse Matrix-Vector Multiplication and Matrix Formats



Источник: [Sparse Matrix Representations & Iterative Solvers]

Structured Mesh



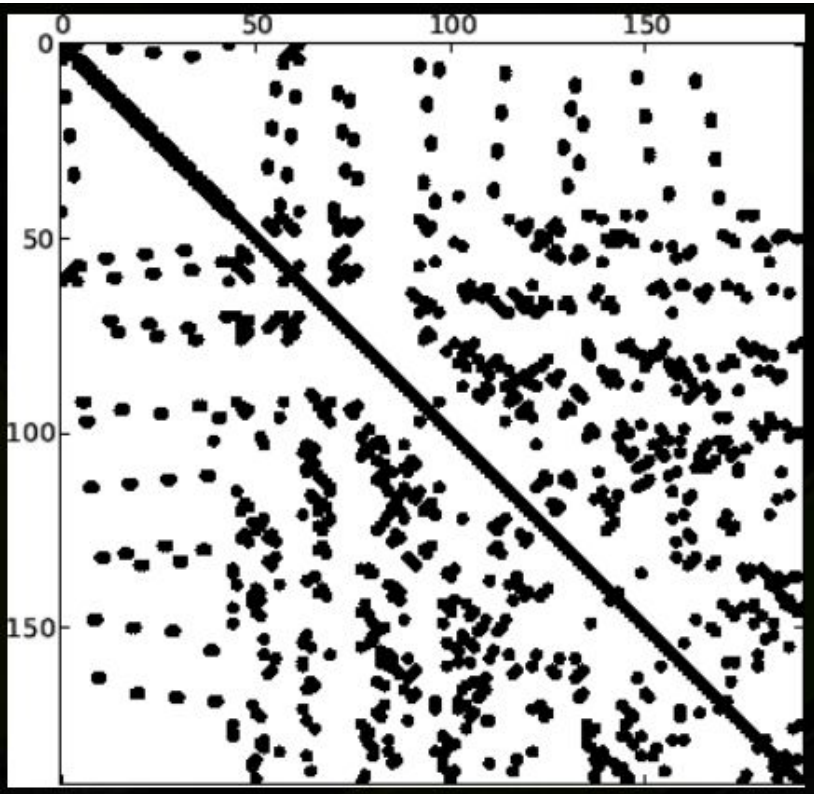
Matrix

Format	float	double
COO	12.00	16.00
CSR	8.45	12.45
DIA	4.05	8.10
ELL	8.11	12.16
HYB	8.11	12.16

Bytes per Nonzero Entry

Источник: Sparse Matrix Representations & Iterative Solvers

Unstructured Mesh



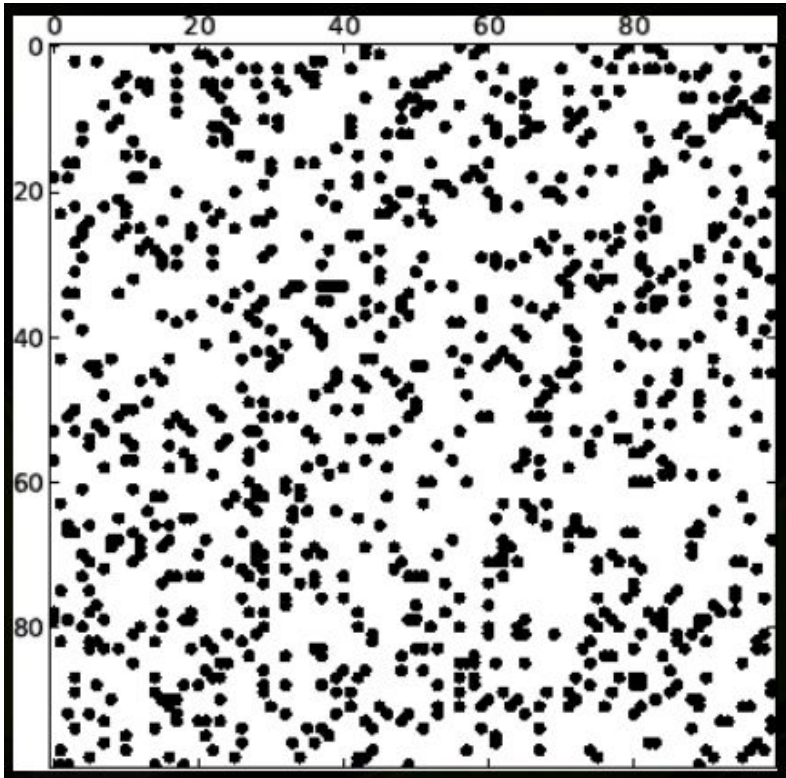
Matrix

Format	float	double
COO	12.00	16.00
CSR	8.62	12.62
DIA	164.11	328.22
ELL	11.06	16.60
HYB	9.00	13.44

Bytes per Nonzero Entry

Источник: Sparse Matrix Representations & Iterative Solvers

Random Matrix



Matrix

Format	Float	Double
COO	12.00	16.00
CSR	8.42	12.42
DIA	76.83	153.65
ELL	14.20	21.29
HYB	9.60	14.20

Bytes per Nonzero Entry

Источник: Sparse Matrix Representations & Iterative Solvers

СЛАУ

Пусть есть разреженная **СЛАУ**

A*x=b

Есть много итеративных методов:

- Якоби
- Гаусса-Зейделя
- Сопряженных Градиентов

Пример - метод Ричардсона:

$$x^{k+1} = x^k - \tau(Ax^k - b)$$

Достаточно научиться **умножать разреженную матрицу на вектор**.

COO

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\begin{array}{rl} \texttt{row} & \begin{bmatrix} 0 & 0 & 1 & 1 & 2 & 2 & 2 & 3 & 3 \end{bmatrix} \\ \texttt{col} & \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 2 & 3 & 1 & 3 \end{bmatrix} \\ \texttt{data} & \begin{bmatrix} 1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4 \end{bmatrix} \end{array}$$

$$\begin{array}{rl} \text{Iteration 0} & \begin{bmatrix} 0 & 1 & 2 & 3 & & & & & \end{bmatrix} \\ \text{Iteration 1} & \begin{bmatrix} & & & & 0 & 1 & 2 & 3 & \end{bmatrix} \\ \text{Iteration 2} & \begin{bmatrix} & & & & & & & & 0 \end{bmatrix} \end{array}$$

Источник: [Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008]

CSR (scalar)

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\mathtt{ptr} = \begin{bmatrix} 0 & 2 & 4 & 7 & 9 \end{bmatrix}$$

$$\mathtt{indices} = \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 2 & 3 & 1 & 3 \end{bmatrix}$$

$$\mathtt{data} = \begin{bmatrix} 1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4 \end{bmatrix}$$

$$\begin{array}{ll} \text{Iteration 0} & \begin{bmatrix} 0 & & 1 & & 2 & & & 3 & \end{bmatrix} \\ \text{Iteration 1} & \begin{bmatrix} & 0 & & 1 & & 2 & & & 3 \end{bmatrix} \\ \text{Iteration 2} & \begin{bmatrix} & & & & & & 2 & & \end{bmatrix} \end{array}$$

Источник: [Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008]

CSR (vector)

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\begin{aligned} \mathtt{ptr} &= \begin{bmatrix} 0 & 2 & 4 & 7 & 9 \end{bmatrix} \\ \mathtt{indices} &= \begin{bmatrix} 0 & 1 & 1 & 2 & 0 & 2 & 3 & 1 & 3 \end{bmatrix} \\ \mathtt{data} &= \begin{bmatrix} 1 & 7 & 2 & 8 & 5 & 3 & 9 & 6 & 4 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} &\text{Warp 0} && \begin{bmatrix} 0 & 0 & & & & & & & \end{bmatrix} \\ &\text{Warp 1} && \begin{bmatrix} & & 1 & 1 & & & & & \end{bmatrix} \\ &\text{Warp 2} && \begin{bmatrix} & & & & 2 & 2 & 2 & & \end{bmatrix} \\ &\text{Warp 3} && \begin{bmatrix} & & & & & & & 3 & 3 \end{bmatrix} \end{aligned}$$

Источник: Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008

ELL

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\mathtt{data} = \begin{bmatrix} 1 & 7 & * \\ 2 & 8 & * \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad \mathtt{indices} = \begin{bmatrix} 0 & 1 & * \\ 1 & 2 & * \\ 0 & 2 & 3 \\ 1 & 3 & * \end{bmatrix}$$

$$\begin{array}{rl} \mathtt{data} & \begin{bmatrix} 1 & 2 & 5 & 6 & 7 & 8 & 3 & 4 & * & * & 9 & * \end{bmatrix} \\ \mathtt{indices} & \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 2 & 2 & 3 & * & * & 3 & * \end{bmatrix} \end{array}$$

$$\begin{array}{ll} \text{Iteration 0} & \begin{bmatrix} 0 & 1 & 2 & 3 & & & & & & & & \end{bmatrix} \\ \text{Iteration 1} & \begin{bmatrix} & & & & 0 & 1 & 2 & 3 & & & & \end{bmatrix} \\ \text{Iteration 2} & \begin{bmatrix} & & & & & & & & 0 & 1 & 2 & 3 \end{bmatrix} \end{array}$$

DIA

В отличии от **ELL**
доступ к **x** - последовательный.

$$\mathtt{A} = \begin{bmatrix} 1 & 7 & 0 & 0 \\ 0 & 2 & 8 & 0 \\ 5 & 0 & 3 & 9 \\ 0 & 6 & 0 & 4 \end{bmatrix}$$

$$\mathtt{data} = \begin{bmatrix} * & 1 & 7 \\ * & 2 & 8 \\ 5 & 3 & 9 \\ 6 & 4 & * \end{bmatrix} \qquad \mathtt{offsets} = \begin{bmatrix} -2 & 0 & 1 \end{bmatrix}$$

$$\mathtt{data} \quad \begin{bmatrix} * & * & 5 & 6 & 1 & 2 & 3 & 4 & 7 & 8 & 9 & * \end{bmatrix}$$

$$\begin{array}{ll} \text{Iteration 0} & \begin{bmatrix} & & 2 & 3 & & & & & & & & \end{bmatrix} \\ \text{Iteration 1} & \begin{bmatrix} & & & & 0 & 1 & 2 & 3 & & & & \end{bmatrix} \\ \text{Iteration 2} & \begin{bmatrix} & & & & & & & & 0 & 1 & 2 & \end{bmatrix} \end{array}$$

Источник: [Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008]

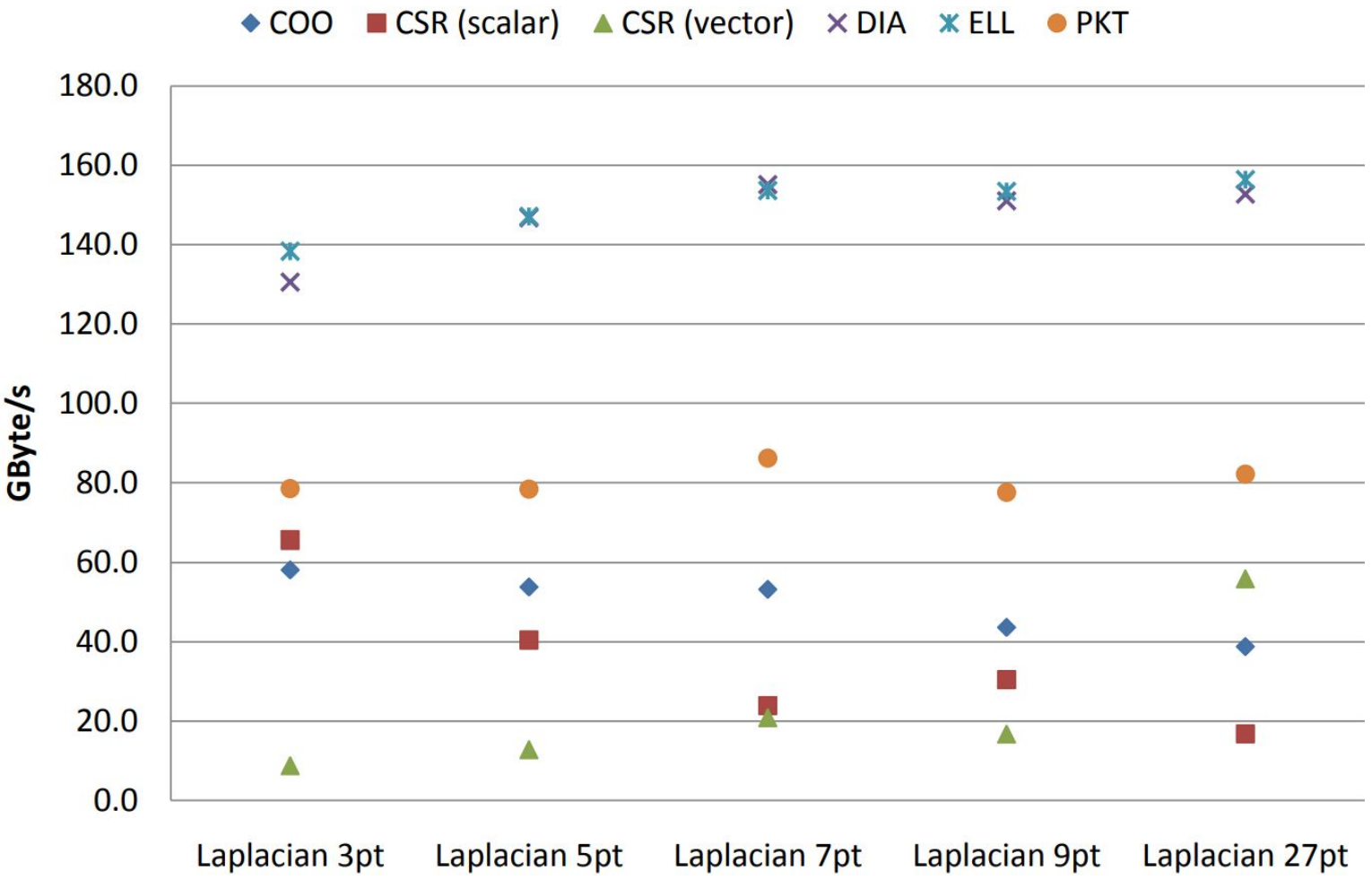
Structured matrices

Matrix	Grid	Diagonals	Rows	Columns	Nonzeros
Laplace 3pt	$(1,000,000)$	3	1,000,000	1,000,000	2,999,998
Laplace 5pt	$(1,000)^2$	5	1,000,000	1,000,000	4,996,000
Laplace 7pt	$(100)^3$	7	1,000,000	1,000,000	6,940,000
Laplace 9pt	$(1,000)^2$	9	1,000,000	1,000,000	8,988,004
Laplace 27pt	$(100)^3$	27	1,000,000	1,000,000	26,463,592

Table 3: Structured matrices used for performance testing.

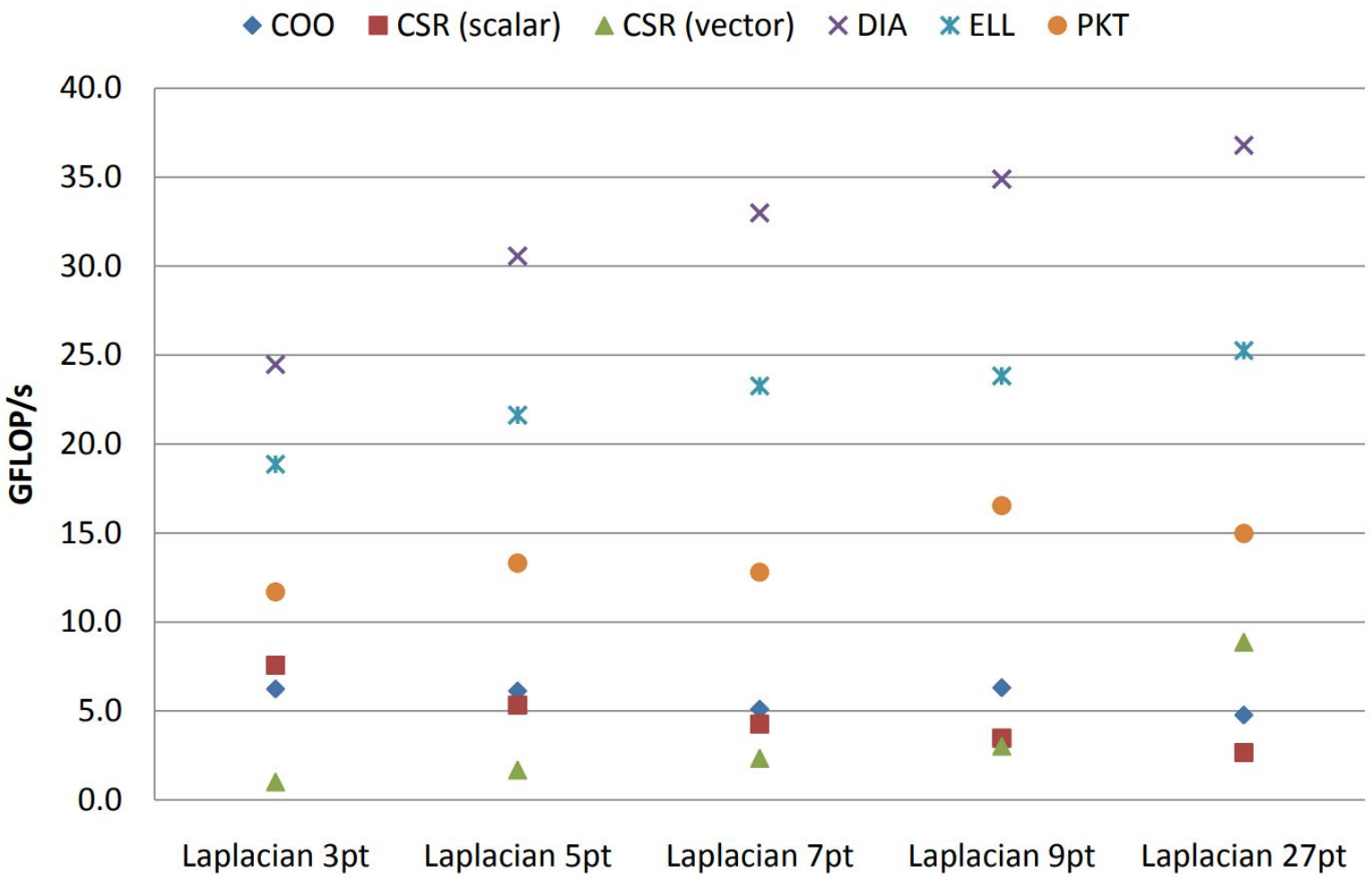
Источник: Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008

Structured matrices - bandwidth



Источник: [Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008]

Structured matrices - gflops



Источник: [Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008]

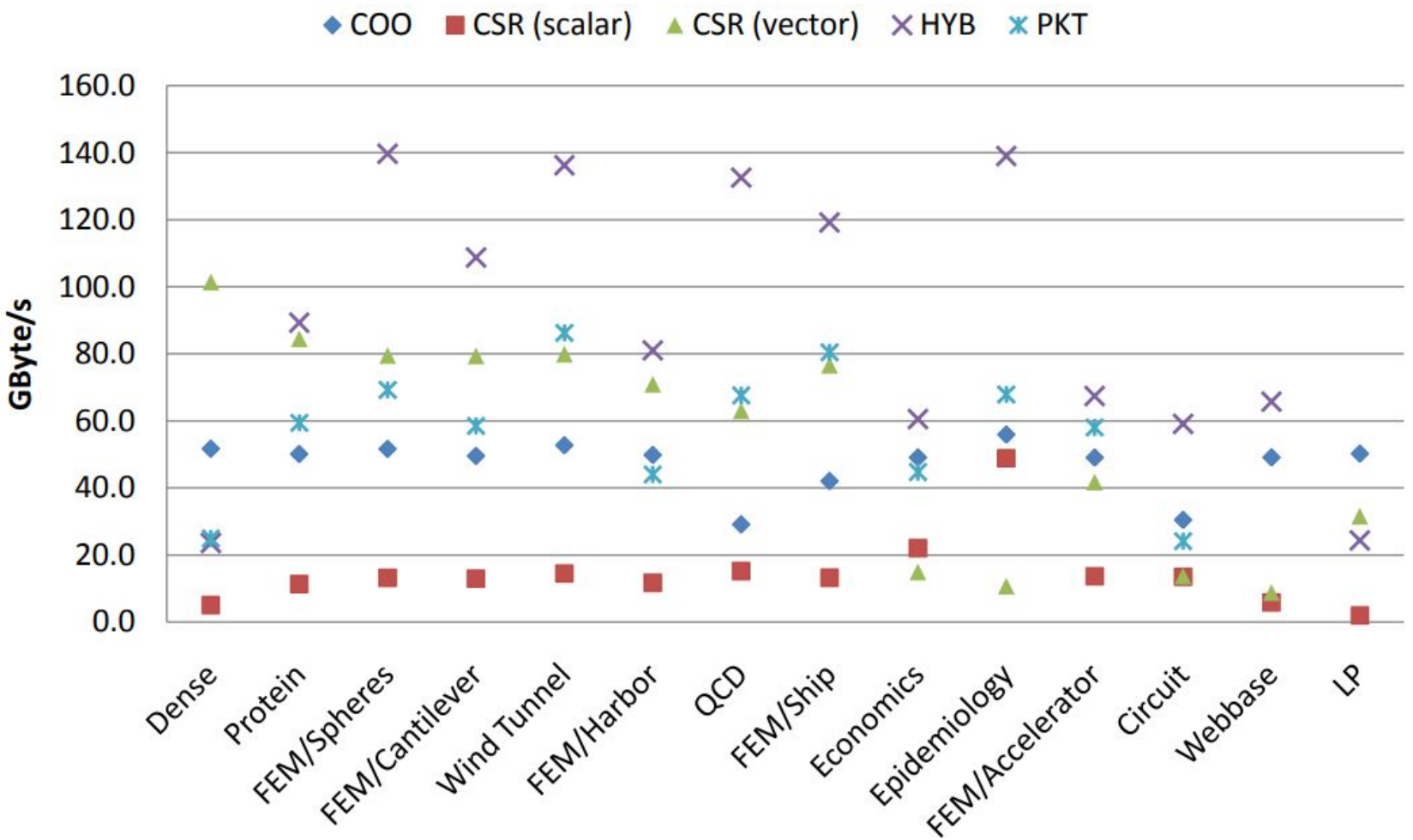
Unstructured matrices

Matrix	Rows	Columns	Nonzeros	Nonzeros/Row
Dense	2,000	2,000	4,000,000	2000.0
Protein	36,417	36,417	4,344,765	119.3
FEM/Spheres	83,334	83,334	6,010,480	72.1
FEM/Cantilever	62,451	62,451	4,007,383	64.1
Wind Tunnel	217,918	217,918	11,634,424	53.3
FEM/Harbor	46,835	46,835	2,374,001	50.6
QCD	49,152	49,152	1,916,928	39.0
FEM/Ship	140,874	140,874	7,813,404	55.4
Economics	206,500	206,500	1,273,389	6.1
Epidemiology	525,825	525,825	2,100,225	3.9
FEM/Accelerator	121,192	121,192	2,624,331	21.6
Circuit	170,998	170,998	958,936	5.6
Webbase	1,000,005	1,000,005	3,105,536	3.1
LP	4,284	1,092,610	11,279,748	2632.9

Table 4: Unstructured matrices used for performance testing.

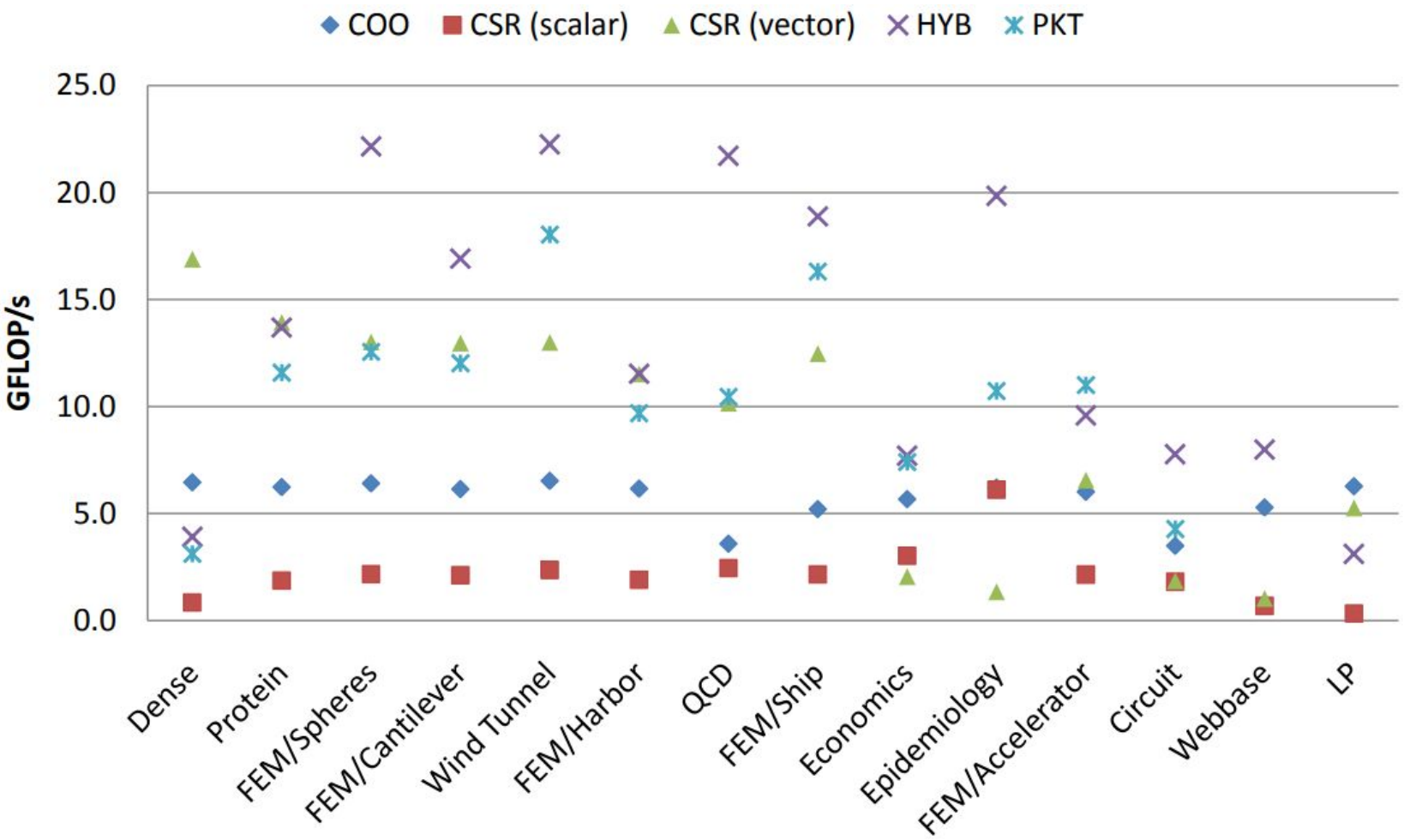
Источник: Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008

Unstructured matrices - bandwidth



Источник: Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008

Unstructured matrices - gflops



Источник: [Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008]

Ссылки

- Sparse Matrix-Vector Multiplication and Matrix Formats
- Sparse Matrix Representations & Iterative Solvers
- Efficient Sparse Matrix-Vector Multiplication on CUDA, Bell et al., 2008

Poisson surface reconstruction

На входе $\vec{V}$ - точки с инвертированными нормалями.
Находим индикатор χ - скалярное поле, чей градиент приближает векторное поле норма $\min_\chi \|\nabla\chi - \vec{V}\|$
Результирующая поверхность ∂M - изоповерхность в поле индикатора.

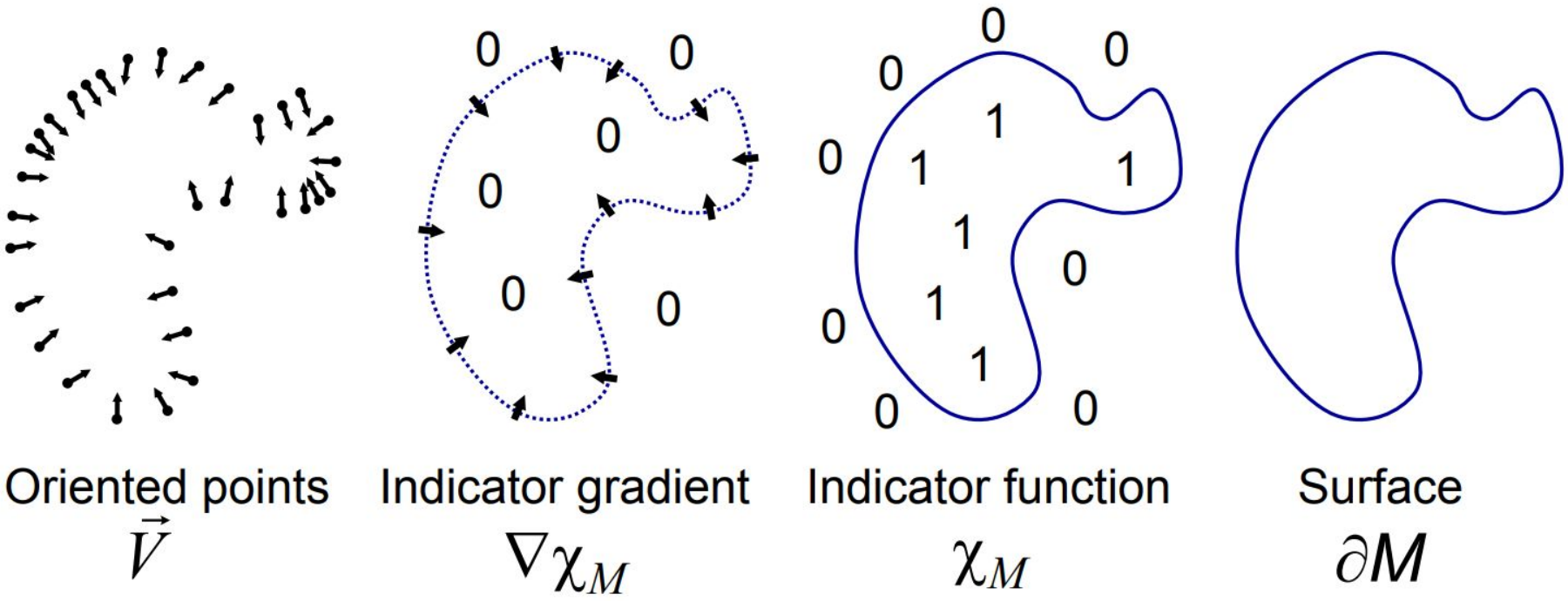


Figure 1: *Intuitive illustration of Poisson reconstruction in 2D.*

Сведение к уравнению Пуассона

Итого хочется найти такой индикатор χ чтобы:

$$\nabla\tilde{\chi} = \vec{V}$$

Явного решения нет, но можно приближенно решить минимизируя квадратичную ошибку оператором дивергенции сведя к уравнению Пуассона:

$$\Delta\chi \equiv \nabla\cdot\nabla\chi = \nabla\cdot\vec{V}$$

$$\Delta\tilde{\chi} = \nabla\cdot\vec{V}$$

Дискретизация

Регулярная решетка - кубическая память.

На самом деле интересно лишь пространство около поверхности. Поэтому можно использовать адаптивное октодерево.

В каждом узле октодерева с центром $o.c$ и шириной $o.w$ находится базовая функция $F : \mathbb{R}^3 \to \mathbb{R}$

$$F_o(q) \equiv F\left(\frac{q - o.c}{o.w}\right)\frac{1}{o.w^3}$$

Т.е. каждая точка делает вклад в базовую функцию своего узла.

Решение

Задача сводится к:

$$\min_{x \in \mathbb{R}^{|\mathcal{O}|}} \|Lx - v\|^2$$

Где L - разреженная, симметричная матрица размера $|\mathcal{O}| \times |\mathcal{O}|$ (квадрат размера октодерева).

Решается методом сопряженных градиентов.

Число столбцов - число соседних узлов чья базисная функция пересекается с узлом текущего ряда. Поэтому $|\mathcal{O}| \times 125$

Подробнее: Poisson Surface Reconstruction, Kazhdan et al., 2006

GPU Poisson reconstruction

1) Как построить октодерево?
2) Как эффективно найти соседние узлы?
3) Как решить уравнение Пуассона?

Как построить октодерево. Структура узла.

1) Shuffled xyz key:

$$x_1 y_1 z_1 x_2 y_2 z_2 \cdots x_D y_D z_D$$

Для октодерева глубины 10 - 30 бит.

2) Точки содержащиеся в узле: **индекс** первой точки и **количество** точек.

3) Указатели:

- родительский узел
- 8 детей
- 27 соседей (включая сам узел)

Итого: >156 байт.

Построение октодерева. Самая глубокая ступень.

1) Нашли bounding box (через редукцию min/max по каждой оси).
2) Для каждой вершины посчитали **shuffled xyz key**.
3) И отсортировали вершины по этому ключу. Теперь вершины с одинаковым ключем (а значит лежащие в одном узле) лежат подряд.
4) Нашли уникальные ключи.
5) По каждому узлу проверили - общий ли родитель с предыдущем по ключу узлом. Если да - положили рядом **0**, если нет - положили рядом **8**.
6) Посчитав префиксную сумму по лежащим рядом числам получили глобальный индекс восьми ячеек в октодереве для нас и наших братьев.

Построение октодерева. Остальные ступени.

Чтобы найти индекс родителя и создать его:

1) Занулить три бита ключа соответствующие самой детальной ступени.

2) Повторить аналогично шагам 5 и 6 из построения самой глубокой ступени.

3) При этом:

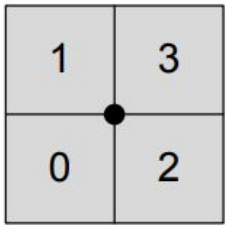
- выставляется ссылка на родителя
- у родителя выставляются ссылки на детей
- у родителя выставляется индекс первой вершины и количество вершин узла (сумма количества вершин в каждом ребенке)

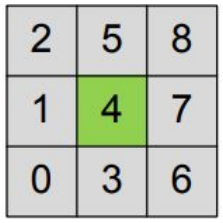
Построение октодерева. Соседи.

У каждого узла до **26** соседей.

Чтобы их найти потребуется сделать **26*27*8=5616** поисков
(26 соседей, 27: родительский узел и его соседи - каждый с 8 детьми).

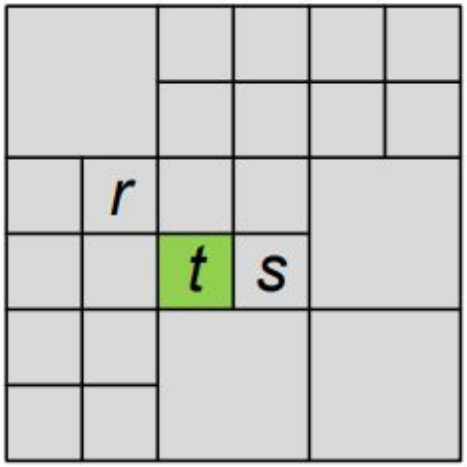
Построение октодерева. Look Up Tables.

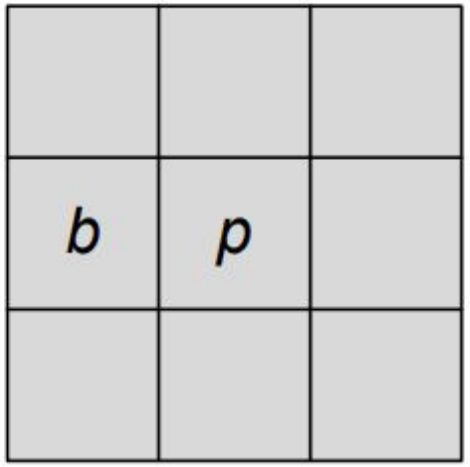




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };





1) LUTparent: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.
Тогда **parent(t.neighs[j]) = LUTparent[i][j]**

2) LUTchild: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.

Пусть **parent(t.neighs[j]) = h**

Тогда **t.neigh[j] = h.children[LUTchild[i][j]]**

Построение октодерева. Look Up Tables.

```
LUTparents = []
for iy in range(2):
    for ix in range(2):
        i = iy * 2 + ix

        parents = []
        for jy in range(3):
            for jx in range(3):
                j = jy * 3 + jx
                globalx, globaly = 2 + ix + (jx - 1), 2 + iy + (jy - 1)
                px, py = int(globalx // 2), int(globaly // 2)
                parents.append(py * 3 + px)

        LUTparents.append(parents)

def LUTparent(i, j):
    ix, iy, iz = i % 2, (i // 2) % 2, (i // 4)
    jx, jy, jz = i % 3, (i // 3) % 3, (i // 9)
    globalx, globaly, globalz = ...
    px, py, pz = ...
    return pz * 3 * 3 + py * 3 + px
```

Решение уравнения Пуассона.

1) Строим СЛАУ $\mathbf{Lx} = \mathbf{b}$
2) Решаем методом сопряженных градиентов

Где L - разреженная, симметричная матрица размера $|\mathcal{O}| \times |\mathcal{O}|$ (квадрат размера октодерева).

Число столбцов - число соседних узлов чья базисная функция пересекается с узлом текущего ряда. Поэтому $|\mathcal{O}| \times 125$ или $|\mathcal{O}| \times 27$ (грубее, но компактнее).

Подробнее: Data-Parallel Octrees for Surface Reconstruction, Zhou et al., 2011

Ссылки

- Poisson Surface Reconstruction, Kazhdan et al., 2006
- Data-Parallel Octrees for Surface Reconstruction, Zhou et al., 2011
- https://devtalk.nvidia.com/default/topic/609551/my-cuda-programming-lecture-and-teaching-of-poisson-parallel-surface-reconstruction-in-a-summer-scho/
- https://www.youtube.com/watch?v=ykUs4MYOwcY