GSKS GSKNN

BLIS-Based High Performance Computing Kernels
in N-body Problems

Chenhan D. Yu

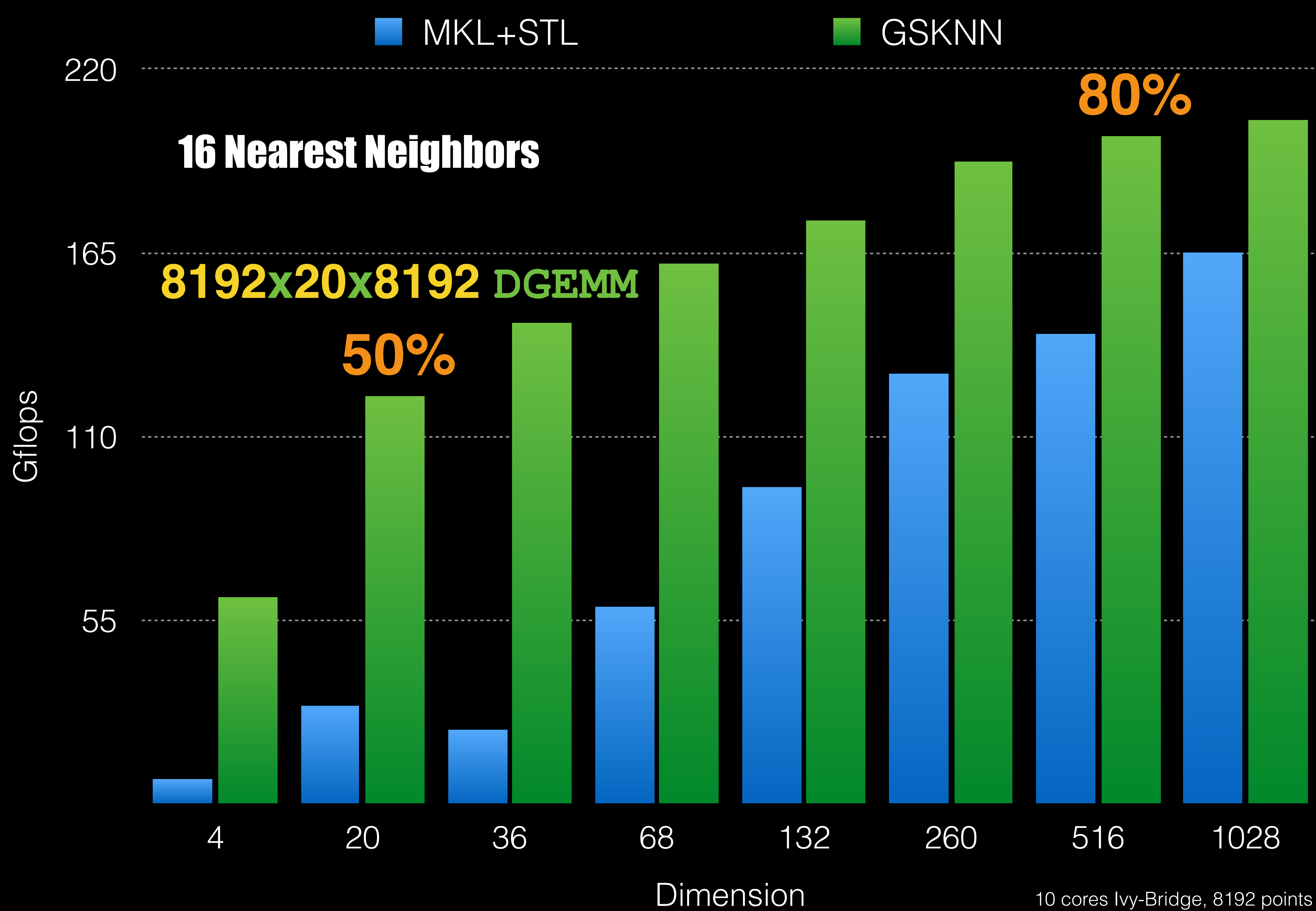
The 3rd BLIS Retreat
Sep 28, 2015

N-body Problems

Hellstorm Astronomy and 3D https://youtu.be/bLLWkx_MRfk

N-body Problems

- N-body problems aim to describe the interaction (relation) of **N** points { **X** } in a **d** dimensional space.
- $\mathcal{K}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{K}_{ij}$ describes the interaction between $\mathbf{x}_i$ and $\mathbf{x}_j$.
- **3** operations: Kernel Summation $\mathbf{u}=\mathbf{K}\mathbf{w}$, Kernel Inversion $\mathbf{w}=(\mathbf{K}+\lambda\mathbf{I})^{-1}\mathbf{u}$ and Nearest-Neighbors.
- **2**D and **3**D applications can be found in computational physics, geophysical exploration and medical imaging.
- High dimension applications in computational statistic include clustering, classification and regression.


MKL+STL
GSKNN
220
165
110
55
Gflops
16 Nearest Neighbors
8192x20x8192 DGEMM
50%
80%
4
20
36
68
132
260
516
1028
Dimension
10 cores Ivy-Bridge, 8192 points

Outline

- Kernel Summation (**u=Kw**) and Nearest-Neighbors.
- How **GEMM** is applied in the conventional approach?
- Why **GEMM** can be memory bound in these operations?
- What insight is required to design an algorithm that avoids redundant memory operations but still preserves the efficiency?
- How GSKS and **GSKNN** are inspired by the BLIS framework in their design?

Linear Kernel: $\mathcal{K}(x_i, x_j) = x_i^T x_j$

x_1

(1,1)

x_2

(0,0)

x_3

(1,0)

R

x_1	x_2	x_3
1	0	1
1	0	0

$Q^T = R$

x_1^T	1	1
x_2^T	0	0
x_3^T	1	0

$K = Q^T R$

2	0	1
0	0	0
1	0	1

Kernel Summation **Kw**

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

Nearest-Neighbors

2	0	1
0	0	0
1	0	1

x_2	x_3
x_1	x_2
x_2	x_1

Other Kernels

$\mathcal{K}(x_i, x_j) = f(x_i^T x_j)$, e.g. Gaussian kernel

$$\mathcal{K}(x_i, x_j) = \exp(-\|x_i - x_j\|_2^2/(2h^2))$$

The expansion exposes **GEMM** operations:

$$\|x_i - x_j\|_2^2 = \|x_i\|_2^2 + \|x_j\|_2^2 - 2x_i^T x_j$$

GEMM

$x_1^T x_1$	2
$x_2^T x_2$	0
$x_3^T x_3$	1

x_1^T	1	1
x_2^T	0	0
x_3^T	1	0

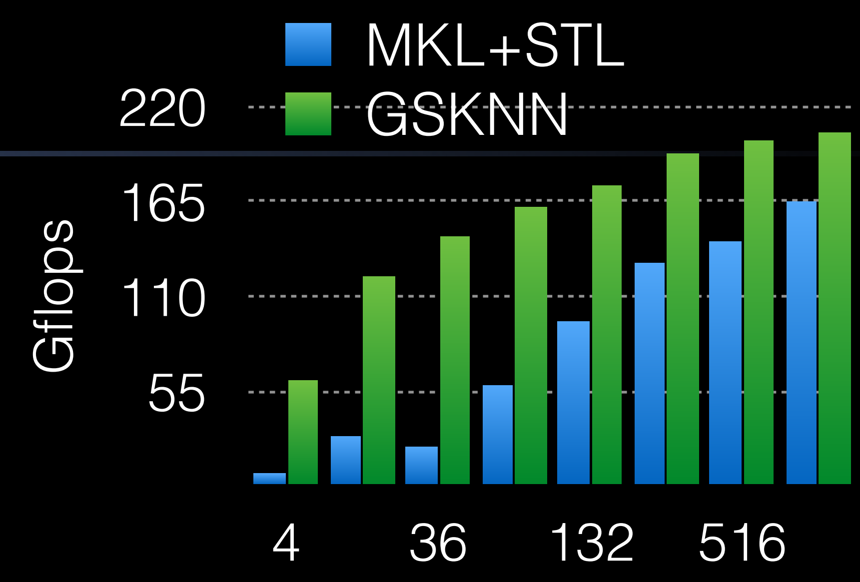
2	0	1
0	0	0
1	0	1

Q^T $\quad K = Q^T R$ $\quad$ 1+2-2*1

The Big Picture

- **Kw** takes $O(N^2)$ if **K** is precomputed, otherwise $O(dN^2)$. The cost is too expensive when **N** is large.
- Exhaustive search requires $O(N^2\log(k))$ if **K** is precomputed, otherwise $O(dN^2+N^2\log(k))$.
- Divide-and-conquer approximation: Barnes-Hut or FMM for kernel summation, and randomized KD-tree or locality sensitive hashing for kNN.
- Still the subproblem of all these algorithms is to solve several smaller dense kernel summation or kNN.
- Solving the subproblem fast benefits all these methods.

Subproblem

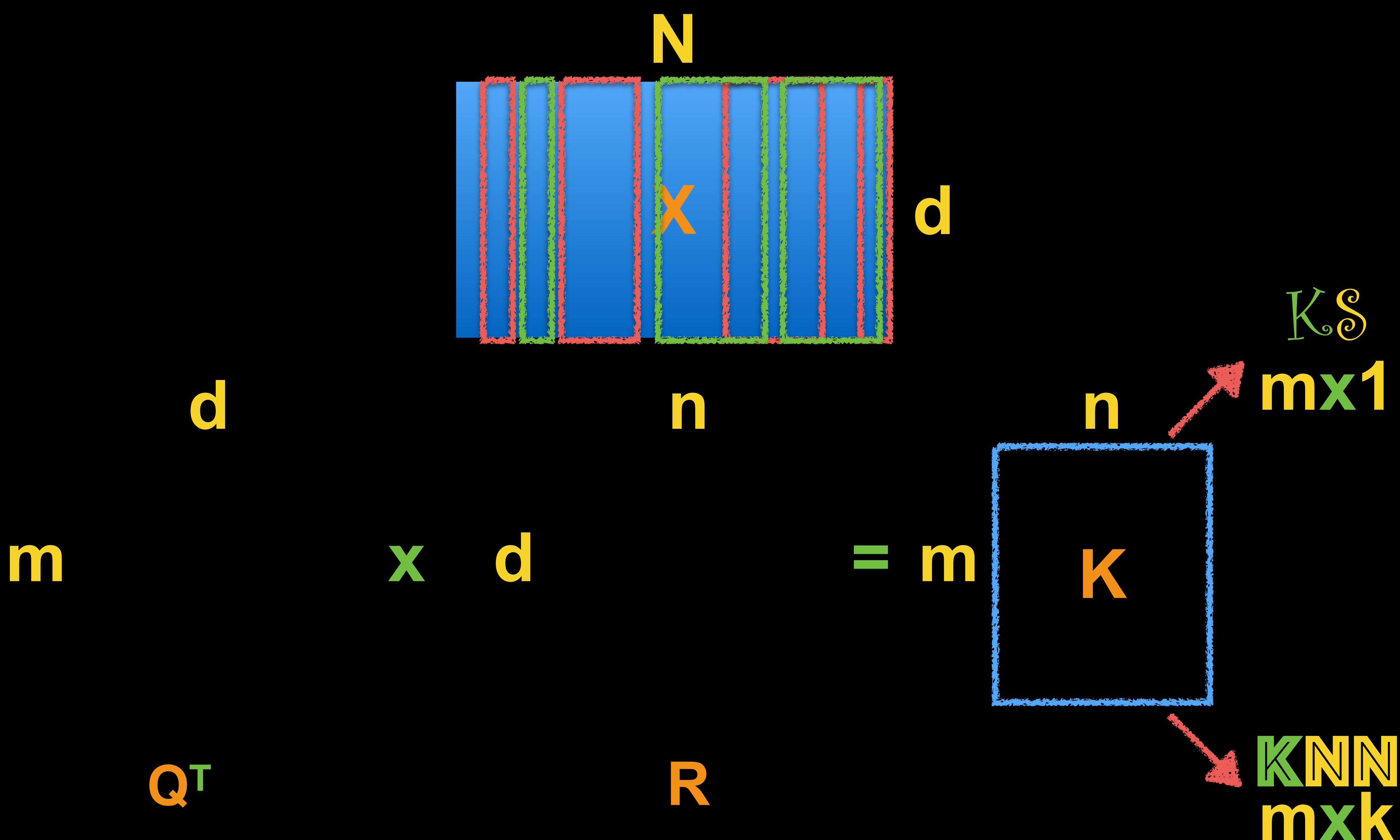


- Take two subsets **Q** and **R** **from X**.
- Compute $\mathcal{K}$**(Q,R)** with **GEMM** using:

$$\|x_i - x_j\|_2^2 = \|x_i\|_2^2 + \|x_j\|_2^2 - 2x_i^T x_j$$

- Compute **Kw** with **GEMV** or select **k** entries in each row.
- Rely on BLAS, VML (Vectorized Math Library) and STL.
- What can possibly go wrong?

Visualization



Insights

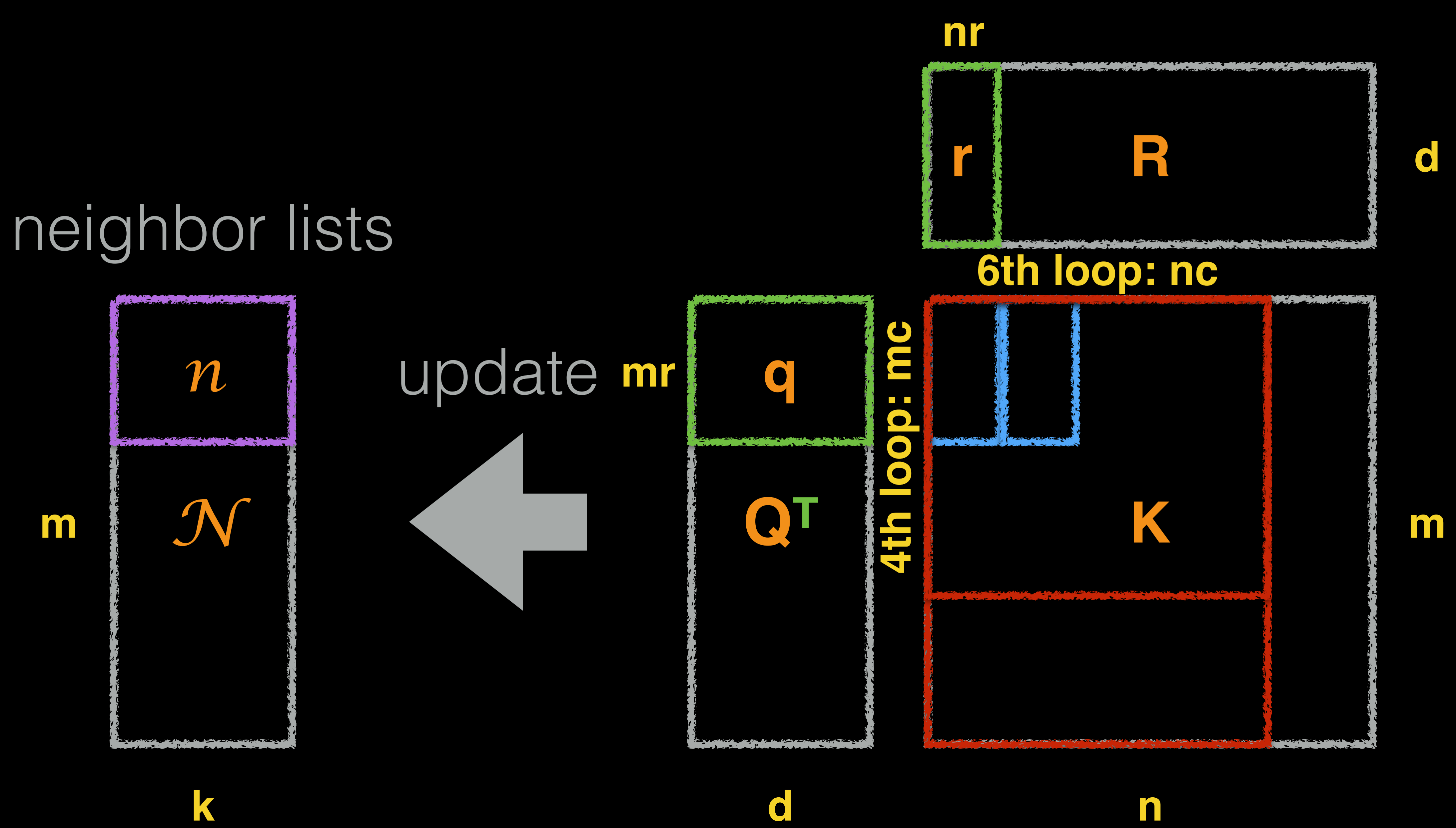
- **Q**, **R** and **K** can't be stored.
- Collet **Q** and **R** from **X** during packing.
- $\mathcal{K}(x_i, x_j) = K_{ij}$ must be computed in registers.
- **Kw** or **k**-select must be completed in registers.
- Only store the output.

i.e.

- We need a special packing routine.
- Fuse **GEMM** with distance calculations, special function evaluations, **Kw** or **k**-select.

Code Fusion in BLIS *Slice and Dice!*

Code fusion is done in micro-kernel,
and the BLIS framework is maintained.



GSKNN and BLIS ($K=Q^TR$)

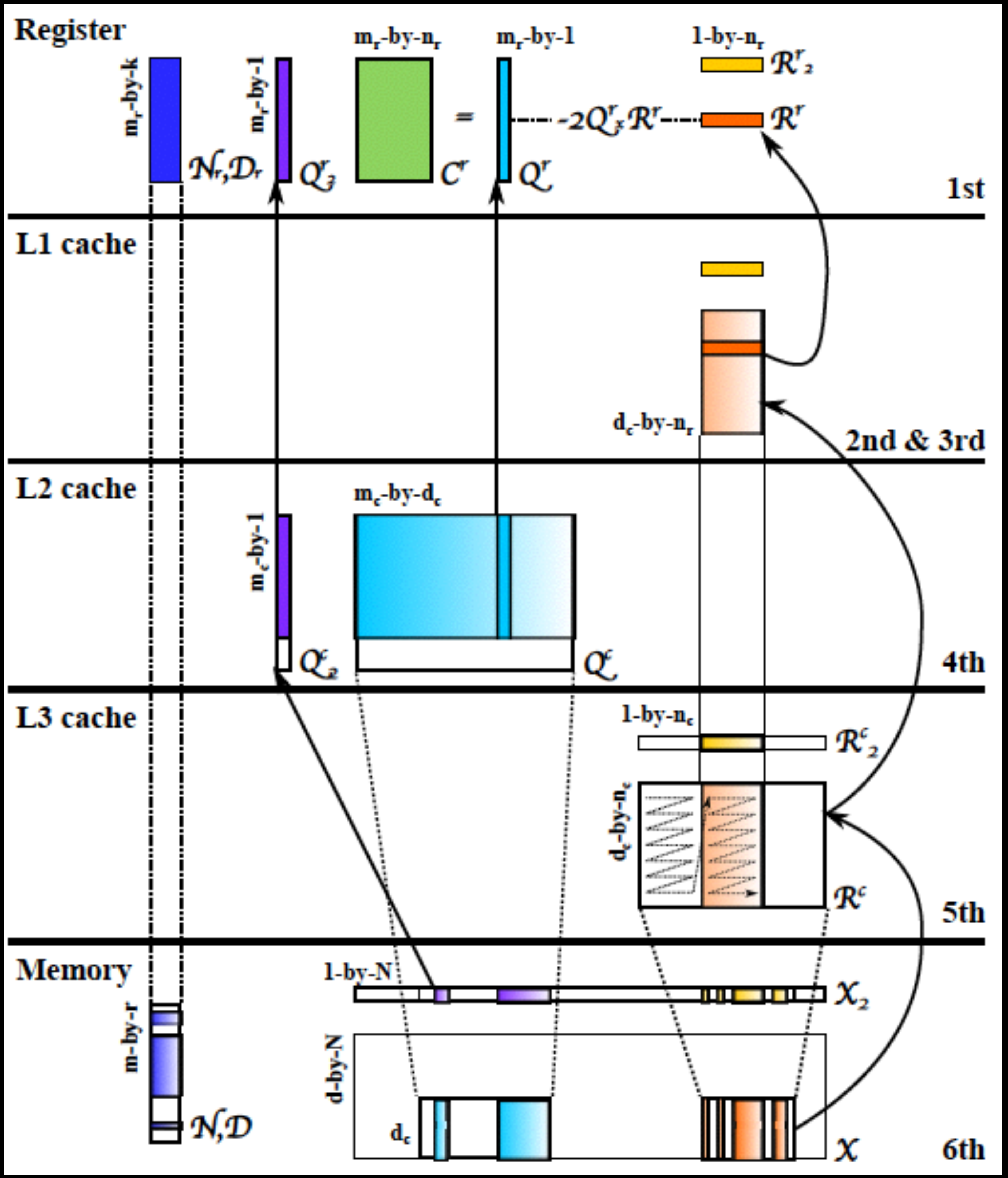
Micro-Kernel

Stream Q^C & Q^C_2

Reuse R^C & R^C_2

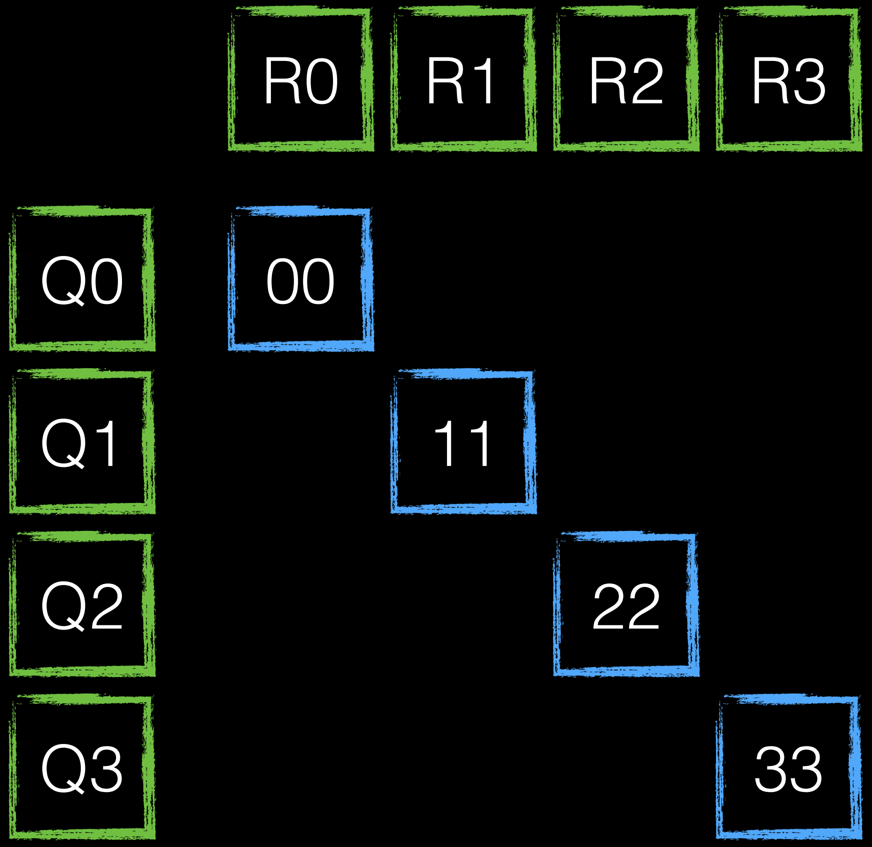
Pack Q^C & Q^C_2

Pack R^C & R^C_2



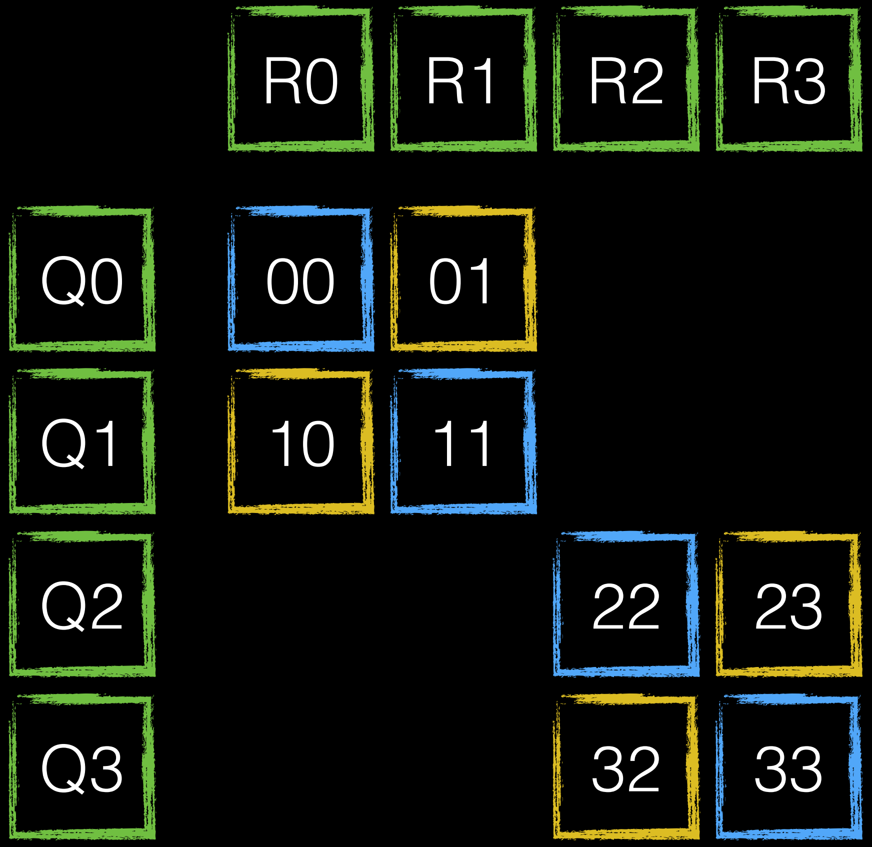
Micro-Kernel

```
LOAD          Q
LOAD          R
FMA           Q, R, C03_0
SHUFFLE
FMA           Q, R, C03_1
PERMUTE2F128
FMA           Q, R, C03_2
SHUFFLE
FMA           Q, R, C03_3
```



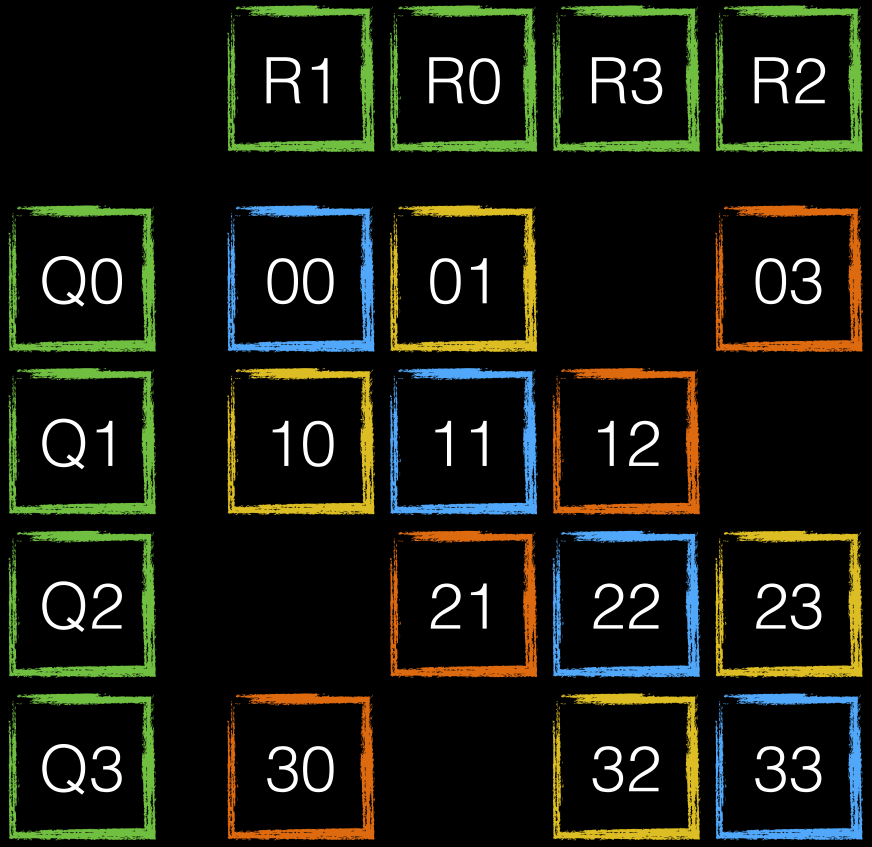
Micro-Kernel

```
LOAD          Q
LOAD          R
FMA           Q, R, C03_0
SHUFFLE
FMA           Q, R, C03_1
PERMUTE2F128
FMA           Q, R, C03_2
SHUFFLE
FMA           Q, R, C03_3
```



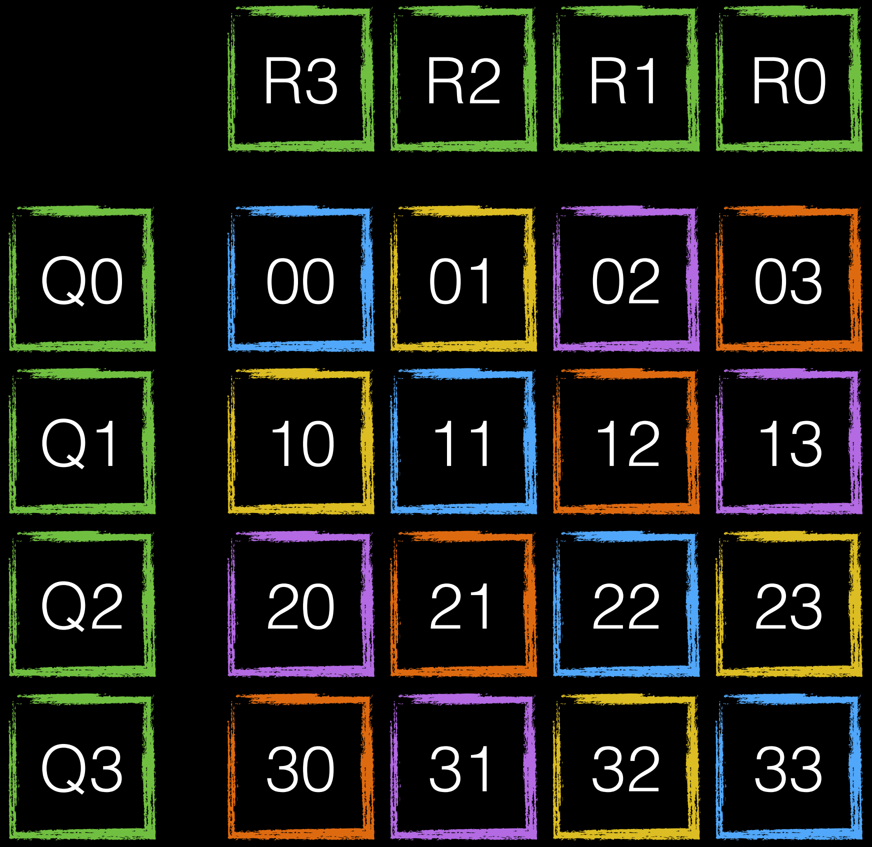
Micro-Kernel

```
LOAD          Q
LOAD          R
FMA           Q, R, C03_0
SHUFFLE
FMA           Q, R, C03_1
PERMUTE2F128
FMA           Q, R, C03_2
SHUFFLE
FMA           Q, R, C03_3
```



Micro-Kernel

```
LOAD            Q
LOAD            R
FMA             Q,  R,  C03_0
SHUFFLE
FMA             Q,  R,  C03_1
PERMUTE2F128
FMA             Q,  R,  C03_2
SHUFFLE
FMA             Q,  R,  C03_3
```



Micro-Kernel with p-norm

```
LOAD         Q
LOAD         R
FMA          Q, R, C03_0
SHUFFLE
FMA          Q, R, C03_1
PERMUTE2F128
FMA          Q, R, C03_2
SHUFFLE
FMA          Q, R, C03_3
```

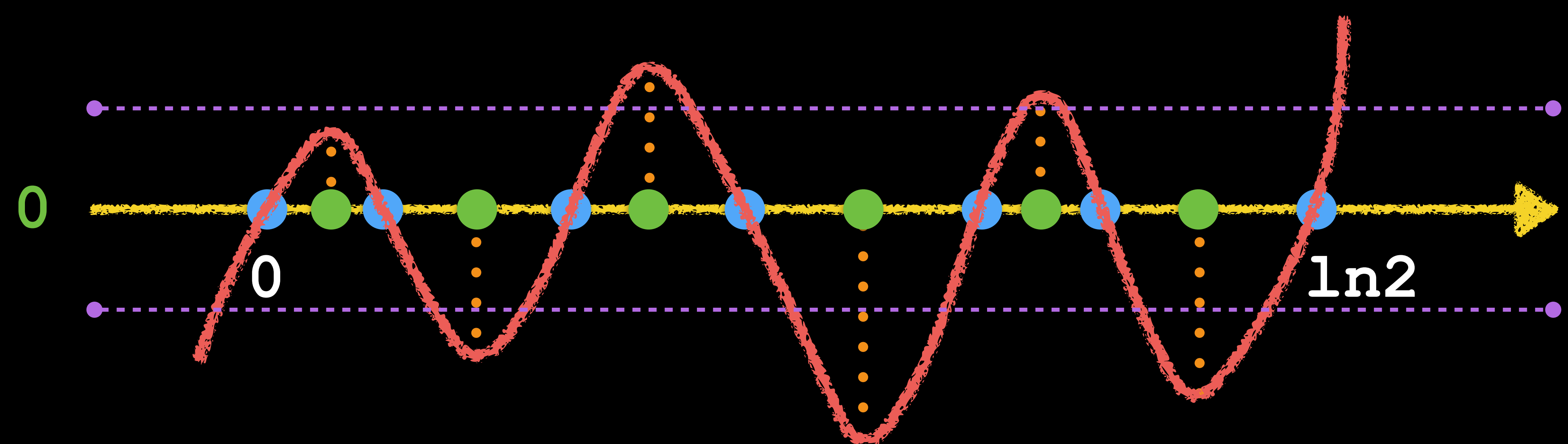
```
1-norm
SUB
AND (flip signed bit)
ADD

inf-norm
SUB
AND (flip signed bit)
MAX

p-norm
SUB
POW (SVML)
ADD
```

Vectorized Math Functions

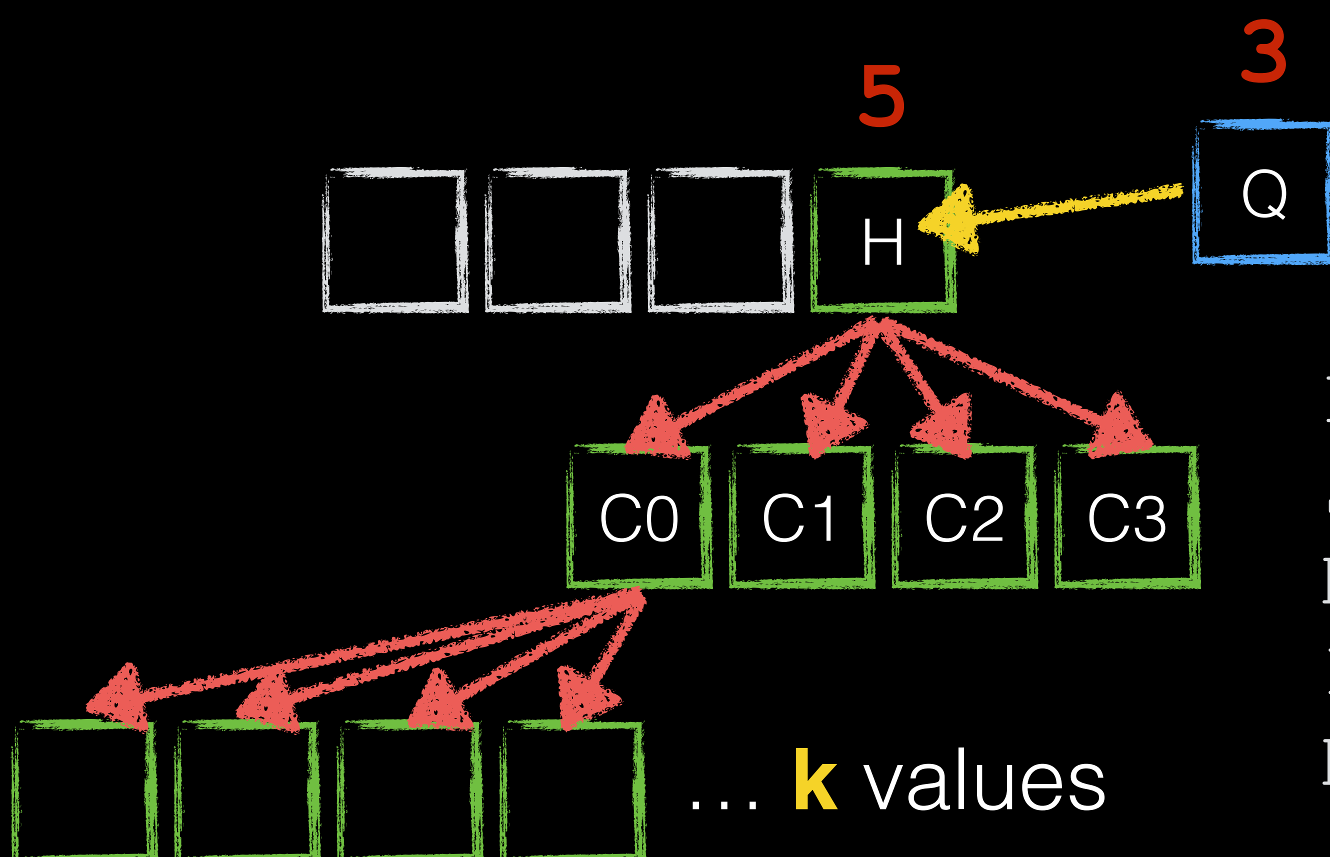
- With a high precision (**20** digits in decimal), **Remez exchange algorithm** can generate an 11 order near minimax polynomial with **1E-18** relative error.

$$P_{11}(x) = c_{11} + (\ldots + (c_5 + (c_4 + (c_3 + (c_2 + (c_1 + c_0 x)x)x)x)x)x\ldots)x$$

= 1ADD + 11FMA



Vectorized Max Heap



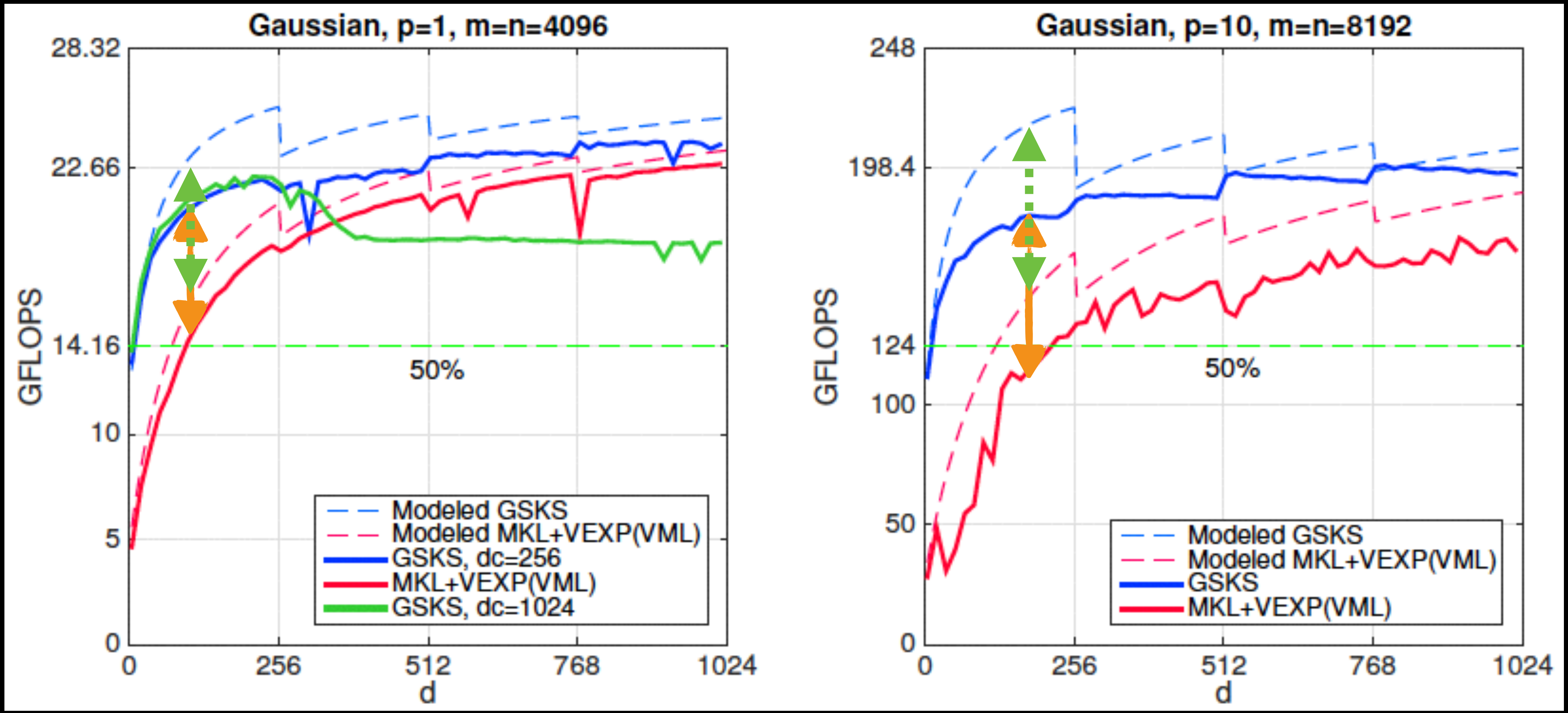

```
LOAD          C
SHUFFLE       D, C, 0x5
MAX           D, C
PERMUTE2F128  C, D, 0x1
MAX           D, C
```

Find the max child

```
C:[1,3,4,2]->[4,3,4,3]->[4,4,4,4]
D:[4,2,1,3]  [3,4,3,4]
```

GSKS Efficiency Analysis

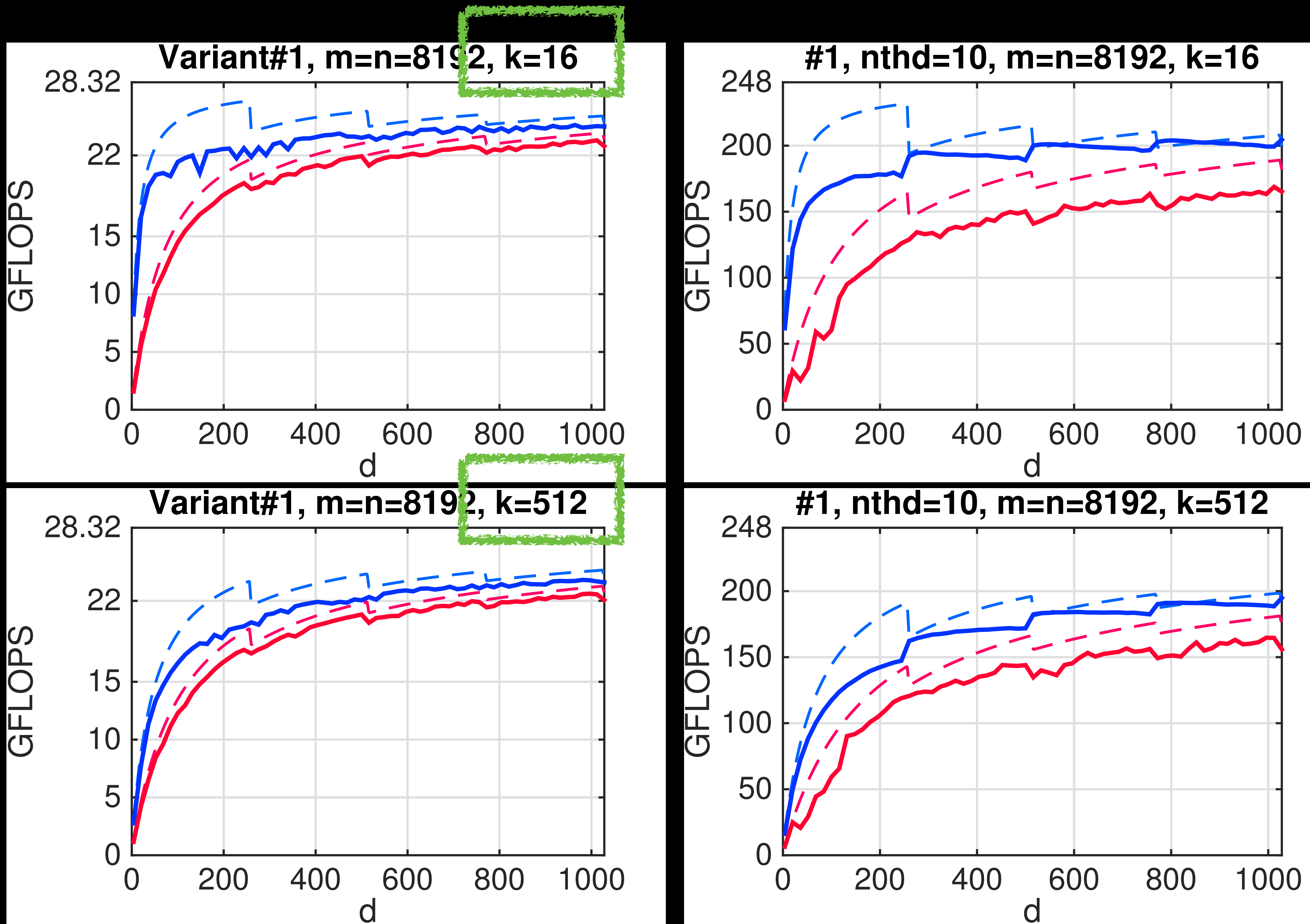
$T_{BLAS} = T_{GSKS} + T_R + T_Q + T_K$



$$\frac{mn(2d+36)}{T_{GSKS}} - \frac{mn(2d+36)}{T_{BLAS}} \stackrel{?}{=}$$

GSKNN Efficiency Graphs

Memory Bound



Conclusion

- The GEMM approach in N-body problems is a good example to show the current BLAS library is lacking flexibility for lower level integration.
- The algorithmic innovation of GSKS and GSKNN is to break through the interface, seeking for the lowest memory complexity.
- We exploit these observations with the help of the **BLIS** framework.
- Ongoing work includes other operations. e.g. kernel inversion, **k**-meaning clustering. Port to GPU and other accelerators.

Question?

GSKS GSKNN

github.com/ChenhanYu/ks github.com/ChenhanYu/rnn