The Power of Less: Harnessing Sparsity for Performance Optimization

Maryam Mehri Dehnavi

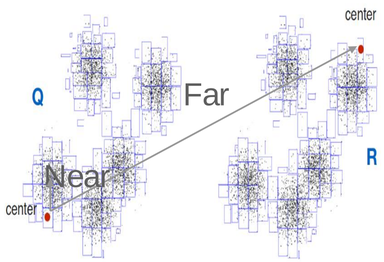
University of Toronto

Microsoft Research

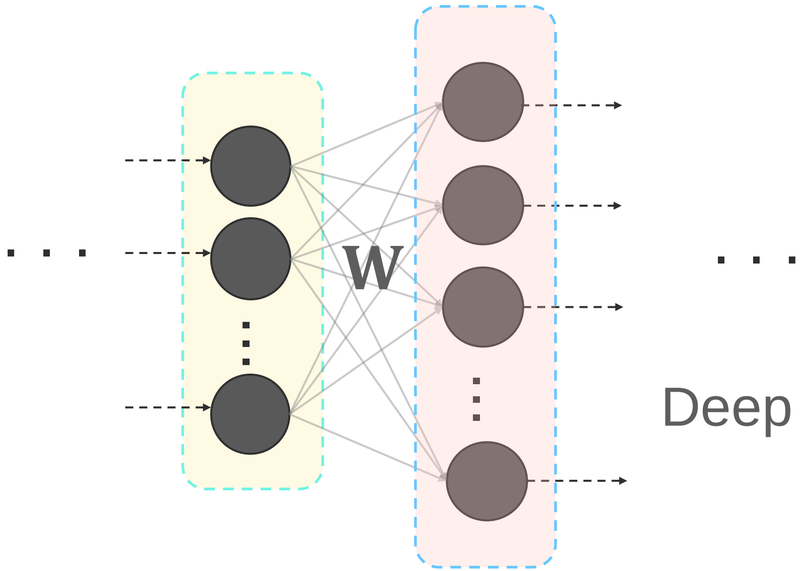
http://www.paramathic.com

Sparsity is Everywhere

Machine Learning

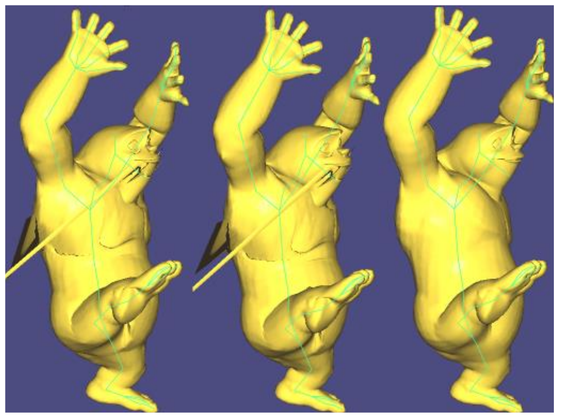


Clustering

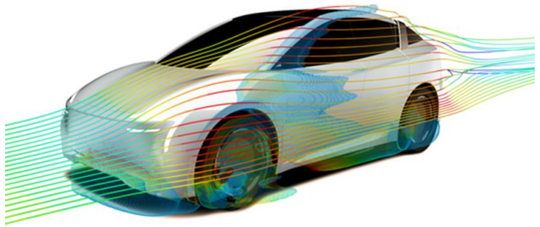


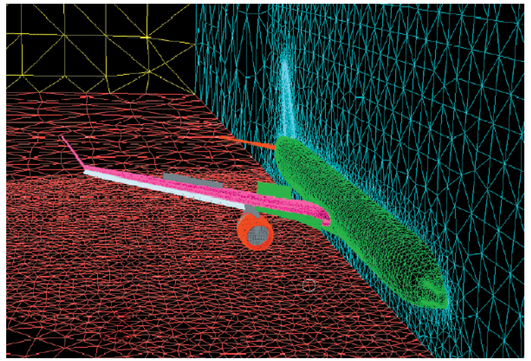
Deep Neural Networks

Science and Engineering



Graphics Simulations



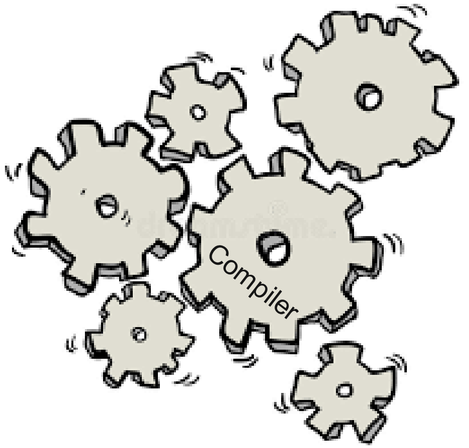


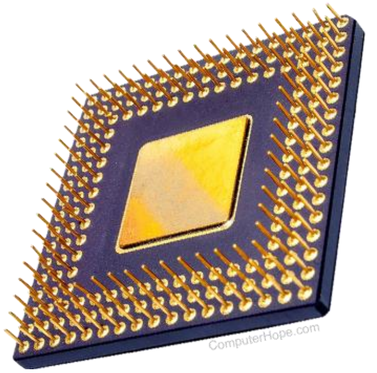
Fluid Dynamics

The World is Built for Dense!

- Hardware utilization optimizations: prefetching, branch predictors, registers, caches, …
- Compiler optimizations: tiling, unrolling, vectorization, parallelization, …
- Libraries: BLAS, LINPACK, LAPACK

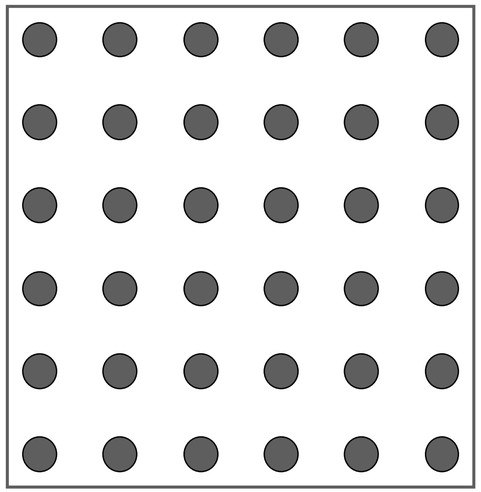
L A P A C K
L -A P -A C -K
L A P A -C -K
L -A P -A -C K
L A -P -A C K
L -A -P A C -K



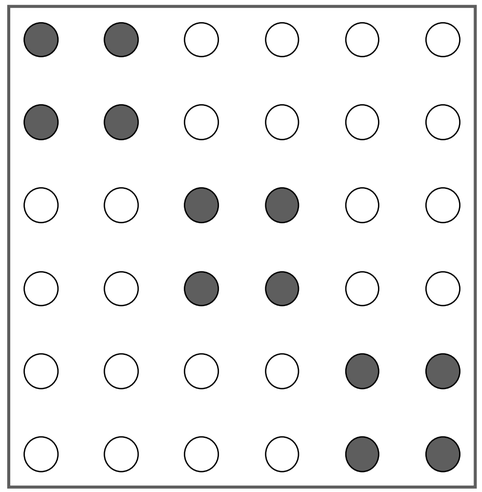


Sparsity Pattern and Ratio

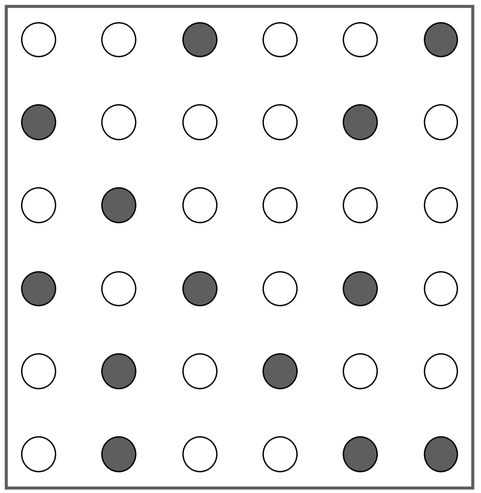
The pattern and location of non-zeros in a sparse matrix is called the **sparsity pattern** and the ratio of non-zeros over all the elements in the matrix is **density ratio**.



Dense A

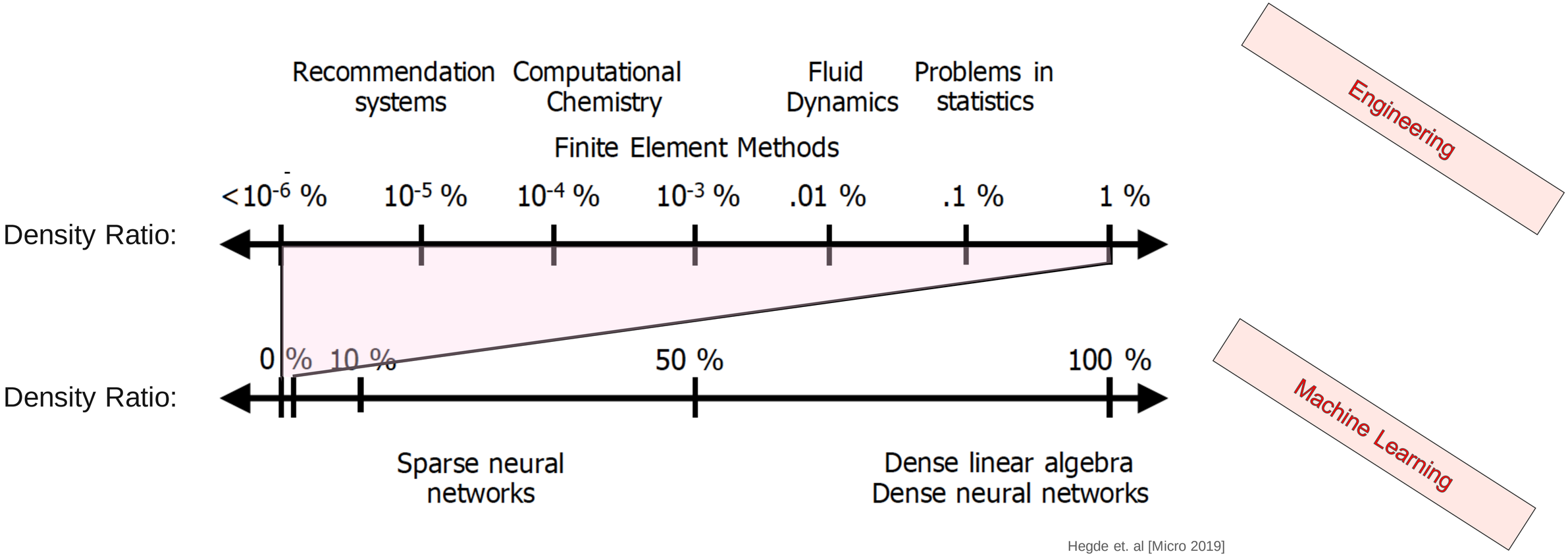


Sparse B



Sparse C

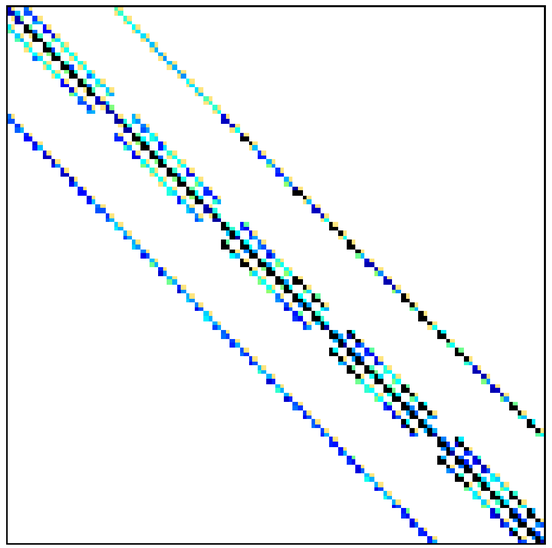
Density Ratios in Machine Learning vs. Engineering



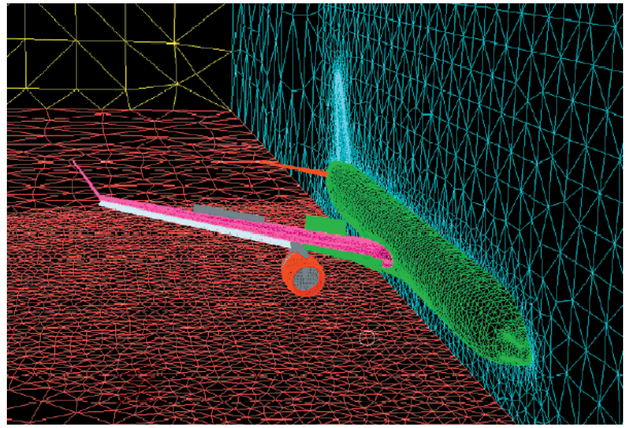
Hegde et. al [Micro 2019]

On average DNN matrices are 13x less sparse compared to scientific matrices.

Sparsity Patterns in Machine Learning vs. Engineering



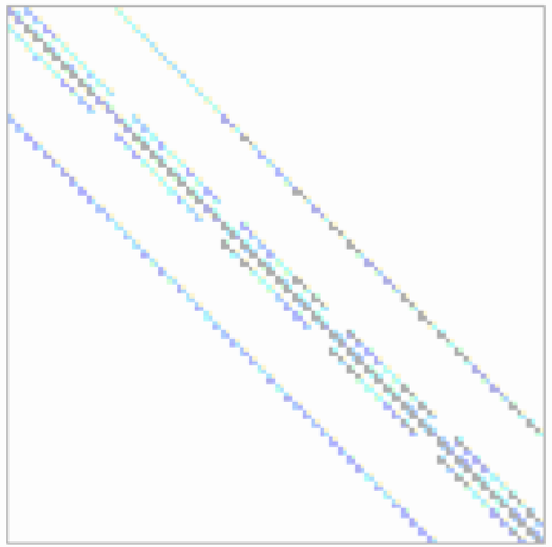
Density Ratio: 2%



Grid structure, physical properties, etc.

Engineering

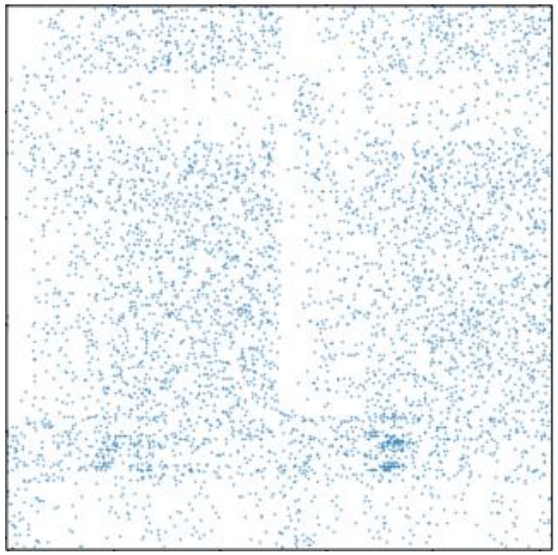
Sparsity Patterns in Machine Learning vs. Engineering



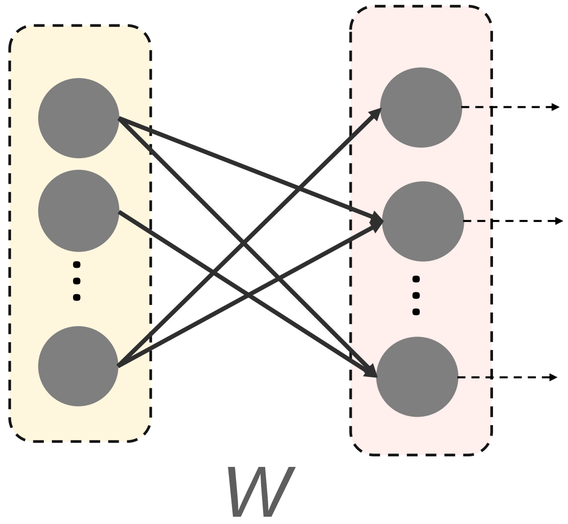
Computational Fluid Dynamics Problem

Density Ratio: 98%

Engineering

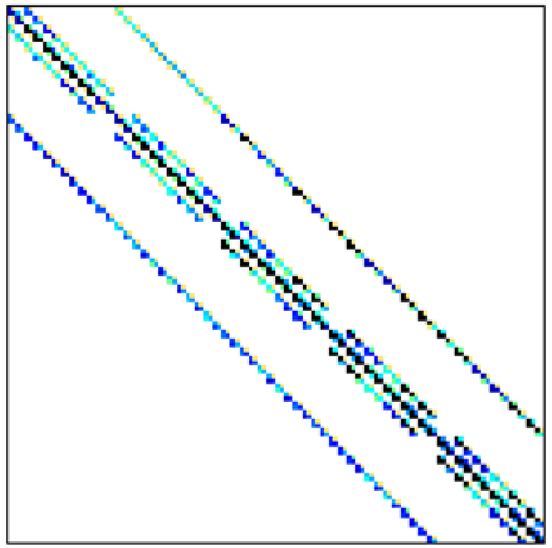


Density Ratio: 2%



Machine Learning

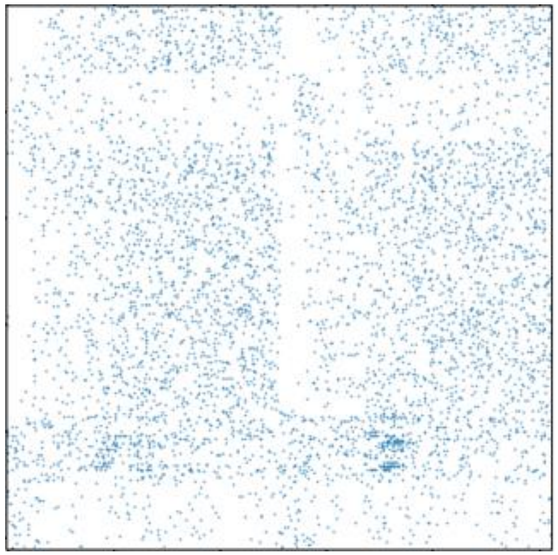
Sparsity Patterns in Machine Learning vs. Engineering



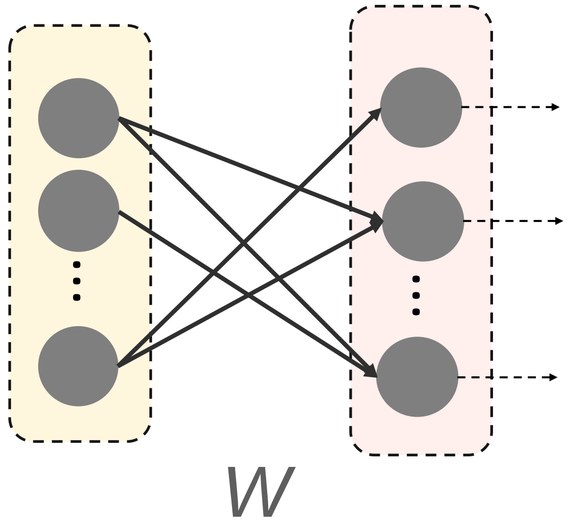
Computational Fluid Dynamics Problem
Density Ratio: 98%

Sparsity is often static or moderately changes

Engineering



Density Ratio: 2%



Weight sparsity static during inference

Machine Learning

There is no One-Size-Fits-All Approach!

Thesis

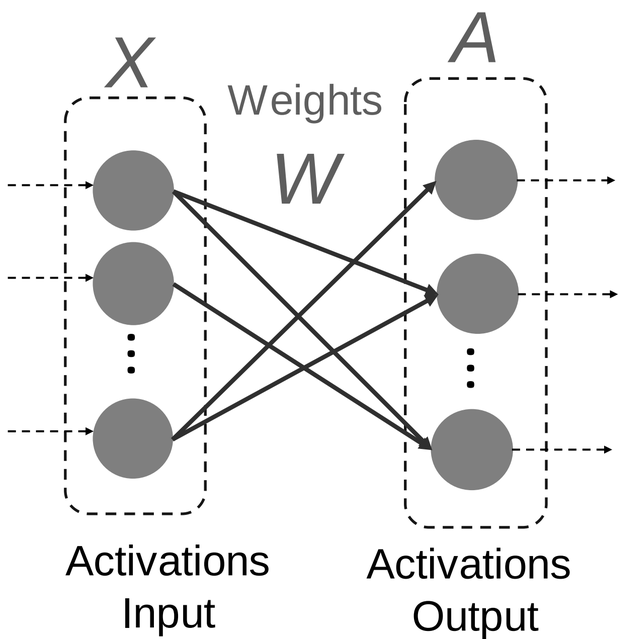
Sparsity is often static or changes moderately in most simulations.

Key contribution

Inspect sparsity patterns to automatically generate highly-optimized code for sparsity.

Part 1: Sparse Matrix Multiplication in ML & Engineering

L, ...⟵ **factorization**(*W*)
while (*not converged*) {
...
Solve: $Lx_{k+1} = b$
$r_{k+1} = r_k - \alpha Wx_{k+1}$
....
}



$A=WX$

Part 1

Matrix Multiplication routines

Engineering

Machine Learning

Part 1: Sparse Matrix Multiplication in ML & Engineering

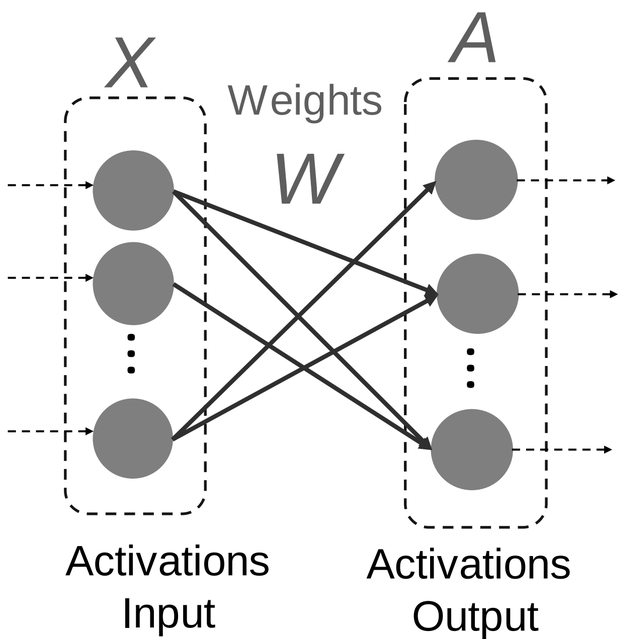
$L, \ldots \longleftarrow$ **factorization**(W)
while (*not converged*) {
…
$Solve: Lx_{k+1} = b$
$r_{k+1} = r_k - \alpha Wx_{k+1}$
….
}

Part 1

Matrix Multiplication routines

Engineering

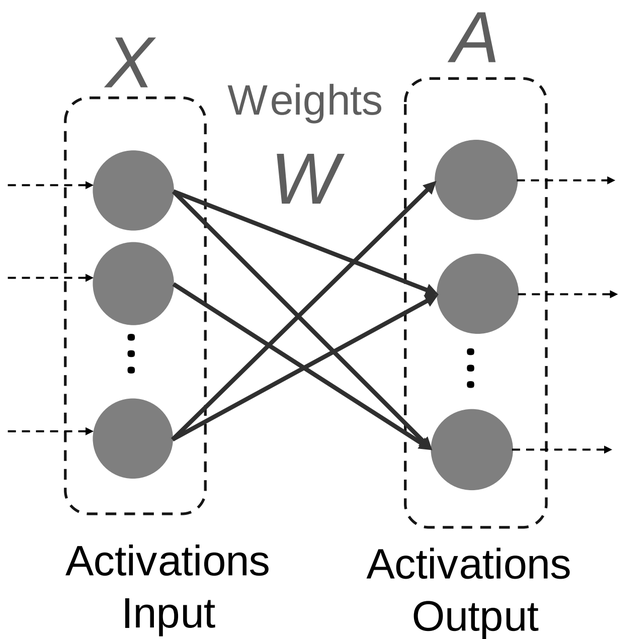
Machine Learning



$A=WX$

Part 1: Sparse Matrix Multiplication in ML & Engineering

```
L, …⟵ factorization(W)
while (not converged) {
…
  Solve: Lx_{k+1} = b
  r_{k+1} = r_k - αWx_{k+1}
….
}
```



$A=WX$

Part 1

Matrix Multiplication routines

Engineering

Machine Learning

Inspect memory access patterns for spatial and temporal locality

Part 2: Sparse Solvers in Engineering

L, …⟵ **factorization**(*W*)
while (*not converged*) {
…
Solve: $Lx_{k+1} = b$
$r_{k+1} = r_k - \alpha W x_{k+1}$
….
}

Engineering

PART 2

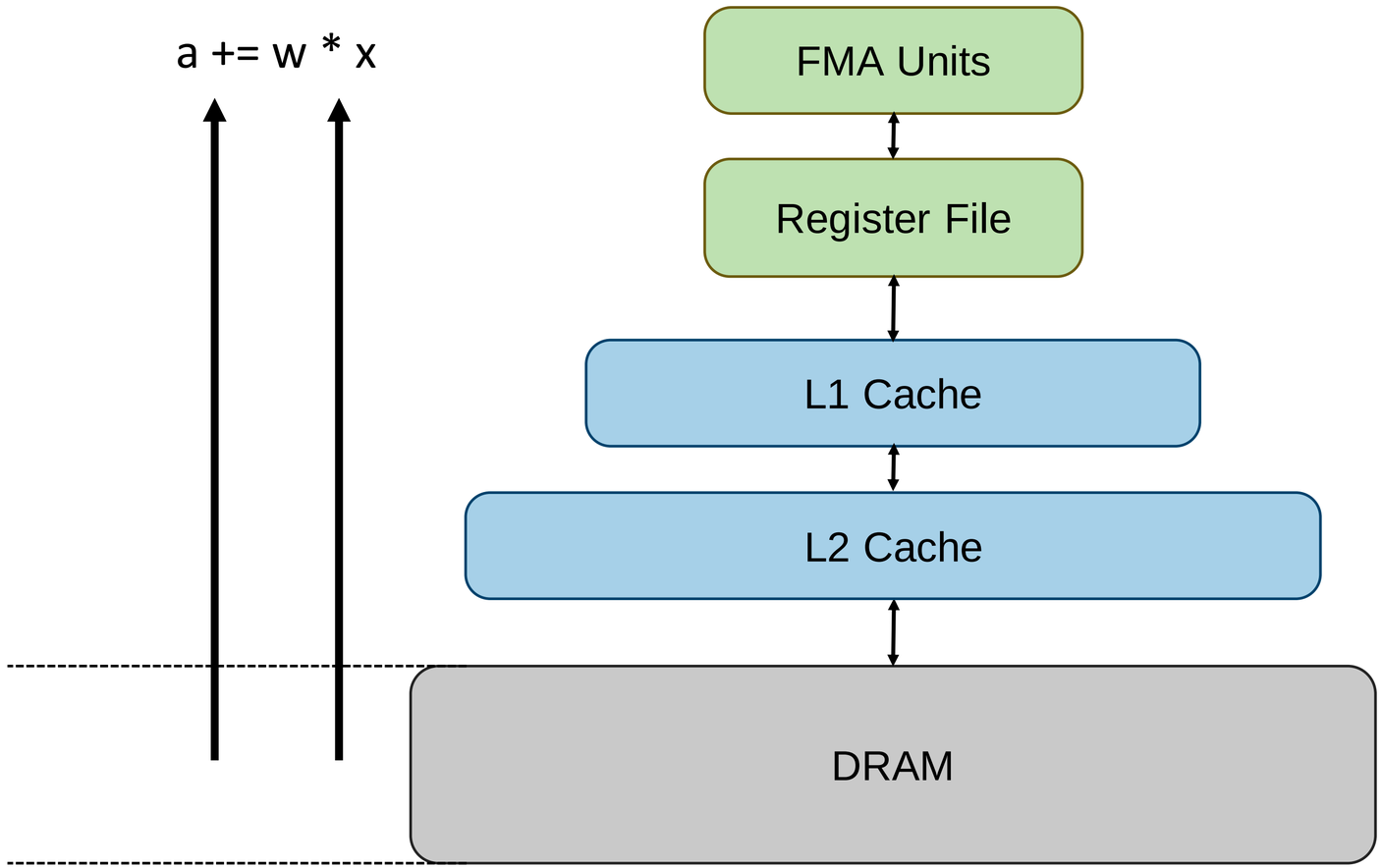
Solver routines

LU factorization, forward-backward solvers, etc.

Inspect data dependence patterns for pruning and parallelism, etc.

CPU Memory Hierarchy

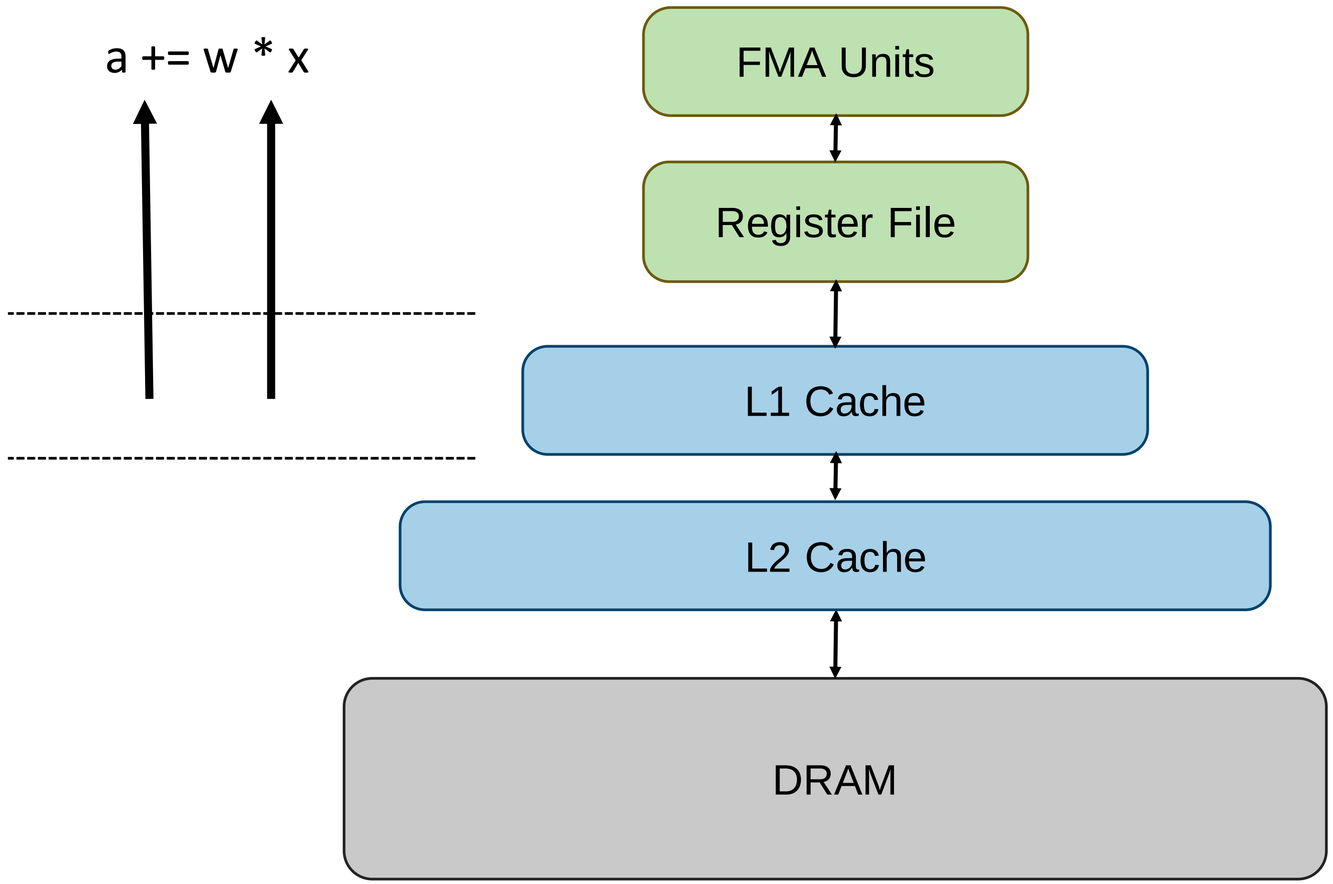
To compute $A = Wx$ values must be moved up the memory hierarchy.



Temporal Locality

To compute $A = Wx$ values must be moved up the memory hierarchy.

Temporal Locality: Reuse data while in fast memory.

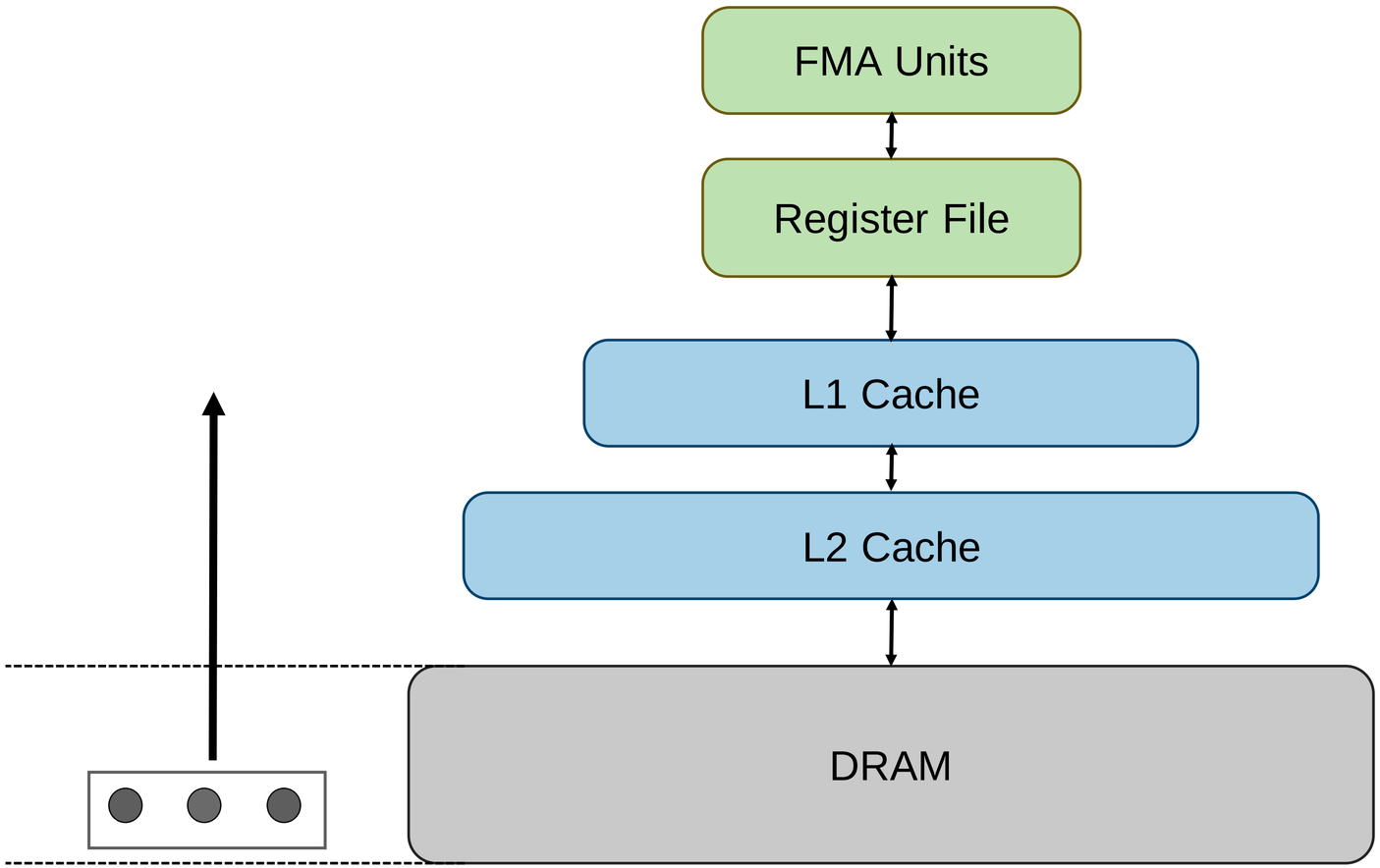


Spatial Locality

To compute $A = Wx$ values must be moved up the memory hierarchy.

Temporal Locality: Reuse data while in fast memory.

Spatial Locality: Access close-by data.

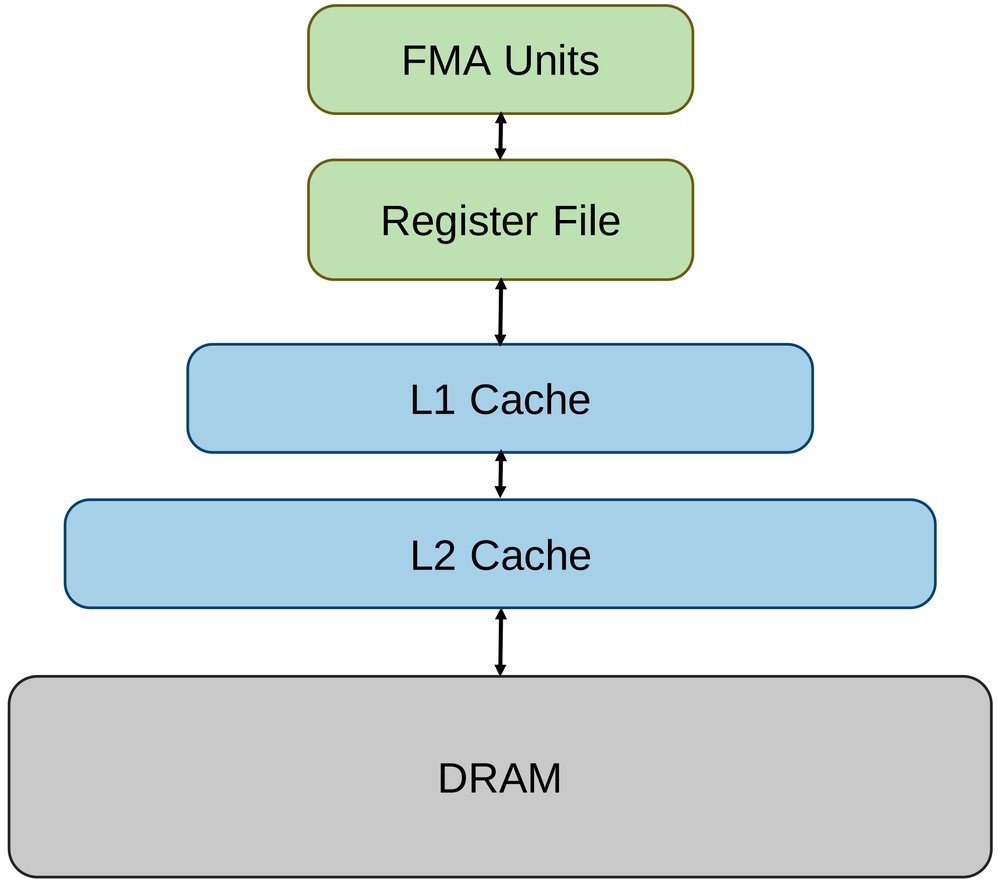


What’s next!

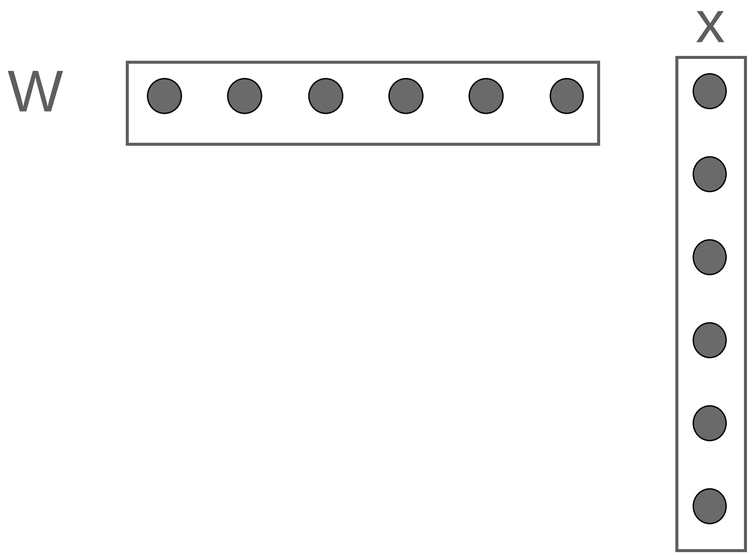
Instruction reordering → Engineering problems ✓

Machine Learning ✗

Sparse Register Tiling → Machine Learning ✓



Strided Memory Accesses and Locality



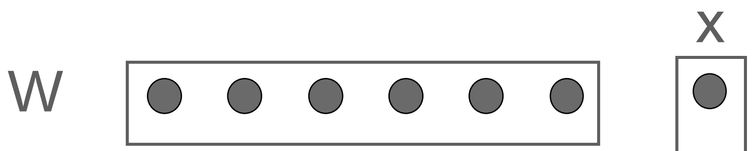

```
for i = 0...5
    a[1] += W[i] * x[1 * i].
```

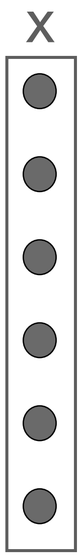
Zero-Strided Access
(Temporal locality)

Unit-Strided Access
(Spatial locality)

✓

Strided Memory Accesses and Locality

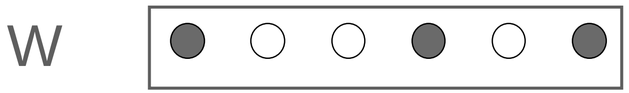




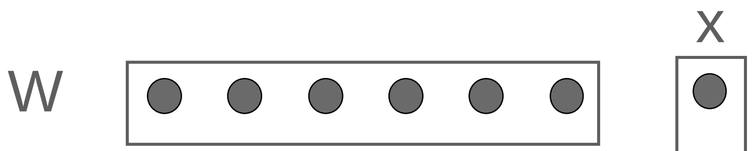

```
for i = 0...5
    a[1] += W[i] * x[1 * i].
```

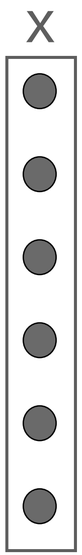
Zero-Strided Access
(Temporal locality)

Unit-Strided Access
(Spatial locality)



Strided Memory Accesses and Locality





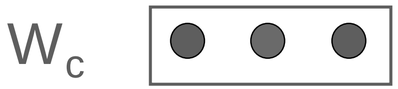

```
for i = 0...5
    a[1] += W[i] * x[1 * i].
```

Zero-Strided Access
(Temporal locality)

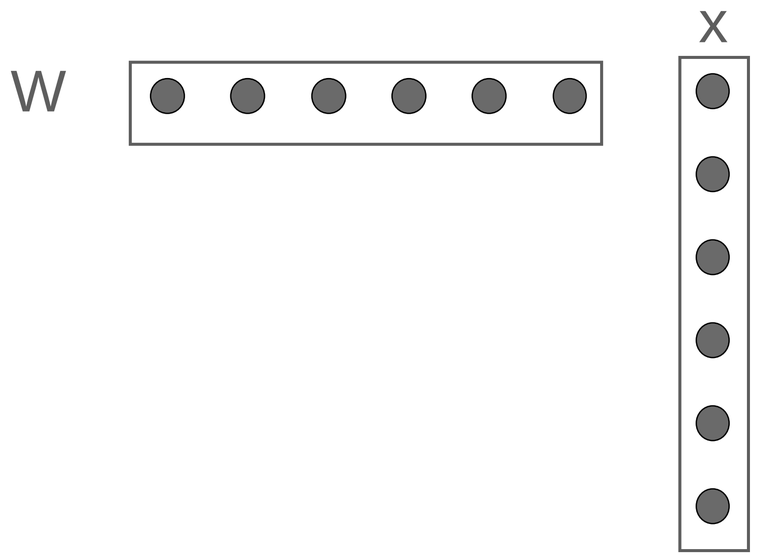
Unit-Strided Access
(Spatial locality)

✓

Column 1 3 6



Strided Memory Accesses and Locality

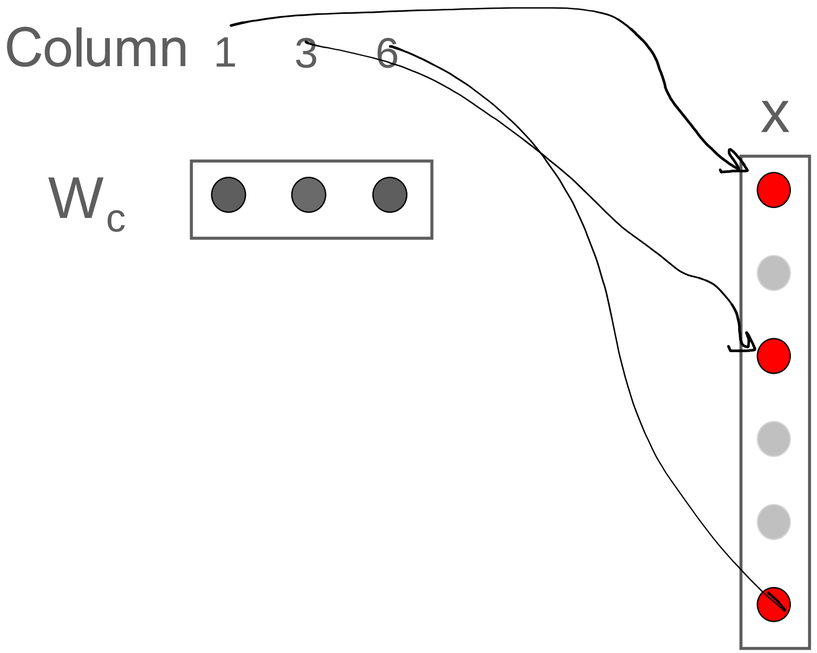



```
for i = 0...5
    a[1] += W[i] * x[1 * i].
```

Zero-Strided Access
(Temporal locality)

Unit-Strided Access
(Spatial locality)

✓

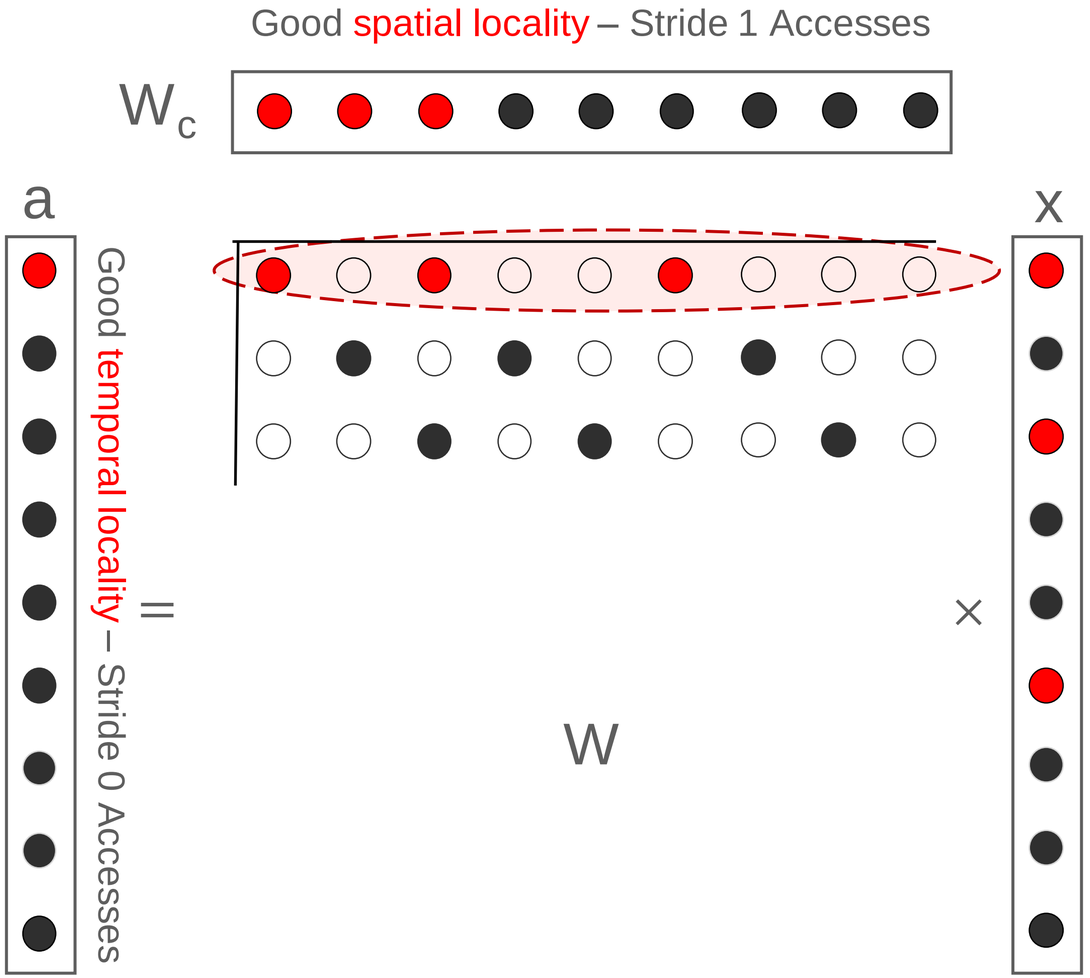



```
for i = 0...2
    a[1] += Wc[i] * x[column[i]]
```

Unknown stride Access

✗

Strided Accesses in SpMV



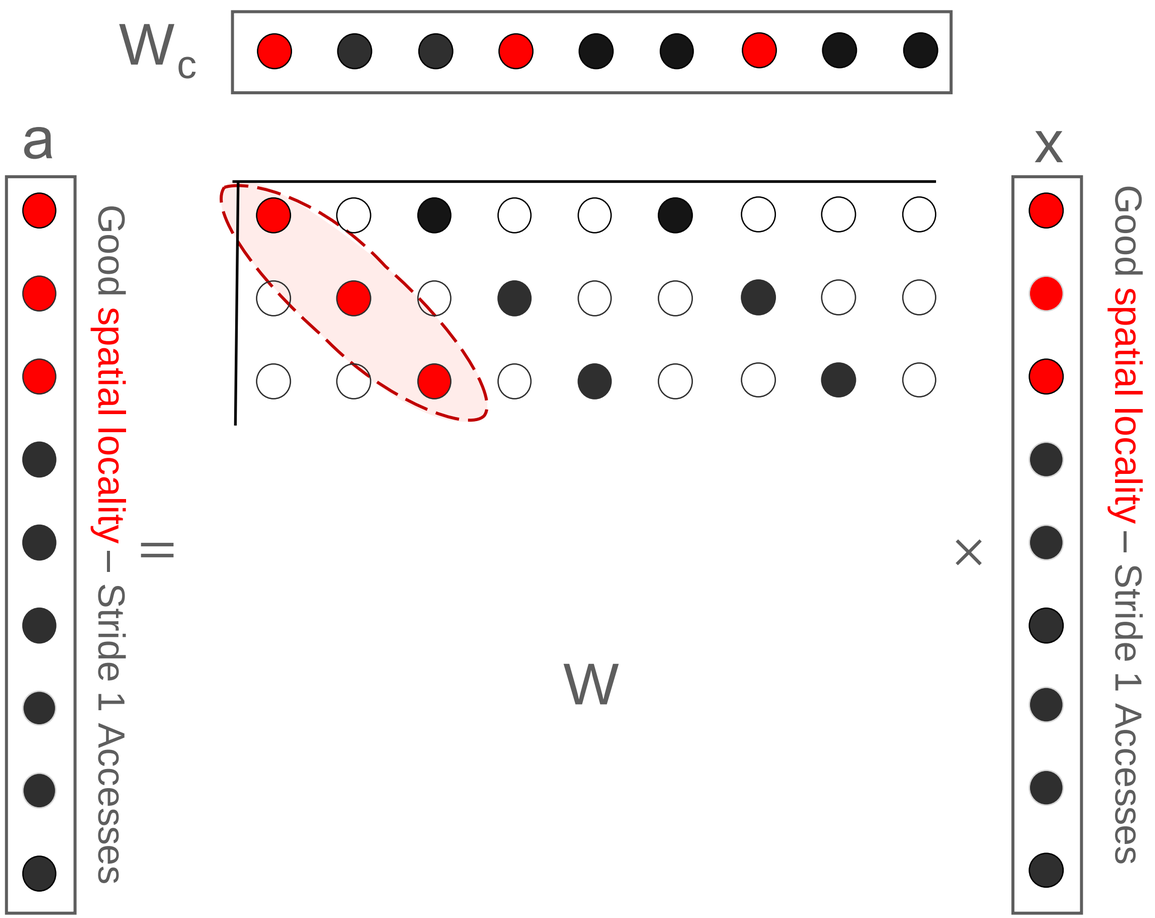
Instruction Order

a[0] += W_c[0] * x[0];
a[0] += W_c[1] * x[2];
a[0] += W_c[2] * x[5];
a[1] += W_c[3] * x[1];
a[1] += W_c[4] * x[3];
a[1] += W_c[5] * x[6];
a[2] += W_c[6] * x[2];
a[2] += W_c[7] * x[4];
a[2] += W_c[8] * x[7];

Code

```
for i = 0...3 {
        a[i] += Wc[i*3] * x[i];
        a[i] += Wc[i*3+1] * x[i+2];
        a[i] += Wc[i*3+2] * x[i+5];
}
```

Strided Accesses in the New Code



Instruction Order

a[0] += W_c[0] * x[0];
a[1] += W_c[3] * x[1];
a[2] += W_c[6] * x[2];
a[0] += W_c[1] * x[2];
a[1] += W_c[5] * x[6];
a[2] += W_c[7] * x[4];
a[0] += W_c[2] * x[5];
a[1] += W_c[4] * x[3];
a[2] += W_c[8] * x[7];

Code

```
for i = 0...3
        a[i] += Wc[i*3] * x[i];
for i = 0...3
        a[i] += Wc[i*3+1] * x[i+2];
for i = 0...3
        a[i] += Wc[i*3+2] * x[i+5];
```

Which Code to Generate?

```
for i = 0...3
        a[i] += Wc[i*3] * x[i];
for i = 0...3
        a[i] += Wc[i*3+1] * x[i+2];
for i = 0...3
        a[i] += Wc[i*3+2] * x[i+5];
```

```
for i = 0...3 {
        a[i] += Wc[i*3] * x[i];
        a[i] += Wc[i*3+1] * x[i+2];
        a[i] += Wc[i*3+2] * x[i+5];
}
```

Strided accesses to a and x

Spatial locality Spatial locality

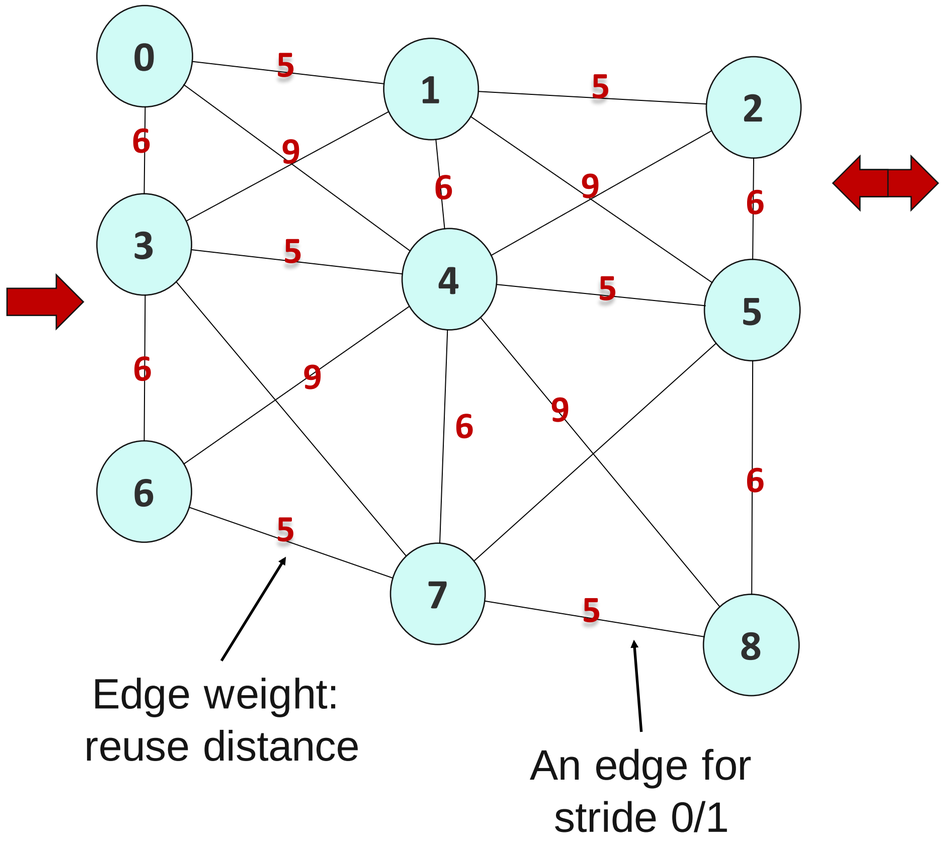
Strided accesses to W_c and a

Spatial locality Temporal locality

Locality-aware Codelet Mining (LCM)

Inspection

```
a[0] += W_c[0] * x[0];
a[0] += W_c[1] * x[2];
a[0] += W_c[2] * x[5];
a[1] += W_c[3] * x[1];
a[1] += W_c[4] * x[3];
a[1] += W_c[5] * x[6];
a[2] += W_c[6] * x[2];
a[2] += W_c[7] * x[4];
a[2] += W_c[8] * x[7];
```



<u>Minimum edge covering problem</u>:
Minimum weight-set of edges guides permutation

+

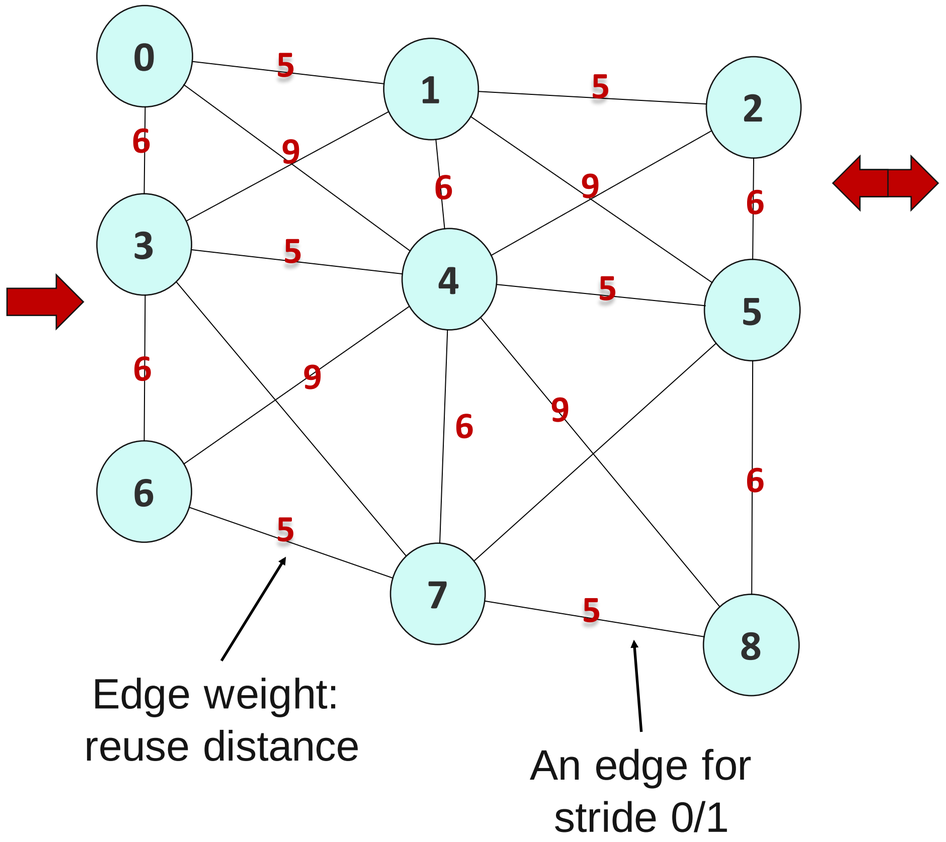
A codelet cost model

Vectorizing Sparse Matrix Computations with Partially-Strided Codelets, Cheshmi, Cetinic, Dehnavi [SC'22]

https://github.com/sparse-specialize/partially-strided-codelet

Locality-aware Codelet Mining (LCM)

```
a[0] += W_c[0] * x[0];
a[0] += W_c[1] * x[2];
a[0] += W_c[2] * x[5];
a[1] += W_c[3] * x[1];
a[1] += W_c[4] * x[3];
a[1] += W_c[5] * x[6];
a[2] += W_c[6] * x[2];
a[2] += W_c[7] * x[4];
a[2] += W_c[8] * x[7];
```



Inspection

Minimum edge covering problem:
Minimum weight-set of edges guides permutation

+

A codelet cost model

Code Gen

Generate strided and partially-strided codelets.

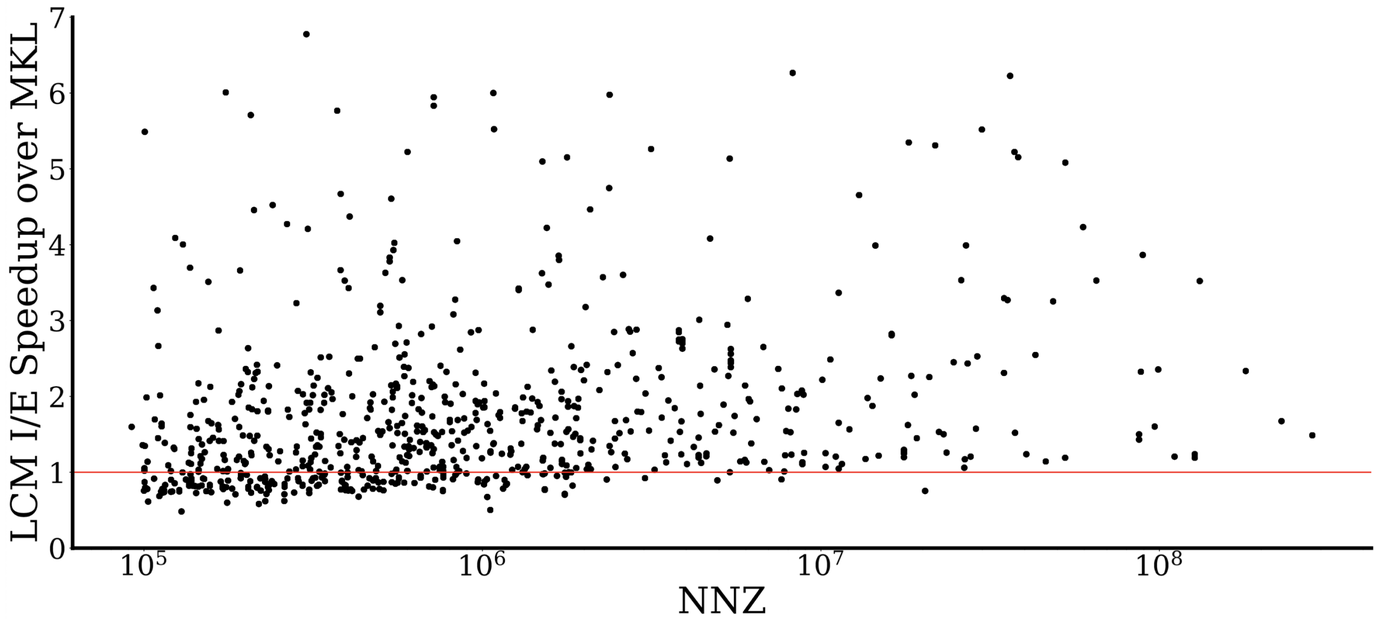
Vectorizing Sparse Matrix Computations with Partially-Strided Codelets, Cheshmi, Cetinic, Dehnavi [SC'22]

https://github.com/sparse-specialize/partially-strided-codelet

LCM on Science and Engineering Problems for SpMM

Testbed: 789 matrices from the SuiteSparse[1] repository on an Intel(R) Xeon(R) Gold 5115 CPU (20 cores, 64GB main memory).

👍 Engineering



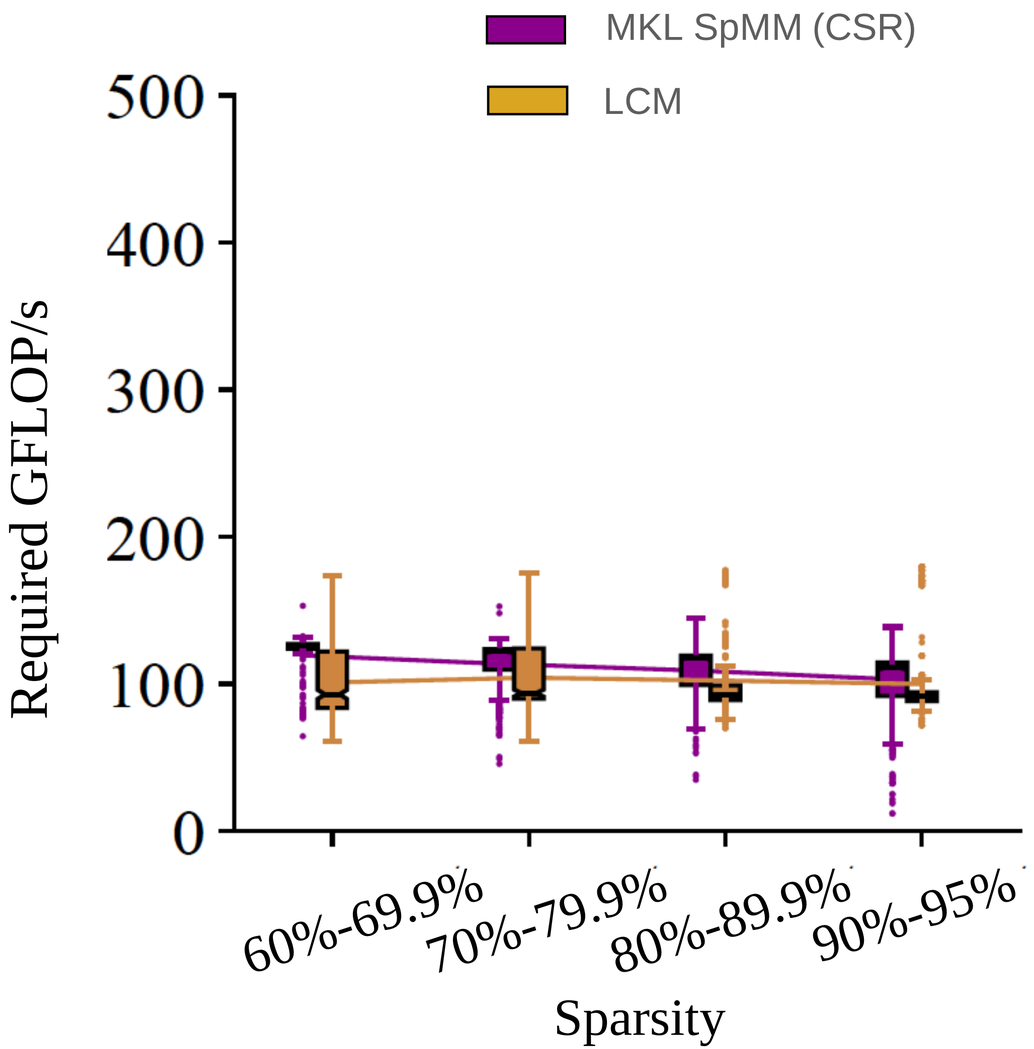
LCM is faster than Intel **MKL[2]** with an average speedup of **1.75x** for *Science and Engineering problems.*

[1] *T. A. Davis and Y. Hu, "The university of florida sparse matrix collection," ACM Transactions on Mathematical Software (TOMS), vol. 38, no. 1, pp. 1–25, 2011.*
[2] *Intel. 2022. Intel Math Kernel Library*

LCM on Machine Learning Matrices

Testbed: Over 3000 matrices from Deep Learning Matrix Collection (DLMC[1]).

👎 Machine Learning

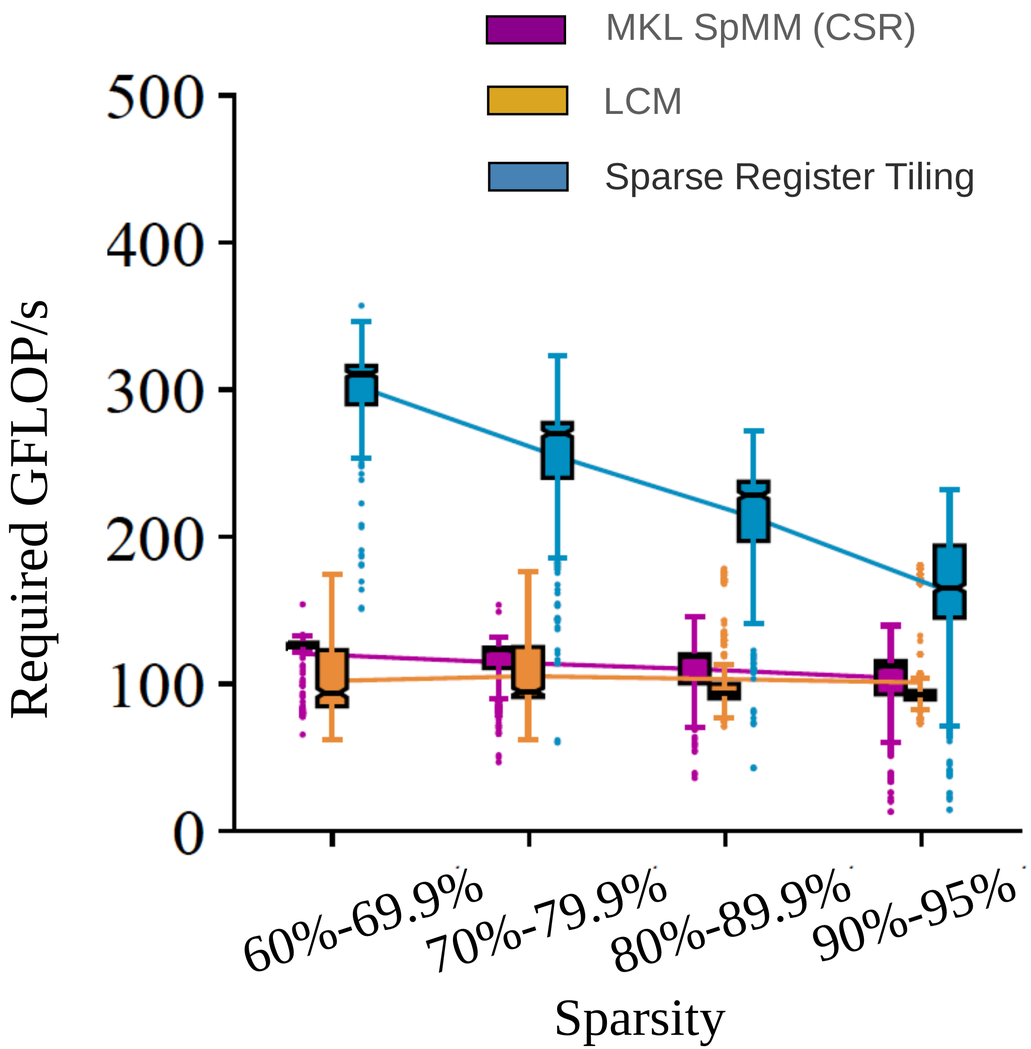


LCM does not improve the performance of *machine learning problems*.

[1]*Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. 2020. Sparse GPU Kernels for Deep Learning. SC '20.*

LCM on Machine Learning Matrices

Testbed: Over 3000 matrices from Deep Learning Matrix Collection (DLMC[1]).



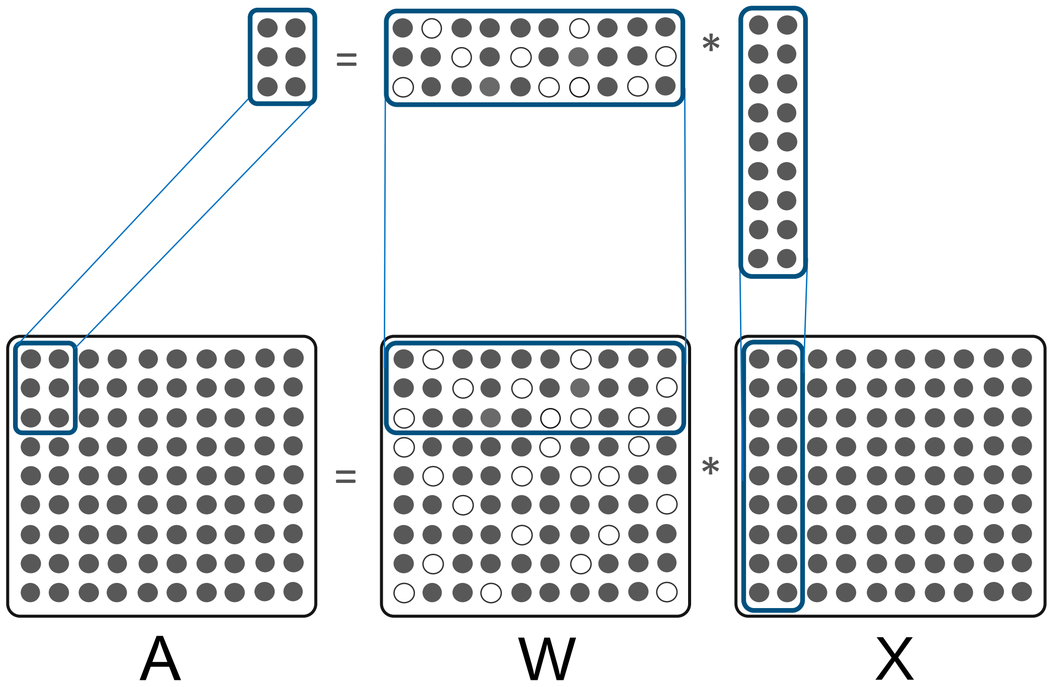
LCM does not improve the performance of *machine learning problems*.

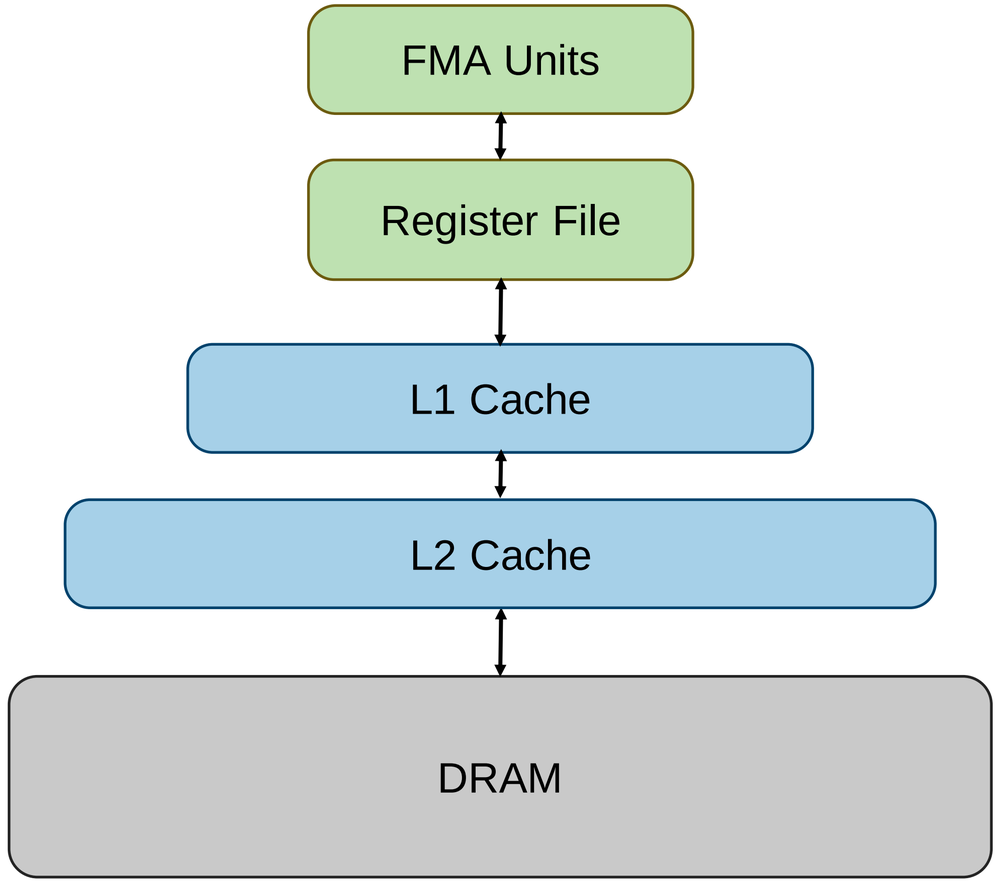
[1]*Trevor Gale, Matei Zaharia, Cliff Young, and Erich Elsen. 2020. Sparse GPU Kernels for Deep Learning. SC '20.*

Cache Tiling for Machine Learning

Because of their random sparsity patterns, in DNNs over 96% of mined patterns are unstrided in at least two of the array accesses!

The better reordering method for unstructured matrices for caches is standard tiling (used in dense problems)*.

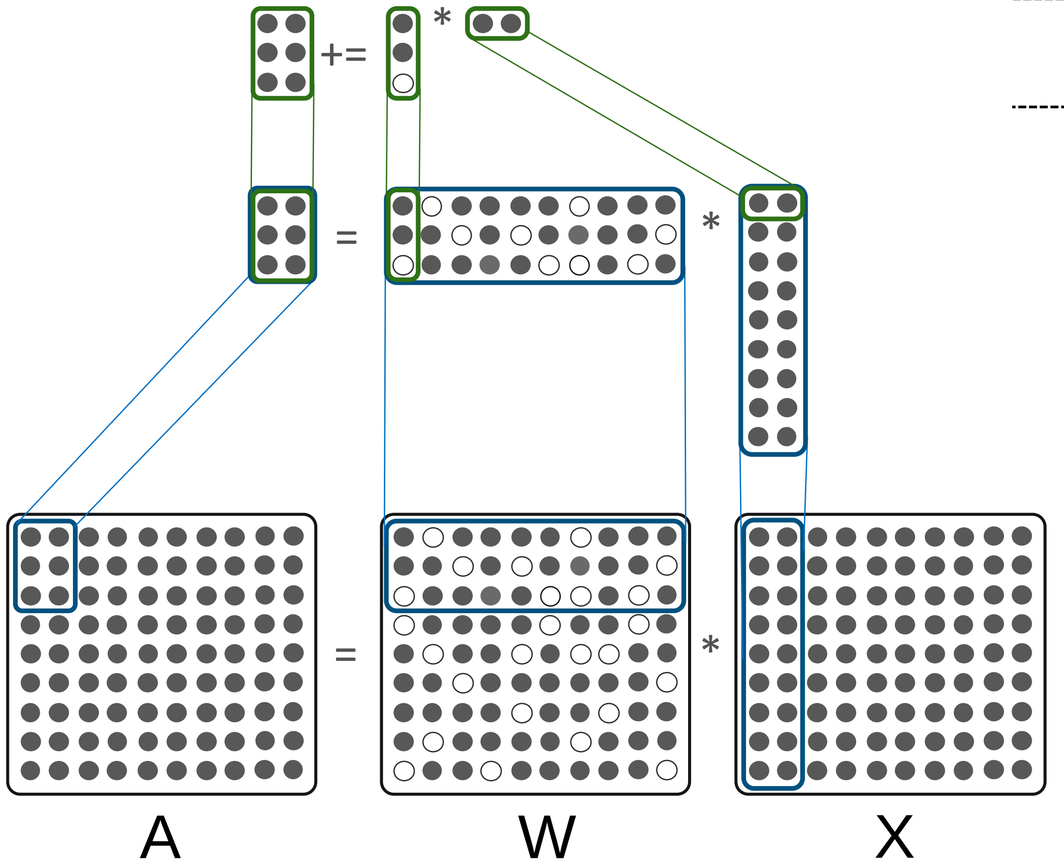


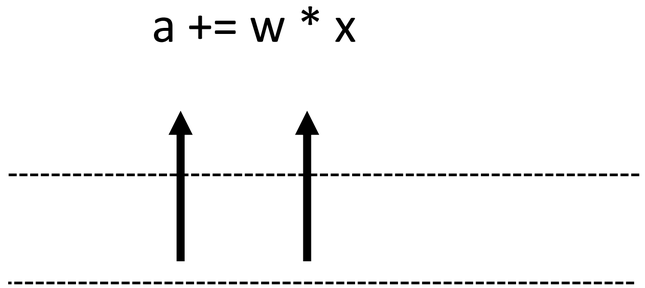


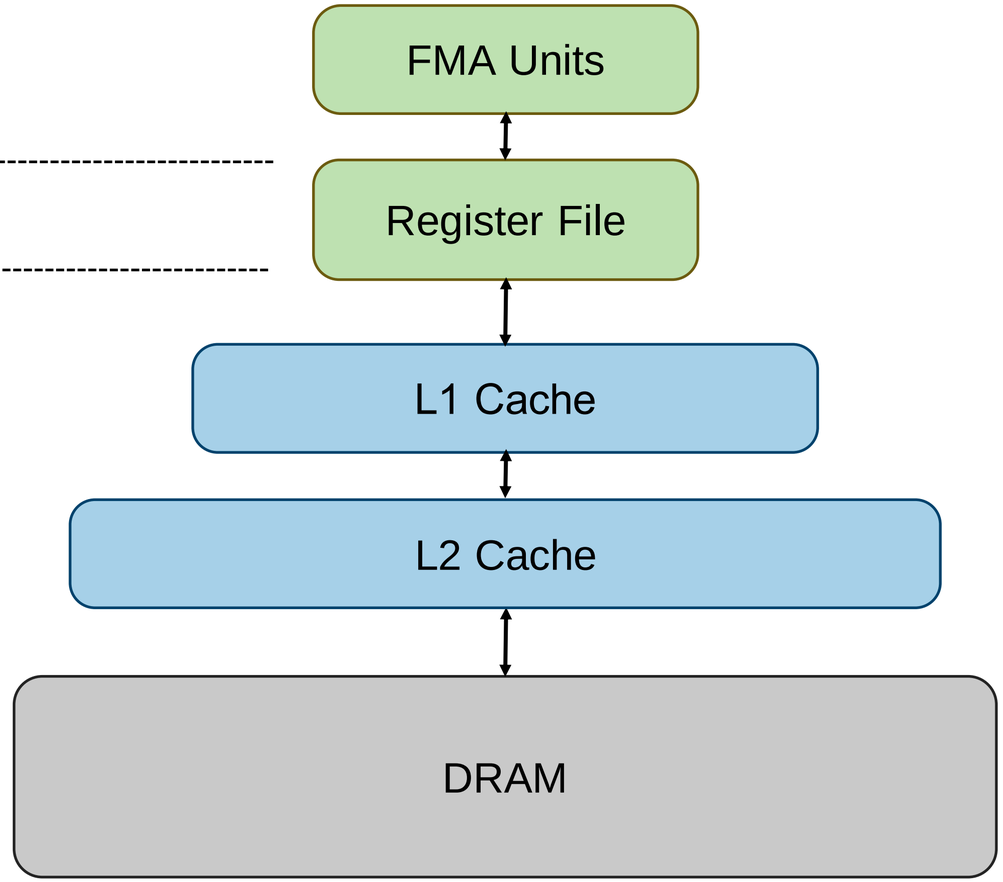
**Working set size discussions in section 3.2 of Register Tiling for Unstructured Sparsity in Neural Network Inference [PLDI'23]*

Register Tiling

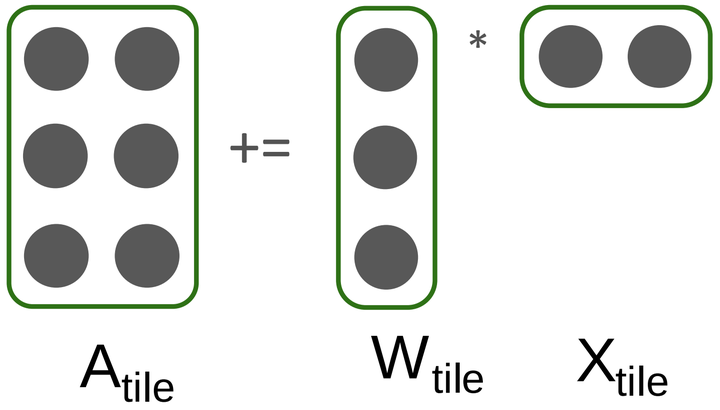
Improve locality in registers!





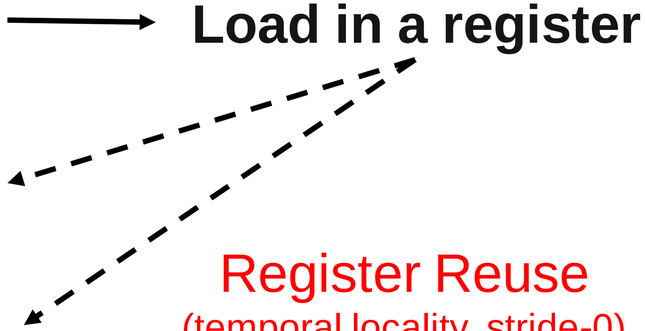


Register Reuse with Dense

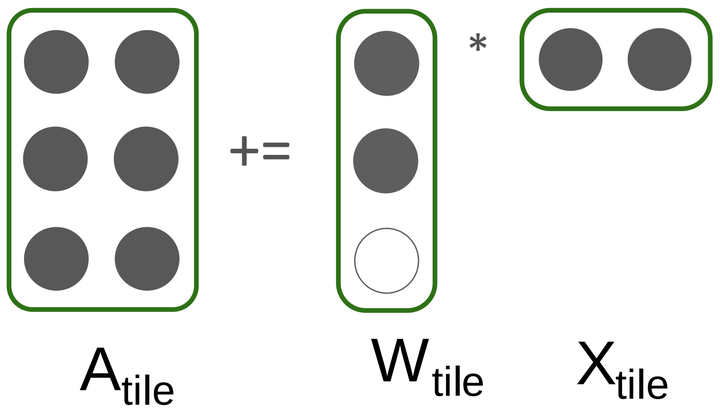


reg1	reg2
reg3	reg4
reg5	reg6
reg7	reg8
reg9	reg10
reg11	reg12

$A_{tile}[0, 0] \mathrel{+}= W_{tile}[0, k] * X_{tile}[k, 0]$
$A_{tile}[0, 1] \mathrel{+}= W_{tile}[0, k] * X_{tile}[k, 1]$
$A_{tile}[1, 0] \mathrel{+}= W_{tile}[1, k] * X_{tile}[k, 0]$
$A_{tile}[1, 1] \mathrel{+}= W_{tile}[1, k] * X_{tile}[k, 1]$
$A_{tile}[2, 0] \mathrel{+}= W_{tile}[2, k] * X_{tile}[k, 0]$
$A_{tile}[2, 1] \mathrel{+}= W_{tile}[2, k] * X_{tile}[k, 1]$



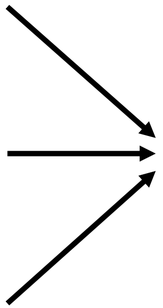
Register Reuse with Sparse



reg1	reg2
reg3	reg4
reg5	reg6
reg7	reg8
reg9	reg10
reg11	reg12

```
if (W_tile[0,k])   {
if (W_tile[1,k])   {
if (W_tile[2,k])   {
```

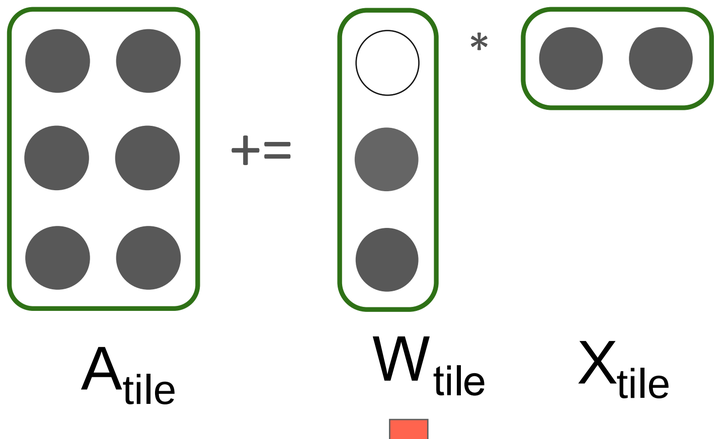
```
A_tile[0, 0] += W_tile[0, k] * X_tile[k, 0]
A_tile[0, 1] += W_tile[0, k] * X_tile[k, 1]   }
A_tile[1, 0] += W_tile[1, k] * X_tile[k, 0]
A_tile[1, 1] += W_tile[1, k] * X_tile[k, 1]   }
A_tile[2, 0] += W_tile[2, k] * X_tile[k, 0]
A_tile[2, 1] += W_tile[2, k] * X_tile[k, 1]   }
```



Unclear which instructions will execute: no register reuse

Overhead of if-conditions

If Tile Sparsity was Known



$A_{tile}[0, 0] \mathrel{+}= W_{tile}[0, k] * X_{tile}[k, 0]$

$A_{tile}[0, 1] \mathrel{+}= W_{tile}[0, k] * X_{tile}[k, 1]$

$A_{tile}[1, 0] \mathrel{+}= W_{tile}[1, k] * X_{tile}[k, 0]$

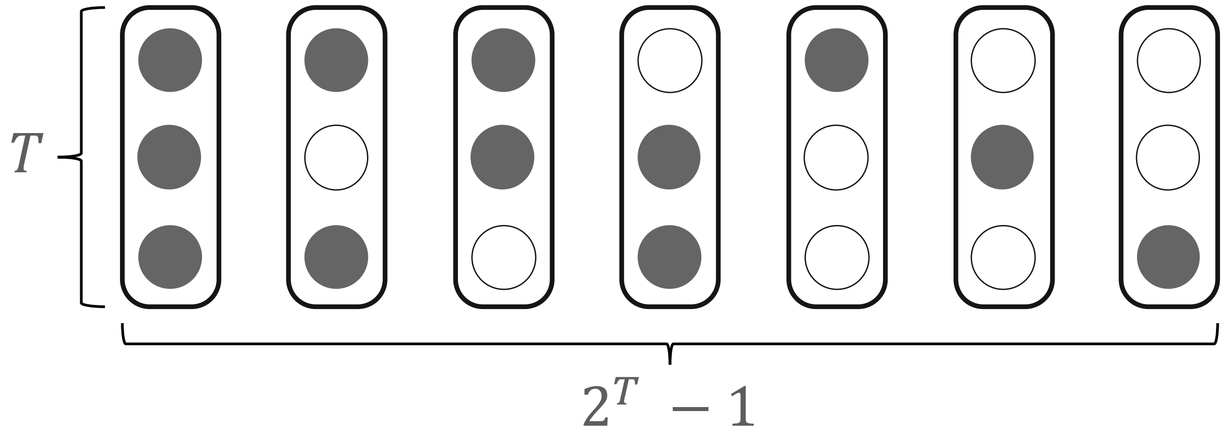
$A_{tile}[1, 1] \mathrel{+}= W_{tile}[1, k] * X_{tile}[k, 1]$

$A_{tile}[2, 0] \mathrel{+}= W_{tile}[2, k] * X_{tile}[k, 0]$

$A_{tile}[2, 1] \mathrel{+}= W_{tile}[2, k] * X_{tile}[k, 1]$

Register reuse will happen because common access patterns are known.

Generate Code for Each Possible Tile Pattern



Inspection

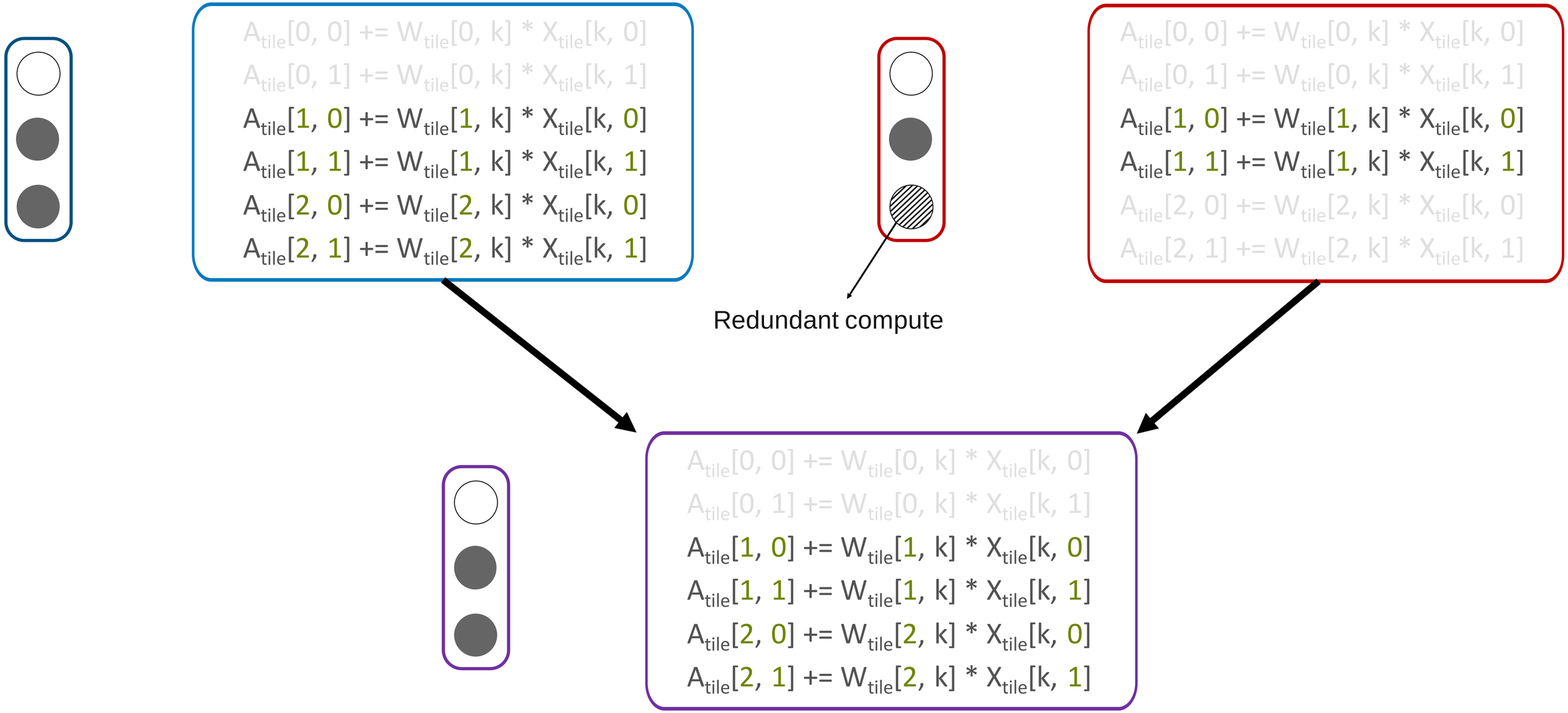
⚠ Code bloat

Solution: Sparse Jam Solver

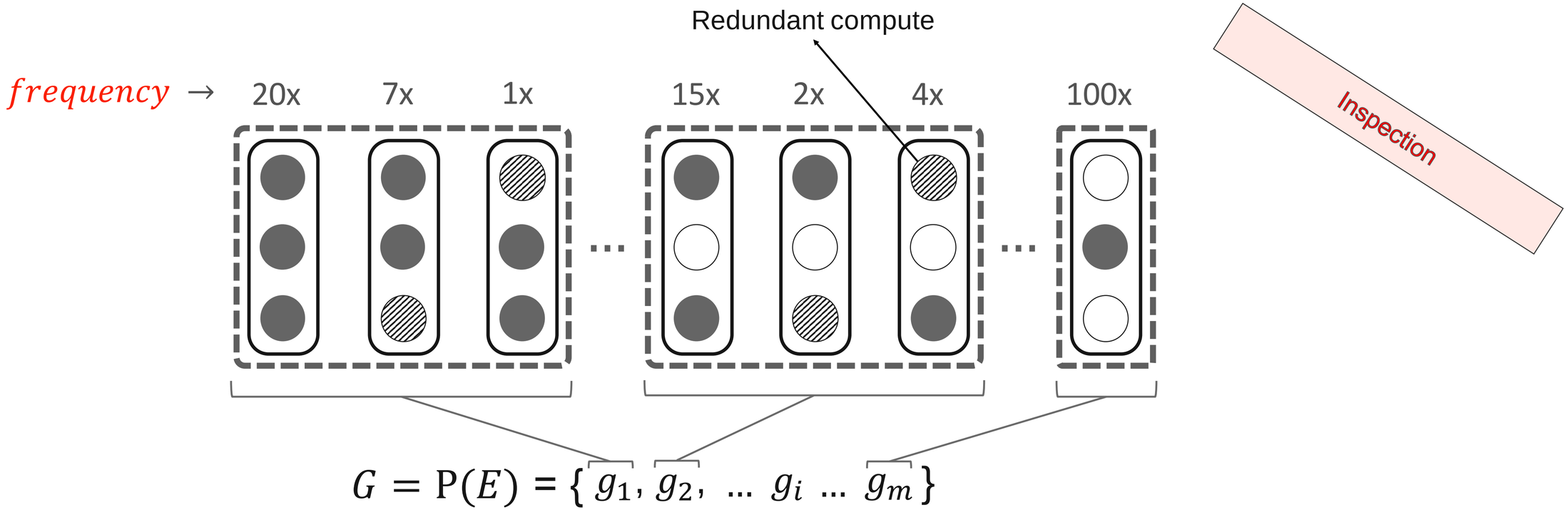
Register Tiling for Unstructured Sparsity in Neural Network Inference, Wilkinson, Cheshmi, Dehnavi [PLDI'23]

https://github.com/SpRegTiling

The Jamming Problem



Sparse-Jam Solver



$$\min \sum_{i=1}^{|G|} x_i * (freq(g_i) * exec\_cost(g_i) + overhead) \qquad x_i \in \{0, 1\}$$

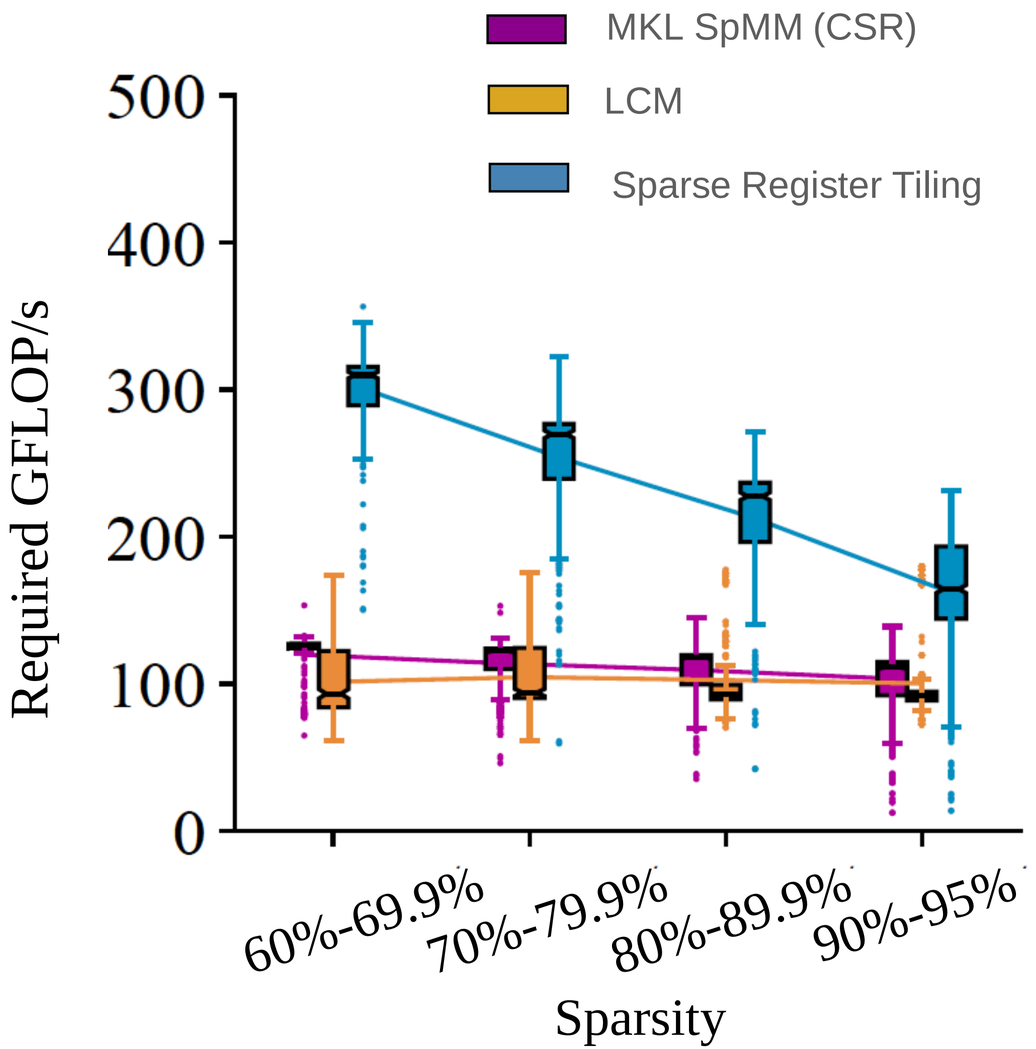
Constraint 1: Ensure all operations are covered

Constraint 2: Size of generated code ≤ $CodeSizeLimit$

Sparse Register Tiling for Machine Learning

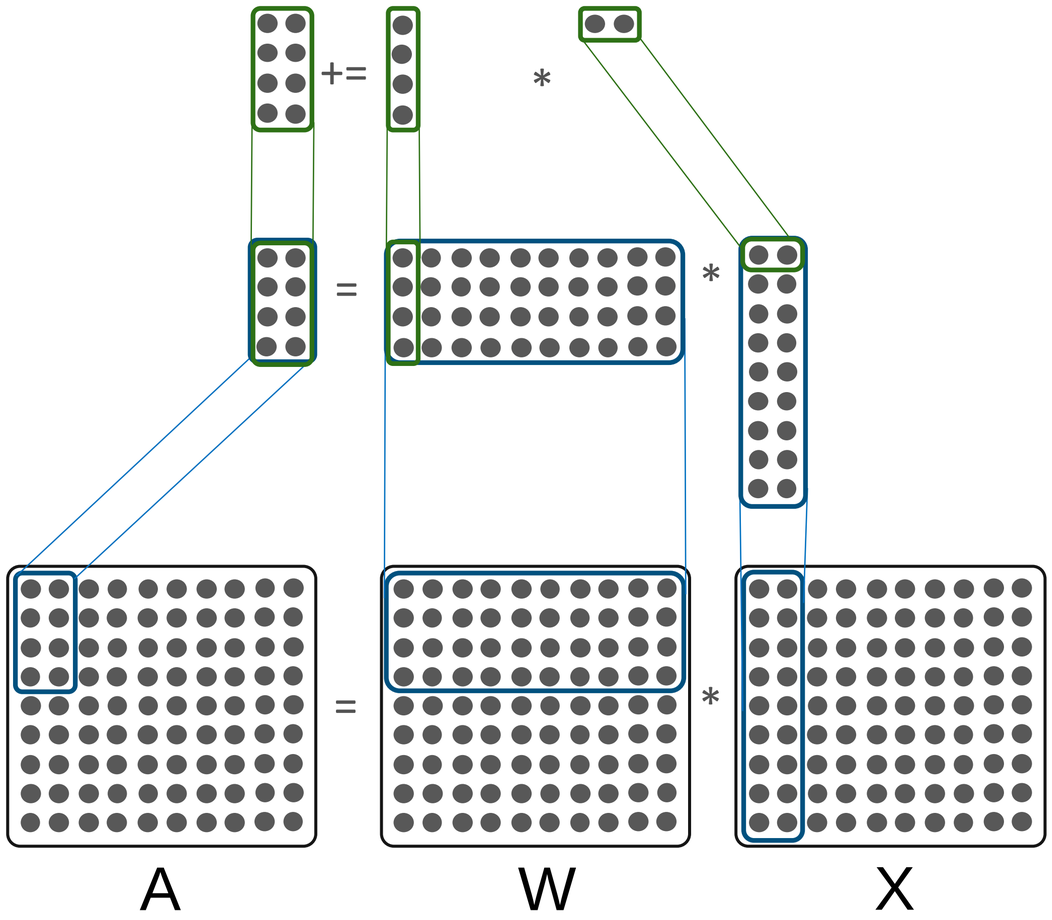
Testbed: **Deep Learning Matrix Collection (DLMC)**[1], a total of **2396 matrices.** A 20 core Cascadelake Xeon(R) Gold 6248 CPU with AVX512.

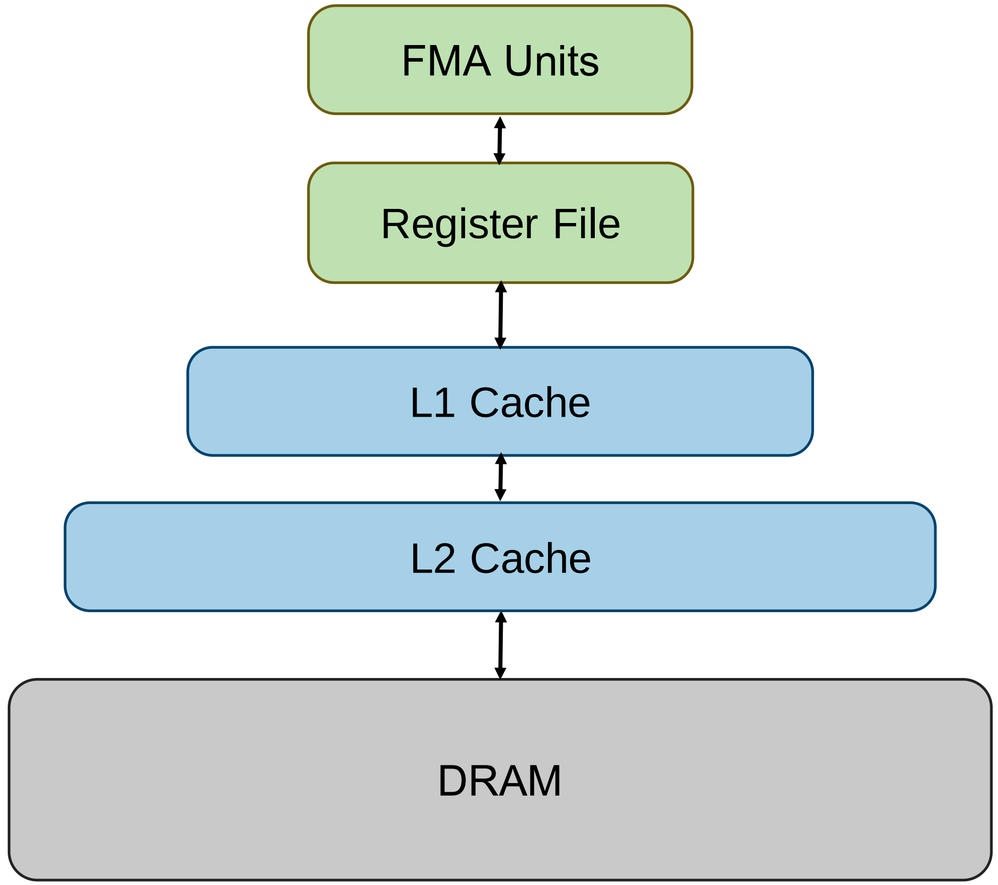
Sparse Register Tiling provides **geomean speedup** of **2.65×**, **1.72×** and **3.23×** over MKL SGEMM, MKL SpMM (CSR) and ASpT[2] respectively.



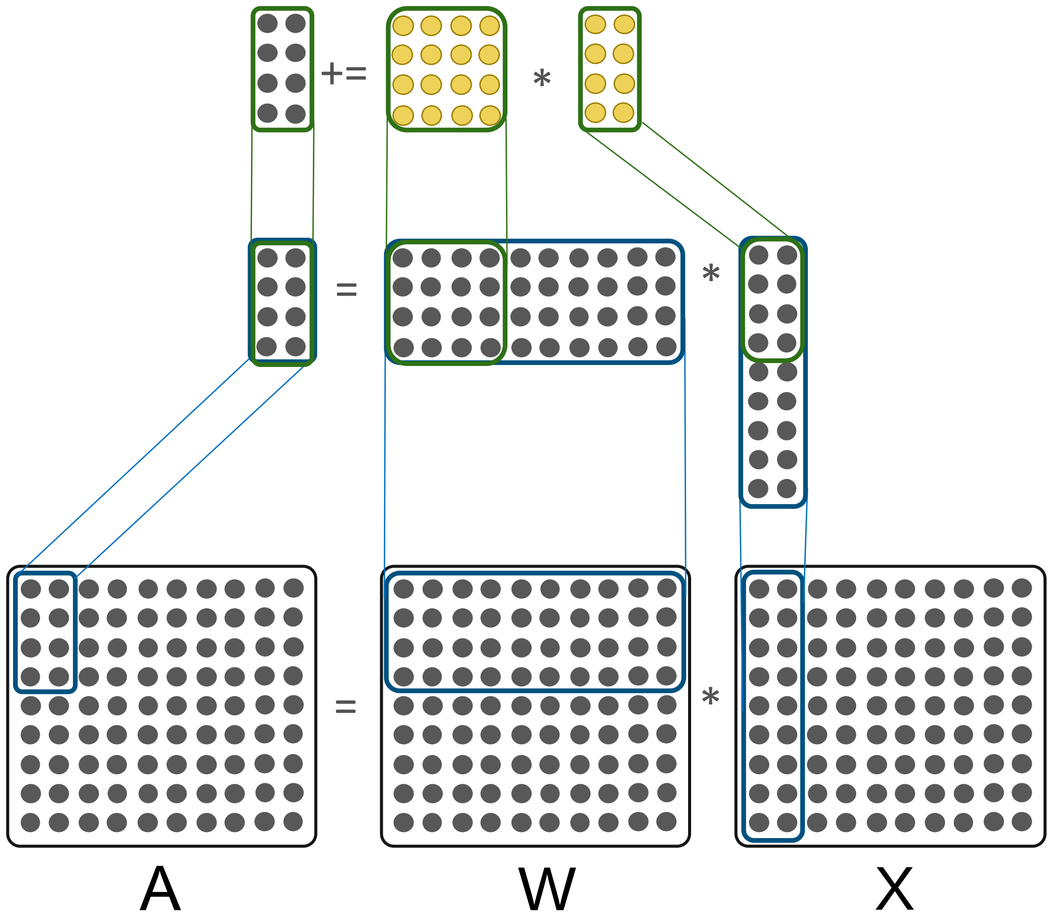
Sparse GPU Kernels for Deep Learning, Gale et. al. SC'20.
[2]Adaptive Sparse Tiling for Sparse Matrix Multiplication, Hong et. al. PPoPP'19.

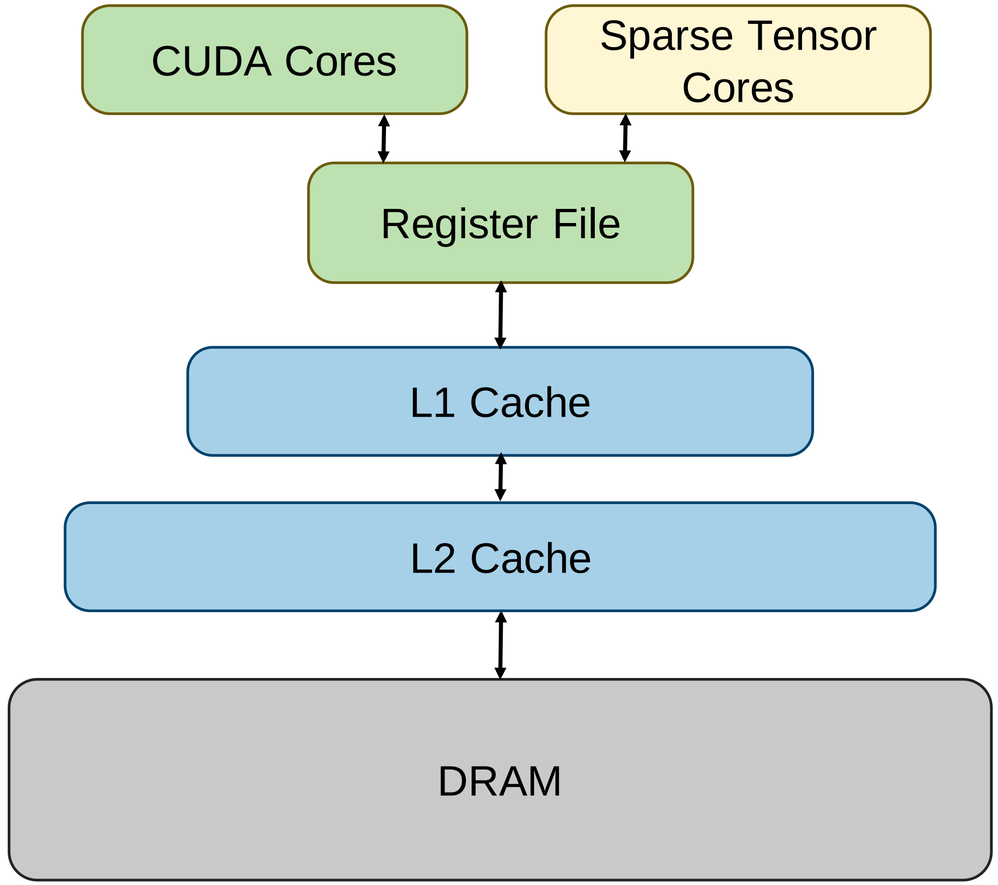
Register Tiling on CPU



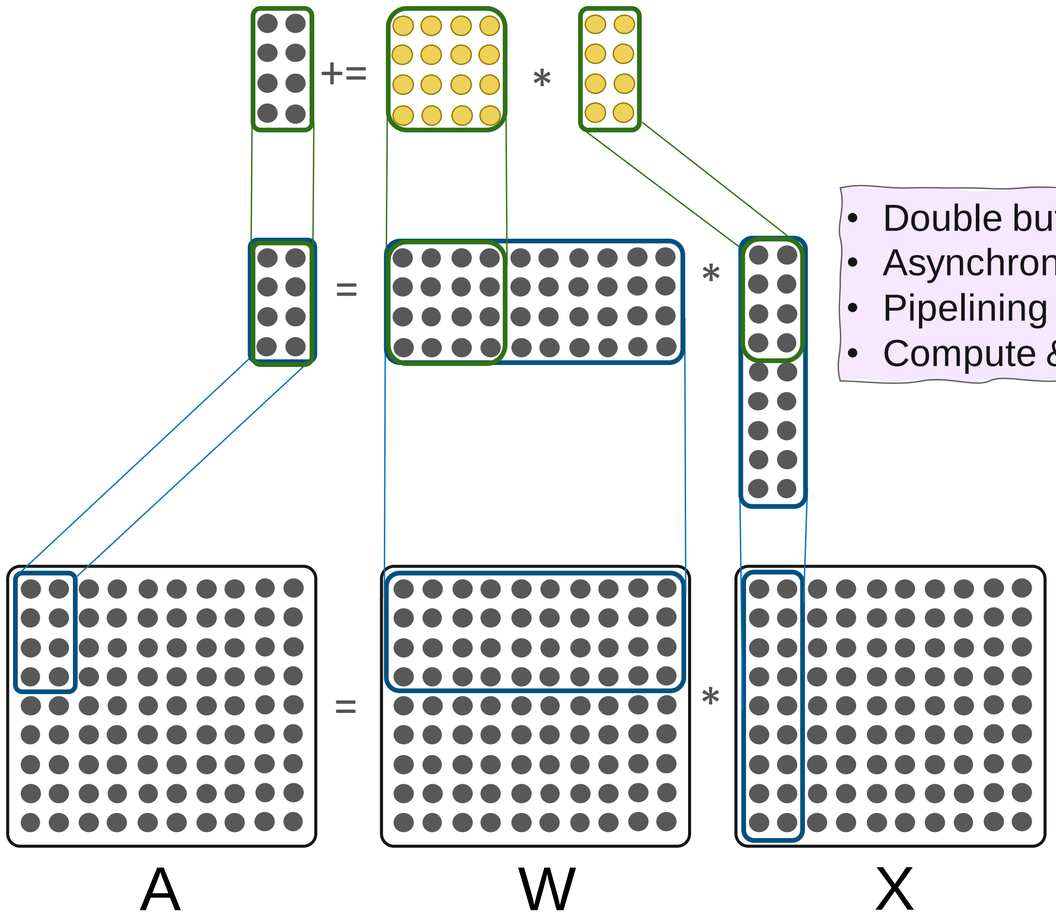


Register Tiling on GPUs

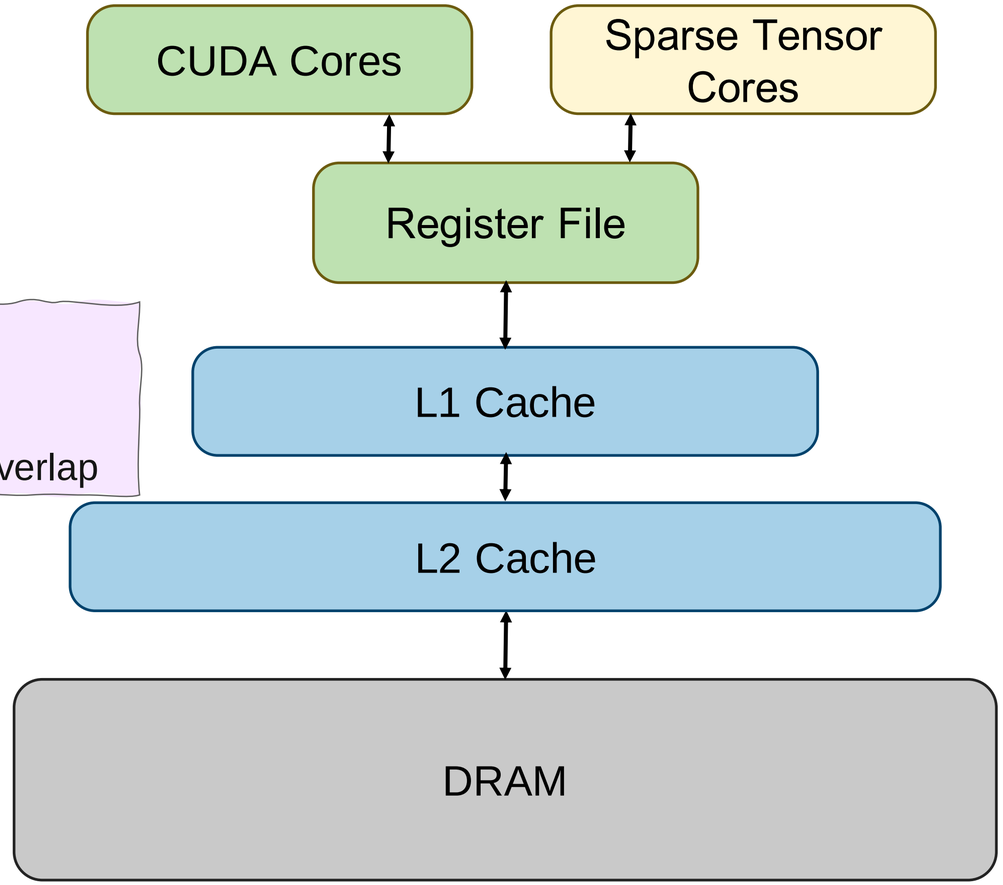




GPU Optimizations Needed to get to Registers

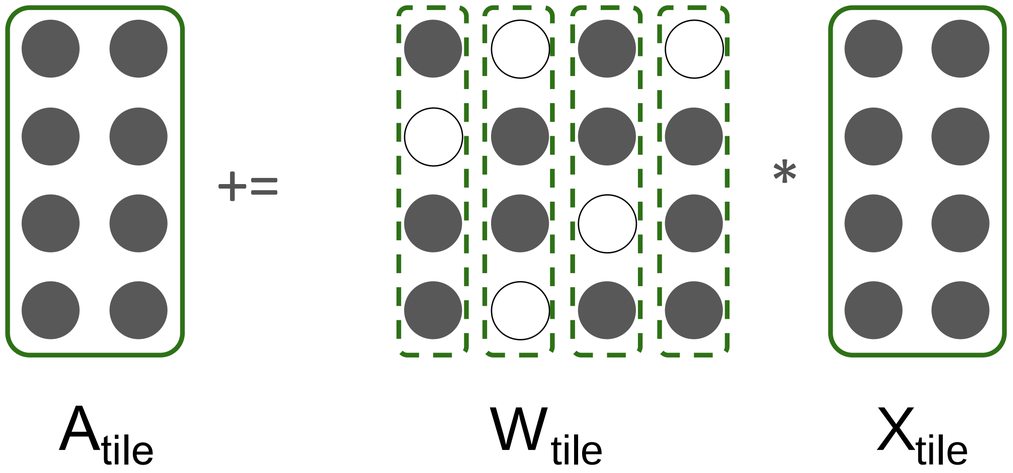


- Double buffering
- Asynchronous mem copies
- Pipelining
- Compute & communication overlap



Sparse Register Tiling on GPUs?

Vertical tiles: register reuse on X



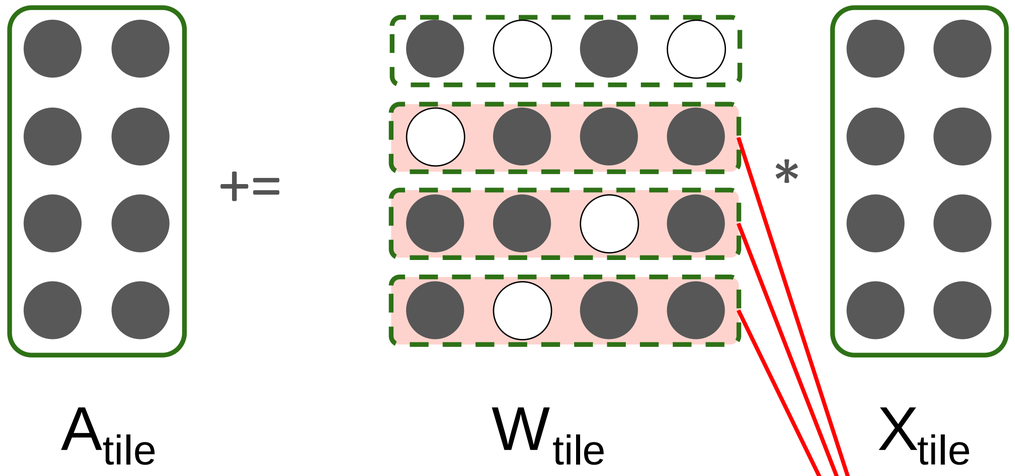
CUDA Cores

~~Sparse Tensor Cores~~

```
__device__ matmul{
    A_tile[0, 0] += W_tile2[0, k] * X_tile[k, 0]
    A_tile[0, 1] += W_tile2[0, k] * X_tile[k, 1]
    A_tile[2, 0] += W_tile2[2, k] * X_tile[k, 0]
    A_tile[2, 1] += W_tile2[2, k] * X_tile[k, 1]
    ...
     }
```

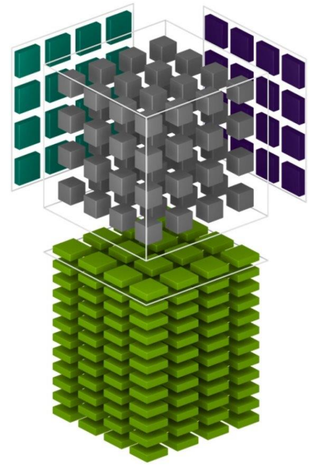
What About the Sparse Tensor Cores?

2:4 Sparsity: At most 2 elements out of every partition of 4 horizontally consecutive elements should be non-zeros

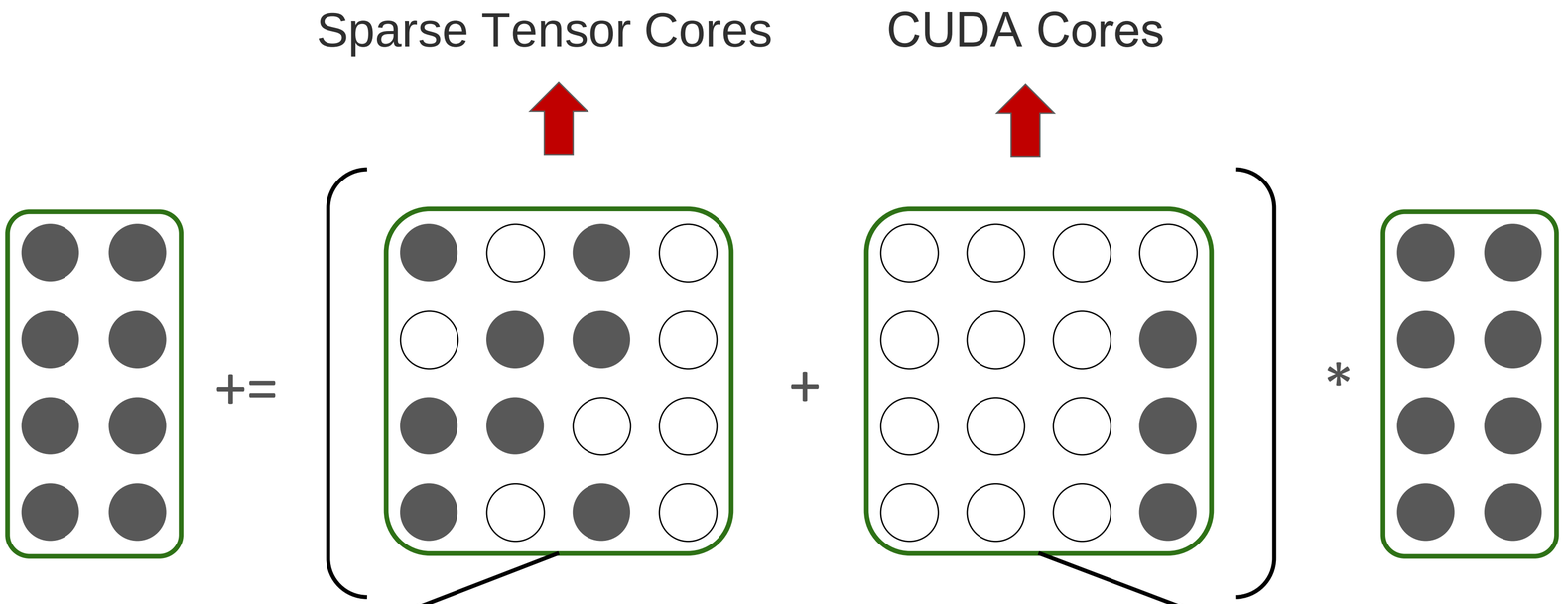


☒ Sparse Tensor Cores

Sparse Tensor Cores



GPU Tile Decomposition

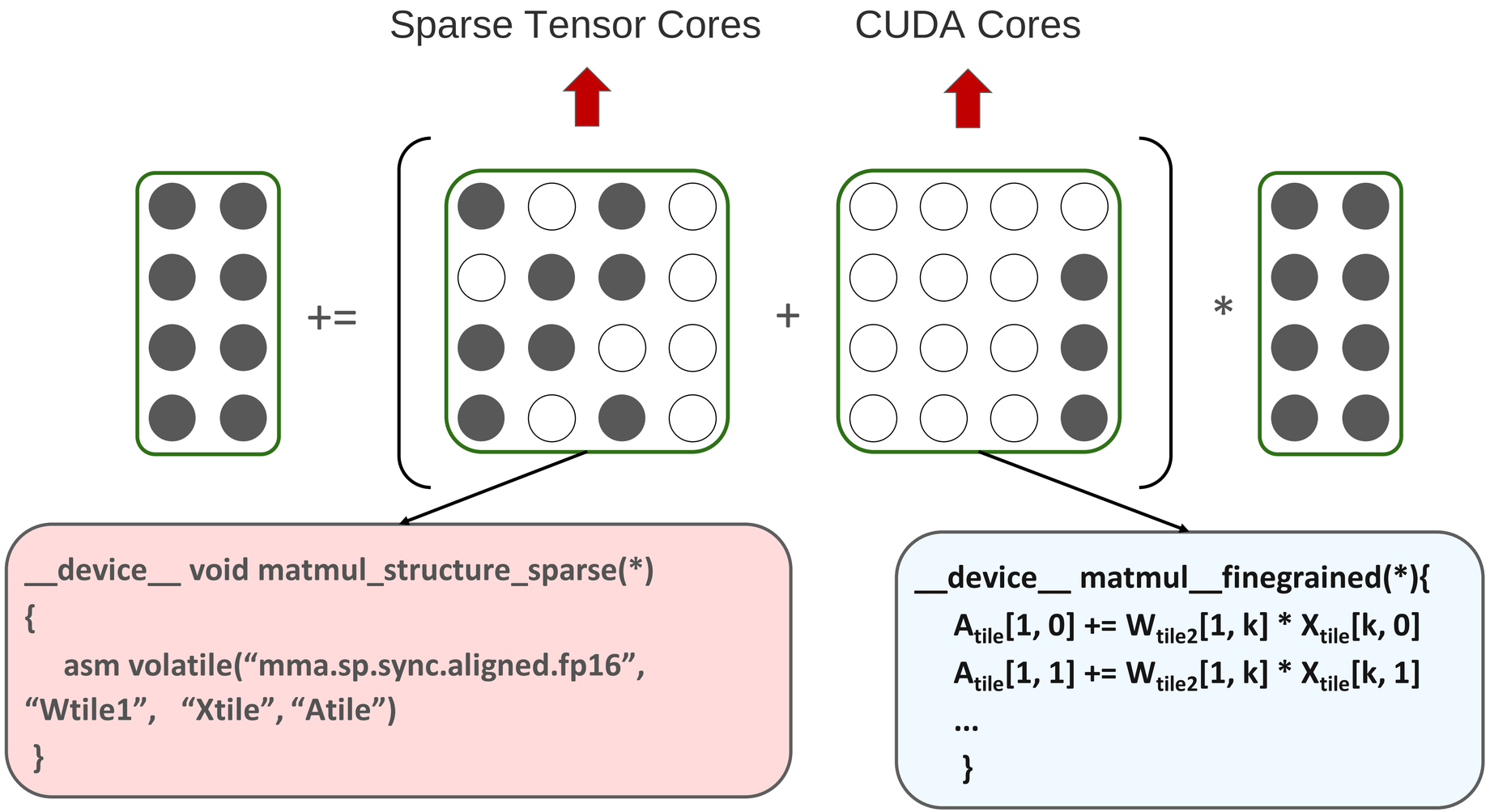



```
__device__ void matmul_structure_sparse(*)
{
    asm volatile("mma.sp.sync.aligned.fp16",
"Wtile1",   "Xtile", "Atile")
 }
```

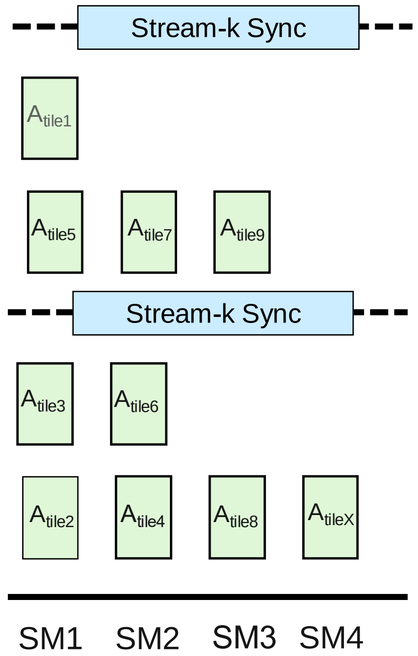
```
__device__ matmul__finegrained(*){
    A_tile[1, 0] += W_tile2[1, k] * X_tile[k, 0]
    A_tile[1, 1] += W_tile2[1, k] * X_tile[k, 1]
    ...
     }
```

Tile Decomposition for Unstructured Sparse Model Inference, Liu, Shahsavan, Dehnavi [in-review]

GPU Tile Decomposition & Tile Scheduling



Tile Scheduling

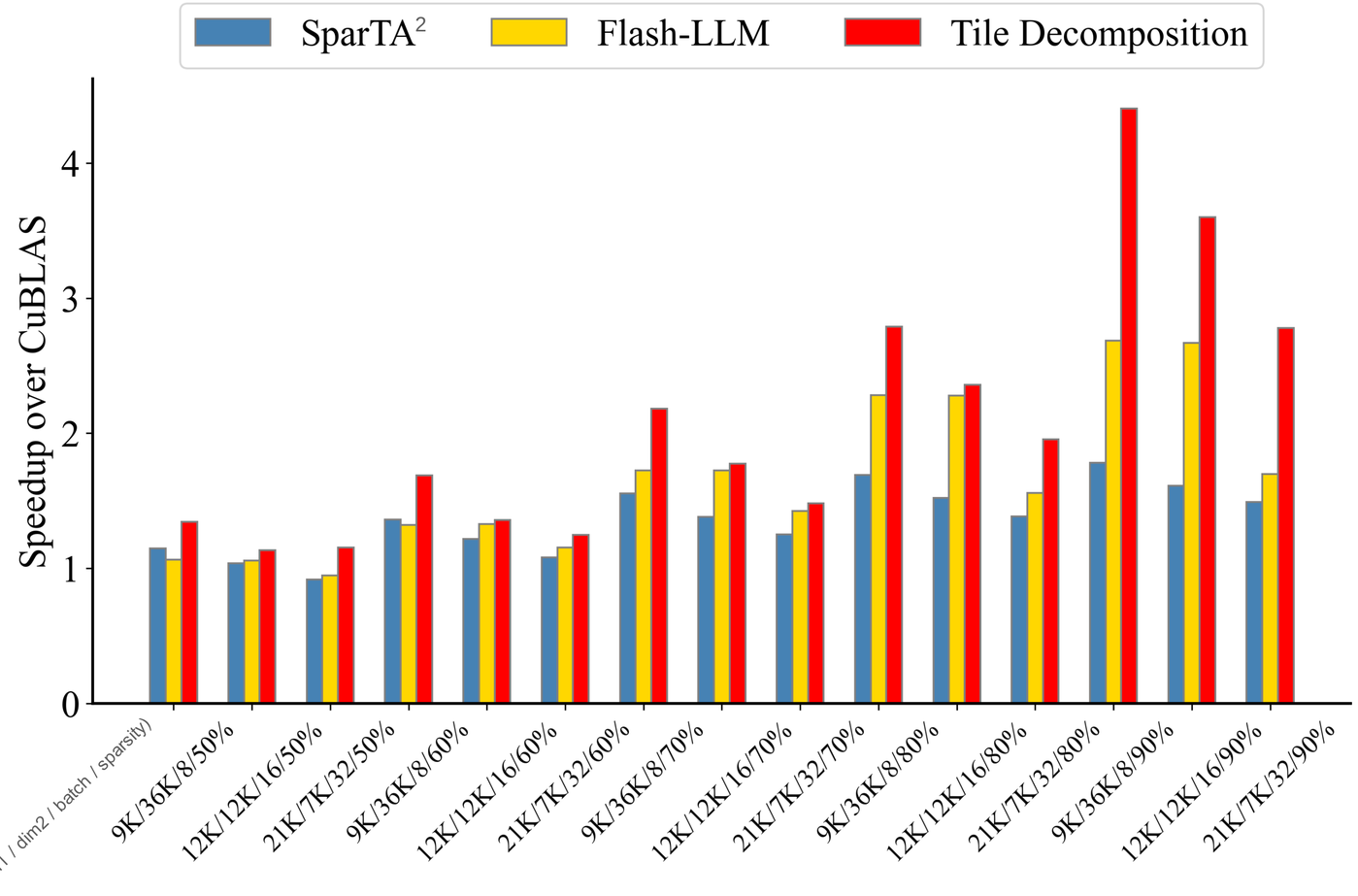


Tile Decomposition for Unstructured Sparse Model Inference, Liu, Shahsavan, Dehnavi [in-review]
A Framework for Fine-Grained Synchronization of Dependent GPU Kernels, Jangda, et. al. [CGO'24]

Sparse Tile Decomposition on GPUs

Testbed: **GPT Matrices from Flash-LLM[1]**, Nvidia RTX 3080 TI, NVCC 12.2

Tile Decomposition provides **geomean speedup** of **2.08×**, **1.47×** and **1.23×** over CuBLAS (Tensor Cores enabled), SparTA[2], and Flash-LLM, respectively, for FP16 data type.



[1]*Flash-llm: Enabling cost-effective and highly-efficient large generative model inference with unstructured sparsity, Xia et. al. VLBD'23*
[2]*SparTA: Deep-Learning Model Sparsity via Tensor-with-Sparsity-Attribute, Zheng et. al. OSDI'22*

Sparsity Patterns in Machine Learning vs. Engineering

L, … ⟵ **factorization**(*W*)
while (*not converged*) {
…
Solve: $Lx_{k+1} = b$
$r_{k+1} = r_k - \alpha W x_{k+1}$
….
}

Engineering

PART 2

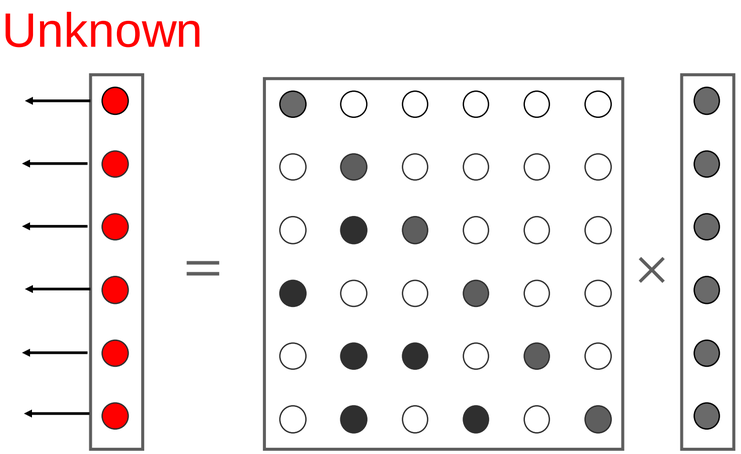
Solver routines

LU factorization, forward-backward solvers, etc.

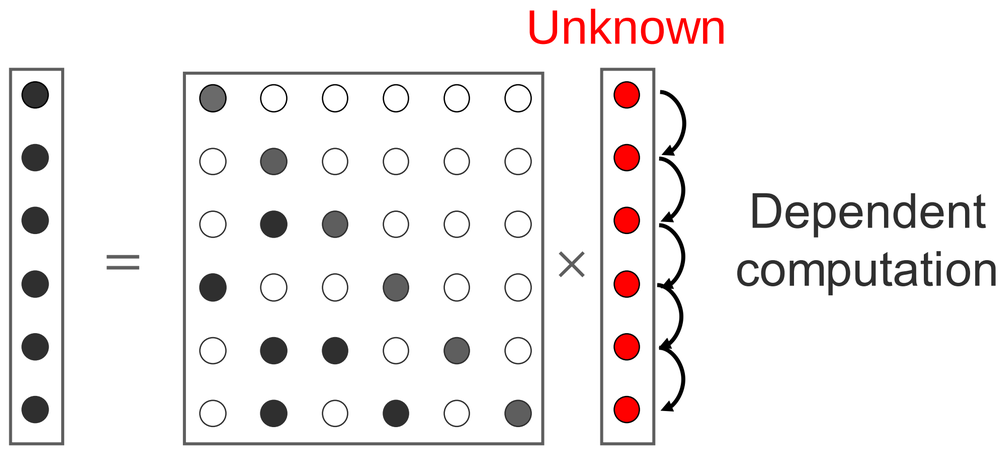
Matrix Multiplication Routines vs. Solvers

Matrix Multiplication routines:

Can be computed independently



Solvers: e.g. sparse triangular system

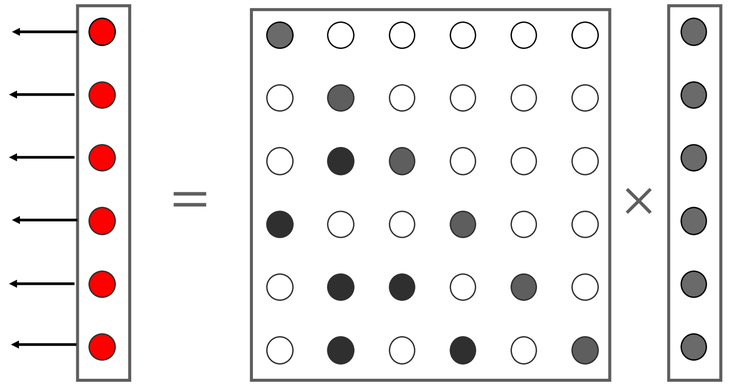


Matrix Multiplication Routines vs. Solvers

Matrix Multiplication routines:

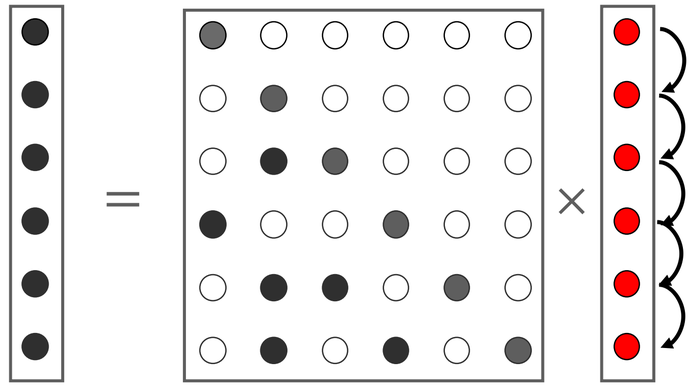
Unknown

Can be computed independently



Solvers: e.g. sparse triangular system

Unknown



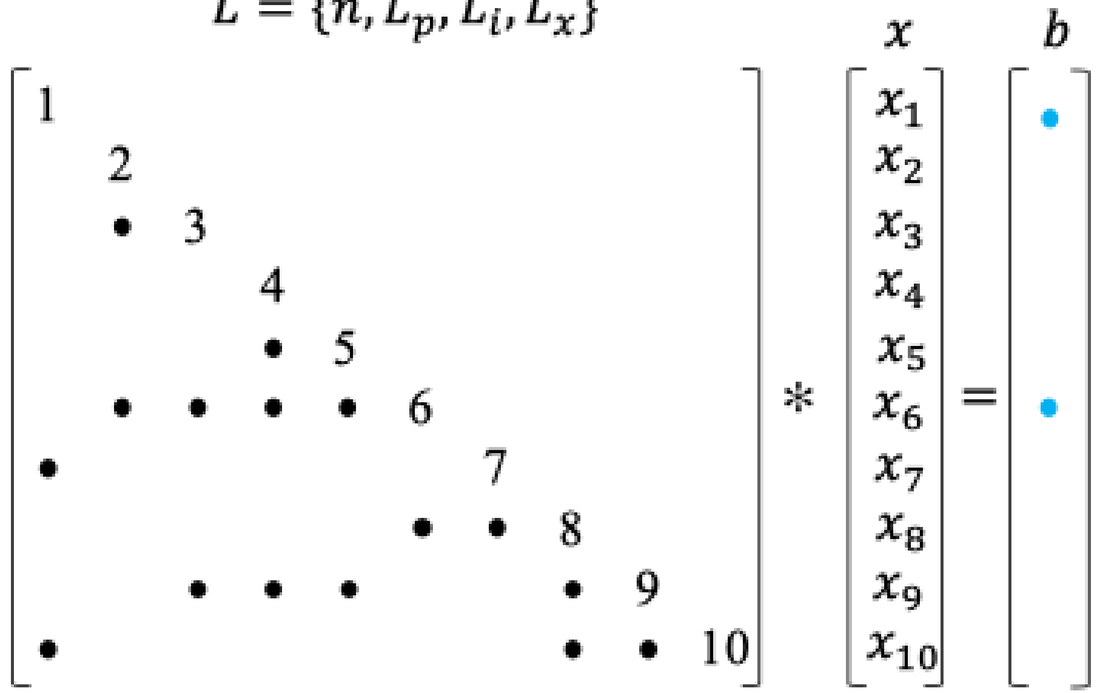
Dependent computation

Inspect data dependence patterns for pruning and parallelism, etc.

Solving a Sparse Triangular System

Find the solution to x, $Lx = b$ where L is sparse lower triangular matrix.

$$L = \{n, L_p, L_i, L_x\}$$




```
for (j=0; j<n; j++){

  x[j]/=Lx[Lp[j]];

  for (p=Lp[j]+1; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
 }
```

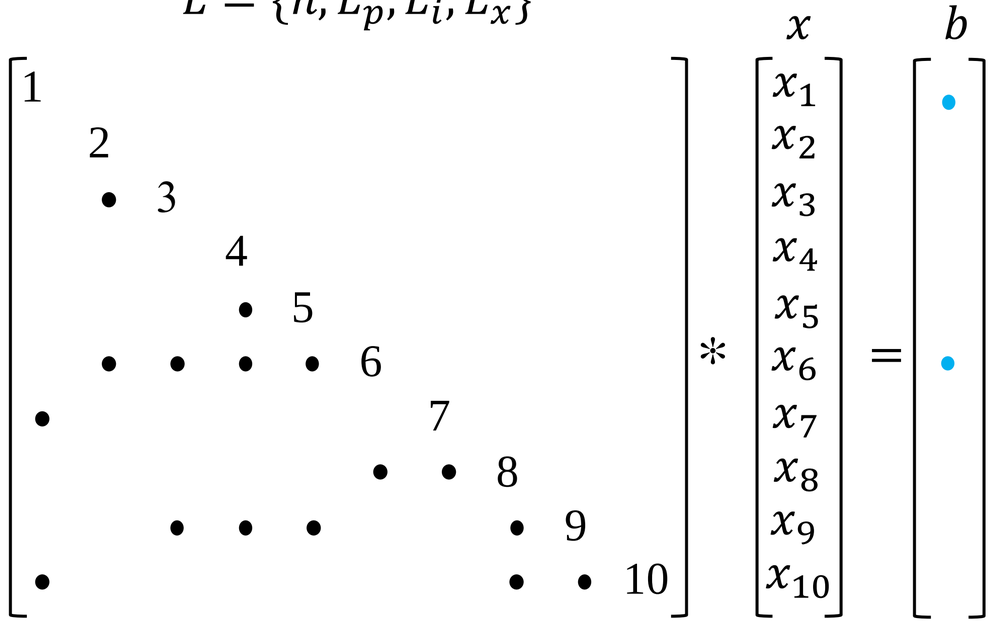
Sparse Analysis in the Sparse Triangular System Solver

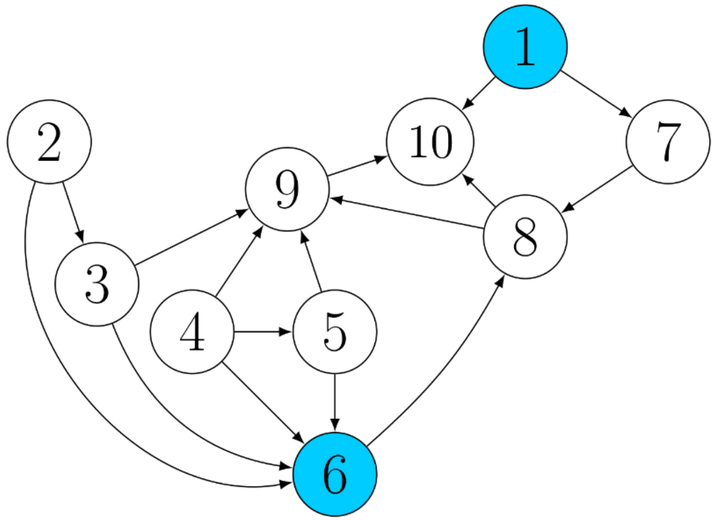
Depth First Search (DFS)

Inspection

Dependence Graph (DG_L)

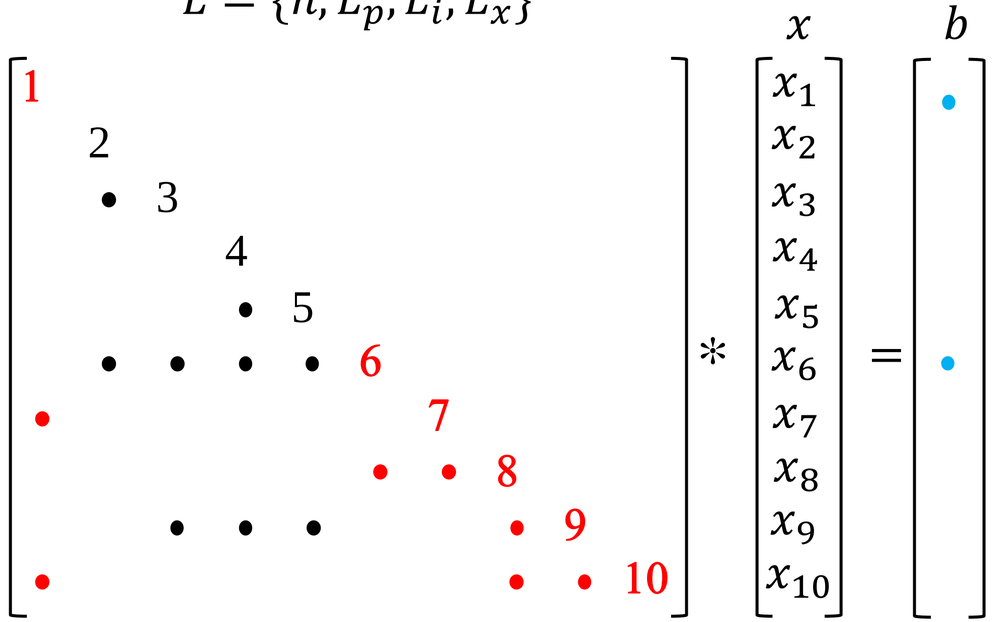
$$L = \{n, L_p, L_i, L_x\}$$





Sparse Analysis in the Sparse Triangular System Solver

$$L = \{n, L_p, L_i, L_x\}$$



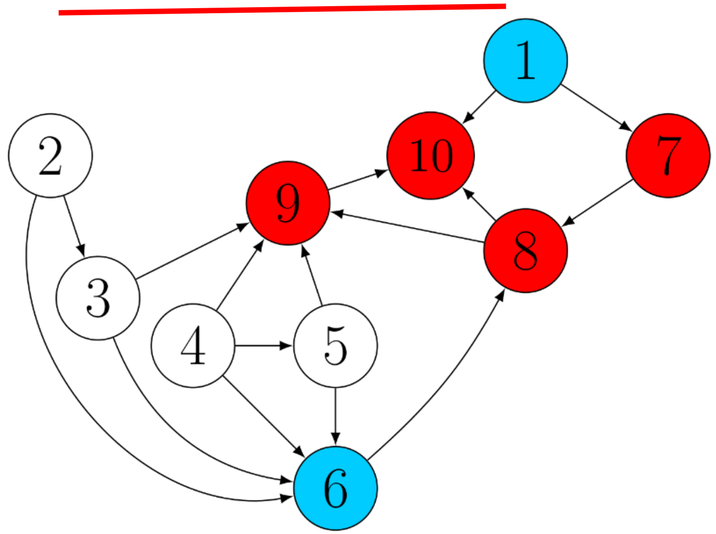
Depth First Search (DFS)

Inspection

Dependence Graph (DG_L)

$\beta = \{i | b_i \neq 0\} = \{1,6\}$

$Reach_L(\beta) = \{1,6,7,8,9,10\}$



Transforming the Sparse Code

Standard Code

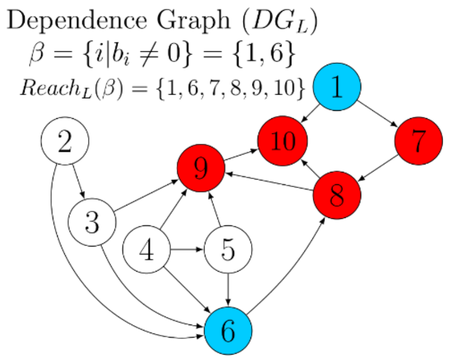
```
for (j=0; j<n; j++){

  x[j]/=Lx[Lp[j]];

  for (p=Lp[j]+1; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
 }
```

Prune the iteration space:
Reachset = {6,1,7,8,9,10}

Code Gen

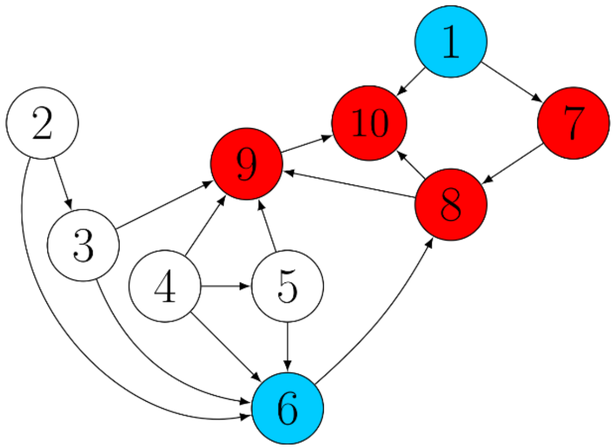
Symbolically-Guided Code

```
for (px=0; px<RSsize; px++) {
  j=reachset[px];          // = 6
  x[j]/=Lx[Lp[j]]
  p=Lp[j]+1;
  for (; p<Lp[j+1]; p++)
    x[Li[p]]-=Lx[p]*x[j];
  }
```

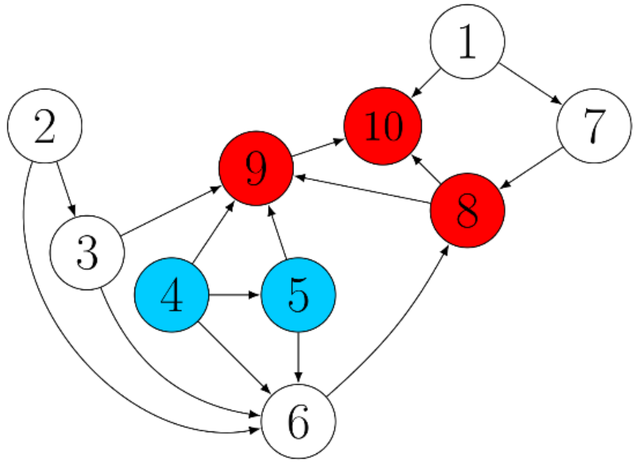


Sparse Inspection and Code Transformations for Sparse Solvers

Sympiler [SC'17]: Inspects for single core optimizations, i.e. pruning, blocking, etc.



Depth First Search
For Pruning



Node Equivalence for
Blocking

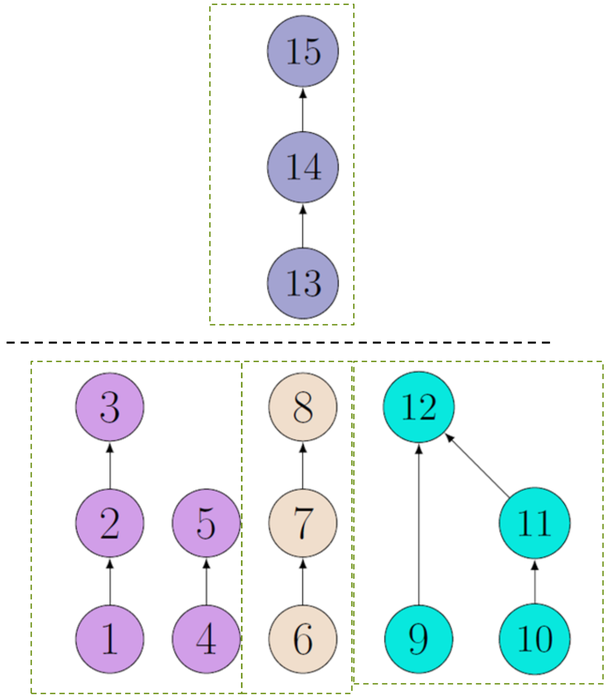
www.sympiler.com

Sparse Inspection and Code Transformations for Sparse Solvers

Sympiler [SC'17]: Inspects for single core optimizations, i.e. pruning, blocking, etc.

+

ParSy [SC'18] and HDAGG [IPDPS'22]: Create well balanced workloads for multicore execution.



Load-Balanced Level Coarsening for Parallelism

www.sympiler.com

Sparse Inspection and Code Transformations for Sparse Solvers

Sympiler [SC'17]: Inspects for single core optimizations, i.e. pruning, blocking, etc.

+

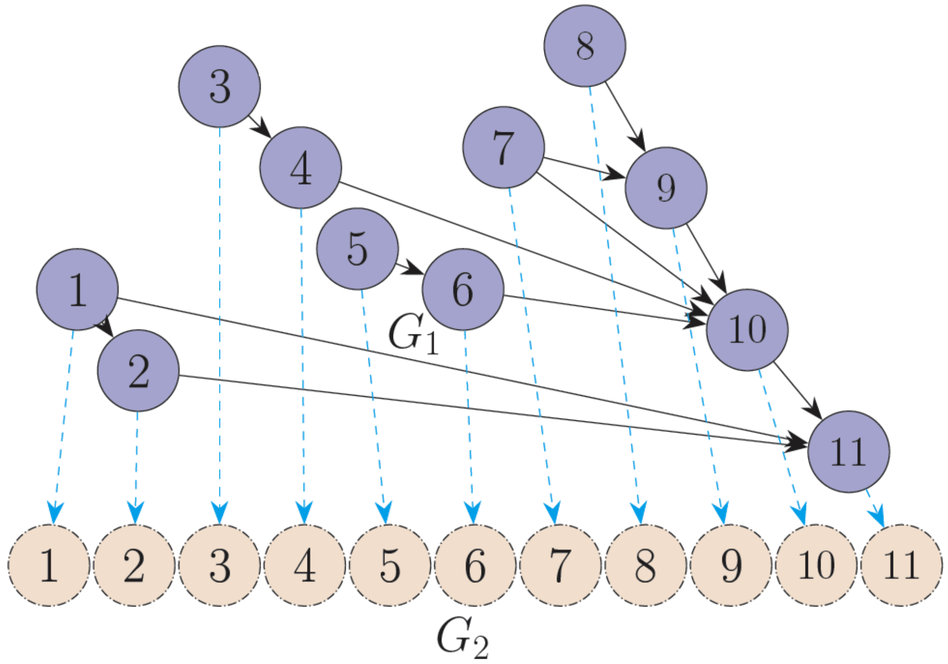
ParSy [SC'18] and HDAGG [IPDPS'22]: Create well balanced workloads for multicore execution

+

Sparse Fusion [SC'23]: Inspects the graphs of multiple operations to generate parallel fused code.

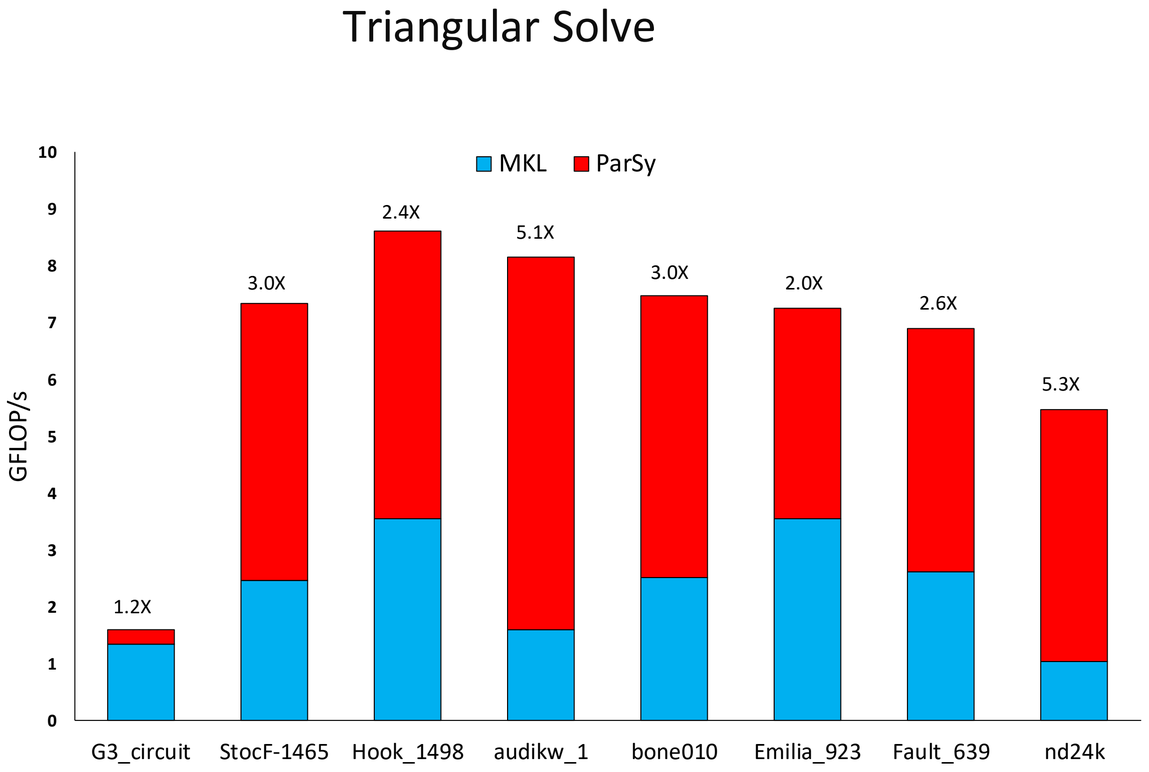
$Lx = b$

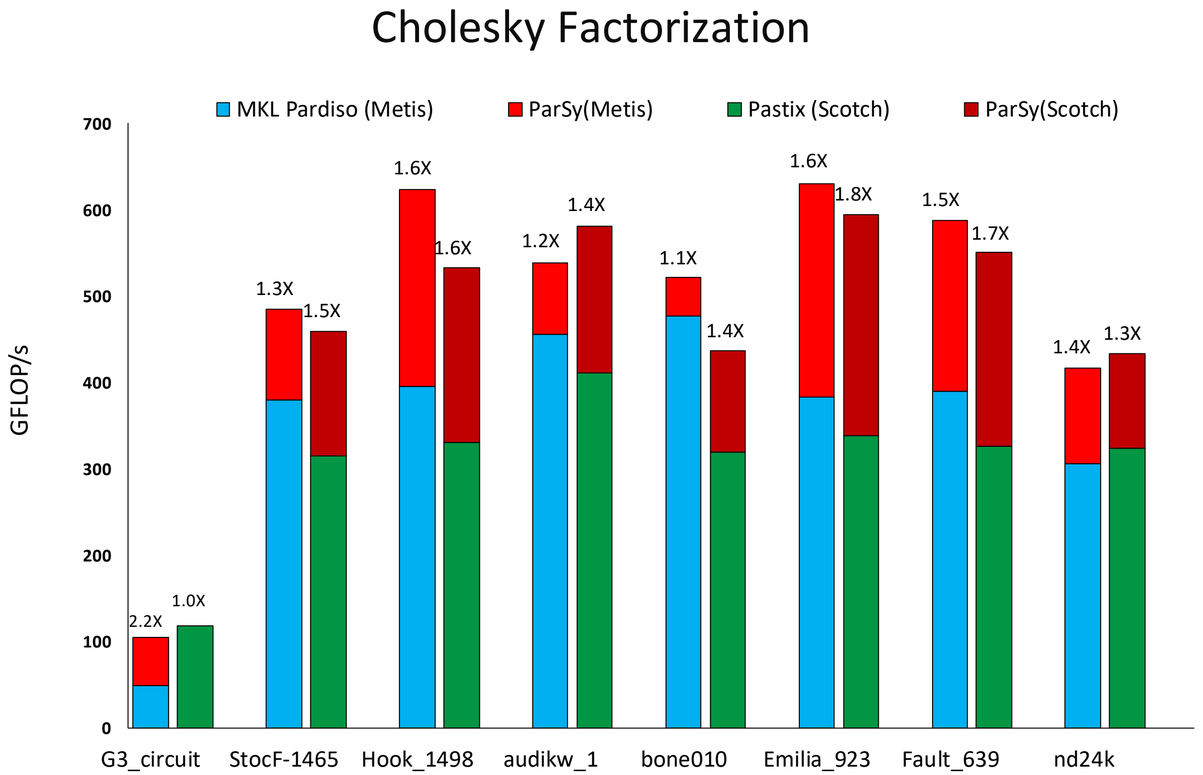
$y = Ax$



www.sympiler.com

ParSy vs Competitors





ParSy is faster than Intel **MKL** with an average speedup of **2.5x**.

ParSy is faster than **Pardiso** with an average speedup of **1.5x**.

***Testbed**: SuiteSparse* repository on an Intel(R) Xeon(R) Platinum 8160 (Skylake).*

There is no One-Size-Fits-All Approach!

Thesis

Sparsity is often static or changes moderately in most simulations.

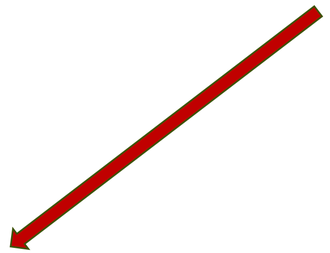
Key contribution

Inspect sparsity patterns to automatically generate highly-optimized code for sparsity.

There is no One-Size-Fits-All Approach!

Thesis

Sparsity is often static or changes moderately in most simulations.



Tweak the systems:

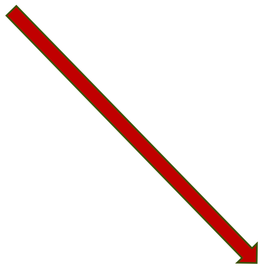
- ❖Relax sparsity-specific specialization
- ❖Runtime checks
- ❖Just-in-time compilation

Typically reduces our performance gains

There is no One-Size-Fits-All Approach!

Thesis

Sparsity is often static or changes moderately in most simulations.



Change the algorithms!

- Constrained/unconstrained optimization in graphics
- Machine Learning Training

A Constrained Optimization Problem: QP Solvers

NASOQ: A Numerically Accurate Sparsity-Oriented Quadratic Program Solver [Siggraph'20]

A case for dynamic sparsity

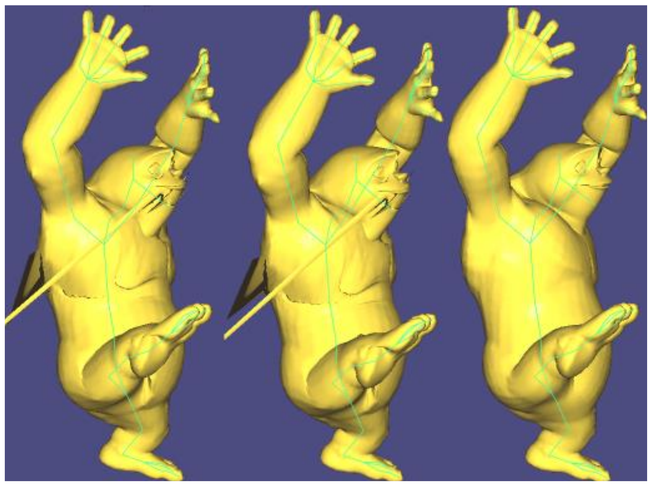
$$\min_x \ \frac{1}{2} x^T H x + q^T x$$

$$Ax = b$$

$$Cx \leq d$$

Objective

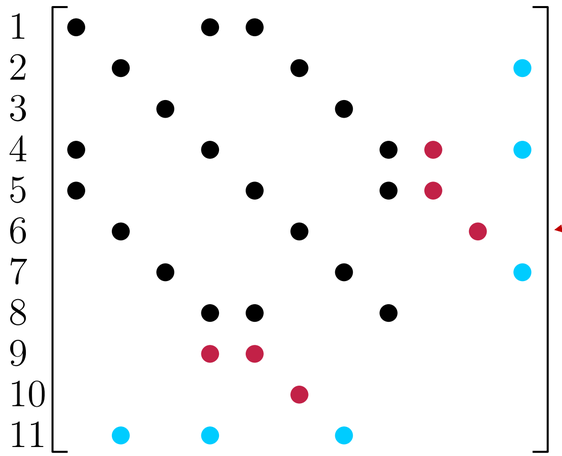
Constraints



https://nasoq.github.io/

The Goldfarb and Idnani Algorithm

In the optimality phase, for each added or removed constraint a new KKT system has to be solved.



KKT matrix

Optimality Phase

while(x_k is not optimal) {

$$[\alpha, \pi, \mu] \leftarrow \begin{bmatrix} H & A^T & C_w^T \\ A & 0 & 0 \\ C_w & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} C_{act} \\ 0 \\ 0 \end{bmatrix}$$

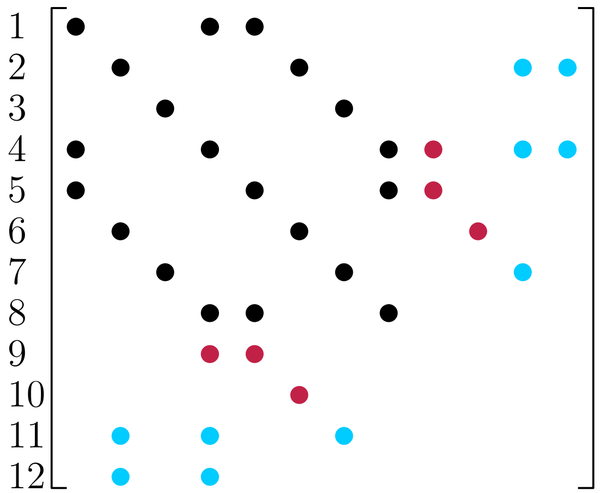
…

}

Factorize the Updated KKT Matrix

The Goldfarb and Idnani Algorithm

Solving incrementally increasing KKT systems, has *large overheads* and thus leads to *poor scalability*.



KKT matrix

Optimality Phase

while(x_k is not optimal) {

$$[\alpha, \pi, \mu] \leftarrow \begin{bmatrix} H & A^T & C_w^T \\ A & 0 & 0 \\ C_w & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} C_{act} \\ 0 \\ 0 \end{bmatrix}$$

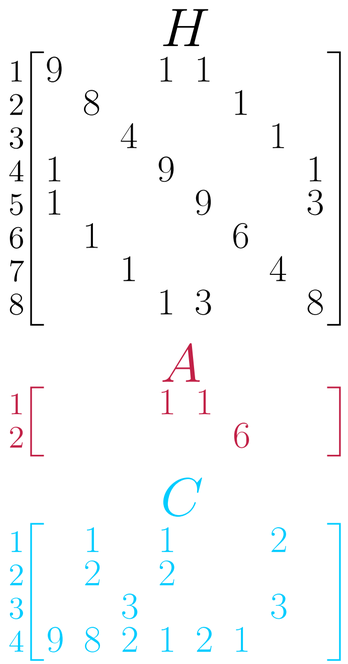
…

}

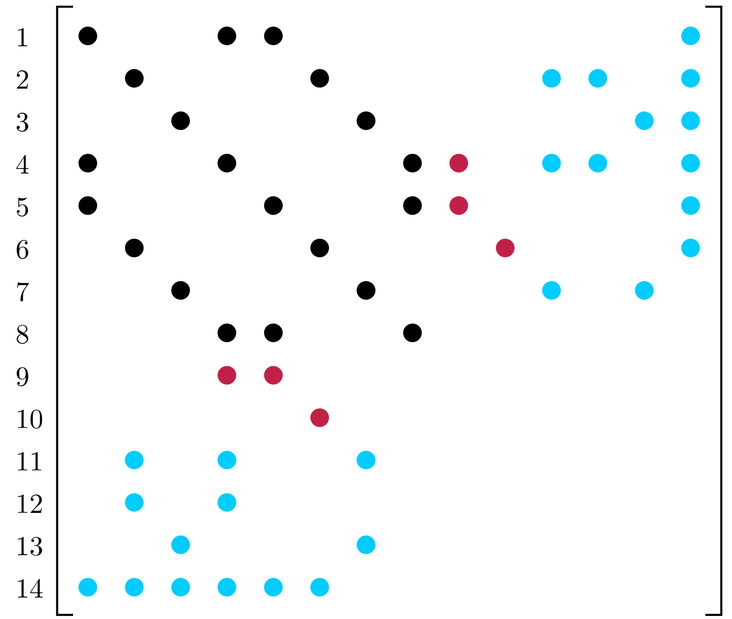
Factorize the Updated KKT Matrix

The Inclusive KKT Matrix in NASOQ

NASOQ adds the symbolic information of all constraints to an *inclusive* KKT matrix prior to solving the QP.

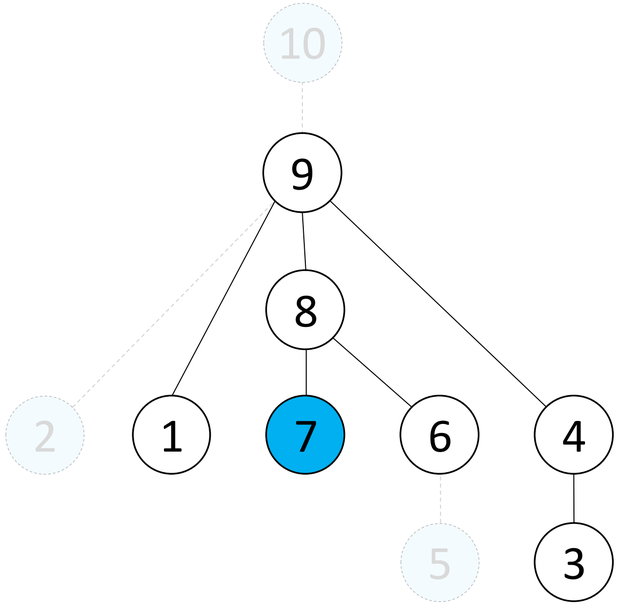


Objective and Constraints Matrices



Inclusive KKT

Sparsity updates: the Optimality Phase



Optimality Phase

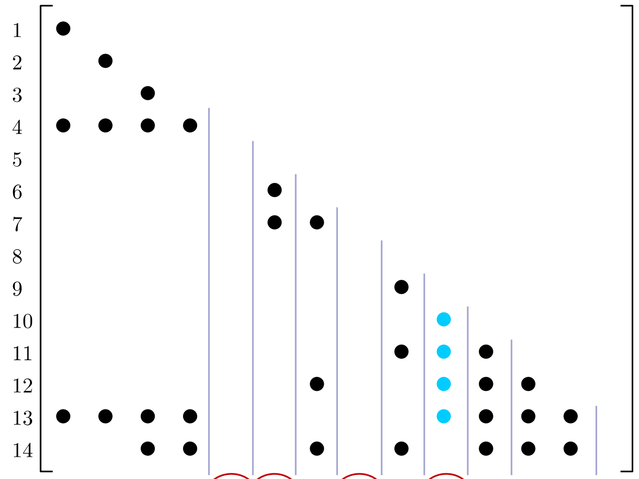
while(x_k is not optimal) {

$$[\alpha, \pi, \mu] \leftarrow \begin{bmatrix} H & A^T & {C_w}^T \\ A & 0 & 0 \\ C_w & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} C_{act} \\ 0 \\ 0 \end{bmatrix}$$

…

}

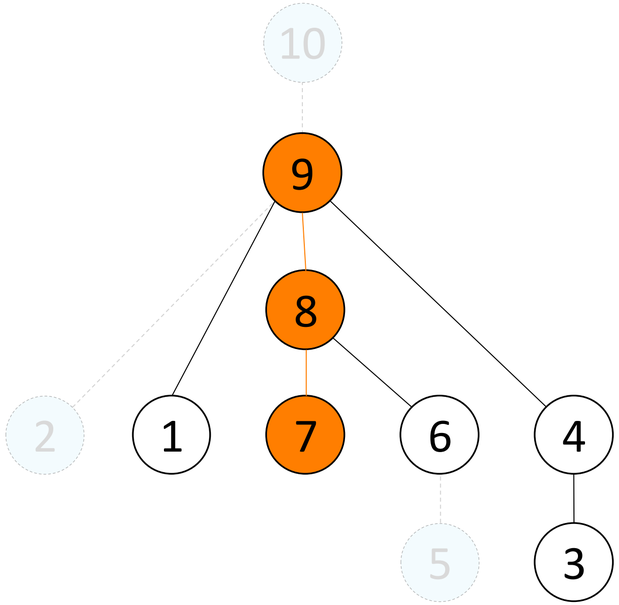
Inclusive KKT



Constraints

For each added inequality constraint, the corresponding column in the inclusive KKT matrix and the dependence graph (elimination tree in this case) are activated.

Sparsity updates: the Optimality Phase



Optimality Phase

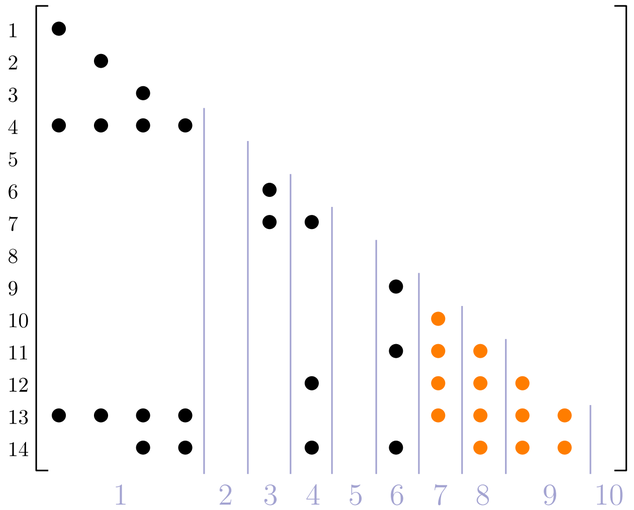
while(x_k is not optimal) {

$$[\alpha, \pi, \mu] \leftarrow \begin{bmatrix} H & A^T & {C_w}^T \\ A & 0 & 0 \\ C_w & 0 & 0 \end{bmatrix}^{-1} \begin{bmatrix} C_{act} \\ 0 \\ 0 \end{bmatrix}$$

…

}

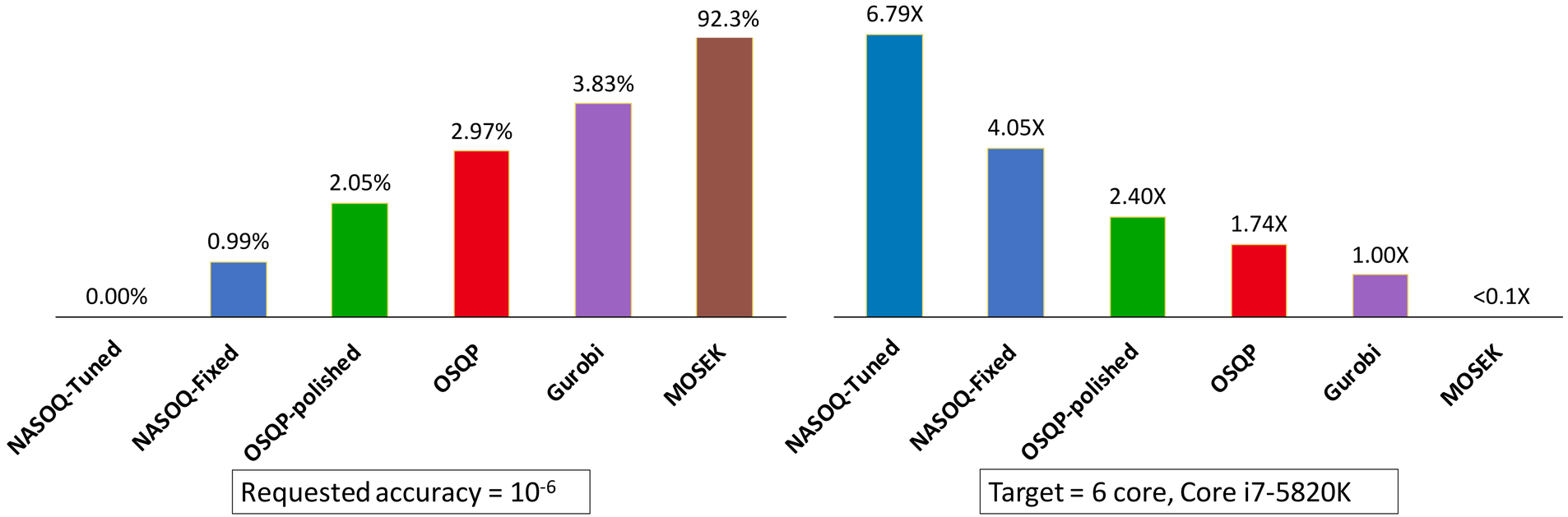
Inclusive KKT



Only the columns affected by the newly added constraint need to be numerically updated, the rest of the values are reused.

Results: NASOQ vs Others

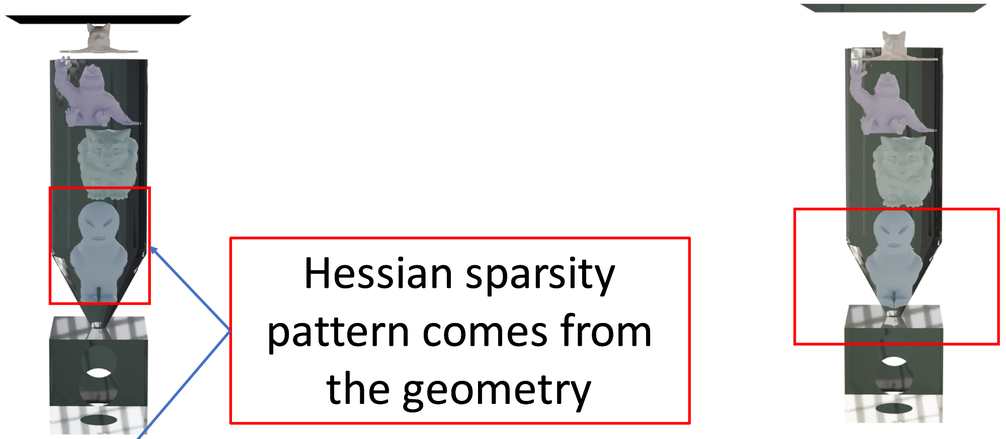
NASOQ has the lowest **failure rate** AND the fastest **solve time**



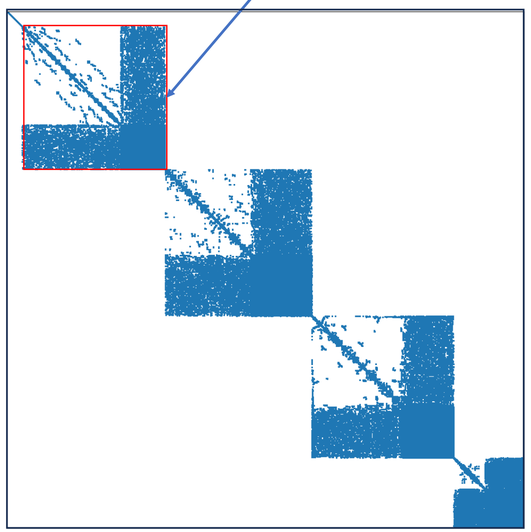
Gurobi is the baseline for speedups

https://nasoq.github.io/

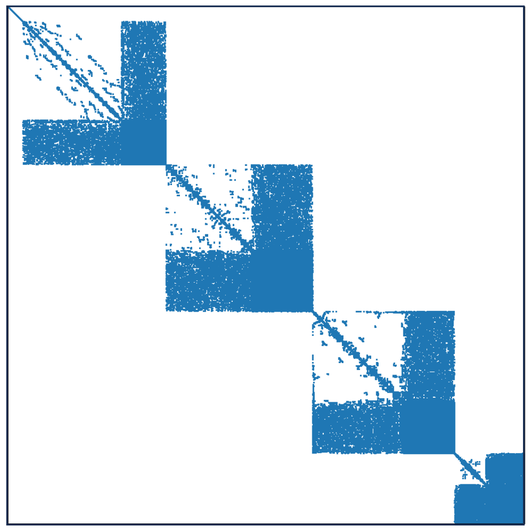
Unconstrained Optimization Problem: Newton Solver for Contact



Hessian sparsity pattern comes from the geometry



Frame 13: H_0



Frame 27: H_{10}

A case for dynamic sparsity

Unconstrained Newton Solver

For each frame:

While(q_k is not optimal){

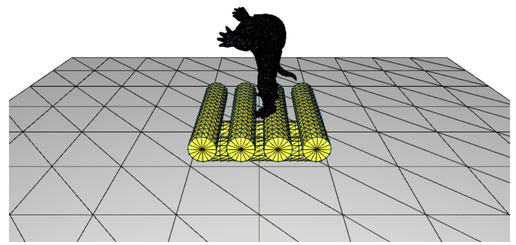
$d \leftarrow -H_k^{-1} g$

Compute step length t

$q_{k+1} = q_k + t * d$

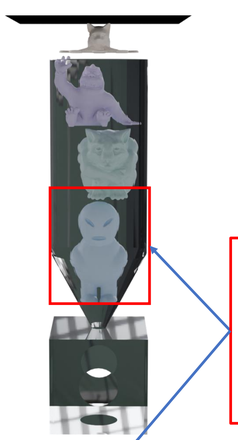
H_{k+1} = computeHessian(H_k)

}

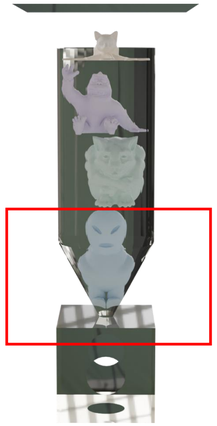


``Squeeze Out'' simulations from Incremental Potential Contact (IPC) simulator - https://ipc-sim.github.io/

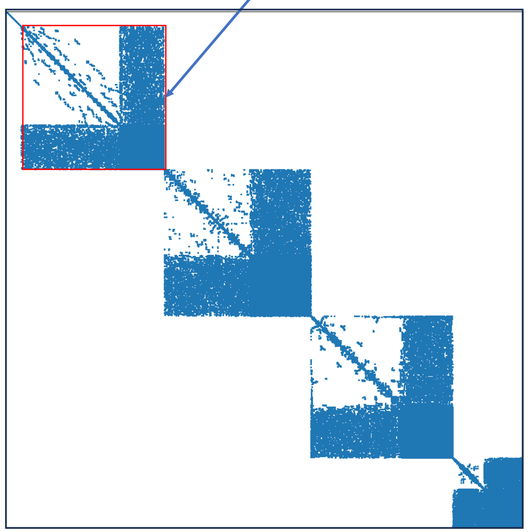
Moderate Changes to the Hessian Sparsity



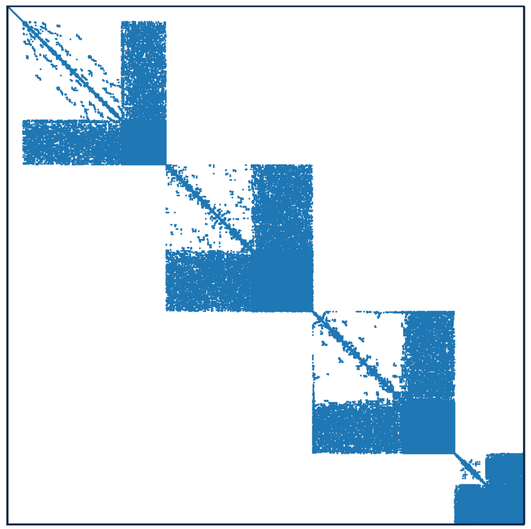
Hessian sparsity pattern comes from the geometry



Contact

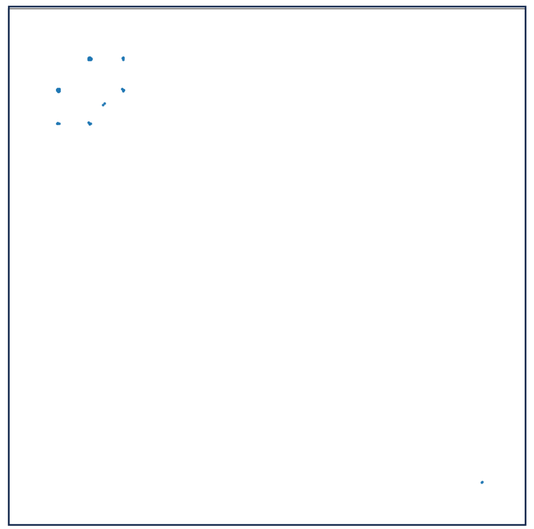


Frame 13: H_0



Frame 27: H_{10}

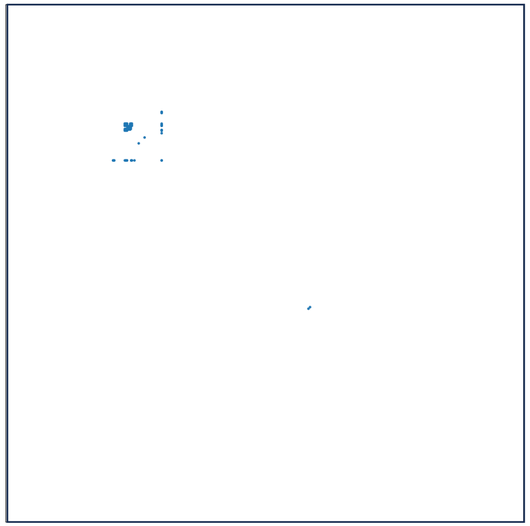
Difference in sparsity



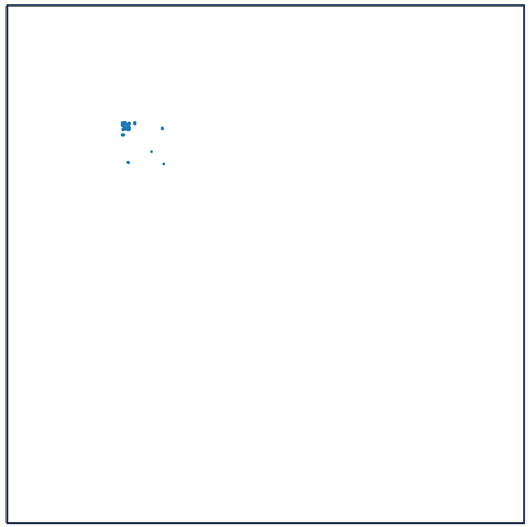
Gradual change in Sparsity

``Squeeze Out'' simulations from Incremental Potential Contact (IPC) simulator - https://ipc-sim.github.io/

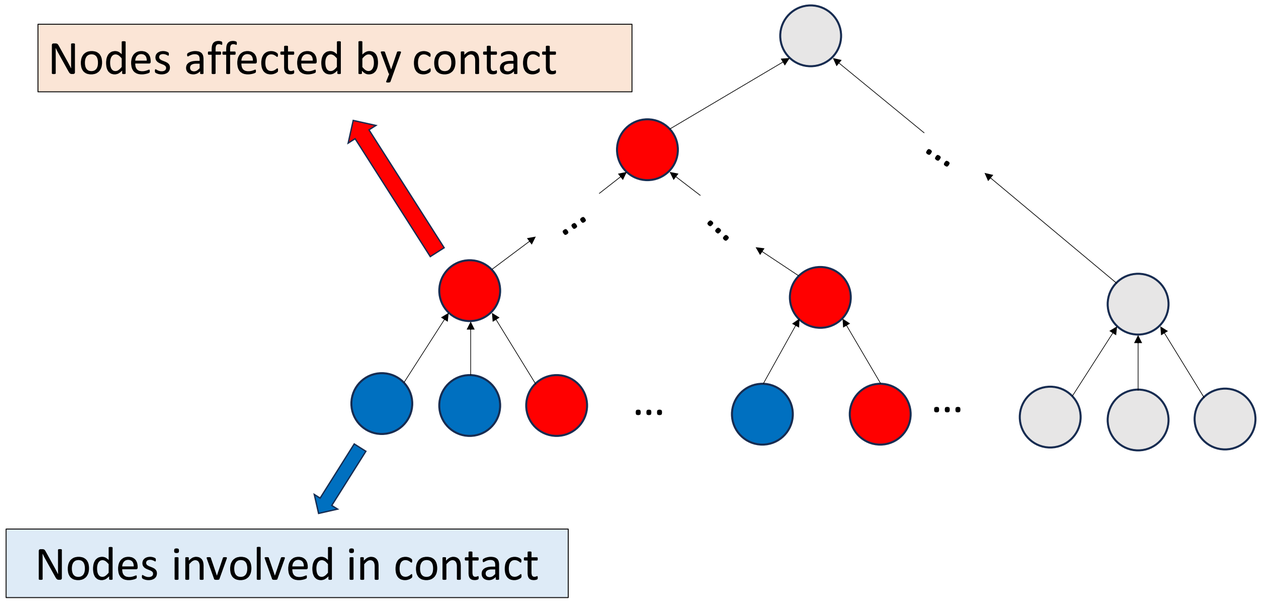
Nodes Involved in Contact

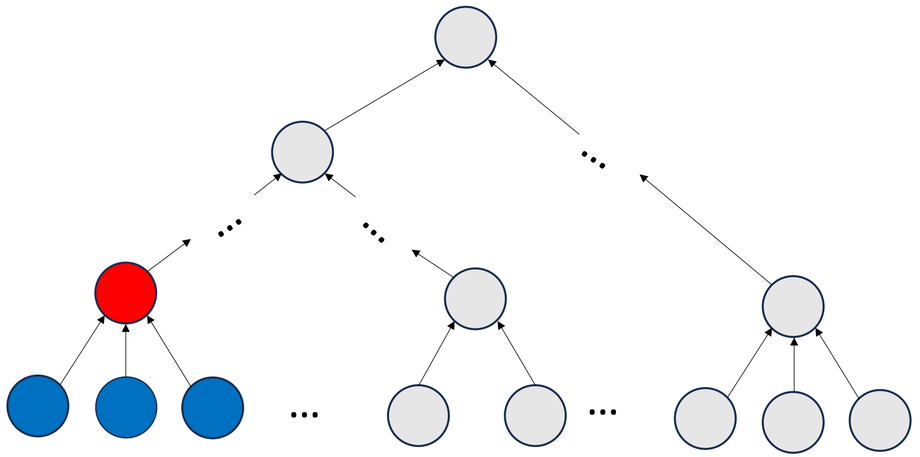


General Re-ordering



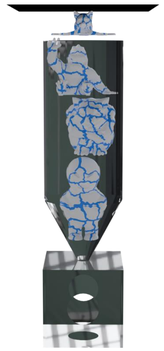
Parth (contact aware) Re-ordering



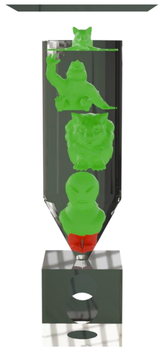


Data Dependence Graph

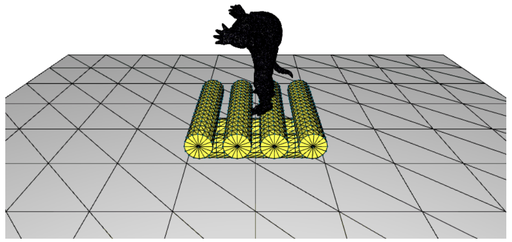
Parth: A Geometry and Re-ordering Aware Mesh Decomposition



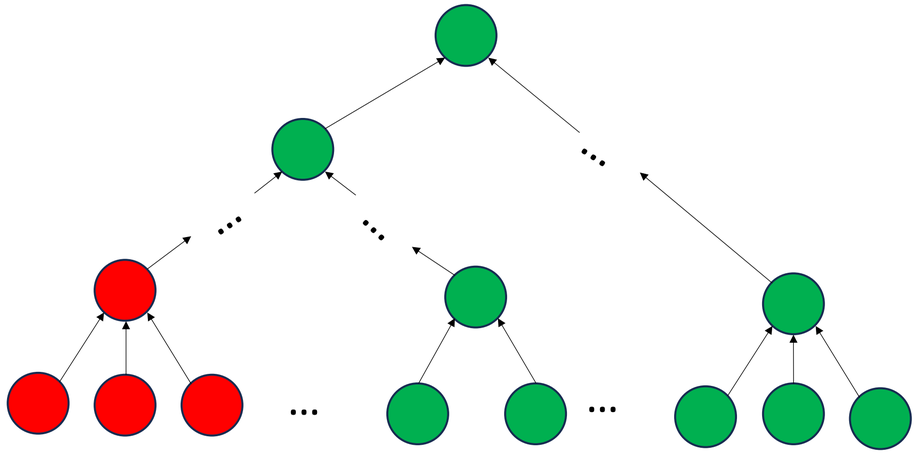
a) Parth decomposition



b) Parth localization



Arma Roller Simulation



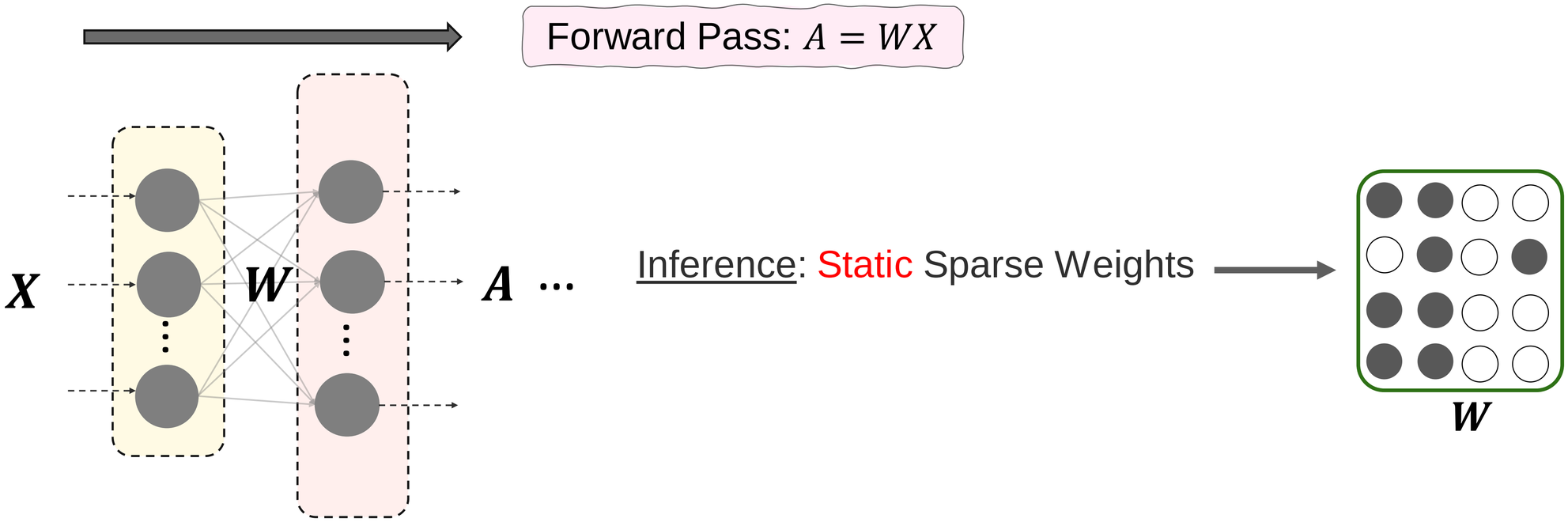
c) Data dependence graph with reuse (colored in green)

IPC Simulations	Apple Accelerate	Intel MKL	CHOLMOD
Dolphin Funnel	2.95x	2.15x	1.97x
Ball Mesh Roller	2.43x	1.87x	1.66x
Mat On Board	2.08x	1.93x	1.72x
Rods Twist	2.07x	1.87x	1.67x
Squeeze Out	2.62x	2.28x	1.99x
Arma Roller	1.33x	1.61x	1.55x

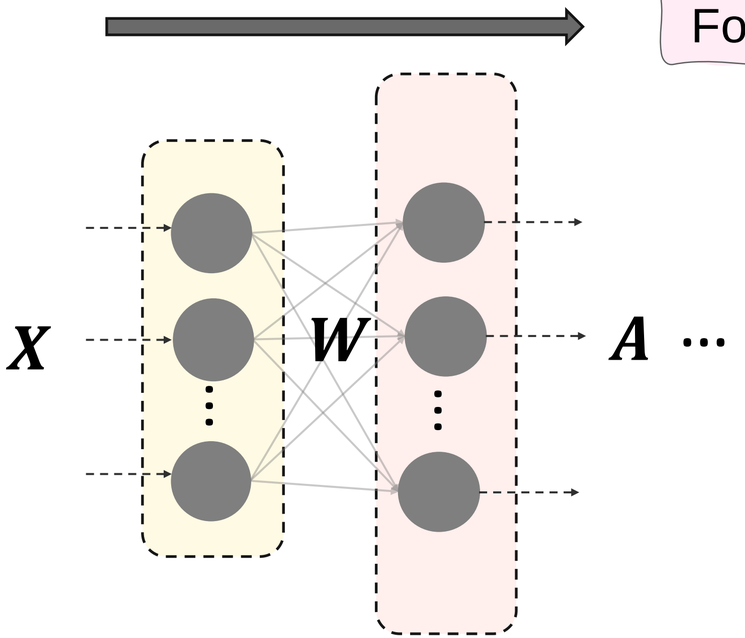
On Intel Xeon with 20 cores and M2 pro with 12 cores

Simulations are from Incremental Potential Contact (IPC) simulator - https://ipc-sim.github.io/

Sparsity in the Neural Network Inference: Forward Pass



Sparsity in the Neural Network Training: Forward/Backward Pass

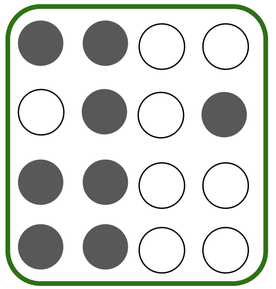


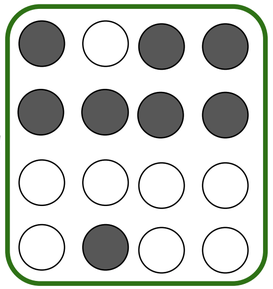
Forward Pass: $A = WX$

Backward Pass: $\nabla_X \mathcal{L} = W^T \nabla_A \mathcal{L}$

A case for dynamic sparsity

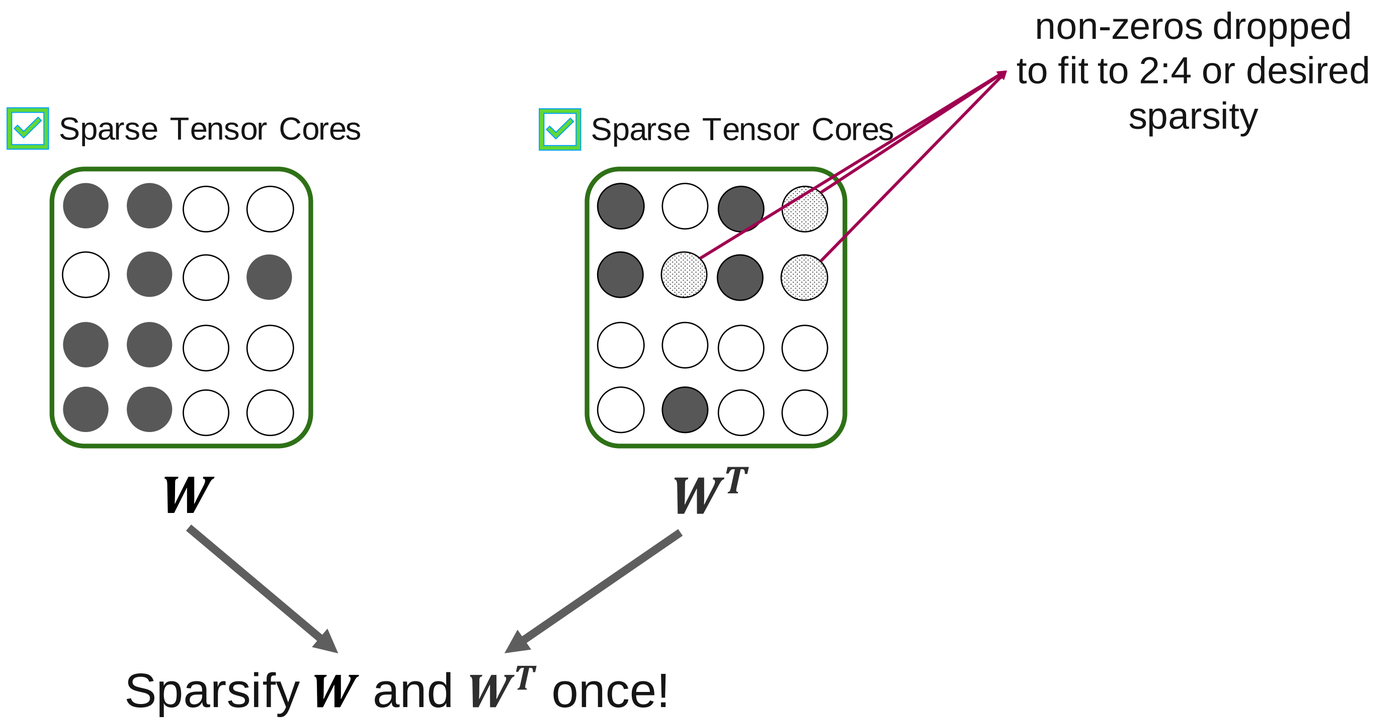
Training: Dynamic Sparse Weights



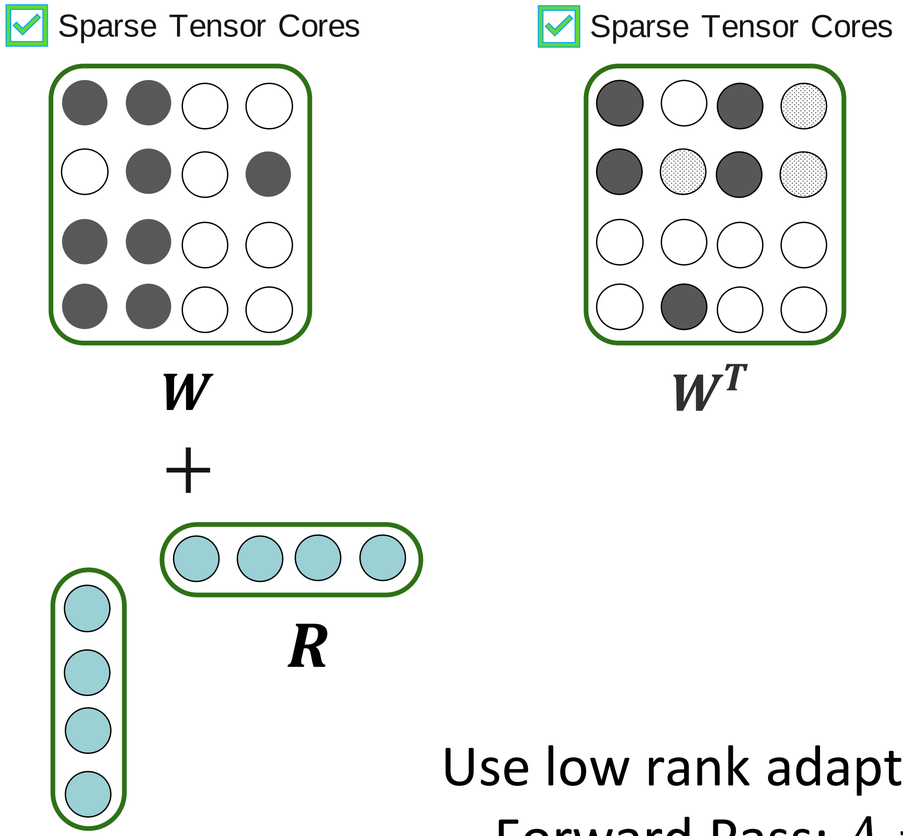


Sparsity patterns change between training iterations!

DNN Training Algorithms with Static Pruning



DNN Training Algorithms with Static Pruning



Model	Bert-Large	GPT2-Xlarge	GPT3-Large	GPT3-Xlarge	GPT3-2.7B
Speedup (×)	1.09	1.12	1.13	1.11	1.10

Dataset	Dense	$r = 0$	$r = 4$	$r = 16$	$r = 64$
SQuAD v1.1	90.4	89.1	89.1	89.2	89.5
GLUE	80.2	77.4	77.7	77.8	78.2

BERT-Large-Uncased Accuracy Results, r is rank

Use low rank adaptors for accuracy

Forward Pass: $A = WX + LRX$

SLoPe: Sparse Plus Lazy Low-rank Pretraining of LLMs, Mozaffari, Yazdanbakhsh, Dehnavi, Submitted to [ICML'24]

DNN Training Algorithms with Static Pruning

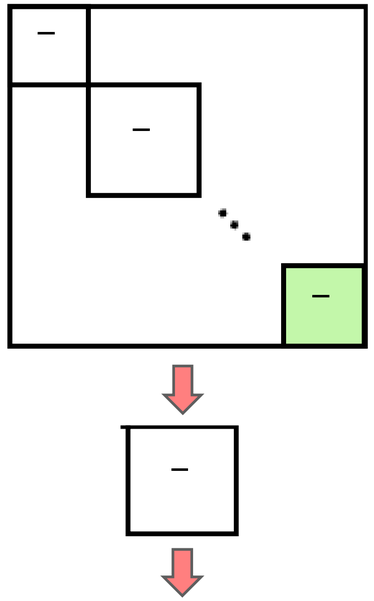
Second-Order optimizers at backward pass

Stochastic Gradient Decent:

$$W \leftarrow W - \alpha \nabla \mathcal{L}(W)$$

Natural Gradient Decent:

$$W \leftarrow W - \alpha F^{-1} \nabla \mathcal{L}(W)$$



Sparsification + quantization + rank-1 updates

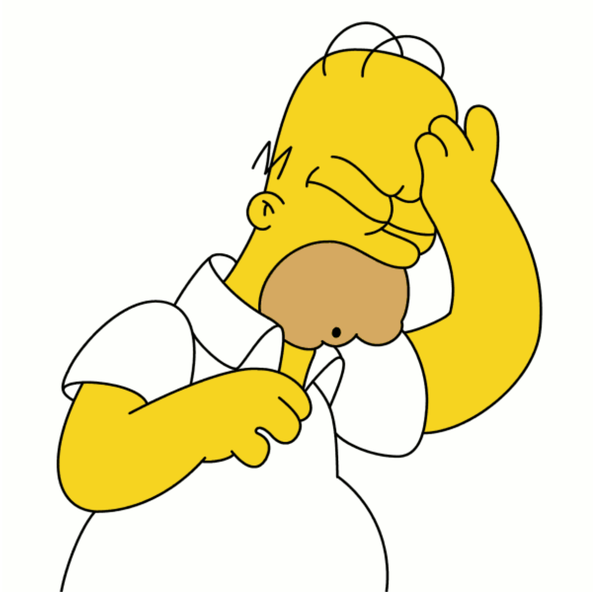
MKOR reduces Bert training time vs. LAMB 2.57 times

Second-Order optimizers for backward pass to control sparsity patterns:

- *HyLo: A Hybrid Low-Rank Natural Gradient Descent Method, Mu, Soori, Dehnavi [SC'22]*
- *MKOR: Momentum-Enabled Kronecker-Factor-Based Optimizer Using Rank-1 Updates, Mozaffari, Zhang, Dehnavi [NeurIPS'23]*

Programmability!

Not another programming language!



Input to the Domain-Specific Compiler

Numerical Method

Sparse matrix-vector multiplication (SpMV)

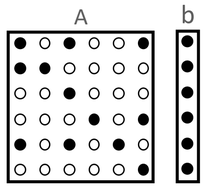
Sparsity info: Data Format, etc.

e.g. A is in Compressed Row (CSR) format

Programmability!

Input Program (NPB Benchmark)

```
for(j = 0; j < lastrow - firstrow + 1; j++){
  sum = 0.0;
  for(k = rowstr[j]; k < rowstr[j+1]; k++){
    sum = sum + A[k]*b[colidx[k]];
  }
  y[j] = sum;
}
```



≠

Input to the Domain-Specific Compiler

Numerical Method

Sparse matrix-vector multiplication (SpMV)

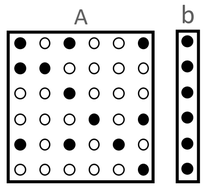
Sparsity info: Data Format, etc.

e.g. A is in Compressed Row (CSR) format

Automatically Translating Non-affine (e.g. sparse) Codes

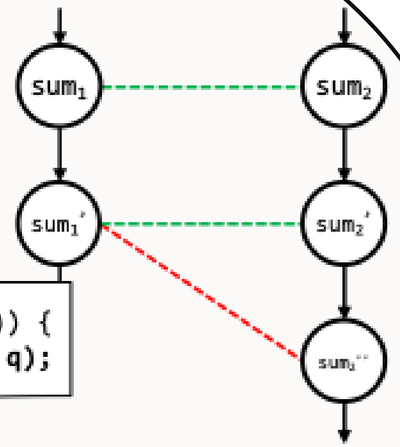
Input Program (NPB Benchmark)

```
for(j = 0; j < lastrow - firstrow + 1; j++){
  sum = 0.0;
  for(k = rowstr[j]; k < rowstr[j+1]; k++){
    sum = sum + A[k]*b[colidx[k]];
  }
  y[j] = sum;
}
```



Automatic Translator

Either, both codes process a non-zero element...




```
if (preconditions(a, rowstr, colidx)) {
  very_fast_spmv(a, rowstr, ..., p, q);
```

Equality Saturation

Backend

```
mkl_sparse_d_mv(...);
```

Backend

```
cusparseSpMV(...);
```

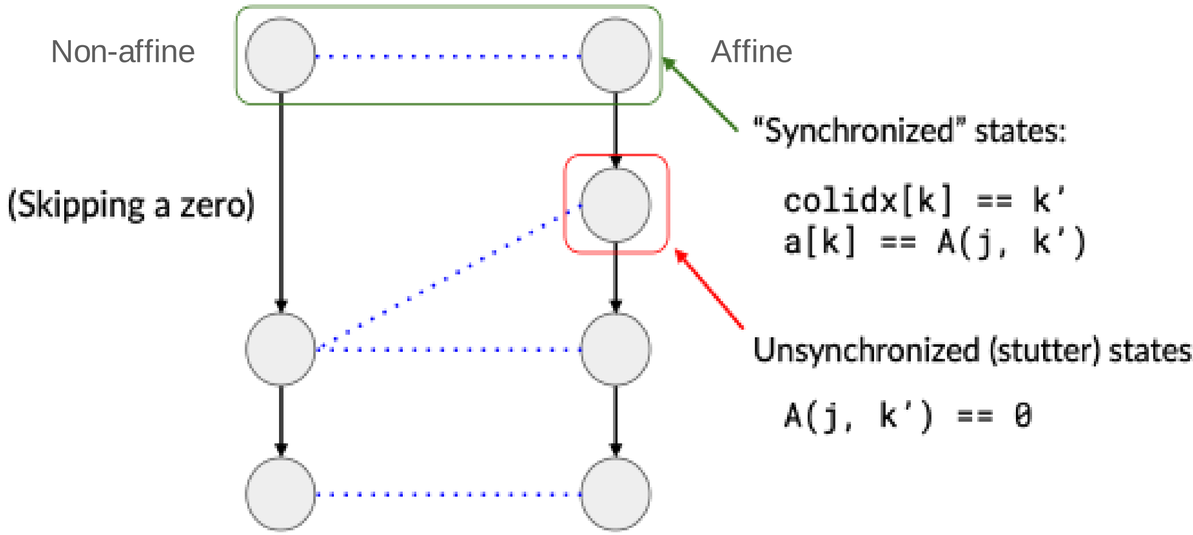
Backend

DOMAIN SPECIFIC LANGUAGE
(SYMPILER, TACO, REV,...)

```
#pragma omp parallel for ...
```

Automatically Translating Non-affine Code

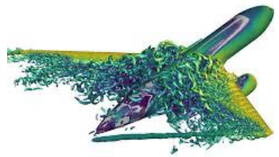
Rev: ***Automatically translating*** non-affine codes with branching bisimulation, *Laird, Liu, Bjorner, Dehnavi, to appear at* [PLDI'24].

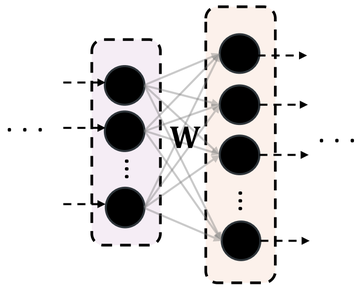


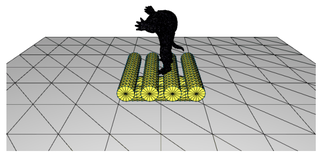
Sparse computation data dependence simplification for efficient ***compiler-generated inspectors***, *Mohammadi, et. al.*[PLDI'19].

Take Aways

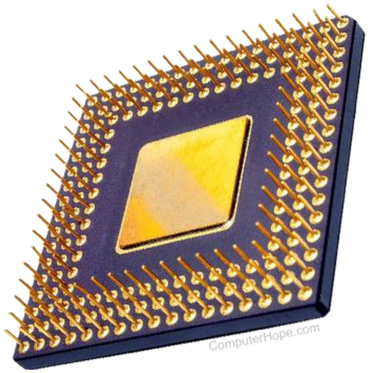
Sparse Algorithms & Applications





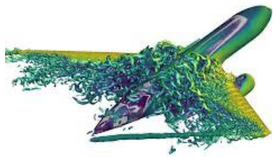


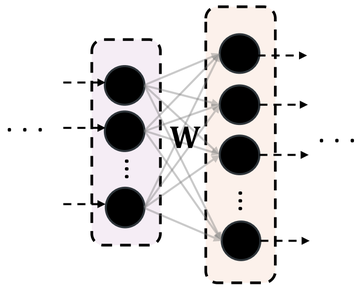
Hardware

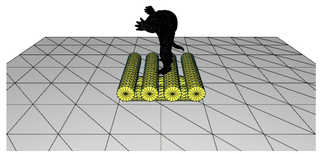


Take Aways

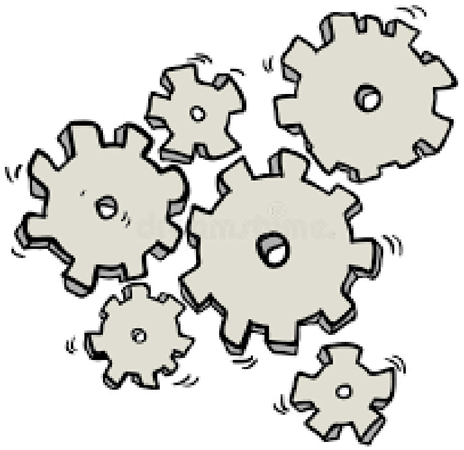
Sparse Algorithms & Applications



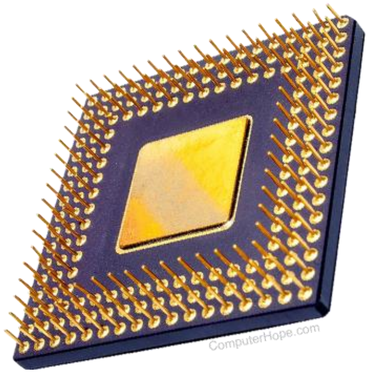




Compilers for sparsity

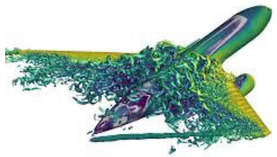


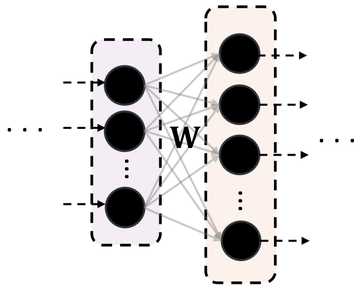
Hardware

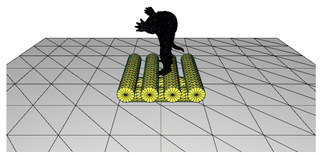


Future!

Sparse Algorithms & Applications

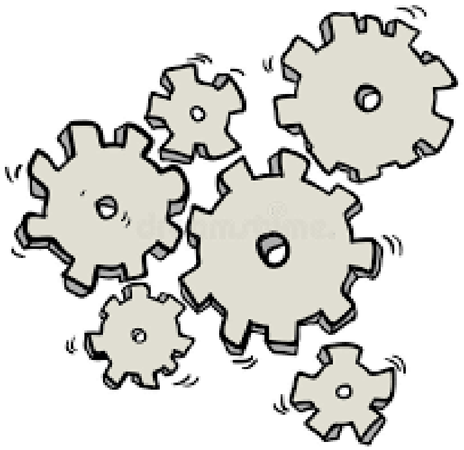




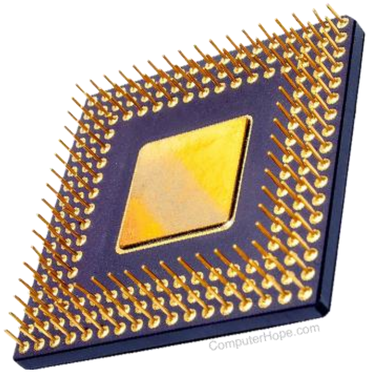


Compilers for sparsity

Hardware



Codesign



http://www.paramathic.com