Scaling the Memory Wall for Sparse Iterative Solvers

Jonas Thies
Delft High Performance Computing Center

Christie Alappatt, Gerhard Wellein & Georg Hager
University of Erlangen-Nuremberg

A “Bread & Butter” Simulation

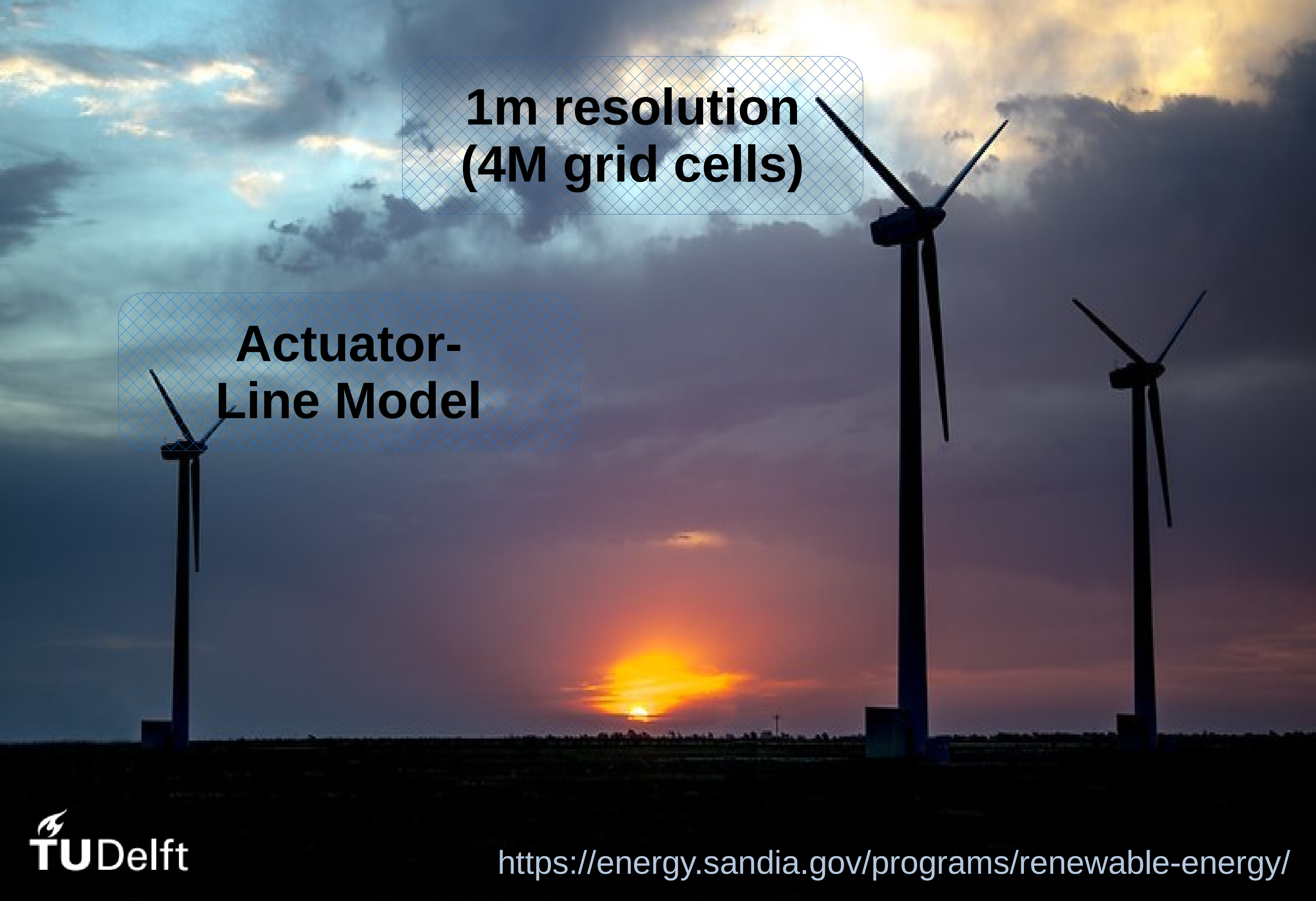
https://energy.sandia.gov/programs/renewable-energy/

A “Bread & Butter” Simulation

https://energy.sandia.gov/programs/renewable-energy/

TUDelft

A “Bread & Butter” Simulation



https://energy.sandia.gov/programs/renewable-energy/

A “Bread & Butter” Simulation

1m resolution (4M grid cells)

Actuator-Line Model

Implicit Time-Stepping

https://energy.sandia.gov/programs/renewable-energy/

A “Bread & Butter” Simulation

**1m resolution
(4M grid cells)**

**Actuator-
Line Model**

**Implicit
Time-Stepping**

**Two Sparse
Systems
per Time-Step**

TU Delft

https://energy.sandia.gov/programs/renewable-energy/

GMRES for the Momentum Equations

Build orthogonal basis of a Krylov subspace

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{q_1, q_2, \ldots, q_m\}$$

Using the Arnoldi process

$$AQ_k = Q_{k+1}H_k.$$

GMRES for the Momentum Equations

Build orthogonal basis of a Krylov subspace

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{q_1, q_2, \ldots, q_m\}$$

Using the Arnoldi process

$$AQ_k = Q_{k+1} H_k .$$

One vector requires 64MB of RAM

GMRES for the Momentum Equations

Build orthogonal basis of a Krylov subspace

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{q_1, q_2, \ldots, q_m\}$$

Using the Arnoldi process

$$AQ_k = Q_{k+1}H_k.$$

One vector requires 64MB of RAM

Needs only 20-40 Iterations

GMRES for the Momentum Equations

Build orthogonal basis of a Krylov subspace

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{q_1, q_2, \ldots, q_m\}$$

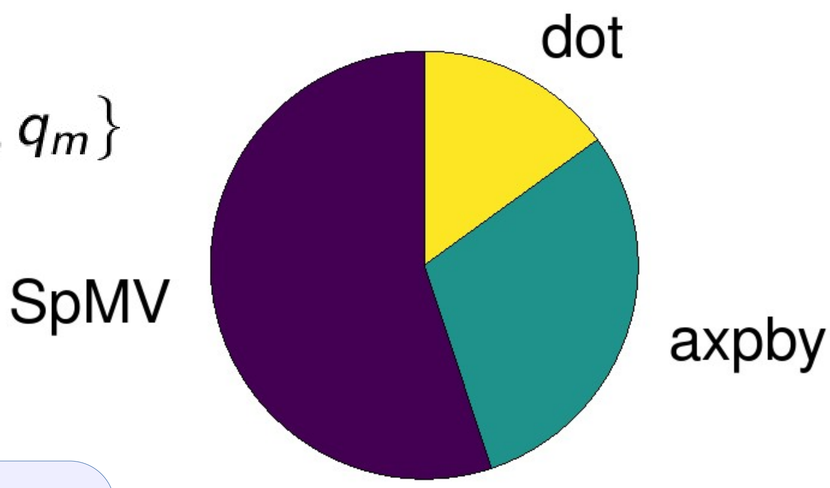
Using the Arnoldi process

$$AQ_k = Q_{k+1}H_k.$$

One vector requires 64MB of RAM

Needs only 20-40 Iterations

Runtime breakdown on 48-core Xeon
(1536 Flops/cycle)



GMRES for the Momentum Equations

Build orthogonal basis of a Krylov subspace

$$\mathcal{K}_m(A, r_0) = \mathrm{span}\{q_1, q_2, \ldots, q_m\}$$

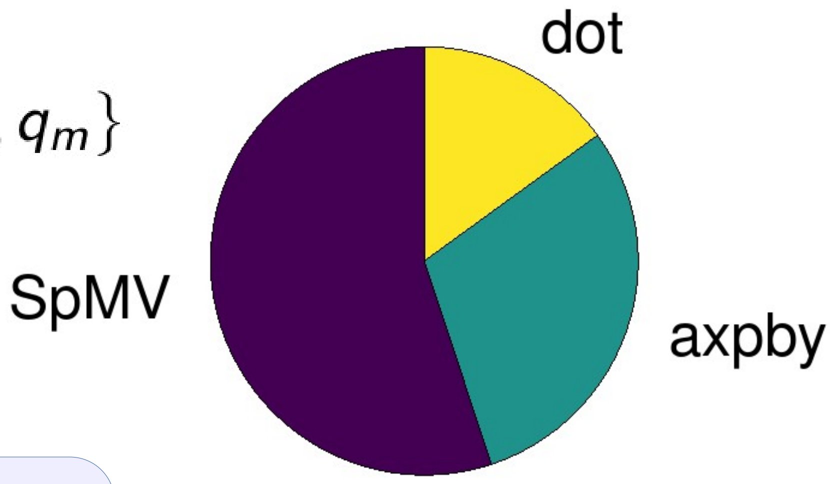
Using the Arnoldi process

$$AQ_k = Q_{k+1}H_k.$$

One vector requires 64MB of RAM

Needs only 20-40 Iterations

Runtime breakdown on 48-core Xeon
(1536 Flops/cycle)



What is “Communication”, And how can it be “avoided”

?

“Communication Avoiding” Krylov

- Partially use polynomial with constant coefficients
- Fewer orthogonalization operations
- More compute-intensive ‘tall&skinny’ operations
- Bundle reductions
- **Glue SpMV’s together (Matrix-Power Kernel):**

$$y^k = \sum_{k=1}^{s} \alpha_k A^k x$$

“Communication Avoiding” Krylov

Synchronization Points

- ...ce polynomial with constant coefficients
- Few... onalization operations
- More c... ensive ‘tall&skinny’ operations
- Bundle re...
- **Glue SpM...’s together (Matrix-Power Kernel):**

$$y^k = \sum_{k=1}^{s} \alpha_k A^k x$$

“Communication Avoiding” Krylov

Synchronization Points

Communication Volume

- …ce polynomial with con… …cients
- Few… …nalization ope…
- More c… …ensive ‘… …ny’ operations
- Bundle re…
- **Glue Sp… …together (Matrix-Power Kernel):**

$$y^k = \sum_{k=1}^{s} \alpha_k A^k x$$

“Communication Avoiding” Krylov

Synchronization Points

Communication Volume

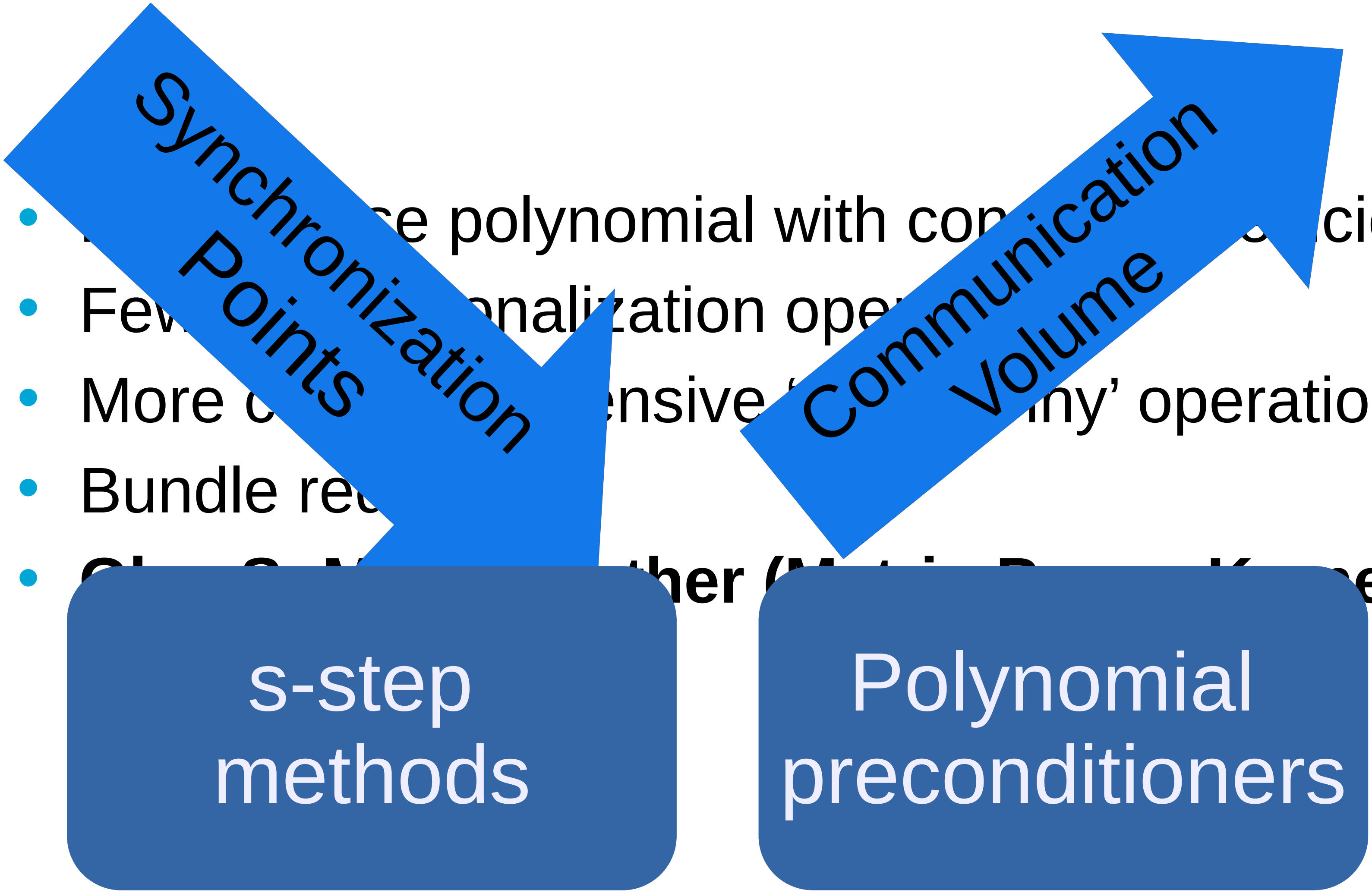
- ...e polynomial with co... ...cients
- Few ...nalization ope...
- More c... ...ensive ‘... ...ny’ operations
- Bundle re...
- **...ther (Matrix-Power Kernel):**

s-step methods

$$y^k = \sum_{k=1}^{s} \alpha_k A^k x$$

“Communication Avoiding” Krylov

- ...e polynomial with co... ...cients
- Few... ...nalization ope...
- More c... ...ensive ‘... ...iny’ operations
- Bundle re...
- **...her (M... ...el):**



Simple Example:
Neumann Polynomial Preconditioner

Simple Example: Neumann Polynomial Preconditioner

Neumann series

$$\frac{1}{x} \approx \sum_{k=1}^{s} \left(1 - x\right)^{k}$$

Simple Example: Neumann Polynomial Preconditioner

Neumann series

$$\frac{1}{x} \approx \sum_{k=1}^{s} (1-x)^k$$

Jacobi splitting

$$A = D - (L + U)$$

Simple Example: Neumann Polynomial Preconditioner

Neumann series

$$\frac{1}{x} \approx \sum_{k=1}^{s} (1-x)^k$$

Jacobi splitting

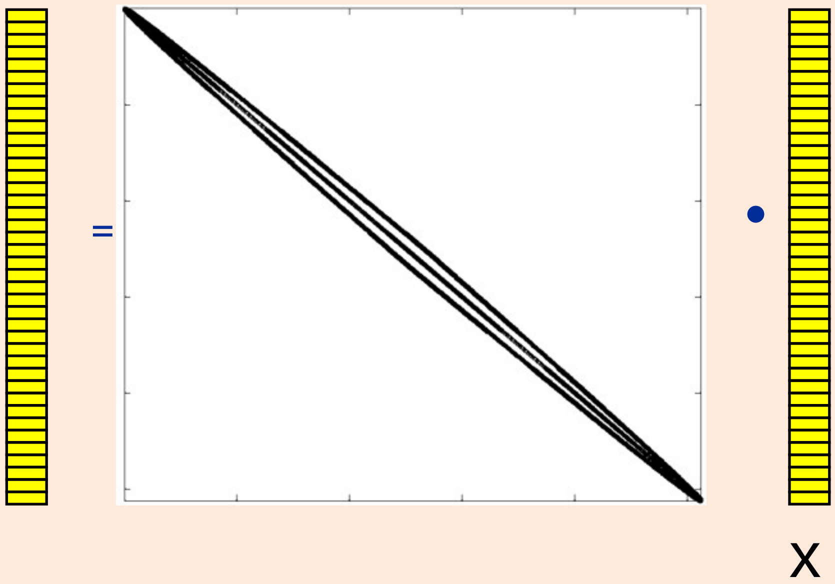
$$A = D - (L + U)$$

Degree-s preconditioner

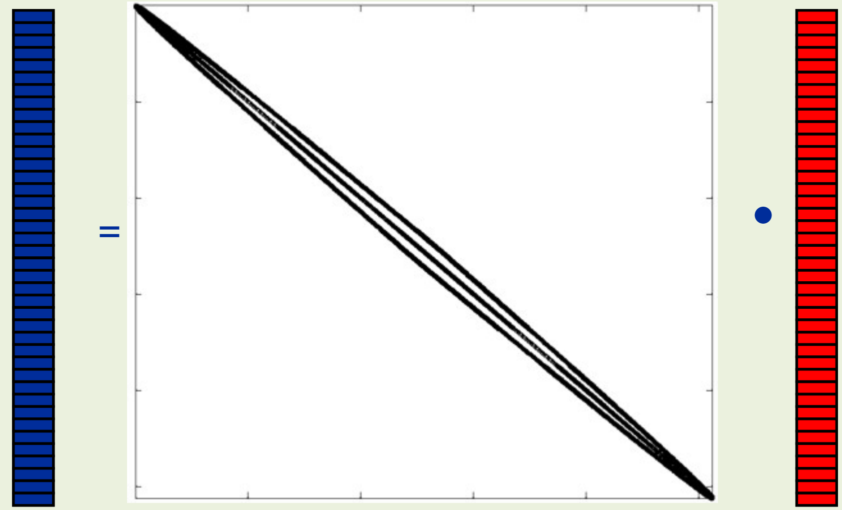
$$(D^{-1}A)^{-1} v \approx [D^{-1}(L+U)]^s D^{-1} v$$

Matrix Power Kernel

“Back to back” SpMVs

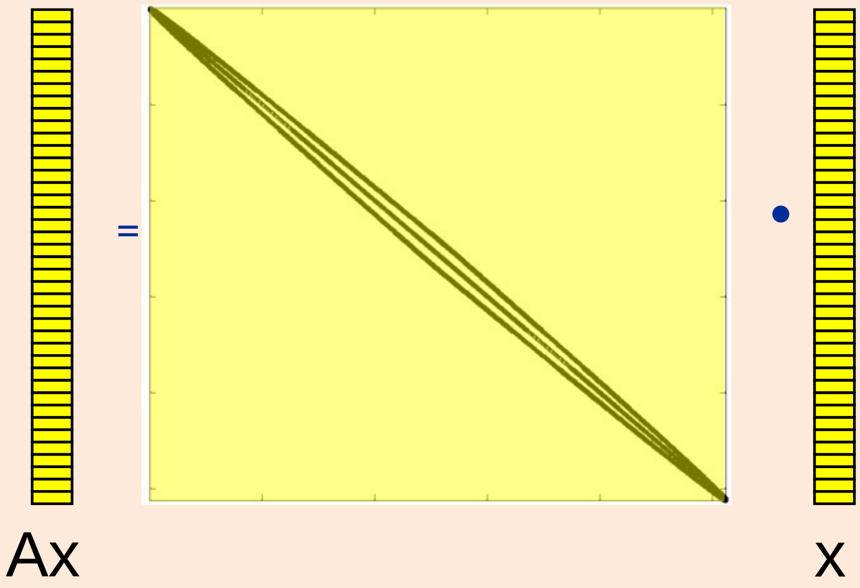


RACE MPK

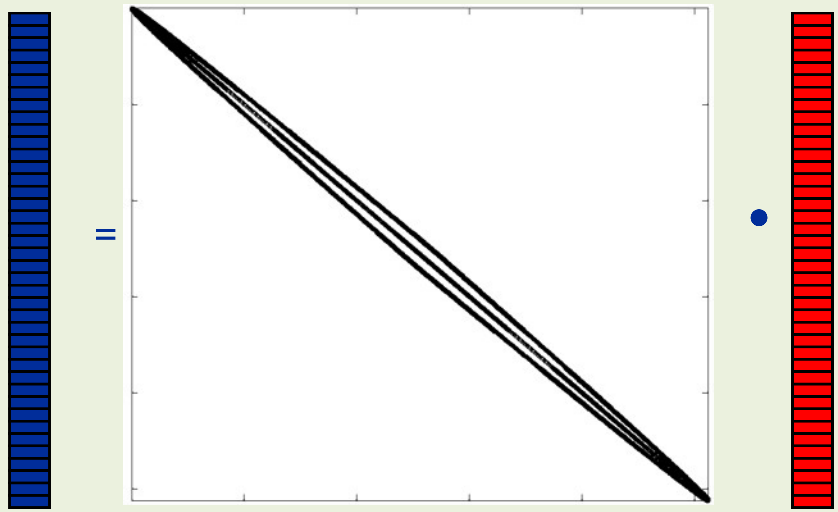


Matrix Power Kernel

“Back to back” SpMVs

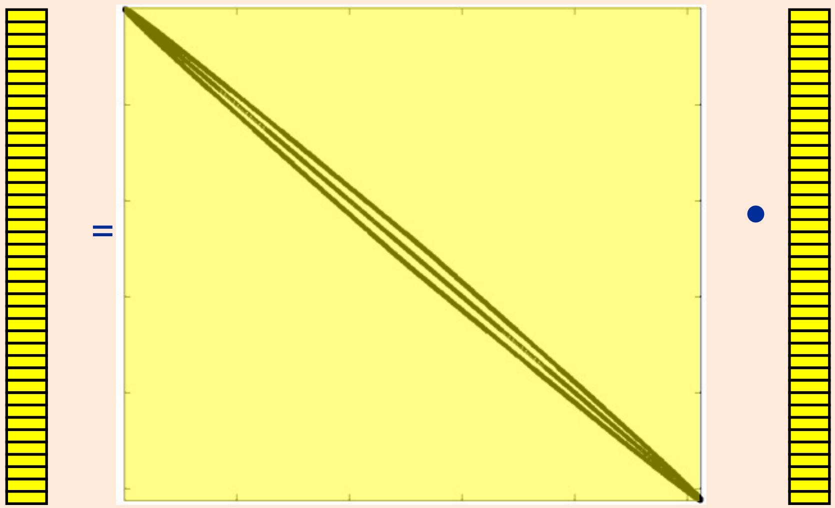


RACE MPK



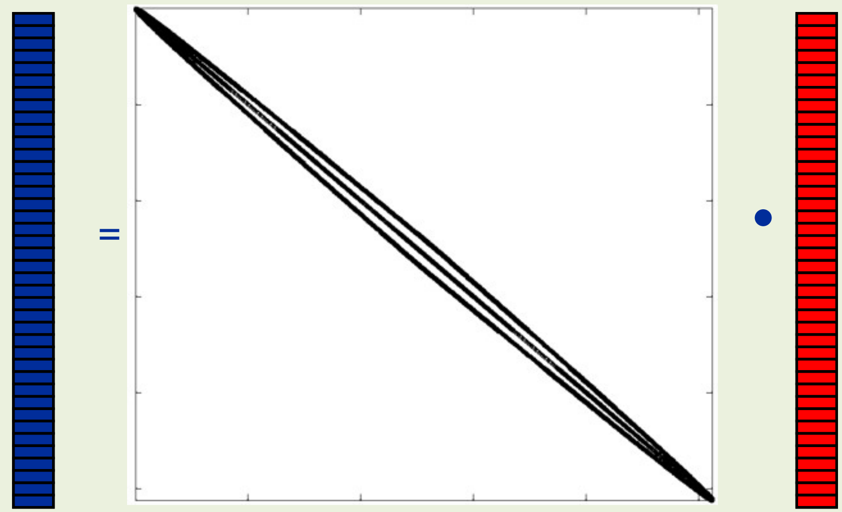
Matrix Power Kernel

“Back to back” SpMVs



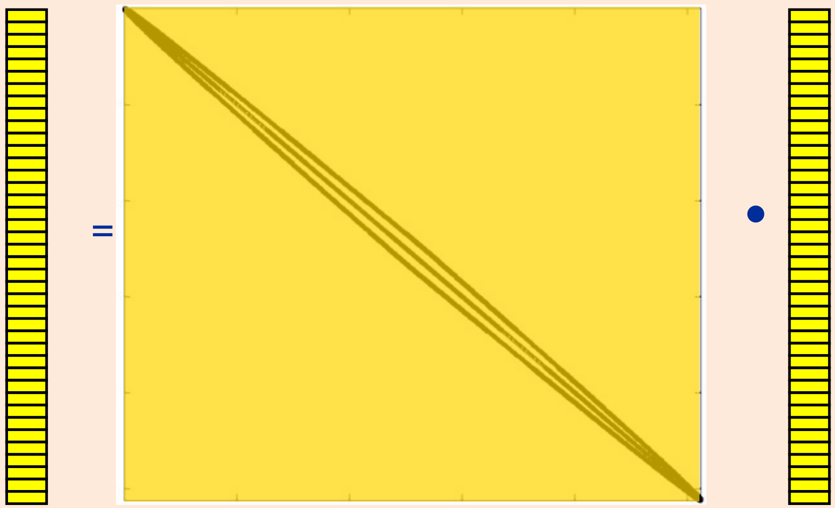
Ax

RACE MPK



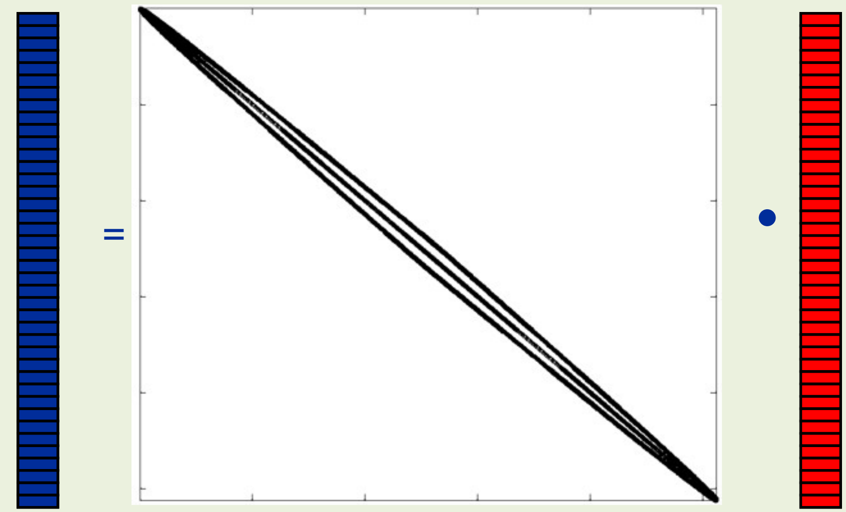
Matrix Power Kernel

“Back to back” SpMVs



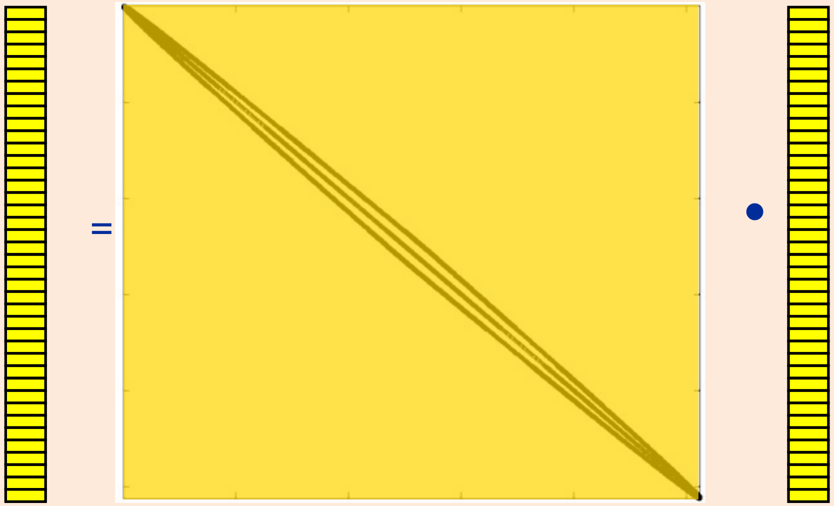
A^2x

RACE MPK



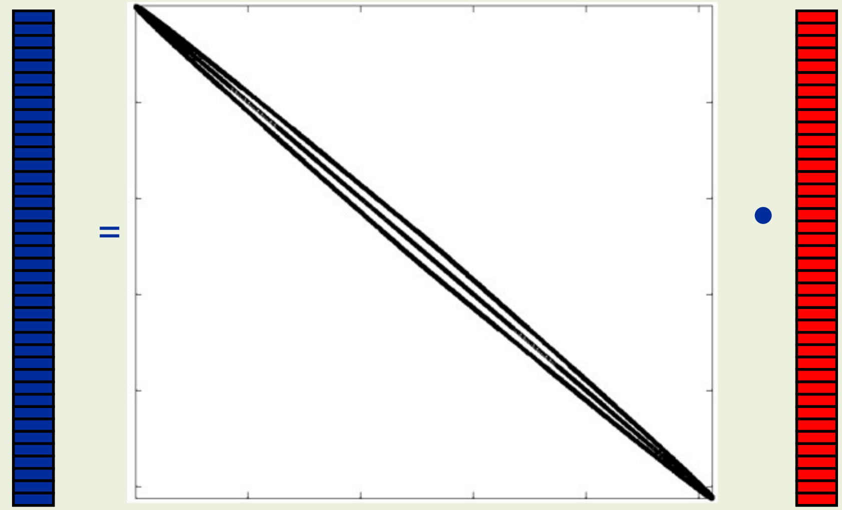
Matrix Power Kernel

“Back to back” SpMVs



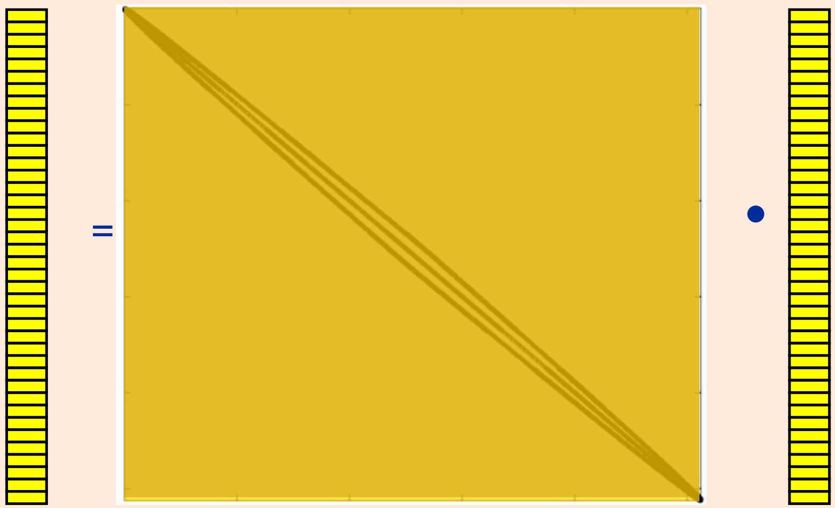
A^2x

RACE MPK



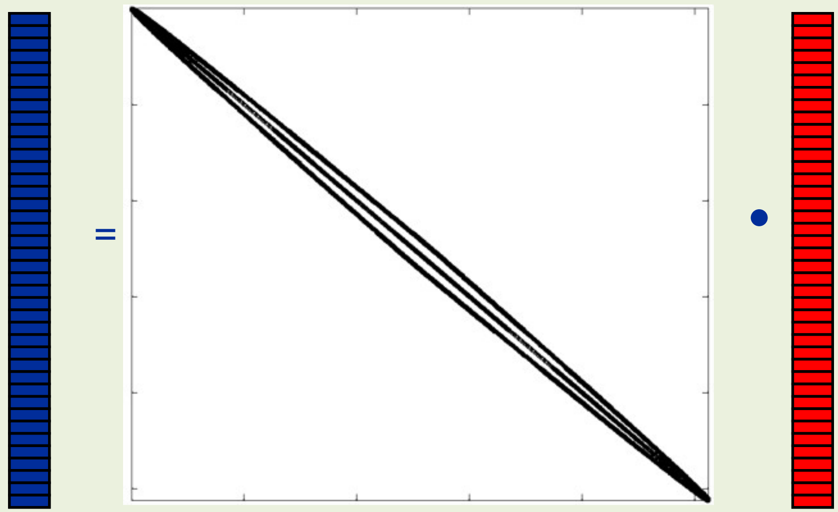
Matrix Power Kernel

“Back to back” SpMVs



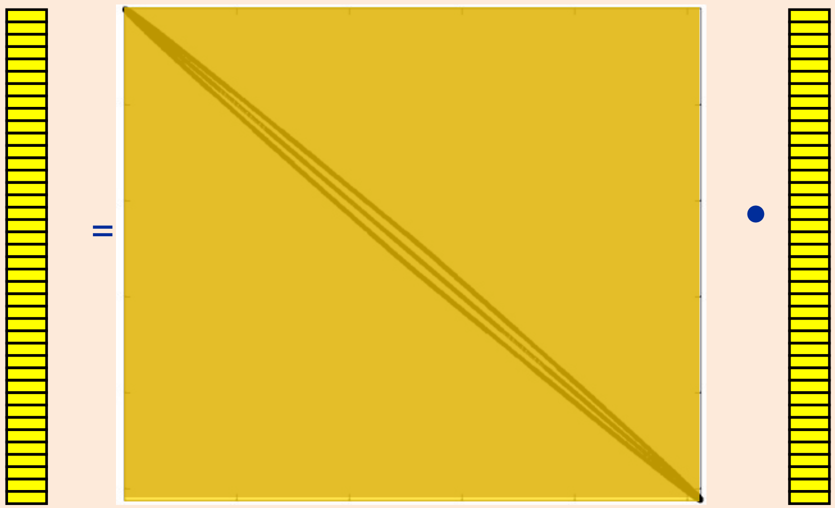
A^3x

RACE MPK



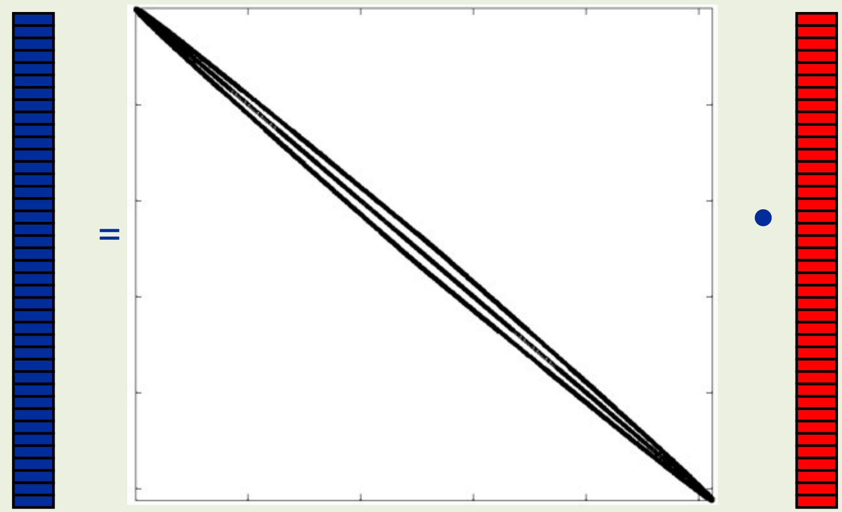
Matrix Power Kernel

“Back to back” SpMVs



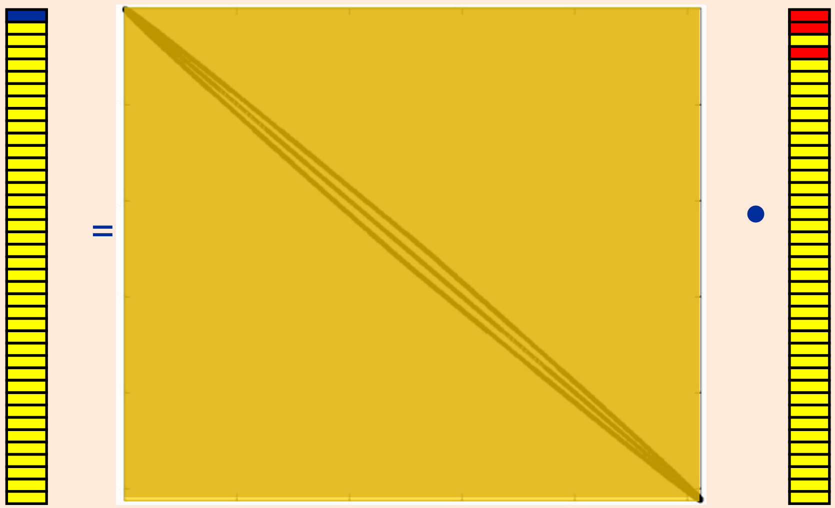
Matrix accessed 3 times from memory

RACE MPK



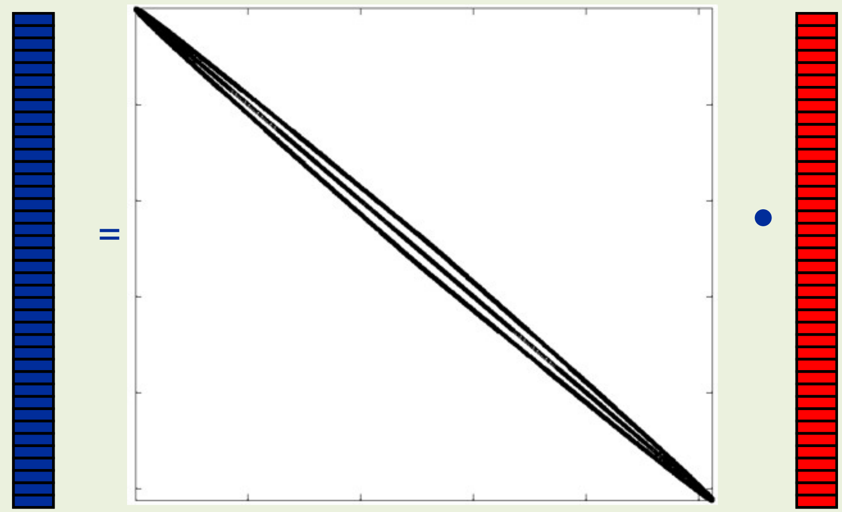
Matrix Power Kernel

“Back to back” SpMVs



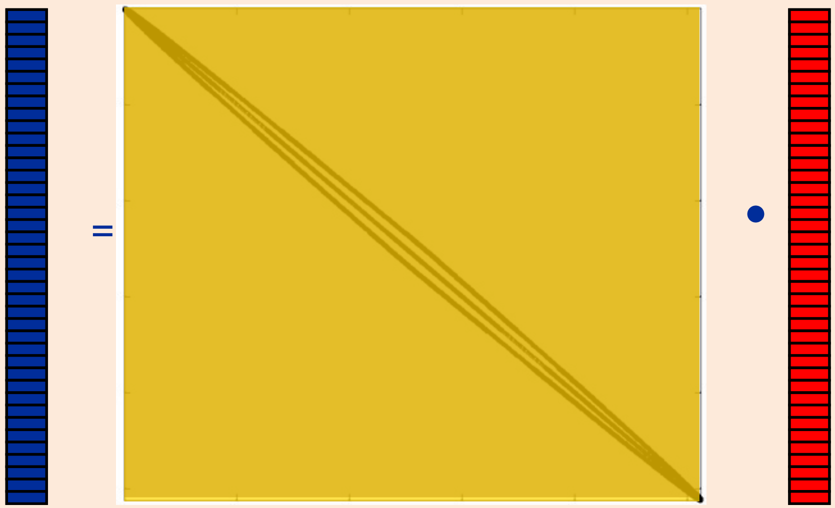
Matrix accessed 3 times from memory

RACE MPK



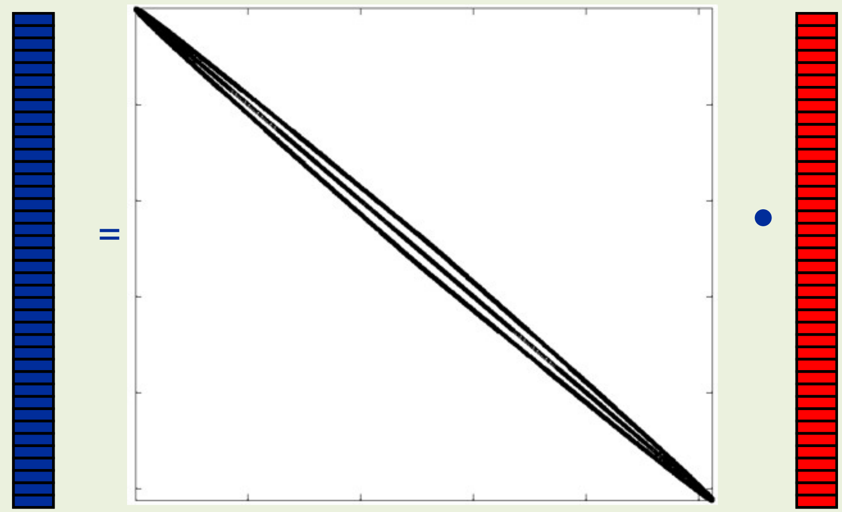
Matrix Power Kernel

“Back to back” SpMVs



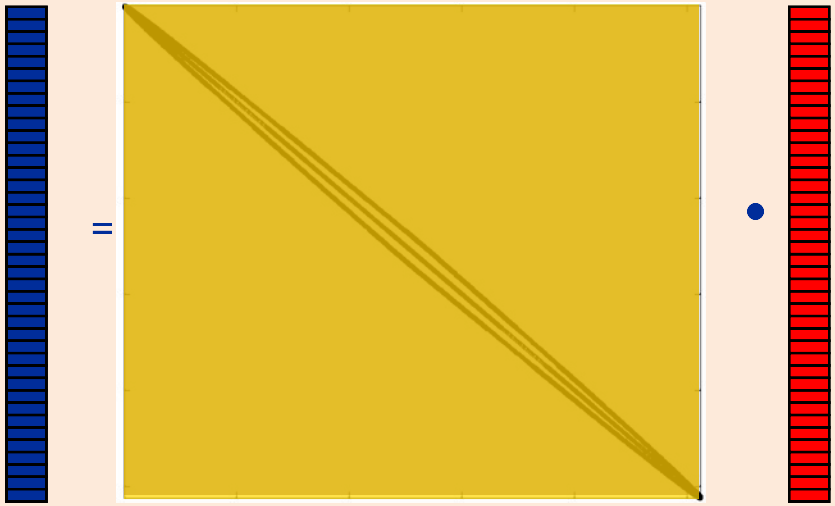
Matrix accessed 3 times from memory X

RACE MPK



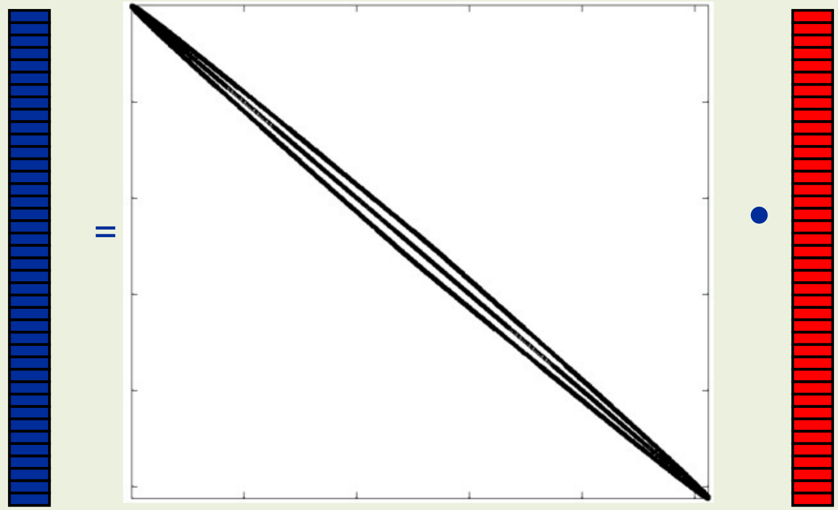
Matrix Power Kernel

“Back to back” SpMVs



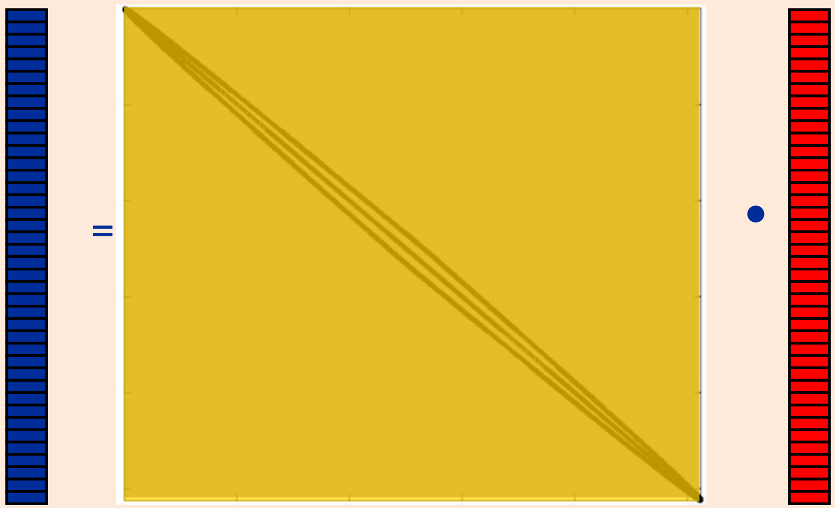
Matrix accessed 3 times from memory X

RACE MPK



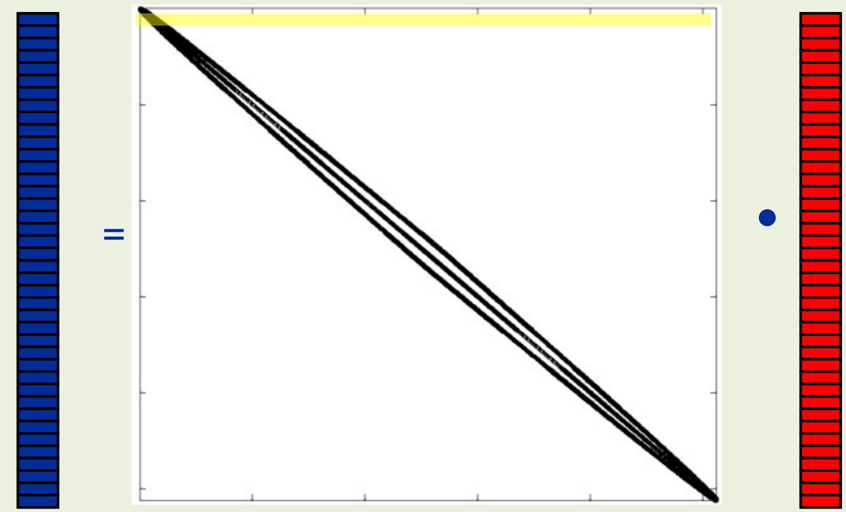
Matrix Power Kernel

"Back to back" SpMVs

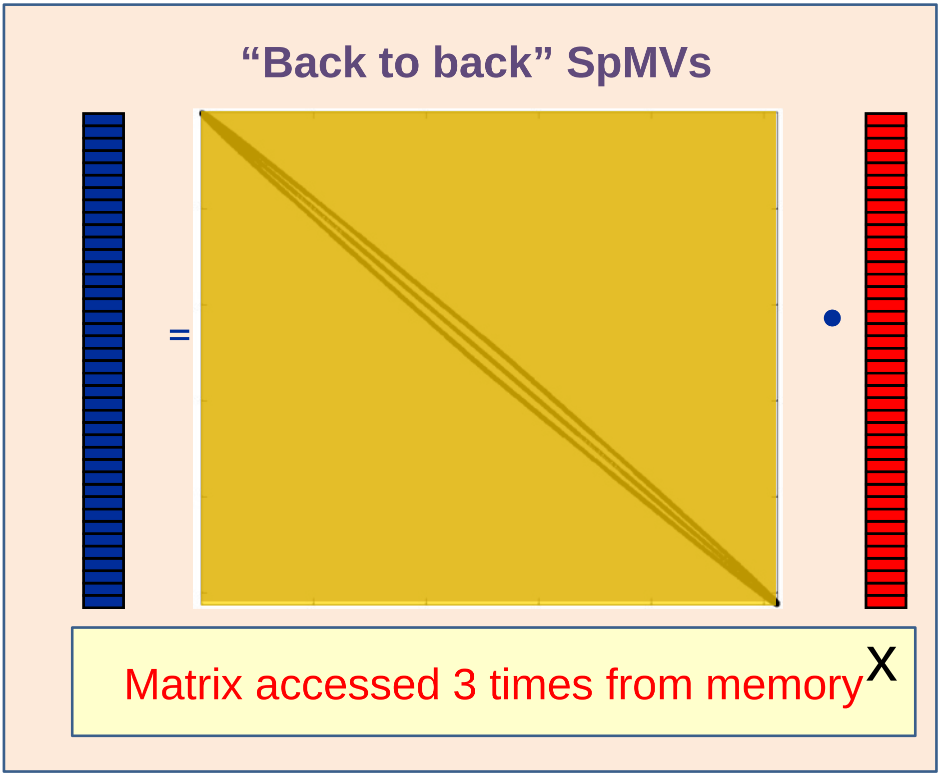


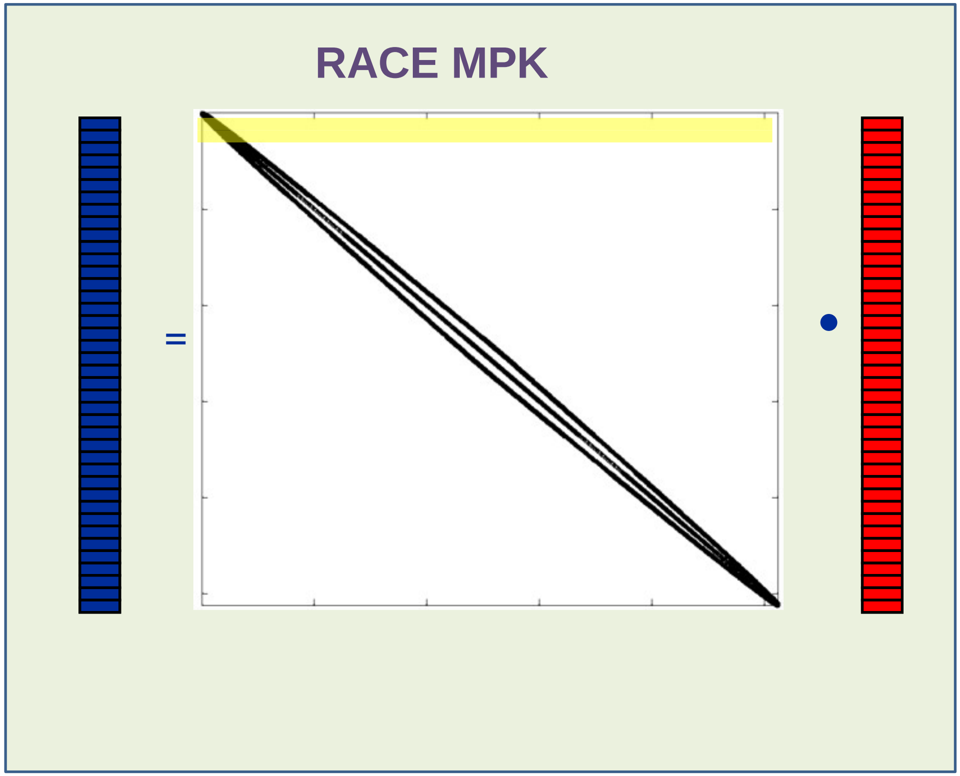
Matrix accessed 3 times from memory X

RACE MPK



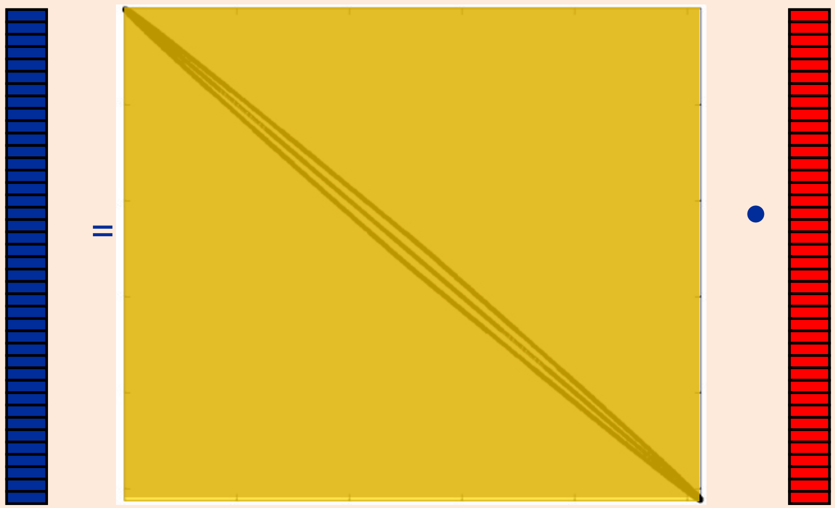
Matrix Power Kernel





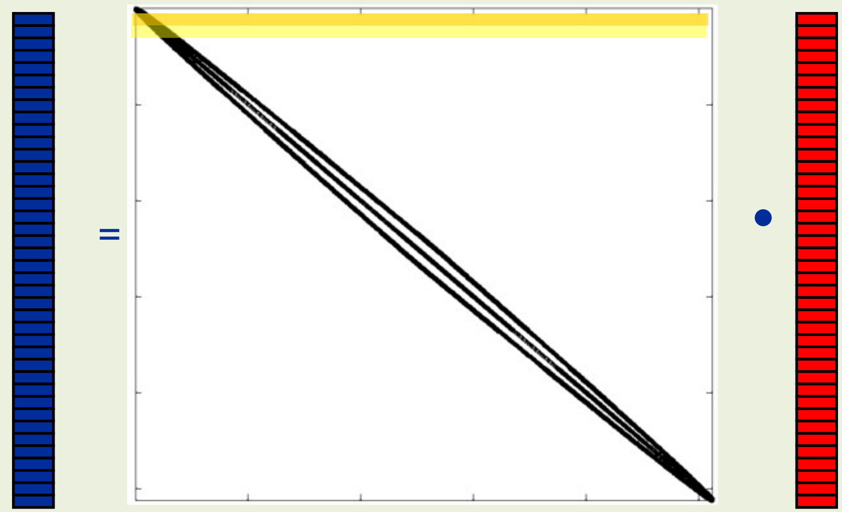
Matrix Power Kernel

“Back to back” SpMVs



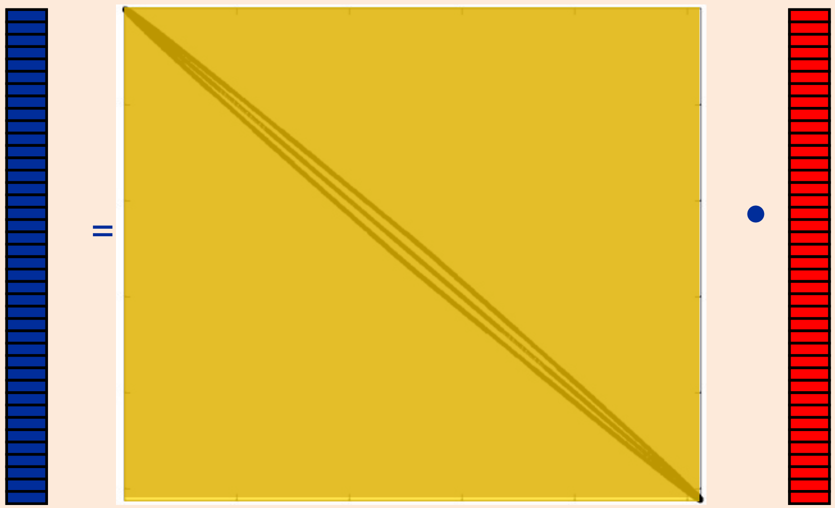
Matrix accessed 3 times from memory X

RACE MPK



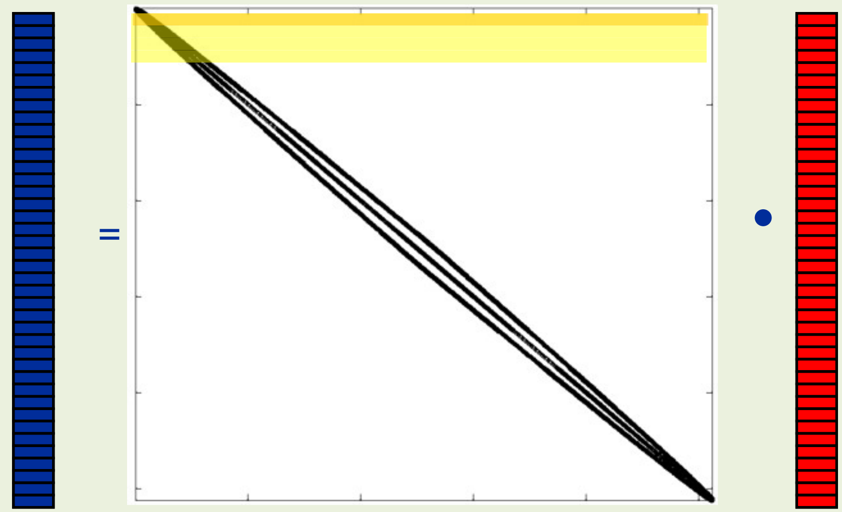
Matrix Power Kernel

“Back to back” SpMVs



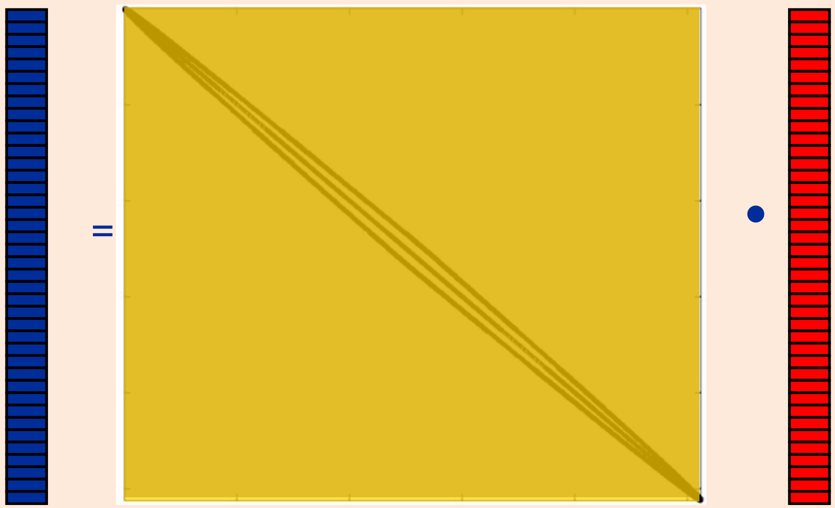
Matrix accessed 3 times from memory X

RACE MPK



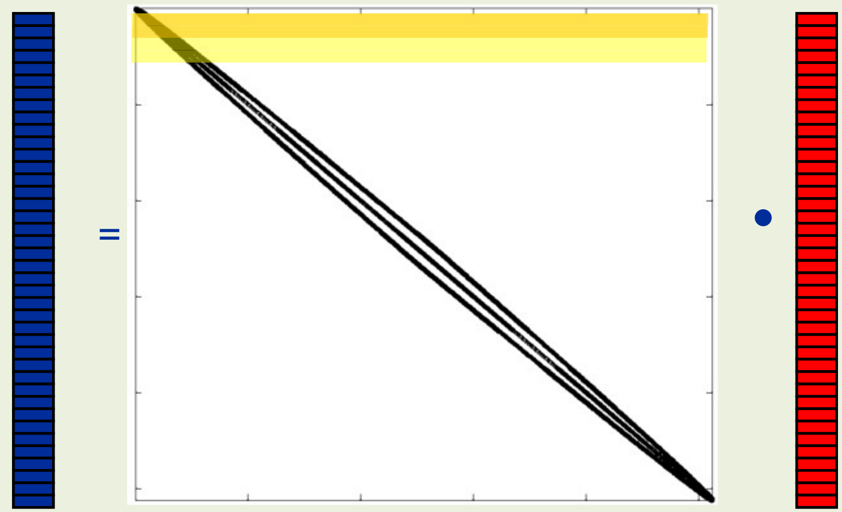
Matrix Power Kernel

“Back to back” SpMVs



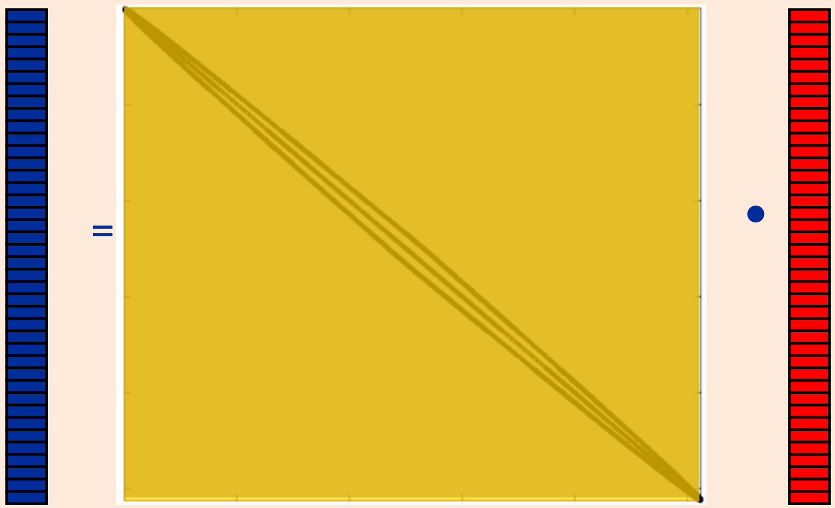
Matrix accessed 3 times from memory X

RACE MPK



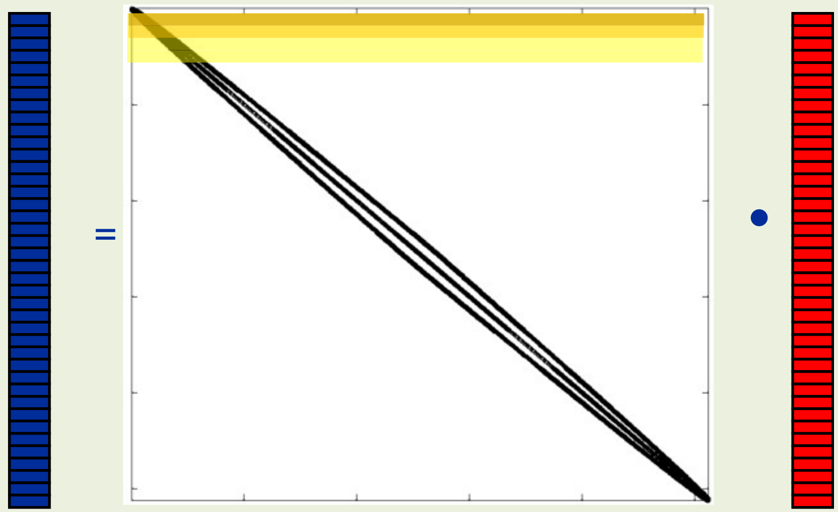
Matrix Power Kernel

"Back to back" SpMVs



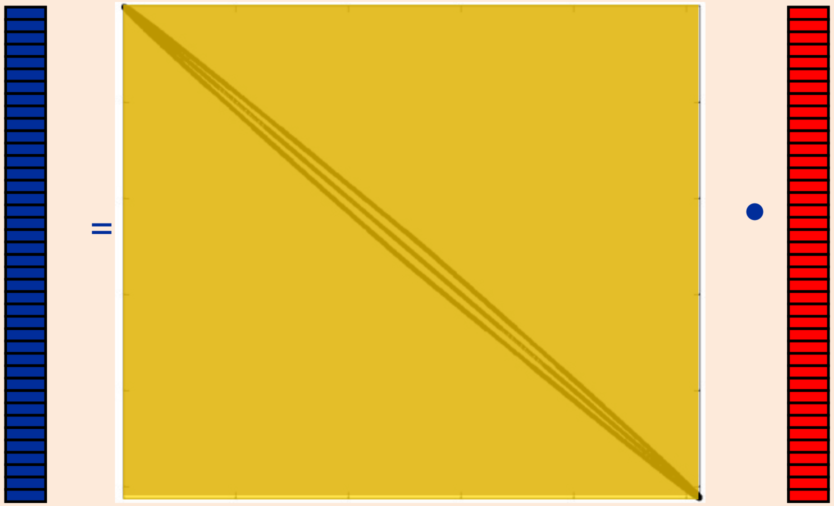
Matrix accessed 3 times from memory

RACE MPK



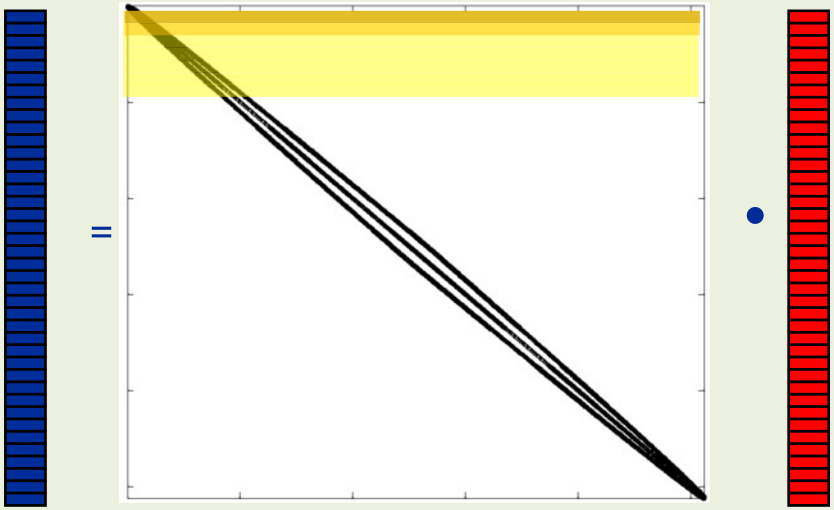
Matrix Power Kernel

“Back to back” SpMVs



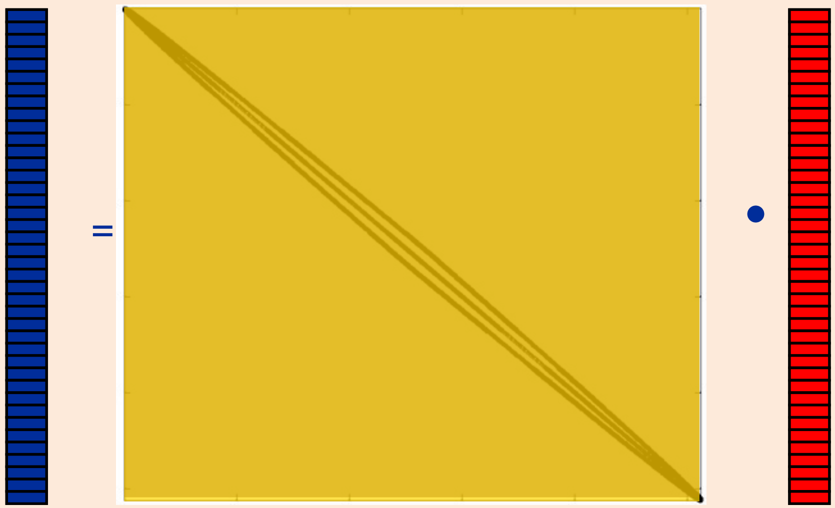
Matrix accessed 3 times from memory X

RACE MPK



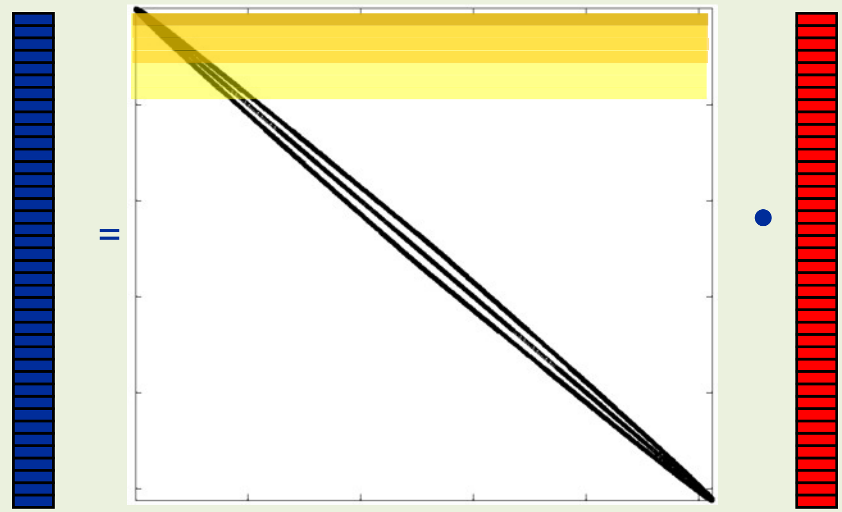
Matrix Power Kernel

“Back to back” SpMVs



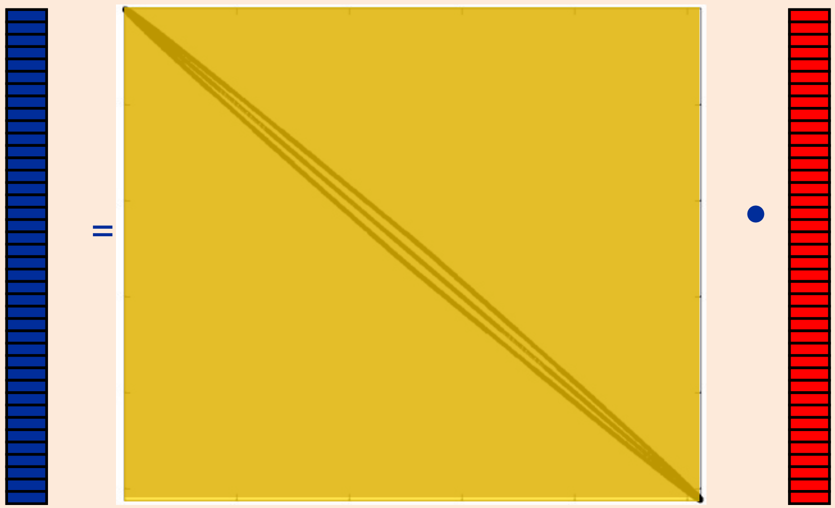
Matrix accessed 3 times from memory X

RACE MPK



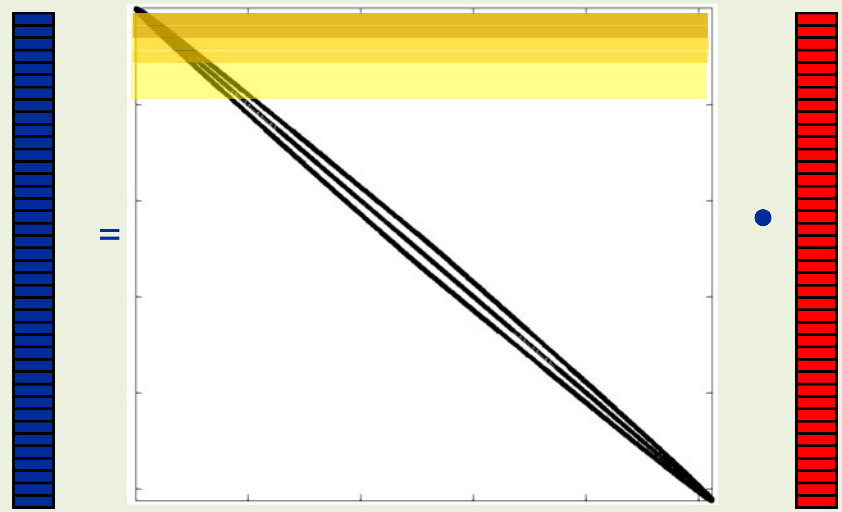
Matrix Power Kernel

“Back to back” SpMVs



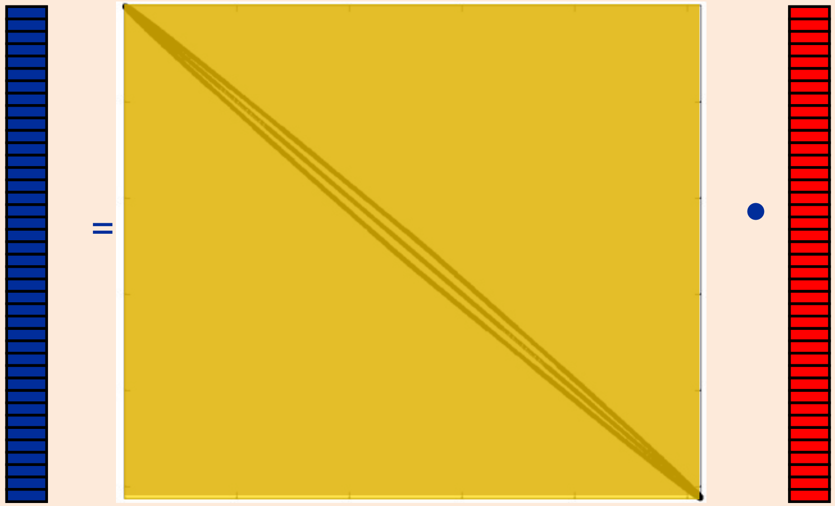
Matrix accessed 3 times from memory X

RACE MPK



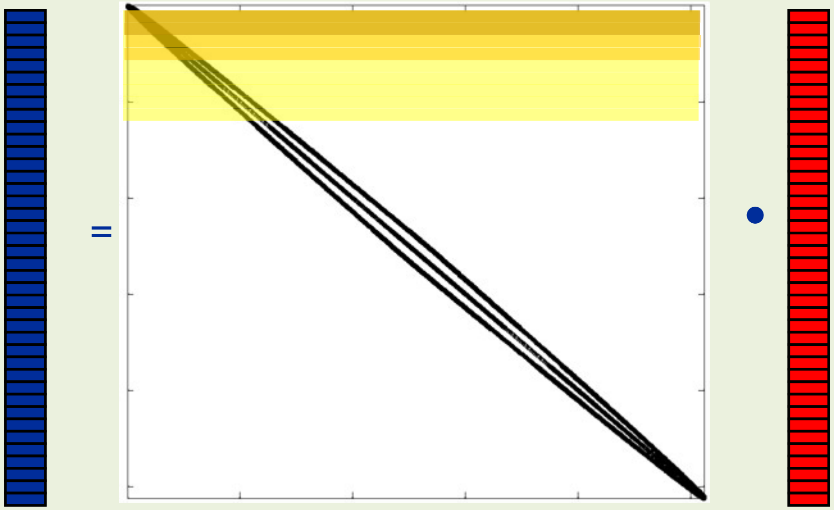
Matrix Power Kernel

"Back to back" SpMVs



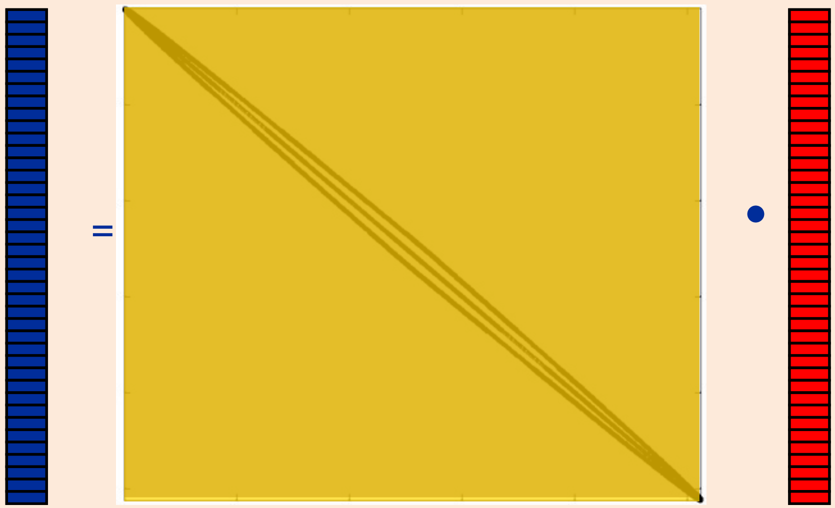
Matrix accessed 3 times from memory X

RACE MPK



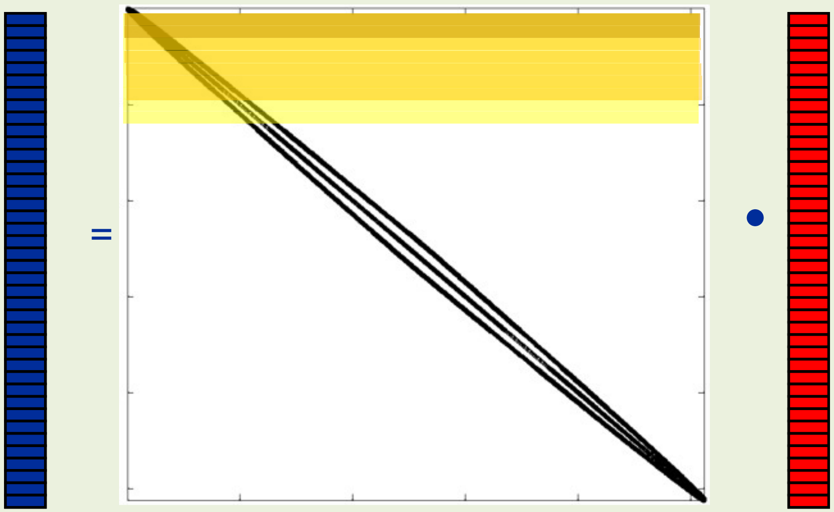
Matrix Power Kernel

“Back to back” SpMVs



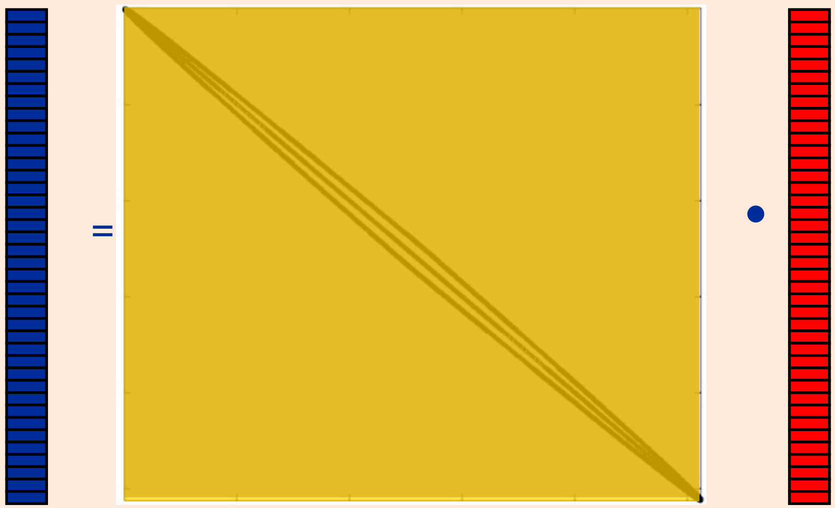
Matrix accessed 3 times from memory X

RACE MPK



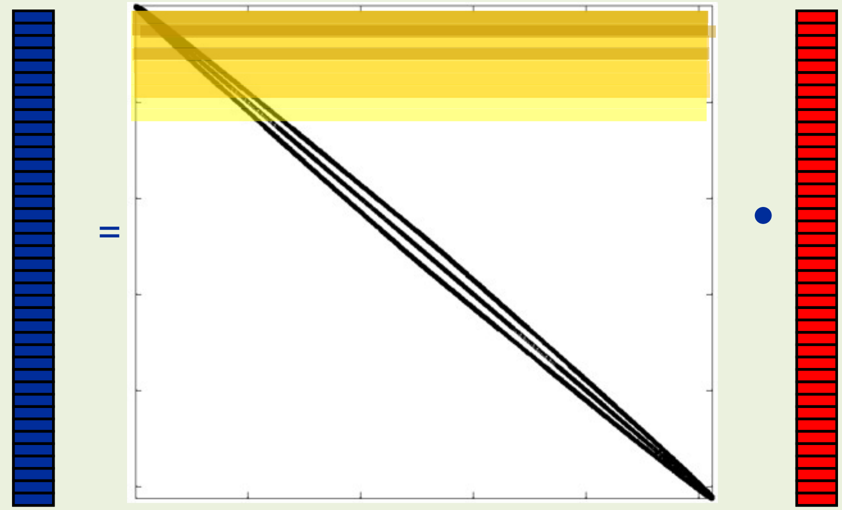
Matrix Power Kernel

“Back to back” SpMVs



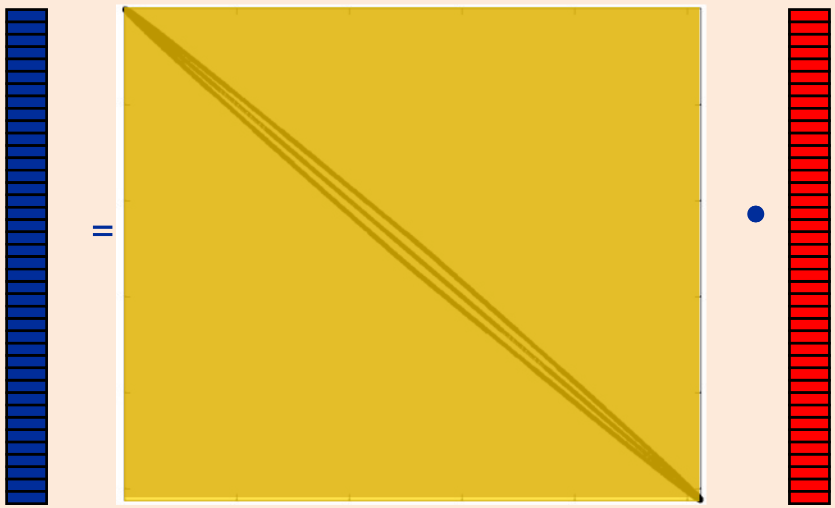
Matrix accessed 3 times from memory

RACE MPK



Matrix Power Kernel

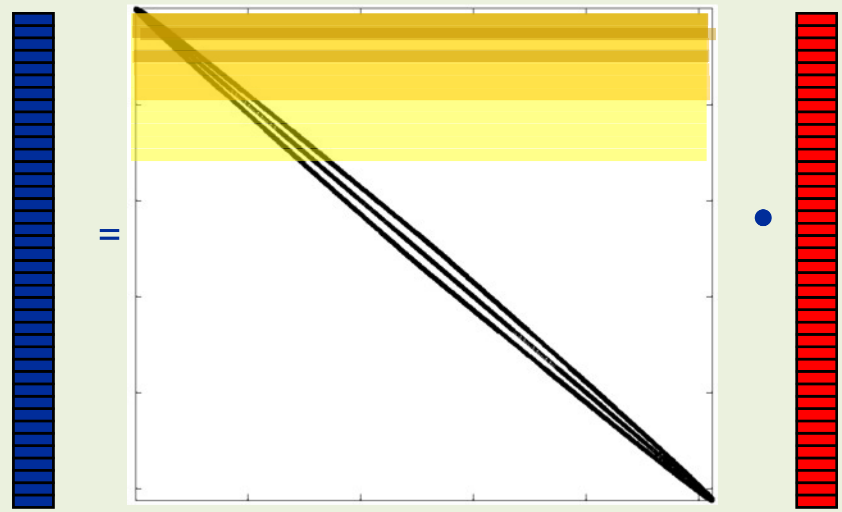
“Back to back” SpMVs



Matrix accessed 3 times from memory

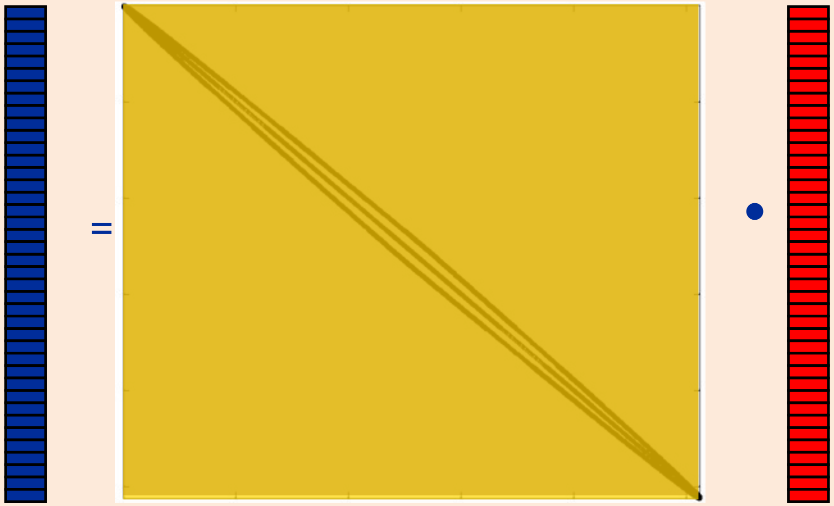
X

RACE MPK



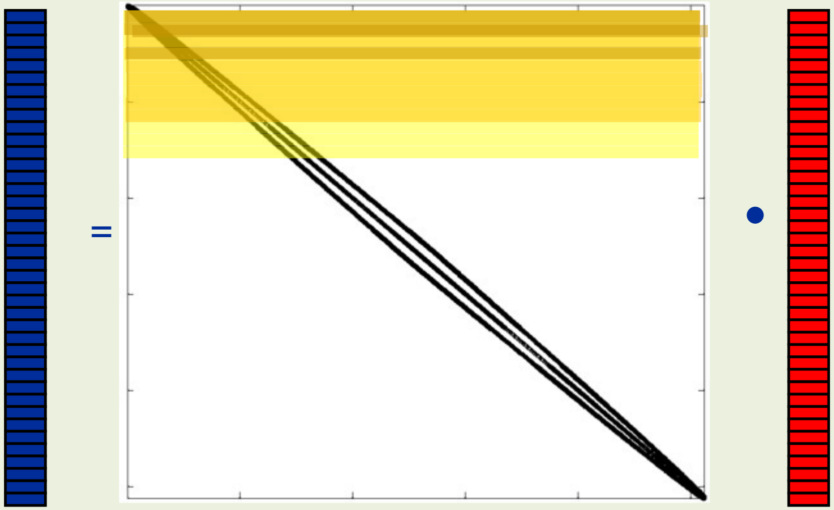
Matrix Power Kernel

“Back to back” SpMVs



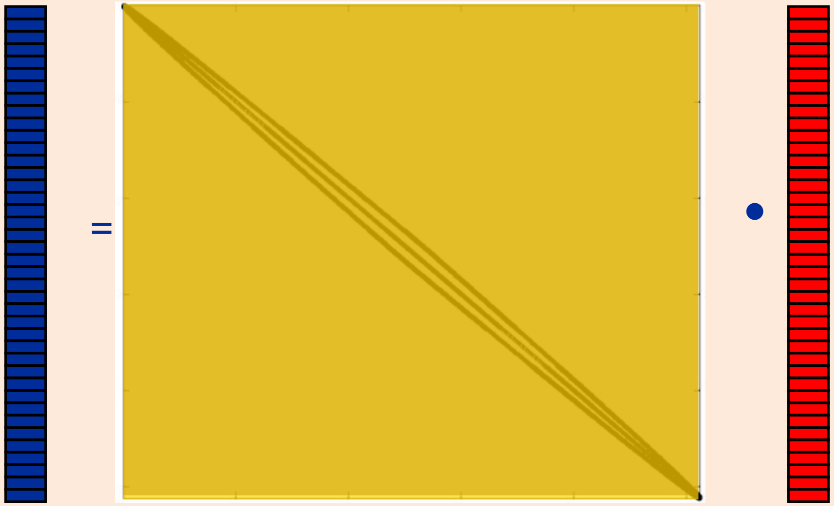
Matrix accessed 3 times from memory

RACE MPK



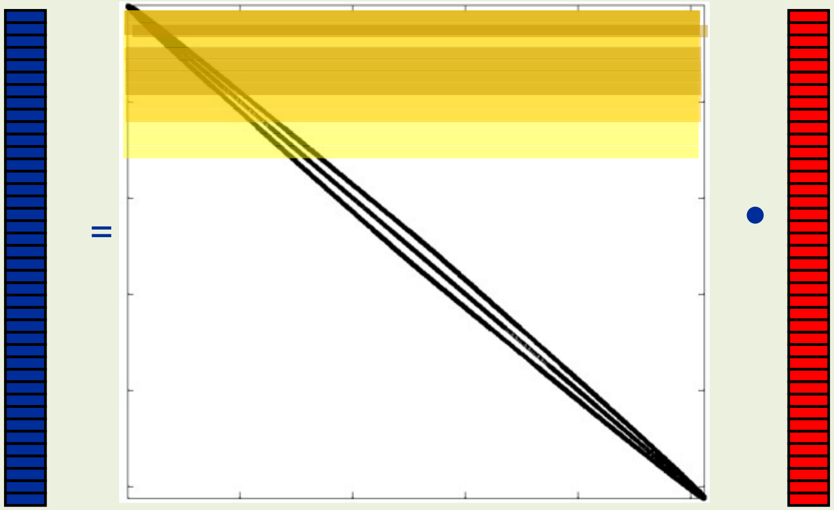
Matrix Power Kernel

“Back to back” SpMVs



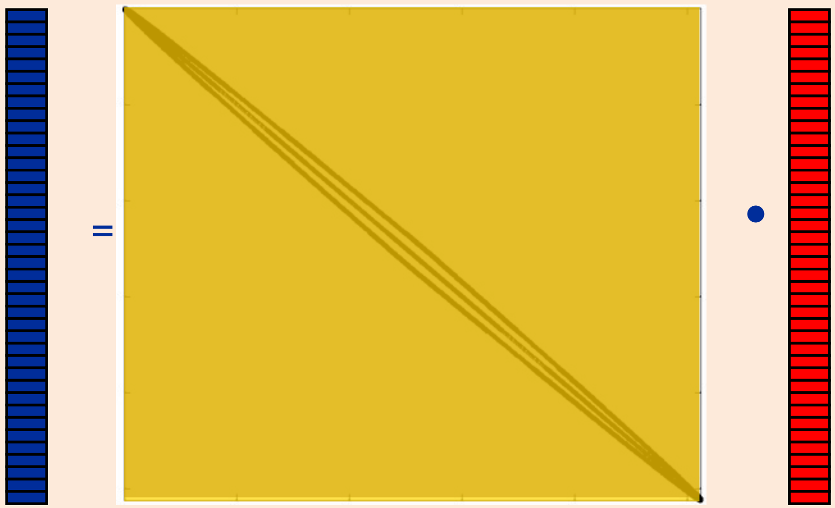
Matrix accessed 3 times from memory X

RACE MPK



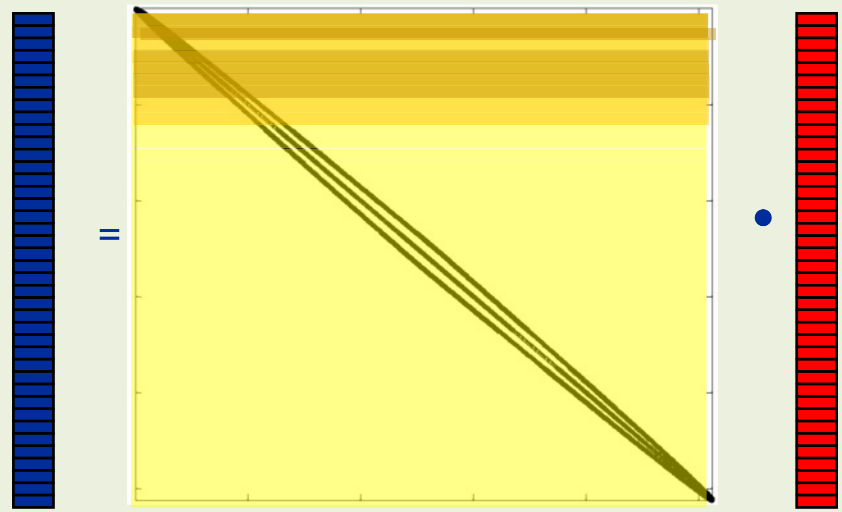
Matrix Power Kernel

"Back to back" SpMVs



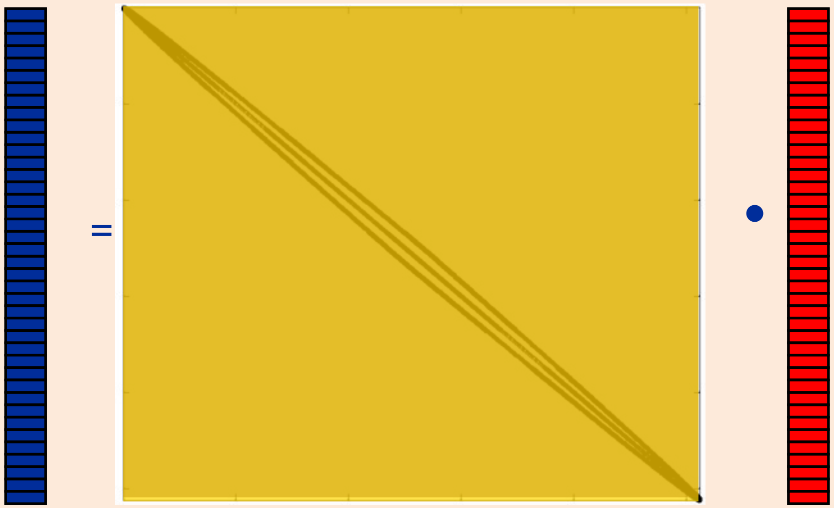
Matrix accessed 3 times from memory X

RACE MPK



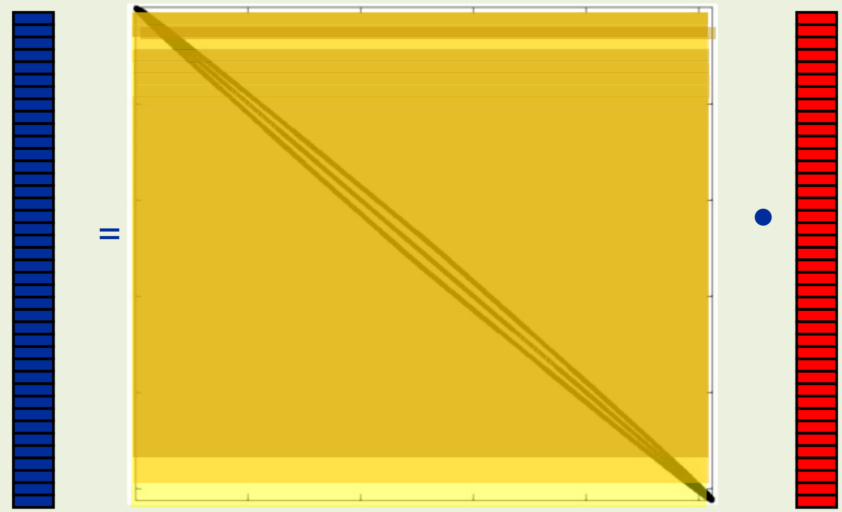
Matrix Power Kernel

“Back to back” SpMVs



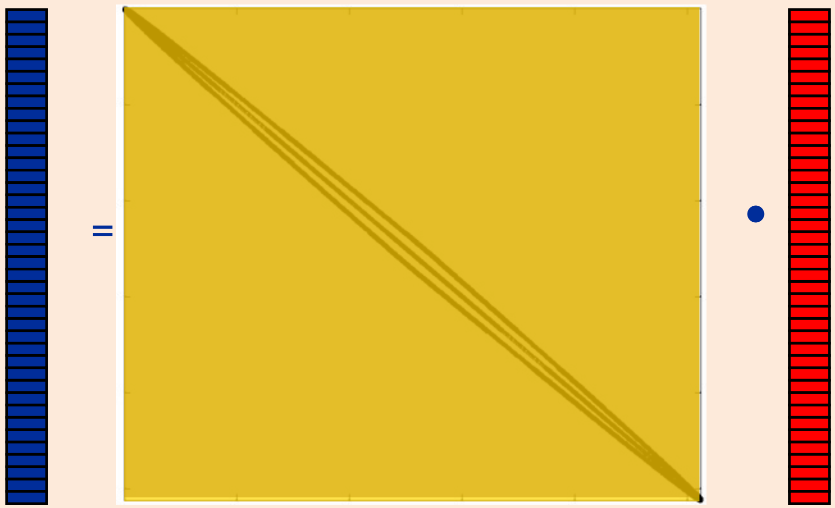
Matrix accessed 3 times from memory X

RACE MPK



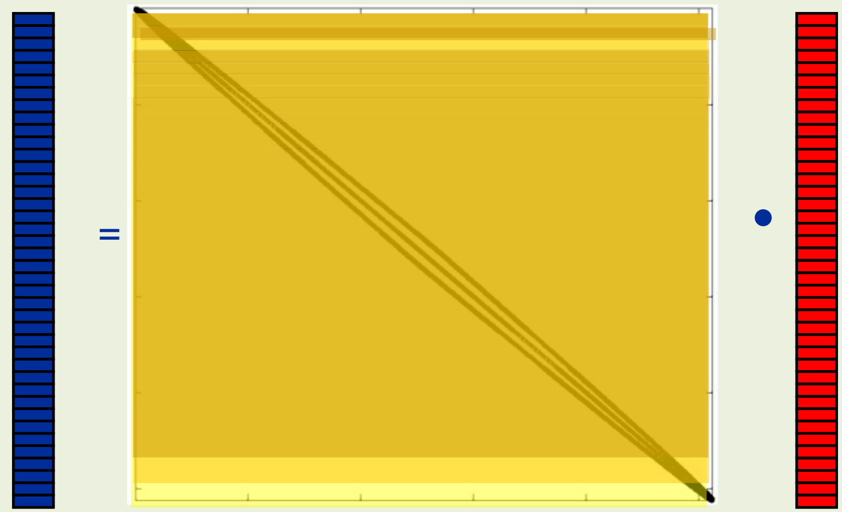
Matrix Power Kernel

“Back to back” SpMVs



Matrix accessed 3 times from memory X

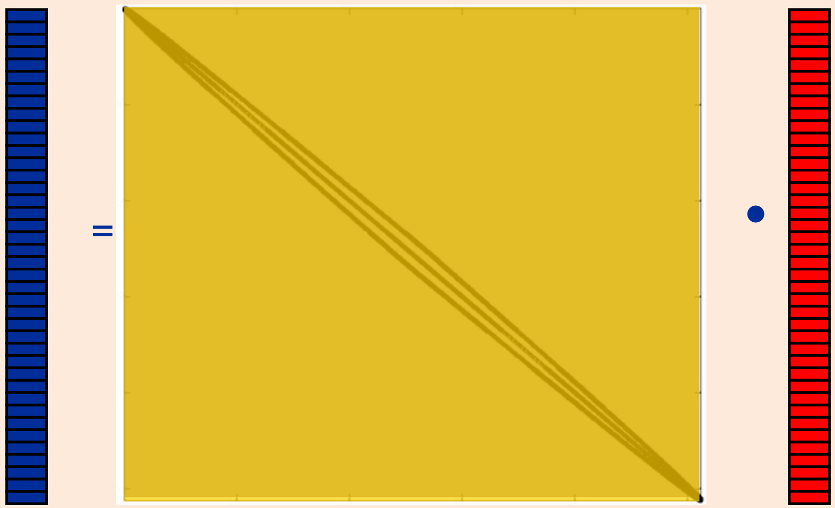
RACE MPK



Matrix accessed 1 time from memory

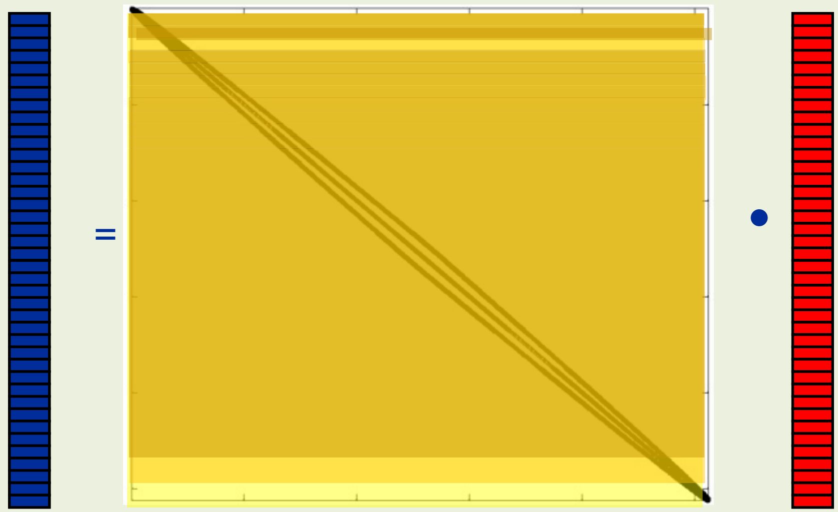
Matrix Power Kernel

"Back to back" SpMVs



Matrix accessed 3 times from memory X

RACE MPK



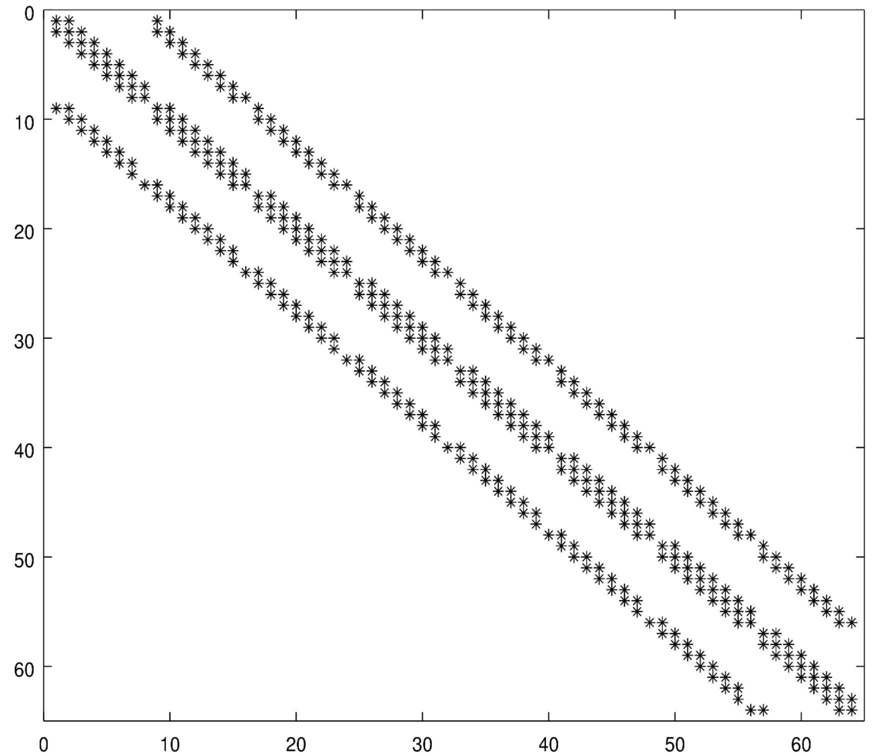
Matrix accessed 1 time from memory

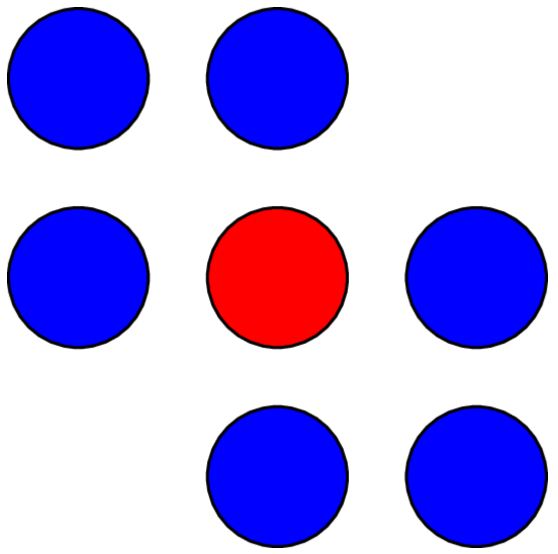
Alappat et al, "Level-Based Blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication," in *IEEE Transactions on Parallel and Distributed Systems*, 2023, doi: 10.1109/TPDS.2022.3223512.

Wavefront passing through matrix.

- More complicated than the Regular example seen here!

Sparse Matrix ✉ Graph





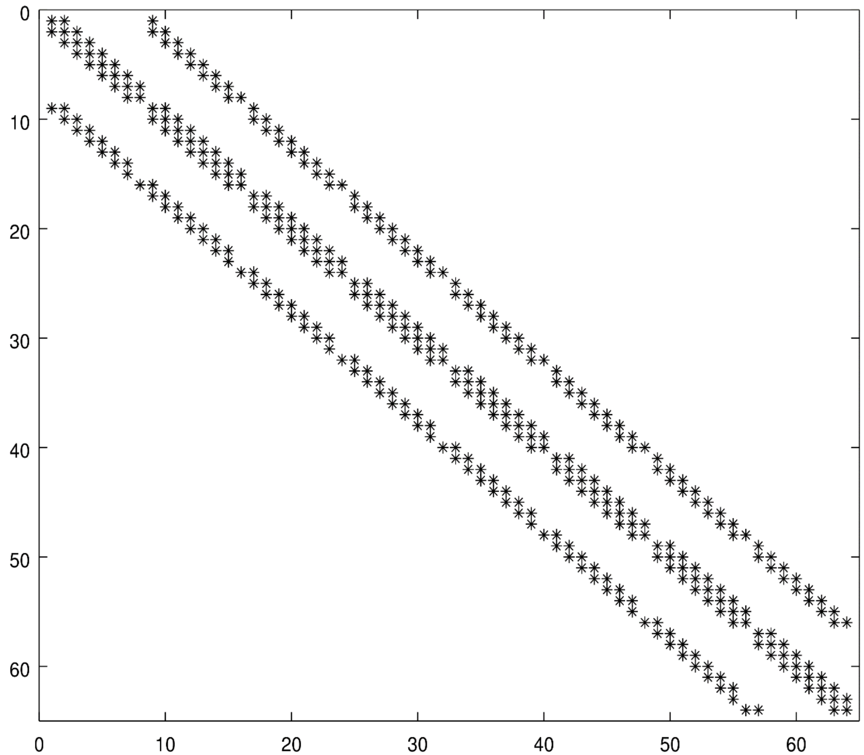
Stencil

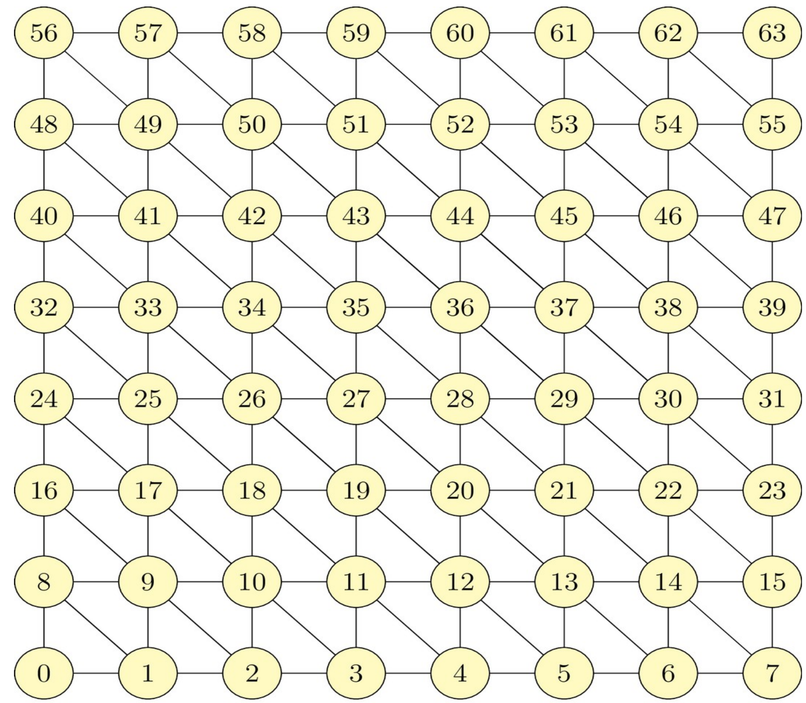
Sparse Matrix

Example of 2D-7 Point stencil

Sparse Matrix ✉ Graph

DELFT BLUE





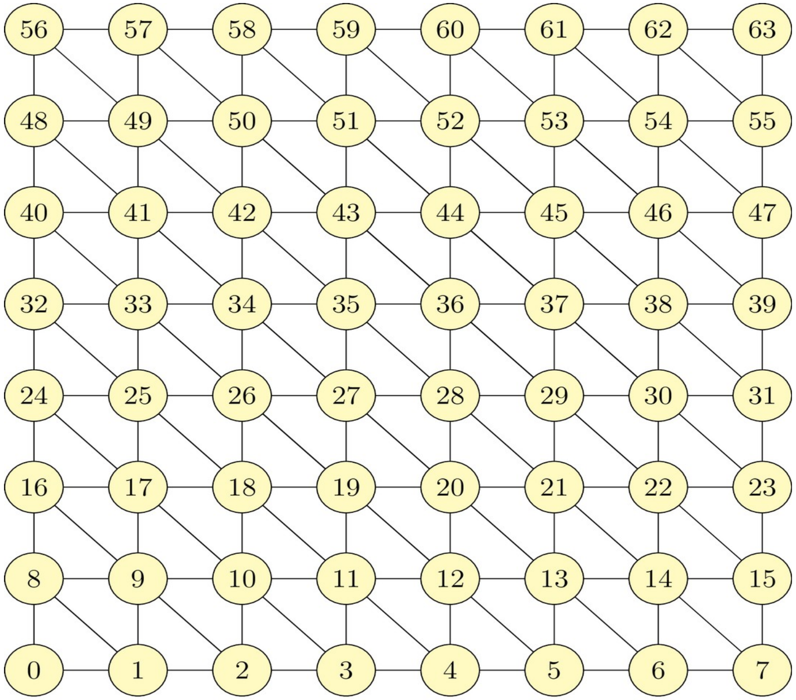
Sparse Matrix

Graph

Example of 2D-7 Point stencil

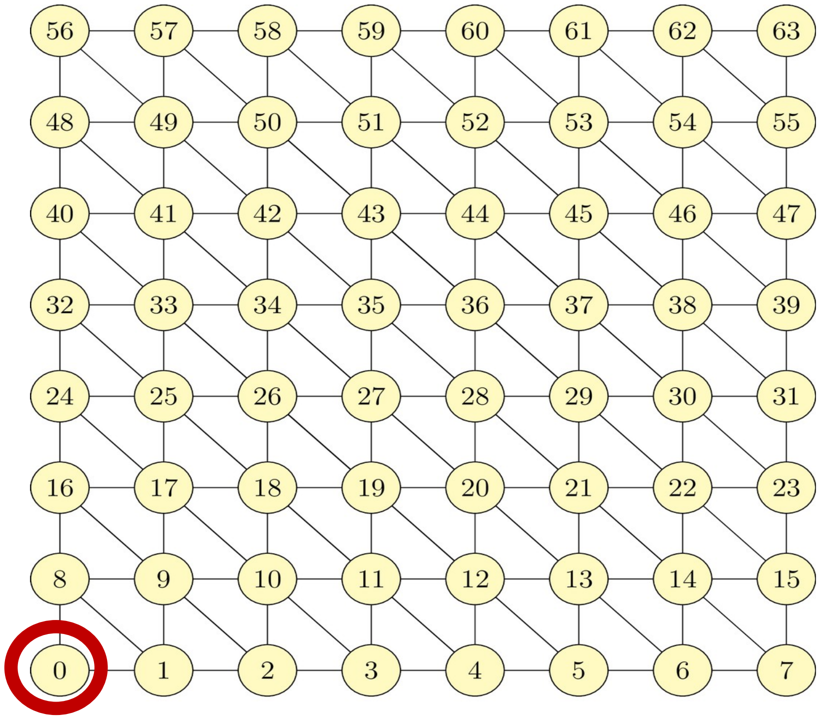
TU Delft

RACE: Recursive Algebraic Coloring



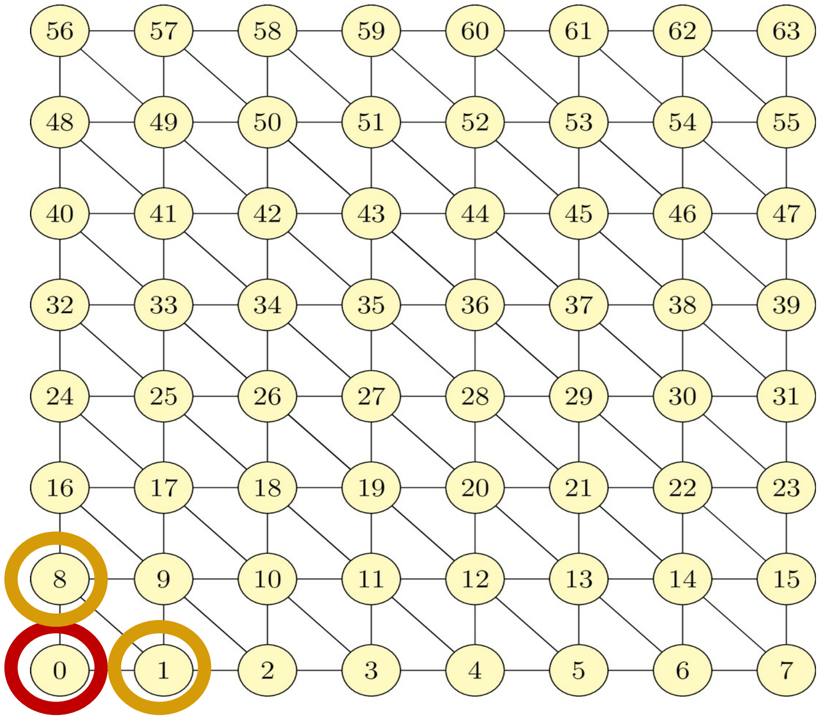
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



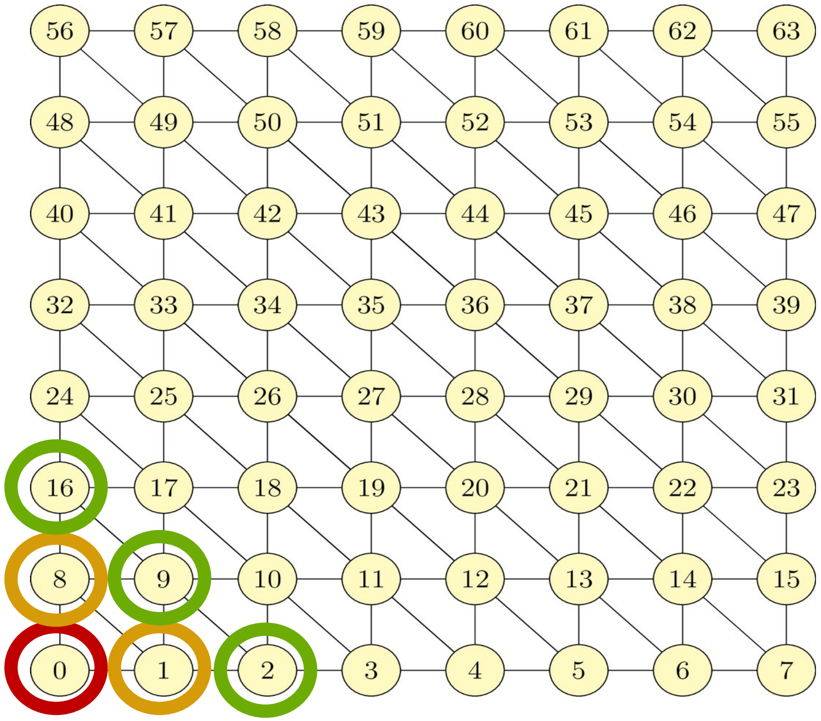
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



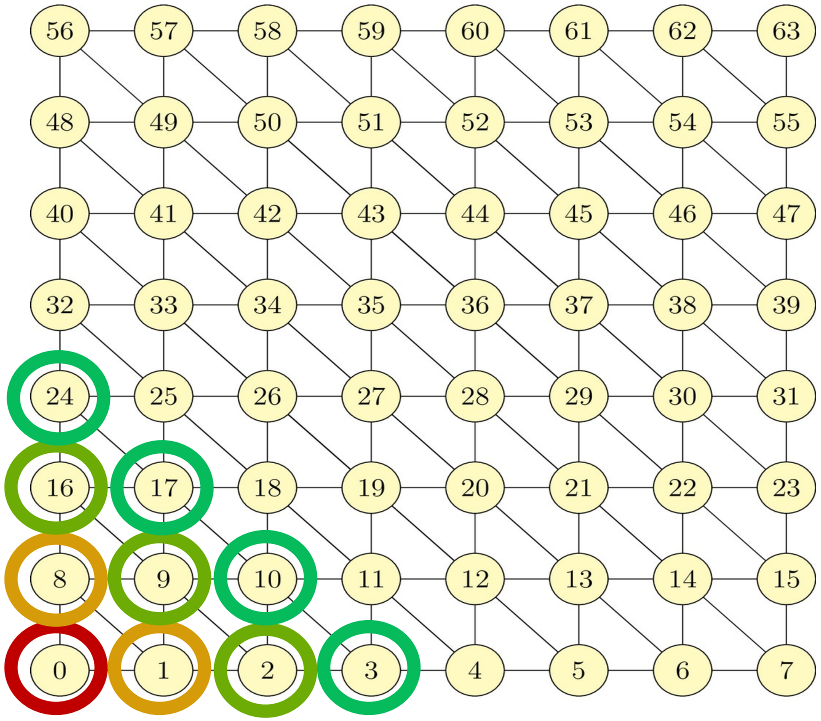
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



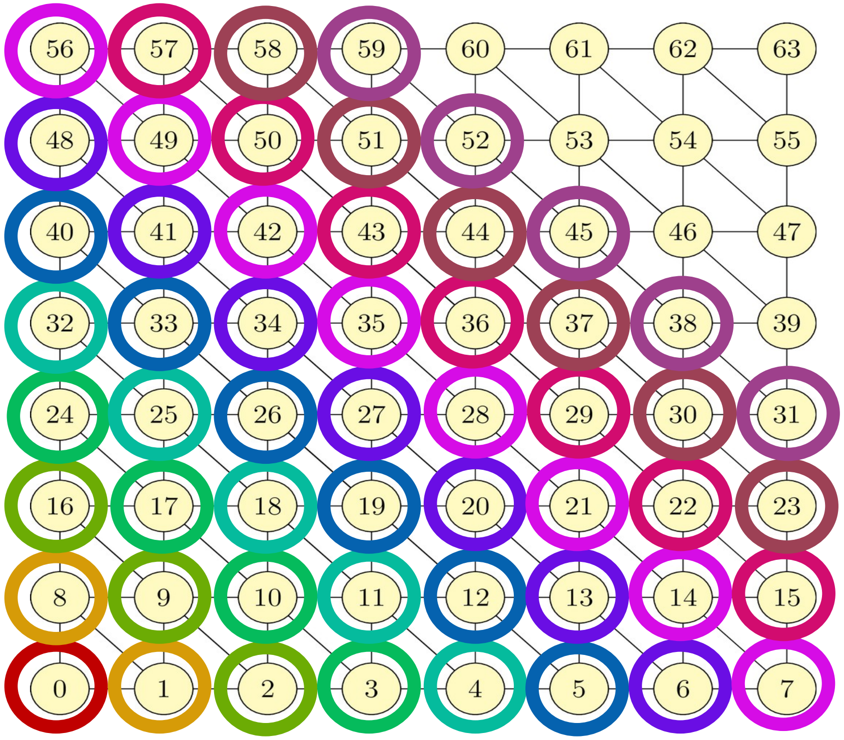
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



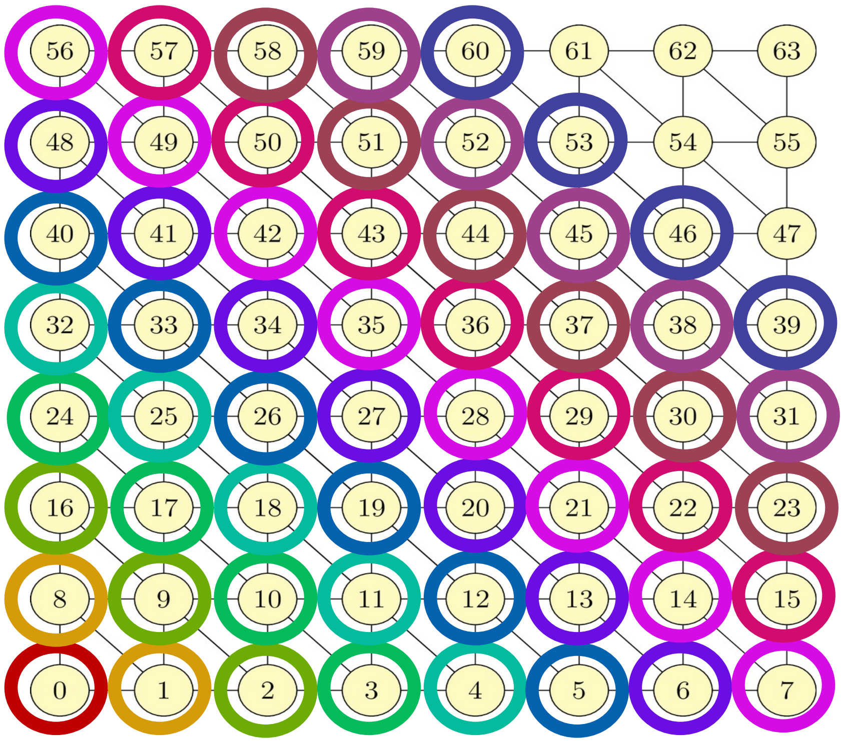
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



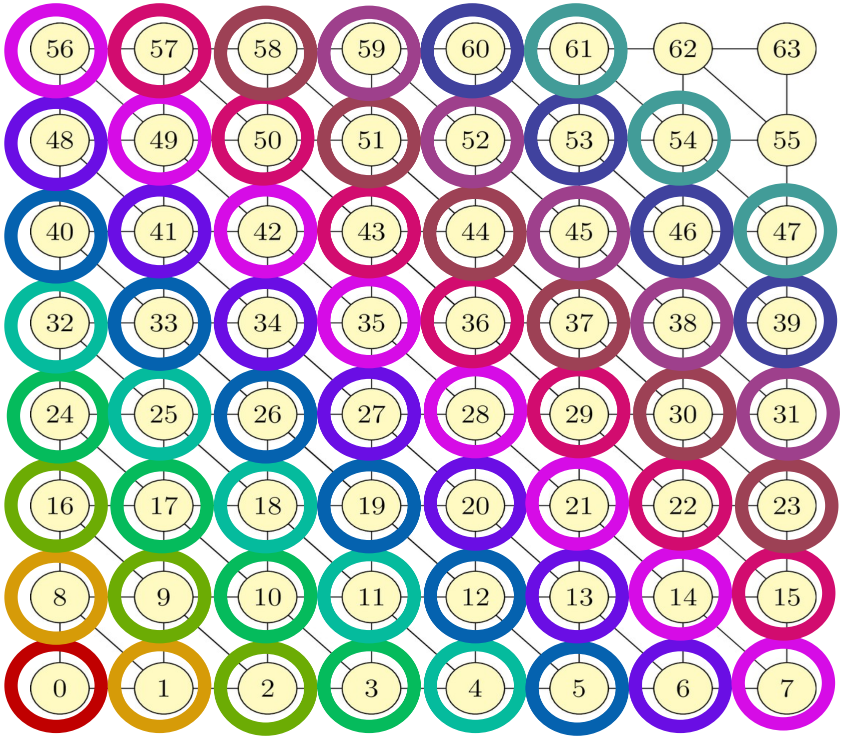
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



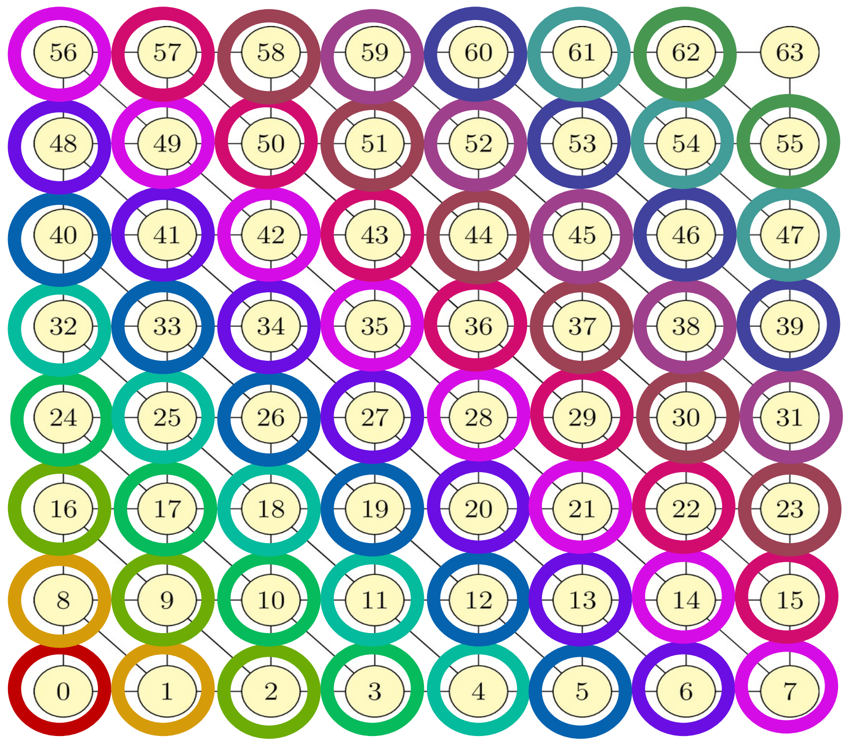
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



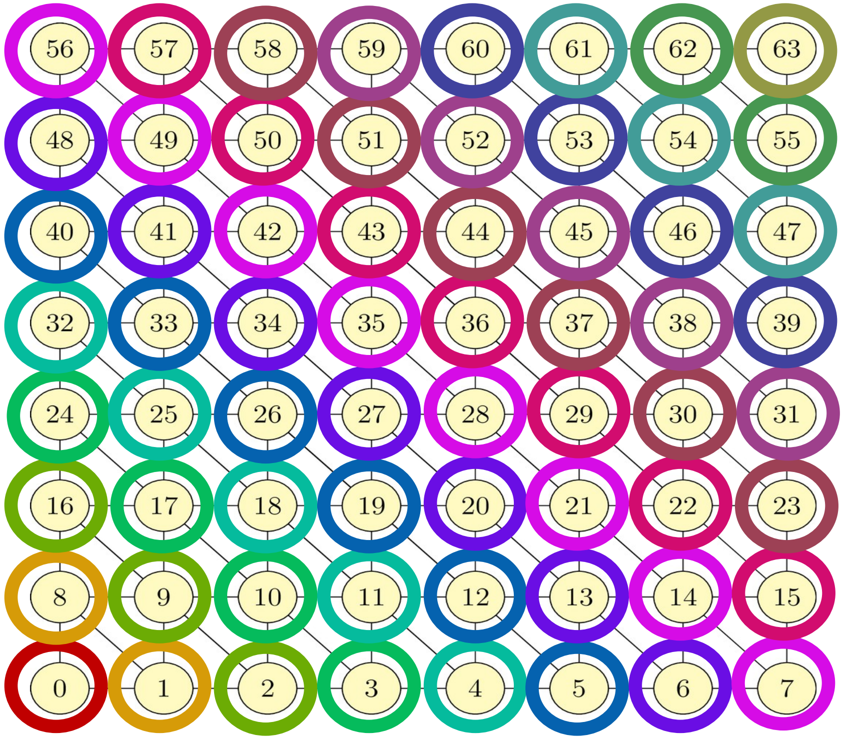
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



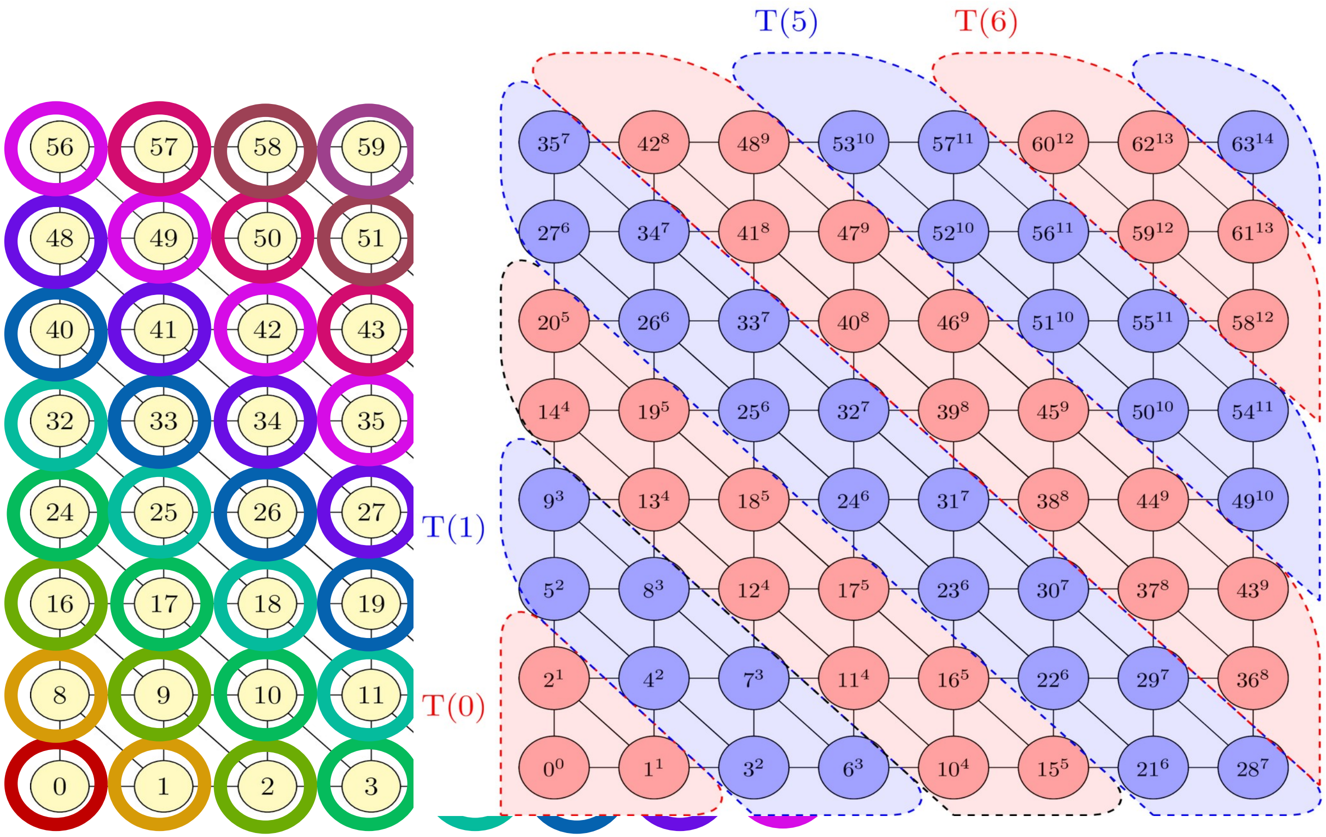
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



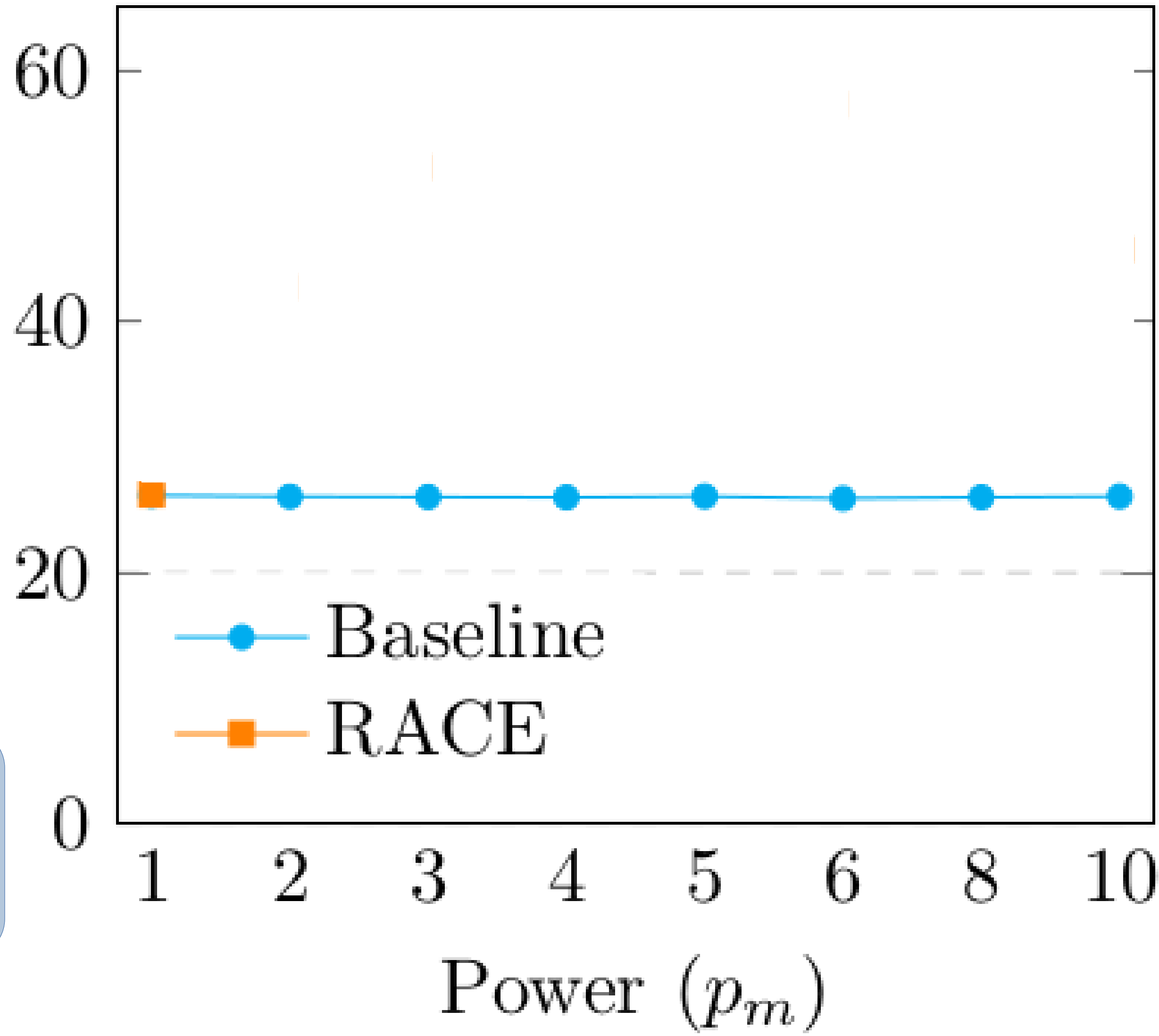
Example of 2D-7 Point stencil

RACE: Recursive Algebraic Coloring



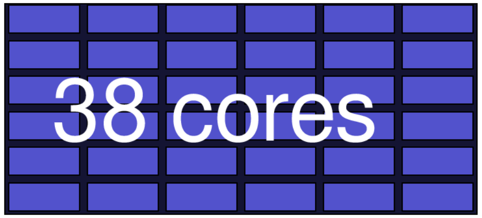
Example of 2D-7 Point stencil

RACE MPK Performance



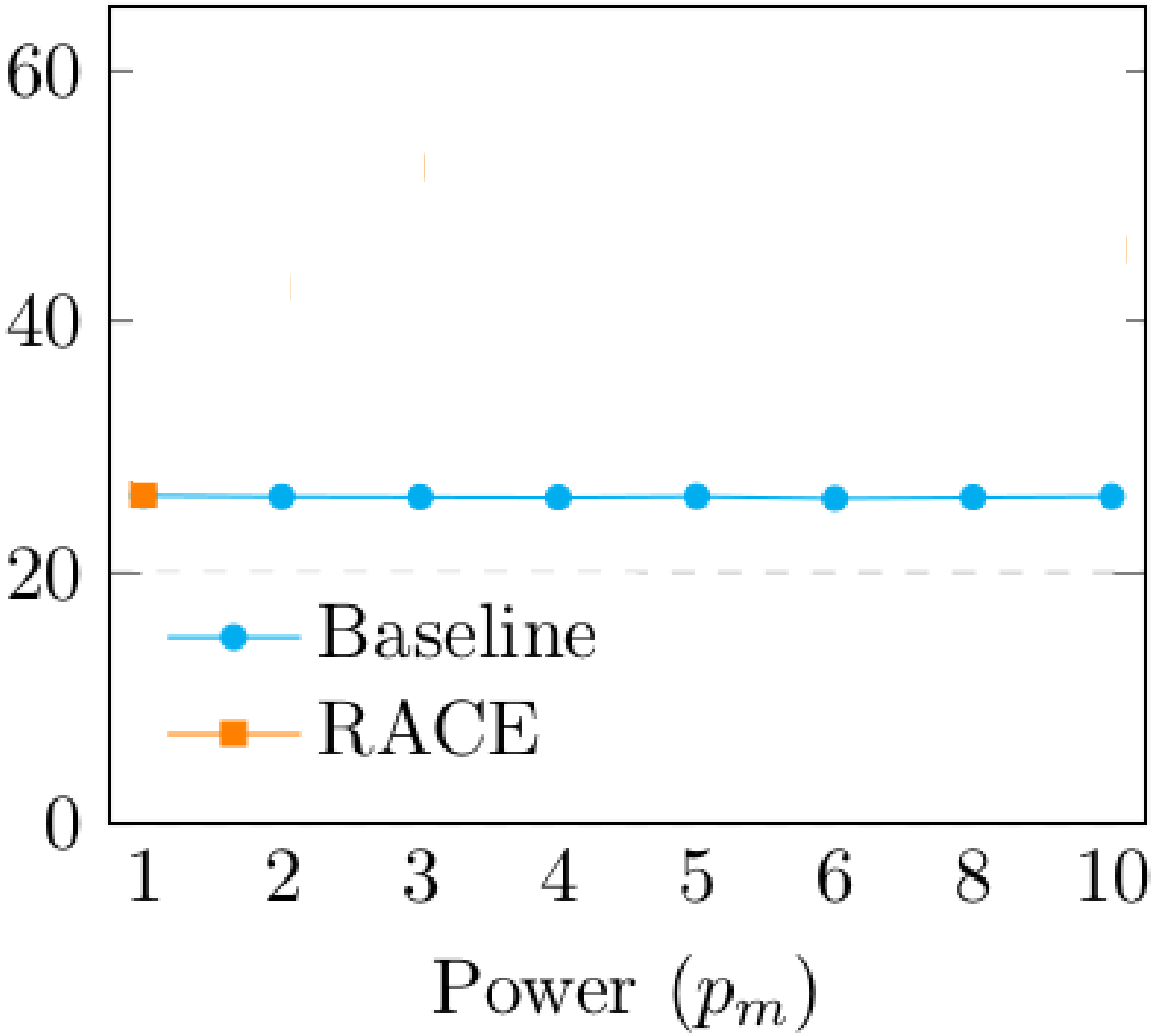
256 GB RAM
(146 GB/s)

RACE MPK Performance

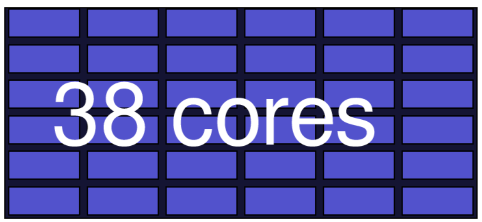


57MB L3 Cache
(420 GB/s)

256 GB RAM
(146 GB/s)

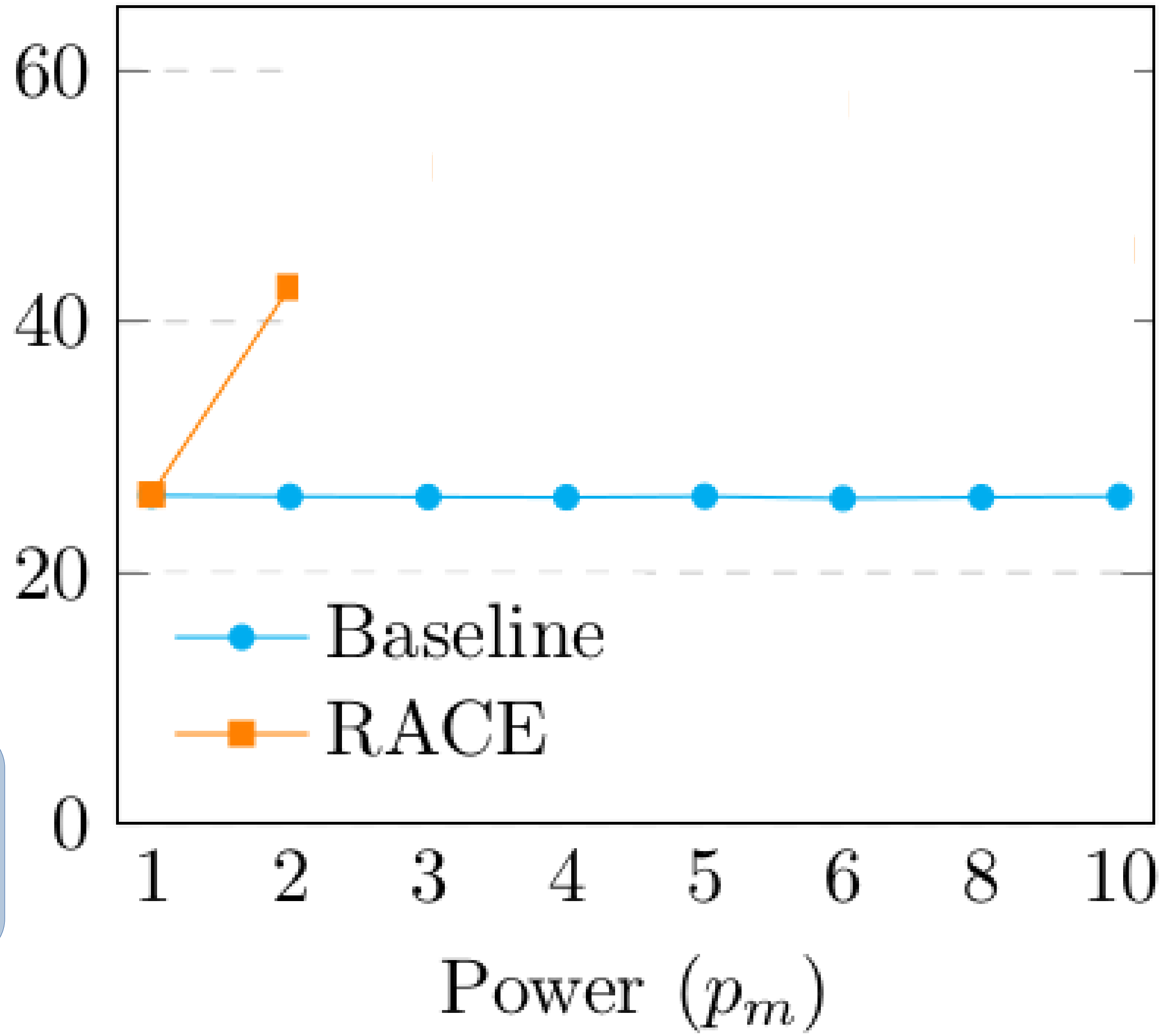


RACE MPK Performance

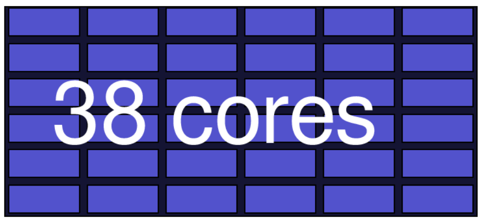


57MB L3 Cache (420 GB/s)

256 GB RAM
(146 GB/s)

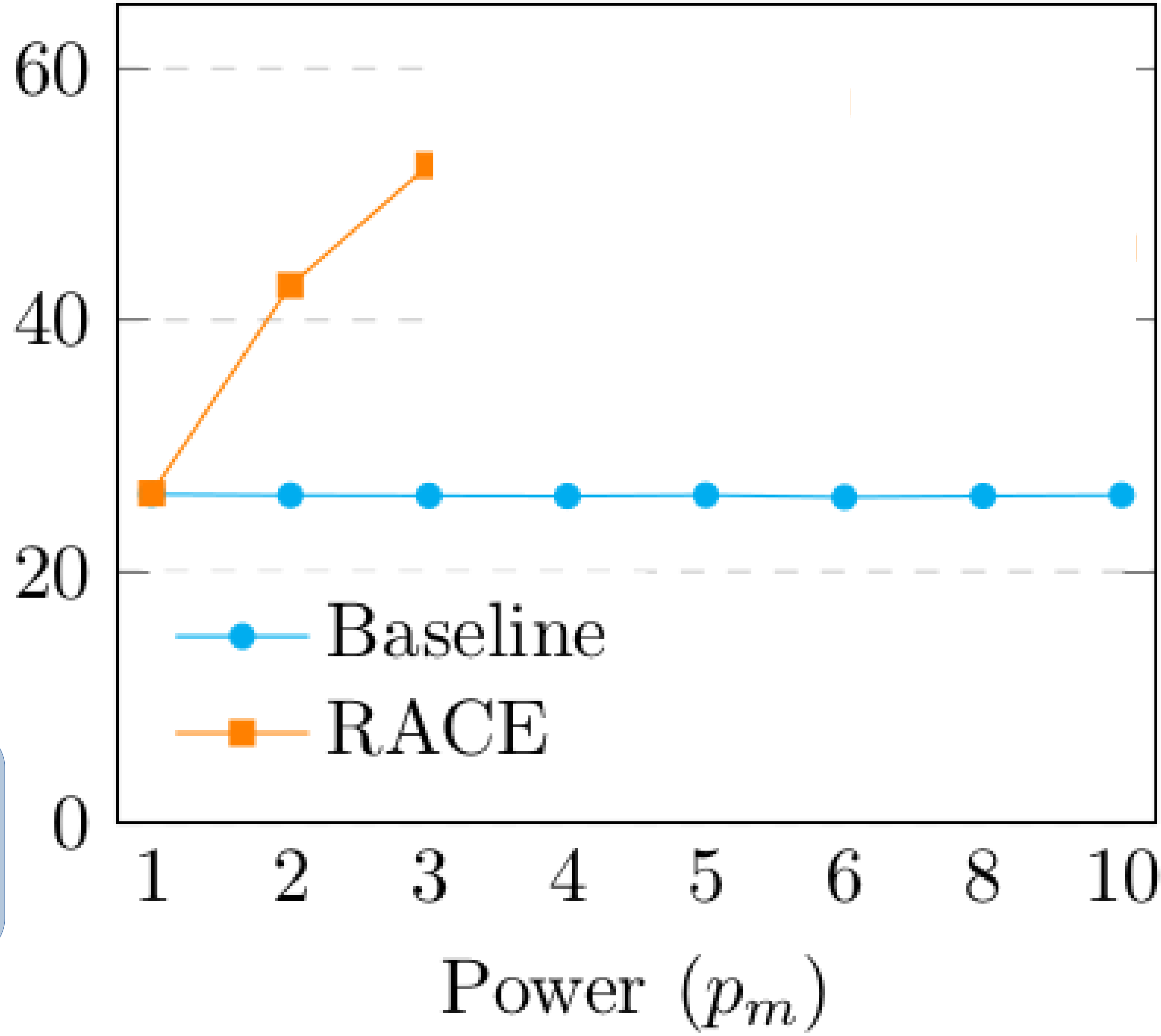


RACE MPK Performance

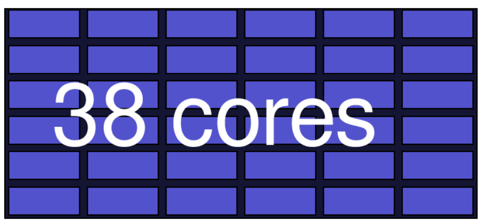


57MB L3 Cache
(420 GB/s)

256 GB RAM
(146 GB/s)

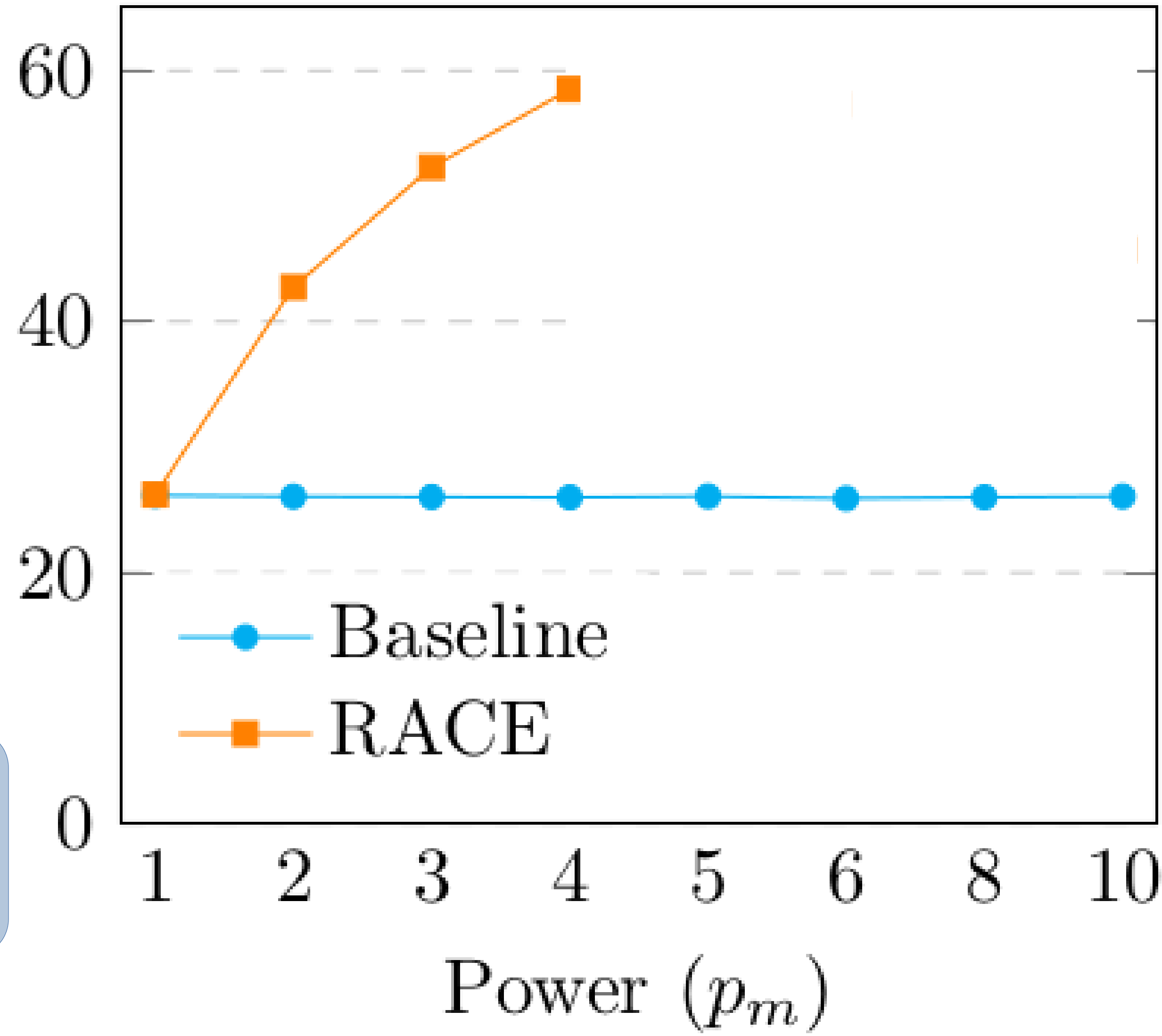


RACE MPK Performance

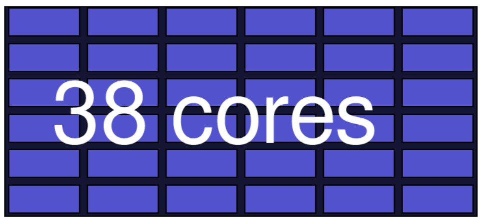


57MB L3 Cache
(420 GB/s)

256 GB RAM
(146 GB/s)

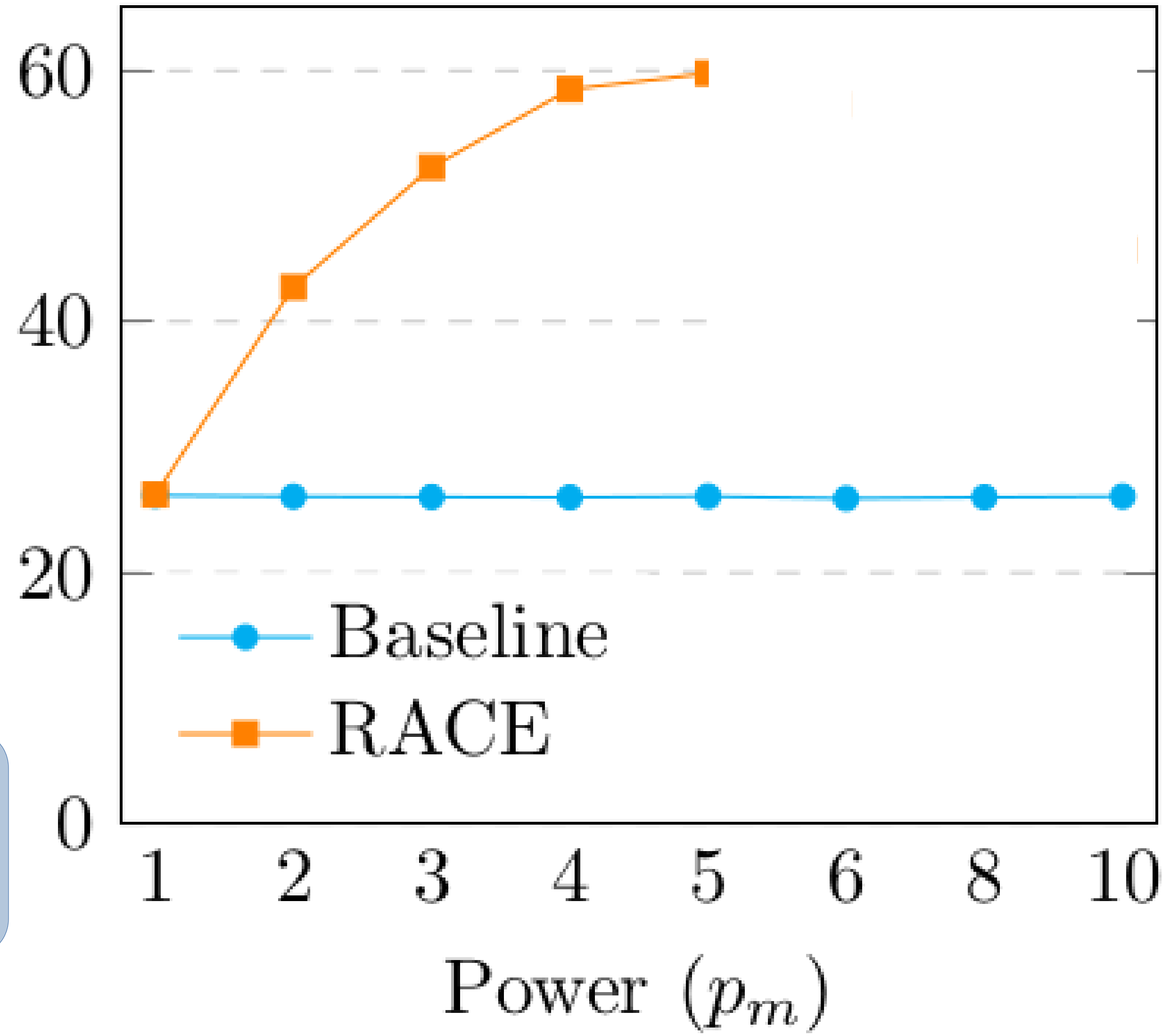


RACE MPK Performance



57MB L3 Cache
(420 GB/s)

256 GB RAM
(146 GB/s)

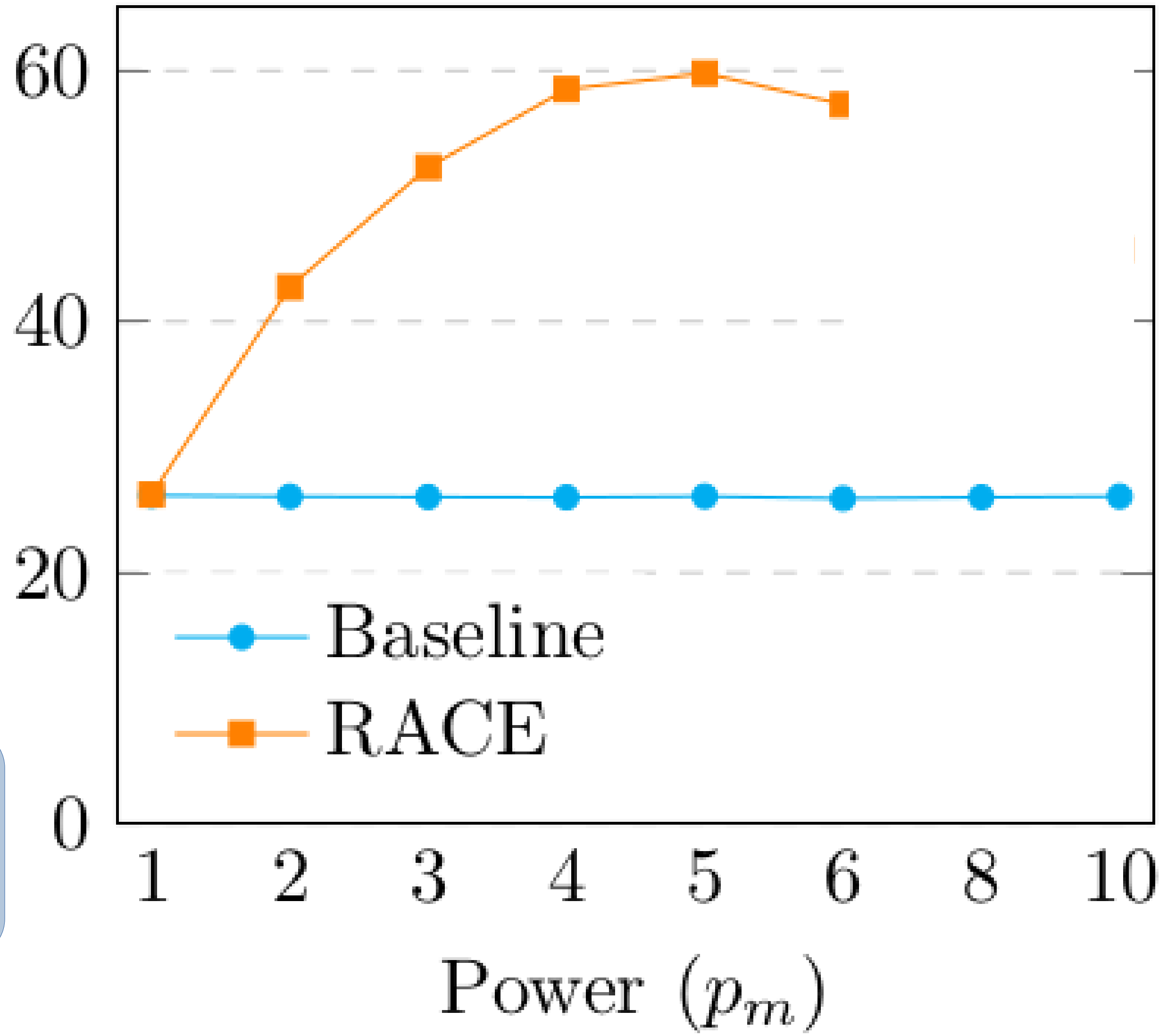


RACE MPK Performance

38 cores

57MB L3 Cache
(420 GB/s)

256 GB RAM
(146 GB/s)

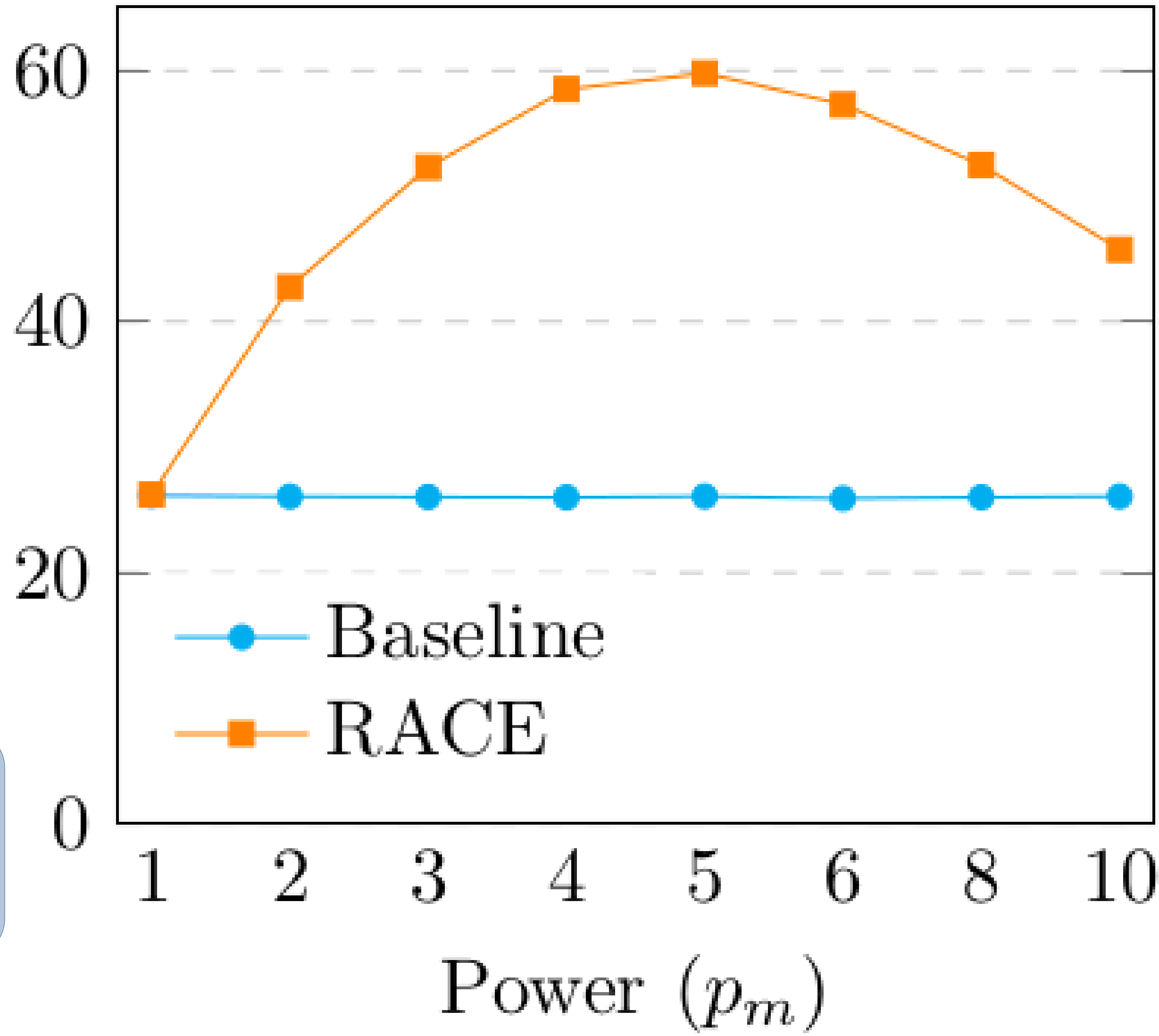


RACE MPK Performance

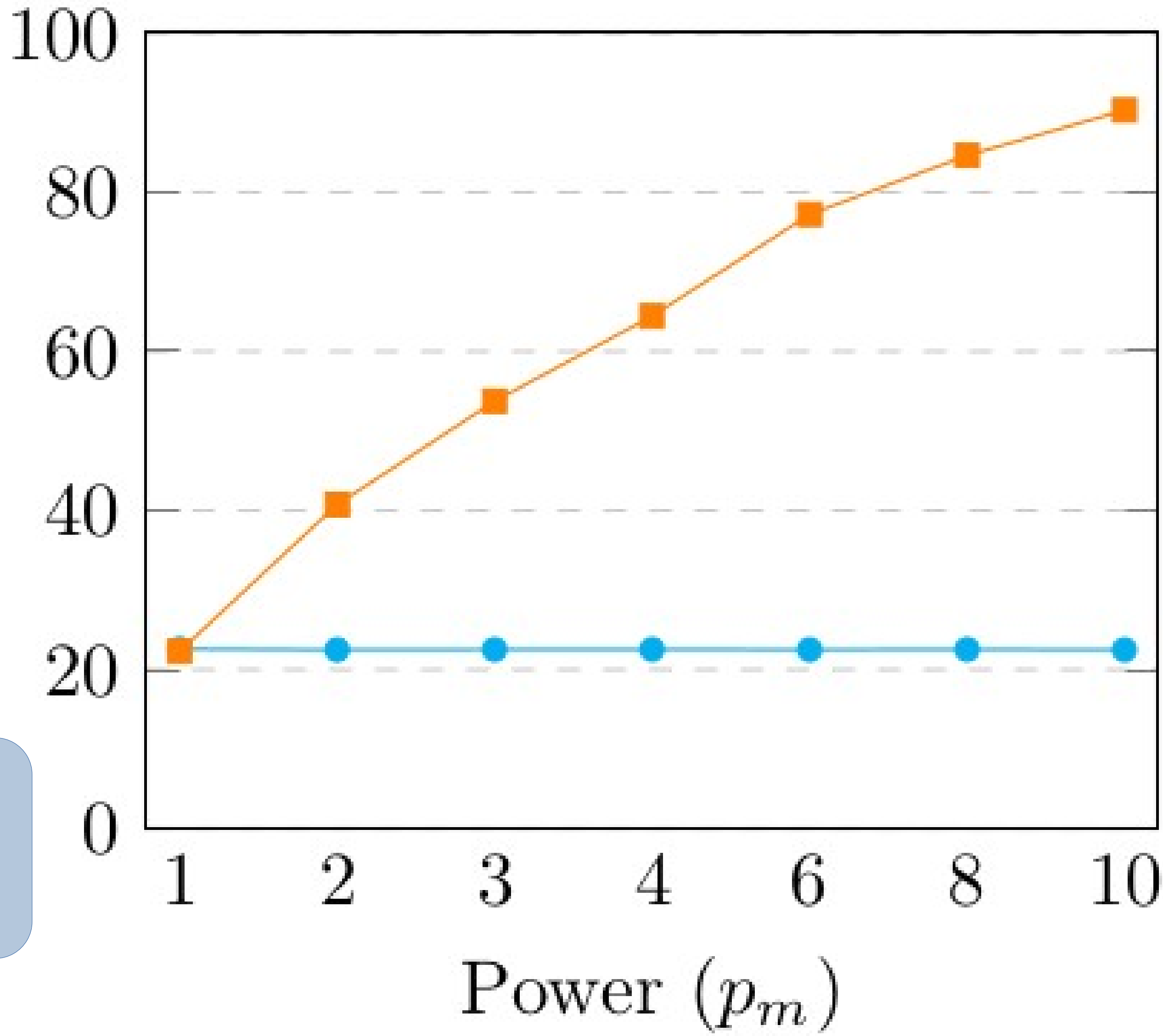
38 cores

57MB L3 Cache
(420 GB/s)

256 GB RAM
(146 GB/s)



RACE MPK Performance



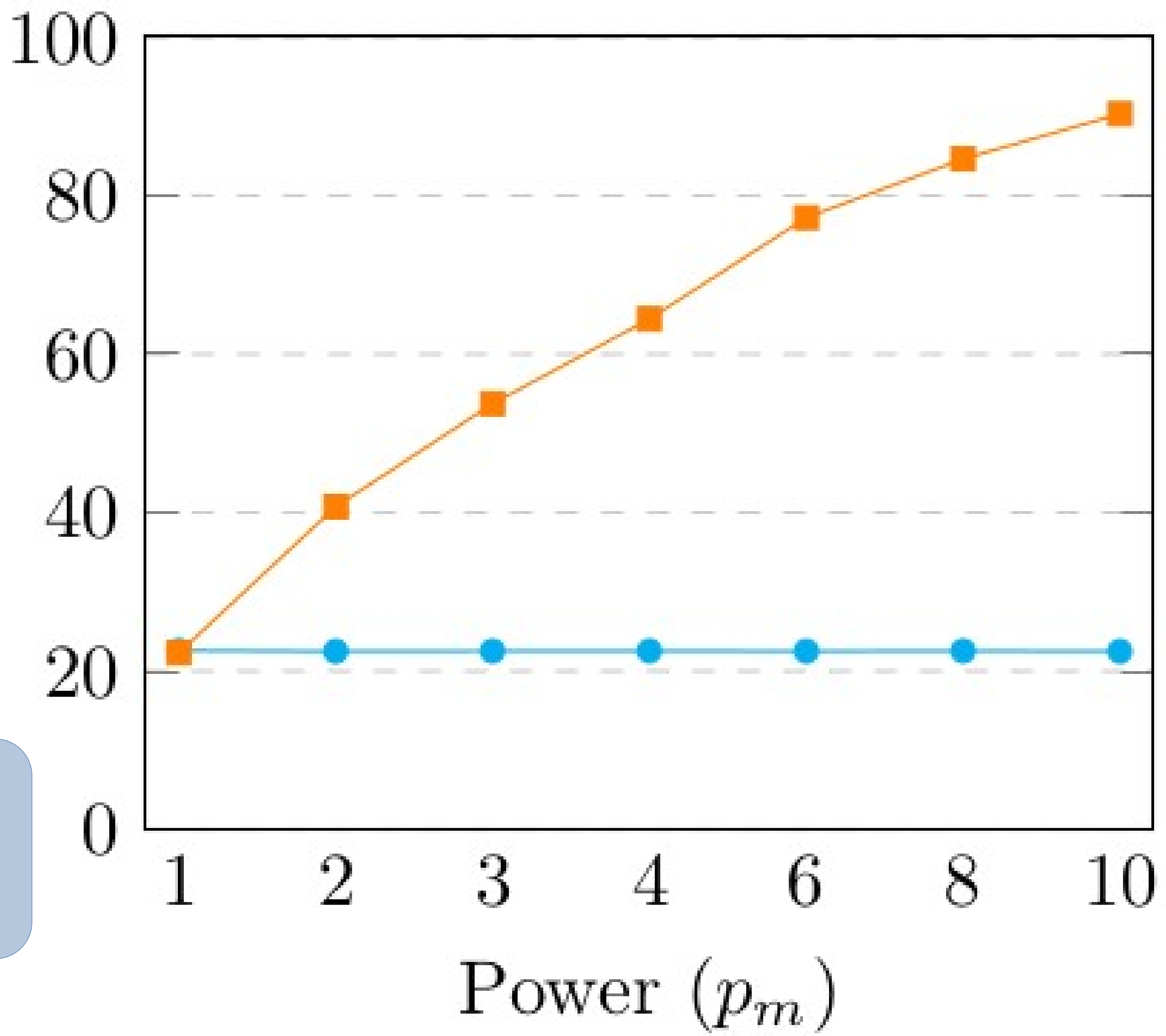
256 GB RAM
(146 GB/s)

RACE MPK Performance

64 cores

256 MB L3 Cache
(2600 G B/s)

256 GB RAM
(146 GB/s)



Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83
Jacobi	24	24	1.73	1.83

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83
Jacobi	24	24	1.73	1.83
GS	13	26	1.73	1.83

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83
Jacobi	24	24	1.73	1.83
GS	13	26	1.73	1.83
Poly(5)	6	30	1.76	2.24

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83
Jacobi	24	24	1.73	1.83
GS	13	26	1.73	1.83
Poly(5)	6	30	1.76	2.24
RACE + Poly(5)	6	30	1.16	1.64

DELFT BLUE

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83
Jacobi	24	24	1.73	1.83
GS	13	26	1.73	1.83
Poly(5)	6	30	1.76	2.24
RACE + Poly(5)	6	30	1.16	1.64
Jacobi(5)	5	25	1.29	1.40

TU Delft

Example: Nalu-Wind Model

Preconditioner	Iter.	#effective SpMVs	Solve time (s)	Time (s)
None	43	43	3.72	3.83
Jacobi	24	24	1.73	1.83
GS	13	26	1.73	1.83
Poly(5)	6	30	1.76	2.24
RACE + Poly(5)	6	30	1.16	1.64
Jacobi(5)	5	25	1.29	1.40
RACE + Jacobi(5)	5	25	0.60	0.70

What else can we do with RACE?

What else can we do with RACE?

s-step
Krylov

What else can we do with RACE?

s-step Krylov

Approx. Triangu-lar solves

What else can we do with RACE?

s-step Krylov

Approx. Triangu-lar solves

Fancier Polynomial Precon-ditioners

What else can we do with RACE?

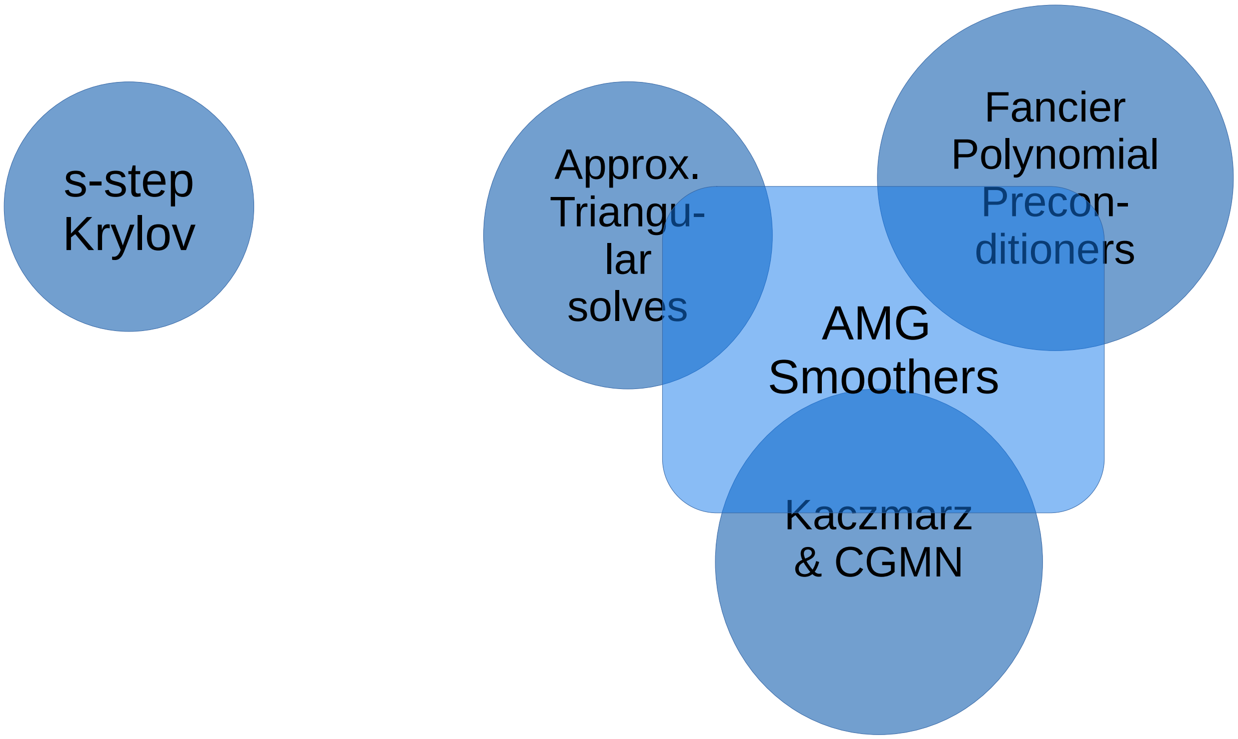
s-step Krylov

Approx. Triangu-lar solves

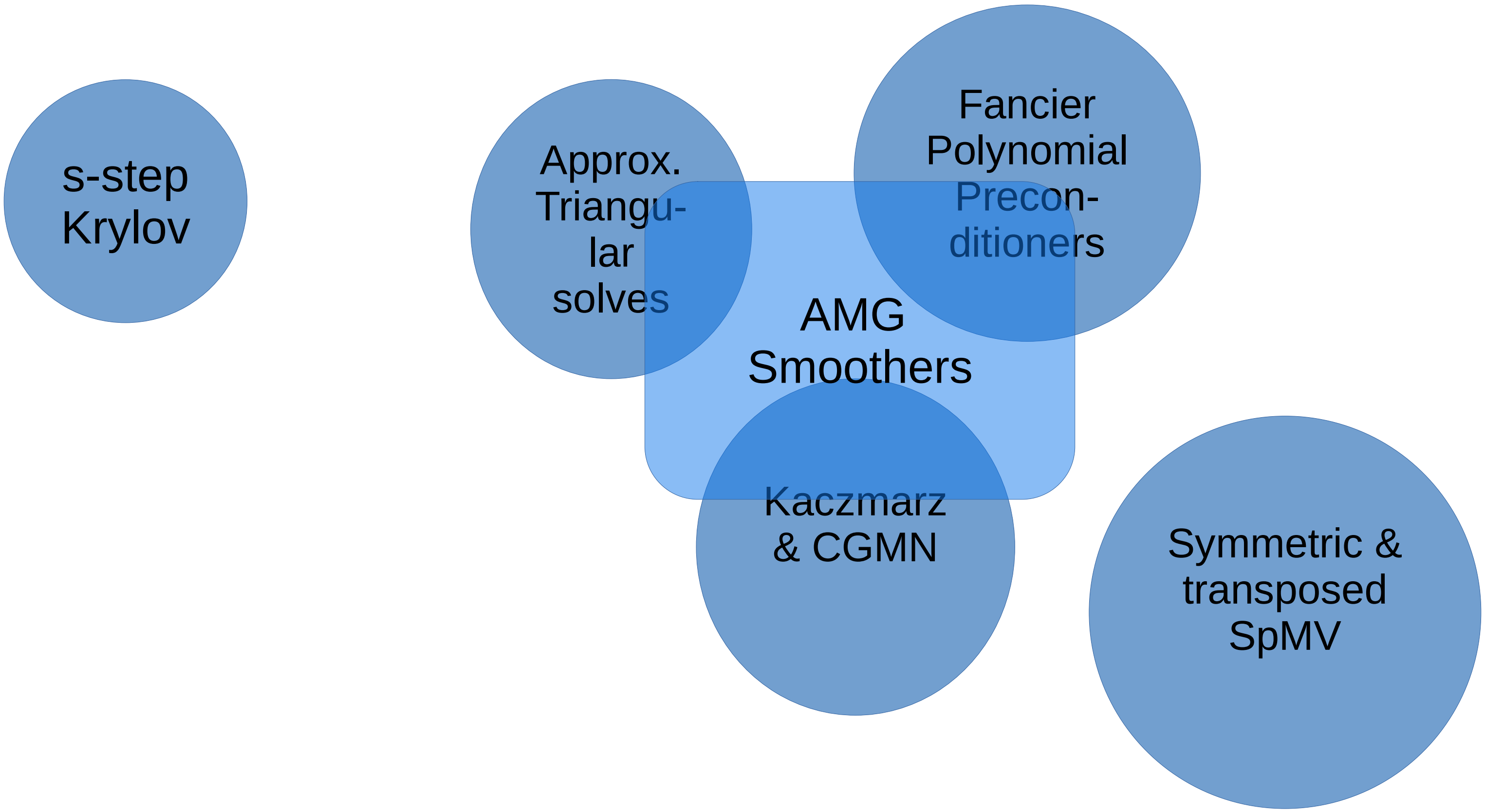
Fancier Polynomial Precon-ditioners

Kaczmarz & CGMN

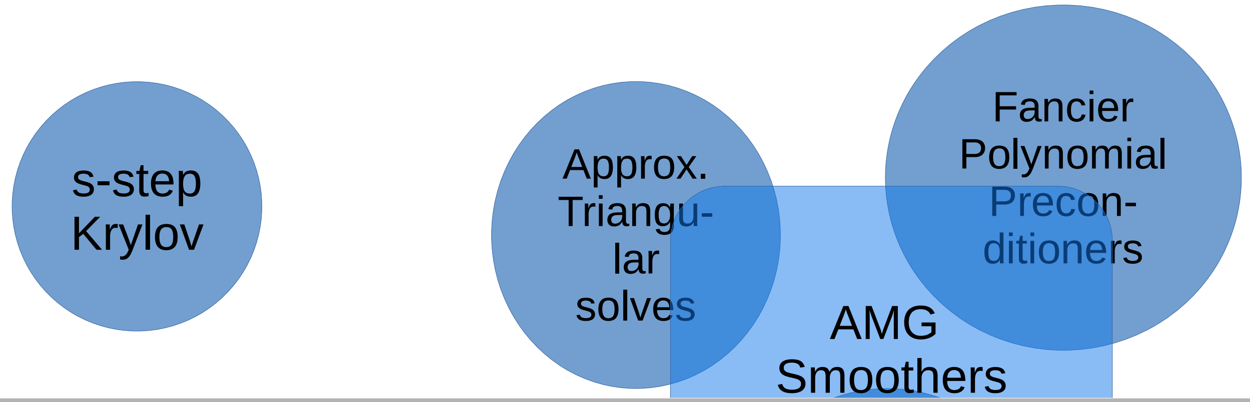
What else can we do with RACE?



What else can we do with RACE?



What else can we do with RACE?



Alappat et al.: A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication, ACM TOPC 7 (3), 2020

Alappat et al.: Level-based blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication. IEEE TPDS 34(2), 2023

Alappat et al.: Algebraic Temporal Blocking for Iterative Solvers on Multi-Core CPUs. To be submitted, will be on arXiv soon.

DCSE Summerschool next week:

Linear Algebra on High Performance Computers

Paolo Bientinesi
(Umeå University)

Gerhard Wellein
(U. Erlangen)

Laura Grigori
(EPFL)

Few places available:
https://www.aanmelder.nl/143287