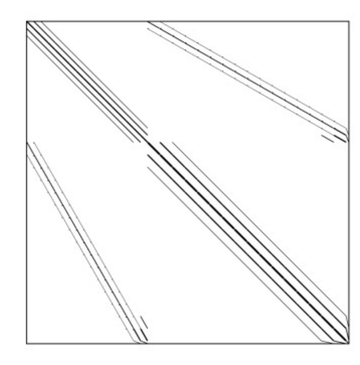
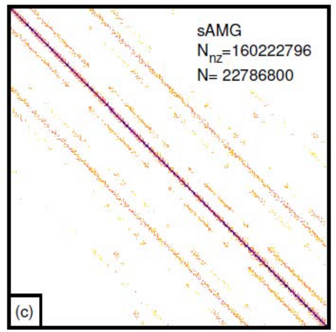
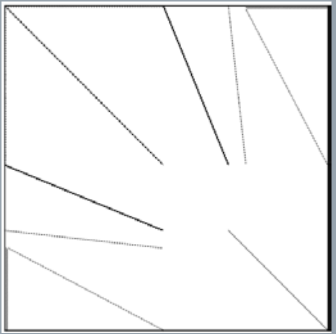
Advanced Matrix-Vector Multiplication – improving Code Balance → RACE

$y = A\,x$; $A^t=A$ → symmetric SpMV

$y = A^p\,x$ → Matrix Power Kernel

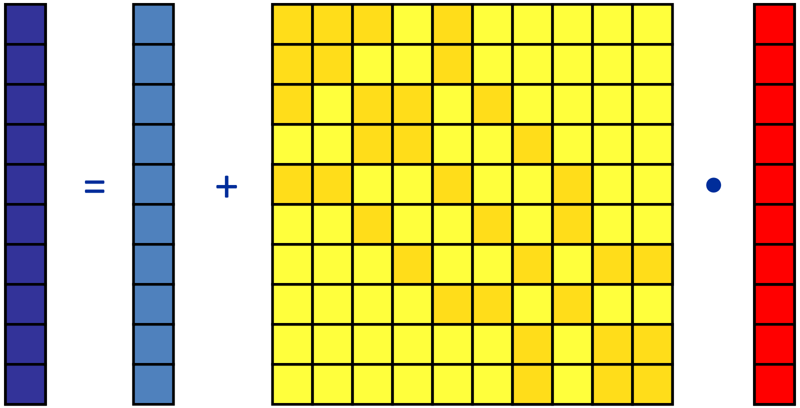






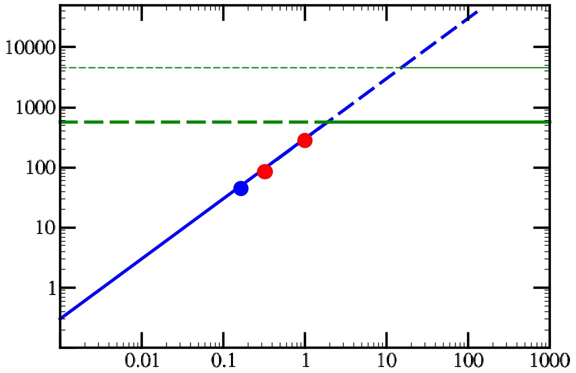
Starting Ground

```
do i = 1, N_r
 do j = row_ptr(i), row_ptr(i+1) - 1
  C(i) = C(i) + val(j) * B(col_idx(j))
 enddo
enddo
```



$$B_{C,min} = \frac{12 + 20/N_{nzr} + 8/N_{nzc}}{2} \frac{B}{F}$$

$$B_C\,(\alpha) = \frac{12 + 20/N_{nzr} + \mathbf{8\,\alpha}}{2} \frac{B}{F}$$

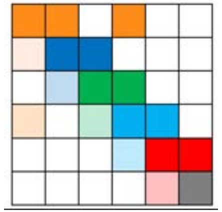


Move upwards in the RLM

Reduce data traffic (B_C) / increase I !

Agenda

- Basic ideas to improve Code Balance for SpMV based algorithms
- RACE: A different approach to Sparse Matrix-Vector Multiplication (SpMV)
- Parallelization of Symmetric SpMV



- Cache blocking for Matrix Power Kernels

$$y = A^p x$$

- Conclusion

!WARNING! This talk considers single multicore processors only:
Cascade Lake, Ice Lake, Rome (20c – 64c)

Motivation – Sparse Matrix Vector Multiplication

- Improve code balance of **SpMV-algorithms**

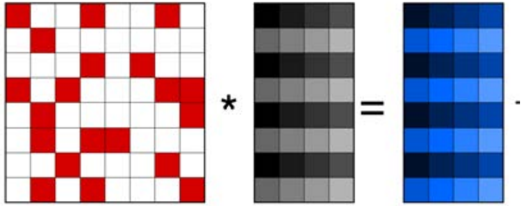
  - Kernel fusion, e.g. HPCG:

Algorithm 2. HPCG

while $k \leq iter$ & $r_{norm}/r_0 > tol$ **do**
  $z = MG(A{,}r)$ $\quad$ −> MG sweep
  $oldrtz = rtz$
  $rtz = \langle r{,}z \rangle$ $\quad$ −> DOT
  $\beta = rtz/oldrtz$
  $p = \beta * p + z$ $\quad$ −> WAXPBY
  $\boxed{Ap = A * p \quad \text{−> SpMPV}}$
  $\boxed{pAp = \langle p{,}Ap \rangle \quad \text{−> DOT}}$
  $\alpha = rtz/pAp$
  $x = x + \alpha * p$ $\quad$ −> WAXPBY
  $r = r - \alpha * Ap$ $\quad$ −> WAXPBY
  $r_{norm} = \langle r{,}r \rangle$ $\quad$ −> DOT
  $r_{norm} = sqrt(r_{norm})$
  $k{+}{+}$

  - Sparse Matrix Multiple Vector Multiplication

    Gropp et al.: Towards Realistic Performance Bounds for Implicit CFD Codes
    ParCFD 1999. https://wgropp.cs.illinois.edu/bib/papers/pdata/1999/pcfd99/gkks.ps



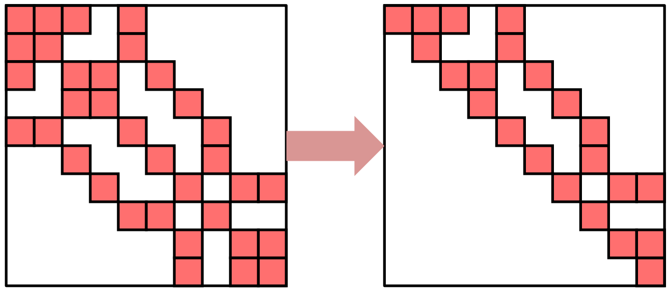
  - Both, e.g. Chebyshev Filter Computation for eigenvalue computations:

    Kreutzer et al.: Chebyshev Filter Diagonalization on Modern Manycore Processors and GPGPUs. ISC 2018.
    DOI: https://dx.doi.org/10.1007/978-3-319-92040-5_17

Motivation – Sparse Matrix Vector Multiplication

Further opportunities to improve code balance of SpMV-kernels:

- Exploit symmetry (SymmSpMV): $A = A^t$
  - Naive speed-up: max. 2



- Cache blocking for Sparse **Matrix Power Kernels** (**MPK**): $y = A^p x$
  - Data locality in „sparse matrix-matrix multiply“ ←→ Irregular sparsity pattern
  - Naive speed up: max. p

Symmetric SpMV – serial execution

- Store only upper (lower) triangular part of symmetric sparse matrix

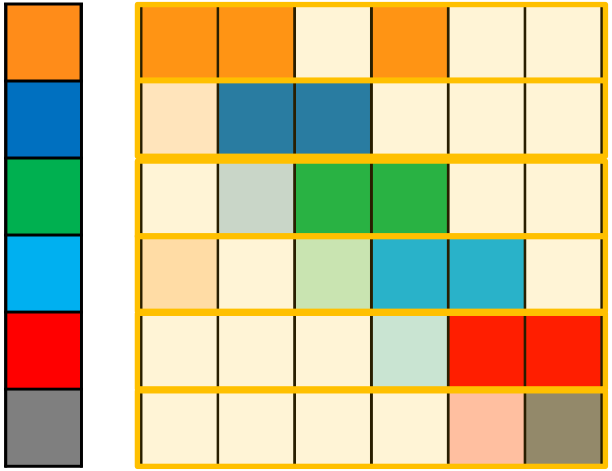
```
do i = 1, N_r
 do j = row_ptr(i), row_ptr(i+1) - 1
  y(i)            = y(i)+ val(j) * x(col_idx(j))
  y(col_idx(j)) = y(col_idx(j)) + val(j) * x(j)
 enddo
enddo
```

- Improve code balance by up to 2x

$$B_C = \frac{\mathbf{12} + 20/N_{nzr} + 8\,\alpha}{\mathbf{2}} \frac{B}{F}$$

☺

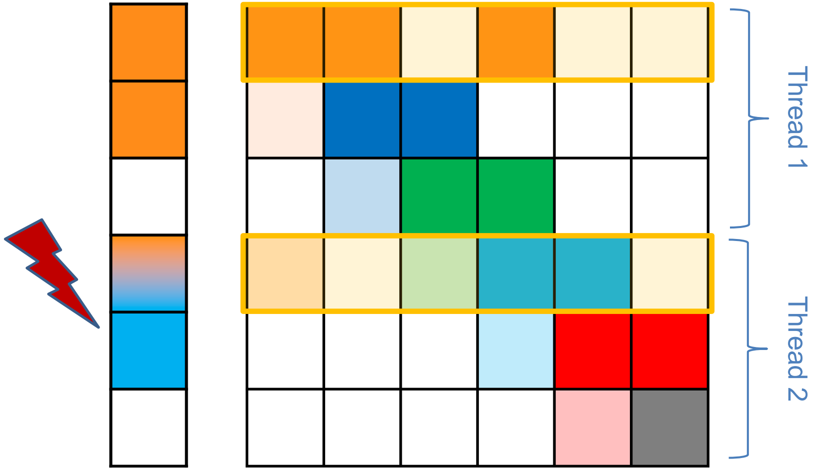
$$B_C^{sym} = \frac{\mathbf{12} + 4/N_{nzr}^{sym} + 24\,\alpha^{sym}}{\mathbf{4}} \frac{B}{F}$$



Symmetric SpMV – parallelisation: write conflicts

```
#pragma omp parallel for
do i = 1, Nr
 do j = row_ptr(i), row_ptr(i+1) - 1
  y(i)             = y(i)+ val(j) * x(col_idx(j))
  y(col_idx(j))   = y(col_idx(j)) + val(j) * x(j)
 enddo
enddo
```

☹



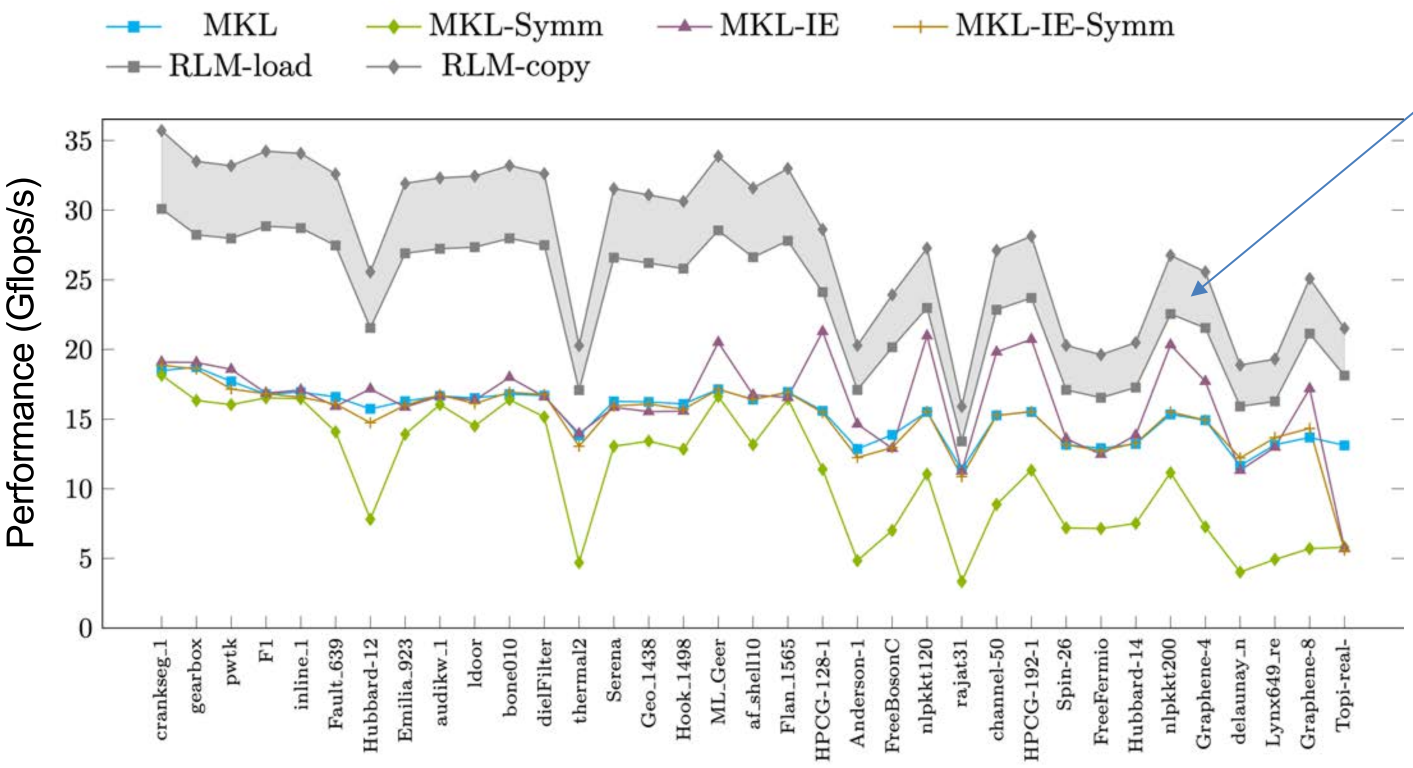
SymmetricSpMV – Intel MKL options

Intel math kernel library (MKL) supports SpMV and SymmSpMV:

- Plain SpMV calls using CRS
  - SpMV-call → MKL
  - SymmSpMV-call → MKL-Symm
- Inspector-Executor takes (1) CRS as input, (2) analyses matrix structure and (3) may optimize (e.g. data structure) and then (4) run the kernel
  - No additonal hints → MKL-IE
  - HINT=symmetric → MKL-IE-Symm

Intel MKL Symmetric SpMV performance

Performance of SymmSpMV on 1 socket of Intel Cascade Lake

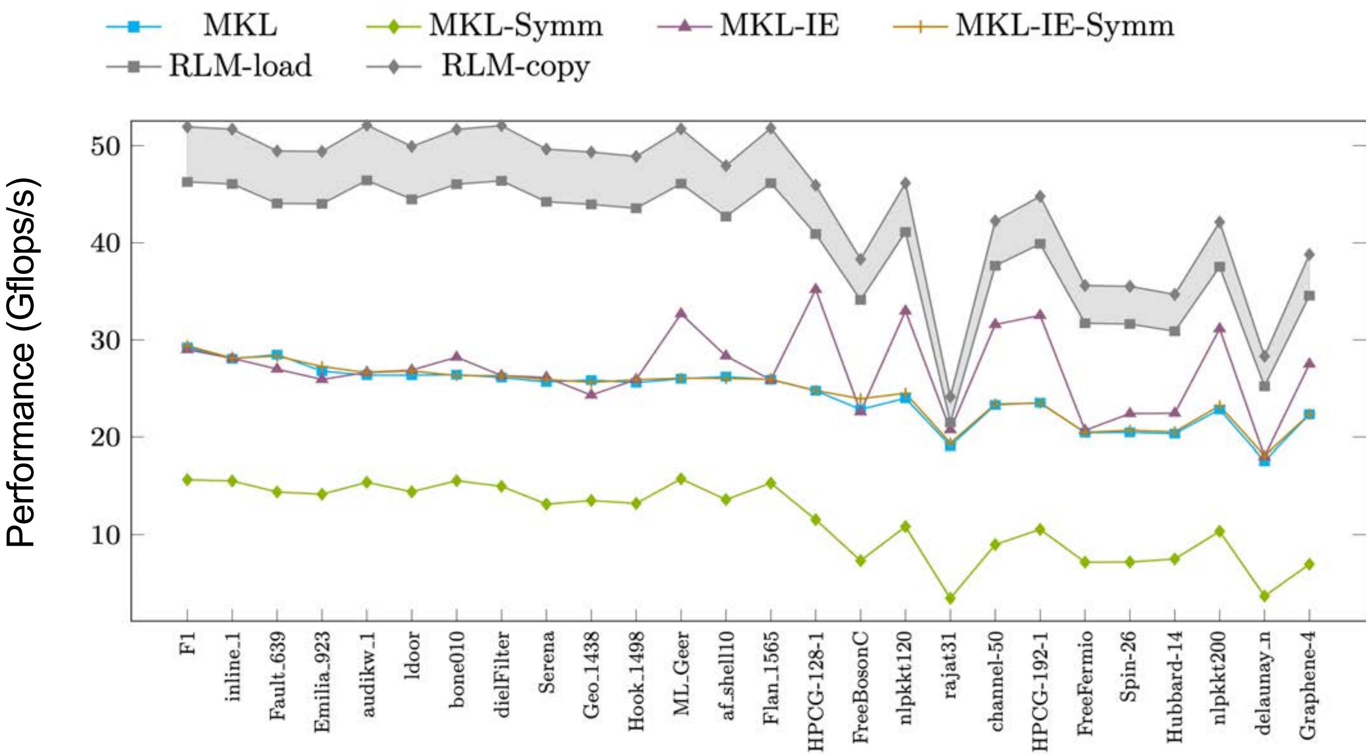


$P = b_S / B_{C,min}$
(load only and copy bandwidth)

Intel Xeon 6248
20 cores
Bandwidth 115 GB/s

Intel MKL Symmetric SpMV performance

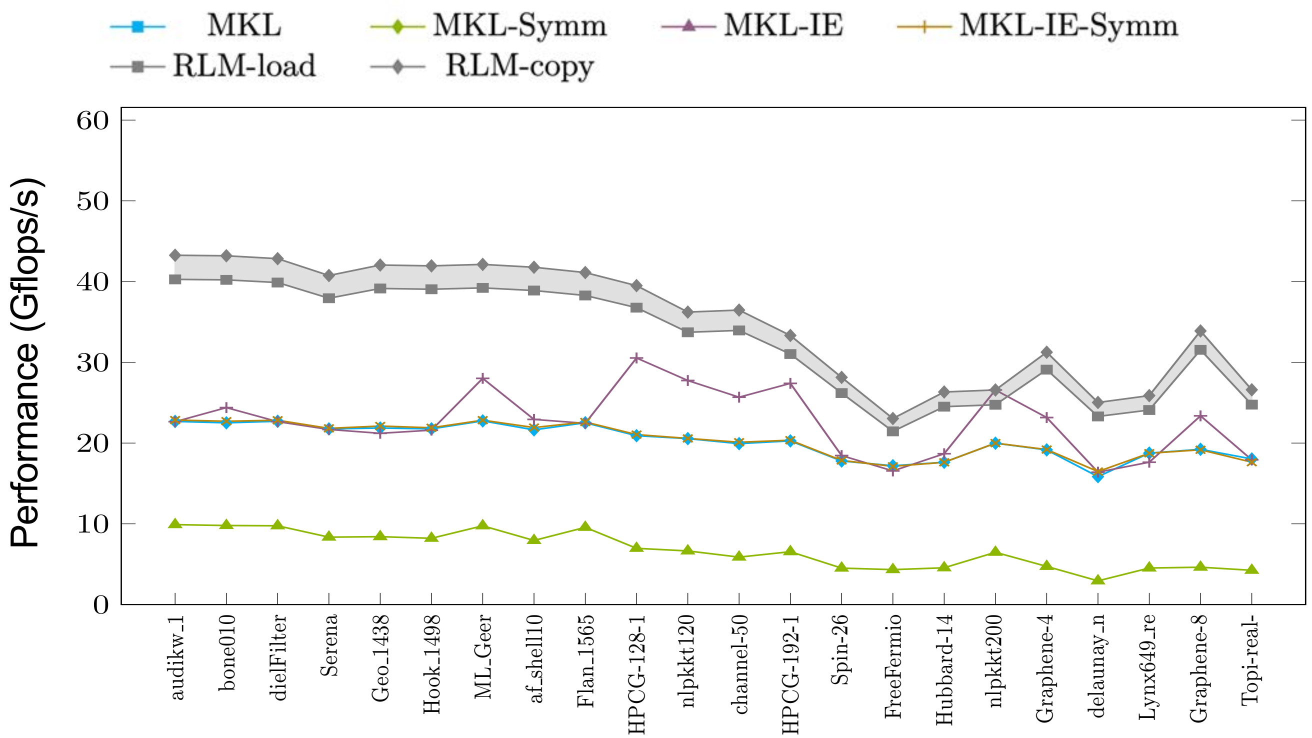
Performance of SymmSpMV on 1 socket of Intel Ice Lake



Intel Xeon 8368
38 cores
Bandwidth 170 GB/s

Intel MKL Symmetric SpMV performance

Performance of SymmSpMV on 1 socket of AMD ROME



AMD EPYC 7662
64 cores
Bandwidth 146 GB/s

How to parallelize SymmSpMV efficiently????

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

SpMV – graph traversal – RACE

RACE Hardware Friendly Coloring

Alappat et al., *A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication*. ACM Trans. Parallel Comput., 2020, DOI:10.1145/3399732

Consider SpMV as „graph traversal"

- Standard view: Run over all `rows i=1,…,n`$_r$ and calculate

$$y_i = \sum_{j \in COL(i)} A_{i,j} x_j$$

where `COL(i)` contains the column indices of the non-zeros in `i-th` row

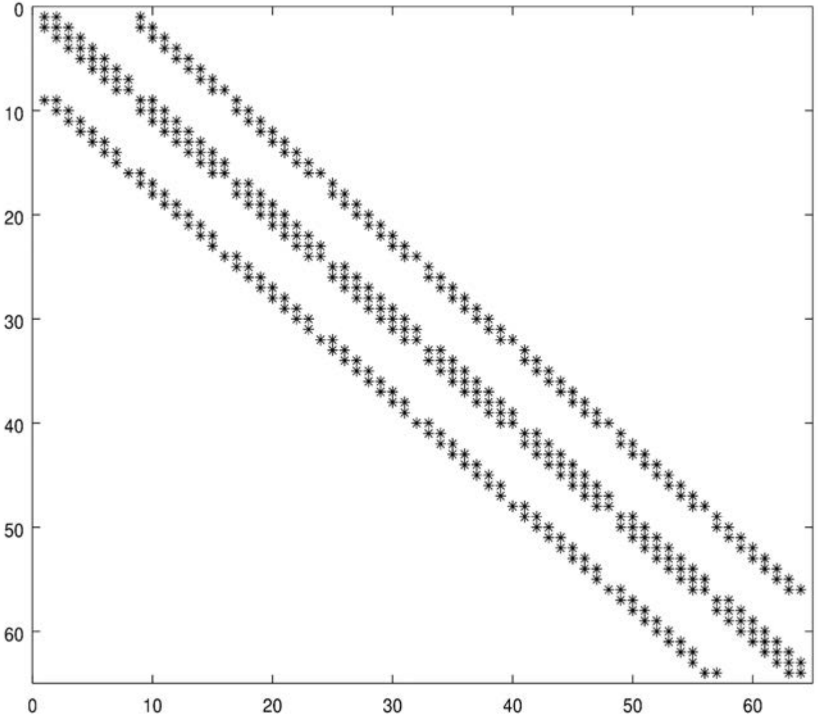
- RACE – **graph-based view**: row ←→ vertex & non-zero entry ←→ edge

In the graph terminology a SpMV operation ($y = Ax$) can be formulated as follows: If $G = (V, E)$ is the graph representation of the sparse matrix A then for every vertex $u \in V(G)$ calculate

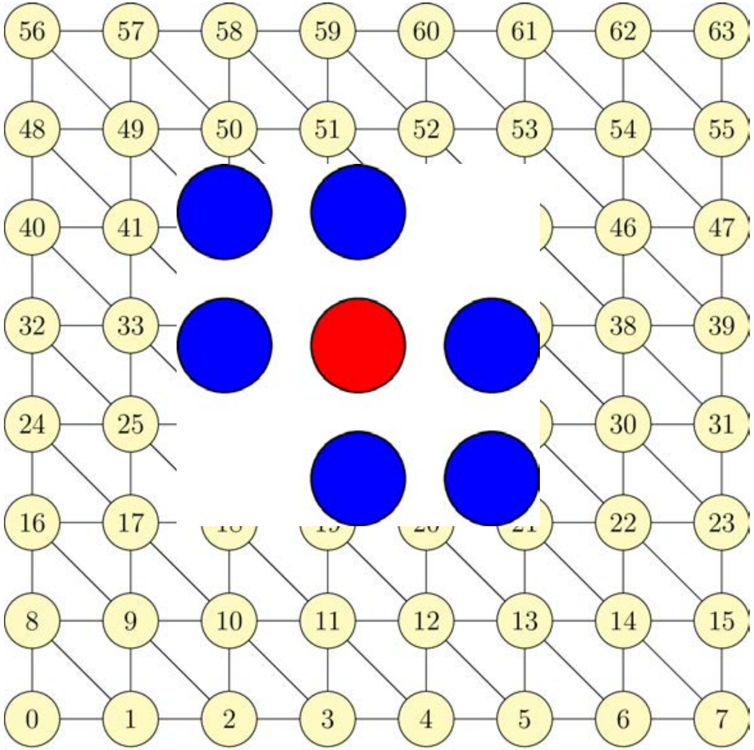
$$y_u = \sum_{v \in N(u)} A_{u,v} x_v \ . \tag{2}$$

Sample Stencil Matrix and its graph representation

Symmetric Matrix (Stencil)

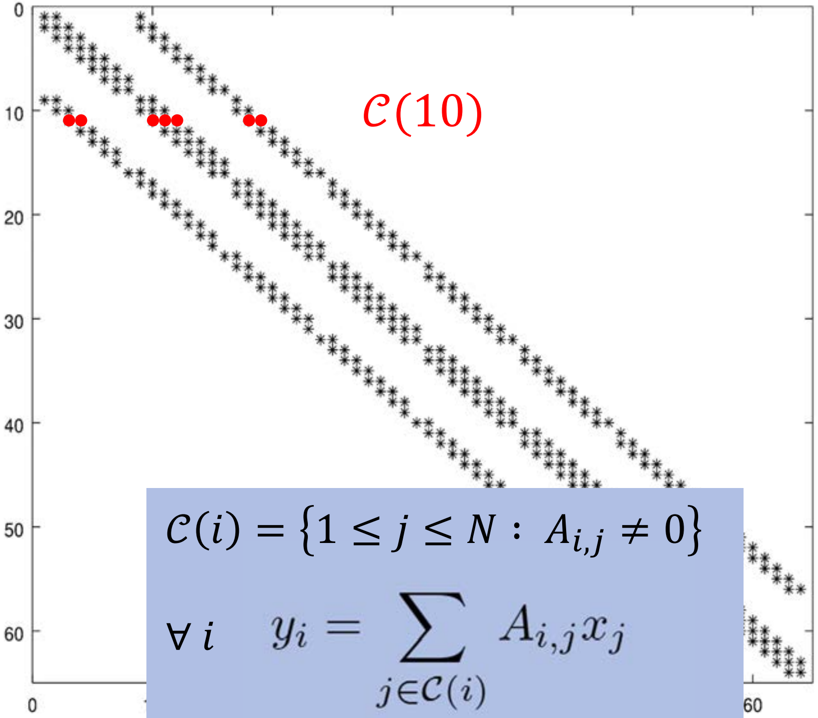


Undirected Graph



Sample Stencil Matrix and its graph representation

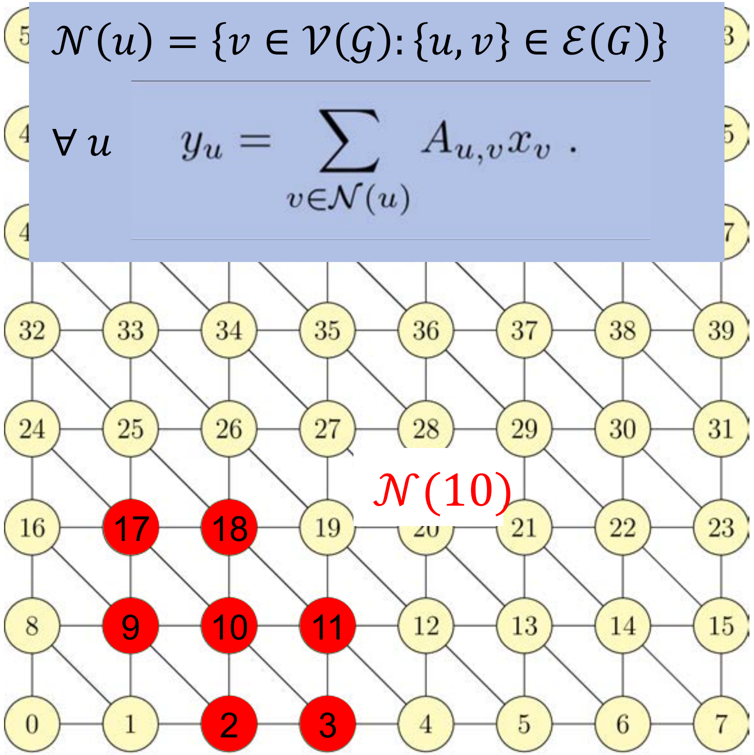
Symmetric Matrix (Stencil)



$$\mathcal{C}(i) = \{1 \le j \le N : A_{i,j} \neq 0\}$$

$$\forall\, i \quad y_i = \sum_{j \in \mathcal{C}(i)} A_{i,j} x_j$$

Undirected Graph

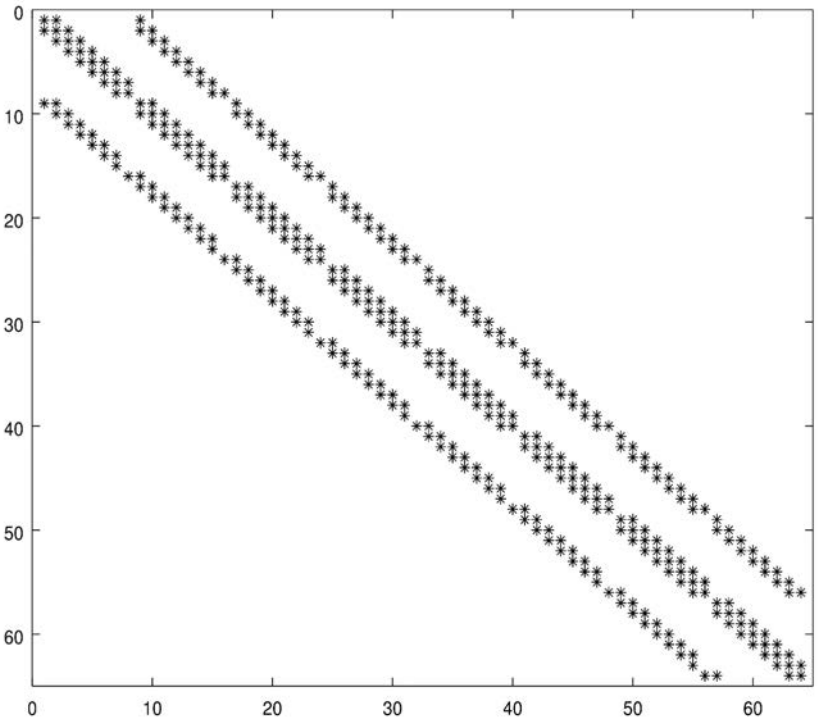


$$\mathcal{N}(u) = \{v \in \mathcal{V}(\mathcal{G}) : \{u, v\} \in \mathcal{E}(G)\}$$

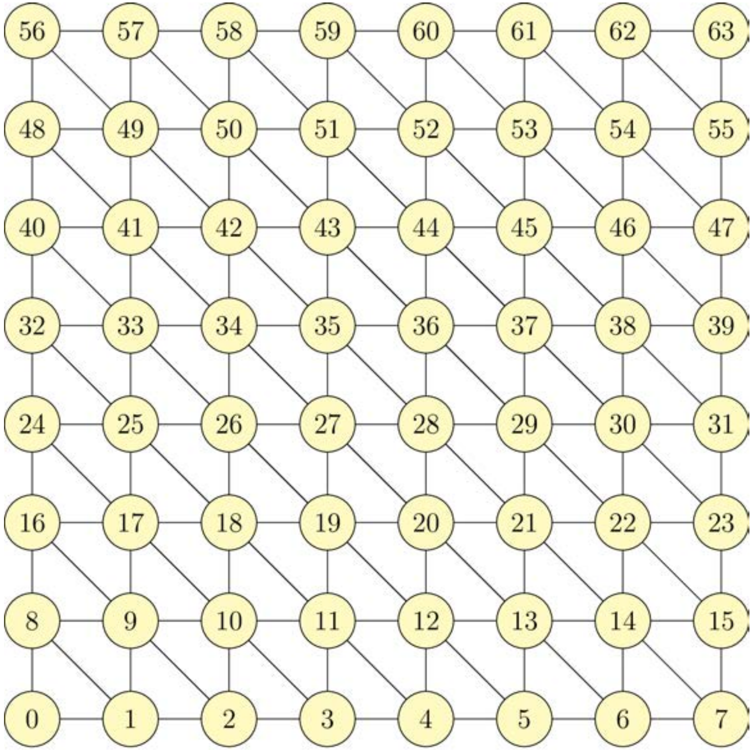
$$\forall\, u \quad y_u = \sum_{v \in \mathcal{N}(u)} A_{u,v} x_v \,.$$

Sample Stencil Matrix and its graph representation

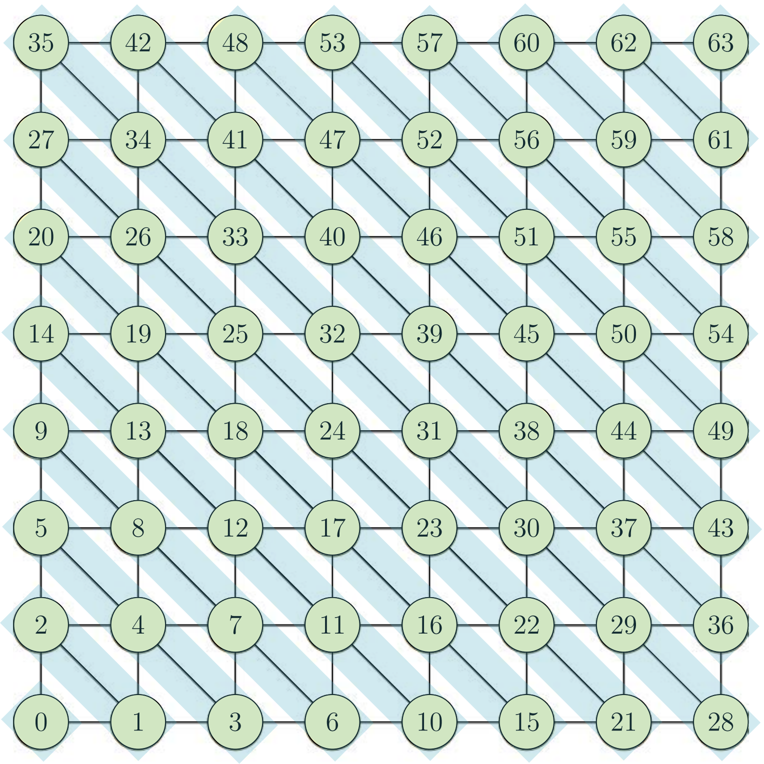
Symmetric Matrix (Stencil)



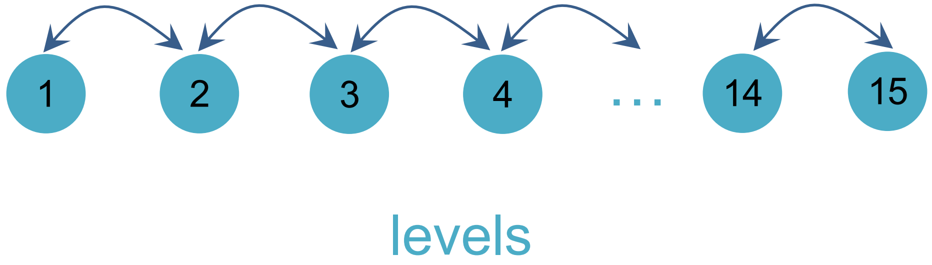
Undirected Graph



RACE: Get BFS-levels

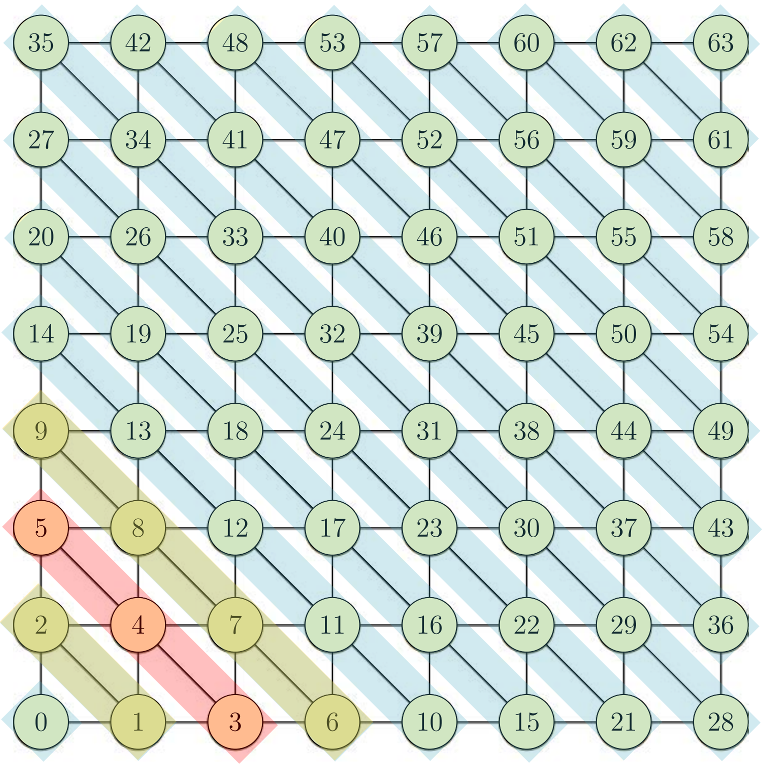


- Get levels of a Breadth-First-Search (BFS)
- Reorder vertices → consecutive witin level

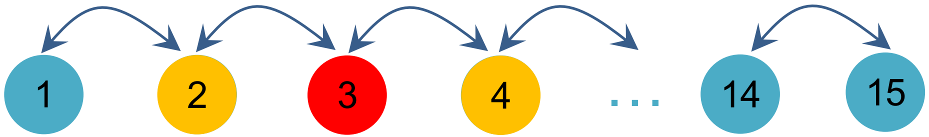


- Dependencies only between neighboring levels!

RACE: SpMV - Dependencies and Implementation



Update 3 → indirect access to 3, 2 and 4

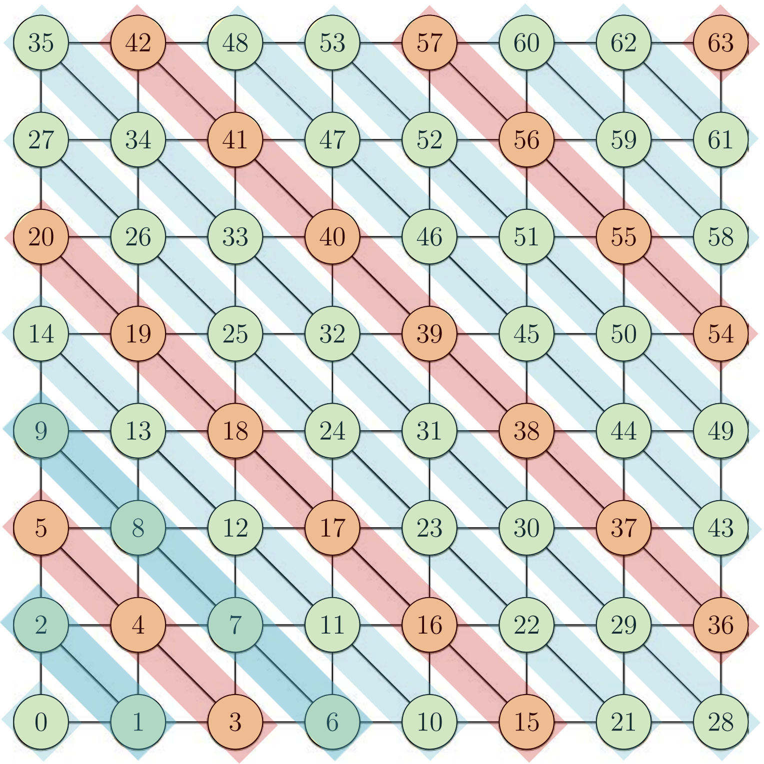


levels

```
do i = 1,L //loop over Levels
 SpMV_CRS(level_ptr[i], level_ptr[i+1])
enddo
```

```
function SpMV_CRS(start, end)
 do i = start, end
  do j = row_ptr(i), row_ptr(i+1) - 1
   y(i) = y(i) + val(j) * x(col_idx(j))
  enddo
 enddo
```

RACE: SymmSpMV - Parallelization

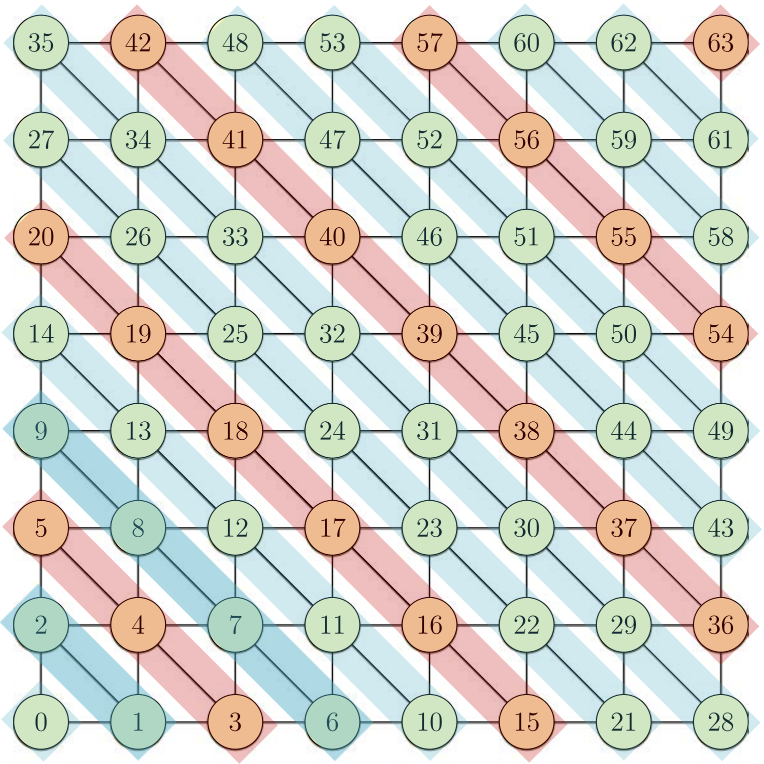


Symmetric SpMV – basic idea:
Compute distance-2 levels in parallel

Computing level 3 and 6 can be done independently → parallel

C. Alappat, A. Basermann, A. R. Bishop, H. Fehske, G. Hager, O. Schenk, J. Thies, and G. Wellein: *A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication.* ACM Trans. Parallel Comput. (2020). DOI: 10.1145/3399732

RACE: SymmSpMV - Parallelization



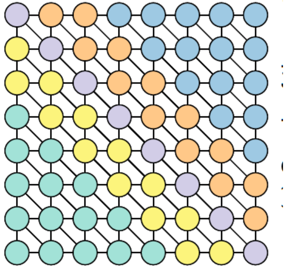
In general:

- „Distance-2“ levels can be updated in parallel (same color)
- Assign levels to threads
- But: Load imbalance!!!!
- #levels >> #threads

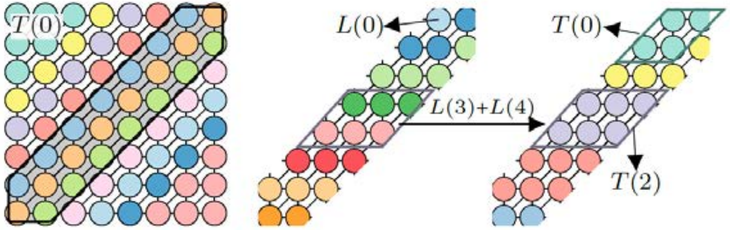
C. Alappat, A. Basermann, A. R. Bishop, H. Fehske, G. Hager, O. Schenk, J. Thies, and G. Wellein: *A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication.* ACM Trans. Parallel Comput. (2020). DOI: 10.1145/3399732

RACE: SymmSpMV - Parallelization

- Further optimization strategies (for details see MPK)
  - Replace levels by level groups
    → better load balancing



  - Recursive level refinement
    → Cache locality / parallelism

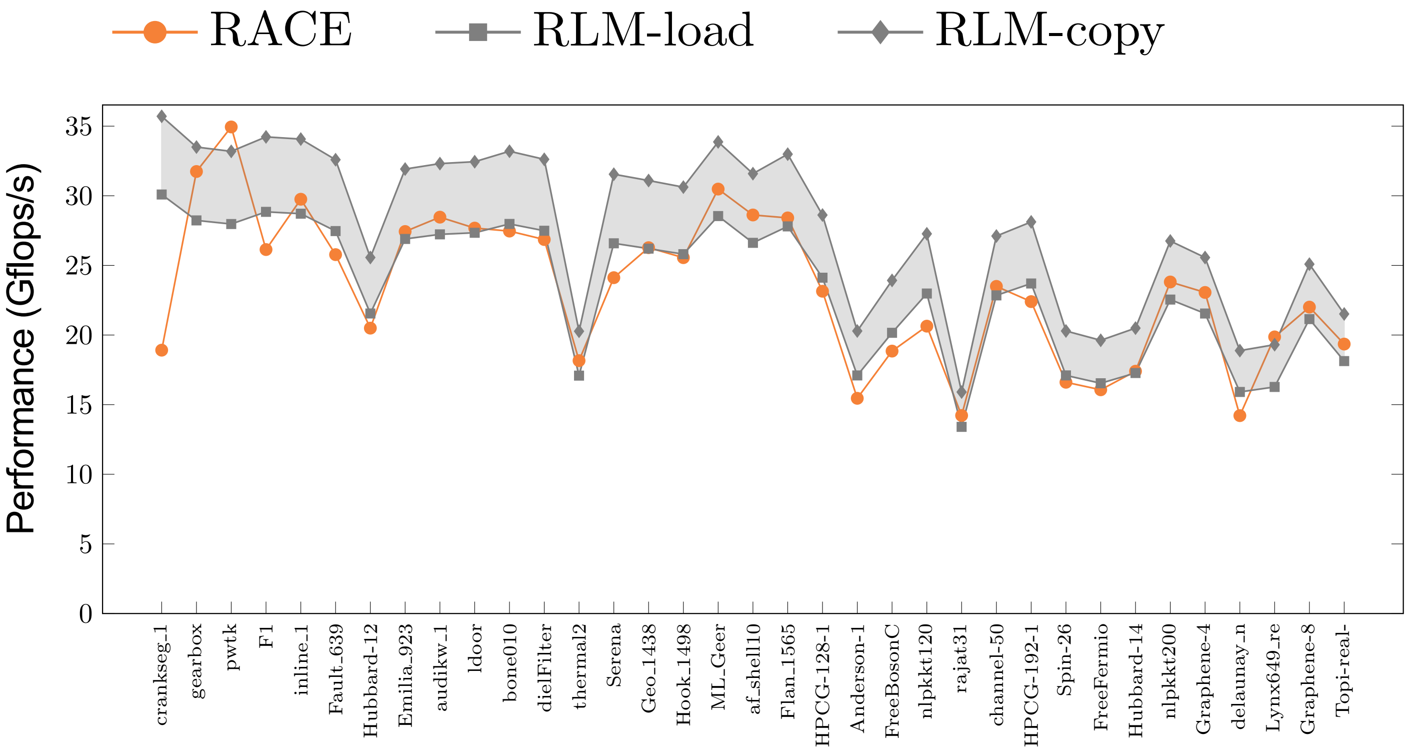


(a) Graph. (b) Levels. (c) Level groups.

  - Avoid global thread sync → Pair-wise if required

MKL Symmetric SpMV performance

Performance of SymmSpMV on 1 socket of Intel Cascade Lake

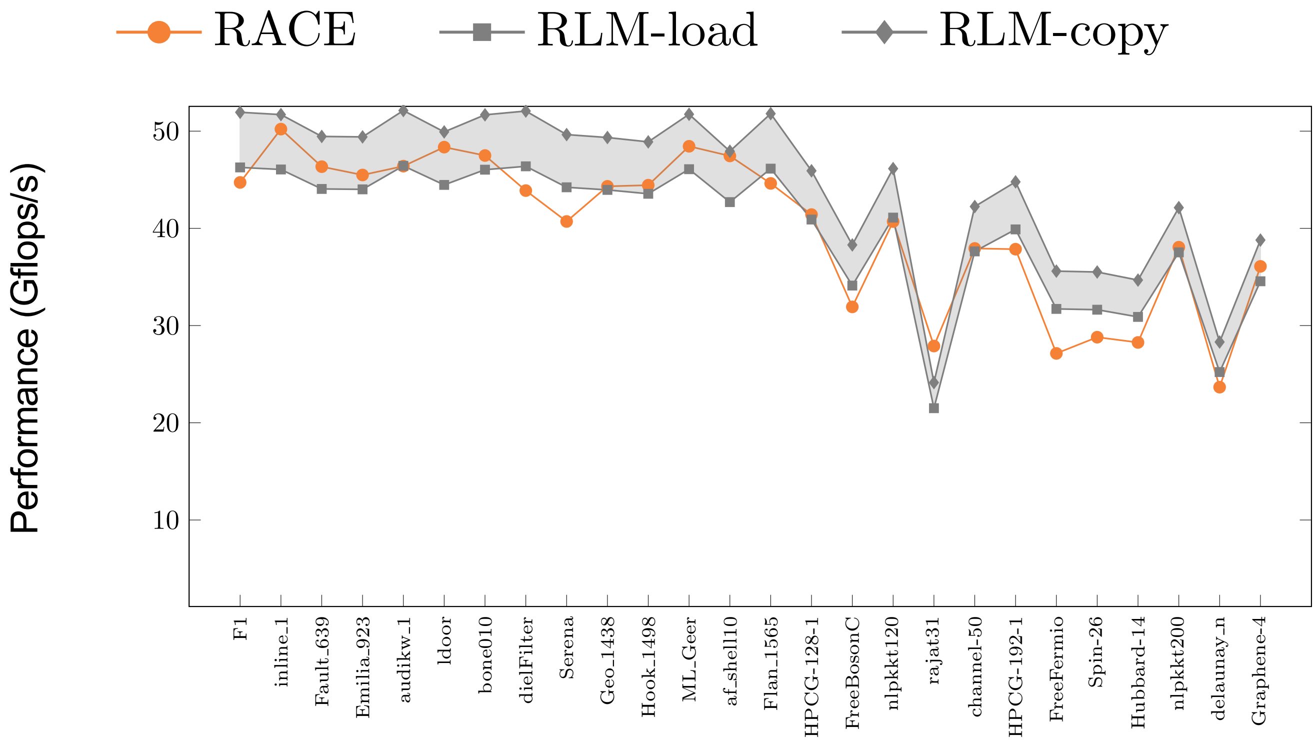


Intel Xeon 6248
20 cores
Bandwidth 115 GB/s

Looks good ☺

MKL Symmetric SpMV performance

Performance of SymmSpMV on 1 socket of Intel Ice Lake

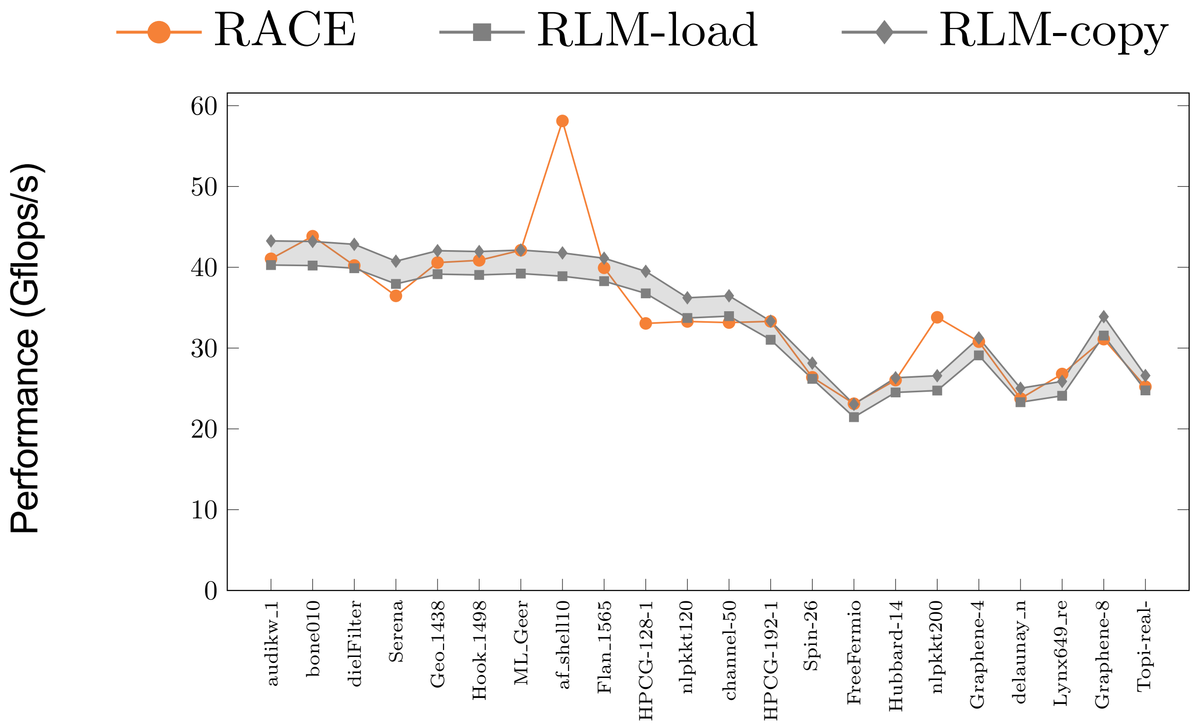


Intel Xeon 8368
38 cores
Bandwidth 170 GB/s

Looks good ☺

MKL Symmetric SpMV performance

Performance of SymmSpMV on 1 socket of AMD ROME



AMD EPYC 7662
64 cores
Bandwidth 146 GB/s

Looks good ☺

RACE & Cache Blocking for MPK

IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, VOL. 34, NO. 2, FEBRUARY 2023 581

Level-Based Blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication

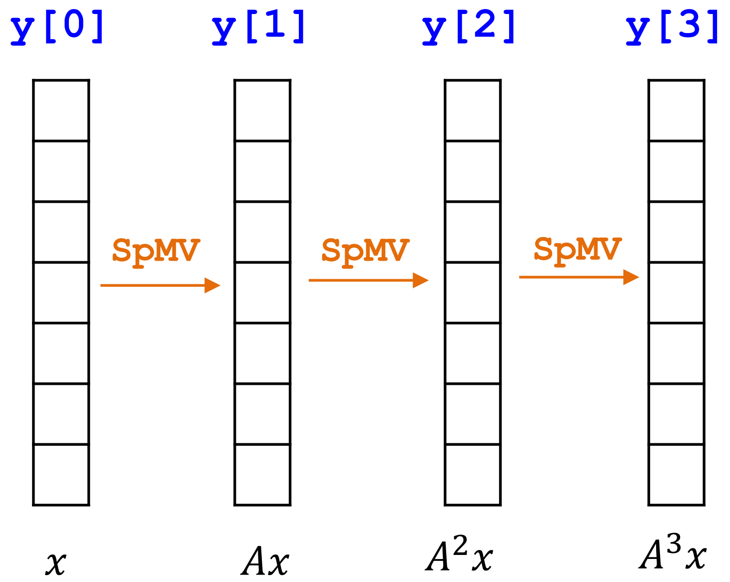
Christie Alappat, Georg Hager, Olaf Schenk, *Senior Member, IEEE*, and Gerhard Wellein

C. Alappat, et al, doi: 10.1109/TPDS.2022.3223512

Motivation – Matrix power kernel (MPK)

- Calculate: $y = A^p x$
- Repeatedly perform back to back SpMVs

```
for k=1:p; do
 y[k] = SpMV(A, y[k-1])
done
```



Same matrix A loaded p times from main memory!!!

How to cache the matrix A across the matrix power calculation?

MPK – existing caching approaches

- Huber et al.: Graph-based higher-order time integration of PDEs[1]
  - "Geometrical approach" based on matrix bandwidth
  - Works for 2D stencil matrices → Runs into problem for 3D and/or unstructured matrices

- Mohiyuddin et al.: Minimizing communication in sparse matrix solvers[2]
  - "Domain decomposition" of underlying graph
  - Requires "ghosting" → Indirect accesses or redundant copies of the matrix entries → Scalability!!

→ Exploit level structure in RACE for cache blocking!

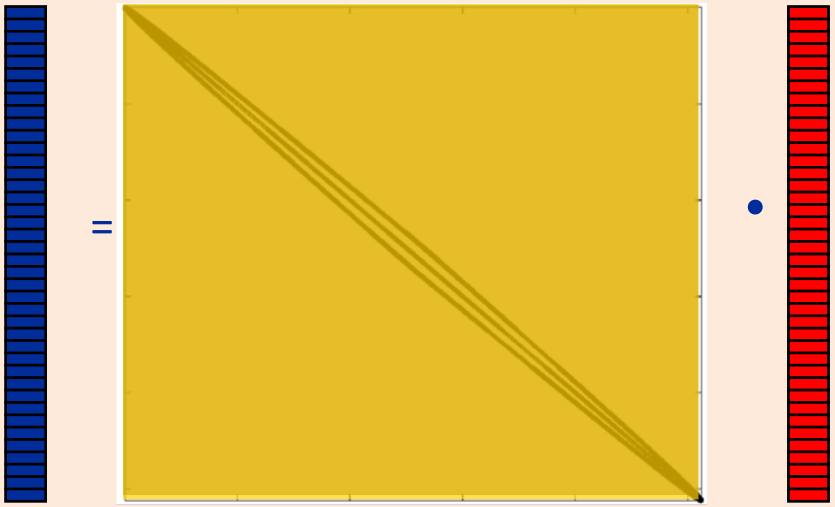
RACE

[1]Huber et al., 2021. Graph-based multi-core higher-order time integration of linear autonomous partial differential equations. J. Comput. Sci. DOI:10.1016/j.jocs.2021.101349
[2]Mohiyuddin et al., 2009. Minimizing communication in sparse matrix solvers. In Proceedings of the SC'09. DOI:10.1145/1654059.1654096

Matrix power – Traditional approach vs. Cache Blocking

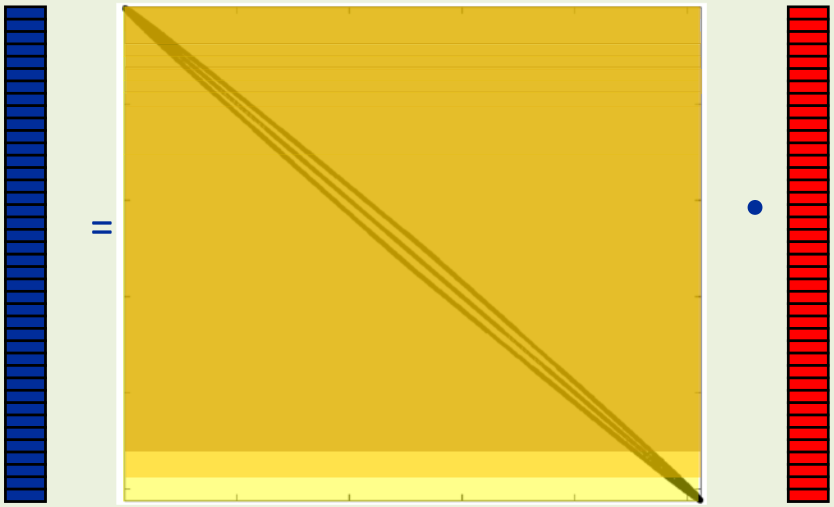
Calculate $y = A^3 x$

TRAD approach



Matrix accessed 3 times from memory

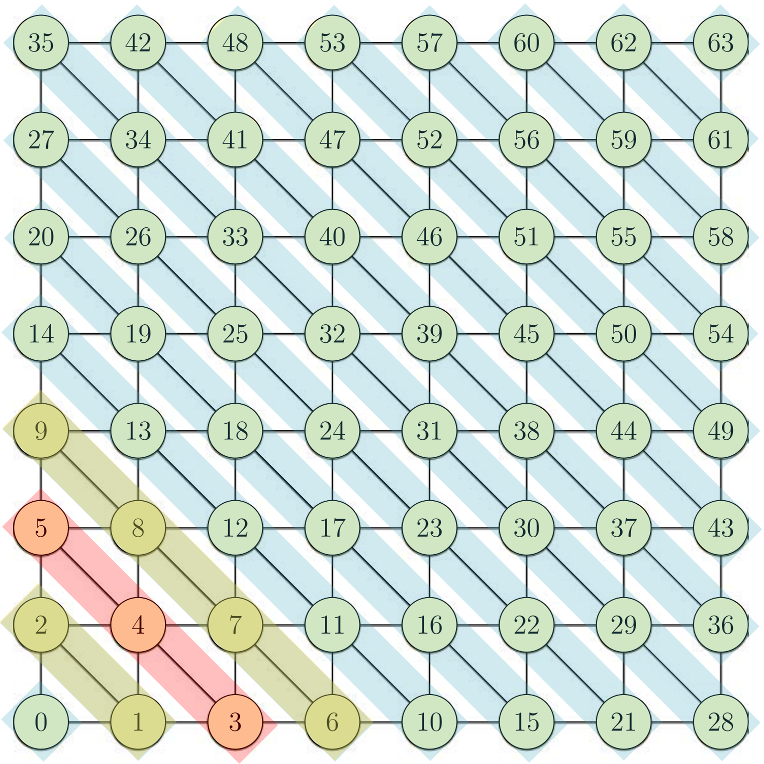
RACE approach

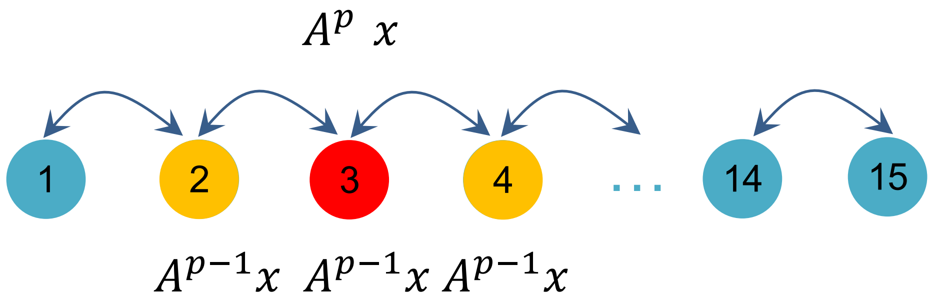


Matrix accessed 1 time from memory

How to do that in general for sparse matrices?

RACE

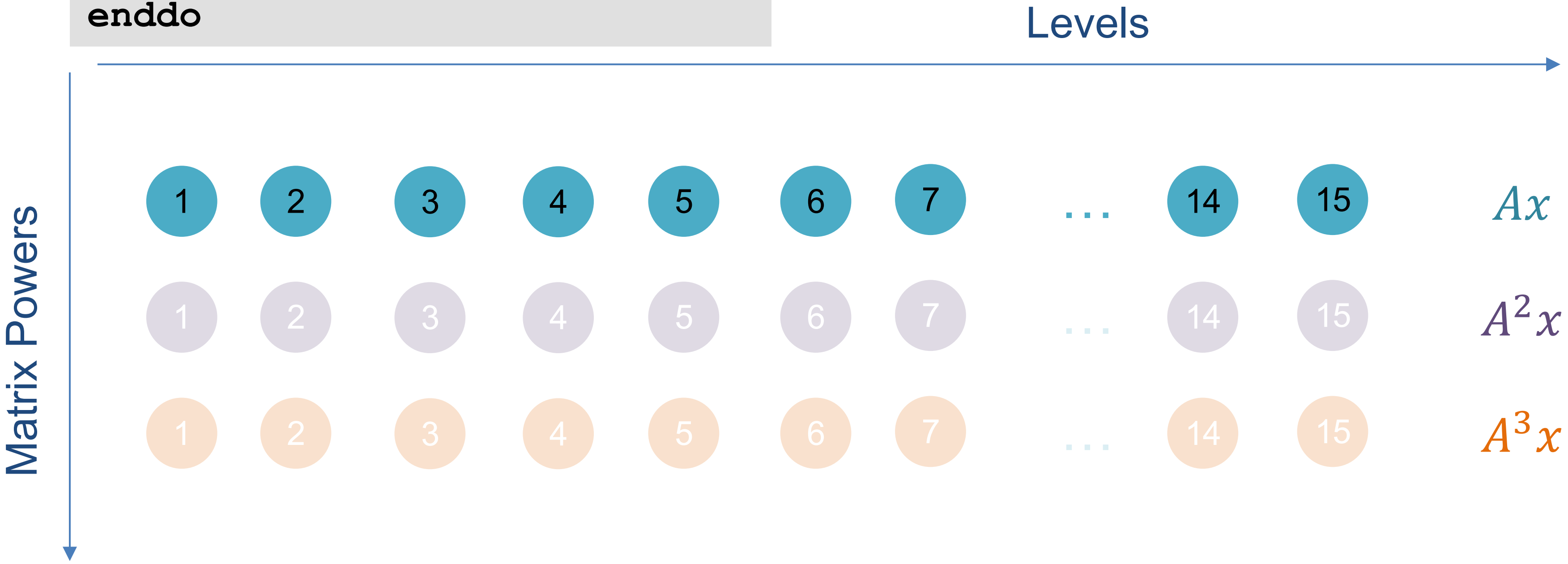




RACE – Level traversal and matrix powers

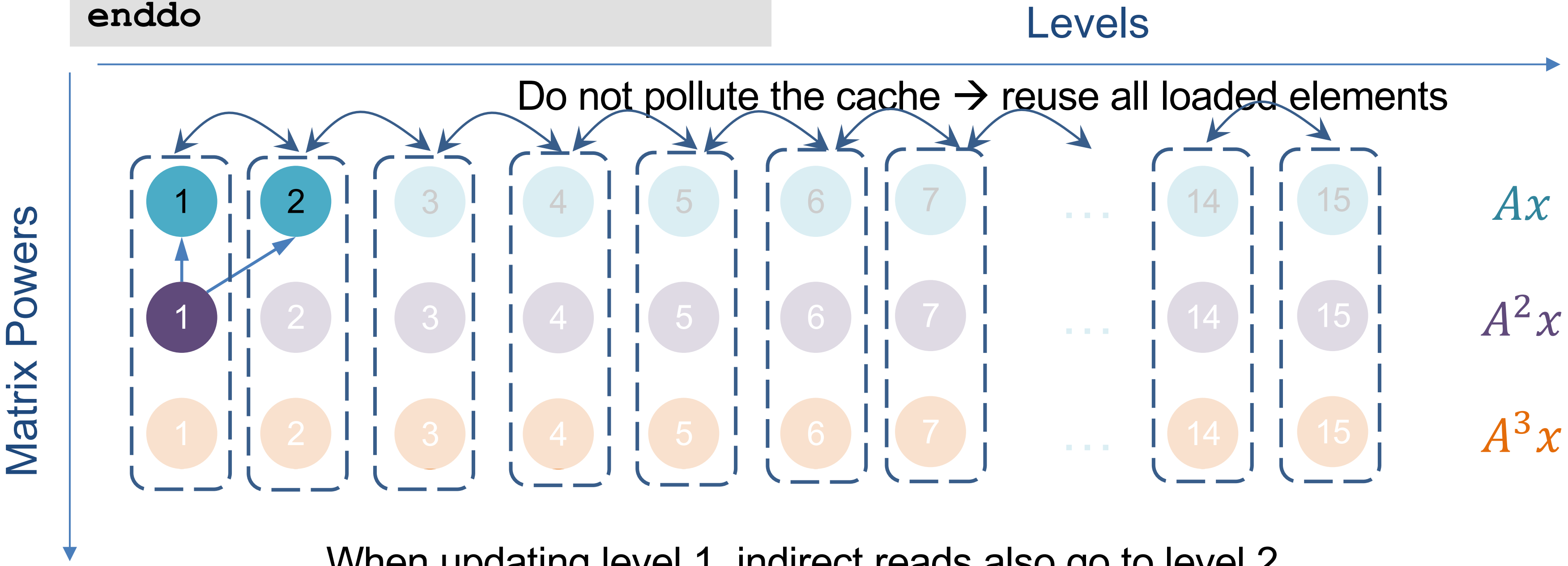
```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```

No cache blocking!



RACE – Level traversal and matrix powers

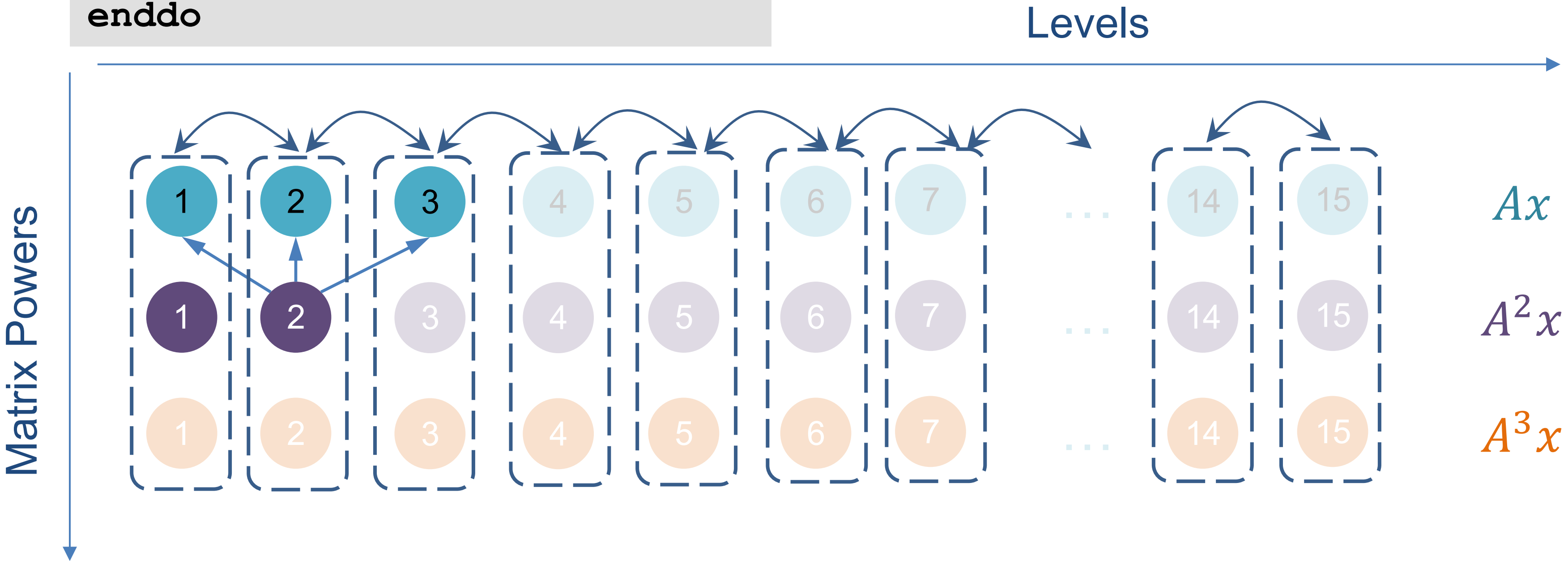
```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



When updating level 1, indirect reads also go to level 2

RACE – Level traversal and matrix powers

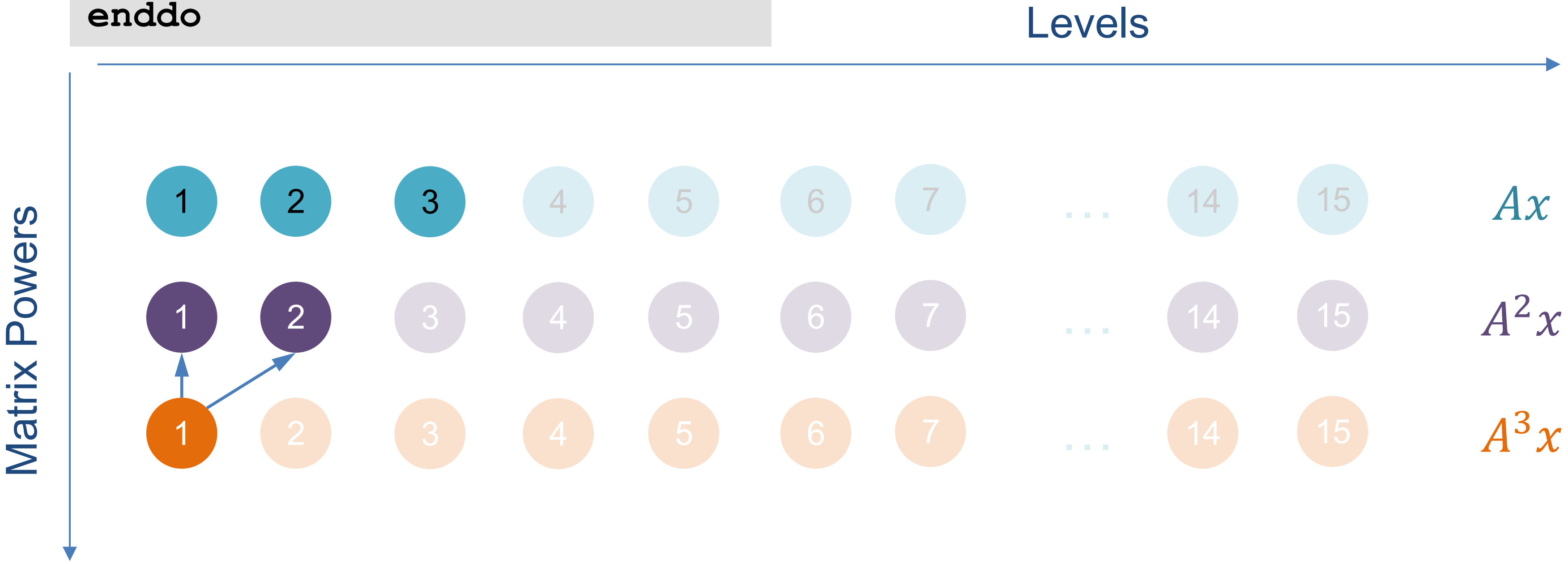
```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



When updating level 2, indirect reads also go to levels 1 and 3

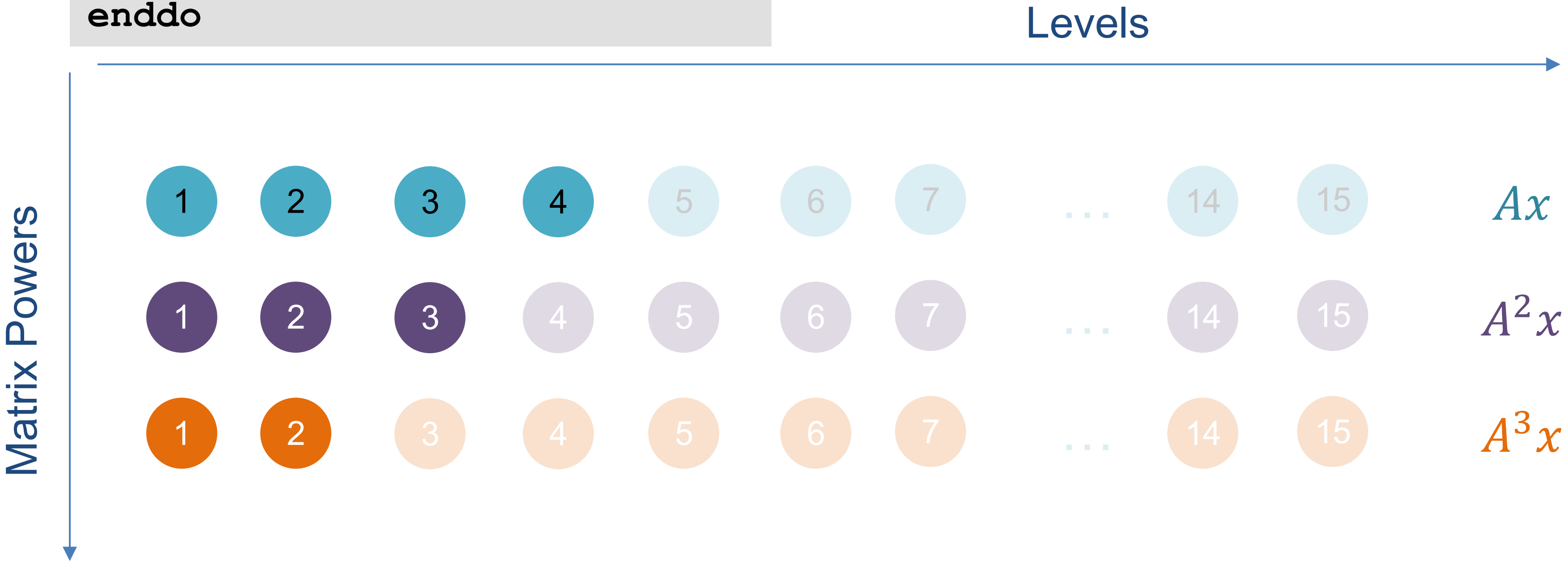
RACE – Level traversal and matrix powers

```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



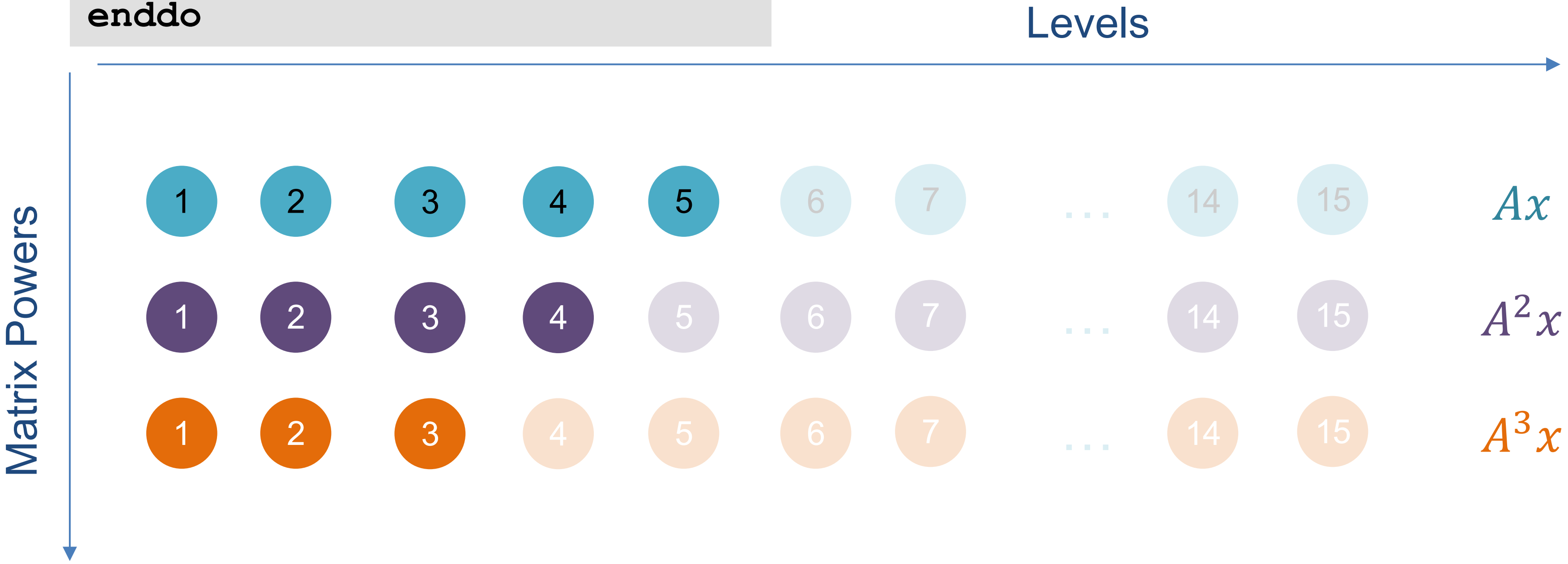
RACE – Level traversal and matrix powers

```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



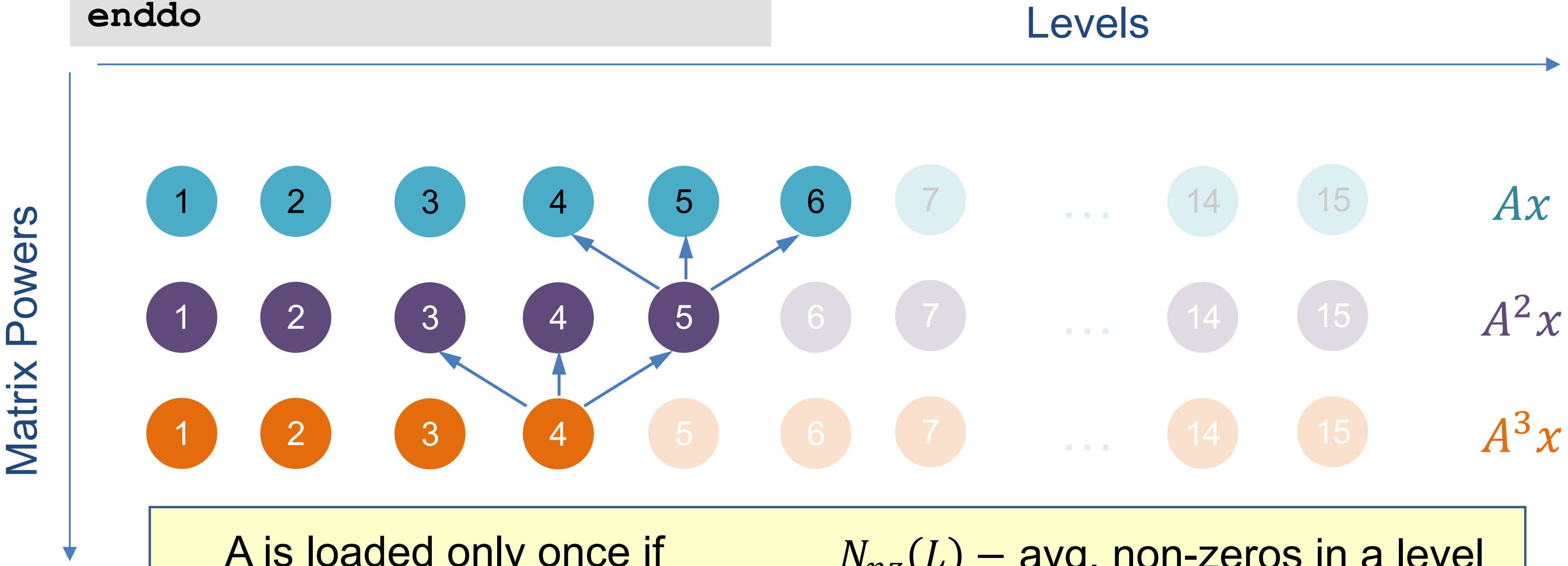
RACE – Level traversal and matrix powers

```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



RACE – Level traversal and matrix powers

```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



A is loaded only once if

$(p+1) \times N_{nz}(L) \times 12$ bytes $< C$

$N_{nz}(L)$ – avg. non-zeros in a level
C – cache size

RACE: MPK Pseudocode

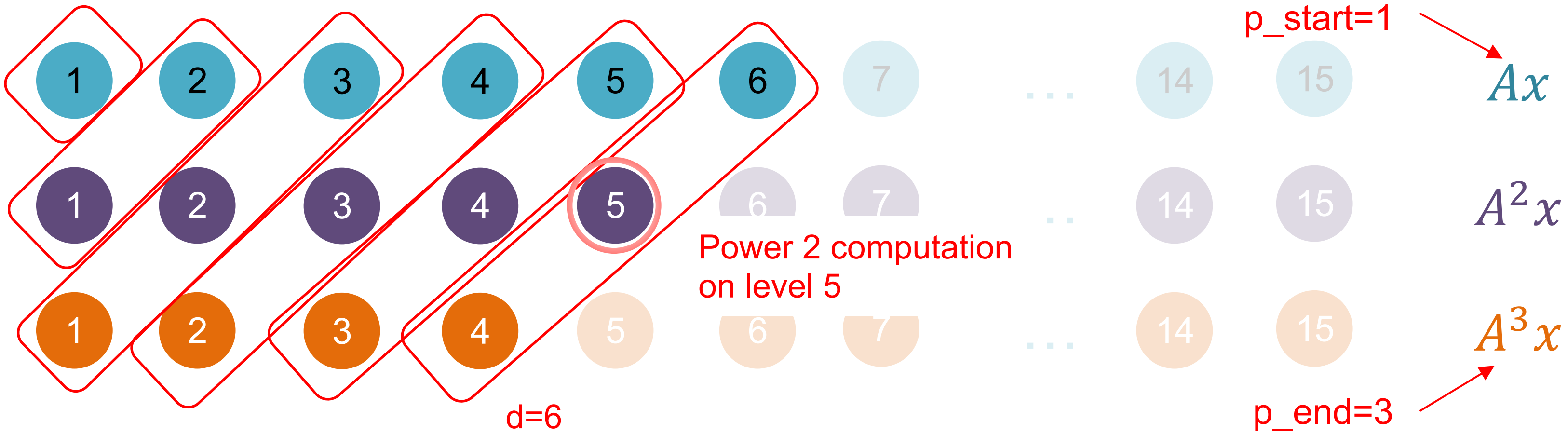
```
do d in 1:L+p-1
 p_start = max(1, d-(L-1))
 p_end = min(d, p)
 do k in p_start:p_end
  l=(d-k+1)
  y(:, k) = SpMV(A(j,:), y(:,k-1), level_ptr[l]:level_ptr[l+1])
 enddo
enddo
```

← Traverse along diagonal

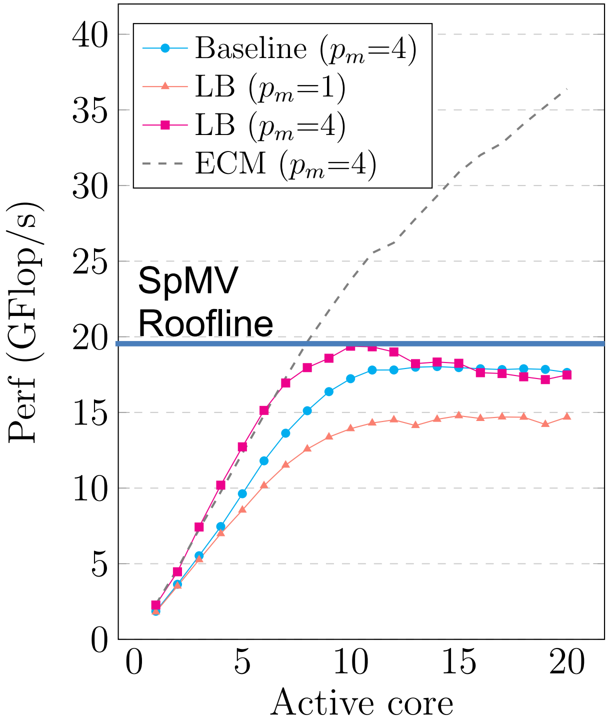
← All powers in diagonal

← Power k computation on level l

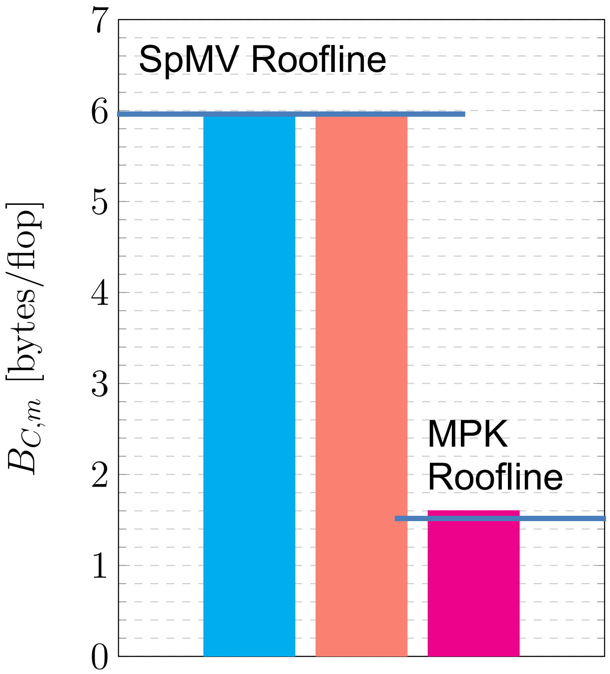
OpenMP parallel SpMV on all vertices in level l



RACE MPK – First Implementation



Performance



Memory traffic

Intel Xeon Gold 6248

- 1 Socket (20c)

`pwtk` matrix

- N_r = 217,918
- N_{nz} = 11,634,424

RACE MPK – Performance Problem Identified

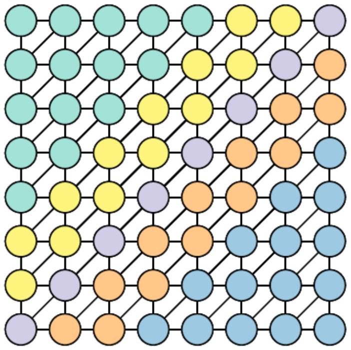
- Scheme seems to work (reduces data traffic) – at least for **`pwtk`**
- But: Performance ☹ !!!!

- Analysis of hardware performance counters (LIKWID) for **`pwtk`** matrix:
  `INSTR_RETIRED_ANY` up 2x for level based SpMV!

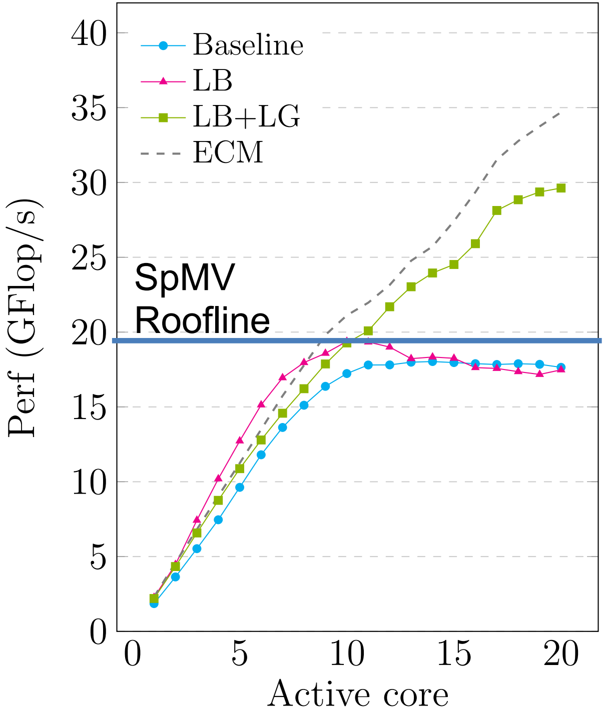
  → Frequent thread syncronisations!

  Reason: After each level threads sync!

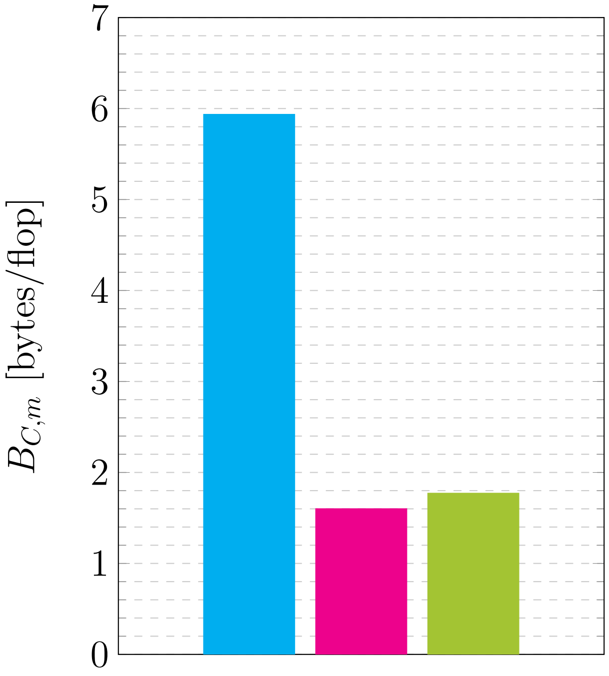
  Measures:
  → Reduce #levels by level aggregation („LG“)
  → Global sync. replaced by point-to-point sync. („p2p“)



RACE MPK – LG optimization



Performance

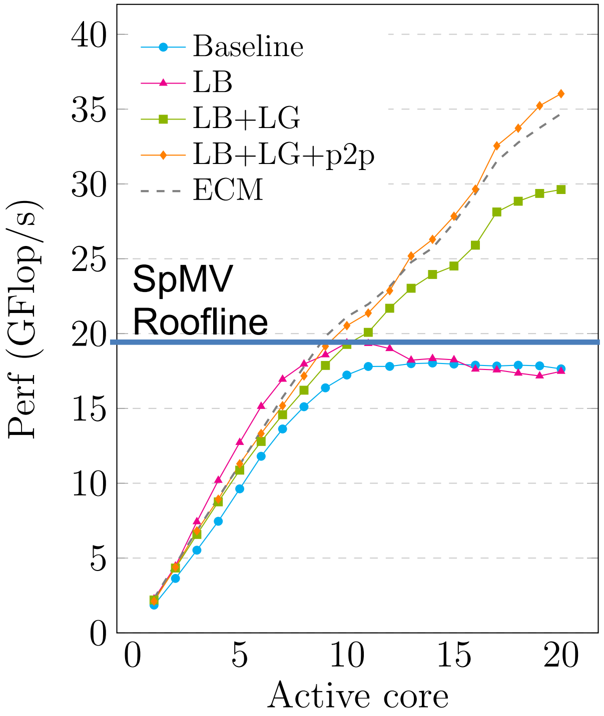


Memory traffic

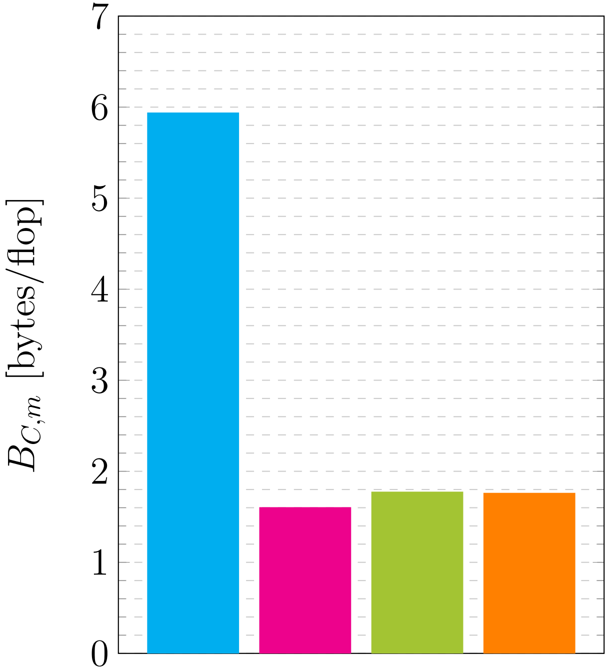
Intel Xeon Gold 6248
- 1 Socket (20c)

`pwtk` matrix
- N_r = 217,918
- N_{nz} = 11,634,424

RACE MPK – LG+p2p optimization



Performance



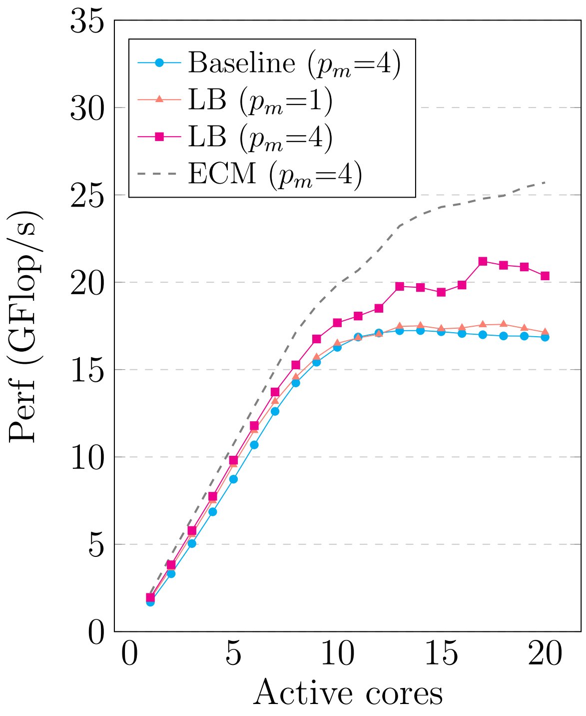
Memory traffic

Intel Xeon Gold 6248
- 1 Socket (20c)

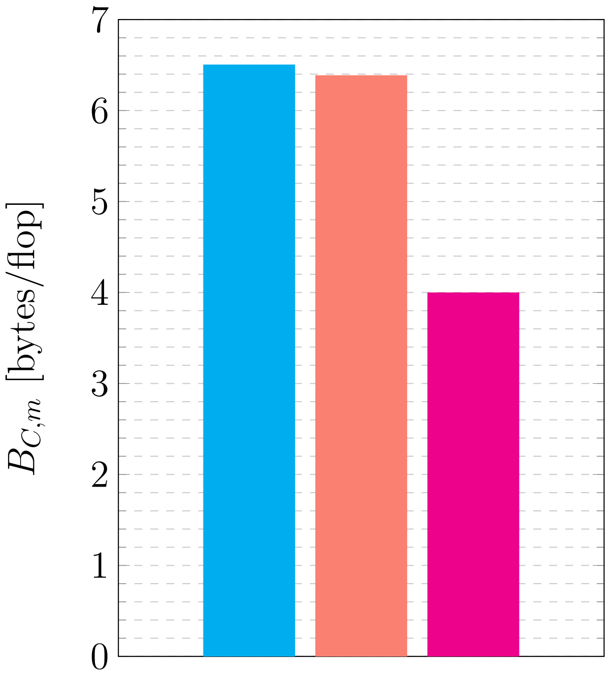
`pwtk` matrix
- N_r = 217,918
- N_{nz} = 11,634,424

☺

RACE MPK – Yet another problem



Performance



Memory traffic

Intel Xeon Gold 6248
- 1 Socket (20c)

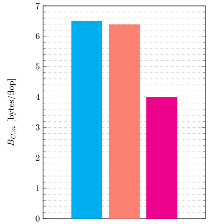
Flan_1565 matrix
- N_r = 1,564,794
- N_{nz} = 117,406,044

Data traffic not reduced by factor of 4

Representative for large matrices!

RACE MPK – Performance Problem Identified (II)

- `Flan_1565` matrix – no significant adequate in data volume
- Analysis of level distribution for `Flan_1565` matrix:



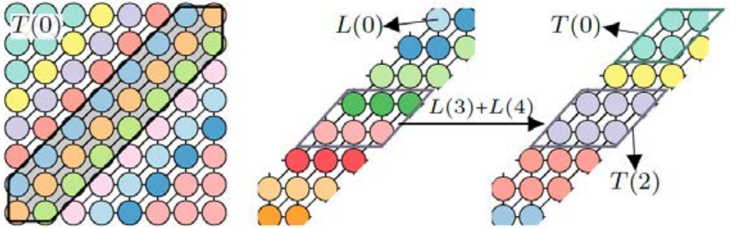
Few large levels → do not fit in cache!

$$(p+1) \times N_{nz}(L) \times 12 \text{ bytes} < C$$

Too big!!!

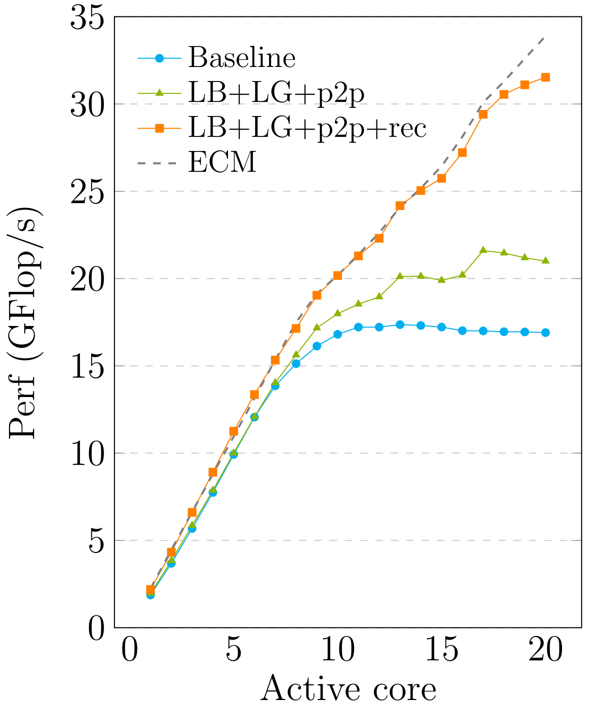
Counter - Measure:

→ Apply RACE to a single / few levels recursively („rec“)!

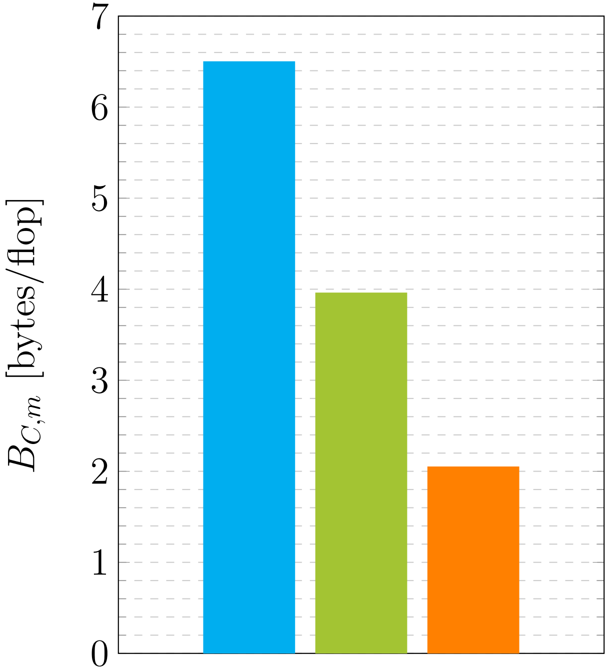


(a) Graph. (b) Levels. (c) Level groups.

RACE MPK – rec



Performance



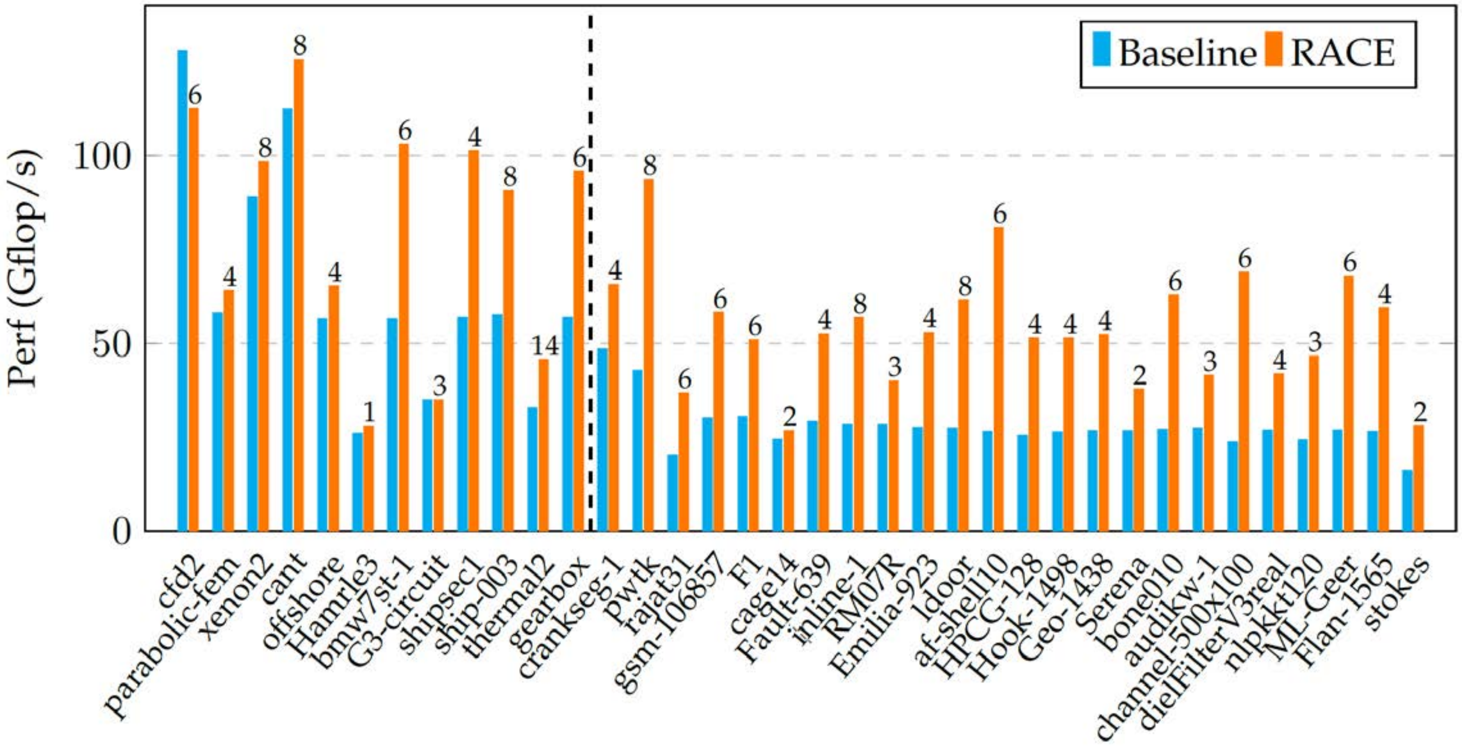
Memory traffic

Intel Xeon Gold 6248
- 1 Socket (20c)

`Flan_1565` matrix
- N_r = 1,564,794
- N_{nz} = 117,406,044

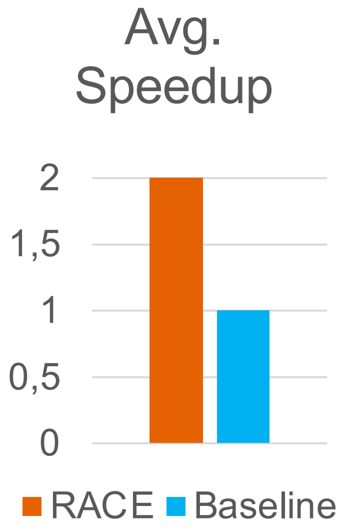
☺

Matrix power kernel: Performance – Intel Ice Lake

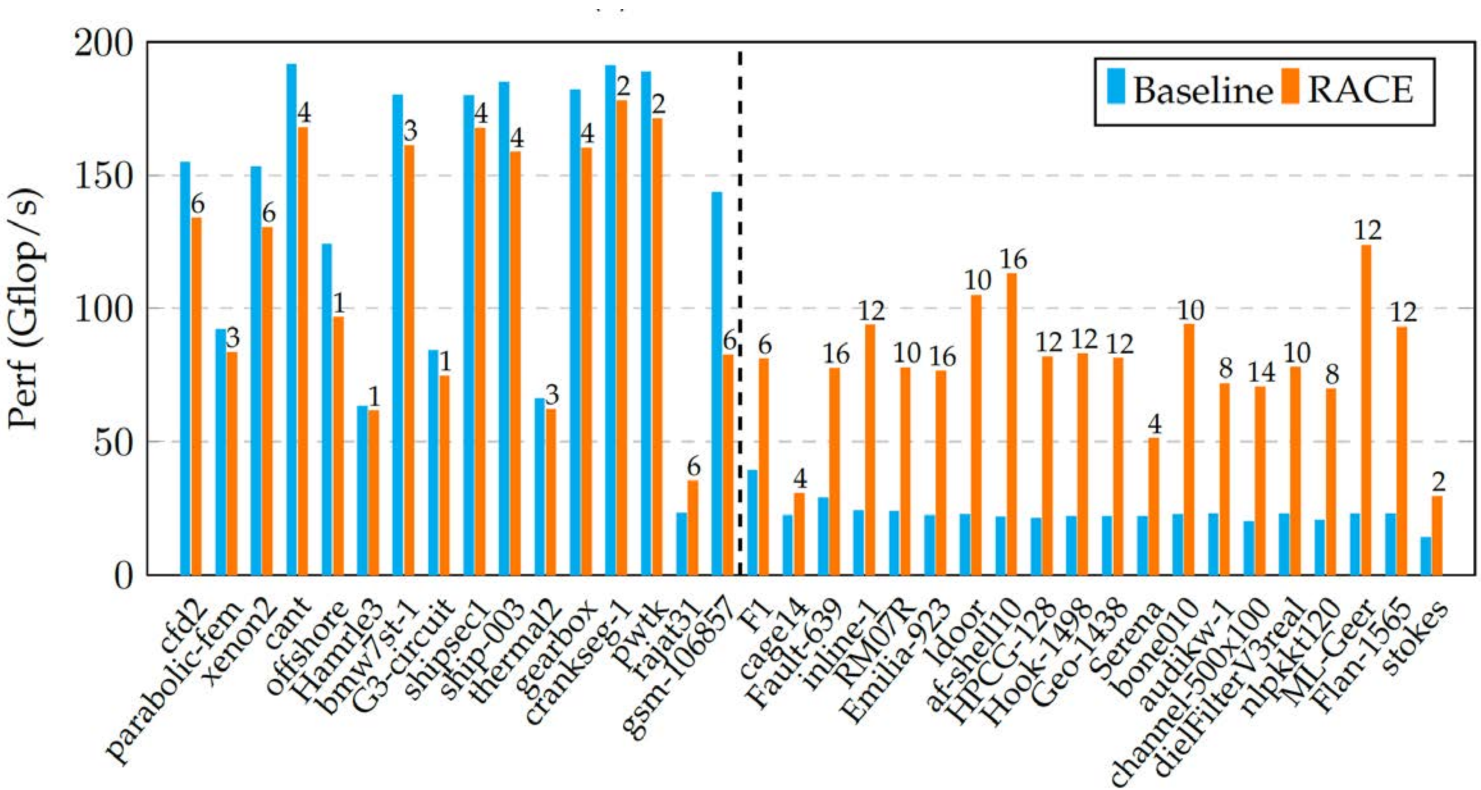


Intel Xeon Platinum 8368

- 38 cores
- 104 MB cache (L2+L3)

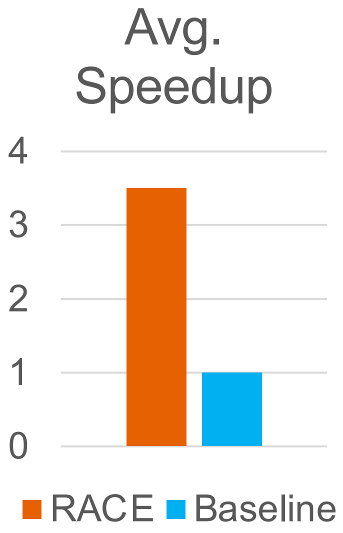


Matrix power kernel: Performance – AMD Rome



AMD EPYC 7662

- 64 cores
- 288 MB cache (L2+L3)



RACE - summary

- Inner kernel: OpenMP parallel standard SpMV routine
- Overhead: BFS & Set up of data structures (approx. ≤ 50 SpMVs)
- Parameters: Power (p_m), Available Cache Size, Max. recursion depth
- Cache size ←→ max. polynomial degree (p_m)
  - Larger caches → larger p_m → better performance
  - Polynomial degree higher than p_m → Computation in chunks of p_m
- No loss of accuracy!

RACE – MPK applications

- Exponential Integrators → Polynomial approximations

- s-step Krylov methods (CA-GMRES)
- Polynomial preconditionung
- Algebraic Multigrid smoothers

- Trilinos interface available

Summary

- MPK kernel's performance → substantial boost by level-based cache blocking using RACE.

- MPK + RACE is attractive for s-step Krylov solvers. For example CA-GMRES solver
- Very promising results if combined with polynomial preconditioners

- Chebyshev smoothers for multigrid

  →Large Scale Sparse Solvers
- Exponential integrators, e.g. Chebyshev time propagation

…

Thank you

Questions

RACE

Hardware Friendly Coloring

https://github.com/RRZE-HPC/RACE

Backup

Matrix power

Calculate $y = A^3 x$

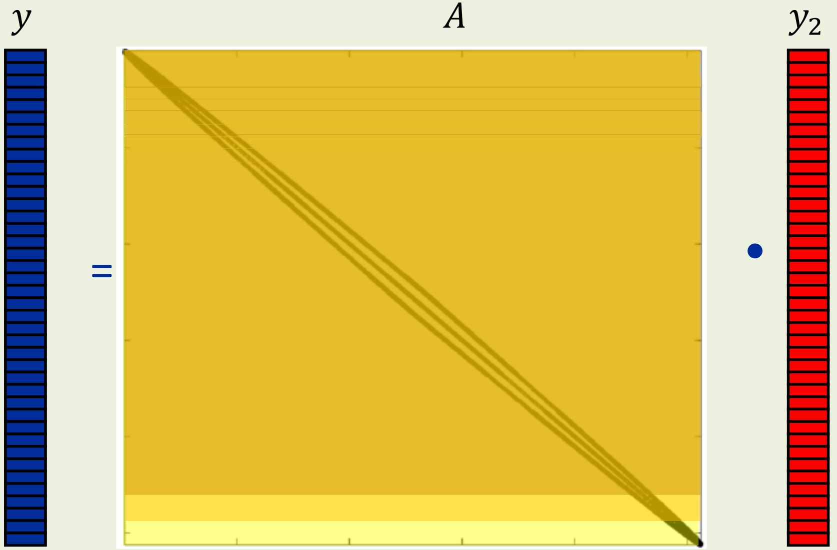
RACE approach

Level view

$y_1 = A\,x$	1	2	3	4	5	6	...
$y_2 = A^2 x$	1	2	3	4	5	6	...
$y \;= A^3 x$	1	2	3	4	5	6	...

For more details on RACE MPK:
C. Alappat, G. Hager, O. Schenk, G. Wellein, 2022, Level-based Blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication, Submitted, Preprint: https://arxiv.org/abs/2205.01598

Matrix view



Matrix accessed 1 time from Memory

Matrix power kernel (MPK)

- Calculate: $y = A^p x$
- Repeatedly perform back to back SpMVs

```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```

Applications

s-step Krylov solvers

- $\boxed{s\text{-step GMRES}}$
- s-step CG

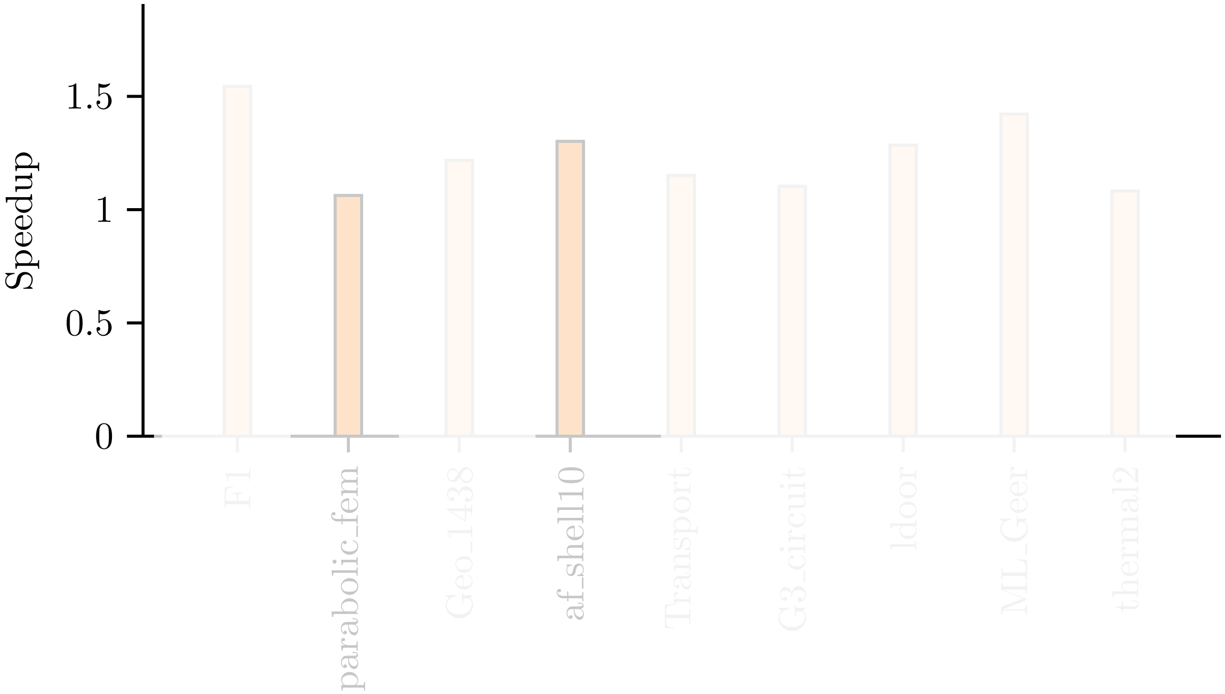
Matrix polynomials

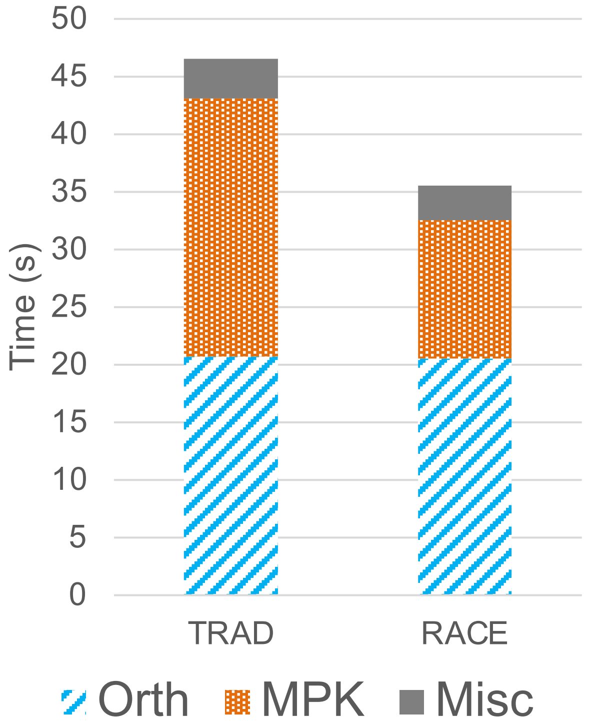
- Chebyshev time propagation
- Exponential integrators
- Polynomial preconditioning

Power iterations

- Eigenvalue computations
- Power iteration clustering

Application: Communication avoiding GMRES

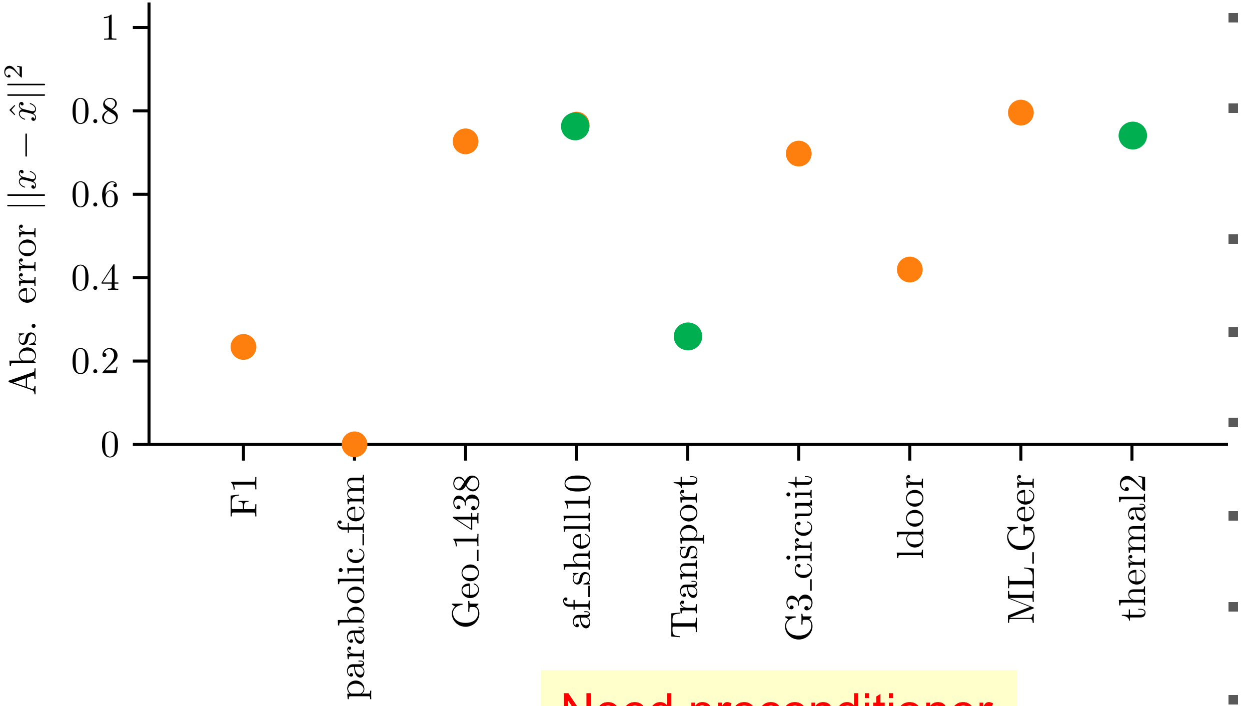




Avg. speedup of 1.2 x with RACE

Orthogonalization dominant, reducing overall speedup

Application: Communication avoiding GMRES



Need preconditioner

- Solve $y = Ax$, with exact solution $||\hat{x}||$ = 1
- TPETRA GMRES S-STEP solver in BELOS
- Max. 5000 iterations
- No preconditioner
- Orthogonalization best of {ICGS, IMGS}
- Restart length best in {30,50,100}
- RACE MPK power set to 4
- 1 socket of Intel Xeon Platinum 8368 (Ice Lake)

Let's choose three matrices for deeper analysis

Polynomial preconditioners ($AMy = b$, $x = My$)

The idea

- Find roots of $(I - Ap(A))$, i.e., solve: $\left(I - Ap(A)\right) = 0$
- The resulting $p(A)$ is an approximation for A^{-1}
- Degree of the polynomial is determined by the number of roots sought after

How to find roots of $\left(I - Ap(A)\right)$?

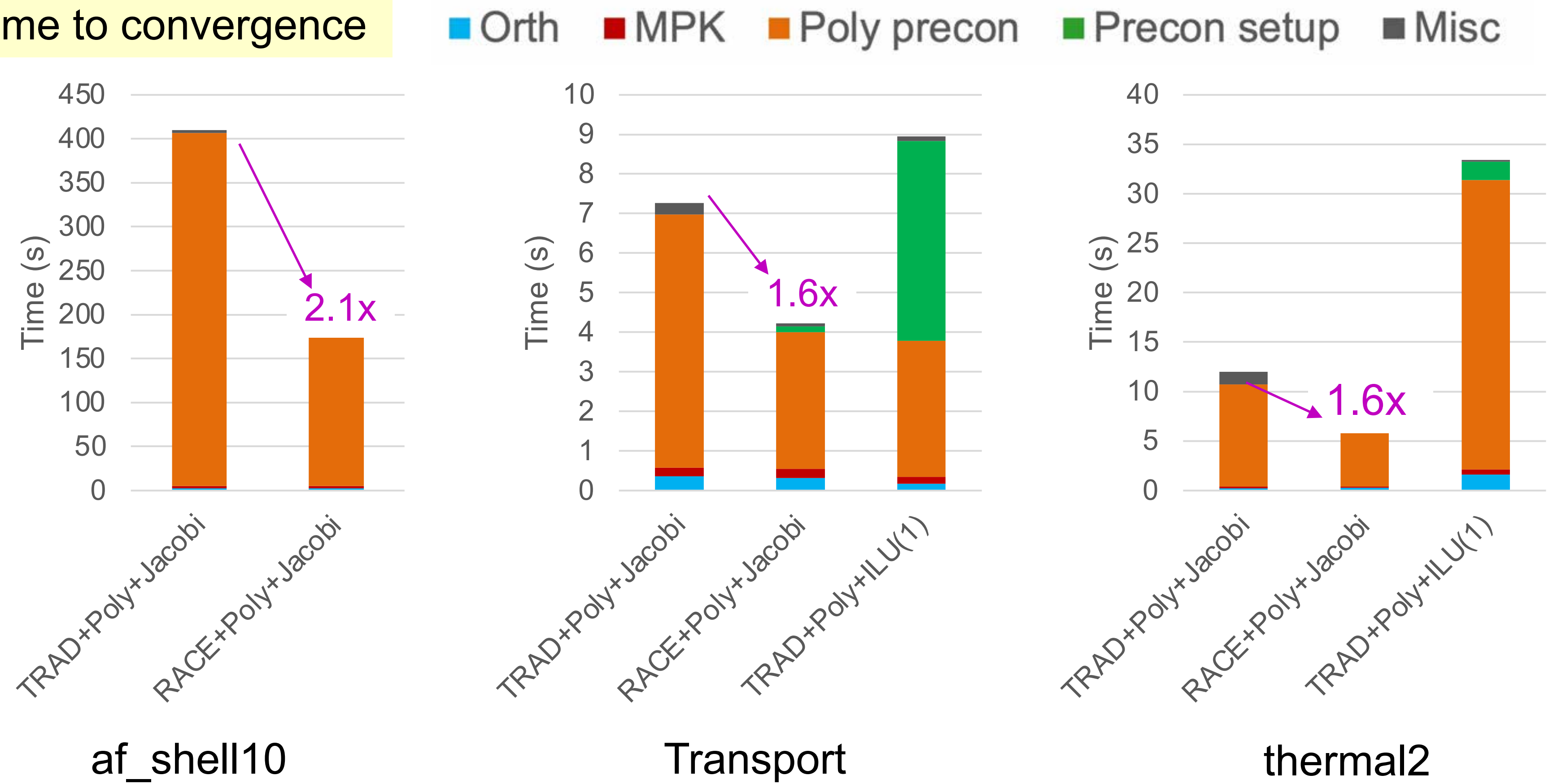
- Higher the degree → more accuracy (in infinite precision)
- But becomes more costly to evaluate

- Run d steps of GMRES with a starting vector (random) b
- Then the solution x in $K(A, b) = span\{b, Ab, A^2b, \ldots\}$
- This implies $x = p(A)b$
- $||r|| = ||b - Ax|| = ||\left(I - Ap(A)\right)b||$ ➔ This residual is what GMRES minimizes

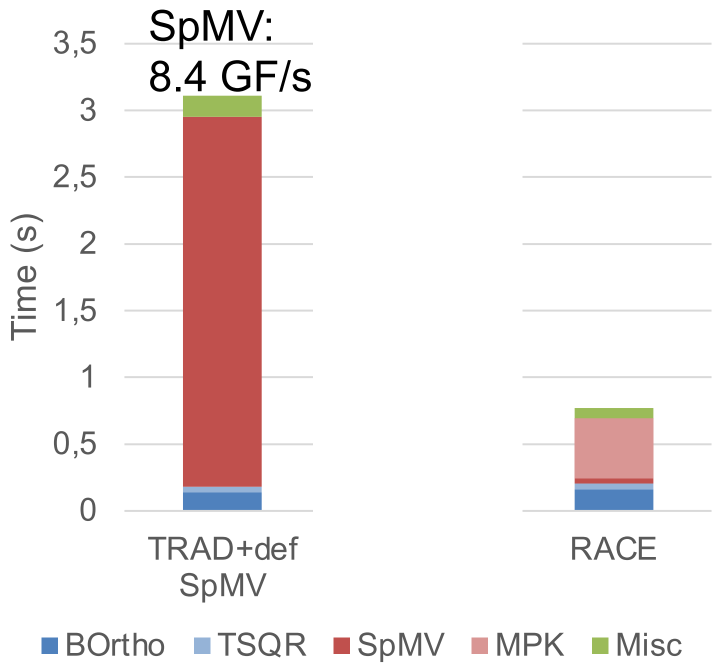
Best thing here is after creating the polynomial $p(A)$,
applying it is just MPK, since $p(A)x = \alpha_0 + \alpha_1 Ax + \alpha_2 A^2 x + \ \ldots$

Application: Polynomial preconditioning

Tiime to convergence



Careful with defaults



SpMV_RLM = 170/6 = 28.3 GF/s

Matrix size = 140 MB

L3+L2 = 104MB

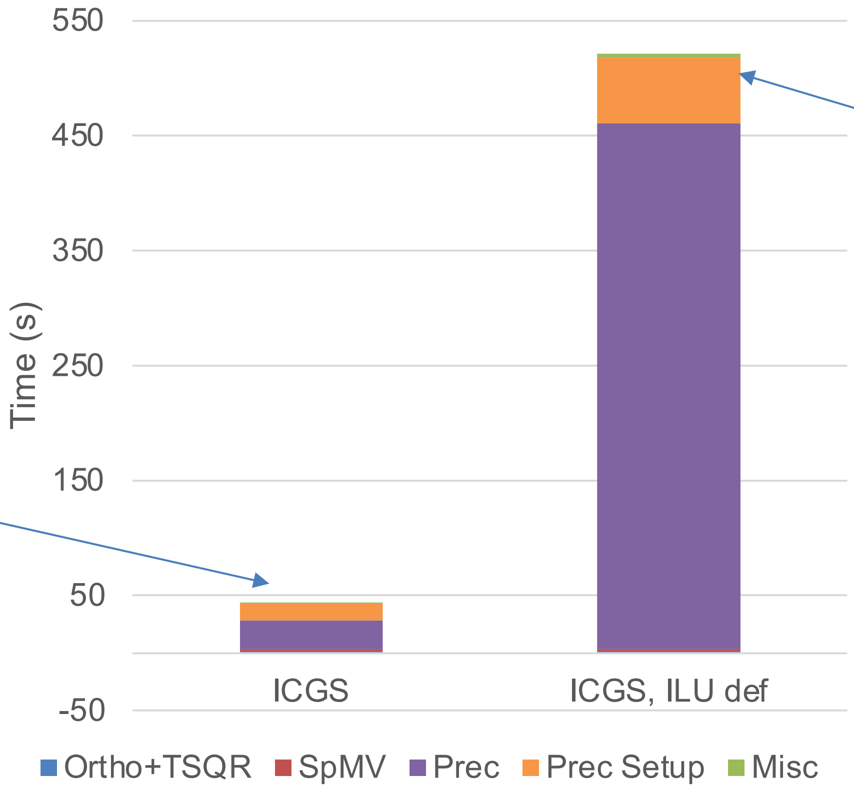
In Kokkos, if Nnz > 1e7 dynamic scheduling

```
if(((A.nnz()>10000000) || use_dynamic_schedule) && !use_static_schedule)
  Kokkos::parallel_for("KokkosSparse::spmv<NoTranspose,Dynamic>",Kokkos::RangePolicy<execution_space, Kokkos::Schedule<Kokkos::Dynamic>>(0, A.numRows()),func);
else
  Kokkos::parallel_for("KokkosSparse::spmv<NoTranspose,Static>",Kokkos::RangePolicy<execution_space, Kokkos::Schedule<Kokkos::Static>>(0, A.numRows()),func);
```

Careful with defaults

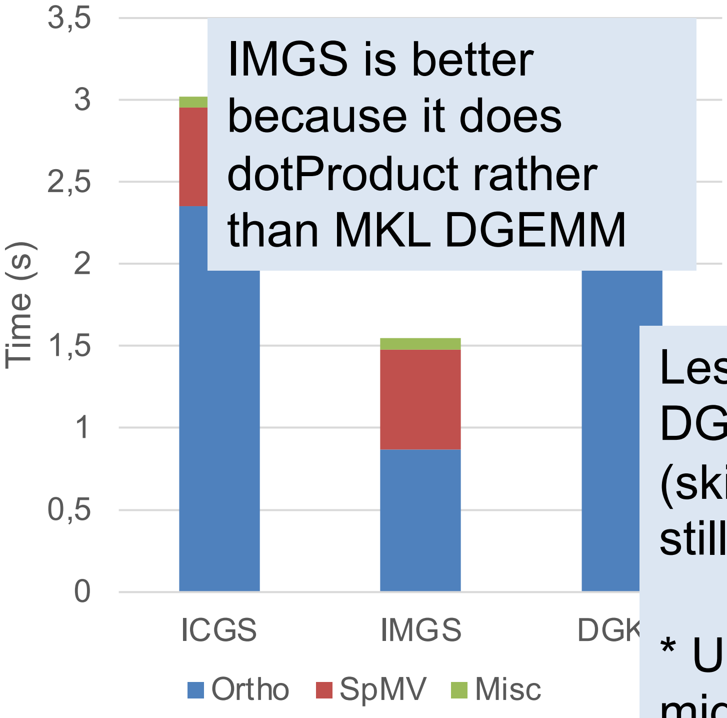
Undocumented backend to SHYLU from Ifpack2

Poly+ILU(1)



Careful with default parameters

Orthogonalization with MKL is not the best

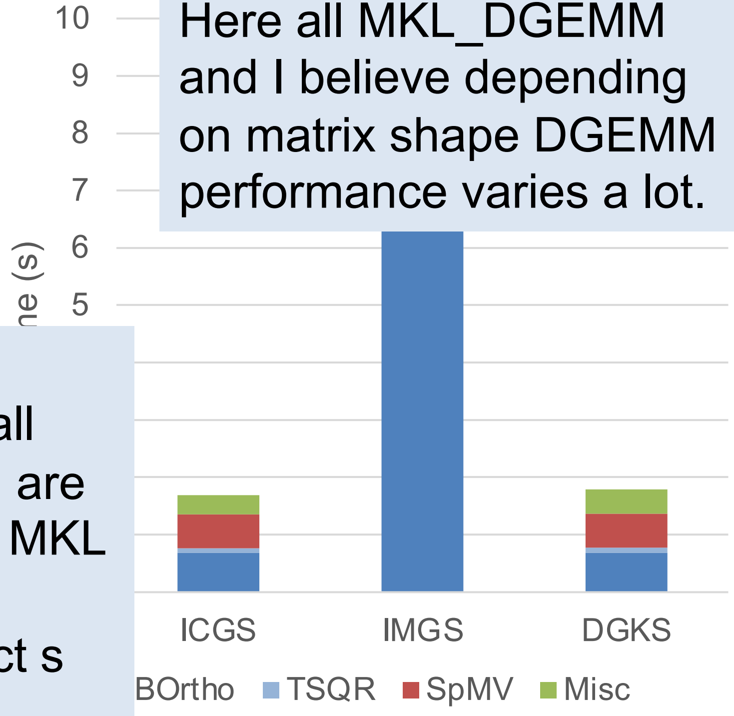


IMGS is better because it does dotProduct rather than MKL DGEMM

TPETRA GMRES

Lesson learned:
DGEMM with small (skinny) matrices are still not optimal in MKL

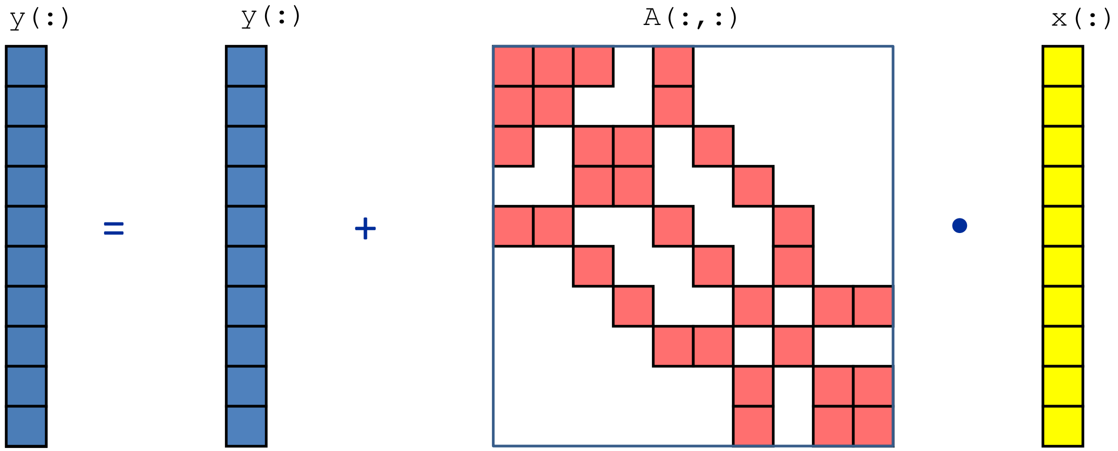
* Using dotProduct s might be better

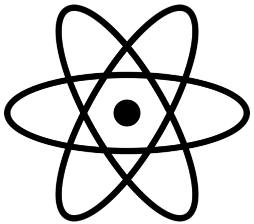


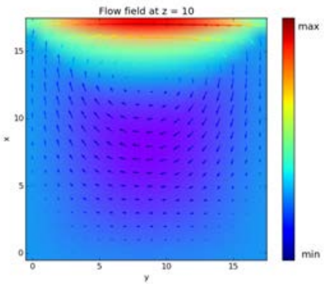
Here all MKL_DGEMM and I believe depending on matrix shape DGEMM performance varies a lot.

TPETRA GMRES S-STEP

SpMV: Multiplication of a sparse matrix (`A`) with a dense vector (`x`)







SpMV: optimizations using RACE

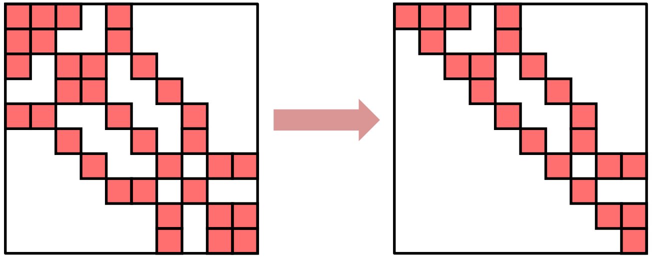
Highly memory bound

↓

Reduce data traffic

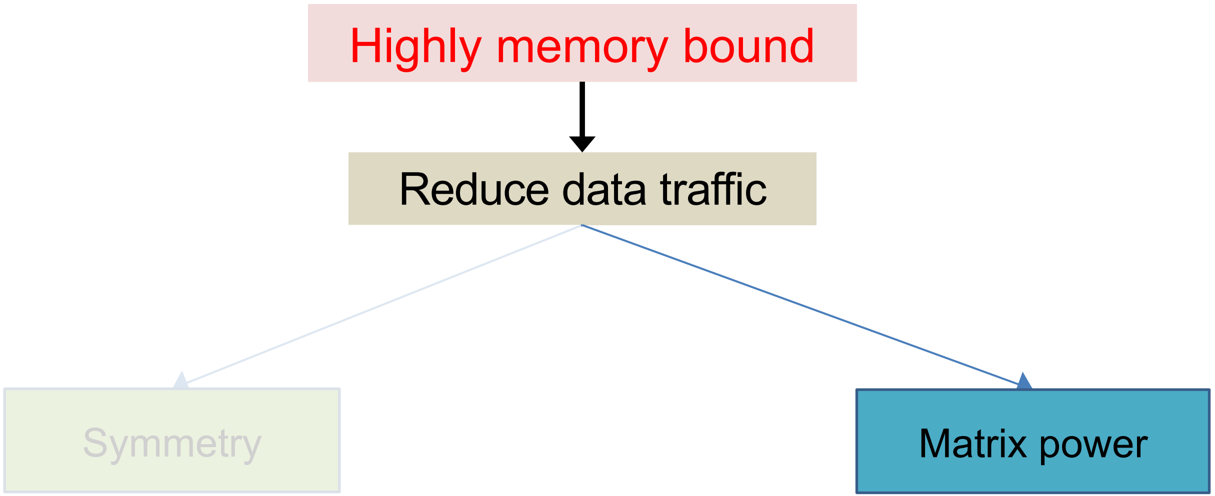
Symmetry | Matrix power

The idea can also be used to efficiently parallelize sparse kernels having dependencies (e.g. Gauss-Seidel, Kaczmarz).



Alappat et al., 2020. A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication. ACM Trans. Parallel Comput. https://doi.org/10.1145/3399732

SpMV: optimizations using RACE

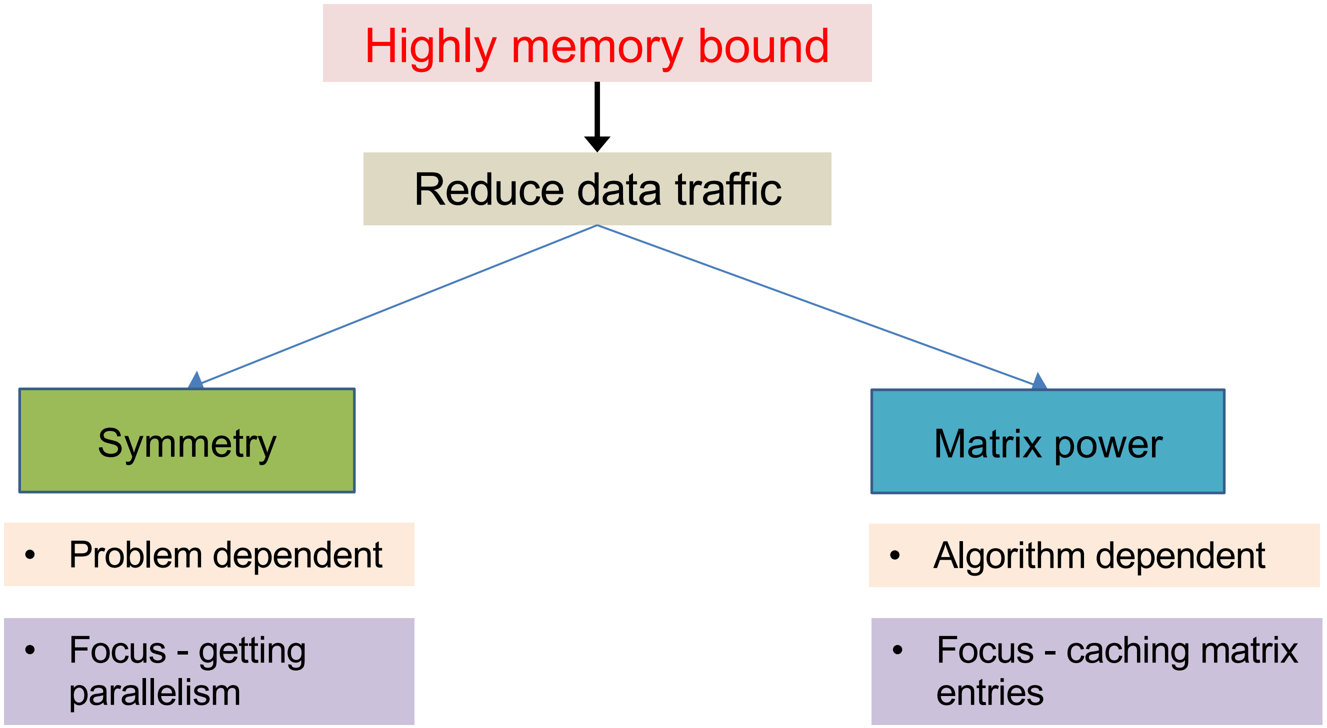


Alappat et al., 2020. Level-based Blocking for Sparse Matrices: Sparse Matrix-Power-Vector Multiplication. Under review. Preprint: https://arxiv.org/abs/2205.01598

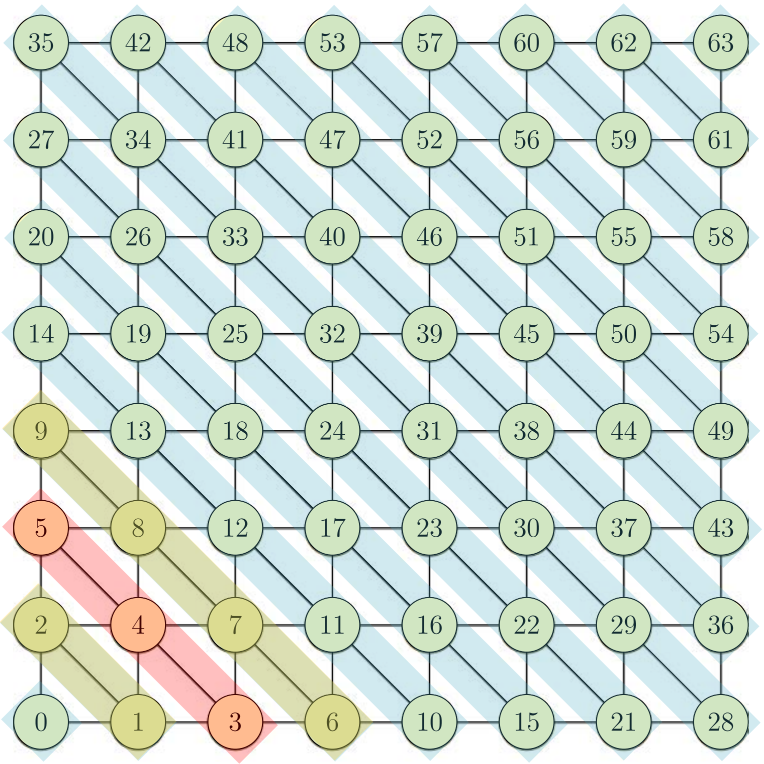
Compute $Ax, A^2x, A^3x, \ldots$

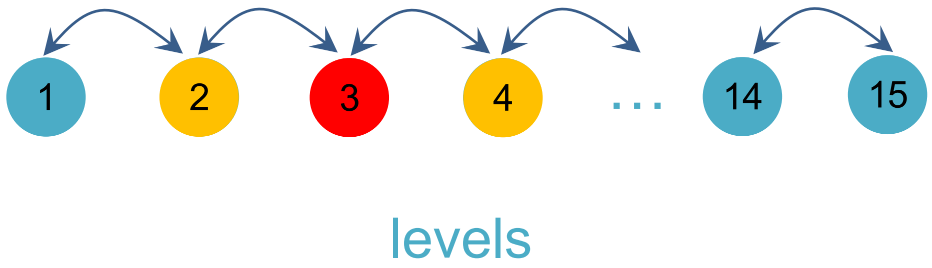
Focus of today's talk: Node-level (CPU) implementation of Matrix power kernels for „large“ matrices

SpMV: optimizations using RACE



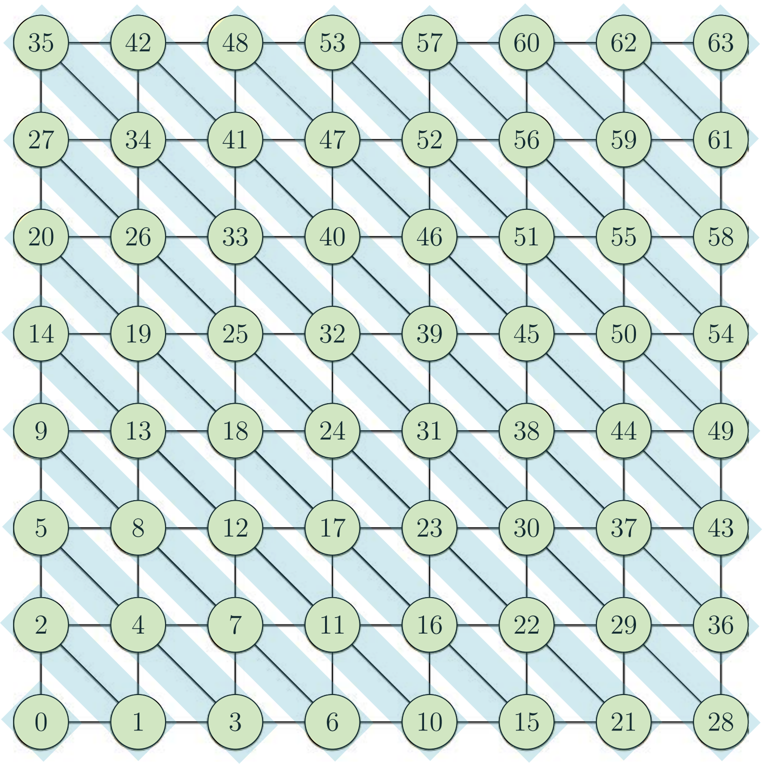
RACE



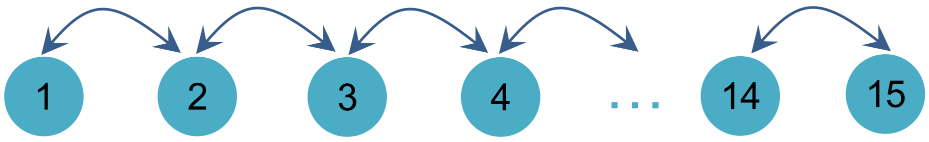


When updating level 3, indirect accesses on levels 3, 2 and 4

RACE



Use Breadth First Search (BFS) Levels



levels

```
do i = 1,L //loop over Levels
 SpMV_CRS(level_ptr[i], level_ptr[i+1])
enddo
```

```
function SpMV_CRS(start, end)
 do i = start, end
  do j = row_ptr(i), row_ptr(i+1) - 1
   y(i) = y(i) + val(j) * y(col_idx(j))
  enddo
 enddo
```