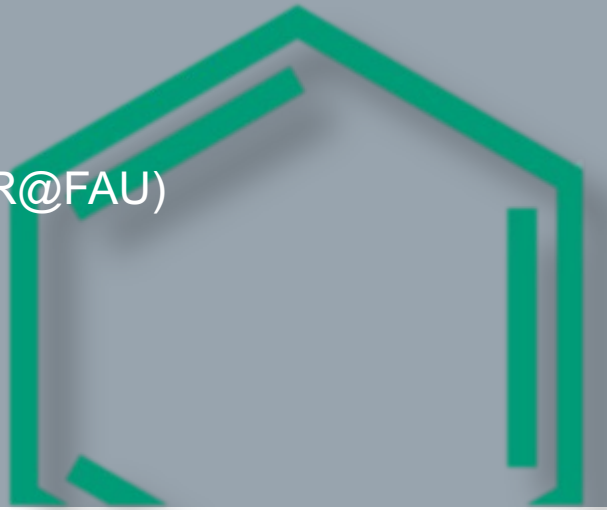
Performance Engineering for Sparse Matrix-Vector Multiplication: Some new ideas for old problems

Christie Alappat, Georg Hager, Gerhard Wellein

Erlangen National High Performance Computing Center (NHR@FAU)

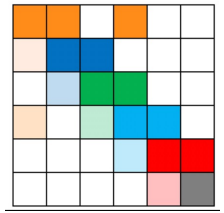
Department of Computer Science

Friedrich-Alexander-Universität Erlangen-Nürnberg



Agenda

- Motivation
- RACE: A different approach to Sparse Matrix-Vector Multiplication (SpMV)
- Parallelization of Symmetric SpMV
- Cache blocking for Matrix Power Kernels
- Conclusion

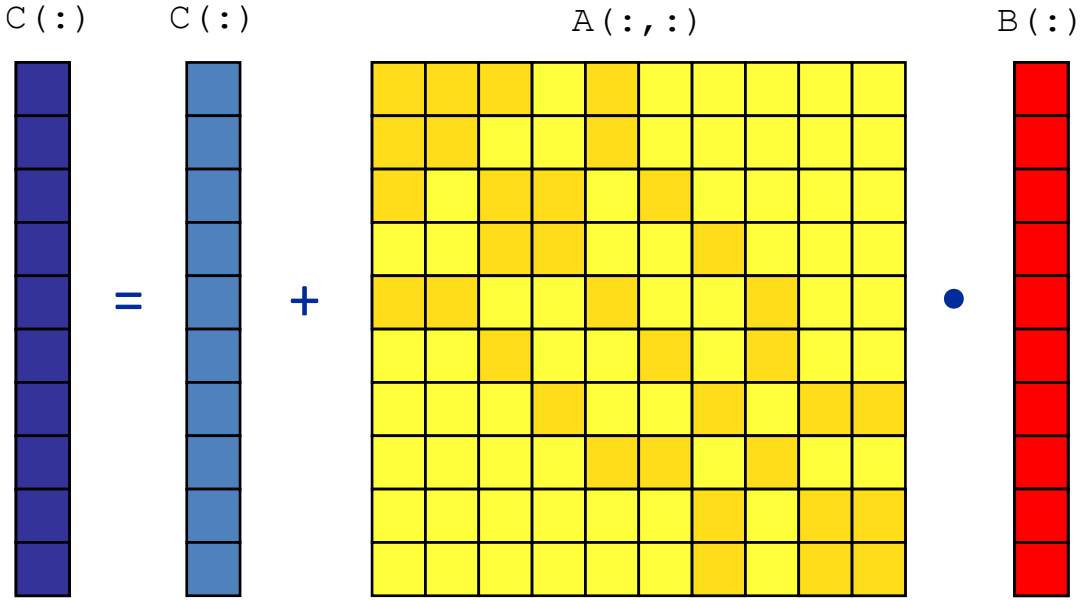


$y = A^p x$

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

Motivation – Sparse Matrix Vector Multiplication

- Easy to parallelize but sparse irregular data structures / accesses
- SpMV Performance ←→ Strongly Memory Bound (high code balance)



Performance of SpMV – Basics: Storage formats

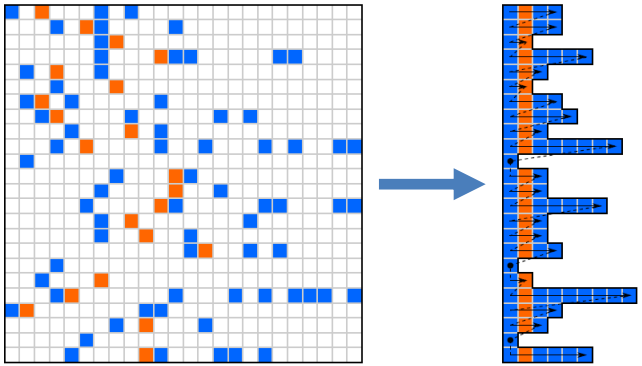
- Store and use non-zero entries only
- Hardware-efficient formats for storing non-zeros in `A(:,:)`
  - **CRS**, COO, DIA,…
  - ELLPACK, hybrid,… → GPU formats, e.g. [1]
  - SELL-C-σ, SELL-P,…
- Best choice ←→ Matrix structure & Architecture

Goals:

- Consecutive access to `A(:,:)`
- Low storage overhea

[1] S. Filippone et al.: Sparse Matrix-Vector Multiplication on GPGPUs. ACM TOMS (2017) https://doi.org/10.1145/3017994

Performance of SpMV – Basics: Performance Model



```
real*8     val[N_nz]
integer*4  col_idx[N_nz]
integer*4  row_ptr[N_r]
real*8     C[N_r]
real*8     B[N_c]
```

A(:,:)

```
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
```

$$B_{C,min} = \frac{12\,N_{nz} + 20\,N_r + 8\,N_c}{2\,N_{nz}}\,\frac{B}{F}$$

Nonzeros per row ($N_{nzr} = {}^{N_{nz}}/_{N_r}$) or column ($N_{nzc} = {}^{N_{nz}}/_{N_c}$)

Lower bound for code balance: $B_{C,min} \geq 6\,\frac{B}{F} \rightarrow I_{max} \leq \frac{1}{6}\,\frac{F}{B}$

Motivation – Sparse Matrix Vector Multiplication

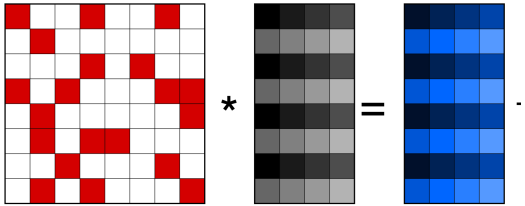
- Improve code balance of **SpMV-algorithms**
  - Kernel fusion, e.g. HPCG:

Algorithm 2. HPCG

```
1:  while k ≤ iter & r_norm/r_0 > tol do
2:      z = MG(A,r)                    -> MG sweep
3:      oldrtz = rtz
4:      rtz = ⟨r,z⟩                    -> DOT
5:      β = rtz/oldrtz
6:      p = β * p + z                  -> WAXPBY
7:      Ap = A * p                     -> SpMPV
8:      pAp = ⟨p,Ap⟩                   -> DOT
9:      α = rtz/pAp
10:     x = x + α * p                  -> WAXPBY
11:     r = r − α * Ap                 -> WAXPBY
12:     r_norm = ⟨r,r⟩                 -> DOT
13:     r_norm = sqrt(r_norm)
14:     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): $A = A^t$
  - Efficient parallelisation ←→ Write conflicts
  - 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

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

SpMV – graph traversal – RACE

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

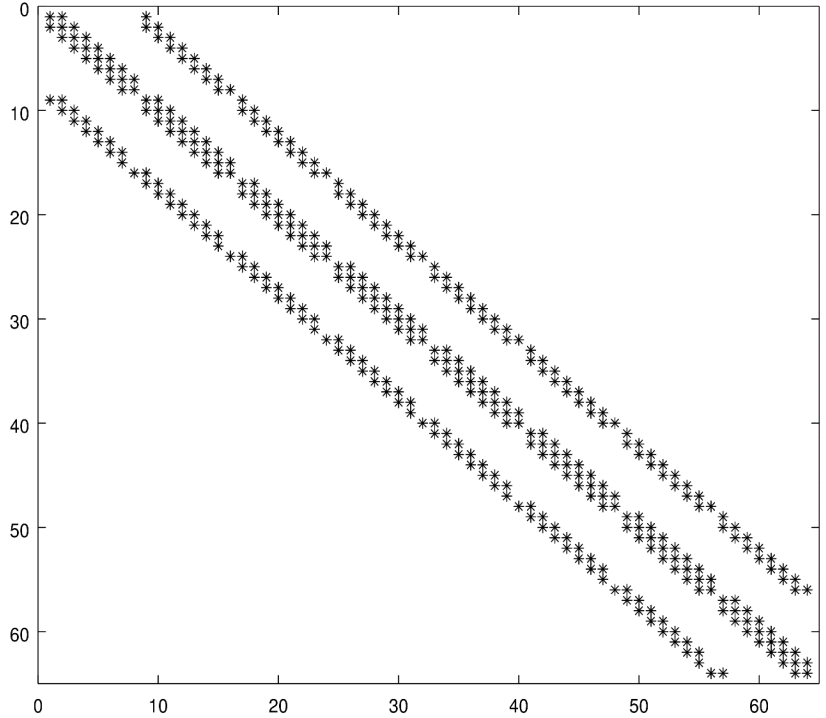
- RACE – **graph-based view**: row ←→ vertex & non-zero entry ←→ edge

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

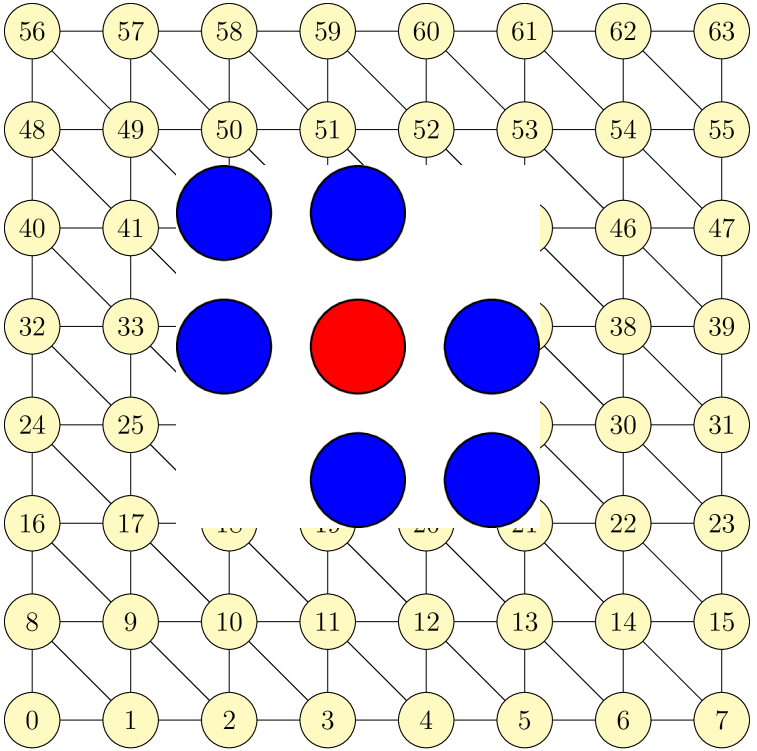
$$y_u = \sum_{v \in N(u)} A_{u,v} x_v \ . \qquad (2)$$

Sample Stencil Matrix and its graph representation

Symmetric Matrix (Stencil)

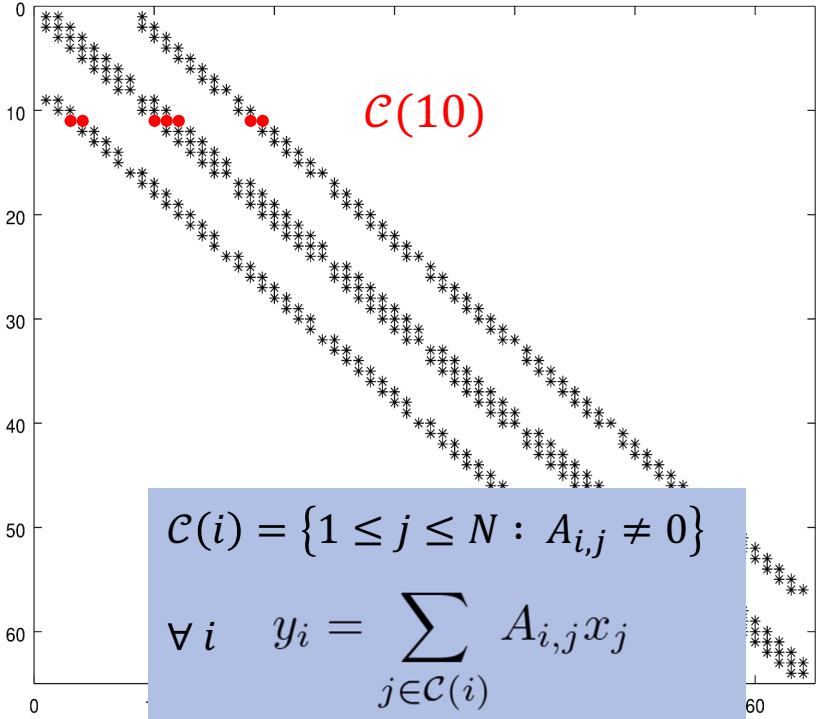


Undirected Graph



Sample Stencil Matrix and its graph representation

Symmetric Matrix (Stencil)



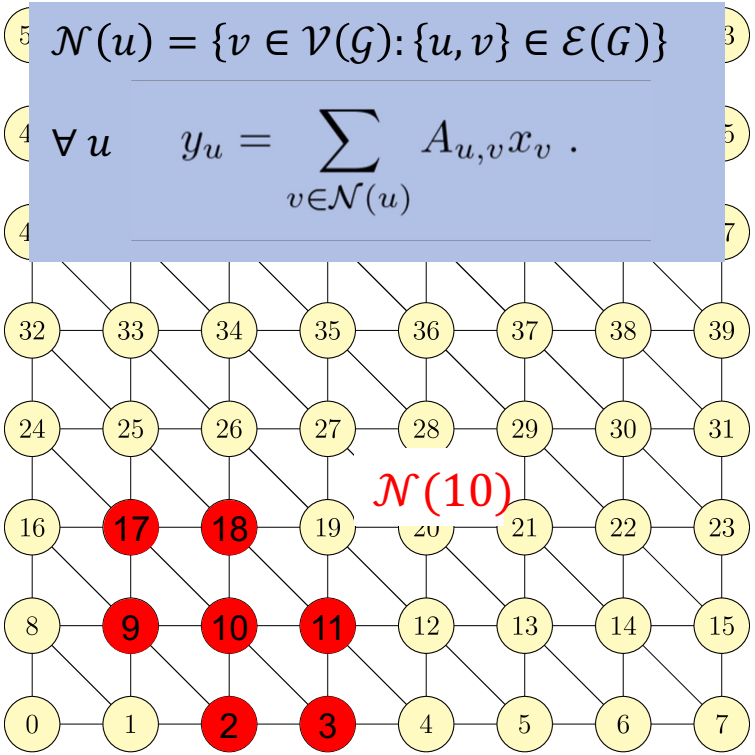
$$\mathcal{C}(i) = \{1 \le j \le N : A_{i,j} \neq 0\}$$

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

Undirected Graph

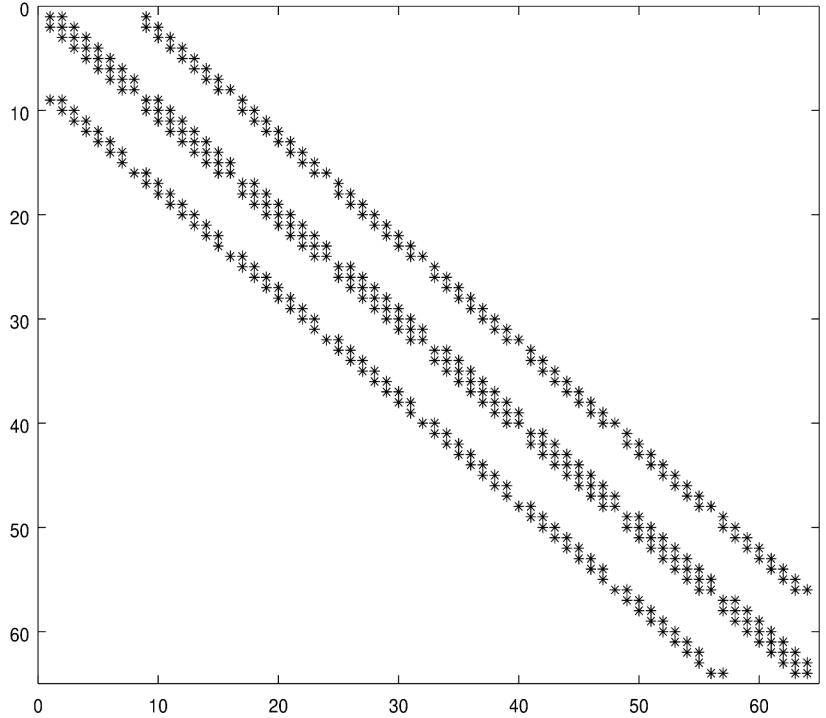
$$\mathcal{N}(u) = \{v \in \mathcal{V}(\mathcal{G}) : \{u, v\} \in \mathcal{E}(G)\}$$

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

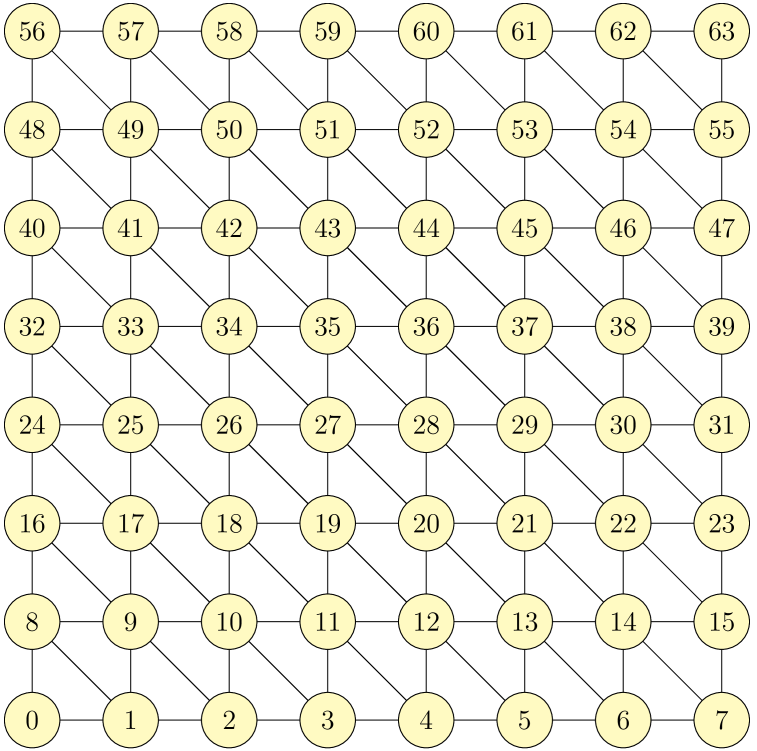


Sample Stencil Matrix and its graph representation

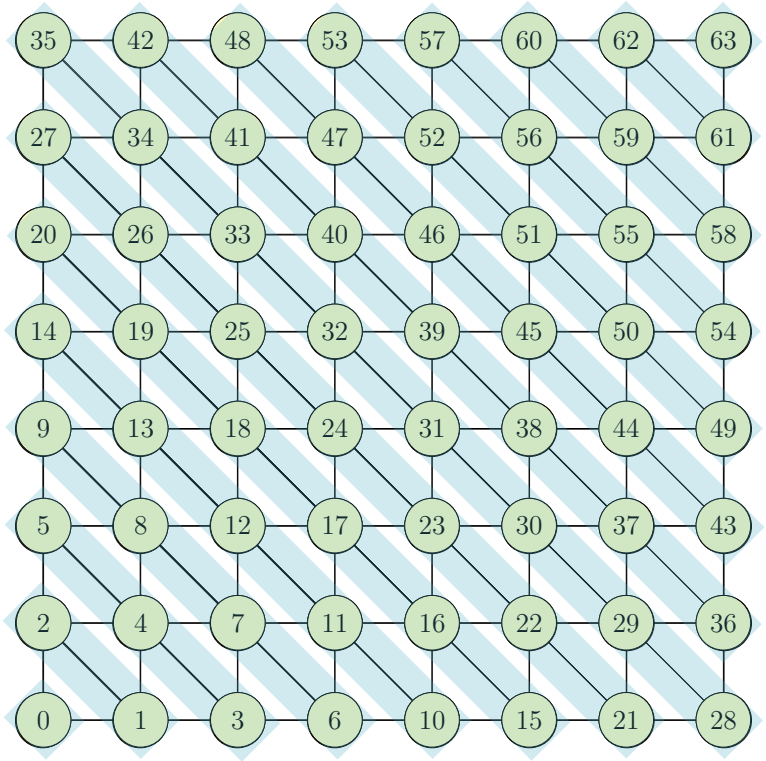
Symmetric Matrix (Stencil)



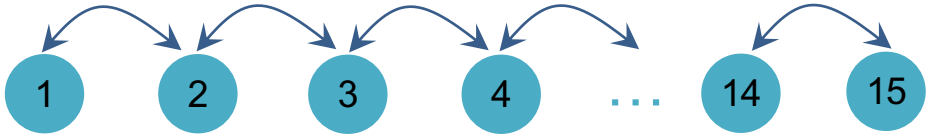
Undirected Graph



RACE: Get BFS-levels



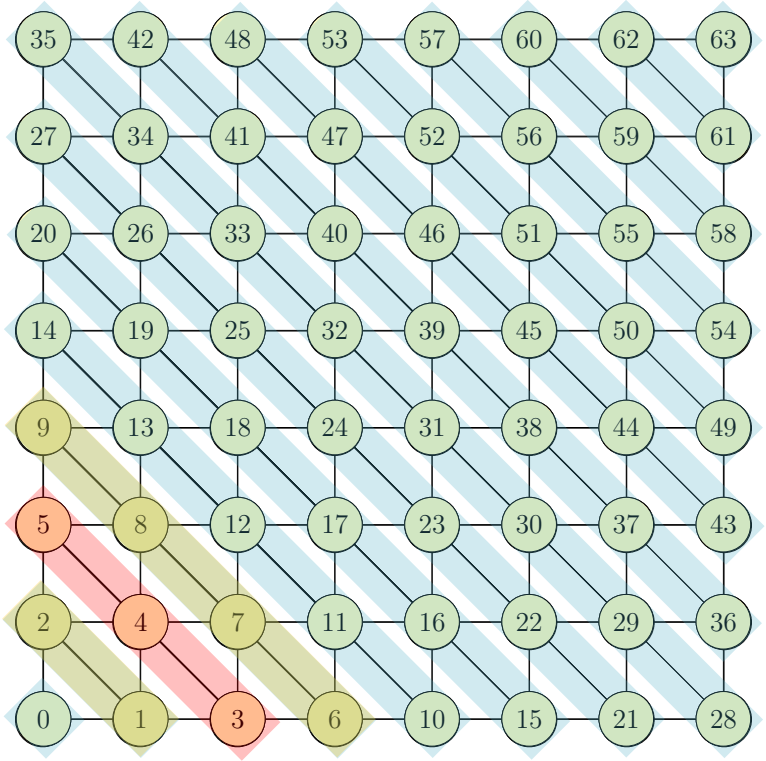
- Get levels of a Breadth-First-Search (BFS)
- Reorder vertices → consecutive witin level



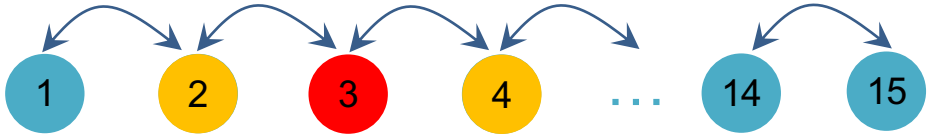
levels

- Dependencies only between neighboring levels!

RACE: SpMV - Dependencies and Implementation



Update 3 → indirect access to 3, 2 and 4

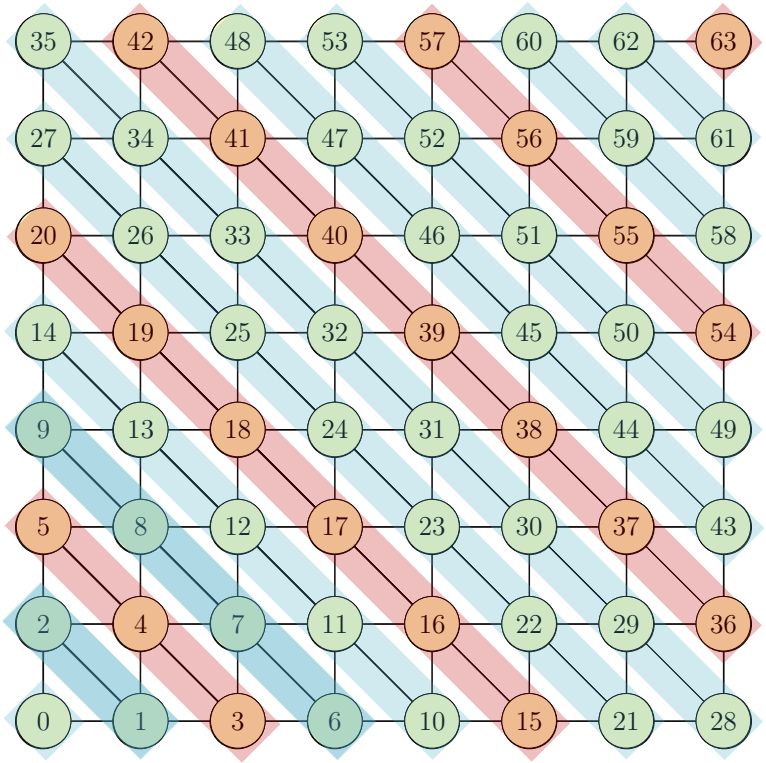


levels

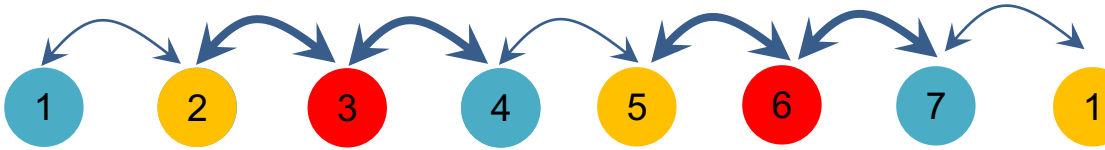
```
do i = 1,L //loop over Levels
 SpMV_CRS(level_ptr[i], level_ptr[i+1])
enddo
```

```
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) * y(col_idx(j))
  enddo
 enddo
```

RACE: SymmSpMV - Parallelization



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



levels

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

NHR FAU

FAU Friedrich-Alexander-Universität Erlangen-Nürnberg

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

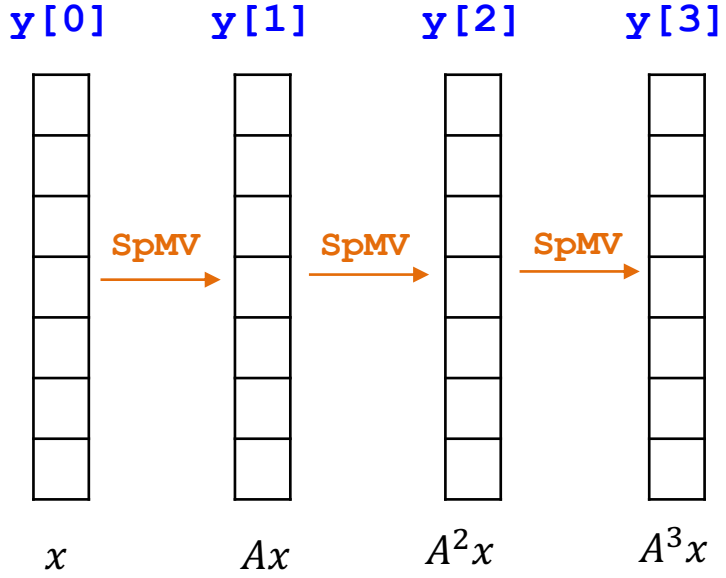
Christie Alappat, Georg Hager, Olaf Schenk, *Senior Member, IEEE*, and Gerhard Wellein

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

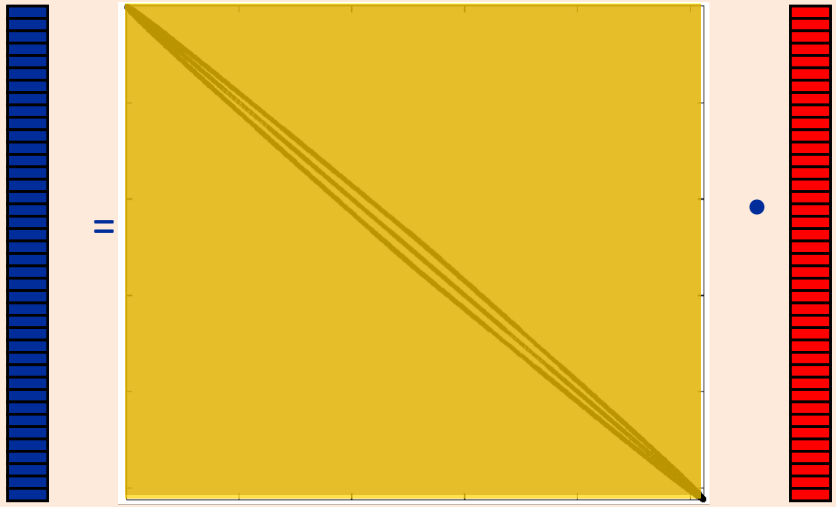
[1]Huber et al., 2021. Graph-based multi-core higher-order time integration of linear autonomous partial differential equations. J. Comput. Sci. DOI:10.1016/j.jocs.2021.101349

[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

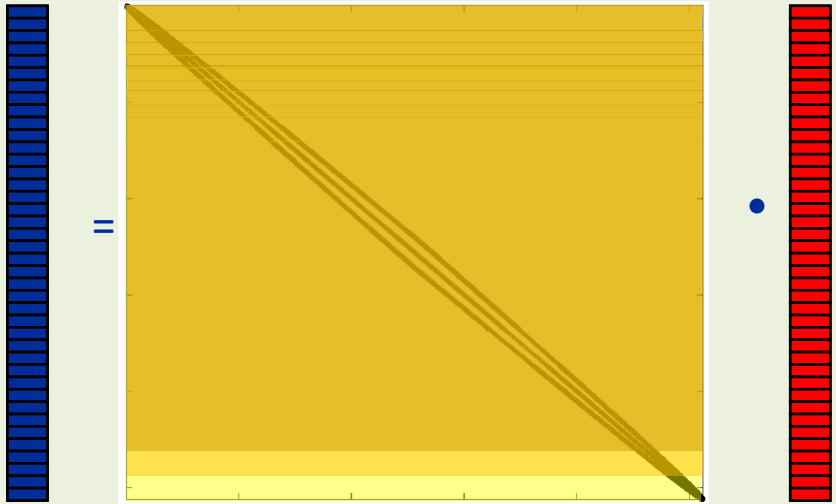
Calculate $y = A^3 x$

TRAD approach



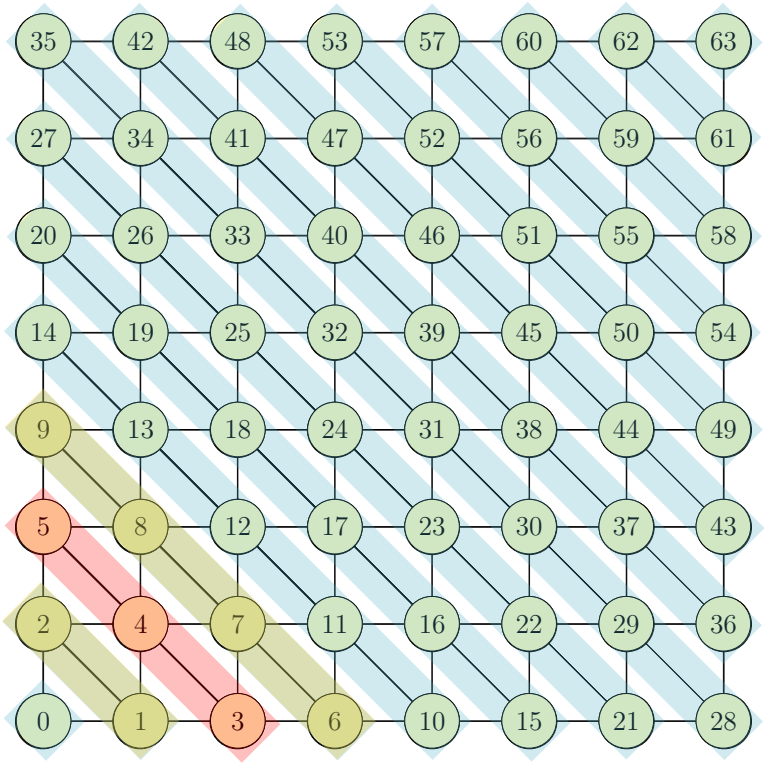
Matrix accessed 3 times from memory

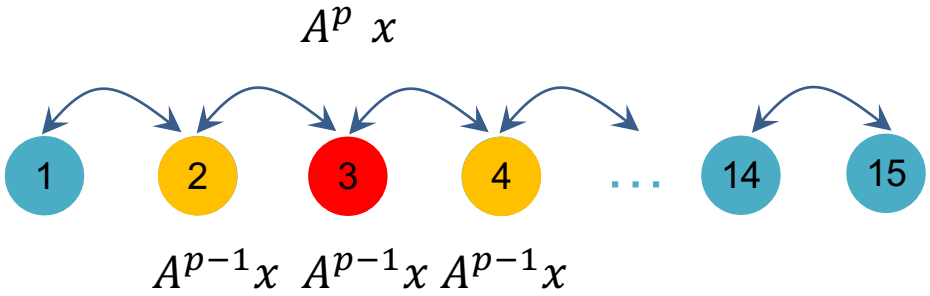
RACE approach



Matrix accessed 1 time from memory

How to do that in general for sparse matrices?

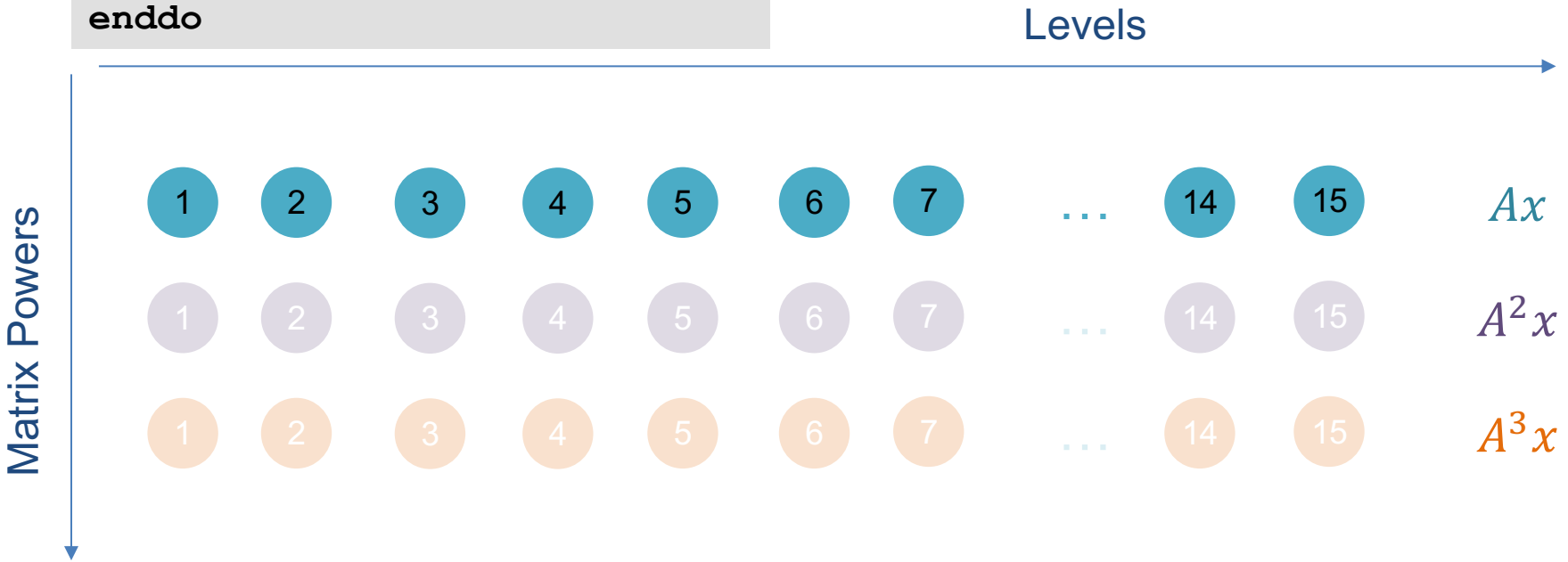

35 42 48 53 57 60 62 63
27 34 41 47 52 56 59 61
20 26 33 40 46 51 55 58
14 19 25 32 39 45 50 54
9 13 18 24 31 38 44 49
5 8 12 17 23 30 37 43
2 4 7 11 16 22 29 36
0 1 3 6 10 15 21 28


$A^p\ x$
1
2
3
4
...
14
15
$A^{p-1}x$ $A^{p-1}x$ $A^{p-1}x$

RACE – Level traversal and matrix powers

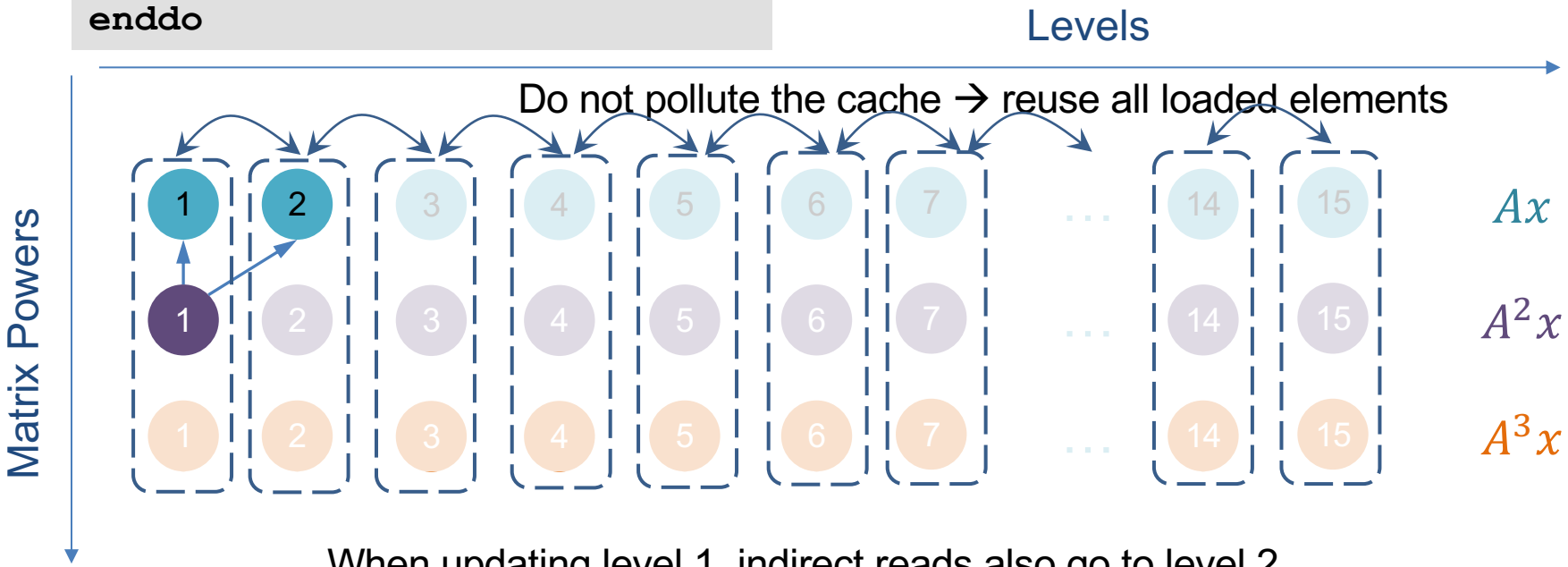
```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```

No cache blocking!



RACE – Level traversal and matrix powers

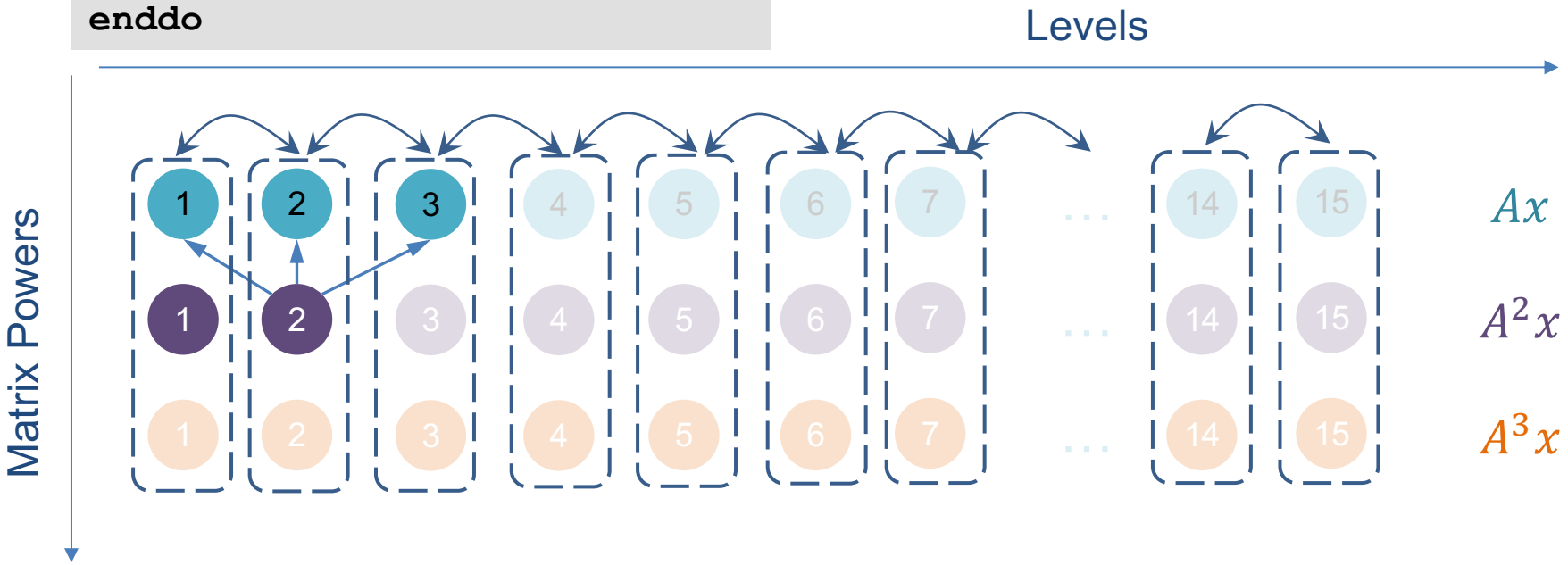
```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



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

RACE – Level traversal and matrix powers

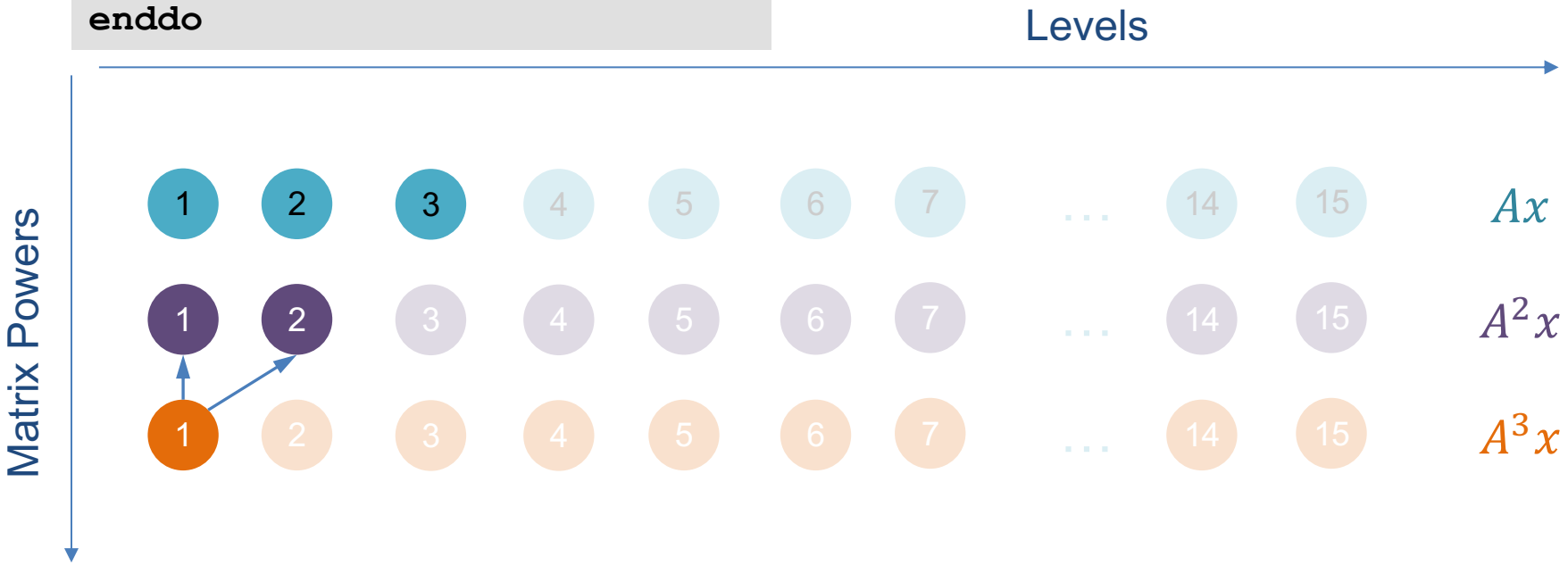
```
do k = 1, p
 y(:, k) = SpMV(A, y(:,k-1))
enddo
```



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

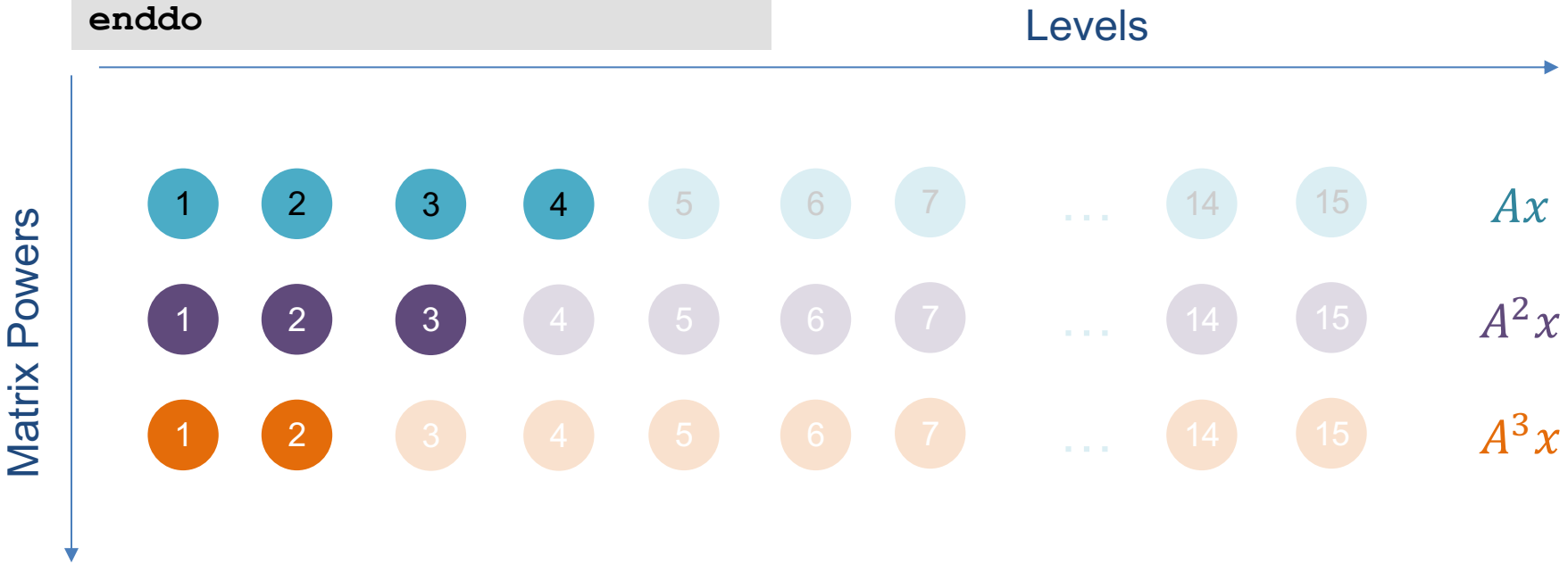
RACE – Level traversal and matrix powers

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



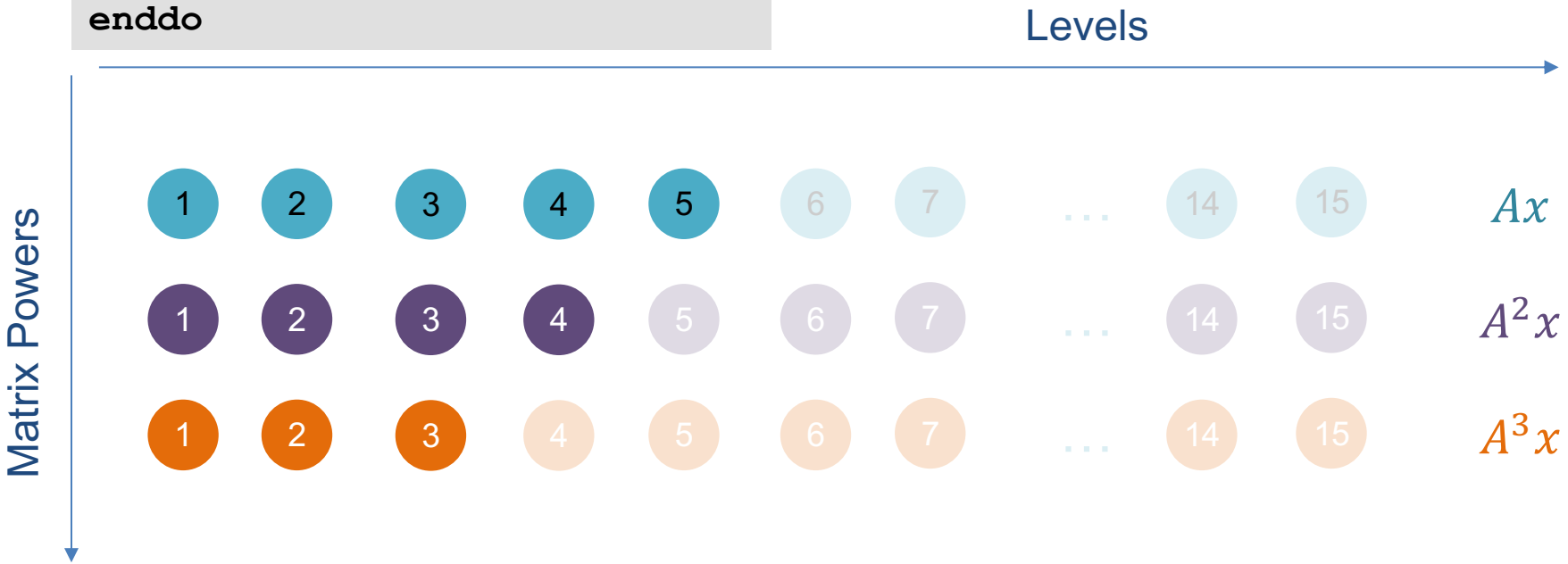
RACE – Level traversal and matrix powers

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



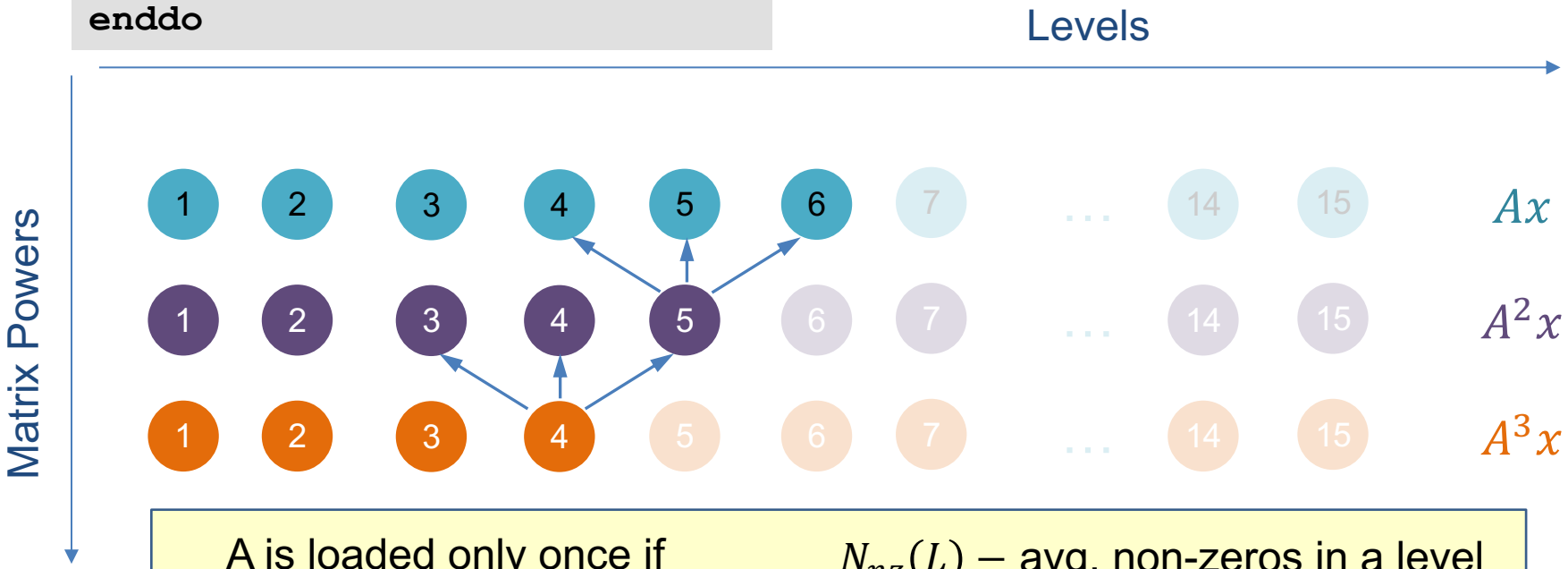
RACE – Level traversal and matrix powers

```
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

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

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

C – cache size

RACE: MPK Pseudocode

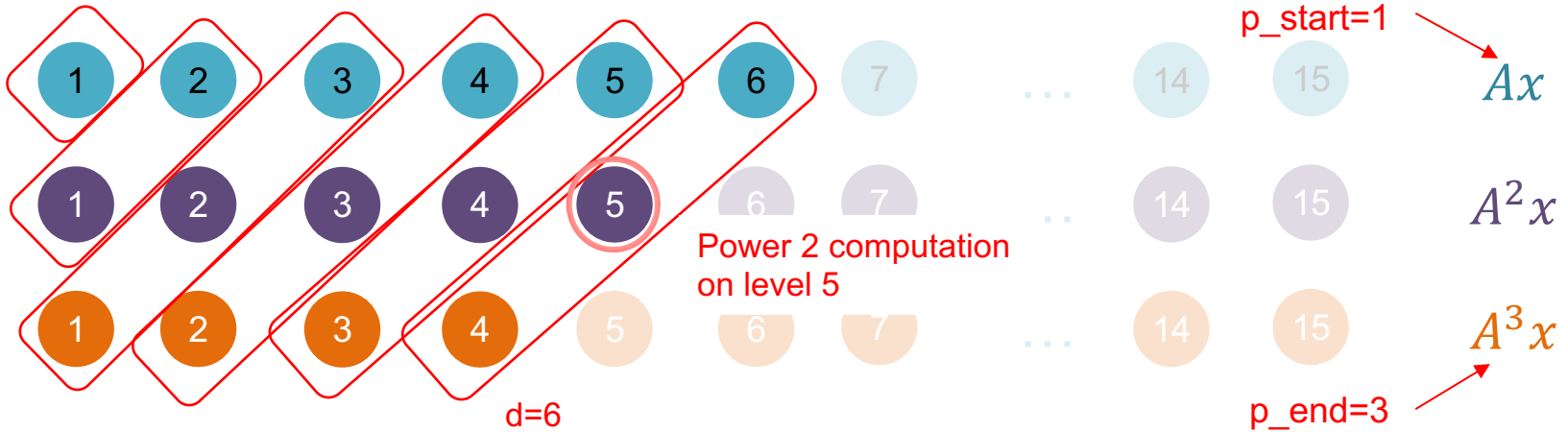
```
do d in 1:L+p-1
 p_start = max(1, d-(L-1))
 p_end = min(d, p)
 do k in p_start:p_end
  l=(d-k+1)
  y(:, k) = SpMV(A(j,:), y(:,k-1), level_ptr[l]:level_ptr[l+1])
 enddo
enddo
```

← Traverse along diagonal

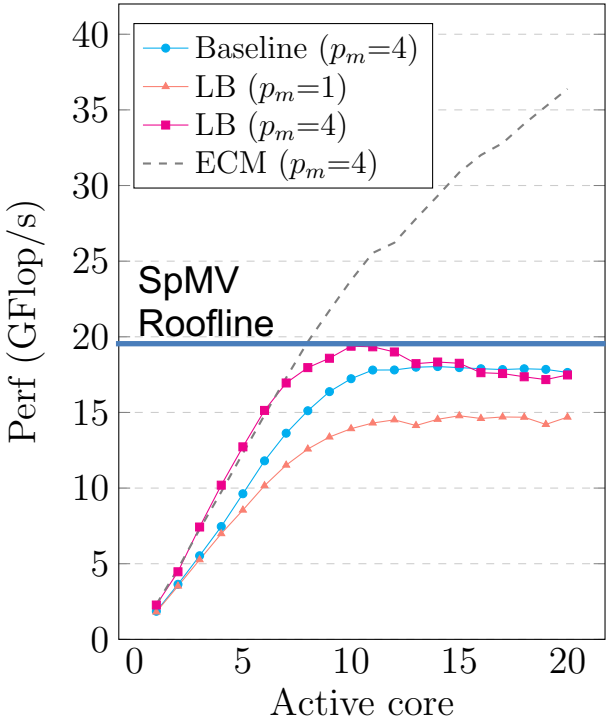
← All powers in diagonal

← Power k computation on level l

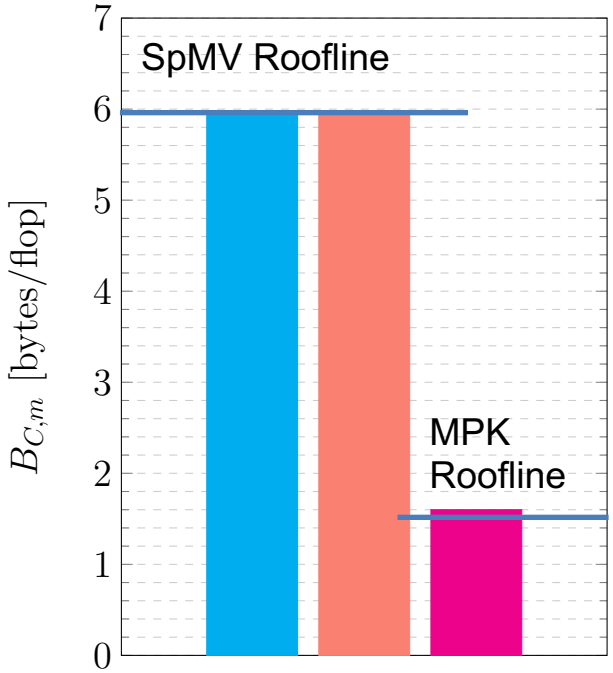
OpenMP parallel SpMV on all vertices in level l



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

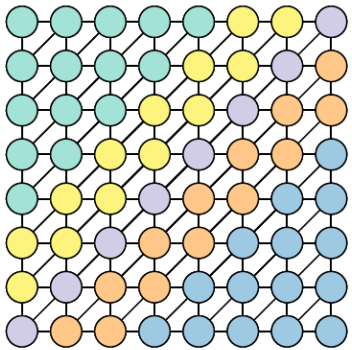
- Scheme seems to work (reduces data traffic) – at least for **`pwtk`**
- But: Performance ☹ !!!!

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

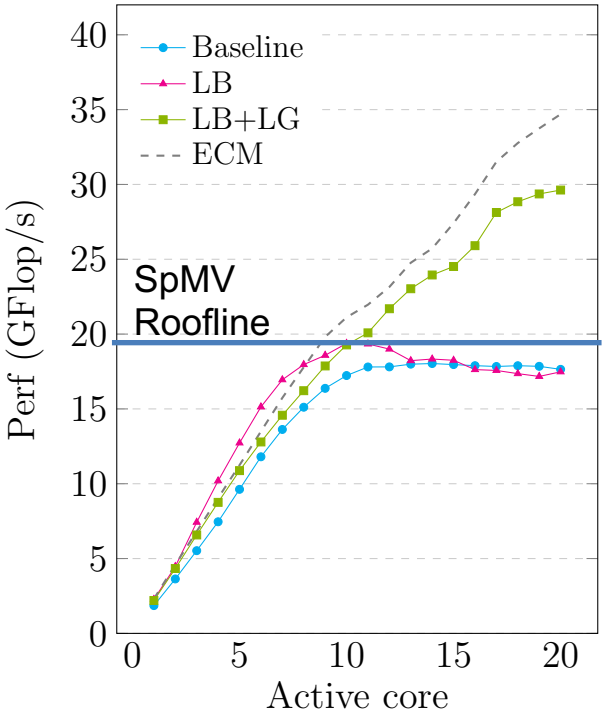
  → Frequent thread syncronisations!

  Reason: After each level threads sync!

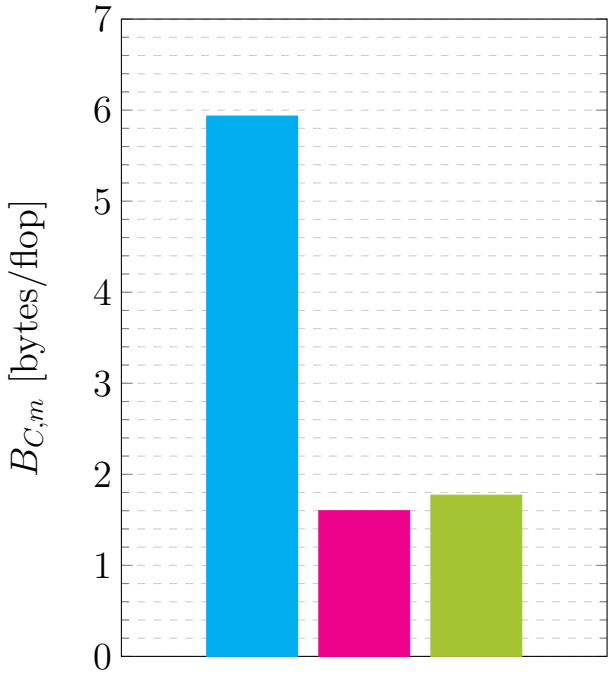
  Measures:
  → Reduce #levels by level aggregation („LG“)
  → Global sync. replaced by point-to-point sync. („p2p“)



RACE MPK – LG optimization



Performance

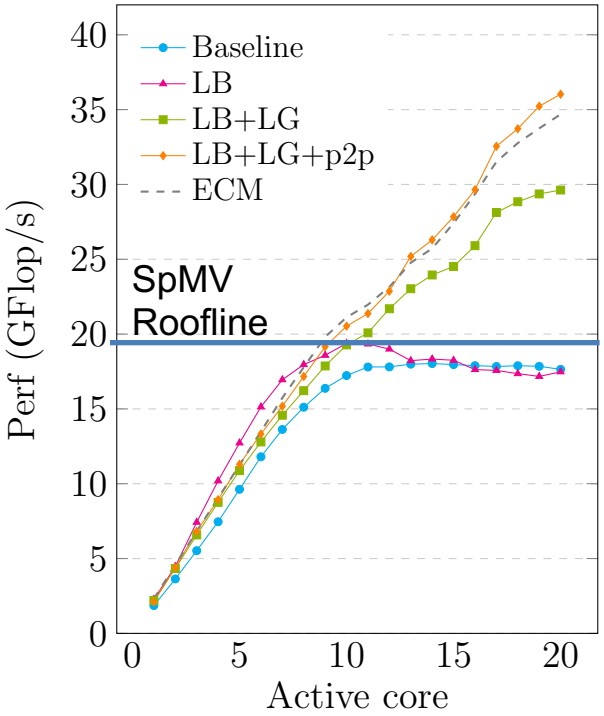


Memory traffic

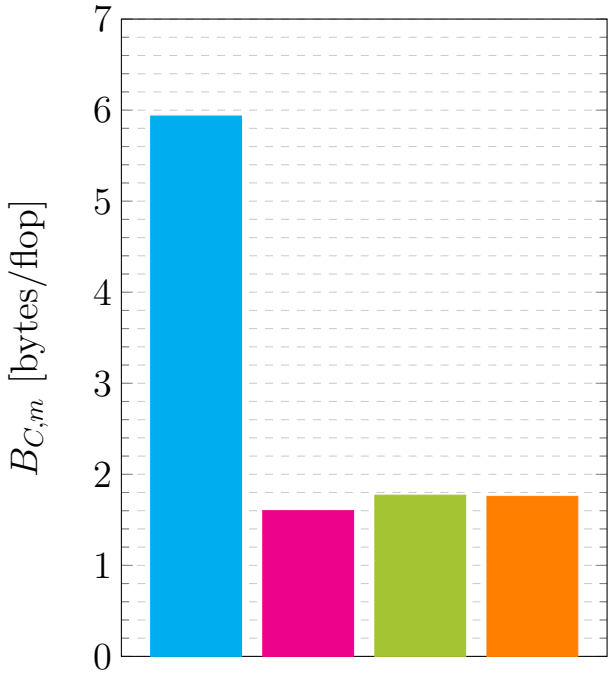
Intel Xeon Gold 6248
- 1 Socket (20c)

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

RACE MPK – LG+p2p optimization



Performance

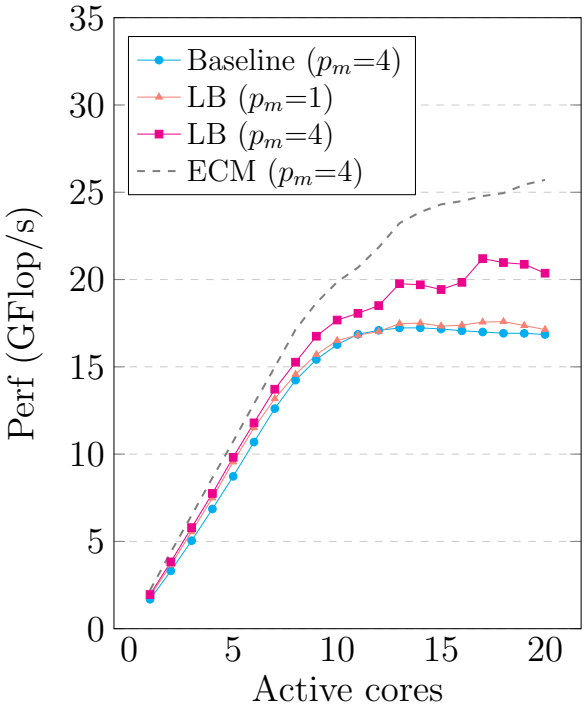


Memory traffic

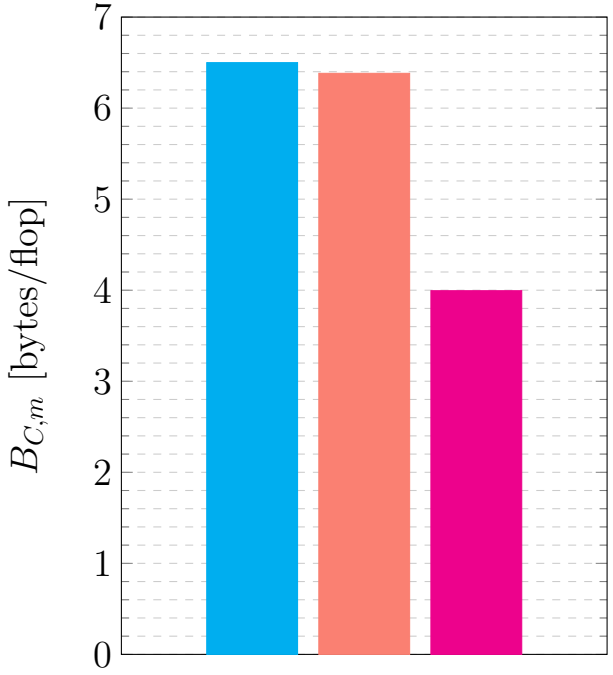
Intel Xeon Gold 6248
- 1 Socket (20c)

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)

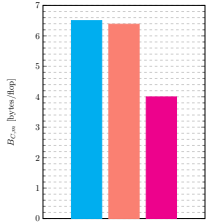
`Flan_1565` matrix
- N_r = 1,564,794
- N_{nz} = 117,406,044

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:



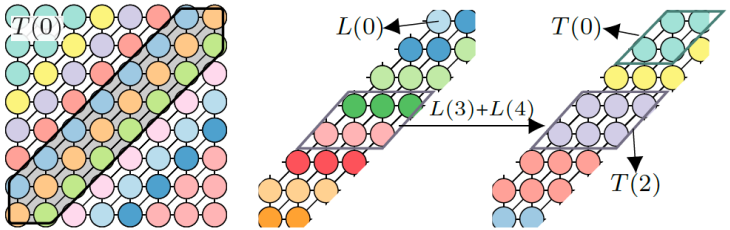
Few large levels → do not fit in cache!

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

Too big!!!

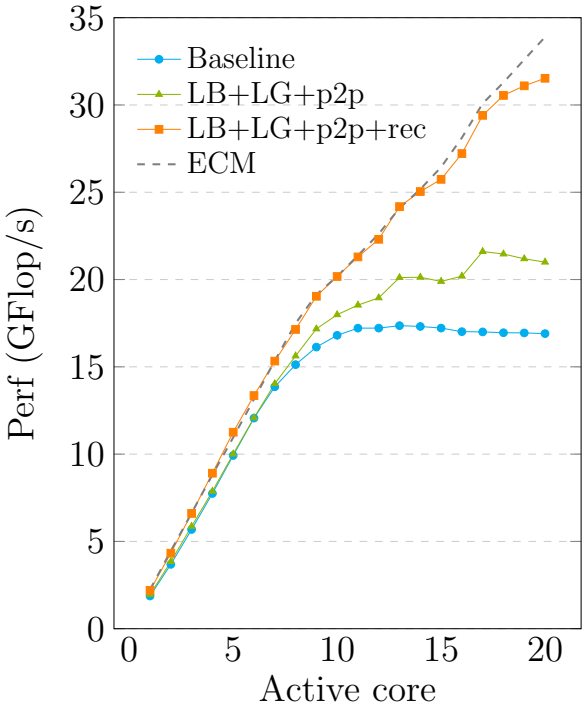
Counter - Measure:

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

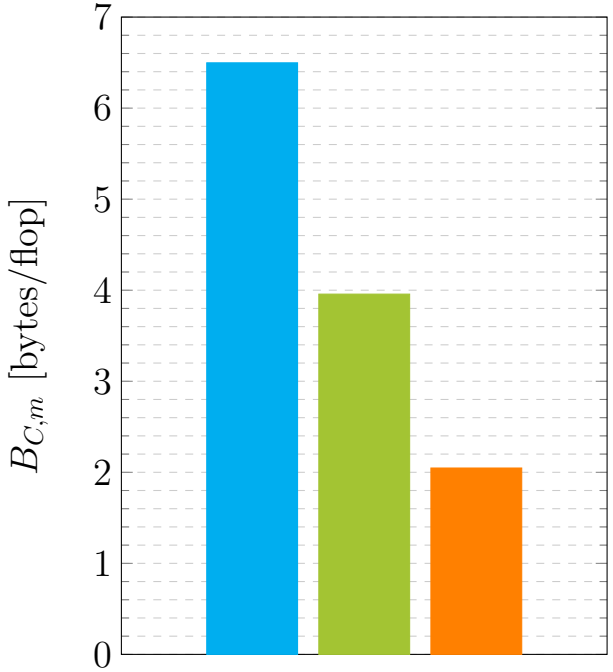


(a) Graph. (b) Levels. (c) Level groups.

RACE MPK – rec



Performance

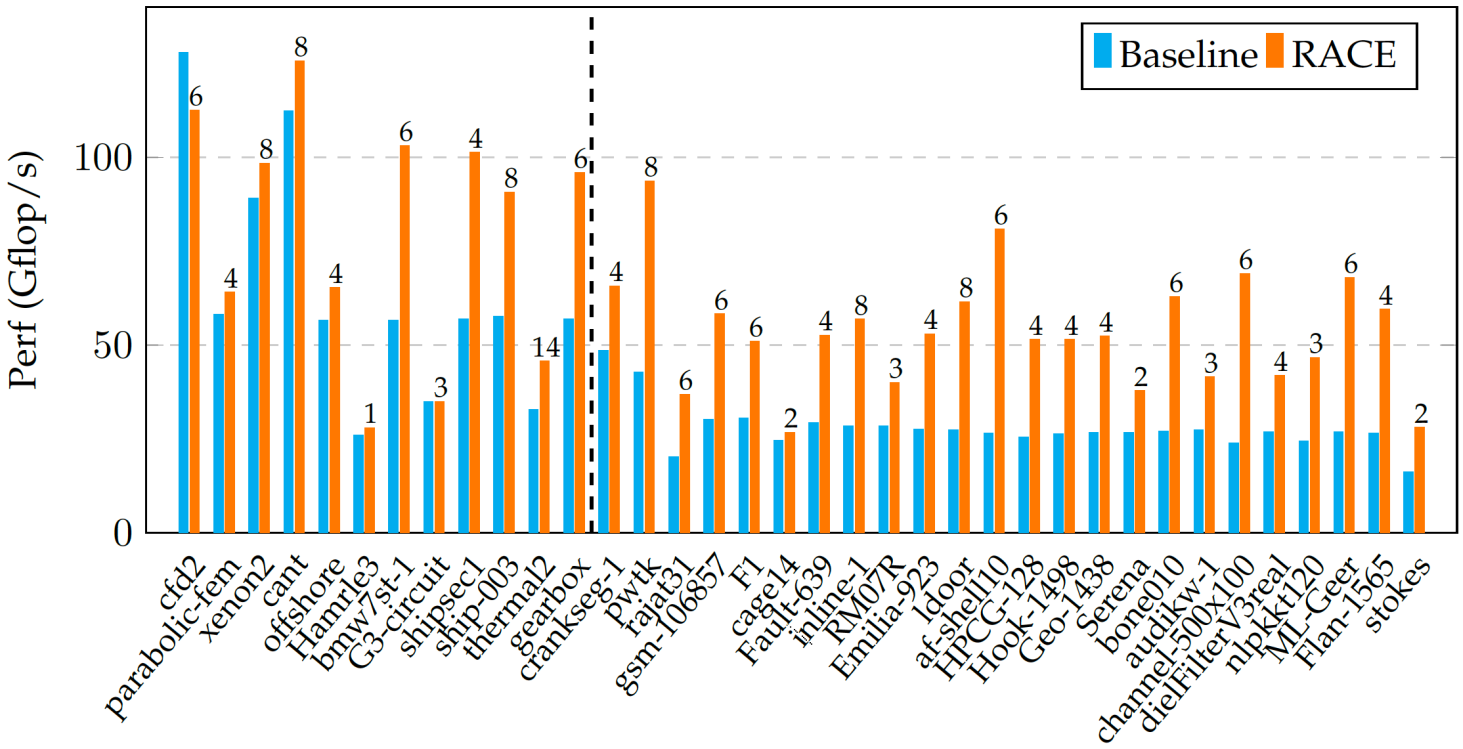


Memory traffic

Intel Xeon Gold 6248
- 1 Socket (20c)

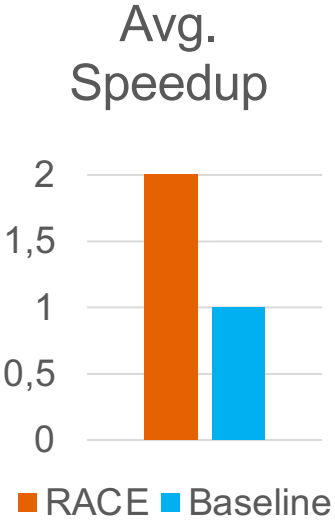
`Flan_1565` matrix
- N_r = 1,564,794
- N_{nz} = 117,406,044

Matrix power kernel: Performance – Intel Ice Lake

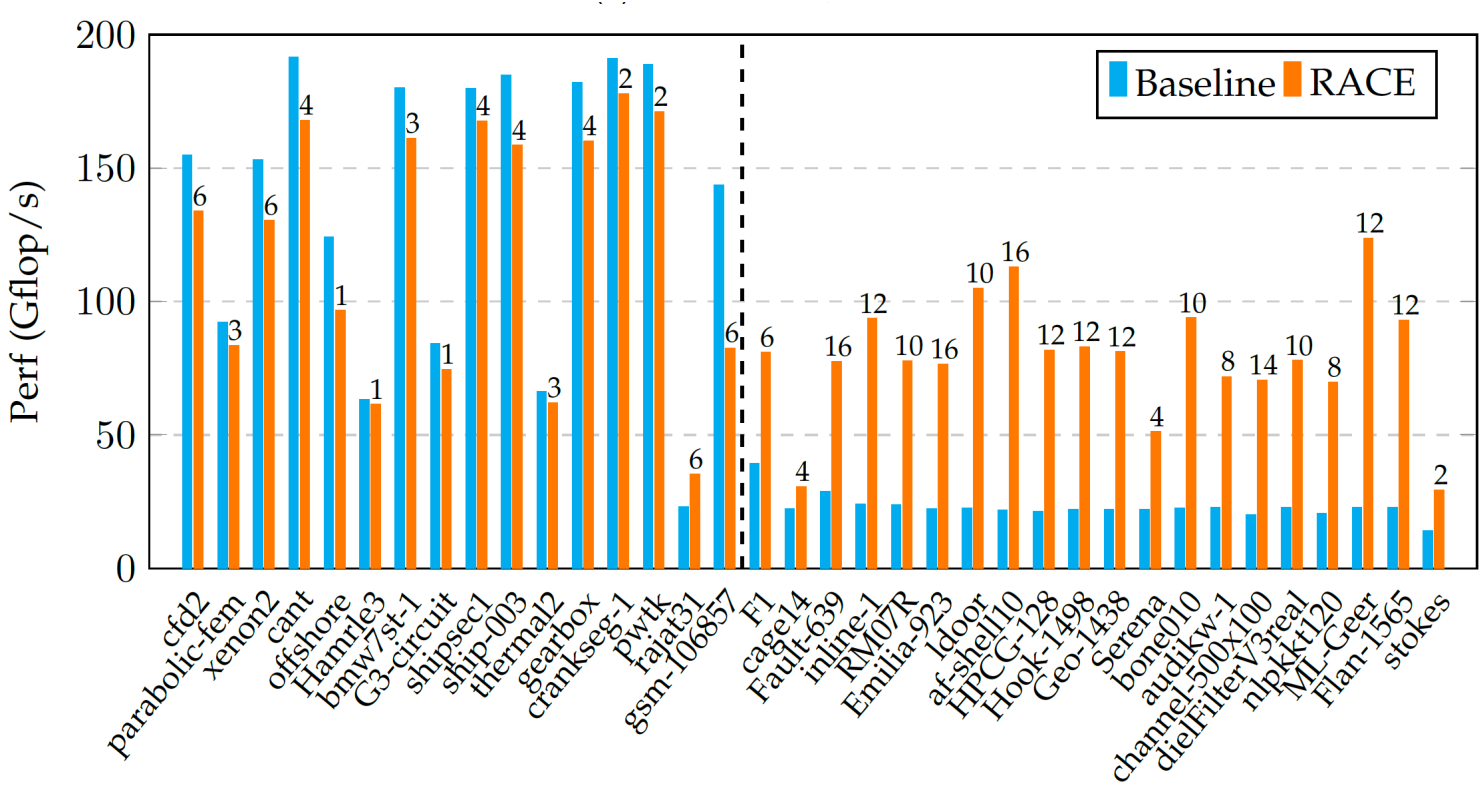


Intel Xeon Platinum 8368

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

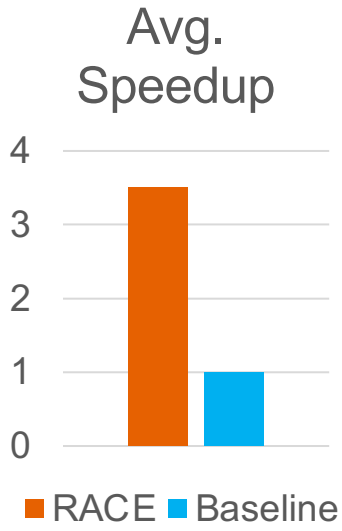


Matrix power kernel: Performance – AMD Rome



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

C. Alappat, J. Thies, G. Hager, H. Fehske, and G. Wellein: *Algebraic Temporal Blocking for Sparse Iterative Solvers on Multi-Core CPUs*. Submitted. Preprint: arXiv:2309.02228