Performance Engineering for Sparse Matrix-Vector Multiplication

C.L. Alappat, G. Hager, G. Wellein

Erlangen National Center for High Performance Computing (NHR@FAU)

Motivation – Sparse Matrix Multiplication

- SpMV: Multiplication of a sparse matrix with a dense vector

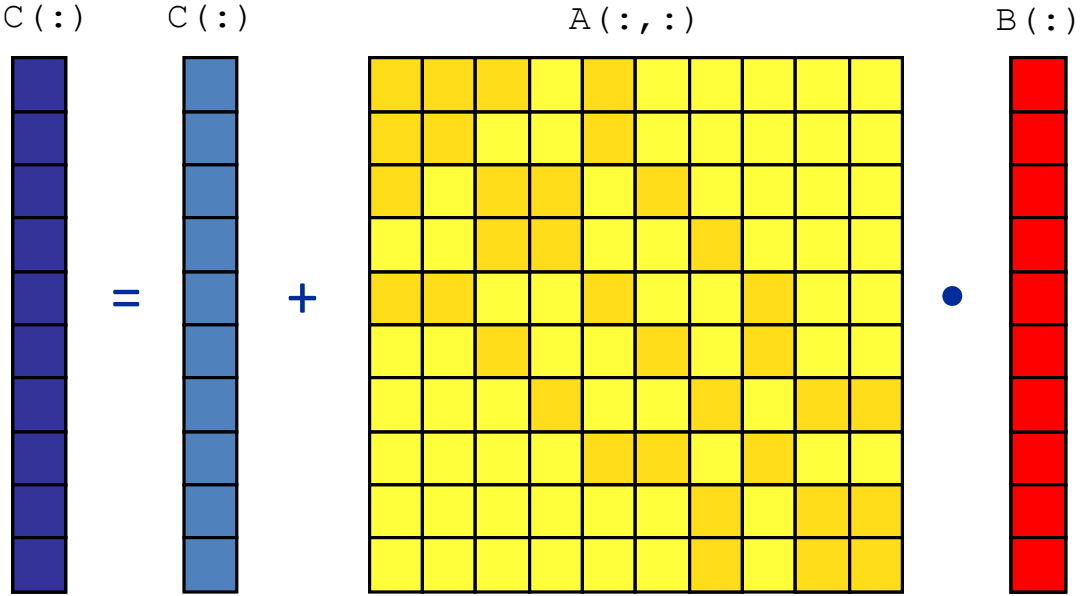
- Core component of sparse algebra alogrithms used in
  - Quantum Physics / Chemsitry / Computing
  - Engineering, e.g. CFD or Structural Mechanics
  - Clustering algorithms
  - …

- (Vendor) optimized libraries available: Intel MKL, NVIDIA cuSPARSE, SparseMAGMA,…

- This talk: Node-level implementation of SpMV for „large“ matrices

Motivation – Sparse Matrix Vector Multiplication

- Easy to parallelize but sparse irregular data structures / accesses
- SpMV Performance ←→ Strongly Memory Bound (high code balance)
- Sparse data format for `A(:,:)`
  - **CRS**, COO,…
  - ELLPACK, hybrid
  - SELL-C-σ, SELL-P
  - GPU formats, e.g. [1]



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

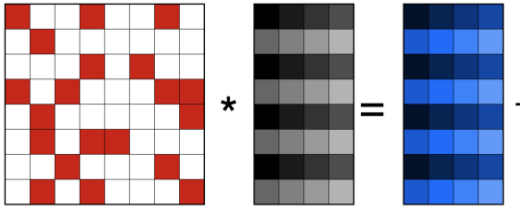
Motivation – Sparse Matrix Vector Multiplication

- 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

THIS TALK: Improve code balance of SpMV-algorithms

- Exploit symmetry: $A = A^t$
  - Many problems in physics, chemistry, engineering
  - Problem: Efficient parallelisation

- Cache blocking for Sparse Matrix Power kernels: $y = A^p x$
  - Matrix polynomials, asynchr. Krylov Methods,…
  - Problem: Data locality in „sparse matrix-matrix multiply“

- Approach: **Recursive Algebraic Coloring Engine** (RACE)

RACE
Hardware Friendly Coloring

PE for SpMV – Agenda

- SpMV Performance Model
- Exploiting Matrix Symmetry: SymmSpMV
- RACE: Hardware efficient SymmSpMV parallelization
- RACE: Hardware efficient Sparse Matrix Powers

SpMV Performance Model – A roofline type approach

$$P_{SpMV} = I * b_S = {}^{b_S}/_{B_C}$$

R.W. Hockney and I.J. Curington: Parallel Computing 10, (1989). DOI: 10.1016/0167-8191(89)90100-2
S. Williams: UCB Tech. Re. No. UCB/EECS-2008-164. PhD thesis (2008)

Gropp et al.: ParCFD1999. https://wgropp.cs.illinois.edu/bib/papers/pdata/1999/pcfd99/gkks.ps
Kreutzer et al.: SIAM SISC **36**(5), C401–C423 (2014). DOI: 10.1137/130930352,

SpMV node performance model – CRS (1)

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

```
real*8     val[Nnz]
integer*4  col_idx[Nnz]
integer*4  row_ptr[Nr]
real*8     C[Nr]
real*8     B[Nc]
```

} A(:,:)

Min. load traffic [B]: **(8 + 4) N_{nz}** + **(4 + 8) N_r** + **8 N_c**

Min. store traffic [B]: **8 N_r**

Total FLOP count [F]: 2 N_{nz}

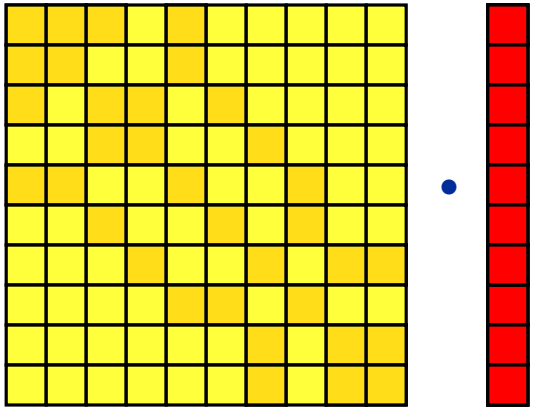
Nonzeros per row ($N_{nzr} = {}^{N_{nz}}/_{N_r}$) or column ($N_{nzc} = {}^{N_{nz}}/_{N_c}$)

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

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

SpMV node performance model – CRS (2)

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



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

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

Parameter (α) quantifies additional traffic for B (irregular access)

$$\alpha \geq 1/N_{nzc}$$

$$\alpha N_{nzc} \geq 1$$

Consider square matrices: $N_{nzc} = N_{nzr}$ and $N_c = N_r$

Note: $B_C\left(1/N_{nzr}\right) = B_{C,min}$

SpMV node performance model – CRS (3)

- Value of α depends on matrix structure ←→ cache subsystem
  - Size of LLC > sizeof(B) → $\alpha \approx {}^{1}/_{N_{nzc}}$
  - $\alpha = 1$: Each load to B main memory access
  - $\alpha > 1$: Highly erratic access

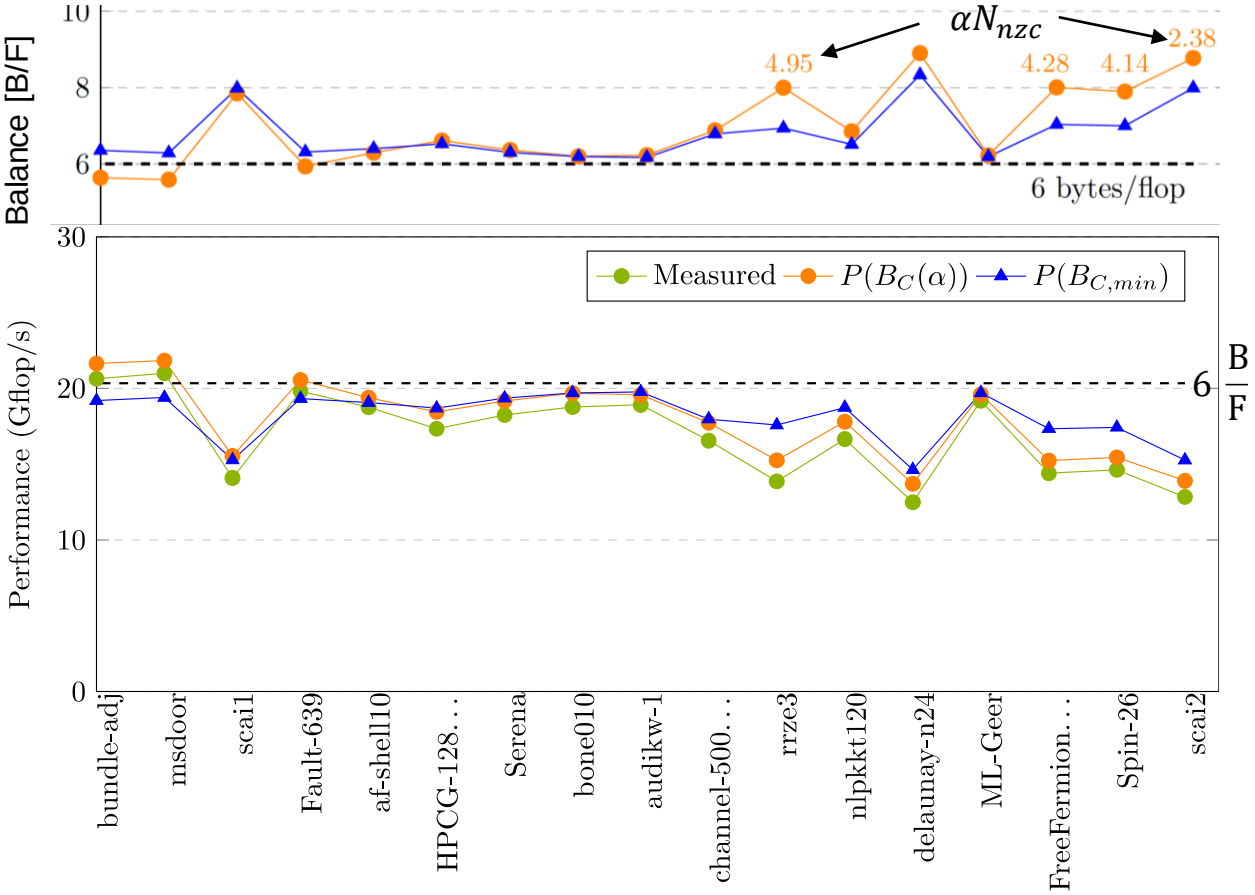
- Determine by measuring the balance, i.e. main memory data volume

$$B_C(\alpha) = \left(6+4\boldsymbol{\alpha}+\frac{10}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}} = \frac{\boldsymbol{V_{meas}}}{N_{nz}\cdot 2\ \mathrm{F}} \quad (= B_C^{meas})$$

- Maximum SpMV performance (roofline): $P_{SpMV} = \frac{b_S}{B_C}$
- More technical details: Node-level Performance Engineering (tutorial)

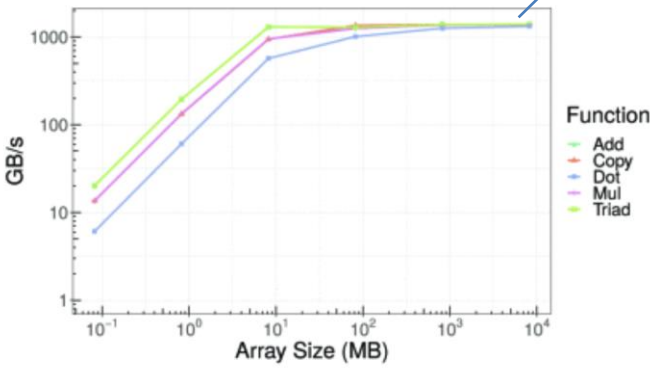
SpMV node performance model – CLX-AP

Intel Xeon Platinum 9242
24c@2.8GHz (turbo)
$b_S = 122\ GB/s$



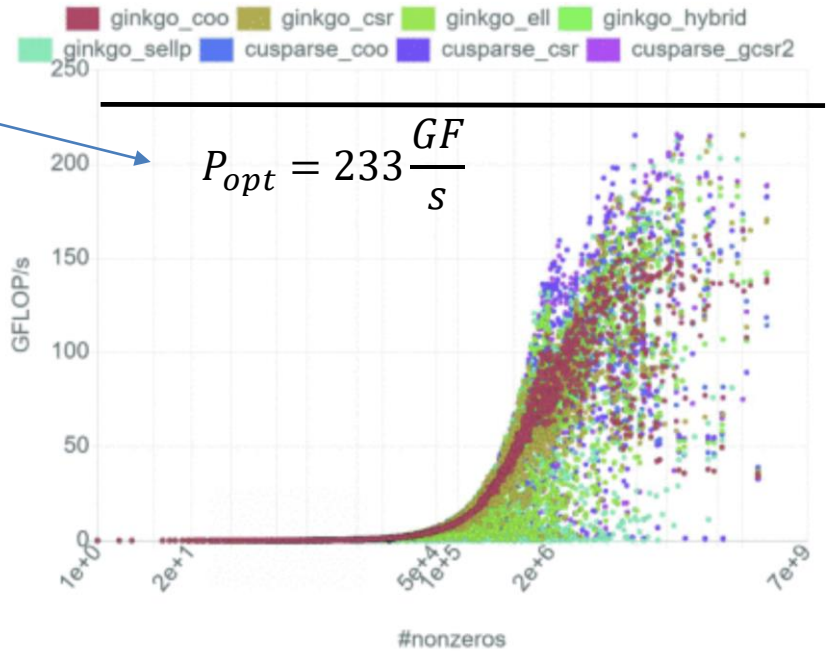
SpMV performance model – GPU lightspeed

$$b_S = 1400 \frac{GB}{s}$$



(b) A100

$$P_{opt} = 233 \frac{GF}{s}$$



(a) SpMV performance profile on A100.SpMV

H. Anzt, et al; *2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS)*, GA, USA, 2020, pp. 26-38, doi: 10.1109/PMBS51919.2020.00009.

NHR FAU

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Exploiting Matrix Symmetry: SymmSpMV

Symmetric SpMV – serial execution

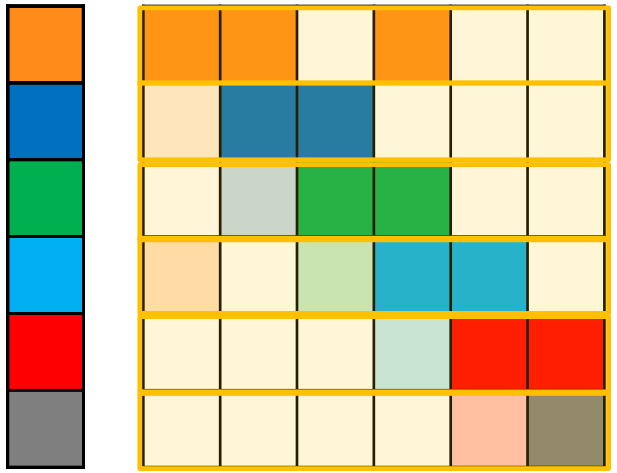
- Store only upper (lower) triangular part of symmetric sparse matrix

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

- Improve code balance by up to 2x

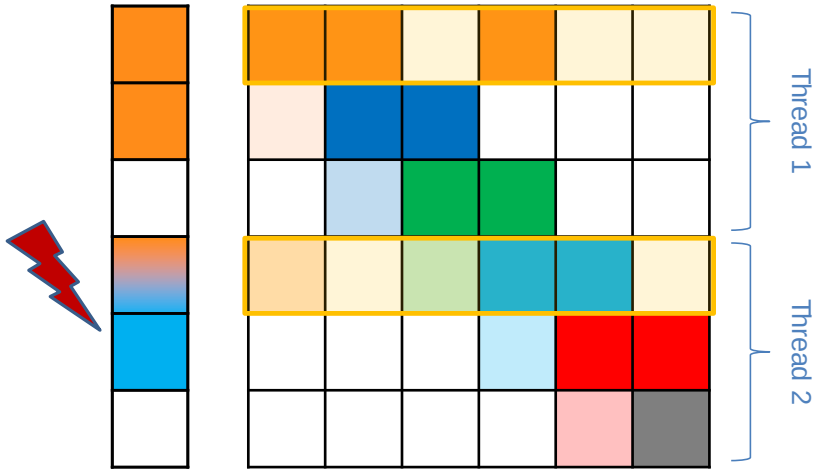
$$B_{C,min} = \frac{\mathbf{12} + 20/\mathrm{N_{nzr}} + 8\,\alpha}{\mathbf{2}}\,\frac{B}{F}$$

$$B^{sym}_{C,min} = \frac{\mathbf{12} + 4/\mathrm{N^{sym}_{nzr}} + 24\,\alpha^{sym}}{\mathbf{4}}\,\frac{B}{F}$$



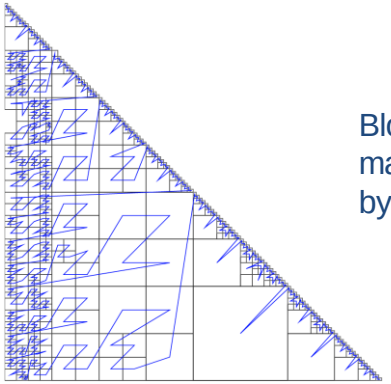
Symmetric SpMV – parallelisation: write conflicts

```
#pragma omp parallel for
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))
  C(col_idx(j)) = C(col_idx(j)) + val(j) * B(j)
 enddo
enddo
```



Symmetric SpMV – parallelisation approaches

- Technical solutions (OpenMP)
  - Locks (`omp_set_lock(&lock)`)
  - Thread-local copies of result vector (`reduction(C(:):+)`)
- Special data formats
  - Compressed Sparse Blocks [1]
  - Recursive Sparse Blocks [2]



Block partitioning of audikw_1 matrix for 16 threads as done by RSB. Picture source [2].

[1] Buluç et al., Parallel sparse matrix-vector and matrix-transpose-vector multiplication using compressed sparse blocks. SPAA'09, 2009, https://doi.org/10.1145/1583991.1584053

[2] Michele Martone, Efficient multithreaded untransposed, transposed or symmetric sparse matrix–vector multiplication with the Recursive Sparse Blocks format, Parallel Computing, 2014, https://doi.org/10.1016/j.parco.2014.03.008

Symmetric SpMV – parallelisation approaches

- D2-coloring of undirected graph (representing symmetric sparse matrix)
  - Vertices/columns with same color can be calculated in parallel
  - Same color matrix entries consecutive in memory (matrix reordering)
- Existing methods
  - Multicoloring [1]
  - ABMC [2]
- Benefits
  - Need no change in data storage format
  - No locking or thread private vectors.

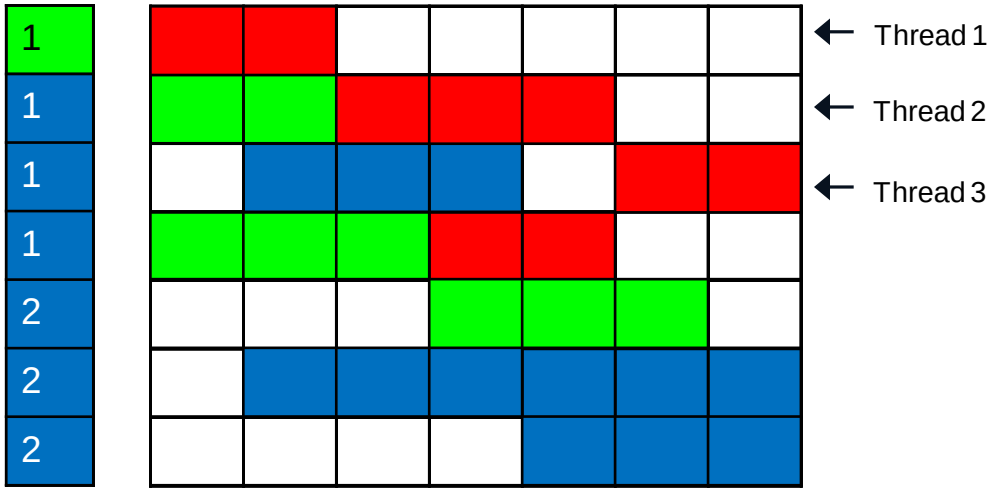
Commonly used strategy for D1 coloring like Gauss-Seidel, ICCG or D2 for KACZ [3]

[1] M. T. Jones and P. E. Plassmann, Scalable iterative solution of sparse linear systems, https://doi.org/10.1016/0167-8191(94)90004-3

[2] Iwashita et al., Algebraic block multi-color ordering method for parallel multi-threaded sparse triangular solver in iccg method, https://doi.org/10.1109/IPDPS.2012.51

[3] Galgon et al., On the parallel iterative solution of linear systems arising in the feast algorithm for computing inner eigenvalues,. https://doi.org/10.1016/j.parco.2015.06.005

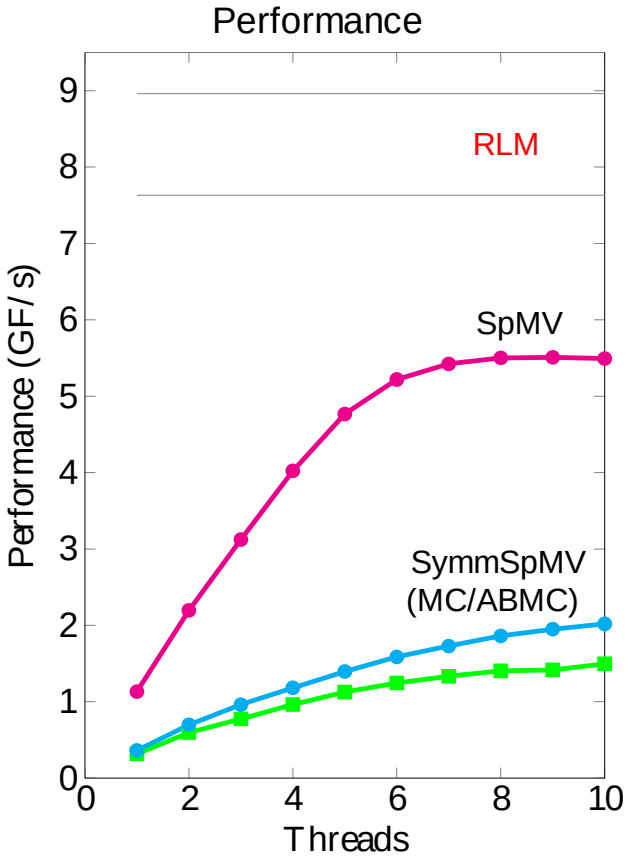
Symmetric SpMV – thread level parallelisation: MC



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

Repeatedly accessing complete vectors
=> α^{sym} may increase!

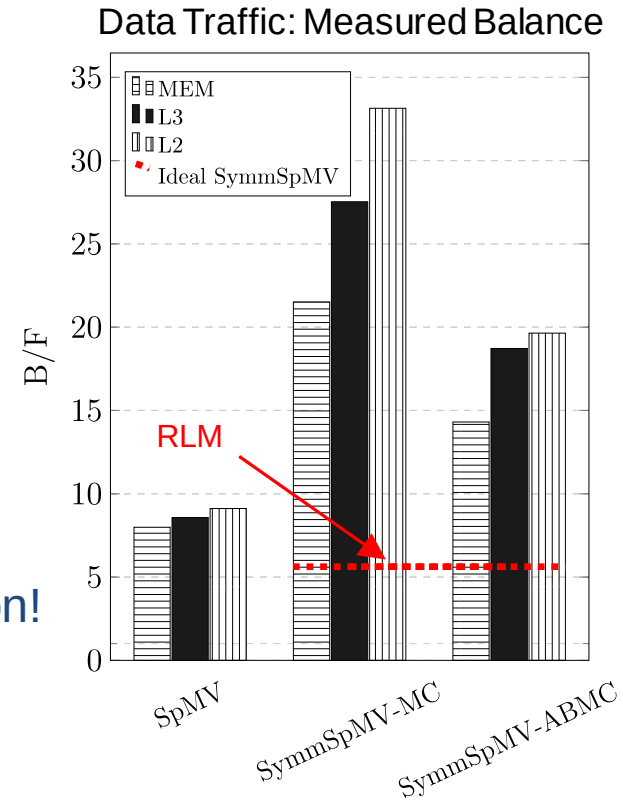
Symmetric SpMV – thread level parallelisation: MC



Intel Xeon E5-2660v2
10c@ 2.2 GHz
b_S= 40 / 47 GB/s

Spin-26 matrix
N_r = 10,400,600
N_{nz} = 145,608,400

Data traffic increases!
Performance degredation!



RACE:
Hardware efficient SymmSpMV parallelization

C. Alappat, et al. (2020). *A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication*. ACM Trans. Parallel Comput. 7, 3, Article 19 (August 2020). DOI: https://doi.org/10.1145/3399732

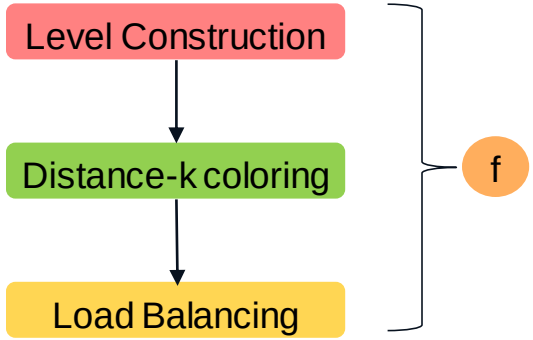
Recursive Algebraic Coloring Engine

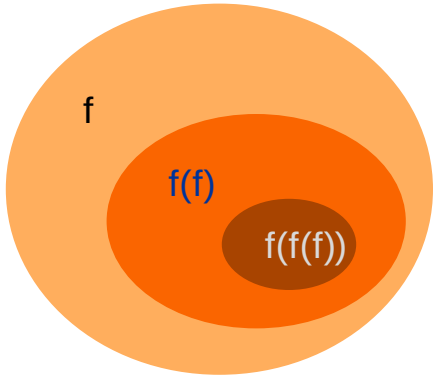
RACE uses a recursive level based method.

Objectives motivated by hardware efficiency

- Preserve data locality (lower α factor).
- Generate sufficient parallelism to support hardware underneath.
- Reduce synchronization overheads.
- Use simple data format like CRS.

3 Steps

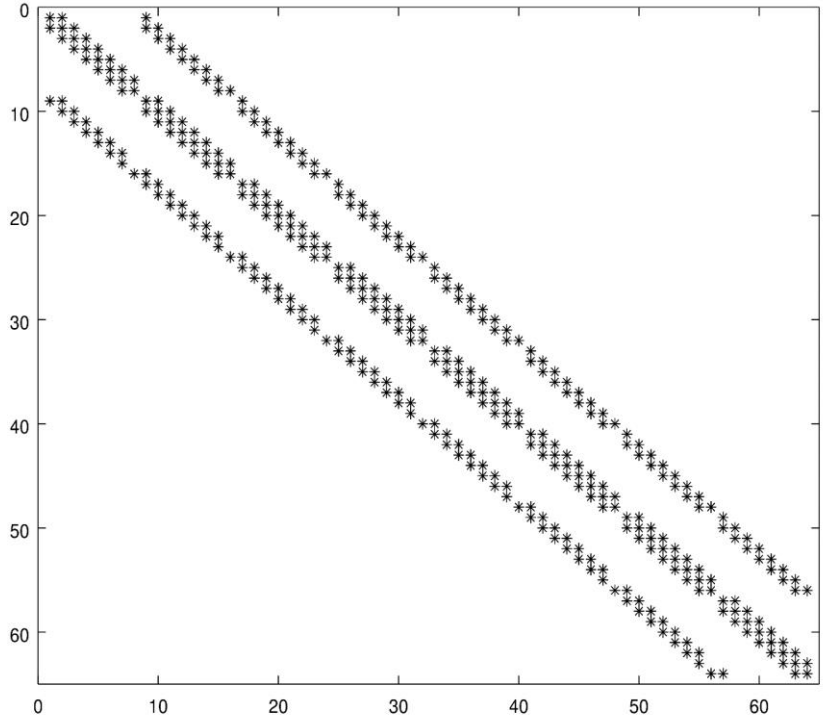




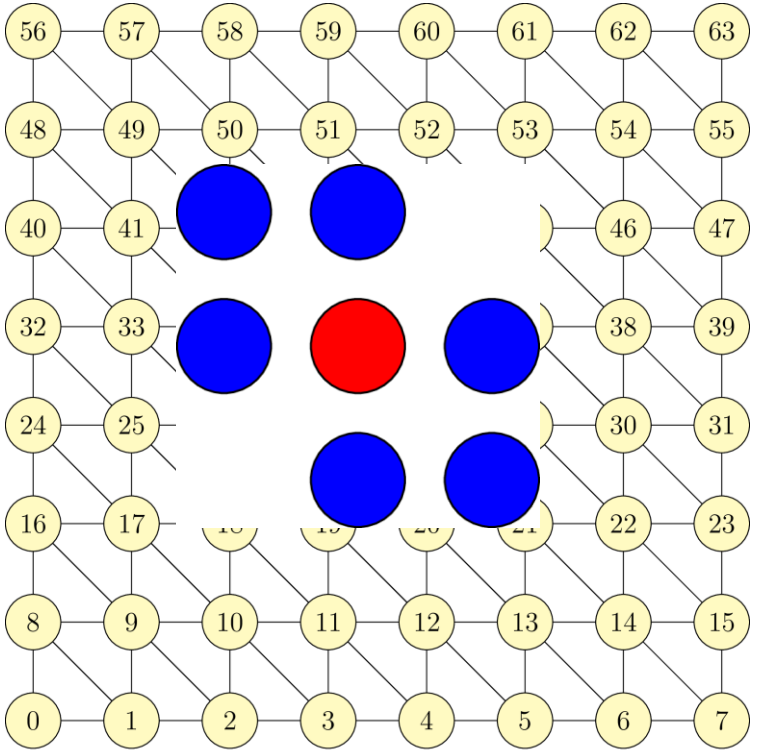
Recursively apply if needed

Sample Stencil Matrix and its graph representation

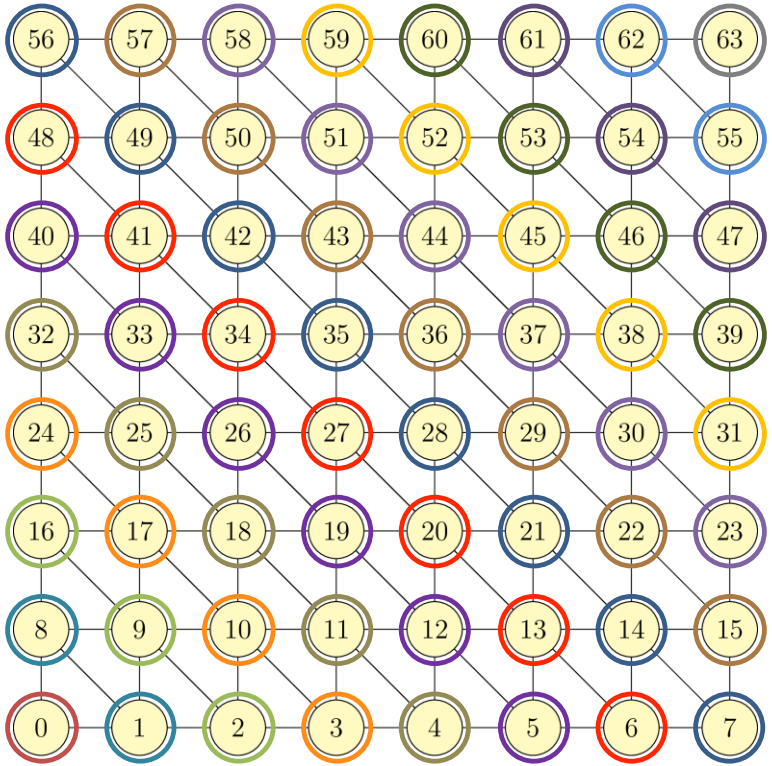
Symmetric Matrix (Stencil)



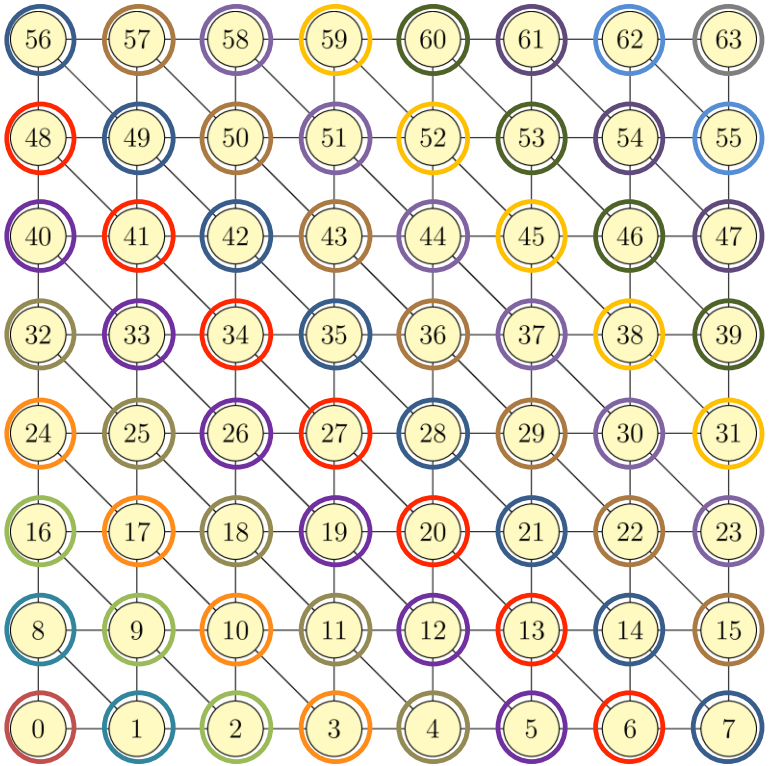
Undirected Graph

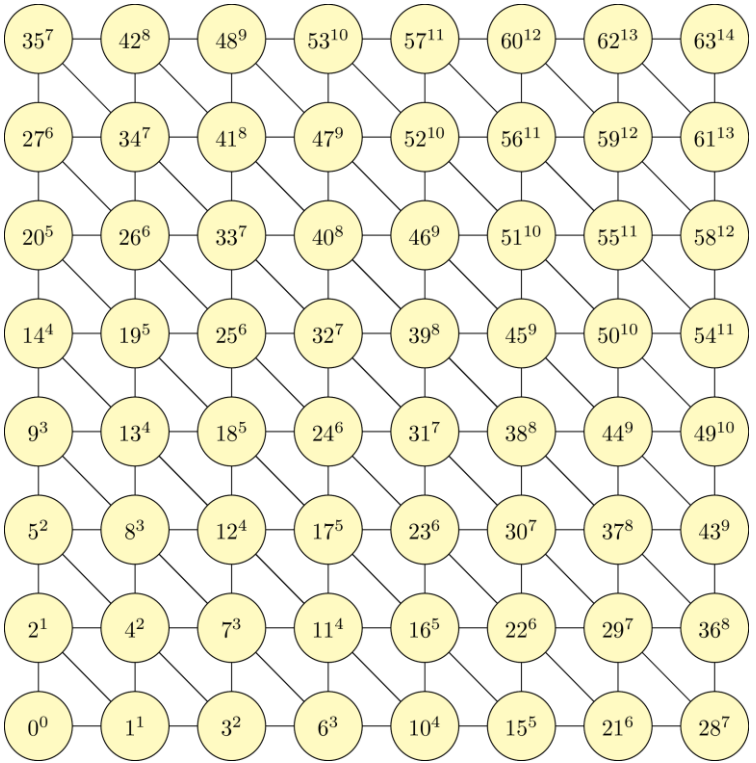


Step 1: Level Construction – BFS

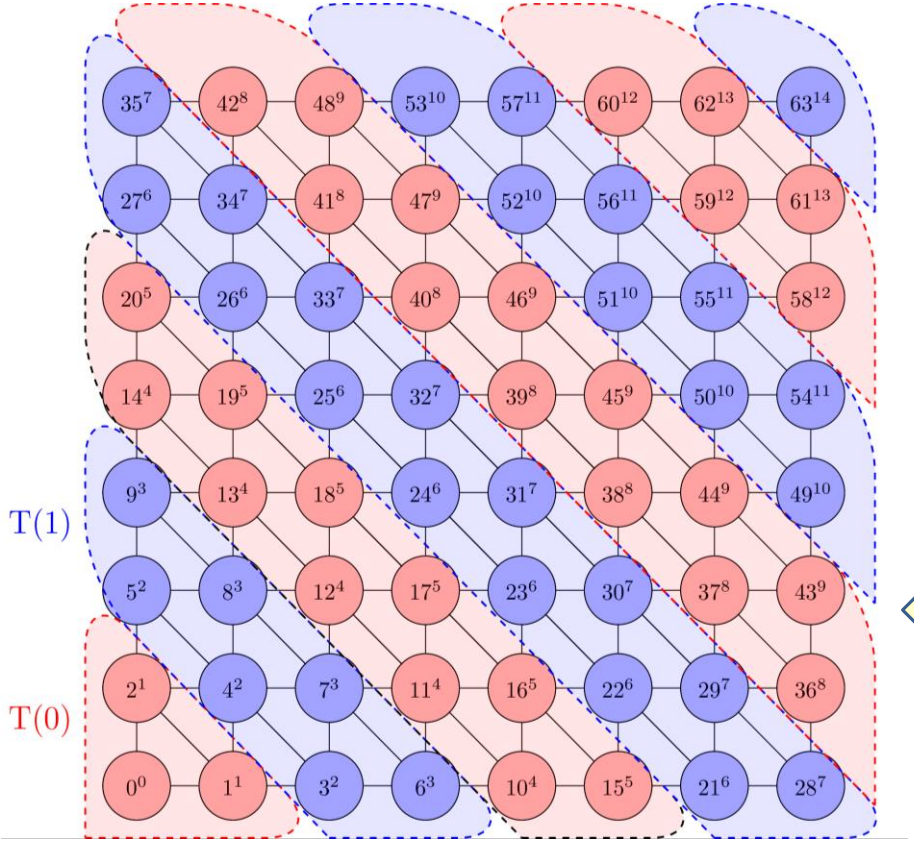


Step 1: Level Construction – Permutation





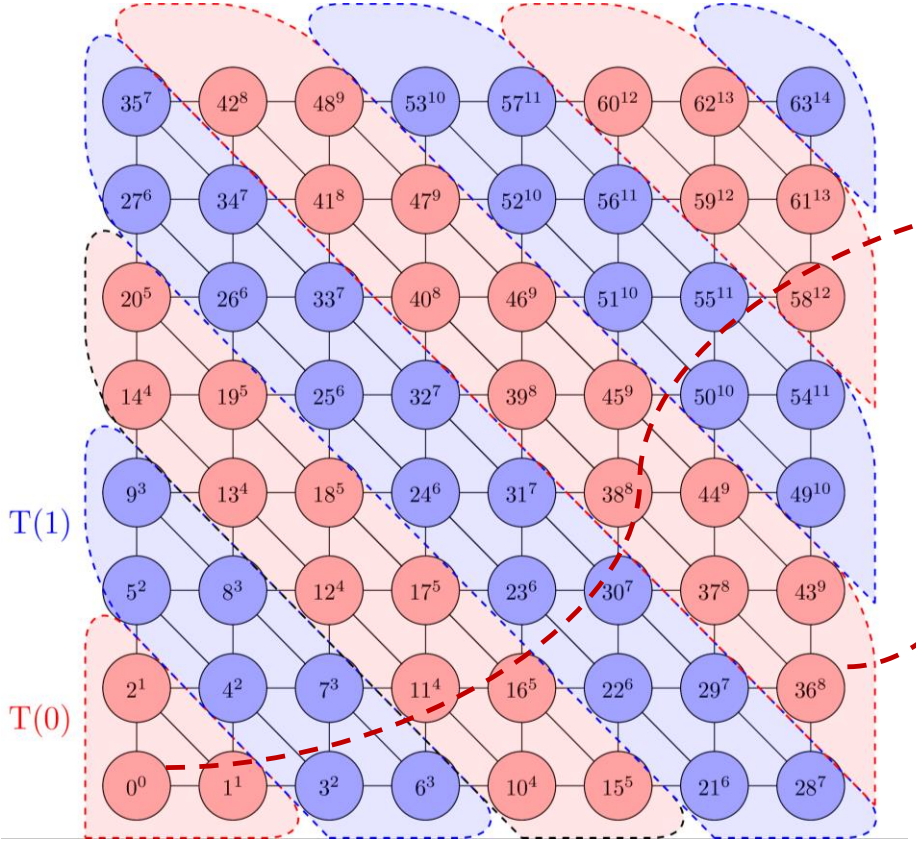
Step 2: Distance-k coloring



For distance-k coloring at least (k-1) levels in a level group (T)

Distance-2 example

But …

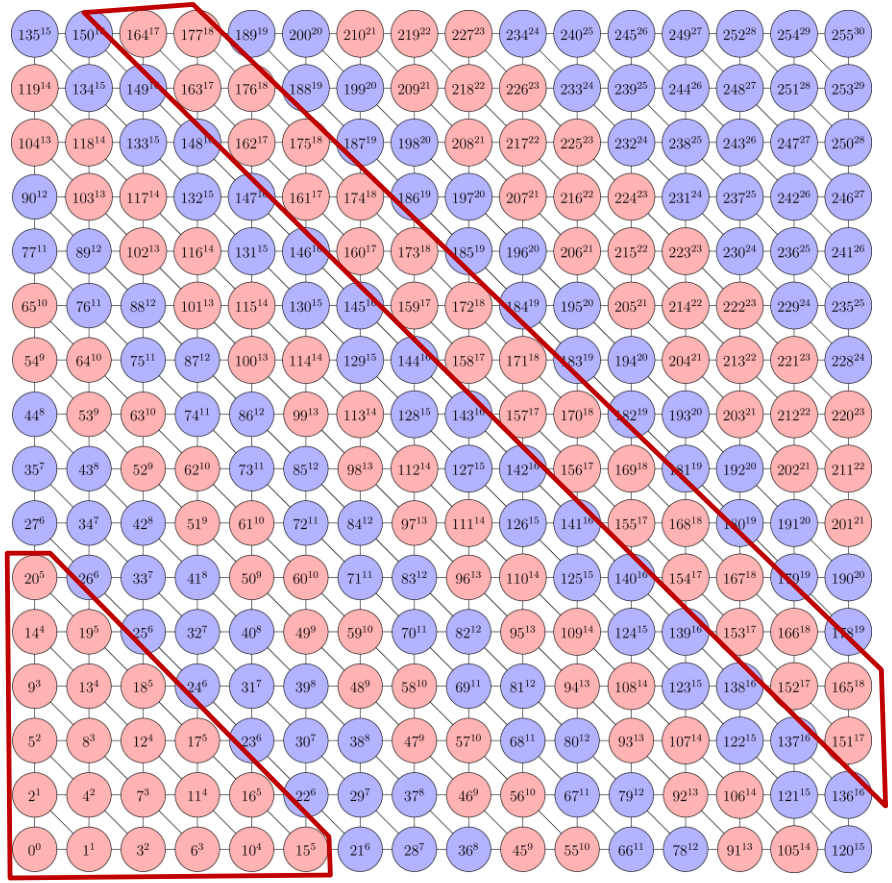


$n_r(T(0)) = 3$

$n_r(T(4)) = 13$

Load imbalance

Step 3: Load balancing



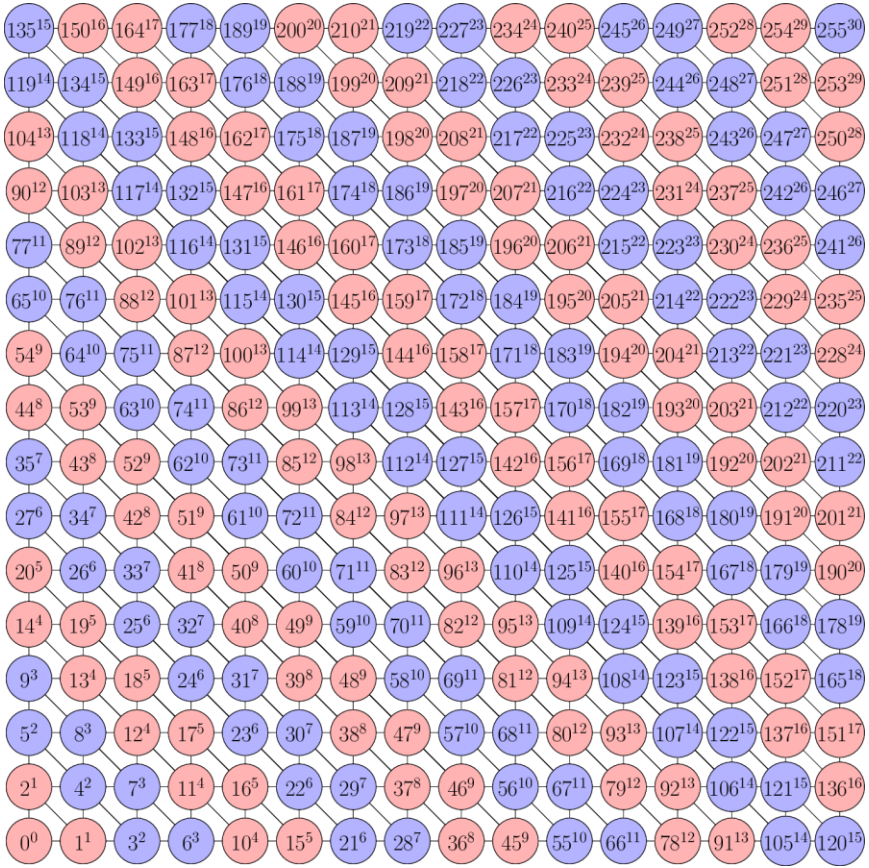
5 threads

More levels since n_r on each levels are small

Just sufficient levels to maintain D2 independency

Step 3: Load balancing

Solution: Find more parallelism using Recursion

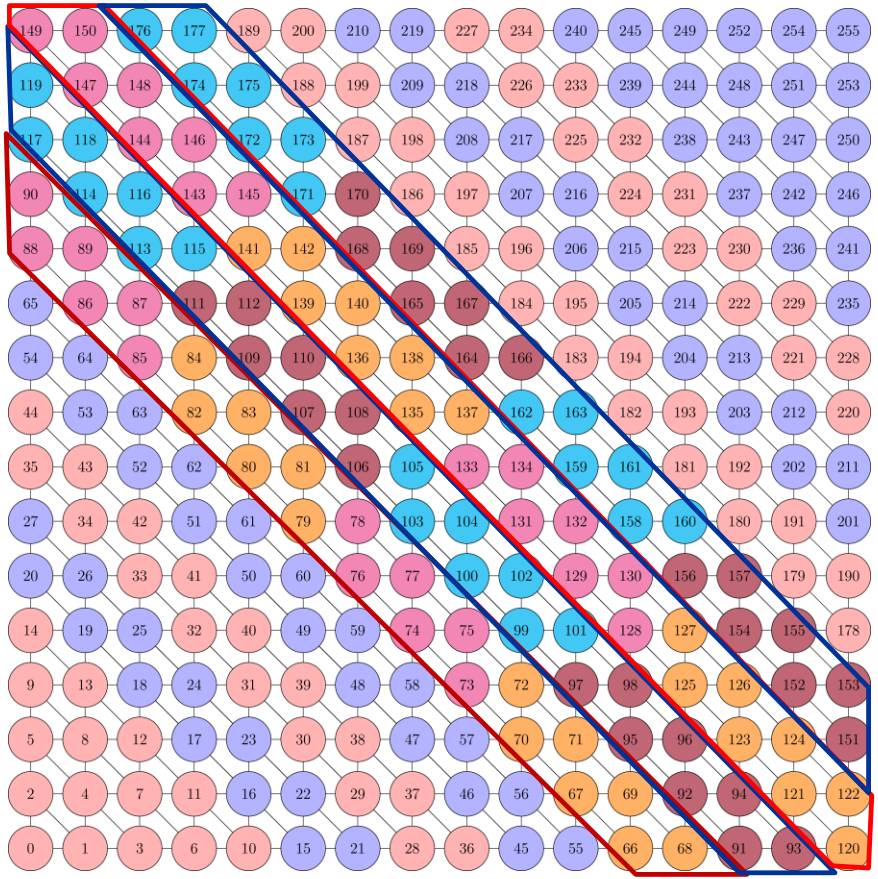


8 threads

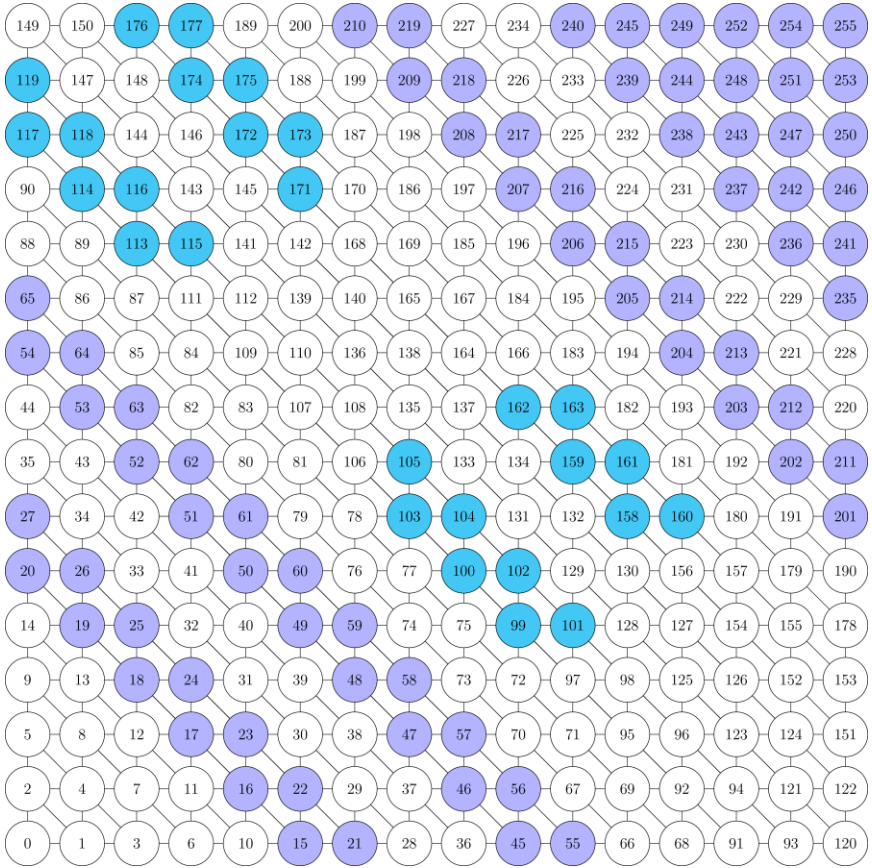
Load imbalance

Parallelism limited by number of levels

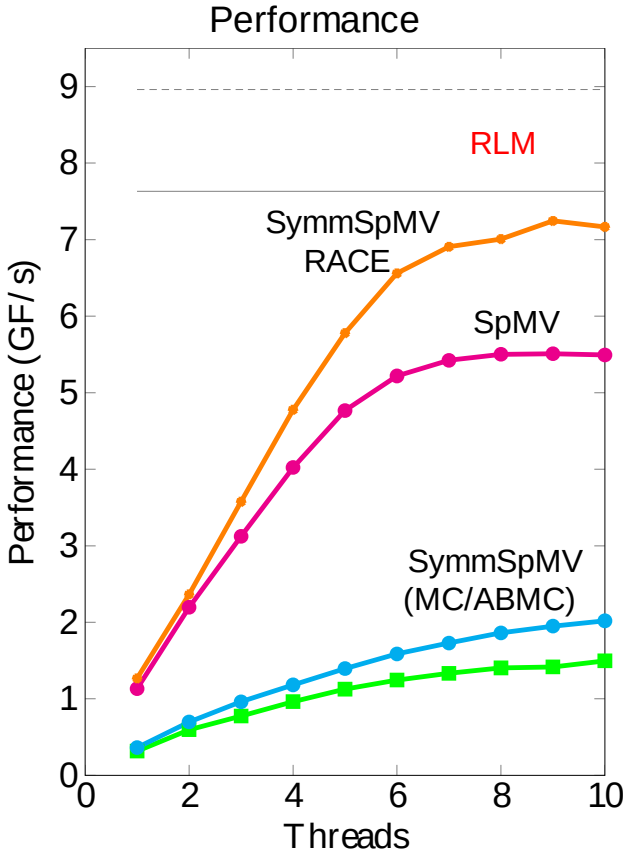
Load balancing: Recursion



Load balancing: Recursion

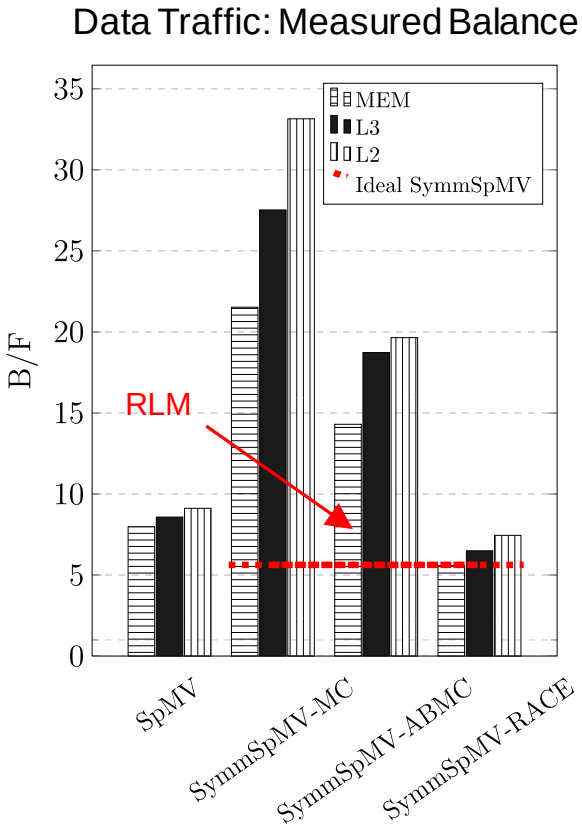


Symmetric SpMV – RACE Perfromance

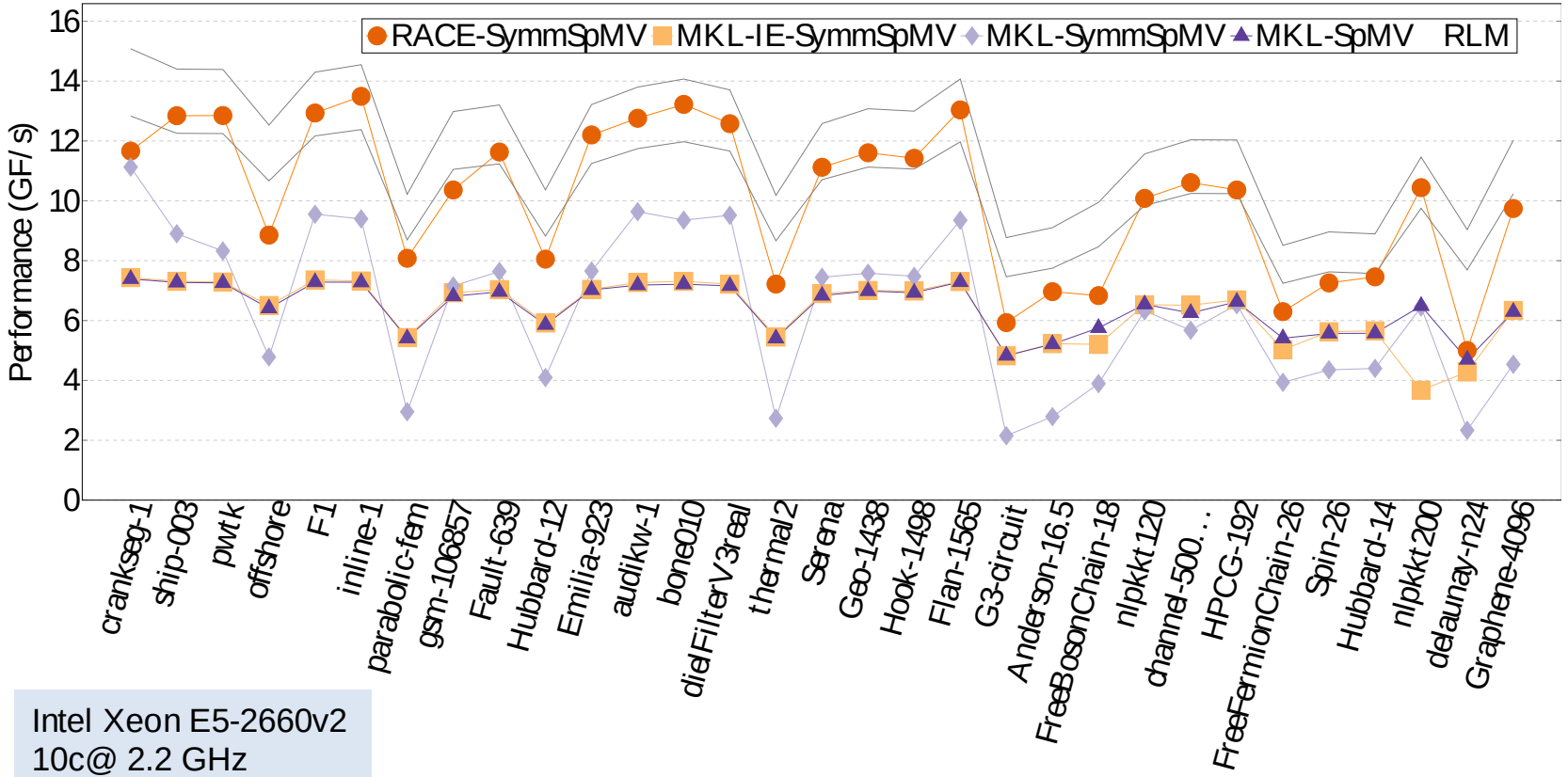


Intel Xeon E5-2660v2
10c@ 2.2 GHz
b_S= 40 / 47 GB/s

Spin-26 matrix
N_r = 10,400,600
N_{nz} = 145,608,400

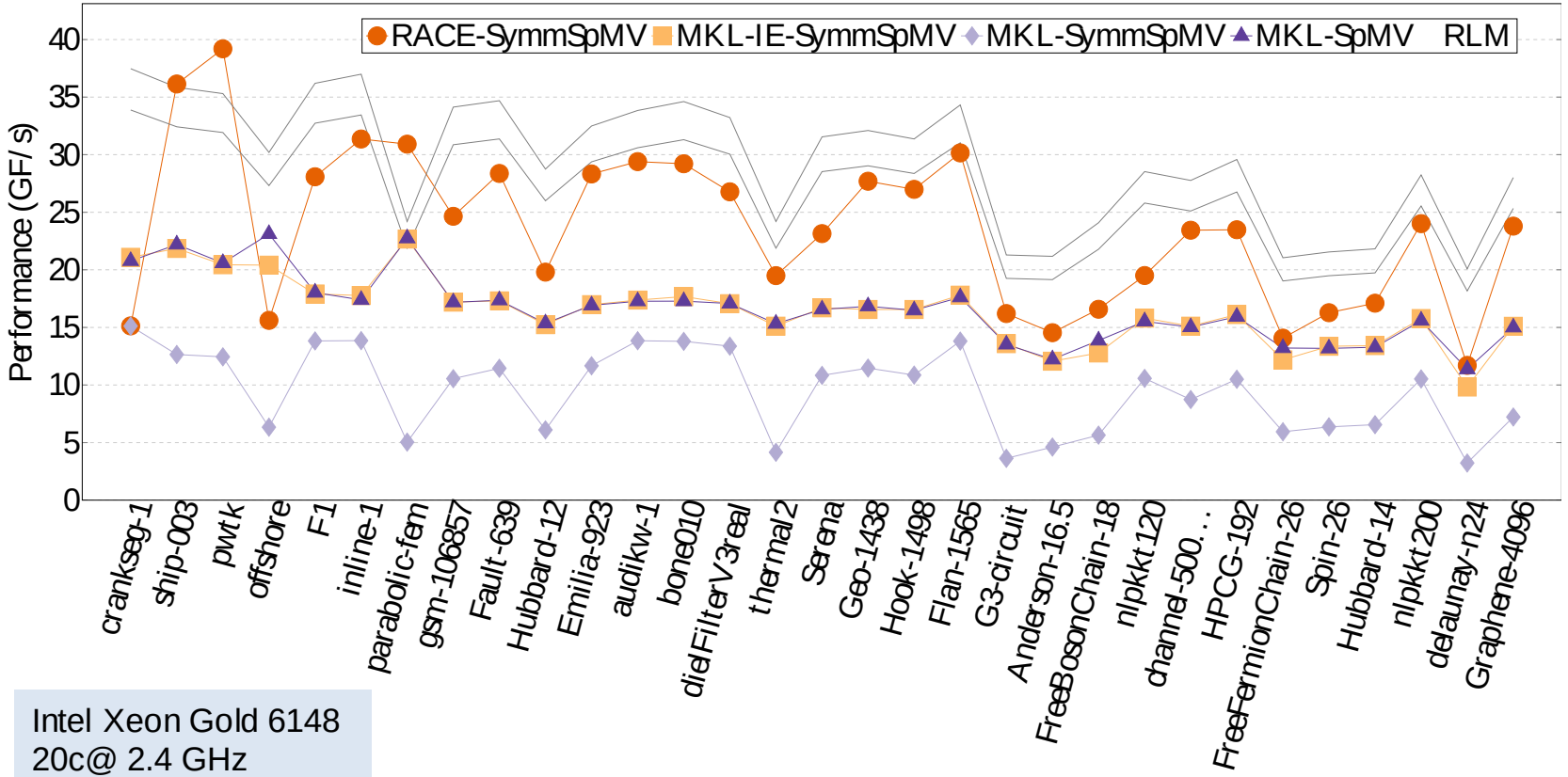


Symmetric SpMV – RACE Perfromance



Intel Xeon E5-2660v2
10c@ 2.2 GHz
b_S= 40 / 47 GB/s

Symmetric SpMV – RACE Perfromance



Intel Xeon Gold 6148
20c@ 2.4 GHz
b_S= 115 / 104 GB/s

Performance comparison

Speedup of RACE compared to other methods

Method	Min.	Max.	Mean
MC (ColPack 2019)	1.40	13.92	3.52
ABMC (ColPack + METIS v 5.1.0)	0.99	4.89	1.66
RSB (v 1.2.0-rc7)	1.02	2.86	1.44

RACE: Hardware efficient Sparse Matrix Powers

Example: $y = A^3x = (A^3)\,x = A(A(Ax))$

C. Alappat, et al. (2021). In preparation

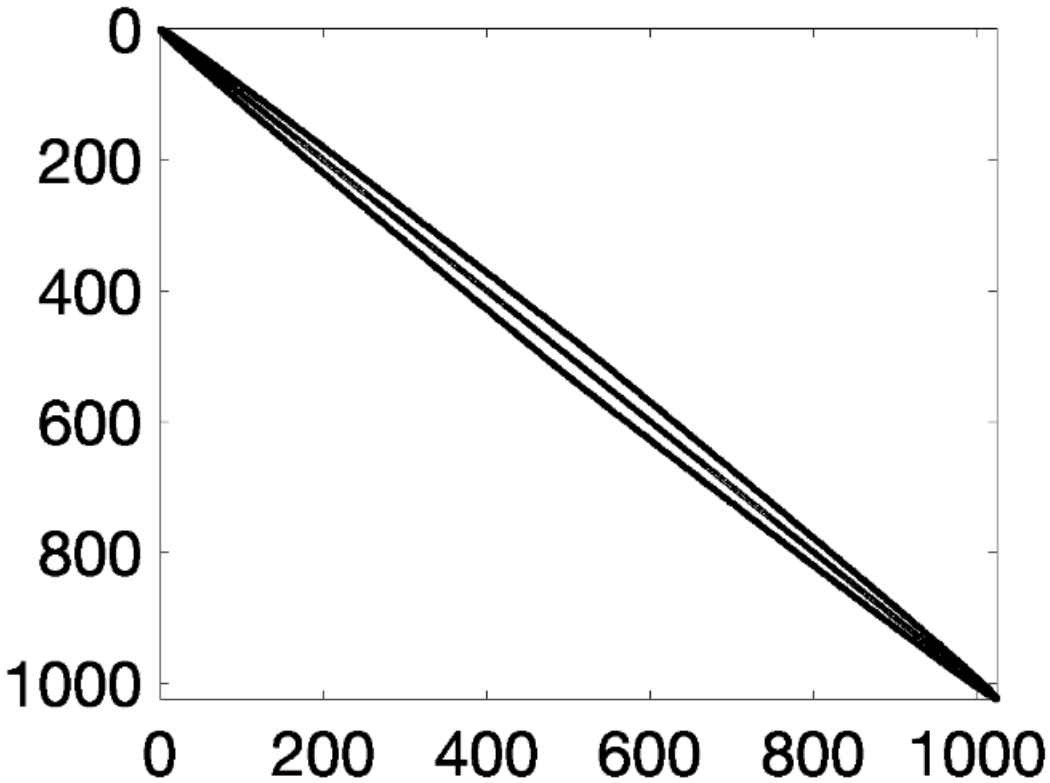
Matrix powers: Concept

Calculate $y = A^3 x$

Traditional approach

$$y_1 = Ax$$
$$y_2 = Ay_1$$
$$y = y_3 = Ay_2$$

Matrix accessed 3 times from main memory

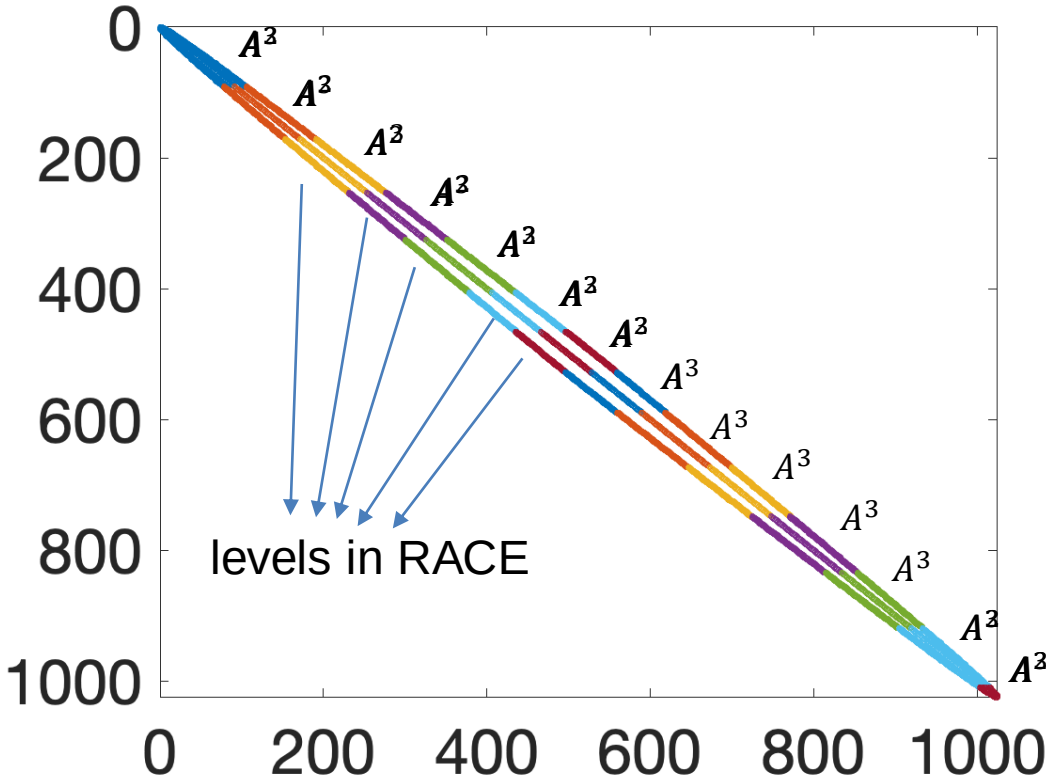


Matrix powers: Concept

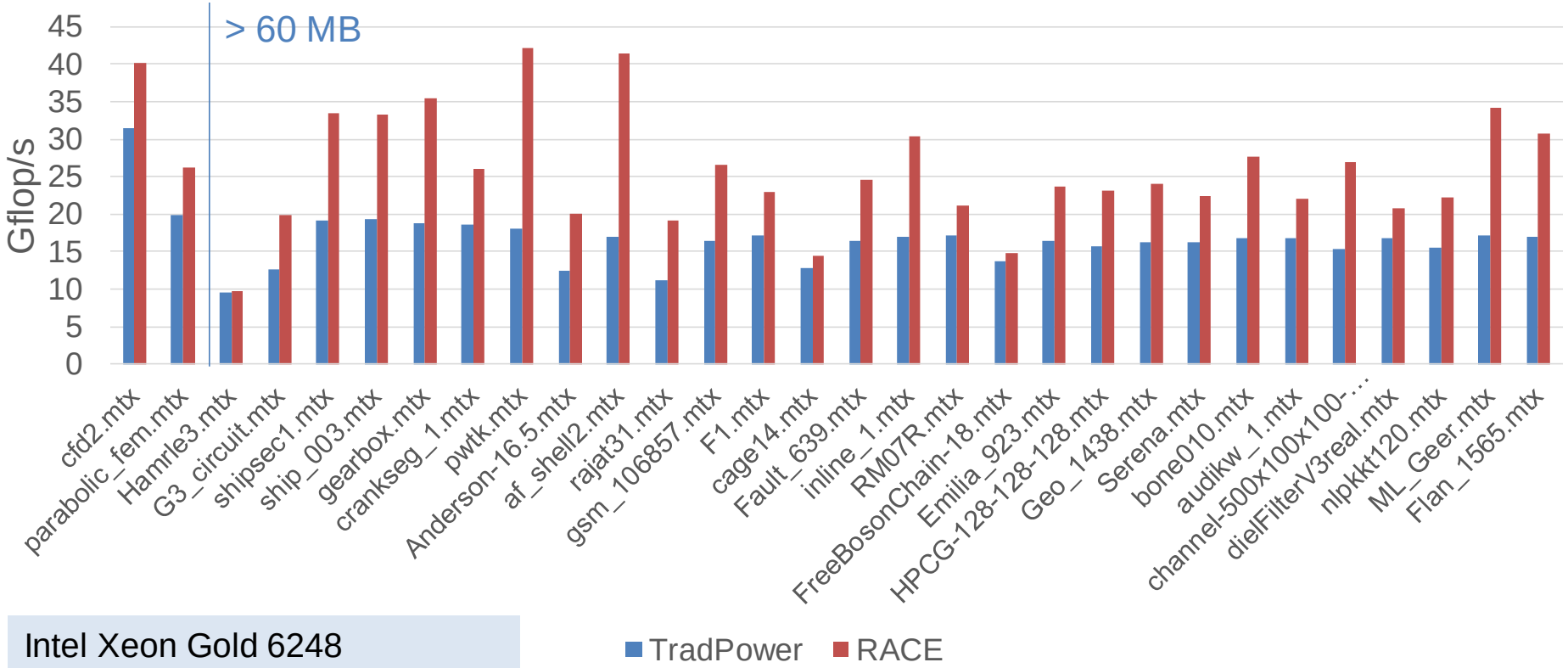
Calculate $y = A^3 x$

RACE approach

A^2x, A^3x computations in-cache

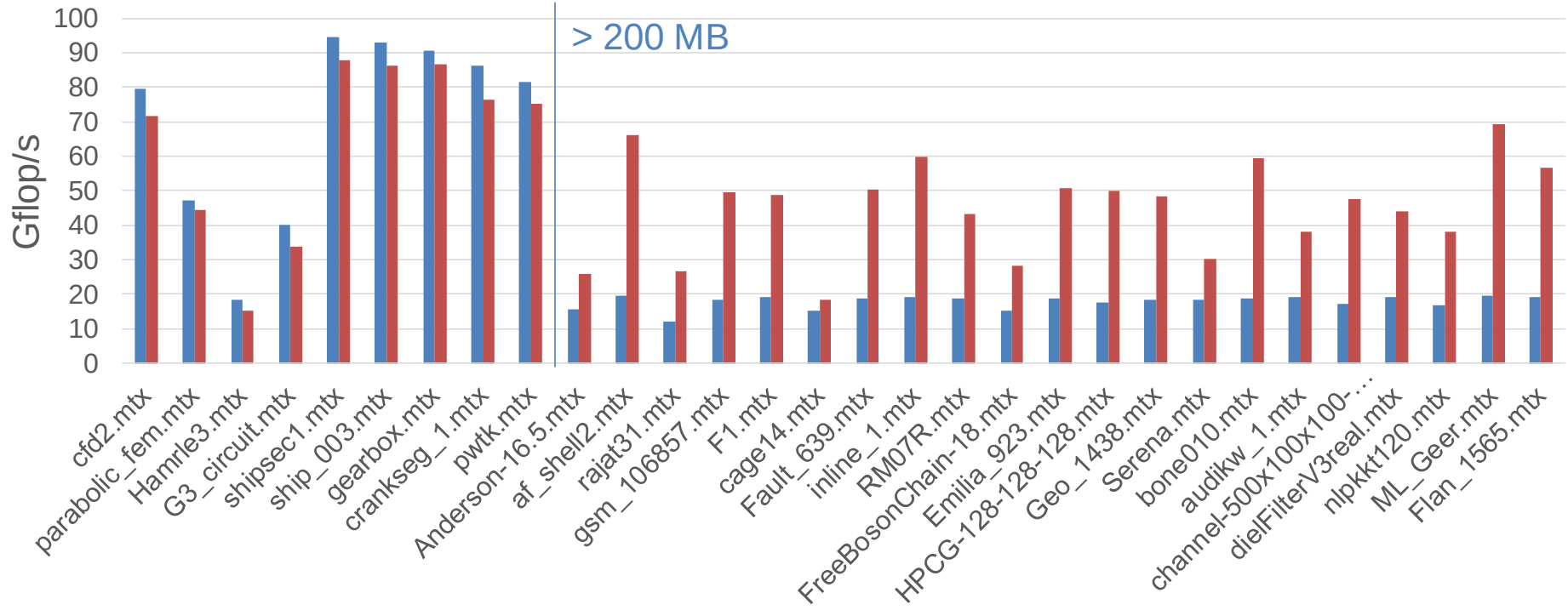


Sparse Matrix Powers: Performance



Intel Xeon Gold 6248
SNC off mode, 20c

Sparse Matrix Powers: Performance



AMD ROME 7452
NPS1 mode, 32c

Summary

- Performance Modeling for SpMV@CPU: useful upper performance bounds
- Plenty of room for performance improvement for SpMV based algorithms
- RACE: Hardware-aware approach to extract parallelism and locality in SpMV operations
- Outlook:
  - Integration in distributed memory parallelisation
  - New architectures: A64FX – CRS data format no longer sufficient

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

C. L. Alappat, N. Meyer, J. Laukemann, T. Gruber, G. Hager, G. Wellein, and T. Wettig: *ECM modeling and performance tuning of SpMV and Lattice QCD on A64FX.*
Preprint: arXiv:2103.03013

C. L. Alappat, J. Laukemann, T. Gruber, G. Hager, G. Wellein, N. Meyer, and T. Wettig: *Performance Modeling of Streaming Kernels and Sparse Matrix-Vector Multiplication on A64FX*. 2020 IEEE/ACM Performance Modeling, Benchmarking and Simulation of High Performance Computer Systems (PMBS), GA, USA, 2020, pp. 1-7.
.DOI: 10.1109/PMBS51919.2020.00006 Preprint: arXiv:2009.13903

C. L. Alappat, J. Hofmann, G. Hager, H. Fehske, A. R. Bishop, and G. Wellein: *Understanding HPC Benchmark Performance on Intel Broadwell and Cascade Lake Processors*. ISC 2020. LNCS. DOI: 10.1007/978-3-030-50743-5_21

Information on test matrices (1)

Table 2. Details of the Benchmark Matrices

Index	Matrix name	N_r	N_{nz}	N_{nzr}	bw	bw_{RCM}
1	crankseg_1* (C)	52,804	10,614,210	201.01	50,388	5,126
2	ship_003*	121,728	8,086,034	66.43	3,659	3,833
3	pwtk*	217,918	11,634,424	53.39	189,331	2,029
4	offshore*	259,789	4,242,673	16.33	237,738	19,534
5	F1	343,791	26,837,113	78.06	343,754	10,052
6	inline_1 (C)	503,712	36,816,342	73.09	502,403	6,002
7	parabolic_fem* (C)	525,825	3,674,625	6.99	525,820	514
8	gsm_106,857*	589,446	21,758,924	36.91	588,744	17,865
9	Fault_639	638,802	28,614,564	44.79	19,988	19,487
10	Hubbard-12* (Q)	853,776	11,098,164	13.00	232,848	38,780
11	Emilia_923	923,136	41,005,206	44.42	17,279	14,672
12	audikw_1	943,695	77,651,847	82.29	925,946	35,084
13	bone010	986,703	71,666,325	72.63	13,016	14,540
14	dielFilterV3real	1,102,824	89,306,020	80.98	1,036,475	25,637
15	thermal2*	1,228,045	8,580,313	6.99	1,226,000	797
16	Serena	1,391,349	64,531,701	46.38	81,578	84,947
17	Geo_1438	1,437,960	63,156,690	43.92	26,018	30,623
18	Hook_1498	1,498,023	60,917,445	40.67	29,036	28,994
19	Flan_1565	1,564,794	117,406,044	75.03	20,702	20,849
20	G3_circuit*	1,585,478	7,660,826	4.83	947,128	5,068
21	Anderson-16.5* (Q)	2,097,152	14,680,064	7.00	1,198,372	24,620
22	FreeBosonChain-18 (Q)	3,124,550	38,936,700	12.46	2,042,975	131,749
23	nlpkkt120	3,542,400	96,845,792	27.34	1,814,521	86,876
24	channel-500x100x100-b050	4,802,000	90,164,744	18.78	600,299	23,766
25	HPCG-192	7,077,888	189,119,224	26.72	37,057	110,017
26	FreeFermionChain-26 (Q)	10,400,600	140,616,112	13.52	5,490,811	434,345
27	Spin-26 (Q)	10,400,600	145,608,400	14.00	709,995	211,828
28	Hubbard-14 (Q)	11,778,624	176,675,928	15.00	3,171,168	425,415
29	nlpkkt200	16,240,000	448,225,632	27.60	8,240,201	240,796
30	delaunay_n24	16,777,216	100,663,202	6.00	16,769,102	32,837
31	Graphene-4096 (C,Q)	16,777,216	218,013,704	13.00	4,098	6,145

N_r is the number of matrix rows, and N_{nz} is the number of nonzeros. $N_{nzr} = N_{nz}/N_r$ is the average number of nonzeros per row. bw and bw_{RCM} refer to the matrix bandwidth without and with RCM preprocessing. The letter "C" in the parentheses of the matrix name indicates a corner case matrix that will be discussed in detail, while the letter "Q" marks a matrix from quantum physics that is not part of the SuiteSparse Matrix Collection. With an asterisk (*), we have labeled all the matrices that are less than 128 MB, which could potentially lead to some caching effects especially on the Skylake SP architecture.

C. Alappat, et al. (2020). *A Recursive Algebraic Coloring Technique for Hardware-efficient Symmetric Sparse Matrix-vector Multiplication*. ACM TOPC 7, 3, Article 19 (August 2020). DOI: https://doi.org/10.1145/3399732

Information on test matrices (2)

TABLE 4 Details of the benchmark matrices. N_r is the number of rows, N_{nz} is the number of nonzeros, and N_{nzr} is the average number of nonzeros per row. Most matrices were taken from the SuiteSparse Matrix Collection [15]. The matrices with ∗ come from other research projects.

Index	Matrix name	N_r	N_{nz}	N_{nzr}
1	scircuit	170,998	958,936	5.61
2	qcd5_4	49,152	1,916,928	39.00
3	pdb1HYS	36,417	4,344,765	119.31
4	Hamrle3	1,447,360	5,514,242	3.81
5	G3_circuit	1,585,478	7,660,826	4.83
6	shipsec1	140,874	7,813,404	55.46
7	pwtk	217,918	11,634,424	53.39
8	kkt_power	2,063,494	14,612,663	7.08
9	Si41Ge41H72	185,639	15,011,265	80.86
10	bundle_adj	513,351	20,208,051	39.36
11	msdoor	415,863	20,240,935	48.67
12	scai1∗	3,405,035	24,027,759	7.06
13	Fault_639	638,802	28,614,564	44.79
14	af_shell10	1,508,065	52,672,325	34.93
15	HPCG-128-128-128	2,097,152	55,742,968	26.58
16	Serena	1,391,349	64,531,701	46.38
17	bone010	986,703	71,666,325	72.63
18	audikw_1	943,695	77,651,847	82.28
19	channel-500x100x100-b050	4,802,000	85,362,744	17.78
20	rrze3∗	6,201,600	92,527,872	14.92
21	nlpkkt120	3,542,400	96,845,792	27.34
22	delaunay_n24	16,777,216	100,663,202	6.00
23	ML_Geer	1,504,002	110,879,972	73.72
24	FreeFermionChain-26∗	10,400,600	140,616,112	13.52
25	Spin-26∗	10,400,600	145,608,400	14.00
26	scai2∗	22,786,800	160,222,796	7.03

C. L. Alappat et al.: *ECM modeling and performance tuning of SpMV and Lattice QCD on A64FX.* Submitted to CCPE. Preprint: arXiv:2103.03013