NHR FAU

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

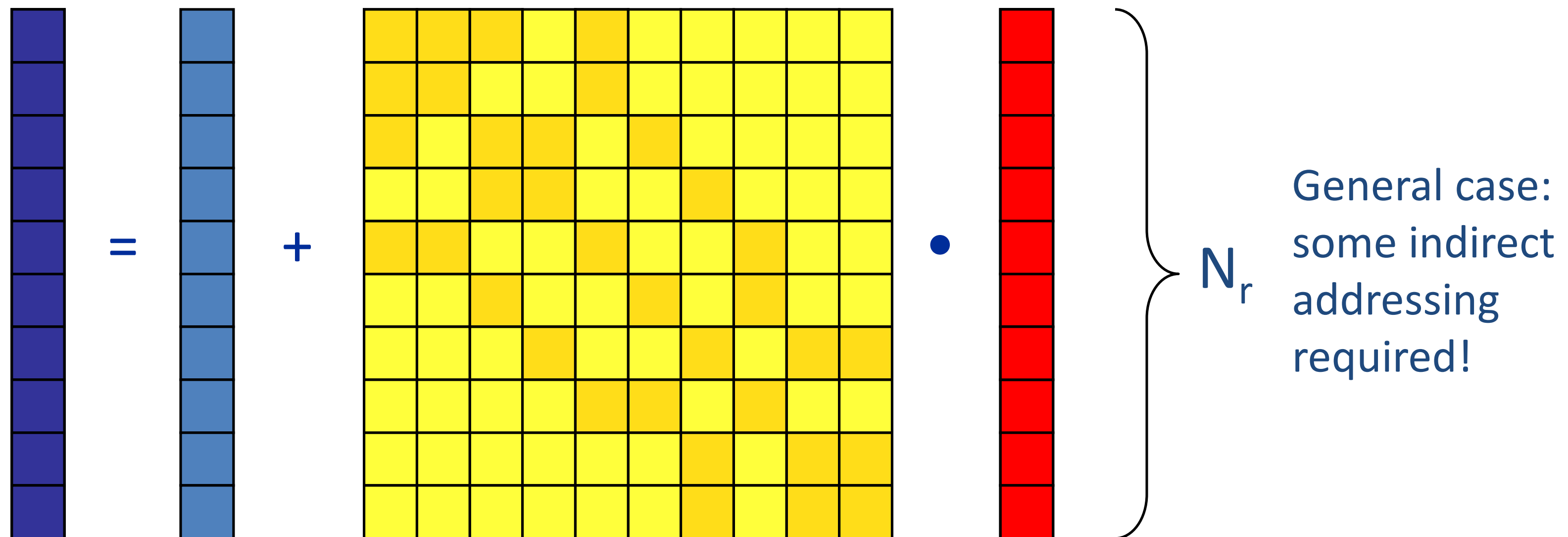
Case study:
Sparse Matrix-Vector Multiplication

HPC High Performance Computing

Sparse Matrix Vector Multiplication (SpMV)

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

- Key ingredient in some matrix diagonalization algorithms
  - Lanczos, Davidson, Jacobi-Davidson
- Store only N_{nz} nonzero elements of matrix and RHS, LHS vectors with N_r (number of matrix rows) entries
- "Sparse": $N_{nz} \sim N_r$
- Average number of nonzeros per row: $N_{nzr} = N_{nz}/N_r$



SpMVM characteristics

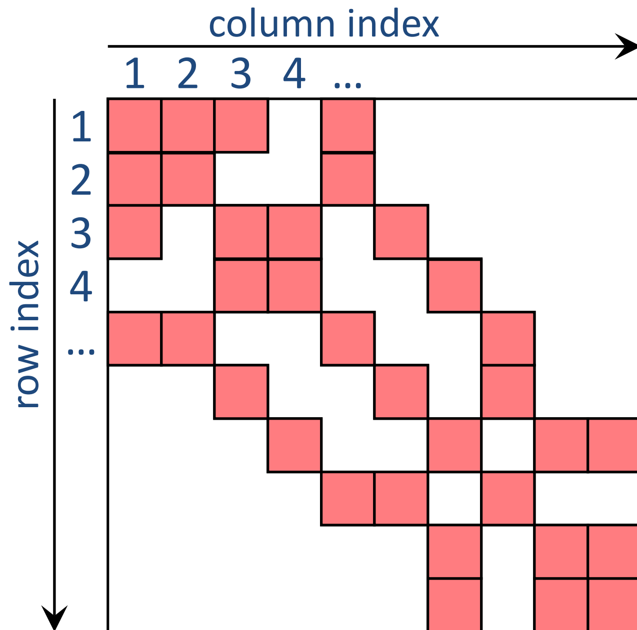
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

- For large problems, SpMV is inevitably memory-bound
  - Intra-socket saturation effect on modern multicores

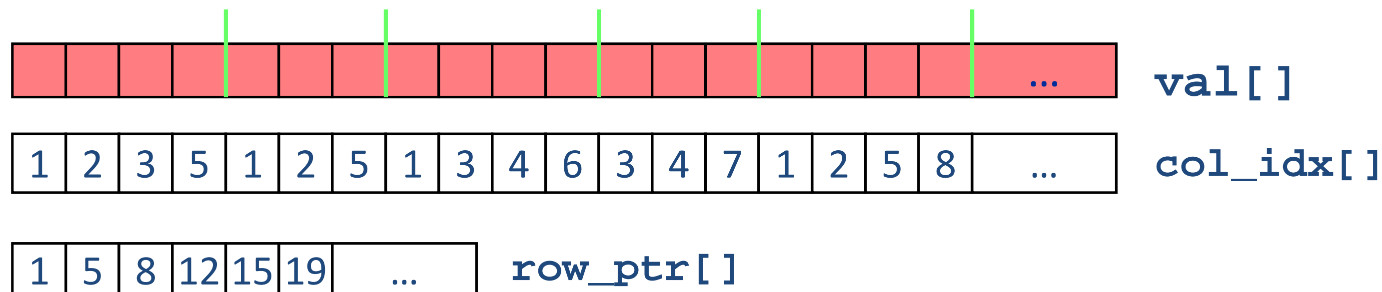
- SpMV is easily parallelizable in shared and distributed memory
  - Load balancing
  - Communication overhead

- Data storage format is crucial for performance properties
  - Most useful general format on CPUs:
    Compressed Row Storage (CRS)
  - Depending on compute architecture

CRS matrix storage scheme



- `val[]` stores all the nonzeros (length N_{nz})
- `col_idx[]` stores the column index of each nonzero (length N_{nz})
- `row_ptr[]` stores the starting index of each new row in `val[]` (length: N_r)



Case study: Sparse matrix-vector multiply

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

- Strongly memory-bound for large data sets
  - Streaming, with partially indirect access:

```
!$OMP parallel do schedule(???)
do i = 1,Nr
 do j = row_ptr(i), row_ptr(i+1) - 1
  C(i) = C(i) + val(j) * B(col_idx(j))
 enddo
enddo
!$OMP end parallel do
```

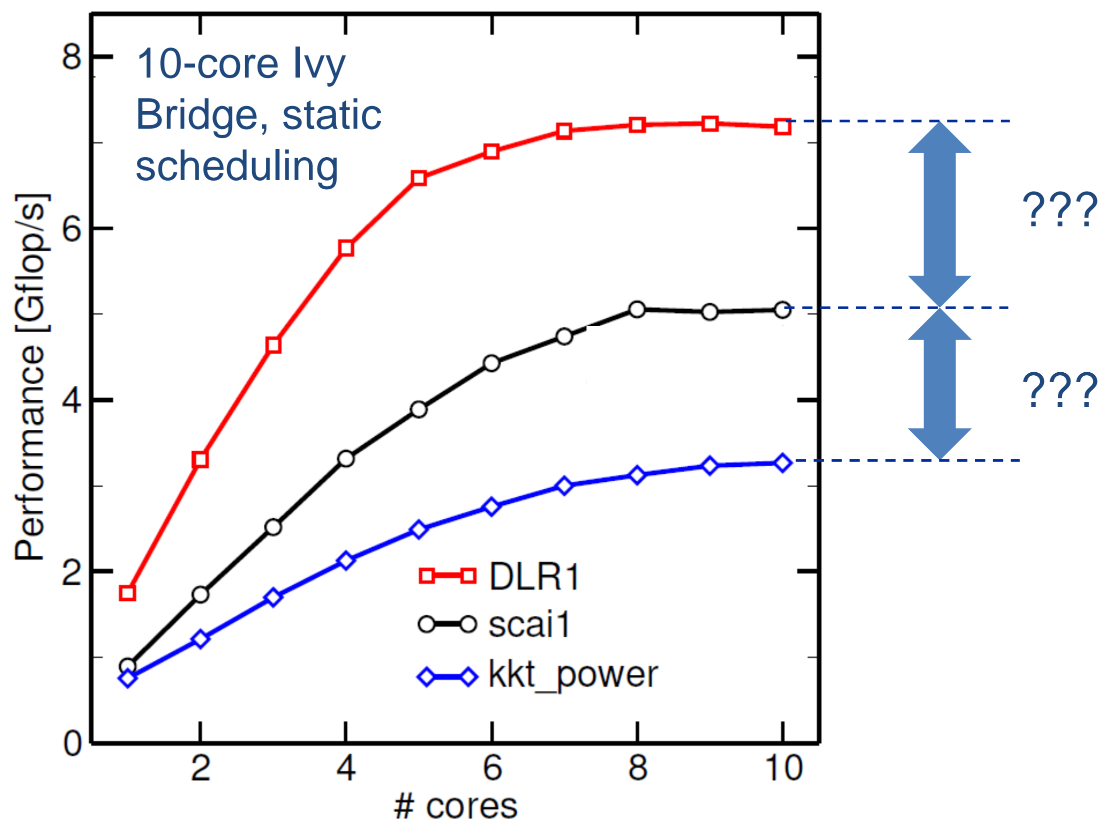
  - Usually many spMVMs required to solve a problem

- Now let’s look at some performance measurements…

Performance characteristics

- Strongly memory-bound for large data sets → saturating performance across cores on the chip
- Performance seems to depend on the matrix

- Can we explain this?
- Is there a “light speed” for SpMV?
- Optimization?



SpMV node performance model – CRS (1)

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

```
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
```

```
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]
```

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]: $2\,N_{nz}$

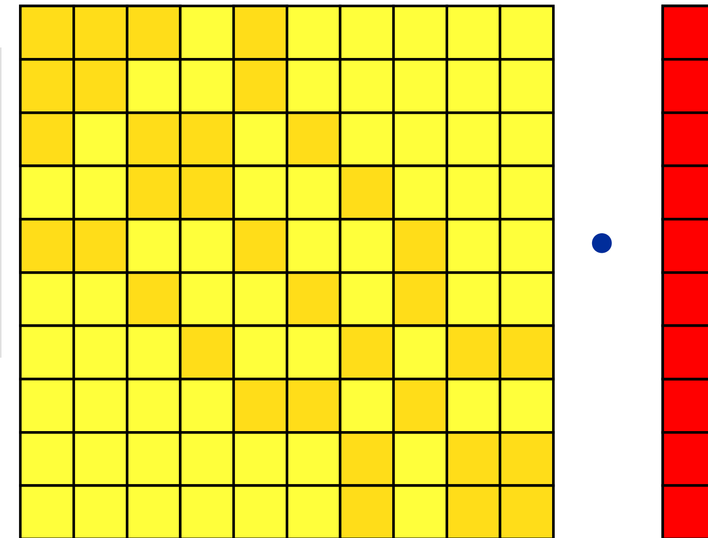
$$B_{C,min} = \frac{12\,N_{nz} + 20\,N_r + 8\,N_c}{2\,N_{nz}}\,\frac{B}{F} = \frac{12 + 20/N_{nzr} + 8/N_{nzc}}{2}\,\frac{B}{F}$$

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

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

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
```



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

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

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

The "α effect"

FAU FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

DP CRS code balance

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

$$= \left(6 + 4\,\alpha + \frac{10}{N_{nzr}}\right)\frac{B}{F}$$

- α quantifies the traffic for loading the RHS
  - $\alpha = 0$ → RHS is in cache
  - $\alpha = 1/\mathrm{N_{nzr}}$ → RHS loaded once
  - $\alpha = 1$ → no cache
  - $\alpha > 1$ → Houston, we have a problem!
- "Target" performance = b_S/B_c
- Caveat: Maximum memory BW may not be achieved with spMVM (see later)

Can we predict α?

- Not in general
- Simple cases (banded, block-structured): Similar to layer condition analysis

→ Determine α by measuring the actual memory traffic
(→ measured code balance B_C^{meas})

Determine α (RHS traffic quantification)

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

- V_{meas} is the measured overall memory data traffic (using, e.g., likwid-perfctr)
- Solve for α:

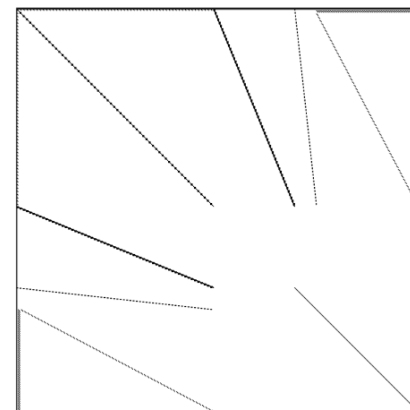
$$\alpha = \frac{1}{4}\left(\frac{V_{meas}}{N_{nz}\cdot 2\ \text{bytes}} - 6 - \frac{10}{N_{nzr}}\right)$$

Example: kkt_power matrix from the UoF collection on one Intel SNB socket

- $N_{nz} = 14.6\cdot 10^6$, $N_{nzr} = 7.1$
- $V_{meas} \approx 258$ MB
- → $\alpha = 0.36$, $\alpha N_{nzr} = 2.5$
- → RHS is loaded 2.5 times from memory
- and:

$$\frac{B_C(\alpha)}{B_{C,min}} = 1.11$$

11% extra traffic → optimization potential!

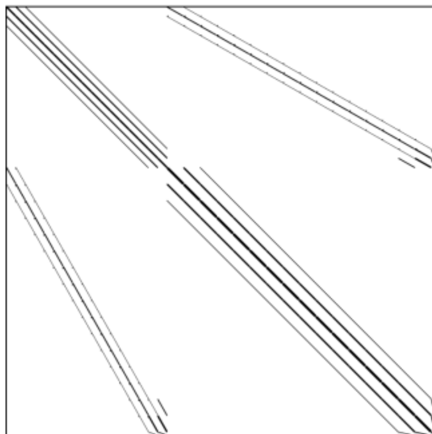


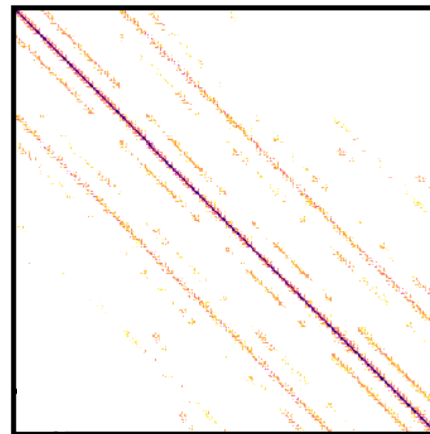
Three different sparse matrices

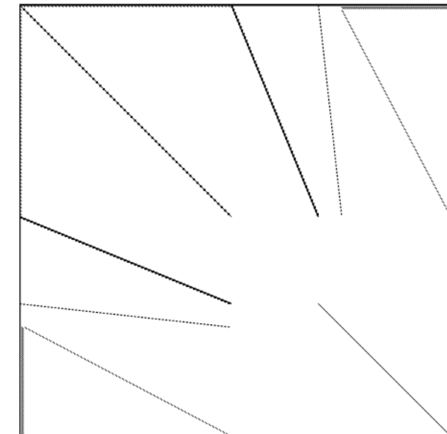
Benchmark system: Intel Xeon Ivy Bridge E5-2660v2, 2.2 GHz, $b_S = 46.6\,\mathrm{GB/s}$

→ Roofline: $P_{opt} = {b_S}/{B_{C,min}}$

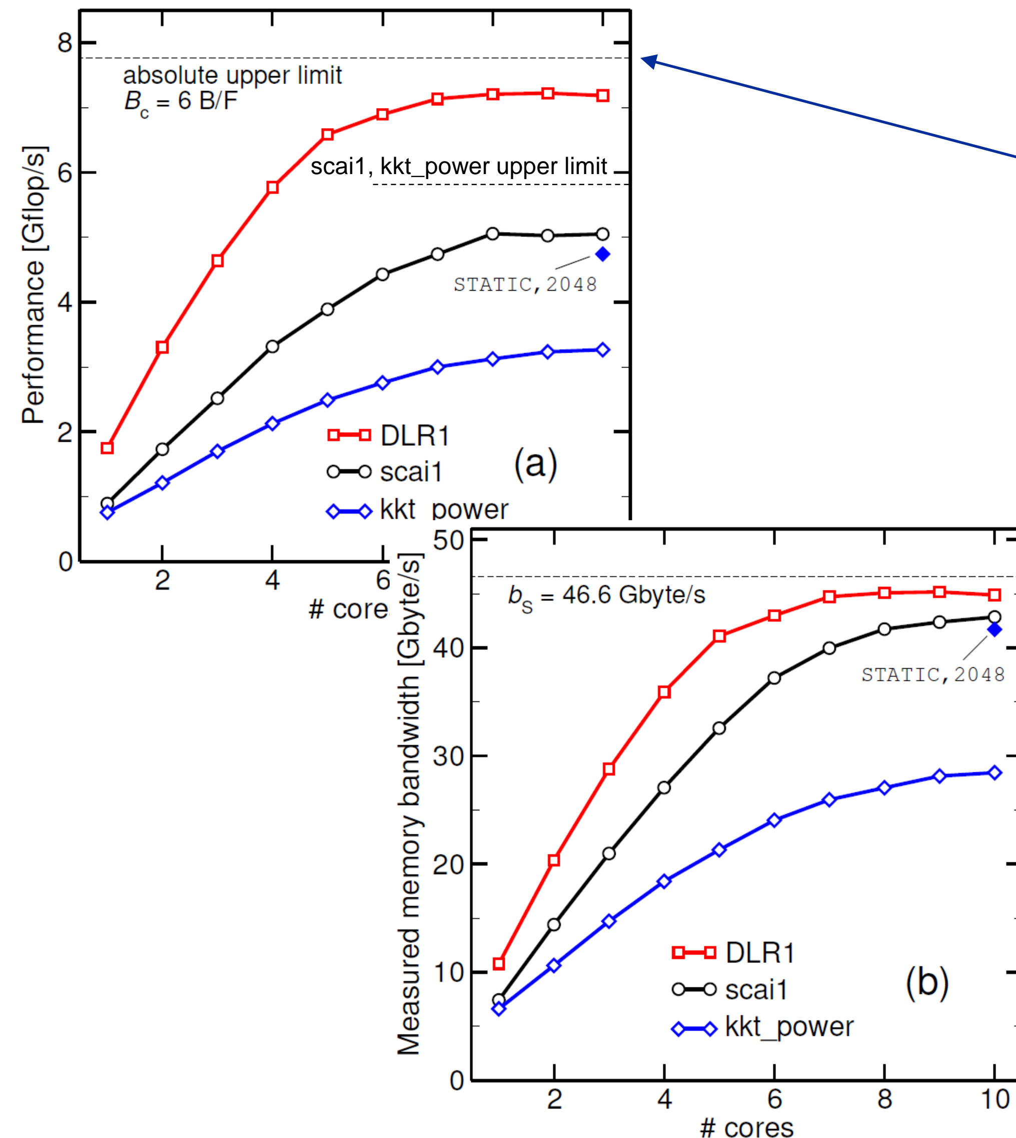
Matrix	N	N_{nzr}	$B_{C,min}$ [B/F]	P_{opt} [GF/s]
DLR1	278,502	143	6.1	7.64
scai1	3,405,035	7.0	8.0	5.83
kkt_power	2,063,494	7.08	8.0	5.83


DLR1


scai1


kkt_power

Now back to the start…



- $b_S = 46.6\,\mathrm{GB/s}, \quad B_c = 6\,\mathrm{B/F}$
- Maximum spMVM performance:

$$P_{max} = 7.8\,\mathrm{GF/s}$$

- **DLR1** causes minimum CRS code balance (as expected)
- **scai1** measured balance:

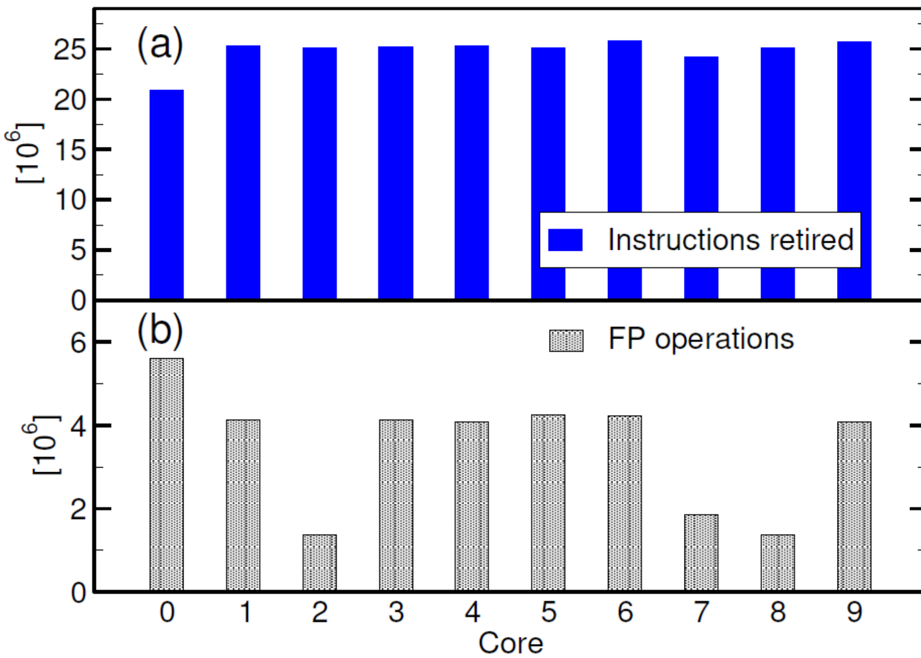
$$B_c^{meas} \approx 8.5\,\mathrm{B/F} > B_{C,min}$$

- → good BW utilization, slightly non-optimal α
- **kkt_power** measured balance:

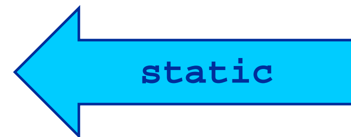
$$B_c^{meas} \approx 8.8\,\mathrm{B/F} > B_{C,min}$$

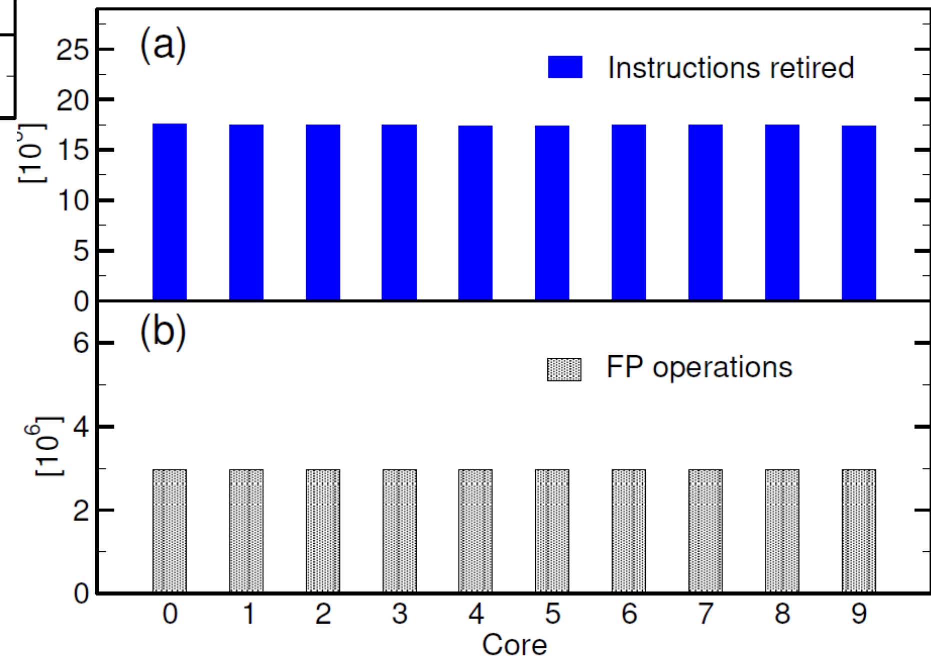
- → performance degraded by load imbalance, fix by block-cyclic schedule

Investigating the load imbalance with kkt_power



Measurements with likwid-perfctr (MEM_DP group)

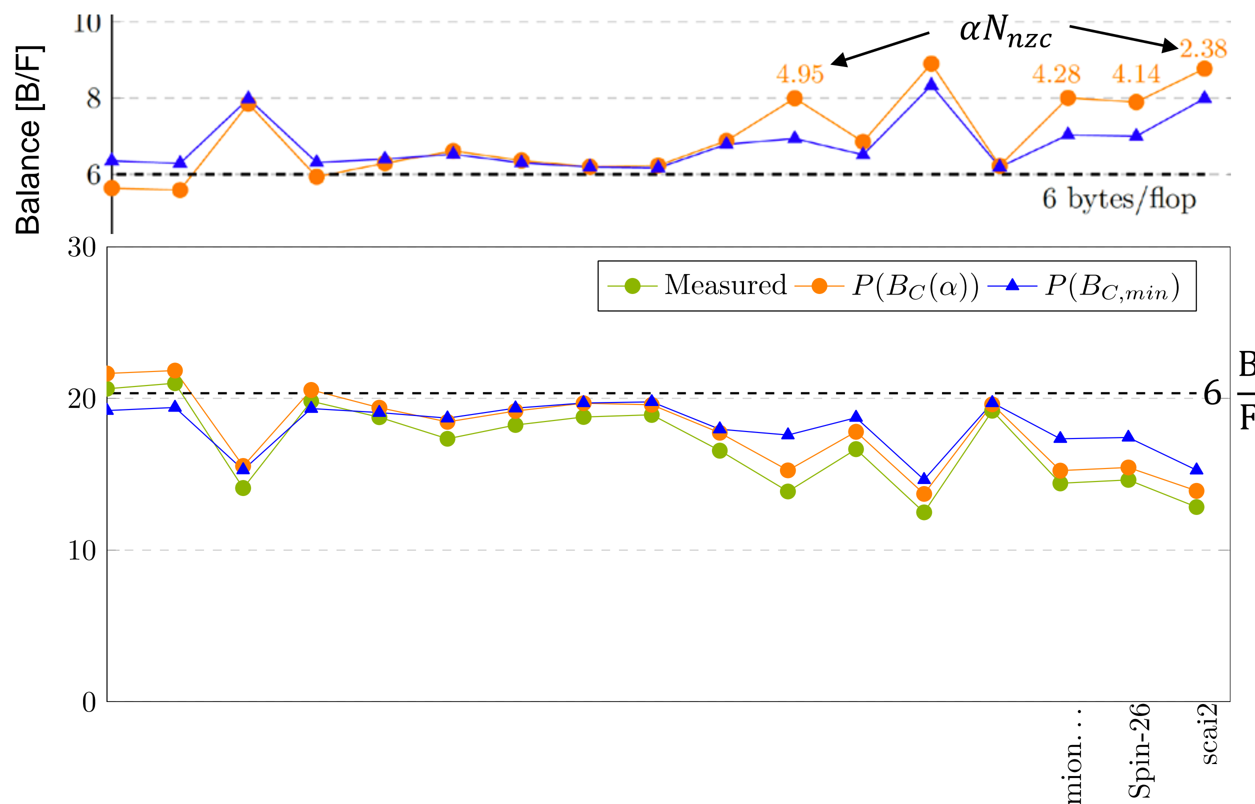




→ Fewer overall instructions, (almost) BW saturation, 50% better performandce with load balancing

→ CPI value unchanged!

SpMV node performance model – CPU



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

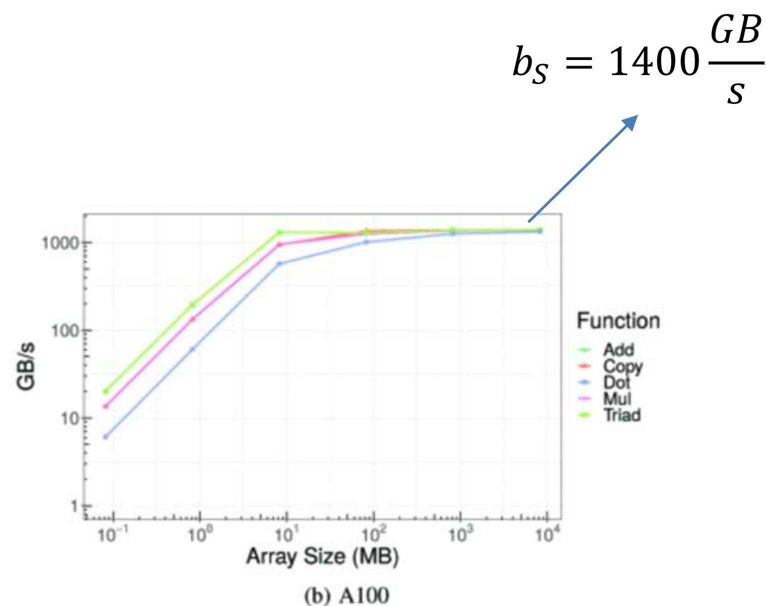
Matrices taken from: C. L. Alappatet al.: *ECM modeling and performance tuning of SpMV and Lattice QCD on A64FX.* Submitted. Preprint: arXiv:2103.0301

SpMV node performance model – GPU

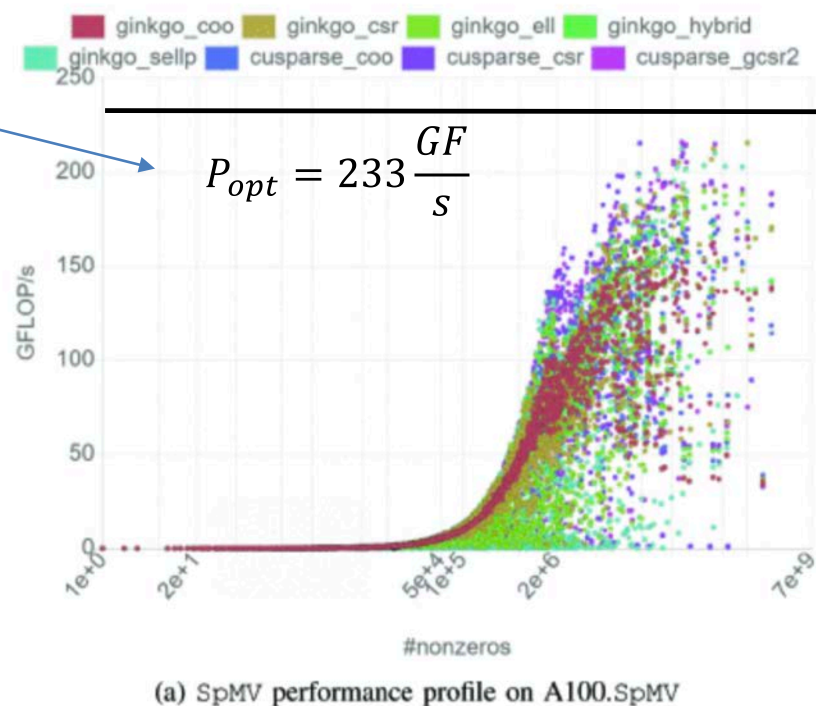
SpMV on NVIDIA A100:

- Different data formats and libraries
- 2800 of matrices (SuiteSparse Matrix collection)

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



(b) A100

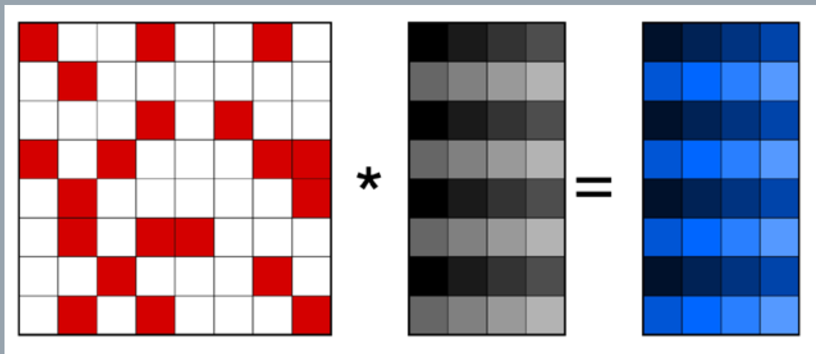


$$P_{opt} = 233 \frac{GF}{s}$$

(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.

Applying sparse matrix to multiple vectors (Sparse Matrix Multiple Vectors; SpMMV)



High Performance Computing

Multiple RHS vectors (SpMMV)

Unchanged matrix applied to multiple RHS (**r**) vectors to yield multiple LHS (**r**) vectors

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

B_c unchanged, no reuse of matrix data

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

Higher B_c due to max reuse of matrix data

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

CL-friendly data structure (row major)

SpMMV code balance

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

One complete inner (**s**) loop traversal:

- $2r$ flops
- 12 bytes from matrix data (value + index)
- $\frac{16r}{N_{nzr}}$ bytes from the r LHS updates
- $\frac{4}{N_{nzr}}$ bytes from the row pointer
- $8r\alpha(r)$ bytes from the r RHS reads

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

$$B_c(r) = \frac{1}{2r}\left(12 + 8r\alpha(r) + \frac{16r+4}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

$$= \left(\frac{6}{r} + 4\alpha(r) + \frac{8+2/r}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

OK so what now???

SpMMV code balance

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

Let's check some limits to see if this makes sense!

$$B_c(r) = \left(\frac{6}{r} + 4\alpha(r) + \frac{8 + 2/r}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

$r = 1$:

$$\left(6+4\alpha+\frac{10}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

reassuring ☺

$r \gg 1$:

$$\left(4\alpha(r) + \frac{8}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

Can become very small for large N_{nzr} → decoupling from memory bandwidth is possible!

$N_{nzr} \gg 1$:

$$\frac{6}{r}\frac{\mathrm{B}}{\mathrm{F}}$$

M. Kreutzer et al.: *Performance Engineering of the Kernel Polynomial Method on Large-Scale CPU-GPU Systems*. Proc. IPDPS15, DOI: 10.1109/IPDPS.2015.76

Roofline analysis for spMVM

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

- Conclusion from the Roofline analysis
  - The roofline model does not “work” for spMVM due to the RHS traffic uncertainties
  - We have “turned the model around” and measured the actual memory traffic to determine the RHS overhead
  - Result indicates:
    1. how much actual traffic the RHS generates
    2. how efficient the RHS access is (compare BW with max. BW)
    3. how much optimization potential we have with matrix reordering
- Do not forget about load balancing!
- Sparse matrix times multiple vectors bears the potential of huge savings in data volume

- Consequence: Modeling is not always 100% predictive. It‘s all about *learning more* about performance properties!