Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# Case study: Sparse Matrix-Vector Multiplication

Erlangen Regional Computing Center

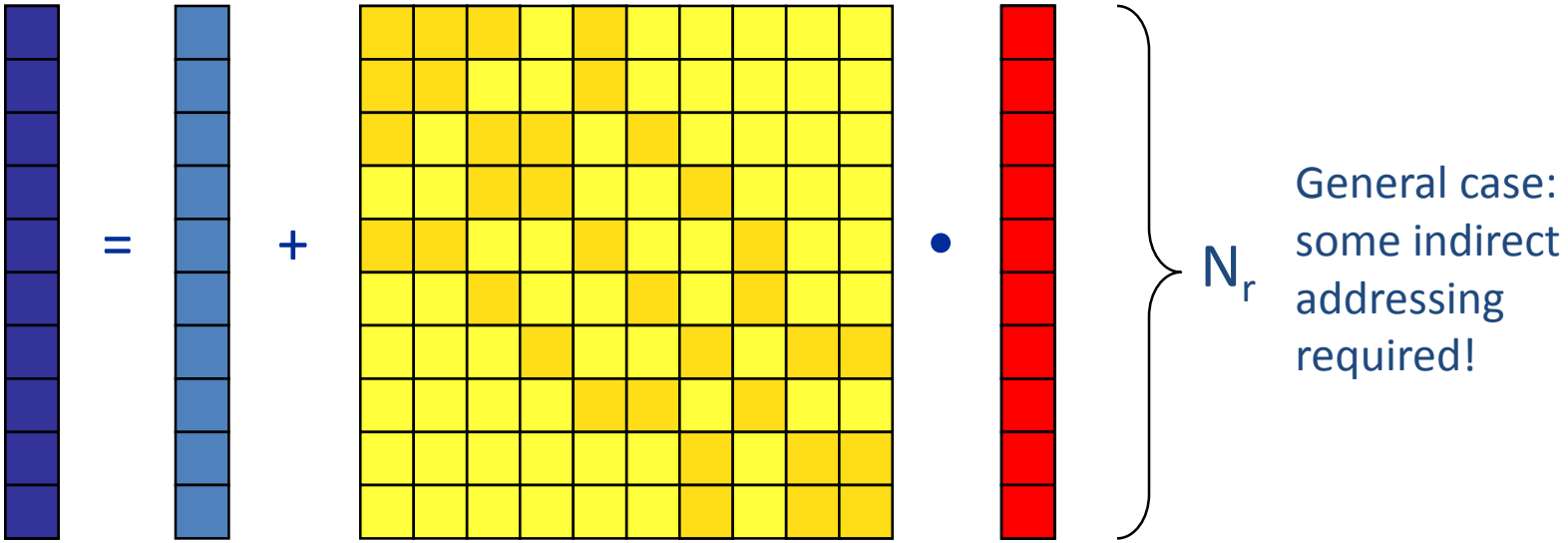
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# SpMVM: The Basics

High Performance Computing

# Sparse Matrix Vector Multiplication (SpMV)

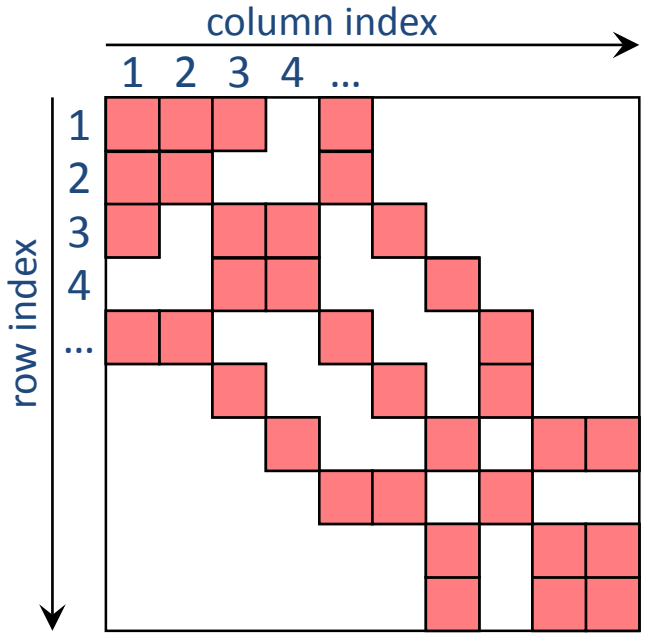
- Key ingredient in some matrix diagonalization algorithms
  - Lanczos, Davidson, Jacobi-Davidson
- Store only $N_{nz}$ nonzero elements of matrix and RHS, LHS vectors with $N_r$ (number of matrix rows) entries
- "Sparse": $N_{nz} \sim N_r$
- Average number of nonzeros per row: $N_{nzr} = N_{nz}/N_r$



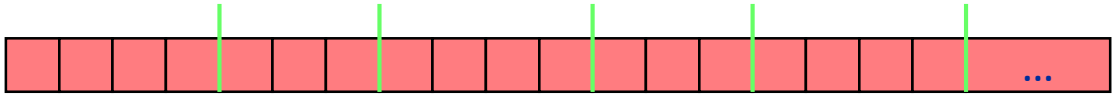
# SpMVM characteristics

- 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: $N_r$)




**`val[]`**

| 1 | 2 | 3 | 5 | 1 | 2 | 5 | 1 | 3 | 4 | 6 | 3 | 4 | 7 | 1 | 2 | 5 | 8 | ... |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

**`col_idx[]`**

| 1 | 5 | 8 | 12 | 15 | 19 | ... |
|---|---|---|---|---|---|---|

**`row_ptr[]`**

# Case study: Sparse matrix-vector multiply

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

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

```
!$OMP parallel do schedule(???)
do i = 1,N_r
 do j = row_ptr(i), row_ptr(i+1) - 1
  c(i) = c(i) + val(j) * b(col_idx(j))
 enddo
enddo
!$OMP end parallel do
```

  - Usually many spMVMs required to solve a problem

- Now let's look at some performance measurements…

Erlangen Regional Computing Center

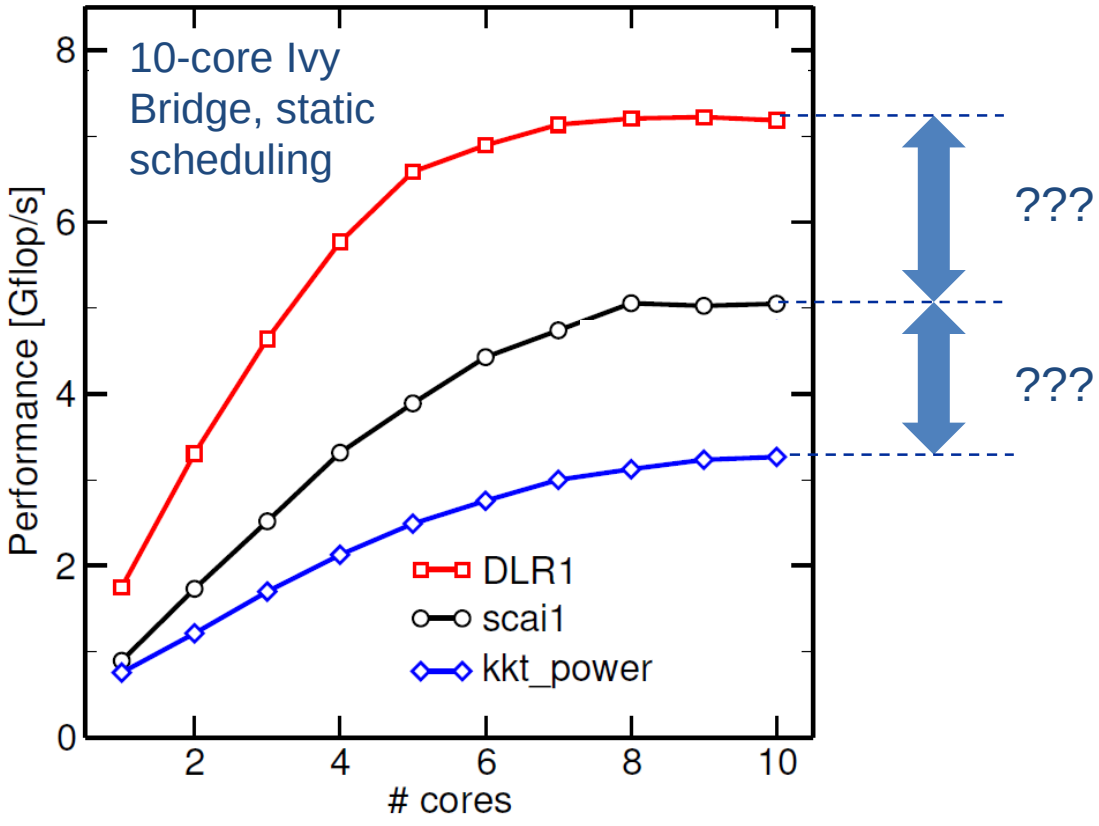
FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# SpMVM: Performance Analysis

# 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

Sparse MVM in
double precision
w/ CRS data storage:

```
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
```

$$B_c^{DP,CRS} = \frac{8+4+8\alpha+20/N_{nzr}}{2}\frac{\mathrm{B}}{\mathrm{F}} = \left(6+4\alpha+\frac{10}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

Absolute minimum code balance: $B_{\min} = 6\ \frac{\mathrm{B}}{\mathrm{F}}$

$\rightarrow I_{\max} = \frac{1}{6}\ \frac{\mathrm{F}}{\mathrm{B}}$

Hard upper limit for in-memory performance: $b_S/B_{\min}$

# The "$\alpha$ effect"

DP CRS code balance

- $\alpha$ quantifies the traffic for loading the RHS
  - $\alpha$ = 0 → RHS is in cache
  - $\alpha$ = 1/$N_{nzr}$ → 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)

$$B_c^{DP,CRS}(\alpha) = \frac{8+4+8\alpha+20/N_{nzr}}{2}\frac{\mathrm{B}}{\mathrm{F}} = \left(6+4\alpha+\frac{10}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}}$$

Can we predict $\alpha$?

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

→ Determine $\alpha$ by measuring the actual memory traffic

# Determine $\alpha$ (RHS traffic quantification)

$$B_c^{DP,CRS} = \left(6+4\alpha+\frac{10}{N_{nzr}}\right)\frac{\mathrm{B}}{\mathrm{F}} = \frac{V_{meas}}{N_{nz}\cdot 2\ \mathrm{F}}$$

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

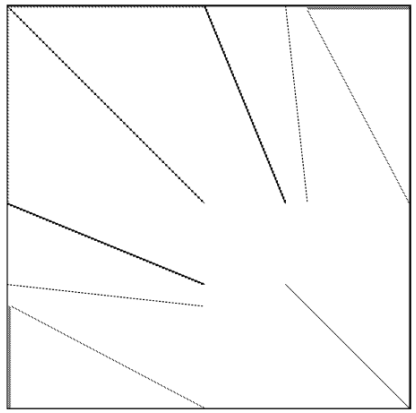
$$\alpha = \frac{1}{4}\left(\frac{V_{meas}}{N_{nz}\cdot 2\ \text{bytes}} - 6 - \frac{10}{N_{nzr}}\right)$$

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

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

$$\frac{B_c^{DP,CRS}(\alpha)}{B_c^{DP,CRS}(1/N_{nzr})} = 1.11$$

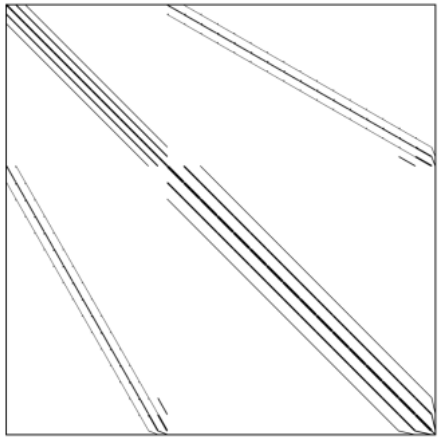
11% extra traffic → optimization potential!



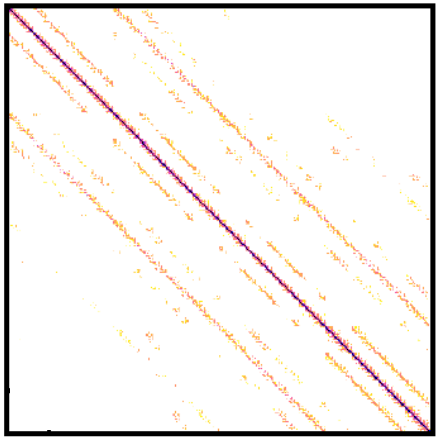
# Three different sparse matrices

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

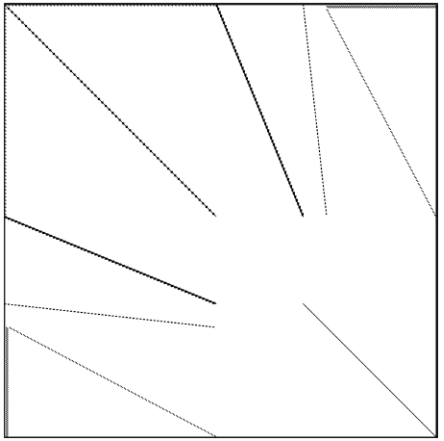
| **Matrix** | $N$ | $N_{nzr}$ | $B_c^{opt}$ [B/F] | $P_{opt}$ [GF/s] |
|---|---|---|---|---|
| DLR1 | 278,502 | 143 | 6.1 | 7.64 |
| scai1 | 3,405,035 | 7.0 | 8.0 | 5.83 |
| kkt_power | 2,063,494 | 7.08 | 8.0 | 5.83 |



DLR1

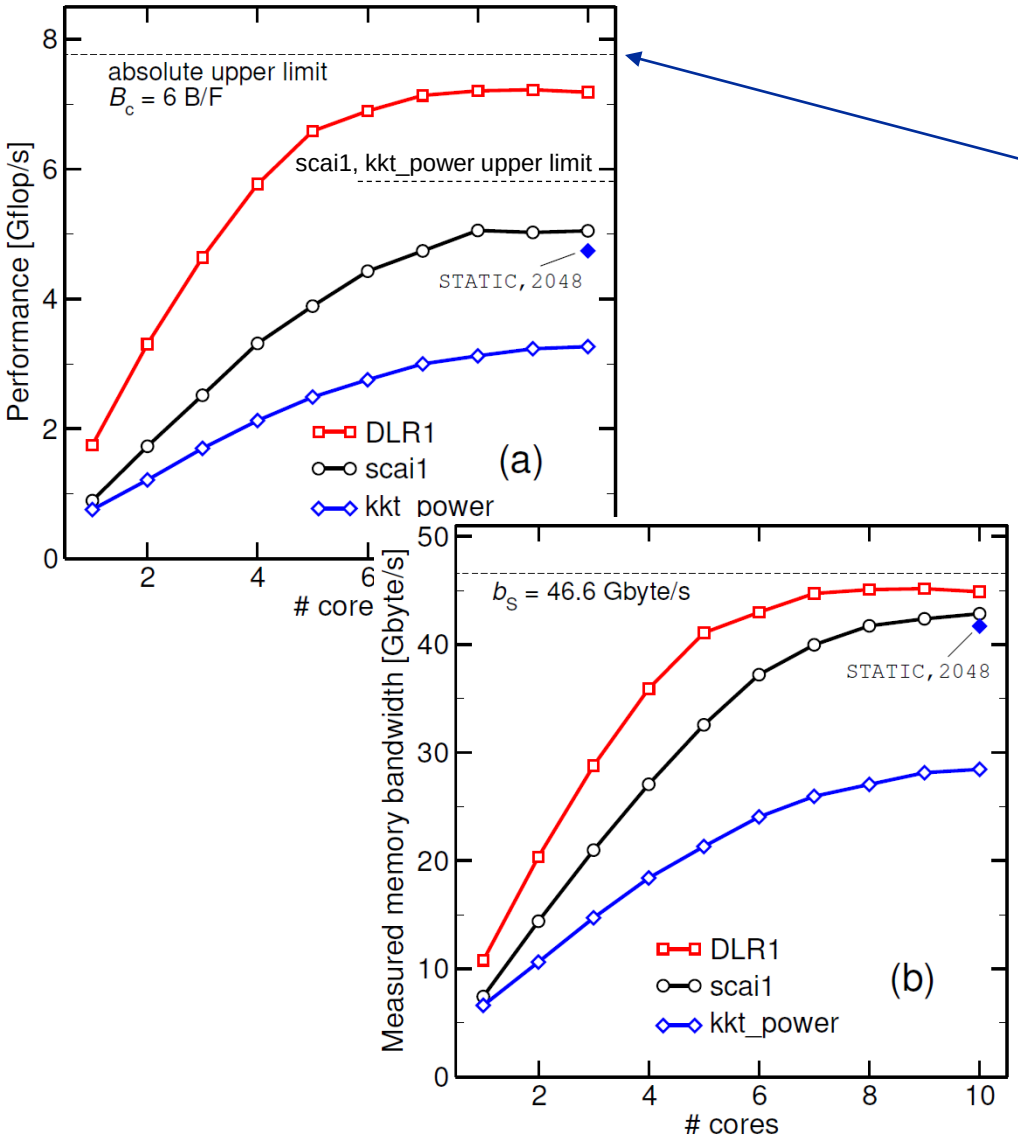


scai1



kkt_power

# Now back to the start…




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

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

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

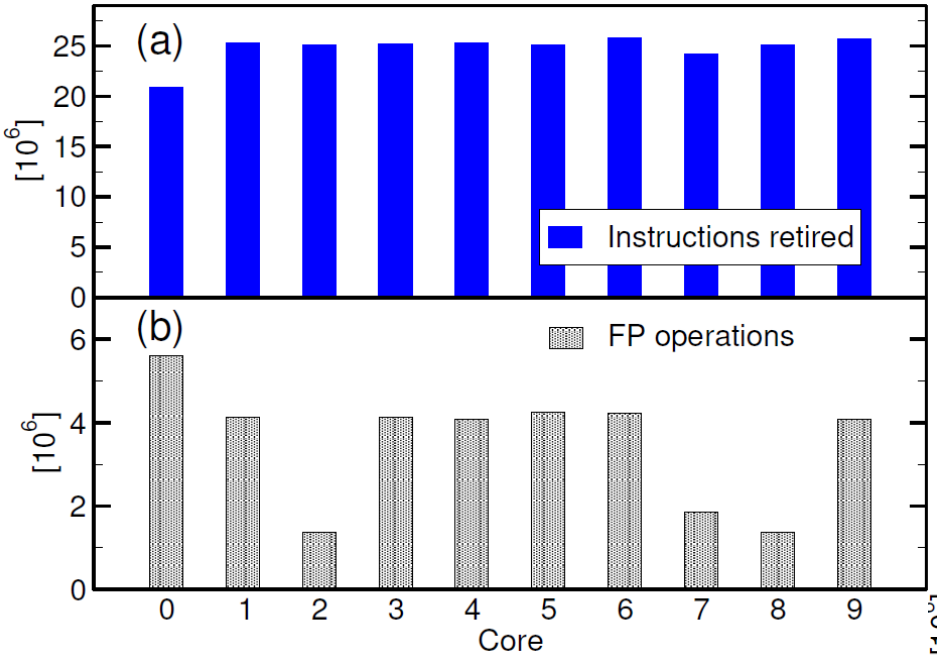
$$B_c^{meas} \approx 8.5\,\mathrm{B/F} > B_c^{opt}$$

- → good BW utilization, slightly non-optimal $\alpha$
- **kkt_power** measured balance:

$$B_c^{meas} \approx 8.8\,\mathrm{B/F} > B_c^{opt}$$

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

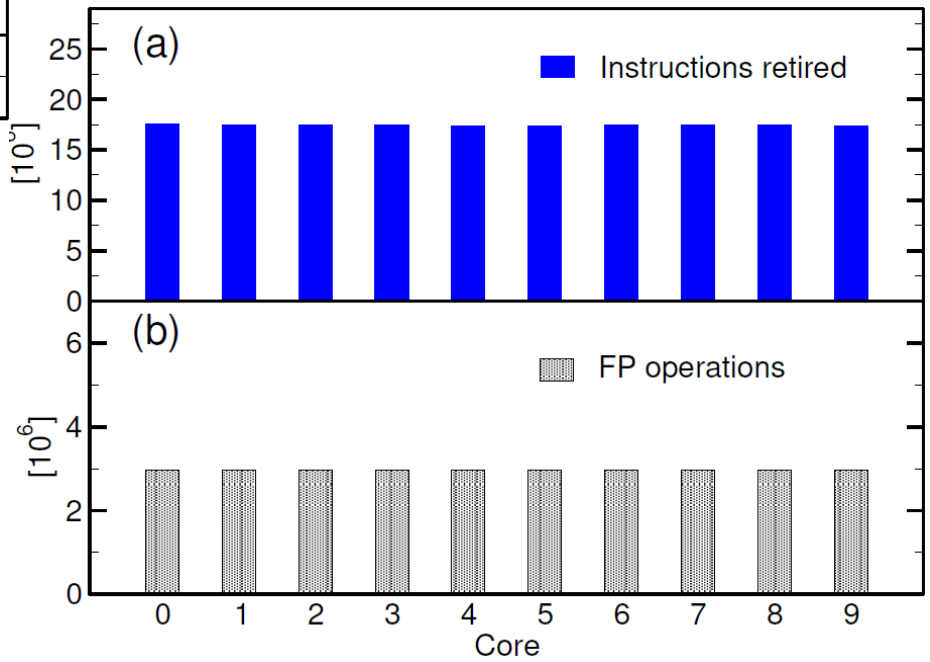
# Investigating the load imbalance with kkt_power




Measurements with likwid-perfctr (MEM_DP group)

`static`

`static,2048`



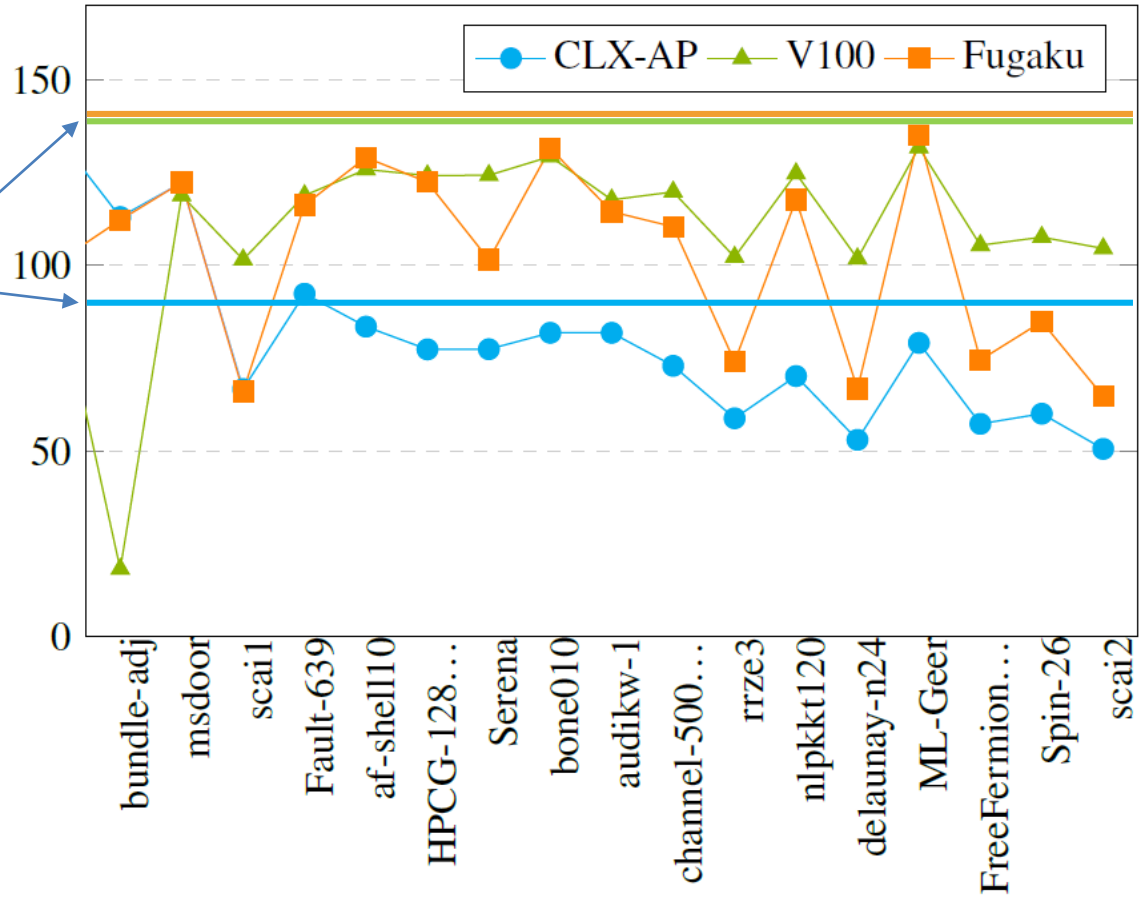

→ Fewer overall instructions, (almost) BW saturation, 50% better performandce with load balancing

→ CPI value unchanged!

# CPU performance comparison

- Cascade-Lake AP: $b_S = 500$ Gbyte/s
- NVIDIA V100: $b_S = 840$ Gbyte/s
- Fujitsu A64FX in Fugaku: $b_S = 859$ Gbyte/s

Absolute upper limits (matrix independent) given by $b_S/(6\ \mathrm{B/F})$




C. L. Alappat et al., DOI: 10.1002/cpe.6512

Erlangen Regional Computing Center

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

# SpMVM with multiple RHS & LHS Vectors

# Multiple RHS vectors (SpMMV)

Unchanged matrix applied to multiple RHS vectors to yield multiple LHS 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
```

Lower $B_c$ 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

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

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 ☺

$N_{nzr} \gg 1$ ↓

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

$r \ll 1$ ↘

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

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

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

- 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!