REGIONALES RECHENZENTRUM ERLANGEN [RRZE]

Some observations on NEC Aurora Tsubasa (10B)

Stencils and sparse matrix-vector multiplication

Georg Hager

Erlangen Regional Computing Center (RRZE)

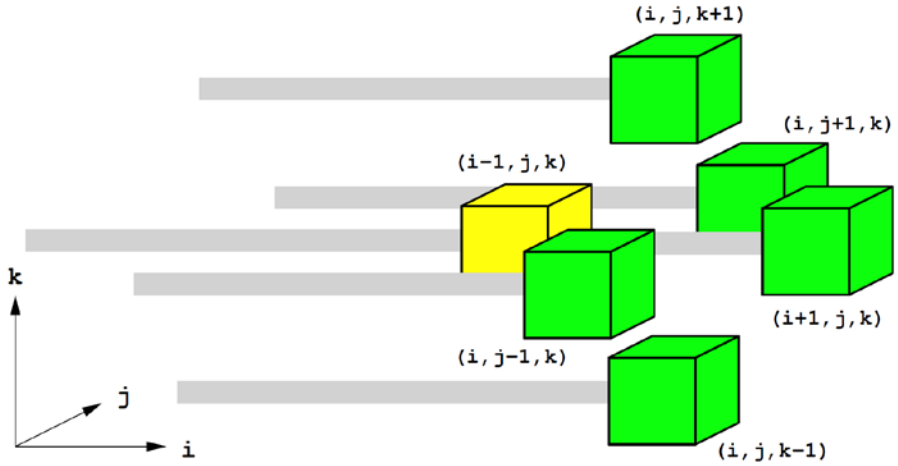
University of Erlangen-Nürnberg

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG

3D 7-pt stencil: recap

Layer conditions (LC) in 3D

- "Outer" LC:
  `3 * imax * jmax`
- "Inner" LC:
  `3 * imax` in the central layer



→ Code balance (incl. write-allocate):

- 16+8 B/LUP if outer LC fulfilled
- 32+8 B/LUP if outer LC broken but inner LC fulfilled
- 48+8 B/LUP if inner & outer LC broken

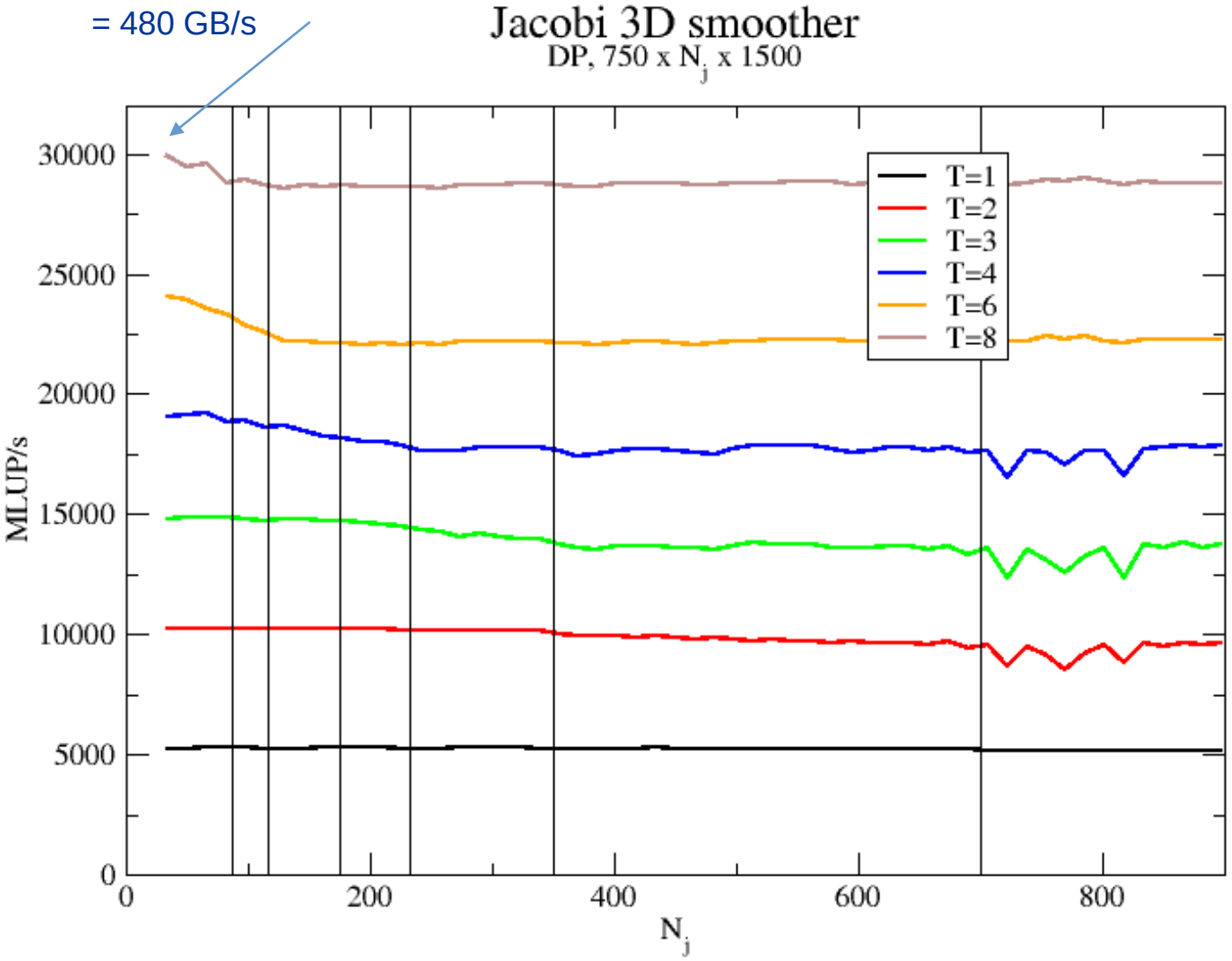
Code

```
52: vec( 109): Vectorization obstructive statement.
52: par(1803): Parallelized by "do".
55: opt(1409): Alternate code generated.: I
55: vec( 101): Vectorized loop.
56: opt(1036): Potential feedback - use directive or compiler option
```

```
47: |         !$omp parallel firstprivate(t0,t1) private(n,jj,ii,k,j,t)
48: |+----->     do n=1,iter
49: ||+---->        do jj=2,njsize-1,bj
51: |||       !$omp do schedule(runtime)
52: |||P--->        do k=2,nksize-1
53: ||||+-->           do j=jj,min(jj+bj-1,njsize-1)
54: |||||     !NEC$ ivdep
55: |||||V->              do i=2,nisize-1
56: ||||||                   phi(i,j,k,t1) = oos*(phi(i-1,j,k,t0)+phi(i+1,j,k,t0)+ &
57: ||||||                                        phi(i,j-1,k,t0)+phi(i,j+1,k,t0)+ &
58: ||||||                                        phi(i,j,k-1,t0)+phi(i,j,k+1,t0))
59: |||||V-              enddo
60: |||||
63: ||||+--           enddo
64: |||P---         enddo
65: |||       !$omp end do
67: |||             call dummy(phi(1,1,1,t0),phi(1,1,1,t1))
68: ||+----         enddo
69: ||              t=t0; t0=t1; t1=t
70: |+-----      enddo
71: |         !$omp end parallel
```

Performance vs. system size (LCs)

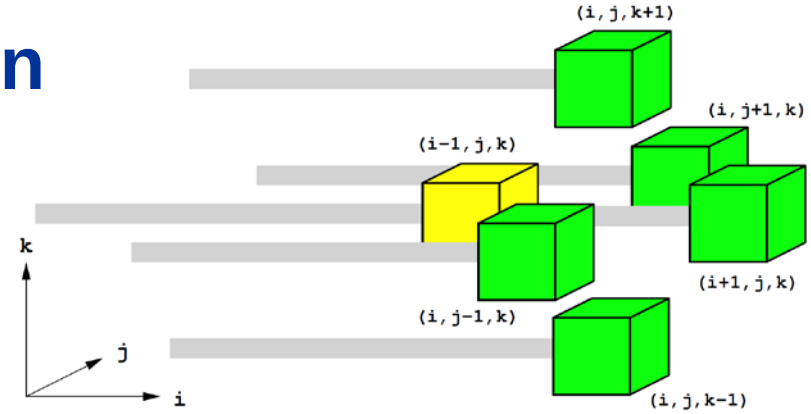
30 GLUP/s * 16 B/LUP
= 480 GB/s



28.5 GLUP/s
* 32 B/LUP
= 912 GB/s

(NEC SCA Lib:
39 GLUP/s)

An attempt at an explanation



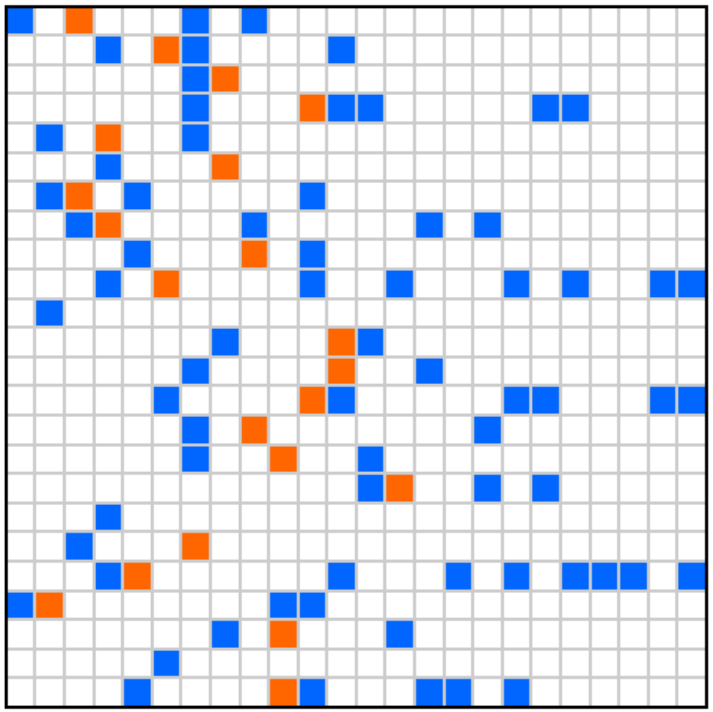
- Assume memory BW $b_S = 900\,\frac{\mathrm{GB}}{\mathrm{s}}$
- Assume LLC BW $b_C = 3\,\frac{\mathrm{TB}}{\mathrm{s}}$
- All LCs satisfied at LLC: $V_{mem} = 16\,\frac{\mathrm{B}}{\mathrm{LUP}}$, $V_{LLC} = 48\,\frac{\mathrm{B}}{\mathrm{LUP}}$
  - Assuming the store does not go through the LLC
- Assuming non-overlapping data transfers Mem ↔ LLC and Reg ↔ LLC, overall transfer time per LUP is

$$T = \frac{16\frac{\mathrm{B}}{\mathrm{LUP}}}{b_S} + \frac{48\frac{\mathrm{B}}{\mathrm{LUP}}}{b_C} \rightarrow P \approx 29.6\,\frac{\mathrm{GLUP}}{\mathrm{s}}$$

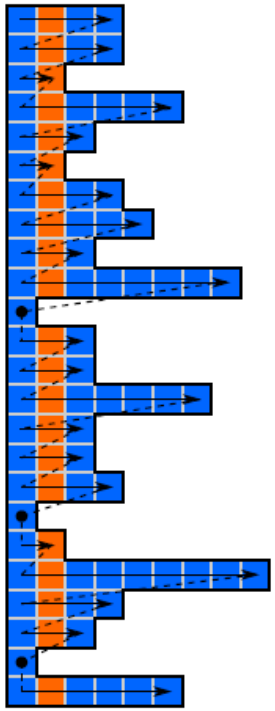
- → indicates that non-overlapping ECM model applies
- → standard spatial blocking will not do
- → register optimizations required!

Sparse MVM with SELL-C-σ

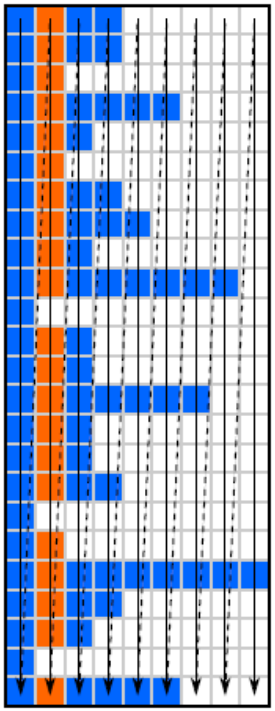
Standard sparse matrix storage formats



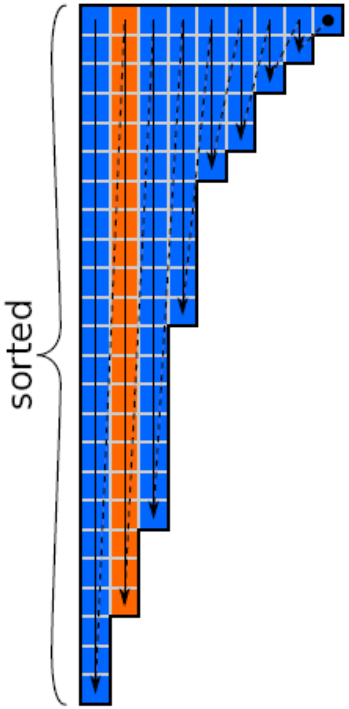
original matrix



CRS



ELLPACK



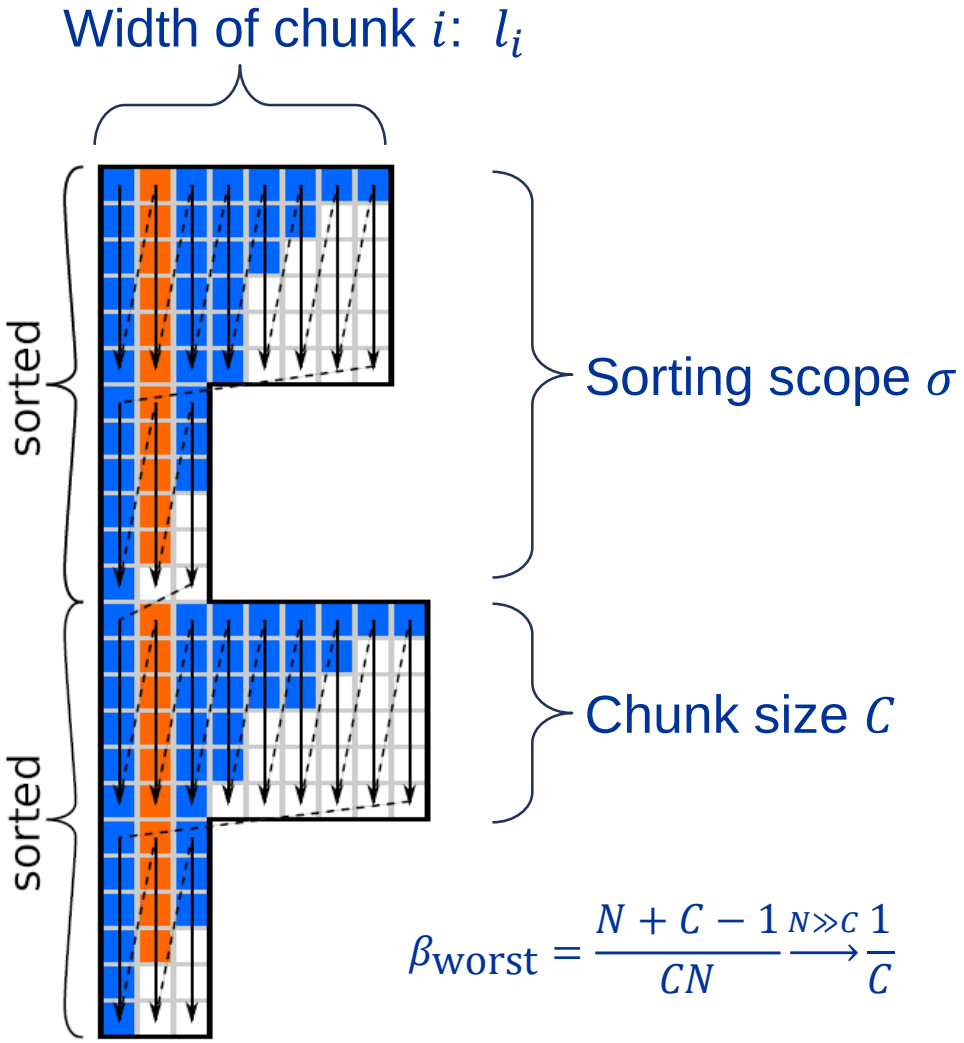
JDS

Enter SELL-C-σ

1. Pick chunk size C (guided by SIMD/T widths)
2. Pick sorting scope σ
3. Sort rows by length within each sorting scope
4. Pad chunks with zeros to make them rectangular
5. Store matrix data in “chunk column major order”

“Chunk occupancy”: fraction of “useful” matrix entries

$$\beta = \frac{N_{nz}}{\sum_{i=0}^{N_c} C \cdot l_i}$$



SELL-6-12
β=0.66

$$\beta_{\text{worst}} = \frac{N + C - 1}{CN} \xrightarrow{N \gg C} \frac{1}{C}$$

Matrix characterization

"Corner case" matrices from "Williams Group":

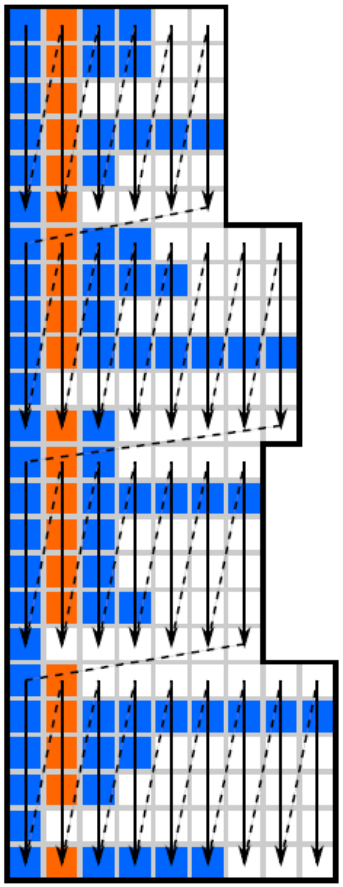
Test case	N	N_{nz}	N_{nzr}	density	$\beta_{\sigma=1}^{C=16}$	$\beta_{\sigma=256}^{C=16}$
RM07R	381,689	37,464,962	98.16	2.57e-04	0.63	0.93
kkt_power	2,063,494	14,612,663	7.08	3.43e-06	0.54	0.92
Hamrle3	1,447,360	5,514,242	3.81	2.63e-06	1.00	1.00
ML_Geer	1,504,002	110,879,972	73.72	4.90e-05	1.00	1.00
pwtk	217,918	11,634,424	53.39	2.45e-04	0.99	1.00
shipsec1	140,874	7,813,404	55.46	3.94e-04	0.89	0.98
consph	83,334	6,010,480	72.13	8.65e-04	0.94	0.97
pdb1HYS	36,417	4,344,765	119.31	3.28e-03	0.84	0.97
cant	62,451	4,007,383	64.17	1.03e-03	0.90	0.98

SELL-C-σ kernel

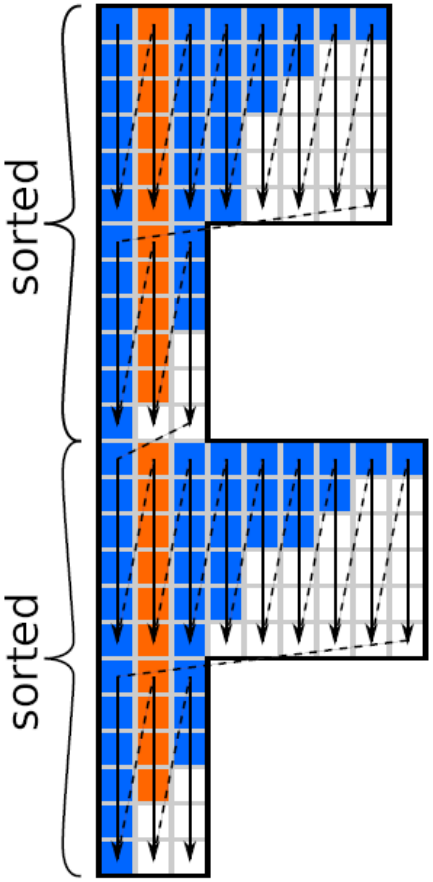
```
for(i = 0; i < N/4; ++i)
{

  for(j = 0; j < cl[i]; ++j)
  {
    y[i*4+0] += val[cs[i]+j*4+0] *
                x[col[cs[i]+j*4+0]];
    y[i*4+1] += val[cs[i]+j*4+1] *
                x[col[cs[i]+j*4+1]];
    y[i*4+2] += val[cs[i]+j*4+2] *
                x[col[cs[i]+j*4+2]];
    y[i*4+3] += val[cs[i]+j*4+3] *
                x[col[cs[i]+j*4+3]];
  }
}
```

$C = 4$

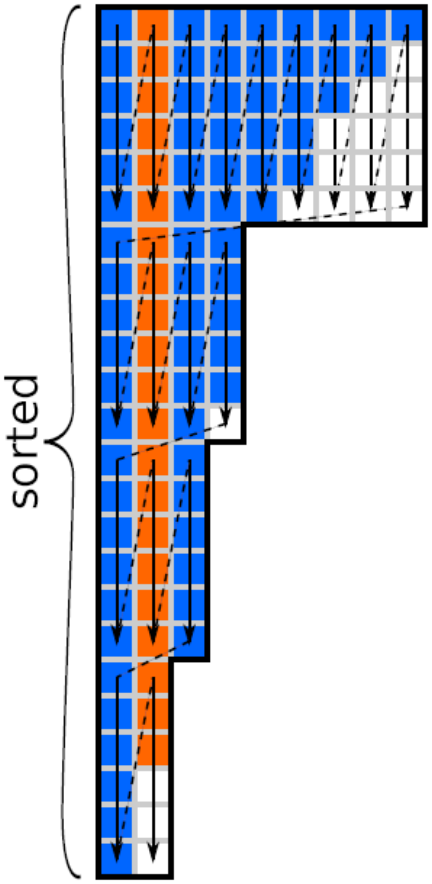
Variants of SELL-C-σ



SELL-6-1
β=0.51



SELL-6-12
β=0.66



SELL-6-24
β=0.84

Roofline performance model for SELL-C-σ

Code balance (double precision FP, 4-byte index):

Matrix data & column index

RHS access

LHS update assuming $y \mathrel{+}= Ax$

$$B_{SELL}(\alpha,\beta,N_{nzr}) = \left(\frac{1}{\beta}\left(\frac{8+4}{2}\right) + 8\alpha + \frac{16/N_{nzr}}{2}\right)\frac{\text{bytes}}{\text{flop}}$$

$$= \left(\frac{6}{\beta} + 4\alpha + \frac{8}{N_{nzr}}\right)\frac{\text{bytes}}{\text{flop}}$$

$$P(\alpha,\beta,N_{nzr},b) = \frac{b_S}{B_{SELL}(\alpha,\beta,N_{nzr})}$$

The α parameter

Corner case scenarios:

1. $\alpha = 0$ → RHS in cache
2. $\alpha = \frac{1}{N_{nzc}}$ → Load RHS vector exactly once

If $N_{nzc} \gg 1$, RHS traffic is insignificant: $\bar{P} = \frac{b\beta}{6\,\text{bytes/flop}}$

3. $\alpha \approx 1$ → Each RHS load goes to memory

4. $\alpha > 1$ → Houston, we've got a problem ☺

Determine α by measuring actual spMVM memory traffic (HPM)

SpMVM on Tsubasa ($b_S \approx 900\ GB/s$), nc++ 2.2.2

Matrix	N	N_{nzr}	B_c^{opt} [B/F]	opt. RL limit [GF/s]	SELL-256-sigma
DLR1	278502	142.72	6.10	148	**109.0**
scai1	3405035	7.06	7.98	113	**56.8**
pwtk	217918	53.93	6.26	144	**102.5**
ML_Geer	1504002	73.72	6.19	145	**93.6**
Hamrle3	1447360	3.81	9.67	93	**39.6**
rrze3_vv	6201600	14.92	6.94	130	**59.8**
kkt_power	2063494	7.08	7.98	113	**46.3**
Dense2	2000	2000	6	150	**79.6**
random2M20	2000000	20	6.70	134	**54.4**

→ Same effect as in stencil code? Still, dense2 too slow.

→ Register blocking techniques seem in order

An attempt at an explanation … again

- Assume memory BW $b_S = 900\,\frac{\mathrm{GB}}{\mathrm{s}}$
- Assume LLC BW $b_C = 3\,\frac{\mathrm{TB}}{\mathrm{s}}$
- Assume $\alpha = \alpha_{\min} = \frac{1}{N_{nzr}}$ and $N_{nzr} \gg 1$

- Assuming non-overlapping data transfers Mem ↔ LLC and Reg ↔ LLC, overall transfer time per MULT-ADD is

$$T = \frac{12\frac{\mathrm{B}}{\mathrm{FMA}}}{b_S} + \frac{20\frac{\mathrm{B}}{\mathrm{FMA}}}{b_C} \rightarrow P \approx 100\,\frac{\mathrm{Gflop}}{\mathrm{s}}$$

- → same conclusions as with stencil codes!

THANK YOU.