NHR FAU

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

SELL-C-σ

A sparse matrix storage format

Reminder: SpMV with CRS on A64FX

Assembly of the short inner loop

```
.L6:
        ld1sw     z0.d, p0/z, [x17, x20, lsl 2]
        ld1d      z2.d, p0/z, [x18, x20, lsl 3]
        ld1d      z3.d, p0/z, [x30, z0.d, lsl 3]
        add       x20, x20, 8
        fmla      z1.d, p0/m, z3.d, z2.d
        whilelo   p0.d, x20, x14
        b.any     .L6

        faddv     d4, p1, z1.d
```

FMA: Update **z1.d**

Latency: 9 cycles

Loop length : $N_{nzr}/8$

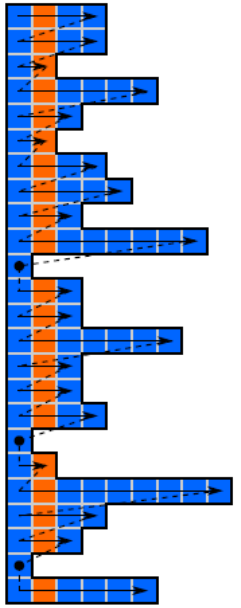
Horizontal add of 512-bit register

Throughput = 11.5

DOI:10.1109/PMBS51919.2020.00006

How can we solve this?

- Three problems
  1. Short inner loop (even shorter due to SIMD)
  2. Loop-carried dependency due to sum reduction
  3. Expensive horizontal add

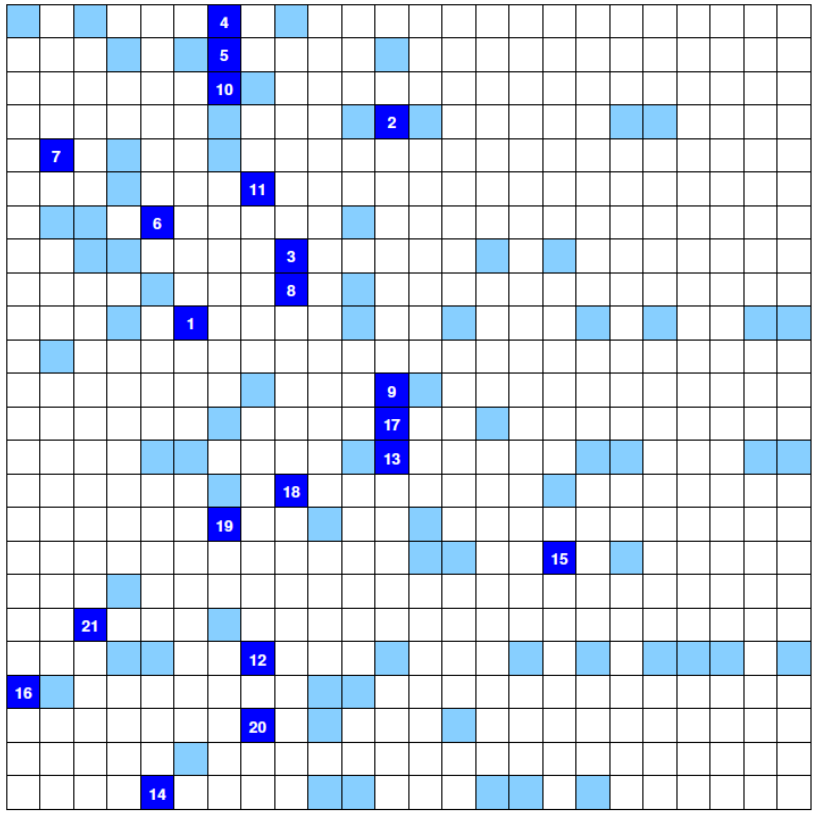


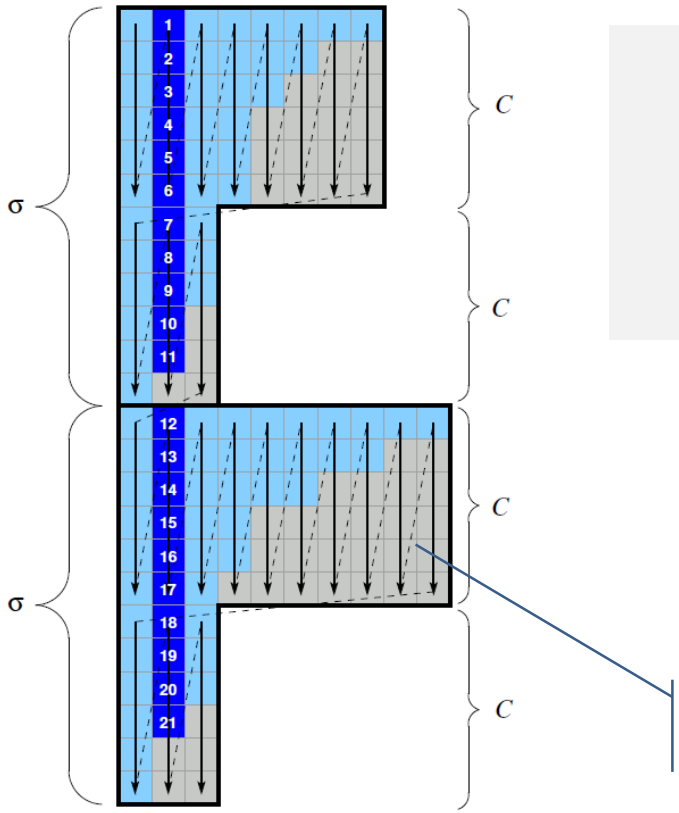
- Remedy: Vector-friendly matrix storage format
  - SELL-C-σ
  - M. Kreutzer et al.: *A Unified Sparse Matrix Data Format For Efficient General Sparse Matrix-vector Multiplication On Modern Processors With Wide SIMD Units*, SIAM SISC 2014, DOI: 10.1137/130930352

SELL-C-σ

Idea

- Sort rows according to length within sorting scope σ
- Store nonzeros column-major in zero-padded blocks of height C





"Chunk occupancy":

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

l_i: width of chunk i

SELL-C-σ kernel

Example $C = 4$ without further unrolling → longer inner loop, but still an LCD

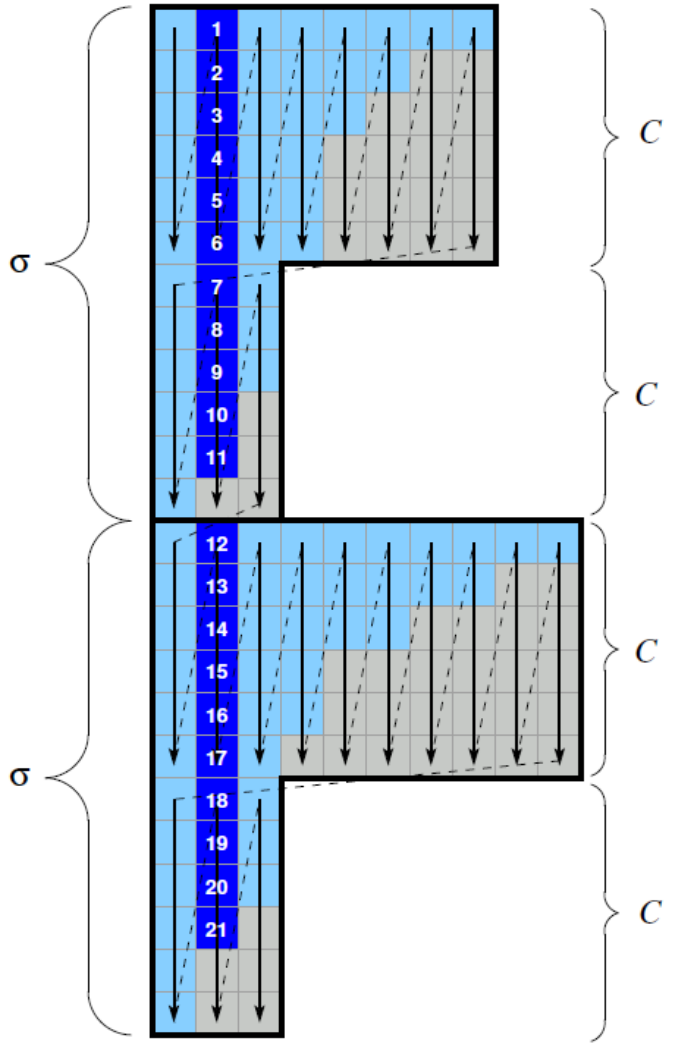
```
for(i = 0; i < N/4; ++i)
{

  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]];
  }
}
```

$C = 4$

How to choose the parameters?

- C
  - $n \times$ SIMD width to allow good utilization of SIMD units
  - $n > 1$ useful for hiding ADD pipeline latency
- σ
  - As small as possible, as large as necessary
  - Large σ reduces zero padding (brings β closer to 1)
  - Sorting alters RHS access pattern → α depends on σ



SELL-32-σ kernel OSACA analysis for A64FX

	0 - 0DV	1	2	3	4	5 - 5D	6 - 6D	7	CP	LCD	
92											.L4:
93						0.50 0.50	0.50 0.50				ld1sw z16.d, p0/z, [x11]
94						0.50 0.50	0.50 0.50				ld1sw z17.d, p0/z, [x11, #1, mul vl]
95						0.50 0.50	0.50 0.50				ld1sw z20.d, p0/z, [x11, #2, mul vl]
96						0.50 0.50	0.50 0.50		8.0		ld1sw z21.d, p0/z, [x11, #3, mul vl]
97	0.00		0.00	0.00	1.00						add x10, x10, 32
98	0.00		0.00	0.00	1.00						add x11, x11, 128
99	0.00		0.00	0.00	1.00						add x12, x12, 256
100						0.50 0.50	0.50 0.50				ld1d z19.d, p0/z, [x12, #-4, mul vl]
101						0.50 0.50	0.50 0.50				ld1d z18.d, p0/z, [x12, #-3, mul vl]
102						0.50 0.50	0.50 0.50				ld1d z25.d, p0/z, [x12, #-2, mul vl]
103						0.50 0.50	0.50 0.50				ld1d z27.d, p0/z, [x12, #-1, mul vl]
104	1.00			1.00		2.00 2.00	2.00 2.00				ld1d z22.d, p0/z, [x3, z16.d, lsl 3]
105	1.00			1.00		2.00 2.00	2.00 2.00				ld1d z23.d, p0/z, [x3, z17.d, lsl 3]
106	1.00			1.00		2.00 2.00	2.00 2.00				ld1d z24.d, p0/z, [x3, z20.d, lsl 3]
107	1.00			1.00		2.00 2.00	2.00 2.00		11.0		ld1d z26.d, p0/z, [x3, z21.d, lsl 3]
108		1.00		1.00							whilelo p1.d, x10, x9
109	0.00		1.00								fmla z4.d, p0/m, z19.d, z22.d
110	0.00		1.00								fmla z5.d, p0/m, z18.d, z23.d
111	0.00		1.00								fmla z6.d, p0/m, z25.d, z24.d
112	0.00		1.00						0.0	9.0	fmla z7.d, p0/m, z27.d, z26.d
113	0.00		0.00	0.00	1.00						mov p0.b, p1.b
114								1.00			b.any .L4
	4.00	1.00	4.00	5.00	4.00	12.0 12.0	12.0 12.0	1.00	19.0	9.0	

SELL-32-σ kernel OSACA analysis for A64FX

```
Loop-Carried Dependencies Analysis Report
-----------------------------------------
 112 |   9.0 | fmla   z7.d, p0/m, z27.d, z26.d       | [112]
 111 |   9.0 | fmla   z6.d, p0/m, z25.d, z24.d       | [111]
 110 |   9.0 | fmla   z5.d, p0/m, z18.d, z23.d       | [110]
 109 |   9.0 | fmla   z4.d, p0/m, z19.d, z22.d       | [109]
  99 |   1.0 | add    x12, x12, 256                   | [99]
  98 |   1.0 | add    x11, x11, 128                   | [98]
  97 |   1.0 | add    x10, x10, 32                    | [97]
```

Code balance of SELL-C-σ

Matrix data & column index

LHS update

chunk index

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

$$= \left(\frac{6}{\beta} + 4\alpha + \frac{\beta(8+2/C)}{N_{nzr}}\right)\frac{\text{bytes}}{\text{flop}}$$

Optimal $\alpha = \frac{\beta}{N_{nzr}}$

When measuring B_C^{meas}, take care to use the "useful" number of flops (excluding zero padding) for work

SpMVM on A64FX Conclusions

- CRS on A64FX has fundamental problems
  - Short reduction loop w/LCD, horizontal ADDs, long ADD latency
- This is why CRS cannot saturate the CMG bandwidth
- SELL-C-σ mitigates these problems by
  - providing a longer inner loop
  - allowing for ADD latency hiding
  - eliminating horizontal ADDs
  - eliminating remainder loops
- ... and it does not hurt on “standard” multicore CPUs and GPUs either
- Drawbacks
  - RHS access modified by sorting and column-major ordering
  - Zero padding introduces work overhead