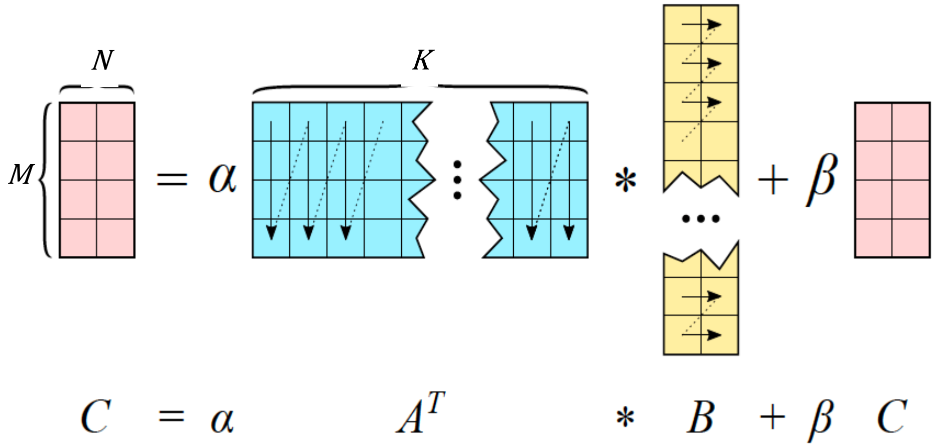
Case study:

Tall & Skinny Matrix-Transpose Times Tall & Skinny Matrix (TSMTTSM) Multiplication

TSMTTSM Multiplication

- Block of vectors → Tall & Skinny Matrix (e.g. 10^7 x 10^1 dense matrix)
- Row-major storage format (see SpMVM)
- Block vector subspace orthogonalization procedure requires, e.g., computation of scalar product between vectors of two blocks
- → TSMTTSM Mutliplication



Assume: $\alpha = 1;\ \beta = 0$

TSMTTSM Multiplication

General rule for dense matrix-matrix multiply: Use vendor-optimized GEMM, (e.g., Intel MKL[1]):

$$C_{mn} = \sum_{k=1}^{K} A_{mk} B_{kn} \ , \qquad m = 1..M, n = 1..N$$

double

System	P_{peak} [GF/s]	b_S [GB/s]	Size	Perf.	Efficiency
Intel Xeon E5 2660 v2 10c@2.2 GHz	176 GF/s	52 GB/s	SQ	160 GF/s	91%
			TS	16.6 GF/s	6%
Intel Xeon E5 2697 v3 14c@2.6GHz	582 GF/s	65 GB/s	SQ	550 GF/s	95%
			TS	22.8 GF/s	4%

complex double

TS@MKL:
Good or bad?

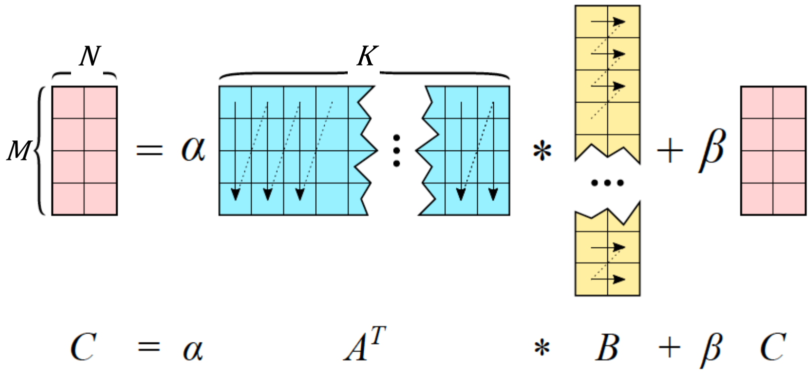
Matrix sizes:
Square (SQ): M=N=K=15,000
Tall&Skinny (TS): M=N=16 ; K=10,000,000

[1]Intel Math Kernel Library (MKL) 11.3

TSMTTSM Roofline model

Computational intensity

$$I = \frac{\#\text{flops}}{\#\text{bytes (slowest data path)}}$$



Optimistic model (minimum data transfer) assuming $M = N \ll K$ and double precision:

$$I_d \approx \frac{2KMN}{8(KM + KN)} \frac{\text{F}}{\text{B}} = \frac{M}{8} \frac{\text{F}}{\text{B}}$$

complex double:

$$I_z \approx \frac{8KMN}{16(KM + KN)} \frac{\text{F}}{\text{B}} = \frac{M}{4} \frac{\text{F}}{\text{B}}$$

TSMTTSM Roofline performance prediction

Now choose $M = N = 16 \rightarrow I_d \approx \frac{16}{8}\frac{\text{F}}{\text{B}}$ and $I_z \approx \frac{16}{4}\frac{\text{F}}{\text{B}}$, i.e. $B_d \approx 0.5\frac{\text{B}}{\text{F}}$, $B_z \approx 0.25\frac{\text{B}}{\text{F}}$

Intel Xeon E5 2660 v2 $(b_S = 52\frac{\text{GB}}{\text{s}}) \rightarrow P = 104\frac{\text{GF}}{\text{s}}$ (double)

Measured (MKL): $16.6\ \frac{\text{GF}}{\text{s}}$

Intel Xeon E5 2697 v3 $(b_S = 65\frac{\text{GB}}{\text{s}}) \rightarrow P = 240\frac{\text{GF}}{\text{s}}$ (double complex)

Measured (MKL): $22.8\ \frac{\text{GF}}{\text{s}}$

→ Potential speedup: 6–10x vs. MKL

Can we implement a better TSMTTSM kernel than Intel?

```
#pragma omp parallel
{
  double c_tmp[n*m] = {0.};

#pragma omp for
  for (int row = 0; row < k-1; row+=2) {
    for (int bcol = 0; bcol < n; bcol++) {
#pragma simd
      for (int acol = 0; acol < m; acol++) {
        c_tmp[bcol*m+acol] +=
          a[(row+0)*m + acol] * b[(row+0)*n + bcol] +
          a[(row+1)*m + acol] * b[(row+1)*n + bcol];
      }
    }
  }

#pragma omp critical
  for (int bcol = 0; bcol < n; bcol++) {
#pragma simd
    for (int acol = 0; acol < m; acol++) {
      c[bcol*m+acol] += c_tmp[bcol*m+acol];
    }
  }
}
```

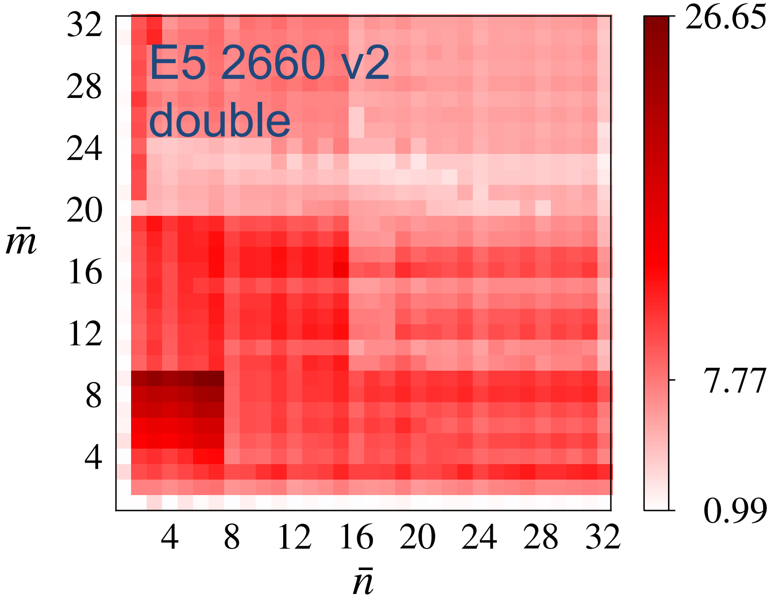
- Thread-local copy of small (results) matrix
- Long Loop (k): Parallel
- Outer Loop Unrolling
- Compiler directives
- Most operations in cache
- Reduction on small result matrix

Not shown: Inner Loop boundaries (n,m) known at compile time (kernel generation), k assumed to be even

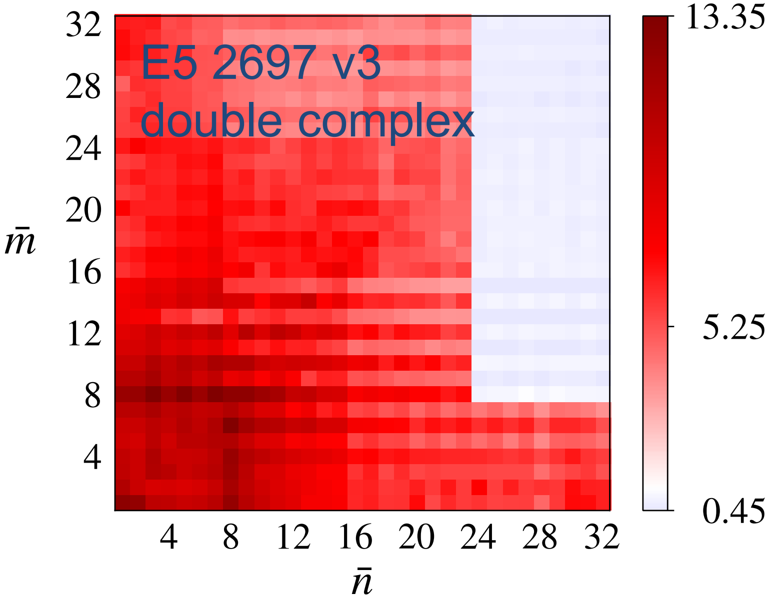
TSMTTSM MKL vs. “hand crafted” (OPT)

TS: M=N=16 ; K=10,000,000

System	P_{peak} / b_S	Version	Performance	RLM Efficiency
Intel Xeon E5 2660 v2 10c@2.2 GHz	176 GF/s 52 GB/s	**TS OPT**	**98 GF/s**	**94 %**
		TS MKL	16.6 GF/s	16 %
Intel Xeon E5 2697 v3 14c@2.6GHz	582 GF/s 65 GB/s	**TS OPT**	**159 GF/s**	**66 %**
		TS MKL	22.8 GF/s	9.5 %



Speedup vs. MKL: 5x – 25x



TSMTTSM conclusion

- Typical example of model-guided optimization
- “Invisible” P_{max} ceiling with Intel MKL (probably wrong loop parallelized)
- Hand-coded implementation ran much closer to limit

- Caveat:
  This is to exemplify the method; current MKL versions might have improved!