EasyBuild's Linear Algebra

Bart Oldeman
McGill University, Calcul Québec, Digital Research Alliance of Canada
Research Support National Team Software Installation Coordinator

Partenaire régional de l'
Alliance de recherche numérique du Canada

A regional partner of the
Digital Research Alliance of Canada

Background

Linear algebra is one of the 4 components of an EasyBuild toolchain:

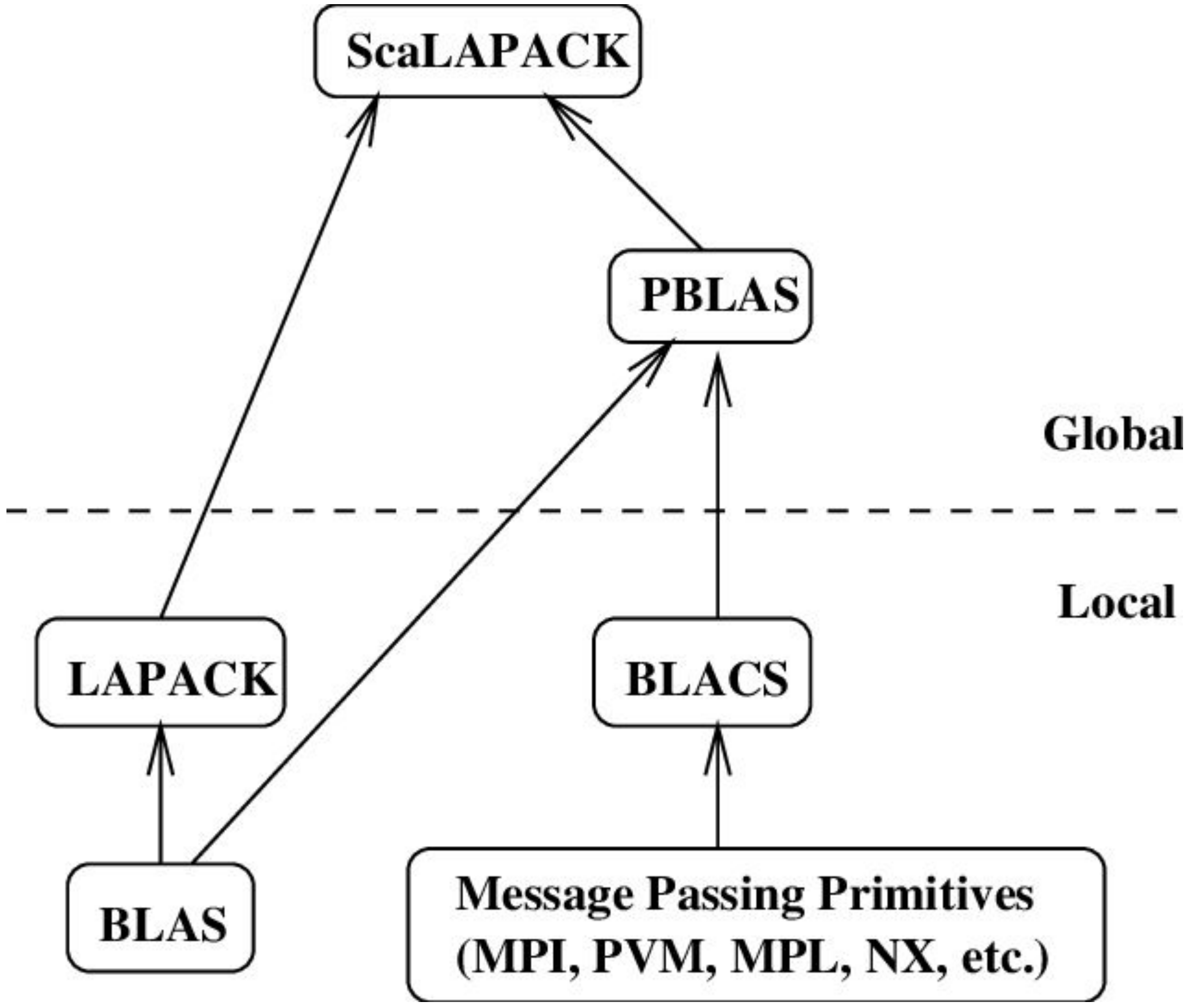
1. Compiler: **G**CC, **I**ntel, **N**VHPC, **C**UDA(*), …
2. MPI: **O**penMPI, **I**ntelMPI, **Q**LogicMPI(**), …
3. Linear algebra: **O**penBLAS, **B**LIS, **F**lexi**B**las, i**M**KL, BLACS, (Sca)LAPACK, …
4. FFT: **F**FTW, **I**ntelFFTW **F**ujitsuFFTW

- A full toolchain combines all 4, e.g. foss = gofbf, intel.
- A partial toolchain always has a compiler but not all of 2, 3, and 4, e.g. gfbf = GCC + FlexiBLAS + FFTW, but no MPI

(*) unused as toolchain component since 2021a

(**) there are many such obsolete implementations still lingering in the framework

Linear Algebra components in EB: BLAS, LAPACK, BLACS, ScaLAPACK



Source: Dongarra & Blackford, 1997

Linear Algebra components in EB

1. BLAS = Basic Linear Algebra Subprograms, 3 levels
    a. BLAS-1: vector-vector, 1979, O(n) e.g. daxpy: y += alpha*x
    b. BLAS-2: matrix-vector, 1988, O(n^2) e.g. dgemv: y = alpha*A*x + beta*y
    c. BLAS-3: matrix-matrix, 1990, O(n^3) e.g. dgemm: C = alpha*A*B + beta*C

    (CBLAS = C interface to BLAS)
2. LAPACK = Linear Algebra Package (1992)
    a. for solving systems of linear equations, eigenvalues/eigenvectors, etc.
    b. uses BLAS-3 where possible for efficiency unlike predecessors (LINPACK/EISPACK); LAPACKE = C interface to LAPACK.
3. BLACS = Basic Linear Algebra Communication Subprograms (1996)
    a. Abstraction layer around PVM (RIP) and MPI for linear algebra
4. ScaLAPACK = Scalable LAPACK (1996)
    a. LAPACK routines on distributed memory
    b. Includes BLACS since version 2.0 (2011)

BLAS implementations in EB

bold-faced: open source, others are closed source vendor implementations

1. ACML: AMD Math Core Library, EOL since ~2014
2. **ATLAS** (1997): Automatically Tuned Linear Algebra Software, last updated 2016
3. **BLIS** (2013): BLAS-Like Library Instantiation Software
4. **FlexiBLAS** (2013): Wrapper for different BLAS and LAPACK implementations
5. FujitsuSSL (2020)
6. **GotoBLAS** (2002): Superseded by OpenBLAS fork in 2011
7. Intel MKL (1994): Math Kernel Library, now called oneMKL
8. Cray LibSci
9. **OpenBLAS** (2011): fork of GotoBLAS

Other implementations, e.g.:

- **Reference BLAS**: Fortran routines on netlib, low performance
- cuBLAS: BLAS for NVIDIA GPUs.
- **rocBLAS**: for AMD GPUs

Example: Netlib DGEMM

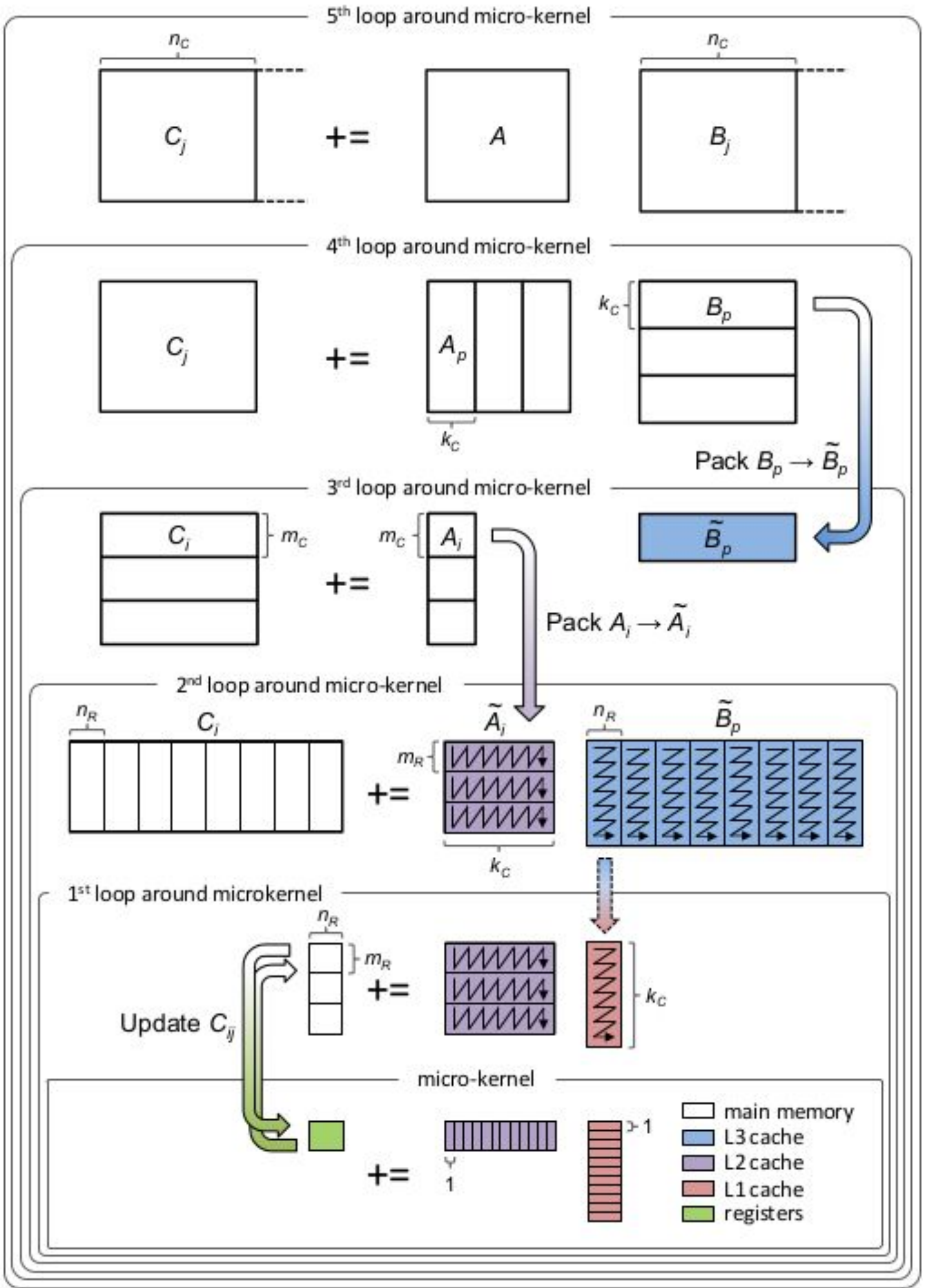
```
*https://www.netlib.org/lapack/explore-3.1.1-html/dgemm.f.html
* first hit if you google dgemm.f
*          Form  C := alpha*A*B + beta*C.
*
           DO 90 J = 1,N
               IF (BETA.EQ.ZERO) THEN
                   DO 50 I = 1,M
                       C(I,J) = ZERO
  50               CONTINUE
               ELSE IF (BETA.NE.ONE) THEN
                   DO 60 I = 1,M
                       C(I,J) = BETA*C(I,J)
  60               CONTINUE
               END IF
               DO 80 L = 1,K
                   IF (B(L,J).NE.ZERO) THEN
                       TEMP = ALPHA*B(L,J)
                       DO 70 I = 1,M
                           C(I,J) = C(I,J) + TEMP*A(I,L)
  70                   CONTINUE
                   END IF
  80           CONTINUE
  90       CONTINUE
```

Ifort optimized this with fast custom code

```
*https://github.com/Reference-LAPACK/lapack/blob/master/BL
AS/SRC/dgemm.f, avoids issues with NaNs
*          Form  C := alpha*A*B + beta*C.
*
           DO 90 J = 1,N
               IF (BETA.EQ.ZERO) THEN
                   DO 50 I = 1,M
                       C(I,J) = ZERO
  50               CONTINUE
               ELSE IF (BETA.NE.ONE) THEN
                   DO 60 I = 1,M
                       C(I,J) = BETA*C(I,J)
  60               CONTINUE
               END IF
               DO 80 L = 1,K

                   TEMP = ALPHA*B(L,J)
                   DO 70 I = 1,M
                       C(I,J) = C(I,J) + TEMP*A(I,L)
  70               CONTINUE

  80           CONTINUE
  90       CONTINUE
```

But not this!

Why DGEMM? Bo Kågström, Per Ling, Charles van Loan (1998) proved everything else can be in terms of GEMM



5th loop around micro-kernel
n_C
C_j
+=
A
B_j
4th loop around micro-kernel
C_j
A_p
k_C
B_p
Pack $B_p \rightarrow \tilde{B}_p$
3rd loop around micro-kernel
C_i
m_C
A_i
$\tilde{B}_p$
Pack $A_i \rightarrow \tilde{A}_i$
2nd loop around micro-kernel
n_R
C_i
$\tilde{A}_i$
m_R
$\tilde{B}_p$
1st loop around microkernel
Update C_{ij}
micro-kernel
1
main memory
L3 cache
L2 cache
L1 cache
registers

GotoBLAS/OpenBLAS/BLIS DGEMM

```
do jc = 1, n, nc ! 5th loop, nc~4096
   do pc = 1, k, kc ! 4th loop, kc~512
       Bp = B(pc:pc+kc-1, jc:jc+nc-1)
       do ic = 1, m, mc ! 3th loop, mc~120
          Ap = A(ic:ic+mc-1, pc:pc+kc-1)
          do jr = 1, nc, nr ! 2nd loop, nr~32 -> macro-kernel
             do ir = 1, mc, mr ! 1st loop, mr~6
                Cp(1:mr, 1:nr) = 0 -> eg. 24 512bit ZMM vectors
                do kr = 1, kc ! 0th loop -> micro-kernel
                   Cp(1:mr, 1:nr) +=
                      Ap(ir:ir+mr-1, kr) * Bp(kr, jr:jr+nr-1)
                enddo
                C(ic+ir-1:ic+ir+mr-2, jc+jr-1,jc+jr+nr-2) +=
                   Cp(1:mr, 1:nr)
             enddo
          enddo
       enddo
   enddo
enddo
```

Algorithm:Goto&Van de Geijn, Anatomy of HP matrix multiplication 2008
Goto/OpenBLAS: optimized macro-kernel, BLIS: optimized micro-kernel
Note: can be easily parallellized in multiple places (jc, ic, jr)!
Figure credit: Van Zee & Van de Geijn, UTexas, Austin

AVX512 microkernel in C and assembly language

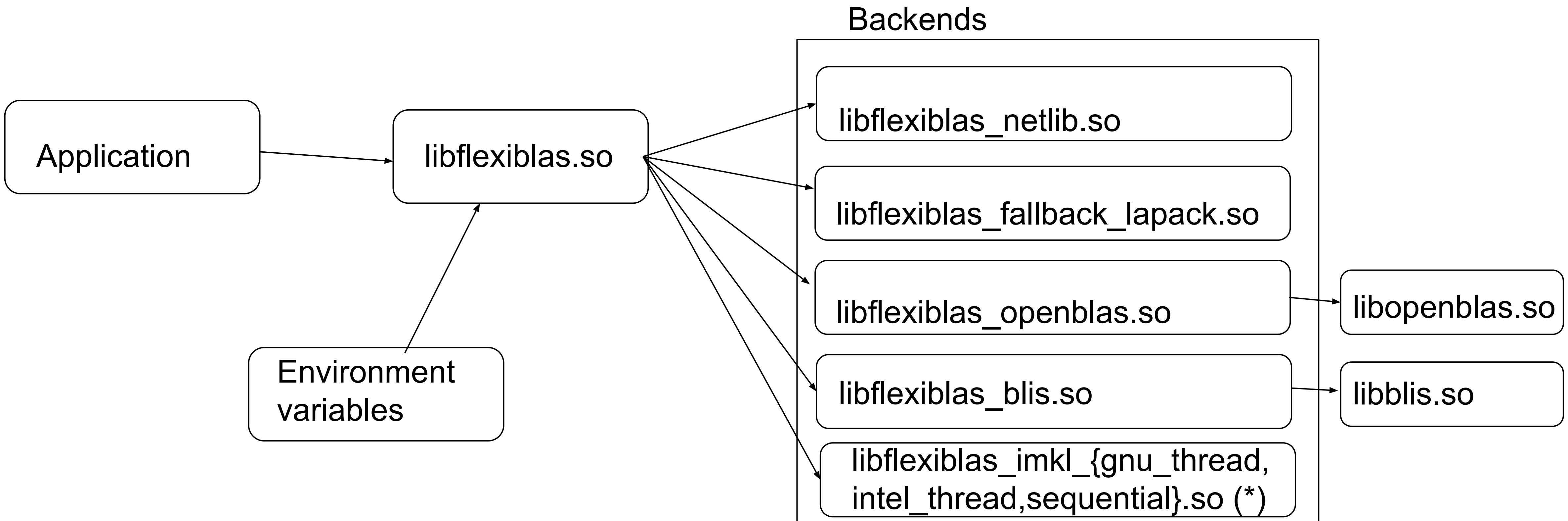
```
#define CACHELINE_SIZE 64
#define TAIL_NITER 5
size_t l = 0; do
{
    size_t i = l + TAIL_NITER*4 + mr - k;
    if ( i  >= 0 && i < mr ) // prefetch parts of c strategically
        for ( dim_t j = 0; j < nr; j+=CACHELINE_SIZE/sizeof(double))
            prefetch( &c[ i*s_c + j ] );
    for ( size_t i = 0; i < mr; ++i ) { //mr=6
        #pragma omp for simd
        for ( size_t j = 0; j < nr; ++j ) { //nr=32
            cr[ i*s_cr + j ] += a[i] * b [j];
        a += cs_a;
        b += rs_b;
    }
} while ( ++l < k );
// update c with cr
for ( size_t i = 0; i < mr; ++i )
    #pragma omp for simd
    for ( dim_t j = 0; j < nr; ++j )
        c [ i*s_c + j ] = alpha * c[ i*s_c + j ] +
            beta * cr [ i * s_cr + j ];
```

```
# zmm0-zmm3 = b[0..31]
# %rax -> a

vbroadcastsd -0x28(%rax),%zmm4
vfmadd231pd %zmm4,%zmm3,%zmm24
vfmadd231pd %zmm4,%zmm2,%zmm23
vfmadd231pd %zmm4,%zmm1,%zmm22
vfmadd231pd %zmm4,%zmm0,%zmm21

vbroadcastsd -0x20(%rax),%zmm4
vfmadd231pd %zmm4,%zmm3,%zmm20
vfmadd231pd %zmm4,%zmm2,%zmm19
vfmadd231pd %zmm4,%zmm1,%zmm18
vfmadd231pd %zmm4,%zmm0,%zmm17

# … (6 of such blocks)
```

A C kernel can be as fast as asm here..
https://github.com/flame/blis/pull/863

FlexiBLAS

FlexiBLAS (Köhler&Saak 2014) allows to swap BLAS implementations at runtime using the $FLEXIBLAS environment variable. For foss toolchains OpenBLAS, BLIS, and Netlib backends are available; MKL as optional add-on.



(*) created from $MKLROOT/tools/builder directory in imkl easyblock, found via $FLEXIBLAS_LIBRARY_PATH at runtime, so no compile-time dep!

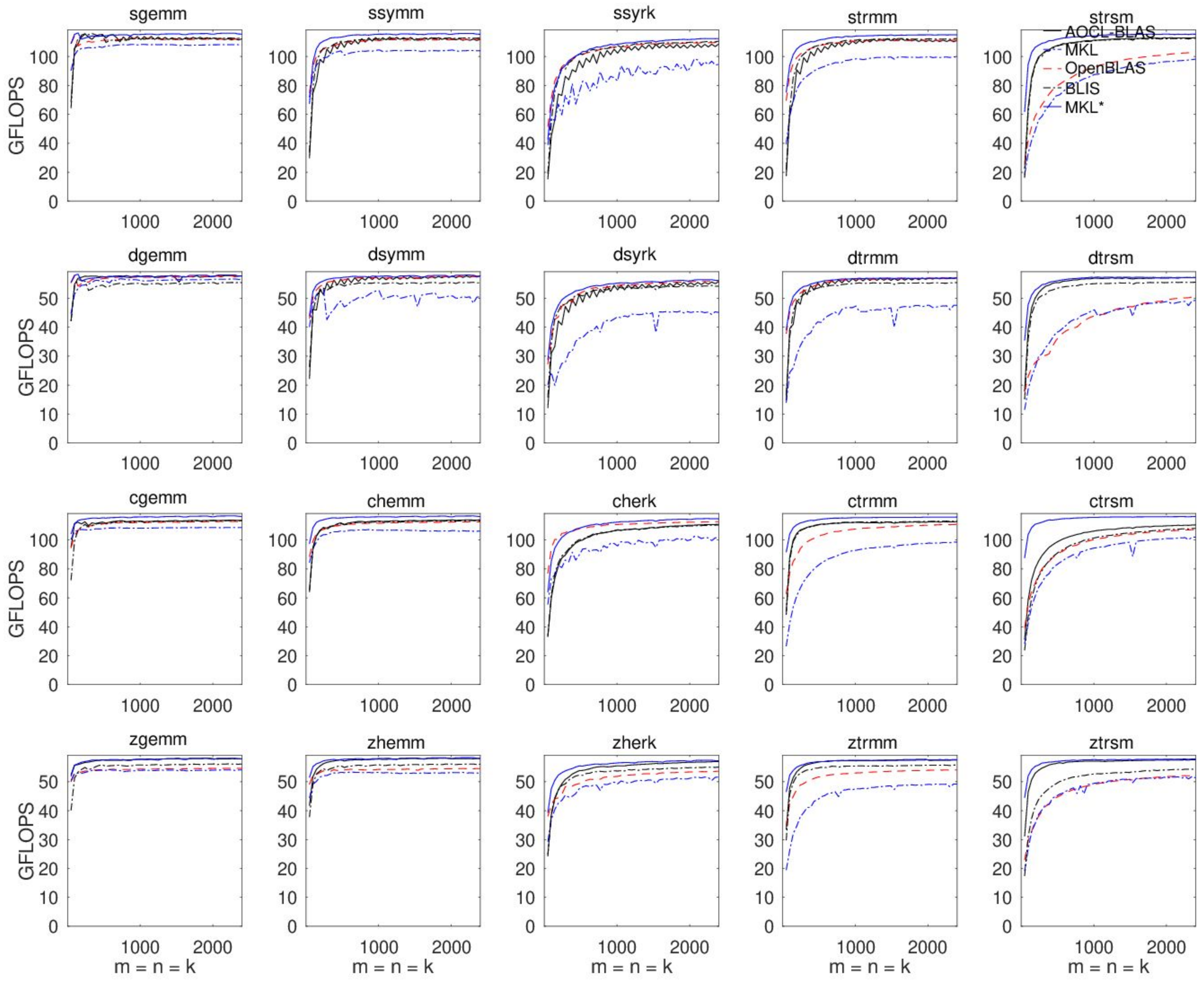
FlexiBLAS

- BLIS, OpenBLAS, and MKL can all do run-time dispatch for microarchitectures
- FlexiBLAS can connect to different backends for different OpenMP implementations (GNU `libgomp`, Intel `libiomp5`, etc)
- Can we then have e.g. `FlexiBLAS-3.4.4-GCCcore-13.3.0.eb` instead of `FlexiBLAS-3.4.4-GCC-13.3.0.eb` (i.e. with one common ABI)?
- Unfortunately on x86 two different calling conventions are used for complex return values (for BLAS-1 functions `cdotc`, `zdotc`, `cdotu`, `zdotu`)
  - the older “g77” stack convention still used by Intel and NVHPC compilers
  - the newer “gfortran” register convention used by `gfortran` and `flang(-new)`.
- Intel/NVHPC have to either use a FlexiBLAS module at the compiler (not GCCcore) level or a separate `libflexiblas_intel.so`, or else a wrong combination will crash.
- Fortunately numpy is calling-convention agnostic, using CBLAS for those 4 BLAS-1 functions, so Python packages are ok even with Intel-compiled code mixed in.

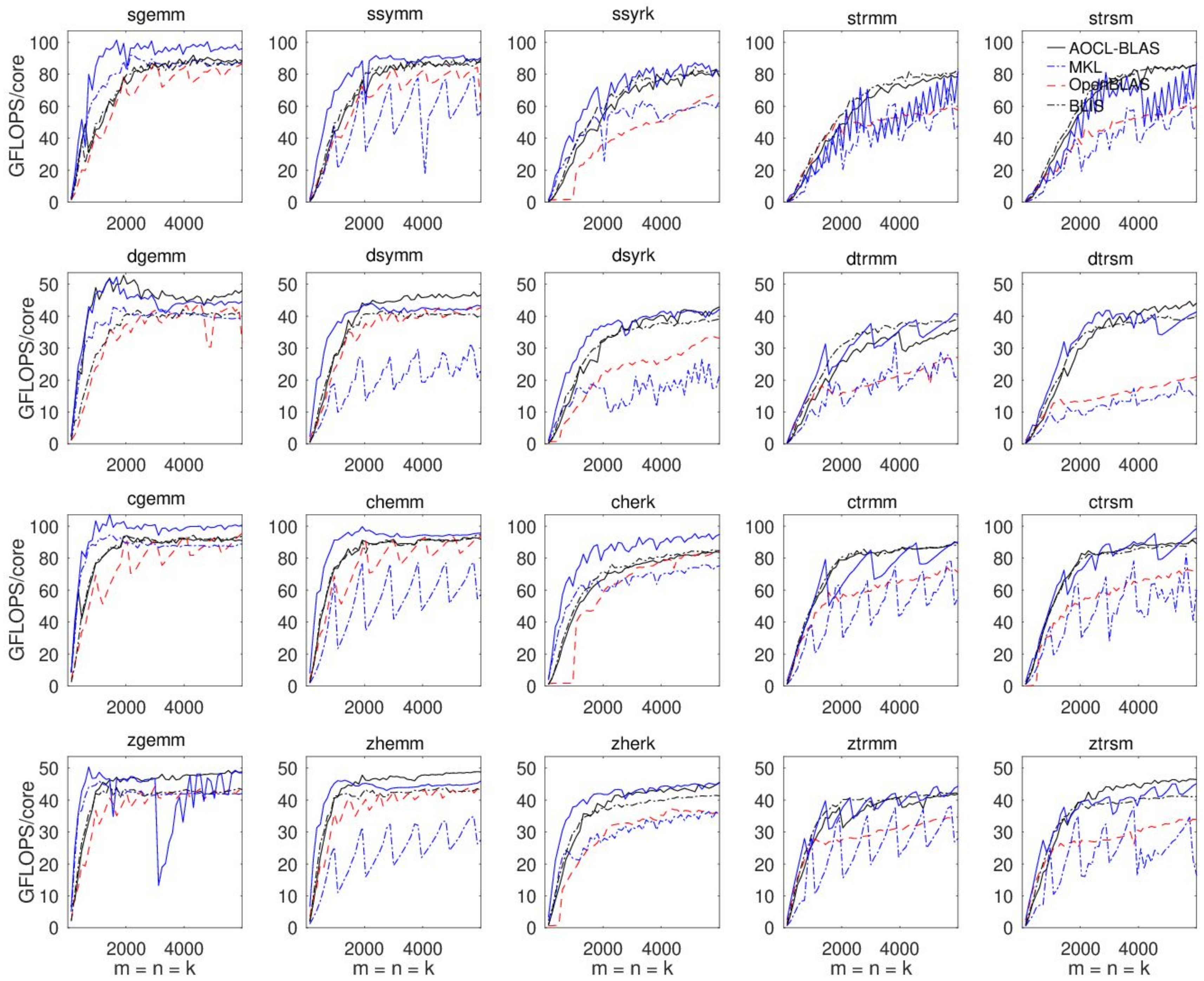
Benchmarks

- Based on https://github.com/flame/blis/blob/master/docs/Performance.md
- Hardware: AMD EPYC 9534 64-Cores, Zen4 (Genoa), single-socket, AVX512
  - Turbo frequency for single core 3.7 GHz, DP peak 3.7*16=59.2 Gflops, SP peak 118.4 Gflops (16=2FMA * 8 DP/ZMM “double-pumped”)
  - Turbo frequency for 64 cores ~ 3.35 GHz
  - 32k L1i + 32k L1d + 1M L2 cache/core + 32M L3 cache/8core-CCD(“chiplet”)
- Compiler: GCC 13.3
- Contenders:
  - BLIS 1.1 (“zen3” as zen4 is not supported yet; “skx” wasn’t faster)
  - AOCL-BLAS 5.0 (AMD’s fork of BLIS; “zen4”)
  - OpenBLAS 0.3.29 (using “COOPERLAKE”)
  - MKL 2025.0
  - MKL 2024.2 with “secret sauce” to make it think it ran on an Intel CPU (didn’t work with MKL 2025.0)

Benchmarks (single core) (*=secret)



Benchmarks (64 cores - 4x4x4 for BLIS jc,ic,jr) (solid blue= “secret”)



Conclusions, references, comments

- AOCL-BLAS overall winner on this processor
  (Intel CPU, not covered in graphs, tends to favour MKL)
- But most implementations do a good job in general
  - upstream BLIS lags a bit in supporting latest CPUs (also vs. OpenBLAS)
  - variation on the small end, but consider libxsmm also
- SHPC at The University of Texas at Austin has a lot of material
  - The Science of High-Performance Computing Group
  - PfHP Welcome to LAFF-On Programming for High Performance
  - https://github.com/flame
  - 6th EasyBuild User Meeting (Jan 25-29 2021) - BLIS talk by Field Van Zee (UTexas)
  - Field Van Zee and Devin Matthews winners of J. H. Wilkinson Prize for Numerical Software for BLIS
- FlexiBLAS: EasyBuild tech talks III: FlexiBLAS
- The field is diversifying beyond level-3 BLAS: mixed-precision, mixed-domain (complex vs real), bandwidth restrictions, Strassen’s algorithm revisited, etc.