BLAS and LAPACK

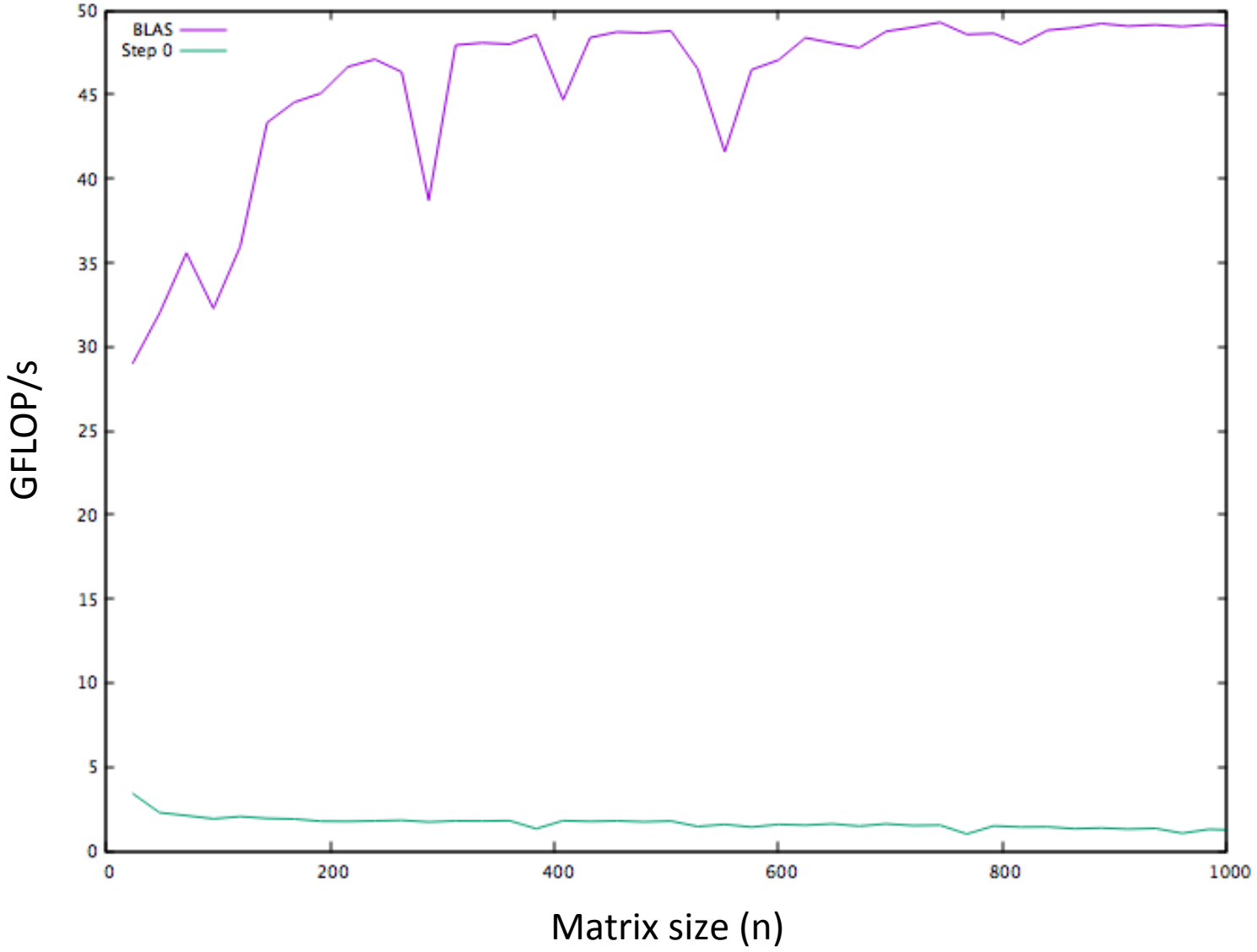
Devin A. Matthews
UT Austin

MolSSI Software Summer School 2017

BLAS and LAPACK

- Dense Linear Algebra (DLA) is the field of operations on **vectors** and **matrices**.
- The “dense” part refers to the fact that all of the vector and matrix elements may be non-zero. There are also extensions of BLAS and LAPACK for sparse matrices.
- The BLAS includes basic operations like matrix multiplication, while LAPACK builds on the BLAS to calculate eigenvalues and –vectors, solutions of linear systems, matrix factorizations, etc.

Why do I need a library?



BLAS and LAPACK

- The traditional BLAS and LAPACK implementations are in Fortran.
- Modern implementations are in C or C++, but still export a Fortran interface.
- This means that:
  - All parameters are pointers.
  - The name is **mangled** (usually a _ gets added).
  - Fortran INTEGER != C/C++ int.

BLAS and LAPACK

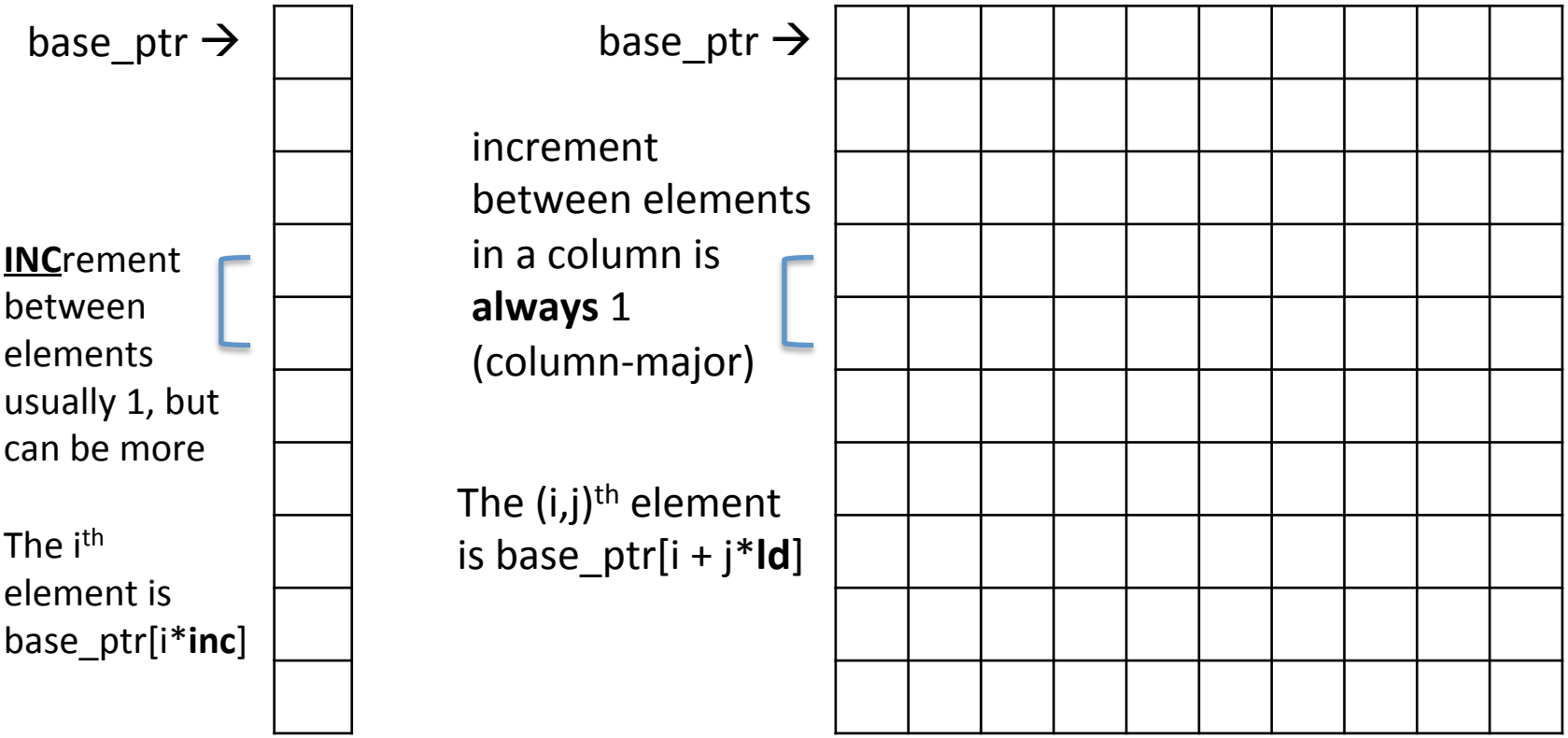
- (Almost) all BLAS and LAPACK routines support four data types:
  - Single precision (float, real*4)
  - Double precision (double, real*8)
  - Single complex (std::complex<float>, complex*8)
  - Double complex (std::complex<double>, complex*16)
- The data type for an operation is denoted by a prefix: s (single), d (double), c (single complex), or z (double complex).
  - Ex: zdotc = double complex dot product with conjugation

The anatomy of vectors and matrices

Vector
length = n

Matrix
rows = m
columns = n

Increment between elements in a row (the **L**eading **D**imension) is usually m, but can be more



BLAS

- BLAS includes three levels:
  - **Level 1**: vector-vector operations such as:
    - Dot product
    - Vector scaling
    - Scale and accumulate (AXPY)
  - **Level 2**: matrix-vector operations such as:
    - Matrix-vector product
    - Outer product (rank-1 update)
  - **Level 3**: matrix-matrix operations such as:
    - Matrix-matrix product
    - Triangular solve
    - Symmetric rank-k update

BLAS level 1 examples

$$y_i = y_i + \alpha * x_i$$

```
daxpy(integer n, real*8 alpha, real*8 x, integer incx,
                               real*8 y, integer incy)
```

$$\text{result} = \Sigma_i \, x_i * \overline{y}_i$$

```
complex*16 zdotc(integer n, complex*16 x, integer incx,
                            complex*16 y, integer incy)

zdotc(integer n, complex*16 x, integer incx,
                 complex*16 y, integer incy,
      complex*16 result)
```

BLAS level 2 examples

$$A_{ij} = \alpha * x_i * y_j + A_{ij}$$

```
dger(integer m, integer n, real*8 alpha,
     real*8 x, integer incx,
     real*8 y, integer incy, real*8 A, integer lda)
```

$$y_i = \Sigma_j \, \alpha * op(A_{ij}) * x_j + \beta * y_i$$

```
  sgemv(character*1 trans, integer m, integer n,
        real*4 alpha, real*4 A, integer lda,
                      real*4 x, integer incx,
        real*4  beta, real*4 y, integer incy)
```

BLAS level 3 example

$$C_{ij} = \Sigma_p \alpha * op_A(A_{ip}) * op_B(B_{pj}) + \beta * C_{ij}$$

```
dgemm(character*1 transa, character*1 transb,
      integer m, integer n, integer k,
      real*8 alpha, real*8 A, integer lda,
                    real*8 B, integer ldb,
      real*8  beta, real*8 C, integer ldc)
```

LAWrap

- This is a header library I wrote to make using BLAS/LAPACK in C and C++ a little easier.
  - Give arguments by values instead of by reference (pointer).
  - Const-correct interface.
  - Function overloading (e.g. “gemm”).
  - Automatically allocate workspace in LAPACK.
- CBLAS and LAPACKe are also good low-level interfaces.

LAPACK

Symmetric (Hermitian) Eigenvalue Problems

$$\langle \phi_i | \hat{H} | \phi_j \rangle \langle \phi_j | \Psi \rangle = E \langle \phi_i | \Psi \rangle$$

Generalized Eigenvalue Problems

$$\tilde{\mathbf{F}}\tilde{\mathbf{C}} = \tilde{\mathbf{C}}\mathbf{E}$$

$$\mathbf{FC} = \mathbf{SCE}$$

Non-symmetric Eigenvalue Problems

$$\left(\hat{H} e^{\hat{T}} \right)_c \hat{R}_i |0\rangle = E_i R_i |0\rangle$$

Matrix Factorizations and Linear Equations

$$\langle K | \tilde{B} | \mu\nu \rangle = \langle K | V | L \rangle^{-\frac{1}{2}} \langle L | B | \mu\nu \rangle$$

(This is Cholesky + triangular solve, not inverse square root!)

LAPACK example

(**FOR EXAMPLE ONLY**, also may have bugs/missed steps)

```
int norb = 1000;
int nocc = 100;
std::vector<double> C(norb*norb), S(norb*norb), F(norb*norb), D(norb*norb);
std::vector<double> e(norb);

C = F;
int info = LAWrap::hegv(LAWrap::AXBX, 'V', 'U',
                        norb, C.data(), norb, S.data(), norb, e.data());

std::cout << "Current SCF Eigenvalues:\n";
std::cout << std::setprecision(12) << std::fixed;
for (int i = 0;i < nocc;i++)
    std::cout << i << " " << e[i] << '\n';
std::cout << "*********************************************************\n";
for (int i = nocc;i < norb;i++)
    std::cout << i << " " << e[i] << '\n';

LAWrap::gemm('N', 'T', norb, norb, nocc,
             1.0, C.data(), norb, C.data(), norb, 0.0, D.data(), norb);
```

BLAS and LAPACK libraries

- Netlib (http://www.netlib.org/blas) (OSS)
  - Reference implementation; *really slow*.
- Intel Math Kernel Library (MKL) (free)
  - Optimized BLAS, LAPACK, FFT, etc.
- AMD Compute Library (ACL) (OSS)
  - AMD vendor BLAS/LAPACK
- IBM Engineering and Scientific Subroutine Library (ESSL)
  - IBM vendor BLAS/LAPACK (e.g. BlueGene/Q, POWER)
- OpenBLAS (OSS)
  - Optimized BLAS and Netlib LAPACK
- BLIS (OSS)
  - Optimized BLAS
- libflame (OSS)
  - Some LAPACK operations optimized
- Apple Accelerate
  - Default BLAS and LAPACK on OSX/macOS

Wrapper libraries

- Wrapper libraries offer convenient interfaces in higher-level languages. Here is a non-exhaustive list for C++:
  - Eigen
  - Boost::uBLAS
  - Armadillo
  - CPPLapack
    - The “CPPBlas” project is unrelated.
- Distributed-memory BLAS and LAPACK:
  - Elemental
  - ELPA
  - PBLAS, ScaLAPACK

Questions