BLAS for Tensors: What, Why, and How

Devin Matthews
Outline

I. The role of interfaces in domain applications

II. BLAS as an example interface

III. BLIS as a better BLAS

IV. Tensors

V. Tensor interfaces now

VI. A “Tensor BLAS”
Math to Code: Fundamental Considerations

What domain scientists want (math):

\[
\langle 0 | \delta \Lambda^{(1)}[[\bar{H}, \delta T^{(1)}], \delta T^{(2)}]|0 \rangle \leftarrow \frac{1}{4} \lambda_{abc}^{ijk(1)} \bar{H}_{tie}^{ma(3)} t_{mjk}^{ebc(1)}
\]

\[
\left( -\frac{i \hbar}{c} \gamma^0 \frac{\partial}{\partial t} - i \hbar \gamma^j \partial_j + mc \right) \psi(x) = (-\gamma^0 p^0 + \gamma^j p^j + mc)\psi(x)
\]

\[
P(Z_{(m,n)} = k | Z_{-(m,n)}, W; \alpha, \beta) \\
\propto P(Z_{(m,n)} = k, Z_{-(m,n)}, W; \alpha, \beta)
\]

\[
= \left( \frac{\Gamma \left( \sum_{i=1}^{K} \alpha_i \right)}{\prod_{i=1}^{K} \Gamma(\alpha_i)} \right)^M \prod_{j \neq m}^{K} \frac{\Gamma(n_{j,(-)} + \alpha_i)}{\Gamma(\sum_{i=1}^{K} n_{j,(-)} + \alpha_i)}
\]
Math to Code: Fundamental Considerations

What domain scientists want (math):

\[
\langle 0 | \delta \Lambda^{(1)} [[\hat{H}, \delta T^{(1)}], \delta T^{(2)}] | 0 \rangle \leftarrow \frac{1}{4} \epsilon^{ijk(1)} \hat{H}_{ie}^{ma(3)} t^{ebc(1)}_{mjk} \\
(\frac{-i\hbar}{c} \gamma^0 \frac{\partial}{\partial t} - i\hbar \gamma^i \partial_j + mc) \psi(x) = (-\gamma^0 p^0 + \gamma^i p^i + mc) \psi(x)
\]

What computer (computational) science provides (code):

- DGEMM (BLAS)
- SCSRMM (SparseBLAS)
- MPI_Reduce_scatter_block (MPI)
- "\" (Matlab)
- ZGGEV (LAPACK)
- contract(...)/*" (i.e. libtensor et al.)
Math to Code: Fundamental Considerations

What domain scientists want (math):

\[ \langle 0 | \delta \Lambda^{(1)} [[H, \delta T^{(1)}], \delta T^{(2)}] | 0 \rangle \leftarrow \frac{1}{4} \lambda_{abc}^{ijk(1)} H_{iae}^{ma(3)} t_{mjk}^{ebe(1)} \]

\[ \left( -\frac{i\hbar}{c} \gamma^0 \frac{\partial}{\partial t} - i\hbar \gamma^j \partial_j + mc \right) \psi(x) = (-\gamma^0 p^0 + \gamma^j p^j + mc) \psi(x) \]

\[ P(Z_{(m,n)} = k \mid Z_{- (m,n)}, W; \alpha, \beta) \times P(Z_{(m,n)} = k, Z_{- (m,n)}, W; \alpha, \beta) \]

\[ = \left( \frac{\prod_{i=1}^{K} \Gamma(\gamma_{i})}{\prod_{i=m}^{K} \Gamma(\gamma_{i})} \right)^{M} \prod_{i=j}^{K} \Gamma(n_{i,j,\alpha} + \alpha) \]

What computer (computational) science provides (code):

DGEMM (BLAS)  SCSRMM (SparseBLAS)  MPI_Reduce_scatter_block (MPI)

“\"” (Matlab)  ZGGEV (LAPACK)  contract(...)/*” (i.e. libtensor et al.)

What no one likes to have to deal with (but is important):

Data layout  Performance  Alignment/caching

Numerical representation  Architectural optimizations
Math to Code: Fundamental Considerations

What domain scientists want (math):

$$\langle 0 | \delta \Lambda^{(1)}[[\vec{H}, \delta T^{(1)}], \delta T^{(2)}]|0 \rangle \leftarrow \frac{1}{4} \lambda^{ijk(1)}_{abc} \vec{H}^{ma(3)}_{ie} t^{e bc(1)}_{m j k}$$

$$\left( -i\hbar \gamma^0 \frac{\partial}{\partial t} - i\hbar \gamma^j \partial_j + mc \right) \psi(x) = \left( -\gamma^0 p^0 + \gamma^j p^j + mc \right) \psi(x)$$

What computer (computational) science provides (code):

- **DGEMM (BLAS)**
- **SCSRMM (SparseBLAS)**
- **MPI_Reduce_scatter_block (MPI)**
- **ZGGEV (LAPACK)**
- **contract(...)/*"** (i.e. libtensor et al.)

What no one likes to have to deal with (but is important):

- Data layout
- Performance
- Alignment/caching
- Numerical representation
- Architectural optimizations
Features of BLAS

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)

User/external code owns the data.

Data and specification are separate. Allows easy hacking (e.g. submatrices, resizing, etc.), but can be error-prone.

All information about the operation is explicit. Can be unwieldy for users but good for flexibility.
BLAS: the Good and the Bad
(a.k.a. How to Start a Flame War)

Good:

• Easier than writing three loops.
• Flexible enough for end users and library writers.
• High-performance implementations exist.
• Bindings in many languages.
• Can be wrapped in a more user-friendly interface.

Bad:

• FORTRAN dependency.
• Requires unit stride.
• No conjugation without transpose.
• Some obvious missing features.
• User must allocate data and deal with alignment etc.
• Opaque.
• Poor threading control.
• No mixed-precision support.
How People Really (Ab)use BLAS

dcopy(n, &constant, 0, array, 1)  (i.e. fill array)

daxpy(n, alpha1, x_1, 1, y, 1)
daxpy(n, alpha1, x_2, 1, y, 1)
daxpy(n, alpha2, x_3, 1, y, 1)

...  

Actually transpose a matrix  
(and other loop code)

dscal(n, beta, y, 1)  
daxpy(n, alpha, x, 1, y, 1)  (i.e. daxpby)

ddot(n, &one, 0, array, 1)  (i.e. non-absolute sum)

Lots of copying and data movement  
BLAS call
Lots more data movement  
Separate real and imaginary arrays/matrices
void bli_dgemm( trans_t transa,
      trans_t transb,
      dim_t   m,
      dim_t   n,
      dim_t   k,
      double*  alpha,
      double*  a,
      inc_t rsa, inc_t csa,
      double*  b,
      inc_t rsb, inc_t csb,
      double*  beta,
      double*  c,
      inc_t rsc, inc_t csc );

void bli_gemm( obj_t*   alpha,
      obj_t*   a,
      obj_t*   b,
      obj_t*   beta,
      obj_t*   c )
Tensors

- Tensors are essentially multi-dimensional arrays:
  \[
  \text{double } T[na][nb][ni][nj]; \quad \leftrightarrow \quad T_{ij}^{ab} \in \mathbb{R}^{n_a \times n_b \times n_i \times n_j}
  \]

- Matrices are 2-D tensors.

- Tensor indices are usually explicit. Einstein notation is very helpful:
  \[
  x^{abc}_{ijk} = (1 + \rho^{ai}_{ck} + \rho^{bj}_{ck}) \left[ A^{abc}_{ijm} r^{c}_{nk} v_{ef} - 2(1 + \rho^{ai}_{bj}) t^{c}_{ijm} r^{c}_{nk} v_{ef} \right. \\
  - 2 r^{c}_{ijm} t^{c}_{nk} v_{ef} - 2 r^{c}_{ijm} t^{c}_{nk} v_{fe} \\
  + (1 + \rho^{ai}_{bj}) t^{c}_{ijm} r^{c}_{nk} v_{ef} + (1 + \rho^{ai}_{bj}) t^{c}_{ijm} r^{c}_{nk} v_{fe} \\
  + t^{c}_{ijm} t^{c}_{nk} v_{ef} + (1 + \rho^{ai}_{bj}) t^{c}_{ijm} t^{c}_{nk} v_{fe} \right].
  \]
Tensors in Chemistry

\[ [\hat{H}, e^{\hat{T}}] \hat{R} |0\rangle = E \hat{R} |0\rangle \]

- Sequence of tensor contractions, summations, etc.
- Need to account for physics: spin, spatial symmetry, and so on.
- Some tensors are very big, some are very small, many different dimensionalities.
- Often need (distributed), out-of-core storage.

Collection of various tensors
Collection of various tensors
Collection of various tensors
High-level Interfaces: Chemistry

\[ F_i^m = f_i^m + \frac{1}{2} v_{mn}^m t_{ef}^m \]

\[ F_e^a = f_e^a - \frac{1}{2} v_{mn}^a t_{mn}^a \]

\[ W_{ij}^{mn} = v_{ij}^{mn} + \frac{1}{2} v_{mn}^e t_{ij}^e \]

\[ W_{ei}^{ma} = v_{ei}^{ma} + \frac{1}{2} v_{mn}^e t_{mn}^a \]

\[ z_{ij}^{ab} = v_{ij}^{ab} + P(ab) F_{ij}^b \]

\[ = v_{ij}^{ab} + P(ab) F_{ij}^b \]

\[ = v_{ij}^{ab} + P(ab) F_{ij}^b \]

\[ = v_{ij}^{ab} + P(ab) F_{ij}^b \]

\[ = v_{ij}^{ab} + P(ab) F_{ij}^b \]

\[ + \frac{1}{2} W_{ij}^{mn} t_{ij}^{mn} \]

\[ + \frac{1}{2} v_{ef}^{mn} t_{ij}^{ef} \]

\[ + P(ab) P(ij) W_{ei}^{ma} t_{mj}^{ab} \]

letter i, j, k, l, a, b, c, d;
btensor<2> f1_oo(oo), f1_vv(vv);
btensor<4> ii_oooo(oooo), ii_ovov(ovov);

\[
\begin{align*}
&\text{Compute intermediates} \\
&\text{f1_oo}(i|j) = \\
&\quad f_{oo}(i|j) + 0.5 \times \text{contract}(k|a|b, i_{ooov}(i|k|a|b), t2(i|k|a|b)); \\
&f1_vv(b|c) = \\
&\quad f_{vv}(b|c) - 0.5 \times \text{contract}(k|l|d, i_{ooov}(k|l|c|d), t2(k|l|b|d)); \\
&ii_oooo(i|j|k|l) = \\
&\quad i_{oooo}(i|j|k|l) + 0.5 \times \text{contract}(a|b, i_{ooov}(k|l|a|b), t2(i|j|a|b)); \\
&ii_ovov(i|a|j|b) = \\
&\quad i_{ovov}(i|a|j|b) - 0.5 \times \text{contract}(k|c, i_{ooov}(i|k|b|c), t2(k|j|c|a)); \\
\end{align*}
\]

\[
\begin{align*}
&\text{Compute updated T2} \\
&t2new(i|j|a|b) = \\
&\quad i_{ooov}(i|j|a|b) \\
&\quad + \text{asymm}(a, b, \text{contract}(c, i_{ooov}(i|j|a|b), f1_vv(b|c))) \\
&\quad - \text{asymm}(i, j, \text{contract}(k, t2(i|k|a|b), f1_oo(j|k))) \\
&\quad + 0.5 \times \text{contract}(k|l, ii_oooo(i|j|k|l), t2(k|l|a|b)) \\
&\quad + 0.5 \times \text{contract}(c|d, i_{vvvv}(a|b|c|d), t2(i|j|c|d)) \\
&\quad - \text{asymm}(a, b, \text{asymm}(i, j, \text{contract}(k|c, ii_ovov(k|b|j|c), t2(i|k|a|c))) ;
\end{align*}
\]
Low-level Interfaces: Blitz++, Eigen, Boost, BTAS, etc.

• Handle allocation, alignment, indexing, etc. “Native” efficiency.

```
Array<double,4> pqrs(np, nq, nr, ns);
pqrs[0][1][6][3] = 0.122;
```

• C++ allows for efficient and expressive interfaces (fixed vs. variable dimensionality, static typing, operator overloading, operation trees, etc.).

```
Array<int,3> A(4, 10, 2);
Array<int,1> G = A(2, 7, Range::all());
```

• Limited computational support (i.e. Level 1-type operations, exptl. contraction support in Eigen).

```
C(i,j) = sum(A(i,k), B(k,j), k) + Cprime(i,j);
```
Under the Hood: BLAS!

- Python/NumPy, Matlab
- libtensor, TCE etc.
- Eigen, BTAS
- Everything else

permute
permute
gemm
permute
Why not BLAS?

Solid lines: matrices
Dashed lines: tensors
Where to Draw the Line?

Domain Science Application (DSA)

Parallelization
- High-level domain logic
- Problem-dependent structure
- Data structure/distribution
- Blocking/pipelining
- Low-level kernels

Ease-of-use
- Error prevention
- Utility per LOC
- Specificity

Flexibility
- Performance and Optimization Opportunities
- Programmer Effort

Generality

Specificity
Where to Draw the Line?

Domain Science Application (DSA)

Parallelization
- High-level domain logic
- Problem-dependent structure
- Data structure/distribution
- Blocking/pipelining
- Low-level kernels

Linear Algebra (Matrices):
- BLAS/LAPACK
- Elemental, ScaLAPACK, etc.
- Global Arrays, etc.
- PETSc, Trilinos, etc.
Where to Draw the Line?

Domain Science Application (DSA)

Paralleling:
- High-level domain logic
- Problem-dependent structure
- Data structure/distribution
- Blocking/pipelining
- Low-level kernels

Multilinear Algebra (Tensors):
- libtensor, TCE, TiledArrays, etc.
- CTF, ROTE
- ??? (BLAS for now)
err_t tensor_dcontract(
   double alpha,
   const double* A,
   gint_t ndim_A,
   const dim_t* len_A,
   const inc_t* stride_A,
   const idx_t* idx_A,
   const double* B,
   gint_t ndim_B,
   const dim_t* len_B,
   const inc_t* stride_B,
   const idx_t* idx_B,
   double beta,
   double* C,
   gint_t ndim_C,
   const dim_t* len_C,
   const inc_t* stride_C,
   const idx_t* idx_C);

• Data externally specified as in BLAS.

• Custom integral types for lengths, strides, number of dimensions.

• Any non-negative number of dimensions.

• Integral error code.

• Contracted/non-contracted dimensions specified by index strings (integral, string literal, etc.) + Einstein notation.

• No “TRANSA” etc.
## What: A (Possible) “Tensor BLAS”

<table>
<thead>
<tr>
<th>BLAS</th>
<th>Level</th>
<th>Tensor BLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>GEMM, TRSM, SYRK, etc.</td>
<td>Level 3</td>
<td>Binary CONTRACT, WEIGHT, etc. → MULT</td>
</tr>
<tr>
<td>GEMV, TRSV, GER, etc.</td>
<td>Level 2</td>
<td>Unary TRANSPOSE, TRACE, etc. → SUM</td>
</tr>
<tr>
<td>DOT, COPY, AXPY, etc.</td>
<td>Level 1</td>
<td>Local REDUCE, SCALE, etc.</td>
</tr>
</tbody>
</table>
Why:

- **Flexibility:**
  - Any storage order is allowed: column-major, row-major, or a mix. Strides do not need to be multiples of each other.
  - Transposition and contracted/non-contracted indices are implicit. Index names are not prescribed.
Why:

• Flexibility:
  – Any storage order is allowed: column-major, row-major, or a mix. Strides do not need to be multiples of each other.
  – Transposition and contracted/non-contracted indices are implicit. Index names are not prescribed.

• Portability:
  – Types (esp. integral) are user-definable, as in BLIS. No 32/64-bit confusion.
  – Pure C interface.
Why:

• Flexibility:
  – Any storage order is allowed: column-major, row-major, or a mix. Strides do not need to be multiples of each other.
  – Transposition and contracted/non-contracted indices are implicit. Index names are not prescribed.

• Portability:
  – Types (esp. integral) are user-definable, as in BLIS. No 32/64-bit confusion.
  – Pure C interface.

• Extensibility:
  – All information needed for the operation is explicit (as in BLAS). Can be used directly or wrapped in another interface.
How: BLIS

1st loop around micro-kernel

2nd loop around micro-kernel

3rd loop around micro-kernel

4th loop around micro-kernel

5th loop around micro-kernel

Tensor physical layout

Matrix-like logical layout

Packed internal layout

Packed "A" → \(\tilde{A}_i\)

Packed "B" → \(\tilde{B}_p\)
Thanks