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):

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

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

$$\begin{aligned} &P(Z_{(m,n)} = k \mid \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ \propto &P(Z_{(m,n)} = k, \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ = &\left(\frac{\Gamma\left(\sum_{i=1}^K \alpha_i\right)}{\prod_{i=1}^K \Gamma(\alpha_i)}\right)^M \prod_{j\neq m} \frac{\prod_{i=1}^K \Gamma(n_{j,(\cdot)}^i + \alpha_i)}{\Gamma\left(\sum_{i=1}^K n_{j,(\cdot)}^i + \alpha_i\right)} \end{aligned}$$

Math to Code: Fundamental Considerations

What domain scientists want (math):

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

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

$$\begin{aligned} & P(Z_{(m,n)} = k \mid \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ \propto & P(Z_{(m,n)} = k, \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ = & \left(\frac{\Gamma\left(\sum_{i=1}^{K}\alpha_i\right)}{\prod_{i=1}^{K}\Gamma(\alpha_i)}\right)^M \prod_{j\neq m}\frac{\prod_{i=1}^{K}\Gamma(n_{j,(\cdot)}^i + \alpha_i)}{\Gamma\left(\sum_{i=1}^{K}n_{j,(\cdot)}^i + \alpha_i\right)} \end{aligned}$$

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):

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

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

$$\begin{aligned} &P(Z_{(m,n)} = k \mid \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ &\propto P(Z_{(m,n)} = k, \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ &= \left(\frac{\Gamma\left(\sum_{i=1}^{K}\alpha_i\right)}{\prod_{i=1}^{K}\Gamma(\alpha_i)}\right)^M \prod_{j\neq m}\frac{\prod_{i=1}^{K}\Gamma(n^i_{j,(\cdot)}+\alpha_i)}{\Gamma\left(\sum_{i=1}^{K} n^i_{j,(\cdot)}+\alpha_i\right)} \end{aligned}$$

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):

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

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

$$\begin{aligned} & P(Z_{(m,n)} = k \mid \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ \propto & P(Z_{(m,n)} = k, \boldsymbol{Z}_{-(m,n)}, \boldsymbol{W}; \alpha, \beta) \\ = & \left(\frac{\Gamma\left(\sum_{i=1}^K \alpha_i\right)}{\prod_{i=1}^K \Gamma(\alpha_i)}\right)^M \prod_{j\neq m} \frac{\prod_{i=1}^K \Gamma(n_{j,(\cdot)}^i + \alpha_i)}{\Gamma(\sum_{i=1}^K n_{j,(\cdot)}^i + \alpha_i)} \end{aligned}$$

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

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

BLIS

```
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)
```

```
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:

`double T[na][nb][ni][nj];` ⟷ $T^{ab}_{ij} \in \mathbb{R}^{n_a \times n_b \times n_i \times n_j}$

- Matrices are 2-D tensors.

- Tensor indices are usually explicit. Einstein notation is very helpful:

$$\begin{aligned}
\check{z}^{abc}_{ijk} &= (1+P^{ai}_{ck}+P^{bj}_{ck})\Big[4\check{t}^{abe}_{ijm}\check{t}^{fc}_{nk}\check{v}^{mn}_{ef} - 2(1+P^{ai}_{bj})\check{t}^{aeb}_{ijm}\check{t}^{fc}_{nk}\check{v}^{mn}_{ef} \\
&\quad -2\check{t}^{abe}_{ijm}\check{t}^{cf}_{nk}\check{v}^{mn}_{ef} - 2\check{t}^{abe}_{ijm}\check{t}^{fc}_{nk}\check{v}^{mn}_{fe} \\
&\quad +(1+P^{ai}_{bj})\check{t}^{aeb}_{ijm}\check{t}^{cf}_{nk}\check{v}^{mn}_{ef} + (1+P^{ai}_{bj})\check{t}^{aeb}_{ijm}\check{t}^{fc}_{nk}\check{v}^{mn}_{fe} \\
&\quad +\check{t}^{abe}_{ijm}\check{t}^{cf}_{nk}\check{v}^{mn}_{fe} + (1+P^{ai}_{bj})\check{t}^{aec}_{ijm}\check{t}^{bf}_{nk}\check{v}^{mn}_{fe}\Big].
\end{aligned}$$

Tensors in Chemistry

Collection of various tensors

Collection of various tensors

Collection of various tensors

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

- 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.

High-level Interfaces: Chemistry

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

$$F_e^a = f_e^a - \frac{1}{2} v_{ef}^{mn} t_{mn}^{af}$$

$$W_{ij}^{mn} = v_{ij}^{mn} + \frac{1}{2} v_{ef}^{mn} t_{ij}^{ef}$$

$$W_{ei}^{ma} = v_{ei}^{ma} + \frac{1}{2} v_{ef}^{mn} t_{in}^{af}$$

$$\begin{aligned} z_{ij}^{ab} =& v_{ij}^{ab} \\ & + P(ab) F_e^a t_{ij}^{eb} \\ & - P(ij) F_i^m t_{mj}^{ab} \\ & + \frac{1}{2} W_{ij}^{mn} t_{mn}^{ab} \\ & + \frac{1}{2} v_{ef}^{ab} t_{ij}^{ef} \\ & + P(ab) P(ij) W_{ei}^{ma} t_{mj}^{eb} \end{aligned}$$

```
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);

// Compute intermediates
f1_oo(i|j) =
        f_oo(i|j) + 0.5 * contract(k|a|b, i_oovv(j|k|a|b), t2(i|k|a|b));
f1_vv(b|c) =
        f_vv(b|c) - 0.5 * contract(k|l|d, i_oovv(k|l|c|d), t2(k|l|b|d));
ii_oooo(i|j|k|l) =
        i_oooo(i|j|k|l) + 0.5 * contract(a|b, i_oovv(k|l|a|b), t2(i|j|a|b));
ii_ovov(i|a|j|b) =
        i_ovov(i|a|j|b) - 0.5 * contract(k|c, i_oovv(i|k|b|c), t2(k|j|c|a));

// Compute updated T2
t2new(i|j|a|b) =
        i_oovv(i|j|a|b)
      + asymm(a, b, contract(c, t2(i|j|a|c), f1_vv(b|c)))
      - asymm(i, j, contract(k, t2(i|k|a|b), f1_oo(j|k)))
      + 0.5 * contract(k|l, ii_oooo(i|j|k|l), t2(k|l|a|b))
      + 0.5 * contract(c|d, i_vvvv(a|b|c|d), t2(i|j|c|d))
      - asymm(a, b, asymm(i, j,
           contract(k|c, ii_ovov(k|b|j|c), t2(i|k|a|c))));
```

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;
```

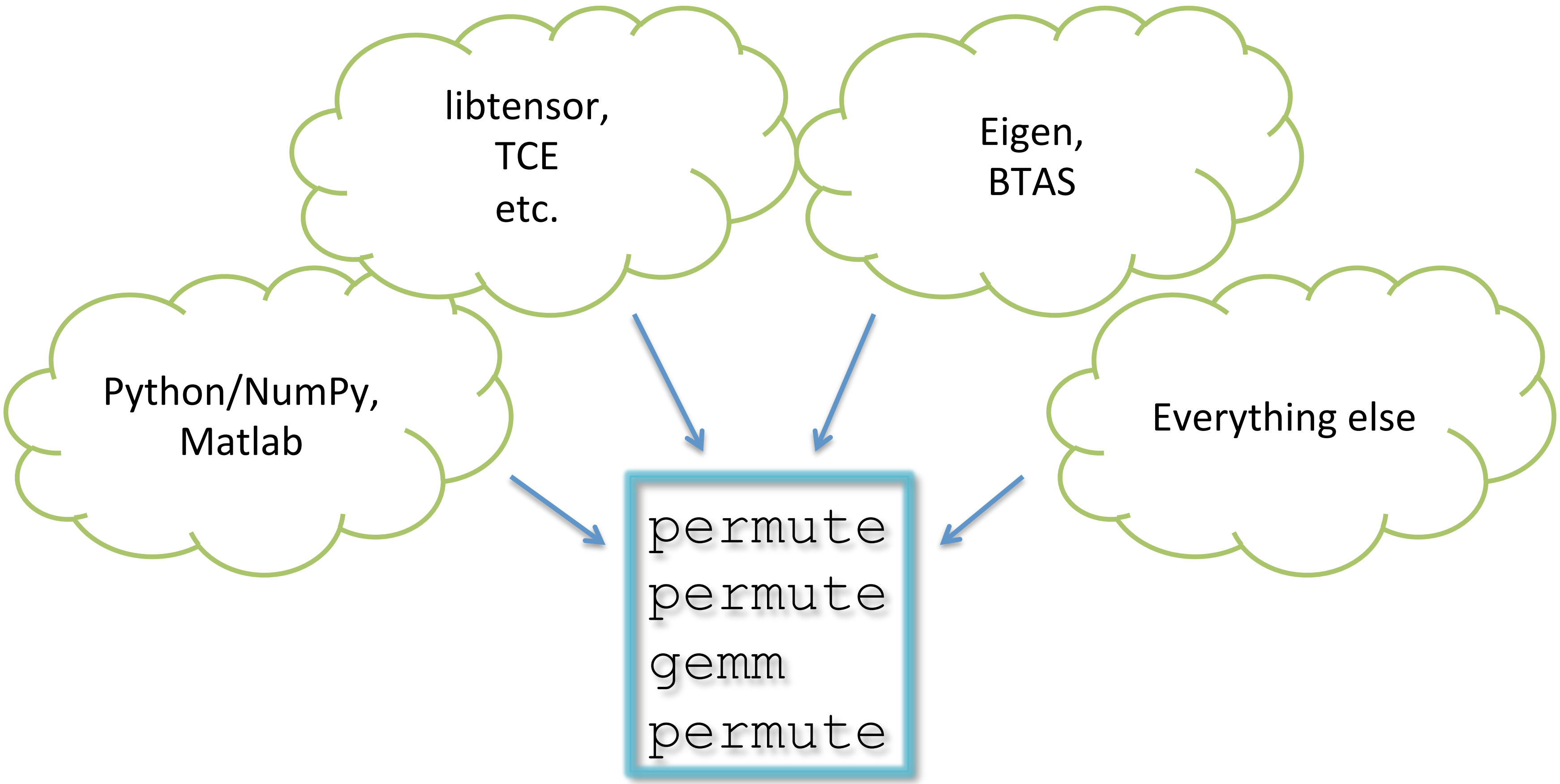
- C++ allows for efficient and expressive interfaces (fixed vs. variable dimensionality, static typing, operator overloading, operation trees, etc.).

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

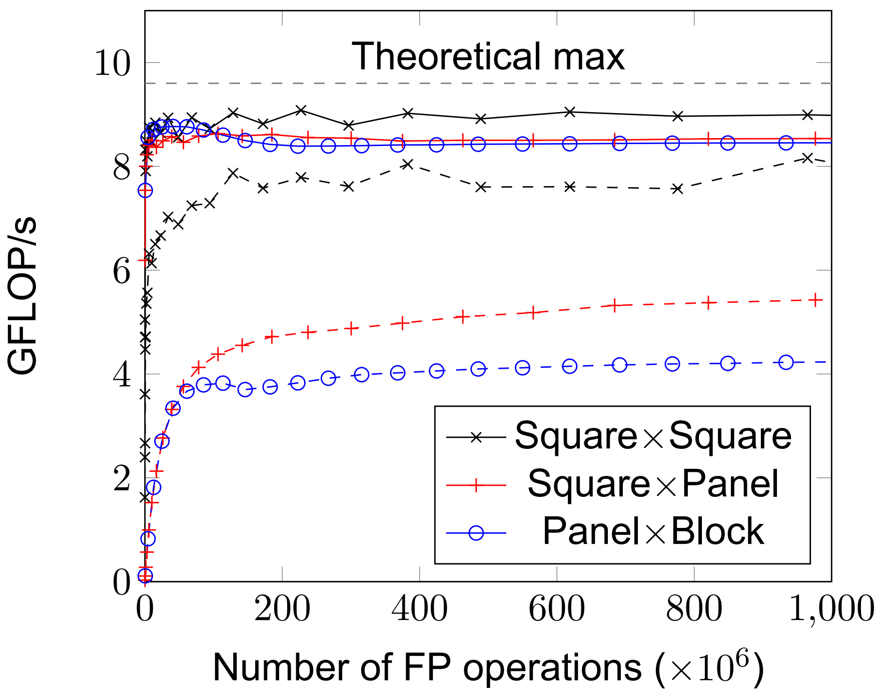
- Limited computational support (i.e. Level 1-type operations, exptl. contraction support in Eigen).

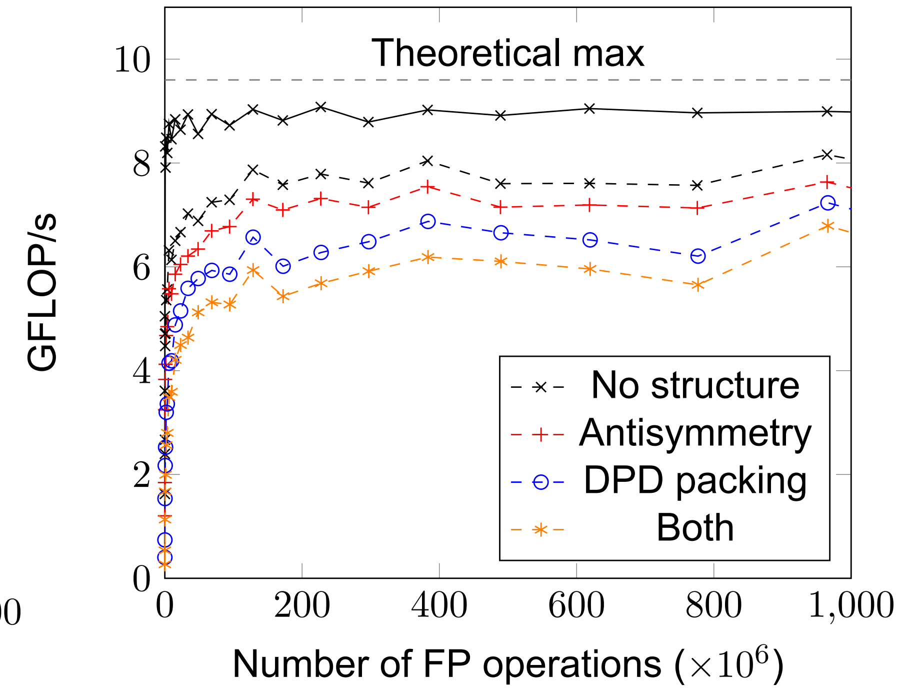
```
C(i,j) = sum(A(i,k), B(k,j), k) + Cprime(i,j);
```

Under the Hood: BLAS!



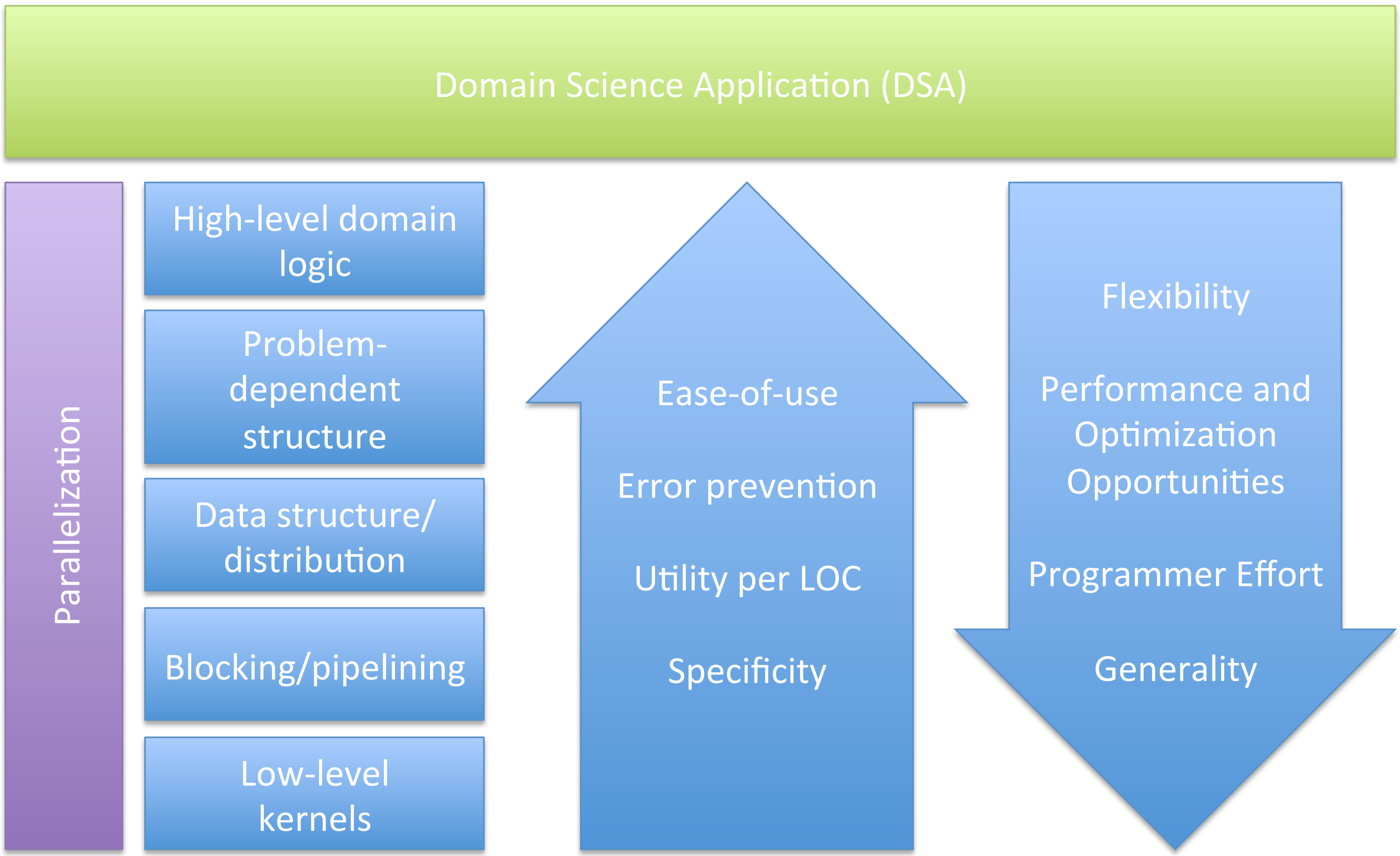
Why not BLAS?



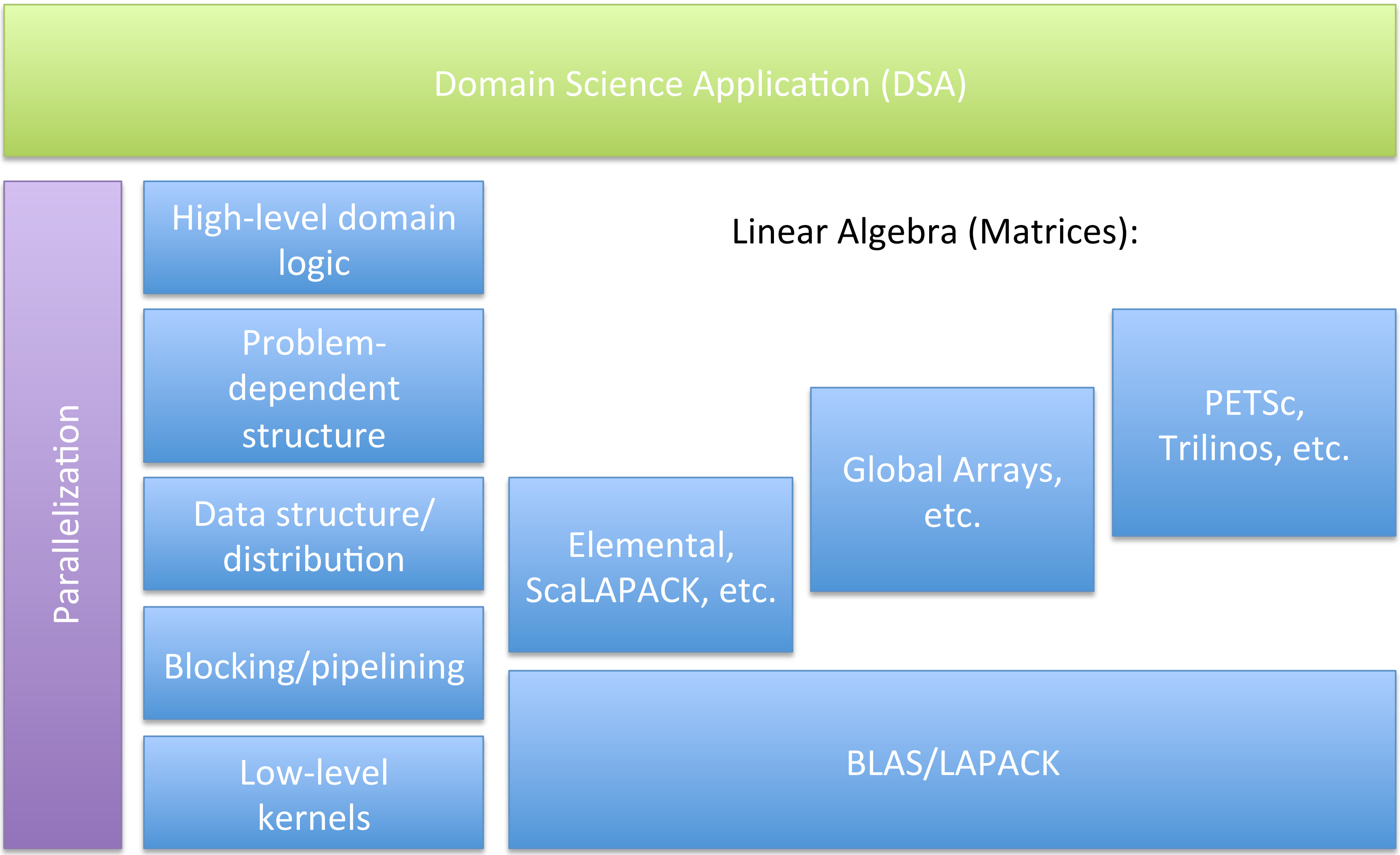


Solid lines: matrices
Dashed lines: tensors

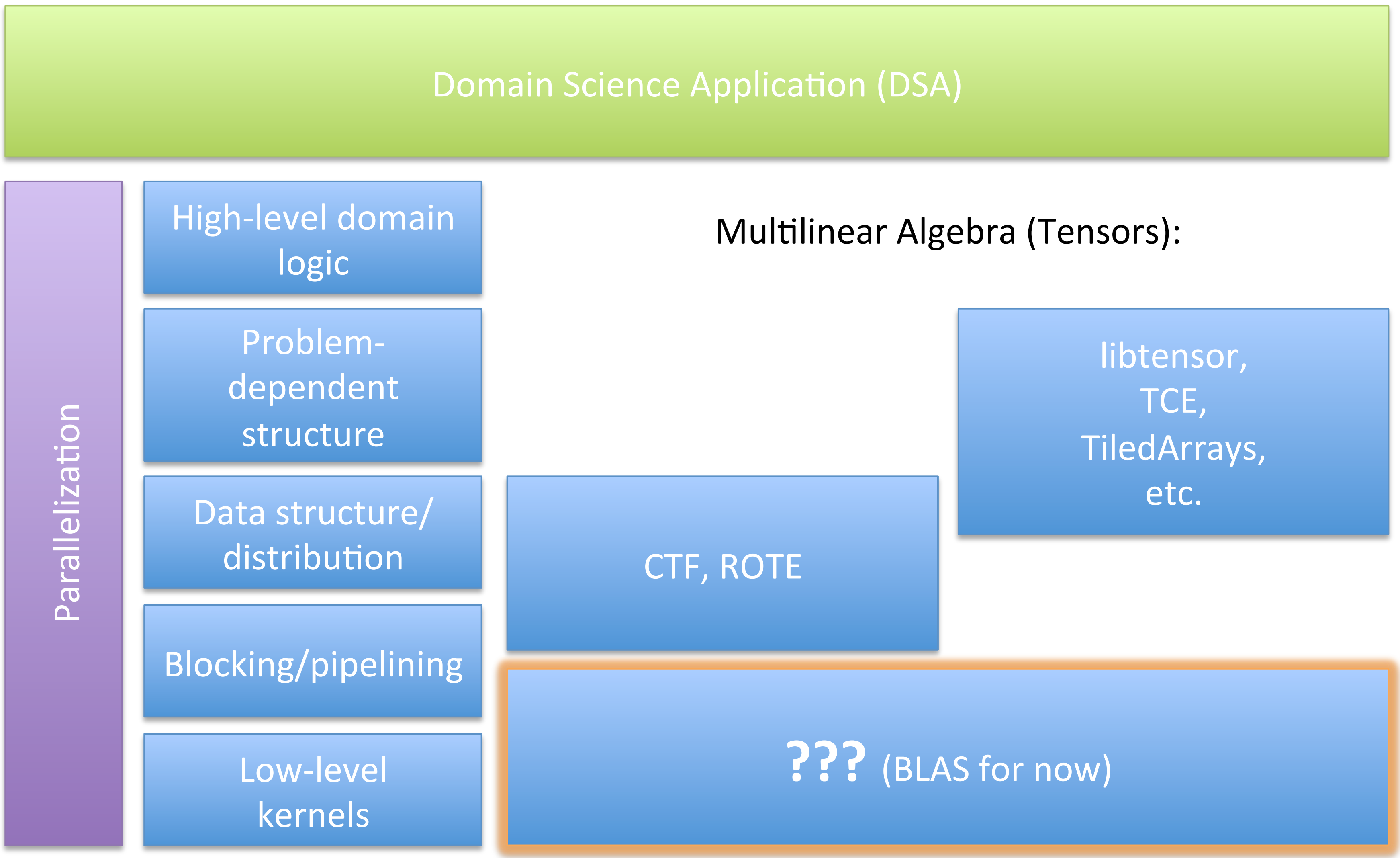
Where to Draw the Line?



Where to Draw the Line?



Where to Draw the Line?



What: A (Possible) “Tensor BLAS”

```
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”

BLAS		Tensor BLAS	
GEMM, TRSM, SYRK, etc.	Level 3	Binary	CONTRACT, WEIGHT, etc. → MULT
GEMV, TRSV, GER, etc.	Level 2	Unary	TRANSPOSE, TRACE, etc. → SUM
DOT, COPY, AXPY, etc.	Level 1	Local	REDUCE, SCALE, etc.

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.

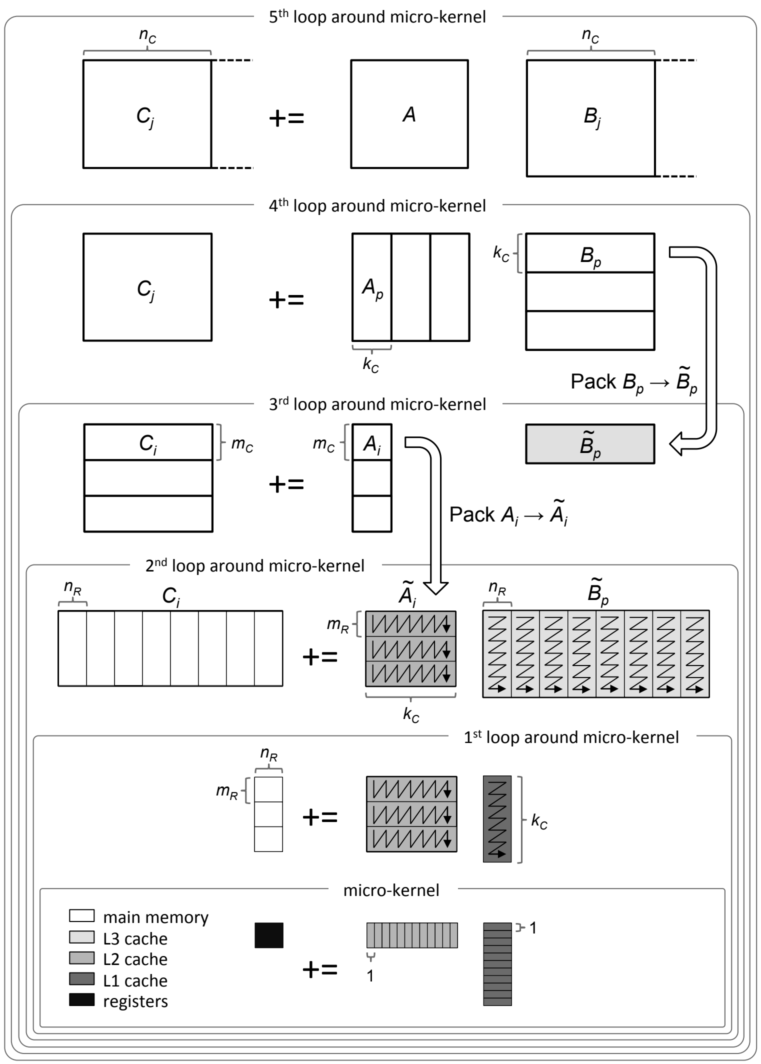
Why:

- 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.

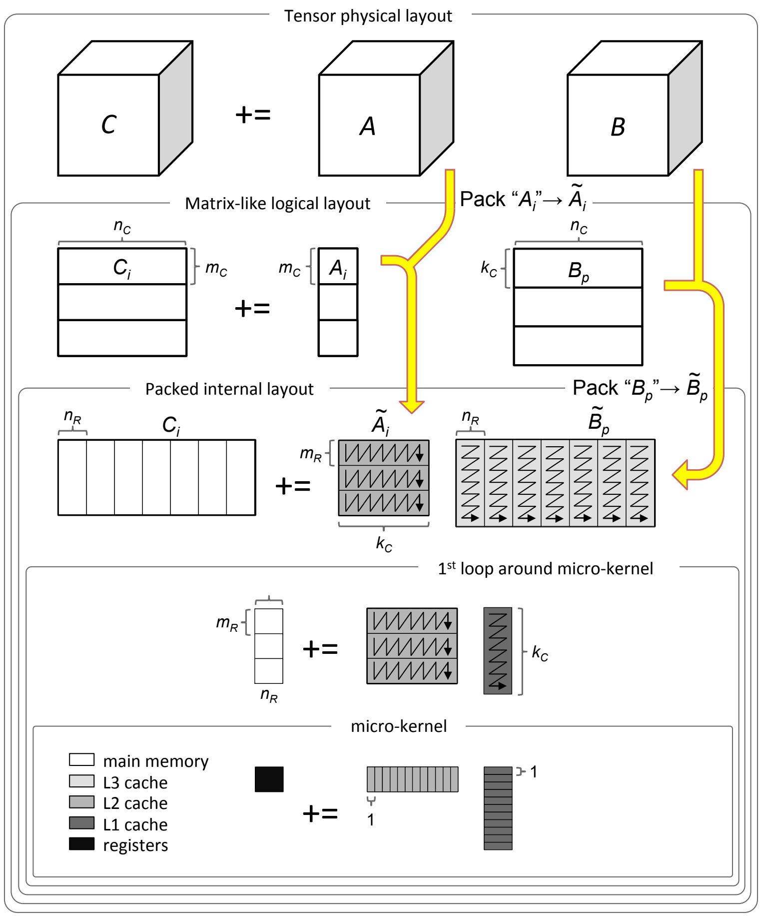
- Portability:
  - Types (esp. integral) are user-definable, as in BLIS. No 32/64-bit confusion.
  - Pure C interface.

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

How: BLIS



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 micro-kernel
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers
1



Tensor physical layout
C
+=
A
B
Pack "A_i" → $\tilde{A}_i$
Matrix-like logical layout
n_C
C_i
m_C
A_i
k_C
B_p
Packed internal layout
Pack "B_p" → $\tilde{B}_p$
n_R
$\tilde{A}_i$
m_R
$\tilde{B}_p$
1st loop around micro-kernel
micro-kernel
main memory
L3 cache
L2 cache
L1 cache
registers
1

Thanks

The Science of
High-Performance
Computing Group

SHPC

THE UNIVERSITY OF
TEXAS
AT AUSTIN

ICES

NSF