Using MLIR to Optimize Tensor Contractions

Rui Li[*], Atanas Rountev[+], Saday Sadayappan[*]

[*]University of Utah
[+]Ohio State University

Tiling Tensor Contractions: Huge Design Space

```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   for (l=0; l<N; l++)
    for (m=0; m<N; m++)
     for (n=0; n<N; n++)
      C[i][j][k][l] += A[i][m][k][n]*B[j][n][l][m];
```

$$C_{ijkl} = \sum_{mn} A_{imkn} \cdot B_{jnlm}$$

- Tensor contraction: high-dimension analog of matrix multiplication
  - Direct implementation as above loop code results in low performance
- Enormous space of loop permutation/tiling + tile size selection:
  - Fully permutable, i.e., 6! permutations
  - For two level memory hierarchy, 2 sets of tiling loops: $(6!)^3$ choices
  - Consider 5 possible values for each tile-size: $(5^{12})^* (6!)^3 = 9.1^*10^{16}$ choices
- Challenging for two-step iterative optimization with polyhedral compilers
  1. Find tiled loop structure for assumed tilesizes, using linear cost model
  2. Auto-tune for different tile size combinations

Domain-Specific Optimization: Tensor Contractions

```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   for (l=0; l<N; l++)
    for (m=0; m<N; m++)
     for (n=0; n<N; n++)
      C[i][j][k][l] += A[i][m][k][n]*B[j][n][l][m];
```

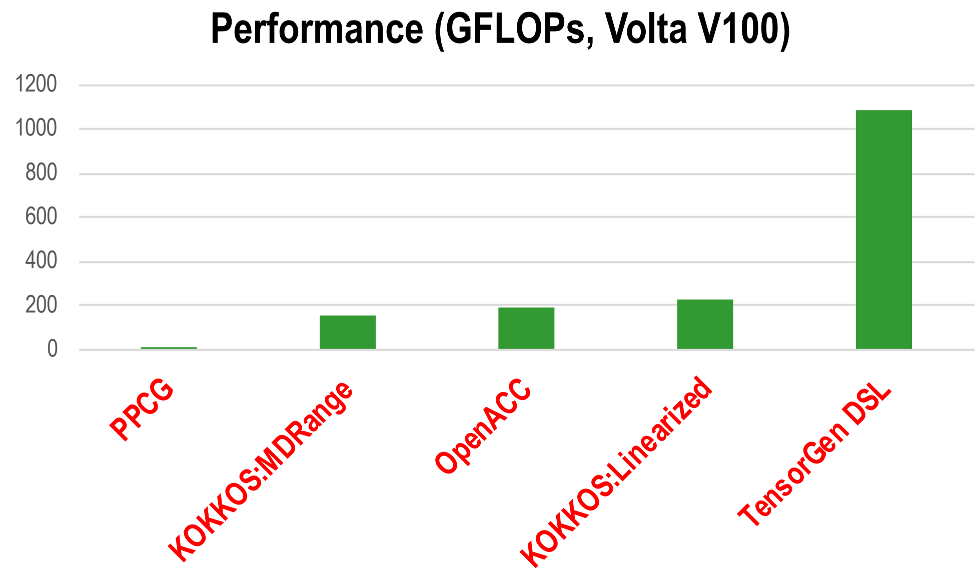
$$C_{ijkl} = \sum_{mn} A_{imkn} \cdot B_{jnlm}$$

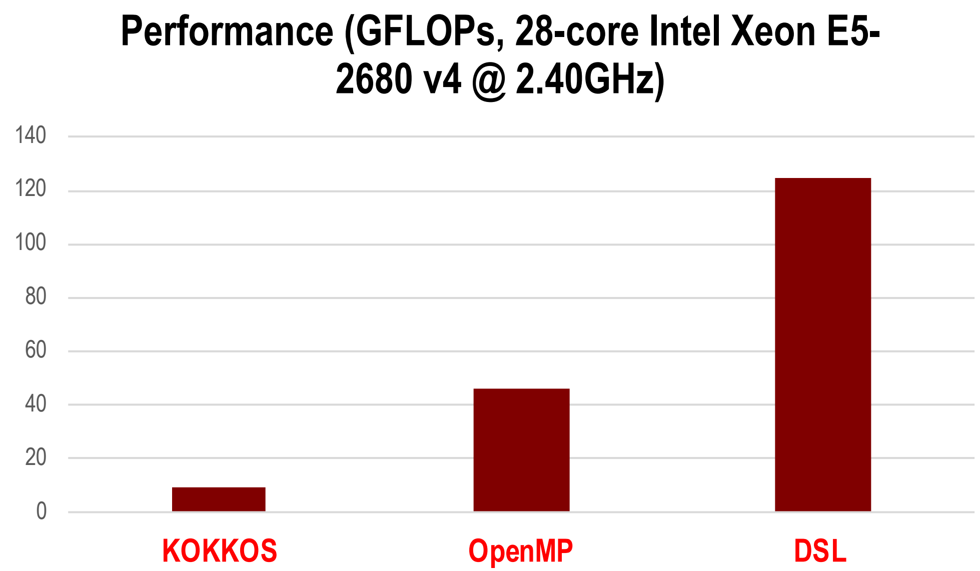
- Enormous space of loop permutation/tiling + tile size selection: very challenging nonlinear optimization problem
  - Linear cost models in polyhedral compilers inadequate for effective selection
  - Domain-specific optimizing compiler can overcome the problem
- Each loop indexes exactly two tensors – group into three sets: C, E1, E2
  - "Contraction index" appears only in input (rhs) tensors: C {m, n}
  - "External index": appears in output tensor and one input tensor: E1 {i, k}, E2{j, l}
- DSL compiler can exploit a key property: Every index within a set is a reuse direction for exactly one of the 3 tensors (the one it does not index)
  - Model degree of reuse for each tensor as product of tile sizes along its reuse-set
  - Very efficient and effective model-driven heuristic search to solve permutation+tile-size-selection problem

CCSD(T) Tensor Contractions: DSL vs. Gen-Purpose

t3[k, j, i, c, b, a] -= t2[d, a, i, j] * v2[d, k, c, b]

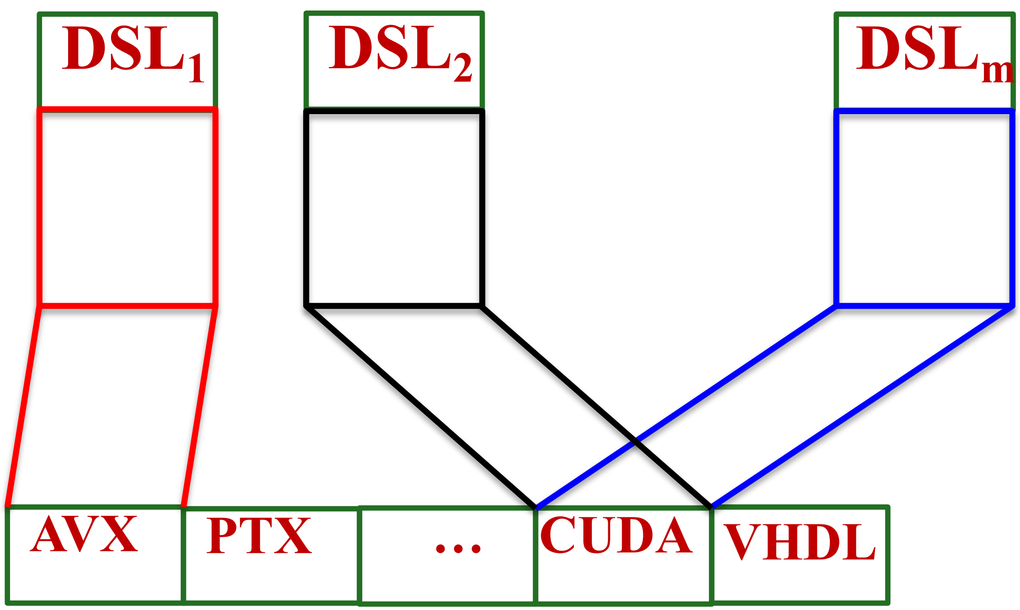
- CCSD(T) is an accurate but extremely compute-intensive method in NWChem
- TensorGen DSL compiler (with PNNL) achieves significantly higher performance than general-purpose compilers
- Question: Can customized optimization be incorporated in compilers/frameworks with broader use?





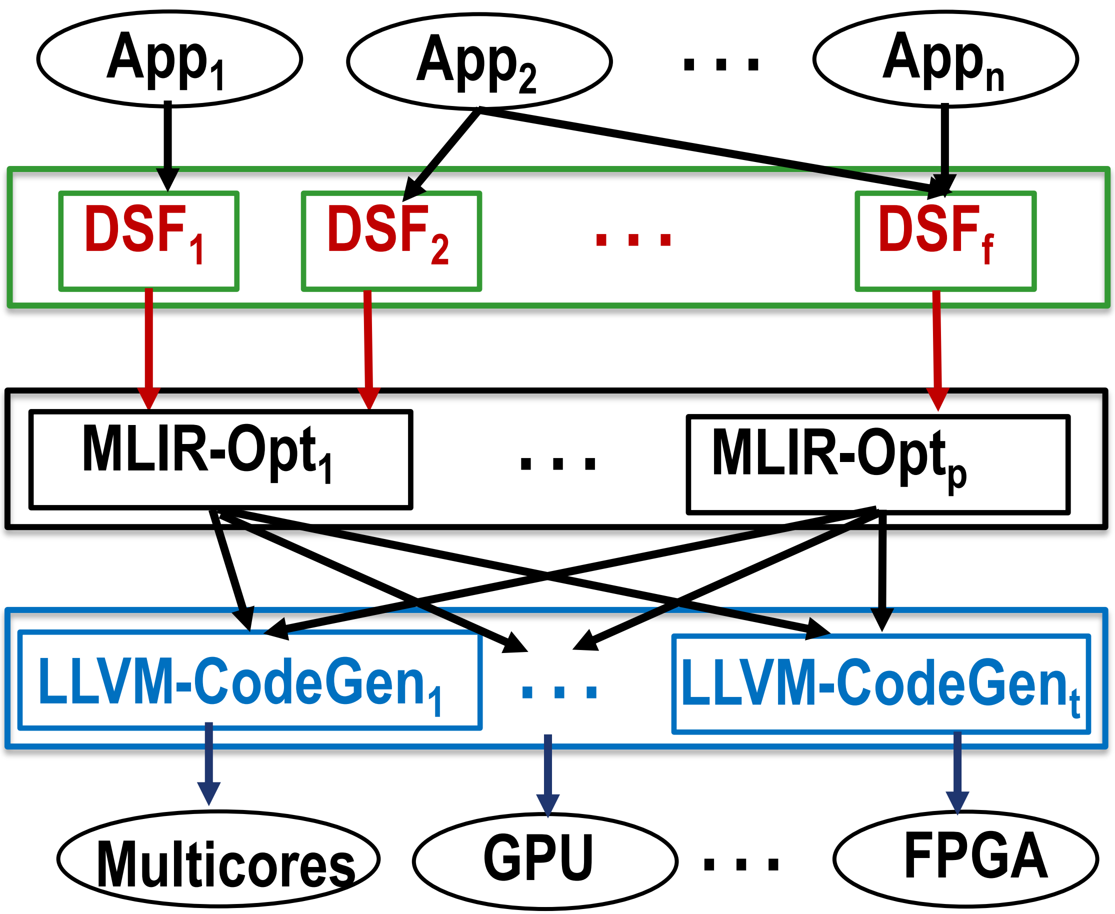
Domain-Specific Compilers: Strengths & Weaknesses

- Multi-target DSLs achieve performance, portability and productivity:
  - Domain-specific internal representation (IR) that facilitates efficient and effective choice of mapping/scheduling of computation/data
  - Separation of high-level target-independent decisions from low-level platform-specific choices
  - Platform-specific code-schema driven by key performance factors
- But each DSL (compiler or library) today is a stand-alone system: no reuse; redundant re-implementation of shared functionality



High-Level Transformations in MLIR

- Layered transformation/optimization
  - DSLs perform domain-specific transformations and invoke pattern-centric optimizers
  - MLIR passes for coarse-grained mapping/scheduling of computations/data-movement using abstracted architectural parameters (e.g., capacities/bandwidths in mem. hierarchy)
  - LLVM for lower-level target-platform-specific code generators generate API-specific code



Using MLIR for Optimizing Tensor Contraction

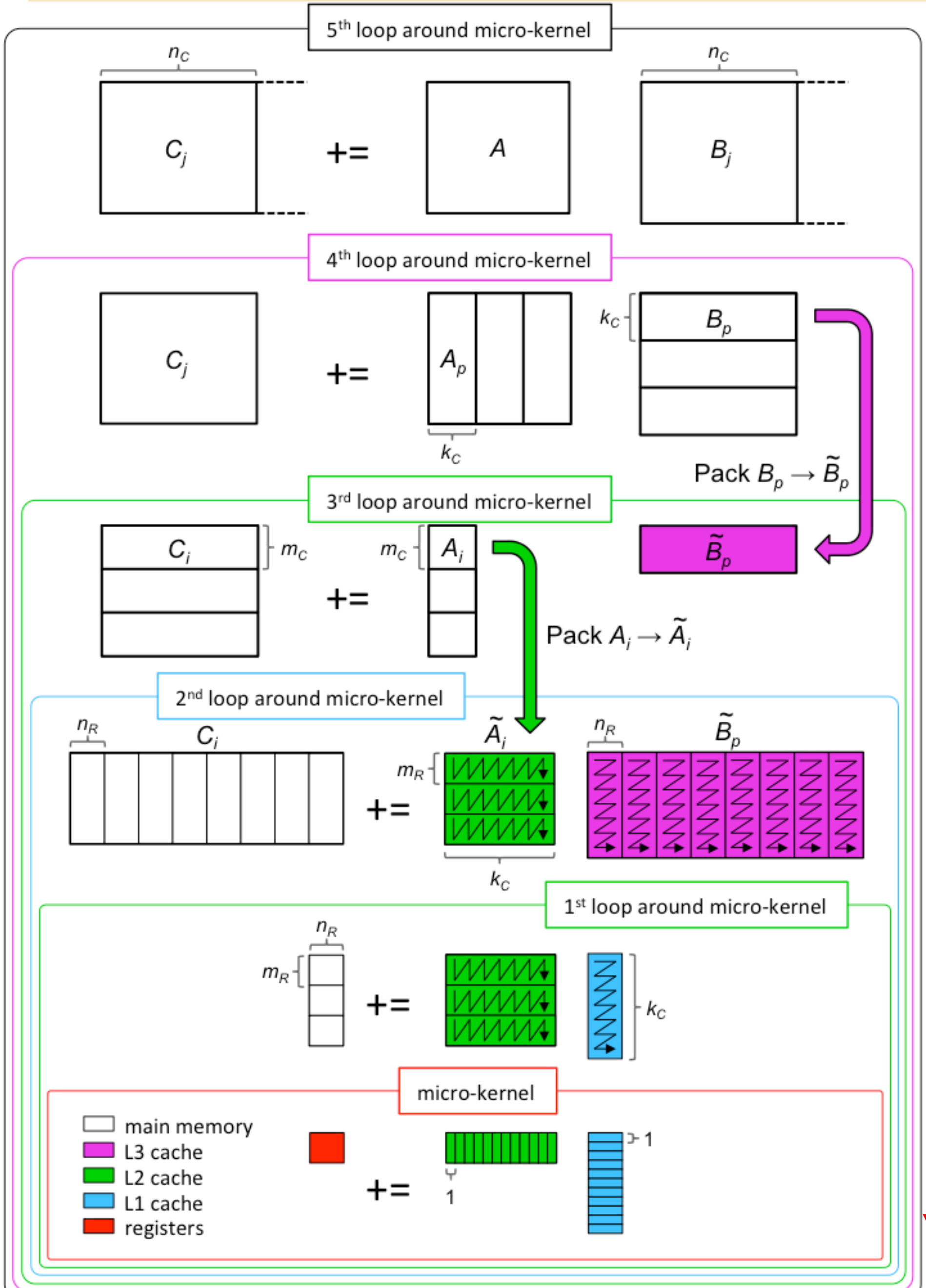
- Efficient tiled execution of tensor contractions

  - Use BLIS microkernel at the lowest level for “panel-panel” outer-product

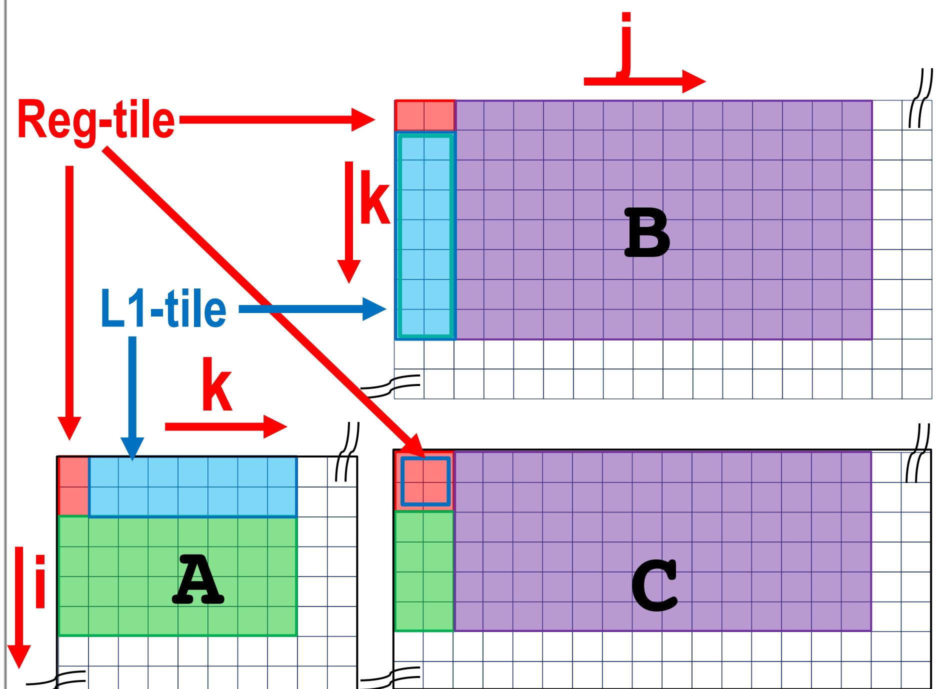
  - Exploit property that loop indices map into one of three groups: contraction-indices, left-external indices, and right-external indices

  - Tile the space of 3 macro-indexes: contraction, ext-left and ext-right; tile sizes determined by solving non-convex tile-size optimization problem

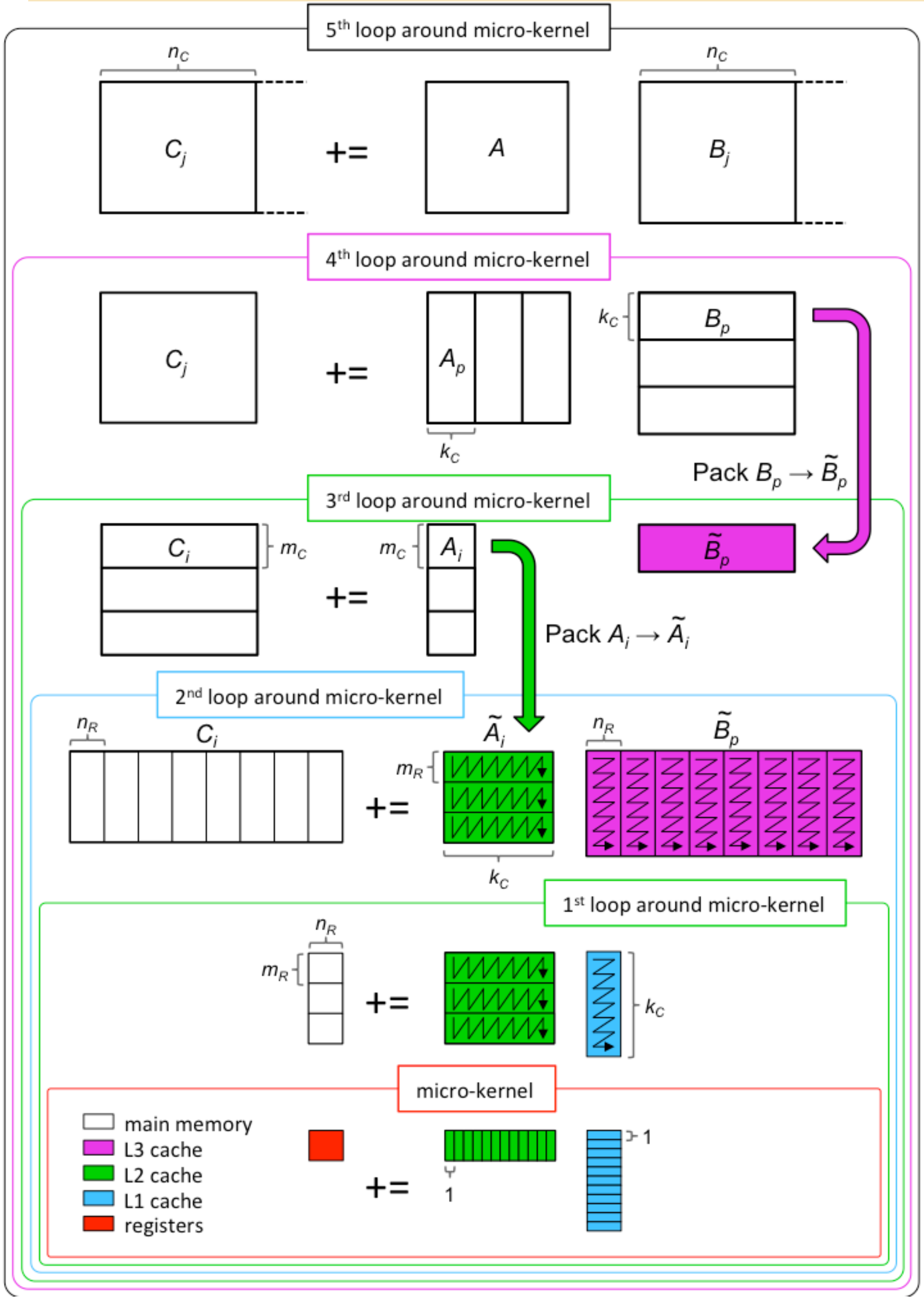
BLIS Schema for Multi-Level Tiled Mat-Mult




```
for (jm=0; jm<N; jm+=Nc)
 for (km=0; km<K; km+=Kc)
 { Pack B[km:km+Kc,jm:jm+Nc] => Bbuf[Kc,Nc];
   for (im=0; im<M; im+=Mc)
   { Pack A[im:im+Mc,km:km+Kc] => Abuf[Nc,Kc]
     for (jc=jm; jc<jm+Nc; jc+=Nr)
       for (ic=im; ic<im+Mc; ic+=Mr)
         BLIS_Kernel(Abuf, Bbuf, @C[ic,jc],Kr);
   }
 }
```



BLIS Schema using MLIR




```
for (jm=0; jm<N; jm+=Nc)
 for (km=0; km<K; km+=Kc)
 { Pack B[km:km+Kc,jm:jm+Nc] => Bbuf[Kc,Nc];
   for (im=0; im<M; im+=Mc)
   { Pack A[im:im+Mc,km:km+Kc] => Abuf[Nc,Kc]
     for (jc=jm; jc<jm+Nc; jc+=Nr)
       for (ic=im; ic<im+Mc; ic+=Mr)
         BLIS_Kernel(Abuf, Bbuf, @C[ic,jc],Kr);
   }
 }
```

```
%Abuf : buffer<589824>       %i2 = range 0:16:1
%Bbuf : buffer<589824>       %j2 = range 0:1:1
%Cbuf : buffer<589824>       %k2 = range 0:1:1
%i0 = range 0:6:1            %i3 = range 0:1:1
%j0 = range 0:16:1           %j3 = range 0:48:1
%i1 = range 0:2:1            %k3 = range 0:1:1
%j1 = range 0:1:1            %i4 = range 0:4:1
%k1 = range 0:256:1          %j4 = range 0:1:1
                             %k4 = range 0:3:1
```

```
%A = view %Abuf[%i4,%i3,%i2,%i1,%i0,%k4,%k3,%k2,%k1]
%B = view %Bbuf[%j4,%j3,%j2,%j1,%j0,%k4,%k3,%k2,%k1]
%C = view %Cbuf[%i4,%i3,%i2,%i1,%i0,%j4,%j3,%j2,%j1,%j0]
for %i4it in %i4, %j4it in %j4, %k4it in %k4  {
 %Aslice = slice %A[%i4it,%i3,%i2,%i1,%i0,%k4it,%k3,%k2,%k1]
 %AbufL3 = alloc_buffer(49152)
 %AL3 = view %AbufL3[%k3,%i3,%k2,%i2,%i1,%k1,%i0]
 copy(%Aslice, %AL3) { outputPermutation =
   (a3,a2,a1,a0,b3,b2,b1) -> (b3,a3,b2,a2,a1,b1,a0) }
 // Similarly for %Bslice, %BbufL3, %BL3
 for %i3it in %i3, %j3it in %j3, %k3it in %k3, %i2it in %i2,
     %j2it in %j2, %k2it in %k2, %i1it in %i1, %j1it in %j1 {
  %Aker = slice %AL3[%k3it,%i3it,%k2it,%i2it,%i1it,%k1,%i0]
  %Bker = slice %BL3[%k3it,%j3it,%k2it,%j2it,%j1it,%k1,%j0]
  %Cker = slice %C[%i4it,%i3it,%i2it,%i1it,%i0,
                   %j4it,%j3it,%j2it,%j1it,%j0]
  blis_kernel(%Aker,%Bker,%Cker) } }
```

Ongoing Work

- Extend data-packing from mat-mult to tensor contractions
  - Exploit property that loop indices map into one of three groups: contraction-indices, left-external indices, and right-external indices
  - Tiling is in the space of 3 macro-indexes: contraction, ext-left and ext-right
  - Need to pack data corresponding to multi-level tile into buffer for contiguous access within each tile
  - Challenge is that data footprints in tiled macro-index space are not regular sections in tensor's index space

```
for (i=0; i<N; i++)
 for (j=0; j<N; j++)
  for (k=0; k<N; k++)
   for (l=0; l<N; l++)
    for (m=0; m<N; m++)
     for (n=0; n<N; n++)
      C[i][j][k][l] += A[i][m][k][n]*B[j][n][l][m];
```

EL = {i,k} **ER = {j,l}** **CI = {m,n}**