A framework for mapping and redistribution of multidimensional distributed arrays

Edgar Solomonik

Department of EECS, UC Berkeley

Dec 5, 2013

Outline

1. Introduction: arrays, tensors, and graphs
   - Arrays are tensors
   - Graphs are matrices
   - Tensors are hypergraphs
2. Mapping a distributed array
   - Why?
   - How does CTF do it?
3. Migrating distributed array data
   - Why?
   - How does CTF do it?
4. Performance
5. Conclusions and future work

Arrays as tensors

An array is a data-storage format for a tensor

- zero-dimensional tensors are scalars (zero-dimensional arrays)
- 1D tensors are vectors (regular arrays)
- 2D tensors are matrices (2D arrays, two nested pointers in C++)
- generally a tensor can be stored in an array of the same dimension

Tensors can also be linearized in memory and stored in a 1D (linear) array

- since memory is linear only one dimension of an array may be accessed at a given time without unit stride
- we can call the stride ordering of the tensor dimensions the embedding or mapping of the tensor in memory

Graphs as matrices

A graph $G = (V, E)$ is a set of vertices V and edges $E \subset V \times V$, for example,

- $V = \{v_1, v_2, v_3, v_4\}$
- $E = \{(v_1, v_2), (v_2, v_4), (v_1, v_4), (v_3, v_4)\}$

Any graph may be represented as a matrix, we may represent the connectivity of the undirected, unweighted graph G as a symmetric adjacency matrix

$$\begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix}$$

The diagonal would be nonzero if the graph had loops, the non-zero entries would be non-unit if the graph was weighted, and the matrix would be non-symmetric if the graph was directed.

Graph operations as matrix operations

Case 1: particle-particle interactions $G = (P, F)$

- the vertices P correspond to particles, which we can store in vector $\mathbf{p}$
- the edges F correspond to forces between particles, which we can store in matrix $\mathbf{F}$
- we define $\odot$ to compute the force between two particles, and $\otimes$ to compute the effect of a force on a particle
- $\mathbf{F} = \mathbf{p} \odot \mathbf{p}$ and $\mathbf{p}^{\text{new}} = \mathbf{F} \otimes \mathbf{p}$
- $F_{ij} = p_i \odot p_j$ and $p_i^{\text{new}} = \sum_j F_{ij} \otimes p_j$

Graph operations as matrix operations

Case 2: single source v_0 shortest paths (Bellman-Ford) with adjacency graph $G = (V, E)$

- we construct the distance graph $G' = (D, E)$ starting with all $d_i \in D$, set to $d_i = \infty$ except for $d_1 = 0$
- we can store D in a vector **d** and E as a matrix **E**
- Bellman-Ford computes for each j, $d_i^{\text{new}} = \min(d_i, E_{ij} + d_j)$
- if we redefine $\odot$ to be addition and $\sum$ to be min
- $d_i^{\text{new}} = \min(d_i, \sum_j E_{ij} \odot d_j)$
- iterate until convergence to get all paths (number of iterations bounded by $|V|$ if there are no cycles in G)

Hypergraphs as tensors

A hypergraph $H = (V, E)$ is a set of vertices V and edges $E \subset V \times V \times V \times \ldots$. In general, a hypergraph edge is a set of any number of vertices. Lets consider a hypergraph where all edges have three vertices,

- $V = \{v_1, v_2, v_3, v_4\}$
- $E = \{(v_1, v_2, v_4), (v_2, v_3, v_4), (v_1, v_3, v_4)\}$

We can represent hypergraph H with a 3-dimensional tensor $\mathcal{T}$ with entries $T_{ijk} \in \mathcal{T}$.

- since the hypergraph is undirected the tensor is symmetric, $T_{ijk} = T_{kij} = T_{jki}$
- the unique non-zero entries in $\mathcal{T}$ are T_{124}, T_{234}, and T_{134}

Coupled Cluster as a hypergraph computation

We can represent the Coupled Cluster method as a directed hypergraph $G = (K, L)$

- $K = P \cup O$ with P being a set of electrons and O being a set of orbitals the electrons may occupy $|O| \geq |P|$
- the edges can be represented as two tensor sets $L = \mathbf{F} \cup \mathcal{V}$ where $\mathbf{F} \in K \times K$ are matrices (one-electron integrals) and $\mathcal{V} \in K \times K \times K \times K$ are 4D tensors (two-electron integrals)
- we would like to compute the 'amplitudes' $\mathcal{T}$ of dimension $2n$ corresponding to the energy contributions of the excitation of n particles to n orbitals
- $\mathcal{T}$ is computed via Jacobi iteration (letting $\mathbf{D}$ be the diagonal of $\mathbf{F}$),

$$\mathcal{T}^{\text{new}} = \mathbf{D}^{-1}\left((\mathbf{F} - \mathbf{D} + \mathcal{V})(1 + \mathcal{T} + \frac{1}{2}\mathcal{T}^2 + \frac{1}{6}\mathcal{T}^3 + \frac{1}{24}\mathcal{T}^4)\right)$$

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?
- blocked or cyclic or block-cyclic? or ...?

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?
- blocked or cyclic or block-cyclic? or ...?
- which dimensions of the tensor/array to decompose?

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?
- blocked or cyclic or block-cyclic? or ...?
- which dimensions of the tensor/array to decompose?
- what is the best order for the local stride of the dimensions of the tensor/array?

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?
- blocked or cyclic or block-cyclic? or ...?
- which dimensions of the tensor/array to decompose?
- what is the best order for the local stride of the dimensions of the tensor/array?
- how to preserve symmetry in the decomposition?

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?
- blocked or cyclic or block-cyclic? or ...?
- which dimensions of the tensor/array to decompose?
- what is the best order for the local stride of the dimensions of the tensor/array?
- how to preserve symmetry in the decomposition?
- is it possible to align to the physical network topology?

Motivation for distributed array mapping

Data should drive the computation, the computation should not drive the data!

In a distributed-memory setting, data should be decomposed so as to minimize communication cost of the forthcoming algorithm

- replicated? to what extent?
- blocked or cyclic or block-cyclic? or ...?
- which dimensions of the tensor/array to decompose?
- what is the best order for the local stride of the dimensions of the tensor/array?
- how to preserve symmetry in the decomposition?
- is it possible to align to the physical network topology?
- answers to these questions depend on problem size, structure of the data, network topology, memory capacity, and theoretical analysis (performance models) of the forthcoming algorithm.

Physical network topology

Assume the network is a torus (mesh), which is a tensor of processors

- e.g. BG/P network is 3D torus, BG/Q network is 5D torus (multiple processes per node can be treated as a 6D torus)

Physical network topology

Assume the network is a torus (mesh), which is a tensor of processors

- e.g. BG/P network is 3D torus, BG/Q network is 5D torus (multiple processes per node can be treated as a 6D torus)
- if the network is not a torus, treat it as one anyway by factoring the number of processes

Physical network topology

Assume the network is a torus (mesh), which is a tensor of processors

- e.g. BG/P network is 3D torus, BG/Q network is 5D torus (multiple processes per node can be treated as a 6D torus)
- if the network is not a torus, treat it as one anyway by factoring the number of processes
    - still exploits locality, since not all processes talk to each other in a torus physical topology

Physical network topology

Assume the network is a torus (mesh), which is a tensor of processors

- e.g. BG/P network is 3D torus, BG/Q network is 5D torus (multiple processes per node can be treated as a 6D torus)
- if the network is not a torus, treat it as one anyway by factoring the number of processes
    - still exploits locality, since not all processes talk to each other in a torus physical topology
- consider all foldings of the physical torus topology (all unique embeddings of a 5D torus into a 4D torus, and so on to 1D)

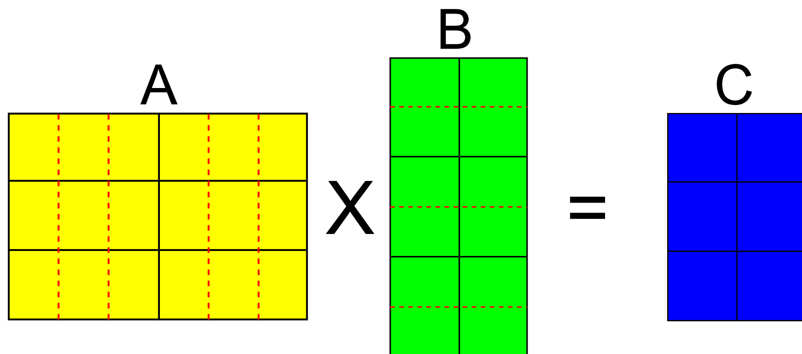
Virtual topology

Need a layer of abstraction between the domain decomposition and the physical network!

- map a tensor with edge lengths $(n_1, n_2, \ldots)$ tensor to a $(p_1, p_2, \ldots)$ via a $(v_1, v_2, \ldots)$ virtual topology, such that
  - $v_i = 0 \mod p_i$ for (enforce load balance)
  - $v_i = v_j$ if tensor dimensions i and j are symmetric (preserve symmetry)
  - typically want to maximize block size, $\prod_i n_i/v_i$
- virtual processes may be seen as blocks and vice-versa in this context
- motivated by the Charm++ programming model, albeit with SPMD control-flow

Virtualization example

Matrix multiply on 2x3 processor grid. Red lines represent virtualized part of processor grid. Elements assigned to blocks by cyclic phase.



The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat

The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)

The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
3. calculate the necessary memory usage and communication cost of the algorithm

The mapping process

Do in parallel over all physical topologies (foldings of the original torus)

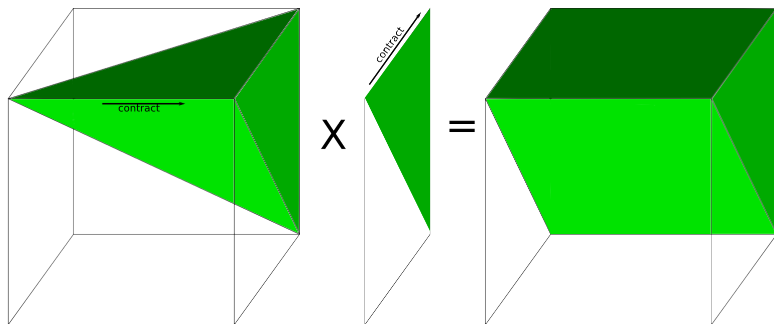
1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
3. calculate the necessary memory usage and communication cost of the algorithm
4. consider whether and what type of redistribution is necessary for the mapping

The mapping process

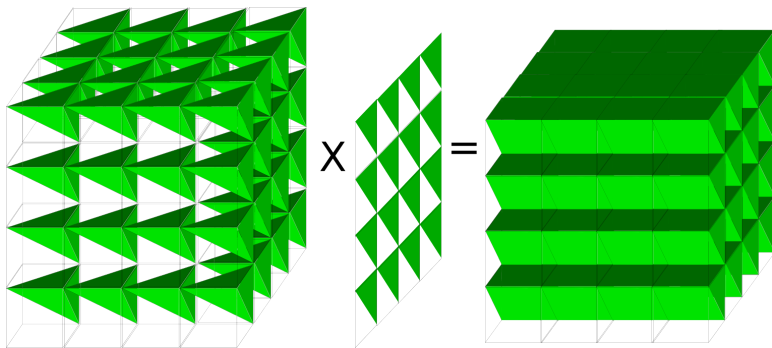
Do in parallel over all physical topologies (foldings of the original torus)

1. map longest physical torus dimension to longest tensor dimension and repeat
2. select virtualization factors to preserve symmetry (as well as to match the algorithmic requirements)
3. calculate the necessary memory usage and communication cost of the algorithm
4. consider whether and what type of redistribution is necessary for the mapping
5. select the best mapping based on a performance model

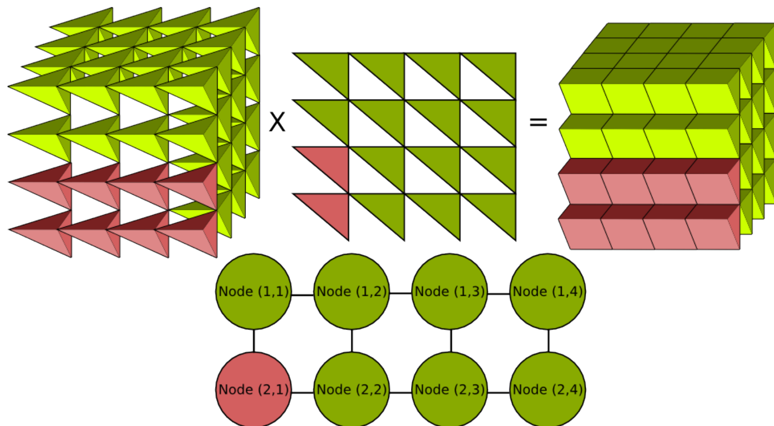
3D tensor mapping



3D tensor mapping



3D tensor mapping



Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method

Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method
- in Density Functional theory each iteration multiplies matrices, runs an orthogonalization process, performs a symmetric eigensolve, and does an FFT

Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method
- in Density Functional theory each iteration multiplies matrices, runs an orthogonalization process, performs a symmetric eigensolve, and does an FFT
  - each of these stages may want a different mapping of the matrix

Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method
- in Density Functional theory each iteration multiplies matrices, runs an orthogonalization process, performs a symmetric eigensolve, and does an FFT
    - each of these stages may want a different mapping of the matrix
- in Coupled Cluster each iteration performs 40-1000 contractions depending on the method

Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method
- in Density Functional theory each iteration multiplies matrices, runs an orthogonalization process, performs a symmetric eigensolve, and does an FFT
  - each of these stages may want a different mapping of the matrix
- in Coupled Cluster each iteration performs 40-1000 contractions depending on the method
  - each contraction wants alignment among different tensor dimensions

Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method
- in Density Functional theory each iteration multiplies matrices, runs an orthogonalization process, performs a symmetric eigensolve, and does an FFT
  - each of these stages may want a different mapping of the matrix
- in Coupled Cluster each iteration performs 40-1000 contractions depending on the method
  - each contraction wants alignment among different tensor dimensions
  - in many cases it is preferable to replicate one or more of the tensors

Why do we need redistributions?

Numerical applications typically have multiple algorithmic stages and die in the glue

- in molecular dynamics, particle-particle interactions and force-particle integration may require different mappings, further long-range force calculations may require different layouts depending on the method
- in Density Functional theory each iteration multiplies matrices, runs an orthogonalization process, performs a symmetric eigensolve, and does an FFT
  - each of these stages may want a different mapping of the matrix
- in Coupled Cluster each iteration performs 40-1000 contractions depending on the method
  - each contraction wants alignment among different tensor dimensions
  - in many cases it is preferable to replicate one or more of the tensors
  - do not want to keep tensors replicated, due to memory overhead

Sparse tensor reads and writes (closest CTF comes to a PGAS model)

In CTF, tensors are defined on a communicator (subset or full set of processors)

- the data pointer is hidden from the user
- the user can perform block-synchronous bulk writes and reads of index-value pairs
- to avoid communication, the user may read the current local pairs
- it is possible to perform overlapped writes (accumulate)
- CTF internal implementation (all parts threaded):
  1. bin keys by processor and redistribute
  2. bin key by virtual processor and then sort them
  3. iterate over the dense tensor, reading or writing keys along the way
  4. return keys to originating location if its a sparse read

Tensor slice and permuted slice

CTF makes it possible to extract sub-tensors of a distributed tensor, into a new distributed tensor

- slice() extracts all values corresponding a contiguous subset of indices along each dimension

Tensor slice and permuted slice

CTF makes it possible to extract sub-tensors of a distributed tensor, into a new distributed tensor

- slice() extracts all values corresponding a contiguous subset of indices along each dimension
- permute() extracts all values corresponding an arbitrary subset of indices along each dimension

Tensor slice and permuted slice

CTF makes it possible to extract sub-tensors of a distributed tensor, into a new distributed tensor

- slice() extracts all values corresponding a contiguous subset of indices along each dimension
- permute() extracts all values corresponding an arbitrary subset of indices along each dimension
- the target and destination tensors can live on different MPI communicators (CTF Worlds)

Tensor slice and permuted slice

CTF makes it possible to extract sub-tensors of a distributed tensor, into a new distributed tensor

- slice() extracts all values corresponding a contiguous subset of indices along each dimension
- permute() extracts all values corresponding an arbitrary subset of indices along each dimension
- the target and destination tensors can live on different MPI communicators (CTF Worlds)
- CTF does not make it possible to create 'views' and operate in-place on sub-tensors

Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs

Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts

Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
- iterate over local piece of old tensor in global order and place keys into bins

Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
- iterate over local piece of old tensor in global order and place keys into bins
- MPI all-to-all-v

Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
- iterate over local piece of old tensor in global order and place keys into bins
- MPI all-to-all-v
- iterate over local piece of new tensor in global order and retrieve keys from bins

Redistribution amongst different mappings

CTF must migrate tensors between different mappings between operations as well as for slice()

- the key idea is to use the fact that the global ordering of the tensor values is preserved to avoid formation of explicit key-value pairs
- iterate over local sub-tensor in linear order and bin keys to determine counts
- iterate over local piece of old tensor in global order and place keys into bins
- MPI all-to-all-v
- iterate over local piece of new tensor in global order and retrieve keys from bins
- kernel is threaded according to a global tensor partitioning

Distributed transposition of a tensor on a virtual processor grid

In some cases, it is necessary to change the assignment of the tensor dimensions to virtual grid dimensions without changing the virtual processor grid itself

- in this case, CTF does not touch data within each block
- redistributed by block instead
- use MPI Isend and MPI Irecv for each sent and received block

Local transposition

Once the data is redistributed into the new mapping, we reorder it locally within blocks

- turns all non-symmetric block contractions into matrix multiplication
- 'preserved' symmetries may be folded into one dimension, but broken ones cannot
- maps dimensions which have symmetry that cannot be folded into matrix multiplication to have the longest stride
- the contraction execution logic becomes
  1. nested distributed SUMMA (matrix multiplication)
  2. nested call to iterate over virtual blocks
  3. nested call to iterate over broken symmetric dimensions
  4. nested call to DGEMM
- claim: algorithms other than contractions may be efficiently expressed in similar nested form

CTF Coupled Cluster performance

CTF CCSD (obtains amplitudes for two-electron excitations) is faster than NWChem and scales to 8K nodes on BG/Q (see paper)

Much progress made this semester on CCSDT (three-electron excitations)

- CCSDT requires 100s of contractions per iteration (more if there is additional symmetric structure in the molecular system)
- Many of these contractions are much smaller than others (tensors of dimensions 2, 4, and 6 are contracted)
- CTF achieves super-linear weak scaling and modest strong scaling
- On 2048 nodes of BG/Q 29 Teraflop/s achieved for 8-water molecules, 192 orbitals
- Important direction forward is to parallelize across multiple small contractions
    1. CTF facilitates this by making inter-communicator tensor algebra possible
    2. the slice() call is also important here, especially when there is block-structure in the Hamiltonian of the physical system

Summary, conclusions, and future work

Cyclops Tensor Framework (CTF)

- `ctf.cs.berkeley.edu`, BSD license, try it, use it
- MPI+OpenMP+BLAS+nothing-else-necessary+keep-your-compiler-its-just-a-library
- Tested on gcc/intel/xlc, Mira/Carver/Hopper/Edison... even Apple
- High performance algebra for your multidimensional symmetric arrays
- In its essence, CTF is a library for mapping and communication orchestration of data via mathematical user-level language (operators)

Backup slides