A parallel library for multidimensional array computations with runtime tuning

Edgar Solomonik

Department of Computer Science
ETH Zurich

Charm++ Workshop, University of Illinois, Urbana-Champaign

April 19, 2016
At the crossroads of parallel libraries and languages

Parallel programming frameworks with multidimensional arrays

- HTA, Charm++ (MSA), Global Arrays, UPC, Titanium, HPF...
- flexible layouts, efficient data transformations and scheduling
- limited API, basic primitives
- rudimentary support for higher dimensional arrays (tensors)

Parallel libraries for matrix computations and numerical methods

- ScaLAPACK, Elemental, DPLASMA, PETSC, Trillinos, ...
- restrictive data distribution assumptions, variable performance
- broad API, powerful functionality
- no support for multidimensional matrices (higher order tensors)

Goal: efficient parallel library with a concise and yet general tensor API

---

Cyclops Tensor Framework

https://github.com/solomonik/ctf
Tensor contractions

For some $s, t, v \geq 0$, a tensor contraction of tensors $A$ and $B$ is

$$C_{\vec{i}\vec{j}} = \sum_{\vec{k}} A_{\vec{i}\vec{k}} \cdot B_{\vec{k}\vec{j}}, \quad \text{alternatively written}, \quad C_{\vec{i}\vec{j}} = \sum_{\vec{k}} A_{\vec{i}\vec{k}} \cdot B_{\vec{k}\vec{j}},$$

where $\vec{i} = \{i_1, \ldots, i_s\}$, $\vec{j} = \{j_1, \ldots, j_t\}$, and $\vec{k} = \{k_1, \ldots, k_v\}$.

Matrix/vector examples:
- $(s, t, v) = (0, 0, 1)$ vector inner product
- $(s, t, v) = (1, 0, 1)$ matrix-vector multiplication
- $(s, t, v) = (1, 1, 0)$ vector outer product
- $(s, t, v) = (1, 1, 1)$ matrix-matrix multiplication
- $(s, t, v) = (s, 1, 1)$ tensor-times-matrix
Applications of higher-order tensor contractions

Some applications of contractions of tensors of order at least three:

- tensor factorization algorithms, e.g. alternating least squares
- high-order numerical solvers for differential equations
- computer vision, graphics, image/video analysis
- deep learning convolutional neural networks
- density matrix renormalization group (DMRG)
- **electronic structure calculations**, e.g. coupled cluster
Coupled cluster methods

Coupled cluster provides a systematically improvable approximation to the manybody time-independent Schrödinger equation

- considers transitions of ‘doubles’, ‘triples’, or ‘quadruples’ of electrons to 2, 3, 4 unoccupied orbitals (CCSD, CCSDT, CCSDTQ)
- encodes their contribution to energy via order 4, 6, 8 tensor $T$
- iteratively solves for $T$, consistent with the mean field Hamiltonian with components $F, V$

$$0 = V_{ij}^{ab} + P_{b}^{a} \sum_{e} T_{ij}^{ae} F_{e}^{b} - \frac{1}{2} P_{j}^{i} \sum_{mnef} T_{im}^{ab} V_{ef}^{mn} T_{jn}^{ef} + \ldots$$

where $P_{y}^{x} f(x,y) := f(x,y) - f(y,x)$

- CCSD has 45 such multilinear terms, CCSDT 100s, CCSDTQ 1000s
Contraction in Coupled Cluster (CCSD method)

\[ W_{ei}^{mn} = V_{ei}^{mn} + \sum_{f} V_{ef}^{mn} t_{i}^{f}, \]

\[ X_{ij}^{mn} = V_{ij}^{mn} + P_{j}^{i} \sum_{e} V_{ie}^{mn} t_{j}^{e} + \frac{1}{2} \sum_{ef} V_{ef}^{mn} T_{ij}^{ef}, \]

\[ U_{ie}^{am} = V_{ie}^{am} - \sum_{n} W_{ei}^{mn} t_{n}^{a} + \sum_{f} V_{ef}^{ma} t_{i}^{f} + \frac{1}{2} \sum_{nf} V_{ef}^{mn} T_{in}^{af}, \]

\[ Q_{ij}^{am} = V_{ij}^{am} + P_{j}^{i} \sum_{e} V_{ie}^{am} t_{j}^{e} + \frac{1}{2} \sum_{ef} V_{ef}^{am} T_{ij}^{ef}, \]

\[ Z_{i}^{a} = f_{i}^{a} - \sum_{m} F_{i}^{m} t_{m}^{a} + \sum_{e} f_{e}^{a} t_{i}^{e} + \sum_{em} V_{ei}^{ma} t_{m}^{e} + \sum_{em} V_{im}^{ae} F_{e}^{m} + \frac{1}{2} \sum_{efm} V_{ef}^{am} T_{im}^{ef}, \]

\[ -\frac{1}{2} \sum_{emn} W_{ei}^{mn} T_{mn}^{ea}, \]

\[ Z_{ij}^{ab} = V_{ij}^{ab} + P_{j}^{i} \sum_{e} V_{ie}^{ab} t_{j}^{e} + P_{b}^{a} P_{j}^{i} \sum_{me} U_{ie}^{am} T_{mj}^{eb} - P_{b}^{a} \sum_{m} Q_{ij}^{am} t_{m}^{b} \]

\[ + P_{b}^{a} \sum_{e} F_{e}^{a} T_{ij}^{eb} - P_{j}^{i} \sum_{m} F_{i}^{m} T_{mj}^{ab} + \frac{1}{2} \sum_{ef} V_{ef}^{ab} T_{ij}^{ef} + \frac{1}{2} \sum_{mn} X_{ij}^{mn} T_{mn}^{ab}, \]
A library for tensor computations

Cyclops Tensor Framework (MPI+OpenMP+CUDA)
- implicit for loops based on index notation (Einstein summation)
- matrix sums, multiplication, Hadamard product (tensor contractions)
- distributed symmetric-packed/sparse storage via cyclic layout
A library for tensor computations

Cyclops Tensor Framework (MPI+OpenMP+CUDA)
- implicit for loops based on index notation (Einstein summation)
- matrix sums, multiplication, Hadamard product (tensor contractions)
- distributed symmetric-packed/sparse storage via cyclic layout

Jacobi iteration (solves $Ax = b$ iteratively) example code snippet

```cpp
Vector<> Jacobi(Matrix<> A, Vector<> b, int n){

    // split A = R + diag(1./d)
    do {
        // parallel for i = 1 to n
        // parallel for j = 1 to n
        x["i"] = d["i"]*(b["i"]-R["ij"]*x["j"]);
        // parallel for i = 1 to n
        // parallel for j = 1 to n
        r["i"] = b["i"]-A["ij"]*x["j"]; // compute residual
    } while (r.norm2() > 1.E-6); // check for convergence

    return x;
}
```
A library for tensor computations

Cyclops Tensor Framework (MPI+OpenMP+CUDA)

- implicit for loops based on index notation (Einstein summation)
- matrix sums, multiplication, Hadamard product (tensor contractions)
- distributed symmetric-packed/sparse storage via cyclic layout

Jacobi iteration (solves $Ax = b$ iteratively) example code snippet

```cpp
Vector<> Jacobi(Matrix<> A, Vector<> b, int n) {
    Matrix<> R(A);
    R["ii"] = 0.0;
    Vector<> x(n), d(n), r(n);
    Function<> inv([[double & d]]{ return 1./d; });
    d["i"] = inv(A["ii"]); // set d to inverse of diagonal of A
    do {
        x["i"] = d["i"]*(b["i"] - R["ij"]*x["j"]);;
        r["i"] = b["i"] - A["ij"]*x["j"]; // compute residual
    } while (r.norm2() > 1.E-6); // check for convergence
    return x;
}
```
Coupled cluster using CTF

Extracted from Aquarius (Devin Matthews’ code, https://github.com/devinamatthews/aquarius)

\[
\begin{align*}
FMI["mi"] & \quad + = \ 0.5 \times WMNEF["mnef"] \times T2["efin"]; \\
WMNIJ["mnij"] & \quad + = \ 0.5 \times WMNEF["mnef"] \times T2["efij"]; \\
FAE["ae"] & \quad - = \ 0.5 \times WMNEF["mnef"] \times T2["afmn"]; \\
WAMEI["amei"] & \quad - = \ 0.5 \times WMNEF["mnef"] \times T2["afin"]; \\
\end{align*}
\]

\[
\begin{align*}
Z2["abij"] & = \ WMNEF["ijab"]; \\
Z2["abij"] & + = \ FAE["af"] \times T2["fbij"]; \\
Z2["abij"] & - = \ FMI["ni"] \times T2["abnj"]; \\
Z2["abij"] & + = \ 0.5 \times WABEF["abef"] \times T2["efij"]; \\
Z2["abij"] & + = \ 0.5 \times WMNIJ["mnij"] \times T2["abmn"]; \\
Z2["abij"] & - = \ WAMEI["amei"] \times T2["ebmj"]; \\
\end{align*}
\]

CTF is used within **Aquarius, QChem, VASP, and Psi4**
Access and write tensor data

CTF abstracts that data distribution from the user
- Access arbitrary sparse subsets of the tensor by global index (coordinate format)
  - T.write(int * indices, double * data) (can also accumulate)
  - T.read(int * indices, double * data) (can also accumulate)
- Matlab submatrix notation: \( A[j : k, l : m] \)
  - T.slice(int * offsets, int * ends) returns the subtensor
  - T.slice(int corner_off, int corner_end) does the same
  - can also sum subtensors
  - different subworlds can read different subtensors simultaneously
- Extract a subtensor of any permutation of the tensor
  - given mappings \( P, Q \), does \( B[i, j] = A[P[i], Q[j]] \) via permute()
  - \( P \) and \( Q \) may access only subsets of \( A \) (if \( B \) is smaller)
  - \( B \) may be defined on subworlds on the world on which \( A \) is defined and each subworld may specify different \( P \) and \( Q \)
Symmetric matrix representation

Symmetric matrix

Unique part of symmetric matrix
Blocked distributions of a symmetric matrix

Naive blocked layout

Block-cyclic layout
Cyclic distribution of a symmetric matrix

Cyclic layout ~ Improved blocked layout

Cyclops Tensor Framework
https://github.com/solomonik/ctf
3D tensor mapping

![Diagram of 3D tensor mapping with nodes and connections](https://github.com/solomonik/ctf)
Tensor decomposition and mapping

CTF tensor decomposition

- cyclic layout used to preserve packed symmetric structure
- overdecomposition (virtualization) employed to decouple the decomposition from the physical processor grid

CTF mapping logic

- arrange physical topology into all possible processor grids
  - assumes a torus topology
  - by default, factorize number of processors to produce torus
- dynamically (in parallel) autotune over all topologies and mappings
  - topologies: all foldings of physical (default) topology
  - mappings: assignments of tensor modes (indices) to torus modes
- select best mapping based on model-based performance prediction
  - performance models for communication, memory-bandwidth, synchronization, and work for contractions and redistributions
  - for sparse tensors, estimate load imbalance

Cyclops Tensor Framework
https://github.com/solomonik/ctf
Algorithms for tensor redistribution

The following three redistribution kernels are provided by CTF

- **Sparse (key-value) redistribution**
  - requires all-to-all-v communication and local binning/sorting
  - well-fit for user-level data entry and sparse tensors

- **Dense mapping-to-mapping (no-explicit-key) redistribution**
  - send/recv counts precomputed analytically via (complicated) recurrence
  - subset of messages packed from data at a time
  - optimized via template-instantiated nested loops, threading

- **Block-to-block redistribution**
  - reassignment of block (virtual) decomposition to processors
  - processors send blocks via point-to-point messages
Benefit of topology aware mapping

CTF can leverage rectangular collectives on BG/Q

The QBall (QBox) density functional theory (DFT) code has leveraged CTF just to accelerate matrix multiplication
Comparison with NWChem

NWChem is the most commonly-used distributed-memory quantum chemistry method suite

- provides Coupled Cluster methods: CCSD and CCSDT
- derives equations via Tensor Contraction Engine (TCE)
- generates contractions as blocked loops leveraging Global Arrays

![Strong scaling CCSD on Edison](image1.png)

Strong scaling CCSDT on Edison

![Strong scaling CCSDT on Edison](image2.png)

Cyclops Tensor Framework [https://github.com/solomonik/ctf](https://github.com/solomonik/ctf)
Coupled cluster on IBM BlueGene/Q and Cray XC30

CCSD up to 55 (50) water molecules with cc-pVDZ
CCSDT up to 10 water molecules with cc-pVDZ

Weak scaling on BlueGene/Q

Weak scaling on Edison

Cyclops Tensor Framework

https://github.com/solomonik/ctf
Performance breakdown on BG/Q

Performance data for a CCSD iteration with 200 electrons and 1000 orbitals on 4096 nodes of Mira 
4 processes per node, 16 threads per process 
Total time: 18 mins 

\( n_v \)-orbitals, \( n_o \)-electrons, \( p \)-processors, \( M \)-local memory size

<table>
<thead>
<tr>
<th>kernel</th>
<th>% of time</th>
<th>complexity</th>
<th>architectural bounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>dgemm</td>
<td>45%</td>
<td>( O(n_v^4 n_o^2 / p) )</td>
<td>flops/mem bandwidth</td>
</tr>
<tr>
<td>broadcasts</td>
<td>20%</td>
<td>( O(n_v^4 n_o^2 / p \sqrt{M}) )</td>
<td>multicast bandwidth</td>
</tr>
<tr>
<td>prefix sum</td>
<td>10%</td>
<td>( O(p) )</td>
<td>allreduce bandwidth</td>
</tr>
<tr>
<td>data packing</td>
<td>7%</td>
<td>( O(n_v^2 n_o^2 / p) )</td>
<td>integer ops</td>
</tr>
<tr>
<td>all-to-all-v</td>
<td>7%</td>
<td>( O(n_v^2 n_o^2 / p) )</td>
<td>bisection bandwidth</td>
</tr>
<tr>
<td>tensor folding</td>
<td>4%</td>
<td>( O(n_v^2 n_o^2 / p) )</td>
<td>memory bandwidth</td>
</tr>
</tbody>
</table>

Cyclops Tensor Framework [https://github.com/solomonik/ctf](https://github.com/solomonik/ctf)
CTF allows a tensor elements to have a user-defined algebraic structure:

<table>
<thead>
<tr>
<th>algebraic str.</th>
<th>add.</th>
<th>add. id.</th>
<th>add. inv.</th>
<th>mul.</th>
<th>mul. id.</th>
</tr>
</thead>
<tbody>
<tr>
<td>set</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>monoid</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>group</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>semiring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>ring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

For each summation or contraction, the user may define custom functions, so we also have

semigroup = set + function
rng = group + function

user-defined types must be static-sized (migratable)
Algebraic Structure Interface

Tropical algebraic semiring

\[ c \oplus a \otimes b := \min(c, a + b) \]

Interface via C++11 Lambdas:

```
Semiring<int> ts(INT_MAX/2, //additive identity
    [](int a, int b){ return min(a,b); }, //add
    MPI_MIN, //add MPI op
    0, //multiplicative identity
    [](int a, int b){ return a+b; }); //mul
```

Adjacency matrix for \( n \)-node undirected graph with integer weights:

```
Matrix<int> A(n,n,SP|SH,ts);
```

SP – sparse, SH – symmetric-hollow (zero diagonal)
Bellman–Ford Algorithm using CTF

CTF code for $n$ node single-source shortest-paths (SSSP) calculation:

```cpp
World w(MPI_COMM_WORLD);
Semiring<int> s(INT_MAX/2,
    [](int a, int b){ return min(a,b); },
    MPI_MIN,
    0,
    [](int a, int b){ return a+b; });

Matrix<int> A(n,n,SP,w,s); // Adjacency matrix
Vector<int> v(n,w,s); // Distances from starting vertex

... // Initialize A and v

//Bellman–Ford SSSP algorithm
for (int t=0; t<n; t++){
    v["i"] += v["j"]*A["ji"];
}
```
Betweenness centrality code snippet, for $k$ of $n$ nodes

```cpp
void btwn_central(Matrix<int> A, Matrix<path> P, int n, int k) {
    Monoid<path> mon(...,
        [](path a, path b) {
            if (a.w < b.w) return a;
            else if (b.w < a.w) return b;
            else return path(a.w, a.m+b.m);
        }, ...);

    Matrix<path> Q(n, k, mon); // shortest path matrix
    Q["ij"] = P["ij"];

    Function<int,path> append([(int w, path p) {
        return path(w+p.w, p.m);
    }];

    for (int i=0; i<n; i++)
        Q["ij"] = append(A["ik"], Q["kj"]) ;
    ...
}
```

Cyclops Tensor Framework https://github.com/solomonik/ctf
Performance of CTF for sparse computations

multiplication of a sparse matrix and a dense matrix

all-pairs shortest-paths based on path doubling with sparsification
Conclusion

Summary of CTF:
- distributed-memory tensor library leveraging MPI, OpenMP, and CUDA
- support for symmetric and sparse tensors
- similarities to Charm++ multidimensional chare arrays
  - overdecomposition (virtualization)
  - topology-aware mapping of tasks(blocks)
  - application-driven development
- differences from Charm++
  - ‘static’ schedule selected at runtime vs dynamic load balancing
  - migration between regular distributions rather than greedy/refined
  - redistributions done processor-to-processor

Future work:
- generation of fast local ‘dgemm’ kernels for user-defined functions
- handling of contractions of two sparse tensors and sparse output
- multi-contraction scheduling, convolutions, tensor factorizations, ...
Backup slides
Matrix multiplication factorization strong scaling on Mira (BG/Q), n=65,536

- 2D MM, custom mapping
- 2D MM, default mapping

Cyclops Tensor Framework
https://github.com/solomonik/ctf
Algebraic shortest path computations

All-pairs shortest-paths (APSP):

- distance matrix is the closure of $A$,

$$A^* = I \oplus A \oplus A^2 \oplus \ldots A^n$$

- Floyd–Warshall = Gauss–Jordan elimination \(\approx\) Gaussian elimination
  - $O(n^3)$ cost, but contains length $n \log n$ dependency path

- path doubling: $\log n$ steps, $O(n^3 \log n)$ cost:

$$B = I \oplus A, \quad B^{2^k} = B^k \otimes B^k, \quad B^n = A^*$$

- sparse path doubling$^1$:
  - let $C$ be subset of $B^k$ corresponding to paths containing exactly $k$ edges,

$$B^{2^k} = B^k \oplus (C \otimes B^k)$$

- $O(n^3)$ cost, dependency paths length $O(\log^2 n)$

---

$^1$ Tiskin, Springer LNCS, 2001
Coupled cluster methods

Coupled cluster provides a systematically improvable approximation to the many-body time-independent Schrödinger equation $H |\Psi\rangle = E |\Psi\rangle$

- the Hamiltonian has one- and two- electron components $H = F + V$
- Hartree-Fock (SCF) computes mean-field Hamiltonian: $F, V$
- Coupled-cluster methods (CCSD, CCSDT, CCSDTQ) consider transitions of (doubles, triples, and quadruples) of electrons to unoccupied orbitals, encoded by tensor operator, $T = T_1 + T_2 + T_3 + T_4$
- they use an exponential ansatz for the wavefunction, $\Psi = e^{T\phi}$ where $\phi$ is a Slater determinant
- expanding $0 = \langle \phi' | H |\Psi\rangle$ yields nonlinear equations for $\{T_i\}$ in $F, V$

$$0 = V_{ij}^{ab} + P(a, b) \sum_e T_{ij}^{ae} F_{e}^{b} - \frac{1}{2} P(i, j) \sum_{mnef} T_{im}^{ab} V_{ef}^{mn} T_{jn}^{ef} + \ldots$$

where $P$ is an antisymmetrization operator