


Towards an algebraic formalism for scalable numerical algorithms

Edgar Solomonik

Department of Computer Science
University of Illinois at Urbana-Champaign

Householder Symposium XX

June 22, 2017

 @CS@Illinois

Communication-synchronization wall

To analyze parallel algorithms, we consider costs along the **critical path** of the execution schedule¹

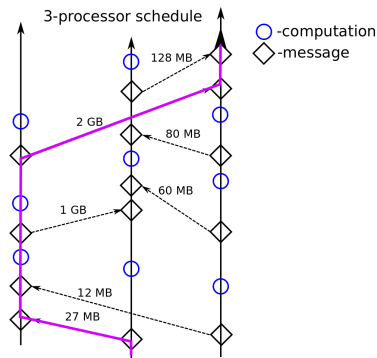
- F – computation cost
- W – horizontal communication cost
- S – synchronization cost

We can show a commonality between

- **Cholesky** of an $n \times n$ matrix and
- n steps of a **9-pt stencil**:

$$W \cdot S = \Omega(n^2)$$

regardless of #processors¹

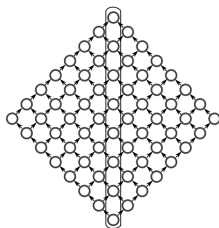


¹E.S., E. Carson, N. Knight, J. Demmel, TOPC 2016

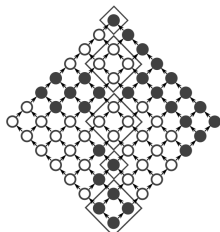
Tradeoffs in the diamond DAG

Computation vs synchronization tradeoff for the $n \times n$ diamond DAG,¹

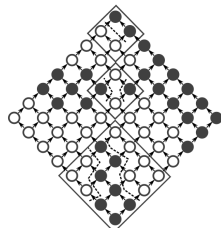
$$F \cdot S = \Omega(n^2)$$



Dependency chain P



Monochrome dependency intervals



Multicolored dependency intervals

In this DAG, vertices denote scalar computations in an algorithm

¹C.H. Papadimitriou, J.D. Ullman, SIAM JC, 1987

Scheduling tradeoffs of path-expander graphs

Definition ((ϵ, σ)-path-expander)

Graph $G = (V, E)$ is a (ϵ, σ)-**path-expander** if there exists a path $(u_1, \dots, u_n) \subset V$, such that the dependency interval $[u_i, u_{i+b}]_G$ for each i, b has size $\Theta(\sigma(b))$ and a minimum cut of size $\Omega(\epsilon(b))$.

Theorem (Path-expander communication lower bound)

*Any parallel schedule of an algorithm with a (ϵ, σ)-**path-expander** dependency graph about a path of length n and some $b \in [1, n]$ incurs computation (F), communication (W), and synchronization (S) costs:*

$$F = \Omega(\sigma(b) \cdot n/b), \quad W = \Omega(\epsilon(b) \cdot n/b), \quad S = \Omega(n/b).$$

Corollary

If $\sigma(b) = b^d$ and $\epsilon(b) = b^{d-1}$, the above theorem yields,

$$F \cdot S^{d-1} = \Omega(n^d), \quad W \cdot S^{d-2} = \Omega(n^{d-1}).$$

Synchronization-communication wall in iterative methods

The theorem can be applied to **sparse iterative methods** on regular grids. For computing s applications of a $(2m + 1)^d$ -point stencil,

$$F_{\text{St}} \cdot S_{\text{St}}^d = \Omega \left(m^{2d} \cdot s^{d+1} \right), \quad W_{\text{St}} \cdot S_{\text{St}}^{d-1} = \Omega \left(m^d \cdot s^d \right)$$

while s -step methods **reduce synchronization**, for large s they **require asymptotically more communication**.

The lower bound is attained by s -step methods when s approaches the dimension of each processor's local subgrid.

A more scalable algorithm for TRSM

For Cholesky factorization with p processors, parallel schedules can attain

$$F = O(n^3/p), \quad W = O(n^2/p^\delta), \quad S = O(p^\delta)$$

for any $\delta = [1/2, 2/3]$. Achieving similar costs for LU, QR, and the symmetric eigenvalue problem requires some [algorithmic tweaks](#).

triangular solve	square TRSM \checkmark^1	rectangular TRSM \checkmark^2
LU with pivoting	pairwise pivoting \checkmark^3	tournament pivoting \checkmark^4
QR factorization	Givens on square \checkmark^3	Householder on rect. \checkmark^5
SVD	singular values only \checkmark^5	singular vectors \times

\checkmark means costs attained (synchronization within polylogarithmic factors).

Ongoing work on [QR with column pivoting](#)

¹B. Lipshitz, MS thesis 2013

²T. Wicky, E.S., T. Hoefler, IPDPS 2017

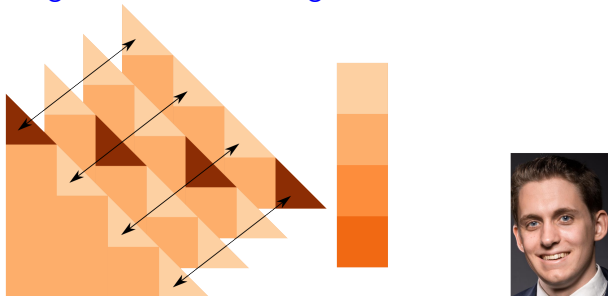
³A. Tiskin, FGCS 2007

⁴E.S., J. Demmel, EuroPar 2011

⁵E.S., G. Ballard, T. Hoefler, J. Demmel, SPAA 2017

New algorithms can circumvent lower bounds

For TRSM, we can achieve a lower synchronization/communication cost by performing **triangular inversion on diagonal blocks**



- **decreases synchronization cost** by $O(p^{2/3})$ on p processors with respect to known algorithms
- optimal communication for **any number of right-hand sides**
- MS thesis work by Tobias Wicky¹

¹T. Wicky, E.S., T. Hoefler, IPDPS 2017

Improving scalability for iterative methods

Randomized-projection methods have potential to significantly improve scalability over iterative Krylov subspace methods

- key idea: **replace sparse mat-vecs with sparse mat-muls**
- define $n \times (k + 10)$ Gaussian random matrix \mathbf{X}
- \mathbf{AX} gives a good representation of the kernel of \mathbf{A}
- accuracy can be improved exponentially with q^1

$$(\mathbf{AA}^T)^q \mathbf{AX}$$

- many related results with high potential for efficiency (e.g. randomized column pivoting for QR ²)

¹N. Halko, P.G. Martinsson, J.A. Tropp, SIAM Review 2011

²P.G. Martinsson, G. Quintana Orti, N. Heavner. R. van de Geijn, SIAM 2017

Need *algorithms and methods* that are *more parallelizable* rather than *parallel schedules* of existing algorithms.

How can we formally define an algorithm?

Formally defining a space of algorithms enables systematic exploration.

Definition (Bilinear algorithms (V. Pan, 1984))

A bilinear algorithm $\Lambda = (\mathbf{F}(\mathbf{A}), \mathbf{F}(\mathbf{B}), \mathbf{F}(\mathbf{C}))$ computes

$$\mathbf{c} = \mathbf{F}(\mathbf{C})[(\mathbf{F}(\mathbf{A})^T \mathbf{a}) \circ (\mathbf{F}(\mathbf{B})^T \mathbf{b})],$$

where \mathbf{a} and \mathbf{b} are inputs and \circ is the Hadamard (pointwise) product.

$$\begin{bmatrix} \mathbf{c} \end{bmatrix} = \begin{bmatrix} \begin{matrix} \times & \times & \times & \times & \times & \times \\ \times & \times & & \times & \times & \times \\ \times & & \times & & \times & \times \\ \times & \times & \times & & \times & \times \\ \times & & \times & \times & \times & \times \\ \times & \times & & \times & \times & \times \end{matrix} \end{bmatrix} \left[\left(\begin{bmatrix} \begin{matrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & & \times & & \times & \times \\ \times & \times & \times & & \times & \times \\ \times & & \times & \times & \times & \times \\ \times & \times & & \times & \times & \times \end{matrix} \end{bmatrix}^T \begin{bmatrix} \mathbf{a} \end{bmatrix} \right) \circ \left(\begin{bmatrix} \begin{matrix} \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \\ \times & \times & \times & & \times & \times \\ \times & & \times & \times & \times & \times \\ \times & \times & \times & \times & \times & \times \end{matrix} \end{bmatrix}^T \begin{bmatrix} \mathbf{b} \end{bmatrix} \right) \right]$$

Bilinear algorithms as tensor factorizations

A bilinear algorithm corresponds to a CP tensor decomposition

$$\begin{aligned}c_i &= \sum_{r=1}^R F_{ir}^{(C)} \left(\sum_j F_{jr}^{(A)} a_j \right) \left(\sum_k F_{kr}^{(B)} b_k \right) \\ &= \sum_j \sum_k \left(\sum_{r=1}^R F_{ir}^{(C)} F_{jr}^{(A)} F_{kr}^{(B)} \right) a_j b_k \\ &= \sum_j \sum_k T_{ijk} a_j b_k \quad \text{where} \quad T_{ijk} = \sum_{r=1}^R F_{ir}^{(C)} F_{jr}^{(A)} F_{kr}^{(B)}\end{aligned}$$

For multiplication of $n \times n$ matrices,

- T is $n^2 \times n^2 \times n^2$
- classical algorithm has rank $R = n^3$
- Strassen's algorithm has rank $R \approx n^{\log_2(7)}$

Expansion in bilinear algorithms

The **communication complexity** of a bilinear algorithm depends on the amount of data needed to compute subsets of the bilinear products.

Definition (Bilinear subalgorithm)

Given $\Lambda = (\mathbf{F}^{(A)}, \mathbf{F}^{(B)}, \mathbf{F}^{(C)})$, $\Lambda_{\text{sub}} \subseteq \Lambda$ if \exists projection matrix \mathbf{P} , so

$$\Lambda_{\text{sub}} = (\mathbf{F}^{(A)}\mathbf{P}, \mathbf{F}^{(B)}\mathbf{P}, \mathbf{F}^{(C)}\mathbf{P}).$$

The projection matrix extracts $\#\text{cols}(\mathbf{P})$ columns of each matrix.

Definition (Bilinear algorithm expansion)

A bilinear algorithm Λ has expansion bound $\mathcal{E}_{\Lambda} : \mathbb{N}^3 \rightarrow \mathbb{N}$, if for all

$$\Lambda_{\text{sub}} := (\mathbf{F}_{\text{sub}}^{(A)}, \mathbf{F}_{\text{sub}}^{(B)}, \mathbf{F}_{\text{sub}}^{(C)}) \subseteq \Lambda$$

we have $\text{rank}(\Lambda_{\text{sub}}) \leq \mathcal{E}_{\Lambda} \left(\text{rank}(\mathbf{F}_{\text{sub}}^{(A)}), \text{rank}(\mathbf{F}_{\text{sub}}^{(B)}), \text{rank}(\mathbf{F}_{\text{sub}}^{(C)}) \right)$

For matrix mult., Loomis-Whitney inequality $\rightarrow \mathcal{E}_{\text{MM}}(x, y, z) = \sqrt{xyz}$

Bilinear algorithms for symmetric tensor contractions

A tensor $\mathbf{T} \in \mathbb{R}^{n_1 \times \dots \times n_d}$ has

- order d (i.e. d modes / indices)
- dimensions n -by- \dots -by- n
- elements $\mathbf{T}_{i_1 \dots i_d} = \mathbf{T}_{\mathbf{i}}$ where $\mathbf{i} \in \{1, \dots, n\}^d$

We say a tensor is **symmetric** if for any $\mathbf{j}, \mathbf{k} \in \{1, \dots, n\}$

$$\mathbf{T}_{i_1 \dots i_j \dots i_k \dots i_d} = \mathbf{T}_{i_1 \dots i_k \dots i_j \dots i_d}$$

A tensor is **partially-symmetric** if such index interchanges are restricted to be within subsets of $\{1, \dots, n\}$, e.g.

$$\mathbf{T}_{kl}^{ij} = \mathbf{T}_{kl}^{ji} = \mathbf{T}_{lk}^{ji} = \mathbf{T}_{lk}^{ij}$$

For any $s, t, v \in \{0, 1, \dots\}$, a tensor contraction is

$$\forall \mathbf{i} \in \{1, \dots, n\}^s, \mathbf{j} \in \{1, \dots, n\}^t, \quad \mathbf{C}_{ij} = \sum_{\mathbf{k} \in \{1, \dots, n\}^v} \mathbf{A}_{ik} \mathbf{B}_{kj}$$

Symmetric matrix times vector

Lets consider the simplest tensor contraction with symmetry

- let \mathbf{A} be an n -by- n symmetric matrix ($\mathbf{A}_{ij} = \mathbf{A}_{ji}$)
- the symmetry is not preserved in matrix-vector multiplication

$$\mathbf{c} = \mathbf{A} \cdot \mathbf{b}$$

$$c_i = \sum_{j=1}^n \underbrace{\mathbf{A}_{ij} \cdot \mathbf{b}_j}_{\text{nonsymmetric}}$$

- generally n^2 additions and n^2 multiplications are performed
- we can perform only $\binom{n+1}{2}$ multiplications using¹

$$c_i = \sum_{j=1, j \neq i}^n \underbrace{\mathbf{A}_{ij} \cdot (\mathbf{b}_i + \mathbf{b}_j)}_{\text{symmetric}} + \underbrace{\left(\mathbf{A}_{ii} - \sum_{j=1, j \neq i}^n \mathbf{A}_{ij} \right) \cdot \mathbf{b}_i}_{\text{low-order}}$$

¹E.S., J. Demmel, 2015

Symmetrized outer product

Consider a rank-2 outer product of vectors \mathbf{a} and \mathbf{b} of length n into symmetric matrix \mathbf{C}

$$\mathbf{C} = \mathbf{a} \cdot \mathbf{b}^T + \mathbf{b} \cdot \mathbf{a}^T$$
$$C_{ij} = \underbrace{a_i \cdot b_j}_{\text{nonsymmetric}} + \underbrace{a_j \cdot b_i}_{\text{permutation}}$$

usually computed via the n^2 multiplications and n^2 additions
new algorithm requires $\binom{n+1}{2}$ multiplications

$$C_{ij} = \underbrace{(a_i + a_j) \cdot (b_i + b_j)}_{\text{symmetric}} - \underbrace{a_i \cdot b_i}_{w_i} - \underbrace{a_j \cdot b_j}_{w_j}$$

Symmetrized matrix multiplication

For symmetric matrices \mathbf{A} and \mathbf{B} , compute

$$\mathbf{C}_{ij} = \sum_{k=1}^n \left(\underbrace{\mathbf{A}_{ik} \cdot \mathbf{B}_{kj}}_{\text{nonsymmetric}} + \underbrace{\mathbf{A}_{jk} \cdot \mathbf{B}_{ki}}_{\text{permutation}} \right)$$

New algorithm requires $\binom{n+2}{3}$ multiplications rather than n^3

$$\begin{aligned} \mathbf{C}_{ij} = & \sum_{k \neq i, j} \underbrace{(\mathbf{A}_{ij} + \mathbf{A}_{ik} + \mathbf{A}_{jk}) \cdot (\mathbf{B}_{ij} + \mathbf{B}_{kj} + \mathbf{B}_{ki})}_{\mathbf{Z}_{ijk} - \text{symmetric}} - \underbrace{\sum_{k \neq i} \mathbf{A}_{ik} \cdot \mathbf{B}_{ik}}_{\mathbf{w}_i - \text{low-order}} - \underbrace{\sum_{k \neq j} \mathbf{A}_{jk} \cdot \mathbf{B}_{jk}}_{\mathbf{w}_j - \text{low-order}} \\ & + \underbrace{\frac{1}{n-2} \left((2-n)\mathbf{A}_{ij} - \mathbf{A}_i^{(1)} - \mathbf{A}_j^{(1)} \right) \cdot \left((n-2)\mathbf{B}_{ij} + \mathbf{B}_i^{(1)} + \mathbf{B}_j^{(1)} \right)}_{\mathbf{U}_{ij} - \text{low-order}} \\ & + \underbrace{\frac{1}{n-2} \left(\mathbf{A}_i^{(1)} + \mathbf{A}_j^{(1)} \right) \cdot \left(\mathbf{B}_i^{(1)} + \mathbf{B}_j^{(1)} \right)}_{\mathbf{V}_{ij} - \text{low-order}} \end{aligned}$$

where $\mathbf{A}_i^{(1)} = \left(\mathbf{A}_{ii} - \sum_{k \neq i}^n \mathbf{A}_{ki} \right)$ and $\mathbf{B}_i^{(1)} = \left(\mathbf{B}_{ii} - \sum_{k \neq i}^n \mathbf{B}_{ki} \right)$.

Symmetrized tensor contraction

Generally consider any **symmetric tensor contraction** for $s, t, v \in \{0, 1, \dots\}$

$$\forall \mathbf{i} \in \{1, \dots, n\}^s, \mathbf{j} \in \{1, \dots, n\}^t, \mathbf{C}_{ij} = \sum_{\mathbf{k} \in \{1, \dots, n\}^v} \mathbf{A}_{ik} \mathbf{B}_{kj} + \text{permutations}$$

- best previous algorithms used roughly $\binom{n}{s} \binom{n}{t} \binom{n}{v}$ multiplications, new algorithm requires roughly $\binom{n}{s+t+v}$ multiplications
- these are **bilinear algorithms** and correspond to a **CP decomposition of the symmetric contraction tensor that defines the problem**
- analysis of bilinear expansion gives us **communication lower bounds**
 - surprising **negative** result – when $s + t + v \geq 4$ and $s \neq t \neq v$ **asymptotically more communication** necessary for new algorithm!
- algorithm can be **nested** in the case of **partially-symmetric** contractions, leads to a **reduction in cost** – manifold cost improvements in some high-order quantum chemistry methods

Analysis of bilinear algorithms

There are a few very fundamental bilinear problems

- matrix multiplication
- symmetric tensor contractions
- **convolution**

The algebraic formulation enables systematic derivation and analysis

- direct proof of correctness
- CP decomposition can be computed numerically to find algorithms
- **numerical stability** easy to infer¹
- communication lower bounds via bilinear expansion²

Some problems are multilinear or correspond to chains of bilinear algorithms, can we provide **useful algebraic formulations** for these problems and the space of algorithms?

¹A.R. Benson, G. Ballard, ACM SIGPLAN 2015

²E.S., J. Demmel, T. Hoefler, 2015

A more complicated case: HSS matrices

Hierarchically-semi-separable (HSS) matrices have the structure



For example the 'prefix-sum' matrix

$$\begin{bmatrix} 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \vdots & & & \vdots \\ 1 & \cdots & 1 & 0 \end{bmatrix}$$

is HSS with rank 1

HSS matrices algebraically

There are a few different ways to think about HSS matrices

- geometrically (FMM) as trees, mat-vecs do up-sweep and down-sweep
- algebraically as a telescoping **block-sparse matrix** factorization¹

$$U^{(3)} (U^{(2)} (U^{(1)} B^{(0)} V^{(1)*} + B^{(1)}) V^{(2)*} + B^{(2)}) V^{(3)*} + D^{(3)}$$

- algebraically as a **dense tensor** factorization like

$$U_{ijk}^{(3)} (U_{jk}^{(2)} (U_k^{(1)} B_c^{(0)} V_c^{(1)}) + \delta_k^c B_c^{(1)}) V_{bc}^{(2)} + \delta_{jk}^{bc} B_{bc}^{(2)} V_{abc}^{(3)}$$

where summations are implicit (Einstein notation) and δ is an identity

- is this representation only a theoretical curiosity?

¹P.G. Martinsson, SIAM Journal on Matrix Analysis and Applications, 2011

A stand-alone library for petascale tensor computations

Cyclops Tensor Framework (CTF)¹

- distributed-memory symmetric/sparse tensors as C++ objects

```
Matrix<int> A(n, n, AS|SP, World(MPI_COMM_WORLD));  
Tensor<float> T(order, is_sparse, dims, syms, ring, world);  
T.read(...); T.write(...); T.slice(...); T.permute(...);
```

- parallel contraction/summation of tensors

```
Z["abij"] += V["ijab"];  
B["ai"] = A["aiai"];  
T["abij"] = T["abij"]*D["abij"];  
W["mnij"] += 0.5*W["mnef"]*T["efij"];  
Z["abij"] -= R["mnje"]*T3["abeimn"];  
M["ij"] += Function<>([](double x){ return 1/x; })(v["j"]);
```

- development (1500 commits) since 2011, open source since 2013



¹E.S., D. Matthews, J.R. Hammond, J. Demmel, JPDC 2014

Coupled cluster: an initial application driver

CCSD contractions from [Aquarius](#) (lead by [Devin Matthews](#))

<https://github.com/devinamatthews/aquarius>

```
FMI["mi"]      += 0.5*WMNEF["mnef"]*T2["efin"];
WMNIJ["nij"]   += 0.5*WMNEF["mnef"]*T2["efij"];
FAE["ae"]      -= 0.5*WMNEF["mnef"]*T2["afmn"];
WAMEI["amei"]  -= 0.5*WMNEF["mnef"]*T2["afin"];

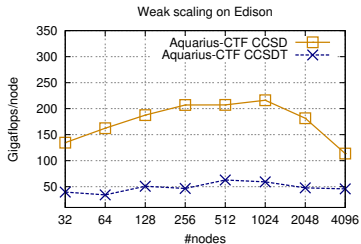
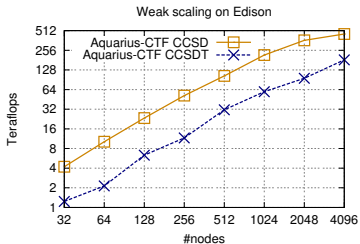
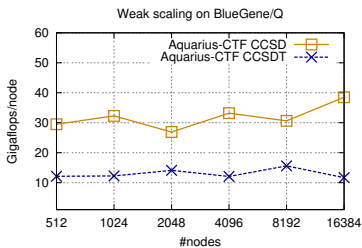
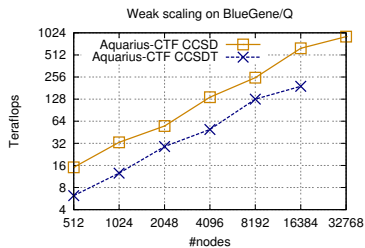
Z2["abij"]    = WMNEF["ijab"];
Z2["abij"]    += FAE["af"]*T2["fbij"];
Z2["abij"]    -= FMI["ni"]*T2["abnj"];
Z2["abij"]    += 0.5*WABEF["abef"]*T2["efij"];
Z2["abij"]    += 0.5*WMNIJ["nij"]*T2["abmn"];
Z2["abij"]    -= WAMEI["amei"]*T2["ebmj"];
```

CTF is used within **Aquarius**, **QChem**, **VASP**, and **Psi4**

Performance of CTF for coupled cluster

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

CCSDT up to 10 water molecules with cc-pVDZ



compares well to **NWChem** (up to 10x speed-ups for CCSDT)

```
Tensor<> Ea, Ei, Fab, Fij, Vabij, Vijab, Vabcd, Vijkl, Vaibj  
... // compute above 1-e an 2-e integrals
```

```
Tensor<> T(4, Vabij.lens, *Vabij.wrld);  
T["abij"] = Vabij["abij"];
```

```
divide_EaEi(Ea, Ei, T);
```

```
Tensor<> Z(4, Vabij.lens, *Vabij.wrld);  
Z["abij"] = Vijab["ijab"];  
Z["abij"] += Fab["af"]*T["fbij"];  
Z["abij"] -= Fij["ni"]*T["abnj"];  
Z["abij"] += 0.5*Vabcd["abef"]*T["efij"];  
Z["abij"] += 0.5*Vijkl["mnij"]*T["abmn"];  
Z["abij"] += Vaibj["amei"]*T["ebmj"];
```

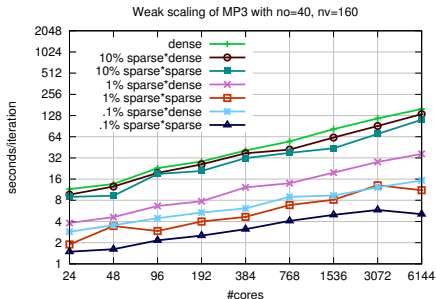
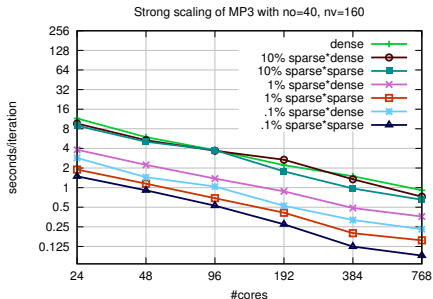
```
divide_EaEi(Ea, Ei, Z);
```

```
double MP3_energy = Z["abij"]*Vabij["abij"];
```


Sparse MP3 code

Strong and weak scaling of sparse MP3 code, with

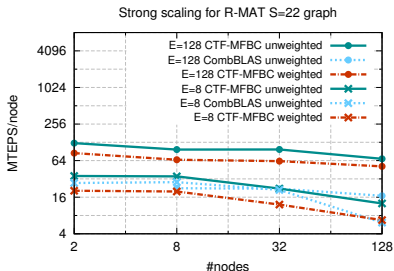
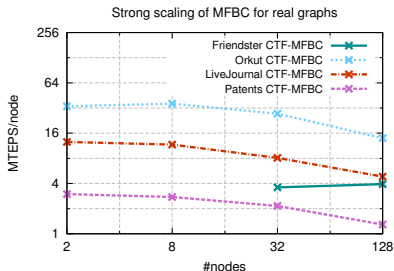
(1) dense V and T (2) sparse V and dense T (3) sparse V and T



CTF for betweenness centrality

Betweenness centrality is a measure of the importance of vertices in the shortest paths of a graph

- can be computed using **sparse matrix multiplication** (SpGEMM) with operations on special **monoids**
- CTF handles this in similar ways to CombBLAS



Friendster has 66 million vertices and 1.8 **billion edges** (results on Blue Waters, Cray XE6)

Communication + synchronization are a fundamental bottleneck in many algorithms

- scalability of standard algorithms for **dense LU, QR, SVD and sparse iterative methods** is **well understood theoretically**
- much to explore in practice, **important algorithms not yet implemented**
- lower bounds motivate more radical algorithmic changes

Bilinear algorithms and tensor factorization representations

- provide an analytical tool for deriving lower-bounds
- demonstrate insights on communication of new algorithms for symmetric tensor contractions
- enable succinct algebraic description in **native dimensionality**
- allow for **effective parallel implementation** based on high-level specification (Cyclops Tensor Framework)

