Parallel Implementation of Tensor Decompositions for Large Data Analysis.

Mark Sears
Brett Bader
Tammy Kolda

Sandia National Laboratories[1]

[1] Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy National Nuclear Security Administration under contract DE-AC04-94AL85000.

Background.

Kolda and Bader developed a MATLAB implementation of several tensor decompositions called the Tensor Toolkit.

Need for high performance implementation.

C++ serial and C++/MPI parallel implementation: expect 10-20x improvement in performance from C++ serial. Parallel implementation could give speedup proportional to number of processors, so possibly could get factors of 100-1000 over current technologies.

Also, implementation in C++/MPI should enable *much* larger tensors to be analyzed.

Tensors.

There are different names for the same thing:

array.

multidimensional array.

N-way array.

Tensor.

A tensor T is a multidimensional array of numbers

$$T(i_1, i_2, \cdots) \quad (1)$$

with dimensions $N_1, N_2, \cdots$. The number of dimensions is called the $order$ of the tensor. Order is two for matrices.

Physicists use the term tensor in a different (and frankly prior) context that has nothing to do with the present work.

Applications.

Tensor or multilinear algebra analysis is used in a wide variety of areas:

- Psychometrics.
- Chemometrics.
- Biometrics.
- Signal and image analysis.
- Text analysis.

Tensor analysis is often useful when the data has a natural description as a high dimensional array.

One unique property of tensors is that the indices do not have to have any associated topology – they can be words or names.

Models.

Tensor decompositions are models of a tensor in a least-squares sense: So if T is a tensor and $\tilde{T}$ is a model tensor then we want to minimize

$$\sigma = \left| T - \tilde{T} \right|^2 \tag{2}$$

or explicitly

$$\sigma = \sum_{i_1, i_2, \cdots} \left| T(i_1, i_2, \cdots) - \tilde{T}(i_1, i_2, \cdots) \right|^2 \tag{3}$$

SVD as a tensor model.

Consider order 2 tensors (aka matrices). Since T is a matrix, look at the SVD:

$$T = U\Sigma V^{\dagger} \tag{4}$$

where U is orthogonal $m \times n$, Σ is $m \times m$ diagonal, and V is $m \times m$ orthogonal.

We can consider the truncated rank R SVD to be a model of T:

$$\tilde{T} = \tilde{U}\tilde{\Sigma}\tilde{V}^{\dagger} \tag{5}$$

The best rank-R approximation to T is given by the R largest singular values and vectors of T.

We just reinvented PCA!

$$\tilde{T}(i,j) = \sum_r \lambda_r U(i,r) V(j,r) \tag{6}$$

Generalizations to higher order.

PARAFAC, CANDECOMP, CP:

$$\tilde{T}(i_1, i_2, i_3, \cdots) = \sum_r \lambda_r U(i_1, r) V(i_2, r) W(i_3, r) \cdots \quad (7)$$

Tucker, HOSVD:

$$\tilde{T}(i_1, i_2, i_3, \cdots) = \sum_{r,s,t} G(r, s, t, \cdots) U(i_1, r) V(i_2, s) W(i_3, t) \cdots \quad (8)$$

Generalizations to collections of tensors.

Might want to fit a collection of tensors T_k where the first dimension of the tensor is allowed to vary.

PARAFAC2:

$$\tilde{T}_k(i_1, i_2, i_3, \cdots) = \sum_r U_k(i_1, r) V(i_2, r) W(i_3, r) \cdots \quad (9)$$

where $U_k = P_k U$ and P_k is an orthogonal $N_1 \times R$ matrix.

We want to minimize

$$\sigma = \sum_k \sigma_k = \sum_k \left| T_k - \tilde{T}_k \right|^2 \quad (10)$$

Weird things.

DEDICOM:

What does large data mean?

In the context of this talk we are interested in generating models of tensors which have modest order (≤ 5, say) with very large dimensions and very sparse.

Example problems:

Enron: order 3, dimensions $197 \times 69157 \times 357$, with 1.77×10^6 nonzero entries.

G1: order 3, dimensions $1000 \times 1000 \times 1000$ with 10^6 nonzero entries.

G2: order 3, dimensions $2000 \times 2000 \times 2000$ with 8×10^6 nonzero entries.

T2: order 2 collection of 10 tensors with dimensions 4000×2000 with 8×10^5 nonzero entries.

T3: order 3 collection of 10 tensors with dimensions $400 \times 200 \times 300$ with 3×10^5 nonzero entries.

Algorithms.

ALS (Alternating Least Squares) algorithms for all of these models have been proposed and analyzed in the literature. These usually construct a single matrix factor at a time.

Example (PARAFAC):

$$\frac{\partial \sigma}{\partial U} = Y_U U - X_U \tag{11}$$

where Y_U is $R \times R$, X_U is $N_1 \times R$.

Construction of X_U requires most work.

$$X_U(i_1, r) = \sum_{i_2, i_3, \cdots} T(i_1, i_2, i_3, \cdots) V(i_2, r) W(i_3, r) \tag{12}$$

Dense version: matricize i_2, i_3 and build matrix of VW products. Then use BLAS.

Sparse version: Stream through nonzero elements of T, building X_U as we go.

A first order parallelization strategy.

Partition nonzero elements of T among P processors, more or less arbitrarily. (setup phase).

Global sum partial versions of X_U.

Computation of Y_U and solve for U is duplicated everywhere.

Comment:

Use existing sparse matrix data structures and code.

Develop something new and specific.

Existing sparse matrix code is heavily oriented towards matrix-vector multiplication. Not really very well suited to the tensor application.

Not very difficult to develop suitable data structures in C++: STL approach, simple table of indices, vector of values.

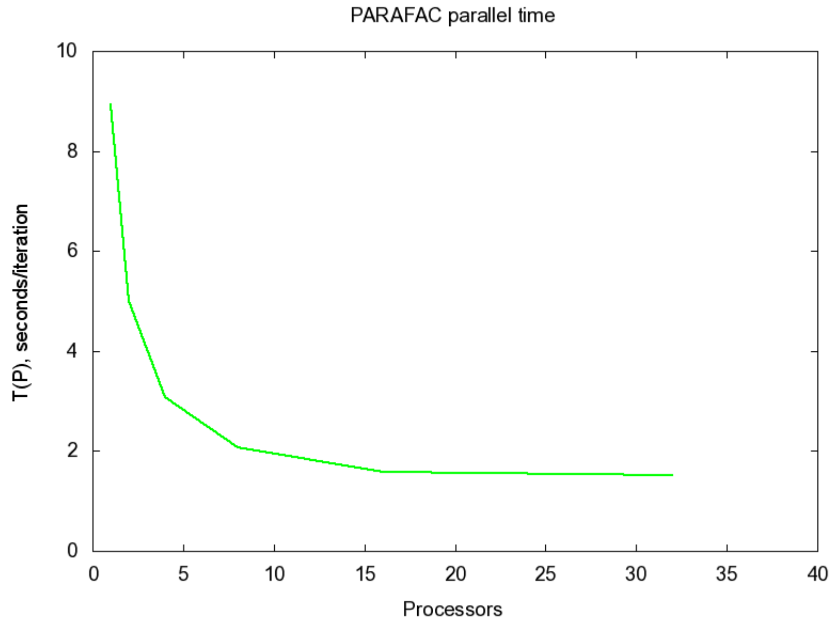
PARAFAC results.

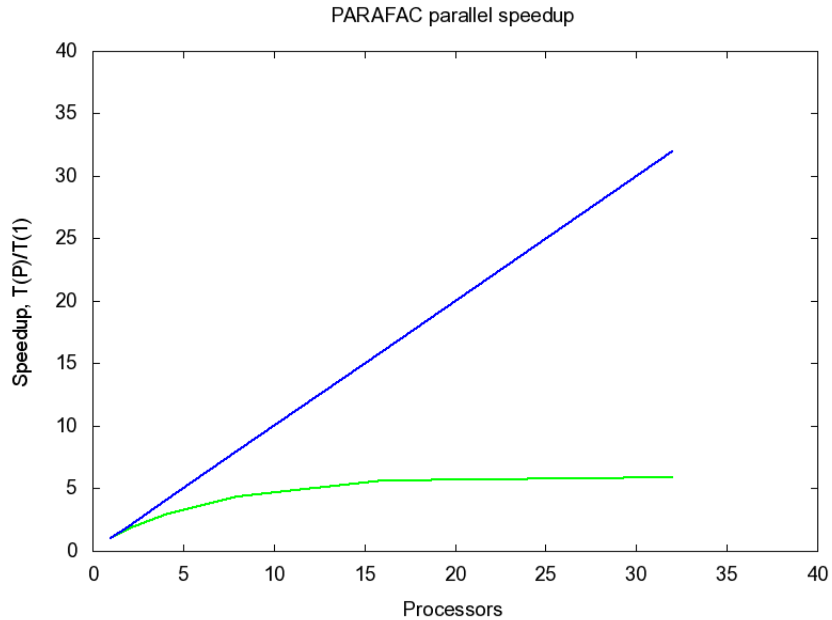
Enron data set:

Dimensions: $197 \times 69157 \times 357$

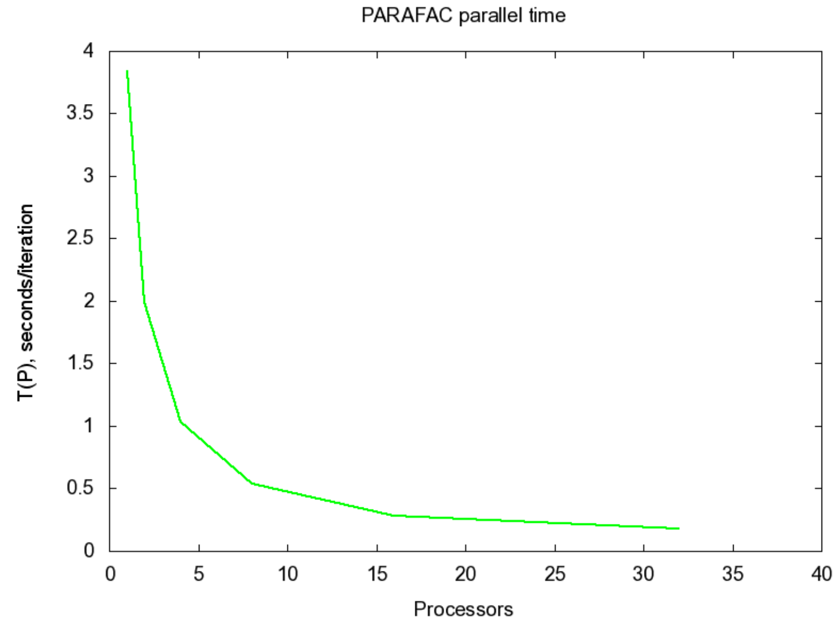
Nonzeros: 1.77×10^6

Model rank: 25





PARAFAC results.

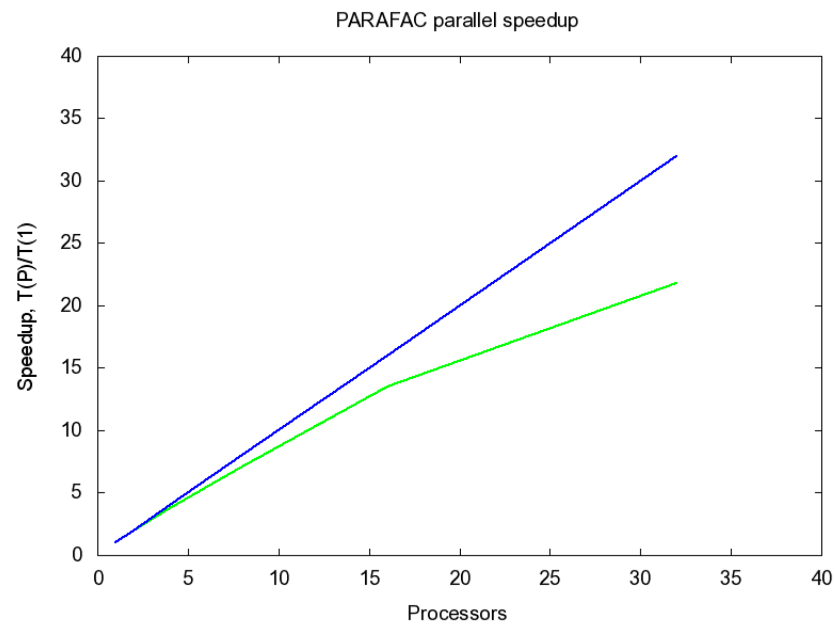


G1 data set:

Dimensions: $1000 \times 1000 \times 1000$

Nonzeros: $1. \times 10^6$

Model rank: 20



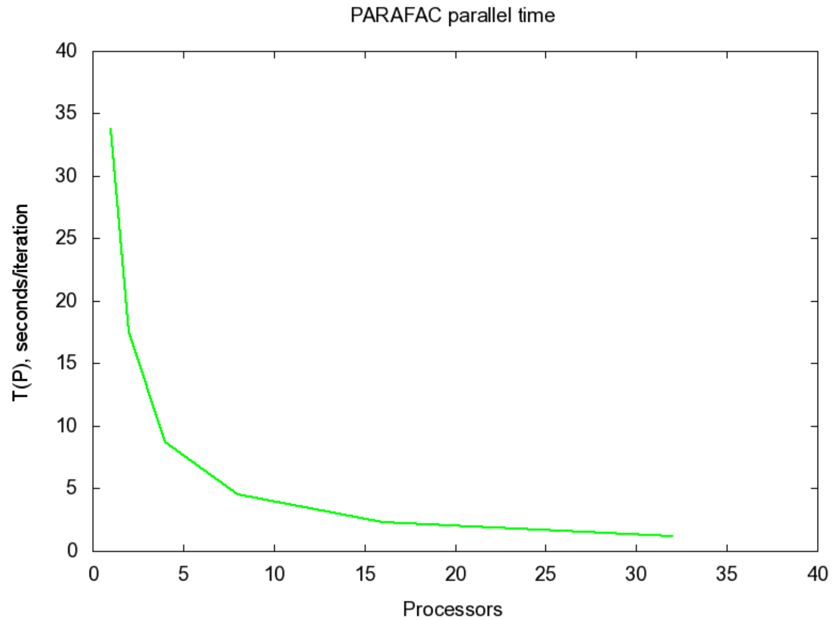
PARAFAC results.

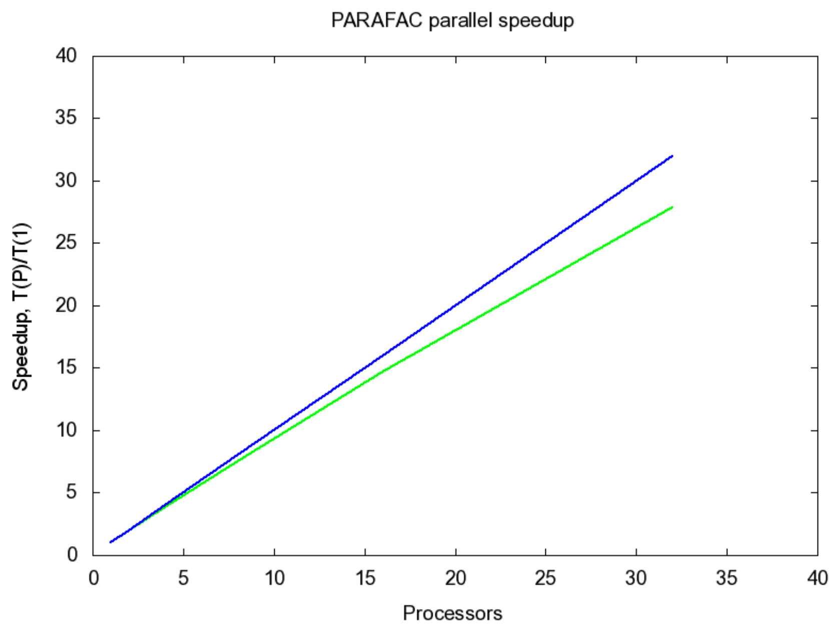
G2 data set:

Dimensions: $2000 \times 2000 \times 2000$

Nonzeros: $8. \times 10^6$

Model rank: 20





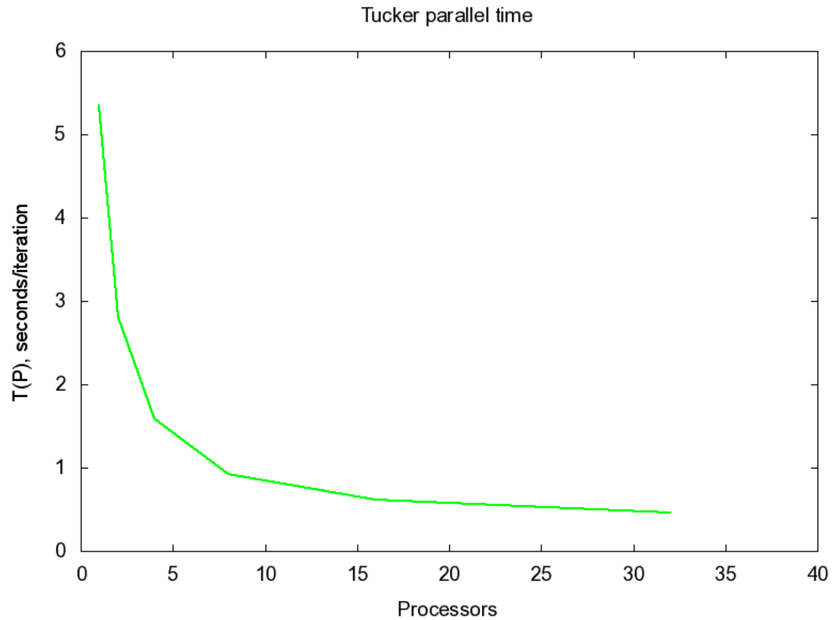
Tucker results.

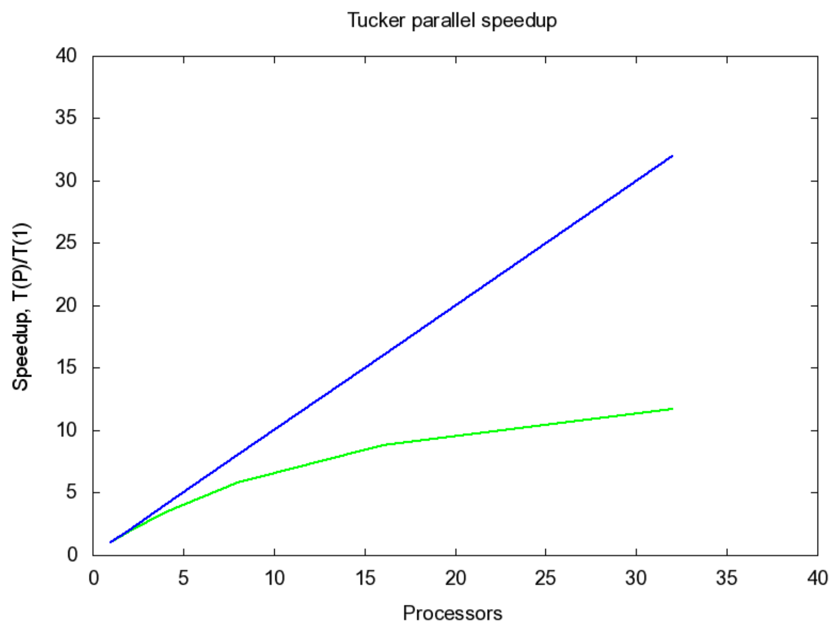
Enron data set:

Dimensions: $197 \times 69157 \times 357$

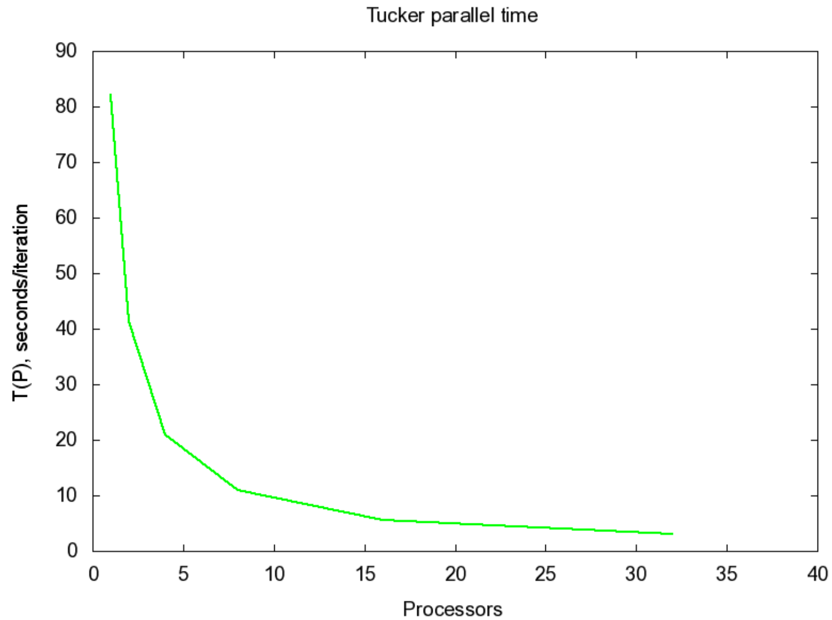
Nonzeros: 1.77×10^6

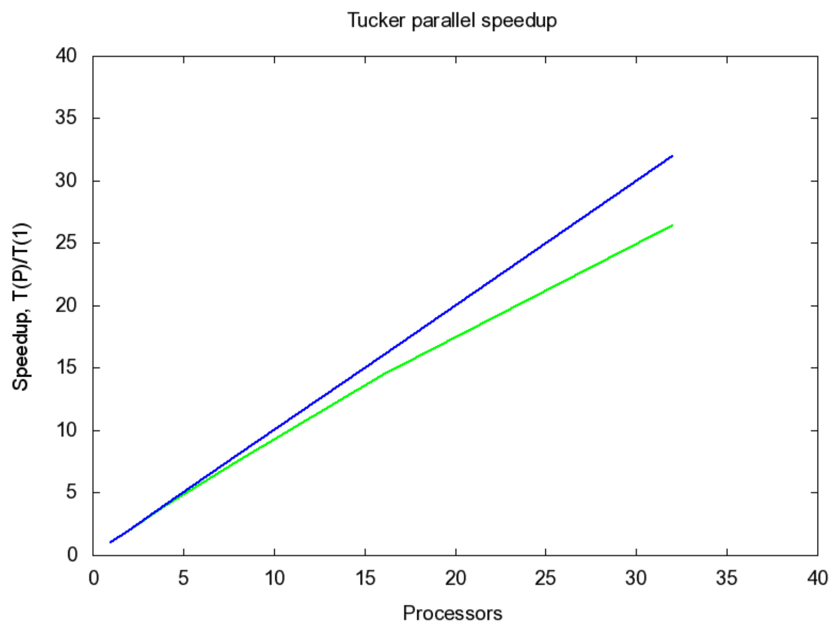
Model rank: $3 \times 3 \times 3$





Tucker results.





G1 data set:

Dimensions: $1000 \times 1000 \times 1000$

Nonzeros: $1. \times 10^6$

Model rank: $3 \times 3 \times 3$

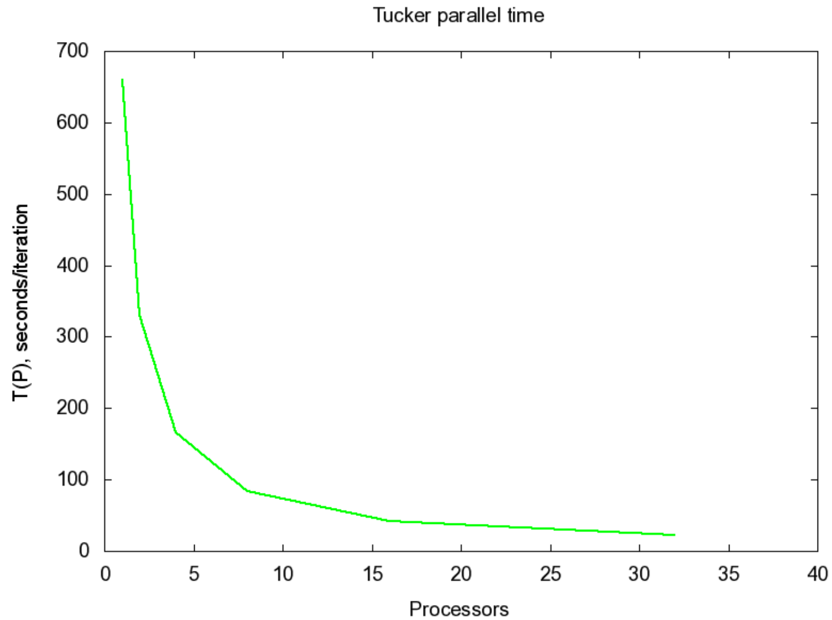
Tucker results.

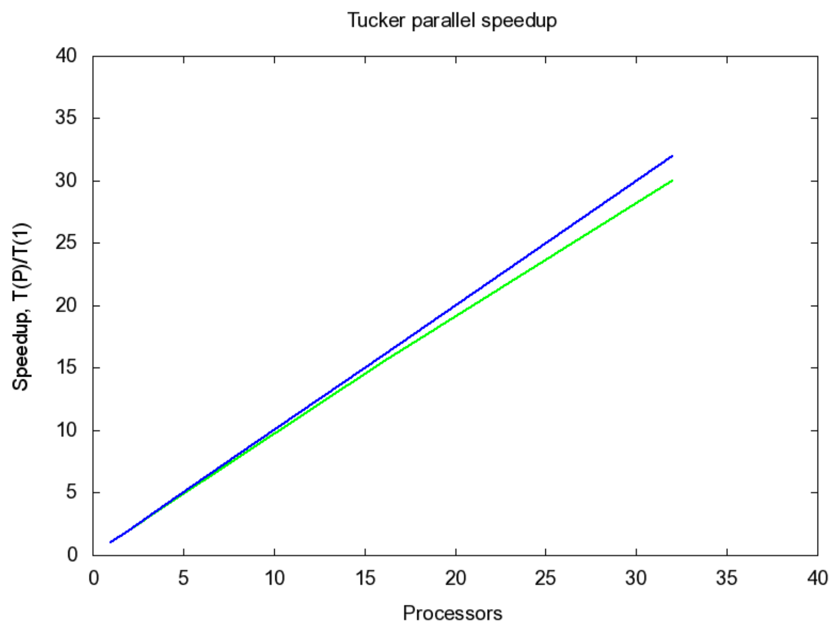
G2 data set:

Dimensions: $2000 \times 2000 \times 2000$

Nonzeros: $8. \times 10^6$

Model rank: $3 \times 3 \times 3$





PARAFAC2 results.

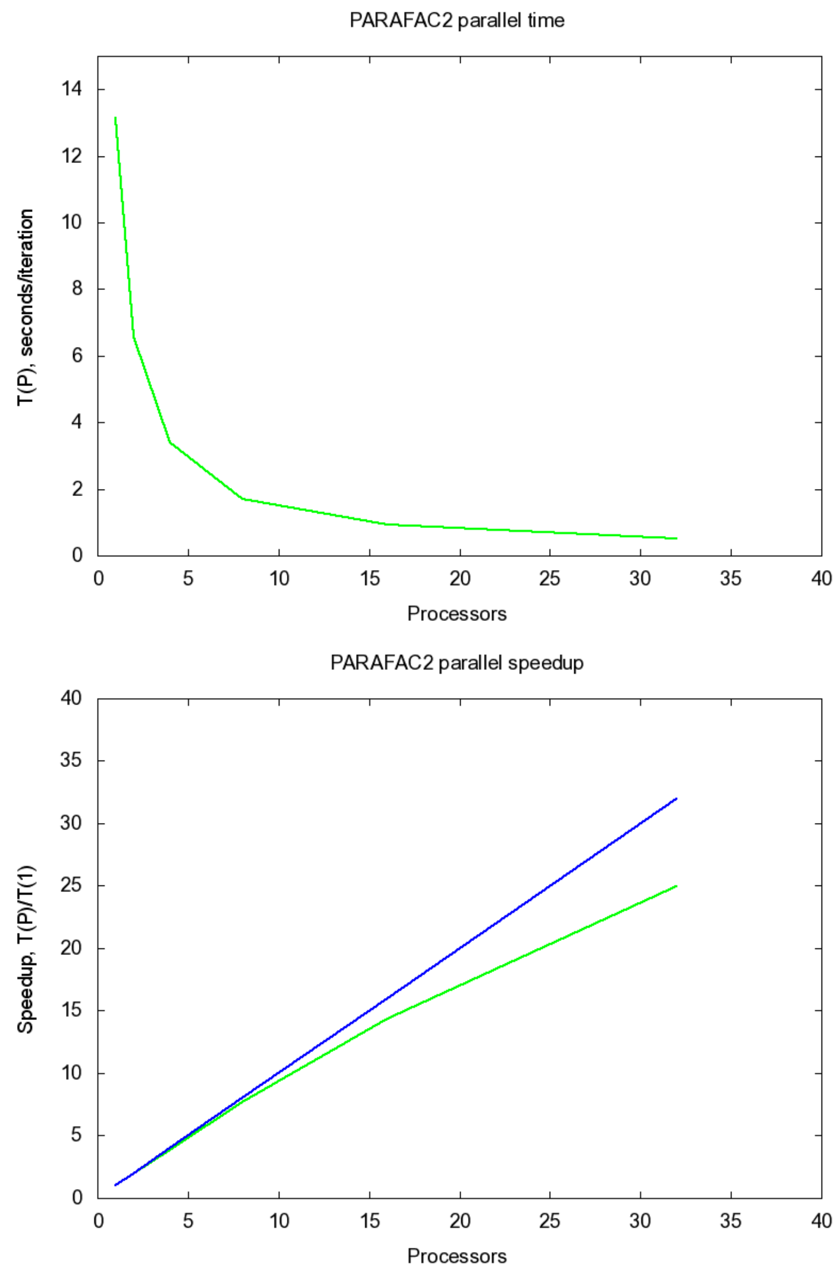
T2 data set:

10 samples

Dimensions: 4000×2000

Nonzeros: $8. \times 10^5$

Model rank: 5



PARAFAC2 results.

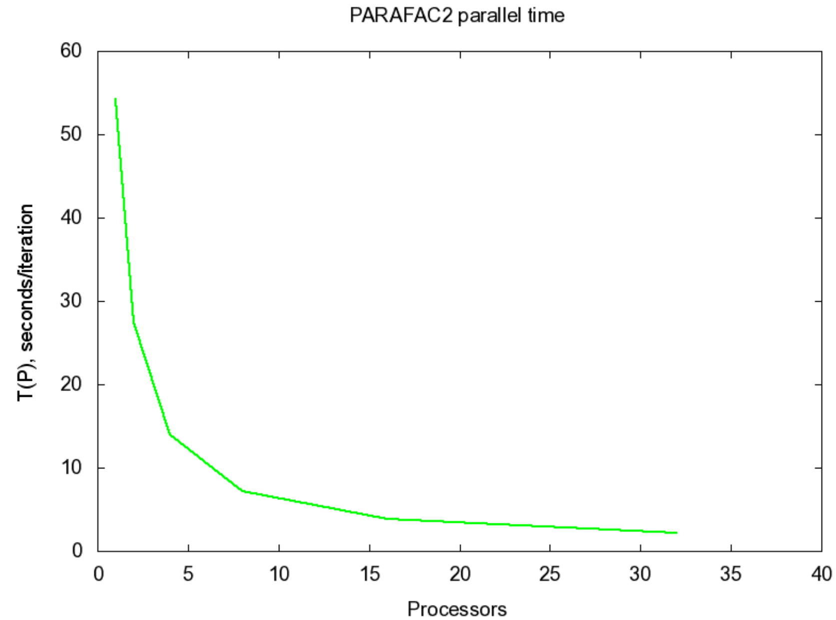
T3 data set:

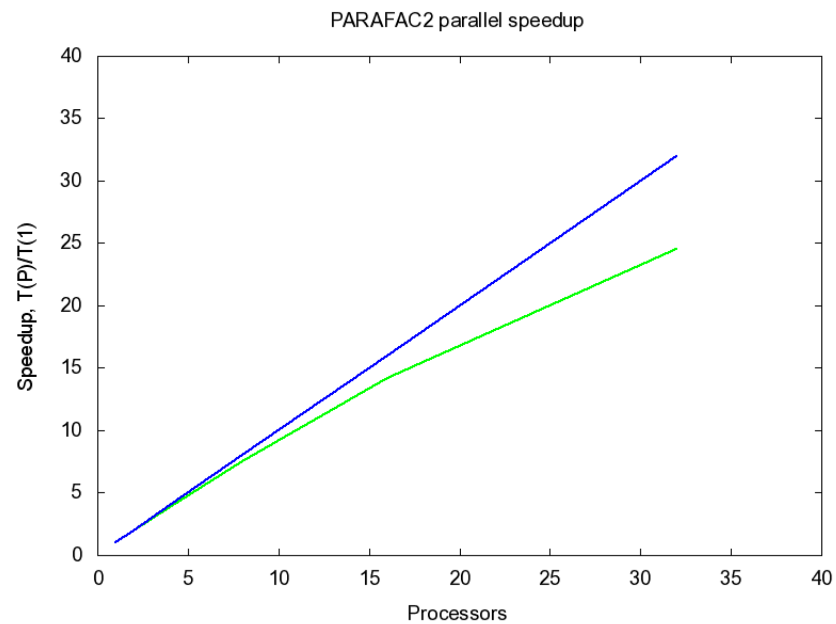
10 samples

Dimensions: $400 \times 200 \times 300$

Nonzeros: $3. \times 10^5$

Model rank: 5





Other issues.

C++ style.

Sparse tensor data formats.

Careful testing.

– Comparison with MATLAB results.
– Initialization.
– Sign and ordering ambiguities.

IO.

Dependencies: LU, small and large SVD, eigensolvers, BLAS, etc.

Things to do.

Better initialization.

Gradient methods.

DEDICOM.

Nonnegative factorizations.

Other decompositions: INDSCAL, CANDELINC, PARATUCK2, PARALIND, Block.

Conclusions.

Implemented C++/MPI library for sparse tensor modelling.

ALS type algorithms for several models are available:

– PARAFAC.
– Tucker.
– PARAFAC2.

Simple parallelization strategy works well.

Much larger tensors can be analyzed.