

SAND2005-1668  
Unlimited Release  
Printed March 2005  
Updated for Trilinos 9.0 February 2009

# Epetra Performance Optimization Guide

Michael A. Heroux  
Scalable Algorithms Department  
Sandia National Laboratories  
P.O. Box 5800  
Albuquerque, New Mexico 87185

## Abstract

Epetra is a package of classes for the construction and use of serial and distributed parallel linear algebra objects [1]. It is one of the base packages in Trilinos [2]. Many Trilinos solver packages can use Epetra object to provide basic linear algebra computations. In these cases, the performance of the solver is often determined primarily by the performance of the kernels in Epetra. For this reason, it is often advantageous to make sure that Epetra kernels are performing optimally. This document describes how to get the best performance from Epetra kernels. The ideas presented here can make a significant difference in the performance of Epetra, sometimes a dramatic difference.

**Acknowledgements:** The author would like to thank the ASC and LDRD programs that have funded this work.

## Table of Contents

1. Introduction.....	6
1.1 Practice Categories.....	6
1.2 Epetra Computational Classes .....	6
2. 3 <sup>rd</sup> Party Libraries: BLAS and LAPACK .....	7
3. Epetra_MultiVector Data Layout .....	8
4. Epetra_CrsGraph Construction .....	8
5. Epetra_CrsMatrix Construction .....	10
6. Selecting the Right Sparse Matrix Class.....	11
7. Parallel Data Redistribution .....	13
8. General Practices .....	13
References.....	14

# 1. Introduction

This document describes issues for the Epetra computational classes that have an impact on performance. We present the topics class by class and indicate via the symbols presented in Section 1.1 the typical impact of each recommended practice.

## 1.1 Practice Categories

Each practice falls into one of three categories:

**VSR**

**Very Strongly Recommended** - Practices necessary for Epetra to perform well.

**SR**

**Strongly Recommended** - Practices that are definitely a good thing or that have proved to be valuable.

**R**

**Recommended** - Practices that are probably a good idea.

## 1.2 Epetra Computational Classes

Epetra contains a base class called Epetra\_CompObject. This is a small class, but the classes that derive from it are exactly those which are the focus of this guide.

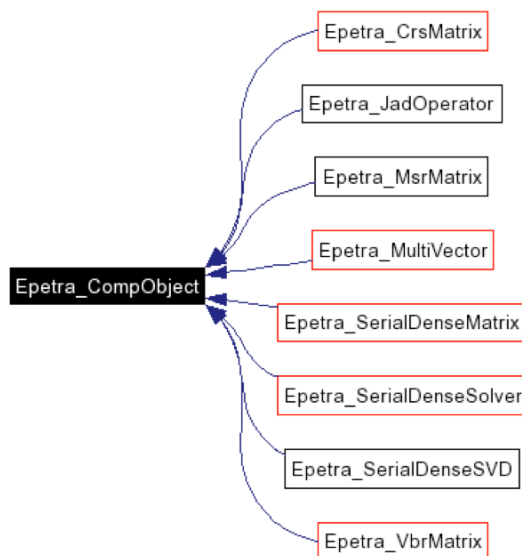


Figure 1: Epetra Computational Classes

## 2. 3<sup>rd</sup> Party Libraries: BLAS and LAPACK

The Basic Linear Algebra Subprograms (BLAS) [3, 4, 5] provide a *de facto* standard interface for a variety of common dense linear algebra kernels such as vector updates and dot products, matrix-vector multiplication and matrix-matrix multiplication. LAPACK [6] provides a very large collection of linear equation solvers, eigenvalue/eigenvector solvers and much more, and is built to get its performance from the BLAS. A number of high-performance implementation of the BLAS exist. In fact, every major computer platform has at least one high-performance BLAS implementation. These high-performance BLAS kernels can provide a speed-up of a factor of ten or more for some important operations. In particular, Level-2 and Level-3 BLAS [4, 5] can be much faster. Two versions that are commonly used are ATLAS [7] and the GOTO [8] BLAS.

A number of Epetra classes use BLAS and LAPACK to perform key calculations. Figure 2 shows the Epetra classes that depend on BLAS via the Epetra\_BLAS class. Epetra\_BLAS is a simple wrapper class that provides portable interfaces to the BLAS Fortran interfaces. It is convenient to use because calling FORTRAN from C++ varies from platform to platform, and Epetra\_BLAS handles the details of this issue.

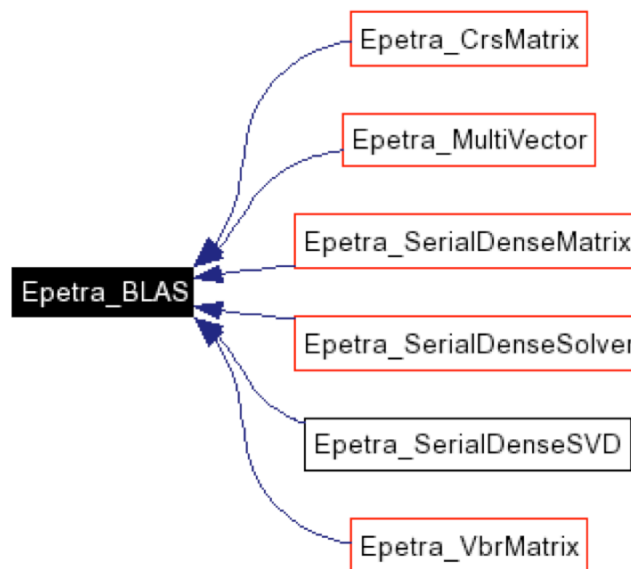


Figure 2: Epetra classes affected by BLAS performance

VSR

- **2.1 Link to high-performance BLAS for Epetra\_MultiVector, Epetra\_VbrMatrix and Epetra\_SerialDense performance:** Programs using Epetra should link to a high-performance BLAS library if
  - Epetra\_MultiVector objects are used for block vector updates and block dot products (via the Multiply() method in the Epetra\_MultiVector class) or
  - Epetra\_VbrMatrix objects are being used. These matrix objects rely on BLAS kernels for performing matrix-vector and matrix-multivector multiplication.

- Any of the Epetra\_SerialDense classes are used. High-performance BLAS are critical to the performance of these classes.

## 3. Epetra\_MultiVector Data Layout

An Epetra\_MultiVector object is a collection of dense vectors. Each vector in a multivector is a contiguous array of double-precision numbers. Generally these arrays are managed by the Epetra\_MultiVector object via an array of pointers. However, whenever possible, Epetra\_MultiVectors are created such that the arrays of double-precision numbers are contiguous, such that the last element of one array is next to the first element of the next array. Such an arrangement of the arrays is referred to as *strided* because the distance from the  $i^{\text{th}}$  element of one array to the  $i^{\text{th}}$  element of the next array is determined by a fixed length. This type of storage association is commonly used because it is exactly how FORTRAN stores two-dimensional arrays. As a result, BLAS kernels assume this type of layout for matrices. Additionally, there is an essential performance advantage when performing matrix computations using strided arrays.

### VSR

**3.1 Create Epetra\_MultiVector objects with strided storage of vectors:** Epetra kernels will tend to perform much better if the underlying vectors have a strided storage associated. Although Epetra\_MultiVectors support non-strided storage, we recommend it be avoided when possible. In particular, use strided storage if:

- Epetra\_MultiVector objects are used for block vector updates and block dot products (via the Multiply() method in the Epetra\_MultiVector class) or
- Epetra\_MultiVectors are being used with Epetra\_CrsMatrix or Epetra\_VbrMatrix object for sparse matrix-vector multiplication or sparse triangular solves. In these instances, the performance of kernels can be up to twice as fast if the multivector has strided storage.

## 4. Epetra\_CrsGraph Construction

An Epetra\_CrsGraph object contains the structural information (the non-zero pattern) for Epetra\_CrsMatrix and Epetra\_VbrMatrix objects. An Epetra\_CrsGraph can be constructed explicitly prior to constructing a matrix, and then passed in to the matrix constructor. Alternatively, if a matrix is constructed without passing in an existing Epetra\_CrsGraph, the Epetra\_CrsGraph will be constructed “on-the-fly”.

Epetra\_CrsGraph objects are constructed in a three-step process:

1. Instantiate the Epetra\_CrsGraph object.
2. Insert graph indices.

3. Call FillComplete() to process the graph data for efficient computation and communication.

Step 1 involves a single call to the constructor and Step 3 to a FillComplete() method. However, Step 2 can be accomplished in a number of ways. Indices can be inserted row-by-row, index-by-index or any combination of these two approaches. Indices can be inserted redundantly if the user does not want to track which indices are already in the graph and indices can be removed. Although this flexibility is convenient, especially in prototyping stages of development, it does introduce overhead in memory and time costs. Therefore, Epetra\_CrsGraph objects can be constructed in more restrictive modes that require more information and attention from the user, but in return the user can significantly reduce memory and time costs.

VSR

**4.1 Construct Epetra\_CrsGraph objects first:** When constructing multiple Epetra\_CrsMatrix or Epetra\_VbrMatrix objects, much of the overhead in sparse matrix construction is related solely to the graph structure of the matrix. By pre-constructing the Epetra\_CrsGraph object, this expense is incurred only once and then amortized over multiple matrix constructions. **Note:** Even when constructing a single matrix, it is often the case that matrix construction is done within a nonlinear iteration or a time-stepping iteration. In both of these cases, it is best to construct the Epetra\_CrsGraph object one time and then reuse it.

VSR

**4.2 When constructing Epetra\_CrsGraph objects, carefully manage the nonzero profile and set StaticProfile to ‘true’ :** Although it is very convenient to use the flexible insertion capabilities of Epetra\_CrsGraph, the performance and memory cost can be substantial. StaticProfile is an optional argument to the Epetra\_CrsGraph constructors that forces the constructor to pre-allocate all storage for the graph, using the argument NumIndicesPerRow as a strict upper limit on the number of indices that will be inserted.

VSR

**4.3 After calling FillComplete(), call OptimizeStorage() :** (Note: With Trilinos 8.0 and later, OptimizeStorage() is called automatically.) The OptimizeStorage() method frees memory that is used only when submitting data to the object. Also, the storage that remains is packed for better cache memory performance due to better spatial locality. If StaticProfile is true, the packing of data is fast and cheap, if needed at all<sup>1</sup>. If StaticProfile is false, then the graph data is not contiguously stored, so we must allocate contiguous space for all entries, copy data from the existing arrays, and then delete the old arrays. This can be a substantial overhead in memory costs and could even double the high-water memory mark. However, the performance improvement for matrix operations can be 20% to a full doubling of performance.

VSR

**4.4 Use Epetra\_FECrsMatrix if you have many repeated indices. Then extract the associated Epetra\_CrsGraph for subsequent use:** Although there is no Epetra\_FECrsGraph class (something that we may introduce in the future), it is typically best to use the Epetra\_FECrsMatrix class to construct matrices when you have many repeated indices. This is typically the case when forming global stiffness matrices from local element

---

<sup>1</sup> If the number of entries in the matrix *exactly* match the profile used to create the matrix, packing is does not need to move any data, but it is not a big performance hit if the profile is somewhat larger.

stiffness matrices in a finite element assembly loop. **Note:** Even though there is no Epetra\_FECrsGraph class, once an Epetra\_FECrsMatrix has been constructed, call it myFEMatrix, there is an Epetra\_CrsGraph that can be accessed via myFEMatrix.Graph() and used to construct further matrices that have the same pattern as myFEMatrix.

SR

**4.5 When inserting indices into Epetra\_CrsGraph objects, avoid large numbers of repeated indices:** To reduce time costs, indices that are inserted into a row are not checked for redundancy at the time they are inserted. Instead, when FillComplete() is called, indices are sorted and then redundant indices are removed. Because of this approach, repeated indices increase the memory used by a given row, at least until FillComplete() is called.

SR

**4.6 Submit large numbers of graph indices at once:** When inserting indices, submit many entries at once if possible. Ideally, it is best to submit all indices for a given row at once.

## 5. Epetra\_CrsMatrix Construction

An Epetra\_CrsMatrix object stores a sparse matrix in a row-biased data structure (although column support is implicitly available because transpose operations are also supported). An Epetra\_CrsMatrix can be constructed using a previously constructed Epetra\_CrsGraph object or from scratch, in which case the graph will be constructed “on the fly”. Matrix entries can be inserted, summed-into or replaced one-at-a-time or as row fragments.

Much like Epetra\_CrsGraph objects, Epetra\_CrsMatrix objects are constructed in a three-step process:

1. Instantiate the Epetra\_CrsMatrix object.
2. Submit matrix values and corresponding column indices via:
  - a. Insertion: New values that have no entry.
  - b. Sum-into: Values that already exist.
  - c. Replacement: Replace an existing value with a new one.
3. Call FillComplete() to process the matrix for efficient computation and communication.

Step 1 involves a single call to the constructor and Step 3 to a FillComplete() method. However, Step 2 can be accomplished in a number of ways. Matrix entries can be submitted row-by-row, index-by-index or any combination of these two approaches. Although this flexibility is convenient, especially in prototyping stages of development, it does introduce overhead in memory and time costs. Therefore, Epetra\_CrsMatrix objects can be constructed in more restrictive modes that require more information and attention from the user, but in return the user can significantly reduce memory and time costs.

VSR

**5.1 When constructing Epetra\_CrsMatrix objects, carefully manage the nonzero profile and set StaticProfile to ‘true’ :** As is true with Epetra\_CrsGraph objects, it is very



convenient to use the flexible insertion capabilities of `Epetra_CrsMatrix`. However, the performance and memory cost of this flexibility can be substantial. `StaticProfile` is an optional argument to the `Epetra_CrsMatrix` constructors that forces the constructor to pre-allocate all storage for the matrix entries (and the `Epetra_CrsGraph`, if it is being constructed on the fly), using the argument `NumIndicesPerRow` as a strict upper limit on the number of matrix entries that will be inserted. **Note: `StaticProfile` is false by default.**

VSR

**5.2 After calling `FillComplete()`, call `OptimizeStorage()` :** (Note: With Trilinos 8.0 and later, `OptimizeStorage()` is called automatically.) As is true with `Epetra_CrsGraph` objects, the `OptimizeStorage()` method frees memory that is used only when submitting data to the object. Also, the storage that remains is packed for better cache memory performance due to better spatial locality. If `StaticProfile` is true, the packing of data is fast and cheap, if needed at all<sup>2</sup>. If `StaticProfile` is false, then the matrix data is not contiguously stored, so we must allocate contiguous space for all entries, copy data from the existing arrays, and then delete the old arrays. This can be a substantial overhead in memory costs and could even double the high-water memory mark. However, the performance improvement for matrix operations can be 20% to a full doubling of performance.

SR

**5.3 Submit large numbers of matrix entries at once:** When submitting entries via insertion, sum-into or replacement, submit many entries at once if possible. Ideally, it is best to submit all entries for a given row at once.

Also see Practices: 3.1, 4.1, and 4.3.

## 6. Selecting the Right Sparse Matrix Class

Epetra provides a base matrix class called `Epetra_RowMatrix`. `Epetra_RowMatrix` is a pure virtual class that provides an interface to sparse matrices, allowing access to matrix coefficients, matrix distribution information and methods to compute matrix multiplication and local triangular solves. Epetra provide four classes that are implementations of `Epetra_RowMatrix`. Specifically:

1. **`Epetra_CrsMatrix`:** Compressed Row Storage scalar entry matrix.
2. **`Epetra_FECrsMatrix`:** Finite Element Compressed Row Storage matrix.
3. **`Epetra_VbrMatrix`:** Variable Block Row block entry matrix.
4. **`Epetra_FEVbrMatrix`:** Finite Element Variable Block Row matrix.

**Note:** `Epetra_JadOperator` does not implement `Epetra_RowMatrix`, but is worth mentioning for users of vector processors. This class is a first version to support vectorization of sparse matrix-vector multiplication.

In addition, there are numerous other implementation of `Epetra_RowMatrix` provided in other packages. For example:

1. `Epetra_MsrMatrix`: This class is provided within AztecOO. The constructor for `Epetra_MsrMatrix` takes a single argument, namely an existing `AZ_DMSR` matrix as

---

<sup>2</sup> If the number of entries in the matrix *exactly* match the profile used to create the matrix, packing is does not need to move any data, but it is not a big performance hit if the profile is somewhat larger.

defined in the Aztec 2.1 User Guide [9]. Epetra\_MsrMatrix does *not* make a copy of the Aztec matrix. Instead it make the Aztec matrix act like an Epetra\_RowMatrix object.

2. Ifpack Filters: Ifpack uses the Epetra\_RowMatrix interface to provide modified views of existing Epetra\_RowMatrix objects. For example, Ifpack\_DropFilter creates a new Epetra\_RowMatrix from an existing one by dropping all matrix values below a certain tolerance.

VSR

**6.1 Use Epetra\_FEVbrMatrix to construct a matrix with dense block entries and repeated summations into block entries:** Epetra\_FEVbrMatrix is designed to handle problems such as finite element, finite volume or finite difference problems where multiple degrees of freedom are being tracked at a single mesh point, and the matrix block entries are being summed into the global stiffness matrix one finite element or control volume cell at a time. In these situations, one can construct an Epetra\_CrsGraph object that encodes the mesh connectivity, using Epetra\_BlockMap objects to describe how many degrees of freedom are being tracked at each mesh node. **Note:** Epetra\_FEVbrMatrix is derived from Epetra\_VbrMatrix. At the end of the construction phase, the Epetra\_FEVbrMatrix object *isa* Epetra\_VbrMatrix object.

R

**6.2 Use Epetra\_VbrMatrix to construct a matrix with dense block entries and few repeated submissions:** Epetra\_VbrMatrix is designed to handle problems such as finite element, finite volume or finite difference problems where multiple degrees of freedom are being tracked at a single mesh point, but matrix block entries are typically not repeated summed into the global stiffness matrix one element or control volume cell at a time. In these situations, one can construct an Epetra\_CrsGraph object that encodes the mesh connectivity, using Epetra\_BlockMap objects to describe how many degrees of freedom are being tracked at each mesh node. **Note:** Presently the Epetra\_VbrMatrix class does not have optimal performance for small block entry matrices. Block entries should be of size 4 or larger before Vbr formats are preferable to Crs formats.

VSR

**6.3 Use Epetra\_FECrsMatrix to construct a matrix with scalar entries and repeated summations into entries:** Epetra\_FECrsMatrix is designed to handle problems such as finite element, finite volume or finite difference problems where a single degree of freedom is being tracked at a single mesh point, and the matrix entries are being summed into the global stiffness matrix one finite element or control volume cell at a time.

VSR

**6.4 Use Epetra\_CrsMatrix in all other cases:** Epetra\_CrsMatrix is the simplest and most natural matrix data structure for people who are used to thinking about sparse matrices. Unless you have one of the above three situations, you should use Epetra\_CrsMatrix.

VSR

**6.5 Use Epetra\_RowMatrix to access matrix entries:** If you are writing code to use an existing Epetra sparse matrix object, use the Epetra\_RowMatrix interface to access the matrix. This will provide you compatibility with all of Epetra's sparse matrix classes, and allow users of your code to provide their own custom implementation of Epetra\_RowMatrix if needed.



## 9. Extended Capabilities

Epetra provides several general-purpose data structures for sparse computations. However, there are other sources for high performance kernels and we provide access to several other libraries. Furthermore, it is often the case that—for users who know their data—special-purpose kernels can provide the best performance.

R

**Use OSKI if it performs well for your problems:** Trilinos provides an interface to OSKI [10] the sparse kernel package from UC Berkeley. Information on its performance characteristics is available [11]. The primary class of interest is `Epetra_OskiMatrix`.

R

**Use PETSc AIJ Matrix wrappers if already using PETSc:** Current PETSc [12] users who have already invested in PETSc data structures can use PETSc matrices and vectors with Trilinos solvers and preconditioners using the `Epetra_PETScAIJMatrix` class, which wraps the PETSc matrix and uses it to satisfy the `Epetra_RowMatrix` interface. Only minimal extra storage is required. This class is part of `EpetraExt`.

R

**Write your own custom matrix class for optimal storage and performance:** Although any of the above approaches can be a good answer for good sparse matrix kernel performance, it is almost always possible for sophisticated users to provide their own implementation of `Epetra_RowMatrix` that specifically addresses their problem and performs better than a generic approach. Trilinos is different than all other library packages that we know about in that all of our solvers and preconditioners use the `Epetra_RowMatrix` interface to access data. Because of this, even if you use your own matrix data structures, you can still use all Trilinos preconditioners and solvers since they are not tied to a data structure, but instead access data through an interface.

# References

- [1] Epetra Home Page: <http://software.sandia.gov/trilinos/packages/epetra>, 11 March 2005.
- [2] Trilinos Home Page: <http://software.sandia.gov/trilinos>, 8 December 2003.
- [3] C. Lawson, R. Hanson, D. Kincaid, and F. Krogh. *Basic linear algebra subprograms for Fortran usage*. ACM Transactions on Mathematical Software, 5, 1979.
- [4] J.J. Dongarra, J. DuCroz, S. Hammarling, and R. Hanson. *An extended set of Fortran basic linear algebra subprograms*. ACM Transactions on Mathematical Software, 14, 1988.
- [5] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. *A set of level 3 basic linear algebra subprograms*. ACM Transactions on Mathematical Software, 16(1):1–17, March 1990.
- [6] E. Anderson, Z. Bai, C. Bischof, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users' Guide*. SIAM Pub., Philadelphia, PA, Third Edition, 1999.
- [7] Atlas BLAS Home Page: <http://math-atlas.sourceforge.net>, 11 March 2005.
- [8] GOTO BLAS Home Page: <http://www.cs.utexas.edu/users/flame/goto>, 11 March 2005.
- [9] Ray S. Tuminaro, Michael A. Heroux, Scott. A. Hutchinson, and J. N. Shadid. *Official Aztec User's Guide, Version 2.1*. Sandia National Laboratories, Albuquerque, NM 87185, 1999.
- [10] OSKI Home Page : <http://bebop.cs.berkeley.edu/oski/>, 8 February 2009.
- [11] Karlin, I. and J. Hu. *Overview and Performance Analysis of the Epetra/OSKI Matrix Class in Trilinos*. in *CSRI Summer Proceedings*. 2008.
- [12] PETSc Home Page : <http://www.mcs.anl.gov/petsc/petsc-as/>, 8 February 2009.