Basic Linear Algebra Subroutines for Use in the System for Population Kinetics

Table of Contents

Introduction

The BLAS (Basic Linear Algebra Subprograms) are building block routines for performing basic vector and matrix operations. Because the System for Population Kinetics (SPK) performs many linear algebra operations, it uses the BLAS equivalent functions from the Numerical Algorithms Group (NAG) C library in multiple places.

The motivation for this research project is the need to remove the NAG library from SPK and to replace it with a different linear algebra library that is fast, free, open-source, and has no restrictions on the use of its source code or binaries derived from that source.

Candidates for Replacing the NAG Linear Algebra Routines

The following table lists possible candidates for replacing the NAG library BLAS equivalent functions used in SPK and for use in other Resource for Population Kinetics (RFPK) software products. All of these candidates are free and have no licensing or use restrictions.

Table 1. BLAS Candidates for Inclusion in SPK

Candidate	Description	Evaluated?
Automatically Tuned Linear Algebra Software (ATLAS)	ATLAS is a software tool that generates C versions of the BLAS library. It uses "Automated Empirical Optimization of Software" in order to provide portable performance. In particular, ATLAS provides many ways of doing the required operations and uses emperical timings to choose the best method for a given architecture. It can be found at math-atlas.sourceforge.net.	Yes.

Candidate	Description	Evaluated?
uBLAS	uBLAS is a C++ template class library that provides BLAS level 1, 2, 3 functionality for dense, packed and sparse matrices. The design and implementation unify mathematical notation via operator overloading and efficient code generation via expression templates. It can be found at www.genesys-e.org/ublas.	Yes.
Fortran77 Reference BLAS	This is a Fortran77 implementation of the BLAS. According to timing results from the ATLAS documentation, this library is 4 to 10 times faster than the ATLAS generated C BLAS and vendor supplied BLAS's on various machines. The Fortran77 Reference BLAS can be found in the Netlib software repository: www.netlib.org/blas.	No. It is not known if Fortran77 would be compatible with the parallel architecture that SPK eventually uses.
Blitz++	The Blitz++ library uses C++ templates to implement fast vectors and multidimensionl arrays. It is competitive with Fortran for some benchmark problems. It can be found at www.oonumerics.org/blitz.	No. This library does not provide any linear algebra support.
Matrix Template Library (MTL)	MTL is a generic component library that supports a wide variety of matrix formats. It can be found at www.osl.iu.edu/research/mtl.	No. This library was previously included in SPK where it performed very poorly.

ATLAS vs. uBLAS: Matrix Multiplication

ATLAS and uBLAS were first compared using matrix multiplication with full matrices that had no sparsity structure.

The following table shows the evaluation times for matrix multiplication of increasingly large double precision matrices. The evaluation times for the "C array" row are those for normal matrix multiplication using C arrays and are included for reference. The number of evaluations is larger for the smaller matrices so that the total times are on the order of a few seconds.

Table 2. Seconds Spent Performing Matrix Multiplication for Various Matrix Sizes

Size of Matrices	3 by 3	10 by 10	30 by 30	100 by 100
Evaluations	**10^7**	**3 x 10^5**	**10^4**	**300**
C array	2.14	1.32	1.19	1.29
uBLAS	3.68	1.72	1.35	1.44
ATLAS	8.90	1.61	0.68	0.46

For small matrices (3 by 3), ATLAS is slower than uBLAS and C arrays. For medium matrices (10 by 10), ATLAS is about the same speed as uBLAS and C arrays. For large

matrices (30 by 30, and larger), ATLAS is faster than uBLAS and C arrays.

Conclusion: For full matrices of the sizes typically found in SPK, ATLAS performs better than uBLAS.

ATLAS vs. uBLAS: Sparse Matrix Multiplication

Since uBLAS has sparse matrix capibilities, ATLAS and uBLAS were next compared using matrix multiplication where the uBLAS matrices were of the sparse type but the ATLAS matrices (and the C arrays) were full matrices that had no sparsity structure.

Note that "sparse type" refers to the C++ data structure provided by uBLAS, while "full" and "diagonal" refer to the actual structure of the matrices, i.e., the locations of the zero values.

The following table shows the evaluation times for matrix multiplication of increasingly large double precision matrices. The evaluation times for the "C array" row are those for normal matrix multiplication using C arrays and are included for reference. The difference between the full and diagonal uBLAS sparse matrices is that full matrices have nonzero values for all of their elements while the diagonal matrices only have nonzero values along their diagonals. They are both of uBLAS sparse type. The number of evaluations is larger for the smaller matrices so that the total times are on the order of a few seconds.

Table 3. Seconds Spent Performing Matrix Multiplication for Various Matrix Sizes

Size of Matrices	**3 by 3**	**10 by 10**	**30 by 30**	**100 by 100**
Evaluations	**10^7**	**3 x 10^5**	**10^4**	**300**
C array	2.10	1.34	1.13	1.24
uBLAS Sparse (Full)	141.41	70.86	41.19	35.85
uBLAS Sparse (Diagonal)	107.06	36.53	11.74	4.09
ATLAS	8.61	1.90	0.59	0.48

For small matrices (3 by 3), ATLAS is faster than sparse uBLAS but slower than C arrays. For medium matrices (10 by 10), ATLAS is faster than sparse uBLAS and about the same speed as C arrays. For large matrices (30 by 30, and larger), ATLAS is faster than uBLAS and C arrays.

It is especially noteworthy that when two sparse type uBLAS matrices are actually full, the performance of uBLAS is extremely poor. Since SPK will not know ahead of time which matrices are sparse and which are full, if sparse uBLAS matrices were included in SPK, then there would be many times where the sparse uBLAS data structures would be full. This would severly degrade the performance of SPK.

Conclusion: For matrices of the sizes typically found in SPK, and even if uBLAS takes advantage of extreme sparseness and ATLAS does not, ATLAS performs better than uBLAS.

ATLAS vs. uBLAS: Software Engineering Issues

The following table compares ATLAS and uBLAS with regards to some software en-

gineering issues.

Table 4. Software Engineering Comparison of ATLAS and uBLAS

ATLAS	uBLAS
Would require minimal SPK code modifications.	Would require major SPK code modifications.
ATLAS is written in C, and the BLAS libraries it generates are also C. ATLAS should produce optimized libraries on almost any platform possessing an ANSI/ISO C compiler, some Unix-like command-line tools, and hierarchical memory. (For more details, see the file ATLAS/doc/atlas_over.ps that is included in the ATLAS distribution.)	uBLAS uses C++ expression templates which are part of the standard C++ library but which are not supported by Microsoft's C++ compiler.
ATLAS is low risk since the number of BLAS users is large in general, since some major players (Maple, Mathematica, Matlab, Octave) use ATLAS, and since it could be replaced with vendor supplied BLAS or other open source BLAS libraries if ATLAS ever stopped being developed.	uBLAS is high risk since it is relatively new, since it is being developed by a small number of people which means it may or may not still be around in 5 years, and since it would require significant modifications to the code in SPK to take it out if uBKAS ever stopped being developed.
Conforms to the usual BLAS paradigm used by the scientific programming community.	Uses new data types and function names to perform BLAS equivalent operations.

Recommendations

ATLAS should be the replacement for the NAG linear algebra routines rather than uBLAS.

Appendix: Installing ATLAS and Generating the C BLAS Library

Here are some basic instructions for installing ATLAS and using it to generate a C BLAS library.

Notes

1. These instructions are a record of how these procedures were performed on a particular machine.
2. The paths used in this document are not necessarily appropriate for other machines nor are they necessarily consistent with the current directory structure for RFPK software products.

Installing ATLAS on Your Machine

1. Go to the ATLAS homepage on the internet:

   `math-atlas.sourceforge.net`

2. Click on the "Software" link on that page, which should take you to

   `https://sourceforge.net/project/showfiles.php?group_id=23725`

3. Download a stable, platform independent version of ATLAS, for example,

   `math-atlas/atlas3.4.1.tar.gz`

   and place it in the appropriate directory, for example,

   `home/watrous/Atlas`

4. Go to the directory where the archive was placed and then extract/uncompress it by typing

   `tar xzf math-atlas/atlas3.4.1.tar.gz`

Generating the C BLAS Using ATLAS

1. Follow the instructions in the file

   `ATLAS/INSTALL.txt`

   which describes how to configure, install, and "sanity" test ATLAS.

2. For the most part, choose the default values provided.

3. At the end of the process there will be several ATLAS generated BLAS libraries on your machine. If your machine is a Pentium IV running Linux, for example, then these libraries will be located in the directory:

   `ATLAS/lib/Linux_P4SSE2`

   The library libatlas.a is the C BLAS library, and the file libcblas.a is the C interface to that library.

Appendix: Installing and Testing uBLAS

Here are some basic instructions for installing and testing uBLAS.

Notes

1. These instructions are a record of how these procedures were performed on a particular machine.

2. The paths used in this document are not necessarily appropriate for other machines nor are they necessarily consistent with the current directory structure for RFPK software products.

Installing uBLAS on Your Machine

1. uBLAS is part of Boost. Information about Boost can be found on its homepage on the internet:

   ```
   www.boost.org
   ```

   The easiest way to install uBLAS is to install the whole Boost library.

2. Go to the download section of the Boost website:

   ```
   http://www.boost.org/more/download.html
   ```

   and click on the "download releases from SourceForge" link on that page. That link should take you to

   ```
   http://sourceforge.net/project/showfiles.php?group_id=7586
   ```

3. Download a platform independent version of Boost, for example,

   ```
   boost_1_30_0.tar.gz
   ```

   and place it in the appropriate directory, for example,

   ```
   home/watrous/Boost
   ```

4. Go to the directory where the archive was placed and then extract/uncompress it by typing

   ```
   tar xzf boost_1_30_0.tar.gz
   ```

5. Instructions for building the Boost library can be found in the file

   ```
   boost_1_30_0/tools/build/index.html
   ```

   Instructions for building the Boost.Jam executable, which is a make-like tool that comes with Boost, are located at the bottom of that page of instructions and also in the file

   ```
   boost_1_30_0/tools/build/jam_src/index.html
   ```

   Before the Boost library can be built, however, the Boost.Jam executable must be built first.

6. In order to build the Boost.Jam executable change to

   ```
   boost_1_30_0/tools/build/jam_src
   ```

   and type

   ```
   sh ./build.sh
   ```

7. Add the Boost.Jam executable's location to the path by typing

   ```
   PATH=$PATH:/home/watrous/Boost/boost_1_30_0/tools/build/jam_src/bin.linuxx8
   ```

8. In order to build the Boost library follow the instructions in the previously mentioned file

   ```
   boost_1_30_0/tools/build/index.html
   ```

9. Add the location of the Boost libraries to the path by typing

   ```
   PATH=$PATH:/home/watrous/Boost/boost_1_30_0
   ```

10. Check the contents of the path by typing

    ```
    $PATH
    ```

Testing uBLAS

1. In order to build all of the tests for uBLAS, first change to

   ```
   boost_1_30_0/libs/numeric/ublas
   ```

   and then type

   ```
   bjam
   ```

   (Note that the locations of the Boost libraries and the Boost.Jam executable's must be added to the path as discussed in the Installation section of these notes.)

2. In order to run one of the tests, e.g. test1, change to

   ```
   /boost_1_30_0/libs/numeric/ublas/test1/bin/test1/gcc/debug/runtime-link-dyn
   ```

   and then type

   ```
   ./test1
   ```

Appendix: Comparing the Performance of ATLAS and uBLAS

This appendix describes how to compare the performance of ATLAS and uBLAS for normal and sparse matrix multiplication.

Notes

1. These instructions are a record of how these procedures were performed on a particular machine.
2. The paths used in this document are not necessarily appropriate for other machines nor are they necessarily consistent with the current directory structure for RFPK software products.
3. The makefiles for this test assume the directory structure is as described in this document. If that is no longer the case, i.e., if the directory structure is different, then the paths in the makefiles will need to be modified.

Comparing ATLAS and uBLAS on Normal Matrix Multplication

1. Copy the files

   ```
   bench1.cpp
   bench13.cpp
   makefile
   ```

   to the directory

   ```
   boost_1_30_0/libs/numeric/ublas/bench1
   ```

   These files are probably located in a subdirectory of the directory where the XML source for this document is located:

   ```
   ~/dvl/doc/decision/research/blas/normal
   ```

2. The file bench1.cpp is a modified version of the original file with the scale variable increased to make the test run long enough for the differences between the methods to be seen. The file bench13.cpp is a modified version of the original file that now contains calls to the ATLAS library.
3. In order to build the release version of the bench1 program, first change to

```
boost_1_30_0/libs/numeric/ublas/bench1
```

and then type

```
make
```

4. In order to run the bench1 program, type

```
./bench1
```

The timing results can be found in the output from this program in the multiple sections for each matrix size that are preceeded with the headings "bench_3" and "prod (matrix, matrix)".

Comparing ATLAS and uBLAS on Sparse Matrix Multplication

1. Copy the files

```
bench2.cpp
bench23.cpp
makefile
```

to the directory

```
boost_1_30_0/libs/numeric/ublas/bench2
```

These files are probably located in a subdirectory of the directory where the XML source for this document is located:

```
~/dvl/doc/decision/research/blas/normal
```

2. The file bench2.cpp is a modified version of the original file with the scale variable increased to make the test run long enough for the differences between the methods to be seen. The file bench23.cpp is a modified version of the original file that now contains calls to the ATLAS library.
3. In order to build the release version of the bench2 program, first change to

```
boost_1_30_0/libs/numeric/ublas/bench2
```

and then type

```
make
```

4. In order to run the bench2 program, type

```
./bench2
```

The timing results can be found in the output from this program in the multiple sections for each matrix size that are preceeded with the headings "bench_3" and "prod (matrix, matrix)".