

TechReview, Part 2: Program applications with the LAPACK library

Learn how to program and performance tune the LAPACK library linear algebra operations in high-performance computing

Skill Level: Introductory

[Kane Scarlett \(kane@us.ibm.com\)](mailto:kane@us.ibm.com)
Editor, Multicore acceleration
IBM

30 Sep 2008

For application programmers using the IBM Software Development Kit for Multicore Acceleration (SDK), this article explains how to program with the IBM Linear Algebra Package (LAPACK) library using a sample application designed to get an inverse matrix. The article also offers 4 pieces of advice on optimizing LAPACK programs, and it outlines the package's optimized APIs. LAPACK is based on a published standard interface for commonly used linear algebra operations in high-performance computing and other scientific domains.

Introduction

This review gives a quick look at the LAPACK library through the eyes of the original documentation, "LAPACK: Linear Algebra Package Library Programmer's Guide and API Reference" (see [Resources](#)). This article focuses on programming basics with the library, advice on optimizing, and the optimized APIs that come in the library. Use this article with the most current version of IBM SDK for Multicore Acceleration (also known as the Cell/B.E.® SDK), which is the version with fixpack 3.0.0.3.

Building the examples

The following sample application shows you how to use the LAPACK library. The application invokes the DGETRF and DGETRI routines to get an inverse matrix. To build the examples, do this:

1. Cut and paste the Makefile source from [Sample Makefile](#) or from the source [programmer's guide](#) into an editor, and save the file as *Makefile*.
2. Cut and paste the example source code from [Example source code](#) or from the [programmer's guide](#) into an editor, and save the file with a name such as `doc_example.c`.
3. Edit the Makefile to include the name of the example source file that you chose in the previous step. If you did not use `doc_example.c`, substitute the name you chose into the lines that contain `doc_example` and `doc_example.a`.
4. Copy the Makefile and the example source file into your development source directory (for example, `/opt/sandbox/`).
5. From a shell prompt, type:

```
$ cd /opt/sandbox
$ export CELL_TOP=/opt/cell/sdk $ make
```

Sample Makefile

Here's that sample Makefile.

```
#
-----
# (C)Copyright 2007,2008
# International Business Machines Corporation
# All Rights Reserved.
#
# Redistribution and use in source and binary forms, with or
# without modification, are permitted provided that the
# following conditions are met:
#
# Redistributions of source code must retain the above
# copyright
# notice, this list of conditions and the following disclaimer.
#
# Redistributions in binary form must reproduce the above
# copyright notice, this list of conditions and the following
# disclaimer in the documentation and/or other materials
# provided with the distribution.
#
# Neither the name of IBM Corporation nor the names of its
# contributors may be used to endorse or promote products
# derived from this software without specific prior written
# permission.
#
# THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND
```

```

# CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES,
# INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
# MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE
# DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR
# CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
# SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT
# NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES;
# LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION)
# HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN
# CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
# OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS
SOFTWARE,
# EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
#
-----
# PROLOG END TAG zYx

#####
#                               Target
#####

PROGRAM_spu := doc_example
LIBRARY_embed := doc_example.a

#####
#                               Local Defines
#####

INCLUDE = -I$(SDKINC)
LDFLAGS += -L$(SDKLIB)

IMPORTS := -lm -lmc_rand -lmisc

#####
#                               make.footer
#####

# If necessary, change this to the location of make.footer:
include /opt/cell/sdk/buildutils/make.footer

#####

```

Example source code

The following sample application shows you how to use the LAPACK library by invoking the DGETRF and DGETRI routines to get an inverse matrix.

```

#include <stdlib.h>
#include <stdio.h>
#include <lapack.h>

int main( int argc, char *argv[] )
{
    int info = 0;
    double *a;
    int *ipiv;
    int n;
    int i,j;

    if( argc < 2 )
    {
        fprintf(stderr, "%s N\n", argv[0]);
        return 0;
    }
}

```

```

n = atoi(argv[1]);

posix_memalign((void **>(&ipiv), 128, sizeof(int)*n);
posix_memalign((void **>(&a), 128, sizeof(double)*n*n);
if( a == NULL || ipiv == NULL )
{
    fprintf(stderr, "a/ipiv malloc error\n");
    return 0;
}

for( i = 0; i < n; i++ )
{
    for( j = 0; j < n; j++ )
    {
        a[i+j*n]= (drand48() - 0.5f)*4;
    }
}

/*-----Call Cell LAPACK library-----*/
dgetrf_(&n, &n, a, &n, ipiv, &info);
if( info != 0 )
{
    fprintf(stderr, "Call dgetrf error\n");
    goto end;
}

/*-----Query workspace-----*/
double workspace;
int tmp=-1;
int lwork;
double *work;
dgetri_(&n, a, &n, ipiv, &workspace, &tmp, &info);
lwork = (int)workspace;
work = malloc(sizeof(double)*lwork);
if(work == NULL)
{
    printf("work malloc error\n");
    goto end;
}

/*-----Call Cell LAPACK library-----*/
dgetri_(&n, a, &n, ipiv, work, &lwork, &info);
if( info != 0 )
{
    fprintf(stderr, "Call dgetri error\n");
    free(work);
    goto end;
}
printf("Inverse matrix completed!\n");

end:
free(ipiv);
free(a);
return 0;
}

```

Getting in tune

The LAPACK library provides additional features for customizing the library. Use the features to efficiently leverage the available resources to boost your performance.

Some advice about optimizing LAPACK programs:

- Make the matrices 128-byte aligned. Memory access is more efficient when data is 128-byte aligned.
- If you want to use more than 8 SPEs, use the NUMA APIs to interleave the memory of the data on different nodes.
- Supply a matrix in which the number of rows and columns is greater than 1024 for better library performance.
- Realize that if you use `DGETRI` with an odd value of `lda`, the performance is approximately 25 percent worse than when the value of `lda` is even. A fix for this discrepancy should be on the horizon.

Your environment variable is `LAPACK_NUMSPES`. It specifies the number of SPEs to use, and its default value is 16.

Using optimized routines

These routines (which come with the LAPACK) have been optimized to use features of the SPEs:

- `DGETRF`: Computes the LU factorization of a general matrix. LU is a matrix decomposition that writes a matrix as the product of a lower and upper triangular matrix.
- `DGETRI`: Computes the inverse of a general matrix using the LU factorization.
- `DGEQRF`: Computes the QR factorization of a general matrix. QR is a decomposition of the matrix into an orthogonal and a triangular matrix.
- `DPOTRF`: Computes the Cholesky factorization of a symmetric positive matrix. Cholesky is a decomposition of a symmetric positive-definite matrix into a lower triangular matrix and the transpose of the lower triangular matrix.
- `DBDSQR`: Computes the singular value decomposition of a real bi-diagonal matrix using the implicit zero-shift QR algorithm.
- `DSTEQR`: Computes the singular value decomposition of a real symmetric tridiagonal matrix using the implicit QR algorithm.

Resources

Learn

- Use an [RSS feed](#) to request notification for the upcoming articles in this series. (Find out more about [RSS feeds of developerWorks content](#).)
- Get the source document for this article, "[LAPACK: Linear Algebra Package Library Programmer's Guide and API Reference](#)" (IBM, April 2008), to learn how to configure the LAPACK library and how to program applications that use LAPACK on the IBM SDK for Multicore Acceleration, Version 3.0.0.3. The guide contains reference information about APIs for the library, and it contains sample applications showing usage of these APIs.
- Refer to "[TechReview, Part 1: Discover the LAPACK library](#)" (developerWorks, September 2008) for the basic structure of the LAPACK library and to read about the four versions of this interface.
- See [Programming with BLAS: The series](#) (developerWorks, November 2007-July 2008) for a roundup of short guides and longer articles to help you understand and use BLAS. (There are two other tech topic roundups available in this format too: [Programming with ALF](#) and [Programming with DaCS](#).)
- Learn more about Cell/B.E. programming from the developerWorks series:
 - "[Programming high-performance applications on the Cell/B.E. processor](#)"
 - "[PS3 fab-to-lab](#)"
 - "[The little broadband engine that could](#)"
- Refer to the [Cell Broadband Engine documentation](#) section of the IBM Semiconductor Solutions Technical Library for a wealth of downloadable manuals, specifications, and more.
- Sign up for the [developerWorks newsletter](#) and get the latest developer news and Cell/B.E. happenings delivered to your inbox each week. Check *Power Architecture*® when you sign up to receive Cell/B.E. news in your newsletter.

Get products and technologies

- Find the [LAPACK routine](#) you need at the [LAPACK section of the Netlib Repository](#), as well as a [user's guide](#), a [user forum](#), a [quick installation guide](#), and more.
- Look for [ScaLAPACK](#) (or Scalable LAPACK) for a subset of LAPACK routines redesigned for distributed-memory MIMD parallel computers.
- Discover [PLAPACK](#) (the Parallel Linear Algebra Package) for an infrastructure for coding linear algebra algorithms at a high level of abstraction.

- Get your copy of the [IBM SDK for Multicore Acceleration 3.0](#) or browse through the [library of Cell/B.E. documentation](#).
- Find all Cell/B.E.-related articles, discussion forums, downloads, and more at the IBM developerWorks [Cell Broadband Engine resource center](#): your definitive resource for all things Cell/B.E.
- Contact IBM about [custom Cell/B.E.-based or custom-processor based solutions](#).

Discuss

- [Participate in the discussion forum for this content](#).
- Check out the [Cell Broadband Engine Architecture forum](#) to get your technical questions about the processor answered. Juicy problems and answers from the forums are rounded up periodically and highlighted in the ["Forum watch" blog series](#).
- Go to the [Cell Broadband Engine/Power Architecture blog](#) for [news](#), downloads, instructional resources, and event notifications for Cell/B.E. and other Power Architecture-related technologies. You can find the popular ["Forum watch"](#) blog series (Q&A roundup), the ["FixIt"](#) technology updates, and the [Infobomb](#) quick-read technology introductions.

About the author

Kane Scarlett

Kane Scarlett is a technology journalist/analyst with 20 years in the business, working for such publishers as National Geographic, Population Reference Bureau, Miller Freeman, and IDG, and managing, editing, and writing for such august journals as *JavaWorld*, *LinuxWorld*, and of course, developerWorks.

Trademarks

IBM, developerWorks, and Power Architecture are trademarks of IBM Corporation in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc.

Other company, product, or service names may be trademarks or service marks of others.