ClearSpeed

CSXL User Guide

06-UG-1357 1.13.2.13

23 April 2007

ClearSpeed

ClearSpeed Technology, Inc.
3031 Tisch Way, Suite 200
San Jose, CA 95128

Tel: 408-557-2067
Fax: 408-557-9054

Email: info@clearspeed.com
Web: www.clearspeed.com

ClearSpeed Technology plc
3110 Great Western Court
Hunts Ground Road
Bristol BS34 8HP
United Kingdom

Tel: +44 (0)117 317 2000
Fax: +44 (0)117 317 2002

Conventions

Convention	**Description**
`commands`	This typeface means that the command must be entered exactly as shown in the text and the `[Return]` or `[Enter]` key pressed.
`Screen displays`	This typeface represents information as it appears on the screen.
`[Key]` names	Key names appear in the text written with brackets. For example `[Return]` or `[F7]`. If it is necessary to press more than one simultaneously, the key names are linked with a plus (+) sign: Press `[Ctrl]`+`[Alt]`+`[Del]`
Bold-face text	Signal names, instructions and register names are displayed in bold. Selections made via the menu hierarchy of a software application.
Words in *italicized* type	Italics emphasize a point, concept or denote new terms.
ⓘ	This symbol indicates important information or instructions.

Contents

1 Overview

1.1 CSXL on the host 5
1.1.1 How it works 5
1.1.2 Using CSXL 5
1.1.3 Using with multiple Advance boards 5
1.1.4 Compatibility 6
Supported host operating systems 6
Supported host libraries 6
Supported compilers 7
1.2 Examples 7

2 Using CSXL

2.1 Installing CSXL 8
Initializing the environment 8
Resetting the board 8
2.2 Using CSXL with your application 8
2.3 Configuration options 9
2.3.1 Format of configuration file 9
2.3.2 Configuration variables 9
General 9
BLAS library 10
Tracing 10
2.3.3 Example 11
2.4 Using CSXL with host applications 12
2.4.1 Linking with CSXL 12
Linux / gcc 12
Microsoft Windows 12
Using Intel’s MKL 12
2.4.2 Existing applications 12
Application option 13
Renaming the library 13
Preloading the library 13
2.4.3 Using CSXL with MATLAB 13
Microsoft Windows 14
Linux 14

MATLAB compatibility . . . 15
2.4.4 Using CSXL with Mathematica . . . 15
Step 1: Create a symbolic link to intercept MKL . . . 15
Step 2: Configure Mathematica to set up the CSXL environment . . . 15
Step 3: Configure a Mathematica kernel to use CSXL . . . 16
Step 4: Select the accelerated kernel for a notebook . . . 16
Mathematica compatibility . . . 16
Accelerated functions . . . 17

3 Optimizing use of DGEMM
3.1 Intel processors . . . 18
3.2 Performance characterization . . . 18
3.3 Calculating host assist . . . 22

4 CSXL library functions
4.1 BLAS functions . . . 23
4.2 LAPACK functions . . . 23
DGEMM . . . 24
Description . . . 24
Fortran interface . . . 24
C prototype . . . 26
DGESV . . . 27
Description . . . 27
Fortran interface . . . 27
C prototype . . . 28
DGETRF . . . 29
Description . . . 29
Fortran interface . . . 29
C prototype . . . 30

5 Bibliography

1 Overview

CSXL, the ClearSpeed math library, provides accelerated versions of a number of standard math functions for use with the ClearSpeed Advance accelerator boards. The supported routines are a subset of the BLAS (Basic Linear Algebra Subprograms) and LAPACK (Linear Algebra PACKage) libraries. The BLAS and LAPACK libraries are widely used for solving a variety of linear algebra problems. The functions supported by CSXL are:

BLAS Level 3

- DGEMM

LAPACK

- DGETRF
- DGESV

See chapter 4, *CSXL library functions*, on page 23 for details of the functions supported by CSXL.

The library which implements CSXL is called `csxl`. See section 2, *Using CSXL*, on page 8 for more information on using the library with your application.

1.1 CSXL on the host

1.1.1 How it works

CSXL consists of two parts: a set of host libraries; and the code which runs on the Advance board. The functions in the host libraries can be called by an application running on the host in the same way as any other implementation of BLAS and LAPACK. If the problem size is suitably large, the CSXL library transparently offloads some or all of the processing to the Advance board. Where no performance benefit would be obtained by using the Advance board, or the function is not supported by CSXL, the routine will be executed on the host by calling a standard host library such as MKL or ACML.

Figure 1.1 shows a typical application calling a host BLAS library and an application making calls via CSXL.

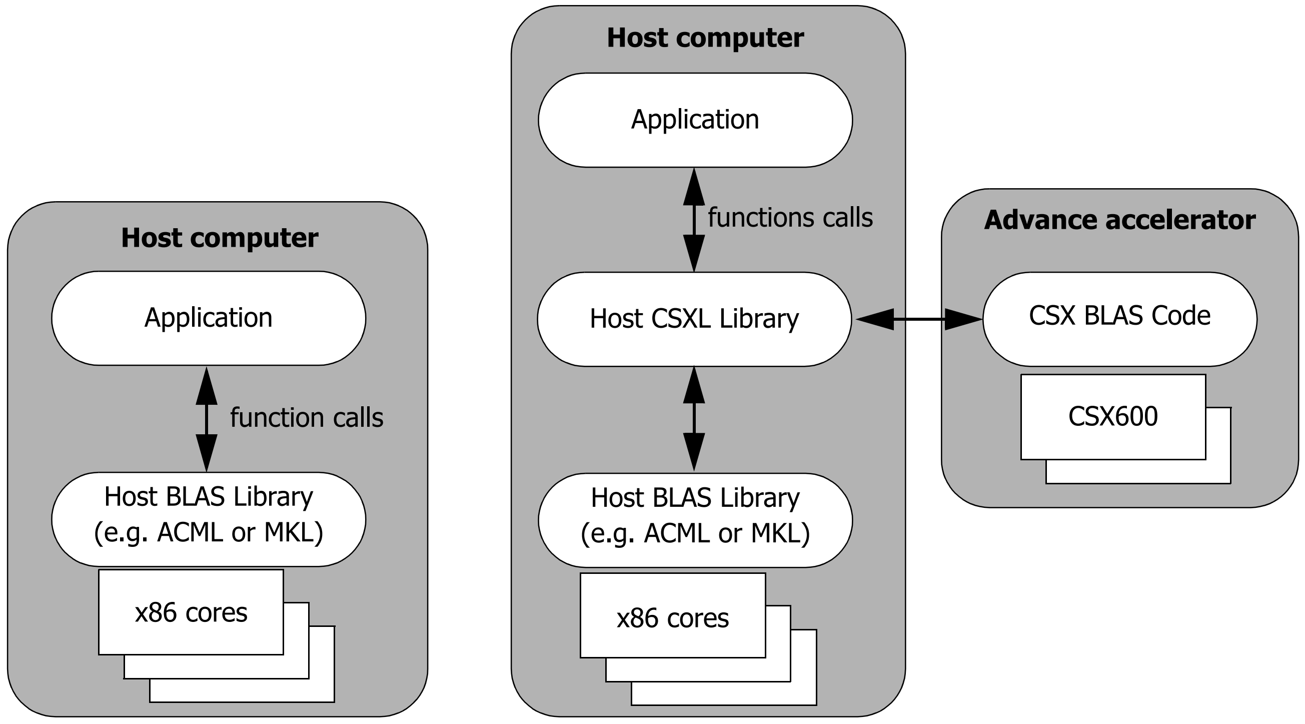
In Figure 1.1, the CSXL library distributes work between the x86 and CSX600 processors. The decision on where data is to be processed is based on factors such as the problem size. When the host CSXL library calls a function which is accelerated on Advance board, the data to be passed as parameters to that function have to be copied across to the Advance board. When the CSX600 processors have completed the function the result is copied back to the host and the CSXL library returns control to the calling application. As long as the time taken to transfer data and process it on the Advance board is less than the time it would have taken to process it on the host, it is beneficial to process it on the Advance board.

1.1.2 Using CSXL

The behavior of the CSXL library can be fine tuned by means of a number of configuration variables (see 2.3, *Configuration options*, on page 9).

1.1.3 Using with multiple Advance boards

The CSXL library supports the use of a single board per process. Multiprocess applications (such as applications using multiple MPI processes per node) can exploit multiple cards per node transparently on a single board per process basis. Furthermore, binding of a specific board to a specific process can be accomplished by setting the `LLDINST` environment variable to the desired board instance number for that process.



Host library only (unaccelerated) Accelerated by CSXL library

Figure 1.1 Host application calling BLAS functions

1.1.4 Compatibility

Supported host operating systems

The following operating systems are supported by the ClearSpeed software.

- RHEL 3 (32 bit)
- RHEL 4 (32 and 64 bit)
- SLES 9.3 (64 bit)
- Windows XP (32 bit)

Supported host libraries

The following host libraries have been tested for compatibility with CSXL:

- ACML
- MKL
- ATLAS

Supported compilers

The use of CSXL has been tested with the following host compilers:

- For Linux: GNU compiler collection and compatibles
- For Windows XP: Visual C++ 2005

1.2 Examples

A number of examples are provided with the CSXL library. These are in the `examples/csxl` directory of the installation. There is some brief documentation of the examples in the `README.txt` file in that directory.

2 Using CSXL

This chapter describes how CSXL is used with various types of application programs.

2.1 Installing CSXL

To install the CSXL library you will need to download the "Base Package" software from the ClearSpeed customer support website. This includes detailed installation instructions. This package contains the CSXL library and the runtime software.

Before installing the CSXL library, you will need to install the runtime software. The runtime software includes a device driver, which provides a software interface to the Advance board, and supporting runtime libraries.

Initializing the environment

After installing the runtime and CSXL software, you will need to setup your environment. A script is provided to do this. You should run this script in any new command window before using the ClearSpeed software. This can be done as follows.

For Linux:

```
source /opt/clearspeed/csx600_m512_le/bin/bashrc
```

For Microsoft Windows:

```
C:\Program Files\clearspeed\csx600_m512_le\bin\setup_env.bat
```

Resetting the board

Before using the Advance board for the first time, it must be reset. This can be done with the `csreset` command:

```
csreset -Av
```

See the *CSX600 Runtime Software User Guide* [5] for more information on the `csreset` command options.

It may also be necessary to reset the board if program execution is terminated abnormally (for example by using [`Ctrl`]+[`C`]).

2.2 Using CSXL with your application

The library which implements the BLAS functions in CSXL is called `csxl`. The library file is:

• `libcsxl.so`	for Linux
• `csxl.dll`	for Microsoft Windows

If the software has been installed in the default location, the library files will be in:

• `/opt/clearspeed/csx600_m512_le/lib/`	for Linux
• `C:\Program Files\clearspeed\csx600_m512_le\bin`	for Windows

These libraries can be linked with a host application to use the functions in the CSXL library. For details of the supported functions see chapter 4, *CSXL library functions*, on page 23.

To use CSXL with a precompiled application see section 2.4, *Using CSXL with host applications*, on page 12.

2.3 Configuration options

The behavior of the CSXL library can be controlled by a number of configuration variables. They can either be set in a configuration file or as environment variables. A value specified as an environment variable overrides the value in the configuration file.

Using a configuration file is convenient on clusters with shared file systems where it may not be easy to pass environment variables to all the nodes or processes.

The following configuration variables must, in general, be specified for the CSXL library to work:

`CS_HOST_BLAS`

`LD_LIBRARY_PATH` (Linux only)

If you use a configuration file, then the environment variable `CSXL_CONFIG_FILE` must be set to specify the full path and file name of the configuration file.

2.3.1 Format of configuration file

A CSXL configuration file consists of a series of lines defining values for configuration variables (defined below). These lines are of the form:

```
variable=value
```

Lines starting with `#` are treated as comments and are ignored. Although a variable can be defined multiple times in the configuration file, only the *first* definition is used.

For example, the host BLAS library could be specified as follows:

```
# Define host BLAS library to use
CS_HOST_BLAS=/opt/lib/ACML2.5/gnu64/lib/libacml.so
```

The CSXL library reads the configuration file and then the environment variables. This means that the values specified in the configuration file can be overridden by defining an environment variable of the same name.

2.3.2 Configuration variables

The configuration variables can be divided into the following groups: general configuration, BLAS library specific variables and control of program tracing. These are described in detail below.

General

`CSXL_CONFIG_FILE`

This variable specifies the name of the CSXL configuration file. The default value is empty, which means no configuration file will be used. This variable can only be specified as an environment variable.

```
CSXL_CONFIG_FILE=/home/jsp/csxl.cfg
```

`CS_HOST_BLAS`

Use this variable to specify which host libraries to search if the Advance board does not accelerate a function. The default value is empty. If this variable is not defined, you get an error message.

This variable can specify a list of library files (in the same way as `LD_LIBRARY_PATH`) to be searched for a function. This enables you to specify a number of host libraries which between them implement the full set of BLAS/

LAPACK functions. For Linux, the items in the list are separated by colons (‘`:`’). For Microsoft Windows, the items in the list are separated by semicolons (‘`;`’).

Symbols are resolved from the list of libraries in the order they appear in `CS_HOST_BLAS`. In the following example, the `dgemm_` function is resolved in the ClearSpeed BLAS library, but `lsame_` is resolved in ACML. For information on which BLAS libraries are supported, see , *Supported host libraries*, on page 6.

```
CS_HOST_BLAS=/var/tmp/clearspeed/libacml.so
```

As another example, Goto BLAS implements all BLAS routines and some LAPACK routines from version 1.0 onwards. However, ACML offers full coverage of LAPACK. If you want to use the functions included in Goto BLAS and access full LAPACK capability from ACML, you can do this by defining `CS_HOST_BLAS` as:

```
CS_HOST_BLAS=/var/tmp/clearspeed/libgoto.so:/var/tmp/clearspeed/libacml.so
```

`LD_LIBRARY_PATH`

This is a standard environment variable for Linux which tells the dynamic linker where to find any dynamically linked libraries. The location of the CSXL library should be added to this variable, before any existing library paths to ensure that the CSXL library is searched before the other host libraries.

```
LD_LIBRARY_PATH=/opt/clearspeed/csx600_m512_le/lib:$LD_LIBRARY_PATH
```

This variable does not exist in Windows systems.

BLAS library

The following variables are specific to the BLAS functions in CSXL.

`CS_BLAS_HOST_ASSIST_PERCENTAGE`

This variable controls how work will be shared between the host and the Advance board(s). This is specified as a percentage. For example, if you set it to 20 then in principle 20% of a BLAS call will be performed on the host and 80% on the Advance board. The value is used as a “hint” to the CSXL library about the relative efficiency of the host and accelerator. The actual split of work between the host and the card depends on the sizes of the matrices being processed. The Advance board will only process matrices of particular sizes. Where the inputs do not match this then the problem will be divided up to provide a good size for the accelerator and the remaining portions will be processed on the host.

The default, if the variable is not defined, is to do as much processing on the card as possible. See also section 3.3, *Calculating host assist*, on page 22.

```
CS_BLAS_HOST_ASSIST_PERCENTAGE=20
```

Tracing

The following variables control the output of information which can be used to trace the calls made to the CSXL library. This can be useful for debugging or optimizing performance.

`CSXL_TRACING`

If this variable is defined, with any value, `then` tracing is enabled. By enabling tracing you can:

- Track calls to functions supported by CSXL.
- View the parameters for traced library functions such as the length of vectors or the number of rows and columns of a matrix.

- Check the timing of function calls. The final trace parameter `_t` is the time spent in the function.

The trace output consists of a series of lines, in square brackets, listing the function name and parameters, and the execution time in microseconds of the function. For example:

```
[ TRACE dgemm_ transa=N transb=N m=768 n=768 k=768 lda=778 ldb=778 ldc=778 ta_len=1
tb_len=1 _t=38727 ]
```

The keyword `_off_` appears in the trace line if the function call has been identified as a candidate for acceleration on the Advance board. Whether or not the function is accelerated depends on a number of factors such as the size and shape of the matrices.

The trace output can be sent to the standard output or to a file. See `CSXL_TRACING_OUTPUT` and `CSXL_CONSOLE_TRACING` for details.

Note that turning on tracing will affect the performance of the CSXL library.

The default, if the variable is not defined is to not generate trace output.

```
CSXL_TRACING=1
```

CSXL_TRACING_OUTPUT

When profiling is enabled (by defining `CSXL_TRACING`), this variable specifies the name of the output file. Normally this file name is used as a base to which the host name, process id and thread id are appended. This allows the trace output to be generated from multiple processes across multiple nodes of a cluster. For example, if `CSXL_TRACING_OUTPUT` is set to `csxl_logfile`, then the output filename will be of the form `csxl_logfile.`*`hostname`*`.`*`pid`*`.`*`tid`*.

If the variable `CSXL_CONSOLE_TRACING` is defined then the filename will be used unchanged. Note that this should only be done when a single instance of CSXL is running.

If the variable `CSXL_TRACING_OUTPUT` is not defined then output is sent to the standard output (typically the terminal window). If the specified file cannot be opened for output then, again, the output will go to standard out.

```
CSXL_TRACING_OUTPUT=/home/fred/trace.out
```

CSXL_CONSOLE_TRACING

If this variable is defined then it is assumed that output is to the console or to the exact file name in `CSXL_TRACING_OUTPUT`. If tracing to the console (`CSXL_TRACING_OUTPUT` undefined), this variable *must* be defined.

```
CSXL_CONSOLE_TRACING=1
```

2.3.3 Example

A typical setup, optimized for LINPACK on a four-core machine, is as follows:

```
CSXL_TRACING=0
CSXL_CONSOLE_TRACING=1
CSXL_TRACING_OUTPUT=/proc/self/fd/1
CS_BLAS_HOST_ASSIST_PERCENTAGE=25
CS_HOST_BLAS=/lib/libgoto_opt-64_1024-r0.97.so
```

2.4 Using CSXL with host applications

There are a number of different methods for using the CSXL library with a host application, depending on the nature of the host code. If it is a program that you are developing then your code can simply be linked with the CSXL library. Existing applications can use a variety of techniques to use CSXL. These different methods of using CSXL are described below, together with the two concrete examples of MATLAB and Mathematica.

2.4.1 Linking with CSXL

In order to link a program with the CSXL library you will need to specify the library to use (`csxl`) and the path to the library. The details of how to do this depend on the development tools being used. A couple of examples are provided below.

Linux / gcc

If you are using the `gcc` compiler for the host application then the library name is specified on the command line using the `-l` option. The library path can either be added to the `LD_LIBRARY_PATH` environment variable or specified on the command line using the `-L` option.

If you link with `csxl` then you will also need to link with the `pthread` and `g2c` system libraries.

A typical command line, for a program that only calls DGEMM, might be:

```
gcc -lm -lg2c -lpthread -L/opt/clearspeed/csx600_m512_le/lib -lcsxl -o dgemm dgemm.c
```

When the program is run, the `LD_LIBRARY_PATH` variable must be set to allow the dynamic library to be found. For example:

```
LD_LIBRARY_PATH=/opt/clearspeed/csx600_m512_le/lib:$LD_LIBRARY_PATH ./dgemm
```

Microsoft Windows

If you are using Visual Studio 2005 to develop the host code then the library (`csxl.dll`) needs to be added to the list of libraries to be linked in the project. To do this select **References...** from the **Project** menu. The library path needs to be added project's properties. For more details see the Visual Studio documentation.

If you are using the Microsoft tools from the command line, then the library path can either be added to the `LIB` environment variable or specified on the command line using the `/LIBPATH` command line option. The library file is given as one of the options to the command line.

Using Intel's MKL

When using MKL as your host libraries, depending on the version of MKL being used you may need to include them in the link line after `-lcsxl`.

2.4.2 Existing applications

To link the CSXL library with a precompiled application there are a number of possibilities. It may be necessary to try several of these to find the one that works best with a given application.

Application option

In the simplest case, an application may provide a way of specifying the library which is to be used. For example, MATLAB uses the `BLAS_VERSION` environment variable to specify the library which should be used for BLAS and LAPACK functions (see section 2.4.3, *Using CSXL with MATLAB*, on page 13). Check your application's documentation to see if there is a similar mechanism available.

Renaming the library

In some cases it is necessary to "fool" the application into using the CSXL library. This can be done by creating a symbolic link (Linux) or copy (Windows) of the CSXL library file with the same name as the library the application is built to link with. This method is used for Mathematica, for example (see section 2.4.4, *Using CSXL with Mathematica*, on page 15).

Before doing this you should rename the original library as a backup copy.

For example, if the application has been built to use MKL then a file with the appropriate name can be created by doing the following:

For Linux:

```
mv libmkl_lapack64.so libmkl_lapack64.so.original
mv libmkl.so libmkl.so.original
ln -s libcsxl.so libmkl_lapack64.so
ln -s libcsxl.so libmkl.so
```

For Windows:

```
rename mkl_lapack64.dll mkl_lapack64.dll.original
rename mkl.dll mkl.dll.original
copy csxl.dll mkl_lapack64.dll
copy csxl.dll mkl.dll
```

Preloading the library

If you are using Linux then it is possible to force the dynamic linker to load a library before it attempts to resolve any symbols. This is done using the environment variable `LD_PRELOAD`. If the first library loaded is CSXL then this will be used for any entry points that it satisfies. The normal BLAS library will then be loaded and used for the remaining functions.

For example:

```
CS_HOST_BLAS=/opt/intel/mkl/8.0.1/lib/em64t/libmkl.so:/opt/intel/mkl/8.0.1/lib/
em64t/libmkl_lapack64.so
LD_PRELOAD=/opt/clearspeed/csx600_m512_le/lib/libcsxl.so
```

2.4.3 Using CSXL with MATLAB

The MATLAB application uses an environment variable, `BLAS_VERSION`, to allow the user to specify which host BLAS library should be used to provide the BLAS functions. This makes it very easy to use with CSXL. You simply need to set this variable to point to the CSXL BLAS library and set the `CS_HOST_BLAS` variable to the host BLAS library (typically the library that `BLAS_VERSION` referenced previously).

After this, MATLAB can be run as normal and any calls to supported BLAS functions will be accelerated.

Details of doing this for Microsoft Windows and Linux operating systems are included below.

If you also wish to use the accelerated LAPACK functions with MATLAB then you will need to use the library renaming technique described above.

Microsoft Windows

The following describes how to use CSXL with MATLAB on the Windows XP operating system. This assumes that the environment for running ClearSpeed software has already been set up by running the batch file:

```
C:\Program Files\clearspeed\csx600_m512_le\bin\setup_env.bat
```

The following description assumes the software is installed as follows:

- MATLAB is installed at:
  `C:\Program Files\matlab\`*`version`*
  where *`version`* is something like `r14sp2` or `r2006a`
- The ClearSpeed software has been installed in the default location:
  `C:\Program Files\clearspeed\csx600_m512_le`

To run MATLAB using CSXL acceleration:

1. Open a DOS shell window and go to the directory where you want to run MATLAB.
2. Ensure the following ClearSpeed environment variables have been set either by typing them on the command line or by creating a batch file:
   ```
   set CS_HOST_BLAS=C:\Program Files\matlab\r14sp2\bin\win32\mkl.dll
   set BLAS_VERSION=C:\Program Files\clearspeed\csx600_m512_le\bin\csxl.dll
   ```
   Note: The path of the host BLAS library may vary depending on the version of MATLAB. The paths specified in these environment variables must not be enclosed in double quotes.
3. Run MATLAB as normal. For example:
   ```
   matlab -nojvm -r csxdgemm
   ```
   where `csxdgemm.m` is an M-file in the local directory which runs DGEMM over a range and plots the result.

Linux

The following describes how to use CSXL with MATLAB on a Linux operating system. See the compatibility information on page 15 for information on which versions of MATLAB can be used with particular Linux distributions.

The following description assumes that the software has been installed in the standard locations, and that the environment for running ClearSpeed software has already been set up by running the `bashrc` script in the software installation; for example:

```
source /opt/clearspeed/csx600_m512_le/bin/bashrc
```

To run MATLAB using CSXL acceleration:

1. Set the `BLAS_VERSION` and `CS_HOST_BLAS` environment variables as described above:
   ```
   export BLAS_VERSION=/opt/clearspeed/csx600_m512_le/lib/libcsxl.so
   export CS_HOST_BLAS=/usr/local/matlabR14/bin/glnx86/libmkl.so
   ```
   The values of these variables will vary depending on the version of MATLAB, which host BLAS library is being used, and so on.
2. Run MATLAB as normal. For example:
   ```
   matlab -nojvm -r csxdgemm
   ```
   where `csxdgemm.m` is an M-file in the local directory which runs DGEMM over a range and plots the result.

MATLAB compatibility

The versions of MATLAB supported by the CSXL library are shown in Table 1.

	MATLAB 7.1 (R14 SP3)	MATLAB 7.2 (R2006a)
RHEL 3 (IA32)	✓	✗
RHEL 4 (IA32)	✗	✓
RHEL 4 (AMD64/EM64T)	✗	✓
SLES 9.3 (AMD64/EM64T)	✓[a]	✗
Windows XP (IA32)	✓	✓

Table 1 MATLAB compatibility

a. Need to replace the library provided with MATLAB (`libstdc++.so.5`) with a symbolic link to the standard system library `/usr/lib64/libstdc++.so.5.0.6`

2.4.4 Using CSXL with Mathematica

The following describes how to use CSXL with Mathematica on a Linux operating system. The CSXL library is not compatible with the current version of Mathematica on Microsoft Windows.

Mathematica 5.2 uses Intel's Math Kernel Library (MKL) as its default standard math library. MKL is a multi-component library with separate shared object files for BLAS, LAPACK, and other sets of math functions.

Step 1: Create a symbolic link to intercept MKL

In order to make Mathematica use the CSXL library it is necessary to create a symbolic link with the name of the MKL library which points to the CSXL BLAS library:

1. Change to the directory:
```
cd /opt/clearspeed/csx600_m512_le/lib
```
2. Create a symbolic link
```
ln -s libcsxl.so libmkl_lapack64.so
ln -s libcsxl.so libmkl.so
```

Step 2: Configure Mathematica to set up the CSXL environment

In the following instructions, the path to the Mathematica installation is referred to as `$MATHEMATICA`.

To enable access to the CSXL library from within Mathematica, perform the following steps once:

1. Log in as the user who installed Mathematica. This is required because Mathematica can only be configured by the person who installed it.
2. Change to the `$MATHEMATICA/Executables` directory:
```
cd $MATHEMATICA/Executables
```
3. Copy the file `math` to a new name, for example `math_csx`:
```
cp math math_csx
```
4. Open the file `math_csx` in a text editor and find the line:
```
exec "${MathKernel}" "$@"
```

Modify the script to load the CSXL library before starting Mathematica by adding the following lines before the `exec` line (this example is specific to SLES 9.3):

```
__PFIX__=/usr/local/Wolfram/Mathematica/5.2/SystemFiles/Libraries/Linux-x86-64
export CS_HOST_BLAS=$__PFIX__/libmkl_lapack32.so:↳
                    $__PFIX__/libmkl_lapack64.so:$__PFIX__/libmkl.so:↳
                    $__PFIX__/libmkl_vml_p4n.so
# Pick up the libraries in CSXL first then the standard ones
export LD_LIBRARY_PATH=/opt/clearspeed/csx600_m512_le/lib/:$LD_LIBRARY_PATH
exec "${MathKernel}" "$@"
```

Where the symbol ↳ indicates that a long line has been broken to fit the page. This should be entered as a single line.

Note: For 32-bit Linux systems, remove `-x86-64` from the `__PFIX__` definition.

5. Use the shell script provided to set up the environment variables:

```
source /opt/clearspeed/csx600_m512_le/bin/bashrc
```

Step 3: Configure a Mathematica kernel to use CSXL

To configure the Mathematica kernel:

1. Start the Mathematica front end in the normal way:

```
mathematica
```

2. Create a kernel entry for the new `math_csx` script. From the **Kernel** menu, select **Kernel Configuration Options...**
3. Click the **Add...** button of the "Properties" dialog box
4. Enter a value for **Kernel Name**, such as "ClearSpeed"
5. Enter "`math_csx`" in the **Shell Command to Launch Kernel** field
6. Click **OK**
7. Click **OK** in the "Properties" dialog

Step 4: Select the accelerated kernel for a notebook

Once a notebook has been opened or created in Mathematica, the CSXL accelerated kernel can be selected from the **Kernel** menu as follows:

1. Select the kernel from the **Notebook's Kernel** menu item.
2. Start the kernel from the **Start Kernel** menu item.
3. Evaluate the notebook by selecting **Evaluate Notebook** from the **Evaluation** menu item.

Mathematica compatibility

The applications and operating systems supported by the CSXL library are shown in Table 2.

	Mathematica version 5.2

Table 2 Mathematica compatibility

RHEL 3 (IA32)	✓
RHEL 4 (IA32)	✓
RHEL 4 (AMD64/EM64T)	✓
SLES 9.3 (AMD64/EM64T)	✓
Windows XP (IA32)	✗

Table 2 Mathematica compatibility

Accelerated functions

The following Mathematica functions are candidates for acceleration by the CSXL library. The degree of acceleration seen is very dependent on the data, for example, the size and shape of the matrices. The greatest acceleration will be achieved for `Dot[]` which calls DGEMM directly.

- `Dot[]`
- `Det[]`
- `LUDecomposition[]`
- `Inverse[]`
- `LinearSolve[]`

3 Optimizing use of DGEMM

This chapter includes graphs that help you analyze the performance of various DGEMM sizes. The following files allow you to generate performance graphs:

- `/opt/clearspeed/csx600_m512_le/examples/csxl/matlab/csxdgemm.m`

  This MATLAB script multiplies two identical, square matrices of varying size and plots the resulting GFLOPS performance on the screen.
- `/opt/clearspeed/csx600_m512_le/examples/csxl/mathematica/csxdgemm.nb`

  This function demonstrates the performance of transparent acceleration of matrix multiply with Mathematica by computing the GFLOPS of a matrix multiply of a square matrix of rank n.

3.1 Intel processors

The performance of the CSXL library may be improved on Intel processors by switching off hyperthreading. This can be typically done via the BIOS. This is a quote from the Release Notes for *Intel Optimized LINPACK Benchmark 2.1.2 for Linux*:

> Intel Optimized Linpack Benchmark is threaded to effectively use multiple processors. Therefore, in MP systems, best performance will be obtained with hyperthreading turned off. This insures that the operating system assigns threads to physical processors only.

3.2 Performance characterization

The following graphs show examples of the performance that can be obtained when using DGEMM to multiply two matrices of different shapes and sizes. The performance graphs were generated on a system with two dual-core Intel Pentium 4, 2.80 GHz processors and one Advance accelerator board.

Figure 3.1 shows a typical performance graph for multiplying square matrices (m=n=k). For the DGEMM implementation in the CSXL library, the best performance is obtained when n and m are multiples of 192 and k is a multiple of 288. Sizes of m=n=k for the results below are chosen to be multiples of 576. Intel's MKL library is used as the host library.

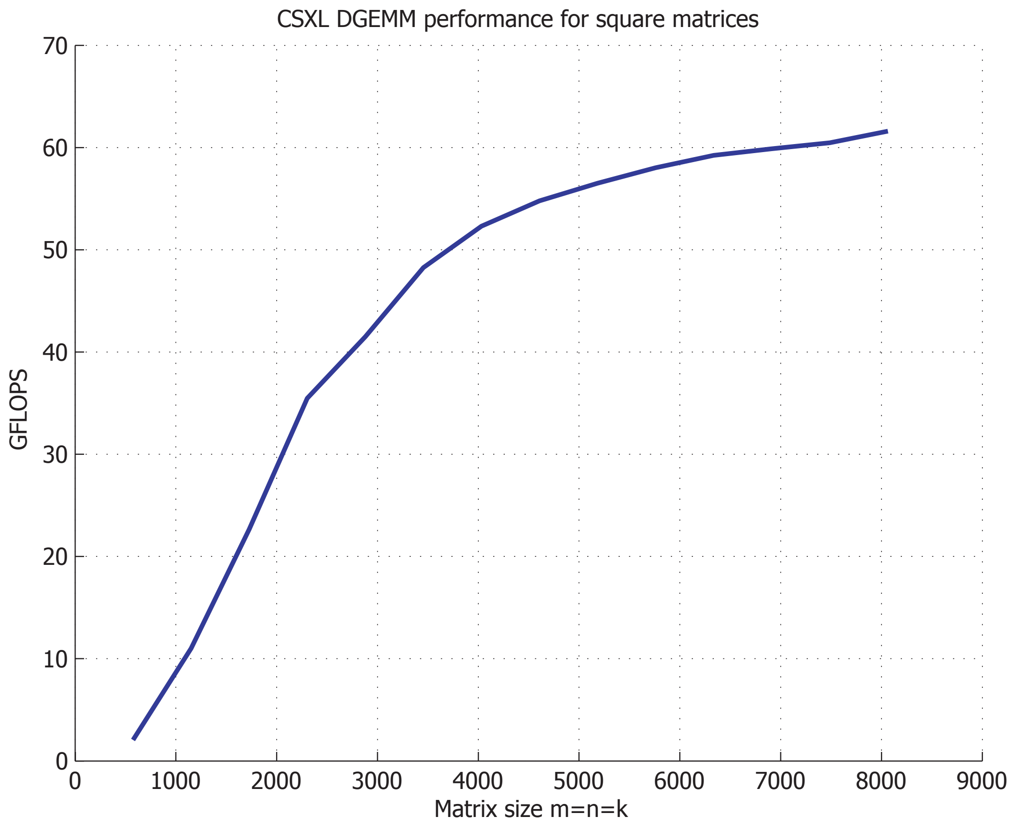


Figure 3.1 CSX600 square DGEMM performance

Figure 3.3 shows DGEMM performance with matrix shapes that are encountered when running the LU factorization phase in Linpack. These matrix multiplications account for a significant percentage of the time in the factorization. Typically m=n, while k is much smaller. This is illustrated in Figure 3.2 where A is high and narrow, B is wide and short, C is large and square.

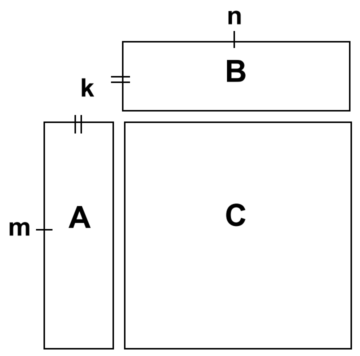


Figure 3.2 Illustration of CSX600 LU-shape DGEMM

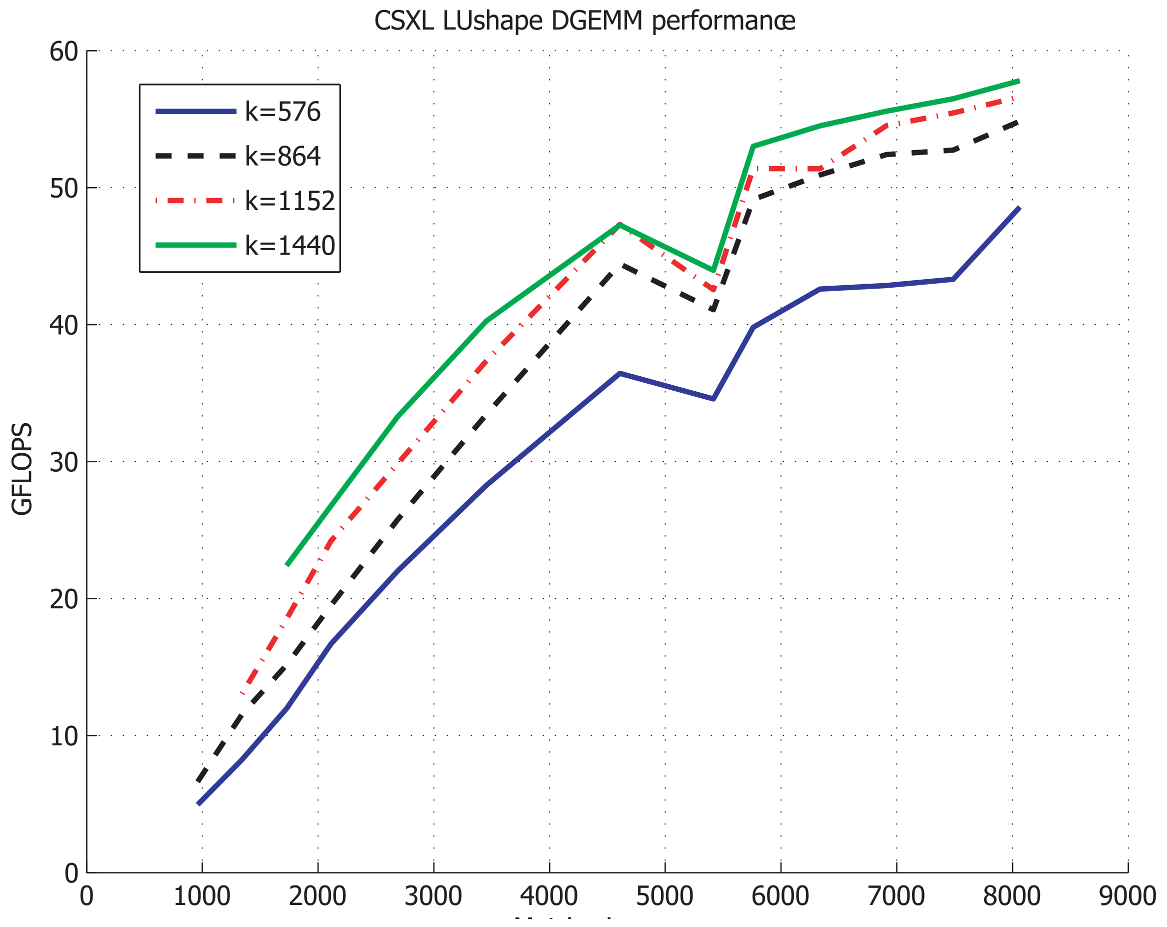


Figure 3.3 CSX600 LU-shape DGEMM performance

In Figure 3.5, we see the performance of DGEMM when m=k and n is much smaller. This shape is also encountered as part of the LU factorization in Linpack. This is illustrated in Figure 3.4 where A is large and square, B is high and narrow, C is high and narrow.

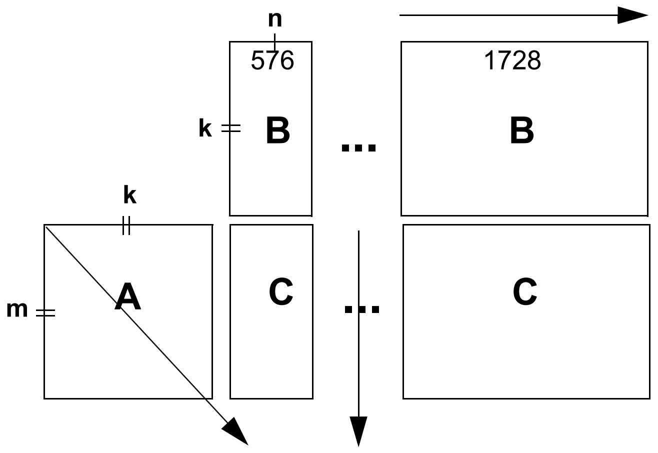


Figure 3.4 Illustration of CSX600 square matrix A DGEMM

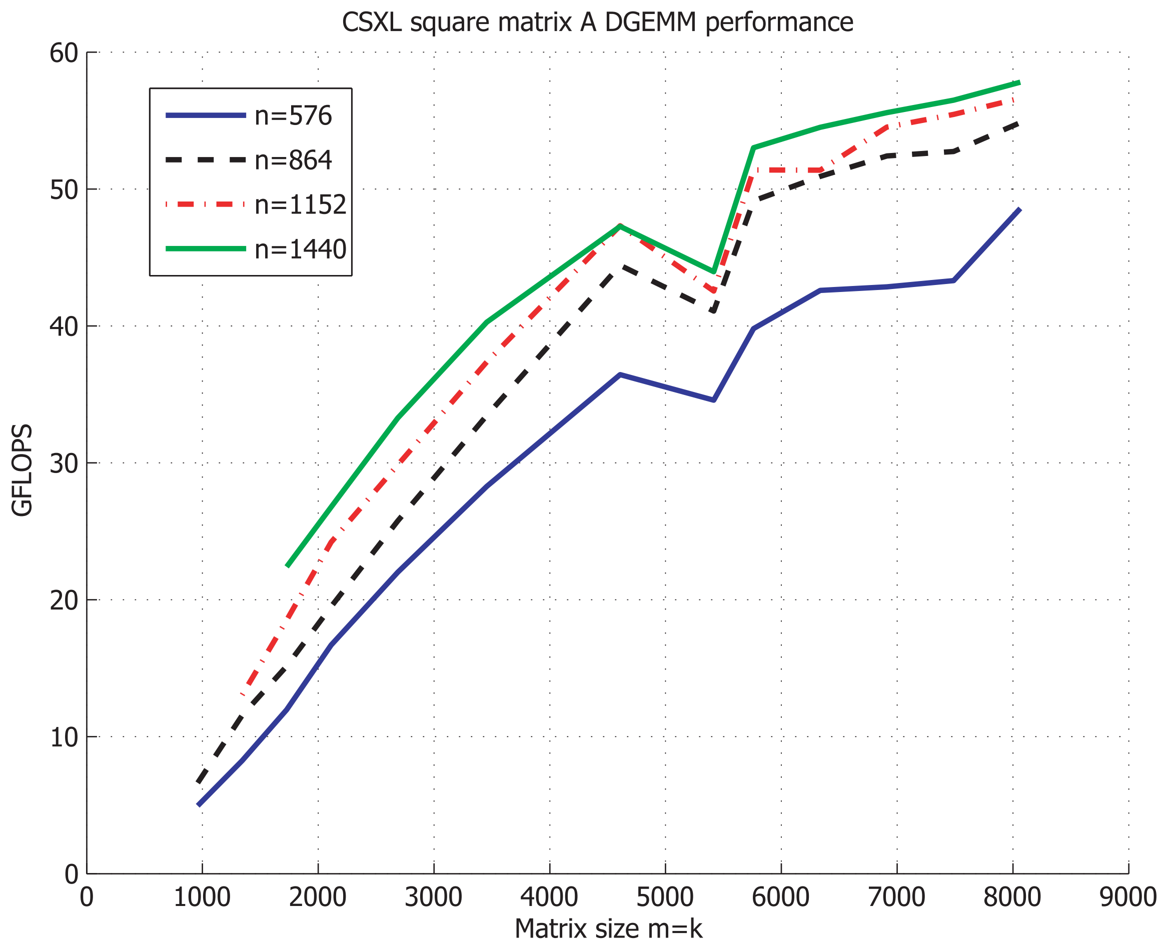


Figure 3.5 CSX600 square matrix A DGEMM performance

3.3 Calculating host assist

This section describes how to balance the performance of the host and the Advance board by partitioning the workload between them. The following instructions take you through the steps needed to calculate the host assist percentage value on Linux platforms.

1. Compile the program `dgemm.c` which is provided in the `examples/csxl/dgemm_example` directory of the installation. This should simply be a matter of running `make` (Linux) in that directory.
2. Setup the CSX runtime and CSXL packages:

   ```
   source /opt/clearspeed/csx600_m512_le/bin/bashrc
   csreset -Av
   export CSXL_TRACING=1
   unset CSXL_TRACING_OUTPUT
   export CSXL_CONSOLE_TRACING=1
   export CS_HOST_BLAS=/usr/lib/libacml.so
   ```
3. Take the following two measurements:
   a. Firstly, measure the capability of the host (*hosttime*):

      ```
      export CS_BLAS_HOST_ASSIST_PERCENTAGE=100
      ./a.out 15360 15360 1152
      ```
      A description of the trace data that is displayed can be found in the section **Tracing**, on page 10.

      The last line will show the time in microseconds. For example:

      ```
      ... _t=35062405 ]
      ```
   b. Next, measure the capability of the installed card (*cardtime*):

      ```
      export CS_BLAS_HOST_ASSIST_PERCENTAGE=0
      ./a.out 15360 15360 1152
      ```
      The last line will show the time in microseconds. For example:

      ```
      ... _t=11226450 ]
      ```
4. Use the above figures (*hosttime*=`35062405` and *cardtime*=`11226450)` to derive an approximation of the require host assist fraction:

   host assist percentage = *cardtime* / (*cardtime* + *hosttime*) =11226450 / 462888555 = 24%
5. You can then set the configuration variable to this value:

   ```
   CS_BLAS_HOST_ASSIST_PERCENTAGE=24
   ```
   You may wish to experiment with this value when running your application to see if a slightly larger or smaller value gives better performance.

Note: The above timings were made with a four-core host. Timings and host assist will vary depending on your system.

4 CSXL library functions

This section provides a definition of the functions implemented in the CSXL library. The functions are documented in alphabetical order.

4.1 BLAS functions

DGEMM 24

4.2 LAPACK functions

DGESV 27
DGETRF 29

DGEMM

Description

DGEMM performs one of the matrix-matrix operations

```
C := alpha*op(A)*op(B) + beta*C
```

where op(X) is one of:

- op(X) = X (identity)
- op(X) = X' (transpose)

alpha and beta are scalars, and A, B and C are matrices, with op(A) an m by k matrix, op(B) a k by n matrix and C an m by n matrix.

Fortran interface

```
      SUBROUTINE DGEMM ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B, LDB,
     $                   BETA, C, LDC )
*     .. Scalar Arguments ..
      CHARACTER*1        TRANSA, TRANSB
      INTEGER            M, N, K, LDA, LDB, LDC
      DOUBLE PRECISION   ALPHA, BETA
*     .. Array Arguments ..
      DOUBLE PRECISION   A( LDA, * ), B( LDB, * ), C( LDC, * )
```

Parameters

`CHARACTER*1 TRANSA`

On entry, `TRANSA` specifies the form of op(A) to be used in the matrix multiplication as follows:

- If `TRANSA` = 'N' or 'n', op(A) = A
- If `TRANSA` = 'T' or 't', op(A) = A'
- If `TRANSA` = 'C' or 'c', op(A) = A'

Unchanged on exit.

`CHARACTER*1 TRANSB`

On entry, `TRANSB` specifies the form of op(B) to be used in the matrix multiplication as follows:

- If `TRANSB` = 'N' or 'n', op(`B`) = B
- If `TRANSB` = 'T' or 't', op(`B`) = B'
- If `TRANSB` = 'C' or 'c', op(`B`) = B'

Unchanged on exit.

`INTEGER M`

On entry, `M` specifies the number of rows of the matrix op(A) and of the matrix C. `M` must be at least zero.

Unchanged on exit.

INTEGER N

On entry, N specifies the number of columns of the matrix op(B) and the number of columns of the matrix C. N must be at least zero.

Unchanged on exit.

INTEGER K

On entry, K specifies the number of columns of the matrix op(A) and the number of rows of the matrix op(B). K must be at least zero.

Unchanged on exit.

DOUBLE PRECISION ALPHA

On entry, ALPHA specifies the scalar alpha.

Unchanged on exit.

DOUBLE PRECISION A()

A is an array of dimensions (LDA, ka), where ka is k when TRANSA = 'N' or 'n', and is m otherwise.

Before entry with TRANSA = 'N' or 'n', the leading m by k part of the array A must contain the matrix A, otherwise the leading k by m part of the array A must contain the matrix A.

Unchanged on exit.

INTEGER LDA

On entry, LDA specifies the first dimension of A as declared in the calling program. When TRANSA = 'N' or 'n' then LDA must be at least max(1, m), otherwise LDA must be at least max(1, k).

Unchanged on exit.

DOUBLE PRECISION B()

B is an array with dimensions (LDB, kb), where kb is n when TRANSB = 'N' or 'n', and is k otherwise.

Before entry with TRANSB = 'N' or 'n', the leading k by n part of the array B must contain the matrix B, otherwise the leading n by k part of the array B must contain the matrix B.

Unchanged on exit.

INTEGER LDB

On entry, LDB specifies the first dimension of B as declared in the calling program. When TRANSB = 'N' or 'n' then LDB must be at least max(1, k), otherwise LDB must be at least max(1, n).

Unchanged on exit.

DOUBLE PRECISION BETA

On entry, BETA specifies the scalar beta. When BETA is supplied as zero then C need not be set on input.

Unchanged on exit.

DOUBLE PRECISION C()

C is an array with dimensions (LDC, n)

Before entry, the leading m by n part of the array C must contain the matrix C, except when beta is zero, in which case C need not be set on entry.

On exit, the array C is overwritten by the m by n matrix (alpha*op(A)*op(B) + beta*C).

INTEGER LDC

On entry, LDC specifies the first dimension of C as declared in the calling program. LDC must be at least max(1, m).

Unchanged on exit.

C prototype

When calling the library from C, the following names can be used: dgemm, dgemm_ or DGEMM. In all cases the prototype is:

```
extern void dgemm_(char *transa, char *transb,
                   int *m, int *n, int *k,
                   double *alpha, double *A, int *lda,
                   double *B, int *ldb, double *beta,
                   double *C, int *ldc,
                   int transa_len, int transb_len);
```

Parameters

The C parameters are the same as the Fortran parameters described above. There are two additional parameters to the C function. These are required to satisfy the Fortran calling conventions and should be set to the value 1.

int transa_len

transa_len specifies the length of the string parameter transa.

int transb_len

transb_len specifies the length of the string parameter transb.

DGESV

Description

DGESV computes the solution to a real system of linear equations

```
A * X = B
```

where A is an N-by-N matrix and X and B are N-by-NRHS matrices. The LU decomposition with partial pivoting and row interchanges is used to factor A as

```
A = P * L * U
```

where P is a permutation matrix, L is unit lower triangular, and U is upper triangular. The factored form of A is then used to solve the system of equations A * X = B.

Fortran interface

```
SUBROUTINE DGETRF( N, NRHS, A, LDA, IPIV, B, LDB, INFO )
*      .. Scalar Arguments ..
       INTEGER            INFO, LDA, LDB, N, NRHS
*      .. Array Arguments ..
       INTEGER            IPIV( * )
       DOUBLE PRECISION   A( LDA, * ), B( LDB, * )
```

Parameters

`INTEGER  N`

The number of linear equations. That is, the order of the matrix A. `N` >= 0.

`INTEGER NRHS`

The number of right hand sides. That is, the number of columns of the matrix B. `NRHS` >= 0.

`DOUBLE PRECISION A`

An array of dimension (`LDA`,`N`).

On entry, the `N`-by-`N` coefficient matrix A.

On exit, the factors L and U from the factorization A = P*L*U; the unit diagonal elements of L are not stored.

`INTEGER LDA`

The leading dimension of the array A. `LDA` >= max(1,`N`).

`INTEGER IPIV`

An array of dimension (`N`).

The pivot indices that define the permutation matrix P; row i of the matrix was interchanged with row `IPIV`(i).

DOUBLE PRECISION B

An array of dimension (LDB,NRHS).

On entry, the N-by-NRHS matrix of right hand side matrix B.

On exit, if INFO = 0, the N-by-NRHS solution matrix X.

INTEGER LDB

The leading dimension of the array B. LDB >= max(1,N).

INTEGER INFO

- = 0: successful exit
- < 0: if INFO = -i, the i-th argument had an illegal value
- > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, so the solution could not be computed.

C prototype

When calling the library from C, the following names can be used: dgesv, dgesv_ or DGESV In all cases the prototype is:

```
extern void dgesv_(int *n, int *nrhs,
                   double *a, int *lda,
                   int *ipiv,
                   double *b, int *ldb,
                   int *info);
```

Parameters

The C parameters are the same as the Fortran parameters described above.

DGETRF

Description

DGETRF computes an LU factorization of a general M-by-N matrix A using partial pivoting with row interchanges.

The factorization has the form:

```
A = P * L * U
```

where P is a permutation matrix, L is lower triangular with unit diagonal elements (lower trapezoidal if m > n), and U is upper triangular (upper trapezoidal if m < n).

This is the right-looking Level 3 BLAS version of the algorithm.

Fortran interface

```
SUBROUTINE DGETRF( M, N, A, LDA, IPIV, INFO )
*     .. Scalar Arguments ..
      INTEGER            INFO, LDA, M, N
*     .. Array Arguments ..
      INTEGER            IPIV( * )
      DOUBLE PRECISION   A( LDA, * )
```

Parameters

INTEGER M

The number of rows of the matrix A. M >= 0.

INTEGER N

The number of columns of the matrix A. N >= 0.

DOUBLE PRECISION A

A is an array of dimension (LDA, N)

On entry, the M-by-N matrix to be factored.

On exit, the factors L and U from the factorization

A = P*L*U; the unit diagonal elements of L are not stored.

INTEGER LDA

The leading dimension of the array A. LDA >= max(1, M).

INTEGER IPIV

IPIV is an array of dimension (min(M,N))

The pivot indices; for 1 <= i <= min(M,N), row i of the matrix was interchanged with row IPIV(i).

`INTEGER INFO`

- = 0: successful exit
- < 0: if INFO = -i, the i-th argument had an illegal value
- > 0: if INFO = i, U(i,i) is exactly zero. The factorization has been completed, but the factor U is exactly singular, and division by zero will occur if it is used to solve a system of equations.

C prototype

When calling the library from C, the following names can be used: `dgetrf`, `dgetrf_` or `DGETRF`. In all cases the prototype is:

```
extern void dgetrf_(int *m, int *n, double *a, int *lda, int *ipiv, int *info);
```

Parameters

The C parameters are the same as the Fortran parameters described above.

5 Bibliography

[1] *A set of level 3 basic linear algebra subprograms*
J. J. Dongarra, J. Du Croz, S. Hammarling, I.S. Duff
ACM Transactions on Mathematical Software (TOMS) Volume 16, Issue 1, 1990.

[2] *HPL - A Portable Implementation of the High-Performance Linpack Benchmark for Distributed-Memory Computers*
A. Petitet, R. C. Whaley, J. Dongarra, A. Cleary
http://www.netlib.org/benchmark/hpl/

[3] *LAPACK: A portable linear algebra library for high-performance computers*
E. Angerson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, D. Sorensen
Proceedings of Supercomputing '90, Nov 1990, pp 2-11.

[4] *SDK Getting Started Guide*
Document Number: 06-UG-1117
ClearSpeed Technology

[5] *CSX600 Runtime Software User Guide*
Document Number: 06-UG-1345
ClearSpeed Technology

[6] *SDK Reference Manual*
Document Number: 06-UG-1136
ClearSpeed Technology