

# *ClearSpeed*<sup>™</sup>

---

CSXL

User Guide

---

**Document No. 06-UG-1584 Revision: 1.B**

August 2008



# Table of contents

- 1 Overview ..... 5**
  - 1.1 CSXL host-side library ..... 5
    - 1.1.1 How it works ..... 5
    - 1.1.2 Using with multiple Advance cards ..... 6
    - 1.1.3 Compatibility ..... 7
  - 1.2 Examples ..... 7
  
- 2 Using CSXL ..... 8**
  - 2.1 Installing CSXL ..... 8
    - 2.1.1 Initializing the environment ..... 8
    - 2.1.2 Resetting the card ..... 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 ..... 10
  - 2.4 Using CSXL with host applications ..... 12
    - 2.4.1 Linking with CSXL ..... 12
    - 2.4.2 Existing applications ..... 13
    - 2.4.3 Using CSXL with MATLAB ..... 14
    - 2.4.4 Using CSXL with Mathematica ..... 15
  
- 3 Optimizing use of DGEMM ..... 18**
  - 3.1 Intel processors ..... 18
  - 3.2 Performance characterization ..... 18
  - 3.3 Calculating host assist ..... 22
  
- 4 CSXL library functions ..... 24**
  
- 5 CSXL card-side functions ..... 47**
  - 5.1 Card-side DGEMM ..... 47
    - 5.1.1 Blocked data format (b96) ..... 47
    - 5.1.2 Notes ..... 51
    - 5.1.3 Example ..... 52
    - 5.1.4 Performance ..... 52

**6**      **Bibliography** ..... **53**

# 1 Overview

CSXL, the ClearSpeed math library, provides accelerated versions of a number of standard math functions for use with the ClearSpeed Advance accelerator cards. 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 directly accelerated by CSXL are:

## BLAS Level 3

- DGEMM
- ZGEMM
- ZGEMM3M
- DTRSM

## LAPACK

- DGETRF
- DGETRS
- DGESV
- DPOTRF
- DPOTRS
- DPOSV
- DGEQRF
- DORGQR
- DORMQR

See [Chapter 4: CSXL library functions](#) for details of the functions supported by CSXL.

See [Chapter 2: Using CSXL on page 8](#) for more information on using the library with your application.

The CSXL library supports two usage models:

- Host-side CSXL library: A library of BLAS/LAPACK functions that can be called from an application running on the host. CSXL will intercept the functions it accelerates and pass the non-accelerated functions back to standard BLAS/LAPACK libraries such as MKL or ACML.
- Card-side CSXL library: A library of BLAS/LAPACK functions callable from **C** code written for the Advance card. This provides direct access to the specialized form of functions from the card side applications.

## 1.1 CSXL host-side library

### 1.1.1 How it works

The CSXL accelerated functions are called by a user's application running on the host using the standard BLAS and LAPACK API. No code changes are required. If the problem size is suitable to gain acceleration, the CSXL library transparently offloads some or all of the processing to the Advance card. Where no performance benefit would be obtained by using the Advance card, or when the function is not supported by CSXL, the routine is executed

on the host by calling a standard host library such as MKL or ACML. To gain the best acceleration from the CSXL library and the Advance card, the application may want to use suitable vector / matrix dimension sizes. This is discussed in detail later. CSXL consists of two parts: host libraries used by the host application; and an executable that runs on the Advance card. The CSXL host library will load and interface with the card executable to gain the best acceleration for the host application.

Figure 1 shows a typical application calling a standard host BLAS library and an application making calls via CSXL. The CSXL library can utilize both the Advance card and the host processor cores to compute the BLAS / LAPACK function. This allows full use of the compute capabilities of the available hardware.

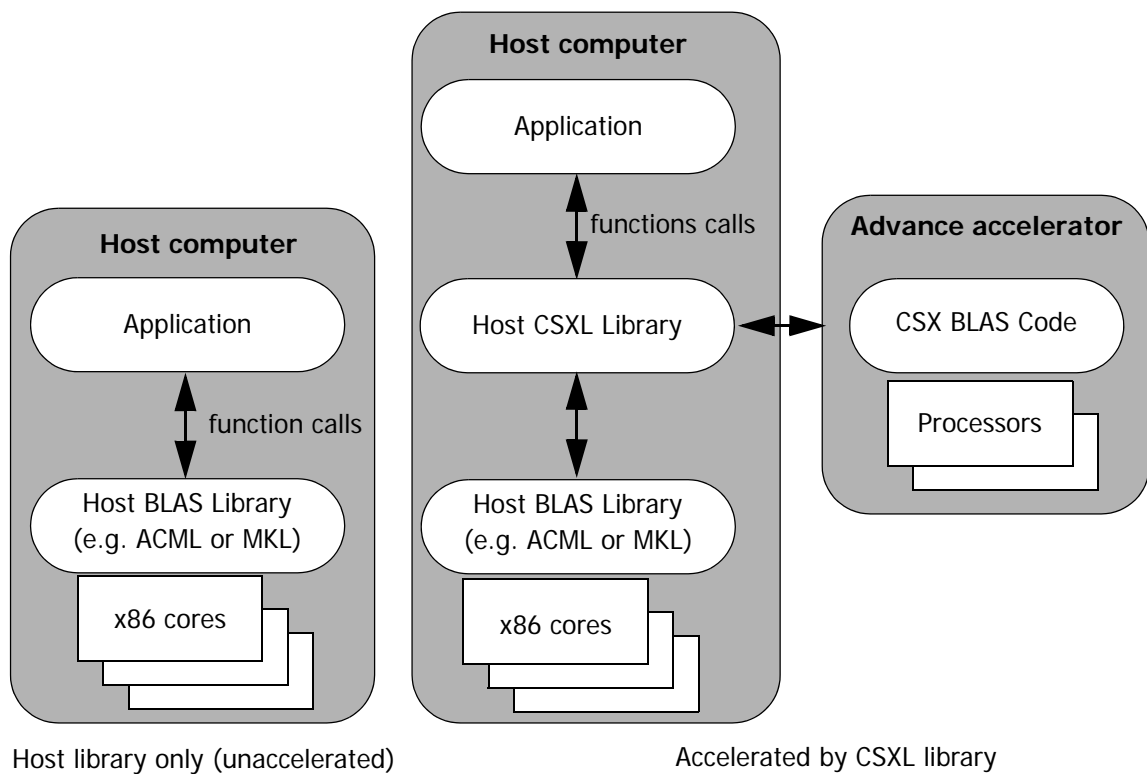


Figure 1. Host application calling BLAS functions

### 1.1.2 Using with multiple Advance cards

The CSXL library supports a single Advance card per process. Multiprocess applications (such as MPI processes) can exploit multiple cards. A process can target a specific card by setting the environment variable LLDINST to the desired Advance card instance number. When LLDINST is not set for a process the first available card is used. If no card is available the function is executed by the host processors.

### 1.1.3 Compatibility

#### Supported host operating systems

Information on the operating systems supported by the ClearSpeed software can be found at: <http://www.clearspeed.com/products/compatibility/>

*Note: ClearSpeed provides a driver for the supported 64-bit Microsoft Windows operating systems (Windows Server 2003 and Compute Cluster Server) but currently we only have 32-bit libraries and tools. It is not possible to mix 32 and 64-bit application software, so only 32-bit applications can be used with the CSXL library.*

*If you are using a 64-bit Windows operating system, you can install the ClearSpeed software for Microsoft Windows. The runtime package will install a 64-bit driver on 64-bit operating systems. You may use these components as you would on a 32-bit operating system with the caveat that the application and all the libraries that you run against the installed 32-bit ClearSpeed software stack must also be 32 bit.*

*For example, if you have a 64-bit native version of MATLAB installed this will not work correctly and you will have to install the 32-bit version of the application software. Similarly, if you build (ATLAS or GotoBLAS) or install (Intel MKL or AMD ACML) math libraries from third party sources, you will need to make sure they are also 32-bit versions.*

#### 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, Intel Fortran compiler
- For Windows XP: Visual C++ 2005, Intel Fortran compiler

## 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 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 card, and supporting runtime libraries.

#### 2.1.1 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/bin/bashrc
```

or:

```
source /opt/clearspeed/bin/cshrc
```

For Microsoft Windows:

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

#### 2.1.2 Resetting the card

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

```
csreset -v
```

See [\[5\]: Runtime Software User Guide](#) for more information on the `csreset` command options.

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

## 2.2 Using CSXL with your application

There are two versions of the library which implements CSXL on the host. These provide support for the different calling conventions of host BLAS and LAPACK libraries. These libraries are:

- `csxl_mkl`: compatible with MKL and most other host math libraries
- `csxl_acml`: compatible with ACML

You must use the CSXL library files appropriate to the host library being used.

The corresponding library files are:

- For Linux
  - libcsxl\_mkl.so
  - libcsxl\_acml.so
- For Microsoft Windows
  - csxl\_mkl.dll
  - csxl\_acml.dll

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

- /opt/clearspeed/lib/ for Linux
- C:\Program Files\clearspeed\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 24](#).

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 be specified for the CSXL library to work:

```
CS_HOST_BLAS
LD_LIBRARY_PATH (Linux only)
```

If you use a configuration file, 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 configuration variables

#### 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 card 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 MKL. For information on which BLAS libraries are supported, see Supported host libraries on page 7.

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

When using MKL 10 on Linux platforms, it is not sufficient to set `CS_HOST_BLAS` to `libmkl.so`. Instead `CS_HOST_BLAS` should explicitly reference the libraries which make up MKL. These are: `libmkl_intel_lp64.so`, `libmkl_intel_thread.so`, `libmkl_core.so`, `libguide.so`. For example:

```
CS_HOST_BLAS=/opt/intel/mkl/10.0.1.014/lib/em64t/libmkl_intel_lp64.so:/opt/intel/mkl/10.0.1.014/lib/em64t/libmkl_intel_thread.so:/opt/intel/mkl/10.0.1.014/lib/em64t/libmkl_core.so:/opt/intel/mkl/10.0.1.014/lib/em64t/libguide.so
```

MKL version 10 does not currently work with CSXL on Microsoft Windows platforms.

As another example, `GotoBLAS` 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 `GotoBLAS` 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/lib:$LD_LIBRARY_PATH
```

This variable does not exist in Windows systems.

### Optional configuration variables

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 card(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 card. 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 card 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 [Section 3.3: Calculating host assist on page 22](#).

```
CS_BLAS_HOST_ASSIST_PERCENTAGE=20
```

#### LLDINST

This variable controls which Advance card CSXL targets. When this variable is not set CSXL uses the first available card in the system.

### Tracing configuration variables

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.

*Note:* [Turning on tracing will affect the performance of the CSXL library. It is recommended that you do not turn on tracing except for specific purposes such as optimization as described in Section 3.3: Calculating host assist on page 22.](#)

The default, if CSXL\_TRACING is not defined, is to generate no trace output.

#### CSXL\_TRACING

If this variable is defined, for example CSXL\_TRACING=0, 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 card. 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.

```
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 (that is, if `CSXL_TRACING_OUTPUT` is undefined), this variable *must* be defined.

```
CSXL_CONSOLE_TRACING=1
```

## 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

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 DGEMM.c -o DGEMM -L/opt/clearspeed/lib -lcsxl_mkl -lpthread -lg2c -lm
```

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/lib:$LD_LIBRARY_PATH ./dgemm
```

## Microsoft Windows

If you are using Visual Studio 2005 to develop the host code, the library (`csxl_mkl.dll/csxl_acml.dll`) needs to be added to the list of libraries to be linked in the project.

To do this:

1. Open the **Project Properties** dialog box.
2. Go to the **Link** tab and add the path to the library folder in your `clearspeed` install directory.
3. Go to the **Input** page and add `csxl_mkl.lib/csxl_acml.lib` to **Additional Dependencies**.

For more details, see the Visual Studio documentation.

If you are using the Microsoft tools from the command line, 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.

## 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 14](#)). 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 [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_mkl.so libmkl_lapack64.so
ln -s libcsxl_mkl.so libmkl.so
```

For Windows:

```
rename mkl_lapack64.dll mkl_lapack64.dll.original
rename mkl.dll mkl.dll.original
copy csxl_mkl.dll mkl_lapack64.dll
copy csxl_mkl.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/lib/libcsxl_mkl.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.

### 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\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
```

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\mk1.dll
set BLAS_VERSION=C:\Progra~1\clearspeed\bin\csxl_mk1.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 <http://www.clearspeed.com/products/compatibility/> 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/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/lib/libcsxl_mk1.so
export CS_HOST_BLAS=/usr/local/matlabR14/bin/glnx86/libmk1.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 can be found at:

<http://www.clearspeed.com/products/compatibility/>

### 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/lib
```
2. Create a symbolic link
 

```
ln -s libcsxl_mkl.so libmkl_lapack32.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):

```
__PREFIX__=/usr/local/Wolfram/Mathematica/5.2/SystemFiles/Libraries/Linux-x86-64
export CS_HOST_BLAS=__PREFIX__/libmkl_lapack32.so: ↵
    __PREFIX__/libmkl_lapack64.so:__PREFIX__/libcsxl_mkl.so: ↵
    __PREFIX__/libmkl_vml_p4n.so
# Pick up the libraries in CSXL first then the standard ones
export LD_LIBRARY_PATH=/opt/clearspeed/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 `__PREFIX__` definition.

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

```
source /opt/clearspeed/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 can be found at:

<http://www.clearspeed.com/products/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[]`
- `CholeskyDecomposition[]`
- `QRdecomposition[]`

## 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/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/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*:

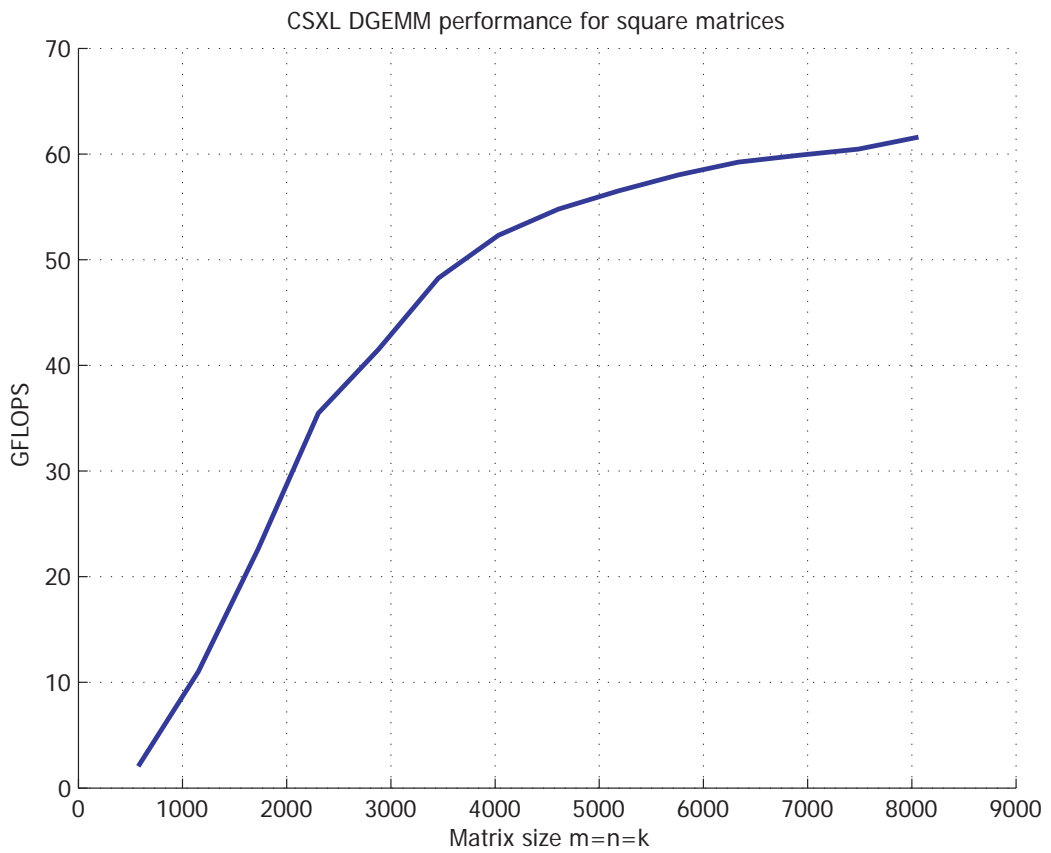
Intel Optimized Linpack Benchmark is threaded to effectively use multiple processors. Therefore, in MP systems, best performance will be obtained with hyperthreading turned off. This insures that the operating system assigns threads to physical processors only.

### 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 card.

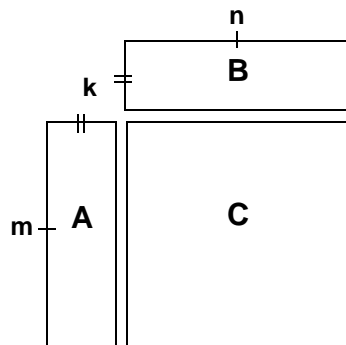
[Figure 2](#) shows a typical performance graph for multiplying square matrices ( $m=n=k$ ). For the DGEMM implementation in the CSXL library, the best performance is obtained when  $n$

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 2. A square DGEMM performance**

Figure 4 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 where  $A$  is high and narrow,  $B$  is wide and short,  $C$  is large and square.



**Figure 3. Illustration of LU-shape DGEMM**

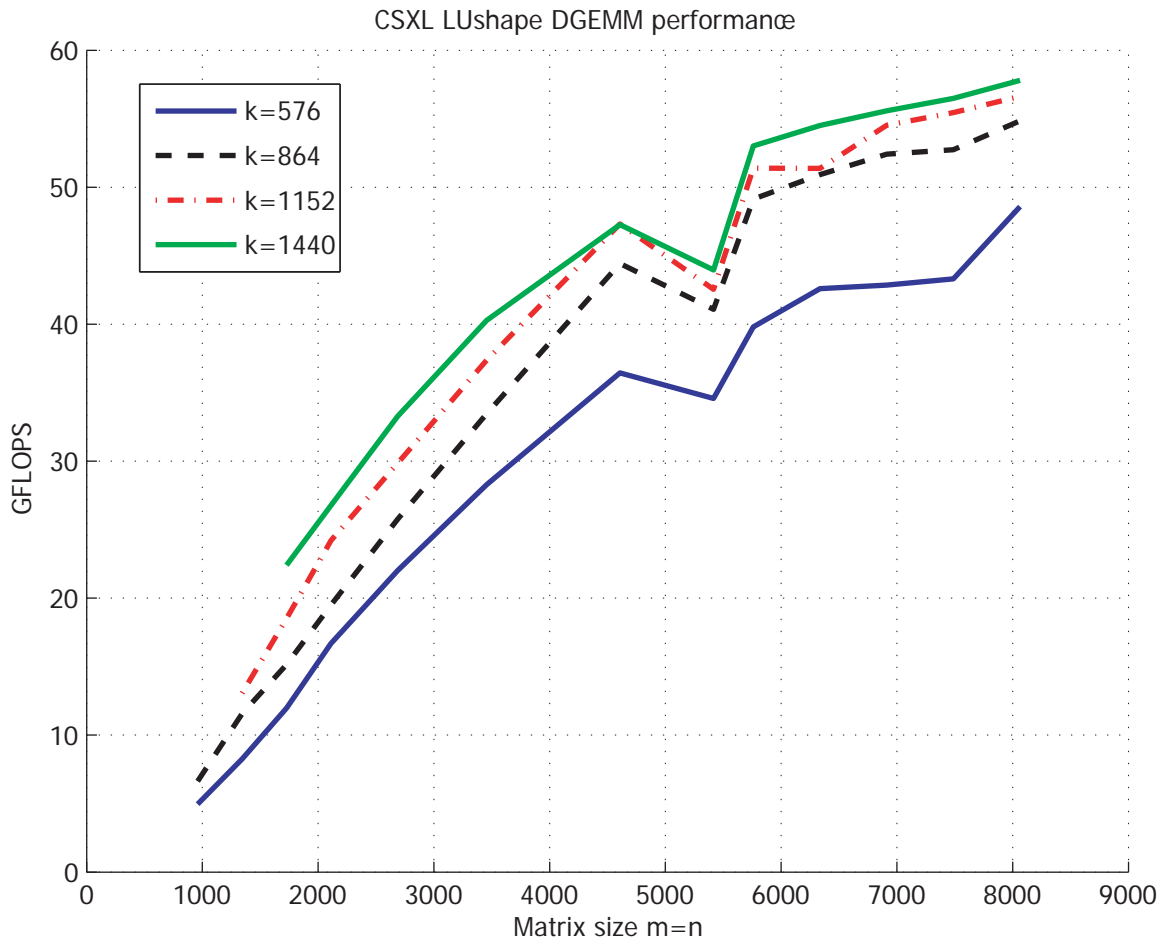


Figure 4. LU-shape DGEMM performance

In [Figure 6](#), 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 5](#) where  $A$  is large and square,  $B$  is high and narrow,  $C$  is high and narrow.

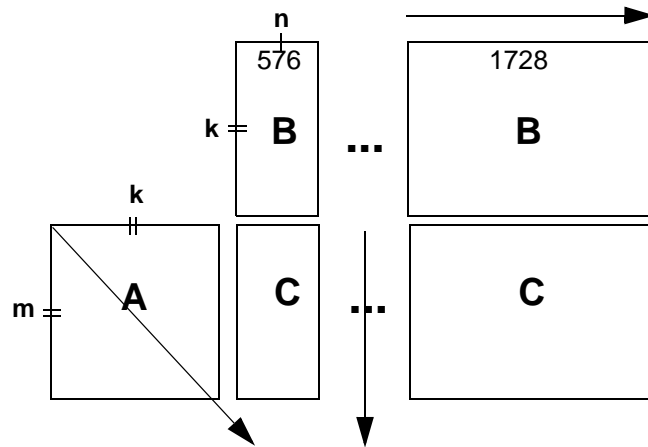


Figure 5. Illustration of a square matrix A DGEMM

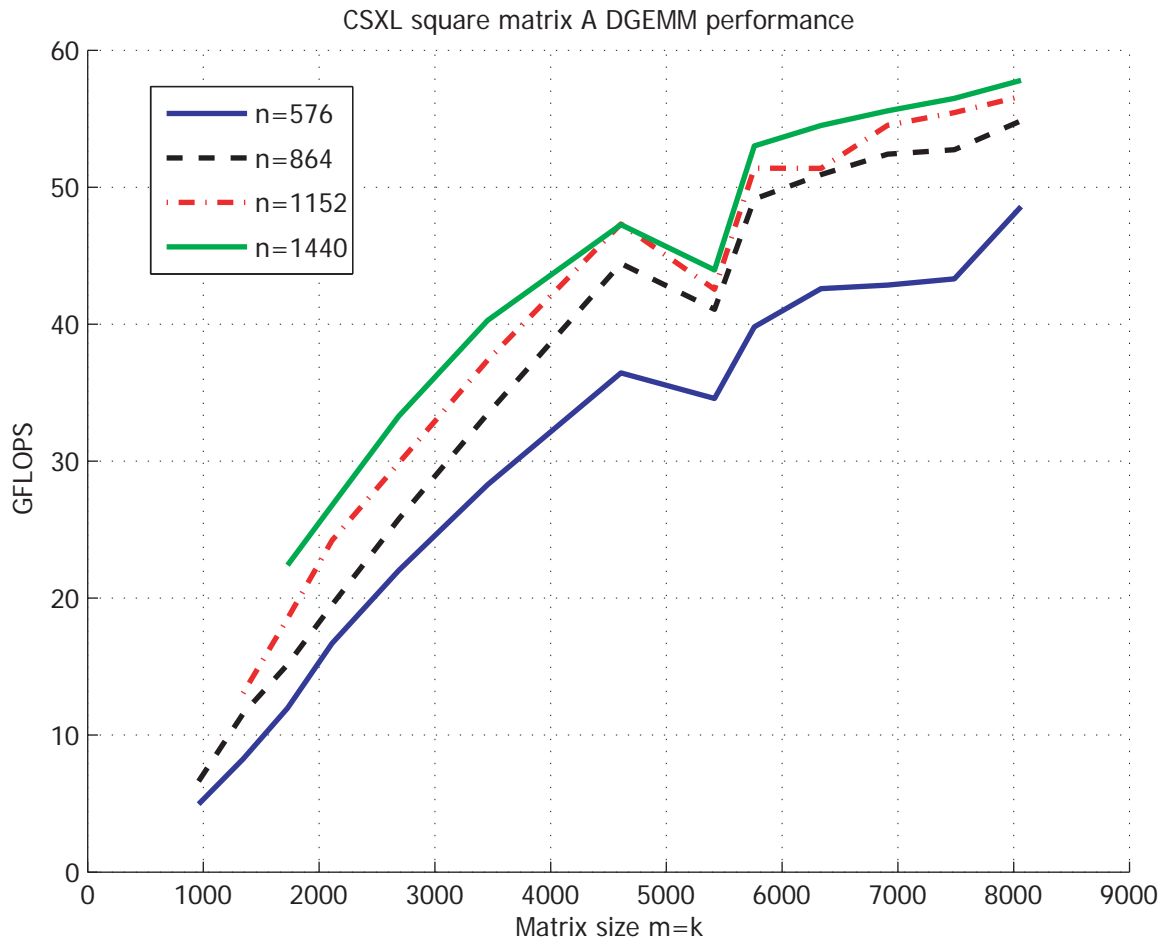


Figure 6. A square matrix A DGEMM performance

### 3.3 Calculating host assist

The CSXL library distributes function execution between the host and the Advance card(s). The `CS_BLAS_HOST_ASSIST_PERCENTAGE` environment variable allows the user to specify how work should be shared between the host and the Advance card(s) and is given as a percentage of the function calculation to be performed on the host. This environment value can be set to divide the function calculation proportionate to the compute capabilities so that the Advance card(s) and the host complete at the same time. This section describes how to find an appropriate value for this environment variable.

The following instructions take you through steps to calculate the host assist percentage value on Linux platforms from a command prompt. Similar steps can be made in a Windows platforms.

1. Change directory into the installed CSXL package: `examples/csxl/dgemm_example/`
2. Compile the `dgemm` program using the makefile provided.
3. Setup the ClearSpeed runtime environment and reset the Advance card(s)  
`source /opt/clearspeed/bin/bashrc`  
`csreset -A`
4. Setup the host blas library:  
`export CS_HOST_BLAS=/usr/lib/libmkl.so`  
`export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/lib/`
5. Setup a host percentage of 100%  
`export CS_BLAS_HOST_ASSIST_PERCENTAGE=100`
6. Run the `dgemm_example` program to find the Gflops (GF/s) score:  
`./dgemm_example -m 5760 -n 5760 -k 5760 -t 2`
7. Setup a host percentage of 0%  
`export CS_BLAS_HOST_ASSIST_PERCENTAGE=0`
8. Run the `dgemm_example` program to find the Gflops (GF/s) score:  
`./dgemm_example -m 5760 -n 5760 -k 5760 -t 2`
9. Use the gained figures (host-score and card-score) to derive an approximation of the require host assist fraction:  
$$\text{host assist percentage} = (\text{host-score} / (\text{card-score} + \text{host-score})) \times 100\% = 24\%$$
10. You can then set the configuration variable to this value:  
`export 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.

## 4 CSXL library functions

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

The BLAS functions are:

- DGEMM [on page 24](#).
- DTRSM [on page 39](#).
- ZGEMM [on page 41](#).
- ZGEMM3M [on page 43](#).

The LAPACK functions are:

- DGEQRF [on page 27](#).
- DGESV [on page 28](#).
- DGETRF [on page 29](#).
- DGETRS [on page 30](#).
- DORGQR [on page 32](#).
- DORMQR [on page 33](#).
- DPOSV [on page 35](#).
- DPOTRF [on page 36](#).
- DPOTRS [on page 37](#).

### DGEMM

#### Description

DGEMM performs one of the matrix-matrix operations:

$$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C$$

where  $\text{op}(X)$  is one of:

- $\text{op}(X) = X$  (identity)
- $\text{op}(X) = X'$  (transpose)

Alpha and beta are scalars. A, B and C are matrices, with  $\text{op}(A)$  an m by k matrix,  $\text{op}(B)$  a k by n matrix and C an m by n matrix.

#### Fortran interface

```

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

```

## Fortran 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.

## DGEQRF

### Description

DGEQRF computes a QR factorization of a real M-by-N matrix A:  $A = Q * R$ .

### Fortran interface

```

SUBROUTINE DGEQRF( M, N, A, LDA, TAU, WORK, LWORK, INFO )
  INTEGER          INFO, LDA, LWORK, M, N
  DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )

```

### Parameters

#### INTEGER M

The number of rows of the matrix A.  $M \geq 0$ .

#### INTEGER N

The number of columns of the matrix A.  $N \geq 0$ .

#### DOUBLE PRECISION A

A is an (input/output) double-precision array of the dimension  $(LDA, N)$ .

On entry, the M-by-N matrix A.

On exit, the elements on and above the diagonal of the array contain the  $\min(M, N)$ -by-N upper trapezoidal matrix R (R is upper triangular if  $m \geq n$ ); the elements below the diagonal, with the array TAU, represent the orthogonal matrix Q as a product of  $\min(m, n)$  elementary reflectors (see: [Further details on page 28](#)).

#### INTEGER LDA

The leading dimension of the array A.  $LDA \geq \max(1, M)$ .

#### DOUBLE PRECISION TAU

TAU is an (output) double-precision array of the dimension  $(\min(M, N))$ .

The scalar factors of the elementary reflectors (see: [Further details on page 28](#)).

#### DOUBLE PRECISION WORK

WORK is a (workspace/output) double-precision array of the dimension  $(\max(1, LWORK))$ .

On exit, if  $INFO = 0$ ,  $WORK(1)$  returns the optimal LWORK.

#### INTEGER LWORK

The dimension of the array WORK.  $LWORK \geq \max(1, N)$ . For optimum performance,  $LWORK \geq N * NB$ , where NB is the optimal block size.

If  $LWORK = -1$ , then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

**INTEGER INFO**

- = 0: successful exit
- < 0: if INFO = -i, the i-th argument had an illegal value

**Further details**

The matrix Q is represented as a product of elementary reflectors

$Q = H(1) H(2) \dots H(k)$ , where  $k = \min(m,n)$ .

Each H(i) has the form

$$H(i) = I - \tau * v * v'$$

where  $\tau$  is a real scalar, and  $v$  is a real vector with

$v(1:i-1) = 0$  and  $v(i) = 1$ ;  $v(i+1:m)$  is stored on exit in  $A(i+1:m,i)$ ,

and  $\tau$  in TAU(i).

**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 DGESV( 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, * )

```

**Fortran parameters****INTEGER N**

The number of linear equations. That is, the order of the matrix A.  $N \geq 0$ .

**INTEGER NRHS**

The number of righthand sides. That is, the number of columns of the matrix B.  $NRHS \geq 0$ .

**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  $A$ .  $LDA \geq \max(1, N)$ .

#### 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

$B$  is an array of dimension  $(LDB, NRHS)$ .

On entry, the  $N$ -by- $NRHS$  matrix of righthand side matrix  $B$ .

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

#### INTEGER LDB

The leading dimension of the array  $B$ .  $LDB \geq \max(1, N)$ .

#### 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.

## 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, * )

```

## Fortran parameters

### INTEGER M

The number of rows of the matrix A.  $M \geq 0$ .

### INTEGER N

The number of columns of the matrix A.  $N \geq 0$ .

### 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.  $LDA \geq \max(1, M)$ .

### INTEGER IPIV

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

The pivot indices; for  $1 \leq i \leq \min(M, N)$ , 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.

## DGETRS

### Description

DGETRS solves a system of linear equations:

$$A * X = B \quad \text{or} \quad A' * X = B$$

with a general N-by-N matrix, A using the LU factorization computed by DGETRF.

### Fortran interface

```

SUBROUTINE DGETRS( TRANS, N, NRHS, A, LDA, IPIV, B, LDB, INFO )
  CHARACTER          TRANS
  INTEGER            INFO, LDA, LDB, N, NRHS
  INTEGER            IPIV( * )
  DOUBLE PRECISION  A( LDA, * ), B( LDB, * )

```

## Parameters

### CHARACTER\*1 TRANS

Specifies the form of the system of equations:

- = 'N':  $A * X = B$  (No transpose)
- = 'T':  $A' * X = B$  (Transpose)
- = 'C':  $A' * X = B$  (Conjugate transpose = Transpose)

### INTEGER N

The order of the matrix A.  $N \geq 0$ .

### INTEGER NRHS

The number of righthand sides, that is, the number of columns of the matrix B.  $NRHS \geq 0$ .

### DOUBLE PRECISION A

A is a double-precision array of dimension  $(LDA, N)$ .

The factors L and U from the factorization  $A = P * L * U$  as computed by DGETRF.

### INTEGER LDA

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

### INTEGER IPIV

IPIV is an (input) integer array of dimension  $(N)$ .

The pivot indices from DGETRF; for  $1 \leq i \leq N$ , row  $i$  of the matrix was interchanged with row  $IPIV(i)$ .

### DOUBLE PRECISION B

B is an (input/output) double-precision array of dimension  $(LDB, NRHS)$ .

On entry, the righthand side matrix B.

On exit, the solution matrix X.

### INTEGER LDB

The leading dimension of the array B.  $LDB \geq \max(1, N)$ .

### INTEGER INFO

- = 0: successful exit
- < 0: if  $INFO = -i$ , the  $i$ -th argument had an illegal value

## DORGQR

### Description

DORGQR generates an M-by-N real matrix Q with orthonormal columns, which is defined as the first N columns of a product of K elementary reflectors of order M:

$$Q = H(1) H(2) \dots H(k)$$

as returned by DGEQRF.

### Fortran interface

```
SUBROUTINE DORGQR( M, N, K, A, LDA, TAU, WORK, LWORK, INFO )
  INTEGER          INFO, K, LDA, LWORK, M, N
  DOUBLE PRECISION A( LDA, * ), TAU( * ), WORK( * )
```

### Parameters

#### INTEGER M

The number of rows of the matrix Q.  $M \geq 0$ .

#### INTEGER N

The number of columns of the matrix Q.  $M \geq N \geq 0$ .

#### INTEGER K

The number of elementary reflectors whose product defines the matrix Q.  $N \geq K \geq 0$ .

#### DOUBLE PRECISION A

A is an (input/output) double-precision array of the dimension (LDA, N).

On entry, the i-th column must contain the vector which defines the elementary reflector H(i), for  $i = 1, 2, \dots, k$ , as returned by DGEQRF in the first k columns of its array argument A.

On exit, the M-by-N matrix Q.

#### INTEGER LDA

The first dimension of the array A.  $LDA \geq \max(1, M)$ .

#### DOUBLE PRECISION TAU

TAU is a double-precision array of dimension (K).

TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

#### DOUBLE PRECISION WORK

WORK is a (workspace/output) double-precision array of the dimension (MAX(1, LWORK)).

On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

**INTEGER LWORK**

The dimension of the array WORK.  $LWORK \geq \max(1, N)$ .

For optimum performance  $LWORK \geq N * NB$ , where NB is the optimal blocksize.

If  $LWORK = -1$ , then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

**INTEGER INFO**

- = 0: successful exit
- < 0: if  $INFO = -i$ , the i-th argument has an illegal value.

**DORMQR****Description**

DORMQR overwrites the general real M-by-N matrix C with:

$$\begin{array}{l} \text{SIDE} = \text{'L'} \quad \text{SIDE} = \text{'R'} \\ \text{TRANS} = \text{'N'}: Q * C C * Q \\ \text{TRANS} = \text{'T'}: Q^{**T} * C C * Q^{**T} \end{array}$$

where Q is a real orthogonal matrix, defined as the product of k elementary reflectors.

$$Q = H(1) H(2) \dots H(k)$$

as returned by DGEQRF. Q is of order M if  $SIDE = \text{'L'}$  and of order N if  $SIDE = \text{'R'}$ .

**Fortran interface**

```

SUBROUTINE DORMQR( SIDE, TRANS, M, N, K, A, LDA, TAU, C, LDC, WORK,
$                LWORK, INFO )
  CHARACTER      SIDE, TRANS
  INTEGER        INFO, K, LDA, LDC, LWORK, M, N
  DOUBLE PRECISION A( LDA, * ), C( LDC, * ), TAU( * ), WORK( * )

```

**Parameters****CHARACTER\*1 SIDE**

- = 'L': apply Q or Q\*\*T from the Left;
- = 'R': apply Q or Q\*\*T from the Right.

**CHARACTER\*1 TRANS**

- = 'N': No transpose, apply Q;
- = 'T': Transpose, apply Q\*\*T.

**INTEGER M**

The number of rows of the matrix C.  $M \geq 0$ .

**INTEGER N**

The number of columns of the matrix C.  $N \geq 0$ .

**INTEGER K**

The number of elementary reflectors whose product defines the matrix Q.

If SIDE = 'L',  $M \geq K \geq 0$ ;

if SIDE = 'R',  $N \geq K \geq 0$ .

**DOUBLE PRECISION A**

A is an (input) double-precision array of dimension (LDA,K).

The i-th column must contain the vector which defines the elementary reflector H(i), for  $i = 1, 2, \dots, k$ , as returned by DGEQRF in the first k columns of its array argument A. A is modified by the routine but restored on exit.

**INTEGER LDA**

The leading dimension of the array A.

If SIDE = 'L',  $LDA \geq \max(1, M)$ ;

if SIDE = 'R',  $LDA \geq \max(1, N)$ .

**DOUBLE PRECISION TAU**

TAU is an (input) double-precision array of dimension (K).

TAU(i) must contain the scalar factor of the elementary reflector H(i), as returned by DGEQRF.

**DOUBLE PRECISION C**

C is an (input/output) double-precision array of dimension (LDC,N).

On entry, the M-by-N matrix C.

On exit, C is overwritten by  $Q^*C$  or  $Q^{**T}C$  or  $C^*Q^{**T}$  or  $C^*Q$ .

**INTEGER LDC**

The leading dimension of the array C.  $LDC \geq \max(1, M)$ .

**DOUBLE PRECISION WORK**

WORK is a (workspace/output) double-precision array of dimension  $(\max(1, LWORK))$ .

On exit, if INFO = 0, WORK(1) returns the optimal LWORK.

**INTEGER LWORK**

The dimension of the array WORK.

If SIDE = 'L', LWORK  $\geq$  max(1,N);

if SIDE = 'R', LWORK  $\geq$  max(1,M).

For optimum performance LWORK  $\geq$  N\*NB if SIDE = 'L', and

LWORK  $\geq$  M\*NB if SIDE = 'R', where NB is the optimal blocksize.

If LWORK = -1, then a workspace query is assumed; the routine only calculates the optimal size of the WORK array, returns this value as the first entry of the WORK array, and no error message related to LWORK is issued by XERBLA.

**INTEGER INFO**

- = 0: successful exit
- < 0: if INFO = -i, the i-th argument had an illegal value

**DPOSV****Description**

DPOSV computes the solution to a real system of linear equations  $A * X = B$ , where A is an N-by-N symmetric positive definite matrix and X and B are N-by-NRHS matrices.

The Cholesky decomposition is used to factor A as:

$A = U^{**T} * U$ , if UPLO = 'U', or

$A = L * L^{**T}$ , if UPLO = 'L',

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

**Fortran interface**

```

SUBROUTINE DPOSV( UPLO, N, NRHS, A, LDA, B, LDB, INFO )
  CHARACTER          UPLO
  INTEGER            INFO, LDA, LDB, N, NRHS
  DOUBLE PRECISION  A( LDA, * ), B( LDB, * )

```

**Parameters****CHARACTER\*1 UPLO**

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

**INTEGER N**

The number of linear equations, that is, the order of the matrix A.  $N \geq 0$ .

**INTEGER NRHS**

The number of righthand sides, that is, the number of columns of the matrix B.  $NRHS \geq 0$ .

**DOUBLE PRECISION A**

A is an (input/output) double-precision array of dimension  $(LDA, N)$ . On entry, the symmetric matrix A. If  $UPLO = 'U'$ , the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If  $UPLO = 'L'$ , the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if  $INFO = 0$ , the factor U or L from the Cholesky factorization  $A = U^*U$  or  $A = LL^*$ .

**INTEGER LDA**

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

**DOUBLE PRECISION B**

B is an (input/output) double-precision array of dimension  $(LDB, NRHS)$ .

On entry, the N-by-NRHS righthand side matrix B.

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

**INTEGER LDB**

The leading dimension of the array B.  $LDB \geq \max(1, N)$ .

**INTEGER INFO**

= 0: successful exit

< 0: if  $INFO = -i$ , the i-th argument had an illegal value

> 0: if  $INFO = i$ , the leading minor of order i of A is not positive definite, so the factorization could not be completed, and the solution has not been computed.

**DPOTRF****Description**

DPOTRF computes the Cholesky factorization of a real symmetric positive definite matrix A. The factorization has the form:

$A = U^*U$ , if  $UPLO = 'U'$ , or

$A = LL^*$ , if  $UPLO = 'L'$ ,

where U is an upper triangular matrix and L is lower triangular. This is the block version of the algorithm, calling Level 3 BLAS.

## Fortran interface

```

SUBROUTINE DPOTRF( UPLO, N, A, LDA, INFO )
    CHARACTER          UPLO
    INTEGER            INFO, LDA, N
    DOUBLE PRECISION  A( LDA, * )

```

## Parameters

### CHARACTER\*1 UPLO

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

### INTEGER N

The order of the matrix A.  $N \geq 0$ .

A is a (input/output) double-precision array of the dimension  $(LDA, N)$ .

On entry, the symmetric matrix A. If `UPLO = 'U'`, the leading N-by-N upper triangular part of A contains the upper triangular part of the matrix A, and the strictly lower triangular part of A is not referenced. If `UPLO = 'L'`, the leading N-by-N lower triangular part of A contains the lower triangular part of the matrix A, and the strictly upper triangular part of A is not referenced.

On exit, if `INFO = 0`, the factor U or L from the Cholesky factorization  $A = U^*U$  or  $A = LL^*$ .

### INTEGER LDA

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

### INTEGER INFO

- = 0: successful exit
- < 0: if `INFO = -i`, the i-th argument had an illegal value
- > 0: if `INFO = i`, the leading minor of order i is not positive definite, and the factorization could not be completed.

## DPOTRS

### Description

DPOTRS solves a system of linear equations  $A^*X = B$  with a symmetric positive definite matrix A using the Cholesky factorization:

$$A = U^*U$$

or

$$A = LL^*$$

computed by DPOTRF.

## Fortran interface

```

SUBROUTINE DPOTRS( UPLO, N, NRHS, A, LDA, B, LDB, INFO )
  .. Scalar Arguments ..
  CHARACTER          UPLO
  INTEGER            INFO, LDA, LDB, N, NRHS
  ..
  .. Array Arguments ..
  DOUBLE PRECISION  A( LDA, * ), B( LDB, * )

```

## Parameters

### CHARACTER\*1 UPLO

= 'U': Upper triangle of A is stored;

= 'L': Lower triangle of A is stored.

### INTEGER N

The order of the matrix A.  $N \geq 0$ .

### INTEGER NRHS

The number of righthand sides, that is, the number of columns of the matrix B.  $NRHS \geq 0$ .

### DOUBLE PRECISION A

A is an (input) double-precision array of dimension  $(LDA, N)$ .

The triangular factor U or L from the Cholesky factorization:

$$A = U^*T^*U \text{ or } A = L^*L^*T$$

as computed by DPOTRF.

### INTEGER LDA

The leading dimension of the array A.  $LDA \geq \max(1, N)$ .

### DOUBLE PRECISION B

B is an (input/output) double-precision array of dimension  $(LDB, NRHS)$ .

On entry, the righthand side matrix B.

On exit, the solution matrix X.

### INTEGER LDB

The leading dimension of the array B.  $LDB \geq \max(1, N)$ .

### INTEGER INFO

- = 0: successful exit.
- < 0: if  $INFO = -i$ , the i-th argument had an illegal value.

## DTRSM

### Description

DTRSM solves one of the matrix equations:

$$\text{op}(A) * X = \alpha * B,$$

or

$$X * \text{op}(A) = \alpha * B,$$

where  $\alpha$  is a scalar,  $X$  and  $B$  are  $m$  by  $n$  matrices;  $A$  is a unit, or non-unit, upper or lower triangular matrix;  $\text{op}(A)$  is one of the following:

- $\text{op}(A) = A$
- $\text{op}(A) = A'$  (transpose)

The matrix  $X$  is overwritten on  $B$ .

### Fortran interface

```

SUBROUTINE DTRSM ( SIDE, UPLO, TRANSA, DIAG, M, N, ALPHA, A,
$                LDA, B, LDB )
CHARACTER*1      SIDE, UPLO, TRANSA, DIAG
INTEGER          M, N, LDA, LDB
DOUBLE PRECISION ALPHA
DOUBLE PRECISION A( LDA, * ), B( LDB, * )

```

### Parameters

#### CHARACTER\*1 SIDE

On entry, `SIDE` specifies whether  $\text{op}(A)$  appears on the left or right of  $X$  as follows:

`SIDE = 'L' or 'l'`  $\text{op}(A) * X = \alpha * B$ .

`SIDE = 'R' or 'r'`  $X * \text{op}(A) = \alpha * B$ .

Unchanged on exit.

#### CHARACTER\*1 UPLO

On entry, `UPLO` specifies whether the matrix  $A$  is an upper or lower triangular matrix as follows:

`UPLO = 'U' or 'u'`  $A$  is an upper triangular matrix.

`UPLO = 'L' or 'l'`  $A$  is a lower triangular matrix.

Unchanged on exit.

#### CHARACTER\*1 TRANSA

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

`TRANSA = 'N' or 'n'`  $\text{op}(A) = A$ .

`TRANSA = 'T' or 't'`  $\text{op}(A) = A'$

`TRANSA = 'C' or 'c'`  $\text{op}(A) = \text{conjg}(A')$

Unchanged on exit.

#### **CHARACTER\*1 DIAG**

On entry, **DIAG** specifies whether or not **A** is unit triangular as follows:

**DIAG** = 'U' or 'u' **A** is assumed to be unit triangular.

**DIAG** = 'N' or 'n' **A** is not assumed to be unit triangular.

Unchanged on exit.

#### **INTEGER M**

On entry, **M** specifies the number of rows of **B**. **M** must be at least zero.

Unchanged on exit.

#### **INTEGER N**

On entry, **N** specifies the number of columns of **B**. **N** must be at least zero.

Unchanged on exit.

#### **DOUBLE PRECISION ALPHA**

On entry, **ALPHA** specifies the scalar alpha. When alpha is zero, then **A** is not referenced and **B** need not be set before entry.

Unchanged on exit.

#### **DOUBLE PRECISION A**

**A** is an array of dimension ( **LDA**, **k** ),

where **k** is **m** when **SIDE** = 'L' or 'l' and is **n** when **SIDE** = 'R' or 'r'.

Before entry with **UPLO** = 'U' or 'u', the leading **k** by **k** upper triangular part of the array **A** must contain the upper triangular matrix and the strictly lower triangular part of **A** is not referenced. Before entry with **UPLO** = 'L' or 'l', the leading **k** by **k** lower triangular part of the array **A** must contain the lower triangular matrix and the strictly upper triangular part of **A** is not referenced. Note that when **DIAG** = 'U' or 'u' (the diagonal elements of **A**) are not referenced either, but are assumed to be unity.

Unchanged on exit.

#### **INTEGER LDA**

On entry, **LDA** specifies the first dimension of **A** as declared in the calling (sub) program.

When **SIDE** = 'L' or 'l' then **LDA** must be at least  $\max(1, m)$ , when **SIDE** = 'R' or 'r' then **LDA** must be at least  $\max(1, n)$ .

Unchanged on exit.

#### **DOUBLE PRECISION B**

**B** is a double-precision array of dimension ( **LDB**, **n** ).

Before entry, the leading  $m$  by  $n$  part of the array  $B$  must contain the righthand side matrix  $B$ , and on exit is overwritten by the solution matrix  $X$ .

#### INTEGER LDB

On entry,  $LDB$  specifies the first dimension of  $B$  as declared in the calling (sub) program.  $LDB$  must be at least  $\max(1, m)$ .

Unchanged on exit.

## ZGEMM

### Description

ZGEMM performs one of the matrix-matrix operations:

$$C := \alpha * \text{op}(A) * \text{op}(B) + \beta * C,$$

where  $\text{op}(X)$  is one of:

- $\text{op}(X) = X$
- $\text{op}(X) = X'$  (transpose)
- $\text{op}(X) = \text{conjg}(X')$  (conjugate transpose)

Alpha and beta are scalars.  $A$ ,  $B$  and  $C$  are matrices with:

- $\text{op}(A)$  an  $m$  by  $k$  matrix
- $\text{op}(B)$  a  $k$  by  $n$  matrix
- $C$  an  $m$  by  $n$  matrix.

### Fortran interface

```

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

```

### Parameters

#### CHARACTER\*1 TRANSA

On entry,  $TRANSA$  specifies the form of  $\text{op}(A)$  to be used in the matrix multiplication as follows:

- If  $TRANSA = 'N'$  or  $'n'$ ,  $\text{op}(A) = A$ .
- If  $TRANSA = 'T'$  or  $'t'$ ,  $\text{op}(A) = A'$ .
- If  $TRANSA = 'C'$  or  $'c'$ ,  $\text{op}(A) = \text{conjg}(A')$ .

Unchanged on exit.

**CHARACTER\*1 TRANSB**

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

- TRANSB = 'N' or 'n',  $op(B) = B$ .
- TRANSB = 'T' or 't',  $op(B) = B'$ .
- TRANSB = 'C' or 'c',  $op(B) = conjg(B')$ .

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.

**COMPLEX\*16 ALPHA**

On entry, ALPHA specifies the scalar alpha.

Unchanged on exit.

**COMPLEX\*16 A**

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

Before entry with TRANSB = '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 (sub) program.

When TRANSB = 'N' or 'n' then LDA must be at least  $\max(1, m)$ , otherwise LDA must be at least  $\max(1, k)$ .

Unchanged on exit.

**COMPLEX\*16 B**

B is an array of dimension ( 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 (sub) 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.

**COMPLEX\*16 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.

**COMPLEX\*16 C**

C is an array of dimension ( 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:

$$( \alpha * \text{op}( A ) * \text{op}( B ) + \text{beta} * C )$$

**INTEGER LDC**

On entry, LDC specifies the first dimension of C as declared in the calling (sub) program.

LDC must be at least  $\max(1, m)$ .

Unchanged on exit.

**ZGEMM3M****Description**

ZGEMM3M performs one of the matrix-matrix operations:

$$C := \alpha * \text{op}( A ) * \text{op}( B ) + \text{beta} * C,$$

where  $\text{op}( X )$  is one of:

- $\text{op}( X ) = X$
- $\text{op}( X ) = X'$  (transpose)
- $\text{op}( X ) = \text{conjg}( X' )$  (conjugate transpose)

Alpha and beta are scalars. A, B and C are matrices with:

- $op( A )$  an  $m$  by  $k$  matrix
- $op( B )$  a  $k$  by  $n$  matrix
- $C$  an  $m$  by  $n$  matrix

ZGEMM3M uses an algorithm requiring three real matrix multiplications and five real matrix additions to compute the complex matrix product; ZGEMM uses four real matrix multiplications and two real matrix additions. The matrix multiplication time is usually the limiting performance factor in ZGEMM, therefore ZGEMM3M may run up to 33 percent faster than ZGEMM. However, due to other overhead associated with the 3M routine, these performance improvements may not always be realized.

If you wish to use ZGEMM3M instead of ZGEMM, you can do so by setting the environment variable CSXL\_ZGEMM3M. No changes in the code to replace ZGEMM with ZGEMM3M are necessary.

### Fortran interface

```

SUBROUTINE ZGEMM3M ( TRANSA, TRANSB, M, N, K, ALPHA, A, LDA, B,
$                   LDB, BETA, C, LDC )
* .. Scalar Arguments ..
DOUBLE COMPLEX ALPHA,BETA
INTEGER          K,LDA,LDB,LDC,M,N
CHARACTER        TRANSA,TRANSB
* ..
* .. Array Arguments ..
DOUBLE COMPLEX 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',  $op( A ) = conjg( A' )$

Unchanged on exit.

#### CHARACTER\*1 TRANSB

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

- TRANSB = 'N' or 'n',  $op( B ) = B$ .
- TRANSB = 'T' or 't',  $op( B ) = B'$ .
- TRANSB = 'C' or 'c',  $op( B ) = conjg( B' )$ .

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.

**COMPLEX\*16 ALPHA**

On entry, *ALPHA* specifies the scalar alpha.

Unchanged on exit.

**COMPLEX\*16 A**

*A* is an array of `DIMENSION ( 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 (sub) 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.

**COMPLEX\*16 B**

*B* is an array of dimension( `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 (sub) 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.

**COMPLEX\*16    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.

**COMPLEX\*16    C**

C is an array of DIMENSION ( 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 (sub) program.

LDC must be at least  $\max( 1, m )$ .

Unchanged on exit.

## 5 CSXL card-side functions

This chapter provides a definition of the card-side functions implemented in the CSXL library.

### 5.1 Card-side DGEMM

This section describes the card-level interface to blocked Double Precision General Matrix Multiplication (DGEMM).

This library allows you to write your own CSX programs and call a card-side version of DGEMM. A card-level library is delivered, named `csxl_blas.csa` and a header file `csxl_blas.h`. This header file will expose only the DGEMM interface. There are some limitations in using the DGEMM calls, specifically the data format.

The card-level library includes custom microcode and ensures that it is loaded as required.

#### 5.1.1 Blocked data format (b96)

The blocked DGEMM can operate on any matrix where the number of rows and the number of columns are multiples of the block size, 96.

*Note:* *The actual number of rows and columns must be a multiple of 96, however the parameters passed to the `bdgemm96` call are expressed as a number of blocks, as described in : [BDGEMM96](#).*

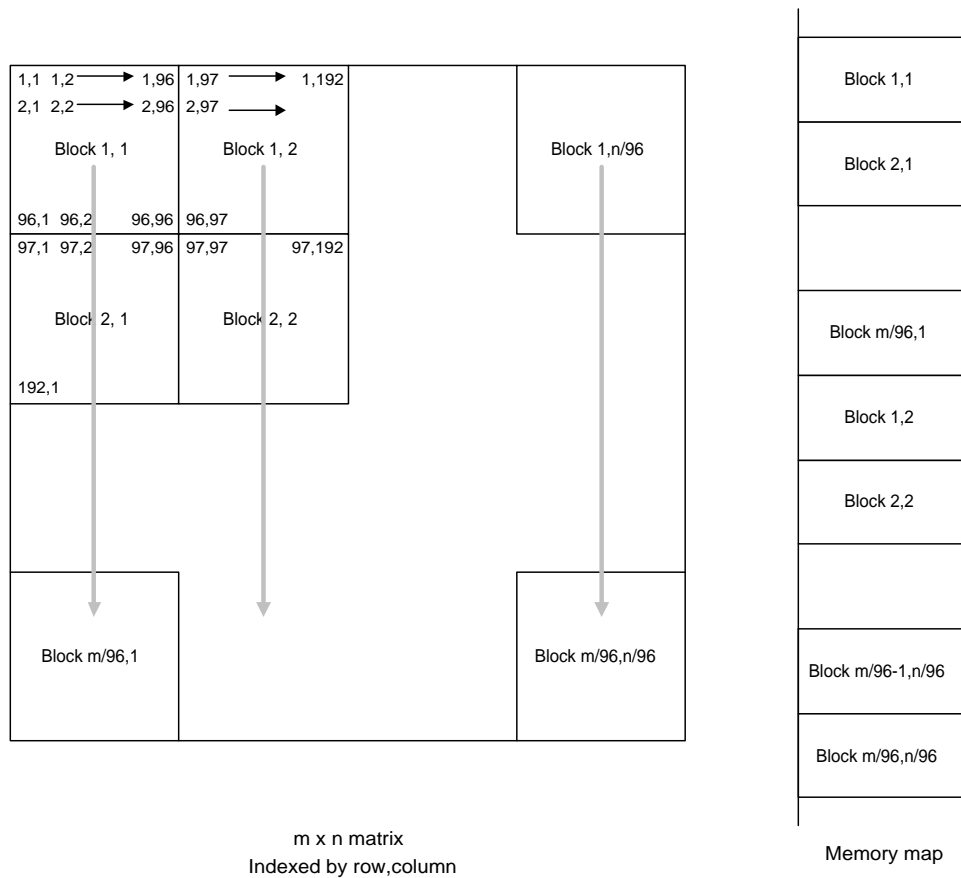
The data must be organized in block format in memory: a 96 x 96 sub-block of the matrix should be stored in a contiguous section of memory in row-major format and these blocks should be stored in column-major format. The data format is illustrated in [Figure 7](#).

If required, the transpose flag can be used to make the function read the data column-wise.

For convenience, the header file `csxl_blas.h` defines the macros `CSXL_BDGEMM96_BLOCKWIDTH`, `CSXL_BDGEMM96_BLOCKHEIGHT` and `CSXL_BDGEMM96_BLOCKSIZE`.

For example, to compute the array index of an element at row `r`, column `c`:

```
index = (r / CSXL_BDGEMM96_BLOCKHEIGHT) * CSXL_BDGEMM96_BLOCKSIZE +
        (c / CSXL_BDGEMM96_BLOCKWIDTH) * (LDA * CSXL_BDGEMM96_BLOCKSIZE) +
        (r % CSXL_BDGEMM96_BLOCKHEIGHT) * CSXL_BDGEMM96_BLOCKWIDTH +
        (c % CSXL_BDGEMM96_BLOCKWIDTH)
```



**Figure 7. Block memory format**

## BDGEMM96

### Description

The blocked DGEMM (BDGEMM96) implements the operation:

$$C \leftarrow \alpha op(A)op(B) + \beta C$$

where  $op(X)$  is one of:

$$op(X) = X \quad or \quad op(X) = X^T$$

$\alpha$  and  $\beta$  are scalars. A, B and C are matrices with  $op(A)$  an M by K matrix,  $op(B)$  a K by N matrix, and C a M by N matrix.

### Interface

The BDGEMM96 interface is a function called `bdgemm96`, as shown below. This assumes data (input and output) is in a blocked format as described in [5.1.1: Blocked data format \(b96\)](#).

```

int bdgemm96(char transA,    // Flag selecting op(A)=transpose(A) or op(A)=A
             char transB,    // Flag selecting op(B)=transpose(B) or op(B)=B
             int block_M,    // Number of block rows in op(A) and in C
             int block_N,    // Number of block columns in op(B) and in C
             int block_K,    // Number of block columns/rows in op(A)/op(B)
             double alpha,   // Scalar multiplier applied to op(A)*op(B)
             double *A,      // Pointer to the matrix A
             int block_LDA,  // Leading dimension of matrix containing A (in blocks)
             double *B,      // Pointer to the matrix B
             int block_LDB,  // Leading dimension of matrix containing A (in blocks)
             double beta,    // Scalar multiplier applied to C (on input)
             double *C,      // Pointer to the matrix C
             int block_LDC)  // Leading dimension of matrix containing A (in blocks)

```

### Parameters

The terms in this code fragment are defined as follows:

#### **transA**

An uppercase or lowercase character where:

- 'N' indicates that matrix A should not be transposed before the multiplication ( $op(X)=X$ ).
- 'T' indicates that matrix A should be transposed before the multiplication ( $op(X)=X^T$ ).
- 'C' indicates that matrix A should be transposed before the multiplication ( $op(X)=X^T$ ).

*Note:* 'C' indicates the conjugate transpose ( $X^*$ ) should be taken, but for non-complex numbers this is equivalent to the transpose ( $X^*=X^T$ ).

- All other characters are invalid and an error is reported (see section 3.1).

#### **transB**

As for **transA**, but applied to matrix B instead of matrix A.

#### **block\_M**

The number of block rows in matrix op(A) and in matrix C. If op(A) is a M by K matrix, then block\_M is M/96.

#### **block\_N**

The number of block columns in matrix op(B) and in matrix C. If op(B) is a K by N matrix, then block\_N is N/96.

#### **block\_K**

The number of block columns in matrix op(A) and block rows in matrix op(B). If op(A) is a M by K matrix, then block\_K is K/96.

**alpha**

The product  $op(A)op(B)$  is multiplied by the scalar alpha as described in the formula above.

**A**

Pointer to the matrix A.

**block\_LDA**

The leading dimension of the matrix A (in blocks) as declared in the calling program. When transA is 'N',  $block\_LDA \geq \max(1, block\_M)$ ; otherwise  $block\_LDA \geq \max(1, block\_K)$

**B**

Pointer to the matrix B.

**block\_LDB**

The leading dimension of the matrix B (in blocks) as declared in the calling program. When transb is 'N',  $block\_LDB \geq \max(1, block\_K)$ ; otherwise  $block\_LDB \geq \max(1, block\_N)$

**beta**

The input matrix C is multiplied by the scalar beta as described in the formula above.

**C**

Pointer to the matrix C.

**block\_LDC**

The leading dimension of the matrix C (in blocks) as declared in the calling program.  
 $block\_LDC \geq \max(1, block\_M)$

To illustrate the semantics of the leading dimension parameters, consider a  $p \times q$  matrix contained within a larger matrix with leading dimension LD (blocks), as shown in [Figure 8](#). If A is the start address of the  $p \times q$  matrix, then the start address of an arbitrary block B (block  $i,j$ ) is given by:

$$address_B = (P + i + j \times LD) \times blocksize$$

where `blocksize` is block height x block width x `sizeof(double)`, that is,  $96 \times 96 \times 8 = 73728$ .

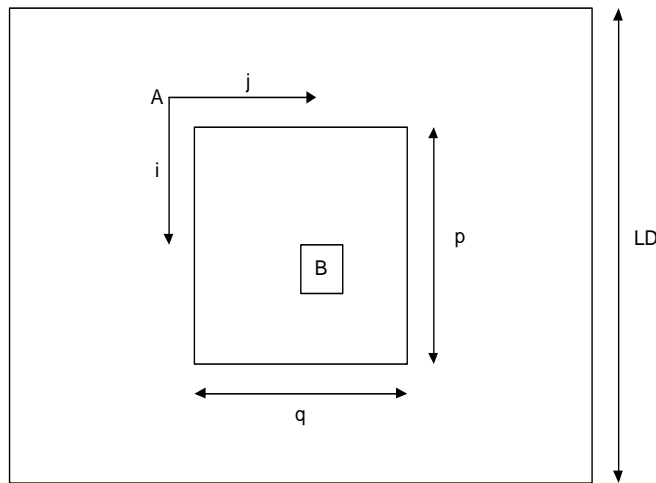


Figure 8. Using leading dimensions to address within a matrix

### 5.1.2 Notes

#### Errors

The function will report errors in the same manner as a standard DGEMM, using the `xerbla` call which will print messages in the form:

"On entry to `bdgemm96`, parameter `n` had an illegal value"

to the standard output. In addition the function will return a nonzero value if an error has occurred, or zero if the functional completed normally.

#### User Requirements

You must set up a poly stack size sufficient for the DGEMM to set up its internal data. The poly stack must be at least 5632 bytes. A sample `.is` file (`bdgemm96_start.is`) is provided which sets up the stack size, see [5.1.3: Example](#).

#### User Limitations

As described in [5.1.1: Blocked data format \(b96\)](#), `bdgemm96` operates on blocks of  $96 \times 96$ ; any multiple of this is acceptable for the `M`, `N`, `K`, `LDA`, `LDB` and `LDC` dimensions. The actual input parameters are expressed as a number of blocks (`block_M = M/96`, `block_LDA = LDA/96`).

The input data must be organized in memory with a block of  $96 \times 96$  elements in a contiguous section of memory, arranged in row-major format (see [Figure 9](#)), although the transpose flag can be used to make `bdgemm96` read the data in column-wise.

#### Semaphores

The `bdgemm96` function uses semaphores that have been set aside for SDK standard libraries (from 96 onwards).

### 5.1.3 Example

An example is included with the CSXL package in the `examples/csxl/cn_bdgemm96` directory. The example compiles (using the makefile provided) to produce a CSX application which calls `bdgemm96` and checks the result.

In addition, the example includes the `bdgemm96_start.is` file which illustrates how to set up the stack.

### 5.1.4 Performance

If the alpha scalar is nonzero, the `bdgemm96` function has its highest performance when neither A nor B is required to be transposed. Transposing A has little effect on performance (slight degradation), however the effect of transposing B is more significant and is illustrated in [Figure 9](#).

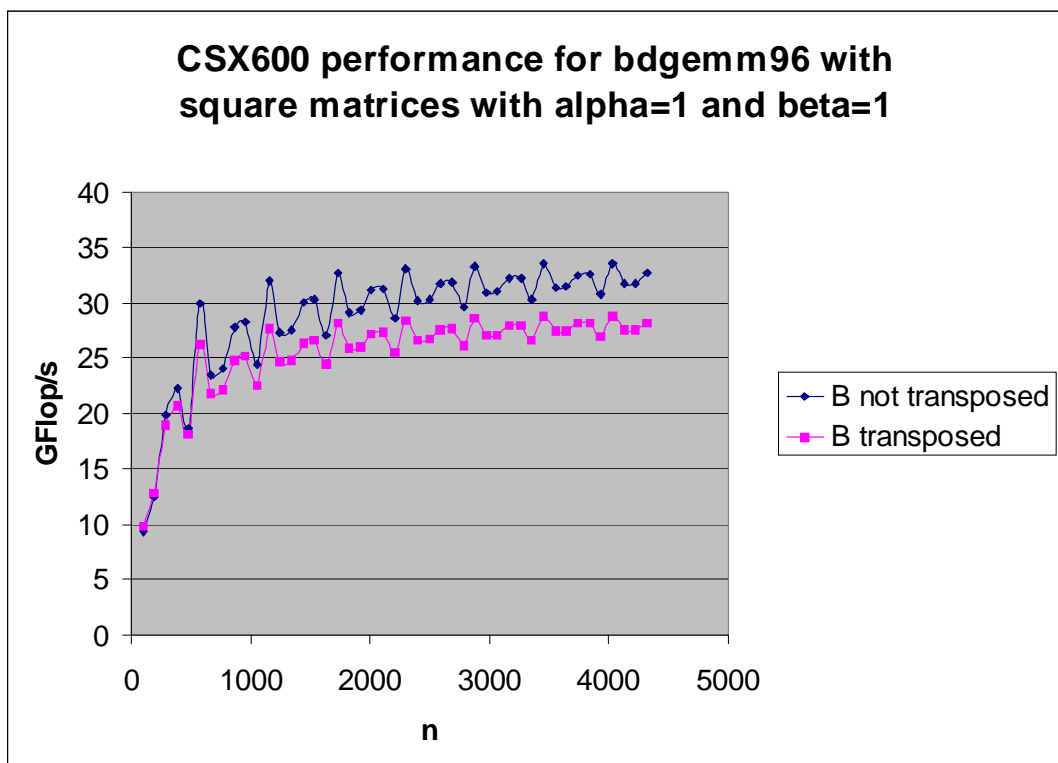


Figure 9. BDGEMM96 performance with square matrices

## 6 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 Introductory Programming Manual*  
Document Number: 06-UG-1117  
ClearSpeed Technology
- [5] *Runtime Software User Guide*  
Document Number: 06-UG-1345  
ClearSpeed Technology
- [6] *SDK Reference Manual*  
Document Number: 06-UG-1136  
ClearSpeed Technology

**ClearSpeed Technology, Inc.**  
3031 Tisch Way, Suite 200  
San Jose, CA 95128  
United States of America

Tel: +1 408 557 2067  
Fax: +1 408 557 9054

Email: [info@clearspeed.com](mailto:info@clearspeed.com)

Web: <http://www.clearspeed.com>

Support: <http://support.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

1. Information and data contained in this document, together with the information contained in any and all associated ClearSpeed documents including without limitation, data sheets, application notes and the like ('Information') is provided in connection with ClearSpeed products and is provided for information only. Quoted figures in the Information, which may be performance, size, cost, power and the like are estimates based upon analysis and simulations of current designs and are liable to change.
2. Such Information does not constitute an offer of, or an invitation by or on behalf of ClearSpeed, or any ClearSpeed affiliate to supply any product or provide any service to any party having access to this Information. Except as provided in ClearSpeed Terms and Conditions of Sale for ClearSpeed products, ClearSpeed assumes no liability whatsoever.
3. ClearSpeed products are not intended for use, whether directly or indirectly, in any medical, life saving and/ or life sustaining systems or applications.
4. The worldwide intellectual property rights in the Information and data contained therein is owned by ClearSpeed. No license whether express or implied either by estoppel or otherwise to any intellectual property rights is granted by this document or otherwise. You may not download, copy, adapt or distribute this Information except with the consent in writing of ClearSpeed.
5. The system vendor remains solely responsible for any and all design, functionality and terms of sale of any product which incorporates a ClearSpeed product including without limitation, product liability, intellectual property infringement, warranty including conformance to specification and or performance.
6. Any condition, warranty or other term which might but for this paragraph have effect between ClearSpeed and you or which would otherwise be implied into or incorporated into the Information (including without limitation, the implied terms of satisfactory quality, merchantability or fitness for purpose), whether by statute, common law or otherwise are hereby excluded.
7. ClearSpeed reserves the right to make changes to the Information or the data contained therein at any time without notice.

© Copyright ClearSpeed Technology plc 2007, 2008. All rights reserved.

Advance is a registered trademark of ClearSpeed Technology plc

ClearSpeed, ClearConnect, Advance and the ClearSpeed logo are trade marks or registered trade marks of ClearSpeed Technology plc. All other brands and names are the property of their respective owners.