# DISTRIBUTED BLAS

## User's Guide

*AUSTRALIAN NATIONAL UNIVERSITY*

*Third Edition : May 1997*

# Contents

# 1 Introduction

The Distributed BLAS (DBLAS) gives an implementation of the Basic Basic Linear Algebra Subroutine library (BLAS) [?] for block-cyclic distributed matrices on the Fujitsu AP1000/AP+ distributed memory multiprocessors. This release of the DBLAS has the following features:

- general parallel algorithms for block-cyclic distributed matrix distribution, which are efficient for all ranges of matrix sizes, minimizing communication startup overhead and volume, while maximizing load balance and cell computation performance.

  The exception are that there are some block alignment requirements, and the symmetric matrix routines are not optimal for when the size of the symmetric matrix is much smaller than that of the other matrix operands.

- a C language interface, and also a Fortran language interface, which is identical to that of the Parallel BLAS (PBLAS) [?].

- can be compiled to interface to the BLAS and BLACS (Basic Linear Algebra Communication Subroutines) libraries, enabling full portability.

- high cell computation performance codes for Sparc versions 7 and 8 architectures.

- relatively small code size and software overheads.

Let {`REGD`} denote the root directory for the DBLAS software. This directory contains the following sub-directories:

- `lib` contains compiled object archives of the DBLAS procedures, suitable for linking to cell programs.

  `DDBLAS.a` is the archive file for the double precision DBLAS; `DDPBLAS.a` is that for the PBLAS front-end.

  `DOUBLEBLASInnerLoopsV4.a` and `DOUBLEBLASInnerLoopsV5.a` are Sparc (version 7) BLAS-like computation kernels. The sources for these are not included in the release, so these files should not be deleted.

- `doc` contains source files for this document and the *Guide to Function* for the DBLAS. Postscript versions of these documents can be made using the provided `Makefile` via the command `make documents`.

- `man` contains the sources for the DBLAS on-line man pages (C interface).

- `src` contains the source code for the DBLAS procedures. The source code is written in ANSI C.

  The file `D_BLAS.h` gives detailed pre/post-conditions for the DBLAS procedures.

  The file `Makefile` can be used to install DBLAS object libraries in the directory {`REGD`}`/lib`.

  The file `Makefile_Generic` contains system-dependent macro definitions used by the makefiles in this and other directories.

  The `README` file gives a description of the contents of the other files. This directory includes the sub-directories:

  - `PBLAS` contains the source files and makefile for the PBLAS (Fortran) interface.
  - `xy` contains the object files and makefile for the stride-ApLib libraries for the AP1000 and AP+.

- `examples` contains test programs to call, time and check the DBLAS procedures.

  `Makefile_Generic` sets up general make macros and targets for the compilation of the host and cell programs with the DBLAS. For the BLAS and BLACS versions, the macros `FTN_BLACS_LIB` and `EXT_FTN_BLAS_LIB` should be redefined to be the names of the local (Fortran) BLACS and BLAS libraries, respectively, together with and other libraries that these in turn require (eg. a Fortran maths library). `Makefile` customizes `Makefile_Generic` for this directory.

  `Gen.host.c` is a generic ApLib host program, similar to the `hap` package.

  `GMM.cell.c` is the source of the test program for the matrix multiply DBLAS procedures (level 1, 2 and 3). The elements of matrices have values given by a linear function of their global indices, so that they can be efficiently generated and checked (independently of any matrix multiply algorithm) on the cells. Command line parameters determine which DBLAS operation is used, and its parameters – see the C include file `GMM.Parameters.c` for full details.

  `TRSM.cell.c` is the source of the test program for the triangular matrix update/solve DBLAS procedures (level 2 and 3). In the case of `DDTRSM (.. ALPHA, A, ..., B, ...)`, $B$ is normally initialized so that the expected result is $\alpha I$; in the case of `DDTRMM`, $B$ is normally initialized to $I$, so that the expected result is $\alpha A'$. However, the `-c` option can be used for the initialization of $B$ to random values; this requires other DBLAS routines (eg.`DDGEMM`) to be reliably implemented. The command line parameters determine which DBLAS operation is used, and its parameters – see the C include file `TRSM.Parameters.c` for the full details.

  The `README` file briefly describes the auxiliary C files required for the test programs.

  The file `sample_runs` shows examples on how to invoke the test programs, and can be executed to test an installation.

As mentioned in the accompanying *Description of Function*, DBLAS-based matrix LU, LLT and QR factorization routines have been developed using the DBLAS and have shown high performance on the AP1000 and AP+. As methods to improve these algorithms further are still under investigation, their codes, and the accompanying test program, are not included in this release. However, they can be obtained by request from the author.

# 2 Environment Requirements and Limitations

There are no limitations on matrix size set by the DBLAS. However, the DBLAS require workspaces of size bounded by $2C_0^{1/2}N$, where $C_0$ is the cache size and $N$ is the largest local matrix size. These workspaces are allocated from dynamic memory; if insufficient memory is available, the DBLAS procedures will abort with an error message.

The following mainly applies to the DBLAS installed on the AP/AP+ platforms, under a version of the ApLib.

For the 'stride ApLib' and ApLinux versions, the logical AP configuration must be a sub-mesh of the physical AP configuration.

For multi-task cell programs, only one task can use the DBLAS at a time, as they use the AP1000's `xy`-communication procedures (which currently have no task specifier), and confusion of `xy` messages between tasks could occur.

The ring buffer size should be set to at least 256 KB, although in some applications (eg. large matrix factorizations with a large matrix block size), a ring buffer of 512 KB may be required. Alternatively, the `MachineParams` variable `OPT_MSG_SIZE_NUM` (see Section 3.2.1) can be reduced if a smaller ring buffer size is desired; this will cause the DBLAS procedures to send messages in smaller pieces.

The DBLAS are by default compiled and executed under the standard AP1000/AP+ system `sys.c7`. However, any version of an AP system may be used, provided the DBLAS are compiled and executed under the same system. In this case, the Unix environment variable `AP1000HOME` should be set to the appropriate directory. The environment `AP1000CCOMPILER` should be set to `gcc`.

| value of COMMUN | name | platforms |
|---|---|---|
| -DAPLIB_STRIDE | stride ApLib | AP1000 only (default on AP_HOST) |
| -DAPLIB_PLUS_STRIDE | stride ApLib | CellOS AP+ only |
| -DAPLIB_LINUX | AP+ Linux | Linux AP+ only (default on APP_HOST) |
| -DAPLIB_STD | standard ApLib | (AP1000, AP+, Sparc SMP) |
| -DFTN_BLACS | BLACS | any platform on which Fortran BLACS is available |
| '' | single processor | any Unix workstation |

Table 1: Versions of Communication Libraries available for the DBLAS

# 3 Installation Guide

The file {REGD}/src/Makefile_Generic contains the system-dependent makefile definitions. This is suitable for both Unix system 4 and 5 platforms. This section describes the main customizations that may be required; there are also other customizations that can be performed; see {REGD}/src/Makefile_Generic for more details. These macros can also be set on the make command line, if desired.

In this release, only double precision DBLAS is supported. However, most of the DBLAS sources and makefiles do have allowances for single precision versions of all routines. The macro FDATA controls this; it has a default value DOUBLE, for double precision compilation.

## 3.1 Installation on AP/AP+ Platforms

Firstly, the macro AP_HOST (APP_HOST) should be set to the name of the font-end to the AP1000 (AP+) to install the library on.

Secondly, if gcc is not available on the front-end, the macro BASECC may be set to any ANSI C compiler [1]. The macros CCFarch and CCFLAGS must also be set accordingly; in particular, -Dinline='' must be added to CCFLAGS if the compiler does not recognize gcc's inline function keyword.

Thirdly, an appropriate version of communication library for the DBLAS must be chosen, and the COMMUN macro must be set to the corresponding value, as described in Table 1. These are listed in order of expected performance on AP/AP+ platforms. Compared with the 'stride ApLib' versions, for some situations, the standard ApLib version will have more communication startups, and the BLACS versions will have more memory copying overheads. Note that for the standard ApLib version on the AP1000, program execution using the -LSEND option is recommended for performance.

To compile for the AP1000 Simulator (CASIM), the standard ApLib library should be selected, with the macro CELLCC being set to mkt.

Finally, an appropriate value of the COMPUTE macro must be chosen. The default is '', which means that the DBLAS cell computation procedures interface directly to BLAS-like computational kernels optimized for Sparc version 7 and 8 cell architectures (but not UltraSparc). The alternative value is -DEXT_FTN_BLAS, which means the DBLAS cell computation procedures interface directly to a standard Fortran BLAS library, which must be available on the platform.

There are a few other system-dependent macros that also may need adjustment. The macro CC_NOALIGN is only important for installation of the standard ApLib version on the AP1000 (and if the -LSEND option is not used for program execution). If this is not the case, CC_NOALIGN can be set to the value of BASECC. If however this is the case, double word alignment of the incoming messages cannot be guaranteed, and to handle this problem, matrix copy and add routines (see {REGD}/src/FDATACellMatCopy.h) that can handle word-aligned data must be provided. Under Solaris/SunOS the cc compiler (without the -dalign option) can be used to compile such versions

[1] Only the file {REGD}/src/MatDim.c does not compile under pre-ANSI C compilers.

of these routines. If this compiler is not available, an alternate compiler that has the same capability will be required, and the CC_NOALIGN and FD_NO_ALIGN_FLAGS macros be set accordingly.

## 3.2 Installation on non-AP/AP+ Platforms

The BLAS [2] and BLACS versions of the DBLAS will be required in this case. The macros BASECC, CCFarch, CCFLAGS, COMMUN and COMPUTE must be set according to the conditions on this platform, as described in the previous section.

On different platforms, there are different conventions for calling Fortran from C. Currently, only the conventions where names of Fortran symbols are in lower case with an underscore added (the Solaris convention), and in lower case with the underscore omitted (this will require -DC_CALL_FTN_NO_CHANGE added to the COMPUTE macro; this is the convention for IBM RS series) are supported. Supporting other conventions should only require modification to the Fortran procedure header files in {REGD}/src and {REGD}/src/PBLAS. It is also assumed that character parameters do not need to be converted between C and Fortran.

For the BLAS version of the DBLAS library, it is also recommended to add -DNO_PARTITION to the value of the COMPUTE, unless it is known that that the BLAS implementation is inefficient on large matrices. Omitting -DNO_PARTITION will cause the DBLAS cell computation procedures to partition local cell matrix multiplications to optimize cache usage. This in turn will require the MachineParams to be appropriately tuned. In the current implementation, the BLAS routine DGEMM() is used even for level 2 matrix multiplications. Thus, it is thus fairly important that this routine be efficient for this case, and also be of low overhead.

In order to compile the test programs, the macro CELLCC must be set to the platform's version of C compiler for parallel cell programs.

Also, in {REGD}/examples/Makefile_Generic, the macros FTN_BLACS_LIB and EXT_FTN_BLAS_LIB must be set to the name(s) of the BLACS and BLAS libraries. As well as the BLAS/BLACS libraries themselves, any other libraries that these in turn require, that are not implicitly linked in by CELLCC, must be included. For example, using BLAS on Solaris 2 platforms from the SunSoft Performance Library requires EXT_FTN_BLAS_LIB to be set to libsunperf.a libsunmath.a. As these test programs are in C, the standard Fortran library must be linked in; its specification must be put in the value of the LIB77 macro (default value is -dn -lF77).

The files Host.h and Cell.h in {REGD}/examples/ contains platform-dependent macros that may need to be modified. The relevant section of Cell.h is where the FTN_BLACS cpp macro is active. Note that the macro HOST in {REGD}/examples/Makefile_Generic should be set to -DHOST if a host program (eg. Gen.host) is needed on the platform, and be set to empty otherwise.

The macro CACHESIZE should also be set to size of the platforms outer-level cache (in KB). This sets the most important parameter of the MachineParams module. Other parameters may also need to be set, as described in the next section.

### 3.2.1 Tuning the DBLAS via the MachineParams module

The file {REGD}/MachineParams.h contains (integer) machine-dependent parameters, which are currently set to appropriate values for AP/AP+ platforms. These parameters depend on the cell characteristics of the target platform, such as (outer-level) cache size and number of registers (ultimately some of the minor parameters depend on the implementation of the cell BLAS being used). This section explains how to customize the values of these parameters for the target platform; the task will be easiest for scalar-based platforms.

The most important parameter is $C_0$ = CACHESIZE, which is set by the Makefile macro of the same name, as mentioned previously. Note however that the former is in units of the number of (double) floating point words, not in KB. The value of $C_0$ is important in the DBLAS cell matrix multiply routines (unless compiled with -DNO_PARTITION, in which case it has little direct effect).

[2] Unless the platform happens to be Sparc (versions 7 and 8) based.

The parameters CACHE_ASSOC and CACHE_LINE_SIZE should be set to the associativity and line size cache (in bytes) of this cache; however, in the current release of the DBLAS, these are not used.

The parameter $S_0 =$ OPT_MSG_SIZE is used only in FDATADistMatrix.c to enable blocking of large messages. The main motivation of this is for systems where incoming messages have to be stored in system buffers (eg. the AP/AP+ ring buffers), the size of unread messages should be not exceed $2S_0$. In some situations, explicit synchronization is required with the blocking to achieve this. The current implementation unfortunately is not able to take advantage of pipelining of messaging via blocking to any significant extent. Setting $S_0 \approx C_0/2$ may also have some performance benefits on some platforms, if received messages are cached. To suppress message blocking, set $S_0 = \infty$ (eg. 1000,000,000).

The parameter MAX_DBLAS_WS_SIZE currently has no effect.

The parameters $K_0 =$ INNER_VECSIZE, $M_0 =$ OUTER_VECSIZE and $N_0 =$ MINOR_OUTER_VECSIZE are important for tuning. These should be set to the (smallest) set of values such that a cell matrix multiply $C = AB$, where $C$ is $M_0 \times N_0$ and $A$ is $M_0 \times K_0$, gives near-optimal performance. The default values of $M_0$ and $K_0$ are set such that $2K_0 = C_0^{1/2}$ and $M_0 = \frac{C_0}{2K_0}$. For most scalar-based architectures, this setting will be appropriate. The optimum value of $N_0$ can be determined empirically, but for most platforms, it will be between 2 and 8. It can be over-estimated without major performance loss, but it should not be under-estimated. Even if the -DNO_PARTITION compilation switch is used, the values of $M_0, N_0$, and $K_0$ are important for the triangular cell matrix multiply routines (DDSYR(2)K() for $M_0$ and $N_0$; DDSYMM() for $K_0$). Note that $M_0$ and $K_0$ are also the default values of $M_0'$ and $K_0'$ below.

For a cell matrix multiply with a very small value of $K$, ie. $K \leq K_0'$, the underlying BLAS routines may switch algorithms, for example to use a rank-1 update based method. In this case, a larger value of $N_0$ may be required. The parameters $N_0' =$ MINOR_OUTER_VECSIZE_K0 and $K_0' =$ K0_MINOR_OUTER_VECSIZE are introduced for this purpose. Their values are only important for DDSYR(2)K() and DDSYR(2)().

Variants of the parameters $M_0$ and $K_0$, here called $M_0'$ and $K_0'$ (having the same names, but prefixed with PAR_) are also available. These are used to partition the global matrix dimensions for the spread/reduce operations in the DBLAS top-level driver routines. By default, $M_0' = M_0$ and $K_0' = K_0$ which will be optimal for most situations; however $M_0 \leq M_0' \leq 2M_0$ and $K_0 \leq K_0' \leq 2K_0$ may be possibly be useful in reducing communication startups, where $B_0$, the matrix distribution blocksize, is of comparable size to $K_0$ or $M_0$.

$V_0 =$ MIN_VECSIZE should be set to smallest vector length for 'reasonable' performance for something like a vector copy. It is used to help determine which loop index will be the inner index in the matrix copy, add and scale routines in {REGD}/src/FDATACell{Matrix,MatCopy}.c. For most scalar based platforms, $8 \leq V_0 \leq 32$ would be appropriate. The default value is 16.

Example: for a vector-based platform with no cache and no limits on message size, the settings:

$$C_0 = S_0 = \infty, M_0 = 1024, N_0 = K_0 = 8, N_0' = N_0, K_0' = 0, V_0 = 128$$

might be appropriate, with the other parameters set to their default values.

The -M $p_n$ $p_v$ option test program GMM.cell can be used to dynamically set the value of the $p_n$th parameter of the module to the value $p_v$ (see Section 5). This is only effective if the test program and the DBLAS are compiled using the -DTUNE switch (which is the default). See {REGD}/MachineParams.h to determine the relationship between $p_n$ and the above parameters. Alternatively, from ScaLAPACK programs, the call:

```
CALL DDBLASSETGET(0, 0, pn, pv)
```

can be used; see {REGD}/src/PBLAS/PBLASCheck.h for details on using this procedure, and other tuning procedures specific for the PBLAS interface.

# 4 Compiling and Linking

From the directory {REGD}/src, the command:

```
make install
```

installs the file DDBLAS.a in {REGD}/lib. This archive will also contain the stride ApLib routines and Sparc BLAS-like computational kernels, provided that they are required by the values of the COMMUN and COMPUTE macros

The command:

```
make envts
```

gives a listing of all of the major makefile macros that can be set by the make command line, with a list of possible valid values.

C cell programs using the DBLAS must be linked with {REGD}/lib/DDBLAS.a. These should also be linked using the -lm option.

From {REGD}/examples, the command:

```
make GMM.cell TRSM.cell
```

will compile the C test programs, and also install the DBLAS library {REGD}/lib/DDBLAS.a.

For systems that require a host program (eg. AP systems excluding the AP+ under Linux), the command:

```
make Gen.host
```

will compile the generic host program. In this case, the cell programs should also be compiled with -DHOST included in the compilation command.

The above indicates how to install the DBLAS enabling run-time internal error checking (with procedure tracing ability) and machine-dependent parameter tuning. However, the former in particular introduces some overhead in both execution time (appreciable only for relatively small computations) and code size.

From the directory {REGD}/src, the command:

```
make COMMON='' install
```

or alternatively, from the directory {REGD}/examples, the command:

```
make COMMON='' GMM.cell TRSM.cell
```

will install a version of the DBLAS without internal error checking or run-time machine-dependent parameter tuning.

See Section 6 for how to run a test suite on a DBLAS installation.

## 4.1 Using the PBLAS Interface in ScaLAPACK applications

ScaLAPACK has been ported to the AP1000, and a BLACS library is available [?]. Once the DBLAS library is installed, the command

```
make install
```

should also be performed from {REGD}/src/PBLAS. This installs the file {REGD}/lib/DDPBLAS.a.

When linking a ScaLAPACK implementation, the order of linking should be:

```
{REGD}/lib/DDPBLAS.a {REGD}/lib/DDBLAS.a PBLAS.a
```

where PBLAS.a is the name of ScaLAPACK's Parallel BLAS archive. This is required since {REGD}/lib/DDPBLAS.a does not include the PBLAS Tools procedures (including the PTOPSET/GET and PBERROR routines, which are called from {REGD}/lib/DDPBLAS.a).

Thus, {REGD}/lib/DDPBLAS.a can be used as an alternative PBLAS implementation. If desired, it is possible to 'mix-and-match' between the two implementations, by deleting the appropriate members of the {REGD}/lib/DDPBLAS.a archive. The command:

```
nm xdxtst | grep UseDBLAS
```

where xdxtst is the name of the ScaLAPACK executable, will indicate which of the PBLAS routines that are DBLAS-based.

### 4.2 Troubleshooting

By default, the DBLAS will be compiled with internal error checking on and a procedure call tracing facility present. This is recommended for the first tests on an installation, and the preliminary tests of any new application using the DBLAS. It is also recommended when errors in the DBLAS are suspected.

Tracing can be turned on using the -t option in the test programs (see Section 5) or by manipulating the trace variables (see {REGD}/src/GenAssert.h).

On AP-systems where a choice of communication libraries are available (see Table 1), re-compiling and running the application under the alternate communication library is also suggested when errors are suspected.

If a DBLAS procedure crashes, or aborts due to an internal error detection, an execution trace (on one of the cells where this occurs) will be useful. Note that a cell exit code of 12 indicates an internal DBLAS error (which should never occur, except possibly in the case of a memory allocation failure, or non-zero value of the AP communication error variable cap_errno (whether the setting of that variable was caused by the DBLAS code or not)).

When a DBLAS procedure detects an error in one of its parameters, a cell exit code of 13 is indicated.

## 5 Explanation of Input Parameters to the DBLAS and the DBLAS Test Programs

See Section 2 of the companion document *Distributed BLAS: Description of Function* for an explanation of the input parameters to the DBLAS procedures themselves. See also Sections 1 and 2 of the *Description of Function* for the mathematical representation of a DistMat parameter, eg. $A = (t, (\mathtt{Ax}, I_{0..1}), D_{0..1})$ where $D_i = (B0_i, B_i, P0_i, P_i)$, which will be used hereafter. These quantities may be superscripted, eg. $t^A$, were necessary to indicate which DistMat operand they refer to.

{REGD}/examples/Gen.host has the usage:

```
Gen.host Ncely Ncelx [caren options] <cell program and parameters>
```

which initiates the cell program with a Ncely $\times$ Ncelx configuration.

The cell program {REGD}/examples/GMM.cell generally performs the distributed matrix computation $C \leftarrow \alpha AB + \beta C$, with $A$ – $M \times K$, $C$ – $M \times N$. It has the usage:

GMM.cell [-a [-RUL] [-TN] [-C] [-O AIO AJO]] [-b [-RUL] [-TN] [-C] [-O BIO BJO]]
[-c [-RUL] [-C] [-O CIO CJO]] [-B BM BN BK] [-e ry rx] [-2] [-l lvl]
[-t trlvl trtype] [-M $p_n$ $p_v$] $MNK$ [$\alpha$ [$\beta$]]

with default values:

`AI0 = AJ0 = BI0 = BJ0 = CI0 = CJ0 = 0;` $\alpha$ = $\beta$ `= 1.0; BM = BN = BK = 1; ry = 2.0, rx = 1.0; lvl = 3; trlvl=-1 trtype=0`

and allowable values:

`0 <= AI0, BI0, CI0, AJ0, BJ0, CJ0; 0 <=` $M, N, K$`; 1 <= BM, BN, BK`
`1 <= lvl < 3; 0 <= trtype <= 5`

It has matrix form options corresponding to the variables:

`-a -R`: `FormA = Rectangular` (default)
`-a -U`: `FormA = Upper` (upper triangular matrix)
`-a -L`: `FormA = Lower` (lower triangular matrix)

(and similarly for `-b ... FormB`, `-c ... FormC`. Note: the `-U` or `-L` options signify that only the upper or lower half, respectively, of the matrix is to be accessed). Only one matrix operand can be non-rectangular.

It has matrix transpose options corresponding to the variables:

`-a -N`: $t^A = 0$ (default)
`-a -T`: $t^A = 1$

(and similarly for `-b ...` $t^B$). If `-C` is specified, the local matrix storage will be column-major (ie. $I_0 = 1$), otherwise it will be row-major (ie. $I_1 = 1$).

Here the block sizes of each matrix are given by:

$$B^A_{t^A} = B^C_0 = \texttt{BM},\ B^A_{1-t^A} = B^B_{t^B} = \texttt{BK},\ B^B_{1-t^B} = B^C_1 = \texttt{BN}$$

and the processor dimension sizes are given by:

$$P^A_0 = P^B_0 = P^C_0 = \texttt{NCely},\ P^A_1 = P^B_1 = P^C_1 = \texttt{NCelx}$$

and the block and processor offsets are given by:

$$\texttt{AI0} = I0^A_0 B^A_0 + B0^A_0,\ \texttt{AJ0} = I0^A_1 B^A_1 + B0^A_1 \ \text{ where } \ P0^A_i = (P_i + \texttt{cid}_i - I0^A_i)\%P_i$$

with $(I0^A_0, I0^A_1)$ being the absolute id of the leading cell, and $(\texttt{cid}_0, \texttt{cid}_1)$ being the absolute id of the current cell. This applies similarly for $B$ and $C$.

Global element $A_{i,j}$ will be initialized to $(\texttt{AI0}+i)*\texttt{ry}+(\texttt{AJ0}+j)*\texttt{rx}$, and similarly for $B_{i,j}, C_{i,j}$.

If `FormA != Rectangular`, $A$ is a symmetric matrix and requires `ry = rx` and also `AI0 = AJ0` (and similarly for $B$ and $C$). Also, note that current DBLAS alignment restrictions also require that $B^A_0 = B^A_1$.

The value of `lvl` can be used to specify which level of the procedure will be called. When `lvl = 2` (and $\min(M,N,K) = 1$), a level 2 procedure will be used. When `lvl = 1` (and two of $M, N, K$ are 1), a level 1 procedure will be used.

If all matrix operands are rectangular, `DDGEMM()` / `DDGEMV()` / `DDGER()` / `DDDOT()` / `DDAXPY()` will be invoked (depending on the values of `lvl` and $M, N, K$). `FormA, FormB != Rectangular` will invoke `DDSYMM() / DDSYMV()`. `FormC != Rectangular` will invoke `DDSYRK() / DDSYR()`; here we will have $B = A^T$ (and require `BI0=AI0, BJ0=AJ0`). If the `-2` option is specified, `DDSYR2K() / DDSYR2()` will be invoked instead, with $B$ and $A$ being independent matrices.

`trlvl` and `trtype` can be set to turn on DBLAS tracing. `trlvl<0` (the default) will suppress tracing. `trlvl=0` will enable tracing of the outer-level DBLAS calls; `trlvl>0` will enable tracing up to a depth of `trlvl` internal calls (6 is the maximum useful value). Generally `trtype=0` (trace all categories of procedures) will be adequate.

The `-M` $p_n$ $p_v$ can be used for the tuning of the `MachineParams` parameters. This can be used repeatedly with different values of $p_n$; the $p_n$th parameter will be set to $p_v$ (only effective if the

DBLAS is compiled with the `-DTUNE` option). This should be used with care; there is currently no 'sanity check' that the `MachineParams` parameters are consistent with each other.

Examples of how to call this program will be given in the next section. See {`REGD`}`/examples/GMM.Parameters.c` for a full listing of all command line options.

The `GMM.cell` program works as follows. First, it allocates memory of $A, B$ and $C$ and initializes them. Then the corresponding DBLAS procedure is called. The call's speed (in MFLOPs/cell) is reported. The values of the result matrix ($C$) are then checked against the expected values (these values can be generated very efficiently and without any communication). It then reports whether any errors were found.

The cell program {`REGD`}`/examples/TRSM.cell` has the usage:

```
TRSM.cell [-f FN] [-S Side] [-U UpLo] [-T TransA] [-D Diag] [-B BM BN BA]
          [-O AIO AJO BIO BJO] [-M n v] [-t trlvl trtype] [-c] [-L lvl] M [N [α]]
```

with default values:

```
M = N, α = 1.0, FN = 'S', Side = 'L', UpLo = 'U', Diag = 'U', TransA = 'N',
AIO = AJO = BIO = BJO = 0; BM = BN = BA = 1; lvl = 3; trlvl=-1, trtype=0
```

and allowable values:

```
FN = 'S', 'M'; Side = 'L', 'R'; UpLo = 'U', 'L'; Diag = 'U', 'N';
TransA = 'N', 'T'; 0 <= M,N; 0 <= AIO, BIO, AJO, BJO; 1 <= BM, BN, BA;
```

It performs the computation $B \leftarrow \alpha AB$, if (`FN != 'S'`), and $B \leftarrow \alpha A^{-1}B$, otherwise. Here $B$ – $M' \times N'$, where $M' = M, N' = N$, if `Side = 'L'`, and $M' = N, N' = M$ otherwise.

The command-line parameters correspond to the ordinary BLAS interface, which is different for `Side = 'R'` to the DBLAS interface. The following table indicates the relationship between $t^A$ and $t^B$ and the command line parameters:

| `Side` | `TransA` | $t^B$ | $t^A$ |
|---|---|---|---|
| 'L' | 'N' | 0 | 0 |
| 'L' | 'T' | 0 | 1 |
| 'R' | 'N' | 1 | 1 |
| 'R' | 'T' | 1 | 0 |

`UpLo` is the same as `FormA` in `GMM.cell`. `Diag = 'U'` means the diagonal of $A$ will be assumed to be unity.

Here the block sizes of each matrix are given by:

$$B_0^B = \texttt{BM},\ B_1^B = \texttt{BN},\ B_{t^A}^A = B_{t^B}^B,\ B_{1-t^A}^A = \texttt{BA}$$

and the processor dimension sizes are given by:

$$P_0^A = P_0^B = \texttt{NCely},\ P_1^A = P_1^B = \texttt{NCelx}$$

and similarly for $B$. `AIO`, ..., `BJO` have the same meaning as for `GMM.cell`.

The variables `lvl`, `trlvl`, `trtype`, $p_n$ and $p_v$ have the same meaning as in `GMM.cell`.

For `lvl = 3` and `FN = 'S'`, the computation `DDTRSM`(`UpLo`$'$, `Diag`, $M', N', \alpha, A, B$) will be invoked. where `UpLo' = UpLo`, if $t^A = 0$, and <the reverse of `UpLo`> otherwise. Examples of how to call this program will be given in the next section. See {`REGD`}`/examples/TRSM.Parameters.c` for a full listing of all command line options.

The `TRSM.cell` program works as follows. First, it allocates memory of $A$ and $B$ and initializes them. $A$ is initialized to random values; $B$ is initialized to $A$, for `DDTRSM() / DDTRSV()`, and to $I$, for `DDTRMM() / DDTRMV()`. Then the corresponding DBLAS procedure is called. The call's speed (in MFLOPs/cell) is reported. The values of the result matrix ($B$) are then checked against the expected value ($\alpha I$ for `DDTRSM() / DDTRSV()`, and $\alpha A$ otherwise). It then reports whether any errors were found.

If the `-c` option is specified, $B$ is initialized to random values $B_0$. In the case of `DDTRMM()/DDTRMV()`, the result $B = \alpha AB_0$ is computed; the unused half of $A$ is then padded with zeros, and $B' = \alpha AB_0$ is computed using `DDGEMM()` and checked against the final value of $B$. In the case of `DDTRSM() / DDTRSV()`, $B$ is initialized to $AB_0$ using `DDTRMV()`, and the final value of $B$ is checked against $\alpha^{-1}B_0$. The `-c` option is recommended for when $\min(M,N) \ll \max(M,N) \leq 500$; its main disadvantages are that it requires more execution time for checking and that for $\max(M,N) > 500$, numerical inaccuracies can accumulate to an extent that they may exceed the current error thresholds in `TRSM.cell`.

Both `GMM.cell` and `TRSM.cell` create log files of the form `Log<program name>.<date>` in the directory that they are run in. It reports the command line parameters, speeds and error checking information. The file {REGD}`/examples/LogIO.h` can be modified to suppress this, if desired.

# 6 Examples and Testing the Installation

The file {REGD}`/examples/example_runs` contains a series of invocations of the above test programs, together with the resulting DBLAS procedure call. The $P0_i$ parameters are those for cell (0,0); note that their effective value is $P0_i\%P_i$. The commands below must be prefixed by a host program specification and a cell configuration specification (eg. `Gen.host -RB 256` $P_0$ $P_1$, or if on AP Linux, `prun -n` $P_1$`x`$P_0$). Any values of $P_0$ and $P_1$ can be chosen that are compatible with the physical cell configuration, provided sufficient cell memory is available.

The command:

```
{REGD}/examples/example_runs [-l] [P1 ...  PN]
```

where the `-l` option must be used for AP Linux systems (only), will perform a testing of the DBLAS under various (small) processor configurations. If included, `[P1 .. PN]` should be a series of cell configuration specifications of the form $P_0$`x`$P_1$.

For workstation versions of the DBLAS (compiled with `COMMUN=''`), the command:

```
{REGD}/examples/example_runs -w
```

will perform the tests (on a logical $1 \times 1$ configuration, which is the only configuration available for this version).

`GMM.cell 1000 1000 250 -1.0`
calls `DDGEMM (1000, 1000, 250, -1.0,` $A$, $B$, `1.0,` $C$`)`
where $t^A = t^B = 0, D_i^A = D_i^B = D_i^C = (0,1,0,P_i), I_1^A = I_1^B = I_1^C = 1$

`GMM.cell -a -T -B 2 2 2 -b -C 100 1000 1000 1.0 0.0`
calls `DDGEMM (100, 1000, 1000, 1.0,` $A$, $B$, `0.0,` $C$`)`
where $t^A = 1, t^B = 0, D_i^A = D_i^B = D_i^C = (0,2,0,P_i), I_1^A = I_0^B = I_1^C = 1$

`GMM.cell -b -T -B 1 3 2 -c -C 1000 10 1000`
calls `DDGEMM (1000, 10, 1000, 1.0,` $A$, $B$, `1.0,` $C$`)`
where $t^A = 0, t^B = 1, D_0^A = (0,1,0,P_0), D_1^A = (0,2,0,P_1), D_0^B = (0,3,0,P_0), D_1^B = (0,2,0,P_1),$
$D_0^C = (0,1,0,P_0), D_1^C = (0,3,0,P_1), I_1^A = I_1^B = I_0^C = 1$

`GMM.cell -a -T -C -b -T -c -C -B 8 8 8 400 500 119`
calls `DDGEMM (400, 500, 119, 1.0,` $A$, $B$, `1.0,` $C$`)`
where $t^A = t^B = 1, D_i^A = D_i^B = D_i^C = (0,8,0,P_i), I_0^A = I_1^B = I_0^C = 1$

`GMM.cell -a -O 1 2 -b -O 4 8 -c -O 3 2 -B 2 3 4 456 237 35 2.0 -1.0`
calls `DDGEMM (456, 237, 35, 2.0,` $A$, $B$, `-1.0,` $C$`)`
where $t^A = 0, t^B = 0, D_0^A = (1,2,0,P_0), D_1^A = (2,4,0,P_1), D_0^B = (0,4,P_0-1,P_0),$
$D_1^B = (2,3,P_1-2,P_1), D_0^C = (1,2,P_0-1,P_0), D_1^C = (2,3,0,P_1), I_1^A = I_1^B = I_1^C = 1$

`GMM.cell -a -O 2 3 -T -b -T -O 5 6 -c -O 4 5 -B 2 3 4 663 837 110 -1.0`
calls `DDGEMM (663, 837, 110, -1.0,` $A$, $B$, `1.0.,` $C$`)`
where $t^A = 1, t^B = 1, D_0^A = (2,4,0,P_0), D_1^A = (1,2,P_1-1,P_1), D_0^B = (2,3,P_0-1,P_0),$
$D_1^B = (2,4,P_1-1,P_1), D_0^C = (0,2,P_0-2,P_0), D_1^C = (2,3,P_1-1,P_1), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -l 2 -B 2 2 2 1000 1 250 -1.0 0.0
```
calls `DDGEMV (1000, 250, -1.0,` $A$`,` $B$`, 1.0,` $C$`)`
where $t^A = t^B = 0, D_i^A = D_i^B = D_i^C = (0, 2, 0, P_i), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -l 2 1000 250 1 -1.0
```
calls `DDGER (1000, 250, -1.0,` $A$`,` $B$`, 1.0,` $C$`)`
where $t^A = 0, t^B = 1, D_i^A = D_i^B = D_i^C = (0, 1, 0, P_i), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -l 1 -a -T -B 8 8 8 1 1 1000 1.0 0.0
```
calls $C_{0,0}$ `= DDDOT (1000,` $A$`,` $B$`)`
where $t^A = t^B = 0, D_i^A = D_i^B = D_i^C = (0, 8, 0, P_i), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -l 1 -B 5 5 5 1 1000 1
```
calls `DDAXPY (1000,` $A_{0,0}$`,` $B$`,` $C$`)`
where $t^A = t^B = 0, D_i^B = D_i^C = (0, 5, 0, P_i), I_1^B = I_1^C = 1$

```
GMM.cell -e 1 1 -a -U -C 297 98 297 -1.0 0.0
```
calls `DDSYMM ("Upper", 297, 98, -1.0,` $A$`,` $B$`, 0.0,` $C$`)`
where $t^A = t^B = 0, D_i^A = D_i^B = D_i^C = (0, 1, 0, P_i), I_0^A = I_1^B = I_1^C = 1$

```
GMM.cell -e 1 1 -B 2 4 4 -b -L 98 297 297
```
calls `DDSYMM ("Lower", 297, 98, 1.0,` $B$`,` $A$`, 1.0,` $C$`)`
where $t^A = t^C = 1, t^B = 0, D_0^A = D_0^B = (0, 4, 0, P_0), D_1^A = (0, 2, 0, P_1),$
$D_0^C = (0, 2, 0, P_0), D_1^B = D_1^C = (0, 4, 0, P_1), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -e 1 1 -l 2 -B 4 2 4 -a -L -0 4 4 -b -T -0 1 0 597 1 597
```
calls `DDSYMV ("Lower", 597, 1.0,` $A$`,` $B$`, 1.0,` $C$`)`
where $t^A = t^C = 0, t^B = 1, D_0^A = (0, 4, P_0 - 1, P_0), D_1^A = (0, 4, P_1 - 1, P_1), D_0^B = (1, 2, 0, P_0),$
$D_1^B = (0, 4, 0, P_1), D_0^C = (0, 4, 0, P_0), D_1^C = (0, 2, 0, P_1), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -B 4 4 1 -b -T -c -L 511 511 88 -1.0
```
calls `DDSYRK ("Lower", 511, 88, -1.0,` $A$`, 1.0,` $C$`)`
where $t^A = 0, D_0^A = D_0^C = (0, 4, 0, P_0), D_1^A = (0, 1, 0, P_1), D_1^C = (0, 4, 0, P_1), I_1^A = I_1^C = 1$

```
GMM.cell -a -T -C -c -U 300 300 32
```
calls `DDSYRK ("Upper", 300, 32, -1.0,` $A$`, 1.0,` $C$`)`
where $t^A = 1, D_0^A = D_0^C = (0, 1, 0, P_0), D_1^A = D_1^C = (0, 1, 0, P_1), I_0^A = I_1^C = 1$

```
GMM.cell -l 2 -B 4 4 1 -a -T -0 0 7 -b -0 0 7 -c -U -0 7 7 310 310 1 -1.0
```
calls `DDSYR ("Upper", 310, -1.0,` $A$`,` $C$`)`
where $t^A = 1, D_0^A = (0, 1, 0, P_0), D_1^A = D_1^C = (3, 4, P_1 - 1, P_1), D_0^C = (3, 4, P_1 - 1, P_0), I_1^A = I_1^C = 1$

```
GMM.cell -a -T -c -L -C -2 300 300 32 -1.0
```
calls `DDSYR2K ("Lower", 300, 32, -1.0,` $A$`,` $B$`, 1.0,` $C$`)`
where $t^A = t^B = 1, D_i^A = D_i^B = D_i^C = (0, 1, 0, P_i), I_1^A = I_1^B = I_0^C = 1$

```
GMM.cell -B 1 1 4 -a -0 1 1 -b -T -c -U -2 320 320 37
```
calls `DDSYR2K ("Upper", 300, 37, -1.0,` $A$`,` $B$`, 1.0,` $C$`)`
where $t^A = t^B = 0, D_0^A = (0, 1, P_0 - 1, P_0), D_1^A = (1, 4, 0, P_1), D_0^B = D_0^C = (0, 1, 0, P_0), D_1^B = (0, 4, 0, P_1)$
$D_1^C = (0, 1, 0, P_1), I_1^A = I_1^B = I_1^C = 1$

```
GMM.cell -l 2 -B 3 3 1 -a -C -b -T -C -c -L -2 411 411 1 1.0
```
calls `DDSYR2 ("Lower", 411, 1.0,` $A$`,` $B$`,` $C$`)`
where $t^A = t^B = 0, D_0^A = D_0^B = D_0^C = (0, 3, 0, P_0), D_1^A = D_1^B = (0, 1, 0, P_1), D_1^C = (0, 3, 0, P_1),$
$I_0^A = I_0^B = I_1^C = 1$

```
TRSM.cell -c -f M -S L -T T -U U -D N 88 400
```
calls `DDTRMM("Lower", "NonUnit", 88, 400, 1.0,` $A$`,` $B$`)`
where $t^A = 1, t^B = 0, D_i^A = D_i^B = D_i^C = (0, 1, 0, P_i), I_1^A = I_1^B = 1$

```
TRSM.cell -c -B 2 1 2 -f M -S R -T N -U L -D N 500 64 -1.0
```
calls `DDTRMM("Upper", "NonUnit", 64, 500, -1.0,` $A$, $B$`)`
where $t^A = t^B = 1, D_0^A = (0,2,0,P_0), D_1^A = (0,1,0,P_1), D_0^B = (0,2,0,P_0), D_1^B = (0,1,0,P_1)$
```
TRSM.cell -c -L 2 -B 4 2 4 -f M -S L -T N -U L -D U 800 1
```
calls `DDTRMV("Lower", "Unit", 800,` $A$, $B$`)`
where $t^A = 0, t^B = 1, D_0^A = (0,4,0,P_0), D_1^A = (0,4,0,P_1), D_0^B = (0,4,0,P_0), D_1^B = (0,2,0,P_1)$
```
TRSM.cell -c -L 2 -B 3 3 3 -f M -S R -T T -U U -D N 1 500
```
calls `DDTRMV("Upper", "NonUnit", 500,` $A$, $B$`)`
where $t^A = 0, t^B = 1, D_i^A = D_i^B = (0,3,0,P_i)$,

```
TRSM.cell -c -f S -S L -T N -U L -D U 88 500 -1.0
```
calls `DDTRSM("Lower", "Unit", 88, 500, -1.0,` $A$, $B$`)`
where $t^A = t^B = 0, D_i^A = D_i^B = D_i^C = (0,1,0,P_i), I_1^A = I_1^B = 1$
```
TRSM.cell -c -f S -B 2 2 2 -S L -T T -U L -D N 500 64
```
calls `DDTRSM("Upper", "NonUnit", 500, 64, 1.0,` $A$, $B$`)`
where $t^A = 1, t^B = 0, D_i^A = D_i^B = D_i^C = (0,2,0,P_i)$
```
TRSM.cell -c -B 2 1 2 -f S -S R -T T -U L -D N 500 64
```
calls `DDTRSM("Lower", "NonUnit", 64, 500, 1.0,` $A$, $B$`)`
where $t^A = 0, t^B = 1, D_0^A = (0,1,0,P_0), D_1^A = (0,2,0,P_1), D_0^B = (0,2,0,P_0), D_1^B = (0,1,0,P_1)$
```
TRSM.cell -c -B 2 2 2 -0 2 2 3 0 -f S -S R -T N -U L -D U 47 400
```
calls `DDTRSM("Upper", "Unit", 400, 47, 1.0,` $A$, $B$`)`
where $t^A = t^B = 1, D_0^A = (0,2,P_0-1,P_0), D_1^A = (0,2,P_1-1,P_1), D_0^B = (1,2,P_0-1,P_0), D_1^B = (0,2,0,P_1)$

```
TRSM.cell -c -L 2 -B 4 2 4 -f S -S L -T T -U U -D N 800 1
```
calls `DDTRSV("Lower", "NonUnit", 800,` $A$, $B$`)`
where $t^A = 0, t^B = 1, D_0^A = (0,4,0,P_0), D_1^A = (0,4,0,P_1), D_0^B = (0,4,0,P_0), D_1^B = (0,2,0,P_1)$
```
TRSM.cell -c -L 2 -B 3 3 3 -f S -S R -T N -U U -D U 1 500
```
calls `DDTRSV("Lower", "Unit", 500,` $A$, $B$`)`
where $t^A = t^B = 1, D_i^A = D_i^B = (0,3,0,P_i)$,

## 7 Contact about this Software

When users have questions about the DBLAS, or when users encounter difficulties in using the DBLAS please contact:
Peter Strazdins, Department of Computing Science, Australian National University
E-mail: Peter.Strazdins@anu.edu.au

## References


[1] R. P. Brent and P. E. Strazdins. Implementation of BLAS Level 3 and LINPACK Benchmark on the AP1000. *Fujitsu Scientific and Technical Journal*, 29(1):61–70, 1993.

[2] J. Choi, J. Demmel, J. Dhillon, J. Dongarra, S. Oustrouchov, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley. ScaLAPACK: A Portable Linear Algebra Library for Distributed Memory Computers - Design Issues and Performance. Technical Report CS-95-283, Computer Science Dept, University of Tennessee, Knoxville, 1995.

[3] J. Choi, J. Dongarra, S. Oustrouchov, A. Petitet, and D. Walker amd R. C. Whaley. A Proposal for a Set of Parallel Basic Linear Algebra SubPrograms. Technical Report CS-95-292, Computer Science Dept, University of Tennessee, Knoxville, 1995.

[4] S. J. Hammarling J. Dongarra, J. J. Du Croz and I. S. Duff. A set of Level 3 Basic Linear Algebra Subprograms. *ACM Transactions on Mathematical Software (16)*, pages 1 – 17, 1990.

[5] A. Rendell. Porting ScaLAPACK (and PBLAS) to the Fujitsu AP1000. Technical report, ANU Supercomputing Facility, Australian National University, 1995.

[6] P. E. Strazdins and R.P. Brent. Implementing Blas Level 3 on the AP1000. In *Proceedings of the CAP Workshop '91*, pages H–1 – H–17, Australian National University, Canberra, November 1991.

[7] P.E. Strazdins. Prototyping Parallel LAPACK using Block-Cyclic Distributed BLAS. In *Third Parallel Computing Workshop for the Fujitsu PCRF*, pages P1–R–1 – P1–R–7, Kawasaki, November 1994.

[8] P.E. Strazdins. Matrix Factorization using Distributed Panels on the Fujitsu AP1000. In *IEEE First International Conference on Algorithms And Architectures for Parallel Processing (ICA3PP-95)*, pages 263–73, Brisbane, April 1995.

[9] P.E. Strazdins. A High Performance, Portable Distributed BLAS Implementation. In *Fifth Parallel Computing Workshop for the Fujitsu PCRF*, pages P2–K–1 – P2–K–10, Kawasaki, November 1996.