Heterogeneous ScaLAPACK Programmers' Reference and Installation Manual

Heterogeneous ScaLAPACK

Parallel Linear Algebra Programs for Heterogeneous Networks of Computers

Version 1.1.0

Ravi Reddy, Alexey Lastovetsky, Pedro Alonso

E-mail: Manumachu.Reddy@ucd.ie, Alexey.Lastovetsky@ucd.ie, palonso@dsic.upv.es

1 May 2009

CONTENTS

1 Introduction

This manual presents Heterogeneous ScaLAPACK, which provides the following high performance parallel linear algebra programs for Heterogeneous Networks of Computers (HNOCs) supporting MPI [1]:

- Dense linear system solvers
- Least squares solvers
- Eigenvalue solvers

The fundamental building blocks of Heterogeneous ScaLAPACK are:

- ScaLAPACK [2]
- PBLAS [3]
- BLAS [4]
- BLACS [5]
- HeteroMPI [6]

The rest of the manual is organized as follows. Section 2 presents the model of a sample Heterogeneous ScaLAPACK program. Section 3 presents the Heterogeneous ScaLAPACK user interface. Section 4 provides the command-line interface to build and run Heterogeneous ScaLAPACK applications. This is followed by installation instructions for UNIX/LINUX platforms in section 5.

2 What is Heterogeneous ScaLAPACK

Heterogeneous ScaLAPACK is a package which provides high performance parallel basic linear algebra programs for HNOCs. It is built on the top of ScaLAPACK software using the multiprocessing approach and thus reuses it completely. The multiprocessing approach can be summarized as follows:

- The whole computation is partitioned into a large number of equal chunks;
- Each chunk is performed by a separate process;
- The number of processes run by each processor is as proportional to its speed as possible.

Thus, while distributed evenly across parallel processes, data and computations are distributed unevenly over processors of the heterogeneous network so that each processor performs the volume of computations proportional to its speed.

To summarize the essential differences between calling a ScaLAPACK routine and a heterogeneous ScaLAPACK routine, consider the four basic steps involved in calling a PDGESV ScaLAPACK routine as shown in Figure 1.

1. Initialize the process grid using `Cblacs_gridinit`;
2. Distribute of the matrix on the process grid. Each global matrix that is to be distributed across the process grid is assigned an array descriptor using the ScaLAPACK TOOLS

```
   int main(int argc, char **argv) {
      int nprow, npcol, pdgesvctxt, myrow, mycol, c__0 = 0, c__1 = -1;
/* Problem parameters */
      int  *N, *NRHS, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *INFO;
      double *A, *B, *IPIV;
/* Initialize the process grid */
      Cblacs_get(c__1, c__0, &pdgesvctxt);
      Cblacs_gridinit(&pdgesvctxt, "r", nprow, npcol);
      Cblacs_gridinfo(pdgesvctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A and B */
      descinit_(DESCA, …, &pdgesvctxt);  /* for Matrix A */
      descinit_(DESCB, …, &pdgesvctxt);  /* for Matrix B */
/* Distribute matrices on the process grid using user-defined pdmatgen */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix A */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix B */
/* Call the PBLAS 'pdgesv' routine */
      pdgesv_(N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO);
/* Release the process grid and Free the BLACS context */
      Cblacs_gridexit(pdgesvctxt);
/* Exit the BLACS */
      Cblacs_exit(c__0);
   }
```

Figure 1. Basic steps involved in calling the ScaLAPACK routine **`PDGESV`**.

routine `descinit`. A mapping of the global matrix onto the process grid is accomplished using the user-defined routine `pdmatgen`;

3. Call the ScaLAPACK routine `pdgesv`;
4. Release the process grid via a call to `Cblacs_gridexit`. When all the computations have been completed, the program is exited with a call to `Cblacs_exit`.

Figure 2 shows the essential steps of the Heterogeneous ScaLAPACK program calling the ScaLAPACK PDGESV routine, which are:

1. Initialize the Heterogeneous ScaLAPACK runtime using using the operation

   ```
   int hscal_init(int * argc, int *** argv)
   ```

   where `argc` and `argv` are the same as the arguments passed to `main`. This routine must be called before any other Heterogeneous ScaLAPACK context management routine and must be called once. It must be called by all the processes running in the Heterogeneous ScaLAPACK application;
2. Get the Heterogeneous ScaLAPACK PDGESV context using the routine **`hscal_pdgesv_ctxt`**. The function call **`hscal_in_ctxt`** returns a value of 1 for the processes chosen to execute the PDGESV routine or otherwise 0;
3. Execute the steps (2) and (3) involved in calling the ScaLAPACK PDGESV routine (shown in Figure 1);
4. Release the context using the context destructor operation

   ```
   int hscal_free_ctxt(int * ctxt);
   ```

```
   int main(int argc, char **argv) {
      int nprow, npcol, pdgesvctxt, myrow, mycol, c__0 = 0;
/* Problem parameters */
      int  *N, *NRHS, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *INFO;
      double *A, *B, *IPIV;
/* Initialize the heterogeneous ScaLAPACK runtime */
      hscal_init(&argc, &argv);
/* Initialize the array descriptors for the matrices A and B
   No need to specify the context argument */
      descinit_(DESCA, …, NULL);  /* for Matrix A */
      descinit_(DESCB, …, NULL);  /* for Matrix B */
/* Get the heterogeneous PDGESV context */
      hscal_pdgesv_ctxt(N, NRHS, IA, JA, DESCA,
                        IB, JB, DESCB, &pdgesvctxt);
      if (!hscal_in_ctxt(&pdgesvctxt)) {
         hscal_free_ctxt(&pdgesvctxt);
         hscal_finalize(c__0);
         return 0;
      }
/* Retrieve the process grid information */
      blacs_gridinfo__(&pdgesvctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A and B */
      descinit_(DESCA, …, &pdgesvctxt);  /* for Matrix A */
      descinit_(DESCB, …, &pdgesvctxt);  /* for Matrix B */
/* Distribute matrices on the process grid using user-defined pdmatgen */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix A */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix B */
/* Call the PBLAS 'pdgesv' routine */
      pdgesv_(N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO);
/* Release the heterogeneous PDGESV context */
      hscal_free_ctxt(&pdgesvctxt);
/* Finalize the Heterogeneous ScaLAPACK runtime */
      hscal_finalize(c__0);
      return 0;
   }
```

Figure 2. Basic steps of the Heterogeneous ScaLAPACK program calling the ScaLAPACK routine **`PDGESV`**.

5. When all the computations have been completed, the program is exited with a call to **`hscal_finalize`**, which finalizes the heterogeneous ScaLAPACK runtime.

It is relatively straightforward for the application programmers to wrap the steps (2) to (4) in a single function call, which would form the heterogeneous counterpart of the ScaLAPACK PDGESV routine. It can also be seen that the application programmers need not specify the process grid arrangement for the execution of the Heterogeneous ScaLAPACK program employing the ScaLAPACK routine, as it is automatically determined in the context constructor routine. Apart from this, the only other major rewriting that the application programmers must perform is the redistribution of matrix data from the process grid arrangement used in the ScaLAPACK program to the process grid arrangement automatically determined in the heterogeneous ScaLAPACK program. The matrix redistribution/copy routines [7, 8], provided

```
   int main(int argc, char **argv) {
      int nprow, npcol, pdgesvctxt, pdposvctxt, myrow, mycol, c__0 = 0;
/* Problem parameters */
      char   *UPLO; int    *N, *NRHS, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *INFO;
      double *A, *B, *IPIV;
/* Initialize the heterogeneous ScaLAPACK runtime */
      hscal_init(&argc, &argv);
/* Initialize the array descriptors for the matrices A and B*/
      descinit_(DESCA, …, NULL);  /* for Matrix A */
      descinit_(DESCB, …, NULL);  /* for Matrix B */
/* Get the heterogeneous PDGESV context */
      hscal_pdgesv_ctxt(N, NRHS, IA, JA, DESCA,
                        IB, JB, DESCB, &pdgesvctxt);
      if (hscal_in_ctxt(&pdgesvctxt)) {
/* Retrieve the process grid information */
         Cblacs_gridinfo(pdgesvctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A and B */
         descinit_(DESCA, …, &pdgesvctxt);  /* for Matrix A */
         descinit_(DESCB, …, &pdgesvctxt);  /* for Matrix B */
/* Distribute matrices on the process grid using user-defined pdmatgen */
         pdmatgen_(&pdgesvctxt, …); /* for Matrix A */
         pdmatgen_(&pdgesvctxt, …); /* for Matrix B */
/* Call the PBLAS 'pdgesv' routine */
         pdgesv_(N, NRHS, A, IA, JA, DESCA, IPIV,
                 B, IB, JB, DESCB, INFO);
      }
/* Release the heterogeneous PDGESV context */
      hscal_free_ctxt(&pdgesvctxt);
/* Initialize the array descriptors for the matrices A and B*/
      descinit_(DESCA, …, NULL);  /* for Matrix A */
      descinit_(DESCB, …, NULL);  /* for Matrix B */
/* Get the heterogeneous PDPOSV context */
      hscal_pdposv_ctxt(UPLO, N, NRHS, IA, JA, DESCA,
                        IB, JB, DESCB, &pdposvctxt);
      if (hscal_in_ctxt(&pdposvctxt)) {
/* Retrieve the process grid information */
         Cblacs_gridinfo(pdposvctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A and B */
         descinit_(DESCA, …, &pdposvctxt);  /* for Matrix A */
         descinit_(DESCB, …, &pdposvctxt);  /* for Matrix B */
/* Distribute matrices on the process grid using user-defined pdmatgen */
         pdmatgen_(&pdposvctxt, …); /* for Matrix A */
         pdmatgen_(&pdposvctxt, …); /* for Matrix B */
/* Call the PBLAS 'pdposv' routine */
         pdposv_(UPLO, N, NRHS, A, IA, JA, DESCA,
                 B, IB, JB, DESCB, INFO);
      }
/* Release the heterogeneous PDPOSV context */
      hscal_free_ctxt(&pdposvctxt);
/* Finalize the Heterogeneous ScaLAPACK runtime */
      hscal_finalize(c__0);
   }
```

Figure 3. Basic steps of the Heterogeneous ScaLAPACK program calling two ScaLAPACK routines **`PDGESV`** and **`PDPOSV`**.

by the ScaLAPACK package for each data type, can be used to achieve this redistribution. These routines provide a truly general copy from any block cyclicly distributed (sub)matrix to any other block cyclicly distributed (sub)matrix. In our future work, we would address this issue of the cost of data redistribution.

Now assume that application programmer has a ScaLAPACK application, which employs more than one ScaLAPACK routine (in this case two routines PDGESV and PDPOSV), then the Figure 3 shows the main steps of the Heterogeneous ScaLAPACK application.

3 Heterogeneous ScaLAPACK Library Interface

In this section, we describe the interfaces to the routines provided by Heterogeneous ScaLAPACK.

3.1 Heterogeneous ScaLAPACK runtime initialization and finalization

`hscal_init`

Initializes Heterogeneous ScaLAPACK runtime system

Synopsis:

```
int
hscal_init
(
    int* argc,
    char*** argv
)
```

Parameters:

`argc` --- Number of arguments supplied to **`main`**
`argv` --- Values of arguments supplied to **`main`**

Description: All processes must call this routine to initialize Heterogeneous ScaLAPACK runtime system. This routine must be called before any Heterogeneous ScaLAPACK context management routine. It must be called at most once; subsequent calls are erroneous.

Usage:

```
int main(int argc, char** argv)
{
   int rc =  hscal_init(
                 &argc,
                 &argv
   );

   if (rc != HSCAL_SUCCESS)
   {
      //Error has occurred
```

```
        }
     }
```

Return values: **`HSCAL_SUCCESS`** on success.

`hscal_finalize`

Finalizes Heterogeneous ScaLAPACK runtime system

Synopsis:

```
int
hscal_finalize
(
    int exitcode
)
```

Parameters:

`exitcode` --- code to be returned to the command shell

Description: This routine cleans up all Heterogeneous ScaLAPACK state. All processes must call this routine at the end of processing tasks. Once this routine is called, no Heterogeneous ScaLAPACK routine (even **`hscal_init`**) may be called.

Usage:

```
int main(int argc, char** argv)
{
    int rc =  hscal_init(
                  &argc,
                  &argv
    );

    if (rc != HSCAL_SUCCESS)
    {
       //Error has occurred
    }

    rc =  hscal_finalize(0);

    if (rc != HSCAL_SUCCESS)
    {
       //Error has occurred
    }
}
```

Return values: **`HSCAL_SUCCESS`** on success.

3.2 Heterogeneous ScaLAPACK Context Management Functions

The main routine is the context creation function, which provides a context for the execution of the ScaLAPACK routine. There is a context creation function for each and every ScaLAPACK routine. This function frees the application programmer from having to specify the process grid arrangement to be used in the execution of the ScaLAPACK routine. It tries to determine the optimal process grid arrangement.

3.2.1 Naming Scheme

All the routines have names of the form **`hscal_pxyyzzz_ctxt`**. The second letter, **`x`**, indicates the data type as follows:

x	**MEANING**
s	Single precision real data
d	Double precision real data
c	Single precision complex data
z	Double precision complex data

Thus **`hscal_pxgesv_ctxt`** refers to any or all of the routines **`hscal_pcgesv_ctxt`**, **`hscal_pdgesv_ctxt`**, **`hscal_psgesv_ctxt`** and **`hscal_pzgesv_ctxt`**.

The next two letters, **`yy`**, indicate the type of matrix (or of the most significant matrix).

`ge` - `ge`neral
`sy` - `sy`mmetric
`he` - `he`rmitian
`tr` - `tr`iangular

The last three letters **`zzz`** indicate the computation performed. Thus **`hscal_pcgels_ctxt`** indicates a context routine for the ScaLAPACK routine **`pcgels`**, which solves overdetermined or underdetermined complex linear systems.

3.2.2 Routines

The Heterogeneous ScaLAPACK and the Heterogeneous PBLAS context creation routines are tabulated below. Only the names are displayed.

Level 1 PBLAS	Level 2 PBLAS	Level 3 PBLAS
hscal_pxswap_ctxt	hscal_pxgemv_ctxt	hscal_pxgemm_ctxt
hscal_pxscal_ctxt	hscal_pxhemv_ctxt	hscal_pxsymm_ctxt
hscal_pxcopy_ctxt	hscal_pxsymv_ctxt	hscal_pxhemm_ctxt
hscal_pxaxpy_ctxt	hscal_pxtrmv_ctxt	hscal_pxsyrk_ctxt
hscal_pxdot_ctxt	hscal_pxtrsv_ctxt	hscal_pxherk_ctxt
hscal_pxdotu_ctxt	hscal_pxger_ctxt	hscal_pxsyr2k_ctxt
hscal_pxdotc_ctxt	hscal_pxgeru_ctxt	hscal_pxher2k_ctxt
hscal_pxnrm2_ctxt	hscal_pxgerc_ctxt	hscal_pxtran_ctxt
hscal_pxasum_ctxt	hscal_pxher_ctxt	hscal_pxtranu_ctxt
hscal_pxamax_ctxt	hscal_pxher2_ctxt	hscal_pxtranc_ctxt
	hscal_pxsyr_ctxt	hscal_pxtrmm_ctxt
	hscal_pxsyr2_ctxt	hscal_pxtrsm_ctxt
		hscal_pxgeadd_ctxt
		hscal_pxtradd_ctxt

ScaLAPACK
hscal_pxgesv_ctxt
hscal_pxgetrs_ctxt
hscal_pxgetrf_ctxt
hscal_pxgeqrf_ctxt
hscal_pxgelqf_ctxt
hscal_pxposv_ctxt
hscal_pxpotrs_ctxt
hscal_pxpotrf_ctxt
hscal_pxgels_ctxt

For example, the context creation function for the PDGESV routine has an interface, which is shown below:

`hscal_pdgesv_ctxt`

Create a heterogeneous context for the execution of PDGESV routine

Synopsis:

```
int hscal_pdgesv_ctxt(
    int * n, int * nrhs,
    int * ia, int * ja, int * desca,
    int * ib, int * jb, int * descb,
    int * octxt)
```

Parameters:

`octxt` --- output context handle to the group of MPI processes

Description: It differs from the PDGESV call in the following ways:

- It returns a context but does not actually execute the PDGESV routine;
- The input arguments are the same as for the PDGESV call except
  - The matrices *A*, *B* and *C* containing the data are not passed as arguments;
  - The context element in the descriptor arrays **`desca`** and **`descb`** need not be filled.
- The output arguments differ as follows:
  - The vector **`ipiv`** and **`info`** are not passed;
  - It has an extra return argument, **`ictxt`**, which contains the handle to a group of MPI processes that is subsequently used in the actual execution of the PDGEMM routine;
  - A return value of **`HSCAL_SUCCESS`** indicates successful execution.

It is a collective operation and must be called by all the processes running in the Heterogeneous ScaLAPACK application. The context contains a handle to a group of MPI processes, which tries to execute the ScaLAPACK routine faster than any other group of processes. This context can be reused in multiple calls of the same routine or any routine that uses similar parallel algorithm as PDGESV. During the creation of the group of MPI processes, the Heterogeneous ScaLAPACK runtime system detects the optimal process arrangement as well as solves the problem of selection of the optimal set of processes running on different computers of the heterogeneous network.

Usage:

```
int rc, octxt;

rc = hscal_pdgesv_ctxt(
          n, nrhs,
          ia, ja, desca,
          ib, jb, descb,
          &octxt
);

if (rc != HSCAL_SUCCESS)
{
   return rc;
}
```

Return values: **`HSCAL_CTXT_UNDEFINED`** is returned if the process is not the member of the context represented by the handle **`ictxt`**. **`HSCAL_SUCCESS`** on success.

hscal_in_ctxt

Am I a member of the context?

Synopsis:

```
int
hscal_in_ctxt
(
    int * ictxt
)
```

Parameters:

`ictxt` --- input context handle to the group of MPI processes.

Description: This function returns **`true`** if the process calling this routine is the member of the context represented by the handle **`gid`** otherwise **`false`**.

Usage:

```
int ictxt;

/* Create context */

if (hscal_is_ctxt(&ictxt))
{
   printf("I'm a member of the context\n");
}
else
{
   printf("I'm not a member of the context\n");
}
```

hscal_get_comm

Returns the MPI communicator

Synopsis:

```
MPI_Comm*
hscal_get_comm
(
    int * ictxt
)
```

Parameters:

`ictxt` --- input context handle to the group of MPI processes.

Description: This function returns the MPI communicator.

Usage:

```
int ictxt;

/* Create context */
```

```
if (hscal_is_ctxt(&ictxt))
{
   MPI_Comm* comm = hscal_get_comm(&ictxt);
}
```

Return values: **`NULL`** is returned if the process is not the member of the context represented by the handle **`ictxt`**.

hscal_timeof

Returns the estimated execution time of the ScaLAPACK routine using the optimal process arrangement

Synopsis:

```
double
hscal_timeof
(
    int * ictxt
)
```

Parameters:

`ictxt` --- input context handle to the group of MPI processes.

Description: This function returns the estimated execution time of the ScaLAPACK routine using the optimal process arrangement. This is only the estimated execution time since the ScaLAPACK routine is not actually executed on the underlying hardware. This routine is serial and can be called by any process, which is participating in the context **`ictxt`**.

Usage:

```
int ictxt;

/* Create PDGESV context using hscal_pdgesv_ctxt */

if (hscal_is_ctxt(&ictxt))
{
   double time_of_pdgesv = hscal_timeof(&ictxt));
   printf(
     "PDGESV estimated execution time is %f\n",
     time_of_pdgemm
   );
}
```

`hscal_free_ctxt`

Free the context

Synopsis:

```
int
hscal_free_ctxt
(
    int * ictxt
)
```

Parameters:

`ictxt` --- input context handle to the group of MPI processes

Description: This routine deallocates the resources associated with a group object **`gid`**. **`HMPI_Group_free`** is a collective operation and must be called by all the processes, which are members of the group **`gid`**.

Usage:

```
if (hscal_is_ctxt(&ictxt))
{
   int rc = hscal_free_ctxt(&ictxt);
   if (rc != HSCAL_SUCCESS)
   {
      /* Problems freeing the context */
   }
}
```

Return values: **`HMPI_SUCCESS`** on success and an appropriate error code in case of failure.

3.3 Heterogeneous ScaLAPACK Auxiliary Functions

In addition to the context management routines, auxiliary routines are provided for each ScaLAPACK (and PBLAS) routine, which determine the total number of computations (arithmetical operations) performed by each process and the total number of communications in bytes between a pair of processes involved in the execution of the ScaLAPACK (and PBLAS) routine. An auxiliary routine is also provided for the serial BLAS equivalent of each PBLAS routine, which determines the total number of arithmetical operations involved in its execution. These routines are serial and can be called by any process. They do not actually execute the corresponding SCALAPACK/PBLAS/BLAS routine but just calculate the total number of computations and communications involved.

The naming scheme and the names of the routines are similar to those discussed in the previous sections and tabulated below.

Level 1 PBLAS	**Level 2 PBLAS**	**Level 3 PBLAS**
hscal_pxswap_info	hscal_pxgemv_info	hscal_pxgemm_info
hscal_pxscal_info	hscal_pxhemv_info	hscal_pxsymm_info
hscal_pxcopy_info	hscal_pxsymv_info	hscal_pxhemm_info
hscal_pxaxpy_info	hscal_pxtrmv_info	hscal_pxsyrk_info
hscal_pxdot_info	hscal_pxtrsv_info	hscal_pxherk_info
hscal_pxdotu_info	hscal_pxger_info	hscal_pxsyr2k_info
hscal_pxdotc_info	hscal_pxgeru_info	hscal_pxher2k_info
hscal_pxnrm2_info	hscal_pxgerc_info	hscal_pxtran_info
hscal_pxasum_info	hscal_pxher_info	hscal_pxtranu_info
hscal_pxamax_info	hscal_pxher2_info	hscal_pxtranc_info
	hscal_pxsyr_info	hscal_pxtrmm_info
	hscal_pxsyr2_info	hscal_pxtrsm_info
		hscal_pxgeadd_info
		hscal_pxtradd_info

ScaLAPACK
hscal_pxgesv_info
hscal_pxgetrs_info
hscal_pxgetrf_info
hscal_pxgeqrf_info
hscal_pxgelqf_info
hscal_pxposv_info
hscal_pxpotrs_info
hscal_pxpotrf_info
hscal_pxgels_info

We explain the details of the interface using one example for ScaLAPACK, PBLAS, and BLAS respectively.

hscal_pdgesv_info

Determines the total number of computations (arithmetical operations) performed by each process and the total number of communications in bytes between a pair of processes involved in the execution of the ScaLAPACK PDGESV routine

Synopsis:

```
int hscal_pdgesv_info(
    int * n, int * nrhs,
    int * ia, int * ja, int * desca,
    int * ib, int * jb, int * descb,
    int nprow, int npcol,
    double * tcomp,
    int * tcomm)
```

Parameters:

nprow --- Number of process rows
npcol --- Number of process columns
tcomp --- Array containing the total number of arithmetic operations
tcomm --- Array containing the total volume of communications between pairs of processes

Description: The matrices ***A*** and ***B*** containing the data are not passed as arguments. It has four parameters in addition to those passed to the PDGESV function call. The parameters `(nprow, npcol)` contain the process arrangement, where `nprow` specifies the number of process rows and `npcol` specifies the number of process columns. The return parameter `tcomp` is a 1D array of size `nprow×npcol` logically representing a 2D array of size `[nprow][npcol]`. Its `[i][j]`–th element contains the total number of arithmetical operations performed by the process with coordinates `(i, j)` during the execution of the PDGESV function call. The return parameter `tcomm` is a 1D array of size `nprow×npcol×nprow×npcol` logically representing an array of size `[nprow][npcol][nprow][npcol]`. Its `[i][j][k][l]`–th element contains the total number of bytes communicated between a pair of processes with coordinates `(i, j)` and `(k, l)` respectively during the execution of the PDGESV function call. **HSCAL_SUCCESS** indicating successful execution or otherwise an appropriate error code is the return value.

Usage:

```
int rc, *tcomm;
double *tcomp;

tcomp = (double*)calloc(nprow*npcol, sizeof(double));
tcomm = (int*)calloc(nprow*npcol*nprow*npcol, sizeof(int));

rc = hscal_pdgesv_info(
          n, nrhs,
          ia, ja, desca,
          ib, jb, descb,
          nprow, npcol,
          tcomp, tcomm
);

if (rc != HSCAL_SUCCESS)
{
   /* Problems querying the information */
}

/* Print the computations and communications information */

free(tcomp);
free(tcomm);
```

Return values: **`HSCAL_SUCCESS`** on success.

`hscal_pdgemm_info`

Determines the total number of computations (arithmetical operations) performed by each process and the total number of communications in bytes between a pair of processes involved in the execution of the PBLAS PDGEMM routine

Synopsis:

```
   int hscal_pdgemm_info(
       char* transa, char* transb,
       int * m, int * n, int * k,
       double * alpha,
       int * ia, int * ja, int * desca,
       int * ib, int * jb, int * descb,
       double * beta,
       int * ic, int * jc, int * descc,
       int nprow, int npcol,
       double * tcomp,
       int * tcomm)
```

Parameters:

`nprow` --- Number of process rows
`npcol` --- Number of process columns
`tcomp` --- Array containing the total number of arithmetic operations
`tcomm` --- Array containing the total volume of communications between pairs of processes

Description: The matrices ***A***, ***B*** and ***C*** containing the data are not passed as arguments. It has four parameters in addition to those passed to the PDGEMM function call. The parameters `(nprow, npcol)` contain the process arrangement, where `nprow` specifies the number of process rows and `npcol` specifies the number of process columns. The return parameter `tcomp` is a 1D array of size `nprow×npcol` logically representing a 2D array of size `[nprow][npcol]`. Its `[i][j]`–th element contains the total number of arithmetical operations performed by the process with coordinates `(i, j)` during the execution of the PDGEMM function call. The return parameter `tcomm` is a 1D array of size `nprow×npcol×nprow×npcol` logically representing an array of size `[nprow][npcol][nprow][npcol]`. Its `[i][j][k][l]`–th element contains the total number of bytes communicated between a pair of processes with coordinates `(i, j)` and `(k, l)` respectively during the execution of the PDGEMM function call. **`HSCAL_SUCCESS`** indicating successful execution or otherwise an appropriate error code is the return value.

Usage:

```
int rc, *tcomm;
double *tcomp;

tcomp = (double*)calloc(nprow*npcol, sizeof(double));
tcomm = (int*)calloc(nprow*npcol*nprow*npcol, sizeof(int));

rc = hscal_pdgemm_info(
          transa, transb,
          m, n, k,
          alpha,
          ia, ja, desca,
          ib, jb, descb,
          beta,
          ic, jc, descc,
          nprow, npcol,
          tcomp, tcomm
);

if (rc != HSCAL_SUCCESS)
{
   /* Problems querying the information */
}

/* Print the computations and communications information */

free(tcomp);
free(tcomm);
```

Return values: **`HSCAL_SUCCESS`** on success.

`hscal_dgemm_info`

Determines the total number of computations (arithmetical operations) involved in the execution of the BLAS DGEMM routine

Synopsis:

```
int hscal_dgemm_info(
    char* transa, char* transb,
    int * m, int * n, int * k,
    double * alpha,
    int * lda, int * ldb,
    double * beta, int * ldc, double *tcomp)
```

Parameters:

`tcomp` --- The total number of arithmetic operations

Description: The matrices ***A***, ***B*** and ***C*** containing the data are not passed as arguments. It has a parameter in addition to those passed to the serial DGEMM function call. This is the return parameter **`tcomp`**, which contains the total number of arithmetical operations performed in the execution of the function call.

Usage:

```
int rc;
double tcomp;

rc = hscal_dgemm_info(
          transa, transb,
          m, n, k,
          alpha,
          lda, ldb,
          beta,
          ldc, &tcomp);

if (rc != HSCAL_SUCCESS)
{
   /* Problems querying the information */
}

/* Print the computations */
```

Return values: **`HSCAL_SUCCESS`** on success.

3.4 Heterogeneous ScaLAPACK Debug Functions

`hscal_set_debug`

Set the debugging diagnostics levels

Synopsis:

```
   HSCAL_LOG_NONE           /* No logging */
   HSCAL_LOG_VERBOSE        /* Most verbose logging */

int
hscal_set_debug
(
    int debug_level
)
```

Parameters:

`debug_level` --- one of the debug levels shown above

Description: Produces detailed diagnostics. Any process can call this function.

3.5 Heterogeneous ScaLAPACK and HeteroMPI

This section explains how to compose a Heterogeneous ScaLAPACK program using Heterogeneous ScaLAPACK functions, which are counterparts of the HeteroMPI functions. It also shows ways to use the `timeof` interfaces cleverly to write a Heterogeneous ScaLAPACK program.

Assuming the application programmer wants to provide the process grid arrangement and not use the Heterogeneous ScaLAPACK runtime system to find it, Figure 4 shows the essential steps. Here the Heterogeneous ScaLAPACK program employs the ScaLAPACK PDGESV routine. The input process grid arrangement is `(nprow, npcol)`. The steps are:

1. Initialize the Heterogeneous ScaLAPACK runtime using using the operation

   ```
   int hscal_init(int * argc, int *** argv)
   ```

   where `argc` and `argv` are the same as the arguments passed to `main`. This routine must be called before any other Heterogeneous ScaLAPACK context management routine and must be called once. It must be called by all the processes running in the Heterogeneous ScaLAPACK application;
2. Updating the estimation of the speeds of the processors using the routine `hscal_pdgesv_recon`, which calls the HeteroMPI function `HMPI_Recon`. A benchmark code representing the core computations involved in the execution of the ScaLAPACK routine PDGESV is provided to this function call to accurately estimate the speeds of the processors. In this case, the benchmark code performs a local GEMM update of

```
   int main(int argc, char **argv) {
      int    nprow, npcol, pdgesvctxt, myrow, mycol, c__0 = 0, ictxt;
/* Problem parameters */
      int    *N, *NRHS, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *INFO;
      double *A, *B, *IPIV;
/* HeteroMPI handle to the group of MPI processes */
      HMPI_Group  gid;
      MPI_Comm    pdgesvcomm;
/* Initialize the heterogeneous ScaLAPACK runtime */
      hscal_init(&argc, &argv);
/* Initialize the array descriptors for the matrices A and B
   No need to specify the context argument */
      descinit_(DESCA, …, NULL);  /* for Matrix A */
      descinit_(DESCB, …, NULL);  /* for Matrix B */
/* Refresh the speeds of the processors */
      hscal_pdgesv_recon(N, NRHS, IA, JA, DESCA,
                         IB, JB, DESCB);
/* Create a HeteroMPI group of processes */
      hscal_pdgesv_group_create(N, NRHS, IA, JA, DESCA,
           IB, JB, DESCB, &nprow, &npcol, &gid, hscal_model_pdgesv);
/* All the processes that are not members of the group exit here */
      if (!HMPI_Is_member(&gid)) {
         HMPI_Group_free(&gid);
         hscal_finalize(c__0);
         return 0;
      }
/* Get the MPI communicator */
      pdgesvcomm = *(MPI_Comm*)HMPI_Get_comm(&gid);
/* Translate the MPI communicator to a BLACS handle */
      ictxt = Csys2blacs_handle(pdgesvcomm);
/* Form BLACS context based on pdgesvcomm */
      Cblacs_gridinit(&ictxt, "r", nprow, npcol);
/* Retrieve the process grid information */
      Cblacs_gridinfo(pdgesvctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A and B */
      descinit_(DESCA, …, &pdgesvctxt);  /* for Matrix A */
      descinit_(DESCB, …, &pdgesvctxt);  /* for Matrix B */
/* Distribute matrices on the process grid using user-defined pdmatgen */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix A */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix B */
/* Call the PBLAS 'pdgesv' routine */
      pdgesv_(N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO);
/* Free the BLACS context */
      Cblacs_gridexit(ictxt);
/* Free the HeteroMPI group */
      HMPI_Group_free(&gid);
/* Finalize the Heterogeneous ScaLAPACK runtime */
      hscal_finalize(c__0);
      return 0;
   }
```

Figure 4. Basic steps of the Heterogeneous ScaLAPACK program calling the ScaLAPACK routine **`PDGESV`**. The HeteroMPI functions are used.

m×b and b×n matrices where b is the data distribution blocking factor and m and n are local number of matrix rows and columns determined based on the problem size solved;

3. Creation of a HeteroMPI group of MPI processes using the routine `hscal_pdgesv_group_create`, which calls the HeteroMPI's group constructor routine `HMPI_Group_create`. One of the inputs to this function call is the handle `hscal_model_pdgesv`, which encapsulates all the features of the performance model in the form of a set of functions generated by the compiler from the description of the performance model of the ScaLAPACK routine. The other input is the process grid arrangement, `(nprow, npcol)`. During this function call, the HeteroMPI runtime system solves the problem of selection of the optimal set of processes running on different computers of the heterogeneous network (mapping problem). The solution is based on the performance model of the ScaLAPACK routine and the performance model of the executing network of computers, which reflects the state of this network just before the execution of the ScaLAPACK routine;
4. The handle to the HeteroMPI group is passed as input to the HeteroMPI routine `HMPI_Get_comm` to obtain the MPI communicator. This MPI communicator is translated to a BLACS handle using the BLACS routine `Csys2blacs_handle`;
5. The BLACS handle is then passed to the BLACS routine `Cblacs_gridinit`, which creates the BLACS context;
6. Execute the steps (2) and (3) involved in calling the ScaLAPACK PDGESV routine (shown in Figure 1);
7. Release the process grid via a call to `Cblacs_gridexit`;
8. When all the computations have been completed, the program is exited with a call to **`hscal_finalize`**, which finalizes the heterogeneous ScaLAPACK runtime.

Now assume that application programmer has a ScaLAPACK application, which employs two routines PDGESV and PDPOSV. Let us also assume that the programmer has to choose between using one of the process arrangements (3,3) and (4,4). Figure 5 shows how the `timeof` interface can be used cleverly to determine the best process arrangement.

```
   int main(int argc, char **argv) {
/* Problem parameters */
      char    *UPLO;
      int    *N, *NRHS, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *INFO;
      double *A, *B, *IPIV, pgatime1[2], pgatime2[2];
      int    pga1[2] = {3, 3}, pga2[2] = {4, 4};
/* Initialize the heterogeneous ScaLAPACK runtime */
      hscal_init(&argc, &argv);
…
/* Refresh the speeds of the processors */
      hscal_pdgesv_recon(N, NRHS, IA, JA, DESCA,
                         IB, JB, DESCB);
/* Use timeof to estimate the execution time of PDGESV for the
   process arrangement (3,3) */
      pgatime1[0] = hscal_pdgesv_timeof(N, NRHS, IA, JA, DESCA,
           IB, JB, DESCB, &pga1[0], &pga1[1], hscal_model_pdgesv);
/* Refresh the speeds of the processors */
      hscal_pdposv_recon(UPLO, N, NRHS, IA, JA, DESCA,
                         IB, JB, DESCB);
/* Use timeof to estimate the execution time of PDPOSV for the
   process arrangement (3,3) */
      pgatime1[1] = hscal_pdposv_timeof(UPLO, N, NRHS, IA, JA, DESCA,
           IB, JB, DESCB, &pga1[0], &pga1[1], hscal_model_pdposv);
/* Refresh the speeds of the processors */
      hscal_pdgesv_recon(N, NRHS, IA, JA, DESCA,
                         IB, JB, DESCB);
/* Use timeof to estimate the execution time of PDGESV for the
   process arrangement (4,4) */
      pgatime2[0] = hscal_pdgesv_timeof(N, NRHS, IA, JA, DESCA,
           IB, JB, DESCB, &pga2[0], &pga2[1], hscal_model_pdgesv);
/* Refresh the speeds of the processors */
      hscal_pdposv_recon(UPLO, N, NRHS, IA, JA, DESCA,
                         IB, JB, DESCB);
/* Use timeof to estimate the execution time of PDPOSV for the
   process arrangement (4,4) */
      pgatime2[1] = hscal_pdposv_timeof(UPLO, N, NRHS, IA, JA, DESCA,
           IB, JB, DESCB, &pga2[0], &pga2[1], hscal_model_pdposv);
/* Use the times obtained to find the best process arrangement */
   if ((pgatime1[0]+pgatime1[1]) < (pgatime2[0]+pgatime2[1]))
      /* Use the process grid arrangement (3,3) */
   else
      /* Use the process grid arrangement (4,4) */
…

/* Finalize the Heterogeneous ScaLAPACK runtime */
      hscal_finalize(c__0);
   }
```

Figure 5. Use of the `timeof` interfaces to choose the best process arrangement between a pair of process arrangements.

Contrast this to the application shown in Figure 3 where the Heterogeneous ScaLAPACK runtime finds the best process grid arrangement.

```
   #include <hscalapack.h>
   #include "hscalapack_pdgesv_parameters.h"

   int main(int argc, char **argv) {
      int nprow, npcol, pdgesvctxt, myrow, mycol, c__0 = 0;
/* Problem parameters */
      int  *N, *NRHS, *IA, *JA, *DESCA, *IB, *JB, *DESCB, *INFO;
      double *A, *B, *IPIV;
/* Initialize the heterogeneous ScaLAPACK runtime */
      hscal_init(&argc, &argv);
/* Initialize the array descriptors for the matrices A and B
   No need to specify the context argument */
      descinit_(DESCA, …, NULL);  /* for Matrix A */
      descinit_(DESCB, …, NULL);  /* for Matrix B */
/* Get the heterogeneous PDGESV context */
      hscal_pdgesv_ctxt(N, NRHS, IA, JA, DESCA,
                        IB, JB, DESCB, &pdgesvctxt);
      if (!hscal_in_ctxt(&pdgesvctxt)) {
         hscal_free_ctxt(&pdgesvctxt);
         hscal_finalize(c__0);
         return 0;
      }
/* Retrieve the process grid information */
      blacs_gridinfo__(&pdgesvctxt, &nprow, &npcol, &myrow, &mycol);
/* Initialize the array descriptors for the matrices A and B */
      descinit_(DESCA, …, &pdgesvctxt);  /* for Matrix A */
      descinit_(DESCB, …, &pdgesvctxt);  /* for Matrix B */
/* Distribute matrices on the process grid using user-defined pdmatgen */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix A */
      pdmatgen_(&pdgesvctxt, …); /* for Matrix B */
/* Call the PBLAS 'pdgesv' routine */
      pdgesv_(N, NRHS, A, IA, JA, DESCA, IPIV, B, IB, JB, DESCB, INFO);
/* Release the heterogeneous PDGESV context */
      hscal_free_ctxt(&pdgesvctxt);
/* Finalize the Heterogeneous ScaLAPACK runtime */
      hscal_finalize(c__0);
      return 0;
   }
```

Figure 6. Basic steps involved in the heterogeneous ScaLAPACK program calling the routine **PDGESV**. The file 'test_hscal_pdgesv.c'.

```
int N                = 1024;
int NRHS             = 1024;
int NB               = 32;
int IA               = 1;
int JA               = 1;
int IB               = 1;
int JB               = 1;
```

Figure 7. The parameters file 'hscalapack_pdgesv_parameters.h'.

4 Heterogeneous ScaLAPACK Program Execution

4.1Building Heterogeneous ScaLAPACK Application

To create the executable of the application shown in Figure 6,

```
shell$ mpicc -I<HSCALAPACKDIR>/include -o test_hscal_pdgesv
test_hscal_pdgesv.c -L<HSCALAPACKDIR>/lib -lhscalapack -lhmpi -
lmpc <ScaLAPACK library> <BLACS library> <BLAS library> <F2C
library>
```

4.2 Running Heterogeneous ScaLAPACK Application

Run the target program using tools provided by MPI such as *mpirun*.

During the Heterogeneous ScaLAPACK runtime initialization, the structure of the network (performances of the processors, values of the parameters of the communication model) is obtained. This information is used by the runtime mapping algorithms. This initialization process takes time. To avoid this penalty, there are two options that can be provided to the Heterogeneous ScaLAPACK program, which allow initialization to take place from a file.

To write the structure/state of the executing network to a file, use the option "*-wtopo*". For example, consider the execution of the program *test_hscal_pdgesv* using OpenMPI on a network consisting of three hosts {hcl01, hcl02, hcl03}. The structure of the network is saved/written to a file "topostruct.txt".

```
shell$ mpirun -np 1 --host hcl01 test_hscal_pdgesv -wtopo
topostruct.txt : -np 1 --host hcl02 test_hscal_pdgesv : -np 1 --
host hcl03 test_hscal_pdgesv
```

To read the structure/state of the executing network from a file, use the option "*-rtopo*". For example, consider the execution of the program *test_hscal_pdgesv* using OpenMPI on three hosts {hcl01, hcl02, hcl03}. The structure of the network is read from the file "topostruct.txt".

```
shell$ mpirun -np 1 --host hcl01 test_hscal_pdgesv -rtopo
topostruct.txt : -np 1 --host hcl02 test_hscal_pdgesv : -np 1 --
host hcl03 test_hscal_pdgesv
```

The topology files can be created once and used for multiple runs.

5 Heterogeneous ScaLAPACK Installation Guide for UNIX

This section provides information for programmers and/or system administrators who want to install Heterogeneous ScaLAPACK for UNIX.

5.1 System Requirements

The following table describes system requirements for Heterogeneous ScaLAPACK for UNIX.

Component	**Requirement**
Operating System	Tested on Linux, Solaris, FreeBSD
C compiler	Any ANSI C compiler
MPI	LAM MPI 6.3.2 or higher MPICH MPI 1.2.0 or higher with chp4 device OpenMPI

5.2 Contents of Heterogeneous ScaLAPACK Installation

Heterogeneous ScaLAPACK installation contains the following:

Directory	**Contents**
bin	Binaries **mpcc**
include	Header files
Lib	Heterogeneous ScaLAPACK library **libhscalapack.a** HeteroMPI libraries **libhmpi.a**, **libmpc.a**

5.3 Installation

You should have MPI installed on your system. Please make sure that `mpicc` and `mpirun` tools are in your `PATH` environment variable.

Unpack the Heterogeneous ScaLAPACK distribution, which comes as a tar in the form heteroscalapack-x.y.z.tar.gz. To uncompress the file tree, use:

```
shell$ tar -zxvf heteroscalapack-x.y.z.tar.gz
```

where x.y.z stands for the installed version of the Heterogeneous ScaLAPACK library (say 1.2.1, 2.0.0, or 3.1.1).

The directory 'heteroscalapack-x.y.z' will be created; execute

```
shell$ cd heteroscalapack-x.y.z
shell$ mkdir build
```

```
shell$ ../configure –prefix=<Installation directory>
shell$ make all install
```

5.4 Testing your Installation

After you have successfully installed Heterogeneous ScaLAPACK, to test the installation, you can test each individual test in the directory "**`heteroscalapack-x.y.z/tests`**". There is a test for each and every ScaLAPACK and PBLAS routine.

Diagnostics are produced showing success or failure of each individual test. The ScaLAPACK tests are in the directory **`tests/SCALAPACK`**.

The following variables must be set in the environment before testing:

1). **`LIBSCALAPACKDIR`**: The location of the scalapack library
`LIBSSCALAPACK`: The name of the scalapack library

The link command would be **`-L$(LIBSCALAPACKDIR) -l$(LIBSSCALAPACK)`**

2). **`LIBBLACSDIR`**: The location of the BLACS libraries
`LIBSBLACS`: The names of the BLACS libraries

The link command would be **`-L$(LIBBLACSDIR) -l$(LIBSBLACS)`**

3). **`LIBBLASDIR`**: The location of the BLAS (optimized or non-optimized) libraries
`LIBSBLAS`: The names of the BLAS libraries

The link command would be **`-L$(LIBBLASDIR) -l$(LIBSBLAS)`**

4). **`LIBF2CDIR`**: The location of the f2c library
`LIBSF2C`: The name of the f2c library

The link command would be **`-L$(LIBF2CDIR) -l$(LIBSF2C)`**

IMPORTANT NOTES: ***In the case of `LIBSSCALAPACK`, `LIBSBLACS`, `LIBSBLAS`, `LIBSF2C`, make sure there is a space in front of the value of the environment variable. For example:***

```
export LIBSSCALAPACK=" -lscalapack"
export LIBSBLACS=" -lblacs -lblacsCinit -lblacs"
export LIBSF2C=" -lf2c"
```

Notice a space in front of `-lblacs, -lscalapack, -lf2c`...

Edit the parameters file to specify the problem parameters. There are separate parameters files for a ScaLAPACK program and the corresponding Heterogeneous ScaLAPACK program. This is so that you can compare the execution times. Each and every test prints the execution time.

For example, to test the ScaLAPACK program employing the PBLAS routine PCGEMM, edit the parameters file `'scalapack_pcgemm_parameters.h'`
For example, to test the Heterogeneous ScaLAPACK program employing the routine PCGEMM, edit the parameters file `'hscalapack_pcgemm_parameters.h'`

Build and run the test using **`mpicc`** and **`mpirun`** tools.

6 References

[1] http://www-unix.mcs.anl.gov/mpi/.
[2] http://www.netlib.org/scalapack/scalapack_home.html.
[3] http://www.netlib.org/scalapack/pblas_qref.html.
[4] http://www.netlib.org/blas/.
[5] http://www.netlib.org/blacs/.
[6] http://hcl.ucd.ie/Software/HeteroMPI.
[7] L. Prylli and B. Tourancheau, "Efficient block cyclic data redistribution," in Proceedings of the Second International Euro-Par Conference on Parallel Processing (EUROPAR'96), Lecture Notes in Computer Science 1123, Springer-Verlag, pp. 155-164, 1996.
[8] R. Whaley, A. Petitet, and J. Dongarra, "Automated Empirical Optimizations of Software and the ATLAS Project," In Parallel Computing, Volume 27, No. (1–2), pp.3–35, January 2001, ISSN 0167-8191.