Intel® Cluster Poisson Solver Library

Reference Manual

October 2008

World Wide Web: http://developer.intel.com

Document number: 320867-001

Contents

Disclaimer and Legal Information

1 Intel® Cluster Poisson Solver Library Implemented

The Intel® Cluster Poisson Solver Library (Intel® CPSL) routines enable approximate solving certain two-dimensional Helmholtz, Poisson, and Laplace problems. The current implementation of Intel CPSL contains only an example of a 3D solver. If you feel that 3D functionality is important, please leave a message at Intel® WhatIf Forum.

Sections below provide details of the problems that can be solved using Intel CPSL.

Two-Dimensional Problems

Notational Conventions

The Intel CPSL interface description uses the following notation for boundaries of a rectangular domain $a_x < x < b_x$, $a_y < y < b_y$ on a Cartesian plane:

$bd\_a_x = \{x = a_x, a_y \le y \le b_y\}$, $bd\_b_x = \{x = b_x, a_y \le y \le b_y\}$

$bd\_a_y = \{a_x \le x \le b_x, y = a_y\}$, $bd\_b_y = \{a_x \le x \le b_x, y = b_y\}$.

The wildcard "+" may stand for any of the symbols a_x, b_x, a_y, b_y, so that bd_+ denotes any of the above boundaries.

Two-dimensional (2D) Helmholtz problem

The 2D Helmholtz problem is to find an approximate solution of the Helmholtz equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} + qu = f(x, y), \quad q = const \ge 0$$

in a rectangle, that is, a rectangular domain $a_x < x < b_x$, $a_y < y < b_y$, with one of the following boundary conditions on each boundary bd_+:

- The Dirichlet boundary condition
  $$u(x, y) = G(x, y)$$
- The Neumann boundary condition
  $$\frac{\partial u}{\partial n}(x, y) = g(x, y)$$

where

$n = -x$ on bd_a_x, $n = x$ on bd_b_x,

$n = -y$ on bd_a_y, $n = y$ on bd_b_y.

Two-dimensional (2D) Poisson problem

The Poisson problem is a special case of the Helmholtz problem, when q=0. The 2D Poisson problem is to find an approximate solution of the Poisson equation

$$-\frac{\partial^2 u}{\partial x^2} - \frac{\partial^2 u}{\partial y^2} = f(x, y)$$

in a rectangle $a_x < x < b_x$, $a_y < y < b_y$ with the Dirichlet or Neumann boundary condition on each boundary *bd_+*. In case of a problem with the Neumann boundary condition on the entire boundary, you can find the solution of the problem only up to a constant. In this case, the Intel CPSL will compute the solution that provides the minimal Euclidean norm of the solution.

Two-dimensional (2D) Laplace problem

The Laplace problem is a special case of the Helmholtz problem, when q=0 and $f(x, y)$=0. The 2D Laplace problem is to find an approximate solution of the Laplace equation

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0$$

in a rectangle $a_x < x < b_x$, $a_y < y < b_y$ with the Dirichlet or Neumann boundary condition on each boundary *bd_+*.

Approximation of 2D problems

To find an approximate solution for any of the 2D problems, a uniform mesh is built in the rectangular domain:

$$x_i = a_x + ih_x,\ y_j = a_y + jh_y$$
$$i = 0,..,n_x,\ j = 0,..,n_y,\ h_x = \frac{b_x - a_x}{n_x},\ h_y = \frac{b_y - a_y}{n_y}$$

Intel CPSL uses the standard five-point finite difference approximation on this mesh to compute the approximation to the solution:

The values of the approximate solution will be computed in the mesh points (x_i , y_j), provided that the user knows the values of the right-hand side $f(x, y)$ in these points and the values of the appropriate boundary functions $G(x, y)$ and/or $g(x,y)$ in the mesh points laying on the boundary of the rectangular domain.

2 Sequence of Invoking the Intel® Cluster Poisson Solver Library Routines

NOTE: This description always considers the solution process for the Helmholtz problem because the Fast Poisson Solver and Fast Laplace Solver are special cases of the Fast Helmholtz Solver (see Intel® Cluster Poisson Solver Library Implemented).

Computation of a solution of the Helmholtz problem using the Intel® Cluster Poisson Solver Library (Intel® CPSL) interface is conceptually divided into four steps, each of which is performed via a dedicated routine. Table 1 lists the routines and briefly describes their purpose.

Currently, all the Intel CPSL routines have versions operating only with double-precision data. The routines for the Cartesian coordinate system have only a 2D version.

Table 1 Intel CPSL Interface Routines

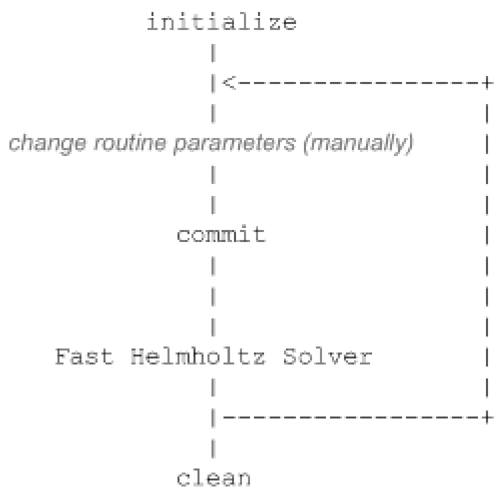
Routine	Description
`dmv0_init_helmholtz_2d`	Initializes basic data structures of Intel CPSL for the Fast Helmholtz Solver.
`dmv0_commit_helmholtz_2d`	Checks consistency and correctness of user's data, as well as creates and initializes data structures needed for the solver.
`dmv0_helmholtz_2d`	Computes an approximate solution of the 2D Helmholtz problem (see Intel® Cluster Poisson Solver Library Implemented) specified by the parameters.
`dmv0_free_helmholtz_2d`	Cleans the memory used by the data structures.

To find an approximate solution of the Helmholtz problem *only once*, the Intel CPSL interface routines are normally invoked in the order in which they are listed in Table 1.

NOTE: Though the order of invoking the Intel CPSL routines may be changed, it is highly recommended to follow the above order of routine calls.

The diagram in Figure 1 indicates a typical order in which the Intel CPSL routines can be invoked in a general case.

Figure 1 Typical Order of Invoking the Intel CPSL Routines



A general scheme of using the Intel CPSL 2D routines is shown below.

```
...
dmv0_init_helmholtz_2d(&ax, &bx, &ay, &by, &nx, &ny, BCtype, &q, ipar, dpar,
&stat);
/* change parameters in ipar and/or dpar, if necessary. */
/* note that the result of the Fast Helmholtz Solver will be in f! If you
want to keep the data stored in f,
save it before the function call below */
dmv0_commit_helmholtz_2d(f, bd_ax, bd_bx, bd_ay, bd_by, &xhandle, ipar,
dpar, cnts, &comm, &stat);
dmv0_helmholtz_2d(f, bd_ax, bd_bx, bd_ay, bd_by, &xhandle, ipar, dpar, work,
cnts, &comm, &stat);
dmv0_free_helmholtz_2d(&xhandle, ipar, &stat);
/* here you may clean the memory used by f, dpar, ipar */
...
```

You can find Fortran 90 and C code examples that use the Intel CPSL routines to solve the 2d Helmholtz and 3D Poisson problem in the `examples` subdirectory of the Intel CPSL installation directory.

NOTE: Although an example of the 3D Poisson solver is provided, there is currently no documentation for the 3D Poisson solver interface.

3 Interface Description

All types in this document are standard C types: `int` and `double`. Fortran 90 users can call the routines with `INTEGER*4` and `DOUBLE PRECISION` Fortran types, respectively (see 2D Helmholtz and 3D Poisson Solver examples in the `examples` subdirectory of the Intel CPSL installation directory).

Routine Options

All the Intel CPSL routines use parameters for passing various options to the routines. These parameters are arrays *`ipar`* and *`dpar`* (see Common Parameters). You can change these values during computations to meet your needs.

WARNING: To avoid failure or wrong results, you must provide correct and consistent parameters to the routines.

User Data Arrays

The Intel CPSL routines take arrays of user data as input. For example, user arrays are passed to the routine `dmv0_helmholtz_2d` to compute an approximate solution to the 2D Helmholtz problem. To minimize storage requirements and improve the overall run-time efficiency, the Intel CPSL routines do not make copies of user input arrays.

NOTE: If you need a copy of your input data arrays, save them yourself.

Distributing Data among Processes in the 2D case

The Intel CPSL 2D routines use an array `f` that initially contains the right-hand side of the problem. After invoking the Intel CPSL routines, it contains the solution to the problem. This array can be represented as a rectangle (1..nx+1,1..ny+1) containing the values of the corresponding function. It is your responsibility to distribute this array among MPI processes so that the MPI process with rank p (p=0,…, $nproc$-1) contains sub-array `f` (1..nx+1,n_p..n_{p+1}-1), where

$n_1 = 1$,

$n_{p+1} = n_p + [(nx+1)/nproc]+1$, if $p< \{(nx+1)/nproc\}$

$n_{p+1} = n_p + [(nx+1)/nproc]$,   if $p>= \{(nx+1)/nproc\}$.

Here $[u/v]$ is the integer part of u/v, $\{u/v\}$ is the remainder $u - [u/v]*v$, and $nproc$ is the number of MPI processes.

4 2D Intel® Cluster Poisson Solver Library Routines

The section describes the 2D Intel® Cluster Poisson Solver Library (Intel® CPSL) routines, their syntax, parameters and return values.

- `dmv0_init_helmholtz_2d`
  Initializes basic data structures of the Fast 2D Helmholtz Solver.
- `dmv0_commit_helmholtz_2d`
  Checks consistency and correctness of user's data, as well as initializes certain data structures required to solve the 2D Helmholtz problem.
- `dmv0_helmholtz_2d`
  Computes the solution of the 2D Helmholtz problem specified by the parameters.
- `dmv0_free_helmholtz_2d`
  Cleans the memory allocated for the data structures used by the Intel MKL FFT interface[1].

dmv0_init_helmholtz_2d

Initializes basic data structures of the Fast 2D Helmholtz Solver.

Syntax

```
void dmv0_init_helmholtz_2d( double* ax, double* bx, double* ay, double* by,
int* nx, int* ny, char* BCtype, double* q, int* ipar, double* dpar, int* stat);
```

Input Parameters

ax	`double*`. The coordinate of the leftmost boundary of the domain along x-axis.
bx	`double*`. The coordinate of the rightmost boundary of the domain along x-axis.
ay	`double*`. The coordinate of the leftmost boundary of the domain along y-axis.
by	`double*`. The coordinate of the rightmost boundary of the domain along y-axis.
nx	`int*`. The number of mesh intervals along x-axis.
ny	`int*`. The number of mesh intervals along y-axis.
BCtype	`char*`. Contains the type of boundary conditions on each boundary. Must contain four characters. Each character can be 'N' (the Neumann boundary condition) or 'D'

[1] Intel CPSL computations use the Intel® Math Kernel Library (Intel® MKL) FFT interface.

	(the Dirichlet boundary condition). The types of boundary conditions for the boundaries must be specified in the following order: *bd_a*$_x$, *bd_b*$_x$, *bd_a*$_y$, *bd_b*$_y$.
q	`double*`. The constant Helmholtz coefficient. To solve the Poisson or Laplace problem, set the value of *q* to 0.

Output Parameters

ipar	`int` array of size 128. Contains integer data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).
dpar	`double array` of size 5**nx*/2+7. Contains floating-point data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).
stat	`int*`. Routine completion status, which is also written to *ipar*[0]. The status should be 0 to proceed to other Intel CPSL routines.

Description

The routine `dmv0_init_helmholtz_2d` initializes basic data structures for Intel CPSL computations. All the routines invoked after a call to the `dmv0_init_helmholtz_2d` routine use values of the *ipar* and *dpar* array parameters returned by the routine. Detailed description of the array parameters can be found in Common Parameters.

WARNING: You can skip calling this routine in your code. However, see Caveat on Parameter Modifications before doing so.

Return Values

stat= 0	The routine successfully completed the task. In general, to proceed with computations, the routine should complete with this *stat* value.
stat= -99999	The routine failed to complete the task because of a fatal error.

dmv0_commit_helmholtz_2d

Checks consistency and correctness of user's data, as well as initializes certain data structures required to solve the 2D Helmholtz problem.

Syntax

```
void dmv0_commit_helmholtz_2d(double* f, double* bd_ax, double* bd_bx, double*
bd_ay, double* bd_by, DFTI_DESCRIPTOR_HANDLE* handle, int* ipar, double* dpar,
int* cnts, MPI_Comm comm, int* stat);
```

Input Parameters

f	`double*`. Contains the right-hand side of the problem packed in a single vector. The size of the vector is (*nx+1*)*(n_{p+1}-n_p) (see Distributing Data among Processes in the 2D case). The value of the right-hand side in the mesh point (*i*, *j*) is stored in *f*[*i*+*j** (*nx+1*)].

To solve the Laplace problem, set all the elements of the array *f* to 0.

Note that the array *f* may be altered by the routine. To preserve the vector, save it in another memory location.

ipar	`int*` array of size 128. Contains integer data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).
dpar	`double*` array of size 5**nx*/2+7. Contains floating-point data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).
bd_ax	`double*`. Contains values of the boundary condition on the leftmost boundary of the domain along *x*-axis. The size of the array is *ny*+1. In case of the Dirichlet boundary condition (the value of *BCtype*[0] is 'D'), contains values of the function $G(ax, y_j)$, j=0, ..., *ny*. In case of the Neumann boundary condition (the value of *BCtype*[0] is 'N'), it contains values of the function $g(ax, y_j)$, j=0, ..., *ny*. The value corresponding to the index *j* is placed in *bd_ax*[*j*].
bd_bx	`double*`. Contains values of the boundary condition on the rightmost boundary of the domain along *x*-axis. The size of the array is *ny*+1. In case of the Dirichlet boundary condition (the value of *BCtype*[1] is 'D'), it contains values of the function $G(bx, y_j)$, j=0, ..., *ny*. In case of the Neumann boundary condition (the value of *BCtype*[1] is 'N'), it contains values of the function $g(bx, y_j)$, j=0, ..., *ny*. The value corresponding to the index *j* is placed in *bd_bx*[*j*].
bd_ay	`double*`. Contains values of the boundary condition on the leftmost boundary of the domain along *y*-axis. The size of the array is $(n_{p+1}-n_p)$. In case of the Dirichlet boundary condition (the value of *BCtype*[2] is 'D'), it contains values of the function $G(x_i, ay)$, $i=n_p, ..., n_{p+1}-1$. In case of the Neumann boundary condition (the value of *BCtype*[2] is 'N'), it contains values of the function $g(x_i, ay)$, $i= n_p, ..., n_{p+1}-1$. The value corresponding to the index *i* is placed in *bd_ay*[*i*].
bd_by	`double*`. Contains values of the boundary condition on the leftmost boundary of the domain along *y*-axis. The size of the array is $(n_{p+1}-n_p)$. In case of the Dirichlet boundary condition (value of *BCtype*[2] is 'D'), it contains values of the function $G(x_i, by)$, $i=n_p, ..., n_{p+1}-1$. In case of the Neumann boundary condition (value of *BCtype*[2] is 'N'), it contains values of the function $g(x_i, by)$, $i= n_p, ..., n_{p+1}-1$. The value corresponding to the index *i* is placed in *bd_ay*[*i*].
cnts	`int*`. Contains internal values needed for communication between MPI processes. The size of the array is 4**nproc*, where *nproc* is the number of MPI processes.
comm	`MPI_Comm*`. MPI Communicator.

Output Parameters

f	`double*`. Vector of the right-hand side of the problem. Possibly, altered on output.
ipar	`int*`. Contains integer data to be used by the Fast Helmholtz Solver. Modified on output as explained in Common Parameters.
dpar	`double*`. Contains floating-point data to be used by the Fast Helmholtz Solver. Modified on output as explained in Common Parameters.
xhandle	`DFTI_DESCRIPTOR_HANDLE*`. Internal data structure.
stat	`int*`. Routine completion status, which is also written to *ipar*[0]. The status should be 0 to proceed to other Intel CPSL routines.

Description

The routine `dmv0_commit_helmholtz_2d` checks consistency and correctness of the parameters to be passed to the solver routine `dmv0_ helmholtz_2d`. It also initializes data structure *xhandle,* as well as arrays *ipar* and *dpar*. Refer to Common Parameters to find out which particular array elements the `dmv0_commit_helmholtz_2d` routine initializes and what values are written there.

The routine performs only a basic check for correctness and consistency. If you are going to modify parameters of Intel CPSL routines, see Caveat on Parameter Modifications. Unlike `dmv0_init_helmholtz_2d`, the routine `dmv0_commit_helmholtz_2d` is mandatory, and you cannot skip calling it in your code. Values of *ax*, *bx*, *ay*, *by*, *nx*, *ny*, and *BCtype* are passed to the routine with the *ipar* array and defined in a previous call to the `dmv0_init_helmholtz_2d` routine.

Return Values

stat= 1	The routine completed without errors and produced some warnings.
stat= 0	The routine successfully completed the task.
stat= -100	The routine stopped because an error in the user's data was found or the data in the *dpar* or *ipar* array was altered by mistake.
stat= -1000	The routine stopped because of an internal fatal interface error.
stat= -10000	The routine stopped because the initialization failed to complete or the parameter *ipar*[0] was altered by mistake.
stat= -99999	The routine failed to complete the task because of a fatal error.

dmv0_helmholtz_2d

Computes the solution of the 2D Helmholtz problem specified by the parameters.

Syntax

```
void dmv0_ helmholtz_2d(double* f, double* bd_ax, double* bd_bx, double*
bd_ay, double* bd_by, DFTI_DESCRIPTOR_HANDLE* handle, int* ipar, double* dpar,
double* work, int* cnts, MPI_Comm comm, int* stat);
```

Input Parameters

f	`double*`. Contains the right-hand side of the problem packed in a single vector. The size of the vector is (*nx+1*)*(n_{p+1}-n_p) (see Distributing Data among Processes in the 2D case). The value of the right-hand side in the mesh point (*i*, *j*) is stored in *f*[*i*+*j** (*nx+1*)] . To solve the Laplace problem, set all the elements of the array *f* to 0. Note that the array *f* may be altered by the routine. To preserve the vector, save it in another memory location.
ipar	`int*` array of size 128. Contains integer data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).
dpar	`double*` array of size 5**nx*/2+7. Contains floating-point data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).
bd_ax	`double*`. Contains values of the boundary condition on the leftmost boundary of the domain along *x*-axis. The size of the array is *ny*+1. In case of the Dirichlet boundary condition (the value of *BCtype*[0] is 'D'), it contains values of the function *G*(*ax*, y_j), *j*=0, ..., *ny*. In case of the Neumann boundary condition (value of *BCtype*[0] is 'N'), it contains values of the function *g*(*ax*, y_j), *j*=0, ..., *ny*. The value corresponding to the index *j* is placed in *bd_ax*[*j*].
bd_bx	`double*`. Contains values of the boundary condition on the rightmost boundary of the domain along *x*-axis. The size of the array is *ny*+1. In case of the Dirichlet boundary condition (the value of *BCtype*[1] is 'D'), it contains values of the function *G*(*bx*, y_j), *j*=0, ..., *ny*. In case of the Neumann boundary condition (the value of *BCtype*[1] is 'N'), it contains values of the function *g*(*bx*, y_j), *j*=0, ..., *ny*. The value corresponding to the index *j* is placed in *bd_bx*[*j*].
bd_ay	`double*`. Contains values of the boundary condition on the leftmost boundary of the domain along *y*-axis. The size of the array is (n_{p+1}-n_p). In case of the Dirichlet boundary condition (the value of *BCtype*[2] is 'D'), it contains values of the function *G*(x_i, *ay*), *i*=n_p, ..., n_{p+1}-1. In case of the Neumann boundary condition (the value of *BCtype*[2] is 'N'), it contains values of the function *g*(x_i, *ay*), *i*= n_p, ..., n_{p+1}-1. The value corresponding to the index *i* is placed in *bd_ay*[*i*].
bd_by	`double*`. Contains values of the boundary condition on the leftmost boundary of the domain along *y*-axis. The size of the array is (n_{p+1}-n_p). In case of the Dirichlet boundary condition (value of *BCtype*[2] is 'D'), it contains values of the function *G*(x_i, b*y*), *i*=n_p, ..., n_{p+1}-1. In case of the Neumann boundary condition (the value of *BCtype*[2] is 'N'), it

	contains values of the function $g(x_i, by)$, $i = n_p, \ldots, n_{p+1}-1$. The value corresponding to the index *i* is placed in *bd_ay*[*i*].
work	`double*`. The internal work array. The size of this array is max((*nx+1*)*([(ny+1)/*nproc*]+1), ([(*nx*+1)/*nproc*]+1)*(*ny+1*)), where *nproc* is the number of MPI processes.
cnts	`int*`. Contains internal values that are needed for communication between MPI processes. The size of this array is 4**nproc*.
comm	`MPI_Comm*`. MPI Communicator.

NOTE: To avoid wrong computation results, do not change arrays *bd_ax*, *bd_bx*, *bd_ay*, *bd_by* between a call to the `dmv0_commit_helmholtz_2d` routine and a subsequent call to the `dmv0_helmholtz_2d routine`.

Output Parameters

f	`double*`. On output, contains the approximate solution to the problem packed the same way as the right-hand side of the problem was packed on input.
xhandle	`DFTI_DESCRIPTOR_HANDLE*`. Internal data structure.
ipar	`int*`. Contains integer data to be used by the Fast Helmholtz Solver. Modified on output as explained in Common Parameters.
dpar	`double*`. Contains floating-point data to be used by the Fast Helmholtz Solver. Modified on output as explained in Common Parameters.
stat	`int*`. Routine completion status, which is also written to *ipar*[0]. The status should be 0 to proceed to other Intel CPSL routines.

Description

The routine computes the approximate solution of the Helmholtz problem defined in previous calls to the initialization and commit routines. The solution is computed according to formulas given in Intel Cluster Poisson Solver Library Implemented. The *f* parameter, which initially holds the packed vector of the right-hand side of the problem, is replaced by the computed solution packed the same way. Values of *ax*, *bx*, *ay*, *by*, *nx*, *ny*, and *BCtype* are passed to the routine with the *ipar* array and defined in a previous call to the `dmv0_init_helmholtz_2d` routine.

Return Values

stat= 1	The routine completed without errors and produced some warnings.

stat= 0	The routine successfully completed the task.
stat= -2	The routine stopped because division by zero occurred. It usually happens if the data in the *dpar* array was altered by mistake.
stat= -3	The routine stopped because the memory was insufficient to complete the computations.
stat= -100	The routine stopped because an error in the user's data was found or the data in the *dpar* or *ipar* array was altered by mistake.
stat= -1000	The routine stopped because of an internal fatal interface error.
stat= -10000	The routine stopped because the initialization failed to complete or the parameter *ipar*[0] was altered by mistake.
stat= -99999	The routine failed to complete the task because of a fatal error.

dmv0_free_helmholtz_2d

Cleans the memory allocated for the internal data structures.

Syntax

```
dmv0_free_Helmholtz_2d(DFTI_DESCIPTOR_HANDLE* xhandle, int* ipar, int* stat);
```

Input Parameters

ipar	`int*` array of size 128. Contains integer data to be used by the Fast Helmholtz Solver (for details, refer to Common Parameters).

Output Parameters

xhandle	`DFTI_DESCIPTOR_HANDLE*`. Internal data structure. Memory allocated for the structures is released on output.
ipar	`int*`. Contains integer data to be used by the Fast Helmholtz Solver. Status of the routine call is written to *ipar*[0].
stat	`int*`. Routine completion status, which is also written to *ipar*[0].

Description

The routine cleans the memory used by the *xhandle* internal data structure. To release memory allocated for other parameters, include cleaning of the memory in your code.

Return Values

stat= 0	The routine successfully completed the task.
stat= -1000	The routine stopped because of an internal fatal interface error.
stat= -99999	The routine failed to complete the task because of a fatal error.

Common Parameters

This section provides description of array parameters *ipar* and *dpar,* which hold the Intel CPSL routine options.

ipar	`int` array of size 128, holds integer data needed for the Fast Helmholtz Solver. Its elements are described in Table 2:

Table 2 Elements of the ipar array

Index	Description
0	Contains status value of the last called Intel CPSL routine. In general, it should be 0 to proceed with Fast Helmholtz Solver. The element has no predefined values.
1	Contains error messaging options: • *ipar*[1]=-1 indicates that all error messages will be printed to the file `MKL_Poisson_Library_log.txt` in the folder from which the routine is called. If the file does not exist, the routine tries to create it. If the attempt fails, the routine prints information that the file cannot be created to the standard output device. • *ipar*[1]=0 indicates that no error messages will be printed. • *ipar*[1]=1, default. Indicates that all error messages will be printed to the preconnected default output device (typically, screen). In case of errors, the *stat* parameter will acquire a non-zero value regardless of the *ipar*[1] setting.
2	Contains warning messaging options: • *ipar*[2]=-1 indicates that all warning messages will be printed to the file `MKL_Poisson_Library_log.txt` in the directory from which the routine is called. If the file does not exist, the routine tries to create it. If the attempt fails, the routine prints information that the file cannot be created to the standard output device. • *ipar*[2]=0 indicates that no warning messages will be printed. • *ipar*[2]=1, default. Indicates that all warning messages will be printed to the preconnected default output device (typically, screen). In case of warnings, the *stat* parameter will acquire a non-zero value regardless of the *ipar*[2] setting.
3	Contains the number of the combination of boundary conditions. In the Cartesian case, it corresponds to the value that the *BCtype* parameter holds. In the 2D case, • 0 corresponds to 'DDDD' • 1 corresponds to 'DDDN' • … • 15 corresponds to 'NNNN'
4	Takes the value of 1 if *BCtype*[0]='N', 0 if *BCtype*[0]='D', and -1 otherwise.

Index	**Description**
5	Takes the value of 1 if *BCtype*[1]='N', 0 if *BCtype*[1]='D', and -1 otherwise.
6	Takes the value of 1 if *BCtype*[2]='N', 0 if *BCtype*[2]='D', and -1 otherwise.
7	Takes the value of 1 if *BCtype*[3]='N', 0 if *BCtype*[3]='D', and -1 otherwise.
8	Reserved.
9	Reserved.
10	Takes the value of *nx,* that is, the number of intervals along *x*-axis.
11	Takes the value of *ny*, that is, the number of intervals along *y*-axis.
12	Reserved.
13	Takes the value of 6, which specifies the internal partitioning of the *dpar* array.
14	Takes the value of *ipar*[13]+*ipar*[10]+1, which specifies the internal partitioning of the *dpar* array.
15	Reserved.
16	Reserved.
17	Takes the value of *ipar*[14]+1, which specifies the internal partitioning of the array.
18	Takes the value of *ipar*[17]+(3**ipar*[10])/2, which specifies the internal partitioning of the *dpar* array.
19	Unused.
20	Unused.
21	Contains message style options: • *ipar*[21]=0 indicates that Intel CPSL routines print all error and warning messages in Fortran-style notations. • *ipar*[21]=1 (default) indicates that Intel CPSL routines print the messages in C-style notations.
22	Contains the number of threads to be used for computations in a multithreaded environment. The default value is 1.
23 through 39	Unused.
40 through 59	Contain internal information.
60 through 79	Contain internal information.

NOTE: You may declare the *ipar* array in your code as `int ipar[80]`. However, for compatibility with later versions of Intel CPSL, which may require more *ipar* values, it is highly recommended to declare *ipar* as `int ipar[128]`.

dpar	array of size 13**nx*/2+7 in the 2D case; initialized in the `dmv0_init_Helmholtz_2d` and `dmv0_commit_Helmholtz_2d` routines. Holds data needed for double-precision Fast Helmholtz Solver computations. Elements of *dpar* array are described in Table 3.

Table 3 Elements of the dpar Array

Index	Description
0	Contains the length of the interval along *x*-axis right after a call to the `dmv0_init_Helmholtz_2d` routine or the mesh size *hx* in the *x* direction (for details, see Cluster Poisson Solver Library Implemented).
1	Contains the length of the interval along *y*-axis right after a call to the `dmv0_init_Helmholtz_2d` routine or the mesh size *hy* in the *y* direction (for details, see Cluster Poisson Solver Library Implemented).
2	Reserved.
3	Contains the value of the coefficient *q* after a call to the `dmv0_init_Helmholtz_2d` `routine`.
4	Contains the tolerance parameter after a call to the `dmv0_init_Helmholtz_2d` routine. This value is used only for the pure Neumann boundary conditions (*BCtype*="NNNN"). This is a special case, because the right-hand side of the problem cannot be arbitrary if the coefficient *q* is zero. Intel CPSL verifies that the classical solution exists (up to rounding errors) using this tolerance. In any case, Intel CPSL computes the normal solution, that is, the solution that has the minimal Euclidean norm. Nevertheless, the `dmv0_Helmholtz_2d` routine informs the user that the solution may not exist in a classical sense (up to rounding errors). The default value for this parameter is 1.0E-10. You can increase the value of the tolerance, for instance, to avoid the warnings that may appear.
ipar[13]-1 through *ipar*[14]-1	Contain the spectrum of the 1D problem along *x*-axis after a call to the `dmv0_commit_Helmholtz_2d` routine.
ipar[17]-1 through *ipar*[18]-1	Take the values of the (staggered) sine/cosine in the mesh points along *x*-axis after a call to the `dmv0_commit_Helmholtz_2d` routine.

NOTE: You may define the array size depending upon the type of the problem to solve.

Caveat on Parameter Modifications

Flexibility of the Intel CPSL interface enables you to skip calling the `dmv0_init_Helmholtz_2d` routine and to initialize the basic data structures explicitly in your code. You may also need to modify contents of the *ipar* and *dpar* arrays after initialization. When doing so, provide correct and consistent data in the arrays. Mistakenly altered arrays cause errors or wrong computation.

You can perform a basic check for correctness and consistency of parameters by calling the `dmv0_commit_Helmholtz_2d` routine; however, this does not ensure the correct solution but only reduces the chance of errors or wrong results.

NOTE: To supply correct and consistent parameters to Intel CPSL routines, you should have considerable experience in using the Intel CPSL interface, as well as good understanding of the solution process, elements that the *ipar* and *dpar* arrays contain, and dependencies between values of these elements.

However, in rare occurrences, even advanced users might fail in tuning parameters for the Fast Helmholtz Solver. In cases like these, refer for technical support at http://www.intel.com/software/products/support/.

WARNING: The only way that ensures a proper solution of the Helmholtz problem is to follow a typical sequence of invoking the routines and not change the default set of parameters. So, avoid modifications of *ipar* and *dpar* arrays unless a strong need arises.

5 Implementation Details

Several aspects of the Intel® Cluster Poisson Solver Library (Intel® CPSL) interface are platform-specific and language-specific. To promote portability across platforms and ease of use across different languages, the Intel CPSL language-specific header files are provided to include in the user's code. Currently, the following header files are available:

- `icpsl.h`, to be used together with `mkl_dfti.h` (from the Intel® Math Kernel Library (Intel® MKL)), for C programs.
- `icpsl.f90`, to be used together with `mkl_dfti.f90` (from Intel MKL), for Fortran 90 programs.

The include files define function prototypes for appropriate languages.

NOTE: Use of the Intel CPSL software without including one of the above header files is not supported.

C-specific Header File

The C-specific header file defines the following function prototypes for the solver:

```
void dmv0_init_helmholtz_2d(double*, double*, double*, double*, int*, int*,
char*, double*, int*, double*, int*);

void dmv0_commit_helmholtz_2d(double*, double*, double*, double*, double*,
DFTI_DESCRIPTOR_HANDLE*, int*, double*, int*, MPI_Comm*, int*);

void dmv0_helmholtz_2d(double*, double*, double*, double*, double*,
DFTI_DESCRIPTOR_HANDLE*, int*, double*, double*, int*, MPI_Comm*,int*);

void dmv0_free_helmholtz_2d(DFTI_DESCRIPTOR_HANDLE*, int*, int*);
```