Dalsoft's Parallel Library

Reference Manual

version 2.6
for the Linux/Windows operating systems

www.products.dalsoft.com/dpl.html

General

dpl - Dalsoft's Parallel Library - allows parallel execution of the set of functions. It is designed to be used on multi core processors with the operating environment that supports OpenMP. Current release is provided for the x86 multi core processors running Linux or Windows operating system; the version of the product for Windows OS is provided as "experimental".

In order to allow parallel execution of the implemented functions, all of the underlying algorithms were subject to a significant mathematical transformations.

dpl assumes that input data to the implemented functions is of the appropriate format and has correct values - e.g. no NULL pointers or NaN's are specified.

Technical specifications

Current version:	2.6
Source code in:	C
Operating system supported:	Linux, Windows
Documentation:	manual
Support:	on-line

See this for technical discussion regarding this product, including:

- analysis of the performance of the implemented functions/algorithms
- analysis of the accuracy of results generated by the implemented functions/algorithms
- the availability of the product and port to different systems

Installation

The release package includes:

dpl.h – include file for Linux/Windows

linux_libdpl.a – library for Linux

windows_libdpl.lib – library for Windows.

To install ***dpl***, copy the provided library and the include file (***dpl.h***) into the appropriate working directories: on Linux it may be */usr/local/include* for ***dpl.h*** and */usr/local/lib64* for ***libdpl.a*** renaming the library as ***libdpl.a***; e.g. on Linux do:

sudo cp linux_libdpl.a /usr/local/lib64/libdpl.a

Compiling and linking with the dpl library

This section describes how to use the ***dpl*** with your programs.

To use ***dpl*** in C/C++ programs, you must **#include** the file **dpl.h** in every source file that calls ***dpl*** functions, accesses ***dpl*** global variables, or uses #defined constants provided by ***dpl***.

The procedures for linking ***dpl*** with the rest of your program vary according to the compiler and method you are using. For example, on a typical Linux system this can be done as following

gcc myprogram.c -ldpl

or

g++ myprogram.cpp -ldpl

It was observed that on Linux, in certain cases, it is necessary to specify **-no-pie** option while linking object files with the ***dpl*** library.

Error Codes

All ***dpl*** functions return an integer value denoting the status of the performed operation. The status is zero if no error is detected, or a specific non-zero error code otherwise.

The possible error codes defined in **dpl.h** file and are:

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_BAD_STRIDE - problematic **stride** parameter specified
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected
DPL_ZERODIV - division by zero attempted
DPL_FAILED_REACH_TOLERANCE - failed to reach specified tolerance
DPL_NO_IMPRVMNT - Gauss-Seidel iteration is not improving the solution
DPL_INTERNAL_ERROR - internal error

In this document for every ***dpl*** function we list the possible error codes it may generate. The global

variable **dpl_error** contains an error code of the last parallel ***dpl*** function executed.

Programming with dpl

This section describes how to make ***dpl*** calls in your program.

Global Variables

dpl_error

int dpl_error;

dpl_error contains the error code from the last executed parallel ***dpl*** function. If the call succeeded, **dpl_error** will be **0** (**DPL_ERR_OK**). A list of error code values, their meanings, and defined constants for them can be found in the section **Error Codes**.

dpl_version

unsigned short dpl_version;

dpl_version contains the ***dpl***'s version in packed BCD format. For example, for the current ***dpl***'s version 2.6, the value of **dpl_version** is 0x0206. The high byte of **dpl_version** represents two digits to the left of the decimal point (one digit per nibble) and the low byte represents two digits to the right of the decimal point (again, one digit per nibble).

Using dpl in C/C++

All functions, provided by ***dpl*** have name that begins with '*dpl_*' and followed by the low-case mnemonic, e.g '*dpl_solve_gs*'.

For every numeric function, the serial implementation of the underlying algorithm is provided. This function has an identical interface as the corresponding parallel version and it name is formed by adding '*_serial*' to the name of the corresponding parallel function. .e.g '*dpl_solve_gs_serial*' for '*dpl_solve_gs*'. Note that these serial functions don't affect the global variable **dpl_error**.

dpl provides the following sets of functions.

- Routines for parallel implementation of basic stencils

  dpl_stencil_1p **dpl_stencil_o_1p**
  dpl_stencil_div_1p **dpl_stencil_div_o_1p**
  dpl_stencil_acc **dpl_stencil_div_acc**
  dpl_stencil_2p
  dpl_stencil

- Routines for parallel implementation of conditional/stochastic functions
  dpl_stencil_min **dpl_stencil_max**

dpl_max_st **dpl_stochastic_max**

- Routine for parallel implementation of the Gauss-Seidel method to solve systems of linear equations

  dpl_solve_gs

- Routine for parallel implementation of the general 2-D 5-points stencil

  dpl_p1p0_p0p1_p0p0_p0n1_n1p0

- Auxiliary routines

  dpl_aux_strerror

Stencils

dpl_stencil_1p

int dpl_stencil_1p(unsigned long n, const double *a, const double *b, double *x, unsigned int stride)

DESCRIPTION

computes **x[i] = a[i]*x[i-stride] + b[i]** for **i** = **stride**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

a, **b**

*Input vectors of size **n**; only elements from **stride** to **n-1** are used.*

x

*Input/output vector of size **n**: elements from **0** to **stride-1** used as an input; elements from **stride** to **n-1** are set by this routine (used as an output).*

stride

*Positive integer parameter. Must be less than **n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_BAD_STRIDE - problematic **stride** parameter specified
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

dpl_stencil_o_1p

int dpl_o_stencil_1p(unsigned long n, const double *c, const double *a, const double *b, double *x, unsigned int stride)

DESCRIPTION

computes **x[i] = c[i]*x[i] + a[i]*x[i-stride] + b[i]** for **i** = **stride**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

c, a, b

*Input vectors of size **n**; only elements from **stride** to **n-1** are used.*

x

*Input/output vector of size **n**: elements from **0** to **n-1** used as an input; elements from **stride** to **n-1** are set by this routine (used as an output).*

stride

*Positive integer parameter. Must be less than **n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_BAD_STRIDE - problematic **stride** parameter specified
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

dpl_stencil_div_1p

int dpl_stencil_div_1p(unsigned long n, const double *a, const double *b, double *x, unsigned int stride)

DESCRIPTION

computes **x[i] = a[i]/x[i-stride] + b[i]** for **i** = **stride**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

a, b

*Input vectors of size **n**; only elements from **stride** to **n-1** are used.*

x

*Input/output vector of size **n**: elements from **0** to **stride-1** used as an input; elements from **stride** to **n-1** are set by this routine (used as an output).*

stride

*Positive integer parameter. Must be less than **n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_BAD_STRIDE - problematic **stride** parameter specified
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected
DPL_ZERODIV - division by zero attempted

dpl_error is set to the returned value.

dpl_stencil_div_o_1p

int dpl_o_stencil_div_o_1p(unsigned long n, const double *c, const double *a, const double *b, double *x, unsigned int stride)

DESCRIPTION

computes **x[i] = c[i]*x[i] + a[i]/x[i-stride] + b[i]** for **i** = **stride**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

c, a, b

*Input vectors of size **n**; only elements from **stride** to **n-1** are used.*

x

*Input/output vector of size **n**: elements from **0** to **n-1** used as an input; elements from **stride** to **n-1** are set by this routine (used as an output).*

stride

*Positive integer parameter. Must be less than **n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_BAD_STRIDE - problematic **stride** parameter specified
DPL_FAILED_ALLOC_MEMORY - failed allocate memory

DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected
DPL_ZERODIV - division by zero attempted

dpl_error is set to the returned value.

dpl_stencil_acc

int dpl_stencil_acc(unsigned long n, const double *a, const double *b, double *x)

DESCRIPTION

computes **acc = a[i]*acc + b[i]** for **i** = **stride**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input vectors **a**, **b**.*

a, **b**

*Input vectors of size **n**; elements from **0** to **n-1** are used.*

x

*Input/output vector of size **1**: element **x[0]** used as an input as well as set by this routine (used as an output).*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

dpl_stencil_div_acc

int dpl_stencil_acc(unsigned long n, const double *a, const double *b, double *x)

DESCRIPTION

computes **acc = a[i]/acc + b[i]** for **i** = **stride**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input vectors **a**, **b**.*

a, **b**

*Input vectors of size **n**; elements from **0** to **n-1** are used.*

x

*Input/output vector of size **1**: element **x[0]** used as an input as well as set by this routine (used as an output).*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_BAD_STRIDE - problematic **stride** parameter specified
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected
DPL_ZERODIV - division by zero attempted

dpl_error is set to the returned value.

dpl_stencil_2p

int dpl_stencil_2p(unsigned long n, const double *a, const double *c, const double *b, double *x)

DESCRIPTION

computes **x[i] = a[i]*x[i-1] * c[i]*x[i-2] + b[i]** for **i** = **2**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input vectors **a**, **b**.*

a, **c**, **b**

*Input vectors of size **n**; elements from **2** to **n-1** are used.*

x

*Input/output vector of size **n**: elements **0** and **1** are used as an input; elements from **2** to **n-1** are set by this routine (used as an output).*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected

DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

dpl_stencil

int dpl_stencil(unsigned int n, const double *a, const double *b, double *x)

DESCRIPTION

computes **x[i] = b[i] +** $\sum_{j=0}^{i-1}$ **a[i][j]*x[j]** for **i** = **0**...**n**-**1**.

INPUT PARAMETERS

n
The number of elements in the input/output vectors.
a
*Input matrix of size **n**x**n**.*
b
*Input vector of size **n**.*
x
*output vector of size **n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

Routines for parallel implementation of plain conditional functions

dpl_stencil_min

int dpl_stencil_min(unsigned long n, const double *a, const double *b, double *x)

DESCRIPTION

computes **x[i] = min(a[i]*x[i-1], b[i])** for **i** = **1**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

a, **b**

*Input vectors of size **n**; only elements from **1** to **n-1** are used.*

x

*Input/output vector of size **n**: element **0** used as an input; elements from **1** to **n-1** are set by this routine (used as an output).*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

dpl_stencil_max

int dpl_stencil_max(unsigned long n, const double *a, const double *b, double *x)

DESCRIPTION

computes **x[i] = max(a[i]*x[i-1], b[i])** for **i** = **1**...**n**-**1**

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

a, **b**

*Input vectors of size **n**; only elements from **1** to **n-1** are used.*

x

*Input/output vector of size **n**: element **0** used as an input; elements from **1** to **n-1** are set by this routine (used as an output).*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region

DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

Routine for parallel implementation of stochastic ordering

dpl_max_st

int dpl_max_st(unsigned long n, const double *a, const double *b, double *x)

DESCRIPTION

computes the stochastic ordering of the input vectors **a** and **b** of the size **n** calculating the output vector **x** of the size **n**

$x = max_{st}(a, b)$

as the solution of the following system of equations:

for every **i** = **0**...**n**-**1**: $\sum_{j=i}^{n-1} x[j] = \max\left(\sum_{j=i}^{n-1} a[j], \sum_{j=i}^{n-1} b[j] \right)$

INPUT PARAMETERS

n

*The number of elements in the input/output vectors **a**, **b** and **x**.*

a, **b**

*Input vectors of size **n**.*

x

*output vector of size **n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

dpl_stochastic_max

int dpl_stochastic_max(unsigned long n, const double *b, double *x)

DESCRIPTION

calculates the output matrix **x** of the size **n**x**n** to be the stochastic matrix of the input matrix **b** of the size **n**x**n** as the following stencil:

(with **m[j,*]** representing **j**'s row of the matrix **m**, $\mathbf{max_{st}}$ being defined in the previous paragraph)

x[0,*] = b[0,*]
x[i,*] = $\mathbf{max_{st}}$(x[i-1,*], b[i,*]) for **i** = **1**...**n**-**1**

INPUT PARAMETERS

n

*The number of rows in the input/output squares matrices **b** and **x**.*

b

*Input matrix of size **n**x**n**.*

x

*output matrix of size **n**x**n**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

Routine for parallel implementation of the Gauss-Seidel method to solve systems of linear equations

dpl_gs

int dpl_gs(unsigned int n, const double *a, const double *b, double *x, double tolerance, unsigned int *iterationcount)

DESCRIPTION

solves a square system of **n** linear equations **ax = b** using Gauss–Seidel method.
Implements interactive process with the next iteration **k+1** $\mathbf{x^{k+1}}$ being calculated by applying Gauss-Seidel method to the value of the initial/previous iterations $\mathbf{x^k}$. After that the norm

$\boldsymbol{\varepsilon_{k+1}} = |\, \mathbf{x^{k+1}} - \mathbf{x^k} |_\infty = \max(|\mathbf{x^{k+1}[i]} - \mathbf{x^k[i]}|)$ for **i** = **0**,...,**n-1**

is calculated.

If $\boldsymbol{\varepsilon_{k+1}}$ **< tolerance** the process is terminated and **DPL_SUCCESS** is returned.

If $\varepsilon_k < \varepsilon_{k+1}$ the process is terminated and **DPL_NO_IMPRVMNT** is returned.
if **k** exceeds the *number of iterations allowed* (explained later) the process is terminated and **DPL_FAILED_REACH_TOLERANCE** is returned.

the *number of iterations allowed* is determine by the **iterationcount** parameter:

if **iterationcount** is **NULL** - no restriction on the number of iterations is imposed otherwise the value that **iterationcount** contains is used as the maximum number of iterations allowed; if this value is equal to **0** - no restriction on the number iterations is imposed.

INPUT PARAMETERS

n

The number of elements in the input/output vectors.

a

*Input matrix of size **n**x**n**.*

b

*Input vector of size **n**.*

x

*output vector of size **n**.*

tolerance

controls termination of the interactive process.

iterationcount

*pointer to the input/output parameter that on input establishes maximum number of iterations to be performed and on output is set to the number of iterations that were performed; may be **NULL**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NOT_ENOUGH_THREADS - the function called requires more execution threads that available
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected
DPL_ZERODIV - division by zero attempted
DPL_FAILED_REACH_TOLERANCE - failed to reach specified tolerance
DPL_NO_IMPRVMNT - Gauss-Seidel iteration is not improving the solution

dpl_error is set to the returned value.

Routine for parallel implementation of the general 2-D 5-points stencil

dpl_p1p0_p0p1_p0p0_p0n1_n1p0

```
int dpl_p1p0_p0p1_p0p0_p0n1_n1p0( unsigned long rows, unsigned long cols,
                                  const double *ap1p0,
                                  const double *ap0p1,
                                  const double *ap0p0,
                                  const double *ap0n1,
                                  const double *an1p0,
                                  const double *b,
                                  double *x
                                )
```

NOTATION

for an elemnt (**r**,**c**) of a 2-dimensional matrix define elements as following:

[pn]#1[pn]#2

meaning

p - previous or the same

n - next

- number of elements from (**r**,**c**) according to [**pn**]

for example **p1n2** is the element (**r**-1,**c**+2)

in the following grid, that defines 2-D 5-points stencil

	c-1	**c**	**c**+1
r-1		**r**-1,**c**	
r	**r**,**c**-1	**r**,**c**	**r**,**c**+1
r+1		**r**+1,**c**	

the elements will be

r-1,**c**	**p1p0**
r,**c**-1	**p0p1**
r,**c**	**p0p0**
r,**c**+1	**p0n1**
r+1,**c**	**n1p0**

DESCRIPTION

computes

```
for ( r = 1 ; r < rows - 1; r++ )
 {
  for ( c = 1 ; c < cols - 1 ; c++ )
```

```
  {
    x[r][c] = x[r-1][c]*ap1p0[r][c] +
              x[r][c-1]*ap0p1[r][c] +
              x[r][c]*ap0p0[r][c] +
              x[r][c+1]*ap0n1[r][c] +
              x[r+1][c]*an1p0[r][c] +
              b[r][c];
  }
}
```

INPUT PARAMETERS

rows

The number of rows of a 2-dimensional matrices involved.

cols

The number of columns of a 2-dimensional matrices involved.

ap1p0, ap0p1, ap0p0, ap0n1, an1p0

*Input 2-dimensional matrices of the size **rows**x**cols**.*

b

*Input 2-dimensional matrix of the size **rows**x**cols**.*

x

*input/output 2-dimensional matrix of the size **rows**x**cols**.*

RETURN VALUES

DPL_SUCCESS - no errors, successful execution, set to **0**
DPL_NOT_ENOUGH_ELEMENTS - more elements necessary to benefit from parallel execution
DPL_FAILED_ALLOC_MEMORY - failed allocate memory
DPL_NO_THREADS - no multi-thread hardware detected
DPL_IN_PARALLEL - call is made inside parallel region
DPL_OVRFLW - overflow was detected

dpl_error is set to the returned value.

Auxiliary routines

dpl_aux_strerror

const char *dpl_aux_strerror()

DESCRIPTION

returns a pointer to a string that describes the error code generated by the last call to a parallel routine from the ***dpl*** library; note that the global variable **dpl_error** contains an error code of the last parallel ***dpl*** function executed.

RETURN VALUES

a pointer to a string that describes the error code generated by the last call to a routine from the

dpl library.

Table of Contents

General....2
Technical specifications....2
Installation....2
Compiling and linking with the dpl library....3
Error Codes....3
Programming with dpl....4
Global Variables....4
dpl_error....4
dpl_version....4
Using dpl in C/C++....4
Stencils....5
dpl_stencil_1p....5
dpl_stencil_o_1p....6
dpl_stencil_div_1p....6
dpl_stencil_div_o_1p....7
dpl_stencil_acc....8
dpl_stencil_div_acc....8
dpl_stencil_2p....9
dpl_stencil....10
Routines for parallel implementation of plain conditional functions....10
dpl_stencil_min....10
dpl_stencil_max....11
Routine for parallel implementation of stochastic ordering....12
dpl_max_st....12
dpl_stochastic_max....12
Routine for parallel implementation of the Gauss-Seidel method to solve systems of linear equations 13
dpl_gs....13
Routine for parallel implementation of the general 2-D 5-points stencil....15
dpl_p1p0_p0p1_p0p0_p0n1_n1p0....15
Auxiliary routines....16
dpl_aux_strerror....16