FUJITSU

FUJITSU Software
Technical Computing Suite V4.0L20

Development Studio
Fast Basic Operations Library for Quadruple Precision User's Guide

J2UL-2576-01ENZ0(00)
February 2020

Preface

This manual describes the use of Fast Basic Operations Library for Quadruple Precision (referred to as fast_dd, throughout this manual).

This manual is organized as follows.

1. Overview

introduces and describes an overview of fast_dd.

2. Use of Fortran version

describes how to use fast_dd in Fortran programs.

3. Use of C++ version

describes how to use fast_dd in C++ programs.

4. Error Messages

describes error messages fast_dd provides.

Export Controls

Exportation/release of this document may require necessary procedures in accordance with the regulations of your resident country and/or US export control laws.

Date of Publication and Version

Version	Manual code
February 2020, 1st Version	J2UL-2576-01ENZ0(00)

Contents

List of Figures

List of Tables

1. Overview

This chapter introduces and gives overview of Fast Basic Operations Library for Quadruple Precision.

1.1 Fast Basic Operations Library for Quadruple Precision

When running computational programs, there are situations where some higher-precision arithmetic operations are required. For example:

- In the double precision programs, there may be some critical parts of code in accuracy where some quadruple precision computation is required.
- In order to check the correctness of results obtained by double precision, the user may need a quadruple precision computation.

Most of compilers provide a quadruple precision as intrinsic data type, so the user can use it. However the implementation relies on software emulation, which results in not a performance of satisfactory.

To reduce such performance issue, Fast Basic Operations Library for Quadruple Precision (referred to as fast_dd, in this manual) is provided. The user would get performance benefit from using fast_dd while obtaining high precision results.

The fast_dd is a library in which a quadruple precision number is expressed in double-double format and arithmetic operations are performed on such formatted numbers. In the format, a quadruple precision number is stored in two double precision variables. Arithmetic operations on such quadruple precision numbers can be processed by using double precision hardware instructions provided on the processor. So, in most cases, it is significantly faster than the intrinsic quadruple precision in the compilers.

In terms of accuracy, the compiler intrinsic quadruple precision has approximately 33 decimal digits while fast_dd double-double data type has approximately 31 decimal digits. If application programs can admit this small loss of accuracy, then fast_dd should be the choice for significantly better performance.

The fast_dd library can be used in both Fortran and C++ programs. Note that its capability and usage is different between Fortran and C++ due to their language specification.

The fast_dd supports assignment, four basic arithmetic operations, comparison, numeric functions and some mathematical functions. For assignment, four basic arithmetic operations and comparison, fast_dd provides the same operators as those provided in Fortran and C++, so it allows the user to write friendly codes. In addition, fast_dd provides multi-vector operation routines. The routines provided by fast_dd are thread safe, so the user can use them from codes that are thread-parallelized by OpenMP or automatic parallelization through compilers.

1.2 double-double format

In the double-double format that fast_dd employs, a quadruple precision number is expressed as a sum of two IEEE double precision numbers.

Figure 1.2.1 double-double format

leading component	s	e	f
	63	62 … 52	51 … 0
trailing component	s	e	f
	63	62 … 52	51 … 0

s : sign 1 bits

e : exponent 11 bits

f : fraction 52 bits

Specifically, a fast_dd double-double number is represented as a structure as follows.

Fortran real double-double format

```
type dd_real
  real(8)::re(2)
end type dd_real
```

Fortran complex double-double format

The cmp(1:2) contains real part and cmp(3:4) contains imaginary part.

```
type dd_complex
  real(8)::cmp(4)
end type dd_complex
```

C++ real double-double format

```
struct dd_real {
        double x[2];
};
```

2. Use of Fortran version

This chapter describes how to use fast_dd from Fortran programs. A simple example code is presented first, then detailed features and specification are explained.

2.1 Example program

Here is a Fortran example code using fast_dd. The code computes the mathematical constant *e* (the base of natural logarithm) of double-double data type by Taylor series.

```
!
!   MAIN PROGRAM
!
     program fast_dd_sample
     use fast_dd                          ! (1)
     type(dd_real)::x,y                   ! (2)
     type(dd_real)::dd_exp_sample
     real(16)::xx,yy,wy
!
!   call dd_exp_sample
!
     x=1.0d0
     y=dd_exp_sample(x)
!
!   call Fortran qexp
!
     xx=x
     yy=qexp(xx)
!
     wy=y
     write(6,100) wy                      ! (4)
     write(6,200) yy
 100  format('dd_exp_sample(1.0) :',F30.25 )
 200  format('qexp(1.0)         :',F30.25 )
     end
!
!   FUNCTION dd_exp_sample
!
     function dd_exp_sample(x)            ! (3)
     use fast_dd                          ! (1)
     type(dd_real)::x,dd_exp_sample       ! (2)
     type(dd_real)::wx,wy,wy0,wc
     real(8)::c
!
!    exp(x)=1 + x + x**2/2! + x**3/3! + x**4/4! + ...
!
     wx=x
     wy0=1.0d0
     c=2.0d0
     wc=1.0d0
     wy =1.0d0+wx*wc    ! wy=1+x
     do while(abs(wy0-wy)>=1d-25)
       wy0=wy
       wc=wc/c
```

```
    wx=wx*x
    wy=wy+wx*wc    ! wy=wy+x**i/i!
    c=c+1.0
  end do
  dd_exp_sample=wy
  end
```

Explanation

(1) The code using fast_dd has to contain use statement for fast_dd. The use statement is required in any of the main program, subroutines, or functions if fast_dd is used in the procedure.

(2) Variables of double-double data type have to be declared by type(dd_real).

(3) Variables of type dd_real can be used as arguments of subroutines or functions, and also as returned values of functions.

(4) Since fast_dd does not support input/output capabilities, the user is required to use variables of type real(16) when the user wants to output fast_dd data type.

2.2 List of features

The features that the Fortran version of fast_dd provides are listed in Table 2.2.1 through Table 2.2.4.

Table 2.2.1 Assignment and Operations

Operator	Operation
=	Assignment
+, -, *, /	Four basic arithmetic operations
-	Unary negation
.EQ. , .NE. , .LT. , .LE. , .GT. , .GE. , == , /= , < , > , <= , >=	Relational operations

Table 2.2.2 Numeric functions

Function name	Operation
ddreal	Type conversion to dd_real
ddcomplex	Type conversion to dd_complex
int	Type conversion from dd_real to integer
real	Real part of dd_complex
dble	Type conversion to real(8)
cmplx	Type conversion to complex(8)
abs	Absolute value
sign	Sign transfer
max	Maximum
min	Minimum
aint	Trancation to integer
anint	Round fraction part in the nearest mode
aimag	Imaginary part of complex number

Function name	Operation
conjg	Conjugate complex

Table 2.2.3 Mathematical functions

Function name	Operation
sqrt	Square root
exp	Exponential
log	Natural logarithm
sin	Sine
cos	Cosine
sincos	Sine and Cosine

Table 2.2.4 Multi-operations and vector operations

Routine name	Operation
m2_add_dd	Two addition
m2_sum_dd	Accumulation
m2_sub_dd	Two subtraction
m2_mul_dd	Two multiplication
v_add_dd	Vector addition
v_sub_dd	Vector subtraction
v_mul_dd	Vector multiplication
vm_add_dd	Vector addition (thread-parallel)
vm_sub_dd	Vector subtraction (thread-parallel)
vm_mul_dd	Vector multiplication (thread-parallel)

2.3 fast_dd module

When using fast_dd from Fortran program, the module fast_dd is used. The user is requested to write the statement "use fast_dd" in any of the main program, subroutine, or function whenever fast_dd features is used in each procedure.

Example: Using fast_dd in the main program and subroutine

```
program main
use fast_dd
   :
end
subroutine sub()
use fast_dd
   :
end subroutine
```

2.4 Variable declaration

The Fortran version of fast_dd provides two data types: dd_real for real numbers and dd_complex for complex numbers.

Example1: Declaration of scalar variable “x” and array “a” of type dd_real

```
type(dd_real):: x,a(100)
```

Example2: Declaration of scalar variable “zx” and array “za” of type dd_complex

```
type(dd_complex):: zx,za(100)
```

2.5 Assignment

The user can use assignment statement with variables of type dd_real and dd_complex. In addition it is also allowed to use assignment between these types and real(8), real(16), integer(4), complex(8), and complex(16). The whole set of assignment features is listed in Table 2.5.1.

Table 2.5.1 Assignment

	Type of left hand side	Type of right hand side
Assignment (=)	dd_real	dd_real
	dd_complex	dd_complex
	real(8)	dd_real
	real(16)	dd_real
	integer(4)	dd_real
	dd_real	real(8)
	dd_real	real(16)
	dd_real	integer(4)
	dd_real	dd_complex
	complex(8)	dd_complex
	complex(16)	dd_complex
	real(8)	dd_complex
	dd_complex	dd_real
	dd_complex	complex(8)
	dd_complex	complex(16)
	dd_complex	real(8)
	dd_complex	integer(4)

Example1: Assignment

```
type(dd_real)::a,b
type(dd_complex)::za,zb
a=b
za=zb
```

Example2: Assignment of integer(4) or real(8) to dd_real

```
type(dd_real)::a
integer::n
real(8)::d
a=n
```

```
a=10
a=d
a=1.0d-5      ! Be careful about precision in assignment of real(8)
```

Example3: Assignment of variables or constants of type integer(4), real(8), or complex(8) to dd_complex

```
type(dd_complex)::a
integer::n
real(8)::d
complex(8)::z
a=n
a=10
a=d
a=z
a=(1.0d0,2.0d0)
```

Example4: Assignment of dd_real to integer(4) or real(8)

```
type(dd_real)::a
integer::n
real(8)::d
d=a
n=a
```

Example5: Assignment of dd_complex to real(8) or complex(8)

```
type(dd_complex)::a
real(8)::d
complex(8)::z
d=a
z=a
```

Example6: Assignment of variables or constants of type real(16) or complex(16) to dd_real or dd_complex

```
type(dd_real)::a
type(dd_complex)::c
real(16)::q
complex(16)::z
a=q
a=0.1q-5
c=z
```

Example7: Assignment to real(16) or complex(16)

```
type(dd_real)::x
type(dd_complex)::c
real(16)::q
complex(16)::z
q=x
z=c
```

2.6 Operations

Four basic arithmetic operations +, –, *, / and unary negation operation – are provided. For arithmetic operations, not only operations between dd_real or dd_complex types but also mixed operations which involve real(8) or integer are provided. The whole set of arithmetic operations available is listed in Table 2.6.1 through Table 2.6.3.

Table 2.6.1 Addition and subtraction

Operation	Type of left hand side	Type of right hand side	Type of result

Operation	Type of left hand side	Type of right hand side	Type of result
Addition and subtraction	dd_real	dd_real	dd_real
	real(8)	dd_real	dd_real
	dd_real	real(8)	dd_real
	integer(4)	dd_real	dd_real
	dd_real	integer(4)	dd_real
	dd_complex	dd_complex	dd_complex
	real(8)	dd_complex	dd_complex
	dd_complex	real(8)	dd_complex
	dd_complex	dd_real	dd_complex
	dd_real	dd_complex	dd_complex

Table 2.6.2 Multiplication

Operation	Type of left hand side	Type of right hand side	Type of result
Multiplication	dd_real	dd_real	dd_real
	real(8)	dd_real	dd_real
	dd_real	real(8)	dd_real
	integer(4)	dd_real	dd_real
	dd_real	integer(4)	dd_real
	dd_complex	dd_complex	dd_complex
	real(8)	dd_complex	dd_complex
	dd_complex	real(8)	dd_complex
	integer(4)	dd_complex	dd_complex
	dd_complex	integer(4)	dd_complex
	dd_complex	dd_real	dd_complex
	dd_real	dd_complex	dd_complex

Table 2.6.3 Division

Operation	Type of left hand side	Type of right hand side	Type of result
Division	dd_real	dd_real	dd_real
	real(8)	dd_real	dd_real
	dd_real	real(8)	dd_real
	integer(4)	dd_real	dd_real
	dd_real	integer(4)	dd_real
	dd_complex	dd_complex	dd_complex

Operation	Type of left hand side	Type of right hand side	Type of result
	dd_complex	dd_real	dd_complex
	dd_real	dd_complex	dd_complex
	dd_complex	real(8)	dd_complex

Example1: Arithmetic operations between two of dd_real

```
type(dd_real)::a,b,c,d,e
c=a+b
c=a-b
c=a*b
c=a/b
c=a*b+d/e
c=-a              ! unary negation
```

Example2: Arithmetic operations between two of dd_complex

```
type(dd_complex)::za,zb,zc,zd,ze
zc=za+zb
zc=za-zb
zc=za*zb
zc=za/zb
zc=za*zb+zd/ze;
```

Example3: For operations with dd_real, variables or constants of type real(8) can appear on the left or right hand side

```
type(dd_real)::a,b,c
real(8)::d
c=d+a
c=a-123.0d0
c=2.0d0*b
c=d/b
```

Example4: For operations with dd_real, variables or constants of type integer(4) can appear on the left or right hand side

```
type(dd_real)::a,b,c
integer(4)::n
c=a+123
c=a-n
c=a*n
c=a/2
```

2.7 Relational operations

Relational operations for types dd_real and dd_complex are provided as Table 2.7.1 shows.

Table 2.7.1 Relational operations

Relational operation	Type of left hand side	Type of right hand side	Type of result
.EQ. , .NE. , == , /=	dd_real	dd_real	logical
	real(8)	dd_real	logical
	dd_real	real(8)	logical
	integer(4)	dd_real	logical

Relational operation	Type of left hand side	Type of right hand side	Type of result
	dd_real	integer(4)	logical
	dd_complex	dd_complex	logical
	dd_real	dd_complex	logical
	dd_complex	dd_real	logical
.GT. , .GE. , .LT. , .LE. , > , >= , < , <=	dd_real	dd_real	logical
	real(8)	dd_real	logical
	dd_real	real(8)	logical
	integer(4)	dd_real	logical
	dd_real	integer(4)	logical

Example1: Relational operations

```
type(dd_real)::a,b,c
type(dd_complex)::x,y
if(a>b) then
   c=0.0d0
end if
if(a>=b.and.(c/=0.0d0.or.c/=1.0d0)) then
   c=0.0d0
end if
if(x==y) then
   y=(0.0d0,0.0d0)
end if
```

2.8 Numeric functions

2.8.1 ddreal

Usage format

y=ddreal(x)

x   either dd_real, integer, real(8) or dd_complex

y   dd_real

Function

When x is either dd_real, integer, or real(8), this function converts x to dd_real. When x is dd_complex, the function returns the real part of x.

Example

```
type(dd_real)::y
type(dd_complex)::z
y=ddreal(1.0d0)
y=ddreal(123)
y=ddreal(z)
```

2.8.2 ddcomplex

Usage format

y=ddcomplex(xr,xi)

xr dd_real

xi dd_real

y dd_complex

or,

y=ddcomplex(xr)

xr either dd_real, real(8) or complex(8)

y dd_complex

Function

This function returns y of type dd_complex whose real and imaginary part is xr and xi, respectively.

If xi is not present, the imaginary part is set to 0.0.

Example

```
type(dd_complex)::y
type(dd_real)::r,c
y=ddcomplex(r,c)
y=ddcomplex(1.0d0)
```

2.8.3 int

Usage format

n=int(x)

x dd_real

n integer

Function

This function converts dd_real to integer. If |x|<1.0 the result is 0. Otherwise, the result y is a maximum integer whose absolute value is less than or equal to |x| and the sign of y retains that of x.

Example

```
type(dd_real)::x
integer::n
n=int(x)+50
```

2.8.4 real

Usage format

y=real(x)

x dd_complex

y dd_real

Function

This function returns the real part of x.

Example

```
type(dd_real)::y
type(dd_complex)::x
y=real(x)
```

2.8.5 dble

Usage format

y=dble(x)

x dd_real

y real(8)

or,

y=dble(x)

x dd_complex

y real(8)

Function

When x is type dd_real, this function converts x to type real(8). When x is type dd_complex, the function converts the real part of x to type real(8).

Example

```
type(dd_real)::x
type(dd_complex)::z
real(8)::d,e
d=dble(x)+1.5
e=dble(z)
```

2.8.6 cmplx

Usage format

y=cmplx(x)

x dd_complex

y complex(8)

Function

This function converts dd_complex to complex(8).

Example

```
type(dd_complex)::x
complex(8)::z
z=cmplx(x)
```

2.8.7 abs

Usage format

y=abs(x)

x either dd_real or dd_complex

y dd_real

Function

This function returns the absolute value of x.

Example

```
type(dd_real)::x,y
type(dd_complex)::z
y=abs(x)
y=abs(z)
```

2.8.8 sign

Usage format

y=sign(x,s)

x dd_real

s	either dd_real or real(8)

y	dd_real

Function

This function returns the absolute value of x times the sign of s.

Example

```
type(dd_real)::x,y
x=10.0d0
y=sign(x,-1.0d0)        ! -10.0 is assigned to y
```

2.8.9 max

Usage format

y=max(x1,x2[,x3,...])

x1, x2,...	dd_real

y	dd_real

Function

This function returns the maximum value. The number of arguments ranges from 2 to 9.

Example

```
type(dd_real)::x1,x2,x3,x4,x5,x6,x7,x8,x9,y
y=max(x1,x2)
y=max(x1,x2,x3,x4,x5,x6,x7,x8,x9)
```

2.8.10 min

Usage format

y=min(x1,x2[,x3,...])

x1, x2,...	dd_real

y	dd_real

Function

This function returns the minimum value. The number of arguments ranges from 2 to 9.

Example

```
type(dd_real)::x1,x2,x3,x4,x5,x6,x7,x8,x9,y
y=min(x1,x2)
y=min(x1,x2,x3,x4,x5,x6,x7,x8,x9)
```

2.8.11 aint

Usage format

y=aint(x)

X	dd_real

Y	dd_real

Function

This function truncates the fraction part of x. If |x|<1.0 the result is 0.0. Otherwise, the result y is a maximum integer whose absolute value is less than or equal to |x| and the sign of y retains that of x.

Example

```
type(dd_real)::x,y
x=5.678d0
y=aint(x)        ! 5.0 is assigned to y
```

```
x=-5.678d0
y=aint(x)         ! -5.0 is assigned to y
```

2.8.12 anint

Usage format

y=anint(x)

x	dd_real
y	dd_real

Function

This function rounds off the fraction part of x in the nearest mode. If x is 0.0 or positive number, the result becomes aint(x+0.5). If x is negative, the result becomes aint(x-0.5).

Example

```
type(dd_real)::x,y
x=5.678d0
y=anint(x)         ! 6.0 is assigned to y
```

2.8.13 aimag

Usage format

y=aimag(x)

x	dd_complex
y	dd_real

Function

This function returns the imaginary part of x.

Example

```
type(dd_complex)::z
type(dd_real)::y
y=aimag(z)
```

2.8.14 conjg

Usage format

y=conjg(x)

x	dd_complex
y	dd_complex

Function

This function returns the conjugate complex of x.

Example

```
type(dd_complex)::y,z
complex(8)::c
z=conjg(y)
```

2.9 Mathematical functions

2.9.1 sqrt

Usage format

y=sqrt(x)

x	dd_real
y	dd_real

Function

This function returns the square root of x. x>=0 must be satisfied.

Example

```
type(dd_real)::x,y
y=sqrt(x)
```

2.9.2 exp

Usage format

y=exp(x)

x	dd_real
y	dd_real

Function

This function returns the exponential of x. x<709.0 must be satisfied.

Example

```
type(dd_real)::x,y
y=exp(x)
```

2.9.3 log

Usage format

y=log(x)

x	dd_real
y	dd_real

Function

This function returns the natural logarithm of x. x>0 must be satisfied.

Example

```
type(dd_real)::x,y
y=log(x)
```

2.9.4 sin

Usage format

y=sin(x)

x	dd_real
y	dd_real

Function

This function returns the sine of x. |x|<1.4e19 must be satisfied.

Example

```
type(dd_real)::x,y
y=sin(x)
```

2.9.5 cos

Usage format

y=cos(x)

x	dd_real
y	dd_real

Function

This function returns the cosine of x. |x|<1.4e19 must be satisfied.

Example

```
type(dd_real)::x,y
y=cos(x)
```

2.9.6 sincos

Usage format

call sincos(x,s,c)

x	input	dd_real
s	output	dd_real
c	output	dd_real

Function

This subroutine returns the sine and cosine of x to s and c, respectively. When both sin(x) and cos(x) for the same x are required in applications this subroutine is useful since it is faster than using the functions sin and cos separately. |x|<1.4e19 must be satisfied.

Example

```
type(dd_real)::x,s,c
call sincos(x,s,c)
```

2.10 Multi-vector operation routines

Multi-operation routines perform two operations by a single call so that arithmetic units can work efficiently. Similarly, vector operation routines perform operations efficiently on an array of type dd_real.

Thread-parallel vector operation routines are also provided which perform OpenMP type of parallel operations within the routines using multiple cores. Note that these routines are thread safe like other fast_dd functions or routines so that the user can call these routines from inside or outside OpenMP parallel regions.

2.10.1 m2_add_dd

Usage format

call m2_add_dd(a,b,c,x,y,z)

a	input	dd_real
b	input	dd_real
c	output	dd_real
x	input	dd_real
y	input	dd_real
z	output	dd_real

Function

This subroutine computes c=a+b, z=x+y.

Example

```
type(dd_real)::a,b,c,x,y,z
call m2_add_dd(a,b,c,x,y,z)
```

2.10.2 m2_sum_dd

Usage format

call m2_sum_dd(a,b,c)

A	input	dd_real
B	input	dd_real
C	input/output	dd_real

Function

This subroutine computes c=c+a+b.

Example

```
type(dd_real)::a,b,c
call m2_sum_dd(a,b,c)
```

2.10.3 m2_sub_dd

Usage format

call m2_sub_dd(a,b,c,x,y,z)

a	input	dd_real
b	input	dd_real
c	output	dd_real
x	input	dd_real
y	input	dd_real
z	output	dd_real

Function

This subroutine computes c=a-b, z=x-y.

Example

```
type(dd_real)::a,b,c,x,y,z
call m2_sub_dd(a,b,c,x,y,z)
```

2.10.4 m2_mul_dd

Usage format

call m2_mul_dd(a,b,c,x,y,z)

a	input	either dd_real or real(8)
b	input	dd_real
c	output	dd_real
x	input	either dd_real or real(8)
y	input	dd_real
z	output	dd_real

Function

This subroutine computes c=a*b, z=x*y.   Data type of a and x must be same.

Example

```
type(dd_real)::a,b,c,x,y,z
```

```
real(8)::p,q
call m2_mul_dd(a,b,c,x,y,z)
call m2_mul_dd(p,b,c,q,y,z)
```

2.10.5 v_add_dd

Usage format

call v_add_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either integer(4) or integer(8), the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This subroutine computes c=a+b for arrays a, b and c.

Example

```
type(dd_real)::a(100),b(100),c(100)
call v_add_dd(a,b,100,c)
```

2.10.6 v_sub_dd

Usage format

call v_sub_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either integer(4) or integer(8), the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This subroutine computes c=a-b for arrays a, b, and c.

Example

```
type(dd_real)::a(100),b(100),c(100)
call v_sub_dd(a,b,100,c)
```

2.10.7 v_mul_dd

Usage format

call v_mul_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either integer(4) or integer(8), the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This subroutine computes c=a*b for arrays a, b, and c.

Example

```
type(dd_real)::a(100),b(100),c(100)
call v_mul_dd(a,b,100,c)
```

2.10.8 vm_add_dd

Usage format

call vm_add_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either integer(4) or integer(8), the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This subroutine is a thread-parallel subroutine to compute c=a+b for arrays a, b, and c.

Example

```
type(dd_real)::a(100),b(100),c(100)
call vm_add_dd(a,b,100,c)
```

2.10.9 vm_sub_dd

Usage format

call vm_sub_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either integer(4) or integer(8), the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This subroutine is a thread-parallel subroutine to compute c=a-b for arrays a, b, and c.

Example

```
type(dd_real)::a(100),b(100),c(100)
call vm_sub_dd(a,b,100,c)
```

2.10.10 vm_mul_dd

Usage format

call vm_mul_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either integer(4) or integer(8), the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This subroutine is a thread-parallel subroutine to compute c=a*b for arrays a, b, and c.

Example

```
type(dd_real)::a(100),b(100),c(100)
call vm_mul_dd(a,b,100,c)
```

2.11 Error handling

Some of fast_dd routines checks input arguments for consistency. See “4 Error Messages” for the details. If the routines detect some errors they set NaN as a value and continue computation. Such error handling can be changed by calling the following routine.

Usage format

call fast_dd_errlvl(n)

n	input	Integer. Specify value from the following selection, which determines action for fast_dd to take after calling this routine. 0: issue no messages and continue computation 10: issue error messages and continue computation 90:issue error messages and terminate computation If n is set to different value from any of the above values, the subroutine does not change error handling.

Function

This subroutine changes the way of error handling in fast_dd.

3. Use of C++ version

This chapter describes how to use fast_dd from C++ programs. A simple example code is presented first, then detailed features and specification are explained.

3.1 Example program

Here is a C++ example code using fast_dd. The code computes the mathematical constant *e* (the base of natural logarithm) of double-double data type by Taylor series.

```
//
//   Include header files
//
#include <iostream>
#include <iomanip>
#include <math.h>
#include <fast_dd.h>                           // (1)

dd_real dd_exp_sample(const dd_real &x);

//
//  MAIN PROGRAM
//
int main()
{
   dd_real x,y;                                // (2)
   long double xx,yy,wy;
//
//  call dd_exp_sample
//
   x=1.0;
   y=dd_exp_sample(x);
//
//  call expl
//
   xx=to_long_double(x);
   yy=expl(xx);
   wy=to_long_double(y);
   std::cout <<"dd_exp_sample(1.0) :"
           <<std::setprecision(20)
           <<std::setw(25)<<wy<<std::endl;   // (4)
   std::cout <<"expl(1.0)        :"
           <<std::setprecision(20)
           <<std::setw(25)<<yy<<std::endl;
   return 0;
}

//
//  Function dd_exp_sample
//
dd_real dd_exp_sample(const dd_real &x)      // (3)
{
   dd_real wx,wy,wy0,wc;                      // (2)
   double c;
```

```
//
// exp(x)=1 + x + x**2/2! + x**3/3! + x**4/4! + ...
//
   wx=x;
   wy0=1.0;
   c=2.0;
   wc=1.0;
   wy=1.0+wx*wc;        //  wy=1+x
   while(abs(wy0-wy)>=1e-20) {
      wy0=wy;
      wc=wc/c;
      wx=wx*x;
      wy=wy+wx*wc;     //  wy=wy+x**i/i!
      c=c+1.0;
   }
   return wy;
}
```

Explanation

(1) The code using fast_dd has to include a header file fast_dd.h.

(2) Variables of double-double data type have to be declared by dd_real.

(3) Variables of type dd_real can be used as arguments of functions, and also as returned values of functions.

(4) Since fast_dd does not support input/output capabilities, the user is required to use variables of type long double when the user wants to output fast_dd data type.

3.2 List of features

The features that the C++ version of fast_dd provides are listed in Table 3.2.1 through Table 3.2.4.

Table 3.2.1 Assignment and Operations

Operator	Operation
=	Assignment
+, -, *, /	Four basic arithmetic operations
-	Unary negation
+=, -=, *=, /=	Assignment operations
== , != , < , > , <= , >=	Relational operations

Table 3.2.2 Numeric functions

Function name	Operation
dd_real	Type conversion to dd_real
to_int	Type conversion from dd_real to integer
to_double	Type conversion from dd_real to double
to_long_double	Type conversion from dd_real to long double
fabs, abs	Absolute value

Function name	Operation
max	Maximum
min	Minimum
aint	Trancation to integer
nint	Round fraction part in the nearest mode

Table 3.2.3 Mathematical functions

Function name	Operation
sqrt	Square root
exp	Exponential
log	Natural logarithm
sin	Sine
cos	Cosine
sincos	Sine and Cosine

Table 3.2.4 Multi-operations and vector operations

Function name	Operation
m2_add_dd	Two addition
m2_sum_dd	Accumulation
m2_sub_dd	Two subtraction
m2_mul_dd	Two multiplication
v_add_dd	Vector addition
v_sub_dd	Vector subtraction
v_mul_dd	Vector multiplication
vm_add_dd	Vector addition(thread-parallel)
vm_sub_dd	Vector subtraction(thread-parallel)
vm_mul_dd	Vector multiplication(thread-parallel)

3.3 Header file

When using fast_dd from C++ program, a header file fast_dd.h has to be included.

Example:

```
#include <fast_dd.h>
```

3.4 Variable declaration

The C++ version of fast_dd provides a data type dd_real. Variables of type dd_real are declared as follows.

Example 1: Declaration of scalar variable "x" and array "a" of type dd_real.

```
dd_real a;
dd_real x[100];
```

Example 2: Declare of some variables and an array of type dd_real and initialize them.

```
dd_real a(1.0,1e-18);     // initialize with leading and trailing component
dd_real b=1;              // initialize with an integer value
dd_real c=1.0;            // initialize with a double precision value
dd_real d=0.1L;           // initialize with a long double value
dd_real x[2]={dd_real(1.0), dd_real(2.0)};  // initialize an array
```

3.5 Assignment

The user can use assignment statement with variables of type dd_real. In addition it is also allowed to use assignment between dd_real and double. The whole set of assignment features is listed in Table 3.5.1

When the right hand side is of neither type dd_real nor type double, it is converted to type double and then assigned to left hand size. When a long double value is assigned to a dd_real variable, the user need to use a conversion function. See example.

Table 3.5.1 Assignment

	Type of left hand side	Type of right hand side
Assignment (=)	dd_real	dd_real
	dd_real	double

Example1: Assignment

```
dd_real a,b;
a=b;
```

Example2: Assignment of integer or double to dd_real

```
dd_real a;
int n;
double d;
a=n;              // This is equivalent to a=(double)n;
a=10;
a=d;
a=1.0e-5;         // Be careful about precision in assignment of double
```

Example3: When user want to assign a long double value to dd_real variable, use the conversion function.

```
dd_real a;
long double q;
a=dd_real(q);
```

Example4: When the left hand side isn't of type dd_real, use the conversion function.

```
dd_real a;
int n;
double d;
long double q;
d=to_double(a);
```

```
n=to_int(a);
q=to_long_double(a);
```

3.6 Operations

Four basic arithmetic operations +, –, *, /, assignment operations +=, -=, *=, /= and unary negation operation – are provided. The whole set of arithmetic operations available is listed in Table 3.6.1 through Table 3.6.2

Table 3.6.1 Basic arithmetic operations

Operation	Type of left hand side	Type of right hand side	Type of result
Addition, Subtraction, Multiplication, Division	dd_real	dd_real	dd_real
	double	dd_real	dd_real
	dd_real	double	dd_real

Table 3.6.2 Assignment operations

Operation	Type of right hand side	Type of result
Addition(+=), Subtraction(-=), Multiplication(*=), Division(/=)	dd_real	dd_real
	double	dd_real

When values or variables of type other than dd_real are specified, they are converted to type double before calculation.

Example 1: Operations between two of dd_real

```
dd_real a,b,c,d,e;
c=a+b;
c=a-b;
c=a*b;
c=a/b;
c=a*b+d/e;
c+=a;
c-=a;
c*=a;
c/=a;
c=-a;       // unary negation
```

Example 2: For operations with dd_real, variables or constants of type double or of type int can appear on the left or right hand side

```
dd_real a,b,c;
double d;
c=d+a;
c=a-123.0;
c=2.0*b;
```

```
c=d/b;
```

Example 3: For operations with long double

```
dd_real x,y;
long double q;
y=x+dd_real(q);  // convert q to type dd_real
```

3.7 Relational operations and equality operations

Relational operations and equality operations for type dd_real are provided as Table 3.7.1 shows.

Table 3.7.1 Relational operations and equality operations

Operation	Type of left hand side	Type of right hand side	Type of result
Relational operations (> , >= , < , <=)	dd_real	dd_real	bool
Equality operations (==, !=)	double	dd_real	bool
	dd_real	double	bool

Example1:Relational operations

```
dd_real a,b,c;
if(a>b) {
    c=0.0;
}
if(a>=b && (c!=0.0 || c!=1.0)) {
    c=0.0;
}
```

3.8 Numeric functions

3.8.1 dd_real

Usage format

y=dd_real(x)

x either dd_real, int, double or long double

y dd_real

Function

This function converts x to dd_real. This is a constructor and can be used as a type conversion function.

Example

```
dd_real y;
dd_real z;
y=dd_real(1.0);
y=dd_real(123);
y=dd_real(1.23L);
y=dd_real(z);
```

3.8.2 to_int

Usage format

n=to_int(x)

x	dd_real

n	int

Function

This function converts dd_real to integer. If |x|<1.0 the result is 0. Otherwise, the result y is a maximum integer whose absolute value is less than or equal to |x| and the sign of y retains that of x.

Example
```
dd_real x;
int n;
n=to_int(x)+10;
```

3.8.3 to_double

Usage format

y=to_double(x)

x	dd_real

y	double

Function

This function converts dd_real to double.

Example
```
dd_real x;
double d;
d=to_double(x)+1.5;
```

3.8.4 to_long_double

Usage format

y=to_long_double(x)

x	dd_real

y	long double

Function

This function converts dd_real to long double

Example
```
dd_real x;
long double d;
d=to_long_double(x);
```

3.8.5 fabs, abs

Usage format

y=fabs(x)
y=abs(x)

x	dd_real

y	dd_real

Function

This function returns the absolute value of x.

Example
```
dd_real x,y;
y=fabs(x);
```

```
y=abs(x);
```

3.8.6 max

Usage format

#include <algorithm>

z=std::max(x,y)

x	dd_real
y	dd_real
z	dd_real

Function

This function returns the maximum value. This is a function of C++ standard library.

Example

```
#include <algorithm>
dd_real x,y,z;
z=std::max(x,y);
```

3.8.7 min

Usage format

#include <algorithm>

z=std::min(x,y)

x	dd_real
y	dd_real
z	dd_real

Function

This function returns the minimum value. This is a function of C++ standard library.

Example

```
#include <algorithm>
dd_real x,y,z;
z=std::min(x,y);
```

3.8.8 aint

Usage format

y=aint(x)

x	dd_real
y	dd_real

Function

This function truncates the fraction part of x. If |x|<1.0 the result is 0.0. Otherwise, the result y is a maximum integer whose absolute value is less than or equal to |x| and the sign of y retains that of x.

Example

```
dd_real x,y;
x=5.678;
y=aint(x);      // 5.0 is assigned to y
x=-5.678;
y=aint(x);      // -5.0 is assigned to y
```

3.8.9 nint

Usage format

y=nint(x)

x	dd_real
y	dd_real

Function

This function rounds off the fraction part of x in the nearest mode. If x is 0.0 or positive number, the result becomes aint(x+0.5). If x is negative, the result becomes aint(x-0.5).

Example

```
dd_real x,y;
x=5.678;
y=nint(x);      //  6.0 is assigned to y
```

3.9 Mathematical functions

3.9.1 sqrt

Usage format

y=sqrt(x)

x	dd_real
y	dd_real

Function

This function returns the square root of x. x>=0 must be satisfied.

Example

```
dd_real x,y;
y=sqrt(x);
```

3.9.2 exp

Usage format

y=exp(x)

x	dd_real
y	dd_real

Function

This function returns the exponential of x. x<709.0 must be satisfied.

Example

```
dd_real x,y;
y=exp(x);
```

3.9.3 log

Usage format

y=log(x)

x	dd_real
y	dd_real

Function

This function returns the natural logarithm of x. x>0 must be satisfied.

Example
```
dd_real x,y;
y=log(x);
```

3.9.4 sin

Usage format

y=sin(x)

x	dd_real
y	dd_real

Function

This function returns the sine of x. |x|<1.4e19 must be satisfied.

Example
```
dd_real x,y;
y=sin(x);
```

3.9.5 cos

Usage format

y=cos(x)

x	dd_real
y	dd_real

Function

This function returns the cosine of x. |x|<1.4e19 must be satisfied.

Example
```
dd_real x,y;
y=cos(x);
```

3.9.6 sincos

Usage format

sincos(x,s,c)

x	input	dd_real
s	output	dd_real
c	output	dd_real

Function

This routine returns the sine and cosine of x to s and c, respectively. When both sin(x) and cos(x) for the same x are required in applications this routine is useful since it is faster than using the functions sin and cos separately. |x|<1.4e19 must be satisfied.

Example
```
dd_real x,s,c;
sincos(x,s,c);
```

3.10 Multi-vector operation routines

Multi-operation routines perform two operations by a single call so that arithmetic units can work efficiently. Similarly, vector operation routines perform operations efficiently on an array of type dd_real.

Thread-parallel vector operation routines are also provided which perform OpenMP type of parallel operations within the routines using multiple cores. Note that these routines are thread safe like other fast_dd functions or routines so that the user can call these routines from inside or outside OpenMP parallel regions.

3.10.1 m2_add_dd

Usage format

m2_add_dd(a,b,c,x,y,z)

a	input	dd_real
b	input	dd_real
c	output	dd_real
x	input	dd_real
y	input	dd_real
z	output	dd_real

Function

This function computes c=a+b, z=x+y.

Example

```
dd_real a,b,c,x,y,z;
m2_add_dd(a,b,c,x,y,z);
```

3.10.2 m2_sum_dd

Usage format

m2_sum_dd(a,b,c)

a	input	dd_real
b	input	dd_real
c	input/output	dd_real

Function

This function computes c=c+a+b.

Example

```
dd_real a,b,c;
m2_sum_dd(a,b,c);
```

3.10.3 m2_sub_dd

Usage format

m2_sub_dd(a,b,c,x,y,z)

a	input	dd_real
b	input	dd_real
c	output	dd_real
x	input	dd_real
y	input	dd_real

z	output	dd_real

Function

This function computes c=a-b, z=x-y.

Example

```
dd_real a,b,c,x,y,z;
m2_sub_dd(a,b,c,x,y,z);
```

3.10.4 m2_mul_dd

Usage format

m2_mul_dd(a,b,c,x,y,z)

a	input	either dd_real or double
b	input	dd_real
c	output	dd_real
x	input	either dd_real or double
y	input	dd_real
z	output	dd_real

Function

This function computes c=a*b, z=x*y. Data type of a and x must be same.

Example

```
dd_real a,b,c,x,y,z;
double p,q;
m2_mul_dd(a,b,c,x,y,z);
m2_mul_dd(p,b,c,q,y,z);
```

3.10.5 v_add_dd

Usage format

v_add_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either int or long int. the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This function computes c=a+b for arrays a, b and c.

Example

```
dd_real a[100],b[100],c[100];
v_add_dd(a,b,100,c);
```

3.10.6 v_sub_dd

Usage format

v_sub_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either int or long int. the number of elements of array a, b and c.

c	output	one-dimensional array of type dd_real and size n.

Function

This function computes c=a-b for arrays a, b, and c.

Example
```
dd_real a[100],b[100],c[100];
v_sub_dd(a,b,100,c);
```

3.10.7 v_mul_dd

Usage format

v_mul_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either int or long int. the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This function computes c=a*b for arrays a, b, and c.

Example
```
dd_real a[100],b[100],c[100];
v_mul_dd(a,b,100,c);
```

3.10.8 vm_add_dd

Usage format

vm_add_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either int or long int. the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This function is a thread-parallel routine to compute c=a+b for arrays a, b and c.

Example
```
dd_real a[100],b[100],c[100];
vm_add_dd(a,b,100,c);
```

3.10.9 vm_sub_dd

Usage format

vm_sub_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either int or long int. the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This function is a thread-parallel routine to compute c=a-b for arrays a, b, and c.

Example

```
dd_real a[100],b[100],c[100];
vm_sub_dd(a,b,100,c);
```

3.10.10 vm_mul_dd

Usage format

vm_mul_dd(a,b,n,c)

a	input	one-dimensional array of type dd_real and size n.
b	input	one-dimensional array of type dd_real and size n.
n	input	either int or long int. the number of elements of array a, b and c.
c	output	one-dimensional array of type dd_real and size n.

Function

This function is a thread-parallel routine to compute c=a*b for arrays a, b, and c.

Example

```
dd_real a[100],b[100],c[100];
vm_mul_dd(a,b,100,c);
```

3.11 Error handling

Some of fast_dd routines checks input arguments for consistency. See “4 Error Messages” for the details. If the routines detect some errors they set NaN as a value and continue computation. Such error handling can be changed by calling the following routine.

Usage format

fast_dd_errlvl(n)

n	input	int. Specify value from the following selection, which determines action for fast_dd to take after calling this routine. 0: issue no messages and continue computation 10: issue error messages and continue computation 90:issue error messages and terminate computation If n is set to different value from any of the above values, the routine does not change error handling.

Function

This routine changes the way of error handling in fast_dd.

4. Error Messages

This chapter describes error messages the fast_dd issues. The user can control whether or not the fast_dd should issue error messages by calling fast_dd_errlvl.

fast_dd-error : 4001 : In exp(x), x>=709.0 : x= ***value***

- Explanation
  The argument x of exp was x>=709.0.
- Explanation about value
  value : shows the value the user has given
- User's action
  correct the argument x to satisfy x<709.0.

fast_dd-error : 4101 : In log(x), x<=0.0 : x= ***value***

- Explanation
  The argument x of log was x<=0.0.
- Explanation about value
  value : shows the value the user has given
- User's action
  correct the argument x to satisfy x>=0.0.

fast_dd-error : 4201 : In sin(x), abs(x)>=1.4e19 : x= ***value***

- Explanation
  The argument x of sin was |x|>= 1.4e19.
- Explanation about value
  value : shows the value the user has given
- User's action
  correct the argument x to satisfy |x|<1.4e19.

fast_dd-error : 4301 : In cos(x), abs(x)>=1.4e19 : x= ***value***

- Explanation
  The argument x of cos was |x|>= 1.4e19.
- Explanation about value
  value : shows the value the user has given
- User's action
  correct the argument x to satisfy |x|<1.4e19.

fast_dd-error : 4401 : In sincos(x,s,c), abs(x)>=1.4e19 : x= ***value***

- Explanation
  The argument x of sincos was |x|>= 1.4e19.
- Explanation about value
  value : shows the value the user has given

- User’s action
  correct the argument x to satisfy |x|<1.4e19.

fast_dd-error : 4501 : In sqrt(x), x<0.0 : x= *value*

- Explanation
  The argument x of sqrt was x<0.0.
- Explanation about value
  value : shows the value the user has given
- User’s action
  correct the argument x to satisfy x>=0.0.

Index