Software Development Kit for Multicore Acceleration Version 3.1

3D Fast Fourier Transform Library Programmer's Guide and API Reference

Note

Before using this information and the product it supports, read the information in Notices on page 23.

Edition notice

This edition applies to version 3, release 1, modification 0 of the IBM Software Development Kit for Multicore Acceleration (Product number 5724-S84) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this publication

This publication describes in detail how to configure the Fast Fourier Transform library (FFT) and how to program applications using it on the IBM Software Development Kit for Multicore Acceleration (SDK). It contains detailed reference information about the APIs for the library as well as sample applications showing usage of these APIs.

Who should use this book

The target audience for this document is application programmers using the SDK. You are expected to have a basic understanding of programming on the Cell Broadband Engine(TM) (Cell/B.E.) platform and common terminology used with the Cell/B.E. platform.

Typographical conventions

The following table explains the typographical conventions used in this document.

Table 1. Typographical conventions

Typeface	Indicates	Example
Bold	Lowercase commands, library functions.	**void sscal_spu (float *sx, float sa, int n)**
Italics	Parameters or variables whose actual names or values are to be supplied by the user. Italics are also used to introduce new terms.	The following example shows how a test program, *test_name* can be run
`Monospace`	Examples of program code or command strings.	`int main()`

Related information

For a list of SDK documentation, see Appendix A. Related documentation.

Part 1. Library Introduction

The Three-Dimensional Fast Fourier Transform (3D FFT) library is a collection of C routines for computing the 3D discrete Fourier transform (DFT) and its inverse.

The 3D FFT library in the IBM Software Development Kit for Multicore Acceleration (SDK) supports single precision and double precision input data formats, hereafter referred to as SP and DP. All SP and DP 3D FFT routines are supported on the Power Processing Element (PPE), and shipped in a 64 bit PPE library.

The current implementation is a prototype and provides the following capabilities:

1. complex to complex transform and inverse transform
2. input data restricted to "tiled" format, explained below
3. sizes of powers of 2 in each dimension of the input data
4. a maximum size of 2048 for the x and y dimensions, and 512 for the z dimension for single precision data
5. a maximum size of 1024 for the x, y and z dimensions for double precision data
6. fft computation is done in-place, overwriting the input data

Part 2. Installing and configuring the library

Installation and configuration of the 3D FFT library occurs during installation of the Software Development Kit for Multicore Acceleration. For details on installing the SDK, see the "Installing the SDK" section of the Software Development Kit 3.1 Installation Guide available at the Cell Broadband Engine Architecture Resource Center developerWorks® Web site:

`http://www-128.ibm.com/developerworks/power/cell`

Part 3 Programming

The following sections provide information about programming with the 3D FFT library:

- Chapter 1, "Basic structure of the 3D FFT library"
- Chapter 2, "Using the 3D FFT library"
- Chapter 3, "Performance tuning a 3D FFT application"

Chapter 1. Basic structure of 3D FFT library

The following two tables detail the location of the various files installed with 3D FFT package.

3DFFT library contents (x86)

Platform	X86 or x86_64 (Development)
PPE 64-bit library	/opt/cell/sysroot/usr/lib64/libfft3d.so.3.1
Library header file	/opt/cell/sysroot/usr/include/libfft3d.h
Example code	/opt/cell/sdk/src/libfft3d-examples-source.tar

3DFFT library contents (Cell/B.E)

Platform	Cell/B.E. or Power Host (Development or execution)
PPE 64-bit library	/usr/lib64/libfft3d.so.3.1
Library header file	/usr/include/libfft3d.h
Example code	/opt/cell/sdk/src/libfft3d-examples-source.tar

The following table describes the key file components of the FFT library.

File description

File	Description
libfft3d.h	Contains the C function interface of 3D FFT PPE library
libfft3d.so	Shared 3D FFT library for Cell/B.E
libfft3d-examples-source.tar	Contains an example that demonstrates how to use the 3D FFT library

Chapter 2. Using the FFT library

The 3D FFT prototype provides a sample application and Makefile that can be used as an example of how to use the library. The sample is provided in a source tar file that is installed in the location described in chapter 1.

To use the library, do the following:

1. Create the 3D input array of data to be processed in tiled format.
2. Initialize the library with either fft_3d_sp_initialize() or fft_3d_dp_initialize().
3. Perform the 3D FFT computation with a call to fft_3d_sp_perform() or fft_3d_dp_perform()
4. Terminate the library with a call to fft_3d_sp_terminate() or fft_3d_dp_terminate().

These steps are demonstrated in the sample application code.

The sample Makefile shows how to build an application that uses the 3D FFT library.

The input data for a 3D FFT is in the form of a rectangular parallelepiped that is described by the sizes in the x, y, and z directions. The maximum number of elements for single precision complex numbers (8 bytes each) is 2^{31} with a maximum size of 2048 for the x and y dimensions, and 512 for the z dimension. The maximum number of elements for double precision complex numbers (16 bytes each) is 2^{30} with a maximum size of 1024 for each of the x, y and z dimensions.

For real data, the natural format in computer memory would be that of a standard 3D array. But FFTs, in general, operate on complex data, and the structure of the input format has an impact on the overall performance of the application.

Several structures are possible when storing a 3D complex array in memory. One natural format is a list of ordered pairs, each pair containing first the real component, and second the imaginary component of the number (R0, I0, R1, I1, R2, I2, etc.), as illustrated in the Figure 1 below.

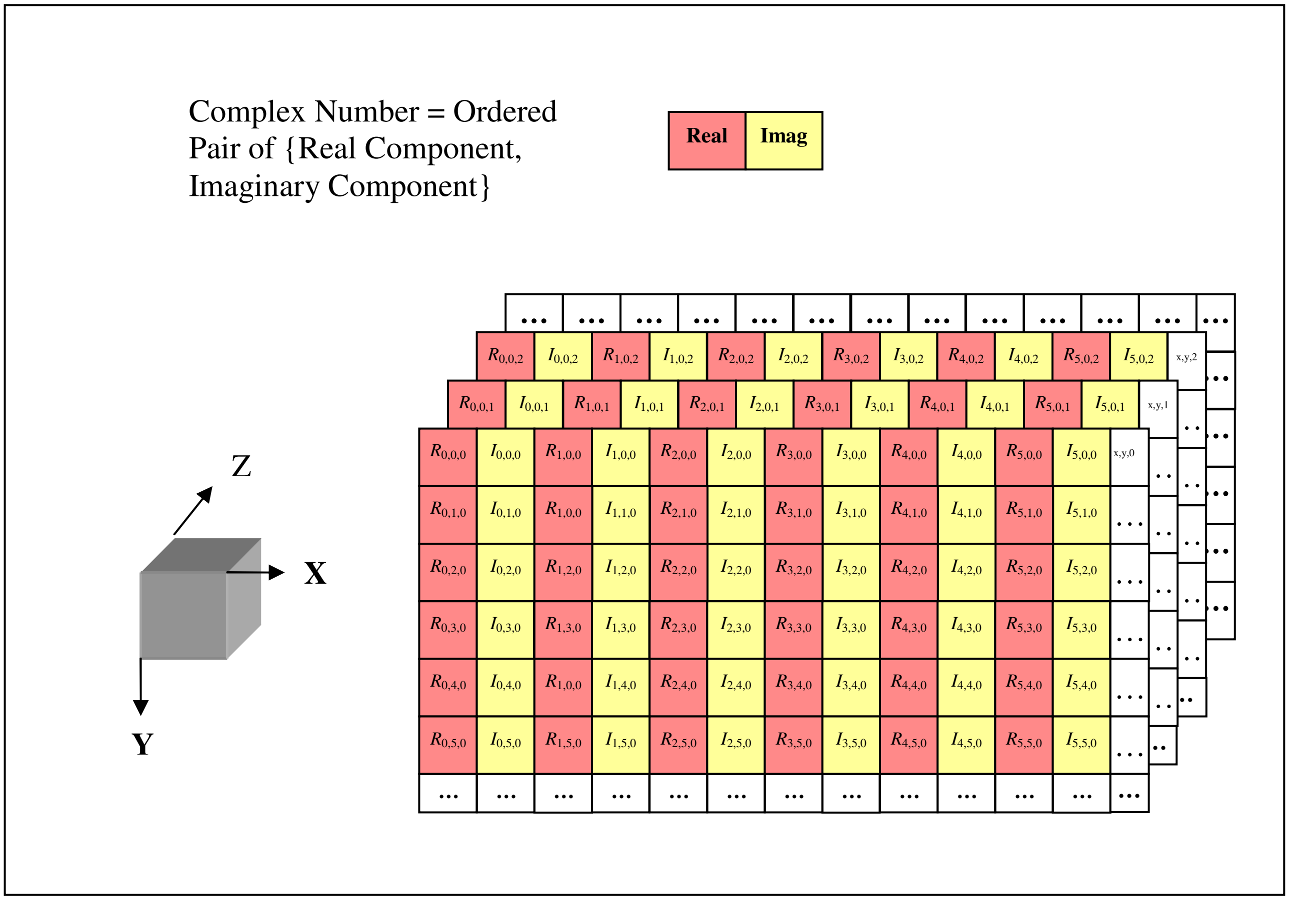

Figure 1: One Natural Way to Represent Complex Data in an Array

Single Precision Data

For single precision SIMD calculations, ideally the data should be ordered in groups of four floats to map directly to the underlying vector data type. Also, FFT calculations operate on complex numbers, and an additional useful requirement is that the real and imaginary values be “close”, or structured in groups of eight floats (4 real, 4 imaginary), again for SIMD reasons (a vector of real, followed by a vector of imaginary). That just leaves the choice of the number of these 4 real, 4 imaginary pairs to be included in a tile. Also a size of 128 bytes is significant for the underlying architecture since it has both DMA and memory bank performance efficiencies associated with it. Therefore the tile size of 4x4x1 complex elements is chosen, because this gives a total tile size of 128 bytes. It is blocked to: real, real, real, real, imaginary, imaginary, imaginary, imaginary, real, real, real, real, etc. The tile structure is depicted in Figure 2.

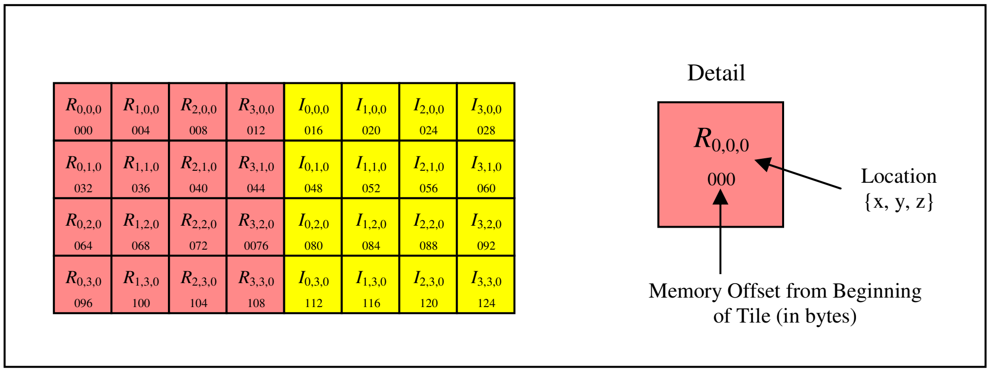

Figure 2: A Single Precision Tile

One final performance consideration is needed before the overall single precision 3D data structure is complete. Memory architecture is such that each consecutive cache line size (128 bytes) of data is mapped sequentially to a different memory bank. There are 16 total memory banks so that addresses that differ in multiples of 2048 (=16x128) map to the same memory bank. Now consider that the FFT calculations operate on rows and columns of tiled data. For sequential tile access in the X direction the memory bank access isn’t an issue since the tiles are drawn from sequential banks naturally. But for tiles in the Y direction there is a tendency for each tile to be aligned in subsets of the memory banks due to the inherent power of 2 nature of the data structure and input size requirements. This situation is clearly undesirable since DMA requests for a row of tiles in the Y direction all target the same subset of memory banks. So a memory bank “bump” pad of 128 bytes is added at the end of each row of tiles to shift consecutive Y tiles out of the same memory bank alignment. Note that there is no pad added to the last row of tiles in each z plane. This is to force the start of consecutive z planes into different memory banks.

When all of these considerations are combined the resulting picture of a single XY plane viewed as tiles is shown in figure 3. The Z direction is simply a stack of these XY planes.

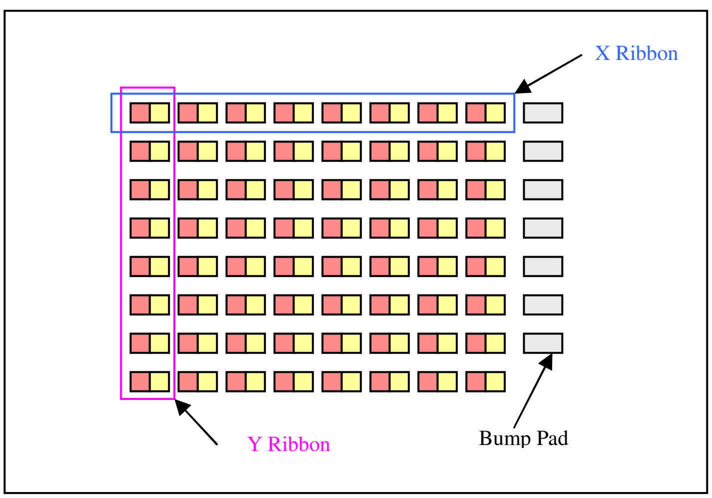

Figure 3: XY Plane Structured as Single Precision Tiles

Double Precision Data

For double precision data the memory bank and alignment described above for the single precision tiled format also applies. The difference is that the underlying vector format is two double precision values versus 4 floats. This leads to a tile that is composed of a block of double precision values of size 2x2x2, and it makes the single and double precision tile sizes the same, that is, 128 bytes. It is blocked to: real (double), real (double), imaginary (double), imaginary (double), real (double), real (double), etc. This tile structure is depicted in Figure 4.

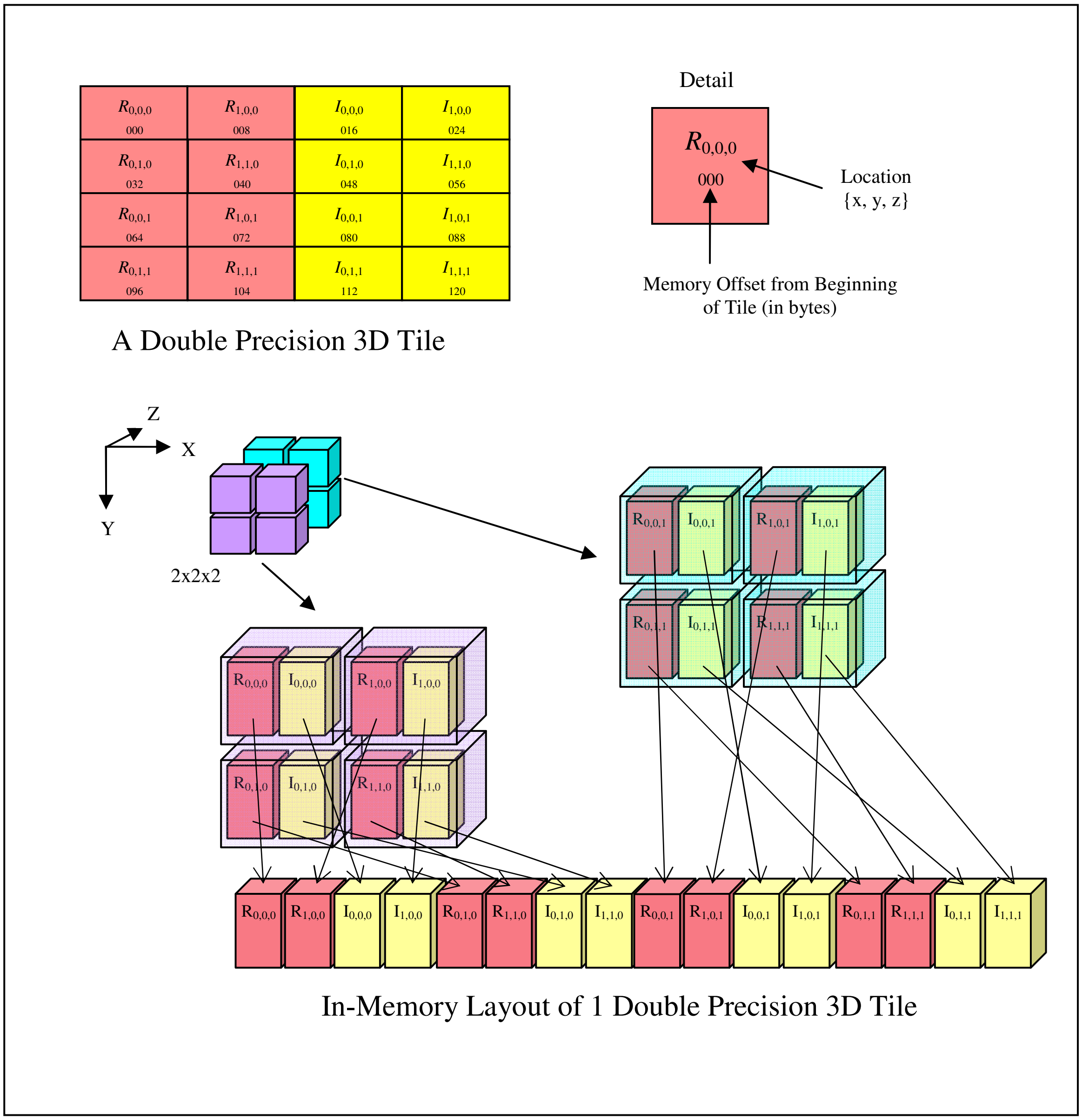

Figure 4: Double Precision Tile Format

Chapter 3. Tuning the 3D FFT Library for Performance

The Cell Broadband Engine provides additional features to more efficiently use the available resources and potentially achieve better performance.

Here are some points to consider when optimizing a 3D FFT program:

1. It is advisable to use huge pages for storing the input/output matrix. This will reduce page faults which may affect performance.
2. Make the input data array 128-byte aligned. Memory access is more efficient when data is 128-byte aligned.
3. If you want to use more than 8 SPEs, use NUMA APIs to interleave the memory of the data on different nodes.

Part 4. 3D FFT API reference

The 3D FFT library provides a PPE interface only. See Chapter 4 for more details.

Chapter 4. PPE APIs

The 3D FFT library provides the following constants to define the minimum and maximum sizes allowed for the X, Y, and Z dimensions for double precision and single precision FFT data arrays.

```
#define FFT_3D_MIN_DIM_X_DP  16
#define FFT_3D_MIN_DIM_Y_DP  16
#define FFT_3D_MIN_DIM_Z_DP  16

#define FFT_3D_MAX_DIM_X_DP  1024
#define FFT_3D_MAX_DIM_Y_DP  1024
#define FFT_3D_MAX_DIM_Z_DP  1024

#define FFT_3D_MIN_DIM_X_SP  16
#define FFT_3D_MIN_DIM_Y_SP  16
#define FFT_3D_MIN_DIM_Z_SP  16

#define FFT_3D_MAX_DIM_X_SP  2048
#define FFT_3D_MAX_DIM_Y_SP  2048
#define FFT_3D_MAX_DIM_Z_SP  512
```

The 3D FFT library provides two sets of FFT APIs. One is for single precision data and the other is for double precision.

fft_3d_dp_initialize

Synopsis

int fft_3d_dp_initialize(fft_3d_handle_t *handle, unsigned int nspus)

Description

This function sets up the environment for performing double precision 3D FFTs, which are rectangular parallelepipeds with the dimensions each being powers of two.

Parameter

handle	is a handle of type *fft_3d_handle_t.
nspus	is the number of SPUs to be used for the problem.

Return Value

FFT_RC_SUCCESS	Success
FFT_RC_NO_SPUS	Insufficient spus are available
FFT_RC_BAD_DIMENSION	Invalid size of array
FFT_RC_FAILED	Generic internal errors

fft_3d_sp_initialize

Synopsis

int fft_3d_dp_initialize(fft_3d_handle_t *handle, unsigned int nspus)

Description

This function sets up the environment for performing single precision 3D FFTs, which are rectangular parallelepipeds with the dimensions each being powers of two.

Parameter

handle	is a handle of type *fft_3d_handle_t.
nspus	is the number of SPUs to be used for the problem.

Return Value

FFT_RC_SUCCESS	Success
FFT_RC_NO_SPUS	Insufficient spus are available
FFT_RC_BAD_DIMENSION	Invalid size of array
FFT_RC_FAILED	Generic internal errors

fft_3d_dp_perform

Synopsis

int fft_3d_dp_perform(fft_3d_handle_t handle, void *src_addr, unsigned int dim_x, unsigned int dim_y, unsigned int dim_z, unsigned int inverse_flag, unsigned int format_flag)

Description

This function accomplishes a double precision 3D complex to complex FFT. The computation is done in-place. The maximum number of elements supported is 2^{30} with any given dimension up to 1024 elements. This allows a maximum shape of 1024x1024x1024.

Parameter

handle	is a handle of type *fft_3d_handle_t created in the initialization step.
src_addr	is an array of addresses of **n_arrays** input arrays. The array contains the data for a particular 3D FFT to be performed.
dim_x	is the size of the X component of FFT array. The minimum supported size is 16. The maximum supported size is 1024. The value must be a power of two.
dim_y	is the size of the Y component of FFT array. The minimum supported size is 16. The maximum supported size is 1024. The value must be a power of two.
dim_z	is the size of the Z component of FFT array. The minimum supported size is 16. The maximum supported size is 1024. The value must be a power of two.
inverse_flag	is the inverse flag. Possible values are: 0 Forward FFT Non-zero Inverse FFT
format_flag	is a bit flag to indicate the input data format. Only one possible value is: FFT_TILED Tiled data format.

Return Value

FFT_RC_SUCCESS	Success
FFT_RC_BAD_PARM	Invalid input parameter.
FFT_RC_FAILED	Generic internal errors.

fft_3d_sp_perform

Synopsis

int fft_3d_sp_perform(fft_3d_handle_t handle, void *src_addr, unsigned int dim_x, unsigned int dim_y, unsigned int dim_z, unsigned int inverse_flag, unsigned int format_flag)

Description

This function accomplishes a single precision 3D complex to complex FFT. The computation is done in-place. The maximum number of elements supported is 2^{31} where the X and Y dimensions can be up to 2048 elements, and the Z dimension can be up to 512 elements. This allows a maximum shape of 2048x2048x512.

Parameter

handle	is a handle of type *fft_3d_handle_t created in the initialization step.
src_addr	is an array of addresses of **n_arrays** input arrays. The array contains the data for a particular 3D FFT to be performed.
dim_x	is the size of the X component of FFT array. The minimum supported size is 16. The maximum supported size is 2048. The value must be a power of two.
dim_y	is the size of the Y component of FFT array. The minimum supported size is 16. The maximum supported size is 2048. The value must be a power of two.
dim_z	is the size of the Z component of FFT array. The minimum supported size is 16. The maximum supported size is 512. The value must be a power of two.
inverse_flag	is the inverse flag. Possible values are: 0 Forward FFT Non-zero Inverse FFT
format_flag	is a bit flag to indicate the input data format. Only one possible value is: FFT_TILED Tiled data format.

Return Value

FFT_RC_SUCCESS	Success
FFT_RC_BAD_PARM	Invalid input parameter.
FFT_RC_FAILED	Generic internal errors.

fft_3d_dp_terminate

Synopsis

int fft_3d_dp_terminate(fft_3d_handle_t handle)

Description

This function terminates the environment for performing double precision 3D FFTs.

Parameter

handle is a handle of type fft_3d_handle_t created in the initialization step.

Return Value

FFT_RC_SUCCESS	Success
FFT_RC_BAD_PARM	Invalid input parameter.
FFT_RC_FAILED	Generic internal errors.

fft_3d_sp_terminate

Synopsis

int fft_3d_sp_terminate(*fft_3d_handle_t handle*)

Description

This function terminates the environment for performing single precision 3D FFTs.

Parameter

handle is a handle of type fft_3d_handle_t created in the initialization step.

Return Value

FFT_RC_SUCCESS	Success
FFT_RC_BAD_PARM	Invalid input parameter.
FFT_RC_FAILED	Generic internal errors.

Part 5. Appendixes

Appendix A. Related documentation

This topic helps you find related information.

Document location

Links to documentation for the SDK are provided on the IBM Information Center site located at:

http://publib.boulder.ibm.com/infocenter/systems/topic/eiccu/eiccukickoff.html

The following documents are available, organized by category:

Architecture

- *Cell Broadband Engine™ Architecture*
- *Cell Broadband Engine Registers*
- *SPU Instruction Set Architecture*

Standards

- *C/C++ Language Extensions for Cell Broadband Engine Architecture*
- *Cell Broadband Engine Linux® Reference Implementation Application Binary Interface Specification*
- *SIMD Math Library Specification for Cell Broadband Engine Architecture*
- *SPU Application Binary Interface Specification*
- *SPU Assembly Language Specification*

Programming

- *Cell Broadband Engine Programmer's Guide*
- *Cell Broadband Engine Programming Handbook*
- *Cell Broadband Engine Programming Tutorial*

Library

- *Accelerated Library Framework for Cell Broadband Engine Programmer's Guide and API Reference*
- *Basic Linear Algebra Subprograms Programmer's Guide and API Reference*
- *Data Communication and Synchronization for Cell Broadband Engine Programmer's Guide and API Reference*
- *Example Library API Reference*
- *Fast Fourier Transform Library Programmer's Guide and API Reference*
- *LAPACK (Linear Algebra Package) Programmer's Guide and API Reference*
- *Mathematical Acceleration Subsystem (MASS)*

- *Monte Carlo Library Programmer's Guide and API Reference*
- *SDK 3.0 SIMD Math Library API Reference*
- *SPE Runtime Management Library*
- *SPE Runtime Management Library Version 1 to Version 2 Migration Guide*
- *SPU Runtime Extensions Library Programmer's Guide and API Reference*

Installation

- *SDK for Multicore Acceleration Version 3.1 Installation Guide*

Tools

- *Getting Started - XL C/C++ Advanced Edition for Linux*
- *Compiler Reference - XL C/C++ Advanced Edition for Linux*
- *Language Reference - XL C/C++ Advanced Edition for Linux*
- *Programming Guide - XL C/C++ Advanced Edition for Linux*
- *Installation Guide - XL C/C++ Advanced Edition for Linux*
- *Getting Started - XL Fortran Advanced Edition for Linux*
- *Compiler Reference - XL Fortran Advanced Edition for Linux*
- *Language Reference - XL Fortran Advanced Edition for Linux*
- *Optimization and Programming Guide - XL Fortran Advanced Edition for Linux*
- *Installation Guide - XL Fortran Advanced Edition for Linux*
- *Using the single-source compiler*
- *Performance Analysis with the IBM Full-System Simulator*
- *IBM Full-System Simulator User's Guide*
- *IBM Visual Performance Analyzer User's Guide*

IBM PowerPC(R) Base

- *IBM PowerPC Architecture(TM) Book*
    - *Book I: PowerPC User Instruction Set Architecture*
    - *Book II: PowerPC Virtual Environment Architecture*
    - *Book III: PowerPC Operating Environment Architecture*
- *IBM PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual*

Appendix B. Accessibility features

Accessibility features help users who have a physical disability, such as restricted mobility or limited vision, to use information technology products successfully.

The following list includes the major accessibility features:

- Keyboard-only operation
- Interfaces that are commonly used by screen readers
- Keys that are tactilely discernible and do not activate just by touching them
- Industry-standard devices for ports and connectors
- The attachment of alternative input and output devices

IBM and accessibility

See the IBM Accessibility Center at http://www.ibm.com/able/ for more information about the commitment that IBM has to accessibility.

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing 2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM Corporation
Software Interoperability Coordinator, Department 49XA
3605 Highway 52 N
Rochester, MN 55901
U.S.A.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Program License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

All IBM prices shown are IBM's suggested retail prices, are current and are subject to change without notice. Dealer prices may vary.

This information is for planning purposes only. The information herein is subject to change before the products described become available.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

Trademarks

Terms and conditions

Permissions for the use of these publications is granted subject to the following terms and conditions.