Software Development Kit for Multicore Acceleration
Version 3.1

Fast Fourier Transform Library Programmer's Guide and API Reference

SC34-2562-00

Software Development Kit for Multicore Acceleration
Version 3.1

Fast Fourier Transform Library Programmer's Guide and API Reference

SC34-2562-00

Note

Before using this information and the product it supports, read the information in "Notices" on page 39.

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™ (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,” on page 35.

How to send your comments

Your feedback is important in helping to provide the most accurate and highest quality information. If you have any comments about this publication, send your comments using IBM Resource Link™ at http://www.ibm.com/servers/resourcelink. Click **Feedback** on the navigation pane. Be sure to include the name of the book, the form number of the book, and the specific location of the text you are commenting on (for example, a page number or table number).

Part 1. Overview of the FFT library

The FFT library is a collection of C routines for computing discrete Fourier transform (DFT). The FFT is a widely used algorithm in science and engineering, with applications in almost every discipline.

This FFT library provides both one-dimensional and two-dimensional FFT routines. These routines are implemented in the C interface. Each FFT routine has three versions:

Data type	Short name
Complex to Complex	C2C
Complex to Real	C2R
Real to Complex	R2C

The FFT library in the SDK supports both single precision (SP) and double precision (DP). All SP and DP FFT routines are supported on the Power Processing Element (PPE). An SPE interface for single precision FFTs is also available.

Each of the different types of FFTs use APIs with the following naming pattern:

- **fft_init**: this routine is called to set up the environment for performing FFT
- **fft_perform**: this routine accomplishes FFT computation one or many FFTs
- **fft_terminate**: this routine is used to clean up the environment

Part 2. Installing and configuring the FFT library

The following section describes how to install and configure the FFT library.

The FFT library is installed and configured during the installation of the SDK. For details about how to install the SDK, see the "Installing the SDK" section of the *SDK Installation Guide* available at the Cell/B.E. Resource Center developerWorks Web site:

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

Part 3. Programming

These topics provide information about how to program with the FFT library.

Chapter 1. Basic structure of the FFT library

This topic describes the FFT library directory paths for each of the supported platform.

The following tables detail the location of the various files installed with the FFT package:

Table 2. FFT library contents (Hybrid)

Platform	X86 or x86_64 (Development)
PPE 32–bit library	/opt/cell/sysroot/usr/lib/libfft.so.3.1
PPE 64–bit library	/opt/cell/sysroot/usr/lib64/libfft.so.3.1
SPE library	/opt/cell/sysroot/usr/spu/lib/libfft.a
PPE/SPE header files	/opt/cell/sysroot/usr/include/libfft.h /opt/cell/sysroot/usr/spu/include/libfft_spu.h
Example code	/opt/cell/sdk/src/libfft-examples-source.tar

Table 3. FFT library contents (Cell/B.E.)

Platform	Cell/B.E. or Power Host (Development or execution including Simulator)
PPE 32–bit library	/usr/lib/libfft.so.3.1
PPE 64–bit library	/usr/lib64/libfft.so.3.1
SPE library	/usr/spu/lib/libfft.a
PPE/SPE header files	/usr/include/libfft.h /usr/spu/include/libfft_spu.h
Example code	/opt/cell/sdk/src/libfft-examples-source.tar

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

Table 4. File description

File	Description
`libfft.h`	Contains the C function interface of FFT on PPE and SPE
`libfft_spu.h`	Contains the C function interface of FFT on PPE and SPE
`libfft.a`	Contains the static FFT library which for SPE
`libfft.so`	Shared FFT library for Cell/B.E
`lifft-examples-source.tar`	Contains examples that demonstrate how to use the FFT library with the SDK

Chapter 2. Using the FFT library

These topics use programming examples to describe how to use the FFT library.

Programming examples

The following example application shows you how to use the FFT library. It invokes the PPE library APIs.

Building the examples

To build the examples listed in this document, follow this procedure:

1. Cut and paste the Makefile source from an online or PDF copy of this document into an editor and save it as Makefile.
2. Cut and paste the example source from an online or PDF copy of this document into an editor and save the file with a name such as `FFT1D_sample.c`.
3. Edit the Makefile to use the name of the source file that you chose in the previous step. If you did not use `fft_example.c` then substitute the name you chose into the lines that contain `FFT1D_sample` and `FFT1D_sample.a`.
4. Copy the Makefile and the example source file into your development source directory (for example into `/opt/sandbox/`).
5. From a shell prompt, type the following commands:

   ```
   $ cd /opt/sandbox
   $ export CELL_TOP=/opt/cell/sdk
   $ make
   ```

   Here is the Makefile:

   ```
   ########################################################################
   #   Target
   ########################################################################

   ifdef CREATE_LIB_PPU64
   PROGRAM_ppu64 = FFT1D_sample
   LDFLAGS += -R/usr/lib64
   else
   PROGRAM_ppu = FFT1D_sample
   LDFLAGS += -R/usr/lib
   endif

   ########################################################################
   #   Objects
   ########################################################################

   IMPORTS   +=-lfft -lm

   ########################################################################
   #   make.footer
   ########################################################################

   ifdef CELL_TOP
     include $(CELL_TOP)/buildutils/make.footer
   else
     include ../../../buildutils/make.footer
   endif
   ```

Example: PPE application

The following sample application shows you how to use FFT library on PPE. It invokes the FFT 1D routine.

```
/*

   FFT1D_sample.c - a simple routine to drive the fft library

*/

/* NOTE:
 * Computing a forward followed by a backward transform (or vice versa) will
 * result in the original data multiplied by the size of the transform
 * (the product of the dimensions).
 */

#include <stdio.h>
#include <stdlib.h>
#include <strings.h>
#include <time.h>
#include <sys/time.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/mman.h>
#include <errno.h>

#include <libfft.h>

#define HUGE_TLB
#define HUGE_PAGE_SIZE (16*1024*1024) /* 16MB */

// Forward declare.
int test_1D(int numberOfFfts, int sizeOfFfts, int spusToUse, int flavor,
int hugepage_flag);


// Main.
int main(int argc, char *argv[])
{
  if (argc < 6 || argc > 7)
  {
    fprintf(stderr, "Usage: %s <function type = c2c, r2c, c2r> <number of ffts>
    <spus to use> <size of each fft> <hugepage_flag>\n", argv[0]);
    exit(1);
  }

  int numberOfFfts = atoi(argv[2]);
  int spusToUse = atoi(argv[3]);
  int sizeOfFfts = atoi(argv[4]);
  int hugepage_flag = atoi(argv[5]);

  int flavor;
  if (strcasecmp(argv[1], "c2c") == 0)
  {
    flavor = FFT_TYPE_C2C;
  }
  else if (strcasecmp(argv[1], "r2c") == 0)
  {
    flavor = FFT_TYPE_R2C;
  }
  else if (strcasecmp(argv[1], "c2r") == 0)
  {
    flavor = FFT_TYPE_C2R;
  }
  else
  {
```

```
    fprintf(stderr, "Bad function type.\n");
    exit(1);
  }

  int res = test_1D(numberOfFfts, sizeOfFfts, spusToUse, flavor, hugepage_flag);

  return res;
} // end main

// Generate complex numbers as input.
void generateC2Cdata(int numberOfFfts, int sizeOfFfts, float **problems)
{
  int i;
  for (i=0; i<numberOfFfts; ++i)
  {
    int j;
    for (j=0; j<sizeOfFfts; ++j)
    {
      problems[i][j*2] = rand() % 1024; // Real
      problems[i][j*2+1] = rand() % 1024; // Imag
    }
  }
}

// Generate packed reals as input.
void generateR2Cdata(int numberOfFfts, int sizeOfFfts, float **problems)
{
  int i;
  for (i=0; i<numberOfFfts; ++i)
  {
    int j;
    for (j=0; j<sizeOfFfts; ++j)
    {
      problems[i][j] = rand() % 1024; // Real
    }
    // Don't care about the elements in the other half of the array,
    // since they don't get used.
  }
}

// Generate complex conjugates as input.
void generateC2Rdata(int numberOfFfts, int sizeOfFfts, float **problems)
{
  int i;
  for (i=0; i<numberOfFfts; ++i)
  {
    problems[i][0] = rand() % 1024; // Real
    problems[i][1] = 0; // Imag
    int j;
    for (j=1; j<(sizeOfFfts+1)/2; ++j)
    {
      problems[i][j*2] = rand() % 1024; // Real
      problems[i][j*2+1] = rand() % 1024; // Imag
      // Complex conjugate.
      problems[i][(sizeOfFfts-j)*2] = problems[i][j*2]; // Real
      problems[i][(sizeOfFfts-j)*2+1] = -problems[i][j*2+1]; // Imag
    }
    if (!(sizeOfFfts % 2)) // Size is even.
    {
      problems[i][sizeOfFfts] = rand() % 1024; // Real
      problems[i][sizeOfFfts+1] = 0; // Imag
    }
  }
}

// Allocate space for data and perform the FFT.
int test_1D(int numberOfFfts, int sizeOfFfts, int spusToUse, int flavor,
```

```
int hugepage_flag)
{
  // Allocate storage for input and output data.
  void *ptr;
  int i;
  posix_memalign(&ptr, 128, sizeof(float *) * numberOfFfts);
  float **input_data = (float **)ptr;
  posix_memalign(&ptr, 128, sizeof(float *) * numberOfFfts);
  float **output_data = (float **)ptr;
  unsigned int mallocLen = sizeof(float) * 2 * sizeOfFfts; // Real + imaginary
  mallocLen += mallocLen % 16;
  unsigned int dataLen = 0;
  if (hugepage_flag)
  /* Using hugepage can significantly reduce the TLB miss thus improve
  the performance */
  {
    int fmem;
    char *mem_file = "/huge/FFT1D_sample_mem.bin";
    if ((fmem = open(mem_file, O_CREAT | O_RDWR, 0755)) == -1)
    {
       fprintf(stderr, "ERROR: unable to open file %s (errno=%d).\n", mem_file, errno);
       return -1;
    }
    else
    {
      remove(mem_file);
      dataLen = numberOfFfts * mallocLen * 2;
      dataLen = ( dataLen + HUGE_PAGE_SIZE-1 ) & ~ (HUGE_PAGE_SIZE-1);
      ptr = mmap(0, dataLen, PROT_READ | PROT_WRITE, MAP_PRIVATE, fmem, 0);
      if (ptr == MAP_FAILED) {
         printf("ERROR: unable to mmap file %s (errno=%d).\n", mem_file, errno);
         close (fmem);
         return -1;
      }
      for (i=0; i<numberOfFfts; i++)
      {
        input_data[i] = (float *)ptr;
        /* If the input data are no longer used after computation,
         * the input and output data can share the same buffer.
         * In this in-place case, TLB miss can be further reduced.
         */
        ptr += mallocLen;
        output_data[i] = (float *)ptr;
        ptr += mallocLen;
      }
    }
  }
  else
  {
    for (i=0; i<numberOfFfts; ++i)
    {
      posix_memalign(&ptr, 128, mallocLen);
      input_data[i] = (float *)ptr;
      posix_memalign(&ptr, 128, mallocLen);
      output_data[i] = (float *)ptr;
    }
  }

  // Populate input data.
  srand(time(NULL));

  if (flavor == FFT_TYPE_C2C)
  {
    generateC2Cdata(numberOfFfts, sizeOfFfts, input_data);
  }
  else if (flavor == FFT_TYPE_R2C)
  {
```

```
    generateR2Cdata(numberOfFfts, sizeOfFfts, input_data);
  }
  else if (flavor == FFT_TYPE_C2R)
  {
    generateC2Rdata(numberOfFfts, sizeOfFfts, input_data);
  }

  // Start timer.
  struct timeval start, end;
  gettimeofday(&start, NULL);

  // Call library to process.
  fft_handle_t handle;
  int res = fft_1d_sp_initialize(&handle, spusToUse);
  if (res == FFT_RC_SUCCESS)
  {
    // Perform the transform.
    res = fft_1d_sp_perform(handle, numberOfFfts, sizeOfFfts, (void **)input_data,
    (void **)output_data, 0, flavor);
    if (res != FFT_RC_SUCCESS)
    {
      fprintf(stderr, "FFT failure: %d\n", res);
    }
    // Cleanup.
    fft_1d_sp_terminate(handle);
  }
  else
  {
    fprintf(stderr, "FFT failed to initialize: %d\n", res);
  }

  // Stop timer.
  gettimeofday(&end, NULL);
  unsigned int elapsed = ((end.tv_sec * 1000000) + end.tv_usec) -
  ((start.tv_sec * 1000000) + start.tv_usec);
  fprintf(stderr, "Calculation time took %u usec.\n", elapsed);

  // Cleanup.
  if (hugepage_flag)
  {
    munmap(input_data[0], dataLen);
  }
  else
  {
    for (i=0; i<numberOfFfts; ++i)
    {
      free(input_data[i]);
      free(output_data[i]);
    }
  }
  free(input_data);
  free(output_data);
  return res;
}
```

Chapter 3. Tuning the FFT library for performance

The FFT library provides additional features for customizing the FFT library. You can use these features to effectively use the available resources and potentially achieve higher performance.

The following are tips for optimizing the FFT program:

- It is advisable to use huge pages for storing the input/output matrix of computing FFT. This reduces page fault which may affect performance.
- Make the data 128-byte aligned. Memory access is more efficient when the data is 128-byte aligned.
- If you want to use more than 8 SPEs, use NUMA APIs to interleave the memory of the data on different nodes.
- The library performs better for 1D FFTs if the vector size is a multiple of 16 * $number\_of\_spu^2$.
- The library performs better for 2D FFTs if the matrix dimensions are powers of 2.
- For 2D FFTs, if input data follows the ’tiled’ format, performance is greatly improved.

Part 4. FFT API reference

The FFT library provides two sets of interfaces.

The interfaces are:

- Chapter 4, "PPE APIs," on page 19
- Chapter 5, "SPE APIs," on page 29

Chapter 4. PPE APIs

The FFT library provides three sets of FFT APIs.

One set is for FFT 1D single precision and the other two sets are for FFT 2D single and double precision.

The following APIs are provided:

FFT 1D single precision APIs

- “fft_1d_sp_initialize” on page 20
- “fft_1d_sp_perform” on page 21
- “fft_1d_sp_terminate” on page 22

FFT 2D single precision APIs

- “fft_2d_sp_initialize” on page 23
- “fft_2d_sp_perform” on page 24
- “fft_2d_sp_terminate” on page 25

FFT 2D double precision APIs

- “fft_2d_dp_initialize” on page 26
- “fft_2d_dp_perform” on page 27
- “fft_2d_dp_terminate” on page 28

fft_1d_sp_initialize

NAME

fft_1d_sp_initialize: Sets up the environment for performing 1D FFTs single precision computation.

SYNOPSIS

int fft_1d_sp_initialize(fft_handle_t *handle, unsigned int nspus);

Parameters

`handle [IN]`	Handle of type **fft_handle_t*.
`nspus [IN]`	Number of SPUs to be assigned to the problem.

DESCRIPTION

This function set up the environment for performing 1D FFTs single precision computation. Number of *spu* and *fft_handle* are input.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_NO_SPUS`	Insufficient SPUs are available
`FFT_RC_BAD_PARM`	Invalid input parameter
`FFT_RC_FAILED`	Generic internal errors

fft_1d_sp_perform

NAME

fft_1d_sp_perform: Accomplishes either one or many 1D complex to complex, complex to real, real to complex FFTs.

SYNOPSIS

int fft_1d_sp_perform(fft_handle_t handle, unsigned int n_arrays, unsigned int num_elems, void **src_addr, void **dst_addr, unsigned int inverse_flag, unsigned int format_flag);

Parameters

`handle[IN]`	Pointer to the handle created in the initialization step.
`n_arrays[IN]`	Number of FFTs that the customer wishes to compute with this call.
`num_elems[IN]`	Number of elements of each FFT.
`src_addr[IN]`	An array of addresses of *n_arrays* input arrays. Each such array contains the data for a particular 1D to be performed.
`dst_addr[IN]`	An array of addresses of *n_arrays* output arrays. Each such array is sized large enough to hold the 1D output of the FFT operation.
`inverse_flag[IN]`	Flag, which is a 0 for a forward FFT, and non-zero for an inverse FFT.
`format_flag[IN]`	Bit flag to set the FFT type. Possible values are: • FFT_TYPE_C2C Do 1D Complex to Complex. • FFT_TYPE_R2C Do 1D Real to Complex. Not implemented if *num_elems* is greater than 10000. The size of output buffer should be double size of input, but the only the first (num_elems/2)+1 complexes in the output buffer are valid. FFT 1D R2C is a forward FFT transform always and *inverse_flag* will be ignored here. • FFT_TYPE_C2R Do 1D Complex to Real FFT. Not implemented if *num_elems* is greater than 10000. FFT 1D C2R is a forward FFT transform always, and *inverse_flag* will be ignored here. Only FFT_TYPE_C2C can be used for an inverse transform.

DESCRIPTION

This function accomplishes either one or many 1D complex to complex, complex to real, real to complex FFTs. According to inverse flag, determine whether an FFT or IFFT is needed. *format_flag* determines the FFT type. FFT_TYPE_R2C and FFT_TYPE_C2R are not implemented if *num_elems* is greater than 10000.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_NOT_IMPLEMENTED`	Not implemented feature
`FFT_RC_BAD_PARM`	Invalid input parameter
`FFT_RC_FAILED`	Generic internal errors

fft_1d_sp_terminate

NAME

fft_1d_sp_terminate - Cleans up the environment after you have finished computing 1D FFTs.

SYNOPSIS

int fft_1d_sp_terminate(fft_handle_t handle);

Parameters

`handle [IN]`	The handle created in the initialization step

DESCRIPTION

This function cleans up the environment after you have finished computing 1D FFTs. *fft_handle* is input.

RETURN VALUE

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

fft_2d_sp_initialize

NAME

fft_2d_sp_initialize: Sets up the environment for performing 2D FFTs single precision computation.

SYNOPSIS

int fft_2d_sp_initialize(fft_2d_handle_t *handle, unsigned int nspus);

Parameters

`handle [IN]`	A handle to the FFT runtime code
`nspus [IN]`	Number of SPUs to be assigned to the problem

DESCRIPTION

This function set up the environment for performing 2D FFTs single precision computation. Number of *spu* and *fft_handle* are input.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_NO_SPUS`	Insufficient SPUs are available
`FFT_RC_BAD_PARM`	Invalid input parameter
`FFT_RC_FAILED`	Generic internal errors

fft_2d_sp_perform

NAME

fft_2d_sp_perform: Completes one 2D complex to complex single precision FFTs.

SYNOPSIS

int fft_2d_sp_perform(fft_2d_handle_t handle, unsigned int xdim, unsigned int ydim, void **src_addr, void **dst_addr, unsigned int inverse_flag, unsigned int format_flag);

Parameters

`handle[IN]`	Pointer to the handle created in the initialization step.
`xdim[IN]`	Size of 2D array in the x dimension.
`ydim[IN]`	Size of 2D array in the y dimension.
`src_addr[IN]`	Input data address.
`dst_addr[IN]`	Output data address.
`inverse_flag[IN]`	Flag, which is a 0 for a forward FFT, and non-zero for an inverse FFT.
`format_flag[IN]`	Bit flag to set the FFT type. Possible values are: • FFT_TILED: The format of all input and output matrices is TILED. • FFT_TYPE_C2C: Do 2D Complex to Complex FFT. • FFT_TYPE_R2C: Do 2D Real to Complex FFT. *xdim* or *ydim* must be to the power of 2. FFT2D R2C is a forward transform always and *inverse_flag* will be ignored here. The size of output buffer should be double size of input. • FFT_TYPE_C2R: Do 2D Complex to Real FFT. *xdim* or *ydim* must be to the power of 2. FFT2D C2R is an inverse transform always, and *inverse_flag* will be ignored here.

DESCRIPTION

This function completes one 2D complex to complex single precision FFTs. According to inverse flag, determine whether a FFT or IFFT is needed. *format_flag* determines FFT type. Also *format_flag* help determines whether input data is tiled format. If *xdim* or *ydim* are not to the power of 2, **fft_2d_sp_perform** does not support FFT_TYPE_C2R and FFT_TYPE_R2C.

RETURN VALUE

`FFT_RC_SUCCESS`	Success.
`FFT_RC_BAD_DIMENSION`	*xdim* or *ydim* is less than 1 or larger than 2048.
`FFT_RC_NOT_IMPLEMENTED`	Not implemented feature.
`FFT_RC_NO_INVERSE`	This is a limitation of the current release. Inverse FFT is not supported in this release.
`FFT_RC_BAD_PARM`	Invalid input parameter.
`FFT_RC_FAILED`	Generic internal errors.

fft_2d_sp_terminate

NAME

fft_2d_sp_terminate: Cleans up the environment after you have finished computing 2D FFTs.

SYNOPSIS

int fft_2d_sp_terminate(fft_2d_handle_t handle);

Parameters

`handle [IN]`	The handle created in the initialization step

DESCRIPTION

This function cleans up the environment after you have finished computing 2D FFTs. *fft_handle* is input.

RETURN VALUE

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

fft_2d_dp_initialize

NAME

fft_2d_2p_initialize: Sets up the environment for performing 2D FFTs double precision computation.

SYNOPSIS

int fft_2d_dp_initialize(fft_2d_handle_t *handle, unsigned int nspus);

Parameters

`handle [IN]`	A handle to the FFT runtime code
`nspus [IN]`	Number of SPUs to be assigned to the problem

DESCRIPTION

This function set up the environment for performing 2D FFTs double precision computation. Number of *spu* and *fft_handle* are input.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_NO_SPUS`	Insufficient SPUs are available
`FFT_RC_BAD_PARM`	Invalid input parameter
`FFT_RC_FAILED`	Generic internal errors

fft_2d_dp_perform

NAME

fft_2d_dp_perform: Completes one 2D complex to complex double precision FFTs.

SYNOPSIS

int fft_2d_dp_perform(fft_2d_handle_t handle, unsigned int xdim, unsigned int ydim, void **src_addr, void **dst_addr, unsigned int inverse_flag, unsigned int format_flag);

Parameters

`handle[IN]`	Handle of type **fft_2d_handle_t* created in the initialization step.
`xdim[IN]`	Size of 2D array in the x dimension.
`ydim[IN]`	Size of 2D array in the y dimension.
`src_addr[IN]`	Input data address.
`dst_addr[IN]`	Output data address.
`inverse_flag[IN]`	Flag, which is a 0 for a forward FFT, and non-zero for an inverse FFT.
`format_flag[IN]`	Bit flag to set the FFT type. Possible values are: • FFT_TYPE_C2C: Do 2D Complex to Complex FFT

DESCRIPTION

This function completes one 2D complex to complex double precision FFTs. According to inverse flag, determine whether a FFT or IFFT is needed. *format_flag* determines the FFT type. For this release, only FFT_TYPE_C2C is supported.

RETURN VALUE

`FFT_RC_SUCCESS`	Success.
`FFT_RC_BAD_DIMENSION`	*xdim* or *ydim* is less than 1 or larger than 2048.
`FFT_RC_NOT_IMPLEMENTED`	Not implemented feature.
`FFT_RC_NO_INVERSE`	This is a limitation of the current release. Inverse FFT is not supported in this release.
`FFT_RC_BAD_PARM`	Invalid input parameter.
`FFT_RC_FAILED`	Generic internal errors.

fft_2d_dp_terminate

NAME

fft_2d_dp_terminate: Cleans up the environment after you have finished computing 2D FFTs.

SYNOPSIS

int fft_2d_dp_terminate(fft_2d_handle_t handle);

Parameters

`handle [IN]`	The handle created in the initialization step

DESCRIPTION

This function cleans up the environment after you have finished computing 2D FFTs. *fft_handle* is input.

RETURN VALUE

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

Chapter 5. SPE APIs

The FFT library provides one set of SPE APIs for single precision FFT 1D computation.

This section describes the following APIs:

fft_1d_c2c_spu

NAME

fft_1d_c2c_spu: Performs a single complex to complex FFT on the SPU.

SYNOPSIS

int fft_1d_c2c_spu(unsigned int num_elems, vector float* srcAddr, vector float* dstAddr, unsigned int inverse_flag);

Parameters

`num_elems[IN]`	Number of elements of each FFT
`srcAddr[IN]`	Address of the first input element
`dstAddr[IN]`	Address of the first output element
`inverse_flag[IN]`	Flag, which is a 0 for a forward FFT, and non-zero for an inverse FFT

DESCRIPTION

This function completes one complex to complex FFT 1D on SPU. *inverse flag* determines whether a FFT or IFFT is needed.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_BAD_DIMENSION`	If *num_elems* > MAX_PROB_SIZE_C2C_1 (10000).
`FFT_RC_BAD_PARM`	Invalid input parameter

fft_1d_c2r_spu

NAME

fft_1d_c2r_spu: Performs a single complex to real FFT on the SPU.

SYNOPSIS

int fft_1d_c2r_spu(unsigned int num_elems, vector float* srcAddr, vector float* dstAddr, unsigned int inverse_flag);

Parameters

`num_elems[IN]`	Number of elements of each FFT
`srcAddr[IN]`	Address of the first input element
`dstAddr[IN]`	Address of the first output element
`inverse_flag[IN]`	Flag, which is a 0 for a forward FFT, and non-zero for an inverse FFT

DESCRIPTION

This function completes one complex to real FFT 1D on the SPU. It only does a forward transform. Inverse transform has not been implemented.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_BAD_DIMENSION`	If *num_elems* > MAX_PROB_SIZE_C2R_1 (10000)
`FFT_RC_BAD_PARM`	Invalid input parameter
`FFT_RC_NOT_IMPLEMENTED`	If *inverse_flag* for C2R/R2C is set to non-zero

fft_1d_r2c_spu

NAME

fft_1d_r2c_spu: Performs a single real to complex FFT on the SPU.

SYNOPSIS

int fft_1d_r2c_spu(unsigned int num_elems, vector float* srcAddr, vector float* dstAddr, unsigned int inverse_flag);

Parameters

`num_elems[IN]`	Number of input
`srcAddr[IN]`	Address of the first input element
`dstAddr[IN]`	Address of the first output element
`inverse_flag[IN]`	Flag, which is a 0 for a forward FFT, and non-zero for an inverse FFT

DESCRIPTION

This function completes one real to complex FFT 1D on the SPU. It only does a forward transform here. Inverse transform has not been implemented.

RETURN VALUE

`FFT_RC_SUCCESS`	Success
`FFT_RC_BAD_DIMENSION`	If *num_elems* > MAX_PROB_SIZE_C2C_1 (10000)
`FFT_RC_BAD_PARM`	Invalid input parameter
`FFT_RC_NOT_IMPLEMENTED`	If *inverse_flag* for C2R/R2C is set to non-zero

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® developerWorks® Web site located at:

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

Click the **Docs** tab.

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*

- *Three dimensional FFT Prototype Library Programmer's Guide and API Reference*

Installation

- *SDK for Multicore Acceleration Version 3.1 Installation Guide*

Tools

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

IBM PowerPC® Base

- *IBM PowerPC Architecture™ 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.

platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

© (your company name) (year). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. _enter the year or years_. All rights reserved.

If you are viewing this information in softcopy, the photographs and color illustrations may not appear.

Trademarks

IBM, the IBM logo, and ibm.com® are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. These and other IBM trademarked terms are marked on their first occurrence in this information with the appropriate symbol (® or ™), indicating US registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A complete and current list of IBM trademarks is available on the Web at http://www.ibm.com/legal/copytrade.shtml.

Adobe®, Acrobat, Portable Document Format (PDF), and PostScript® are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc., in the United States, other countries, or both and is used under license therefrom.

Linux® is a trademark of Linus Torvalds in the United States, other countries, or both.

Other company, product or service names may be trademarks or service marks of others.

Terms and conditions

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

Personal Use: You may reproduce these publications for your personal, noncommercial use provided that all proprietary notices are preserved. You may not distribute, display or make derivative works of these publications, or any portion thereof, without the express consent of the manufacturer.

Commercial Use: You may reproduce, distribute and display these publications solely within your enterprise provided that all proprietary notices are preserved. You may not make derivative works of these publications, or reproduce, distribute or display these publications or any portion thereof outside your enterprise, without the express consent of the manufacturer.

Glossary

API

Application Program Interface.

Broadband Engine

See *CBEA*.

CBEA

Cell Broadband Engine Architecture. A new architecture that extends the 64-bit PowerPC Architecture. The CBEA and the Cell Broadband Engine are the result of a collaboration between Sony, Toshiba, and IBM, known as STI, formally started in early 2001.

Cell/B.E. processor

The Cell/B.E. processor is a multi-core broadband processor based on IBM's Power Architecture.

Cell Broadband Engine processor

See *Cell/B.E* processor.

compiler

A programme that translates a high-level programming language, such as C++, into executable code.

DFT

Discrete Fourier transform. One of the specific forms of Fourier analysis. It transforms one function into another, which is called the frequency domain representation, or the DFT, of the original function. The DFT requires an input function that is discrete and whose non-zero values have a limited (finite) duration.

FFT

Fast Fourier Transform. An algorithm to compute the discrete Fourier transform (DFT) and its inverse.

FORTRAN

FORmula TRANslator). A high-level programming language for problems that can be expressed algebraically.

PDF

Portable document format.

PPE

PowerPC Processor Element. The general-purpose processor in the Cell.

PPU

PowerPC Processor Unit. The part of the *PPE* that includes the execution units, memory-management unit, and L1 cache.

SDK

Software development toolkit for Multicore Acceleration. A complete package of tools for application development.

SPE

Synergistic Processor Element. Extends the PowerPC 64 architecture by acting as cooperative offload processors (synergistic processors), with the direct memory access (DMA) and synchronization mechanisms to communicate with them (memory flow control), and with enhancements for real-time management. There are 8 SPEs on each cell processor.

SPU

Synergistic Processor Unit. The part of an SPE that executes instructions from its local store (LS).

vector

An instruction operand containing a set of data elements packed into a one-dimensional array. The elements can be fixed-point or floating-point values. Most Vector/SIMD Multimedia Extension and SPU SIMD instructions operate on vector operands. Vectors are also called SIMD operands or packed operands.

Index

Printed in USA

SC34-2562-00