Cell Broadband Engine

Software Development Toolkit Installation and User's Guide Version 1.1

SC33-8323-00

Cell Broadband Engine

Software Development Toolkit Installation and User's Guide Version 1.1

SC33-8323-00

Note: Before using this information and the product it supports, read the general information in Appendix B, “Notices,” on page 41.

First Edition (September 2006)

This edition applies to the BladeCenter QS20 and to all subsequent preleases and modifications until otherwise indicated in new editions.

Contents

Chapter 1. Introduction 1
About this book 1
New in this release 1
Supported platforms 2
How to use the SDK 2
Prerequisites 2
Licenses 3
Getting support 3
Related documentation 3
Cell BE processor 3
Cell BE programming using the SDK 4
IBM Full-System Simulator 4
PowerPC base 4

Chapter 2. Installing Linux Fedora Core 5 5
Downloading the Fedora Core 5 installation packages 5
Copying the installation files 5
Installing Fedora Core 5 on an x86 or PPC64 server 6
Installing Fedora Core 5 on a BladeCenter QS20 6
Fedora Core 5 installation overview 6
The network installation environment 7
Setting up a netboot environment 8
Setting up a network installation environment 8
Installing Fedora Core 5 8
Starting the installation 8
Copying a network-enabled kernel version 11
Additional installation steps for installed InfiniBand cards 12
Rebooting the BladeCenter QS20 12
Checking and adapting /etc/yaboot.conf 12
Checking and setting up a swap area (if required) 13
Configuring yum (if required) 13
Managing a BladeCenter QS20 13
Checking the firmware version 13
Booting a BladeCenter QS20 13
Shutting down and restarting the BladeCenter QS20 14
Finishing the Fedora Core 5 installation 14
gcc 14
make 14
perl 14
freeglut 15
libX11 15
TK 15
netpbm 15

Chapter 3. Installing the SDK 17
SDK components 17
Simple install of the SDK 17
Upgrading SDK 1.1 to SDK 1.1.1 18
Other ways to install the SDK 19
Uninstalling the SDK 19
Other script tasks and options 20
RPMs in SDK 21

Chapter 4. Contents of the SDK 23
GNU tool chain 23
IBM XL C/C++ compiler. 23
IBM Full-System Simulator 24
System root image for the simulator 25
Linux kernel 26
Linux support libraries 26
SPE runtime management library 26
numactl 27
Libraries and samples 27
Libraries and samples subdirectories 27
SPU timing tool. 29

Chapter 5. Using the SDK 31
Running the Full-System Simulator 31
The callthru utility 32
Read and write access to the simulator sysroot image 33
Enabling Symmetric Multiprocessing Support 33
SDK programming samples 33
Changing the default compiler 34
Building and running a specific program. 34
System root directory 34
Support for huge TLB file systems 35
SDK development best practices 35
Developing applications with a user (non-root) account 35
Using a shared development environment 36
Restrictions and limitations 36

Appendix A. Quaternion Julia Set Ray-tracing sample 37
Algorithm overview 37
Converting Cg to cell code 37
Load balancing framework. 38
Software cache. 39
SOA verses AOS data structures 40

Appendix B. Notices 41
Edition notice 42
Trademarks 44

Glossary 45

Index 49

Chapter 1. Introduction

The Software Development Toolkit (SDK) for the Cell Broadband Engine (Cell BE) is a complete package of tools to allow developers firsthand programming experience with the Cell BE Processor. The SDK is composed of development tool chains, software libraries and sample source files, a system simulator, and a Linux kernel that fully support the capabilities of the Cell BE.

The complete SDK is available as a .iso image, or as individual components. Alternatively you can download individual SDK components. The packages are available from the IBM@server alphaWorks Web site http://www.alphaworks.ibm.com/tech/cellsw and the Barcelona Supercomputing Center (BSC) Web site http://www.bsc.es/projects/deepcomputing/linuxoncell/. All of the files in the SDK are distributed as RPM packages except for the SDK install script (cellsdk).

The alphaWorks Web site hosts the IBM software including the Cell BE Full System Simulator, XL C/C++ compiler, libraries and samples, Synergistic Processor Unit (SPU) timing tool, and the cellsdk script.

The BSC Web site hosts the open source-derived packages such as the GCC-based tool chain for the Cell BE, the Linux kernel and support libraries, and system support for the Cell BE Full System Simulator.

About this book

Each section covers a different topic:

- Chapter 2, “Installing Linux Fedora Core 5,” on page 5 explains how to install the Fedora Core 5 Linux operating system on the supported hardware platforms
- Chapter 3, “Installing the SDK,” on page 17 explains how to install the BladeCenter QS20 SDK
- Chapter 4, “Contents of the SDK,” on page 23 describes each of the components of the SDK in more detail and what is needed for each of the supported platforms and configurations
- Chapter 5, “Using the SDK,” on page 31 describes some additional features and best practices for developing application using the SDK

In addition, there is a programming tutorial provided on the SDK CD which gives an introduction to writing applications for the Cell BE platform.

New in this release

SDK 1.1 contains a number of significant enhancements over the Cell BE SDK 1.0 and 1.0.1 and completely replaces these SDKs versions. These enhancements include:

- Linux kernel (2.6.16) and library support for BladeCenter QS20s that contain two Cell BE Processors giving a total of 16 Synergistic Processor Elements (SPEs)
- Support for PowerPC 64-bit hardware such as Apple Power Mac G5 and IBM PowerPC as development platforms
- Added C++ support to the XL C compiler for PowerPC Processing Unit (PPU) applications
- Added support for Gnu Debugger (GDB) server running in both PowerPC Processing Elements (PPEs) and SPEs

- Upgraded GNU Compiler Collection (GCC) compiler for PPU and SPU programs to version 4.0.2
- Upgraded `binutils` to version 2.16.1
- Additions and updates to the libraries and samples including a new sample that ray traces the quaternion Julia Set
- Added support for Non-Uniform Memory Access (NUMA) to improve the performance of memory accesses between SPEs
- Improved installation by using a new process and RPMs

SDK 1.1 has been refreshed to coincide with the general availability of the BladeCenter QS20. This refresh contains BladeCenter QS20 documentation on the ISO image and an updated IBM Full-System Simulator that runs faster on 64-bit platforms and can simulate a SMP configuration with dual Cell BE processors as are used in the BladeCenter QS20.

Supported platforms

Cell BE applications can be developed on the following platforms:

- x86
- x86-64
- 64-bit PowerPC (PPC64)
- BladeCenter QS20

How to use the SDK

The SDK includes both PPU and SPU compilers for all the supported platforms. A Cell BE application can be run either natively on a BladeCenter QS20 or in the IBM Full-System Simulator (simulator), which is supported on all of the host platforms. The simulator on the BladeCenter QS20 is useful for debugging or verifying a problem. For example, it is possible to build an application on an x86 system, test under the simulator on that system, and then later run the same binary natively on a BladeCenter QS20.

Prerequisites

The Cell BE SDK runs in Fedora Core 5, which must be installed before you install the SDK, see Chapter 2, “Installing Linux Fedora Core 5,” on page 5 for details.

Note: This SDK may work on other Linux distributions, but it has only been tested and certified to run on Fedora Core 5. A key requirement for any Linux installation is glibc Version 2.4 or newer.

Table 1 shows the recommended minimum configuration on each platform.

Table 1. Configurations

System	Recommended minimum configuration
x86	2GHz Pentium 4 processor
PowerPC	64-bit PPC with a clock speed of 1.42 GHz. 32-bit PPC platforms are not supported.
BladeCenter QS20	Must be at revision 31 or greater and have a minimum firmware level of FW6.14.7 or later (see “Checking the firmware version” on page 13)

All systems must have:

- Hard disk space: 5 GB (minimum) to install the source package and the accompanying development tools
- 1 GB RAM (minimum) on the host system

Note: The simulator requires that the minimum amount of RAM must be twice the amount of simulated memory. For example, to simulate a system with 512 MB of RAM, the host system must have at least 1 GB of RAM.

Licenses

The source code and binaries that are part of the total SDK package are distributed with different licenses. The packages on the BSC Web site are generally open source and use the General Public license (GPL) (http://www.gnu.org/copyleft/gpl.html) or Lesser General Public (LGP) (http://www.gnu.org/licenses/licenses.html#LGPL) license. If you are not already familiar with either, visit the Free Software Foundation (FSF) for more information.

The Common Public License (CPL), is used for projects like Eclipse and is the license for the SDK libraries and samples package. It is the successor to the IBM Public License (IPL) and you can find a FAQ (http://www-128.ibm.com/developerworks/library/os-cplfaq.html) about the CPL license on the IBM developerWorks. The Cell Full System Simulator and SPU Timing tool are licensed under the IBM International License Agreement for Early Release of Programs (ILAR). The XL C/C++ compiler for Cell Processor is licensed under a modified ILAR.

Getting support

The BladeCenter QS20 SDK is supported through the Cell BE architecture forum on the developerWorks Web site at http://www.ibm.com/developerworks/power/cell/. There is also support for the IBM Cell Full-System Simulator and XL C/C++ Compiler through their individual alphaWorks forums. If in doubt, start with the Cell architecture forum.

As mentioned earlier this version of the Cell BE SDK 1.1 supersedes all previous versions of the SDK.

Note: The Cell BE SDK is provided on an “as-is” basis. Wherever possible, workarounds to problems will be provided in the respective forums.

Related documentation

There is a set of tutorial and reference documentation for the Cell BE stored in the IBM online technical library http://www.ibm.com/chips/techlib/techlib.nsf/products/Cell_Broadband_Engine. The following documentation is available:

Cell BE processor

- *Cell Broadband Engine Architecture*
- *Cell Broadband Engine Programming Handbook*
- *Cell Broadband Engine Registers*
- *SPU C/C++ Language Extensions*
- *Synergistic Processor Unit (SPU) Instruction Set Architecture*

- *SPU Application Binary Interface Specification*
- *Assembly Language Specification*
- *Cell Broadband Engine Linux Reference Implementation Application Binary Interface Specification*

Cell BE programming using the SDK

After you have installed the SDK, you can find the following PDFs in the `/opt/IBM/cell-sdk-1.1/docs` directory:

- *Cell Broadband Engine Programming Tutorial documentation*
- *SPE Runtime Management library documentation*
- *SDK Sample Library documentation*
- *IDL compiler documentation*

IBM Full-System Simulator

The following PDFs can be found in the `/opt/IBM/systemsim-cell/doc` directory after installing the SDK:

- *IBM Full-System Simulator Users Guide*
- *IBM Full-System Simulator Command Reference*
- *Performance Analysis with the IBM Full-System Simulator*
- *IBM Full-System Simulator BogusNet HowTo*

PowerPC base

The following documents can be found on the developerWorks Web site at http://www.ibm.com/developerworks/eserver/library:

- *PowerPC Architecture Book, Version 2.02*
  - *Book I: PowerPC User Instruction Set Architecture*
  - *Book II: PowerPC Virtual Environment Architecture*
  - *Book III: PowerPC Operating Environment Architecture*
- *PowerPC Microprocessor Family: Vector/SIMD Multimedia Extension Technology Programming Environments Manual Version 2.06c*

Chapter 2. Installing Linux Fedora Core 5

Before you install the Cell BE SDK, you must install Fedora Core 5.

This section describes how to:

- Download Fedora Core 5
- Install Fedora Core 5 on an x86, a PowerPC or a BladeCenter QS20
- Install prerequisite packages

Downloading the Fedora Core 5 installation packages

You download Red Hat Fedora Core 5 installation packages from the download site, http://fedora.redhat.com. Downloads are available as .iso images which can be used to create CDs or DVDs, or as an unpacked installation tree. The Fedora Core site also has the installation instructions available for download.

Note: You should allow approximately three hours for installation and configuration.

Use your browser or FTP to download the .iso images. If you wish to download to an existing Linux server or workstation, you can use `wget` to download the unpacked installation tree. To do this, issue the following command:

```
wget -r http://download.fedora.redhat.com/pub/fedora/linux/core/5/ppc/os/
```

Note: Depending on your environment the simplest way is to download the *.iso images and then unpack them on your HTTP/FTP server rather than downloading the unpacked installation tree. You do not need the rescue image.

Copying the installation files

If you are already running Linux, you can save the .iso images to a directory on your server or another server on the network. You can then mount the image and copy the files to a directory which then becomes the installation source directory. Alternatively, you can make CDs or DVDs and mount them in your CD or DVD drive.

1. Create the installation directory from iso images.

   The installation files can be copied from the .iso images to any directory accessible over the network. The following example shows how to mount the .iso images and copy the files to a directory named `/srv/repos/p/FC5`.

   a. Create directory for each CD:

   ```
   mkdir /mnt/cd1
   mkdir /mnt/cd2
   ```

   b. Mount the CDs:

   ```
   cd /srv/repos/p
   mount -o loop FC5/FC5-ppc-disc1.iso /mnt/cd1
   mount -o loop FC5/FC5-ppc-disc2.iso /mnt/cd2
   ```

   c. Copy each CD:

   ```
   cp -rv /mnt/cd1/* /srv/repos/p/FC5
   cp -rv /mnt/cd2/* /srv/repos/p/FC5
   ```

   d. Cleanup:

   ```
   umount /mnt/cd1
   umount /mnt/cd2
   ```

2. Download all .iso images and burn them on CDs or DVDs (this can be done on a different system)
   a. Make a directory where the CD or DVD images will be copied
   b. Copy the images to that directory from the CD or DVD.

Installing Fedora Core 5 on an x86 or PPC64 server

If you are using an x86 or PPC64 host server, you can install Fedora Core 5 from the CDs or DVD and go to “Finishing the Fedora Core 5 installation” on page 14.

Installing Fedora Core 5 on a BladeCenter QS20

This section describes how to install Fedora Core 5 for PPC64 on a BladeCenter QS20.

Note the following:

- Fedora Core 5 asks you at the end of the installation process to reboot the BladeCenter QS20. At this point DO NOT REBOOT the blade, DO NOT CLICK the reboot button otherwise you will have a non-working Linux® on your BladeCenter QS20 and you will have to restart the installation.
- The kernel shipped with Fedora Core 5 does not support the full Cell BE-based blade functionality. The easiest way to get the updated kernel is to install the Cell BE SDK as described in Chapter 3, “Installing the SDK,” on page 17.
- You must download an updated `ppc64bcmfix.img` from http://www.bsc.es/projects/deepcomputing/linuxoncell/ to your install server. The original `ppc64.img` does not support the network hardware of this hardware revision and is therefore not usable for a network installation.

Fedora Core 5 installation overview

The outline for the installation is as follows:

1. Set up a netboot environment.
2. Set up a net install environment.
3. Perform a manual installation.

The Fedora Core 5 installation process starts by booting a kernel with the install `initrd` from the network device (this is the only supported installation method on a BladeCenter QS20). The init process `/sbin/init` starts `/sbin/loader` prompts you for the installation language and installation method. For a network installation, the loader also configures the network and queries the parameter for the install server, before it downloads the secondary stage image `Fedora/base/stage2.img` from the install server. After mounting the disk image, loader passes control to the Python script `anaconda`.

The Anaconda installer is the main installation program for Fedora Core 5 and performs the remaining steps of the installation, either manually through configuration screens or automatically using the `kickstart` configuration file. This includes downloading all RPMs, which are selected for installation from the install server.

The network installation environment

Notes:

1. The IP addresses used in the examples below are for illustrative purposes only. Use IP addresses allocated to your network.
2. During installation, you are prompted for the directory containing the Fedora Core 5 installation files. The illustrations below show `/fedoratree` as the source containing the installation files. Change this to the path containing your installation files.

For the remainder of this document, it is assumed that you have the following environment:

- A BladeCenter QS20 (10.32.5.11). This is the installation target.
- A DHCP/BOOTP server (10.32.0.1).
- An install server (10.32.0.1) running a TFTP server, with the installation source. This server must also be able to run Fedora Core 5 if the installation material requires modification.
- An HTTP or FTP server (10.64.0.31) with the installation source.

The HTTP/FTP server can reside on the same server as the DHCP/BOOTP/TFTP server.

Figure 1 shows a typical network installation environment.

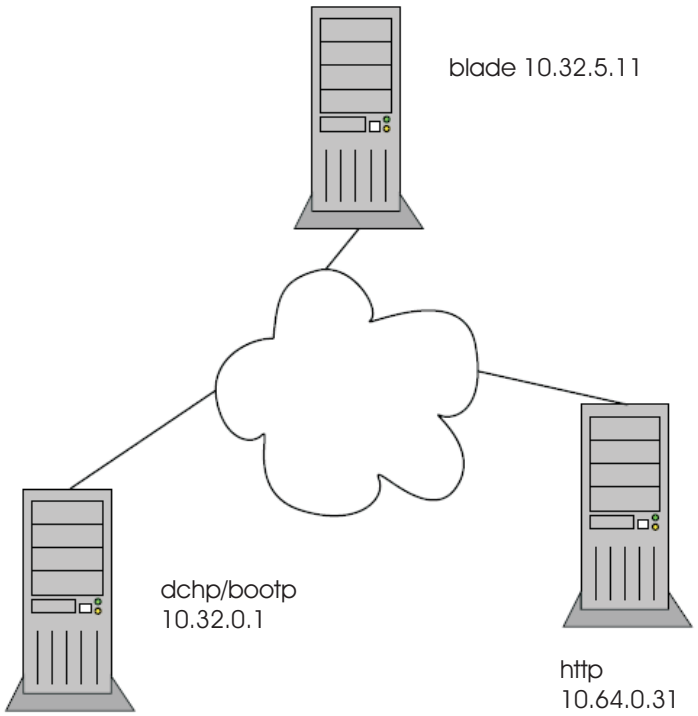


Figure 1. The network installation environment

Setting up a netboot environment

You must download an updated `ppc64bcmfix.img` from http://www.bsc.es/projects/deepcomputing/linuxoncell/ to your install server. The original `ppc64.img` does not support the network hardware of this hardware revision and is therefore not usable for a network installation.

Copy the new `ppc64bmcfix.img` file to the `/tftpboot` directory of the TFTP/BOOTP server and make sure that it matches the respective entry in `/etc/dhcpd.conf`.

Note: Rebuilding the `ppc64bmcfix.img` is possible but outside the scope of this document.

Setting up a network installation environment

Fedora Core 5 is installed over the network using TFTP. Generally, a TFTP server is used as the install server.

Installing Fedora Core 5

This section describes how to install Fedora Core 5.

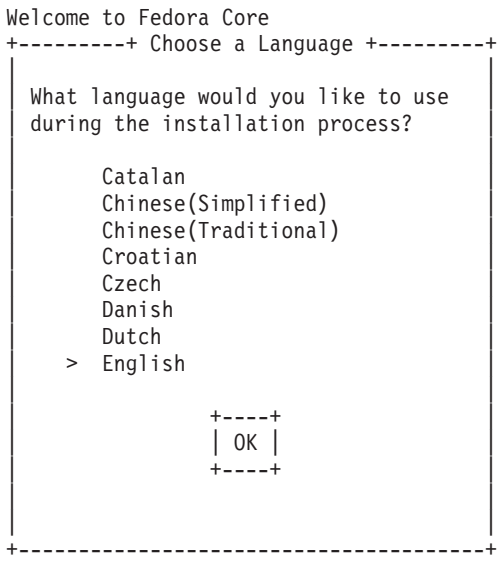
Starting the installation

To install Fedora Core 5, do the following:

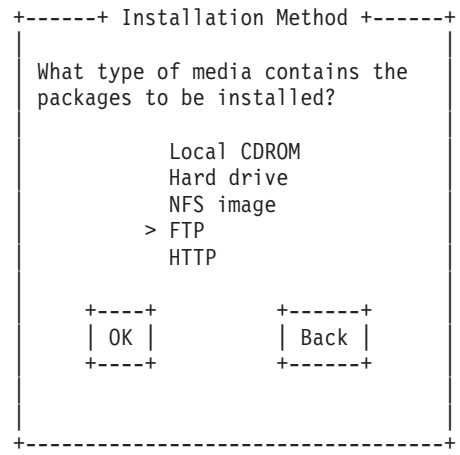
1. Either insert a new hard disk into the BladeCenter QS20, or overwrite an existing hard disk with an existing Linux.
2. Connect the BladeCenter QS20 to a serial console (115200,N,1,8, no handshake) and boot it up to the firmware prompt.
3. To start the installation, enter the following:

   ```
   > netboot vnc console=hvc0
   ```

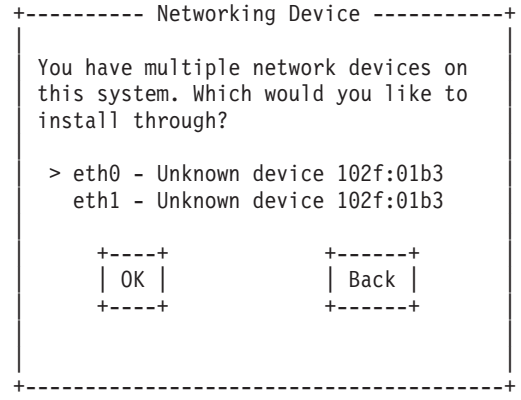
4. Select the language you would like to use for the installation.



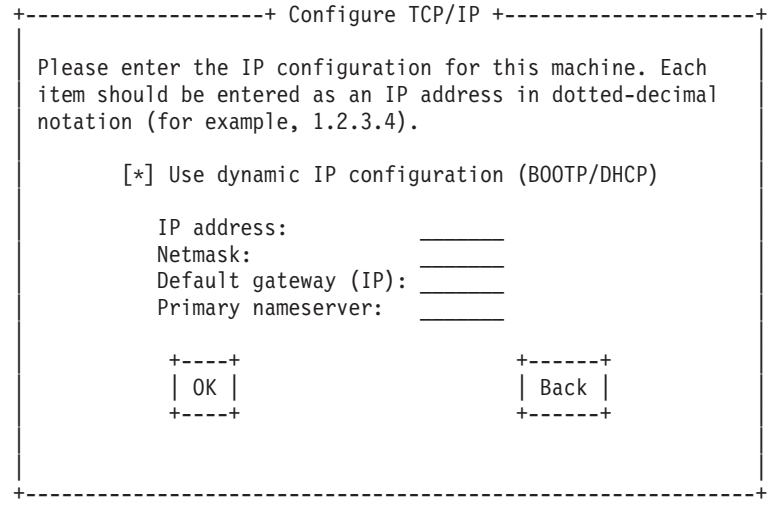
5. Select the media type that contains the installation packages.



6. Select the network device.



7. Enter the IP configuration for your server. Select **Use dynamic IP configuration (BOOTP/DHCP)**. Fedora Core 5 determines the host name and domain from the dhcp/bootp server.



8. Enter the name the FTP site name and the directory that contains the Fedora Core:

```
+-----------------+ HTTP Setup +------------------+
|                                                  |
| Please enter the following information:          |
|                                                  |
|     o the name or IP number of your FTP server   |
|     o the directory on that server containing    |
|       Fedora Core for your architecture          |
|                                                  |
|   FTP site name:    10.32.0.1_____________       |
|                                                  |
|   Fedora core directory: /fedoratree_________    |
|                                                  |
|  [*] Use non-anonymous ftp                       |
|                                                  |
|         +----+                +------+          |
|         | OK |                | Back |          |
|         +----+                +------+          |
|                                                  |
|                                                  |
+--------------------------------------------------+
```

9. Enter an FTP account name and password.

```
+----------------  Further FTP Setup  --------------+
|                                                   |
| If you are using non anonymous ftp, enter the     |
| account name and password you wish to use below.  |
|                                                   |
|      Account name:  userid_________________       |
|      Password:      password_________________     |
|                                                   |
|          +----+                +------+           |
|          | OK |                | Back |           |
|          +----+                +------+           |
|                                                   |
|                                                   |
+---------------------------------------------------+
```

10. The following is displayed:

```
+-------------------------------------------------------------------------+
| Running anaconda, the Fedora Core system installer - please wait...     |
| Framebuffer ioctl failed. Exiting.                                      |
| Probing for video card:   Unable to probe                               |
| Probing for monitor type:   Unknown monitor                             |
| Probing for mouse type:   No — mouse                                    |
| No video hardware found, assuming headless                              |
| Starting VNC...                                                         |
|                                                                         |
| WARNING!!! VNC server running with NO PASSWORD!                         |
| You can use the vncpassword=<password> boot option                      |
| if you would like to secure the server.                                 |
|                                                                         |
| The VNC server is now running.                                          |
| Please connect to 10.32.5.11:1 to begin the install...                  |
|                                                                         |
| Press <enter> for a shell                                               |
| Starting graphical installation...                                      |
+-------------------------------------------------------------------------+
```

11. Start a VNC session on another computer in the network. At the command prompt of that computer enter `vncviewer <target IP>`, where `<target IP>` is the address of the BladeCenter QS20 being installed, for example, 10.32.5.11:1. Continue the install process from the computer running the vncviewer session, not the BladeCenter QS20 where the install is actually taking place.

Note: Do NOT close the serial console on the blade server. This session is still needed at the end of the installation process.

12. Continue the installation, completing each section as required. Take note of the following:

    When you partition the hard disk, select **Create Custom Layout**. Figure 2 is an example of what a custom layout should look like.

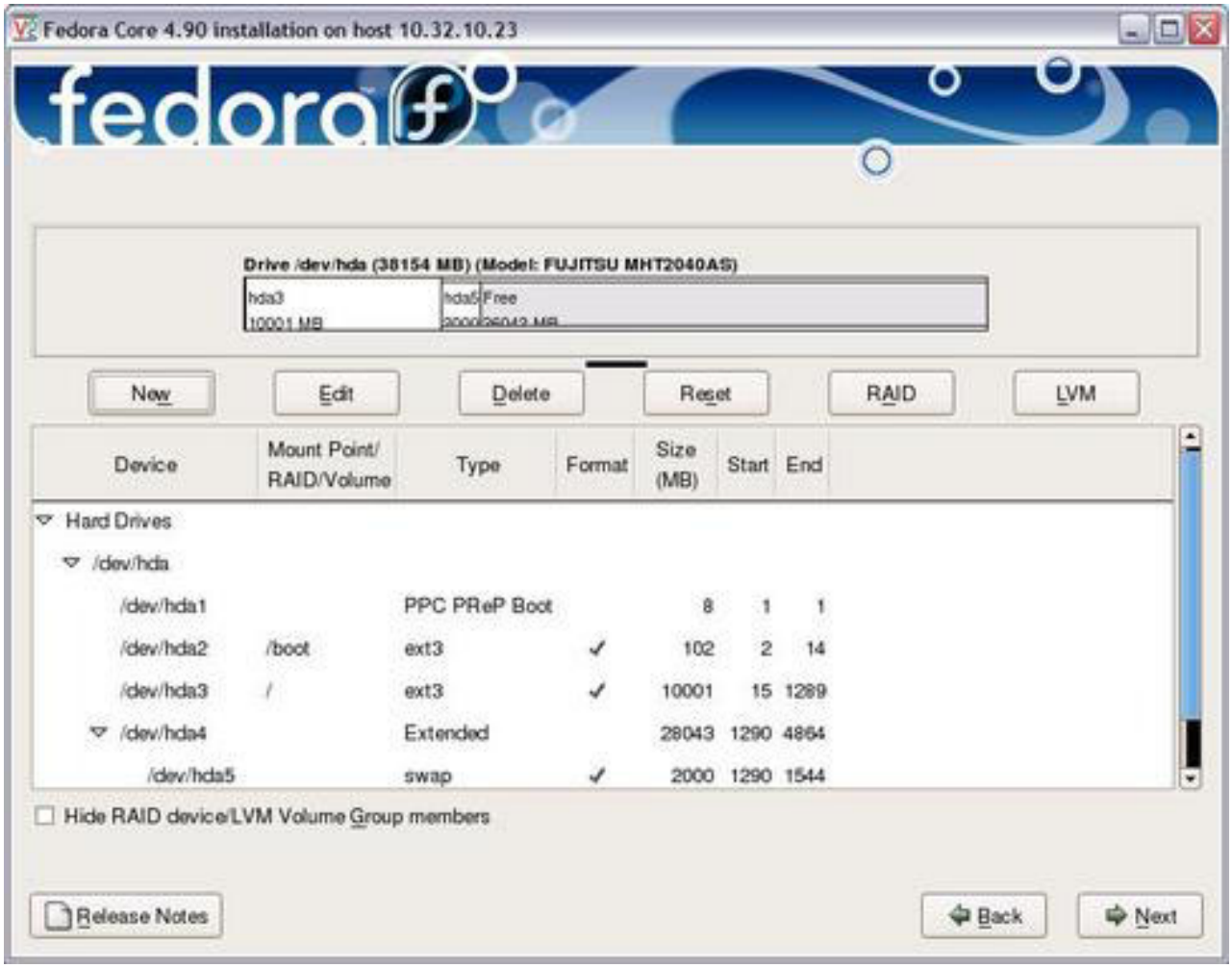


Figure 2. Example of a custom layout

- Fedora Core 5 is not able to boot larger partitions crossing certain sector boundaries, this means that you need a small boot partition of approximately 100MB
- The partition containing the `yaboot.conf` must be readable at boot time (and therefore cannot be an LVM partition but must be ext3).
- If Fedora Core 5 asks you at the end of the installation to reboot the blade, DO NOT REBOOT the BladeCenter QS20 and DO NOT CLICK the reboot button otherwise you will have a non-functional Linux on your BladeCenter QS20. Depending on your firmware, or hardware revisions, or both, you need to perform additional steps.

Copying a network-enabled kernel version

The unpatched kernel from Fedora Core 5 does not support the network hardware and therefore leaves a bootable blade with no network support.

To correct this, you MUST either:

- Copy the network modules from the install image to the server hard disk

  or

- Copy and install the new kernel to the blade BEFORE you reboot the BladeCenter QS20.

```
cp *.ko /mnt/sysimage/lib/modules/2.6.15-1.2054_FC5/kernel/drivers/net/
```

or use the serial console to download the new kernel RPM to the hard disk:

1. Download the BladeCenter QS20 RPMs from the BSC Web site if you have not already done so (for example, via your HTTP/FTP server) .
2. Copy the BladeCenter QS20 RPMs to the blade (for example: `scp server:/...kernel-2.6.15.4-bsc3.0.ppc64.rpm /...`)
3. Perform the following command to install the RPMs: For example:

   ```
   ftp 10.32.0.1
   cd <directory>
   get <kernel_name>
   mv <kernel_name> /mnt/sysimage/root
   ```

   where `<directory>` is the name of the directory containing the RPMs.

Additional installation steps for installed InfiniBand cards

If you have InfiniBand cards installed, you must issue the following command:

```
echo blacklist ib_mthca >> /mnt/sysimage/etc/modprobe.conf
```

After you have installed the Fedora Core 5 kernel or the SDK 1.1 you may need to upgrade the kernel. A kernel update is available that fixes an issue with InfiniBand and the second Gigabit Ethernet interface and is intended only if you wish to use both. See http://www.bsc.es/projects/deepcomputing/linuxoncell/stable/linuxkernel.html for more information.

Rebooting the BladeCenter QS20

Reboot the system from the installation screen.

Checking and adapting /etc/yaboot.conf

If the installation has completed successfully, `yaboot.conf` shows an entry for the Fedora Core 5 kernel and an entry for the patched kernel. Check that the patched entry is correct and edit the label variable to something meaningful.

The following is a sample `yaboot.conf` file:

```
# yaboot.conf generated by anaconda

boot=/dev/hde1
init-message=Welcome to Fedora Core!\nHit <TAB> for boot options

partition=2
timeout=20
install=/usr/lib/yaboot/yaboot
delay=5
enablecdboot
enableofboot
enablenetboot
nonvram
fstype=raw

image=/vmlinuz-2.6.16-1.17.driver.be0613
        label=cell                      << change default name to user friendly name
        read-only
        initrd=/initrd-2.6.16-1.17.driver.be0613.img
        append="console=hvc0 rhgb quiet root=LABEL=/"
```

```
image=/vmlinuz-2.6.15-1.1826.2.10_FC5
        label=linux
        read-only
        initrd=/initrd-2.6.15-1.1826.2.10_FC5.img
        append="console=hvc0 rhgb quiet root=LABEL=/"
        ppc64bcmfix
```

Checking and setting up a swap area (if required)

If required, you can set up a swap area as follows:

```
free                                  << check if swap space is available
swapon -a                             << enable devices and files for swapping
mkswap /dev/VolGroup00/LogVol01       << set up a Linux swap area
```

Configuring yum (if required)

If required, configure the `/etc/yum.conf` file so that it points to the HTTP server. You must change the `baseurl` entry:

```
[main]
  cachedir=/var/cache/yum
  debuglevel=2
  logfile=/var/log/yum.log
  pkgpolicy=newest
  distroverpkg=redhat-release
  tolerant=1
  exactarch=1
  retries=20
  obsoletes=1
  gpgcheck=0

# PUT YOUR REPOS HERE OR IN separate files named file.repo
# in /etc/yum.repos.d

/etc/yum.repos.d/fedora-core.repo

[base]
name=Fedora Core $releasever - $basearch - Base
#baseurl=http://download.fedora.redhat.com/pub/fedora/linux/core/$releasever/$basearch/os/
baseurl=http://10.64.0.31/                  <<<< modify baseurl here
mirrorlist=http://fedora.redhat.com/download/mirrors/fedora-core-$releasever
enabled=1
gpgcheck=0
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-fedora
```

Managing a BladeCenter QS20

This section describes how to check the firmware version, boot, shut down, and restart a BladeCenter QS20.

Checking the firmware version

To check the firmware version, do the following:

1. Access the BladeCenter Management Module.
2. Select **Monitors** → **Firmware VPD**. The **Blade Server Firmware VPD** window contains the build identifier, release, and revision.

Booting a BladeCenter QS20

To boot a BladeCenter QS20, do the following:

1. Open the BladeCenter Management Module.
2. Set the appropriate boot device (network, hard disk) for the BladeCenter QS20 by selecting **Blade Tasks** → **Configuration** → **Boot Sequence**.
3. Power on the BladeCenter QS20 by selecting **Blade Tasks** → **Power/Restart** → checkmark the blade → **Power On Blade**.

Note:

- The boot process of the BladeCenter QS20 can only be monitored with a serial console (115200,N,1,8, no handshake) connected to the BladeCenter QS20's serial port on the front bezel
- You may need a null-modem cable
- To force the Blade into the OpenFirmware Prompt press **s** on the serial console during the early stages of the boot process

Shutting down and restarting the BladeCenter QS20

Always shutdown and restart a blade that has been booted to the Linux prompt with one of the following commands from a Linux shell on the BladeCenter QS20:

```
shutdown -g0 -i0 -y
halt
reboot
shutdown -r now
```

Do not use the Blade Center Management Module to power down or restart the Blade (using **Blade Tasks** → **Power/Restart** → checkmark the blade → **Power Off Blade / Restart Blade**) as you can corrupt the file system: the Blade Center will power off the BladeCenter QS20 without first notifying the operating system.

Finishing the Fedora Core 5 installation

At this point you should have Fedora Core 5 installed and rebooted on your machine, which is either an x86, a PowerPC or a BladeCenter QS20. It is recommended that you do a yum update regularly to keep up to date with the latest Fedora Core 5 changes. In addition, there are up to seven other dependencies that need to be installed before you install the Cell BE SDK.

Note: A default Fedora Core 5 install does **not** have all the required packages.

gcc

The regular Fedora Core 5 gcc GNU compiler is used to build two samples that are run on the host machine rather than on a Cell BE Processor. To verify if it is installed, run `rpm –q gcc`. The result should be similar to:

```
gcc-4.1.1-1.fc5
```

To get the gcc package and its dependencies, you must do a `yum install gcc`.

make

Make is used to build the libraries and sample programs. To verify if it is installed, run `rpm -q make`. The result should be similar to:

```
make-3.80-10.2
```

To get the make package, you must do a `yum install make`.

perl

Perl is required by several parts of the SDK. To verify if it is installed, run `rpm –q perl`. The result should be similar to:

```
perl-5.8.8-5
```

To get the perl package, you must run `yum install perl`.

freeglut

Two of the sample programs depend on OpenGL and require glut.h to compile successfully. To verify if it is installed, run `rpm –q freeglut-devel`. The result should be similar to:

```
freeglut-devel-2.4.0-4
```

To get the freeglut-devel package, you must run `yum install freeglut-devel`.

libX11

To execute the applications that depend on OpenGL, various X11 libraries are needed. To get the required packages, run:

```
yum install libX11
yum install libXau
yum install libXdmcp
```

TK

The simulator requires `libtk8.4.so`. To verify if it is installed, run `rpm –q tk`. The result should be similar to:

```
tk-8.4.13-1.1
```

If the tk package is not installed you see the following message when you run the full system simulator:

```
systemsim-cell/run/cell/../../bin/systemsim-cell: error while loading
shared libraries: libtk8.4.so: cannot open shared object file: No such file or directory.
```

To get the tk (and tcl) package, run `yum install tk`.

netpbm

The Julia set sample needs the netpbm and netpbm-devel packages on the Cell BE-based blade server. To verify if installed do a `rpm –qa | grep netpbm`. The result should be similar to:

```
netpbm-devel-10.33-1.fc5
netpbm-10.33-1.fc5
```

If these packages are not installed, you must do a `yum install netpbm`.

Chapter 3. Installing the SDK

This section assumes that you have installed Fedora Core 5 on your host system and are now ready to install the Cell BE SDK. The following Web sites are relevant:

- http://www.alphaworks.ibm.com/tech/cellsw

  The IBM alphaWorks site contains emerging technologies. In particular it contains the IBM Source Libraries and Samples, the Full-System Simulator and the Cell BE XL C/C++ compiler.
- http://www.bsc.es/projects/deepcomputing/linuxoncell/cbexdev.html

  The Linux on Cell BE-based systems web site at the Barcelona Supercomputing Center (BSC) provides information about how to enable Linux on Cell BE platforms. It hosts all of the open source patches and binaries needed to install SDK 1.1.
- http://www.ibm.com/developerworks/power/cell/

  The IBM developerWorks site contains technical articles and support forums for Cell BE.

SDK components

The Cell BE SDK consists of a set of RPM files and a script to install the appropriate RPMs for the target platform. The components are explained in more detail in Chapter 4, “Contents of the SDK,” on page 23. The table below shows combinations of platforms and components:

Table 2. Supported combinations of platforms and components

Component	x86/x86-64	PPC64	BladeCenter QS20
GNU Tool chain	Mandatory	Mandatory	Mandatory
XL C/C++ Compiler	Optional	Not applicable	Not applicable
Linux Kernel	Built-in to system root image	Built-in to system root image	Mandatory
Linux support libraries	Built-in to system root image	Built-in to system root image	Mandatory
Full-System Simulator	Mandatory	Mandatory	Optional
System Root Image for Simulator	Mandatory	Mandatory	Optional
Libraries and Samples	Mandatory	Mandatory	Mandatory

Some components consist of one or more RPM files and the detailed list of RPMs for each component and platform is described in “RPMs in SDK” on page 21.

Simple install of the SDK

The simplest way to install the SDK is to download the Cell BE SDK ISO image from the IBM alphaWorks Web site http://www.alphaworks.ibm.com/tech/cellsw. The ISO image contains all of the IBM licensed software and also all of Cell BE and SDK technical documentation in Adobe PDF format. You can also create a physical CD from this ISO image and read it on non-Linux systems.

The `cellsdk` shell script handles the SDK installation. The SDK's tools are primarily installed in `/opt/IBM`, although some of the toolchain commands are installed in the regular path. for example, the `spu-gcc` command is installed in `/usr/bin` on a PPC machine and in `/opt/sce` on an x86 machine. The simulator is installed in `/opt/IBM/systemsim-cell`, and the SDK in `/opt/IBM/cell-sdk-1.1/`.

The script begins by asking you whether you have read the associated license agreements. You must type **yes** to continue; **y** is not accepted as confirmation. The script checks to see what features might be needed and what features are available. If some of the other installation files (from the BSC Web site) are needed, they are automatically downloaded to the directory `/tmp/cellsdk-1.1`.

Do the following to install the SDK, and to upgrade the SDK, see "Upgrading SDK 1.1 to SDK 1.1.1":

1. As root download the SDK ISO disk image, `CellSDK11.iso` from the Cell BE SDK alphaWorks Web site http://www.alphaworks.ibm.com/tech/cellsw.
2. Create a mount directory and make sure nothing else is mounted on this directory:
   ```
   mkdir –p /mnt/cellsdk
   ```
3. Mount the disk image on the mount directory:
   ```
   mount –o loop CellSDK11.iso /mnt/cellsdk
   ```
4. Change directory to `/mnt/cellsdk/software`:
   ```
   cd /mnt/cellsdk/software
   ```
5. Install the SDK by using the following command and answer any prompts:
   ```
   ./cellsdk install
   ```
6. Change directory to any directory which is not the mount directory or below it:
   ```
   cd /
   ```
7. Unmount the disk image:
   ```
   umount /mnt/cellsdk
   ```

If you have previously installed SDK 1.1, download the latest SDK from alphaWorks and use the following command to upgrade your system to the SDK 1.1.1.

```
./cellsdk upgrade
```

During the install process you might be asked questions about which components to install or reinstall. Refer to Table 2 on page 17 to understand what the mandatory and optional components are. As part of the install procedure the SDK libraries and samples are compiled and linked by default using the GNU tools. See "Other ways to install the SDK" on page 19 for other cellsdk tasks and options that allow you to compile the libraries and samples with the XL C compiler or install the SDK without prompts.

You can run the cellsdk script multiple times to install or reinstall components. The install process also builds all of the libraries and samples without stopping if there is an error. Use the following command can be used to keep a log of what was done during the install process in case there is a problem:

```
./cellsdk install 2>&1 | tee /tmp/cellsdk-install.log
```

Upgrading SDK 1.1 to SDK 1.1.1

If you have previously installed SDK 1.1, download the latest SDK from alphaWorks and use the following command to upgrade your system to the SDK 1.1.1.

```
./cellsdk upgrade
```

Other ways to install the SDK

The install approach described in “Simple install of the SDK” on page 17 may not work in your environment. This section describes several other methods to get the components of the SDK and install it.

One problem is that the download from the BSC Web site may not work if you access the internet through a proxy. One solution to this problem is to use the `http_proxy` environment variable to tell `wget` about the proxy. For example, if you use a proxy with the IP address 192.168.10.11 on port 8080, set the environment variable as follows:

```
export http_proxy="http://username:password@192.168.10.11:8080"
```

If this does not work for you because of firewalls or other network issue, download the RPM files manually. Table 3 on page 21 gives a complete list of all the RPMs needed for a given hosting environment. You can also use the `cellsdk` script itself to determine if all of the dependent RPMs have been acquired. Issue the following command to see the list of missing files:

```
./cellsdk depend
```

Here is some example output from this command:

```
Do you plan to install the Cell Full System Simulator? (y/n):y

WARNING! The following required file(s) for SDK install were not found
-------------------------------------------------------------------------
  Missing file(s):
    ppu-gcc-c++-3.2-4.ppc.rpm
    ppu-gcc-3.2-4.ppc.rpm
    ppu-binutils-3.2-4.ppc.rpm
    spu-newlib-3.2-6.ppc.rpm
    spu-gcc-c++-3.2-6.ppc.rpm
    spu-gcc-3.2-6.ppc.rpm
    spu-binutils-3.2-6.ppc.rpm
    spu-gdb-3.2-6.ppc.rpm

  Paths searched:
    /home/user23
    /tmp/cellsdk-1.1

  These files may be downloaded from:
    http://www.bsc.es/projects/deepcomputing/linuxoncell/cellsimulator/sdk1.1/


Would you like to attempt automatically downloading these files
to /tmp/cellsdk-1.1? (y/n):n


ERROR: Missing SDK dependencies
```

Uninstalling the SDK

If you need to uninstall the SDK for whatever reason this can be done as root by executing the script as follows:

```
./cellsdk uninstall
```

Note: To ensure that the system is left in a usable state, the script does not uninstall the kernel on BladeCenter QS20s. You must change the yaboot configuration file (`/etc/yaboot.conf`) to load a different kernel at the next reboot and use the following command to uninstall the kernel:

```
rpm –e kernel-2.6.16-bsc4.2.
```

Other script tasks and options

The SDK script `cellsdk` implements a number of tasks and options for those tasks. If you run the script without any parameters, it displays the usage statement as follows:

```
Usage: cellsdk <install|uninstall|depend|build|synch|upgrade|verify|version> [OPTION...]

Task 'install' and 'depend' options:
  -f, --force    assume 'yes' to all questions.
  -x, --xlc      use XLC to compile the libraries and samples (x86 only)
  -g, --gcc      use GCC to compile the libraries and samples (default)
      --nosim    do not install the IBM Cell BE Full System Simulator
      --nokernel do not install the Cell BE kernel (CBE hardware only)

Task 'uninstall' options:
  -f, --force    assume 'yes' to all questions.

Task 'build' options:
  -x, --xlc      use XLC to compile the libraries and samples (x86 only)
  -g, --gcc      use GCC to compile the libraries and samples (default)
```

The script understands the following tasks:

Task	Description
`install`	Install the SDK as explained in “Simple install of the SDK” on page 17
`uninstall`	Uninstall the SDK as explained “Uninstalling the SDK” on page 19
`depend`	Determine if all of the required RPMs are available for an install as explained “Other ways to install the SDK” on page 19
`build`	Build or rebuild the libraries and samples. **Note:** A `make clean` is also done before the build.
`synch`	Synchronizes between the `/opt/IBM/cell-sdk-1.1/sysroot` directory and the system root image for the Full System Simulator. See “System root image for the simulator” on page 25 for more information.
`upgrade`	Only available for the SDK 1.1 Refresh to upgrade SDK 1.1 to the SDK 1.1 Refresh.
`verify`	Determines if the correct component versions have been installed.

`version`	Prints the SDK version string:
	`IBM Cell Broadband Engine SDK 1.1 (build 20060705)`
	
	OR for the SDK 1.1 refresh:
	`IBM Cell Broadband Engine SDK 1.1.1 (build 20060918)`

The script also supports the following five flags which are ignored if not specified for the task:

Flag	Description
`-f` or `--force`	Forces an install or uninstall with minimal prompting.
`-g` or `--gcc`	Selects the GNU/GCC tool chain on an x86 machine for compiling the libraries and samples.

Flag	Description
--nokernel	Do not install the kernel on the BladeCenter QS20. By default installing the kernel puts it first in the yaboot configuration file and it is used on the next reboot of the server. In most cases you should use this kernel as it has full support for the BladeCenter QS20.
--nosim	Do not install the simulator. This option only applies to a BladeCenter QS20 where a simulator is not required. Otherwise it is recommended that the simulator is installed so that it can be used for debugging or problem recreation.
-x or --xlc	Selects the XL C/C++ compiler on an x86 machine for compiling the libraries and samples. The GNU/GCC linker is used to build the complete executable.

RPMs in SDK

The table below shows a complete list of RPMs that can be installed for each supported platform. The rows in bold are RPMs that you can download from IBM alphaWorks Web site http://www.alphaworks.ibm.com/tech/cellsw. You can download the rows that are not in bold from the BSC Web site http://www.bsc.es/projects/deepcomputing/linuxoncell/cbexdev.html.

Table 3. RPMs that can be installed for each supported platform

Component area	x86/x86_64	PPC64	BladeCenter QS20
GNU Tool Chain for PPU	ppu-binutils-3.2-4.i686.rpm ppu-gcc-3.2-4.i686.rpm ppu-gcc-c++-3.2-4.i686.rpm ppu-sysroot-3.2-4.noarch.rpm ppu-sysroot64-3.2-4.noarch.rpm	ppu-binutils-3.2-4.ppc.rpm ppu-gcc-3.2-4.ppc.rpm ppu-gcc-c++-3.2-4.ppc.rpm	
GNU Tool Chain for SPU	spu-binutils-3.2-6.i686.rpm spu-gcc-3.2-6.i686.rpm spu-gcc-c++-3.2-65.i686.rpm spu-newlib-3.2-6.i686.rpm spu-gdb-3.2-6.i686.rpm	spu-binutils-3.2-6.ppc.rpm spu-gcc-3.2-6.ppc.rpm spu-gcc-c++-3.2-6.ppc.rpm spu-newlib-3.2-6.ppc.rpm spu-gdb-3.2-6.ppc.rpm	
IBM XL C/C++ Compiler	**xlc-cell-lib-1.1-9.i386.rpm** **xlc-cell-cmp-1.1-9.i386.rpm** **xlcpp-cell-lib-1.1-9.i386.rpm** **xlcpp-cell-cmp-1.1-9.i386.rpm**	Not Supported for PPC64	
Linux Kernel	reintegrated within system root image for simulator		kernel-2.6.16-bsc4.2.ppc64.rpm kernel-cbesim-2.6.16-bsc4.2.noarch.rpm
Library SPE	libspe-1.1.0-1.ppc.rpm libspe-devel-1.1.0-1.ppc.rpm elfspe-1.1.0-1.ppc.rpm libspe-1.1.0-1.ppc64.rpm libspe-devel-1.1.0-1.ppc64.rpm		
NUMA Control	Pre-integrated within system root image for simulator		numactl-0.9.8-1.cbe.ppc.rpm
OProfile	Pre-integrated within system root image for simulator		oprofile-0.9.1-8.cbe.ppc.rpm oprofile-devel-0.9.1-8.cbe.ppc.rpm
IBM Full System Simulator (SDK 1.1)	**systemsim-cell-1.1-6.i386.rpm**	**systemsim-cell-1.1-6.ppc.rpm**	
IBM Full System Simulator (SDK 1.1 Refresh)	**systemsim-cell-1.1-10.i386.rpm** **systemsim-cell-1.1-10.x86_64.rpm**	**systemsim-cell-1.1-10.ppc64.rpm**	
System root image for simulator	sysroot_image-1.1-011.noarch.rpm		
IBM Samples and Libraries	**cell-sdk-lib-samples-1.1-10.noarch.rpm**		
IBM SPU Timing	**cell-spu-timing-1.1-2.i686.rpm**	**cell-spu-timing-1.1-2.ppc.rpm**	

Chapter 4. Contents of the SDK

This section describes the contents of the SDK, where it is installed on the system, and how the various components work together.

GNU tool chain

The GNU tool chain contains the GCC compiler for the PPU and SPU implemented by Sony Computer Entertainment. For the PPU it is a cross-compiler on x86 and a replacement for the native GCC compiler on PPC platforms. The GCC compiler for the PPU is preferred and the make files are configured to use it when building the libraries and samples.

The GCC compiler also contains a separate SPE cross-compiler that supports the standards defined in the SPU C/C++ Language Extensions V2.1, SPU Application Binary Interface Specification V1.4, and Synergistic Processor Unit (SPU) Instruction Set Architecture V1.0 documents.

The associated assembler and linker additionally support the SPU Assembly Language Specification V1.3. The assembler and linker are common to both the GCC and XL C/C++ compilers. GDB support is provided for both PPU and SPU debugging, and the debugger client can be in the same process or a remote process.

On a non-PPC machine, the GNU Tool Chain is installed into the `/opt/sce/toolchain-3.2/ppu` and `/opt/sce/toolchain-3.2/spu` directories, containing the separate tool chains for the PPU and SPU, respectively. On a PPC64 or BladeCenter QS20, both tool chains are installed into `/usr`.

The patches to the standard 4.0.2 GNU/GCC compiler are available on the BSC Web site and are distributed under the GPL license.

IBM XL C/C++ compiler

The IBM XL C/C++ compiler for Cell BE Processor is a cross-compiler for x86 platforms only. This C/C++ cross-compiler generates code for the PowerPC Processor Unit (C and C++) or Synergistic Processor Unit (C only) and is tuned for the Cell BE Architecture The compiler requires the GCC Tool chain for Cell BE, which provides tools for cross-assembling and cross-linking applications for both the PPE and SPE.

The compiler supports the latest ISO C 1999 (International Standard ISO/IEC 9899:1999) standard, also known as C99. It also supports the C89 Standard and K&R style. To help port the code that was originally written for GCC, a subset of features related to GNU C is supported by XL C/C++. Other GNU compiler features are recognized (accepted and ignored). This helps minimize the number of changes that you must make to your GCC source code to ensure that it compiles successfully with XL C.

The XL C/C++ compiler provides three invocation commands:

- `ppuxlc`
- `ppuxlc++`
- `spuxlc`

The commands `ppuxlc` and `ppuxlc++` are used to generate code for the PPU, and `spuxlc` is used to generate code for the SPU.

The compiler invocation commands for the PPU perform all necessary steps for compiling C/C++ source files by `ppuxlc` or (C++ source using `ppuxlc++`) into `.o` files and linking the object files and libraries by `ppu-ld` into an executable program. Similarly, the compiler invocation command for the SPU performs all necessary steps for compiling C source files by `spuxlc` into `.s` files, assembling `.s` files into `.o` files by `spu-as`, and linking the object files and libraries into an executable program by `spu-ld`. The `ppu-embedspu` tool that is part of the GNU tool chain is used to link a PPU executable and a SPU executable into a single Cell BE executable.

The XL C/C++ compiler includes five base optimization levels:

- -O0: almost no optimization
- -O2: strong, low-level optimization that benefits most programs
- -O3: intense, low-level optimization analysis with basic loop optimization
- -O4: all of -O3 and detailed loop analysis and good whole-program analysis at link time
- -O5: all of -O4 and detailed whole-program analysis at link time.

Auto-SIMDization is enabled at O3 `-qhot` or O4 and O5 by default for the PPE, and at O3 `-qhot` or O4 and O5 for the SPE. SIMD is an abbreviation for Single Instruction Multiple Data; this is one of the key performance boosters in the Cell BE architecture.

The XL C/C++ compiler is installed into the `/opt/ibmcmp/xlc/1.0` directory and is distributed under the IBM ILAR license.

Note: The source code is not available.

IBM Full-System Simulator

The IBM Full System Simulator (simulator) is a software application that emulates the behavior of a full system that contains a Cell BE Processor. You can boot a Linux operating system on the simulator and run applications on the simulated operating system. The simulator also supports the loading and running of statically-linked executables and standalone tests without an underlying operating system.

The simulator infrastructure is designed for modeling processor- and system-level architecture at levels of abstraction, which vary from functional to performance simulation models with a number of hybrid fidelity points in between:

- **Functional-only simulation:** Models the program-visible effects of instructions without modeling the time it takes to execute these instructions. Functional-only simulation assumes that each instruction can be run in a constant number of cycles. Memory accesses are synchronous and are also performed in a constant number of cycles. This simulation model is useful for software development and debugging when a precise measure of execution time is not significant. Functional simulation proceeds much more rapidly than performance simulation, and so is also useful for fast-forwarding to a specific point of interest.
- **Performance simulation:** For system and application performance analysis, the simulator provides performance simulation (also referred to as timing simulation.) A performance simulation model represents internal policies and mechanisms for system components, such as arbiters, queues, and pipelines. Operation latencies

are modeled dynamically to account for both processing time and resource constraints. Performance simulation models have been correlated against hardware or other references to acceptable levels of tolerance. The Full System Simulator for Cell BE Processor provides a cycle-accurate SPU core model that can be used for performance analysis of computationally-intense applications. However, this model can not be used for measuring or tracking memory access latencies. Refer to the IBM developerWorks SPU Pipeline Examination article http://www.ibm.com/developerworks/power/library/pa-cellspu/ for additional information.

- The simulator can also be configured to fast-forward the simulation, using a functional model, to a specific point of interest in the application and to then switch to a timing-accurate mode to conduct performance studies. Then various types of operational details can be gathered to gain insight into the behavior of real-world hardware and software systems. This option affords a high level of configurability: users can dynamically trade detail and accuracy of timing for faster simulation speed.

The version of the simulator shipped with the Cell BE SDK is compiled to run on Fedora Core 5. See the `/opt/IBM/systemsim/docs` subdirectory for complete documentation including a simulator user's guide. The pre-release name of the simulator is "Mambo" and this name may appear in some of the documentation.

The simulator for the Cell BE Processor is also available as an independent technology at http://www.alphaworks.ibm.com/tech/cellsystemsim.

The simulator is installed into the `/opt/IBM/systemsim-cell` directory and is distributed under the IBM ILAR license.

Note: The source code for the simulator is not available.

System root image for the simulator

The system root image for the system simulator is a file that contains a disk image of Fedora Core 5 files, libraries and binaries that can be used within the system simulator. This disk image file is preloaded with a full range of Fedora Core 5 utilities and also includes all of the Cell BE Linux support libraries described in "Linux support libraries" on page 26. This RPM file is by far the largest of the RPM files and when installed takes up 1.6 GB on the host server's hard disk.

The system root image for the simulator must exist in the same directory as the Linux kernel which is either the current directory when starting the simulator or the default `/opt/IBM/systemsim-cell/images/cell` directory. The cellsdk script takes care of automatically putting the system root image in the default directory.

This system root image can be mounted to see what it contains. Assuming a mount point of `/mnt/cell-sdk-1.1-sysroot`, which is the mount point used by the cellsdk script, the command to mount the system root image is:

```
mount -o loop /opt/IBM/systemsim-cell/images/cell/sysroot_disk /mnt/cell-sdk-1.1-sysroot/
```

The command to unmount the image is:

```
umount /mnt/cell-sdk-1.1-sysroot/
```

Do not attempt to mount the image on the host system while the simulator is running. The system root image should always be unmounted before you start the

simulator. You should not mount the system root image to the same point as the root on the host server as the system can become corrupted and fail to boot.

There are three ways to change files on the system root image disk:

- Mount it as described above. Then change directory (cd) to the mount point directory or below and modify the file using host system tools such as `vi` or `cp`.
- Use the `./cellsdk synch` command to synchronize the system root image with the `/opt/IBM/cell-sdk-1.1/sysroot` directory for libraries and samples (see "System root directory" on page 34) that have been cross-compiled and linked on a host system and need to be copied to the target system.
- Use the `callthru` mechanism (see "The callthru utility" on page 32) to source or sink the host system file during execution of the simulator. This is only method that can be used while the simulator is running.

The source for the system root image is available on the BSC Web site and is distributed under the GPL license.

Linux kernel

A number of patches have been made to the Linux 2.6.16 kernel to provide services needed to support the hardware facilities of the Cell BE Processor.

For the BladeCenter QS20 the kernel is installed into the `/boot` directory, `yaboot.conf` is modified and a reboot is required to activate this kernel. The `cellsdk` install task (see "Simple install of the SDK" on page 17) provides an option (`--nokernel`) not to install this kernel.

Note: The `cellsdk uninstall` command does not automatically uninstall the kernel to avoid leaving the system in an unusable state.

The patches for the 2.6.16 kernel are available on the BSC Web site directory and are distributed under the GPL license.

Linux support libraries

The following support libraries are provided by the Cell BE SDK to aid development and performance test of Cell BE applications.

SPE runtime management library

The SPE runtime management library (`LIBSPE`) contains an SPE thread programming model for Cell BE applications. The `ELFSPE` enables direct SPE executable execution from a Linux shell without the need of a PPE application creating an SPE thread.

For BladeCenter QS20s the `LIBSPE` headers, libraries and binaries are installed by the SDK into the `/usr` directory and the standalone SPE executive, `ELFSPE`, is registered during system root by commands added to `/etc/rc.d/init.d`.

For the simulator the `LIBSPE` and `ELFSPE` binaries and libraries are preinstalled in the same directories in the system root image and no further action is required at install time.

The source for the SPE runtime management library is available on the BSC Web site and is distributed under the GPL license.

numactl

numactl is used to control NUMA policy for processes or shared memory. An application can take advantage of this facility by binding a region of virtual storage to a specific bank of memory as well as binding a process to a specific NUMA node.

Note: The standard Fedora Core 5 version of `numactl` does not work correctly with the Cell BE processor. Instead the latest version of the numactl-0.9.8 for Fedora Core 6 is installed by the SDK on BladeCenter QS20s. The Fedora Core 6 version of `numactl` is available on the BSC Web site and distributed under the GPL license.

Libraries and samples

The libraries and samples RPM package provides a rich set of optimized standard SPE C library routines that greatly reduce the development cost and enhance the performance of SPU programs. A variety of application-oriented libraries, including Fast Fourier Transform (FFT), image, audio resample, math, game math, intrinsics, matrix operation, multi-precision math, noise generation, oscillator, surface, synchronization, and vector are also included in order to demonstrate the versatility of Cell BE architecture. Additional samples and workloads demonstrate how a programmer can exploit the on-chip computational capacity. Included is a large FFT workload that showcases performance that is more than an order of magnitude higher than a traditional processor.

The libraries and sample sources are installed in the `/opt/IBM/cell-sdk-1.1` directory under the IBM CPL license.

Libraries and samples subdirectories

The libraries and samples RPM has been partitioned into the following subdirectories.

Table 4. Subdirectories for the libraries and samples RPM

Subdirectory	Description
bin	Executables directory containing the SPU Timing tool and the ILAR license for this tool.
docs	Contains documentation about libraries and tools such as IDL compiler.
host	Host system executables, headers and libraries for the IDL tool.
license	Contains the text for the CPL license.
src/include	System header files. These files are exported to the $SDKINC_<target> (where target is either **ppu** or **spu**) directory for general use throughout the SDK.
scr/lib	Series of libraries and reusable header files. These are exported to `$SDKLIB_<target>` or `$SDKINC_<target>` directories (respectively). Complete documentation for all library functions and available in `/opt/IBM/cell-sdk-1.1/docs/lib/libraries.pdf`.

Table 4. Subdirectories for the libraries and samples RPM (continued)

Subdirectory	Description
src/samples	The samples directory contains examples of Cell BE programming techniques. Each program shows a particular technique, or set of related techniques, in detail. You may review these programs when you want to perform a specific task, such as double-buffered DMA transfers to and from a program, performing local operations on an SPU, or provide access to main memory objects to SPU programs. Some subdirectories contain multiple programs. The sync subdirectory has examples of various synchronization techniques, including mutex operations and atomic operations. The Julia subdirectory, which is new for SDK 1.1, contains sample code that ray traces the quaternion Julia Set into a bitmap image – see Appendix A, “Quaternion Julia Set Ray-tracing sample,” on page 37 for more details. The spulet subdirectory of samples includes customized startup code to let simple programs be run on the SPU directly from a Linux command prompt. For instance, the hello example prints a traditional greeting. The spulet model is intended to encourage testing and refinement of programs that need to be ported to the SPUs; it also provides an easy way to build filters that take advantage of the huge computational capacity of the SPUs, while reading and writing standard input and output.
src/tests	The tests directory defines some regression tests for the system. These programs exercise key system components. For instance, the asm/verify coverage test contains a non-executable sample program for the SPU which exists only to be assembled; it is not actually meant to execute, but it validates the functionality of an SPU assembler. The events subdirectory provides a sample program showing the handling of user-defined events; this program helps verify the function of the simulator.
src/tools	The tools directory contains tools that are useful for software development such as the Interface Definition Language (IDL) compiler and the `callthru` program. The IDL compiler reads high-level interface specifications and generates PPU and SPU stub functions to allow PPU code to ″call″ a function that is implemented and runs on the SPU. This is most useful for cases where you have a computationally intensive task that is well suited to simply being handed off to an SPU for processing. The code this generates is informative about PPU/SPU communications, but is primarily intended as a prototyping tool rather than a learning tool. The IDL tool has its own samples which show how to offload some processing to the SPU. The IDL’s PPU stub code supports dynamic allocation of multiple SPUs to handle simultaneous offloaded functions, and multiple functions can be loaded on a single SPU, if they are small enough. Some features are still under development such as double buffering support.
src/workloads	The workloads directory provides a handful of examples that can be used to better understand the performance characteristics of the Cell BE Processor. There are four sample programs, which contain insights into how real-world code should run. **Note:** Running these examples using the simulator takes much longer than on the native Cell BE-based hardware. The performance characteristics in wall-clock time using the simulator are extremely inaccurate, especially when running on multiple SPUs. Instead the emulator CPU cycle counts need to be examined. For example, the `matrix_mul` program lets you perform matrix multiplications on one or more SPUs. Matrix multiplication is a good example of a function which the SPUs can accelerate dramatically. Unlike some of the other sample programs, these examples have been hand-tuned to get the best performance. This makes them harder to read and understand, but it gives an idea for the kind of performance code that can be written for the Cell BE processor.
sysroot	Contains some of the headers and libraries used during cross-compiling and contains the compiled results of the libraries and samples. This can be synched up with the system root image by using the command: `/opt/IBM/cell-sdk-1.1/cellsdk synch`

SPU timing tool

The SPU static timing tool, `spu_timing`, annotates an SPU assembly file with scheduling, timing, and instruction issue estimates assuming straight, linear execution of the program. The tool generates a textual output of the execution pipeline of the SPE instruction stream from this input assembly file. Execute `spu_timing –help` to see its usage syntax.

The SPU timing tool is distributed as an RPM under the IBM ILAR license and is located in the `/opt/IBM/cell-sdk-1.1/bin` directory.

Note: The source code is not available.

Chapter 5. Using the SDK

This section is a short introduction into using the SDK. Refer to the programming tutorial, the Full-System Simulator user's guide, and other documentation for more details.

Running the Full-System Simulator

To verify that the Full-System Simulator (simulator) is operating correctly and then run it, issue the following commands:

```
# mkdir sandbox
# cd sandbox
# cp /opt/IBM/systemsim-cell/run/cell/linux/.systemsim.tcl .
# export PATH=/opt/IBM/systemsim-cell/bin:$PATH
# systemsim -g
```

The `.systemsim.tcl` file is a startup script for the simulator that configures the system and prepares it to boot and run the Linux operating system.

The `systemsim` script found in the simulator's bin directory launches the simulator and the `–g` parameter starts the graphical user interface.

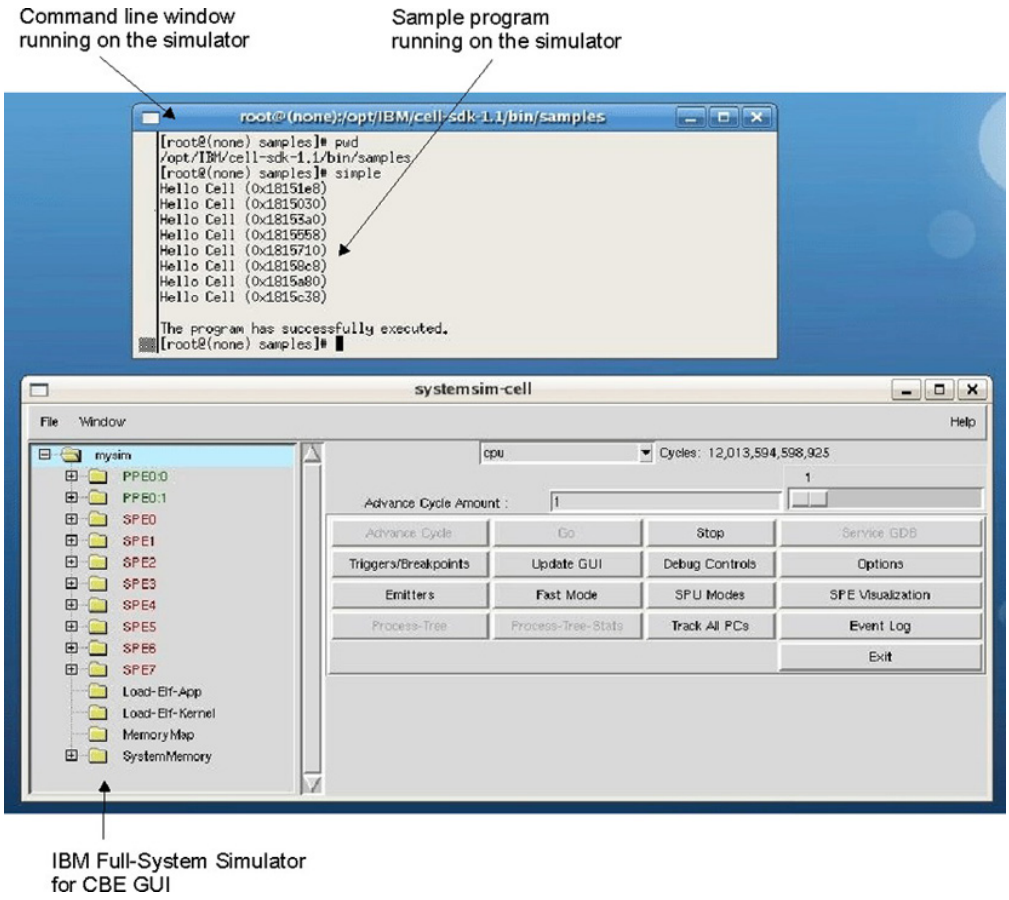


Figure 3. Running the simulator

Notes:

1. You must be on a graphical console, or at least have the `DISPLAY` environment variable pointed to a friendly X server to run the simulator's graphical user interface.
2. If an error message about `libtk8.4.so` is displayed, you must load the TK package as described in “TK” on page 15

When the GUI is displayed, click **Go** to start the simulator.

Note: To make the simulator run in fast mode, you can click **Fast Mode** and then **Go**. This forces the simulator to bypass its standard analysis and statistics collection features. Fast mode is useful if you want to advance the simulator through setup or initialization functions that are not the focus of analysis, such as, the Linux boot processing. You should disable fast mode when you reach the point at which you wish to do detailed analysis or debug the application.

You can use the simulator's GUI to get a better understanding of the Cell BE architecture. For example, the simulator shows two sets of PPE registers. This is because the PPE processor core is dual-threaded and each thread has its own registers and context. You can also look at the 128-bit register arrays provided on the SPEs.

The syntax of the `systemsim` command is:

```
systemsim [-f file] [-g] [-n]
```

where:

Parameter	Description
-f <filename>	specifies an initial run script (TCL file)
-g	specifies GUI mode, otherwise the simulator starts in command-line mode
-n	specifies that the simulator should not open a separate console window

You can find documentation about the simulator including a user’s guide in the `/opt/IBM/systemsim-cell/doc` directory.

The callthru utility

The `callthru` utility allows you to copy files to or from the simulated system while it is running. The utility is located in the simulator system root image in the `/usr/bin` directory.

If you call the utility as:

- `callthru sink <filename>`, it writes its standard input into `<filename>` on the host system
- `callthru source <filename>`, it writes the contents of `<filename>` on the host system to standard output.

Redirecting appropriately lets you copy files to and from the host. For example, when running on the host, you could copy a Cell BE application into `/tmp`:

```
cp matrix_mul /tmp
```

Then, in the console window of the simulated system, you could access it like this:

```
callthru source /tmp/matrix_mul > matrix_mul
chmod +x matrix_mul
./matrix_mul
```

Note: The use of the **>** redirect. If you omit **>** (callthru source) and **<** (callthru sink) this can cause problems when you are debugging.

The `/tmp` directory is shown as an example only.

The source files for the `callthru` utility are in `/opt/IBM/systemsim-cell/sample/callthru` and the `Makefile` to build the utility is in `/opt/IBM/cell-sdk-1.1/src/tools/callthru`.

Read and write access to the simulator sysroot image

By default the simulator does not write changes back to the simulator system root (sysroot) image, this means that the simulator always begins in the same initial state of the sysroot image. As you become more experienced, you can overwrite this default behavior so that any file changes made by the simulated system to the sysroot image apply to subsequent simulator sessions.

To specify that you want to use changes you have made to the sysroot image, change the `newcow` parameter on the `mysim bogus disk init` command in `.systemsim.tcl` to `rw` (specifying read/write access) and remove the last two parameters. Here is the changed line from `.systemsim.tcl`:

```
mysim bogus disk init 0 $sysrootfile rw
```

Enabling Symmetric Multiprocessing Support

By default the simulator provides an environment that simulates one Cell BE processor. To simulate a blade environment where two Cell BE processors exist, you must enable Symmetric Multiprocessing (SMP) support. To do this, edit the startup script that your simulator is using, which by default is `.systemsim.tcl`. Add these three lines to the start of the file:

```
proc config_hook { conf } {
    config_dual_be $conf
}
```

When the simulator is started, it has access to 16 SPEs across two Cell BE processors.

SDK programming samples

Each of the samples has an associated `README.txt` file. There is also a top-level `readme` in the `/src` directory, which introduces the structure of the sample code source tree. There are a number of PDF documents in the `/opt/IBM/cell-sdk-1.1/docs` directory including a programming tutorial.

Almost all of the samples run both within the simulator and on the BladeCenter QS20. Some samples include SPU-only programs that can be run on the system simulator in standalone mode.

Code, which is specific to a given Cell BE processor unit type, is in the corresponding subdirectory within a given sample's directory:

- `ppu` for code compiled to run on the PPE
- `spu` for code compiled to run on an SPE

- `spu_sim` for code compiled to run on an SPE under the system simulator in standalone environment natively on a BladeCenter QS20

Changing the default compiler

In `/opt/IBM/cell-sdk-1.1` there are some top level makefiles that control the build environment for all of the samples. Most of the directories in the libraries and samples contain a makefile for that directory and everything below it. All of the samples have their own makefile but the common definitions are in the top level makefiles.

The build environment makefile are documented in `/opt/IBM/cell-sdk-1.1/README_build_env.txt`.

Environment variables in the `/opt/IBM/cell-sdk-1.1/make.env` makefile are used to determine which compiler is used to build the samples.

The cellsdk script contains a task which automatically switches the compiler, does a make clean and then a make which rebuilds all of the samples and libraries. The syntax of this command is:

```
./cellsdk build [-x | -g ]
```

where the `–x` flag selects the XL C/C++ compiler (x86 only) and the `–g` flag selects the GCC compiler.

After you have selected a particular compiler, that same compiler is used for all future builds. By default the GCC compiler is selected for compiling the samples. For x86 hosts, you can overwrite this default at install time by using the `–x` option with the `cellsdk install` command.

Building and running a specific program

You do not need to build all the sample code at once; you can build each program separately. To start from scratch, issue a `make clean` using the makefile in the `/opt/IBM/cell-sdk-1.1/src` directory or anywhere in the path to a specific library or sample.

If you have performed a `make clean` at the top level, you need to rebuild the include files and libraries first before you compile anything else. To do this run a `make` in the `src/include` and `src/lib` directories.

For example, build the `hello` spulet. The spulet examples require a special version of `crt0`, built from `src/samples/spulet/crt0`. After the include files and libraries are built, change directory to `src/samples/spulet/hello`, and run `make`.

System root directory

When you build the libraries and samples output files are copied into a special directory named `/opt/IBM/cell-sdk-1.1/sysroot`. This directory has a very similar structure to a normal system root directory (that is, /) and contains the usual subdirectories such as `/bin`, `/usr` and `/etc`.

On a BladeCenter QS20 the samples can be run directly from the subdirectories under the `/opt/IBM/cell-sdk-1.1/sysroot` directory.

After you have logged in as `root`, you can synchronize this `sysroot` directory with the simulator sysroot image file. To do this, use the cellsdk script with the synch task. The command is:

```
./cellsdk synch
```

This command is very useful whenever a library or sample has been recompiled. This script reduces user error because it provides a standard mechanism to mount the system root image, rsync the contents of the two corresponding directories, and unmount the system root image.

Support for huge TLB file systems

The SDK supports the huge translation lookaside buffer (TLB) file system, which allows you to reserve large pages of pinned, contiguous memory. This feature is particularly useful for some Cell BE applications that operate on large data sets such as the FFT16M workload sample and Cell BE Terrain Rendering Engine.

To configure the Cell BE-based blade server for 20 pages (320 MB), run the following commands:

```
mkdir -p /huge
echo 20 > /proc/sys/vm/nr_hugepages
mount -t hugetlbfs nodev /huge
```

If you have difficulties configuring adequate huge pages, it could be that the memory is fragmented and you need to reboot. You can add the command sequence shown above to a startup initialization script, such as `/etc/rc.d/rc.sysinit`, so that the large TLB file system is configured during the system boot.

To verify the large memory allocation, run the command `cat /proc/meminfo`. The output is similar to:

```
MemTotal:      1010168 kB
MemFree:        155276 kB
. . .

HugePages_Total:    20
HugePages_Free:     20
Hugepagesize:    16384 kB
```

SDK development best practices

This section documents some best practices in terms of developing applications using the Cell BE SDK. See also developerWorks articles about programming tips and best practices for writing Cell BE applications at http://www.ibm.com/developerworks/power/cell/.

Developing applications with a user (non-root) account

The libraries and sample programs are installed in `/opt/IBM/cell-sdk-1.1/src`. To develop applications using your regular user account rather than root, change the ownership of the `/opt/IBM/cell-sdk-1.1` and `/opt/IBM/systemsim-cell` directories to your regular user account. To do this, issue the following command:

```
chown -R $USER /opt/IBM/cell-sdk-1.1 /opt/IBM/systemsim-cell
```

You can do this as soon as the cellsdk script has finished building the environment. Then log on with your user account.

Note: You still need root access to synch with the simulator sysroot image, because it must be mounted and unmounted. You should not leave this file permanently mounted as it can interfere with the running of the simulator and changes are not stored in the file until the simulator's sysroot image is unmounted.

Using a shared development environment

If multiple people are using the same machine, it is advisable to set up different sandboxes for each user. The simplest way to do this is to copy the `/opt/IBM/cell-sdk-1.1` directory to each user's home directory. Even if you are the only user, it is advisable to set up one or more sandboxes so you can experiment with the SDK samples and libraries.

Multiple users should not update the common simulator sysroot image file by mounting it read-write in the simulator. In this case, the `callthru` utility (see "The callthru utility" on page 32) can be used to get files in and out of the simulator. Alternatively, users can copy the sysroot image file to their own sandbox area and then mount this version read/write to make persistent updates to the image.

If multiple users need to run Cell applications on a BladeCenter QS20, you need a machine reservation mechanism to reduce collisions between two people who are using SPEs at the same time. This is because SPE threads are not fully pre-emptable in this version of the SDK.

Restrictions and limitations

This section documents some minor restrictions and limitations in the SDK.

- XDR memory should be initialized by the application before it is used
- The function `spe_get_context` has been implemented
- The function `spe_set_context` has not been implemented
- There is limited checking on exceeding the 256K memory limitation for a SPU so a message from the spu-gcc compiler such as:

  ```
  relocation truncated to fit: SPU_ADDR16 against `.bss'
  ```

  means that you have probably need to reduce the memory usage in the SPU.

Appendix A. Quaternion Julia Set Ray-tracing sample

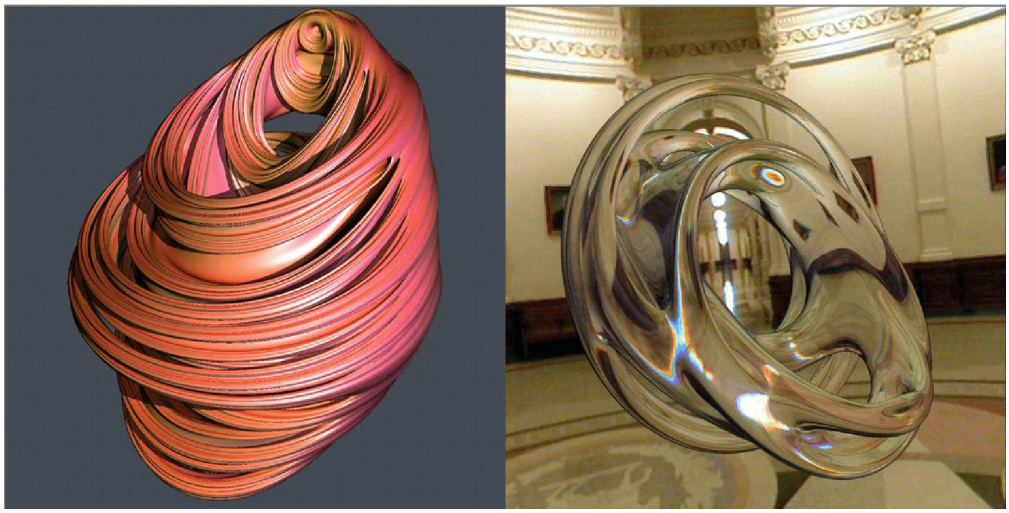
The description and pictures in this chapter replaces the text provided with the Julia sample, which can be found in `/opt/IBM/cell-sdk-1.1/src/samples/julia` directory.

This sample was inspired by Keenan Crane's work to implement Julia Set ray-tracers on GPUs using NVIDIA's Cg language. Procedurally generated surfaces are a hot topic relative to the Cell BE Processor due to their dynamic resolution independent nature, high computational intensity, and low memory foot print. This sample attempts to preserve Keenan's Cg coding style and shows the flexibility of Cell BE programming models.

The Julia sample demonstrates several Cell BE software technologies including a SPE centric load balancing framework first written for our Terrain Rendering Engine (TRE) and a software cache optimized for Single Instruction Multiple Data (SIMD) texture performance. The sample also provides a demonstration platform for the advantage of structure of array (SOA) over array of structures (AOS) data organization when running on SIMD architectures.

Algorithm overview

You can find the original ray-tracing algorithm for the Julia set in the paper *Ray Tracing Deterministic 3-D Fractals*, J. Hart, D. Sandin and L. Kauffman, _SIGGRAPH_, 1989, pp. 289-296. Keenan Crane's Cg code base was used as the starting point for this sample.



Converting Cg to cell code

The code port consists of the following steps:

1. Convert from Cg style infix code to SPE intrinsic style prefix code. This step could easily be performed using a simple code translation tool.
2. Implement a framework that could execute these Cg shader kernels. SPE Quad interpolators were written to break down the screen regions into initial rays on the front end and a frame buffer data structure was created to store the resulting fragments on the back end. Work is communicated to the SPE in the

form of screen regions which are vertical screen strips. Each SPE is assigned a different screen region.

Figure 4. Cg to cell code conversion

3. Each screen region is broken down into rays and processed by the shader kernels.
4. Completed image fragments are gathered in temporary local store vertical column buffers and then written out to the system memory frame buffer as they are completed via DMAs.

Load balancing framework

Screen regions are computed and dispatched to the SPEs using an SPE centric load balancing framework. In this framework compute tasks are prepared by the PPE and processed by multiple SPEs in a data parallel fashion. The PPE only manages the preparation and synchronization of the work at a very high level. All data loading, processing, and storing is handled by the heaving lifting SPEs. Using this model one PPE is able to manage many SPEs without becoming the bottleneck. As more SPEs are added performance continues to scale linearly.

The framework provides a simple command/data interface to the SPEs using PPE to SPE mailboxes. Supported commands include the loading of a rendering context which contains attributes that are constant across all frames of an animation, loading of a work block communicated in the form of a screen region and surface basis, SPE flush which forces all finished fragments to be written to system memory, and SPE exit or release.

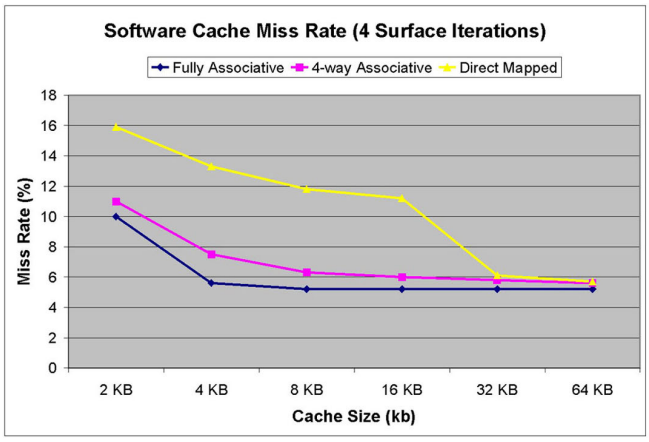
The PPE performs work load balanced across multiple SPE by adjusting the size of each screen region based on the finish order of the prior screen regions. A small percentage of the slowest SPE's work is transferred to the fastest SPE after each frame is completed. By implementing this dynamic scheme complex performance issues like local and remote memory, object screen space distribution, and variable super sampling just balance themselves out. SPE can also be added and remove dynamically by simply adding or removing them from the available pool and resetting the work weights. This framework has been reused in several Cell

applications. It can easily be adapted to many rendering and imaging workloads providing a good starting point for SPE centric coding.

Software cache

The Julia sample leverages a software cache abstraction layer for the SPE giving us the ability to both hide the complexity of DMAs and benefit from transparent data reuse. Given the lessons learned from this paper on “The Design and Analysis of a Cache Architecture for Texture Mapping”, we tiled our textures, optimized our access patterns, and implemented several cache replacement policies. We then rewrote the shader to add five cubemap texture lookup passes - 3 refraction lookups, a reflection lookup, plus a background lookup. These five texture lookups were then blended together with a fresnel calculation and modulated with the base lighting computation to form the final sample color.

We found that even with small 4-way set associative software cache sizes (8 KB), miss rates for this renderer were a low 7% and hit access times were only 12 SPE cycles.



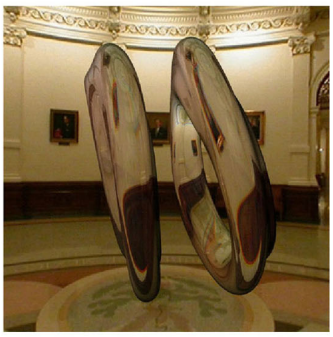
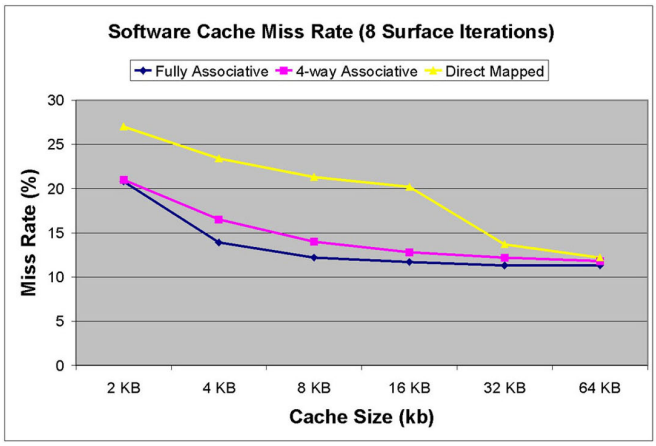


Figure 5. Software cache miss rate (4 surface iterations)



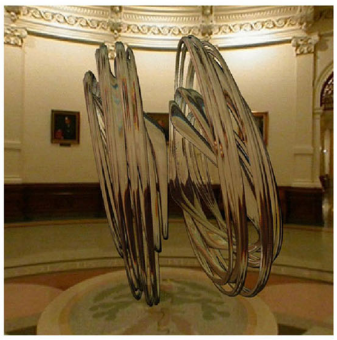


Figure 6. Software cache miss rate (8 surface iterations)

In the Julia sample you will find the code for a 4-way set associative SPE cache along with a set of 16x16 tiled textures. These textures are loaded and repacked into a 565 16 bit per texel format in system memory. The cache abstraction

provides lookups in both a single effective address (EA) form and a four EA SIMD form. In the SIMD cache lookup, we employ a novel technique to avoid cast outs due to collisions. When any element in the SIMD lookup misses, a bitmask of available slots is generated.

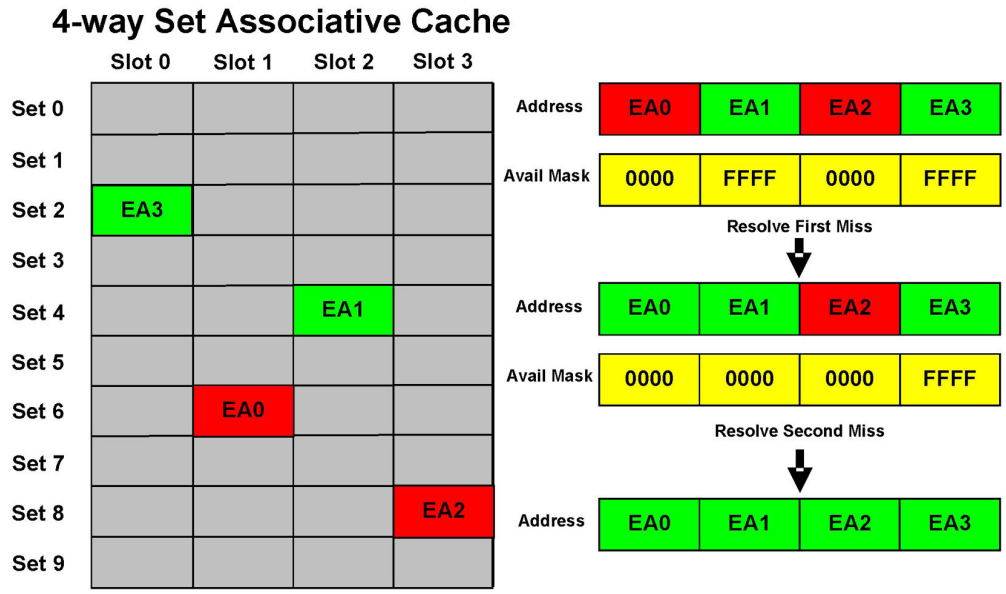


Figure 7. 4-way set associative cache

Slots that are found to contain hits (green) are first removed from the available mask. Next slots are allocated for misses (red) using the available mask and a revolving starting slot. The first available slot in the available mask is selected and the corresponding slot of the cache set is allocated to this EA. This slot is then removed from the available mask and the next miss is processed. All misses are issued under the same tag group allowing for one tag wait per group of four thereby reducing the load latency to the length of just one DMA regardless of the number of misses.

SOA verses AOS data structures

The way CG structures its SIMD computation is inefficient as it causes large percentages of the code to execute in scalar mode. This is due to the way they structure their vector data, AOS vs. SOA. By converting the shader from AOS to SOA form, SIMD utilization greatly increases allowing for much higher performance. Analysis of IBM XLC compiled SPE code shows the cycles per instruction (CPI) drops from 1.68 to 1.20 using this transformation and the number of SPE cycles required to compute an image are cut in half. Both forms of the shader are made available in the sample with the exception of the texture shader which is only available in the SOA form.

Appendix B. Notices

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

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

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

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: THIS INFORMATION IS PROVIDED "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. The manufacturer 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 Web sites not owned by the manufacturer 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 product and use of those Web sites is at your own risk.

The manufacturer 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 the manufacturer.

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, IBM License Agreement for Machine Code, or any equivalent agreement between us.

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

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

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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to the manufacturer, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. The manufacturer, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

SUBJECT TO ANY STATUTORY WARRANTIES WHICH CAN NOT BE EXCLUDED, THE MANUFACTURER, ITS PROGRAM DEVELOPERS AND SUPPLIERS MAKE NO WARRANTIES OR CONDITIONS EITHER EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OR CONDITIONS OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT, REGARDING THE PROGRAM OR TECHNICAL SUPPORT, IF ANY.

UNDER NO CIRCUMSTANCES IS THE MANUFACTURER, ITS PROGRAM DEVELOPERS OR SUPPLIERS LIABLE FOR ANY OF THE FOLLOWING, EVEN IF INFORMED OF THEIR POSSIBILITY:

1. LOSS OF, OR DAMAGE TO, DATA;
2. SPECIAL, INCIDENTAL, OR INDIRECT DAMAGES, OR FOR ANY ECONOMIC CONSEQUENTIAL DAMAGES; OR
3. LOST PROFITS, BUSINESS, REVENUE, GOODWILL, OR ANTICIPATED SAVINGS. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU.

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

Edition notice

U.S. Government Users Restricted Rights — Use, duplication, or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

BladeCenter®
e-business logo
@server
IBM
IBM (logo)
IntelliStation®
POWER™
Power Architecture™
PowerPC®
Predictive Failure Analysis®
pSeries®
ServerProven®
xSeries®

Intel®, MMX, and Pentium® are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft®, Windows®, and Windows NT® are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX® is a registered trademark of The Open Group in the United States and other countries.

Java™ and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Adaptec and HostRAID are trademarks of Adaptec, Inc., in the United States, other countries, or both.

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

Red Hat, the Red Hat “Shadow Man” logo, and all Red Hat-based trademarks and logos are trademarks or registered trademarks of Red Hat, Inc., in the United States and other countries.

InfiniBand is a trademark of the InfiniBand Trade Association.

XDR is a trademark of Rambus Inc. in the United States and other countries.

PCI Express is a trademark of PCI-SIG.

Cell Broadband Engine is a trademark of Sony Computer Entertainment Inc.

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

Glossary

This glossary contains terms and abbreviations used in Cell BE systems.

anaconda. The main installation program for Linux Fedora Core.

AOS. Array of structures. A method of organizing related data values. Also called vector-across form. See *SOA*.

atomic operation. A set of operations, such as read-write, that are performed as an uninterrupted unit.

Barcelona Supercomputing Center. Spanish National Supercomputing Center, supporting Bladecenter and Linux on cell.

BSC. See *Barcelona Supercomputing Center*.

BE. Broadband Engine.

BOOTP. Bootstrap Protocol. A UDP network protocol used by a network client to obtain its IP address automatically. Replaced in many networks by DHCP.

Broadband Engine. See *CBEA*.

built-ins. A type of C and C++ programming language intrinsic that is similar to generic intrinsics, except that built-ins map to more than one SPU instruction. These intrinsics are prefaced by `spu_`.

C++. Deriving from C, C++ is an object-orientated programming language.

cache. High-speed memory close to a processor. A cache usually contains recently-accessed data or instructions, but certain cache-control instructions can lock, evict, or otherwise modify the caching of data or instructions.

CBEA. Cell Broadband Processor 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 BE. Cell Broadband Engine. See *CBEA*.

Cell Broadband Engine Linux application. An application running on the PPE and SPE. Each such application has one or more Linux threads and some number of SPE threads. All the Linux threads within the application share the application's resources, including access to the SPE threads.

Cell Broadband Engine program. A PPE program with one or more embedded SPE programs.

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

CPI. Cycles per instruction. Average number of clock cycles taken to perform one CPU instruction.

CPL. Common Public License.

Cycle-accurate simulation. See *Performance simulation*.

cycle. Unless otherwise specified, one tick of the PPE clock.

DHCP. Dynamic Host Configuration Protocol. Similar to BOOTP, DHCP is a protocol for assigning IP addresses to client devices on a network.

DMA. Direct Memory Access. A technique for using a special-purpose controller to generate the source and destination addresses for a memory or I/O transfer.

EA. See *Effective address*.

effective address. An address generated or used by a program to reference memory. A memory-management unit translates an effective address (EA) to a virtual address (VA), which it then translates to a real address (RA) that accesses real (physical) memory. The maximum size of the effective address space is 2^{64} bytes.

ELF. Executable and Linking Format. The standard object format for many UNIX operating systems, including Linux. Originally defined by AT&T and placed in public domain. Compilers generate ELF files. Linkers link to files with ELF files in libraries. Systems run ELF files.

ext3. Extended file system 3. One of the file system options available for Linux partitions.

Fedora Core. An operating system built entirely from open-source software and therefore freely available. Often, but mistakenly, known as Fedora linux.

FFT. Fast Fourier Transform.

FIFO. First In First Out. Queue type in which elements are processed in order of entry. See also *LIFO*.

firmware. A set of instructions contained in ROM usually used to enable peripheral devices at boot.

FSF. Free Software Foundation. Organization promoting the use of open-source software such as Linux.

FSS. Full System Simulator. IBM's tool which simulates the cell processor environment on other host computers.

FTP. File Transfer Protocol. An application layer protocol that uses the TCP/IP suite of services to transfer bulk-data files between machines or hosts.

GCC. GNU C compiler.

gdb. GNU application debugger. A modified version of `gdb`, `ppu-gdb`, can be used to debug a Cell Broadband Engine program. The PPE component runs first and uses system calls, hidden by the SPU programming library, to move the SPU component of the Cell Broadband Engine program into the local store of the SPU and start it running. A modified version of gdb, spu-gdb, can be used to debug code executing on SPEs.

GPL. GNU General Public License. Guarantees freedom to share, change and distribute free software.

GNU. GNU is Not Unix. A project to develop free Unix-like operating systems such as Linux.

HTTP. Hypertext Transfer Protocol. A method used to transfer or convey information on the World Wide Web.

I/O device. Input/output device. From the viewpoint of software I/O devices exist as memory-mapped registers that are accessed in main-storage space by load/store instructions. .

IDL. Interface definition language. Not the same as CORBA IDL

ILAR. IBM International License Agreement for early release of programs.

initrd. A command file read at boot.

intrinsic. A C-language command, in the form of a function call, that is a convenient substitute for one or more inline assembly-language instructions. Intrinsics make the underlying ISA accessible from the C and C++ programming languages.

ISO image . Commonly a disk image which can be burnt to CD. Technically it is a disk image of and ISO 9660 file system.

Julia set. Fractal shapes defined on a complex number plane

kernel. The core of an operating which provides services for other parts of the operating system and provides multitasking. In Linux or UNIX operating system, the kernel can easily be rebuilt to incorporate enhancements which then become operating-system wide.

K&R programming. A reference to a well-known book on programming written by Dennis Kernighan and Brian Ritchie.

L1. Level-1 cache memory. The closest cache to a processor, measured in access time.

L2. Level-2 cache memory. The second-closest cache to a processor, measured in access time. A L2 cache is typically larger than a L1 cache.

latency. The time between when a function (or instruction) is called and when it returns. Programmers often optimize code so that functions return as quickly as possible; this is referred to as the low-latency approach to optimization. Low-latency designs often leave the processor data-starved, and performance can suffer.

libspe. A SPU-thread runtime management library.

LIFO. Last In First Out. Queue type in which elements are processed in reverse order of entry. See also *FIFO*.

Linux. An open-source Unix-like computer operating system.

LGPL. Lesser General Public License. Similar to the *GPL*, but does less to protect the user's freedom.

local store. The 256-KB local store associated with each SPE. It holds both instructions and data.

LS. See *local store*.

LVM. Logical volume manager. An abstraction of a physical hard disk which allows the changing the size of a volume without having to reboot the system.

LVM partition. A portion of a physical drive managed by the LVM.

main memory. See *main storage*.

main storage. The effective-address (EA) space. It consists physically of real memory (whatever is external to the memory-interface controller, including both volatile and nonvolatile memory), SPU LSs, memory-mapped registers and arrays, memory-mapped I/O devices (all I/O is memory-mapped), and pages of virtual memory that reside on disk. It does not include caches or execution-unit register files. See also *local store*.

makefile. A descriptive file used by the `make`command in which the user specifies: (a) target program or library, (b) rules about how the target is to be built, (c) dependencies which, if updated, require that the target be rebuilt.

MFC. Memory Flow Controller. Part of an SPE which provides two main functions: it moves data via DMA between the SPE's local store (LS) and main storage, and it synchronizes the SPU with the rest of the processing units in the system.

netboot. Command to boot a device from another on the same network. Requires a TFTP server.

NUMA. Non-uniform memory access. In a multiprocessing system such as the Cell BE, memory is configured so that it can be shared locally, thus giving performance benefits.

parallel array form. See *SOA*.

PDF. Portable document format.

Performance simulation. Simulation by the IBM Full System Simulator for the Cell Broadband Engine in which both the functional behavior of operations and the time required to perform the operations is simulated. Also called cycle-accurate simulation.

PERL. Practical extraction and reporting language. A scripting programming language.

PowerPC. Of or relating to the *PowerPC Architecture* or the microprocessors that implement this architecture.

PPC-64. 64 bit implementation of the *PowerPC Architecture*.

PowerPC Architecture. A computer architecture that is based on the third generation of RISC processors. The PowerPC architecture was developed jointly by Apple, Motorola, and IBM.

PPC. See *Power PC*.

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

PPSS. PowerPC Processor Storage Subsystem. Part of the *PPE*. It operates at half the frequency of the *PPU* and includes a L2 cache and a Bus Interface Unit (BIU).

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

proxy. Allows many network devices to connect to the internet using a single IP address. Usually a single server, often acting as a firewall, connects to the internet behind which other network devices connect using the IP address of that server.

python. An open-source interpretative scripting language.

RA. See *real address*.

real address. An address for physical storage, which includes physical memory, the PPE's L1 and L2 caches, and the SPE's local stores (LSs) if the operating system has mapped the LSs to the real address space. The maximum size of the real address space is 2^{42} bytes.

RPM. Originally an acronym for Red Hat Package Manager, and RPM file is a packaging format for one or more files used by many Linux systems when installing software programs.

Sandbox. Safe place for running programs or script without affecting other users or programs.

scalar. An instruction operand consisting of a single value.

SDK. Software development toolkit. A complete package of tools for application development. The Cell BE SDK includes sample software for the Cell Broadband Engine.

SIMD. Single Instruction Multiple Data. Processing in which a single instruction operates on multiple data elements that make up a vector data-type. Also known as vector processing. This style of programming implements data-level parallelism.

SIMDize. To transform scaler code to vector code.

SOA. Structure of arrays. A method of organizing related data values. Also called parallel-array form. See also *AOS*.

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.

SPE thread. A thread scheduled and run on a SPE. A program has one or more SPE threads. Each such thread has its own SPU local store (LS), 128 x 128-bit register file, program counter, and MFC Command Queues, and it can communicate with other execution units (or with effective-address memory through the MFC channel interface).

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

spulet. 1) A standalone SPU program that is managed by a PPE executive. 2) A programming model that allows legacy C programs to be compiled and run on an SPE directly from the Linux command prompt.

tag group. A group of DMA commands. Each DMA command is tagged with a 5-bit tag group identifier. Software can use this identifier to check or wait on the completion of all queued commands in one or more tag groups. All DMA commands except `getllar`, `putllc`, and `putlluc` are associated with a tag group.

Tcl. Tool Command Language. An interpreted script language used to develop GUIs, application prototypes,

Common Gateway Interface (CGI) scripts, and other scripts. Used as the command language for the Full System Simulator.

TFTP. Trivial File Transfer Protocol. Similar to, but simpler than the Transfer Protocol (FTP) but less capable. Uses UDP as its transport mechanism.

thread. A sequence of instructions executed within the global context (shared memory space and other global resources) of a process that has created (spawned) the thread. Multiple threads (including multiple instances of the same sequence of instructions) can run simultaneously if each thread has its own architectural state (registers, program counter, flags, and other program-visible state). Each SPE can support only a single thread at any one time. Multiple SPEs can simultaneously support multiple threads. The PPE supports two threads at any one time, without the need for software to create the threads. It does this by duplicating the architectural state.

TLB. Translation Lookaside Buffer. An on-chip cache that translates virtual addresses (VAs) to real addresses (RAs). A TLB caches page-table entries for the most recently accessed pages, thereby eliminating the necessity to access the page table from memory during load/store operations.

TRE. Terrain Rendering Engine. An important early demo of cell technology.

UDP. User Datagram Protocol. Transports data as a connectionless protocol, i.e. without acknowledgement or receipt. Fast but fragile.

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.

Vector/SIMD Multimedia Extension. The SIMD instruction set of the PowerPC Architecture, supported on the PPE.

virtual memory. The address space created using the memory management facilities of a processor.

virtual storage. See *virtual memory*.

VNC. Virtual Network Computing. A desktop sharing system which uses the RFB (Remote FrameBuffer) protocol to control another computer remotely.

VPD. Vital product data. Part of the firmware, a ROM chip containing information about the system board and/or other components. May be updatable via firmware updates.

workload. A set of code samples in the SDK that characterizes the performance of the architecture, algorithms, libraries, tools, and compilers.

XDR. Rambus ® Extreme Data Rate DRAM memory technology.

xlc. The IBM optimizing C/C++ compiler.

x86. Generic name for Intel-based processors.

yaboot. Linux utility which is a boot loader for PowerPC-based hardware.

yum. Yellow dog Updater, Modified. A package manager for RPM-compatible Linux systems.

Index

A

Anaconda 6

B

best practices 35
booting 13
  after installation 12

C

callthru utility 32
Cell BE processor
  documentation 3
Cell BE programming
  documentation 4
cellsdk 19
  build task 20
  depend task 20
  displaying usage statement 20
  install task 20
  parameters 20
  synch task 20
  uninstall task 20
  upgrade task 20
  verify task 20
  version task 20
compiler
  changing 34
  gcc 14
configuring
  sample yum configuration 13
  swap area 13
  yaboot.conf 12

D

directory structure
  libraries and samples 27
  programming sample 33
  system root 34
documentation
  Cell BE processor 3
  Cell BE programming 4
  PowerPC base 4
  simulator 4
download
  Fedora Core 5 tree 5
  file 5

E

environment
  for network installation 7

F

fast mode
  Full-System Simulator 32
Fedora Core 5
  downloading 5
  finishing the installation 14
  installation files 5
  installing 5
  installing on BladeCenter QS20 6
  installing on PPC64 6
  installing on x86 6
  with InfiniBand 12
file
  download 5
files
  Fedora Core 5 installation 5
firmware
  checking which version 13
freeglut 15
Full-System Simulator
  .systemsim.tcl 31
  See also PowerPC base
  callthru utility 32
  description 24
  documentation 4
  fast mode 32
  RPM 21
  running 31
  system root image 25, 33
  systemsim 31

G

gcc 14
GNU tool chain 23

H

hard disk
  custom layout 11
  partitioning 11
hardware
  ppc64bcmfix.img 6, 8

I

InfiniBand
  operating system 12
init process 6
installing
  different ways to install SDK 19
  Fedora Core 5 5
  freeglut-devel 15
  libX11 15
  make package 14
  netpbm 15
  netpbm-devel 15

installing *(continued)*
operating system 5
overview 6
perl package 14
replacing kernel 11
starting 8
tcl package 15
tk package 15
iso image 5

K

kernel 26
replacing 11
RPM 21
uninstalling 19
kickstart 6

L

libraries and samples 27
subdirectories 27
libX11 15
licenses 3
limitations 36

M

make 14
makefile
for samples 34

N

netboot
setting up installation environment 8
netpbm 15
netpbm-devel 15
network
installation environment 7
setting up installation environment 8
numactl 27

O

operating system
installing 5

P

perl 14
PowerPC base
documentation 4
ppuxlc 23
ppuxlc++ 23
prerequisites
hard disk space 3
RAM for the simulator 3
RAM on host 3
SDK 2
programming sample
building and running 34
compiler 34
directory structure 33
spulet 34

R

restarting
BladeCenter QS20 14
restrictions 36
RPM 21
Full-System Simulator 21
GNU tool chain for PPU 21
GNU tool chain for SPU 21
kernel 21
library SPE 21
NUMA 21
OProfile 21
Samples and Libraries 21
SPU timing 21
system root image 21
XL C/C++ compiler 21

S

sandbox
setting up 36
script 20
.systemsim.tcl 31
cellsdk 19
systemsim 31
SDK
components 17
how to use 2
installing 17
prerequisites 2
supported platforms 17
upgrading to SDK 1.1.1 18
shared development environment 36
shutting down
BladeCenter QS20 14
simulator
See fss
SPE runtime management library 26
spe_get_context 36
spe_set_context 36
SPU timing tool 29
spulet 34
spuxlc 23
support libraries 26
supported platforms 2
swap area 13
symmetric multiprocessing support 33
sysroot
Full-System Simulator 33
system root
directory 34
system root image 25

T

tk 15
TLB file system
  configuring 35
trademarks 44

U

uninstalling
  kernel 19
  SDK 19
upgrading
  SDK version 18
user account 35
utility
  callthru 32

X

XL C/C++ compiler 23
  commands 23
  optimization levels 24

Y

yaboot.conf 12
yum configuration 13

Part Number: 42C4915

Printed in USA

(1P) P/N: 42C4915

SC33-8323-00