Cell Programming link on sti.cc.gatech.edu

Installing the system takes up to 20 minutes.

Prerequisites

- Installed and working VMWare Player.
- About 8 gigabytes of free hard drive storage.
- At least 512 megabytes of system RAM.

Downloading the System

Cell SDK3.0: (last updated: 17 July 2008)

We recommend that you use the Torrent, but also make available zip files for download.

- BitTorrent: Cell-VMWare-F7-SDK30-3Mar2008.torrent
- HTTP: The files you need are:
  - Cell-VMWare-F7-SDK30-3Mar2008-Part1of5.zip
  - Cell-VMWare-F7-SDK30-3Mar2008-Part2of5.zip
  - Cell-VMWare-F7-SDK30-3Mar2008-Part3of5.zip
  - Cell-VMWare-F7-SDK30-3Mar2008-Part4of5.zip
  - Cell-VMWare-F7-SDK30-3Mar2008-Part5of5.zip

Image contains VI, Kate, Kedit, eclipse, but no emacs???
Running the VMWare workshop image

You can get the full version of VMWare (for Mac as well) from OIT.
Starting the simulator GUI

```
[root@localhost ~]# cd /opt/ibm/systemsim-cell/run/cell/linux
[root@localhost linux]# ../../../run_gui
```

![Simulator GUI interface](image)
Set everything to Fast
Starting the simulator

```
X
root@(none):~

Uniform Multi-Platform E-IDE driver Revision: 7.00alpha2
ide: Assuming 50MHz system bus speed for PIO modes; override with idebus=xx
mice: PS/2 mouse device common for all mice
usbcore: registered new interface driver hiddev
usbcore: registered new interface driver ushbid
drivers/hid/usbhid/hid-core.c: v2.6:USB HID core driver
TCP cubic registered
Initializing XFRM netlink socket
NET: Registered protocol family 1
NET: Registered protocol family 17
drivers/rtc/hctosys.c: unable to open rtc device (rtc0)
md: Autodetecting RAID arrays.
md: autorun ...
md: ... autorun DONE.
Initializing disk 0 with devsz 1843200
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 448k freed
INIT: version 2.86 booting
     Welcome to Fedora release 7 (Moonshine)
     Press 'I' to enter interactive startup.
eth0: bogus network driver initialization
No IRQ retrieved
INIT: Entering runlevel: 2
[root@(none) ~]
```
Running Simple Hello Worlds

```
[root@(none) ~]# callthru source /opt/cell_class/Hands-on-30/hello/hello_ppu/hello_ppu
[root@(none) ~]# chmod 755 hello_ppu
[root@(none) ~]# ./hello_ppu
Hello World!
[root@(none) ~]# callthru source /opt/cell_class/Hands-on-30/hello/hello_spu/hello_spu
[root@(none) ~]# chmod 775 hello_spu
[root@(none) ~]# hello_spu
Hello World!
[root@(none) ~]#
```
Running Synergistic Hello Worlds

```
[root@none ~]# callthru source /opt/cell_class/Hands-on-30/hello/hello_be1-syn
cc/hello_be1 > hello_be1
[root@none ~]# chmod 755 hello_be1
[root@none ~]# hello_be1
Hello World!
[root@none ~]# callthru source /opt/cell_class/Hands-on-30/hello/hello_be1-asyn
c/hello_be1 > hello_be1
[root@none ~]# chmod 755 hello_be1
[root@none ~]# hello_be1
Hello World!
[root@none ~]# 
```
CellBuzz Cluster link on sti.cc.gatech.edu

Fill out this today!

Read this!!!
Using CellBuzz

• `ssh` to `cell-user.cc.gt.atl.ga.us`
• Use `passwd` to change your password
• `cell-user` is not a Cell machine; do not try to run your Cell programs on it!
• Request interactive session: `qsub -I`
• Can `sftp` your executables to CellBuzz from the Fedora Core image
• See CellBuzz users guide about submitting batch jobs
hello_ppu

Makefile

PROGRAM_ppu = hello_ppu
include $(CELL TOP)/buildutils/make.footer

hello_ppu.c

#include <stdio.h>

int main()
{
  printf("Hello World!\n");
  return 0;
}
hello_spu

Makefile

PROGRAM_spu = hello_spu
SPU_TIMING=1
include $(CELL_TOP)/buildutils/make.footer

hello_spu.c

#include <stdio.h>

int main()
{
  printf("Hello World!\n");
  return 0;
}
hello_be1 – sync version

```
[root@localhost hello_be1-sync]# pwd
/opt/cell_class/Hands-on-30/hello/hello_be1-sync
[root@localhost hello_be1-sync]# ls
hello_be1 hello_be1.d Makefile
hello_be1.c hello_be1.o spu
[root@localhost hello_be1-sync]# ls spu
hello_spu hello_spu-embed.o Makefile
hello_spu.a hello_spu.o Makefile~
hello_spu.c hello_spu.s
hello_spu.d hello_spu.s.timing
[root@localhost hello_be1-sync]#  
```
Synergistic PPE and SPE

• Applications use software constructs called SPE contexts to manage and control SPEs.
• Linux schedules SPE contexts from all running applications onto the physical SPE resources in the system for execution according to the scheduling priorities and policies associated with the runnable SPE contexts.
• libspe provides the means for communication and data transfer between PPE threads and SPEs.

Slide from IBM Presentation, “Hands-on – The Hello World! Program”
How does a PPE program start an SPE thread?

4 basic steps must be done by the PPE program:

• Create an SPE context
• Load an SPE executable object into the SPE context local store
• Run the SPE context
  – This transfers control to the operating system, which requests the actual scheduling of the context onto a physical SPE in the system.
• Destroy the SPE context.

Slide from IBM Presentation, “Hands-on – The Hello World! Program”
hello_be1 – sync version – spu code

Makefile

PROGRAM_spu .  = hello_spu
LIBRARY_embed   = hello_spu.a

#CPPFLAGS_gcc  = -S
SPU_TIMING      = 1
#CC_OPT_LEVEL   = -04

#include $(TOP)/buildutils/make.footer

hello_spu.c (for the hello_be1 example)

#include <stdio.h>

int main(unsigned long long speid,
           unsigned long long argp,
           unsigned long long envp)
{
    printf("Hello World!\n");
    return 0;
}
hello_be1 – sync version – ppu code (1)

Makefile

```
DIRS  .   = spu
PROGRAM_ppu  .  = hello_be1
IMPORTS = -lspe2 -lpthread spu/hello_spu.a
include $(CELL_TOP)/buildutils/make.footer
```

hello_be1.c

```c
#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <libspe2.h>
```
extern spe_program_handle_t hello_spu;

int main(void)
{
    // Structure for an SPE context
    spe_context_ptr_t speid;
    unsigned int flags = 0;
    unsigned int entry = SPE_DEFAULT_ENTRY;
    void * argp = NULL;
    void * envp = NULL;
    spe_stop_info_t stop_info;
    int rc;

    // Create an SPE context
    speid = spe_context_create(flags, NULL);
    if (speid == NULL) {
        perror("spe_context_create");
        return -2;
    }
// Load an SPE executable object into the SPE context local store
if (spe_program_load(speid, &hello_spu)) {
    perror("spe_program_load");
    return -3;
}

// Run the SPE context
rc = spe_context_run(speid, &entry, 0, argp, envp, &stop_info);
if (rc < 0)
    perror("spe_context_run");

// Destroy the SPE context
spe_context_destroy(speid);
return 0;