



ECE4893A/CS4803MPG:
MULTICORE AND GPU PROGRAMMING FOR VIDEO GAMES
 Introduction to Cell Programming



Prof. Aaron Lanterman
 School of Electrical and Computer Engineering
 Georgia Institute of Technology



Cell Programming link on sti.cc.gatech.edu

Georgia Tech, STI Workshop on Software and Applications for the Cell/B.E. Processor

http://sti.cc.gatech.edu/programming.html

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:
 - Image contains VI, Kate, Kedit, eclipse, but no emacs???

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



Running the VMWare workshop image

Georgia Tech, STI Workshop on Software and Applications for the Cell/B.E. Processor

http://sti.cc.gatech.edu/programming.html

```
for i in *.zip; do unzip $i; done
```

Running the System


Before running the system for the first time you may need to adjust the size of allocated memory for the system. Do this by editing the line memsize = "xxxx" in the configuration file Cell Broadband.vmx. We recommend reserving about half the size of your physical RAM or at least 256.

1. Start the VMWare Player.
2. Select the Cell Broadband.vmx image.
3. Answer all the questions with their default values.
4. Login with root/innovate at the login screen.

VMware Players

- Windows_VMware-player-1.0.3-34682.exe (Windows)
- Linux_VMware-player-1.0.3-34682.i386.rpm (Linux)

You can get the full version of VMWare from OIT



Starting the simulator GUI

root@localhost:~/opt/ibm/systemsim-cell/run/cell/linux - Shell - Konsole

Session Edit View Bookmarks Settings Help

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

GUI started

License (C) All

systemsim-cell

File Window Help

- myasm
- PPE0.0
- PPE0.1
- SPE0
- SPE1
- SPE2
- SPE3
- SPE4
- SPE5
- SPE6
- SPE7
- Load-Elf-App
- Load-Elf-Kernel
- MemoryMap
- SystemMemory

cpu Cycles: 0

Advance Cycle Amount: 1

Advance Cycle	Go	Stop	Service GDB
Triggers/Breakpoints	Update GUI	Debug Controls	Options
Emiters	Mode	SFU Modes	SPE Visualization
Process-Tree	Process-Tree-Stats	Track All PCs	Event Log
Exit			

Set everything to Fast

The screenshot shows the simulator's configuration interface. At the top, there are buttons for 'Update GUI', 'Debug Controls', 'Mode', 'SPU Modes', 'Process-Tree-Stats', and 'Track All PCs'. Below this, the 'simmode' window shows 'Simulator is in mode SIMPLE' with 'Simple Mode' selected. The 'spumodes' window shows settings for BE0, with each SPU (SPU0-SPU7) and the BE0 itself set to 'Fast' for both 'Pipe' and 'Instruction'.

Starting the simulator

```

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 usbhid
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 devz 1843200
VFS: Mounted root (ext2 filesystem).
Freeing unused kernel memory: 448k freed
INIT: version 2.86 booting
        Welcome to Fedora 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):~
[root@(none)]# callthru source /opt/cell_class/Hands-on-30/hello/hello_ppu/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 > hello_spu
[root@(none)]# chmod 775 hello_spu
[root@(none)]# ./hello_spu
Hello World!
[root@(none)]#
    
```

Running Synergistic Hello Worlds

```

root@(none):~
[root@(none)]# callthru source /opt/cell_class/Hands-on-30/hello/hello_be1-src/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-async/hello_be1 > hello_be1
[root@(none)]# chmod 755 hello_be1
[root@(none)]# ./hello_be1
Hello World!
[root@(none)]#
    
```

http://wiki.cc.gatech.edu/cellbuzz/index.php/Main_Page

Log in / create account

article discussion edit history

Main Page CellBuzz Cluster link on sti.cc.gatech.edu

The CellBuzz wiki captures documentation supporting the configuration and usage of the IBM QS20 dual-Cell blade cluster at Georgia Tech, and publications related to the Cell BE processor. If you would like to add material to this wiki, please contact Prof. David A. Bader for an account on this wiki.

Georgia Tech Cell/B.E. QS20 CellBuzz Cluster

- Please acknowledge the use of the Georgia Tech CellBuzz Cell/B.E. BladeCenter Cluster in any research report, journal, or publication, that has benefited from access to these resources. The recognition of the Cell/B.E. resources is important for acquiring funding for the next generation of cyberinfrastructure. Our suggested acknowledgment is:
 - The authors acknowledge Georgia Institute of Technology, its Sony-Toshiba-IBM Center of Competence, and the National Science Foundation, for the use of Cell Broadband Engine resources that have contributed to this research.
- Account Request Form ← *Fill out this today!*
- CellBuzz User Guide ← *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 = -O4
```

```
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

```

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <libspe2.h>

```

hello_be1 – sync version – ppu code (2)

```

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;
    }

```

hello_be1 – sync version – ppu code (3)

```

// 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;
}

```

hello_be1 – async version – spu code

Makefile

```

PROGRAM_spu .  = hello_spu
LIBRARY_embed = hello_spu.a

#CPPFLAGS_gcc = -S
SPU_TIMING    = 1
#CC_OPT_LEVEL = -O4

```

(same as for
sync version)

```

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 – async version – ppu code (1)

Makefile

```

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

```

hello_be1.c

```

#include <errno.h>
#include <stdio.h>
#include <stdlib.h>
#include <libspe2.h>
#include <pthread.h>

```

hello_be1 – async version – ppu code (2)

```

typedef struct ppu_thread_data{
    spe_context_ptr_t context;
    pthread_t pthread;
    unsigned int entry;
    unsigned int flags;
    void *argp;
    void *envp;
    spe_stop_info_t stopinfo;
} ppu_thread_data_t;

```

hello_be1 – async version – ppu code (3)

```

void *ppu_thread_function(void *arg)
{
    ppu_thread_data_t *datap = (ppu_thread_data_t *)arg;
    int rc;
    rc = spe_context_run(datap->context, &datap->entry,
        . . . . . datap->flags, datap->argp,
        . . . . . datap->envp, &datap->stopinfo);
    pthread_exit(NULL);
}

```

hello_be1 – async version – ppu code (4)

```

extern spe_program_handle_t hello_spu;

int main(void) {
    ppu_thread_data_t data;
    data.context = spe_context_create(0, NULL);
    spe_program_load(data.context, &hello_spu);
    data.entry = SPE_DEFAULT_ENTRY;
    data.flags = 0;
    data.argp = NULL;
    data.envp = NULL;
    pthread_create(&data.pthread, NULL,
        . . . . . &ppu_thread_function, &data);
    pthread_join(data.pthread, NULL);
    spe_context_destroy(data.context);
    return 0;
}

```

simple-multispu/simple – spu code

Makefile

```
PROGRAMS_spu . := simple_spu
LIBRARY_embed := lib_simple_spu.a
include $(CELL_TOP)/builddutils/make.footer
```

simple_spu.c

```
#include <stdio.h>

int main(unsigned long long id) {
    printf("Hello Cell (0x%llx)\n", id);
    return 0;
}
```

simple-multispu/simple – ppu code (1)

Makefile

```
DIRS . := . spu
PROGRAM_ppu := . simple
IMPORTS = spu/lib_simple_spu.a -lspe2 -lpthread
INSTALL_DIR = $(SDKBIN)/samples
INSTALL_FILES = $(PROGRAM_ppu)
include $(CELL_TOP)/builddutils/make.footer
```

simple.c

```
#include <stdlib.h>
#include <stdio.h>
#include <errno.h>
#include <libspe2.h>
#include <pthread.h>
```

simple-multispu/simple – ppu code (2)

```
extern spe_program_handle_t simple_spu;
#define SPU_THREADS . 6

void *ppu_thread_function(void *arg) {
    spe_context_ptr_t ctx;
    unsigned int entry = SPE_DEFAULT_ENTRY;

    ctx = *((spe_context_ptr_t *)arg);
    if (spe_context_run(ctx, &entry, 0, NULL, NULL, NULL) < 0) {
        perror ("Failed running context");
        exit (1);
    }
    pthread_exit(NULL);
}

int main() {
    int i;
    spe_context_ptr_t ctxs[SPU_THREADS];
    pthread_t threads[SPU_THREADS];
```

simple-multispu/simple – ppu code (3)

```
/* Create several SPE threads to execute 'simple_spu'. */
for(i=0; i<SPU_THREADS; i++) {
    /* Create context */
    if ((ctxs[i] = spe_context_create (0, NULL)) == NULL) {
        perror ("Failed creating context");
        exit (1);
    }
    /* Load program into context */
    if (spe_program_load (ctxs[i], &simple_spu)) {
        perror ("Failed loading program");
        exit (1);
    }
    /* Create thread for each SPE context */
    if (pthread_create (&threads[i], NULL,
        &ppu_thread_function, &ctxs[i])) {
        perror ("Failed creating thread");
        exit (1);
    }
}
```

simple-multispu/simple – ppu code (4)

```

/* Wait for SPU-thread to complete execution.
*/
for (i=0; i<SPU_THREADS; i++) {
    if (pthread_join (threads[i], NULL)) {
        perror("Failed pthread_join");
        exit (1);
    }
}

printf("\nThe program has successfully executed.\n");

return (0);
}

```

simple-multispu/simple - let's run it!

```

[root@none]~]# callthru source /opt/cell_class/Hands-on-30/simple-multispu/sim
ple > simple
[root@none]~]# chmod 755 simple
[root@none]~]# simple
Hello Cell (0x1820008)
Hello Cell (0x1820688)
Hello Cell (0x1820900)
Hello Cell (0x1820b78)
Hello Cell (0x1820df0)
Hello Cell (0x1821068)

The program has successfully executed.
[root@none]~]# █

```