

ECE4893A/CS4803MPG:

# MULTICORE AND GPU PROGRAMMING FOR VIDEO GAMES

Lecture 25: Asynchronous Hello Cell



Prof. Aaron Lanterman



School of Electrical and Computer Engineering

Georgia Institute of Technology

# 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.thread, NULL,
                  &ppu_thread_function, &data);
    pthread_join(data.thread, 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)/buildutils/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)/buildutils/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/simple > 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) ~]# █