Lecture 24: Hello Cell

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

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

Running the VMWare workshop image

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 (for Mac as well) from OIT

Starting the simulator GUI

You can get the full version of VMWare (for Mac as well) from OIT
Set everything to Fast

Starting the simulator

Running Simple Hello Worlds

Running Synergistic Hello Worlds
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**

```bash
[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

---

**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**

```makefile
hello_spu.c (for the hello_be1 example)
#include <stdio.h>

int main(unsigned long long spuid,
          unsigned long long args,
          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 (sp eid == 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(sped, &hello_spu)) {
    perror("spe_program_load");
    return -3;
  }

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

  // Destroy the SPE context
  spe_context_destroy(sped);
  return 0;
}