Full System Simulator

SDK includes simulator preconfigured for Cell.
Usage:

- can be started running Linux as OS from the Linux run directory or without OS from the standalone directory
- can be started with GUI (../run_gui) or with command line only (../run_cmdline)

Files can be sent to the simulator using:

callthru source <fileOnRealSystem> > <fileOnSimulator>.

These files are not stored permanently. By

mount -o loop sysroot_disk /mnt

the files can be copied into the simulated environment permanently. Must be unmounted before running the simulator.

GDB – The GNU Project Debugger

Modified version of the GDB source-level debugger
Usage:

- add "CFLAGS= -g" to the makefile
- copy source and binary to the simulator
- *gdb <ppu-binary>*, for PPU-code and *spu-gdb <spu-binary>* fur SPU-code
- *gdb --tui <binary>* for window view

Important Commands

b source.c:57 if i=5	Break execution at a particular source code location (under condition)
info {command}	Information about…
d {breakpoint}	Delete a breakpoint
r	Run the debugged program.
n	Step to next statement and over routine calls
s	Step to next statement and into called functions.
c	Continue program until next breakpoint
finish	Step until end of current function
until {location}	Step until location
p {expression}	Print variable or content of a memory address
ba	Backtrace / Display stack
x {address}	Examine data at address
l	Show surrounding source code

OProfile

A statistical, kernel-based, profiler that is not yet available for Cell.
Usage:

- Point OProfile to the *vmlinux* file corresponding to the running kernel: *opcontrol --vmlinux=/boot/vmlinux*
- Start the daemon with *opcontrol --start*
- Use *opreport* the get summaries of data. *opreport -l <binary>* provides only data regarding the given program
- *opannotate --source <binary>* produces annotated source if binary was built with -g

Static timing analysis

SPU-gcc_timing is part of the SDK.
Usage:

- *make <spu-source.s>*
- *spu-gcc_timing <spu_source.s>*
- the annotated machine code can be found in *<spu_source.s.timing>*

The timing-file can be interpreted as follows:

0/1	indicates the pipeline that issued an instruction
D/d/	"D" signifies a successful dual-issue, "d" signifies a dual-issue did not occur due to dependencies and no entry signifies that issue rules were not satisfied
0-9	Each number represents one clock cycle that was taken for the instruction
-	Represents a dependency stall

Dynamic timing analysis

This is part of the simulator.
Usage:

- SPU must be set to *pipeline mode* in order to collect performance data
- Performance statistics fear each SPE can be accessed from the simulator under *SPUStats*
- To start, stop and reset the performance counter from the SPU-program *#include <profile.h>* and use *prof_start(), prof_stop()* and *prof_clear()*

Important indicators include cycles per instruction, single cycles, dual cycles, stalls due to branch miss and due to dependency and register use.