Programming Environments
&
Debugging

Greg Watson

Introduction

- Parallel Tools Platform
  - Provides basic functionality for developing parallel codes
  - Provides a platform for the integration of parallel tools
- Starting to get some experiences with tool integration
- Convincing argument for using an IDE?
- Debugging

Case Study 1: Cell SDK

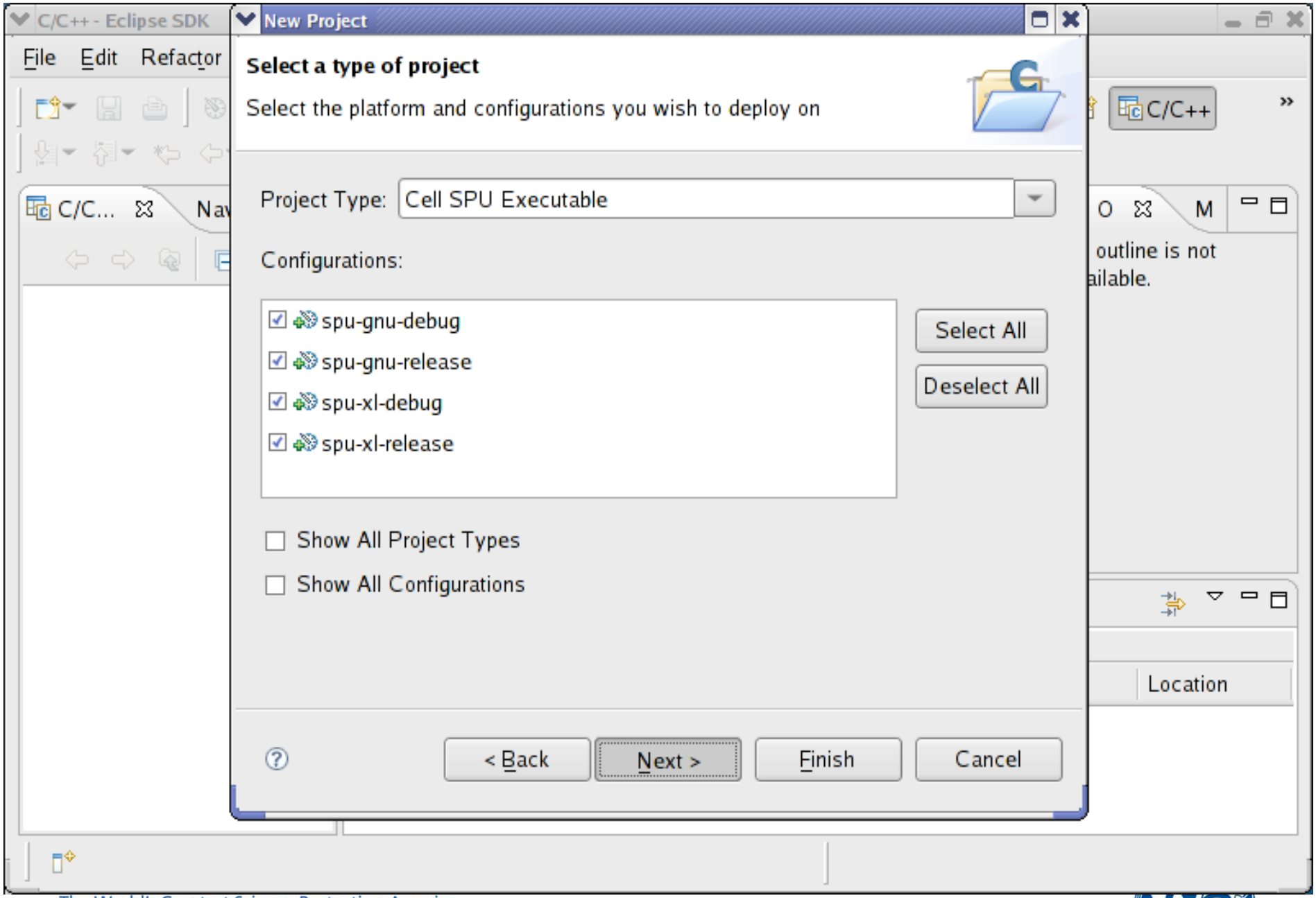
- Traditional build model
  - Requires three directories: project, PPE, SPU
  - Manually edit makefiles and include template
  - Specify SPU imports in project
  - Partition code into PPE and SPU (!)
  - During build process:
    - Compiles PPE code
    - Cross-compiles SPU code
    - Runs tool to embed SPU code into PPE executable

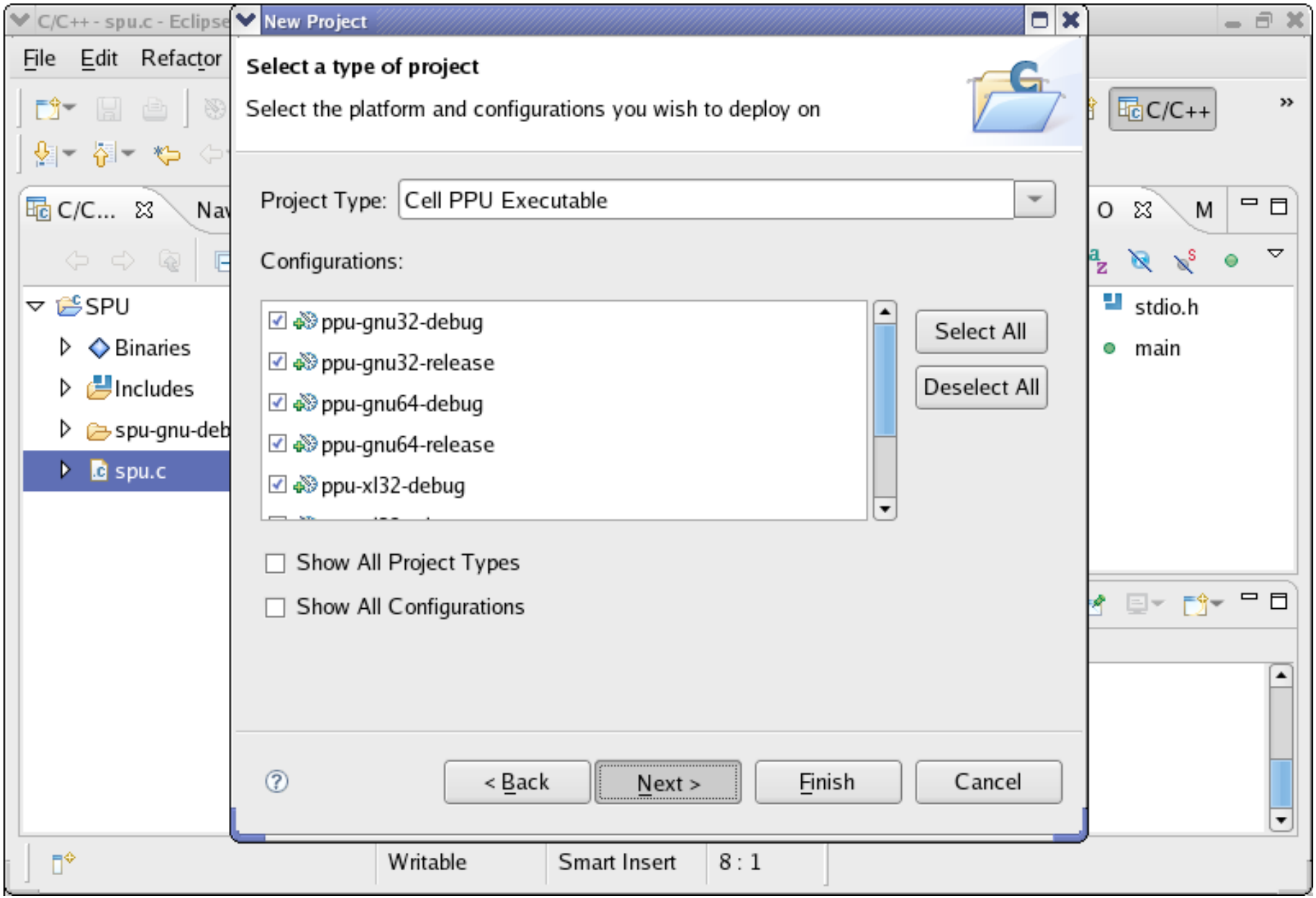
Case Study 1 (cont…)

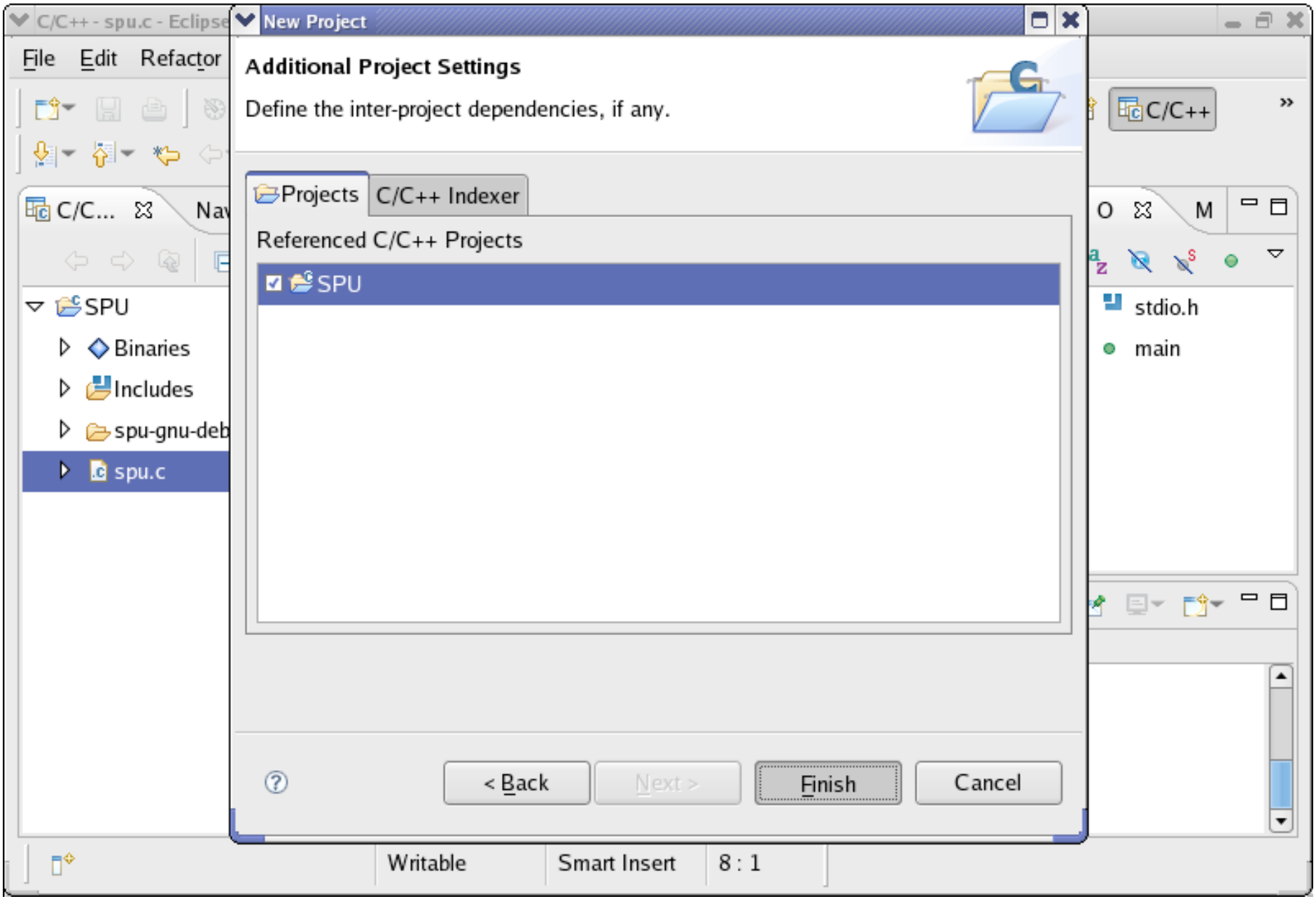
- Traditional (cont…)
  - Copy PPE executable to target
  - Run PPE executable
  - If debugging:
    - Launch separate debugger for PPE or SPU target
    - Connect to running SPU process as remote target
- Cell SDK
  - Version 2.0 available 15 December
  - Provides combined PPE/SPU debugger

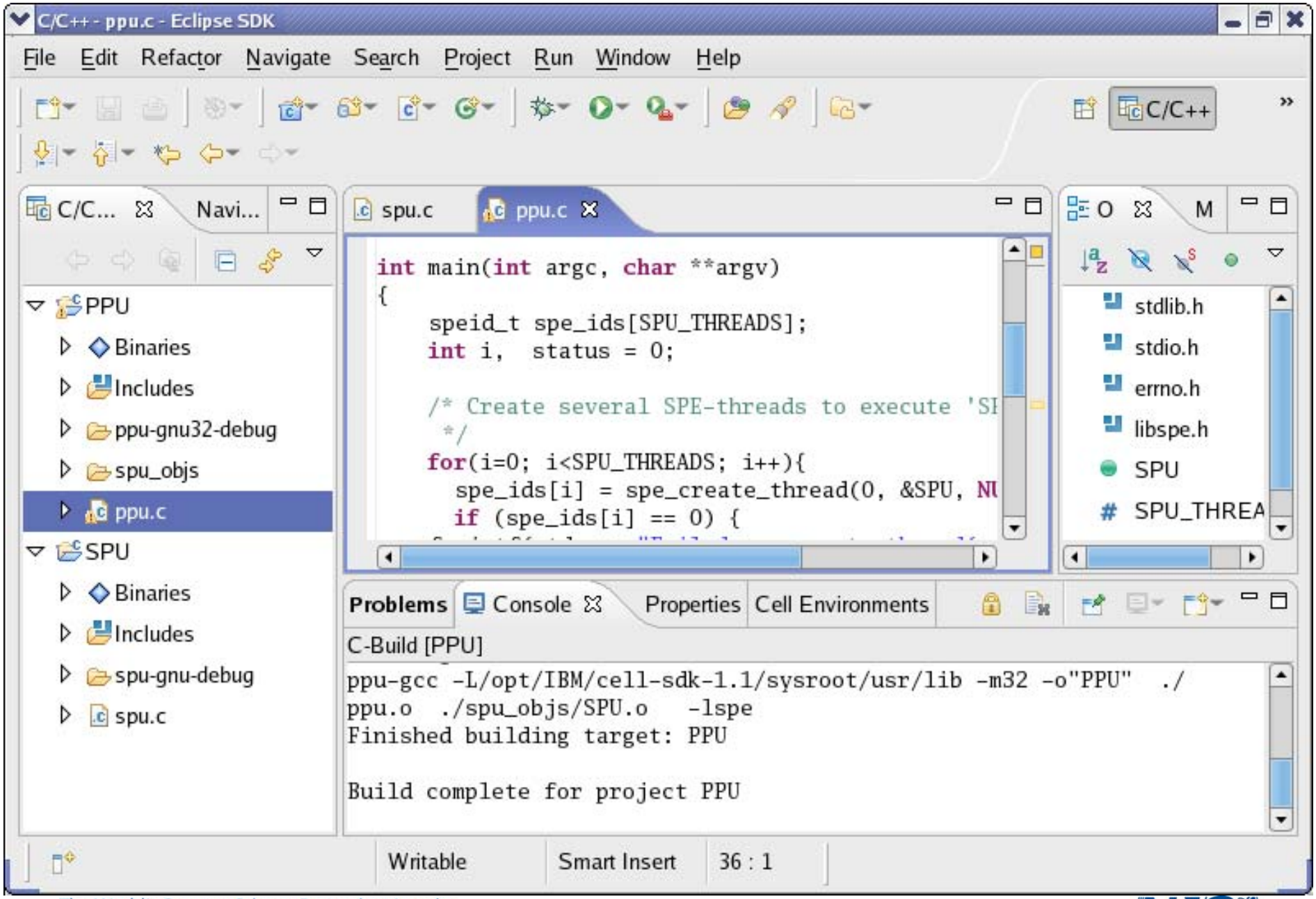
Case Study 1 (cont…)

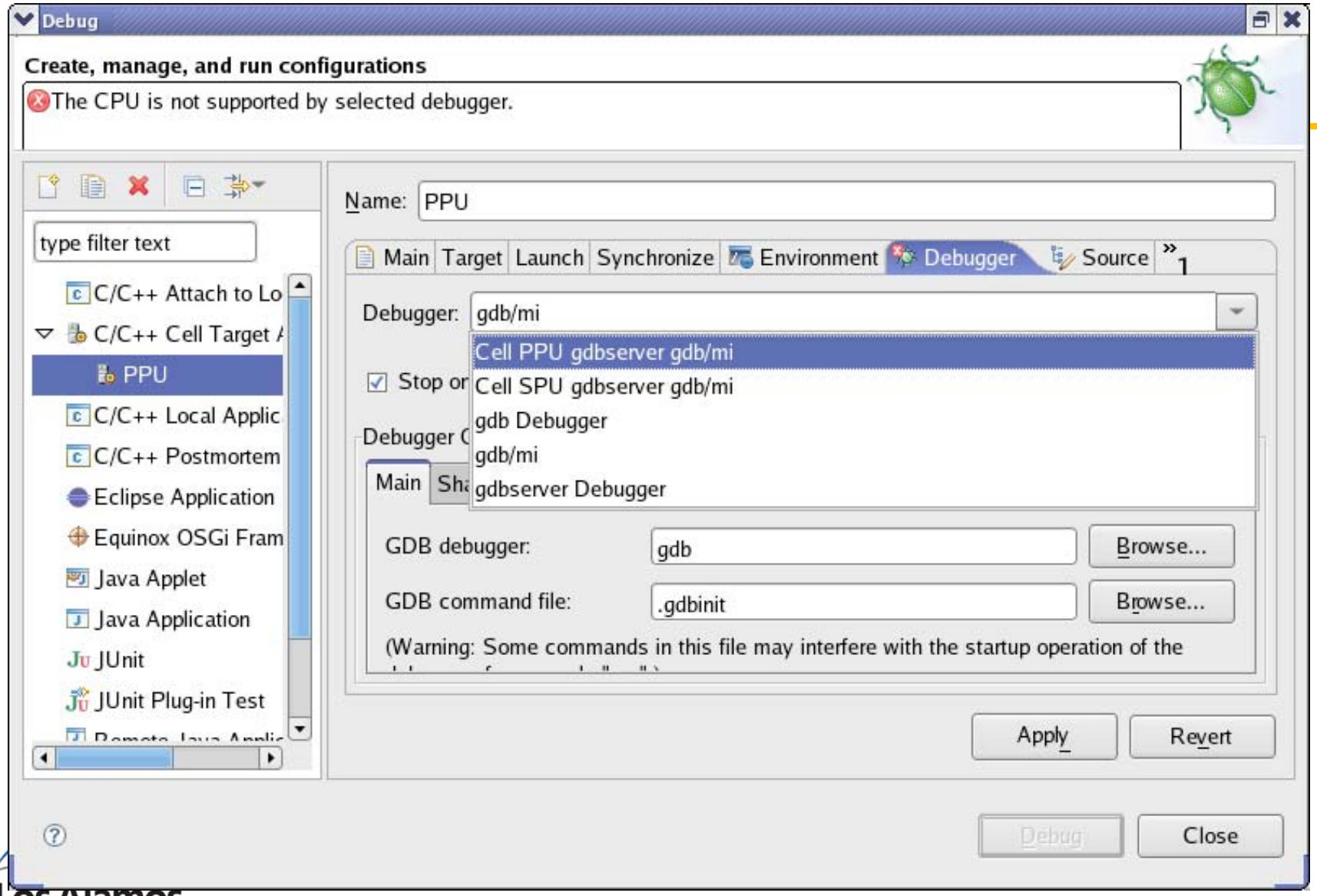
- Cell IDE
  - Eclipse based

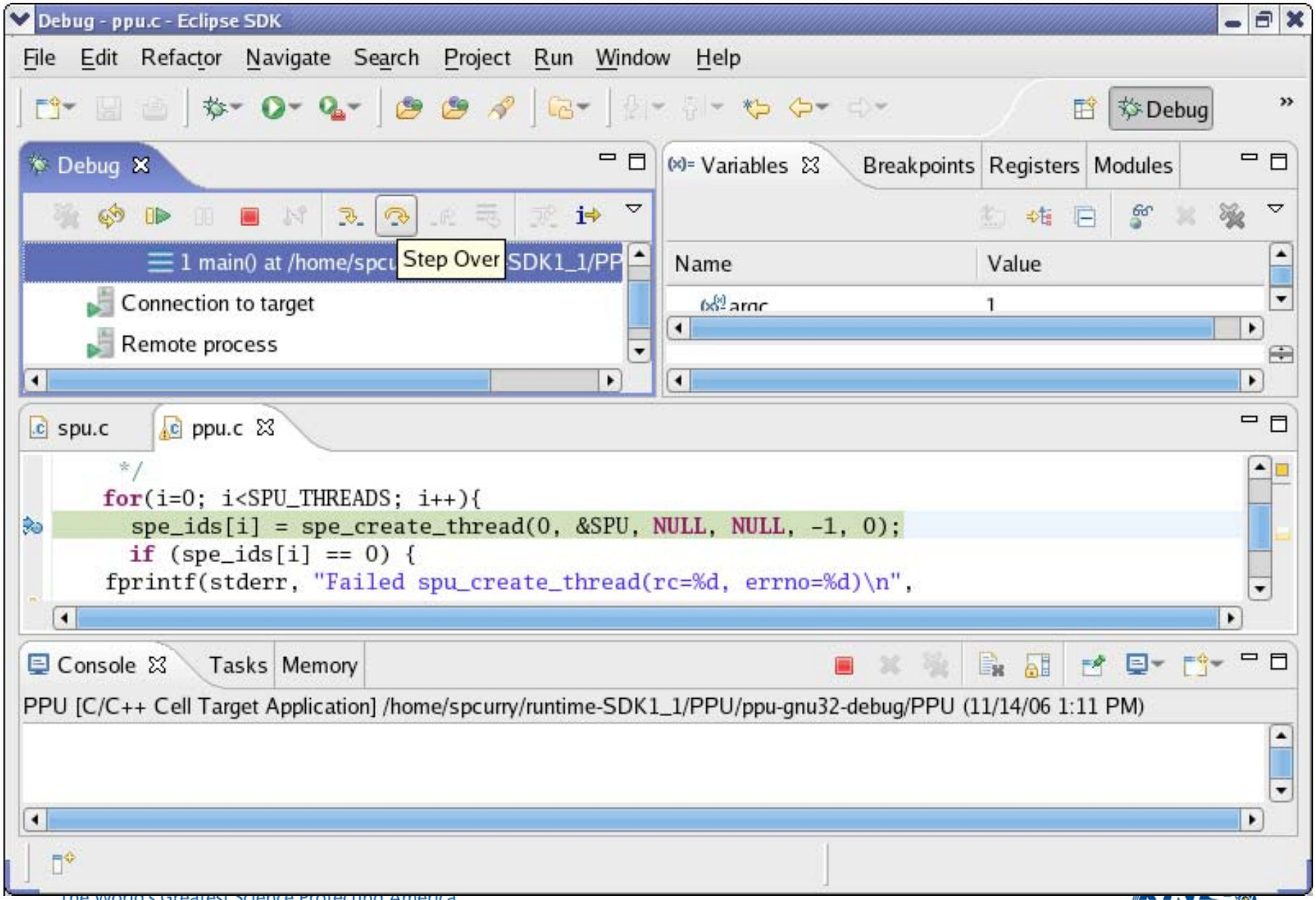

C/C++ - Eclipse SDK
File Edit Refactor
C/C++
New Project
Select a type of project
Select the platform and configurations you wish to deploy on
Project Type: Cell SPU Executable
Configurations:
spu-gnu-debug
spu-gnu-release
spu-xl-debug
spu-xl-release
Select All
Deselect All
Show All Project Types
Show All Configurations
< Back
Next >
Finish
Cancel
outline is not ailable.
Location


C/C++ - spu.c - Eclipse
File Edit Refactor
C/C++
New Project
Select a type of project
Select the platform and configurations you wish to deploy on
Project Type: Cell PPU Executable
Configurations:
ppu-gnu32-debug
ppu-gnu32-release
ppu-gnu64-debug
ppu-gnu64-release
ppu-xl32-debug
Select All
Deselect All
Show All Project Types
Show All Configurations
< Back
Next >
Finish
Cancel
SPU
Binaries
Includes
spu-gnu-deb
spu.c
stdio.h
main
Writable
Smart Insert
8 : 1


C/C++ - spu.c - Eclipse
File Edit Refactor
C/C++
C/C... Nav
SPU
Binaries
Includes
spu-gnu-deb
spu.c
O M
stdio.h
main
New Project
Additional Project Settings
Define the inter-project dependencies, if any.
Projects
C/C++ Indexer
Referenced C/C++ Projects
SPU
< Back
Next >
Finish
Cancel
Writable
Smart Insert
8 : 1


C/C++ - ppu.c - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
C/C++
C/C... Navi...
PPU
Binaries
Includes
ppu-gnu32-debug
spu_objs
ppu.c
SPU
Binaries
Includes
spu-gnu-debug
spu.c
spu.c
ppu.c
int main(int argc, char **argv)
{
speid_t spe_ids[SPU_THREADS];
int i, status = 0;
/* Create several SPE-threads to execute 'SI
*/
for(i=0; i<SPU_THREADS; i++){
spe_ids[i] = spe_create_thread(0, &SPU, NU
if (spe_ids[i] == 0) {
O M
stdlib.h
stdio.h
errno.h
libspe.h
SPU
SPU_THREA
Problems Console Properties Cell Environments
C-Build [PPU]
ppu-gcc -L/opt/IBM/cell-sdk-1.1/sysroot/usr/lib -m32 -o"PPU" ./
ppu.o ./spu_objs/SPU.o -lspe
Finished building target: PPU
Build complete for project PPU
Writable
Smart Insert
36 : 1


Debug
Create, manage, and run configurations
The CPU is not supported by selected debugger.
type filter text
C/C++ Attach to Lo
C/C++ Cell Target A
PPU
C/C++ Local Applic
C/C++ Postmortem
Eclipse Application
Equinox OSGi Fram
Java Applet
Java Application
JUnit
JUnit Plug-in Test
Name: PPU
Main
Target
Launch
Synchronize
Environment
Debugger
Source
Debugger: gdb/mi
Cell PPU gdbserver gdb/mi
Cell SPU gdbserver gdb/mi
gdb Debugger
gdb/mi
gdbserver Debugger
Stop or
Main
GDB debugger: gdb
Browse...
GDB command file: .gdbinit
Browse...
(Warning: Some commands in this file may interfere with the startup operation of the
Apply
Revert
Debug
Close


Debug - ppu.c - Eclipse SDK
File Edit Refactor Navigate Search Project Run Window Help
Debug
Debug
Step Over
1 main() at /home/spcu
SDK1_1/PP
Connection to target
Remote process
Variables
Breakpoints
Registers
Modules
Name
Value
argc
1
spu.c
ppu.c
*/
for(i=0; i<SPU_THREADS; i++){
spe_ids[i] = spe_create_thread(0, &SPU, NULL, NULL, -1, 0);
if (spe_ids[i] == 0) {
fprintf(stderr, "Failed spu_create_thread(rc=%d, errno=%d)\n",
Console
Tasks
Memory
PPU [C/C++ Cell Target Application] /home/spcurry/runtime-SDK1_1/PPU/ppu-gnu32-debug/PPU (11/14/06 1:11 PM)

Case Study 2: TAU

- Traditional
  - Stub makefile and library created when TAU installed
  - Manually include stub makefile in build system
  - Enable/disable instrumentation using TAU_DEFS environment variable
  - Launch application, selective instrumentation controlled by TAU_THROTTLE environment variable or by editing instrumentation file
  - Manually collect data
  - Run visualizer on collected instrumentation data

Case Study 2 (cont…)

- Eclipse
  - Uses existing Eclipse-based project
  - Click on Profile button instead of Run button
  - Select instrumentation from GUI
    - Automatically rebuilds application if necessary
  - Profile data automatically added to project database
  - Right-click to launch paraprof

Lessons

- Complex tool chains can be significantly simplified
  - Hide unnecessary details from the user
  - Presents much more logical workflow
- Advanced interface features
  - Greatly simplify user interaction with tool chain
  - Can provide features difficult or impossible otherwise

Lessons (cont…)

- New tool design and development simplified
  - No need to re-implement shared services
  - Can focus on core tool functionality
- Training and support costs are lower
  - Much easier to learn development tools
  - Support can focus on single environment across a range of architectures/machines

Other Advantages

- Managing legacy codes
  - Refactoring C structs to MPI types
  - Converting Fortran common blocks to derived types
  - Conversion to new languages
- Meta-compiler error checking
  - Concurrency checking for OpenMP
  - Fortran default single precision constants
- Programming model support
  - Visual aids for program design
  - Source code generation (e.g. MPI communicators)

Debugging

- Traditional debugging methodology
  - B.I.S.R. (breakpoint, inspect, step, repeat)
  - printf
  - May not work with advanced languages
  - Will not work with peta-scale architectures
- Assumption is that gdb and TotalView will solve the problem

Debugging Advances

- Identifying error occurrence
  - Predicates/assertions
  - Message/data patterns
- Locating error data
  - Visualize global program state
  - Comparison/search across application
- Rewinding/replaying
- As easy as printf

Debugging Advances (cont…)

- *and*
  - how does this work in heterogeneous environments?

What is needed?

- Universal parallel debugging platform
- Rich user interface
- Community willing to undertake research
  - Open source
- Confidence
  - Technology can be advanced
  - Will aid developers, not hinder
  - Longevity