IA-64 Linux Device Driver Porting

John A. Ronciak
Staff Engineer
NCG/NID/LAO
Intel Corp.

February 15-17, 2000

Intel Developer Forum Spring 2000

Intel Labs

Agenda

- Learning the Cross Development Platform
- Make the Device Driver IA-64 Clean
- Building an IA-64 Linux Driver
- Debugging an IA-64 Linux Driver
  - Code Examples (Case Studies)

Cross Development Platform

IA-64 SDK for Linux

- Components for Cross Development:
  - Latest Trillian Dev. Release for IA-64
    - Include Header Files and Libraries
    - Based on 2.3.35 Linux Kernel
  - Latest Build Tools
    - Compiler, assembler, etc.
  - IA-32 Linux distribution
    - i.e. Redhat 6.1, Suse 6.2, Caldera 2.3, etc.

Tools Available Now

Cross Development Platform

IA-64 SDK for Linux (cont.)

- System Hardware requirements:
  - IA-32 Machine
    - 350 MHz Pentium® II processor
    - 64MB RAM
    - 1 4-GB Hard Drive

Hint: Bigger and faster is better here.

Tools Available Now

Cross Development Platform

SDK Environment

- Standard Linux install
  - Nothing special needs to be done
- Test the install for functionality
  - Need to know things are working
- Load the Trillian Dev. Release
  - IA-64 Kernel Source and Build tools
    - independent location from regular kernel

Intel Developer Forum Spring 2000

Tools Available Now

Intel Labs

Cross Development Platform

Building the kernel

- Modify the Build Environment
  - Change the Upper-level Makefile
    - Point to where you installed the tools
- Configure the Kernel
  - Make menuconfig, xconfig, etc.
- Build the IA-64 Linux Kernel
  - Run the Upper-level Makefile

Tools Available Now

Cross Development Platform

Building the kernel (cont.)

- Check the Build for Errors
  - Need to Build Without Any Errors
- Check the Configuration
  - Make Sure All the Devices for Your Software Engineering System are Configured into the Kernel

IA-64 Clean

Source Code Cleaning

- Generic IA-64 Considerations
  - Data Models
  - Data Types
  - Pointers
  - Compiler Switches
- 64-bit Driver Source Cleanup
- Regression Test “Cleaned” Driver on IA-32

IA-64 Clean

Generic IA-64 Considerations

- Data Models
  - Uniform Data Model
    - Same source runs on IA-32 and IA-64 systems
  - LP64 Data Model
    - longs and pointers are 64-bit
    - ints remain 32-bit

IA-64 Clean

Generic IA-64 Considerations

- Data Types
  - New Data Types
    - int64_t and uint64_t
    - int32_t and uint32_t
    - int16_t and uint16_t
    - int8_t and uint8_t

IA-64 Clean

Generic IA-64 Considerations

- Pointers
  - All Pointers will be system defined
    - i.e. 32-bit pointers for IA-32, 64-bit for IA-64
- Compiler Switches
  - ARCH = ia64
  - CROSS_COMPILE = <path>/ia64-cygnus-linux-
  - -D__LP64__ and -D__LINUX__

IA-64 Clean

64-bit Driver Cleaning

- Use the new data types
  - IA-32 ulong's to uint32_t
    - long's are now 64-bit
  - ushort's to uint16_t
  - Check all hardware specific types and convert them accordingly
- Do not cast pointers to int's or long's
  - Write "clean" C code

IA-64 Clean

Regression Testing

- After a Clean IA-64 Compile
  - Compile same code on IA-32
  - Test the new IA-32 driver
    - easier to test with existing IA-32 machine
- Validate for All Needed Functionality
- Rebuild in IA-64 environment if needed

Building the Driver

Milestones so far

- Driver Cleanly Compiles for IA-64
- Driver Source is validated on IA-32
- Driver is integrated into IA-64 Linux kernel
- Loadable Module support as needed

Ready for Porting Now

Building the Driver

Linux and the SES

- Start the Linux Software Engineering System
- Put the Driver Source on the Software Engineering System
- Compile both the Linux Kernel and Driver on the Software Engineering System with the Native Tools

Building the Driver

Linux and the SES

- Alternatively use the Trillian Native kernel and compiler
  - Start from a working cross-development kernel
  - Apply the Native kernel source and tools
  - Compile and run as normal

Building the Driver

The New Kernel

- Reboot the Software Engineering System with the New Linux Kernel
- Test the New Driver
  - Start with basic functionality
  - Add more and more stress to the driver

Debugging the Driver

Linux Debug Tools

- Standard Printk's still work
  - Be careful with 64-bit pointers
- Debugger Built into the Linux Kernel
  - It's able to:
    - Dump and modify memory
    - Disassemble code
    - Set breakpoints

Debugging the Driver

Kernel Debugger

- Some Debugger commands:
  - md and mds displays memory
  - mm modifies memory
  - id disassembles code
  - go continues execution
  - rd and rm diplays and modifies registers
  - ef display exception frame

Debugging the Driver

Kernel Debugger (cont.)

- bt and btp stack backtrace and for pid
- bp, bl, bpa, bc, be bd set and manipulate breakpoints
- ss and ssb single step and step to branch
- env and set show and set environment variables
- cpu switch to new cpu
- reboot will immediately reboot machine

Debugging the Driver

Kernel Debugger (cont.)

- Use the Debugger
  - Saves development time
  - Use to test error/unused code paths
  - Enabled for panic backtraces
  - Use to check for correct types

Case Studies

Code Examples

- Structure packing
  - Compiler generates codes with naturally aligned boundaries.
- Structures use hard-coded size for padding

```
Struct Buffer {
        void   *Ptr[10];
        char    Padding[88];
        }
```

To pad data to 128-byte chunk (4*10 + 88) in IA-32

Case Studies

Code Examples (cont.)

- Structures Padding (cont.)
  - Pointer is 8-bytes in IA-64! Better to use

    Char Padding[128 - (10 * sizeof(void *))]
- Performance impact
  - due to padding error

Case Studies

Code Examples (cont.)

- Don’t use Unaligned data access in kernel
- Use “__unaligned” qualifier to access unaligned data, if necessary

```
__unaligned * pmyStruct;
```

- Performance impact on
  - Code size increased and Slow IO

Case Studies

Code Examples (cont.)

```
Struct  {
        ULONG  Space;
        PVOID    Buffer;
        ULONG  Offset;
        ULONG  Length;
} IoBlock;
```

```
Struct  {
        PVOID    Argument1;
        PVOID  Argument2;
        PVOID    Argument3;
        PVOID    Argument4;
} OtherStruct;
```

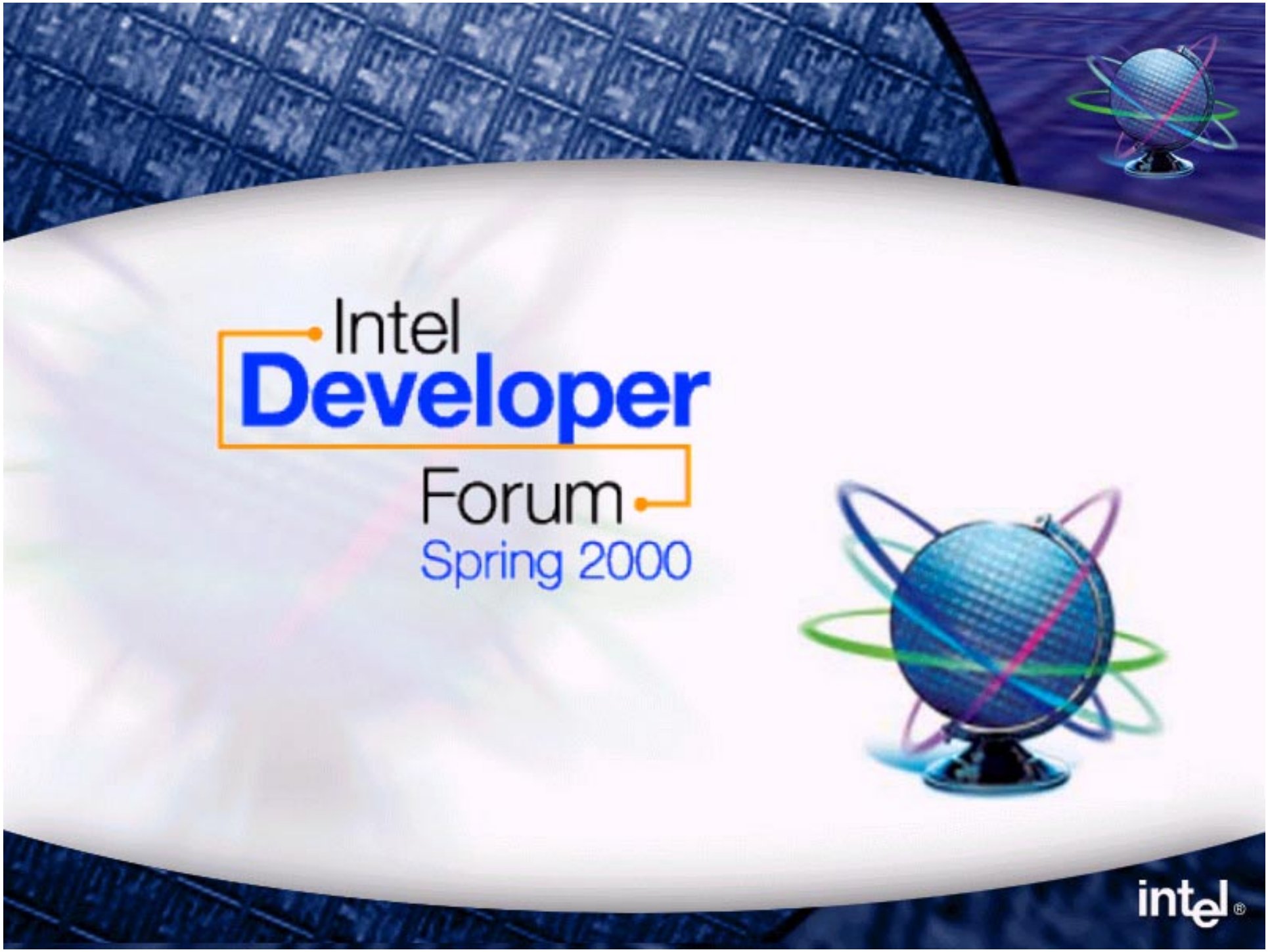
- In IA-32, it is ok to UNION structures “IoBlock” and “OtherStruct”
  - and use data structures interchangeably
- In IA-64, the data structs will be corrupted

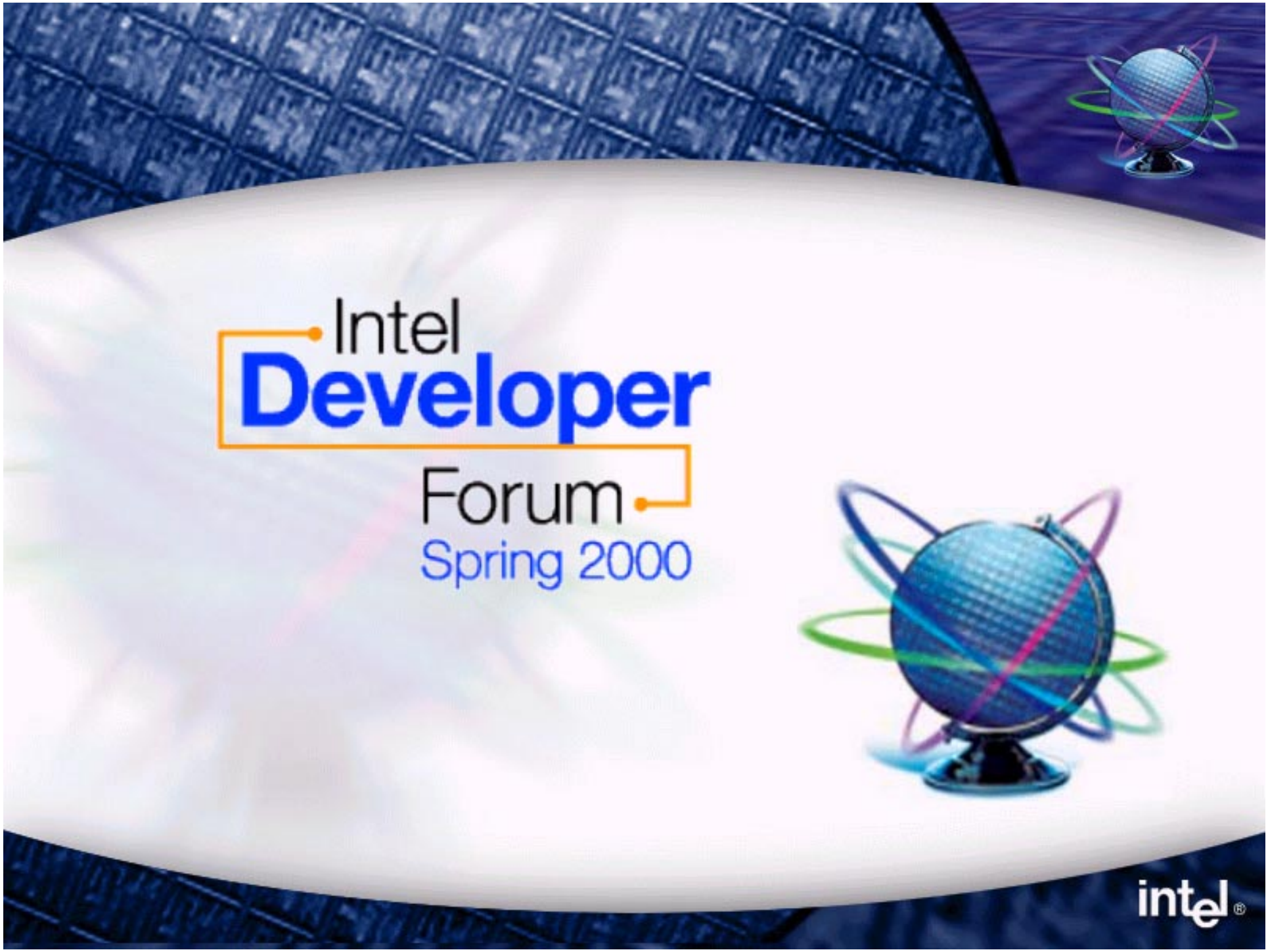
Useful URLs and References

- URLs
  - http://developer.intel.com/design/ia-64/index.htm
    - For IA-64 information from Intel
  - http://reality.sgi.com/slurn_engr/
    - For kernel debugger information(IA-32)
  - http://www.ia64linux.org
    - For Linux on IA-64 information

In Summary

- Porting Tools are Available Now
- The Porting Procedures are Easy
- Ready for the Porting Process Now
- Get an Early Start, Don’t Wait


Intel
Developer
Forum
Spring 2000
intel


Intel
Developer
Forum
Spring 2000
intel