Porting to IA-64 Linux

Michael Tiemann
CTO
Red Hat, Inc.

February 16, 2000
Agenda

- Trillian, GNU and Cygnus / Red Hat
- Porting to IA-64 Linux
  - IA-64 Linux (Your Target) Characteristics
  - Know Your Starting Point
  - Porting Issues/Checklist
  - Using GNU Tools to Ease Porting
    - other helpful tools
- Call to action
What’s GNU?

- **GNU is Not Unix!**
- Non-proprietary, cooperative development of tools, applications and systems
  - very efficient at creating *very good* peer-reviewed software
  - identical tools across many computer systems
  - naturally promotes portable software
- **Open source**
  - implementations are publicly readable
  - excellent learning tool

GNU tools are used by 95% of all Linux developers!!
Some GNU History

1984: GNU project starts - emacs
1985: GNU Debugger GDB
1987: GCC and G++
1989: First commercial GNU tools
1991: GCC 2.0, GDB 4.0
2000: GCC for IA-64, GCC 3.0, GDB 5.0
Red Hat (Cygnus) Trillian Effort

- IA-64 work started with HP’s initial port of GCC compiler in May’99
- Contributed GNUPro components for the Trillian effort
  - gcc, g++, gas, gld
  - Provides both native and cross development environments
  - Only LP64 data model supported
  - Supports development of EFI applications
IA-64 GNU tools – Future Work

• What’s in the works
  – Exception handling support
  – Assembler dependency violation support (DVLOC)
  – Intrinsics
    (efficiency of assembly within C/C++ code)
  – IA-64 -specific Assembler directives
  – Full gdb support
IA-64 GNU tools – Future Work

- Compiler Optimizations planned
  - New hardware pipeline model
  - Scheduling of instructions latencies
  - Better loop aware alias analysis
  - Software pipelining
  - Predication
  - Speculation
IA-64 UNIX ABI Work

- Intel, SCO, HP, IBM, SGI, SUN, Red Hat, VA Linux
- Agreements on
  - Common object format - ELF and DWARF
  - Standardized ELF extensions for better code generation
    - String section – allows compressions of string table program wide
    - Extended number of sections – more optimizations to reduce program size
    - Standardized intrinsic functions to allow access processor features through from C/C++ programs
  - more…
IA-64 C++ ABI Work

- IA-64 C++ ABI Group
  - SGI, Intel, HP, IBM, SUN, Red Hat and SCO
- Agreements on
  - standardized object layouts
  - common virtual table implementation
  - standardized (and optimized) name mangling
- Allows some C++ binaries from one Unix version to another UNIX version for IA-64 platforms
Porting to IA-64 Linux
Essence of IA-64 Linux

IA-64 Linux as seen by *your* code:

- **LP64**
  - Like other 64-bit UNIX machines
  - ILP32 *only* via execution of x86 binaries
    - Caution: x86 and IA-64 processes can share data via shared memory, files, sockets, ...

- **Standards compliant (mostly)**
  - ISO/ANSI C, C++ (ISO Standards C90 and C99)
  - most UNIX/98 APIs
Essence of IA-64 Linux

- little-endian (by default)
  - no big-endian support

- C/C++ pre-defines you can use:
  - established at compile time:
    __unix, __linux, __ia64, __LP64, __ELF__, __GNUC__
  - established by header #include files:
    __WORDSIZE (== 64)
Know Your Source Code

- Is your source ISO/ANSI compliant?
- What data structures are externally visible (files, sockets, shared memory, pipes, ...) to other architectures?
- What platform ports are already done?
- Is your source to be built on both UNIX and Windows?
Porting to IA-64 Linux

- If you are starting from Windows*
  - Set goals for Linux functionality
  - Pick tools and toolkits for Linux
  - Use C runtime library and POSIX-like functions instead of proprietary MS interfaces
    - Can use cygwin.dll as a bridge on IA-32 Linux

Porting to IA-32 Linux first is recommended, but you can port to IA-64 Linux directly

* All trademarks and brands are the property of their respective owners
Porting to IA-64 Linux

- **Guidelines:**
  - Involves mostly recompiling if your source is already 64-bit clean
  - Common object file format: ELF. All Unices will use it (no XCOFF, no SOM)
  - Standardized ELF extensions allow better object generation. E.g.
    - The string section will allow compression of string table program-wide
    - The extended number of sections will allow putting functions (esp C++) in different sections
  - Common binary format allows to develop tools usable for all different Unix platforms
Porting to IA-64 Linux

- **Step 1**: Get GNUPro for IA-32 Linux
- **Step 2**: Port apps to IA-32 Linux
  - Use `--Wall --Wconversion --Wsighn-compare`
- **Step 3**: Cleanup
  - `--pointer/integer warnings`
  - `--signed/unsigned warnings`
- **Step 4**: Reconfigure and recompile
Challenge of Portable Code

- Legacy code often ignores portability issues
- Many programmers are simply not taught to write portable software
- Writing and maintaining portable software is challenging, but worth it

GNU Software is exemplary… learn from it!
Writing Portable Software

- Follow GNU guidelines
  - Write code to be able to compile with ISO C or ISO C++ compilers
  - Configuration / support packages for compatibility
    - autoconf, automake, libtool
- Example: GNU ‘hello’
  - GNU coding standards
  - Internationalization support
Porting Rules for IA-64

- **DO**
  - Use `#include` to get system data types

- **DON’T**
  - declare things you don’t define (required by ISO C)

- **NEVER**
  - declare a structure the system defines
Porting Checklist

- **Fix casts that can truncate**
  - `(int)*xyz, (long)*xyz ⇒ (uintptr_t)*xyz`

- **Use `sizeof()` and `offsetof()`**
  - hard-coded numbers - “4” - don’t port
  - n.b. args to `malloc()`, pointer arithmetic

- **Use `size_t` instead of `int` or `long`**
  - `int ln= strlen(...) ⇒ size_t ln= str...
  - other revised APIs
Porting Checklist

• Use sized types for external structures
  – ISO/ANSI data types
    ```
    struct on_disk { int reclen; ... ⇒
    struct on_disk { int32_t reclen; ...
    ```
  – also for “on-wire” and shared memory structs

• Accommodate packing differences
  – IA-64 defaults to “natural alignment”
  – use `#pragmas` for shared legacy structures
Porting Checklist

- Revise `printf()`, `scanf()` specifiers
  
  ```c
  "%08lx", &foo) ⇒ ("%p", ...; "%d" ⇒ "%ld"; ...)
  ```

- `#error case at end of #ifdef chains`
  - prevents unexpected/silent use of some default case

  ```c
  ...
  #elif defined(__ia64)
  ...
  #else
  #error Update for new platform!
  #endif
  ```
Porting Checklist

- Use API calls to get system params
  
  `getpagesize()`, `sysconf(_SC_CLK_TICK)`

- Use header file mnemonics
  
  `SEEK_END (not 2), INT_MIN (not -2147483648)`

- Inline assembly must be re-written
Other things you can do

- **Sort your structures**
  - Always optimize for speed and not space
    - Put the structure elements together that are used often to find them in the same cache line
  - You should also
    - Put pointers together
    - Put shorts together
    - Put chars together
  - Net result: improved structure packing

- **-Wpadded**
  - warn when gcc pads to alignment boundary
Porting with GNUPro Tools

- Compilers enhanced to check for type compatibility problems
  - size incompatibilities
  - alignment/padding messages
  - bad comparisons

- Both native and cross compilers available
Sample (Buggy) Program

```c
int bugs (int foo, unsigned bar)
{
    signed int si;
    unsigned int ui;

    si = -1;
    ui = ~0;
    ui = ~0U;
    ui = ~0UL;
    if (si < ui)
        return 1;
    return -1;
}

int main (int argc, char **argv)
{
    int i;
    int *p = &i;
    bugs (p, i);
    bugs ((int)p, -1);
    return 0;
}
```
GNUPro tools on IA-32

tiemann@happy$ more ../linux/config.status
#!/bin/sh
# This file was generated automatically by configure. Do not edit.
# This directory was configured as follows:
/work/devo/configure --with-gcc-version-trigger=/work/devo/gcc/version.c
--host=i686-pc-linux-gnu -v --srcdir=/work/devo --norecursion
# using "mh-frag" and "mt-frag"
tiemann@happy$ gcc -Wall -Wsign-compare -Wconversion bugs.c -S
bugs.c: In function `bugs':
bugs.c:7: warning: negative integer implicitly converted to unsigned type
bugs.c:10: warning: comparison between signed and unsigned
bugs.c: In function `main':
bugs.c:19: warning: passing arg 1 of ‘bugs’ makes integer from pointer without a cast
bugs.c:19: warning: passing arg 2 of ‘bugs’ as unsigned due to prototype
bugs.c:20: warning: passing arg 2 of ‘bugs’ as unsigned due to prototype
bugs.c:20: warning: negative integer implicitly converted to unsigned type
GNUPro tools cross IA-64

tiemann@happy$ more ../linux-linux64/config.status
#!/bin/sh
# This file was generated automatically by configure. Do not edit.
# This directory was configured as follows:
/work/devo/configure --with-gcc-version-trigger=/work/devo/gcc/version.c
--host=i686-pc-linux-gnu --target=ia64-linux-gnu --srcdir=/work/devo --norecursion
# using "mt-frag"
tiemann@happy$ ../linux-linux64/gcc/xgcc -Wsign-compare -Wconversion bugs.c -S
bugs.c: In function `bugs':
bugs.c:7: warning: negative integer implicitly converted to unsigned type
bugs.c:9: warning: large integer implicitly truncated to unsigned type
bugs.c:10: warning: comparison between signed and unsigned
bugs.c: In function `main':
bugs.c:19: warning: passing arg 1 of `bugs' makes integer from pointer without a cast
bugs.c:19: warning: passing arg 2 of `bugs' as unsigned due to prototype
bugs.c:20: warning: cast from pointer to integer of different size
bugs.c:20: warning: passing arg 2 of `bugs' as unsigned due to prototype
bugs.c:20: warning: negative integer implicitly converted to unsigned type
bash-2.03$ /usr/cygnus/bin/gcc -Wall -Wsign-compare -Wconversion bugs.c
bugs.c: In function `bugs':
bugs.c:7: warning: negative integer implicitly converted to unsigned type
bugs.c:9: warning: large integer implicitly truncated to unsigned type
bugs.c:10: warning: comparison between signed and unsigned
...
... -- the same thorough diagnostics as when using the cross-compiler
...
bash-2.03$ gdb a.out
GNU gdb 4.18.1
Copyright 1998 Free Software Foundation, Inc.
...
This GDB was configured as "ia64-pc-linux"...
Portability Warnings

- Integer/Pointer assumptions
  - type size
  - Roundtrip conversion (int->void* - >int)

- Signed/Unsigned mismatches
  - Comparisons
  - Conversions

- Shifts and Masks
  - Many hash algorithms assume 32-bit ints!
What Makes Porting Easy?

- GNU-based compilers are used for 99% of all Linux development
- The Linux APIs are clean
  - IA-64 Linux is a pure 64-bit model
  - No legacy 32-bit complexity
- GNUPro compilers have special features to help port code
  - Between big and little endian
  - Across different integer and pointer sizes
  - To systems with different alignment rules
Call to Action

- Get the IA-64 Linux OS kernel
  - Available at www.ia64linux.org
- Get GNUPro for IA-32 and IA-64
- Adapt your build environment to configure and/or autoconf
  - This makes it easy to maintain a single source base for IA-32 and IA-64 targets
Additional Information

- See http://www.fsf.org/software/lists GNU software, including *hello*
- Visit gcc.gnu.org if you want to contribute to the compiler work
- http://developer.intel.com/design/ia-64 is a resource for additional IA-64 information