
OpenIMPACT at System Library and Kernel Levels

Nacho Navarro

University of Illinois at Urbana-Champaign
<http://gelato.uiuc.edu>

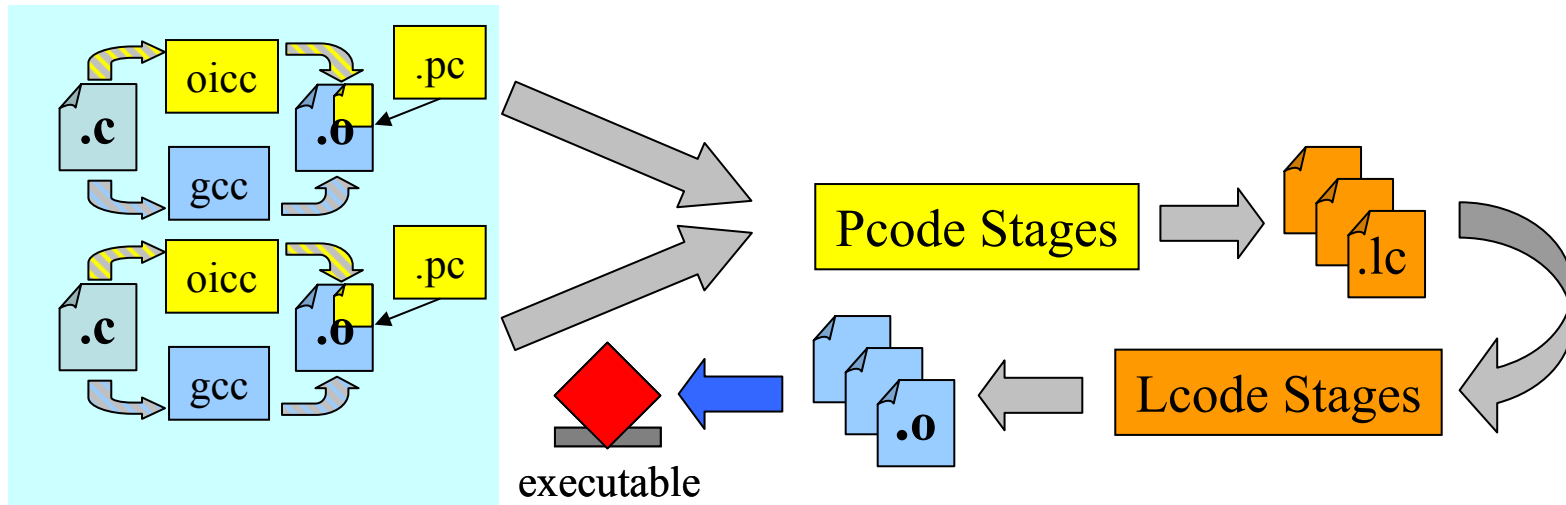
Agenda

- Status of the compilation/ linking process
- OpenIMPACT compatibility
- The nature of Linux kernel and library code
- Enabling more deep analysis
- Code coverage and profiling
- Next steps
- Your questions and comments

OpenIMPACT Kernel Compilation challenges

- Goal: Compile and optimize the Linux OS
 - Release a Linux runtime environment under Gelato
 - Benefit from the global analysis of application + libraries + Linux kernel
 - Cooperation of static analysis and runtime profiling
- OpenIMPACT and gcc compatibility
 - New compiler front-end and binary tools compatible with gcc environment and language extensions
 - *“Linux is written for the GNU gcc compiler. Trying to compile it with any other C compiler will result in complete failure”*
(Linux Internals, Moshe Bar, Mc Graw Hill, 2000)

OpenIMPACT Two Stage Compilation



- Makefile compatibility
 - Make and binutils can be used without modification
 - Compiler output is intermediate representation: new ELF section
- Interprocedural and aggressive optimizations at “link” stage
 - oicc picks up IR, performs global deep analysis and profiling
 - New Pcode symbol table is preserved and info will be persistent

OpenIMPACT Linux Compilation Issues

- Specific gcc / ld flags
- Language issues
 - Attributes in functions and variable declarations
 - Sections
 - Functions, data alignment
 - Packed structures
 - Declarations used only inside assembly code
 - Zero length arrays, variable length array, bounded array initialization, empty structures
 - Inline assembly, volatile
 - Extern inline functions
 - IA64 registers support

Gcc Compatibility: Resolved Items

- Statement Expressions
- Locally Declared Labels
- Naming Types
- Referring to a Type with typeof
- Generalized Lvalues
- Conditionals with Omitted Operands
- Long Long
- Hex Floats
- Variadic Macros
- Slightly Looser Rules for Escaped Newlines
- Multi-line Strings
- Arithmetic on void- and Function-Pointers
- Non-Constant Initializers
- Compound Literals
- Designated Initializers
- Cast to Union
- Case Ranges
- Mixed Declarations and Code
- Attribute Syntax
- Function Prototypes
- C++ Style Comments
- Dollar Signs in Identifier Names
- The Character <ESC> in Constants
- Inquiring about Alignment of Types or Variables
- An Inline Function is As Fast As a Macro
- Alternate Keywords
- Incomplete enum Types
- Pragmas Accepted by GCC
- Unnamed struct/union fields within structs/unions.

gcc Compatibility Summary

- Statement Expressions
- Locally Declared Labels
- Naming Types
- Referring to a Type with typeof
- Generalized Lvalues
- Conditionals with Omitted Operands
- Long Long
- Hex Floats
- Variadic Macros
- Slightly Looser Rules for Escaped Newlines
- Multi-line Strings
- Arithmetic on void- and Function-Pointers
- Non-Constant Initializers
- Compound Literals
- Designated Initializers
- Cast to Union
- Case Ranges
- Mixed Declarations and Code
- Attribute Syntax
- Function Prototypes
- C++ Style Comments
- Dollar Signs in Identifier Names
- The Character <ESC> in Constants
- Inquiring about Alignment of Types or Variables
- An Inline Function is As Fast As a Macro
- Alternate Keywords
- Incomplete enum Types

- Pragmas Accepted by GCC
- Unnamed struct/union fields within structs/unions.

- Declaring Attributes of Functions
- Specifying Attributes of Variables
- Specifying Attributes of Types

- Labels as Values
- Constructing Function Calls
- Arrays of Length Zero
- Arrays of Variable Length
- Non-Lvalue Arrays May Have Subscripts
- Assembler Instructions with C Expression Operands
- Constraints for asm Operands
- Controlling Names Used in Assembler Code
- Variables in Specified Registers
- Function Names as Strings
- Getting the Return or Frame Address of a Function
- Vector instructions through built-in functions
- Other built-in functions provided by GCC

- Nested Functions
- Complex Numbers
- Target Specific Built-in Functions

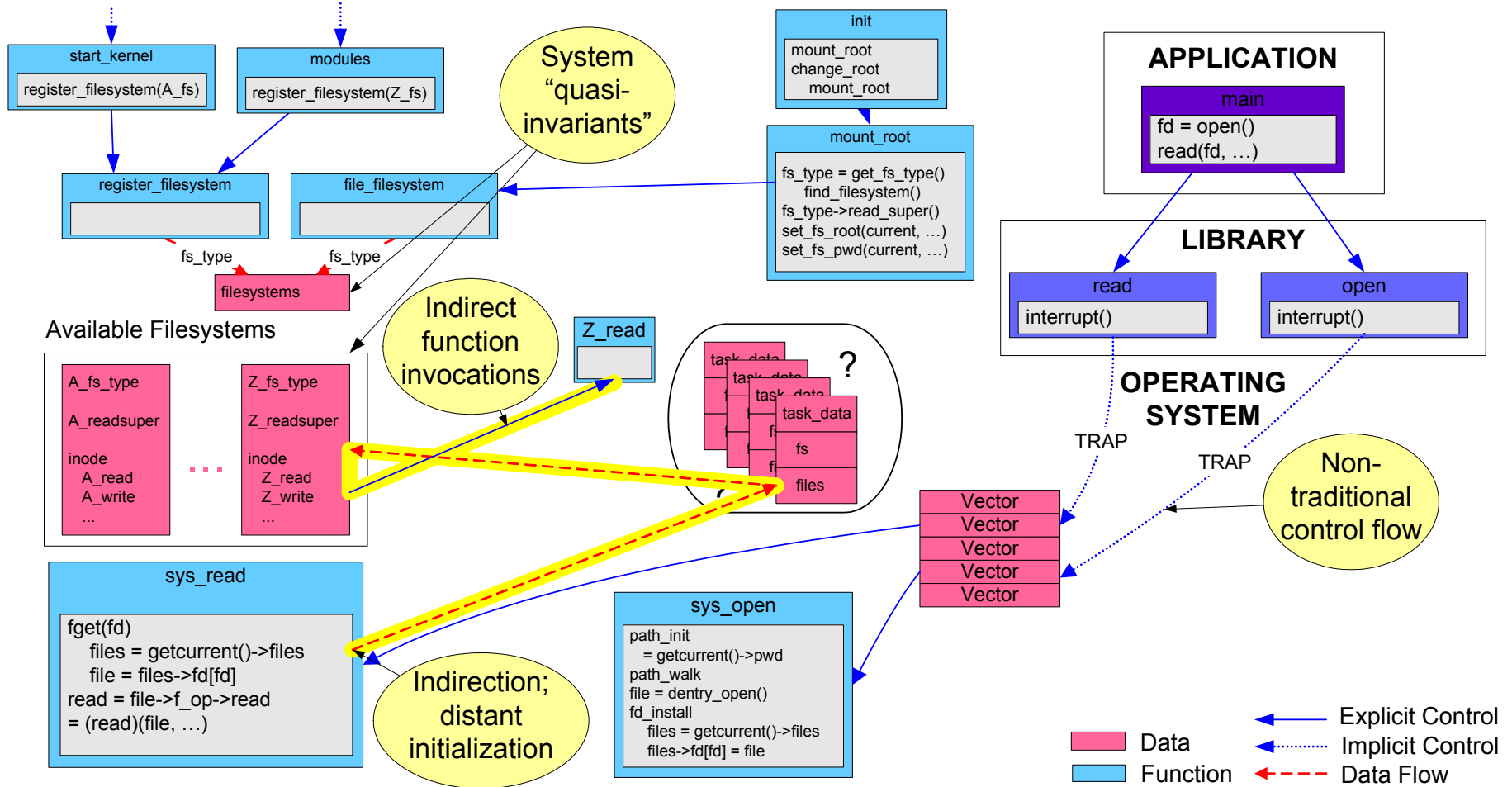
System Libraries Optimization Wish List

- Libraries features
 - Symbol resolution priority, weak symbols
 - Static and shared objects linking support
 - Well defined interfaces (malloc)
 - Assembly code integration (memcpy)
- Libraries deep analysis
 - Usually IR was not persistent, we want to incorporate the libraries into application interprocedural analysis
 - Global symbol database (Pcode archive and efficient selection at link time)
- Release optimized versions of libc, libm ...
 - Preprocessed, Pcode with partial analysis
 - Ready for Pcode inlining
 - Prepared for runtime optimizations
 - Keep compatibility (static and shared objects)

System source code characteristics

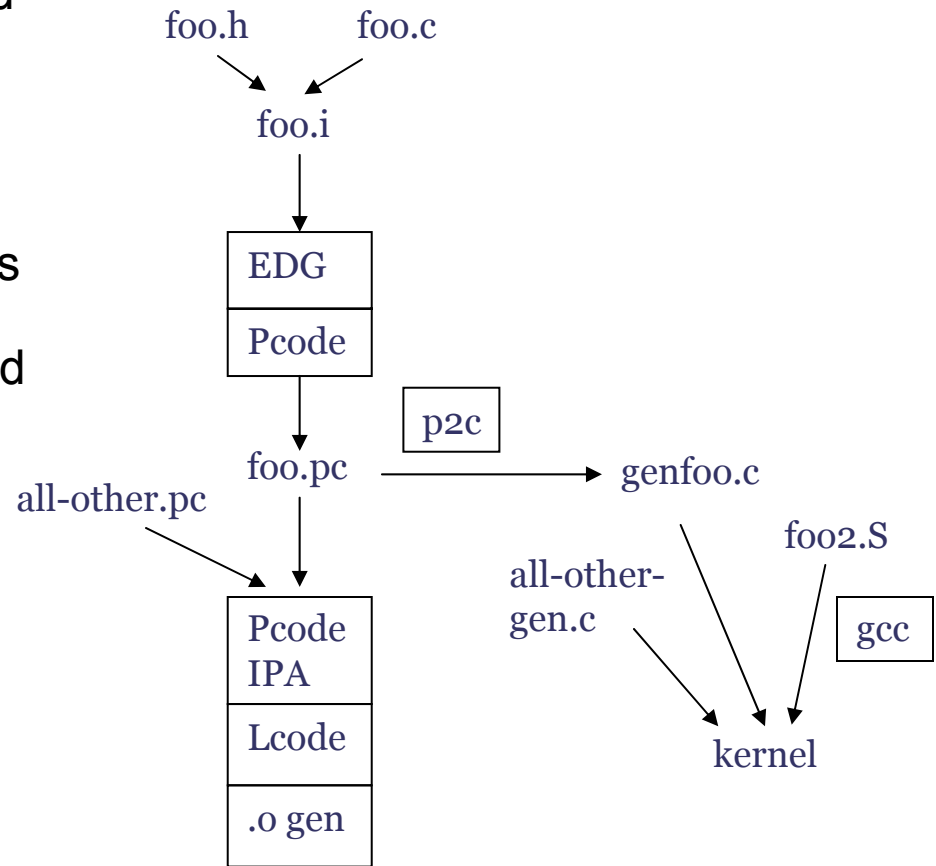
- Kernel and system libraries have stable API, and are related
 - Subsystems and device drivers follow well known programming guidelines, but not underneath
 - Challenge of let the compiler know system level architectural properties, stable conventions
- General purpose code
 - Tricky to characterize, new territory
- Plenty of indirections, initialized at boot time
- Domains, protection, parameter checking

Challenges in Unraveling O/S Code



Status of Linux Kernel Compilation

- GCC specific support
 - Kernel source can be compiled to Pcode
 - Generate back to C source code is completed
- Assembly code support
 - Inline assembly code is kept as black box
 - Assembly files not yet compiled with OpenIMPACT
- Test of regenerated C code
 - Compiled and linked with assembly files (gcc)
 - Boot under debugging



Linux kernel Optimization

- Kernel performance is not critical ?
- Gcc compilation at `-O2` is conservative
- Good stress test environment for features
 - Stability and reliability of our transformations
 - Debugging
- Metrics
 - speedup, boot time, footprint, cache contamination, time to market
- Configurations
 - workstations, servers, data centers (virtual machines)
- Kernel runtime phases: boot, steady state
- Modules extensibility vs. optimization

Profiling the Kernel

- OpenIMPACT iteration of compilation, profiling and recompilation
- What's a “typical” input set?
- Need the ability to boot and profile
 - Scripts with representative workload
 - Collection of profile counters
 - Launch using virtual machine, full system emulation
 - Deterministic execution in emulation run
- But less than half kernel code is executed

Next steps

- Make the kernel bootable, debug, stabilize
- Provide optimized versions of libraries
- Global deep analysis with interprocedural information database
- Deep analysis scalability, incremental compilation
- Merge analysis and profiling information