From Fortran to Fortran: The Porting Guide

Igor S. Gerasimov

November 5, 2025

FortranCon 2025

Fortran compilers

- **gfortran** — from GNU Compiler Collection (GCC).
- **LLVM flang** — LLVM-based Fortran front-end; successor to flang-classic.
- **flang-classic** — Original LLVM Flang front-end; basis for:
  - NVIDIA HPC SDK (former PGI)
  - AMD Compiler Collection (AOCC)
  - ArmFlang
- **ifort & ifx** — Intel's classic and LLVM-based compilers.
- **Oracle/Sun Fortran compiler** — Oracle's Fortran compiler for SPARC and x86.
- **IBM XL Fortran** — IBM's compiler for Power and z systems.
- **Cray Fortran compiler** — for Cray machines.
- **NAG Fortran compiler** — by Numerical Algorithms Group; has a lot diagnistics.
- ...

🔗 List of Fortran compilers

Classes of porting issues

Compile-time errors and warnings

- Invalid or non-standard code
- Compiler bugs

Classes of porting issues

Compile-time errors and warnings

- Invalid or non-standard code
- Compiler bugs

Link-time errors and warnings

- Missing symbols
- Incompatible interfaces between separately compiled units

Classes of porting issues

Compile-time errors and warnings

- Invalid or non-standard code
- Compiler bugs

Link-time errors and warnings

- Missing symbols
- Incompatible interfaces between separately compiled units

Run-time errors

- Segmentation faults (`SIGSEGV`) or bus errors (`SIGBUS`)
- Accessing uninitialized or out-of-bounds arrays
- Numerical exceptions (division by zero) and instabilities

Porting plan

For consistency during the presentation and porting process, the following procedure is applied:

Algorithm

for issue_class $\in$ {Compile-time, Link-time, Run-time}:
  for compiler $\in$ {gfortran, ifort/ifx, flang, xl, cray, nag}:
    build some source with compiler
    fix produced errors and, if possible, all warnings
    check that it did not affect already fixed compilers
  end for
end for
return consistent builds and results across compilers

Porting plan

For consistency during the presentation and porting process, the following procedure is applied:

Algorithm

for issue_class $\in$ {Compile-time, Link-time, Run-time}:
  for compiler $\in$ {gfortran, ifort/ifx, flang, xl, cray, nag}:
    build some source with compiler
    fix produced errors and, if possible, all warnings
    check that it did not affect already fixed compilers
  end for
end for
return consistent builds and results across compilers

Main idea: get the information about possible bugs as earlier as possible.

Before start...

A large set of Link-time errors comes from using of compiler extensions not supported by other compilers.

- getenv → get_environment_variable
- system → execute_command_line
- exit(0) → stop
- exit(1) → error stop
- call flush → flush
- derf → erf
- dasinh → asinh
- dfloat → dble
- ...

NOTE: Assume that for **gfortran** code works fine.

Compile-time errors and warnings

- Invalid or non-standard code
- Compiler bugs

ifx: interfaces mismatch

```
subroutine call_op(a, b, op_id)
  interface
    subroutine add; end
    subroutine sub; end
  end interface

  real(8) :: a, b
  integer :: op_id

  if (op_id == 1) call do_op(add, a, b)
  if (op_id == 2) call do_op(sub, a, b)
end subroutine call_op

subroutine add(a, b)
  real(8) :: a, b
  a = a + b
end subroutine add
```

ifx error #5633:

```
**Internal compiler error:
segmentation violation signal
raised**
```

ifx: interfaces mismatch

```
subroutine call_op(a, b, op_id)
  interface
    subroutine add; end
    subroutine sub; end
  end interface

  real(8) :: a, b
  integer :: op_id

  if (op_id == 1) call do_op(add, a, b)
  if (op_id == 2) call do_op(sub, a, b)
end subroutine call_op

subroutine add(a, b)
  real(8) :: a, b
  a = a + b
end subroutine add
```

ifx error #5633:

```
**Internal compiler error:
segmentation violation signal
raised**
```

Solutions:

Use **external** declarations

```
external add
external sub
```

Or proper explicit interfaces

```
interface
  subroutine add(a, b)
    real(8) :: a, b
  end subroutine add
```

Link-time errors and warnings

- Missing symbols
- Incompatible interfaces between separately compiled units

Incompatible Fortran calls

file1.f90:

```
subroutine example(skeylen, skey)
  integer :: skeylen
  character :: skey(*)
end subroutine example
```

file2.f90:

```
subroutine call_example()
  call example(4.8, "1234")
end subroutine call_example
```

Link-time optimizations and Fortran calls

file1.f90:

```
1 subroutine example(skeylen, skey)
2   integer :: skeylen
3   character :: skey(*)
4 end subroutine example
```

file2.f90:

```
1 subroutine call_example()
2   call example(4.8, "1234")
3 end subroutine call_example
```

Check with LTO:

```
#  gfortran -flto file1.f90 file2.f90 main.f90
file2.f90:2:34: warning: type of 'example' does not match original
    declaration [-Wlto-type-mismatch]
  2 | call example(4.8, "1234")
    |                           ^
file1.f90:1:18: note: 'example' was previously declared here
  1 | subroutine example(skeylen, skey)
    |                  ^
```

The same approach can be applied for **bind(C, name=)**.

Implicit Fortran ABI

```
subroutine example(skeylen, skey)
  integer :: skeylen
  character :: skey(*)
end subroutine example
```

For **gfortran**:

```
// gfortran -fc-prototypes-external -fsyntax-only file1.f90
...

void  example_  (int *skeylen, char *skey, size_t skey_len);

...
```

Other compilers? Can be different.

Implicit Fortran ABI

```
1 // gfortran -fc-prototypes-external -fsyntax-only file1.f90
2 ...
3 void  example_  (int *skeylen, char *skey, size_t skey_len);
4       ^        ^^
5 ...
```

Many variants:

- `example` (LFortran)
- `EXAMPLE` (ifort on Windows ∼15 years ago)
- `_example_` (gfortran on MacOS, but linker do work)
- `example_` (gfortran, flang, ...)
- `example__` (g77)

That is still a problem for OpenBLAS and other projects.

Run-time errors and warnings

- Segmentation faults (`SIGSEGV`) or bus errors (`SIGBUS`)
- Accessing uninitialized or out-of-bounds arrays
- Numerical exceptions (division by zero) and instabilities

SegFault with AOCC 4.2.0

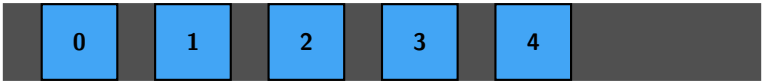
```
program main
  use, intrinsic :: iso_fortran_env, only: real64
  use, intrinsic :: iso_c_binding, only: c_loc
  implicit none
  real(real64), target :: static_memory(1)
  call get_c_ptr(c_loc(static_memory(1)))
  print *, static_memory(1)
end program main

subroutine get_c_ptr(cptr)
  use, intrinsic :: iso_fortran_env, only: int64
  use, intrinsic :: iso_c_binding, only: c_ptr, c_f_pointer
  implicit none
  type(c_ptr) :: cptr
  integer(int64), pointer :: ival
  call c_f_pointer(cptr, ival)
  ival = 0
end subroutine get_c_ptr
```

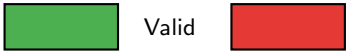
Memory layout and Address sanitizer

Default allocations layout

Allocations after enabling Address sanitizer



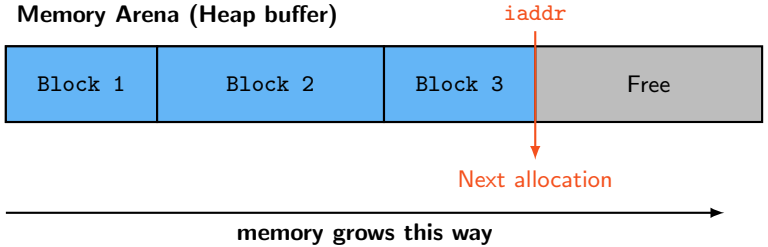
Allocations and redzones



K&R stack allocator

A simple linear allocator that manages memory as a **stack**:

- Memory is a single contiguous region
- Each allocation moves a pointer `iaddr` forward
- Deallocation simply resets the pointer backward
- Extremely fast; no fragmentation



Accessing to Address sanitizer API

LLVM: compiler-rt/include/sanitizer/asan_interface.h

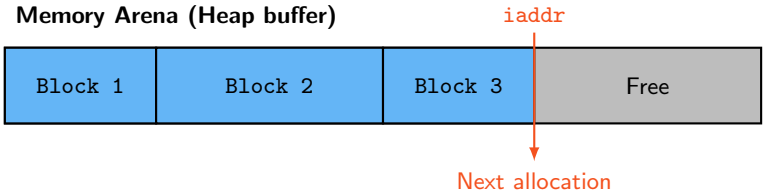
```
...
/// Marks a memory region as unaddressable.
    void __asan_poison_memory_region((addr), (size));
...
/// Marks a memory region as addressable.
    void __asan_unpoison_memory_region((addr), (size));
...
/// Checks if an address is poisoned.
    int __asan_address_is_poisoned(void const volatile *addr);
...
/// Checks if a region is poisoned.
    void* __asan_region_is_poisoned(void *beg, size_t size);
...
```

Accessing to Address sanitizer API

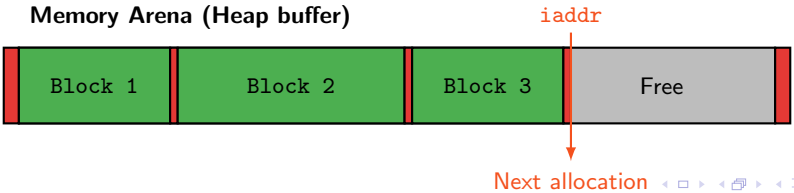
```
1     subroutine asan_poison_memory_region(addr, size) bind(C, name="
          __asan_poison_memory_region")
2       use iso_c_binding, only: C_ptr, C_size_t
3       type(c_ptr),        value, intent(in) :: addr
4       integer(c_size_t), value, intent(in) :: size
5     end subroutine asan_poison_memory_region
6     subroutine asan_unpoison_memory_region(addr, size) bind(C, name="
          __asan_unpoison_memory_region")
7       use iso_c_binding, only: C_ptr, C_size_t
8       type(c_ptr),        value, intent(in) :: addr
9       integer(c_size_t), value, intent(in) :: size
10    end subroutine asan_unpoison_memory_region
```

K&R stack allocator

No Address sanitizer:



With Address sanitizer:



Run-time behaviour

- `-O0` / `-O3` / `-O5` optimization levels
- for **gfortran**: `-finit-integer=n -finit-logical=<true|false> -finit-real=<zero|inf|-inf|nan|snan> -finit-character=n`
- `-ffast-math`
- `-march=...`, `-mavx512f` *(found a bug with K&R allocator and OpenMP)*
- different ISAs: x86_64, AArch64, RISC-V, Power
- different OS: Linux, AIX, MacOS, FreeBSD, Windows, ...
- `MALLOC_PERTURB_=n` env var for glibc

Not necessary to have a good set of tests, just generate some inputs and compare outputs.

Numerical checks

Module IEEE_EXCEPTION has

- IEEE_INVALID, always nice to catch
- IEEE_DIVIDE_BY_ZERO, also nice to catch
- IEEE_INEXACT
- IEEE_UNDERFLOW
- IEEE_OVERFLOW

```
1    use, intrinsic :: ieee_exceptions, only: ieee_set_flag,
         ieee_set_halting_mode, ieee_invalid
2    call ieee_set_flag(ieee_invalid, .false.)
3    call ieee_set_halting_mode(ieee_invalid, .true.)
```

Works well on x86_64, RISC-V CPUs have a lot of issues

Does not work with Reference-LAPACK because of IEEECK routine

Numerical checks

IEEE_ARITHMETIC is really useful with IEEE_EXCEPTION:

```
1     use, intrinsic :: ieee_arithmetic, only: ieee_value, ieee_signaling_nan
2     real(8), allocatable :: allocated_data(..)
3 ...
4     allocated_data = ieee_value(1.0_real64, ieee_signaling_nan)
```

In K&R allocators can be used between allocated blocks instead of ASan poisoning.

Domain specific checks

In Quantum Chemistry: H atom, He (triplet state)

In general: special cases

Take home messages

- Compilers can be used for validation of Fortran code
- Variety of compilation flags may discover hidden bugs
- LTO in not only optimizations
- Address sanitizer can work with K&R stack allocators
- IEEE intrinsic may help to stabilize numerical results

Thank you

Contact Information

@foxtran

foxtranigor@gmail.com

github.com/foxtran