毕昇编译器
2.1.0
用户指南

文档版本 01

发布日期 2021-12-30

华为技术有限公司

商标声明


和其他华为商标均为华为技术有限公司的商标。
本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受华为公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，华为公司对本文档内容不做任何明示或暗示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

华为技术有限公司

地址： 深圳市龙岗区坂田华为总部办公楼 邮编：518129

网址： https://www.huawei.com

客户服务邮箱： support@huawei.com

客户服务电话： 4008302118

目 录

1 毕昇编译器介绍

前言

本手册提供毕昇编译器的使用方法以及构建业务场景的注意事项。

概述

毕昇编译器是针对鲲鹏平台的高性能编译器。它基于开源LLVM开发，并进行了优化和改进，同时将Flang作为默认的Fortran语言前端编译器。

功能介绍

除LLVM通用功能和优化外，毕昇编译器的工具链对中端及后端的关键技术点进行了深度优化，并集成Auto-tuner特性支持编译器自动调优。自动调优操作指导可以参考《 Autotuner特性指南 》。

部分通用信息请参考LLVM的用户指导**https://llvm.org/docs/UserGuides.html**，毕昇编译器新增的自定义选项参考**自定义优化选项**章节。

支持的编程语言

LLVM是一种涵盖多种编程语言和目标处理器的编译器，毕昇编译器聚焦于对C、C++、Fortran语言的支持，利用LLVM的Clang作为C和C++的编译和驱动程序，Flang作为Fortran语言的编译和驱动程序。

C，C++程序

Clang不仅仅是可以将C, C++程序编译为LLVM中间表示的IR，它也是一个驱动程序，会调用所有以代码生成为目标的LLVM优化遍，直到生成最终的二进制文件。毕昇编译器提供了端到端编译程序所需的所有工具和库。

Fortran程序

Flang是专为LLVM集成而设计的Fortran前端，由两个组件flang1和flang2组成。它也是一个驱动程序，将源代码转换为LLVM IR，前端驱动程序将IR传输下去进行优化和目标代码生成。

2 毕昇编译器安装使用

2.1 环境依赖

- 内存：8GB以上
- 操作系统：openEuler21.03、openEuler 20.03 (LTS)、CentOS 7.6、Ubuntu 18.04、Ubuntu 20、麒麟V10、UOS 20
- 架构：AArch64
- GCC版本：4.8.5以上
- glibc版本：2.17以上
- libatomic版本：1.2.及以上

2.2 获取毕昇编译器

在**毕昇编译器产品页**选择“毕昇编译器软件包下载”获取毕昇编译器软件包。

软件包名称：bisheng-compiler-2.1.0-aarch64-linux.tar.gz

目录结构

bisheng-compiler-2.1.0-aarch64-linux

-- bin

-- include

-- lib

-- libexec

-- share

完整性校验

获取软件包后，需要校验软件包，确保与网站上的原始软件包一致。**毕昇编译器产品页**提供sha256sum文件用于软件包的完整性校验（见“毕昇编译器 sha256”），用户可使用以下命令生成哈希值对比确认：

```
sha256sum bisheng-compiler-2.1.0-aarch64-linux.tar.gz
```

2.3 安装毕昇编译器

本节介绍毕昇编译器的安装步骤，以下操作均使用root用户执行。

步骤1 获取毕昇编译器软件包，并校验完整性后将其上传到目标执行机。

步骤2 设置安装目录

1. 创建毕昇编译器安装目录（这里以/opt/compiler为例）

```
mkdir -p /opt/compiler
```

2. 将毕昇编译器压缩包拷贝到安装目录下：

```
cp bisheng-compiler-2.1.0-aarch64-linux.tar.gz /opt/compiler
```

步骤3 进入压缩包目录，执行命令解压缩软件包。解压完成后在当前目录下出现名为bisheng-compiler-2.1.0-aarch64-linux的目录。

```
cd /opt/compiler
tar -zxvf bisheng-compiler-2.1.0-aarch64-linux.tar.gz
```

步骤4 配置毕昇编译器的环境变量

```
export PATH=/opt/compiler/bisheng-compiler-2.1.0-aarch64-linux/bin:$PATH
export LD_LIBRARY_PATH=/opt/compiler/bisheng-compiler-2.1.0-aarch64-linux/lib:$LD_LIBRARY_PATH
```

须知

以上步骤是以/opt/compiler目录举例，若您的安装目录不同，请以实际目录为准。

步骤5 安装完毕后执行如下命令验证毕昇编译器版本：

```
clang -v
```

若返回结果已包含bisheng compiler版本信息，说明安装成功。

说明

毕昇编译器基于开源LLVM，其命令clang，clang++, flang的使用方法与开源LLVM相同。

----结束

2.4 使用毕昇编译器

编译运行 C/C++程序

```
clang [command line flags] hello.c -o hello.o
./hello.o
```



```
clang++ [command line flags] hello.cpp -o hello.o
./hello.o
```

编译运行 Fortran 程序

```
flang [command line flags] hello.f90 -o hello.o
./hello.o
```

指定链接器

毕昇编译器指定的链接器是LLVM的lld，若不指定它则使用默认的ld。

```
clang [command line flags] -fuse-ld=lld hello.c -o hello.o
./hello.o
```

2.5 安全加固

目的

缓冲区溢出攻击是利用缓冲区溢出漏洞所进行的攻击行动。缓冲区溢出是一种非常普遍、非常危险的漏洞，在各种操作系统、应用软件中广泛存在。利用缓冲区溢出攻击，可以导致程序运行失败、系统关机、重新启动等后果。

在当前网络与分布式系统安全中，被广泛利用的50%以上都是缓冲区溢出，而缓冲区溢出中，最为危险的是堆栈溢出，因为入侵者可以利用堆栈溢出，在函数返回时改变返回程序的地址，让其跳转到任意地址，带来的危害一种是程序崩溃导致拒绝服务，另外一种就是跳转并且执行一段恶意代码，例如跳转到shell执行恶意代码。

方案

提供栈保护编译选项进行加固软件的安全性，防范堆栈溢出攻击。关于栈保护选项可以使用命令 clang --help | grep stack-protector 查看。

3 毕昇编译器选项说明

3.1 默认选项

- 支持LLVM的所有优化等级（O0/O1/O2/O3/Ofast）。
- 支持**Clang的默认编译选项**和**Flang的默认编译选项**。
- 支持fsanitize=address/leak/memory等选项。

3.2 指定数学库

optimized-routines是针对ARM体系处理器各种库函数的优化实现，毕昇编译器将其以动态链接库的方式集成到工具链中，支持数学函数的标量和矢量实现。

指定标量数学库：

```
clang -O3 -lmathlib -lm
```

指定矢量数学库：

```
clang -O3 -fveclib=MATHLIB -lmathlib -lm
```

须知

mathlib函数支持不完全，需要增加-lm选项链接libm，并且libm的链接顺序必须在mathlib之后。

3.3 指定 jemalloc

毕昇编译器支持jemalloc库的使用，jemalloc是一个通用的malloc实现，主要是为了减少内存碎片和提高并发性能，以动态库的方式集成到工具链中。

使能方式：

```
clang -O3 -ljemalloc
```

须知

jemalloc的动态链接库文件存放于bisheng-compiler-2.1.0-aarch64-linux/lib文件中，将该目录加入LD_LIBRARY_PATH后才可以直接使用-ljemalloc，否则编译时需要添加-L$(library)指定库路径。

3.4 LTO 优化

毕昇编译器支持因某些优化需要分析整个程序，在链接时使用-flto启用链接时优化(LTO)。

3.5 浮点运算控制选项

编译器经过一系列优化后可能出现浮点运算结果不一致，此类情况一般是由以下几个原因导致的：

- 浮点精度有限
- 每个编译器都有影响浮点计算结果的选项
- 选项因编译器而异，并且具有不同的默认值
- 某些选项可以被其它看似无关的选项显式地启用/禁用
- 通过启用/禁用正确的选项，编译器可以控制浮点运算结果

默认选项

毕昇编译器的浮点运算控制选项基于LLVM官方文档，可参阅：**https://clang.llvm.org/docs/UsersManual.html#controlling-floating-point-behavior**。

-Mflushz

该选项用于控制非规范化的浮点值刷新为零，与其它不安全的浮点优化分开。开源Clang和Flang仅在x86架构支持该选项，毕昇编译器将AArch64架构加入支持范围。使用方式与开源版本相同。

参考文档：**https://en.wikipedia.org/wiki/Denormal_number**。

-ffp-contract=style

该选项的值可以是off/on/fast，毕昇编译器将其值默认设为fast，使能浮点数乘加操作，将乘法和加法合并为一条乘加运算从而提升性能。

-faarch64-pow-alt-precision=18/21

Flang选项, 仅对fortran代码起效。用于更改对于pow函数的优化策略，使得pow函数的计算结果与非ARM平台保持一致。

-faarch64-minmax-alt-precision

Flang选项, 仅对fortran代码起效。用于更改对于min/max函数的优化策略，使得min/max函数的计算结果与非ARM平台保持一致。

-mllvm -disable-sincos-opt

llvm选项，更改sin、cos函数的优化策略，使得sin、cos函数的计算结果与非ARM平台保持一致。

-mllvm -aarch64-recip-alt-precision

llc选项，使用软浮点补偿，使得recip倒数指令的计算结果与非ARM平台保持一致。

-mllvm -aarch64-rsqrt-alt-precision

llc选项，使用软浮点补偿，使得rsqrt倒数开方指令的计算结果与非ARM平台保持一致。

-mllvm -enable-alt-precision-math-functions

llvm选项，作用是将数学函数__mth_i_cosd，__mth_i_asind及__pd_powi_1的函数名替换为cosdf，asindf及powr8i4，从而实现控制数学函数精度的效果（需要结合kml数学库使用），此选项仅在O1及以上优化起效。

-mllvm -enable-18-math-compatibility

llvm选项，作用是将数学函数tgammaf，cbrt，log，log10 等数学函数换作有_18后缀的函数，从而实现控制数学函数精度的效果（需要结合kml数学库使用），此选项仅在O1及以上优化，且开启-mllvm -enable-alt-precision-math-functions 时起效。

-ffp-compatibility=17/18/21

通用选项，用于统一控制为保持计算结果与非ARM平台保持一致需要打开的所有选项。

-ffma-combine-fdiv

通用选项，用于将表达式 a/b+c 优化为 fma(a, 1/b, c)，有利于保持计算结果与非ARM平台保持一致，仅在-ffp-contract=fast时起效。

-ffma-reverse-associative

通用选项，用于将表达式 ab+cd 优化为 fma(a, b, c*d) ，有利于保持计算结果与非ARM平台保持一致，仅在-ffp-contract=fast时起效。

-Hx,124,0xc00000

flang 选项，用于保持使用常量初始化的舍入方式与非ARM计算平台保持一致，仅在fortran起效。

3.6 自定义优化选项

毕昇编译器支持通过-mllvm驱动的自定义优化选项，由于基于鲲鹏架构优化，使能自定义优化选项需要指定鲲鹏架构，如-mcpu=tsv110。

-mllvm -force-customized-pipeline=<true|false>

强制使用自定义的pass顺序。设为true开启该优化，默认关闭。

-mllvm -sad-pattern-recognition=<true|false>

对差值的绝对值求和运算（ sum += abs(a[i] - b[i]) ）进行优化，生成更简单高效的运算序列。设为true开启该优化，默认开启。

-mllvm -instcombine-ctz-array=<true|false>

实现对De Bruijn序列查表计算的优化。设为true开启该优化，默认开启。

-mllvm -aarch64-loopcond-opt=<true|false>

减少某些条件下的循环条件判断中的冗余指令，生成更优的代码。设为true开启该优化，默认开启。

-mllvm -aarch64-hadd-generation=<true|false>

对于矢量化的运算 (x[i] + y[i] + 1) >> 1，使用一条ARM NEON指令**URHADD**完成运算，从而生成更优的代码。设为true开启该优化，默认开启。

-mllvm -enable-loop-split=<true|false>

使能循环拆分优化，可以将符合条件的一个循环拆分成多个循环从而有助于实现冗余循环删除等优化。设为true开启该优化，默认开启。

-mllvm -enable-mem-chk-simplification=<true|false>

LLVM循环矢量化常常需要生成运行时检查，此优化致力于简化运行时检查的逻辑，从而生成更优的循环矢量化代码。设为true开启该优化，默认开启。

-mllvm -aarch64-ldp-stp-noq=<true|false>

禁止生成**stp/ldp q1**, **q2**, **addr**形式的指令，此形式的指令性能较差。设为true开启该优化，默认开启。

-mllvm -enable-func-arg-analysis=<true|false>

增强LLVM的值域分析能力，使LLVM的function specialization优化可以覆盖更多的函数场景。设为true开启该优化，默认开启。

-mllvm -ipsccp-enable-function-specialization=<true|false>

增强function specialization优化，使得此优化可以对函数参数为函数指针的场景生效。设为true开启该优化，默认开启。

-mllvm -enable-modest-vectorization-unrolling-factors=<true|false>

使得步长较小的循环更容易被矢量化。设为true开启该优化，默认开启。

-mllvm -instcombine-shrink-vector-element=<true|false>

通过提高矢量化指令的并行度，消除矢量化过程中生成的标量中间值，达到增强循环矢量化的效果。设为true开启该优化，默认开启。

-mllvm -instcombine-reorder-sum-of-reduce-add=<true|false>

通过更改reduction操作的顺序生成更优的reduction代码。设为true开启该优化，默认开启。

-mllvm -replace-fortran-mem-alloc=<true|false>

在fortran代码中，对于已知大小的内存分配（如数组），使用栈内存代替堆内存，从而提升性能。设为true开启该优化，默认开启。

须知

此优化生效的内存大小由-mllvm -max-fortran-heap-to-stack-size=<Int number>指定，默认为4096。

-mllvm -enable-pg-math-call-simplification=<true|false>

优化fortran多个数学库函数调用，提升fortran数学函数调用的性能。设为true开启该优化，默认开启。

-mllvm -instcombine-gep-common=<true|false>

优化多维数组在复杂场景下（如嵌套多层循环）的元素地址计算，减小寄存器压力，提高程序性能。设为true开启该优化，默认开启。

-mllvm -disable-extra-gate-for-loop-heuristic=<true|false>

添加一些条件来确定是否需要启用循环的分支预测优化。设为true开启该优化，默认开启。

-mllvm -enable-sroa-after-unroll=<true|false>

使能循环展开后添加SROA的功能，减少访存操作，将变量保存在寄存器中。设为true开启该优化，默认开启。

-mllvm -enable-fp-aggressive-interleave=<true|false>

对循环中A = A + B的累加场景进行优化，根据寄存器压力选择interleave值，对累加表达式做循环展开。打开开关会有浮点精度损失。设为true开启该优化，默认关闭。

-mllvm -disable-recursive-bonus=<true|false>

使递归函数中的函数调用更容易被inline，可以给调用频繁的递归函数带来更好的性能。设为true关闭该功能，默认为false，使能inline操作。

-mllvm -disable-recip-sqrt-opt=<true|false>

在fastmath场景下，对 A = (C / sqrt(Y)); B = A * A 的形式进行优化，使用更少的指令完成运算。设为true关闭该优化，默认false，使能该优化。

-mllvm -disable-loop-aware-reassociation=<true|false>

在Reassociate Pass中增加循环感知，将一些操作限制在循环边界内，避免因循环内部的指令数量增加导致的性能下降。设为true关闭该优化，默认false，使能该优化。

-Hx,70,0x20000000

O1/O2/O3时毕昇编译器使能了flang1阶段内联minloc和maxloc，内联后，函数调用成为简单的for-loop，在LLVM中可以进一步优化。使用本选项可以禁用内联功能，行为与O0一致。

-gep-common

通过删除用作索引的add指令，为来自同一指令的GEP cluster生成一个公共父代。

- **-mllvm -gep-common=<true|false>** 控制该优化，设为true开启优化，默认开启。
- **-mllvm -gep-cluster-min=<Int number>** 设置GEP cluster阈值，默认为3。
- **-mllvm -gep-loop-mindepth=<Int number>** 设置循环阈值，默认为3。

-array-restructuring

数组重排优化，改进程序中一个或多个数组的内存访问模式，进行数组重排，从而减少运行时间。

- **-mllvm -enable-array-restructuring=<true|false>** 控制该优化，设为true开启优化，默认开启。
- **-mllvm -skip-array-restructuring-codegen=<true|false>** 禁用该优化pass的指令生成部分，设为true禁用，默认false。

-struct-peel

结构体剥离优化，提高访问结构体数组中的结构字段时的局部缓存，从而减少运行时间。

- **-mllvm -enable-struct-peel=<true|false>** 控制该优化，设为true开启优化，默认开启。

- **-mllvm -struct-peel-skip-transform=<true|false>** 禁用该优化pass的指令生成部分，设为true禁用，默认false。
- **-mllvm -struct-peel-this=...** 在合法的前提下，强制剥离指定的结构体。

4 Fortran 语言引导语

毕昇编译器支持部分Fortran语言的引导语，用于指示编译器的优化行为。

!$mem prefetch

内存引用方面的引导语，指示编译器将特定数据从main memory加载到L1/L2 cache。用法：

```
!$mem prefetch array1, array2, array2(i + 4)
DO i=1,100
   array1(i - 1) = array2(i - 1) + array2(i + 1)
END DO
```

!dir$ ivdep

指示编译器忽略迭代循环的内存依赖性，并进行向量化。用法：

```
!dir$ ivdep
DO i=1, ub
   array1(i) = array1(i) + array2(i)
END DO
```

!$omp simd

指示编译器将循环转换为SIMD形式。这是一个OpenMP指令，需要指定选项‘-fopenmp’才能生效。用法：

```
!$omp simd
DO i=1, ub
   array1(i) = array1(i) + array2(i)
END DO
```

须知

该引导语当前不支持任何子句。

!dir$ vector always

通过忽略基于cost的依赖来强制编译器进行循环矢量化，立即作用于紧随其后的循环。该引导语需要添加关键字“always”，并且不支持带关键字“never”。用法：


```
!dir$ vector always
DO i=1, ub
   array3(i) = array1(i) - array2(i)
END DO
```

!dir$ novector

指示编译器不要进行循环矢量化操作，且与优化等级无关，立即作用于紧随其后的循环。用法：

```
!dir$ novector
DO i=1, ub
   array3(i) = array1(i) - array2(i)
END DO
```

!dir$ inline

指示编译器对函数进行内联操作，且与优化等级无关，立即作用于紧随其后的循环。用法：

```
!dir$ inline
real function inline_func (num)
   implicit none
   real :: num
   inline func = num + 1234
   return
end function
```

!dir$ noinline

指示编译器不要对函数进行内联操作，且与优化等级无关，立即作用于紧随其后的循环。用法：

```
!dir$ noinline
subroutine noinline_func (a, b)
   integer, parameter :: m = 10
   integer :: a(m), b(m)
   integer :: i
   do i = 1, m
      b(i) = a(i) + 1
   end do
end subroutine noinline_subr
```

!dir$ unroll

指示编译器进行循环展开操作，且与优化等级无关，立即作用于紧随其后的循环。共有3种用法：

```
!dir$ unroll  -- 全展开
DO i=1, ub
   array3(i) = array1(i) - array2(i)
END DO
!dir$ unroll (4)  -- 展开4次
DO i=1, ub
   array3(i) = array1(i) - array2(i)
END DO
!dir$ unroll = 2  -- 展开2次
DO i=1, ub
   array3(i) = array1(i) - array2(i)
END DO
```

!dir$ nounroll

指示编译器阻止循环展开操作，立即作用于紧随其后的循环，用法如下：


```
!dir$ nounroll
DO i=1, 100
    array1(i) = 5
END DO
```


5 GDB 调试

5.1 约定

使用gdb调试毕昇编译器编译程序生成的可执行文件时，存在部分功能不支持的情况，该类问题与gdb本身能力有关，毕昇编译器不作保证。推荐使用gdb 10.1版本调试毕昇Clang/Flang生成的二进制可执行文件，用户体验较好。

5.2 不支持场景（按需更新）

5.2.1 Fortran 函数入参调试信息丢失

使用gdb调试毕昇编译器编译Fortran代码产生的可执行程序时，存在函数参数的调试信息丢失的问题，导致无法用p命令查看参数内容。如图2所示，在函数内部无法打印入参alat数组内容。

图 5-1 预期输出

```
Breakpoint 1, mydepart (rr0=..., alat=..., n=10) at licm-print.F90:16
16                if (rr0(k,j) < 0.0d0) rr0(k,j) = rr0(k,j) + 2.0*cos(alat(j))
(gdb) p alat
$1 = (10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
(gdb)
```

图 5-2 实际输出

```
Breakpoint 1, mydepart (rr0=..., alat=..., n=10) at licm-print.F90:16
16                if (rr0(k,j) < 0.0d0) rr0(k,j) = rr0(k,j) + 2.0*cos(alat(j))
(gdb) p alat
$1 = ()
(gdb)
```

5.2.2 无法进入 Fortran 代码的函数内部进行单步调试

使用gdb调试毕昇编译器编译Fortran代码产生的可执行程序时，在某些场景下存在无法正常进入函数调试的问题，导致用n命令无法正常进入函数进行单步调试。如图2所示，无法通过s命令后使用n命令进行单步调试。

图 5-3 预期输出

```
(gdb) b in29.f90:53
Breakpoint 1 at 0x1284: file in29.f90, line 53.
(gdb) r
Starting program: /home/yansendao/boole-compiler/llvm-project/flang/test/f90_correct/src/a.out

Breakpoint 1, p () at in29.f90:53
53        call checkc4( cf_rslt, cf_expct, N, rtoler=(0.0000003,0.0000003))
(gdb) s
check_mod::checkc4 (reslt=..., expct=..., np=10, atoler=<error reading variable: Cannot access memory at address 0x0>, rt
    ulptoler=<error reading variable: Cannot access memory at address 0x0>, ieee=<error reading variable: Cannot access m
640           anytolerated = present(atoler) .or. present(rtoler) .or. present(ulptoler)
(gdb) n
641           ieee_on = .false.
```

图 5-4 实际输出

```
Breakpoint 1, p () at in29.f90:53
53        call checkc4( cf_rslt, cf_expct, N, rtoler=(0.0000003,0.0000003))
(gdb) s
0x0000000000216a0c in check_mod::checkc4 (reslt=..., expct=..., np=<optimized out>, atoler=<optimized out>, rtoler=<optim
    ieee=<optimized out>)
(gdb) n
Single stepping until exit from function __check_mod_MOD_checkc4,
which has no line number information.
test number 3 tolerated res 14825725.000 -12091979.000 (0x4B6238FD) (0xCB38824B) exp 14825724.000 -12091979.000 (0x4B6238
test number 4 tolerated res -17384.793 -4576.485 (0xC687D196) (0xC58F03E1) exp -17384.793 -4576.485 (0xC687D196) (0xC58F0
test number 5 tolerated res 6.142 15.724 (0x40C48C87) (0x417B96B1) exp 6.142 15.724 (0x40C48C88) (0x417B96B1)
test number 8 tolerated res 11335543.000 -13773353.000 (0x4B2CF777) (0xCB522A29) exp 11335543.000 -13773354.000 (0x4B2CF7
10 tests completed. 10 tests PASSED. 4 tests tolerated
 PASS
p () at in29.f90:54
54        call checkc8( cd_rslt, cd_expct, N, rtoler=(0.0000003_8,0.0000003_8))
```

使用lldb可以正常调试。

5.3 通过升级 gdb 版本解决部分问题

使用gdb调试毕昇编译器编译Fortran代码产生的可执行程序时，在某些场景下会报以下警告，导致无法正常调试。如图2所示，打了断点后无法进行调试。

```
BFD: warning:libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0xc0000000 from gdb
```

图 5-5 预期输出

```
(gdb) b hello.f90:1
Breakpoint 1 at 0x8ec: file hello.f90, line 1.
(gdb) r
Starting program: /home/yansendao/tmp/a.out

Breakpoint 1, MAIN__ () at hello.f90:1
1       print*,"Hello World!"
(gdb) n
 Hello World!
2       end
(gdb) q
A debugging session is active.

        Inferior 1 [process 14188] will be killed.

Quit anyway? (y or n) y
```

图 5-6 实际输出

```
(gdb) b hello.f90:1
Breakpoint 1 at 0x2109b0
(gdb) r
Starting program: /home/yansendao/tmp/a.out
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflang.so: unsupported GNU_PROPERTY_TYPE (5) type: 0x
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
BFD: warning: /home/yansendao/software/boole-compiler-binary/lib/libflangrti.so: unsupported GNU_PROPERTY_TYPE (5) type:
[Thread debugging using libthread_db enabled]
Using host libthread_db library "/lib/aarch64-linux-gnu/libthread_db.so.1".

Breakpoint 1, 0x00000000002109b0 in MAIN ()
(gdb) n
Single stepping until exit from function MAIN,
which has no line number information.
 Hello World!
0x000000000021098c in main ()
(gdb) q
A debugging session is active.

        Inferior 1 [process 60581] will be killed.

Quit anyway? (y or n) y
```

GDB版本高于8.3.1则可以正常调试。

6 已知故障与解决方法（按需更新）

6.1 约定

本章节所描述的已知故障均为OS或硬件相关，编译器无法处理，仅提供规避方案。请避免触发该类故障。

6.2 编译极复杂表达式导致栈耗尽

如下例，表达式中有10000个变量连续相加：

```
Sum += g0 + g1 + g2 + … + g9998 + g9999   // 10000个变量连续相加
```

在主机系统默认栈大小有限的情况下，可能会出现segmentation fault错误，原因是该表达式极其复杂，编译时引起Clang进程使用的栈大小超过系统限制。

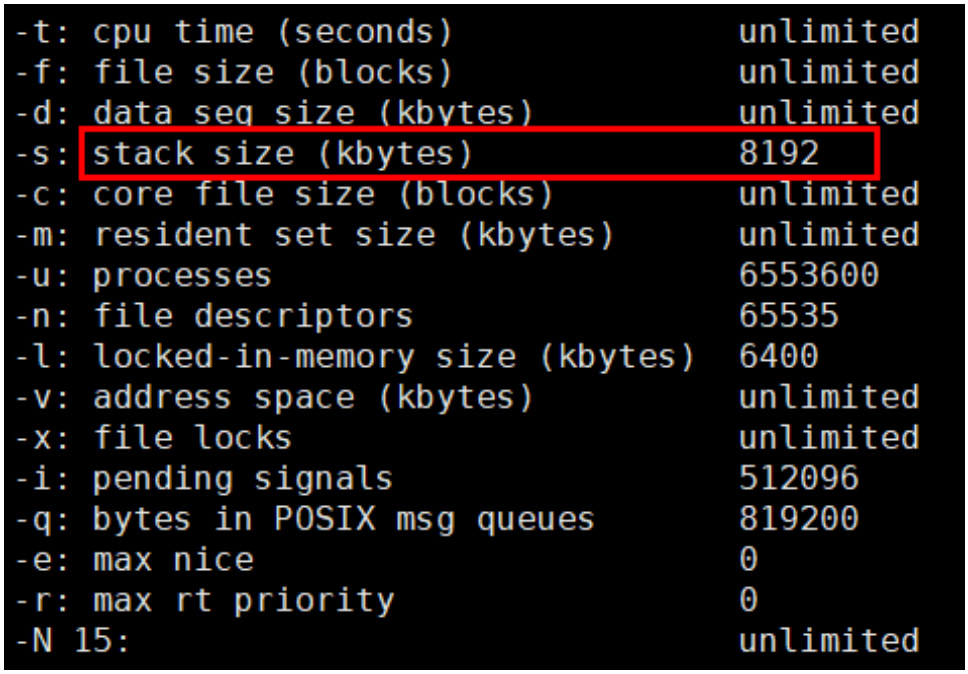
在较高的优化等级下，一部分循环会被展开，也会导致生成极复杂的表达式。

规避方案

通过ulimit –s将栈空间设大，如ulimit -s 20000（栈大小上限设为20000K），或ulimit -s unlimited（栈大小不受限），可规避该问题。

ulimit -a命令可以查看系统默认栈大小限制。

图 6-1 系统信息



7 兼容性说明

7.1 概述

毕昇编译器基于开源LLVM开发，相比GCC和ICC，LLVM前端Clang对语法的检查更严谨，严格匹配语言标准，Clang的常见兼容性和可移植性问题，请参考开源官方文档**https://clang.llvm.org/compatibility.html**。本文主要列出一些Clang相对GCC不支持的问题以及部分固有实现，以供用户参考。

7.2 Clang 不支持问题

7.2.1 不支持 print-multi-os-directory

错误信息

```
ERROR: Problem encountered: Cannot find elf_aarch64_efi.lds
```

问题介绍

GCC使用**print-multi-os-directory**该选项返回../lib64，而Clang不支持该选项，故无法组成完整的路径，因此找不到某些文件。

解决方案

在编译时，对需要通过选项获取lib64路径的代码进行硬编码。

7.2.2 不支持选项-fstack-clash-protection

错误信息

```
无报错信息
```

问题介绍

-fstack-clash-protection是一个安全编译的选项，Clang并未支持，开源社区只能查到检视中的patch：

Support -fstack-clash-protection for x86

解决方案

后续支持。

7.2.3 不支持__builtin_longjmp/__builtin_setjmp

错误信息

```
error: __builtin_longjmp is not supported for the current target
  __builtin_longjmp (buf, 1);
  ^~~~~~~~~~~~~~~~~~~~~~~~~~
error: __builtin_setjmp is not supported for the current target
  int r = __builtin_setjmp (buf);
          ^~~~~~~~~~~~~~~~~~~~~~
```

问题介绍

当前AArch64后端不支持内置函数 __builtin_longjmp和__builtin_setjmp。

解决方案

- 使用标准库中的setjmp/longjmp；
- 后续支持。

7.2.4 不支持__uint128_t

错误信息

```
clang-10: warning: unknown platform, assuming -mfloat-abi=soft
error: unknown type name '__uint128_t'
```

问题介绍

在32位模式下编译（例如添加了-m32参数）报错：__uint128不被支持。

除此之外，还可能看到一个平台无法识别的warning：使用-mfloat-abi=soft参数将浮点运算编译为软浮点。

解决方案

- 去除-m32参数，使用64位模式编译；
- 检查Makefile、configure等构建脚本的平台判断流程，使用正确的平台选项编译。

7.2.5 不支持选项-aux-info

错误信息

```
无报错信息
```

问题介绍

-aux-info选项用于将编译单元中声明或定义的所有函数(包括头文件中的函数)输出到给定的输入文件。

该选项一般用于从*.c自动生成*.h文件。

Clang 不支持这个选项。

解决方案

避免使用Clang不支持的选项。

如果确实需要这些信息，可以通过预处理器处理之后，用**sed**/**aux**等命令提取出想要的函数信息。

例如：

```
clang -E -xc test.c | sed -n 's/^extern * int *\(\w*\) *(.*$/\1/p'
```

7.2.6 不支持__ builtin___snprintf_chk

错误信息

```
error: no member named '__builtin___snprintf_chk' in
```

问题介绍

Clang暂不支持检查built-in函数格式化输出，会报出warning，当开启**-Werror**选项时升级为error。

解决方案

添加编译选项**-Wno-builtin-memcpy-chk-size**规避该告警

7.2.7 不支持 NEON 指令

错误信息

```
error: unknown register name 'q0' in asm
      : "memory", "cc", "q0"
                        ^
```

问题介绍

Clang中不支持NEON指令Q寄存器设计

代码示例

```
$ cat bar.c
int foo(void) {
```



```
    __asm__(""::"q0");
    return 0;
}
$  clang bar.c
bar.c:2:16: error: unknown register name 'q0' in asm
  __asm__(""::"q0");
               ^
1 error generated.
```

解决方案

修改qX寄存器为vX寄存器。

7.2.8 不支持部分运行库

错误信息

```
undefined reference to `__muloti4'
```

问题介绍

某符号不在libgcc中，但是在compiler-rt中，特别是使用Clang的__builtin_*_overflow家族的内联函数时。

解决方案

使用**--rtlib=compiler-rt**来启用compiler-rt，注意目前并不支持所有平台。

如果使用libc++ 或者 libc++abi，使用compiler-rt而不是libgcc_s，通过在cmake中添加**-DLIBCXX_USE_COMPILER_RT=YES**和 **-DLIBCXXABI_USE_COMPILER_RT=YES**实现。否则可能会链接两个运行时库，虽然不影响功能但是造成性能浪费。

参考LLVM官方说明：**https://clang.llvm.org/docs/Toolchain.html**。

7.2.9 不支持原子类型(atomic type)的类型转换

错误信息

```
error: used type '_Atomic(int_fast32_t)' where arithmetic or pointer type is required
                uint32_t(_Atomic(int_fast32_t))(1)
                         ^
```

问题介绍

Clang拒绝对非标准类型的类型转换，并且目前不支持对原子类型(atomic type)的转换。所支持的类型在clang/include/clang/AST/BuiltinTypes.def中有列出，不包括原子类型。

解决方案

避免对原子类型使用类型转换。

7.3 链接问题

7.3.1 指定 pic、pie

错误信息

某些动态库在未使用pic/pie选项的情况下，会报符号表缺失的错误，比如：

```
undefined reference to `cmsPlugin'
```

问题介绍

生成动态库与pie可执行文件编译或链接时未带对应pic/pie选项，导致符号表缺失，需要手动指定。

代码示例

执行如下命令检查是否为PIC库，检索到textrel符号则表明是PIC库。

```
readelf -a libhello.so |grep -i textrel
```

检查是否为PIE共享文件，使用file命令查看，或者用**size --format=sysv**查看基地址是否在0附近。

解决方案

编译选项增加**-fPIC**，链接选项增加**-pie**。

须知

Clang严格区分编译和链接选项，不能通过cflags将-pie传递给链接器，否则会报错：

```
clang-10: error: argument unused during compilation: '-pie' [-Werror,-Wunused-command-line-argument]
```

7.3.2 给链接器参数加上-Wl

错误信息

```
clang-10: error: unknown argument: '-znow'
clang-10: error: unsupported option `--whole-archive`
clang-10: error: unsupported option `--no-whole-archive`
clang-10: error: unknown argument: `-soname`
```

问题介绍

Clang和GCC的选项传递实现有点区别，有一些传给链接器的参数必须添加**-Wl**，才能传递给链接器。

包括但不限于：

- -znow
- --whole-archive
- --no-whole-archive
- -soname

如果出现unknown argument或者unsupported option，并且该选项是应该传给链接器的，则需要加上**-Wl**。

解决方案

这些参数前面添加**-Wl,**。例如：

```
-Wl,-znow -Wl,--whole-archive -Wl,--no-whole-archive -Wl,-soname
```

7.3.3 Clang 不再默认传递--build-id 到链接器

错误信息

```
ERROR: No build ID note found in xxx.so
```

问题介绍

为了避免额外链接器开销，Clang 不再默认传递**--build-id**到链接器。

解决方案

编译目标源码时增加**-Wl,--build-id**选项可以临时传递。

7.3.4 系统 libstdc++库版本过低导致符号未定义或运行结果错误

错误信息

无法找到高版本C++标准库函数定义的接口：

```
undefined reference to `std::xxx`
```

问题介绍

Clang默认使用系统路径下的libstc++.so动态库，过低的系统libstdc++.so库版本可能不支持用户代码中使用的高版本特性，导致链接时出现未定义符号或运行结果错误。

解决方案

链接时加入-stdlib=libc++或-lc++选项，使用Clang提供的libc++.so库中提供的标准C++库实现。

7.4 其它类兼容问题

7.4.1 Clang 预处理器结果与 GCC 存在较大差异

错误信息

- 格式错误：syntax error, unexpected IDENT
- 找不到头文件

问题介绍

Clang的预处理器实现和GCC有比较大的不同，例如：

- Clang会保留每行开头的空白符；

- Clang会保留引入的头文件的绝对路径；
- 其它的不一一列举。

有一些程序会使用预处理器来处理源码文件，但是因为Clang和GCC的预处理器的行为有一些不同，可能会因此导致一些问题。

解决方案

修改源码使得其能被Clang的预处理器正确处理。例如：

- 删除代码行前的空白符；
- 保证include的文件能被找到。

7.4.2 Clang 不支持在使用'-o'指定输出时直接添加头文件

错误信息

```
clang test.h test.c -o test
clang-10: error: cannot specify -o when generating multiple output files
```

问题介绍

Clang不支持在使用'-o'指定输出文件时直接添加头文件，但允许在编译命令中使用预编译头文件，以减少编译时间。

```
$ cat test.c
#include "test.h"
```

生成预编译头文件的命令：

```
clang -x c-header test.h -o test.h.pch
```

可以通过添加 '-include' 命令使用预编译头文件：

```
clang -include test.h test.c -o test
```

Clang会先检查test.h对应的预编译头文件是否存在；如果存在，则会使用对应的预编译头文件对test.h进行处理，否则，Clang会直接处理test.h的内容。

如果设法在Clang编译命令中保留头文件，则可以通过添加命令'-include'的方法使其通过编译。

官方参考文档：**https://clang.llvm.org/docs/UsersManual.html**

解决方案

避免在 Clang 的编译命令中直接添加头文件，或者按照上述方法使用预编译功能。

7.4.3 不同编译器对 built-in includes 的实现不同

错误信息

```
error: typedef redefinition with different types ('__uint64_t' (aka 'unsigned long') vs 'UINT64' (aka 'unsigned long long'))

error: unknown type name 'wchar_t'
```


问题介绍

某些头文件（例如“stdatomic.h、stdint.h”）是由编译器实现的，不同的编译器对于这些文件的实现存在差异，因此使用GCC头文件实现的程序，切换到Clang之后，用户自定义的代码可能会与Clang头文件发生冲突。

比如说重定义问题：在Clang的某些头文件中定义了一些在对应GCC的头文件中没有定义的变量，而用户在自己编写的或引入的其它库的头文件也定义了该变量，变量被重复定义，导致redefinition错误。

又或者：在GCC的built-in头文件中定义了一些变量，而Clang对应的头文件中没有定义该变量，用户在自己编写的代码中直接使用了该变量，结果就会导致unknown type的错误。

解决方案

建议修改源码。

7.4.4 不同编译器链接的 OpenMP 运行时库不同

错误信息

- 测试错误
- 无法加载OpenMP运行时库

问题介绍

Clang编译的可执行程序链接的OpenMP运行时库叫libomp.so，GCC链接的叫libgomp.so。

解决方案

确保能够找到libomp.so：

- 将libomp.so所在的目录（例如{$INSTALLATION_HOME}/lib，INSTALLATION_HOME为安装根目录）添加到环境变量**LD_LIBRARY_PATH**；
- 或者安装libomp：
  ```
  yum install libomp -y
  ```

7.4.5 __builtin_prefetch 语义检查错误

错误信息

```
error: argument to '__builtin_prefetch' must be a constant integer
  __builtin_prefetch(address, forWrite);
  ^
```

问题介绍

在这段代码中，因为**__builtin_prefetch**的第二个参数需要是常量，所以先用**__builtin_constant_p**检查forWrite是否是常量。但是，对于Clang而言，会出现语义检查错误。

代码示例

```
static void prefetchAddress(const void *address, bool forWrite) {
    if (__builtin_constant_p(forWrite)) {
        __builtin_prefetch(address, forWrite);
    }
}
```

解决方案

将函数转换为宏函数：

```
##define prefetchAddress(address,forWrite) do{\
  if (__builtin_constant_p(forWrite)) {      \
    __builtin_prefetch(address, forWrite);   \
  }                                          \
}while(0)
```

7.4.6 找不到符号 perl_tsa_mutex_lock

错误信息

```
Can't load 'xxx.so' for module threads: xxx.so: undefined symbol: perl_tsa_mutex_lock at xxx
```

问题介绍

在文件/usr/lib64/perl5/CORE/perl.h中有如下的定义：

```
##if ...
    defined(__clang__)
    ...
##  define PERL_TSA__(x)   __attribute__((x))
##  define PERL_TSA_ACTIVE
##else
##  define PERL_TSA__(x)   /* No TSA, make TSA attributes no-ops. */
##  undef PERL_TSA_ACTIVE
##endif

##ifdef PERL_TSA_ACTIVE
EXTERN_C int perl_tsa_mutex_lock(perl_mutex* mutex)
  PERL_TSA_ACQUIRE(*mutex)
  PERL_TSA_NO_TSA;
EXTERN_C int perl_tsa_mutex_unlock(perl_mutex* mutex)
  PERL_TSA_RELEASE(*mutex)
  PERL_TSA_NO_TSA;
##endif##endif
```

由于针对Clang使用的mutex相关的符号是有线程安全标记的**perl_tsa_***，但是libperl.so并不包含这些符号，故而出现链接错误。

解决方案

- 使用包含**perl_tsa_***符号的libperl.so（在编译libperl.so时，加上宏**USE_ITHREADS**和**I_PTHREAD**）；
- 去除预定义宏__clang__:

```
clang -U__clang__ ...
```


7.4.7 Clang 宏问题

问题介绍

程序代码逻辑使用了__GNUC__的宏作为判断依据，但是GCC与Clang中定义的宏内容不一致，可以使用如下命令确认Clang中宏定义的值。

```
clang -x c /dev/null -dM -E >clang.log;cat clang.log|grep '__GNUC__'
```

解决方案

若宏内容不一致导致报错，可以在编译选项加入`**-D__GNUC__=x**`进行适配修改。

7.4.8 支持的 Attributes 集合

毕昇编译器只支持Clang框架中的attributes，请参考链接：**https://clang.llvm.org/docs/AttributeReference.html**

链接中未提到attributes暂不支持。

7.4.9 -march 选项在架构扩展特性上的使用说明

Clang使用架构扩展特性时，需在**-march=arch_name**后加上扩展特性的名称，包括架构默认支持的扩展特性。例如，DotProd特性是Armv8.4架构默认支持的，可使用**-march=armv8.4-a+dotprod**使能该特性。

7.4.10 -mgeneral-regs-only 选项的使用说明

使用该选项时将生成仅使用通用寄存器的代码，这会阻止编译器使用浮点或高级SIMD寄存器。因此当编译时加入该选项，编译器应避免使用浮点运算指令，如果程序中有浮点运算，毕昇编译器将会调用compiler-rt中的库函数进行运算。因此，链接时要配合添加**-rtlib=compiler-rt -l gcc_s**选项。

7.4.11 -ffixed-line-length-n 选项的使用说明

-ffixed-line-length-n该选项中的“n”在Flang中暂时只支持“72”或“132”，若“n”为其他值会导致编译失败。因此当编译选项中“n”不为“72”或“132”时，需要将其修改为“72”或“132”。

7.4.12 -mllvm -unroll-runtime 选项的使用说明

毕昇编译器默认使能-unroll-runtime选项，可能会对编译时间产生一定的影响。因此，当对编译时间敏感时，建议加上选项-mllvm -unroll-runtime=false。

7.4.13 依赖操作系统 libatomic 库的使用说明

毕昇编译器当前基于LLVM 12.0.1版本开发，且构建发布时使用了自举构建来保证版本质量。当前开源LLVM12及以上版本在自举构建的编译器版本使用时新增了对操作系统环境中libatomic库的依赖。因此如操作系统中没有安装该依赖库，需要用户自行安装，可以通过如下方式解决：

1. 在操作系统中通过

```
sudo yum install libatomic
```

或者

```
sudo apt install libatomic
```

下载安装；

2. 通过下载rpm包解决，下载地址：

https://rpmfind.net/linux/rpm2html/search.php?query=libatomic&submit=Search+...&system=&arch=

7.4.14 128 位浮点数学库支持说明

鲲鹏场景硬件浮点运算只支持到64位浮点，超过64位的浮点计算精度取决于使用的软浮点库精度。默认使用环境中已有的四精度浮点数学库。如有需要也可以加 --rtlib=compiler-rt 选项使用compiler-rt库中的四精度浮点数学库。

7.5 Flang 兼容性

7.5.1 Fortran 与 C 互操作，main 函数双重定义

毕昇编译器在install_path/lib/目录下有libflangmain.a库，此静态库提供一个main函数供操作系统调用。所以Fortran程序只需要有自己的program函数就可以了。

当在Fortran与C互操作场景下，用户可能在C程序中提供自己的main函数，这样在链接时可能出现main双重定义的情况，如下

```
/bin/ld: c_call_fortran.o: in function `main':
c_call_fortran.c:(.text+0x0): multiple definition of `main';  /install/bin/../lib/
libflangmain.a(flangmain.c.o):flangmain.c:(.text.main+0x0): first defined here
```

这时，只需要在编译链接选项中加入**-fno-fortran-main**即可。

7.5.2 Flang 中调用 getpid()等系统调用需额外声明

gfortran可以直接调用getpid()。毕昇编译器需要加上对应的声明才能正确调用getpid()，同样的函数还有getuid()，加上对应的声明才能正确调用。

```
PROGRAM test
  INTEGER :: getpid

  print *, getpid()   !gfortran 只需要这一行，毕昇编译器需要上面的声明
end PROGRAM test
```

7.5.3 Flang 中将算术表达式直接作为 NORM2 函数的输入时需要额外选项

gfortran可以直接将类似 x+y （ x和y都是数组 ）的算术表达式作为NORM2的参数，即NORM2(x+y)，但是毕昇编译器语法检查更为严格，需要转换为以下表达：

```
temp = x + y
NORM2(temp)
```

如果不用temp，那么需要在编译时加上特定**xflag**，即**-Hx,70,0x200000**，它的作用是：

- When inlining code for F90 sum-like reduction intrinsics, don't use a temp if the argument is 'simple-enough' to evaluate in-line, and the DIM argument is one.

7.5.4 Flang 对多维数组的大小支持有限

对于多维数组，当每一维的元素个数比较多，生成的IR大于2GB时，会编译失败。失败信息如下：

```
clang::FileID clang::SourceManager::createFileID(const clang::SrcMgr::ContentCache*, llvm::StringRef, clang::SourceLocation, clang::SrcMgr::CharacteristicKind, int, unsigned int): Assertion `NextLocalOffset + FileSize + 1 > NextLocalOffset && NextLocalOffset + FileSize + 1 <= CurrentLoadedOffset && "Ran out of source locations!"' failed.
```

或如下：

```
fatal error: sorry, this include generates a translation unit too large for Clang to process.
```

7.6 Intrinsic Procedures

Intrinsic procedures和编译器的具体实现相关，本章只列出兼容性和其他编译器有差异部分的Intrinsic procedures。

7.6.1 etime is not an intrinsic function

错误信息

```
F90-S-0126-Name etime is not an intrinsic function (../INSTALL/second_INT_ETIME.f: 53)
```

问题介绍

Clang比GCC检查更加严格，DTIME 或者 ETIME 函数非标准Fortran函数，他们是通过固有函数CPU_TIME 和SYSTEM_CLOCK构造而成。而GCC忽略掉了ETIME非固有函数的错误只进行告警。

解决方案

参考网上用例手写DTIME或者ETIME函数：

```
! ETIME in standard Fortran.
Real Function etime(time)
  Real time(2)
  Call Cpu_Time(etime)
  time(1) = etime
  time(2) = 0
End Function
```

7.6.2 CPU_TIME

```
CALL CPU_TIME(TIME)
```

CPU_TIME返回代码执行的CPU时间，以秒为单位。如果获取失败，返回参数TIME=-1.0。

7.6.3 SYSTEM_CLOCK

```
CALL SYSTEM_CLOCK([COUNT, COUNT_RATE, COUNT_MAX])
```

SYSTEM_CLOCK函数可用于计算elapsed time，计算方式是两次调用SYSTEM_CLOCK函数并获取二者的差值。这个Intrinsic函数的返回值，依赖不同编译器的实现。

COUNT：可选的Integer类型输出参数，表示当前的处理器时钟计数，该值的范围为[0, **COUNT_MAX**]，达到最大值**COUNT_MAX**后，会从0开始重头计数。

COUNT_RATE：可选的Integer或者Real类型输出参数，表示处理器每秒的clock次数。

- 如果kind=4或者kind=2时，**COUNT_RATE**为1000，**COUNT**的值代表毫秒milliseconds，如果从0开始计数，大约25天COUNT计数会再次重置为0。
- 如果kind=8或者更大，**COUNT_RATE**为1000000000，**COUNT**代表纳秒。

当入参均为INTEGER时，所有参数的kind应当相同。不支持kind=1的情况，此时**COUNT**, **COUNT_RATE, COUNT_MAX**全返回0。

如果获取系统时钟失败，**COUNT=-HUGE(COUNT)**, **COUNT_RATE**=0, **COUNT_MAX**=0。

7.6.4 RTL 库函数

作为fortran语言的前端，Flang内部包含有RTL(run time library)库函数例程清单，RTL库函数清单所含的函数为运行时阶段内嵌库函数。在用户层面使用fortran语言进行编码时，应避免调用同名函数，以防止函数重定义的出现。

接口举例

```
FtnRteRtn ftnRtlRtns[] = {
{"f90io_aux_init", "", false, ""},
   {"f90io_backspace", "", false, ""},
   {"f90io_begin", "", false, ""},
.....
```

使用举例

这些IO运行时函数，命名以特殊的f90io_为前缀，错误的使用案例如下：

```
INTEGER FUNCTION f90io_sc_i_ldw(i, j)
  IMPLICIT NONE
  INTEGER, INTENT(in):: i
  INTEGER, INTENT(INOUT):: j
  print *, i,j
  j = j + 1
END FUNCTION f90io_sc_i_ldw
```

该用例使用fortran语言‘print’的上述RTL里的运行时库函数f90io_sc_cf_ldw，导致编译时出现如下错误：

```
F90-W-0155-f90io_sc_i_ldw - PURE subprograms may not contain external I/O statements
(f90io_sc_i_ldw.f90: 5)
  0 inform,   1 warnings,   0 severes, 0 fatal for f90io_sc_i_ldw
F90-S-0000-Internal compiler error. gen_funcret: illegal dtype, sym       0  (f90io_sc_i_ldw.f90: 7)
F90-S-0000-Internal compiler error. get_llvm_name: bad stype for       0  (f90io_sc_i_ldw.f90: 7)
.....
F90-S-0000-Internal compiler error. get_llvm_name: bad stype for       0  (f90io_sc_i_ldw.f90: 7)
F90-F-0000-Internal compiler error. make_lltype_from_sptr(), no incoming arguments       0
(f90io_sc_i_ldw.f90: 7)
```

7.6.5 Neon Intrinsic

Neon Intrinsic和编译器的具体实现相关，Clang的neon Intrinsic的功能与官方文档《Arm Neon Intrinsics Reference》（以下简称ANIR文档）一致。

但生成ANIR文档上指定的汇编指令，需指定优化级别大于O0。

ANIR文档获取链接：**https://developer.arm.com/documentation/ihi0073/g**

使用举例

以内容如下的test.c为例：

```
#include <arm_neon.h>
int32x2_t test_vsudot_lane_s32(int32x2_t r, int8x8_t a, uint8x8_t b) {
  return vsudot_lane_s32(r, a, b, 0);
}
```

表 7-1 ANIR 文档上 vsudot_lane_s32 的描述

Intrinsic	Argument Preparation	Instruction	Result	Supported Architectures
int32x2_t vsudot_lane_s32(int32x2_t r, int8x8_t a, uint8x8_t b, const int lane)	r -> Vd.2S a -> Vn.8B b -> Vm.4B 0 <= lane <= 1	SUDOT Vd.2S,Vn.8B,Vm.4B[lane]	Vd.2S -> result	A32/A64

使用命令**clang -march=armv8.6-a+i8mm test.c -O0 -S**生成的结果是以**mov**、**dup**和**usdot**等多条指令组合的形式。

```
test_vsudot_lane_s32:                   // @test_vsudot_lane_s32
// %bb.0:                               // %entry
        sub     sp, sp, #112            // =112
        str     d0, [sp, #72]
        str     d1, [sp, #64]
        str     d2, [sp, #56]
        ldr     d0, [sp, #72]
        str     d0, [sp, #48]
        ldr     d0, [sp, #64]
        str     d0, [sp, #40]
        ldr     d0, [sp, #56]
        str     d0, [sp, #32]
        ldr     d0, [sp, #32]
        str     d0, [sp, #16]
        ldr     d0, [sp, #48]
        ldr     d1, [sp, #16]
                                        // implicit-def: $q3
        mov     v3.16b, v1.16b
        dup     v1.2s, v3.s[0]
        ldr     d2, [sp, #40]
        str     d0, [sp, #104]
        str     d1, [sp, #96]
        str     d2, [sp, #88]
        ldr     d0, [sp, #104]
        ldr     d1, [sp, #96]
        ldr     d2, [sp, #88]
        usdot   v0.2s, v1.8b, v2.8b
        str     d0, [sp, #80]
        ldr     d0, [sp, #80]
        str     d0, [sp, #24]
        ldr     d0, [sp, #24]
        str     d0, [sp, #8]
        ldr     d0, [sp, #8]
        add     sp, sp, #112            // =112
        ret
```

使用命令**clang -march=armv8.6-a+i8mm test.c -O1 -S**生成结果与ANIR文档一致。

```
test_vsudot_lane_s32:                   // @test_vsudot_lane_s32
// %bb.0:                               // %entry
                                        // kill: def $d2 killed $d2 def $q2
        sudot   v0.2s, v1.8b, v2.4b[0]
        ret
```

7.7 HPC Workload 应用支持范围

应用名称	版本号
GRAPES	3.2
GFS	14.1.7
NEMO	3.6
CESM	2.1.1
OpenFOAM	1906
QE	6.4.1
Gromacs	2019.3
Lammps	5-6月-19
CP2K	7.1
WRF	4.2.1

8 附录

8.1 问题反馈

在使用过程中遇到问题，需要技术支持时，请反馈问题信息至**毕昇论坛**。

8.2 修订记录

发布日期	修订记录
2021-12-30	第一次正式发布。文档内容更新如下： 1）更新软件包版本号； 2）浮点运算控制选项章节修改1个描述 3）Flang兼容性章节新增1个问题； 4）兼容性说明新增2个问题； 5）新增HPCworkload应用支持范围