PhyCC 1.0 用户手册

（V1.0）

飞腾信息技术有限公司
www.phytium.com.cn
2024 年 03 月 27 日

商标声明

Phytium 和其他飞腾商标均为飞腾信息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。

特别提示

本文档仅作为使用指导，飞腾对本文档内容不做任何明示或暗示的声明或保证。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。

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，如有变更，恕不另行通知。

目录

1 PhyCC 1.0 简介

PhyCC1.0 是基于 LLVM14 研发的一款 C/C++/Fortran 语言高性能编译器，兼容主流 llvm 特性，针对飞腾微架构订制了一系列优化，并集成性能更优的 malloc、math 等基础库，是一款 Linux 下深度适配飞腾 CPU 的高性能编译器，该编译器当前仅支持 AArch64 架构。

2 PhyCC 1.0 编译器使用

PhyCC1.0 是一款针对 C,C++，Fortran 程序的高性能编译器，该编译器优化任务分为 2 大类，一类是与架构相关的优化（如 AArch64 架构的飞腾处理器）,另一类是与架构无关的优化。

Phycc1.0 编译器可以用来编译多种程序语言，C 和 C++程序请使用 clang 编译，Fortran 程序请使用 Flang 编译器，当前编译器仅支持 64 位模式。

2.1 编译器组成及编译过程介绍

PhyCC1.0Clang 和 Flang 在编译程序时，包含预处理、词法分析、语义分析、优化、汇编和链接等过程，Clang 和 Flang 作为驱动，会调用预处理、编译器、汇编器、链接器等工具完成上述编译步骤。

Clang 和 Flang 是高度集成的驱动，用户理解编译过程的各个阶段，对了解编译器非常重要。这些阶段按以下顺序执行：

2.1.1 驱动

Clang 作为前端工具，主要功能是将一个程序编译为 LLVM 中间表示（IR），此外，clang 也作为驱动，会驱动预处理器，编译器、汇编器、连接器运行。与 Clang 类似，Flang 是 Fortran 前端编译器，由以下两个组件组成：

· flang1：由前端驱动程序调用，负责将 Fortran 程序转换为标记。解析器将这些标记转换为抽象语法树（AST）。然后将 AST 转换为用于生成 ILM 代码的规范形式。

· flang2：使用 flang1 生成的 ILM 代码，并将其转换为 ILI，然后由内部优化器进行优化。优化后的 ILI 然后转换为 LLVMIR。然后，前端驱动程序将此 LLVMIR 传输给 LLVM 优化器进行优化和目标代码生成。

针对一般程序员，您可以将 Clang 和 Flang 用作端到端的驱动程序。但是，对于需要了解编译过程的开发人员，您可以手动执行每个编译阶段。

2.1.2 预处理

预处理阶段处理输入源文件的标记化、宏展开、#include 展开和处理其他预处理器指令。该阶段的输出通常称为.i（对于 C）、.ii（对于 C++）或.i（对于 Fortran）文件。

2.1.3 词法分析和语义分析

该阶段解析输入文件，将预处理器标记转换为解语法树。当代码以语法树的形式表示时，能对表达式的计算类型应用语义分析，并确定代码是否形式良好。该阶段负责生成大多数编译器警告和解析错误。Clang 的这个阶段的输出是一个抽象语法树（AST）。使用 Flang，Flang1 将被调用来将程序标记转换为 AST，然后转换为规范形式-用于生成 ILM 代码。

2.1.4 LLVM IR 代码生成

在 Clang 中，这个阶段将 AST 转换为低级中间代码（LLVMIR）。使用 Flang，Flang2 接管了 Flang1 生成的 ILM 代码，并将其转换为 ILI，然后经过内部优化器优化，最后转换为 LLVMIR。

2.1.5 Phycc 优化器

该阶段负责优化生成的 LLVMIR，主要是与架构无关的代码优化。

2.1.6 机器码生成

该阶段从优化的 LLVMIR 执行特定于目标的代码生成。这个阶段的输出通常称为.s 或汇编文件。

Clang 和 Flang 还支持使用集成汇编器，代码生成器直接产生目标文件。这避免了生成.s 文件然后调用目标汇编器的开销。

2.1.7 汇编器

该阶段目标是将汇编代码转成目标对象文件。这个阶段的输出通常称为.o 或对象文件。

2.1.8 链接器

该阶段链接器将多个对象文件合并成可执行文件或动态库。输出通常称为 a.out、.dylib 或.so 文件。

2.2 标准支持

PhyCC 1.0 支持如下语言标准：

- C:

  C17 standard (ISO/IEC9899:2018) (Default)

C11 standard (ISO/IEC9899:2011)

C99 standard (ISO/IEC9899:1999)

●C++:

C++20 standard (ISO/IEC14882:2020)1

C++17 standard (ISO/IEC14882:2017) (Default)

C++14 standard (ISO/IEC14882:2014)

C++11 standard (ISO/IEC14882:2011)

C++98 standard (ISO/IEC14882:1998)

●Fortran:

Fortran-95 (ISO/IEC1539:1997)

Fortran-2003 (ISO/IEC1539:2004)

Partial support of Fortran-2008 (ISO/IEC1539:2010)2

●OMP 4.5 and OMP 5.0 standards for C/C++ programming3

●OMP 4.5 standards for Fortran programming

●DWARF 5 standards for C,C++,and Fortran debuggability

3 PhyCC 1.0 安装说明

3.1 环境准备

基于 Ubuntu 20.04.5 LTS 操作系统，AArch64 机器。

3.2 安装 PhyCC 1.0

以下操作均使用 root 用户权限执行

a).创建任意安装目录,如/xxx/。

```
$mkdir-p/xxx/
```

b).拷贝二进制包

```
$cpphycc.tar.gz/xxx/
```

c).解压缩二进制包

```
$cd/xxx/&&tarxfphycc.tar.gz
```

d).配置环境变量

执行以下命令，对于非 root 用户，需要再重启机器或注销登录后环境变量才能生效：

```
$echo “exportPATH=/xxx/phycc/bin:$PATH” >>/etc/profile
$source/etc/profile
```

e).配置基础库

将 libftmath、libftmalloc 库拷贝至系统库默认搜索路径下，如“/lib”或“/usr/lib”。

f).编译器版本确认执行以下命令检查 PhyCC 的版本：

```
$clang--version
```

如果返回结果中包含 PhyCC 的版本信息，如“phycc1.0...”，则表明 PhyCC 已安装成功。

3.3 PhyCC 1.0 使用示例

3.3.1 clang 和 clang++ 使用

将 C 语言编写的源码编译为可执行文件 a.out：

```
$clanga.c-oa.out
$./a.out
```

将 C++编写的源码编译为可执行文件 a.out：

```
$clang++a.cpp-oa.out
$./a.out
```

3.3.2 Flang 使用

将 Fortran 编写的源码编译为可执行文件 a.out：

```
$flanga.f-oa.out
$./a.out
```

3.3.3 使用 LLD 链接器

编译时启动 LLD 链接器对程序进行链接：

```
$clang-fuse-ld=llda.c-oa.out
$./a.out
```

3.3.4 -lftmalloc 链接飞腾 malloc 库

适配飞腾处理器的内存分配库,在链接选项中加入-ftmalloc,可使用高性能库优化程序。

3.3.5 -lftmath 链接飞腾数学库

适配飞腾处理器的基础数学库，与-lm 组合使用，如：-lftmath-lm，可使用高性能数学库。

4 PhyCC 1.0 编译和链接选项介绍

4.1 架构选项

4.1.1 -mcpu=ft862

使用飞腾硬件架构所订制流出模版调度指令，生成更优的指令排布。

4.2 驱动选项

4.2.1 -mllvm

通过-mllvm 标识，选项通过前端传递到 opt，使能某个优化特性。例如-mllvm-struct-array-layout。

4.2.2 -fuse-ld=lld

启动 lld 链接器对程序进行链接。

4.3 优化等级选项

4.3.1 -O0

无优化,该级别编译速度最快，生成的代码可调试性最强。

4.3.2 -O1

开启 O0 与-O2 之间的一些优化。

4.3.3 -O2

开启大多数稳定编译器优化 pass。

4.3.4 -O3

开启所有优化，为了拿到最佳优化效果，某些优化选项会增加代码 codesize。

4.3.5 -Os

开启优化 pass 与-O2 类似，但进行了额外的优化以减少代码量。

4.3.6 -Oz

开启优化 pass 与-Os（以及-O2）类似，但进一步减小了代码大小。

4.3.7 -O5

开启优化 pass 与-O3 类似，但进一步加入激进的优化 pass,提升整型程序性能。

4.3.8 -O5fast

开启优化 pass 与-O3 类似，但进一步加入激进的优化 pass,提升浮点型程序性能。

4.3.9 -lto

适用于多文件做链接时优化，将多个文件优化信息共享，在链接时进行更多编译优化。

4.4 优化选项

4.4.1 -fstruct-peel

clang 选项，开启结构体剥离优化（需开启 LTO），将结构体拆解成冷热子结构体，优化 Dcache 布局。

4.4.2 -enable-struct-reordering=true

编译选项，开启结构体成员重排序优化（需开启 LTO），默认为关闭状态。

4.4.3 -struct-redundancy-elim=true

编译选项，开启结构体冗余成员删除优化（需开启 LTO），默认为关闭状态。

4.4.4 -struct-array-relayout

编译选项，使用-Wl,-mllvm,-struct-array-relayout 传递给优化器（需开启 LTO）,默认为关闭状态。主要对特定场景的结构体数组进行拆分，如果数组长度小于 2^16 等长度，在此基础上进行指针压缩，进一步提升 Dcache 利用率。

4.4.5 -min-vectorize-width=

编译选项：循环向量化中，并入同一矢量类型的最小数据个数。提高该阈值用于提高向量寄存器的利用效率。默认为零。

循环向量化的最小宽度。

- 参数类型：整数

4.4.6 -loop-distribute-rtcheck=true

循环拆分时使能运行时校验,默认为关闭状态。

4.4.7 -min-prefetch-cover=

循环数据预取的最小缓存覆盖率。

- 参数类型：整数
- 参数值范围：0≤n≤8
- 默认值：0

4.4.8 unroll-runtime-param

优化程序 runtime-unroll 展开阈值，当循环体内参数的数量小于该值，扩大 runtime-unroll 展开阈值。

4.4.9 address-step-relevancy

优化程序 runtime-unroll 展开阈值，当循环体内的聚合体类型参数的数量大于该值时，优化展开阈值策略。

4.4.10 -enable-function-specialization

对函数进行特异化，对多个常量调用点生成不同的实现副本；添加对多个参数的识别和特异化处理。

4.4.11 乘法相关指令优化

根据操作数的类型生成更符合该操作数的性能更高的指令（默认开启）。

4.4.12 为特定后端添加 lsl 相关特性

在指令选择时可以选择更好的指令（默认开启）。

4.4.13 -instcombine-cttz-table=true

对使用 deBruijn 序列查找数据最低位 1 所在位置的运算进行优化（默认开启）。

4.4.14 -argpromote-bitcast-arg=true

在参数提升优化过程中添加对指针参数的使用是 bitcast+load 场景的识别与处理，优化 load 指令（默认开启）。

4.4.15 -inline-minloc-maxloc

lang 选项，在 lang 对 minloc/maxloc 库函数进行处理时内联原本的库函数调用，优化库函数性能。

4.4.16 -function-specialization-for-literal-constant=true

函数特异化对标量等简单常量的参数进行优化，默认为关闭状态。

4.4.17 -func-specialization-max-iters=

函数特异化进行迭代的次数，多次迭代会尝试对新生成的特异化函数进行特异化优化。

- 参数类型：整数
- 默认值：1

4.4.18 -sink-funcspec-pass=true

调整函数特异化 PASS 的位置，能增加函数特异化的优化场景与优化效果。默认为关闭状态。

4.4.19 -foutput-intentin

lang 选项，对声明了 INTENT(in)属性的参数在生成 LLVMIR 时输出 intentin 属性关键字。

4.4.20 -foutput-internal

lang 选项，对作用域只在当前模块内部的接口添加 internal 属性关键字，便于后续 pass 做优化处理。

4.4.21 -instcombine-fold-not-into-cmp=true

减少冗余的条件判断，精简代码逻辑。默认为关闭状态。

4.4.22 -enable-customized-pipeline=true

开启自定义 pipeline，优化表达式重组策略，可更好的促进后续优化。默认为关闭状态。

4.4.23 -aarch64-insert-extract-base-cost=

由用户指定插入和提取代价值，方便用户调整向量计算开销,默认值为 3。

4.4.24 -fveclib=FTMATH

编译选项，指定向量数学库，调用到向量库函数，提高程序执行效率。

4.4.25 -loop-bound-split

编译选项，使用-mllvm-loop-bound-split=true 将循环迭代范围拆分为多个连续部分，每个连续部分对应一个新的循环，循环迭代范围拆分后，循环体内某些分支指令得以消除，使得循环体可以得到进一步优化。

A 更新记录

发布日期	说明
2024-03-27	初稿