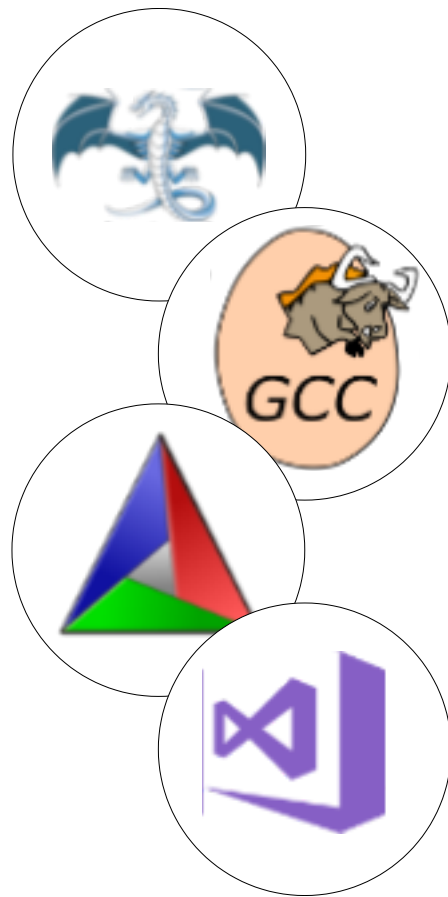


Modern C++ Toolchain

从零开始的配 C++ 工具链生活

CUHKSZ-HPC 技术分享

Mar. 24th, 2023



编译器

- g++: 老牌编译器 GCC 的一部分, GPL 和 LGPL 许可证, 最普适, 支持冷门 CPU 架构, linux 系统默认必备, 一般认为是标准。
- clang++: LLVM 的一部分, Apache 2.0 许可证, 属于宽松许可证, 模块化, 支持静态分析和代码格式化, 还支持 cuda 编程。
- MSVC: 微软出品, 免费但专有, 下载麻烦, Windows 编程必备 (使用 Windows API 的 C++ 程序)。

编译器

- Apple Clang: 和 LLVM 是一家，但是很苹果，不建议使用，使用原版就行。
- EDG eccp: 您好！买！
- Intel® oneAPI DPC++/C++: 开源，对英特尔 CPU 和 GPU 特化，在 HPC 中优化非常离谱地好，后端是 LLVM。不支持为 AMD CPU 优化。
- IBM XLC++: 您好！买！看起来不是很现代的样子。
- Sun/Oracle C++: 免费但专有。看起来不是很现代的样子。
- Embarcadero C++ Builder: 您好！买！
- Cray: 找了半天没找到下载地址。好像是为了 Cray System 专门设计的，后端是 LLVM。
- Nvidia HPC C++: GPU 编程，你就说你要不要吧。
- Nvidia nvcc: GPU 编程，你就说你要不要吧。

编译器选择

在绝大多数情况下，我推荐你使用 clang (报错全而友好)

```
#include <iostream>
int main() {
    int value = 4;
    const char *s = "Hello, World!" + value;
    std::cout << s << std::endl;
    return 0;
}
```

```
> g++ -Wall -Wextra -Wpedantic warning.cpp # gcc version 12.2.1 20230201
> clang++ -Wall -Wextra -Wpedantic warning.cpp # clang version 15.0.7
warning.cpp:4:35: warning: adding 'int' to a string does not append to the string [-Wstring-plus-int]
    const char *s = "Hello, World!" + value;
                                ~~~~~^~~~~~
warning.cpp:4:35: note: use array indexing to silence this warning
    const char *s = "Hello, World!" + value;
                                ^
                                &      [      ]
1 warning generated.
```

<https://easyaspi314.github.io/gcc-vs-clang.html>

编译器选择

在绝大多数情况下，我推荐你使用 clang

易于下载

无论是 Windows, Linux 还是 MacOS 都支持 clang++。

g++ 不行么？

Windows 上需要使用 mingw, mingw 原版还不维护了，需要使用 mingw-w64，这个软件还不能单独下，需要和 cygwin 和 msys2 配合使用。

其他方面

截至 2023/4/4，两者编译出来的二进制文件运行速度互有胜负。

<https://benchmarksgame-team.pages.debian.net/benchmarksgame/fastest/c.html>

构建系统

- GNU Make: 使用大家熟悉的 Makefile 构建 C++ 项目。

优点：大家都在用，很多老牌项目如 Linux, Sqlite 依然在用。

缺点：手写麻烦，坑很多，比如不能使用空格缩进，必须用 `tab` 缩进。

- Ninja: 新出现的构建系统，构建速度比 GNU Make 快很多。

优点：速度快，并行化构建。构建时提示友好，不会像 GNU Make 一样一直输出，让整个 terminal 没法看。

缺点：基本上不支持手写，不支持字符串相关操作，需要和元构建系统配合使用。

对于使用现代化元构建系统的项目，因为可选择生成 GNU Makefile 或 Ninja build file，为了生成速度我都推荐使用 Ninja 作为构建系统。












元构建系统

- Autotools: 一个老牌的元构建系统，可以生成 GNU Makefile。
- CMake: 一个跨平台的构建系统，可以生成 GNU Makefile 或 Ninja build file。
- Bazel: Google 出品，可以生成 GNU Makefile 或 Ninja build file。
- Meson: 一个跨平台的构建系统，可以生成 GNU Makefile 或 Ninja build file。
- GN: Google 出品，可以生成 Ninja build file。

Autotools

我们不会重点去谈这个构建系统，但是因为这个构建系统的历史悠久，很多老项目都在使用，所以我们需要知道它的一些基本用法。

比如 HPC 常用的集群管理软件 Slurm。

 Makefile.am	Add make clean-contrib
 Makefile.in	Add auxdir/gtk-2.0.m4 so we can bring it up to date.
 NEWS	Fix typo for initialized in description for ESLURM_PLUGIN_NOT_LOADED.
 README.rst	Docs - Remove DejaGNU from documentation
 RELEASE_NOTES	Fix typo for initialized in description for ESLURM_PLUGIN_NOT_LOADED.
 aclocal.m4	Add auxdir/gtk-2.0.m4 so we can bring it up to date.
 config.h.in	switch/hpe_slingshot - use new cxil_get_svc_rsrc_list routines
 configure	Merge cons_common into cons_tres.
 configure.ac	Merge cons_common into cons_tres.
 make_ref.include	Fix Makefile *.bino generation error for FreeBSD.
 slurm.spec	Merge branch 'slurm-23.02'

以 ArchLinux 软件仓库的 PKGBUILD 为例，我们可以看到它的构建过程：

```
$ autoreconf

$ ./configure \
  --disable-developer \
  --disable-debug \
  --without-rpath \
  --enable-optimizations \
  --prefix=/usr \
  --sbindir=/usr/bin \
  --sysconfdir=/etc/slurm-llnl \
  --localstatedir=/var \
  --enable-pam \
  --with-hdf5 \
  --with-hwloc \
  --with-rrdtool \
  --with-munge

$ make -j
```

第一步重新使用 `autoreconf` 是为了防止开发者在修改后忘了重新生成 `configure` 脚本，或者目录里根本没有该脚本。

CMake

C++ 元构建系统事实上的标准。

```
$ cmake \  
-DCMAKE_EXPORT_COMPILE_COMMANDS:BOOL=TRUE \  
-DCMAKE_BUILD_TYPE:STRING=Debug \  
-DCMAKE_C_COMPILER:FILEPATH=/usr/bin/clang \  
-DCMAKE_CXX_COMPILER:FILEPATH=/usr/bin/clang++ \  
-DCMAKE_CXX_FLAGS:STRING="-Wall -Wextra -Wpedantic" \  
-DCMAKE_LINKER:FILEPATH=/usr/bin/ld.lld \  
-Bbuild \  
-G Ninja \  
.  
  
$ cd build  
  
$ ninja
```

有用的工具

- clangd: 基于 clang 开发的 LSP 服务器，可以提供语法高亮，自动补全，跳转等功能。

为什么你不该使用 微软的 vscode-cpptools 下的 cpp intellisense?

```
public int strength;  
public int intelligence;  
public int |outtelligence;
```

不管是功能、速度还是稳定性，clangd 都全方位吊打 vscode-cpptools。

如果期望使用 AI 技术补全代码的话，我的建议是使用 Github Copilot 而不是 vscode-cpptools。

clangd 使用的是 clang 的报错方式，报错信息更全更友好，warning 不用编译就能看到，节省大量时间。

有用的工具

- CodeLLDB: 基于 LLDB 开发的 extension, 提供调试功能。

和 vscode-cpptools, 我体验下来, 除了语法不一致, 并无太大区别。

特别地, 因为 lldb-mi 被废弃了, 所以如果你要使用 lldb 调试的话, 为了方便, 只建议使用 CodeLLDB。

GDB 的话, 我推荐使用 vscode-cpptools。

- CMake && CMake Tools

提供一些 CMake 语法补全, 和快捷操作的图形化接口, 比较方便, 倒也不是必须。

有用的工具

- Catch2

一个测试框架，比 Google Test 更加易用。比如创建窗口，销毁窗口这样的代码，只需要写一次就可以应用在其下的所有测试上。

- Perf

Linux 上的性能分析工具，可以用来分析程序的性能瓶颈。

一般配合 cargo-flamegraph 使用，可以生成火焰图。

Linux 用户可以尝试使用 <https://github.com/KDAB/hotspot>，图形化，功能强大。

- cargo-flamegraph

flamegraph 火焰图的 rust 实现，易用性上有全方位的提升。

<https://github.com/flamegraph-rs/flamegraph>

