CPUBench测试指导书

计算产品性能基准工作组

2024年3月

目录

一、 CPUBench 介绍

1.1 工具介绍

CPUBench 作为一款开放的通用计算 CPU 性能评测基准工具，其测试负载来源于 HPC、大数据，数据库等常用通用计算领域的典型业务场景，定位于对通用计算场景下的 CPU、内存子系统以及所依赖的编译器进行综合计算能力评估。

CPUBench 工具由计算产品性能基准工作组设计开发，并在 2021 年世界计算机大会上进行发布，参与单位包括 CPU 厂商、整机厂商、用户单位、研究机构等 100 余家单位。工具详细介绍及性能测试结果数据库见计算产品性能基准工作组官网：www.cppb-wg.com

1.2负载设计

CPUBench 根据计算类型（整型、浮点型）和并发数（单并发、多并发）设计四大测试套件，每个套件支持 typical 和 extreme 两种测试模式，形成八大性能指标，如下表所示：

测试套件	性能指标	指标含义
IntSingle	int_single_typical int_single_extreme	评估单核整型运算能力
IntConcurrent	int_concurrent_typical int_concurrent_extreme	评估多核整型运算能力
FloatSingle	float_single_typical float_single_extreme	评估单核浮点运算能力
FloatConcurrent	float_concurrent_typical float_concurrent_extreme	评估多核浮点运算能力

表 1-1 测试套分类及介绍

- typical 模式：要求测试套件中相同语言的负载必须采用相同的性能优化措施，体现性能优化的通用性。
- extreme 模式：允许不同的负载采用不同的性能优化措施，体现该系统极限优化的性能水平。

从各种典型的业务场景中抽取计算密集型部分，覆盖各种应用运行特征，覆盖主流编程语言 C、C++、Fortran、Java。具体 workload 如下：

测试套件	测试负载	应用领域
IntSingle IntConcurrent	x264	视频编码
	gcc	编译器
	gzip	数据压缩
	tpcc	事务处理
	tpch	在线分析
	kmeans	机器学习-聚类
	wordcount	大数据-统计
	velvet	基因拼接
	openssl	加解密
	rapidjson	数据序列化
	python	Python 解析器
	xz	数据压缩

表 1-2 整型测试套负载及介绍

测试套件	测试负载	应用领域
FloatSingle FloatConcurrent	lightgbm	机器学习-决策树
	nektar	计算气动声学
	phenglti	计算流体力学
	phyml	基因分析
	gromacs	分子动力学-有机物
	povray	光线追踪
	openfoam	计算电磁学
	lammps	分子动力学-无机物
	cube	天体物理-N 体模拟
	wrf	气象-天气预报

表 1-3 浮点型测试套负载及介绍

1.3 度量方法

CPUBench 的各子项计分规则如下：

$$W_i = R_{time} / E_i$$

W_i为各子项得分；

R_{time}为参考时间，基于 Intel Skylake 3106 CPU（1.7GHz 8Core）的 HPE DL380 Gen10 服务器测试得出的时间；

E_i为各个子项实际运行的时间。

CPUBench 的测试套总分为各子项得分的几何平均值：

$$\mathrm{G} = \sqrt[n]{\prod_{i=1}^{n} Wi}$$

二、 CPUBench 测试

2.1 环境要求

运行 CPUBench 之前，请参照表 2-1 确认运行环境是否符合要求。

类别	配置要求	说明
软件要求	C、C++、Fortran 编译器，版本>=5.3.0。 JDK。 make。	CPUBench 内置 x86_64 、aarch64 、mips64 、sw_64、loongarch64、ppc64le 架构 CMake 和 Python3 (包括 PyPDF2 和 reportlab)工具，用户只需安装 C、C++、Fortran 编译器、JDK、perl 及 make。 说明： 如果操作系统环境不能联网，工具安装过程中需要操作系统镜像文件用于安装依赖包，请自行获取对应操作系统版本的镜像文件。 Redhat 系： yum install -y perl make gcc gcc-c++ gcc-gfortran java Debian 系： apt-get install -y perl make gcc g++ gfortran default-jdk Suse： zypper in -y perl make gcc gcc-c++ gcc-fortran java*
硬件要求	工具运行需要约 2G 内存每实例。 工具运行时/dev/shm 目录至少需要 3G 空间。 运行 Single 模式时至少需要 20G 硬盘空间，运行 Concurrent 模式时，每实例增加 2G 硬盘空间。	运行目录默认为 CPUBench/work_dir，用户可手动配置。

表 2-1 CPUBench 运行环境要求

2.2 安装说明

CPUBench 采用免安装模式，将 CPUBench 软件包 CPUBench-vx.x.x.tar.gz 上传到被测系统并解压即可：

```
# tar -zxvf CPUBench-vx.x.x.tar.gz
```

2.3 配置选项

根据被测服务器系统信息及测试目的，修改 config/config-template.ini 文件里的配置项。主要关注以下配置选项，更详细的配置请参考 2.6 节：

选项	默认值	作用域	含义
action	standard	common	指定本次工具的活动。完整流程：standard 编译：build 运行：run 清理数据：clean
benchmarks	[]	common	指定本次运行的子项集合或套件
tune	typical	common	调优级别，可以选择经典(typical)或最优(extreme)。
jobs	1	common	运行实例个数，1 表示单核单实例，大于 1 表示多核多实例。
iterations	3	common	运行的迭代次数，最后得分取均值。
taskset	[]	common	绑核列表，由逗号分隔的一系列 cpu 核心编号构成，同时支持通过-分隔符来标识一个编号范围，如 1-5 等价于 1,2,3,4,5。
lang_dir	/usr/bin	namespace	编译器的 bin 目录
LD_LIBRARY_PATH	[]	namespace	若用户使用非系统自带编译器，需配置编译器 lib 库完整路径
CC	${lang_dir}/gcc	namespace	使用的 C 语言编译器
CFLAGS	-O3	namespace	使用的 C 语言编译选项
CXX	${lang_dir}/g++	namespace	使用的 C++语言编译器
CXXFLAGS	-O3	namespace	使用的 C++语言编译选项
LIBS	[]	namespace	链接库
AR	ar	namespace	使用的静态库制作工具，如不使用系统默认，请注意配置完整路径
RANLIB	ranlib	namespace	静态库的符号索引表更新工具，如不使用系统默认，请注意配置完整路径
FC	${lang_dir}/gfortran	namespace	使用的 fortran 语言编译器
FFLAGS	-O3	namespace	配置 fortran 语言编译选项
java_options	[]	namespace	JVM 配置参数

表 2-2 CPUBench 主要配置项说明

主要规则说明：

- 若 benchmarks 配置为多并发测试套(IntConcurrent、FloatConcurrent)，jobs 必

须大于 1，否则强制运行单并发测试套(IntSingle、FloatSingle)。若 benchmarks 配置为单并发测试套，jobs 无论配置为多少，均强制运行单并发测试套。

- 设置 taskset 时，当该列表长度大于 jobs 时，只会选前 jobs 的 cpu 核进行绑定，若小于 jobs，则会在超出列表长度时，从列表头重新绑定。
- java_options 选项传递给 JVM 的调优参数，通过 spark.driver.extraJavaOptions 生效。禁止通过 spark.driver.extraJavaOptions=-Xmx 设置最大堆容量，故禁止在本配置项设置-Xmx。
- iterations 必须设置为 3，否则测试报告无效
- action 必须为 standard，否则测试报告无效
- 必须包含 typical 模式下的测试结果，否则测试报告无效

2.4 运行说明

切换到 CPUBench 的安装目录，直接执行 cpubench.sh。

 # ./cpubench.sh [options] [arguments]

具体运行选项和参数，可通过./cpubench.sh -h 查看，详细说明如下：

选项	含义
--benchmarks/-b	指定测试项
--action/-a	指定测试阶段
--config/-c	指定配置文件，可以是绝对路径或文件名或去后缀名，如： /CPUBench/config/config-template.ini 或 config-template.ini 或 config-template
--jobs	指定运行副本数
--tune/-T	指定测试模式
--iterations/-i	指定运行轮次
--work_dir	指定工作目录
--version/-V	输出工具版本
--verbose/--debug	指定是否开启 debug 日志
--rebuild/-R	指定负载运行前是否重新编译
--perf	指定是否开启性能采集
--perf_target/--ptarget	指定性能采集类别
--skip_verify	指定是否跳过工具校验
--help/-h	输出帮助信息

表 2-3 CPUBench 主要配置项说明

若相同的配置项既出现在命令行又出现在配置文件，以命令行为主。

以下为 3 个运行举例：

 # ./cpubench.sh -c config-template.ini -b x264 gzip xz -i 3 --jobs 8

表示指定 config 目录下的 config-template.ini 为配置文件，仅测试 int_x264，int_gzip，int_xz 三个子项，各子项迭代测试 3 次，各子项的实例为 8 个

 # ./cpubench.sh -c config-template.ini -b x264 -a build

表示指定 config 目录下的 config-template.ini 为配置文件，仅对 int_x264 子项进行编译测试

 # ./cpubench.sh -c config-template.ini -b x264 --skip_verify 1

表示指定 config 目录下的 config-template.ini 为配置文件，仅对 int_x264 子项进行测试，且跳过工具校验，一般修改工具源代码，进行调试时，增加该编译选项，但会导致最终的测试报告无效。

2.5 结果说明

2.5.1 生成内容说明

CPUBench 测试报告包含六个部分：

- 运行日志（log）：CPUBench 运行日志与终端日志相同，记录工具的运行情况。
- 环境信息文件（CPUBench_Environment.txt）：记录当次运行所采集的环境信息，方便用户更全面的了解运行环境各参数的配置情况。
- 当次运行配置文件（ini）：用户使用--config 或-c 指定的配置文件，若用户未指定，则默认为 config-template.ini 文件。
- 中间文件（json）：为 Json 格式文件。该文件包含测试信息，软硬件信息以及负载运行信息（得分，时间，运行轮次等）。
- PDF 报告：以 PDF 测试套得分、用户配置测试信息、软硬件信息、负载运行信息、构建运行信息以及优化选项。
- HTML 报告：以 HTML 的格式显示的报告，内容同 pdf 保告。
- CSV 报告：记录测试负载运行信息，包括负载最终得分，运行副本数，负载每轮运行时间及运行得分，以及测试套得分。

2.5.2 PDF 报告/HTML 报告说明

PDF/HTML 报告会标识本次运行是否有效，若您需要合法的报告，须确保以下行为：

- 必须包含 typical 模式下的测试结果。
- 必须开启工具自校验。
- 必须运行整个测试套。

- 必须将迭代次数设置为 3。
- typical 模式下同语言测试负载必须拥有相同的优化选项。
- 测试套中所有负载的运行结果必须有效。

2.6 详细配置说明

2.6.1 配置文件说明

- 配置文件定义了一系列配置项，指导了 CPUBench 的编译、运行、日志和发布等行为，是 CPUBench 与测试环境的重要交互方式。
- 想要复现某个测试结果，需满足三个条件：
  1) 相同版本的 CPUBench 工具，并通过相同的方式调用它。
  2) 相同的软硬件环境。
  3) 相同的配置文件。
- 配置文件是 ini 格式的文件，遵循业界通用的 ini 文件语法。

2.6.2 配置文件规则

1. 大小写敏感。
2. 配置文件提供#或;注释功能，注释不会被视为配置项。
3. 缩进代表本行内容为前一个配置项的值。如：

```
hardware_others = CPU Prefetching: Enable
                  Turbo: Enable
software_others = swapoff -a
```

此时 hardware_others 的值为：

```
CPU Prefetching: Enable\nTurbo: Enable
```

2.6.3 配置文件结构

整个配置文件分为 common 区和命名区：

- 通用区以“[common]”打头，配置通用配置项，比如 action、benchmarks 等
- 命名区以一个区域标识行打头，区域标识行由一到二个字符串通过&相连。

  如：“[IntConcurrent&typical]”，多个命名区可能有相同的配置项，此时需要根据命名区优先级规则来决定配置项的取值。详细规则可参考命名区规则

2.6.4 通用配置项

配置文件提供了与命令行相似的通用配置项，可以有效降低命令行的复杂度。如在配置文件中定义：

action = standard
benchmarks = IntConcurrent
tune = typical
那么以下两条命令行指令完全一致：
 # ./cpubench.sh -c test.ini
 # ./cpubench.sh -c test.ini -a standard -b IntConcurrent -T typical

详细通用配置项说明如下：

选项	默认值	作用域	含义
action	standard	common	指定本次工具的活动。
benchmarks	[]	common	指定本次运行的 benchmarks 或 suite。
tune	typical	common	调优级别，可以选择经典(typical)或最优(extreme)。
tag	""	common	用于标记可执行文件，构建目录和运行目录。
jobs	1	common	支持在多个核上跑多个复制。
verbose	0	common	值为 1 时，开启 debug 模式。
work_dir	work_dir	common	设置工作目录，所有运行数据都会存放到该路径下。
iterations	3	common	要运行的迭代次数。
taskset	[]	common	绑核列表，由逗号分隔的一系列 cpu 核心编号构成，同时支持通过-分隔符来标识一个编号范围，如 1-5 等价于 1,2,3,4,5。当该列表长度大于 jobs 时，只会选前 jobs 的 cpu 核进行绑定，若小于 jobs，则会在超出列表长度时，从列表头重新绑定。
arch	auto	common	指定 CPU 架构，auto 表示自动识别，否则用户手动指定参数。该选项用于不规范 CPU Architecture 命名情况。
rebuild	0	common	值为 1 时，表示在运行前重新编译负载。

表 2-4 CPUBench 通用配置项说明

2.6.5 测试和环境信息

配置文件支持描述配置项，这类配置项方便用户理解测试信息。

选项	默认值	作用域	含义
cpu_name	[]	common	制造商确定的处理器名称。
machine_name	[]	common	机器名。
cpu_max_mhz	[]	common	芯片供应商指定的 CPU 的最大速度，以 MHz 为单位。
cpu_nominal_mhz	[]	common	芯片供应商指定的 CPU 速度，以 MHz 为单位。
disk	[]	common	运行目录的磁盘信息。
memory	[]	common	主内存的大小。
l1_cache	[]	common	1 级（主）缓存。
l2_cache	[]	common	2 级缓存。
l3_cache	[]	common	3 级缓存。
manufacturer	[]	common	硬件制造商。
os	[]	common	操作系统的名称和版本。
compiler	[]	common	编译器的名称和版本。
file_system	[]	common	运行目录的文件系统（ntfs，ufs，nfs 等）。
run_level	[]	common	运行级别。
ptrsize	[]	common	基准测试使用的指针位数。
huge_page_size	[]	common	大页内存。
transparent_huge_pages	[]	common	透明大页内存。
test_date	[]	common	测试时间。
test_sponsor	[]	common	赞助此测试的实体。
license_id	[]	common	许可证编号。
software_available_time	[]	common	软件发布时间。
hardware_available_time	[]	common	硬件发布时间。
software_others	[]	common	其他软件相关信息。
hardware_others	[]	common	其他硬件相关信息。

表 2-5 CPUBench 测试和环境信息配置选项

该配置选项规则如下：

- 优先级：用户填写 > 系统获取
- 若用户不填写 manufacturer 字段，报告显示为 Unknown/未知。
- 若用户不填写 test_date、test_sponsor、license_id、software_available_time、hardware_available_time、software_others、hardware_others 字段，报告显示为空。
- 其余字段信息在用户未填写的情况下，工具会自动获取，若获取不到，则报告显示为 Unknown/未知。

2.6.6 编译选项

配置文件提供配置项控制 workload 的编译行为。

选项	默认值	作用域	含义
LD_LIBRARY_PATH	[]	namespace	动态链接库搜索路径设置。
CC	/usr/bin/gcc	namespace	如何调用 C 编译器。
CC_VERSION	-v	namespace	如何获取 C 编译器版本。
CFLAGS	[]	namespace	既不是优化也不是可移植性的 C 语言编译标志。
CXX	/usr/bin/g++	namespace	如何调用 C++编译器。
CXX_VERSION	-v	namespace	如何获取 C++编译器版本。
CXX_FLAGS	[]	namespace	既不是优化也不是可移植性的 C++语言编译标志。
FC	/usr/bin/gfortran	namespace	如何调用 Fortran 编译器。
FC_VERSION	-v	namespace	如果获取 Fortran 编译器版本。
FFLAGS	[]	namespace	执行 Fortran 语言编译选项。
CLD	[]	namespace	在编译 C 语言的程序时，如何调用链接器。

CXXLD	[]	namespace	在编译 C++语言的程序时，如何调用链接器。
FLD	[]	namespace	在编译 Fortran 语言的程序时，如何调用链接器。
CLD_FLAGS	[]	namespace	适用所有 C Workload 使用的链接标志。
CXXLD_FLAGS	[]	namespace	适用所有 CXX Workload 使用的链接标志。
FLC_FLAGS	[]	namespace	适用所有 Fortran Workload 使用的链接标志。
LD_FLAGS	[]	namespace	适用所有模块时使用的链接标志。
LIBS	[]	namespace	指定链接库选项。
AR	ar	namespace	应用于创建静态库，但是启用特殊选项（如 LTO）时，需要编译器自带 AR 工具（如 gcc-ar）。
RABLIB	ranlib	namespace	应用于静态库建立索引，以便编译器加速查找符号表。启用特殊选项（如 LTO）时，需要使用编译器自带的 RANLIB 工具（如 gcc-ranlib）。

表 2-6 CPUBench 编译选项配置说明

2.6.7 JAVA 运行参数

配置文件提供配置项以方便用户进行 JVM 调优。

选项	默认值	作用域	含义
java_options	[]	namespace	传递给 JVM 的调优参数，通过 spark.driver.extraJavaOptions 生效。

表 2-7 CPUBench JVM 运行参数配置说明

2.6.8 调试选项

配置文件提供配置项控制 workload 运行过程的性能数据采集行为。调试选项仅在开发和测试过程使用，采集数据会对负载运行有一定影响，请勿随意使用。

选项	默认值	作用域	含义
perf	0	common	是否执行采集任务，0 不采集，1 采集。
perf_timeout	60	common	当 workload 运行结束后，继续等待 perf_timeout 指定的时间，若采集任务还未结束，则强制终止采集。单位：秒。
perf_info	{}	common	用户自定义的采集命令。 该选项属于可变选项，格式为 perf_{target}={command}。其中{target}可由用户自定义，用于解释采集目标，区分采集数据。{command}为字符串，识别为一条采集命令。用户需要在采集命令中通过{pid}来预置传 PID 的位置。若采集命令不是通过标准输出展示采集信息，通常会通过-o 选项来指定数据的保存路径。用户可直接在命令中指定目录，也可通过{output}来使输出文件保存到框架默认的采集数据存储目录。 说明： 框架内置采集命令为： CPUBENCH#IO: pidstat 1 -d -p {pid} CPUBENCH#CPU: pidstat 1 -p {pid} CPUBENCH#MEM: pidstat 1 -r -p {pid} CPUBENCH#STAT: perf stat -ddd -p {pid} -o {output}/perf.txt CPUBENCH#HOTSPOT: perf record -a -s -d -p {pid} -o {output}/perf.data
perf_target	[]	common	当用户只想执行特定几个采集命令时，可以通过该选项来指定这

			些命令。选项的值由逗号分隔的采集目标构成，例如：perf_target = IO,MEM。

表 2-8 CPUBench JVM 调试选项配置说明

2.6.9 命名区规则

- 命名区以区域标识行打头，延续到下一个标识行。
- 配置文件支持多个命名区，它们的顺序不会影响配置项的最终值。
- 如果区域标识行有重复，它们标识命名区的内容会自动合并。
- Typical 模式下，测试套件中相同语言的负载需配置相同调优选项，Extreme 模式下，不同负载可根据自身调优方式，配置不同调优选项。

区域标识行的格式如下，分为两个标识符，如“[benchmark&tune]”包含 suite 以及特定的 workload 名称。如果区域标识行的尾部标识是 default，可以省略。如：“[int_x264&default]” 等同于 “[int_x264]”

配置文件通过优先级规则来组织命名区的区域合并。优先级规则作用于标识行内的两个标识符。

对 benchmark 标识符，优先级如下：

- 优先级：workload > suite > default
- 对 tune 标识符，优先级：高优先级 typical = extreme > default
- 若按照以上进行各个标识符的优先级排序后，区域内的配置项之间没有冲突，则直接合并，若有冲突，按照以下标识符间优先级来判断，优先级一致，则以最后设置的值为主，优先级：benchmark > tune

举例如下：

```
[default]
CC = ${lang_dir}/gcc
CC_VERSION = -v
CXXFLAGS = -O0

[IntConcurrent]
CXXFLAGS = -O3

[default&extreme]
CXXFLAGS = -O2
```

第一条命令：

```
# ./cpubench.sh --benchmarks x264 --jobs 3
```

指定了 benchmark，会匹配前两个命名空间，由于优先级冲突，选择优

先级大的-O3。
第二条命令：

 # ./cpubench.sh --benchmarks x264 --jobs 1 --tune extreme

指定了 tune 和 benchmark，按照各个标识符优先级排序，会匹配第一和第三个命名空间，但是由于配置项冲突，则按照标识符间优先级排序，tune 优先级更高，选择-O2。

2.7 常见问题

2.7.1 系统时间戳引起的相关问题

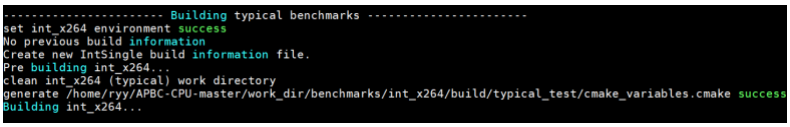
CPUBench 工具依赖 CMake 构建机制编译测试负载，在 Linux 环境，CMake 生成 Makefile 脚本，通过 make 执行 Makefile 进行编译构建，在编译过程中扫描源代码文件，根据文件时间戳判断文件是否发生变化进行编译，当系统时间未设置正确时间（比如默认 1970-1-1 01:01:01），而 CPUBench 软件包里的相关源文件的时间戳是当前时间（比如 2022-3-15 01:01:01 等），会导致 CMake 误以为文件出现变化需要重新编译，在复杂工程里容易出现编译死循环。

1）x264 长时间编译

- 现象：

  X264 编译时屏幕回显长时间停留 building。

  图 9-1 x264 长时间编译



- 可能原因：

  系统时间在 CPUBench 工具包的时间戳之前。

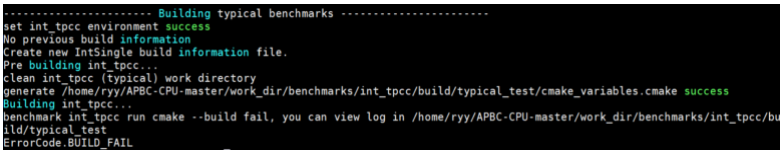
- 解决办法：修改系统时间到当前时间，重新运行 CPUBench 工具即可

2）tpcc/tpch 编译失败

- 现象：

  运行 CPUBench 工具编译 tpcc 负载或者 tpch 负载时失败。

  tpcc/tpch 编译失败：



- 可能原因：

  系统时间在 CPUBench 工具包的时间戳之前，查看 mysql 本身的日志如下：

  mysql 报错日志：

- 解决办法：

  修改系统时间到当前时间，重新运行 CPUBench 工具即可。

3）时间戳导致的其他 cmake 编译问题

- 现象：

  编译第一个负载就报编译错误。

- 可能原因：

  查看环境中当前时间和 CPUBench 工具包时间，当前时间是否在工具包的时间戳之前。

- 解决方法：

  修改系统时间到当前时间，重新运行 CPUBench 工具即可。

2.7.2 依赖工具、编译器、JDK 相关问题

1）源码安装 cmake 失败

- 现象：

  在龙芯或申威 cpu 架构上源码安装 cmake 失败，Utilities\KWIML\include\kwiml\abi.h 报错，无法识别 CPU 架构的字节序

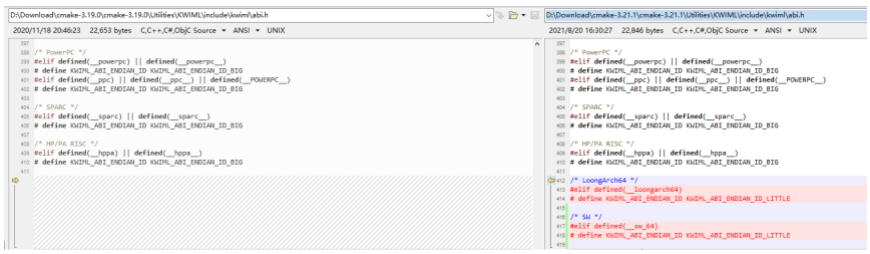
  ```
  # error "Byte order of target CPU unknown."
  ```

- 原因：

  低版本的 cmake 不支持最新的 loongarch64、sw_64 架构。

- 解决方法：

  使用 cmake3.21 版本的源码编译安装可解决龙芯的源码安装问题，申威架构需要修改 abi.h 文件，如下图所示：



2）cmake 自动去重问题

- 现象：

  使用类似-mllvm flag1 -mllvm flag2...类型的编译选项时，会出现编译报错 unrecognized command line option。

  查看编译日志，发现实际应用的编译选项变为-mllvm flag1 flag2，丢失了第二个-mllvm 导致 flag2 识别失败。

- 原因：

  多个具有重复字段的编译选项时，由于 cmake 自身具有自动去重的机制，会造成 config 中传递的编译选项的字段损失，导致编译失败。

- 解决方法：

  CMake 官方方案为通过增加“SHELL:”前缀避免去重，本工具延用该方法，且由于自身逻辑要求，通过以下格式进行防去重操作：

  -O3; SHELL: -mllvm flag1; -mllvm flag2; -mllvm flag3

3）aarch64 架构 OpenJDK8 运行 wordcount 偶现测试结果异常

- 现象：

  在 aarch64 架构上，使用 OpenJDK8 运行 workload 时，约 10%概率触发 JDK 的 bug，导致测试时长异常增加至正常的 3 倍左右。

- 原因：

  这是 JDK8 的 bug，详细请查看下面链接

  [8183543: Aarch64: C2 compilation often fails with "failed spill-split-recycle sanity check"]

  http://hg.openjdk.java.net/jdk9/jdk9/hotspot/rev/f739cf1a4ab8

- 解决方法：

  使用 openJDK11，或手工合入 patch 重新构建 JDK。

4）Java workload 在使用 jdk17 时运行失败

- 现象：

  java workload(kmeans, wordcount)在使用 jdk17 时，运行失败。

- 原因：

  当前 java workload 包含基于 spark 的 kmeans 和 wordcount，选取的 spark 版本为 spark-3.0.1，该版本需要添加额外的 jdk 参数才能支持 jdk17（更高版本均如此）。

- 解决方法：

  增加额外 jdk 选项: --add-opens=java.base/sun.nio.ch=ALL-UNNAMED。

5）tpcc 或其他负载使用-flto 选项编译失败

- 现象：

  在配置文件添加 CFLAGS/CXXFLAGS/FFLAGS=-flto，编译失败。

- 原因：

  编译过程中静态库创建需要使用编译器自带 AR 和 RANLIB 工具。

- 解决方法：

  使用 AR=${lang_dir}/gcc-ar，RANLIB=${lang_dir}/gcc-ranlib

  AR 和 RANLIB 请根据当前使用编译器来指定，上述是 gcc 编译器的示例。

6）gcc/tpcc/tpch/gromacs 使用 llvm 系编译器+flto 编译失败

- 现象：

  在配置文件添加 CFLAGS/CXXFLAGS/FFLAGS/LD_FLAGS=-flto，且配置 llvm-ar 和 llvm-ranlib，编译提示 BUILD_FAIL，查看负载编译日志 cmake_build_out_log，有错误提示"file format not recognized"。

- 原因：

  编译过程需要指定链接器为 lld。

- 解决方法：

  使用 LD_FLAGS=-flto -fuse-ld=lld。

7）wrf 或其他负载使用-flto 选项编译失败

- 现象：

  在配置文件设置 FFLAGS=-flto，且配置编译器自带的 AR 和 RANLIB 后（比如 gcc-ar, gcc-ranlib），编译提示 BUILD_FAIL。

- 可能原因：

  缺少某些未知编译选项支持，且编译选项在-O3 中开启。

- 解决方法：

  使用 FFLAGS=-O3 -flto。

8）wrf 编译提示参数类型不匹配

- 现象：

  使用 gcc10 及以上版本编译器编译 wrf，编译失败。

- 原因：

  gfortran 10 版本以后，对代码的语法检查更严格，默认把调用参数不匹配视为 Error。

- 解决方法：

  在 FFLAGS 字段中添加-fallow-argument-mismatch -fallow-invalid-boz 编译选项。

`-fallow-argument-mismatch`

Some code contains calls to external procedures with mismatches between the calls and the procedure definition, or with mismatches between different calls. Such code is non-conforming, and will usually be flagged with an error. This options degrades the error to a warning, which can only be disabled by disabling all warnings via '`-w`'. Only a single occurrence per argument is flagged by this warning. '`-fallow-argument-mismatch`' is implied by '`-std=legacy`'.

Using this option is *strongly* discouraged. It is possible to provide standard-conforming code which allows different types of arguments by using an explicit interface and `TYPE(*)`.

`-fallow-invalid-boz`

A BOZ literal constant can occur in a limited number of contexts in standard conforming Fortran. This option degrades an error condition to a warning, and allows a BOZ literal constant to appear where the Fortran standard would otherwise prohibit its use.

9）kmeans 运行成功，最后校验数据失败

- 现象：

多实例运行 kmeans，运行结束是报“RUN_JOB_AFTER_FAIL”

- 原因：

  最后校验结果的时候报错，使用 md5sum 命令查看每个生成结果，发现有结果不一致的情况。该问题属于 openjdk11 的 bug。该 bug 已在 openeuler 社区较新的 jdk 上解决。

- 解决方法：

  更新 jdk：java-11-openjdk-11.0.22.7-0.oe2203sp2.aarch64.rpm

2.7.3 测试环境不满足工具运行必要条件

1）内存不足，多实例运行过程中卡死

- 现象：

  多实例运行 TPCC 或者 TPCH 负载时卡死。

- 可能原因：

  负载设计原则是每个负载单实例的内存占用可以在 2G 以内，多实例下服务器内存不够。

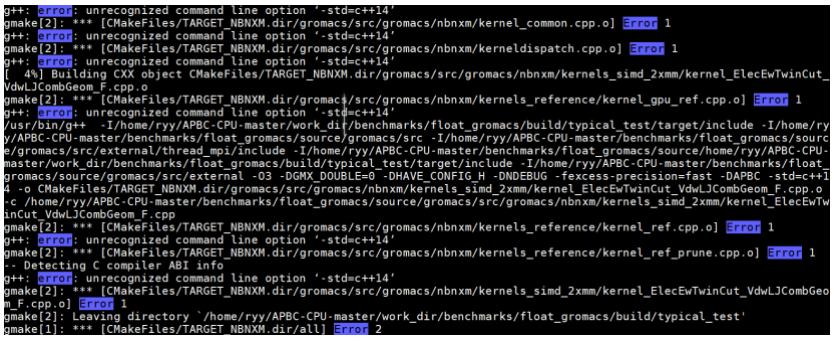
- 解决方法：

  查看环境中内存空间大小，估算多实例需要的内存大小，为服务器增加足够的内存条之后再次运行。

2）编译器版本低引起的编译错误

- 现象：

  编译负载时（例如 gromacs，tpcc，tpch 等）出现编译错误，打开 debug 日志后搜索 error 查看具体错误信息如下。



- 可能原因：

  查看环境中当前的 gcc 版本和配置文件中配置的编译器路径，有些负载使用了 c++14，需要 5.3.0 以上版本的 gcc 编译器。

- 解决方法：升级当前环境中的 gcc 版本，或者如果当前环境中存在符合条件的高版本 gcc，在配置文件里配置其二进制文件所在路径。

2.7.4 系统环境变量配置引起的相关问题

1）工具提示 UnicodeEncodeError

- 现象：

  运行 CPUBench 工具时，终端提示 UnicodeEncodeError。

- 可能原因：

  环境变量 LC_ALL 的值设定为'C'。

- 解决方法：

  执行'export LC_ALL='，将该环境变量置空，重新运行 CPUBench 工具即可。

2）编译报错，提示 lib 库不存在

- 现象：

  运行 CPUBench 工具编译 nektar 负载时提示 libxxx.so 库不存在，实际环境中存在该 lib 库。

- 可能原因：

  系统未配置 LD_LIBRARY_PATH 环境变量。

- 解决方法：将所使用编译器的 lib 库路径配置到 LD_LIBRARY_PATH 中，重新运行 CPUBench 工具即可。

3）tpcc/tpch 负载编译失败，提示“Unsupported platform”

- 现象：

  运行 tpcc/tpch 时，cmake_build_out_log 提示“Unsupported platform”。

- 可能原因：

  系统未配置 LD_LIBRARY_PATH 环境变量。

- 解决方法：将所使用编译器的 lib 库路径配置到 LD_LIBRARY_PATH 中，重新运行 CPUBench 工具即可。

4）tpcc/tpch 负载编译失败，提示“sys_gettid was not declared in this scope”

- 分析：

  1、cd work_dir/benchmarks/int_tpcc/build/typical_test 进入编译目录。
  2、查看 cmake_build_out_log 日志文件得到的提示“sys_gettid was not declared in this scope”可能并不是根因，需要进一步分析 CMake 的详细日志。
  3、进入 CMakeFiles 目录。
  4、打开 CMakeError.log 和 CMakeOutput.log 文件查看更详细的构建信息，如"error while loading shared libraries:libatomic.so.1:cannot open shared object file"。

- 解决方法：yum install libatomic

三、 CPUBench 调优

3.1 OS 调优

OS 调优手段，主要通过修改 OS 启动参数，运行时参数，对进程进行绑核等操作，从而提升 CPU 的性能。以下为常用的调优手段：

1）仅安装必须的软件，关闭无用的服务

评测时安装 Linux 时选择最小安装模式，或者手工关闭非必要的后台服务，减少后台服务进程切换引起干扰。如:

```
# systemctl disable auditd.service
# systemctl disable libvirtd
# systemctl disable firewalld
# systemctl disable irqbalance
# systemctl disable postfix
# systemctl disable crond
```

2）修改 OS 启动参数

步骤 1　vim /etc/default/grub，找到 GRUB_CMDLINE_LINUX=xx

步骤 2　在 GRUB_CMDLINE_LINUX=中添加需要的 grub 启动参数，如 selinux=0，也可以是多个参数，保存文件

```
GRUB_TERMINAL_OUTPUT="console"
GRUB_CMDLINE_LINUX="rhgb quiet crashkernel=
0me selinux=0 "
GRUB_DISABLE_RECOVERY="true"
```

步骤 3　yum 依赖的类似 CentOS 系统执行如下，grub.cfg 使用对应的文件，X86 可能是别的路径的 grub.cfg

```
grub2-mkconfig -o /boot/efi/EFI/*/grub.cfg
```

如果是 apt 的类似 Ubuntu 系统执行如下

```
update-grub
```

步骤 4　重启 OS，执行 cat /proc/cmdline 检查是否包含 selinux=0

3）绑核优化

测试时对整机进行绑核可以提升性能，如服务器有 128 核可以按照 taskset=0-127 绑定。

CPUBench 可以通过修改配置文件的方式绑核：

```
#=====================================================================
[common]
action = standard          # options: standard, build, run, clean
benchmarks = IntConcurrent     # Example: x264,gcc or lightgbm
arch = auto                # arch: auto, aarch64, x86_64, mips64el
tag = test
tune = typical             # options: typical, extreme, both
jobs = 128                   # 1: single-core performance. 1+: Multi-Core Performance.
iterations = 3
verbose = 0                # options: 0(close), 1(open)
work_dir = bench_line        # Any directory with the write permission.
rebuild = 0                # options: 0(close), 1(open)
taskset = 0-127   # Set the number of the bound CPU core.
```

3.2 编译调优

编译调优主要是从编译器的角度对应用进行优化，在编译过程中的优化器阶段，gcc 提供了近百种优化选项，用来对编译时间、目标文件长度、执行效率这个三个方向进行不同的取舍和平衡。以下为通常考虑的 3 种优化编译选项：

1）O2/O3 及相关 flag 优化

O0 为默认的编译选项，减少编译时间和生成完整的调试信息。

O1 对程序做部分编译优化，例如尝试减小生成代码的尺寸，以及缩短执行时间，但并不执行需要占用大量编译时间的优化。

O2 比 O1 更加优化，尝试更多的寄存器级的优化以及指令级的优化，提高

程序的执行速度。具体区别如下：

a. O2 会进行更多的优化，包括循环优化，常量传播等，可以使得代码更加高效。

b. O2 会使用更多的寄存器来存储变量，减少了内存的访问次数，提高了程序的执行速度。

c. O2 会对代码进行更加严格的优化，可能会导致一些代码的行为发生变化，例如浮点数的精度可能会有所损失。

O3 在包含 O2 所有的优化的基础上，使用普通函数的内联，以及针对循环的更多优化。

可通过 gcc -Q -O2 --help=optimizers 查看对应 O2 级别开启的优化列表，同理可查其他级别。若是不想使能其中一个或多个优化项，可使用-fno-xxx 选项关闭。若是同时使用多个不同级别的优化选项，会根据最后一个-O 的 level 来采用优化级别。

除了一些开关编译选项，gcc 还有一些参数型的编译优化选项，可通过 gcc -Q -O2 --help=params 命令进行查看，不同的编译选项开关，编译选项参数，能够达到不同的优化效果。如：

CFLAGS = -O3 -mcpu=native -fno-default-inline -finline-limit=400

表示开启编译器-O3 级别的优化，使编译器自动检测处理器，在函数体内定义的函数不做内联处理，限制决定函数是否能被 inline 的伪指令长度为 400。

具体的寻优组合，及各个编译选项的作用可以参考各编译器的手册，如 gcc 可参考：gcc 编译选项优化说明

2）LTO 优化

LTO（Link Time Optimization）：传统的编译方式只能将优化局限于单个编译单元（如.c 文件），链接时优化可以跨编译单元全局优化，从而实现跨文件函数内联、函数特化、常量传播等优化。链接时优化是对整个程序的分析和跨模块的优化，因此会增加链接编译时间。

- gcc 编译时添加-flto。
- 可以设置并发增加编译速度，-flto=N。

3）使能硬件 feature

一些架构本身有对应的特性能够使能优化，比如 ARM 的 CRC 循环冗余校验，CRC32 主要是计算字符串的 32 位 CRC 校验码，输入待计算的字符串，输出 32 位 CRC 多项式校验码。常用于数据正确性校验，多用在网络通信、存储等方面。CRC 的本质是模-2 除法的余数，采用的除数不同，CRC 的类型也就不一样。编译器的作用，实际上就是使能硬件 feature 的过程。

对一些内核态有大量网络通信且调用 CRC 校验的场景 workload，能够使用该硬件特性进行优化。

可用“cat /proc/cpuinfo”命令查看 ARM 硬件是否支持，在 Features 一行有 crc32 即为支持硬件指令加速。

要使用该特性，只要在 ARM 平台中增加-march=armv8-a+crc 编译选项即可，其它服务器可查看对应的硬件特性进行调优。

3.3 其它优化手段

除了上述的一些调优方法外，还可以借助一些工具，如 perf top 查看热点函数，分析热点函数，针对性地对热点函数进行优化。比如，当分析热点函数 malloc 和 free 占比较高时，可以考虑使用其它内存分配器优化，常用的有 jemalloc/tcmalloc。

另外，一些厂商也提供了对应的数学库和加速库，可以对计算密集型 HPC 进行优化，如 Intel 提供的 MKL，AMD 提供的 ACML，鲲鹏提供 KML 对浮点测试套件有所提升。