基准评测

郭健美

2024年秋

SOLE 系统优化实验室
华东师范大学

Benchmarking 基准评测

- 基准点：A surveyor's bench mark (two words) defines a known reference point by which other locations may be measured.

- 计算机性能基准评测：A computer benchmark performs a known set of operations by which computer performance can be measured.

http://spec.org/cpu2017/Docs/overview.html

为何需要基准评测?

Amazon Announces Graviton2 SoC Along With New AWS Instances: 64-Core Arm With Large Performance Uplifts

by Andrei Frumusanu on December 3, 2019 12:30 PM EST

- *All of these performance enhancements come together to give these new instances a significant performance benefit over the 5th generation (M5, C5, R5) of EC2 instances. Our initial benchmarks show the following per-vCPU performance improvements over the M5 instances:*

- *SPECjvm® 2008: +43% (estimated)*
- *SPEC CPU® 2017 integer: +44% (estimated)*
- *SPEC CPU 2017 floating point: +24% (estimated)*
- *HTTPS load balancing with Nginx: +24%*
- *Memcached: +43% performance, at lower latency*
- *X.264 video encoding: +26%*
- *EDA simulation with Cadence Xcellium: +54%*

https://www.anandtech.com/show/15189/amazon-announces-graviton2-soc-along-with-new-aws-instances-64core-arm-with-large-performance-uplifts

为何需要基准评测?

数据说话！UCloud「硬刚」腾讯云，高性能 AMD 云主机哪家强?

2020-05-28 17:49

5月25日，网络上有两家 IT 自媒体分别发布了一篇关于云主机能力的开发者测评文章，分别为 **《腾讯云 vs Azure vs UCloud：基于AMD EPYC™ ROME云主机选型指南》**和 **《高性能AMD云主机如何选？AWS、谷歌云、UCloud、腾讯云测试大PK》**。

https://www.sohu.com/a/398302358_115128

阿里云开发者大会技术解读：JVM性能提升50%背后的秘密武器

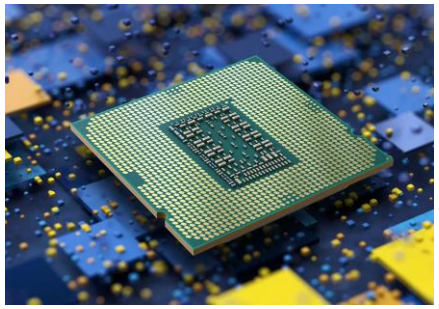
新版本做出了综合的性能优化,对SPECjbb2015提升显著。

https://developer.aliyun.com/article/787110

为何需要基准评测?

- **标准化评价 Standardized evaluation** of performance and energy consumption
- **竞品分析 Competitive analysis**
  - For manufacturers: product design optimization (performance, power, cost, …)
  - For users: product selection (performance, price, …)

CPU



intel AMD

AMPERE. Kunpeng

Server

DELL Technologies Lenovo

hp inspur

Cloud Instance

aws Microsoft Azure

Google Cloud Alibaba Cloud

[J. Guo. From SPEC Benchmarking to Online Performance Evaluation in Data Centers. LTB@ICPE 2022 Keynote.]

内容

- 基准评测程序
- 标准化基准评测套件
- 基准评测的策略
- 阿姆达尔定律
- 古斯塔夫森定律

基准评测程序 Benchmark Programs

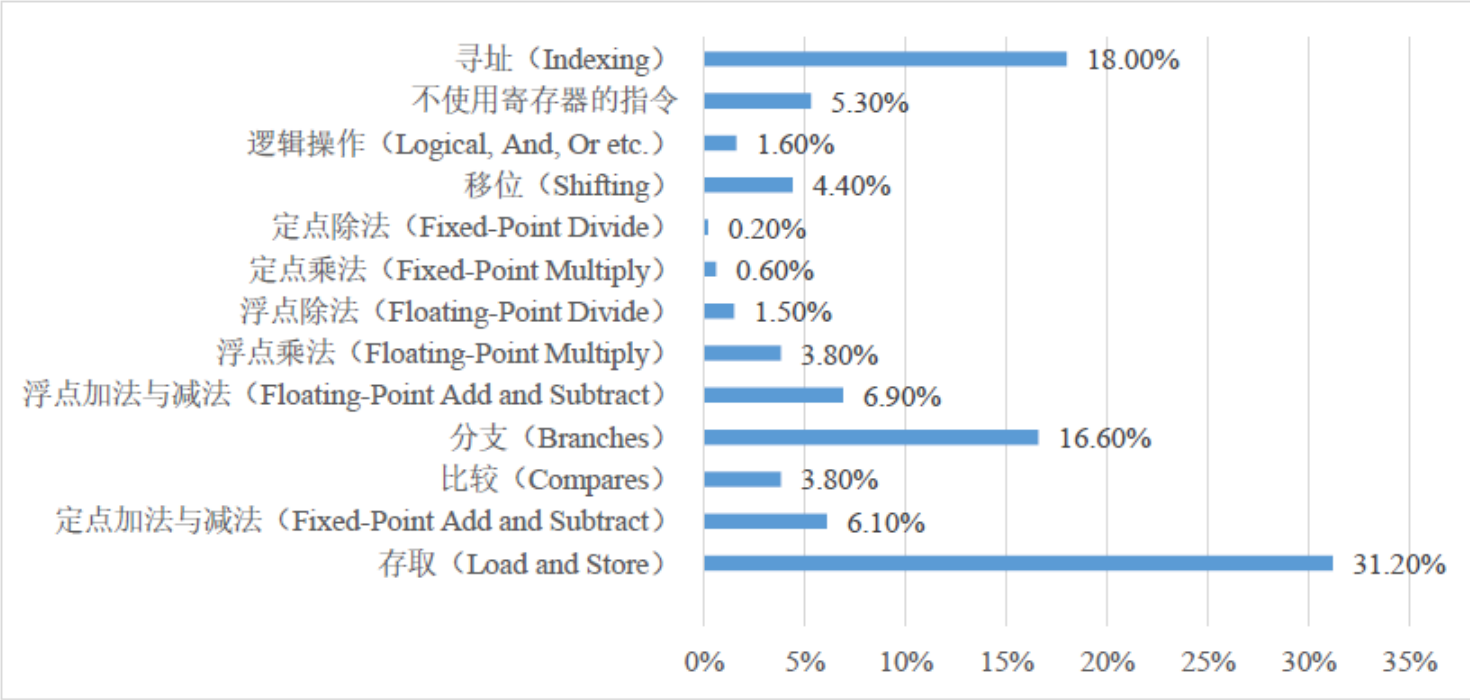
- 单一指令
- 指令组合（instruction mix）
- 合成程序（synthetic programs）
- 程序内核（program kernels）
- 微基准评测程序（microbenchmark programs）
- 应用基准评测程序/套件（application benchmark programs/suites）

单一指令

- 最早和最普遍接受的性能衡量标准之一是执行单个操作（例如，最常见的加法指令）所需的时间
- 由于早期指令集架构较为简单，执行指令消耗的时钟周期通常是相同的。

指令组合

- 一个指令组合是由多种类型指令以及它们各自使用频率定义的，其中不同指令的使用频率来源于真实系统
- Gibson指令组合来源于 IBM 704/650 系统



不足?

合成程序

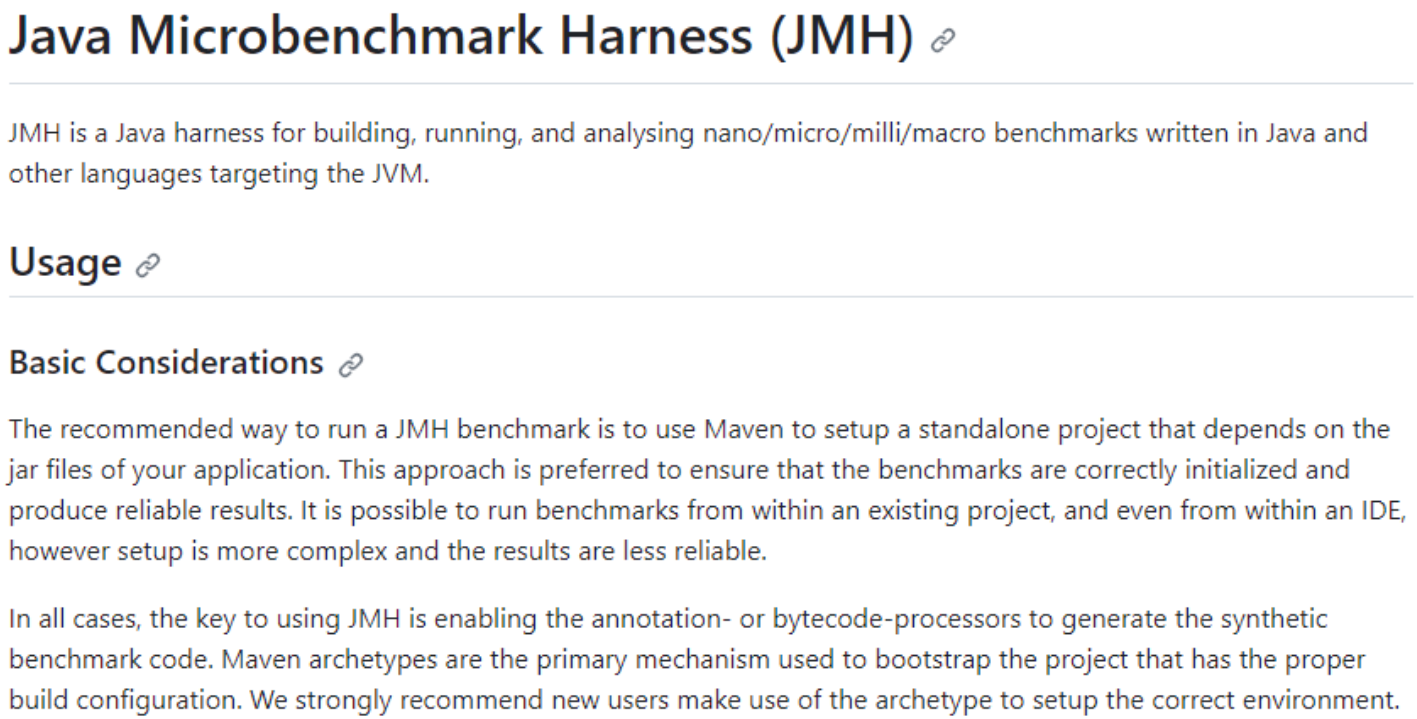
- 基本思想：编写一个合成程序，使其与预期的或所需的指令组合相匹配。
  - 指令组合思想的一种增强和补充。
- 合成程序是没有实际用途的人工程序，无法切实模拟真实应用的行为
  - 不能准确地模拟由于特定的指令排序而引起的指令之间的交互及其对性能的影响

程序内核

- 表示应用程序的核心或基本部分，本质上是一个指令组合的泛化形式，通常是一组频繁使用的操作。
- 例如，科学计算常用的矩阵乘法、矩阵求逆等，以及应用程序常用的排序函数、查找函数等。

微基准评测程序

- 专门设计的小程序，主要用于评测计算机系统的某个特定组件，例如，内存接口芯片、输入/输出子系统或浮点执行单元等



Java Microbenchmark Harness (JMH)

JMH is a Java harness for building, running, and analysing nano/micro/milli/macro benchmarks written in Java and other languages targeting the JVM.

Usage

Basic Considerations

The recommended way to run a JMH benchmark is to use Maven to setup a standalone project that depends on the jar files of your application. This approach is preferred to ensure that the benchmarks are correctly initialized and produce reliable results. It is possible to run benchmarks from within an existing project, and even from within an IDE, however setup is more complex and the results are less reliable.

In all cases, the key to using JMH is enabling the annotation- or bytecode-processors to generate the synthetic benchmark code. Maven archetypes are the primary mechanism used to bootstrap the project that has the proper build configuration. We strongly recommend new users make use of the archetype to setup the correct environment.

https://github.com/openjdk/jmh

应用基准评测程序/套件

- 应用基准评测程序：直接使用真实的应用作为基准评测程序。这些程序是完整的真实程序，能完成实际的操作并产生有意义的结果。
- 应用基准评测套件：单一应用通常代表性有限，为了提高基准评测的代表性，一个常见的做法是收集一组应用程序。
- 组成套件的应用程序可以是同一类的应用，比如都是科学计算应用，可以覆盖大部分科学计算的应用场景；也可以是不同类的应用，比如包含了编译器、编解码、科学计算、物理仿真等应用，可以覆盖某个计算机系统上大部分常用的应用场景。

标准化基准评测套件

- 为了进一步提高基准评测的代表性和公信力，一些标准化组织相继成立，他们开发和维护标准化基准评测套件（standardized benchmark suite），评审各个厂商产品的性能并公开发布标准化评测报告，逐渐形成行业标准。

LINPACK benchmarks

Original author(s)	Jack Dongarra, Jim Bunch, Cleve Moler, and Gilbert Stewart
Initial release	1979
Website	netlib.org/benchmark/hpl/

标准性能评估公司（The Standard Performance Evaluation Corporation, SPEC）

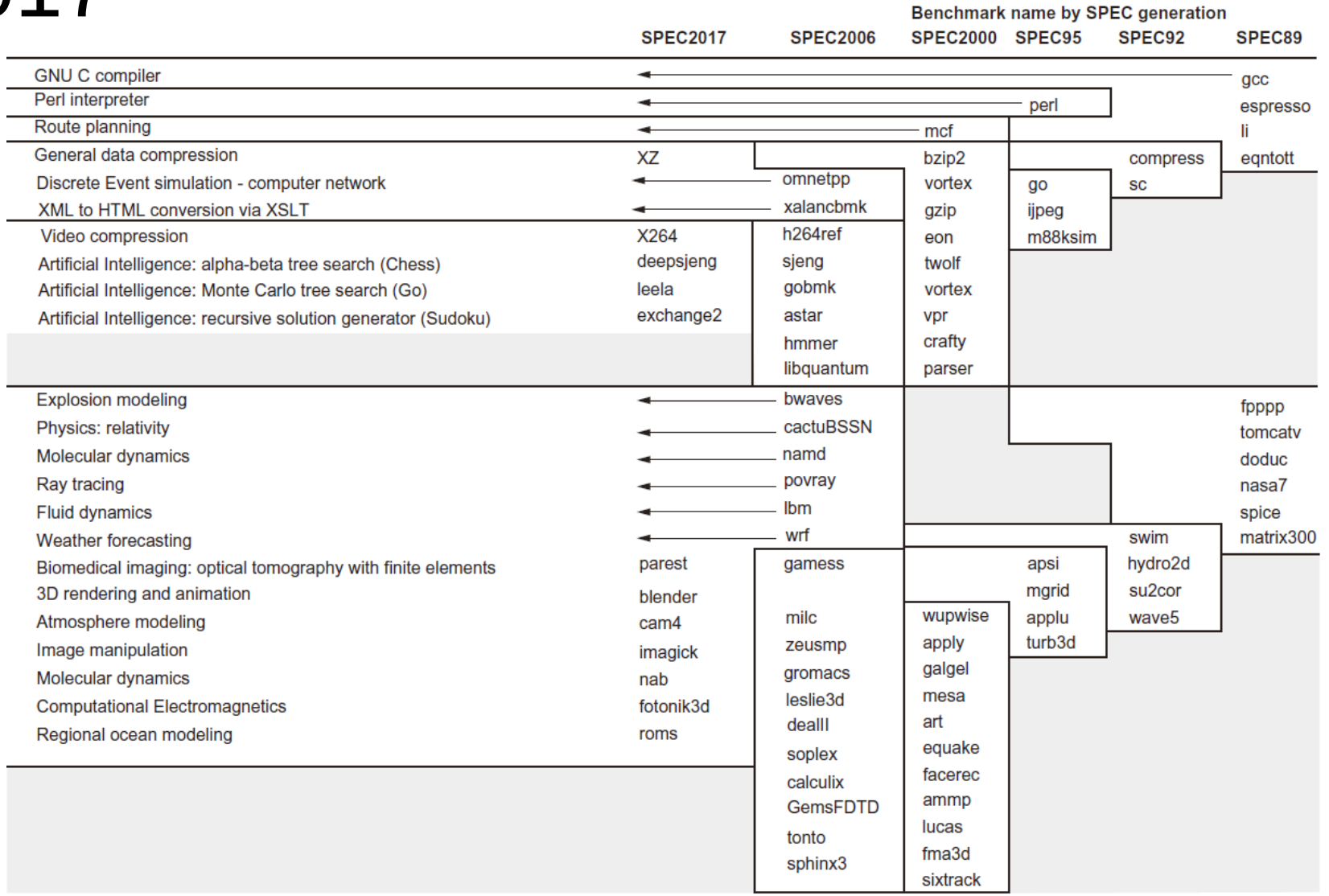
- SPEC 是一家非营利性组织，旨在为计算机系统的性能和能效等建立、维护和背书一套标准化基准评测。
- SPEC 开发了若干基准评测套件，同时，负责审核各个厂商的评测结果，并将结果公开发布在其网站上。

评测场景	套件名称	简介
云计算	SPEC Cloud IaaS 2018	运行典型的云服务工作负载并评估基础设施即服务的云平台性能
处理器	SPEC CPU 2017	运行计算密集型的整型和浮点型应用并评估处理器性能
高性能计算	SPEC OMP 2012	运行基于 OpenMP 的程序并评估并行处理应用的性能
Java 服务器	SPECjbb 2015	运行分布式 Java 应用并评估 Java 服务器的性能
服务器能效	SPECpower_ssj 2008	运行多种不同的工作负载并评估服务器的功耗和性能的关系
虚拟化平台	SPEC virt_sc 2013	运行服务器整合场景的应用并评估虚拟化平台的性能

www.spec.org

SPEC CPU 2017

- 最为广泛使用的用于评估 CPU 处理器性能的基准评测套件
- 1989 年被首次推出，之后经过多次更新



[John L. Hennessy, David A. Patterson: Computer Architecture - A Quantitative Approach, 6th Edition. Morgan Kaufmann 2017.]

SPEC CPU 2017

- 43个应用基准评测程序
- C/C++/Fortran
- 不同的应用领域

SPECrate 2017 Integer	SPECspeed 2017 Integer	编程语言[1]	千行代码数[2]	应用领域
500.perlbench_r	600.perlbench_s	C	362	Perl 解释器
502.gcc_r	602.gcc_s	C	1 304	GNU C 编译器
505.mcf_r	605.mcf_s	C	3	路径规划
520.omnetpp_r	620.omnetpp_s	C++	134	离散事件模拟 - 计算机网络
523.xalancbmk_r	623.xalancbmk_s	C++	520	通过 XSLT 进行 XML 到 HTML 的转换
525.x264_r	625.x264_s	C	96	视频压缩
531.deepsjeng_r	631.deepsjeng_s	C++	10	人工智能：alpha-beta 树搜索（国际象棋）
541.leela_r	641.leela_s	C++	21	人工智能：蒙特卡洛树搜索（围棋）
548.exchange2_r	648.exchange2_s	Fortran	1	人工智能：递归解生成器（数独）
557.xz_r	657.xz_s	C	33	通用数据压缩
SPECrate 2017 Floating Point	SPECspeed 2017 Floating Point	编程语言	千行代码数	应用领域
503.bwaves_r	603.bwaves_s	Fortran	1	爆炸建模
507.cactuBSSN_r	607.cactuBSSN_s	C++, C, Fortran	257	物理：相对论
508.namd_r	-	C++	8	分子动力学
510.parest_r	-	C++	427	生物医学成像：有限元光学断层扫描
511.povray_r	-	C++, C	170	光线追踪
519.lbm_r	619.lbm_s	C	1	流体动力学
521.wrf_r	621.wrf_s	Fortran, C	991	天气预报
526.blender_r	-	C++, C	1 577	3D 渲染与动画
527.cam4_r	627.cam4_s	Fortran, C	407	大气建模
-	628.pop2_s	Fortran, C	338	大范围海洋建模（气候级）
538.imagick_r	638.imagick_s	C	259	图像处理
544.nab_r	644.nab_s	C	24	分子动力学
549.fotonik3d_r	649.fotonik3d_s	Fortran	14	计算电磁学
554.roms_r	654.roms_s	Fortran	210	区域海洋建模

[1] 对于多编程语言基准测试，列表中的第一个编程语言决定了库和链接选项（详细信息见文档）。

[2] KLOC = 用于构建的源文件行数（包括注释/空白行）/ 1000。

SPEC CPU 2017

- Speed 方式：只执行程序的一个副本，但允许使用多线程加速程序运行（启用OpenMP）
- Rate 方式：并行地执行若干程序副本，每一个程序是单线程的（禁用OpenMP)
- 评价指标：基于时间 & 基于吞吐量

$$\mathrm{SPECspeed} = \frac{T_{\mathrm{ref}}}{T}$$

$$\mathrm{SPECrate} = N_{\mathrm{copy}} \times \frac{T_{\mathrm{ref}}}{T}$$

SPEC CPU 2017

- 评测基本步骤

1. 检查待测系统是否满足配置要求；
2. 使用安装包来安装 SPEC CPU 2017；
3. 选择使用哪一个子套件进行评测；
4. 生成一个配置文件，定义如何编译和运行基准评测程序；
5. 使用 runcpu 工具执行基准评测；
6. 如果需要发布结果，需要遵循预先规定的运行与报告的规则。

SPEC CPU 2017

- 如何生成一个已知的最佳配置（Best-Known Configuration, BKC）
- 竞品分析的关键

1. 根据待测的软硬件环境，选定一个初始配置，记为 config A，一般会从 SPEC 网站发布的公开报告里选择与待测环境类似的 BKC；
2. 测量 config A 的性能；
3. 为了获取更优性能，采用各种配置优化方法，生成新的配置 config A'；
4. 测量 config A' 的性能；
5. 如果 config A' 的性能优于 config A，那么用 config A' 替换 config A；
6. 评价当前 config A 的性能，如果仍然不满足需求，则回到第 3 步继续调优。

SPEC CPU 2017 的配置优化（下节课的主题）

SPEC CPU®2017 Integer Rate Result

Copyright 2017-2021 Standard Performance Evaluation Corporation

Huawei (Test Sponsor: Peng Cheng Laboratory)
Huawei TaiShan 200 Server (Model 2280)
(2.6 GHz,Huawei Kunpeng 920 7260)

SPECrate®2017_int_base = 389
SPECrate®2017_int_peak = Not Run

CPU2017 License:	5036	Test Date:	Sep-2021
Test Sponsor:	Peng Cheng Laboratory	Hardware Availability:	Sep-2019
Tested by:	Peng Cheng Laboratory	Software Availability:	Jul-2021

Benchmark result graphs are available in the PDF report.

Hardware

CPU Name:	Huawei Kunpeng 920 7260
Max MHz:	2600
Nominal:	2600
Enabled:	128 cores, 2 chips
Orderable:	1,2 chips
Cache L1:	64 KB I + 64 KB D on chip per core
L2:	512 KB I+D on chip per core
L3:	64 MB I+D on chip per chip
Other:	None
Memory:	512 GB (16 x 32 GB 2Rx8 PC4-2933P-R)
Storage:	15 TB LVM RAID-0 stripe on 4 x 3.638 TB SATA HDD, 7200 RPM
Other:	None

Software

OS:	openEuler release 20.03 (LTS-SP2) 4.19.90-2106.3.0.0095.oe1.aarch64
Compiler:	C/C++/Fortran: Version 1.3.3 of BiSheng
Parallel:	No
Firmware:	Huawei Corp. Version 1.80 released Sep-2021
File System:	ext4
System State:	Run level 3
Base Pointers:	64-bit
Peak Pointers:	Not Applicable
Other:	jemalloc memory allocator V5.2.1
Power Management:	BIOS and OS set to prefer performance at the cost of additional power usage

Results Table

Benchmark	Base							Peak						
	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio	Copies	Seconds	Ratio	Seconds	Ratio	Seconds	Ratio
500.perlbench_r	128	**562**	**362**	562	363	565	361							
502.gcc_r	128	724	250	726	250	**724**	**250**							
505.mcf_r	128	**597**	**347**	596	347	598	346							
520.omnetpp_r	128	**972**	**173**	971	173	973	173							
523.xalancbmk_r	128	**433**	**312**	433	312	433	312							
525.x264_r	128	**240**	**934**	240	934	241	932							
531.deepsjeng_r	128	305	481	**304**	**483**	303	484							
541.leela_r	128	**497**	**427**	498	426	497	427							
548.exchange2_r	128	305	1100	304	1100	**305**	**1100**							
557.xz_r	128	**620**	**223**	621	223	620	223							
SPECrate®2017_int_base			389											
SPECrate®2017_int_peak			Not Run											

Results appear in the order in which they were run. Bold underlined text indicates a median measurement.

Operating System Notes

Stack size set to unlimited using "ulimit -s unlimited"

Environment Variables Notes

Environment variables set by runcpu before the start of the run:
LD_LIBRARY_PATH =
"/root/install_dir/bisheng-compiler-1.3.3-aarch64-linux/lib:"
PATH =
"/root/install_dir/bisheng-compiler-1.3.3-aarch64-linux/bin:/home/spec2017/bin:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/root/bin"

Base Optimization Flags

C benchmarks:

-flto=full -Wl, --build-id -z muldefs -Wl,-mllvm, -licm-safe-hoist=true -enable-diamond-load-hoist=true -enable-loop-load-widen=true -disable-extra-gate-for-loop-heuristic=false -aarch64-optimize-vector-mul=true -enable-struct-padding=false -enable-struct-repacking=true -ljemalloc -O3 -mcpu=native -mllvm -enable-loopinterchange-boole=true -fno-strict-aliasing -lmathlib

C++ benchmarks:

-std=c++03 -flto=full -Wl, --build-id -Wl,-mllvm, -licm-safe-hoist=true -ljemalloc -O3 -mcpu=native -mllvm -enable-loopinterchange-boole=true -lmathlib

Fortran benchmarks:

-Mallocatable=03 -flto=full -Wl, --build-id -Wl,-mllvm, -enable-large-loop-bp-enhancement=true -ljemalloc -O3 -mcpu=native -Kieee -mllvm -enable-loopinterchange-boole=true -lmathlib

Base Other Flags

C benchmarks:

-v -fuse-ld=lld

C++ benchmarks:

-v -fuse-ld=lld

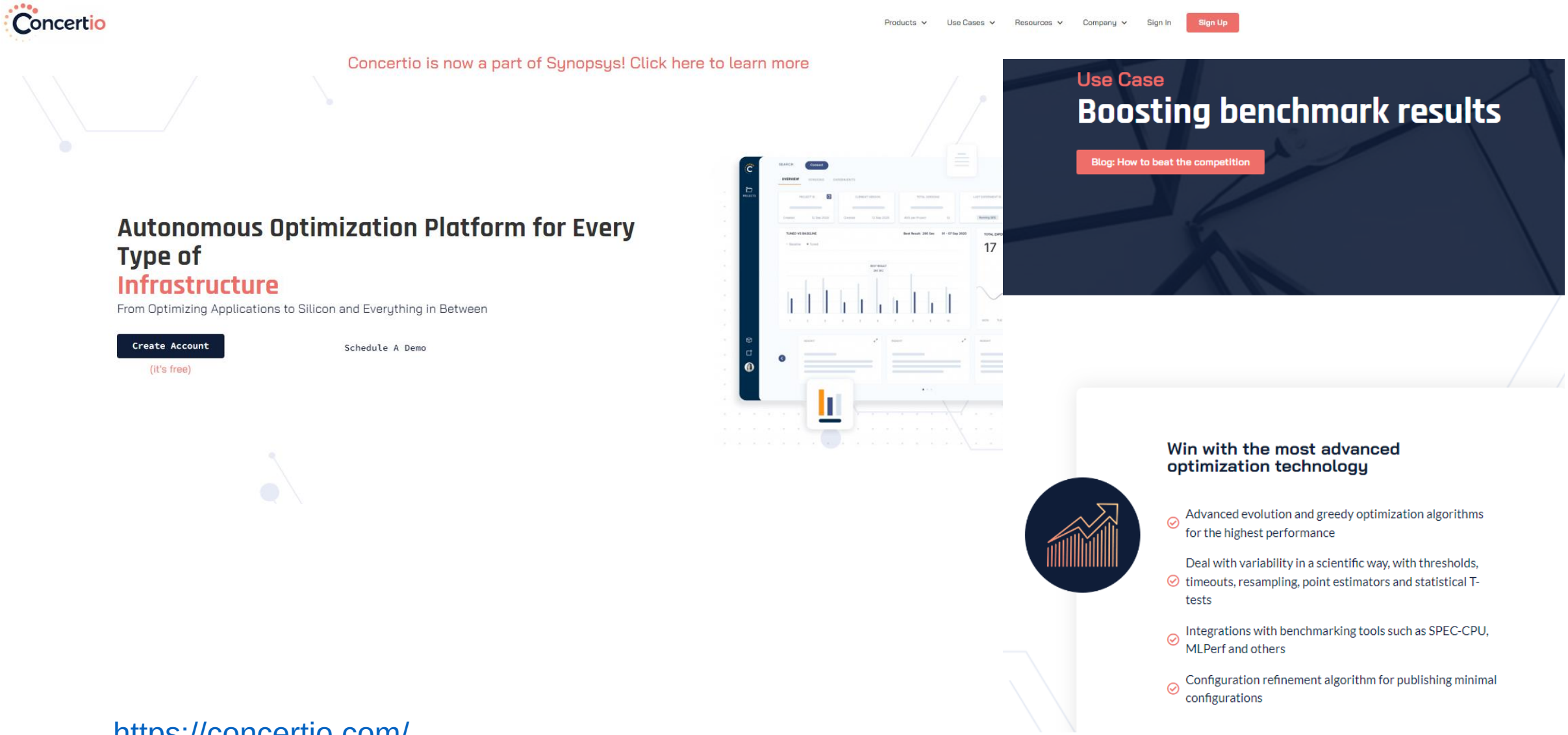
Fortran benchmarks:

-v -fuse-ld=lld

巨大的配置空间

http://spec.org/cpu2017/results/res2021q4/cpu2017-20211012-29727.html

配置优化和竞品分析



https://concertio.com/

基准评测套件的开发标准

- **“The best choice of benchmarks to measure performance is real applications.”** [J. L. Hennessy, D. A. Patterson: Computer Architecture - A Quantitative Approach, 6th Edition. Morgan Kaufmann 2017]
- 但是真实应用可能存在部署困难、输入规模多变和运行时间不可控等限制。
- 基准评测套件易于建立性能基线，也便于不同产品的性能比较。

基准评测套件的开发标准

- **“The best choice of benchmarks to measure performance is real applications.”** [J. L. Hennessy, D. A. Patterson: Computer Architecture - A Quantitative Approach, 6th Edition. Morgan Kaufmann 2017]
- 但是真实应用可能存在部署困难、输入规模多变和运行时间不可控等限制。
- 基准评测套件易于建立性能基线，也便于不同产品的性能比较。

1. 严格定义工作量，即保证每一次测试中待测机器需要完成的操作是一致的，可以通过校验基准评测程序的输出以检验是否完成指定的工作量。
2. 结果可复现，多次运行得到的结果应当不存在较大的性能波动。
3. 报告至少一个性能指标，该数值能够表示待测机器的性能，通常是响应时间或吞吐量。
4. 结果可比较，多个系统之间的性能指标应能够用于评估性能差异。
5. 定义运行与报告的规则，规定应当如何运行基准评测程序，以及测试结果如何汇总与报告。

基准评测的策略

- 固定计算的基准评测
- 固定时间的基准评测
- 可变计算和可变时间的基准评测

固定计算的基准评测

记待测机器运行基准评测程序完成的工作量为W，这里的工作量是一个抽象的概念，在不同的场景下会有不同的解释，比如，事务或指令等；执行时间为T。

那么可以定义待测机器的“执行速率”为 R = W/T。

$$R_1 = \frac{W_1}{T_1}, \quad R_2 = \frac{W_2}{T_2} \qquad \text{speedup} = \frac{R_1}{R_2} = \frac{W_1 \ / \ T_1}{W_2 \ / \ T_2}$$

固定计算的基准评测

记待测机器运行基准评测程序完成的工作量为W，这里的工作量是一个抽象的概念，在不同的场景下会有不同的解释，比如，事务或指令等；执行时间为T。

那么可以定义待测机器的“执行速率”为 R = W/T。

$$R_1 = \frac{W_1}{T_1}, \quad R_2 = \frac{W_2}{T_2} \qquad \text{speedup} = \frac{R_1}{R_2} = \frac{W_1 \,/\, T_1}{W_2 \,/\, T_2}$$

设定要对比的两个系统需要完成的工作量是一致的，即有 $W_1 = W_2$

$$\text{speedup} = \frac{T_2}{T_1}$$

例子：SPECspeed

$$\text{SPECspeed} = \frac{T_{\text{ref}}}{T}$$

Calculating SPECspeed® Metrics
1 copy of each benchmark in a suite is run.
The tester may choose how many OpenMP threads to use.
For each benchmark, a performance ratio is calculated as: `time on a reference machine / time on the SUT`
Higher scores mean that less time is needed.
Example: • The reference machine ran 600.perlbench_s in 1775 seconds. • A particular SUT took about 1/5 the time, scoring about 5. • More precisely: `1775/354.329738 = 5.009458`

https://spec.org/cpu2017/Docs/overview.html

固定时间的基准评测

- 在某些应用场景下，程序需要完成的工作量可能很大，这时固定计算的策略可能就不适用于基准评测了。
- 例如，评测一个天气预报系统，如果每次预报第二天天气的任务都需要24 个小时以上的运行时间，这个评测本身是没有用处的。

$$R_1 = \frac{W_1}{T_1}, \quad R_2 = \frac{W_2}{T_2} \qquad \text{speedup} = \frac{R_1}{R_2} = \frac{W_1 \ / \ T_1}{W_2 \ / \ T_2}$$

根据固定时间的策略，即有 $T_1 = T_2$

$$\text{speedup} = \frac{W_1}{W_2}$$

例子：SLALOM

- 该评测衡量性能的指标是，在一分钟内待测机器对一个问题计算答案的准确程度。
- 该程序会持续进行迭代，在一分钟的时间限制内，完成的迭代次数越多，得到的答案越准确。
- 该基准评测没有限定求解问题的算法，以答案的准确程度作为衡量待测系统性能的依据，目的是为了体现系统解决实际问题的能力。

http://www.johngustafson.net/benchmarks/slalom/slalom.htm

可变计算和可变时间基准评测

- 以兼顾工作量和执行时间的一个函数 $f(W,T)$ 作为性能评价的指标

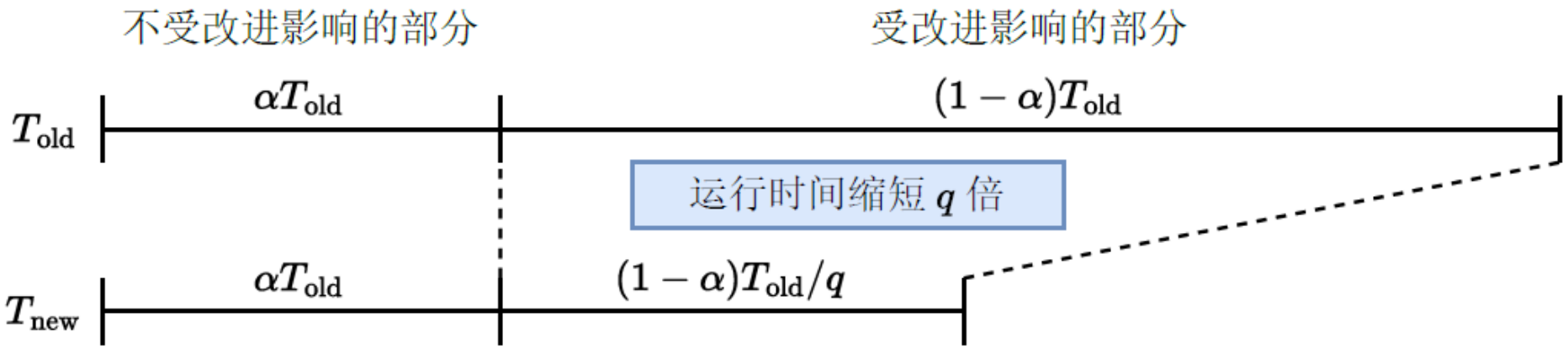
- 例子：HINT Benchmarks
  - 它严格定义了一个数学问题，同时定义了该问题解决方案的质量。
  - 解决方案的执行需要一定的时间，同时解决方案的质量可以通过额外的计算不断提高，因而，这个质量指标与达到该质量水平所需时间的比值就是性能评测指标，定义为 QUIPS (QUality Improvements Per Second)，即每秒获得的质量提升。

http://www.johngustafson.net/pubs/pub47/Hint.htm

阿姆达尔定律 Amdahl's Law

- 根据固定计算的基准评测策略，由于固定了待测系统需要完成的工作量，一旦改进系统并采用了更快的执行方式，则系统完成工作量的时间会缩短，进而相比改进前的加速比会提高。
- 那么，一个有意思的问题是，改进待测系统所能带来的最大性能提升是多少?

阿姆达尔定律 Amdahl's Law



$$T_{\text{new}} = \alpha T_{\text{old}} + \frac{(1-\alpha)T_{\text{old}}}{q}$$

$$\text{speedup} = \frac{T_{\text{old}}}{T_{\text{new}}} = \frac{T_{\text{old}}}{\alpha T_{\text{old}} + \frac{(1-\alpha)T_{\text{old}}}{q}} = \frac{1}{\frac{1}{q} + \alpha(1-\frac{1}{q})}$$

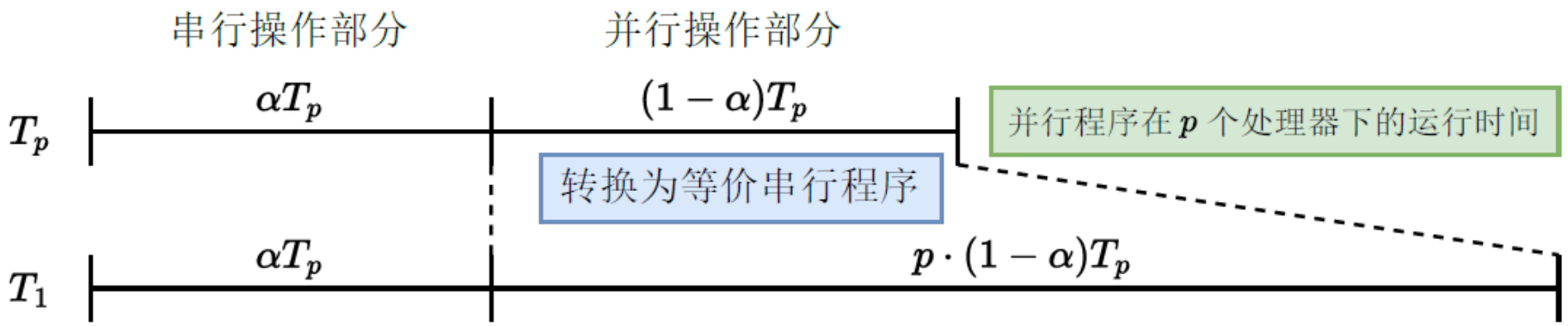
$$\lim_{q \to +\infty} \frac{1}{\frac{1}{q} + \alpha(1-\frac{1}{q})} = \frac{1}{\alpha}$$

程序性能提升的上限取决于程序不受改进影响部分的比例

古斯塔夫森定律 Gustafson's Law

- 根据阿姆达尔定律，如果一个程序可并行部分的比例不大，无论增加多少个处理器，实际上能够获得的加速比也是十分有限的。这样的结论是悲观的，这意味着即便是使用超级计算机，也没法得到更大的性能提升。
- 然而，阿姆达尔定律实际上是一个固定计算的模型，工作量的大小是固定的，进而可并行化的规模也是固定的。
- 实际操作中，我们不会用一台超级计算机来处理一个小问题。当待测机器的运算能力增加之后，可以也应该去解决规模更大的问题。
- 古斯塔夫森认为，工程师应当调整工作量以充分利用待测机器的运算能力，如果有更快的设备，就可以在相同时间内解决更大规模的问题。

古斯塔夫森定律 Gustafson's Law



$$T_1 = \alpha T_p + p \cdot (1-\alpha)T_p$$

$$\text{speedup} = \frac{T_1}{T_p} = \frac{\alpha T_p + p \cdot (1-\alpha)T_p}{T_p} = \alpha + (1-\alpha)p = p + \alpha(1-p)$$

$$W_p = \alpha W + p(1-\alpha)W$$

当问题的规模足够大时，α接近于0，即有 $speedup \approx p$

局限?

小结

- 基准评测为标准化性能评价和竞品分析提供了有效手段和数据基础
- 基准评测的一个重要原则是尽可能代表真实应用，同时结果可复现
- 基准评测的策略
  - 固定计算
  - 固定时间
  - 可变计算和可变时间
- 两条定律
  - 阿姆达尔定律
  - 古斯塔夫森定律