基准评测

郭健美

2023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?

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?

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

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

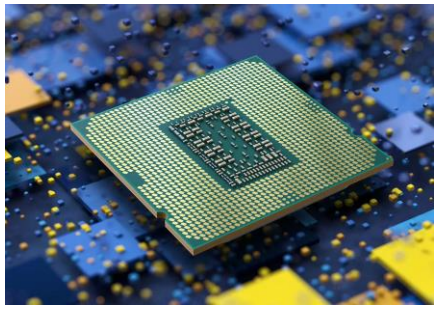
新版本做出了综合的性能优化,对SPECjbb2015提升显著。

https://new.qq.com/omn/20210602/20210602A08Y1O00.html

为何需要基准评测?

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

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.]

Benchmark Programs 基准评测程序

- 单一指令
- 指令组合（instruction mixes）
- 合成程序（synthetic programs）
- 程序内核（program kernels）
- 微基准评测程序（microbenchmarks）
- 应用基准评测程序（application benchmarks）

Addition Instruction

- Processors were the most expensive and most used components of the system
- Addition was the most frequent instruction

Instruction Mixes

- Instruction mix = instructions + usage frequency
- Gibson mix: Developed by Jack C. Gibson in 1959 for IBM 704 systems.

1.	Load and Store	31.2
2.	Fixed-Point Add and Subtract	6.1
3.	Compares	3.8
4.	Branches	16.6
5.	Floating Add and Subtract	6.9
6.	Floating Multiply	3.8
7.	Floating Divide	1.5
8.	Fixed-point Multiply	0.6
9.	Fixed-point Divide	0.2
10.	Shifting	4.4
11.	Logical, And, Or, etc.	1.6
12.	Instructions Not Using Registers	5.3
13.	Indexing	18.0
	Total	100.0

Instruction Mixes (Cont)

- Disadvantages:
  - Complex classes of instructions not reflected in the mixes.
  - Instruction time varies with:
    - Addressing modes
    - Cache hit rates
    - Pipeline efficiency
    - Interference from other devices during processor-memory access cycles
    - Parameter values
    - Frequency of zeros as a parameter
    - The distribution of zero digits in a multiplier
    - The average number of positions of preshift in floating-point add
    - Number of times a conditional branch is taken

Instruction Mixes (Cont)

- Performance Metrics:
  - MIPS = Millions of Instructions Per Second
  - MFLOPS = Millions of Floating Point Operations Per Second

Synthetic Programs

- To measure I/O performance lead analysts
  ⇒ Exerciser loops
- The first exerciser loop was by Buchholz (1969) who called it a synthetic program.
- Advantage:
  - Quickly developed and given to different vendors.
  - No real data files
  - Easily modified and ported to different systems.
  - Have built-in measurement capabilities
  - Measurement process is automated
  - Repeated easily on successive versions of the operating systems
- Disadvantages:
  - Too small
  - Do not make representative memory or disk references
  - Mechanisms for page faults and disk cache may not be adequately exercised.
  - CPU-I/O overlap may not be representative.
  - Loops may create synchronizations ⇒ better or worse performance.

microbenchmarks

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

Kernels

- Kernel = nucleus
- Kernel= the most frequent function
- Commonly used kernels: Sieve, Puzzle, Tree Searching, Ackerman's Function, Matrix Inversion, and Sorting.
- Disadvantages: Do not make use of I/O devices

Application Benchmarks

- For a particular industry: Debit-Credit for Banks
- Benchmark = workload (Except instruction mixes)
- Some Authors: Benchmark = set of programs taken from real workloads
- Popular Benchmarks

Benchmarking

- “The best choice of benchmarks to measure performance is real applications.”

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

What is a (good) benchmark?

- A surveyor‘s *bench mark* defines a known reference point by which other locations may be measured.
- A computer *benchmark* performs a known set of operations by which computer performance can be measured.

Table 1: Characteristics of useful performance benchmarks

Specifies a *workload*	A strictly-defined set of operations to be performed.
Produces at least one *metric*	A numeric representation of performance. Common metrics include: • Time - For example, seconds to complete the workload. • Throughput - Work completed per unit of time, for example, jobs per hour.
Is *reproducible*	If repeated, will report similar (*) metrics.
Is *portable*	Can be run on a variety of interesting systems.
Is *comparable*	If the metric is reported for multiple systems, the values are meaningful and useful.
Checks for correct operation	Verify that meaningful output is generated and that the work is actually done. *"I can make it run as fast as you like if you remove the constraint of getting correct answers." (**)*
Has *run rules*	A clear definition of required and forbidden hardware, software, optimization, tuning, and procedures.

(*) "Similar" performance will depend on context. The benchmark should include guidelines as to what variation one should expect if the benchmark is run multiple times.

(**) Richard Hart, Digital Equipment Corporation, approximately 1982.

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

The Standard Performance Evaluation Corporation (SPEC)

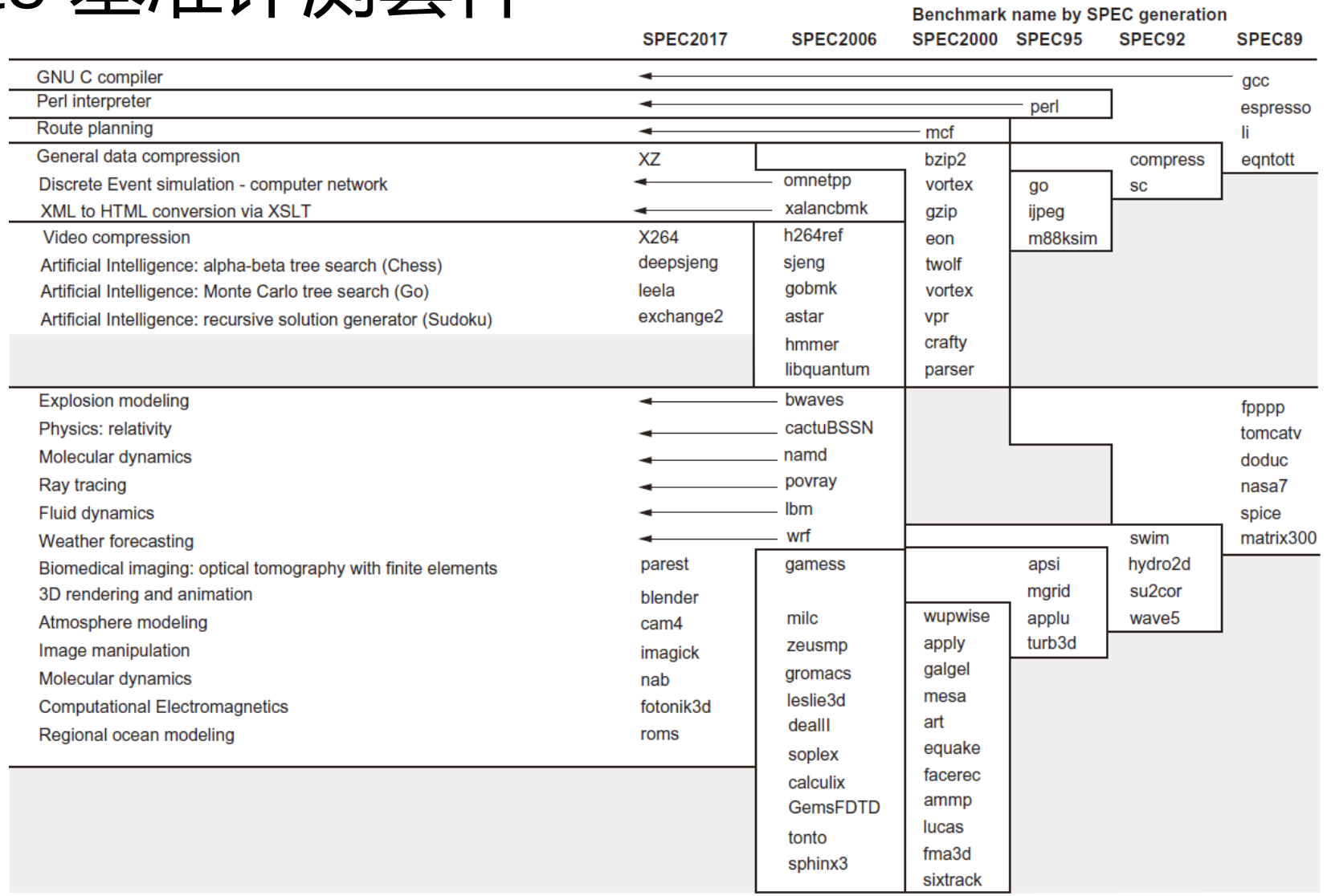
- www.spec.org
- SPEC is a non-profit corporation formed to **establish, maintain and endorse** standardized benchmarks and tools to **evaluate** performance and energy efficiency for the newest generation of computing systems.
- SPEC **develops** benchmark suites and also **reviews** and **publishes** submitted results from our member organizations and other benchmark licensees.

Category	Name	Measures performance of
Cloud	Cloud_IaaS 2016	Cloud using NoSQL database transaction and K-Means clustering using map/reduce
CPU	CPU2017	Compute-intensive integer and floating-point workloads
Graphics and workstation performance	SPECviewperf® 12	3D graphics in systems running OpenGL and Direct X
	SPECwpc V2.0	Workstations running professional apps under the Windows OS
	SPECapcSM for 3ds Max 2015™	3D graphics running the proprietary Autodesk 3ds Max 2015 app
	SPECapcSM for Maya® 2012	3D graphics running the proprietary Autodesk 3ds Max 2012 app
	SPECapcSM for PTC Creo 3.0	3D graphics running the proprietary PTC Creo 3.0 app
	SPECapcSM for Siemens NX 9.0 and 10.0	3D graphics running the proprietary Siemens NX 9.0 or 10.0 app
	SPECapcSM for SolidWorks 2015	3D graphics of systems running the proprietary SolidWorks 2015 CAD/CAM app
High performance computing	ACCEL	Accelerator and host CPU running parallel applications using OpenCL and OpenACC
	MPI2007	MPI-parallel, floating-point, compute-intensive programs running on clusters and SMPs
	OMP2012	Parallel apps running OpenMP
Java client/server	SPECjbb2015	Java servers
Power	SPECpower_ssj2008	Power of volume server class computers running SPECjbb2015
Solution File Server (SFS)	SFS2014	File server throughput and response time
	SPECsfs2008	File servers utilizing the NFSv3 and CIFS protocols
Virtualization	SPECvirt_sc2013	Datacenter servers used in virtualized server consolidation

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

Benchmark Suite 基准评测套件

- Collections of benchmark applications, called benchmark suites, are a popular measure of performance of processors with a variety of applications
- Of the 10 SPEC2017 integer programs, 5 are written in C, 4 in C++., and 1 in Fortran. For the floating-point programs, the split is 3 in Fortran, 2 in C++, 2 in C, and 6 in mixed C, C++, and Fortran.
- The version of the program changes and either the input or the size of the benchmark is often expanded to increase its running time and to avoid perturbation in measurement or domination of the execution time by some factor other than CPU time.



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

Common Benchmarking Mistakes

Critical thinking!

If the benchmark description says:	There may be potential difficulties:	Solutions
1. It runs Loop 1 billion times.	Compiler X runs 1 billion times faster than Compiler Y, because compilers are allowed to skip work that has no effect on program outputs ("dead code elimination").	Benchmarks should print something.
2. Answers are printed, but not checked, because Minor Floating Point Differences are expected.	• What if Minor Floating Point Difference sends it down Utterly Different Program Path? • If the program hits an error condition, it might "finish" twice as fast because half the work was never attempted.	Answers should be validated, within some sort of tolerance.
3. The benchmark is already compiled. Just download and run.	You may want to compare new hardware, new operating systems, new compilers.	Source code benchmarks allow a broader range of systems to be tested.
4. The benchmark is portable. Just use compiler X and operating system Y.	You may want to compare other compilers and other operating systems.	Test across multiple compilers and OS versions prior to release.
5. The benchmark measures X.	Has this been checked? If not, measurements may be dominated by benchmark setup time, rather than the intended operations.	Analyze profile data prior to release, verify what it measures.
6. The benchmark is a slightly modified version of Well Known Benchmark.	• Is there an exact writeup of the modifications? • Did the modifications break comparability?	Someone should check. Create a process to do so.
7. The benchmark does not have a Run Rules document, because it is obvious how to run it correctly.	Although "obvious" now, questions may come up. A change that seems innocent to one person may surprise another.	Explicit rules improve the likelihood that results can be meaningfully compared.
8. The benchmark is a collection of low-level operations representing X.	How do you know that it is representative?	Prefer benchmarks that are derived from real applications.

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

What does SPEC provide?

- SPEC CPU 2017 is distributed as an ISO image that contains:
  - Source code for the benchmarks
  - Data sets
  - A tool set for compiling, running, validating and reporting on the benchmarks
  - Pre-compiled tools for a variety of operating systems
  - Source code for the SPEC CPU 2017 tools, for systems not covered by the pre-compiled tools
  - Documentation
  - Run and reporting rules

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

How to Conduct a Competitive Benchmarking?

Basic Steps to run SPEC CPU 2017

1. Meet the System Requirements
2. Install the benchmark
3. Determine a suite
4. **Find** a *config* file and **customize** it
5. Use the `runcpu` command
6. For publishing results, adhere to Run Rules

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

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

How to Conduct a Competitive Benchmarking?

Basic Steps to run SPEC CPU 2017

1. Meet the System Requirements
2. Install the benchmark
3. Determine a suite
4. **Find** a *config* file and **customize** it
5. Use the `runcpu` command
6. For publishing results, adhere to Run Rules

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

Basic Steps to acquire a **Best-Known Config (BKC)**

1. Start from a config A (e.g., similar to competitors)
2. Measure the performance of A
3. **Change** A and produce a hopefully faster A'
4. **Measure** the performance of A'
5. If A' beats A, set A = A'
6. If A is still not competitive enough, go to Step 3

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

Exploration of Configuration Space

SPEC CPU®2017 Integer Rate Result

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

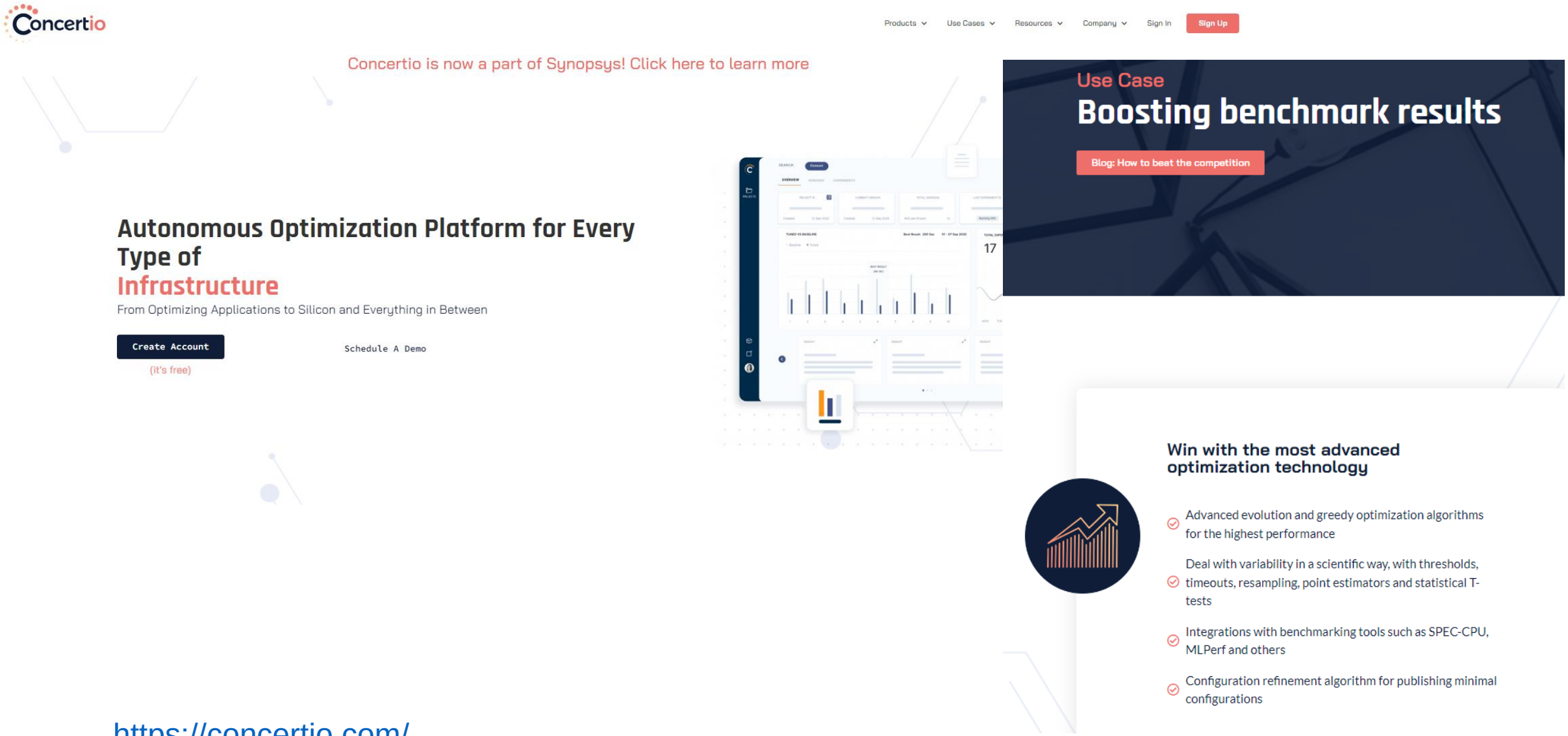
Fortran benchmarks:

-v -fuse-ld=lld

Huge Configuration Space!

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

A startup on Benchmarking Optimization



https://concertio.com/

基准评测策略

- Fixed-computation benchmarking
  - Amdahl’s Law
- Fixed-time benchmarking
  - Gustafson’s Law
- Variable-computation and variable-time benchmarking

固定计算基准评测

What we would like to do is define a computer system's *speed* or *execution rate*, denoted R_1, to be $R_1 = W_1/T_1$, where T_1 is the time required to execute the computation W_1.

we define the amount of computation performed by a specific program to be constant regardless of how many instructions are actually required to execute the program on either system. Thus, we simply define the amount of computation completed by the system when executing the benchmark program to be W so that $W_1 \equiv W_2 \equiv W$. Then the speedup of machine 1 relative to machine 2 is

$$S = \frac{R_1}{R_2} = \frac{W/T_1}{W/T_2} = \frac{T_2}{T_1}.$$

SPEC CPU 2017 Metrics

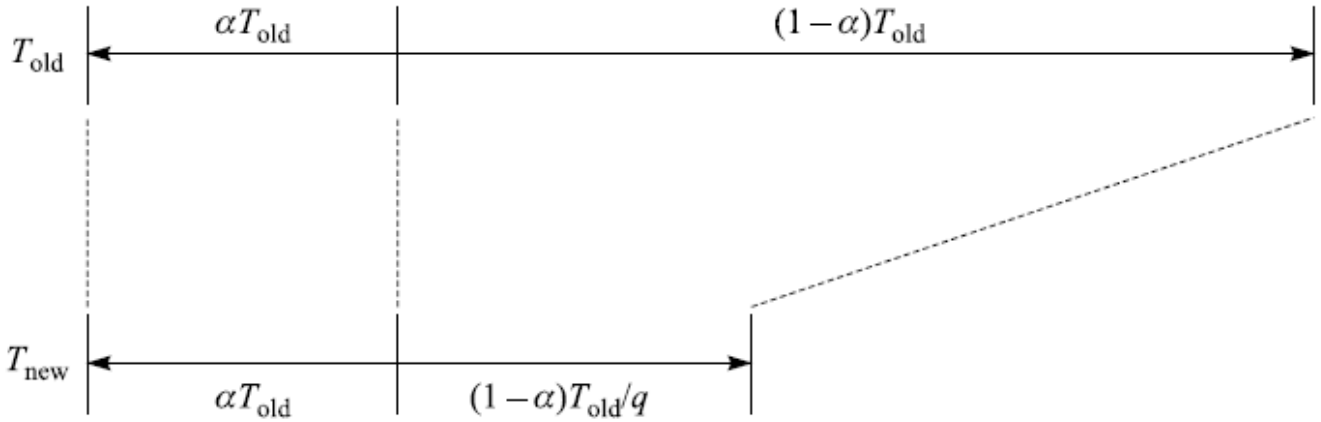
- There are many ways to measure computer performance. Among the most common are:
  - Time - For example, seconds to complete a workload.
  - Throughput - Work completed per unit of time, for example, jobs per hour.
- SPECspeed is a time-based metric; SPECrate is a throughput metric.

Calculating SPECspeed® Metrics	Calculating SPECrate® Metrics
1 copy of each benchmark in a suite is run.	The tester chooses how many concurrent copies to run
The tester may choose how many OpenMP threads to use.	OpenMP is disabled.
For each benchmark, a performance ratio is calculated as: `time on a reference machine / time on the SUT`	For each benchmark, a performance ratio is calculated as: `number of copies * (time on a reference machine / time on the SUT)`
Higher scores mean that less time is needed.	Higher scores mean that more work is done per unit of time.
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`	Example: • The reference machine ran 1 copy of 500.perlbench_r in 1592 seconds. • A particular SUT ran 8 copies in about 1/3 the time, scoring about 24. • More precisely: `8*(1592/541.52471) = 23.518776`
For both SPECspeed and SPECrate, in order to provide some assurance that results are repeatable, the entire process is repeated. The tester may choose: a. To run the suite of benchmarks three times, in which case the tools select the medians. b. Or to run twice, in which case the tools select the lower ratios (i.e. slower).	
For both SPECspeed and SPECrate, the selected ratios are averaged using the Geometric Mean, which is reported as the overall metric.	

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

Amdahl's Law

The concept of fixing the amount of computation to be executed in a program leads to an upper bound on how much the overall performance of any computer system can be improved due to changes in only a single component of the system.



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

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

固定时间基准评测

An alternative view argues that users with large problems to solve are willing to wait a fixed amount of time to obtain a solution. The allowable time may be determined simply by the users' patience, or by some external factor. For instance, a system that is used to compute a weather forecast is useless if it takes more than 24 h to produce the next day's forecast.

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

Gustafson's Law （Scaling Amdahl's Law）

Gustafson estimated the speedup S gained by using N processors (instead of just one) for a task with a serial fraction s (which does not benefit from parallelism) as follows:[2]

$$S = N + (1 - N)s$$

Using different variables, Gustafson's law can be formulated the following way:[*citation needed*]

$$S_{\text{latency}}(s) = 1 - p + sp,$$

where

- S_{latency} is the theoretical speedup in latency of the execution of the whole task;
- s is the speedup in latency of the execution of the part of the task that benefits from the improvement of the resources of the system;
- p is the percentage of the execution workload of the whole task concerning the part that benefits from the improvement of the resources of the system *before the improvement*.

Gustafson's law addresses the shortcomings of Amdahl's law, which is based on the assumption of a fixed problem size, that is of an execution workload that does not change with respect to the improvement of the resources. Gustafson's law instead proposes that programmers tend to set the size of problems to fully exploit the computing power that becomes available as the resources improve. Therefore, if faster equipment is available, larger problems can be solved within the same time.

https://en.wikipedia.org/wiki/Gustafson%27s_law

可变计算和可变时间基准评测

- The measured performance is a function of the computation & the execution time.
- HINT Benchmarks

The HINT benchmark is a good example of this variable-computation, variable-time benchmark-program strategy. It rigorously defines the 'quality' of a solution for a given mathematical problem that is to be solved by the benchmark program. The solution's quality, based on the problem's definition, can be continually improved by doing additional computation. The ratio of this quality metric to the time required to achieve that level of quality is then used as the measure of performance. The HINT benchmark expresses this final performance metric as QUIPS, or *quality improvements per second*.

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