How Google collected these benchmark results

Results in the blog post were collected on general-purpose VMs providing 32 vCPUs and 128 GB RAM. For the Google Cloud Tau VMs, results are based on pre-release VMs. All other results are based on production VMs offered by two other leading cloud vendors - Amazon Web Services and Microsoft Azure. Only the VMs that showed the best price-performance from each cloud vendor are shown in the charts.

The VMs were provisioned with PerfKitBenchmarker (PKB). To get started with PKB, we recommend starting with this walkthrough. To run the benchmarks on VMs offered by different cloud providers with PKB, you only need to vary 3 flags including cloud, zone, and machine type. We showed results for the following machines:

Machine	Cloud (--cloud=)	Zone (--zone=)	Machine (--machine_type=)
t2d-standard-32[1]	GCP	us-central1-f	t2d-standard-32
m6g.8xlarge	AWS	us-east-1a	m6g.8xlarge
Standard_D32s_v4	Azure	westus2	Standard_D32s_v4
Standard_D32as_v4	Azure	westus2	Standard_D32as_v4

Note that we tested on a wider set of machine types and in a wider set of regions and zones.

For collecting CoreMark and estimated SPECrate®2017_int_base results with GCC, we used a single PKB command line that also coordinated the installation of software, ran the benchmark, and parsed the data. For collecting SPECrate®2017_int_base compiled with ICC or AOCC, we used PKB to provision the VMs and then followed vendor instructions for running the SPEC binary. Reproducing these results requires reaching out to Intel or AMD.

To reproduce the estimated SPECrate®2017_int_base results using PKB, you first need to purchase SPEC CPU® 2017 and download the cpu2017-1.1.8.iso. Then you can run PKB and point it to this ISO file. The following command line will provision a GCP t2d-standard-32 machine in

[1] Repro instructions for the t2d-standard-32 are shown as they are expected to be once in GCP production environments. These commands will not be possible until the machines are available publicly.

us-central1-f, upload the ISO to the VM, install all necessary software, build the binary, run the benchmark, and parse the results:

```
COMMON_GCC_FLAGS="-Ofast -funroll-loops -flto -ljemalloc -z muldefs"

./pkb.py --cloud=GCP \
         --zone=us-central1-f \
         --machine_type=t2d-standard-32 \
         --benchmarks=speccpu2017 \
         --data_search_paths=<directory-containing-iso> \
         --os_type=ubuntu2004 \
         --enable_transparent_hugepages=True \
         --build_fortran=True \
         --force_build_gcc_from_source=True \
         --runspec_build_tool_version=11.1.0 \
         --runspec_estimate_spec=True \
         --runspec_iterations=1 \
         --runspec_keep_partial_results=True \
         --spec17_subset=intrate \
         --spec_runmode=base \
         --spec17_gcc_flags="-march=znver3  $COMMON_GCC_FLAGS" \
         --runspec_config=pkb-gcc-linux-x86.cfg
```

The above command is tuned for AMD Milan machines.

For AMD Rome machines, the last two arguments should be:

```
         --spec17_gcc_flags="-march=znver2  $COMMON_GCC_FLAGS" \
         --runspec_config=pkb-gcc-linux-x86.cfg
```

For Intel Cascade Lake machines, the last two arguments should be:

```
         --spec17_gcc_flags="-march=cascadelake $COMMON_GCC_FLAGS" \
         --runspec_config=pkb-gcc-linux-x86.cfg
```

For ARM machines, the last two arguments should be:

```
         --spec17_gcc_flags="-march=armv8.2-a  $COMMON_GCC_FLAGS" \
         --runspec_config=pkb-gcc-linux-aarch64.cfg
```

Note that we also tested with GCC using -O3, but we saw better performance with -Ofast on all machines tested. An interesting note is that while we saw a 56% estimated SPECrate®2017_int_base performance uplift on the t2d-standard-32 over the m6g.8xlarge when we used AMD's optimizing compiler, which could take advantage of the AMD architecture, we also saw a 25% performance uplift on the t2d-standard-32 over the m6g.8xlarge when using GCC 11.1 with the above flags for both machines.

To reproduce the estimated SPECrate®2017_int_base results using PKB but running the AMD or Intel binaries, use PKB to provision the VM and then follow the binary instructions to reproduce the results. The following command line will provision a GCP t2d-standard-32 machine in us-central1-f:

```
./pkb.py --cloud=GCP \
         --zone=us-central1-f \
         --machine_type=t2d-standard-32 \
         --benchmarks=cluster_boot \
         --os_type=ubuntu2004 \
         --enable_transparent_hugepages=True
```

Note that you may need a larger boot disk size to run the benchmark than is set by default for the cluster_boot benchmark. To increase the size of the boot disk, use an additional flag or change the YAML configuration to specify a "boot_disk_size" of 500 GB. On GCP, you may specify this with the flag:

```
--config_override=cluster_boot.vm_groups.default.vm_spec.GCP.boot_disk_size=500
```

To reproduce the CoreMark results using PKB, you can run a single command line. The following command line will provision a GCP t2d-standard-32 machine in us-central1-f and produce CoreMark results:

```
./pkb.py --cloud=GCP \
         --zone=us-central1-f \
         --machine_type=t2d-standard-32 \
         --benchmarks=coremark \
         --os_type=ubuntu1804 \
         --coremark_parallelism_method=SOCKET \
         --run_stage_iterations=5
```

All performance results presented in this blog were created with the above methodology and are in alignment with internal testing of GCP. Results may vary due to changes to the underlying configuration, updates to PKB, and other conditions such as the placement of the VM and its resources, optimizations and other changes made by the cloud service providers, accessed cloud regions, co-tenants, and the types of other workloads exercised at the same time on the system.

We used list prices offered by respective cloud providers to create the price-performance graph. List price from the following regions were used:

- AWS: US East
- Azure: East US
- GCP: us-central1