Geekbench 4
CPU Workloads

PRIMATE LABS

Introduction

This document outlines the workloads included in the Geekbench 4 CPU Benchmark suite.

CPU Benchmark scores are used to evaluate and optimize CPU and memory performance using workloads that include artificial intelligence, data compression, image processing, and physics simulation. Performance on these workloads is important for a wide variety of applications including web browsers, image editors, and developer tools.

Platform Support

Platform	Minimum Version	Comment
Android	Android 5.0 “Lollipop”	
iOS	iOS 9.0	
Linux	Ubuntu 14.04 LTS	CentOS 7.5 also supported
macOS	macOS 10.11	
Windows	Windows 7	

Compilers

Geekbench 4.0 is built using the following compilers:

Platform	Compiler	Comment
Android	Clang 3.8	Provided by NDK r12
iOS	Xcode 7.3	
Linux	Clang 3.8	
macOS	Xcode 7.3	
Windows	Visual Studio 2015	

Geekbench 4.1 (and later) is built using the following compilers:

Platform	Compiler	Comment
Android	Clang 3.8	Provided by NDK r13b
iOS	Xcode 8.2	
Linux	Clang 3.9.1	
macOS	Xcode 8.2	
Windows (Intel)	Visual Studio 2015	
Windows (ARM)	Visual Studio 2017	

Runtime

Geekbench 4 groups CPU workloads into two sections:

1. Single-Core Workloads
2. Multi-Core Workloads

Each section is grouped into four subsections:

1. Cryptography Workloads
2. Integer Workloads
3. Floating-Point Workloads
4. Memory Workloads

Geekbench inserts a pause (or gap) between each workload to minimize the effect thermal issues have on workload performance. Without this gap, workloads that appear later in the benchmark would have lower scores than workloads that appear earlier in the benchmark.

The default gap is 2 seconds for both single-core and multi-core workloads.

Scores

Geekbench 4 provides two composite scores: single-core and multi-core. These scores are computed using a weighted arithmetic mean of the subsection scores. The subsection scores are computed using the geometric mean of the scores of the workloads contained in that section.

Subsection	Weight
Cryptography	5%
Integer	45%
Floating Point	30%
Memory	20%

Cryptography Workloads

AES

The Advanced Encryption Standard (AES) defines a symmetric block encryption algorithm. AES encryption is widely used to secure communication channels (e.g., HTTPS) and to secure information (e.g., storage encryption, device encryption).

The AES workload in Geekbench 4 encrypts a 32MB string using AES running in CTR mode with a 256-bit key. Geekbench will use AES instructions when available, and fall back to software implementations otherwise.

Superior AES performance can translate into improved usability for mobile devices. See, e.g., the Ars Technica review of the Moto E.

Integer Workloads

LZMA Compression

LZMA (Lempel-Ziv-Markov chain algorithm) is a lossless compression algorithm. The algorithm uses a dictionary compression scheme (the dictionary size is variable and can be as large as 4GB). LZMA features a high compression ratio (higher than bzip2).

The LZMA workload compresses and decompresses a 450KB HTML ebook using the LZMA compression algorithm. The workload uses the LZMA SDK for the implementation of the core LZMA algorithm.

JPEG Compression

JPEG is a lossy image compression algorithm. The algorithm works by encoding an image in 8x8 blocks. Each block is transformed using a discrete cosine transform (DCT). Each DCT coefficient is quantized to achieve compression. The coarseness of the quantization determines the quality of the compressed image.

The JPEG workload compresses and decompresses an image using the JPEG compression method. The JPEG workload sets the quality parameter to “90”, a commonly-used setting for users who desire high-quality images.

The workload uses libjpeg for the implementation of the core JPEG algorithm.

Canny

Canny Edge Detection is a widely used technique in image processing and computer vision -- the paper that introduced the algorithm has over 20,000 citations according to Google Scholar. Canny is a component of many computer vision and image processing algorithms, including registration, stereo correspondence, and object recognition.

Canny is a sophisticated workload with four separate components:

- Noise removal step (Gaussian Blur)
- Gradient calculation
- Minimum maximum threshold pass
- Edge following step (BFS)

Lua

Lua is a scripting language commonly used as an embedded scripting language in larger applications (notable examples include Adobe Photoshop Lightroom and World of Warcraft).

The Lua workload executes a Lua script using the standard Lua interpreter. The script parses data from a JSON file and uses Mustache to combine the data with a template to produce an HTML file.

Dijkstra

The Dijkstra workload computes driving directions between a sequence of destinations. Similar techniques are used to compute paths in games, to route computer network traffic, and to route driving directions. The dataset for this workload is based on Open Street Map data for the Waterloo Region. The dataset has 79,392 nodes and 162,644 edges with integer weights approximating travel time along the section of road represented by the edge. The route includes a source and nine ordered destinations.

SQLite

SQLite is a self-contained SQL database engine, and is the most widely deployed database engine in the world.

The SQLite workload executes SQL queries against an in-memory database.The database is synthetically created to mimic financial data, and is generated using techniques outlined in “Quickly Generating Billion-Record Synthetic Databases” by J.Gray et al. The workload is designed to stress the underlying engine using a variety of SQL features (such as primary and foreign keys) and query keywords such as: SELECT, COUNT, SUM, WHERE, GROUP BY, JOIN, INSERT, DISTINCT, and ORDER BY. This workload measures the transaction rate a device can sustain with an in-memory SQL database.

HTML 5 Parse

Construct a parse tree from a large HTML5 document using Gumbo, Google’s HTML5 parsing library. The HTML5 document selected for this workload is the Wikipedia article “List of city nicknames in the United States”, a 1.2 MB document selected because it contains the most internal references of any Wikipedia article, and therefore contains a large amount of links, list items, superscript numbers, and text. This workload tests how

quickly a device can process an HTML5 document into a data structure required by web browsers.

HTML 5 DOM

This workload constructs a HTML5 Document Object Model (DOM) based on the specification supplied at https://dom.spec.whatwg.org/ and performs manipulations of that DOM. The document selected for this workload is the 'Processor Benchmarks' page from browser.primatelabs.com. This workload performs several manipulations, including node insertion and deletion, and selective node class decoration, to measure how responsive those operations would be during a normal web browsing session.

Histogram Equalization

Histogram equalization is a method in image processing of contrast adjustment using the image's histogram. The Histogram Equalization workload performs this adjustment on an 2576×3872 image.

PDF Rendering

The Portable Document Format (PDF) is a standard file format used to present and exchange documents independent of software or hardware. PDF files are used in numerous ways, from government documents and forms to e-books.

The PDF workload parses and renders a 29-page PDF document. The document is mostly text with a few small images placed throughout the document. The PDF workload uses the PDFium library (which is used by Google Chrome to display PDFs).

LLVM

LLVM is a compiler infrastructure library that provides a source- and target-independent optimizer. LLVM also provides code-generation support for several architectures.

The LLVM workload processes an LLVM IR (intermediate representation) file through the LLVM optimizer and code-generation routines. The LLVM IR file was generated from a 3,900 line C source file using Clang. The workload uses ARM as the target architecture for code generation.

Camera

Camera replicates a photo sharing application like Instagram. Camera merges several steps into one workload:

- AES decryption
- AES key generation using PBKDF HMAC SHA1
- SHA2 checksum generation
- JSON parsing
- JPEG compression and decompression
- PNG decompression
- Image compositing
- Image filters (gaussian blur, contrast)
- SQLite (INSERT the processed image metadata into a database)

All steps run on the CPU and are not accelerated by the GPU.

Floating Point Workloads

SGEMM

GEMM (General Matrix Multiplication) computes the result C = AB + C, where A, B, and C are matrices.

The GEMM workload uses 512x512 single-precision matrices. The GEMM implementations are written using hand-tuned vector instructions for the following instruction sets:

- AVX512 (Geekbench 4.1 and later)
- AVX + FMA
- AVX
- SSE3
- SSE2
- ARMv8 NEON
- ARMv7 NEON

SFFT

FFT (Fast Fourier Transform) decomposes an input signal into a linear combination of a basis of trigonometric polynomials. FFT is a core algorithm in many signal-processing applications.

The FFT workload executes an FFT on a 32MB input buffer operating in 16KB chunks. This is similar to how FFT is used to perform frequency analysis in an audio processing application.

The FFT implementations are written using hand-tuned vector instructions for the following instruction sets:

- AVX512 (Geekbench 4.1 and later)
- AVX2
- AVX
- SSE3
- SSE2
- ARMv8 NEON
- ARMv7 NEON

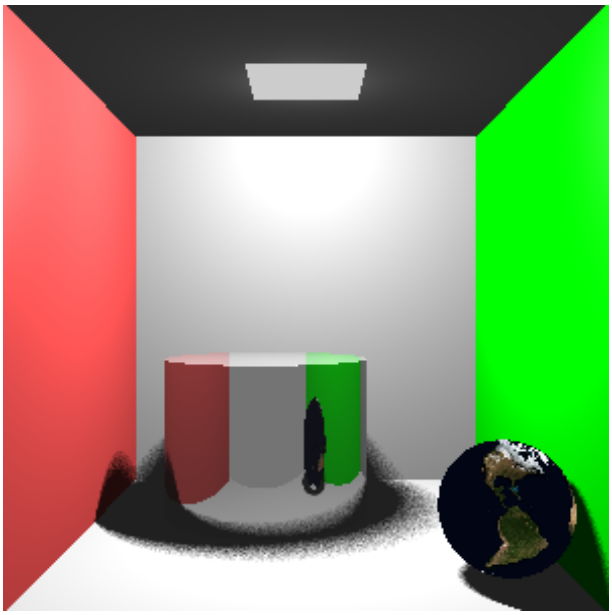
N-Body Physics

Computes a 3D gravitation simulation using the Barnes-Hut method. To compute the exact gravitational force acting on a particular body x in a field of N bodies requires N – 1 force computations. The Barnes-Hut method reduces the number of force computations by approximating as a single body any tight cluster of bodies that is far away from x. It does this efficiently by dividing the space into octants — eight cubes of equal size — and recursively subdividing each octant into octants, forming a tree, until each leaf octant contains exactly one body. This recursive subdivision of the space requires floating point operations and non-contiguous memory accesses.

Ray Tracing

The Ray Tracing workload is an expanded implementation of the same workload from Geekbench 3. Reflection, refraction, and translucency effects have been added, along with texture mapping and bump mapping. Ray Trace also implements soft shadows via area lights.

The Ray Trace workload renders our version of the Cornell Box, which is traditionally used to test rendering software. The scene includes a reflective cylinder and a texture-mapped sphere. The two objects are surrounded by a mesh (the box) and a diffuse light source is positioned at the top of the box. The rendered image is 300x300 pixels.



Rigid Body Physics

Computes a 2D physics simulation for rigid bodies that includes collisions and friction. The simulation is implemented using the popular Box2D physics library.

HDR

Processes the raw data from a digital camera sensor into an RGB image. RAW Processing has several steps: debayering, denoising, colour correction, gamma correction, sharpen, and quantization.

Gaussian Blur

Gaussian Blur blurs an image using a Gaussian spatial filter. Gaussian Blurs are widely used in software, both in operating systems to provide interface effects, and in image editing software to reduce detail and noise in an image. Gaussian Blurs are also used in computer vision applications to enhance image structures at different scales.

The Gaussian Blur workload blurs an image using a Gaussian spatial filter. While the Gaussian Blur implementation supports an arbitrary sigma, the workload uses a fixed sigma of 1.0f. This sigma translates into a filter diameter of 9 pixels.

Speech Recognition

Perform recognition of arbitrary English speech using PocketSphinx, a widely used library that uses HMM (Hidden Markov Models) to perform speech recognition. Using speech to interact with smartphones is becoming more popular with the introduction of Siri, Google Now, and Cortana, and this workload tests how quickly a device can process sound and recognize the words that are being spoken.

Face Detection

Implementation of face detection using the algorithm presented in “Rapid Object Detection using a Boosted Cascade of Simple Features” (2001) by Viola and Jones.

Memory Workloads

Memory Copy

Measures the performance of the system-provided memcpy() routine using different sizes and randomized offsets in a large memory buffer. The sizes are selected based on the distribution of memcpy() calls in user applications.

Memory Latency

Measures the latency of system memory by traversing a circular linked-list. The nodes of the linked list are arranged in such a way so as to reduce the number of TLB cache misses (1 per page rather than 1 per access).

Memory Bandwidth

Measures sustained memory bandwidth. The implementation is written using hand-tuned vector instructions for the following instruction sets:

- AVX
- SSE2
- ARMv7 NEON