Geekbench 5
CPU Workloads

Introduction

This document outlines the workloads included in the Geekbench 5 CPU Benchmark suite.

CPU Benchmark scores are used to evaluate and optimize CPU and memory performance using workloads that include data compression, image processing, machine learning, 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 7.0 “Nougat”	
iOS	iOS 12	
Linux	Ubuntu 16.04 LTS	CentOS 7, RHEL 7 also supported
macOS	macOS 10.13	
Windows	Windows 10	

Compilers

Geekbench 5.0 is built using the following compilers:

Platform	Compiler	Comment
Android	Clang 8.0	Provided by NDK r20
iOS	Xcode 10.3	
Linux	Clang 8.0	
macOS	Xcode 10.3	
Windows	Clang 8.0	

Geekbench 5.1 and later is built using the following compilers:

Platform	Compiler	Comment
Android	Clang 9.0	Provided by NDK r21
iOS	Xcode 11.2	
Linux	Clang 9.0	
macOS	Xcode 11.2	
Windows	Clang 9.0	

Runtime

Geekbench 5 groups CPU workloads into two sections:

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

Each single-core workload has a multi-core counterpart, and vice versa. Each section is grouped into three subsections:

1. Cryptography Workloads
2. Integer Workloads
3. Floating-Point 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 1second for both single-core and multi-core workloads.

Scores

Geekbench 5 scores are calibrated against a baseline score of 1,000 (which is the score of a Dell Precision 3430 with a Core i3-8100 processor). Higher scores are better, with double the score indicating double the performance.

Geekbench 5 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 subsection.

Subsection	Weight
Cryptography	5%
Integer	65%
Floating Point	30%

Cryptography Workloads

AES-XTS

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-XTS workload in Geekbench 5 encrypts a 128MB buffer using AES running in XTS mode with a 256-bit key. The buffer is divided into 4K blocks. For each block, the workload derives an XTS counter using the SHA-1 hash of the block number. The block is then processed in 16-byte chunks using AES-XTS, which involves one AES encryption, two XOR operations, and a GF(2^{128}) multiplication.

Geekbench will use AES (including VAES) and SHA-1 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

Text Compression

The Text Compression workload uses LZMA to compress and decompress an HTML ebook. 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 Text Compression workload compresses and decompresses a 2399KB HTML ebook using the LZMA compression algorithm with a dictionary size of 2048KB. The workload uses the LZMA SDK for the implementation of the core LZMA algorithm.

Image Compression

The Image Compression workload compresses and decompresses a photograph using the JPEG lossy image compression algorithm, and compresses and decompresses a CSS sprite using the PNG lossless image compression algorithm.

The photograph is a 24 megapixel image, and the JPEG quality parameter is set to “90”, a commonly-used setting for users who desire high-quality image. JPEG compression is implemented by the libjpeg-turbo library.

The CSS sprite is a 3 megapixel image. PNG compression is implemented by the libpng and zlib-ng libraries.

The Image Compression workload compresses and decompresses a photograph using JPEG, and a CSS sprite using PNG. The workload sets the JPEG quality parameter to “90”, a commonly-used setting for users who desire high-quality images.

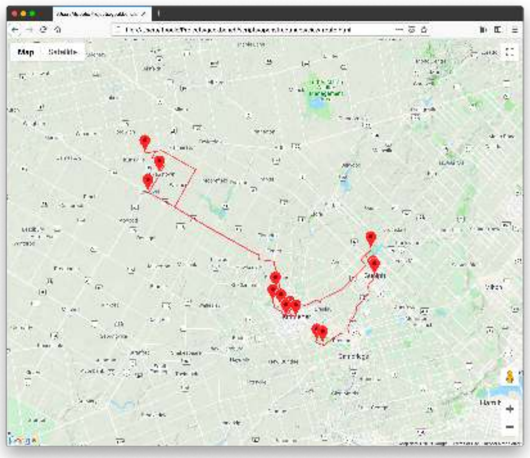
The workload uses libjpeg-turbo for the implementation of the core JPEG algorithm, and libpng for the implementation of the core PNG algorithm.

JPEG Image

PNG Image (detail)

Navigation

The Navigation workload computes driving directions between a sequence of destinations using Dijkstra's algorithm. Similar techniques are used to compute paths in games, to route computer network traffic, and to route driving directions. The dataset contains 216,548 nodes and 450,277 edges with weights approximating travel time along the road represented by the edge. The route includes 13 destinations. The dataset is based on Open Street Map data for Ontario, Canada.



Navigation Route

HTML5

The HTML5 workload models DOM creation from both server-side rendered (SSR) and client-side rendered (CSR) HTML5 documents. For the SSR document, the HTML5 workload uses the Gumbo HTML5 parser to create the DOM by parsing an HTML file. For the CSR document, the HTML5 workload uses the Gumbo HTML5 parser to create the DOM by parsing an HTML file, then uses the Duktape JavaScript engine to extend the DOM.

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.

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 PDF map of Crater Lake National Park at 200dpi. The PDF workload uses the PDFium library (which is used by Google Chrome to display PDFs).

Text Rendering

The Text Rendering workload parses a Markdown-formatted document and renders it as rich text to a bitmap. The Text Rendering workload uses the following libraries as part of the workload:

- GitHub Flavored Markdown, used to parse the Markdown document.
- FreeType, used to render fonts.
- ICU (International Components for Unicode), used for boundary analysis.

The Text Rendering workload input file is 1721 words long, and produces a bitmap that is 1275 pixels by 9878 pixels in size.

Clang

Clang is a compiler front end for the programming languages C, C++, Objective-C, Objective-C++, OpenMP, OpenCL, and CUDA. It uses LLVM as its back end.

The Clang workload compiles a 1,094 line C source file (of which 729 lines are code). The workload uses AArch64 as the target architecture for code generation.

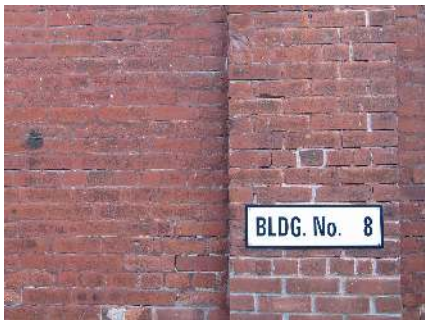
Camera

The Camera workload simulates some of the actions a camera application or photo-sharing social network application might perform. The Camera workload simulates applying a filter to an image and preparing it for upload:

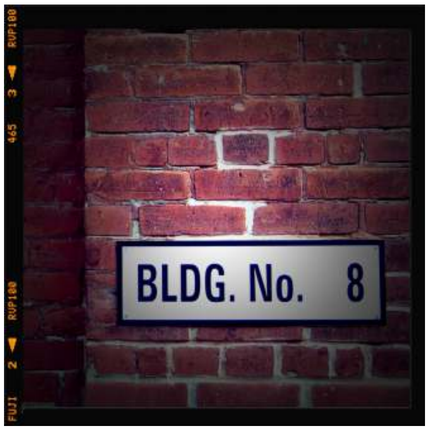
- Crop an image to a square aspect ratio.
- Load a filter definition from a JSON file and execute the individual filter operations:
  - Adjust image contrast.
  - Blur the image.
  - Composite a vignette effect onto the image.
  - Composite a border onto the image.
- Compress the output image into a JPEG file.
- Compute the SHA-2 hash of the JPEG file.

The Camera workload also simulates preparing photos for display in a UI picker by generating thumbnails for image:

- Query a SQLite database to determine which images are missing thumbnails.
- Generate a thumbnail with a longest edge of 224 pixels.



Camera Input Image



Camera Output Image

Floating Point Workloads

N-Body Physics

The N-Body Physics workload 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.

The N-Body Physics workload operates on 16,384 bodies arranged in a "flat" galaxy with a massive black hole at its centre.

Rigid Body Physics

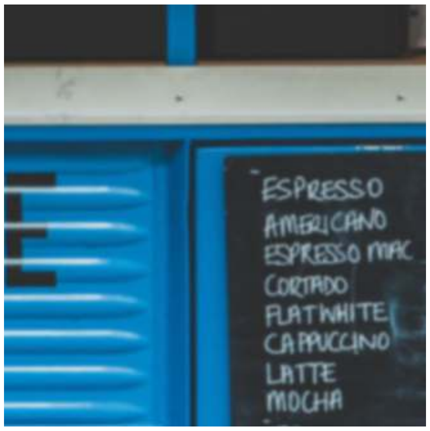
The Rigid Body Physics workload computes a 2D physics simulation for rigid bodies that includes collisions and friction. The workload uses the Lua programming language to initialize and manage the physics simulation, and uses the Box2D physics library to perform the actual physics calculations.

Gaussian Blur

The Gaussian Blur workload 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 24 megapixel image using a Gaussian spatial filter. While the Gaussian blur implementation supports an arbitrary sigma, the workload uses a fixed sigma of 3.0f. This sigma translates into a filter diameter of 25 pixels by 25 pixels.

Gaussian Blur Input Image (Detail)



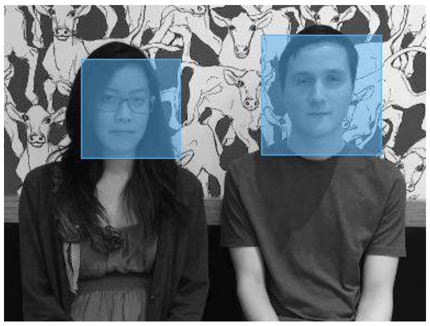
Gaussian Blur Output Image (Detail)

Face Detection

Face detection is a computer vision technique that identifies human faces in digital images. One application of face detection is in photography, where camera applications use face detection for autofocus.

The Face Detection workload uses the algorithm presented in “Rapid Object Detection using a Boosted Cascade of Simple Features” (2001) by Viola and Jones. The algorithm can produce multiple boxes for each face. These boxes are reduced to a single box using non-maximum suppression.

Face Detection Input Image

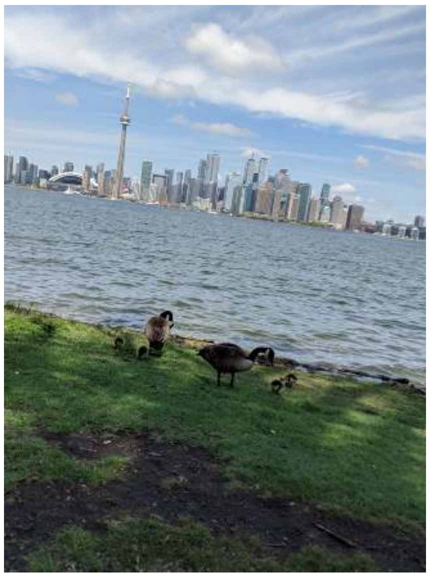

Face Detection Output

Horizon Detection

The Horizon Detection workload searches for the horizon line in an image. If the horizon line is found, the workload rotates the image to make the horizon line level.

The workload first applies a Canny edge detector to the image to reduce details, then detects lines in the image using the Hough transform, and then picks the line with the maximum score as the horizon. The workload rotates the image so the horizon line is level in the image.

The input image is a 9 megapixel photograph.


Horizon Detection Input Image

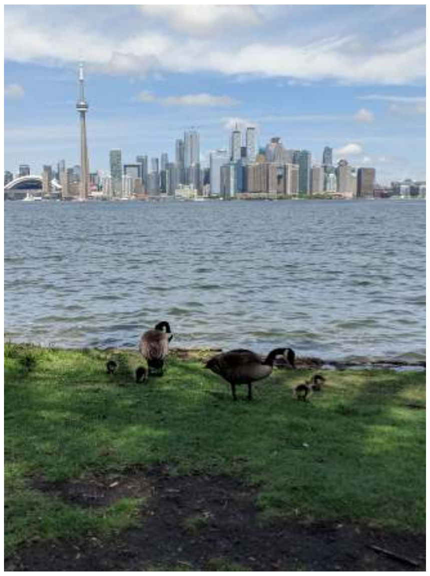
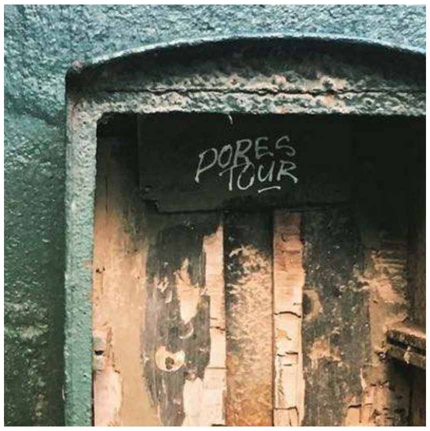

Horizon Detection Output Image

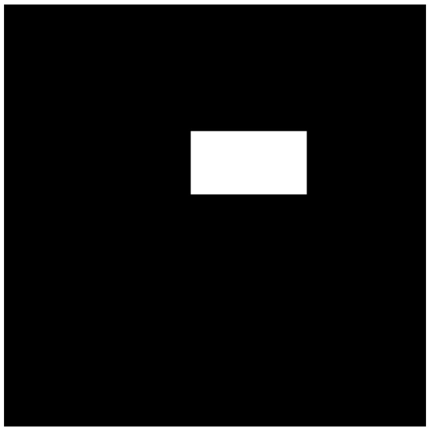
Image Inpainting

The Image Inpainting workload takes an input image with an undesirable region (indicated via a mask image) and uses an inpainting scheme to reconstruct the region using data from outside the undesirable region.

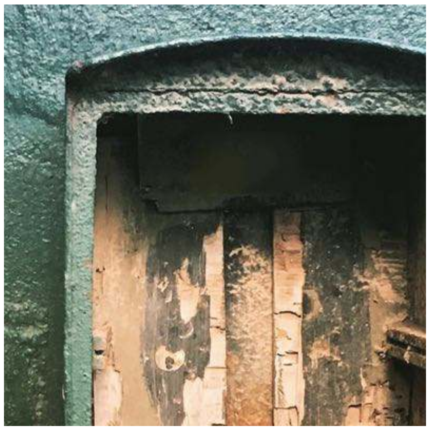
The Image Inpainting workload operates on 1 megapixel images.



Inpaint Input Image (Detail)



Inpaint Mask Image (Detail)



Inpaint Output Image (Detail)

HDR

The HDR workload takes four standard dynamic range (SDR) images and produces a high dynamic range (HDR) image. Each input image is 3 megapixels in size. The HDR workload uses the algorithm described in the paper, "Dynamic Range Reduction inspired by Photoreceptor Physiology" by Reinhard and Devlin, and produces superior images as compared to the tone mapping algorithm in Geekbench 4.

Ray Tracing

Ray tracing is a rendering technique. Ray tracing generates an image by tracing the path of light through an image plane and simulating the effects of its encounters with virtual objects. This method is capable of producing high-quality images, but these images come at a high computational cost.

The Ray Tracing workload uses a k-d tree, a space-partitioning data structure, to accelerate the ray intersection calculations.

The Ray Tracing workload operates on a scene with 3,608 textured triangles. The rendered image is 768 pixels by 768 pixels.

Ray Tracing Output Image

Structure from Motion

Augmented reality (AR) systems add computer-generated graphics to real-world scenes. The systems must have an understanding of the geometry of the real-world scene in order to properly integrate the computer-generated graphics. One approach to calculating the geometry is through Structure from Motion (SfM) algorithms.

The Structure from Motion workload takes two 2D images of the same scene and constructs an estimate of the 3D coordinates of the points visible in both images.

Speech Recognition

The Speech Recognition workload performs recognition of arbitrary English speech using PocketSphinx, a widely used library that uses HMM (Hidden Markov Models).

Using speech to interact with smartphones is becoming more popular with the introduction of Siri, Google Assistant, and Cortana, and this workload tests how quickly a device can process sound and recognize the words that are being spoken.

Machine Learning

The Machine Learning workload is an inference workload that executes a Convolutional Neural Network to perform an image classification task. The workload uses MobileNet v1 with an alpha of 1.0 and an input image size of 224 pixels by 224 pixels. The model was trained on the ImageNet dataset.

Workload Characteristics

Geekbench 5 CPU workloads exercise a wide range of device subsystems, and can be partly characterized by the following characteristics.

Instructions Per Cycle

Instructions per Cycle, or IPC, is a measure of the effective instruction throughput of a processor, which correlates with higher performance. It is measured as the number of instructions executed for a workload divided by the number of cycles used for that workload.

Branch Miss Rate

Branch miss rate, or branch misprediction rate, is a measure of how frequently a system incorrectly predicts a code branch, which correlates with lower performance. It is measured as a percentage of total branches. Whenever a code path branches into multiple cases, a system executing that code attempt to predict which case will be true in order to pre-fetch data or pre-execute instructions. When operating on well-ordered data, systems can correctly predict (or "hit") branches more frequently, improving performance.

Working Set Size

Working Set size is a measure of the amount of memory that a program uses, either by reading from it or writing to it. It is expressed in bytes, but operates by measuring memory used at the granularity of pages. Programs with a large working-set size run on systems with small caches can encounter more cache misses, where the program cannot find the data it needs in a cache and must query a larger cache or main memory. Cache misses negatively impact performance.

Cache Hit and Miss Rates

Cache hit and miss rates are a measure of how frequently a program finds data in a system's cache when attempting to read from or write to memory. Cache misses correlate with lower performance. A hit or miss rate for a cache is measured as a percentage of total accesses to that cache. When a program attempts to access data, it first checks if that data has been copied to a lower level of cache (e.g. the L1d cache). If it has been, it can operate on it there quickly. If it is not cached there, the program must check higher levels of cache or main memory, which negatively impacts performance.

As an example of cache measurements, if a workload cannot find data in the L1d cache or the L2 cache, but finds it in the L3 cache, it is recorded as an L1d miss, an L2 miss, and an L3 hit. If data is accessed regularly, or if the system predicts that data will be accessed soon, it is more likely to be cached.

Most systems have an L1 Instruction , L1 Data, L2, and L3 cache. The L1 Data, L2, and L3 caches are all used to improve the speed of fetching data, and are increasingly large and slow to access. If a program misses the L3 cache, and there is no L4 cache, it accesses main memory which is particularly large and slow to access. The L1 Instruction cache is an instruction cache, used to improve the speed of fetching instructions.

Workload Characteristics Data

The following tables report the following characteristics for the single-core and multi-core CPU workloads:

- Runtimes
- Instructions Per Clock
- Branch Miss Rate
- Working Set Size
- L1 Instruction Cache Miss Rate
- L1 Data Cache Miss Rate
- L2 Cache Miss Rate
- L3 Cache Miss Rate

The data was collected on an Intel Core i5-6400 running Ubuntu 18.04.

Workload Runtimes

	Single-Core Runtime	Multi-Core Runtime
AES-XTS	60	22
Text Compression	977	1211
Image Compression	289	309
Navigation	424	728
HTML5	93	100
SQLite	76	89
PDF Rendering	278	314
Text Rendering	41	51
Clang	272	297
Camera	104	117
N-Body Physics	717	207
Rigid Body Physics	275	289
Gaussian Blur	507	134
Face Detection	150	158
Horizon Detection	421	471
Image Inpainting	156	213
HDR	506	565
Ray Tracing	105	170
Structure from Motion	1122	1236
Speech Recognition	388	728
Machine Learning	32	66

Single-Core Workload Characteristics

	IPC	Branch Miss	Working Set (Bytes)
AES-XTS	1.9	0.1	268,882,739
Text Compression	1.2	10.5	23,225,958
Image Compression	2.7	2.1	48,694,067
Navigation	0.9	5.0	42,862,182
HTML5	2.6	0.5	30,010,572
SQLite	2.4	0.8	4,440,064
PDF Rendering	2.9	1.3	96,491,110
Text Rendering	2.5	0.9	52,686,848
Clang	1.5	2.7	23,905,075
Camera	3.2	0.3	96,450,150
N-Body Physics	0.9	5.7	11,009,228
Rigid Body Physics	1.9	1.8	3,202,252
Gaussian Blur	2.7	0.1	191,755,059
Face Detection	2.7	0.2	2,016,051
Horizon Detection	2.2	2.6	140,408,422
Image Inpainting	1.4	0.4	52,924,416
HDR	3.2	0.1	109,992,345
Ray Tracing	2.0	2.0	28,647,424
Structure from Motion	2.9	0.9	122,825,113
Speech Recognition	1.1	4.9	55,885,824
Machine Learning	2.8	0.2	95,648,153

Single-Core Workload Cache Characteristics

	L1I Miss	L1D Miss	L2 Miss	L3 Miss
AES-XTS	0.0	0.4	3.6	9.6
Text Compression	0.0	3.9	55.4	8.5
Image Compression	0.0	1.5	3.5	3.8
Navigation	0.0	7.8	43.0	28.5
HTML5	0.3	0.7	39.3	26.2
SQLite	3.3	2.1	8.9	0.0
PDF Rendering	0.5	0.8	11.4	41.8
Text Rendering	0.5	0.5	25.1	45.5
Clang	6.1	3.1	38.9	1.8
Camera	0.0	6.3	5.1	5.4
N-Body Physics	0.0	24.0	55.3	2.0
Rigid Body Physics	0.4	7.3	2.0	0.0
Gaussian Blur	0.0	15.7	12.4	0.0
Face Detection	0.0	1.5	12.8	0.0
Horizon Detection	0.0	3.2	10.6	81.0
Image Inpainting	0.0	9.0	57.5	8.3
HDR	0.0	0.2	5.0	23.6
Ray Tracing	0.0	24.6	6.6	2.0
Structure from Motion	0.0	3.3	7.9	37.1
Speech Recognition	0.0	10.8	69.3	14.0
Machine Learning	0.0	6.6	25.6	14.5

Multi-Core Workload Characteristics

	IPC	Branch Miss	Working Set (Bytes)
AES-XTS	1.9	0.1	268,939,264
Text Compression	1.0	10.5	101,616,844
Image Compression	2.7	2.1	185,000,755
Navigation	0.6	5.0	170,098,688
HTML5	2.6	0.5	113,223,270
SQLite	2.4	0.8	16,428,236
PDF Rendering	2.7	1.3	379,792,588
Text Rendering	2.3	0.9	207,525,478
Clang	1.5	2.7	42,861,363
Camera	3.1	0.3	376,146,329
N-Body Physics	0.9	5.7	11,141,939
Rigid Body Physics	1.8	1.8	9,274,163
Gaussian Blur	3.2	0.1	191,800,115
Face Detection	2.7	0.2	5,337,088
Horizon Detection	2.1	2.6	487,233,126
Image Inpainting	1.2	0.4	196,447,436
HDR	3.1	0.1	456,980,889
Ray Tracing	2.3	0.6	50,994,380
Structure from Motion	2.9	0.9	468,773,273
Speech Recognition	0.7	4.9	219,734,835
Machine Learning	1.8	0.2	381,136,896

Multi-Core Workload Cache Characteristics

	L1I Miss	L1D Miss	L2 Miss	L3 Miss
AES-XTS	0.0	0.4	6.8	13.3
Text Compression	0.0	4.0	55.6	23.2
Image Compression	0.0	1.5	3.4	5.8
Navigation	0.0	8.2	45.5	39.2
HTML5	0.3	0.6	35.0	27.4
SQLite	3.3	2.2	9.3	3.2
PDF Rendering	0.5	0.8	11.8	52.9
Text Rendering	0.5	0.5	24.5	57.5
Clang	6.1	3.1	39.2	8.3
Camera	0.0	6.5	5.1	6.4
N-Body Physics	0.0	24.0	56.1	2.7
Rigid Body Physics	0.4	7.3	3.3	0.0
Gaussian Blur	0.0	6.1	1.4	80.0
Face Detection	0.0	1.5	15.9	0.0
Horizon Detection	0.0	3.2	11.6	67.6
Image Inpainting	0.0	9.2	57.7	32.7
HDR	0.0	0.3	6.7	26.3
Ray Tracing	0.0	25.3	5.4	1.1
Structure from Motion	0.0	3.2	8.9	41.9
Speech Recognition	0.0	10.8	70.8	59.2
Machine Learning	0.0	6.7	33.1	47.9