

OABench™ 2.0

software
benchmark
data book



An Industry-Standard Benchmark Consortium

Table of Contents

Bezier	2
Dithering	4
Ghostscript®	10
Rotate	14
Text (Text Parsing)	20



OABench™ Version 2.0

Benchmark Name: Bezier

Highlights

- Benchmarks the classic Bezier curve algorithm
- Interpolate a set of points defined by the four points of a Bezier curve (two end points, two intermediate points)
- Fixed point and floating point versions available
- A component of the EEMBC OAV2mark™
- Four new data files implemented in Version 2
- Bezier curves are the backbone of computer graphics, font renderings and design, and computer graphics.
- Implements Cyclical Redundancy Checksum (CRC) for self-checking in integer mode, and SNR for self checking in floating point mode.

History, Application, and Restrictions

Pierre Étienne Bézier, a French engineer, created a mathematical numerical analysis technique for drawing parametric curves. The problem solved was how to draw curves based on fixed data points. The creator of the first algorithm to implement Bezier curves was Paul de Casteljau.

Bezier curves can be linear, quadratic, cubic, or even triangles. In computer science, one of the primary applications of Bezier curves is the creation and smoothing of fonts on-screen and in a printer for the printed page. For example, TrueType® fonts use Bezier curves. TrueType, PostScript®, Ghostscript, The GIMP, and many other applications use Bezier splines with cubic Bezier curves for drawing shapes. Translation, scaling, and rotation on the curve can be accomplished by applying the respective transform on the control points of the curve (the points).

As with all EEMBC source code, the Bezier benchmark is not to be used in any commercial product whatsoever.

Benchmark Description

In EEMBC's OABench office automation benchmark suite, the calculations interpolate a set of points defined by the four points of a Bezier curve. Two endpoints and two control points define the curve. The points are in 2D space, and are defined using floating point (double precision) or integer variables. The algorithm makes use of configuration constants in the header file `bez.h`. This includes the number of points to interpolate for each curve as well as the overall loop count. The main function makes use of a call to the test harness `malloc()` to create an array of curve structures for all the input data before processing starts. The first line in the input file defines the number of points following in the rest of the file.

This benchmark evaluates the parametric function for Bezier curve



An Industry-Standard Benchmark Consortium

Benchmark Description (continued)

$$P(t) = p_0 * (1-t)^3 + 3 * p_1 * t * (1-t)^2 + 3 * p_2 * (t^2) * (1-t) + p_3 * t^3$$

1000 iterations is the default, 10 for CRC verification runs. There are four data sets, one of which is reserved for profiling initialization.

Analysis of Computing Resources

The benchmark uses division, multiplication, and scalar processing. There are two loops (inner and outer), so efficient compilers and architectures can take advantage of this, but the function `interpolatePoints()` cannot be optimized away. This benchmark is almost exclusively CPU bound, and the quality of the math library has an effect on performance.

Optimizations Allowed

Out-of-the-Box / Standard C Full Fury / Optimized

- The C code must not be changed for Out-of-the-Box unless it must be modified to get it to compile. All changes must be documented, authorized by the certification authority, and must not have a performance impact.
- For Out-of-the-Box, additional hardware can be used if it does not require code changes.
- All Optimized libraries must be part of the standard compiler package, and/or available to all customers.
- Test harness changes may be made for portability reasons if they do not impact performance.
- For Optimized, the basic algorithm may not be changed, but the code may be rewritten in assembler. Re-writing the code to take advantage of parallelism is allowed so long as the correct answers are achieved using any arbitrary keys (not just those supplied in the benchmark code). You may not optimize out the function `interpolatePoints()`.
- For Optimized, optimized libraries can be used if they are publicly available.
- For floating point, SNR is used to evaluate the quality of the output. Double precision is required to achieve certifiable SNR.
- For Optimized, in lining is allowed.
- Additional data files may be used during certification to ensure the correctness of the optimized benchmark. You should **not** assume data patterns during optimization.
- Profile directed optimization is allowed using training data set 1, `bezdata1.txt`.



OABench™ Version 2.0

Benchmark Name: Dithering

Highlights

- Benchmarks potential performance of a printer application
- Uses Floyd-Steinberg Error Diffusion Dithering Algorithm (1975)
- Converts 8bpp grayscale image to 1bpp monochrome.
- Largely integer math with shifts and logical compares
- A component of the EEMBC OAV2mark™
- Implements 11 new data files compared with OABench Version 1.1
- Implements cyclical redundancy checksum (CRC) for self-checking as well as the ability to view processed output files (new in Version 2)
- Jarvis Grayscale Dithering included for debugging purposes

History, Application, and Restrictions

The dithering benchmark is representative of color and monochrome printer applications. The algorithm converts a grayscale image into a form ready for printing using the Floyd-Steinberg Error Diffusion dithering algorithm. This algorithm propagates an error quantity from image row to image row, effectively diffusing errors from the rendering calculations and preventing unwanted printing artifacts, such as banding.

References: Robert Ulichney (1987); *Digital Halftoning*, The MIT Press, Cambridge, Massachusetts; pp. 239-242

Benchmark Description

The benchmark changes a grayscale 8bpp image to a 1bpp monochrome image, using a Floyd-Steinberg Error Diffusion dithering algorithm. It uses two image buffers (one for the source image and a second for the generated output) and two line buffers to hold error data. Two “error” arrays are used — one for saving the errors from the current row (used to dither the next row) and one from the previous row, used to diffuse the errors from that row to the current pixel. This array must be zeroed out before the first row, to ensure that no spurious data is left there.

The error array is created such that there is one extra value at either end. This eliminates special processing at the start and end of each row (but requires zeroing the additional columns).

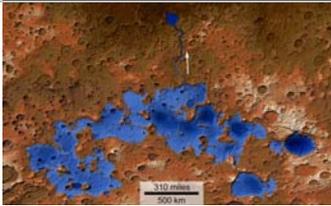
Each pixel of the input image file is processed as follows:

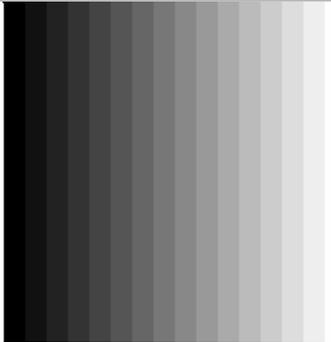
1. Calculate an “error” value using the history buffer (weighted values of surrounding pixels).
2. Calculate a monochrome output pixel value and store.
3. Store “error” value to next line history buffer.

Benchmark Description (continued)

For Version 2, datasets are now taken from external data files (the same .pgm files as found in DENbench Version 1.0), and data is output to files as well to aid in verification. The input data files are:

Data Name	Data File	Attributes	Picture
Data 1	DavidAndDogs	564x230, 256 shades of gray. The image has 215 unique colors.	
Data 2	DragonFly	606x896, 16 million colors. The image has 162,331 unique colors. Highlights, wide range of contrasts.	
Data 3	EEMBCGroup Shot-Miami	EEMBCGroupShotMiami: 640x480, 16 million colors. The image has 181,872 unique colors. Large number of fleshtones, highest number of unique colors in data set.	

Data 4	Galileo	290x415, 16 million colors. The image has 36,557 unique colors, and also contains "real black" for over 30% of the picture, which is interesting from an optimization perspective.	
Data 5	Goose	320x240, 256 colors. The image has 22921 unique colors.	
Data 6	Mandrake	320x240, 16 million colors. The image has 71,482 unique colors.	
Data 7	MarsFormer Lakes	800x482, 16 million colors. The image has 91,152 unique colors.	

Data 8	Rose256	227x149, 256 colors. The image contains 256 unique colors.	
Data 9	Dragon	370 x 384, 256 colors, 88 unique colors.	
Data 10	Gradient	A grayscale gradient shading test pattern. 256 x 256, 256 colors.	

Data 11	Medium	A long, thin, black and white picture having 37 x 345 pixels, 256 colors, and 255 unique colors.	
200 iterations are the default, 2 for CRC verification runs.			

Analysis of Computing Resources

The benchmark effectively stresses four areas of the target CPU:

- Indirect references used for managing internal buffers
- Manipulation of large data sets, since large images will stress the cache
- Ability to manipulate packed-byte quantities, used to hold grayscale pixel information
- Ability to perform four byte-wide multiply-accumulate operations per pixel

The instruction mix for this benchmark is very architecture and compiler dependent, since the main part of the inner loop can be implemented with add/sub/shift, or multiplies, or MAC instructions depending on hardware characteristics.

Analysis of Computing Resources (continued)

The C library function `memset()` is called twice per iteration (for the output buffer and for the error buffers). No floating-point calculations are used. The code size is small and the data size is large. By using multiple data sets (and private EEMBC data for certification), data-focused optimization is eliminated.



An Industry-Standard Benchmark Consortium

Optimizations Allowed **Out of the Box / Standard C Full Fury / Optimized**

- The C code must not be changed for Out-of-the-Box unless it must be modified to get it to compile. All changes must be documented, authorized by the certification authority, and must not have a performance impact.
- For Out-of-the-Box, additional hardware can be used if it does not require code changes.
- All optimized libraries must be part of the standard compiler package, and/or available to all customers.
- Test harness changes may be made for portability reasons if they do not impact performance.
- For Optimized, the basic algorithm may not be changed, but the code may be rewritten in assembler. Rewriting the code to take advantage of parallelism is allowed so long as the correct answers are achieved using any arbitrary keys (not just those supplied in the benchmark code).
- For Optimized, optimized libraries can be used if they are publicly available.
- For Optimized, in lining is allowed.
- Additional data files may be used during certification to ensure the correctness of the optimized benchmark. You should *not* assume data patterns during optimization.
- Profile directed optimization is allowed using train data set 1, DavidAndDogs.pgm.



OABench Version 2.0

Benchmark Name: Ghostscript®

Highlights

- Benchmarks potential performance of a PostScript® printer engine
- 9 different test files stress different printer aspects
- 6 different output drivers stress different aspects of output formatting and rendering
- Based on open source AFPL Ghostscript® code base

Application

The Ghostscript benchmark provides an indication of the potential performance of an embedded processor running a PostScript printer engine. Performance is measured using nine input files reflecting different aspects of PostScript language processing, as well as six output drivers reflecting the different types of processing found in common printer engines.

Benchmark Description

Ghostscript is an application developed to render print format files on a host application. This task must perform all of the processing normally found in a PostScript printer engine. Additionally, this task must produce final output for a wide range of available printers and fax formatted results. This makes Ghostscript an ideal application to measure embedded processor capabilities to perform printer functions with a single consolidated score.

EEMBC Ghostscript* contains a significant number of functions and algorithms used in PostScript printer engines. These functions are implemented within a portable test harness that allows execution on a wide range of processors and DSPs without source code changes in the main application. This benchmark is supported by an embedded compressed RAMfile system required to support this application.

Every PostScript printer engine contains a built-in interpreter that executes PostScript instructions. The engine also contains modules for reading a variety of input raster formats, fonts for rendering text characters, and an output driver to produce each page in a format for the printer engine.

The EEMBC Ghostscript application benchmark provides:

- An interpreter for the PostScript language;
- Input modules (utilities) for reading a variety of formats, including Postscript and Encapsulated PostScript;
- Output modules (drivers) for a wide variety of raster file formats, and printers.



An Industry-Standard Benchmark Consortium

Benchmark Description (continued)

- The Ghostscript library, a set of procedures to implement the graphics and filtering capabilities that are primitive operations in the PostScript language.

Libraries are incorporated to handle graphics formats such as BMP, JPEG, PNG and tiff.

Ghostscript Benchmark Printer Drivers

Ghostscript is an interpreter for the PostScript language. A PostScript interpreter takes as input a set of commands. The output is usually a page bitmap, which is then sent to an output device such as a printer or display. PostScript is embedded in many printers.

The following printer output drivers are implemented in EEMBC Ghostscript and applied to each input file:

- TIFF G4 (Fax formatting)
- 256 Color BMP Bitmap format
- CMYK - 1bpp/2bpp/4bpp/8bpp color separated CMYK data

PostScript™ Features Supported

There are three versions of PostScript: Level 1, Level 2, and PostScript 3. Level 2 PostScript, which was released in 1992, has better support for color printing. PostScript 3, released in 1997, supports more fonts, has better graphics handling, and includes several features to speed up PostScript printing.

The following PostScript Language Level 3 features are available in EEMBC Ghostscript:

- Anti-aliased text and graphics on continuous-tone devices
- Alpha value for displays
- ICC-based color support
- Device "N" color support (6+ colors at 8 bits)
- "Argyll" color management system
- Band-at-a-time rendering for high-resolution printers

Ghostscript Input Data Set Descriptions

The following section contains a description of each input file used in the EEMBC Ghostscript application benchmark.

Rotate-fontlist	This dataset is a full listing of all standard ASCII characters of 2 fonts, printed in landscape orientation.
Banner	This dataset is comprised of several lines of text, warped to create text effects (Circle, wave). Sixteen copies are sent and printed n-up 4x4.
Presentation	This dataset is a typical business Powerpoint® presentation and includes text combined with diagonal lines for background and a vertical gradient. There is a PS version that is printed with four copies n-up (all slides are on



An Industry-Standard Benchmark Consortium

the same output page). There is a PDF version that is printed one slide per output page.

Mandel	This postscript data file includes a mathematical equation describing the Mandelbrot set combined with a text caption. This dataset results in a significant amount of floating point computation to create the output pixels. The output is set to full page at 300 DPI color.
Fractal Fern	This postscript includes a mathematical equation that yields a leaf-like shape. This dataset results in a significant amount of floating point computation to create the output pixels. The output is set to full page at 72 DPI, monochrome.
Spreadsheet	This dataset is a typical excel spreadsheet that results in two pages, with tables and charts. The output is n-up 2x2, with two copies of the spreadsheet printed to a single page.
Photo	This postscript file contains a full-color encapsulated JPEG image of a firefighter at a burning house. A caption is included. Four copies are printed full color, one copy per page.
Ebreadme	This dataset is a typical text document and includes two pages of text taken from the EnergyBench readme file. Eight copies are sent and printed n-up 4x4 on a single output page.

Training datasets only

3Dcolor:	This dataset includes postscript instructions to cover all available color space in a 3D-like cube.
Font catalogue	This dataset prints a full listing of each available font. This is based on the default script included with Ghostscript.

Analysis of Computing Resources

Ghostscript is a fully functional printer application with PostScript language interpretation, low-level graphics conversions, and printer drivers. Internally, the file system required for font selection and processing is also implemented with over 400 resources used during the processing on an embedded platform.

Profiling Analysis

The Ghostscript benchmark with the above-mentioned datasets was profiled, and the resulting data categorized to show that the following functions are being performed by the benchmark.

- Dithering / half toning (Color and Monochrome)
- Error diffusion
- Color adjustment
- Color conversion
- Image transforms (rotate, scale, clip, mirror, etc.)
- Compression
- Fill
- PostScript Interpreter



An Industry-Standard Benchmark Consortium

Optimizations Allowed

Out of the Box/Standard C Full Fury/Optimized

- The C code must not be changed for Out-of-the-Box unless it must be modified to get it to compile. All changes must be documented, authorized by the certification authority, and must not have a performance impact.
- For Out-of-the-Box, additional hardware can be used if it does not require code changes.
- All optimized libraries must be part of the standard compiler package, and/or available to all customers.
- The EEMBC Test Harness Lite must be used. Test harness changes may be made for portability reasons if they do not impact performance
- For Optimized, the basic algorithm may be changed and/or the code can be rewritten in assembler, as long as the output is identical (bit-exact) to output produced by Out-of-the-box implementation on the same platform.
- For Optimized, optimized libraries can be used if they are publicly available.
- For Optimized, hardware-assist can be used if it is on the same processor as that being benchmarked.
- For Optimized, in-lining is allowed.
- Additional data files may be used during certification to ensure the correctness of the optimized benchmark. You should not assume data patterns during optimization.
- Profile directed optimization is allowed using the training data set, colormap.ps and/or the font catalogue data set.

*This version of Ghostscript is based on AFPL 8.54.

OABench is a trademark of the Embedded Microprocessor Benchmark Consortium. PostScript is a registered trademark of Adobe Systems. Ghostscript is a registered trademark of Artifex Software, Inc. Powerpoint is a registered trademark of Microsoft Corporation. All other trademarks appearing herein are the property of their respective owners.



OABench™ Version 2.0

Benchmark Name: Rotate (Image Rotation)

Highlights

- Benchmarks potential performance of a printer application
- Uses a bitmap rotation algorithm to perform a clockwise 90° rotation on a binary image
- Largely integer math with shifts and logical compares
- Tests bit manipulation, comparison, and indirect reference capabilities
- Largely logical compares/branches and integer addition/subtraction
- A component of the EEMBC OAV2mark™
- 11 dataset files
- Implements cyclical redundancy checksum (CRC) for self-checking as well as the ability to view resultant processed output files (new in Version 2)

History, Application and Restrictions

The Rotate (Image Rotation) benchmark is representative of monochrome printer applications that must rotate binary images 90° (for example, to switch between portrait and landscape modes). This benchmark uses a bitmap rotation algorithm to perform a clockwise 90° rotation on a binary image. Rotated images are assumed to be a complete image (i.e. not rotating a bitmap within a larger image), with rows padded out to byte boundaries.

Benchmark Description

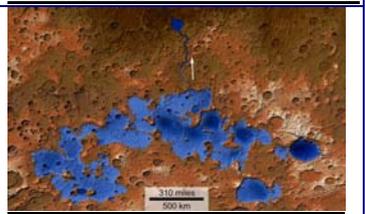
The bitmap rotation algorithm is primarily aimed at testing the bit manipulation, comparison, and indirect reference capabilities of the microprocessor. The algorithm uses a series of indirect references and bit masks to check and set individual bits in a data buffer representing a binary image. The implementation supports 8-, 16- and 32-bit data as well as little and big endian memory architectures. Two buffers are used, one for input and one for output, rather than trying to rotate the image in place.

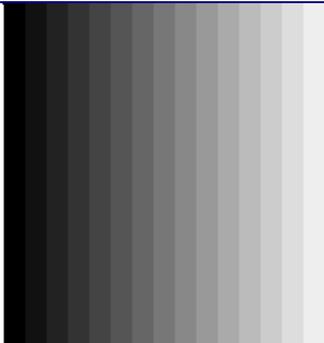
There are multiple input data buffers available to debug the benchmark. The input buffer is included in the benchmark is statically initialized data and the output buffer is created by calling the test harness memory allocation routine, `th_malloc()`. After the timed iterations have been completed, the test is run one additional time so that the results can be checked by calculating a CRC of the output buffer. The benchmark assumes 1 bit per pixel.

The C library routine `memset()` is called at the beginning of each iteration to set the output buffer to zeroes.

In OABench Version 2, datasets are taken from external data files (the same .pgm files as found in DENbench Version 1.0), and data is output to files as well to aid in verification. The input data files are:

Data Name	Data File	Attributes	Picture
Data 1	DavidAndDogs	564x230, 256 shades of gray. The image has 215 unique colors.	
Data 2	DragonFly	606x896, 16 million colors. The image has 162,331 unique colors.	
Data 3	EEMBCGroupShot-Miami	EEMBCGroupShotMiami: 640x480, 16 million colors. The image has 181,872 unique colors. Large number of fleshtones, highest number of unique colors in data set.	
Data 4	Galileo	290x415, 16 million colors. The image has 36,557 unique colors, and also contains "real black" for over 30% of the picture, which is interesting from an optimization perspective.	

Data 5	Goose	320x240, 256 colors. The image has 22921 unique colors.	
Data 6	Mandrake	320x240, 16 million colors. The image has 71,482 unique colors.	
Data 7	MarsFormerLakes	800x482, 16 million colors. The image has 91,152 unique colors.	
Data 8	Rose256	227x149, 256 colors. The image contains 256 unique colors.	
Data 9	Dragon	370 x 384, 256 colors, 88 unique colors.	

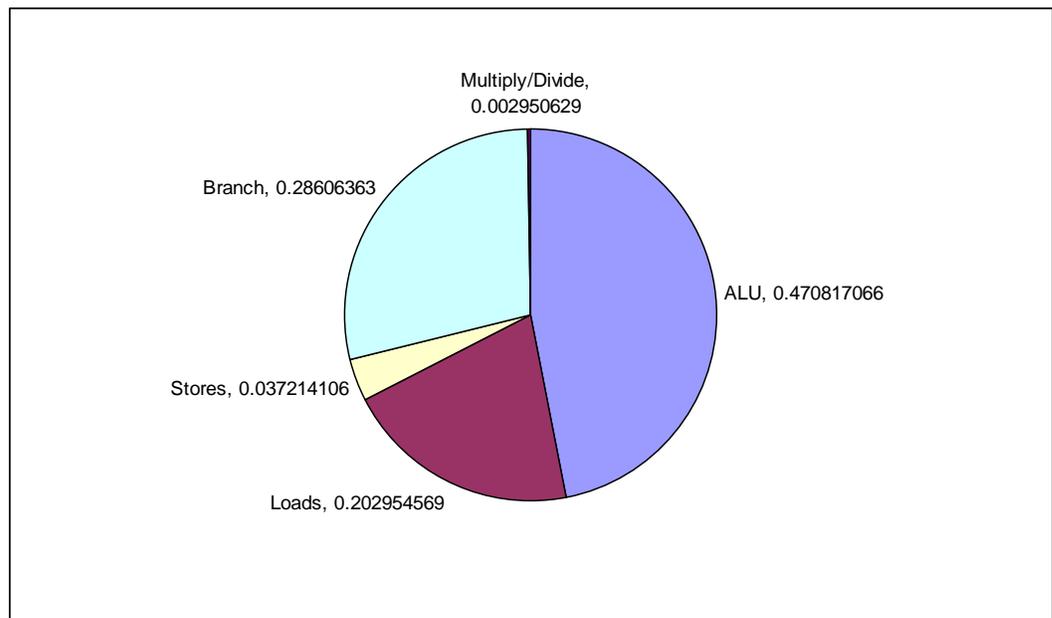
Data 10	Gradient	A grayscale gradient shading test pattern. 256 x 256, 256 colors.	
Data 11	Medium	A long, thin, black and white picture having 37 x 345 pixels, 256 colors, and 255 unique colors.	

50 iterations are the default, 2 for CRC verification runs.

Analysis of Computing Resources

The benchmark effectively stresses the bit manipulation capabilities of the target CPU.

Dynamic Instruction Mix:



The percentages are approximate and may vary across architectures. The C library function `memset()` is called once per iteration to initialize the output buffer to zeroes. No floating-point calculations are used. The code size is small and the data size is moderate. Efficient multiplication and division as well as bit-shifting. By using multiple data sets (and proprietary EEMBC Technology Center data for certification), data-focused optimization is eliminated.

Optimizations Allowed **Out of the Box / Standard C**
Full Fury / Optimized

- The C code must not be changed for Out-of-the-Box unless it must be modified to get it to compile. All changes must be documented, authorized by the certification authority, and must not have a performance impact.
- For Out-of-the-Box, additional hardware can be used if it does not require code changes.
- All optimized libraries must be part of the standard compiler package, and/or available to all customers.
- UNROLL may be selected using a `#define` to fully unroll the inner loop for Out-of-the-Box certification.
- Bits must be defined to 32 for Out-of-the-Box certification.
- Test harness changes may be made for portability reasons if they do not impact performance.
- For Optimized, the basic algorithm may not be changed, but the code may be rewritten in assembler. Rewriting the code to take advantage of parallelism is allowed so long as the correct answers are achieved using any arbitrary keys (not just those supplied in the benchmark code).
- For Optimized, the source code may be changed to take advantage of



An Industry-Standard Benchmark Consortium

additional hardware.

- For Optimized, optimized libraries can be used if they are publicly available.
- For Optimized, in lining is allowed.
- Additional data files may be used during certification to ensure the correctness of the optimized benchmark. You should **not** assume data patterns during optimization.
- Profile directed optimization is allowed using training data set 1, DavidAndDogs.pgm.



OABench™ Version 2.0

Benchmark Name: Text (Text Parsing)

Highlights

- Benchmarks potential performance of a printer interpretive control language
- Parses Boolean expressions made up of text strings
- Tests bit manipulation, comparison, and indirect reference capabilities.
- Largely shift/rotates with integer math and logical compares/branches
- A component of the EEMBC OAV2mark™
- Three data files
- Implements cyclical redundancy checksum (CRC) for self-checking

History, Application and Restrictions

The Text (Text Parsing) Benchmark is representative of a printer application where an interpretive control language like PCL or PostScript is parsed. The algorithm parses Boolean expressions represented as text lines made up of variables, constants, and operators. The variables are space separated words, from 1 to 64 characters long, the constants are single character "T" or "F" and the operators may either be single-character symbols (& | !) or their phonetic equivalents (and, or, not). Standard precedence rules for expression parsing apply.

Benchmark Description

Input to the benchmark consists of rule data files that are loaded via the EEMBC RAMfile system. OABench Version 2 has four dataset files, one of which is used for profiling, compared with just one for OABench Version 1.1. These files are found in the libtxt directory. Much of the code is generated by the cheader subsystem in Version 2. The strings consist of variables, constants, and operators separated by spaces. For example:

"sss and fred implies (red & blue) or fred"

The expression is broken down into a binary tree structure, with each branch on the tree being an operand (a single variable, or a constant, or a reference to yet another tree node representing another expression). Unary operators are stored as modifiers to each of the branches. The resulting structure is then traversed to evaluate the value of the expression.

The benchmark avoids calling a memory allocation routine by statically declaring and managing a 1,000-node buffer. After the timed iterations have been completed, the test is run one additional time and a CRC is calculated for the binary tree to be used for checking for correct operation.

For each line, the benchmark breaks the expression into a binary tree structure, where each node contains a binary expression with two operands (each with a possible unary operator) and a binary operator. The operands may be variables, constants, or pointers to further nodes which themselves



An Industry-Standard Benchmark Consortium

represent binary operations, etc.

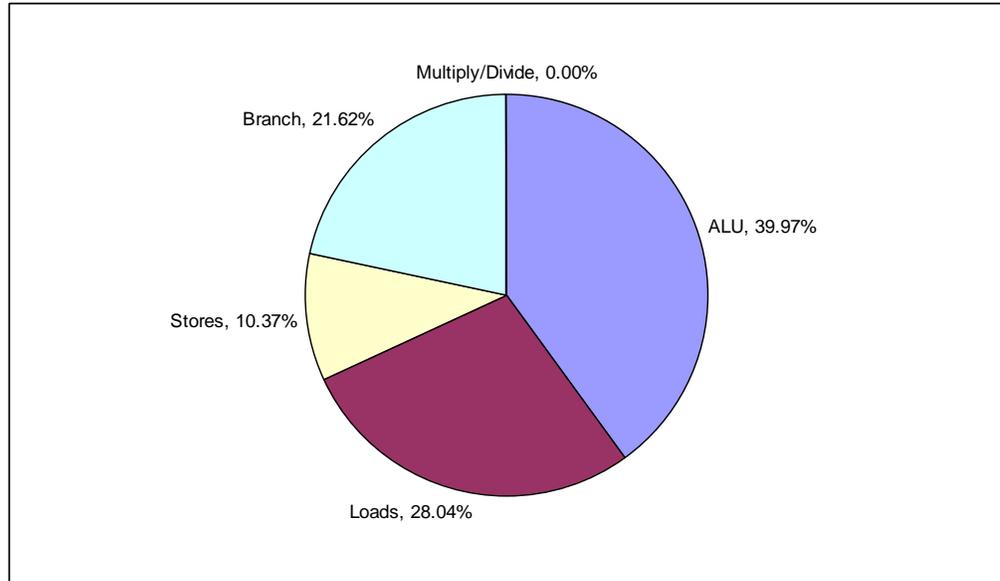
A debug mode is provided (in this case, the `#define BENCHMARK`) to activate the main timed loop, controlled by the test harness. If this is not defined, then the program goes into an interactive mode where each "rule" (Boolean expression) is entered by the user, parsed, and then printed out as a truth table

If the program is in debug mode (i.e., not in benchmark mode), then the program evaluates the expression for all possible values of the variable list. This is done with a recursive function to set the variables, and then by evaluating the expression stored in the binary tree.

1,000 iterations are the default, and 1,000 for CRC verification runs.

Analysis of Computing Resources

This benchmark exercises the byte manipulation, pointer comparison, indirect reference handling and stack manipulation capabilities of a processor.



The instruction mix is shown in the pie chart. The percentages may vary across architectures. The C library functions `strcmp()` and `strncpy()` are used extensively by this benchmark, and a well-designed and optimized C library would improve performance. Unlike other EEMBC benchmarks, dynamic memory allocations via `malloc()` are avoided. No floating-point calculations are used. The code size and the data size are moderate.



An Industry-Standard Benchmark Consortium

Optimizations Allowed	Out of the Box / Standard C Full Fury / Optimized
----------------------------------	--

- The C code must not be changed for Out-of-the-Box unless it must be modified to get it to compile. All changes must be documented, authorized by the certification authority, and must not have a performance impact.
- For Out-of-the-Box, additional hardware can be used if it does not require code changes.
- All optimized libraries must be part of the standard compiler package, and/or available to all customers
- The EEMBC Test Harness Lite must be used. Test harness changes may be made for portability reasons if they do not impact performance.
- For Optimized, the basic algorithm may not be changed, but the code may be rewritten in assembler. Re-writing the code to take advantage of parallelism is allowed so long as the correct answers are achieved using any arbitrary keys (not just those supplied in the benchmark code).
- For Optimized, optimized libraries can be used if they are publicly available.
- For Optimized, in lining is allowed.
- Additional data files may be used during certification to ensure the correctness of the optimized benchmark. You should **not** assume data patterns during optimization.
- Profile directed optimization is allowed using training data set 1, ruledata1.txt.