



An Industry-Standard Benchmark Consortium

## 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



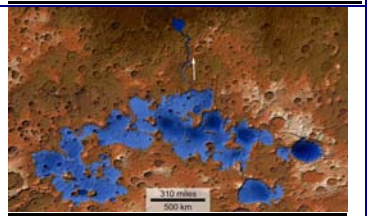

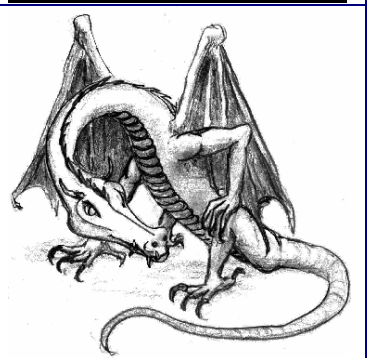
The bitmap rotation algorithm is primarily aimed at testing the bit manipulation, comparison, and indirect reference capabilities of the microprocessor. The algorithm uses a series of indirect references and bit masks to check and set individual bits in a data buffer representing a binary image. The 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.

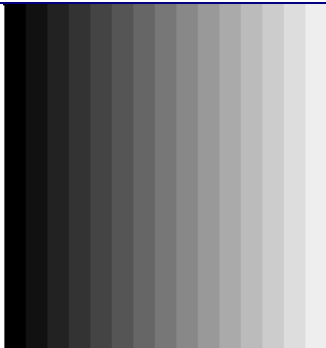

There are multiple input data buffers available to debug the benchmark. The input buffer is included in the benchmark is statically initialized data and the output buffer is created by calling the test harness memory allocation routine, `th_malloc()`. After the timed iterations have been completed, the 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.	

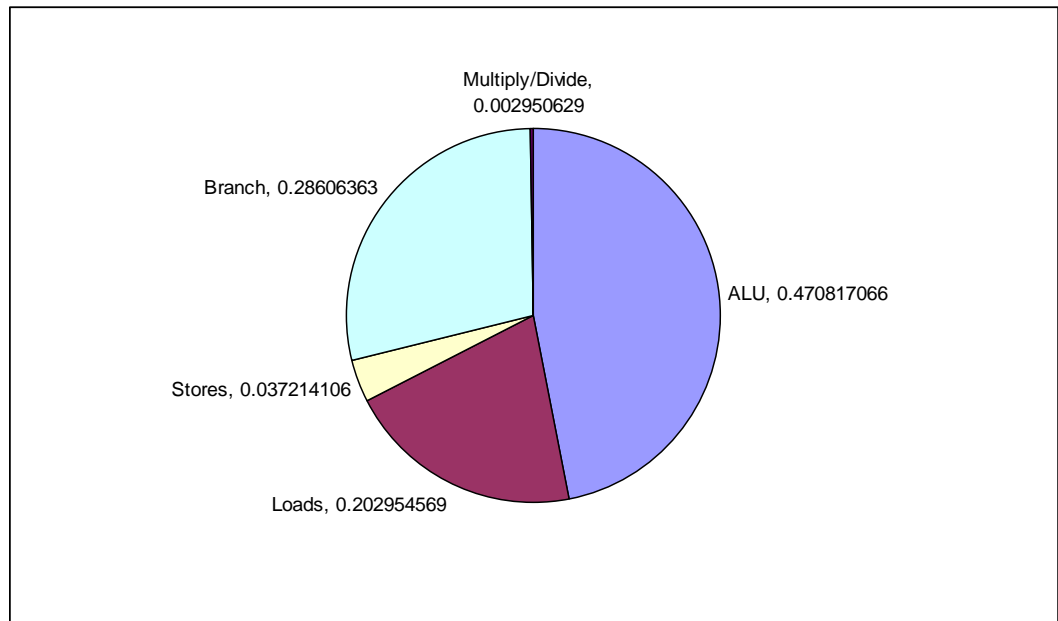
Data 10	Gradient	A grayscale gradient shading test pattern. 256 x 256, 256 colors.	
Data 11	Medium	A long, thin, black and white picture having 37 x 345 pixels, 256 colors, and 255 unique colors.	

***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.