
# EEMBC Symmetric-Multicore Benchmark User Guide

Version 2.1.4

Contact: peter.torelli@eembc.org

Table of Contents



# 1 Introduction

MultiBench™ is a comprehensive suite of benchmarks for evaluating the performance of scalable symmetric multiprocessor architectures employed within embedded multicore platforms. MultiBench allows you to test:

- Scalability where contexts can exceed available resources
- Single core versus multiprocessor/multicore
- Memory bandwidth
- OS scheduling support
- Compiler benchmarking

The key element of MultiBench is the unique Multi-Instance Test Harness (MITH) that provides both a framework for coordinating scalability analysis, as well as an abstraction layer to facilitate porting to different platforms.

MultiBench is comprised of a wide-variety of application-focused workloads from several industries such as automotive, networking and office automation. These workloads can also be used for general-purpose analysis.

While *MultiBench* is the name given to the original scalability benchmark, EEMBC also provides other products built on the same framework for: floating-point scalability, called FPMark™; automotive-centric analysis, called AutoBench™ 2.0; and CoreMark®-Pro, the successor to the popular CoreMark® benchmark which encompasses workloads most representative of consumer-centric experiences.

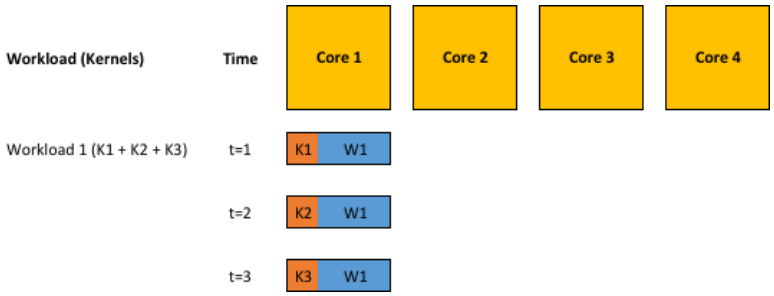
 

## 2 Theory of Operation

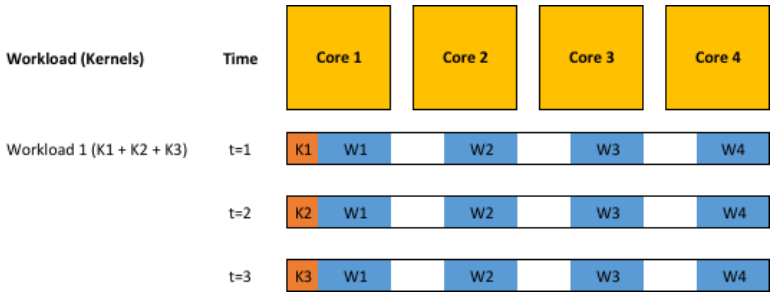
As a starting point in this benchmark development, EEMBC members provided the real-world, industry code kernels such as angle-to-time computation, finite impulse response filtering, IP packet checking, image rotation, and video encoding. MITH provides a convenient method to analyze compute scalability through a platform framework that can link together these individual kernels into workloads with varying degrees of complexity. In turn, the benchmark's workloads represent common occurrences of these kernels from real-world applications. For example, a simple workload may contain a single kernel such as *ippktcheck-4Mw1*. On the other hand, a complex workload may contain 5 kernels, such as *4M-check-reassembly-tcp-cmykw2-rotatew2*.

For portability, MITH provides an abstraction layer of interface functions for porting the test harness to most any hardware (homogeneous multicore or single core).

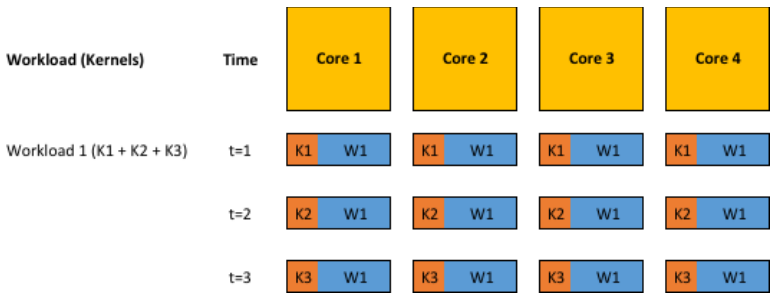
1. One context and one worker per context, `XCMD='-c1 -w1'`:




2. One context and four workers per context, `XCMD='-c1 -w4'`:

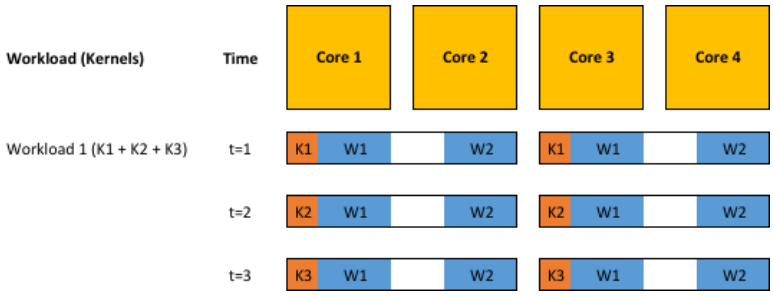



3. Four contexts and one worker per context, `XCMD='-c4 -w1'`:




4. Two contexts and two workers per context, `XCMD='-c2 -w2'`:




## 2.1 Performance Metrics

All MITH benchmark suites utilize two performance metrics for comparing configurations and platforms:

### 2.1.1 Throughput

Defined in iterations per second, each platform executes a workload a specific number of times per second. The user adjusts the number of contexts to find the optimal throughput for each workload for the given hardware.

### 2.1.2 Performance Scaling

This unit-less value defines how well performance scales with more computing resources assigned to the workloads. This is done by comparing a workload running with a single worker to that same workload with multiple workers. The user adjusts the number of workers to find the optimal scaling for each workload.

The MultiBench and FPMark suites combine the throughput of each workload into a single score (a Mark) which is reported in the output file . By default, the suite runs all of the workloads once using a single resource, and then again with the user-defined best configuration. The performance scaling of a platform is reported as the ratio of the Marks for these two runs. More information on this mark is given in the Certification Marks for MultiBench section.

# 3 Installing and Running

Each suite is named by the following convention `suite_x.x.x.tgz`, where *suite* can be *multibench*, *fpmark*, *coremarkpro*, or *autobench*.

1. Unpack the suite archive. It will create a new folder with the following contents:

```
Makefile
Makefile.mak
benchmarks/      ; Benchmark kernels
docs/            ; Release information
mith/            ; MITH framework
util/            ; Support tools
workloads/       ; Workloads & workload datasheets
```

2. Build the suite. Currently the build system supports the Linux *gcc* 32- and 64-bit environments. While there are many toolchains in the util/make area, not all are up-to-date and are provided as a reference. This build step creates all of the executables and runs them once to verify the workload runs against reference data.

```
% make TARGET=<target>
```

3. Build and run the certification to obtain official EEMBC results for all the workloads:

```
% make TARGET=<target> certify-all
```

   See the "Benchmark Technical Overview" section for details of what occurs in this step.

After the last step completes, the results are stored under the build path:

```
builds/<target>/<toolchain>/
```

The workloads' executables are stored under the build path in the bin area:

```
bin/<workload>.exe
```

Note that some workloads request input data in a hardcoded path (*../data*) so they can only be run from within the *bin* directory.

Some workloads require input data files for verification. For example, the FPMark workloads that contain floating-point functions use a set of golden results in order to compare SNR accuracy of the target systems' FP fidelity. The same is true for the video codec libraries found in MultiBench. The comparison data can be found under:

```
bin/data*/
```

Each official run creates a time-stamped directory containing a log file with all results. FPMark and MultiBench create a "Mark" score, which will be located in the same path as the log file, but with the suffix *mark*:

```
cert/<date>/<target>.<toolchain>.log  ; results log
cert/<date>/<target>.<toolchain>.mark ; summary & mark
```

# 4 Benchmark Technical Overview

## 4.1 Certification Marks for MultiBench

MultiBench consists of three separate marks: *MultiMark*, *ParallelMark*, and *MixMark*. Each of these marks are comprised of approximately two dozen workloads. The MultiBench Expansion Pack contains an additional 96 workloads that are not associated with any mark, but provide additional analysis capabilities. The workloads in the MultiBench Expansion Pack are executed automatically when the user runs the certification run.

### 4.1.1 MultiMark

MultiMark consolidates the best throughput using workloads with only one work item, each of which uses only one worker. The calculated throughput factor is 10 times the geometric mean of the iterations per second achieved with the best configuration for each workload (Note: 10 is a multiplication factor). Each work item uses only one worker, `-w1`, and multiple copies of the task can be performed in parallel to take advantage of concurrent hardware resources, `-cN`. All workloads in this mark use a 4MB dataset.

| MultiMark | -w | -c | Description |
|---|---|---|---|
| iDCT-4Mw1 | ✗ | ✓ | Inverse DCT |
| ippktcheck-4Mw1 | ✗ | ✓ | IP Packet Header Check |
| ipres-4Mw1 | ✗ | ✓ | IP Reassembly |
| md5-4Mw1 | ✗ | ✓ | MD5 Checksum |
| rgbcmyk-4Mw1 | ✗ | ✓ | RGB to CMYK Conversion |
| rotate-4Ms1w1 | ✗ | ✓ | Image rotation with one slice |
| rotate-4Ms64w1 | ✗ | ✓ | Image rotation with 64 slices |
| x264-4Mqw1 | ✗ | ✓ | Encode YUV data to h.264 |

*Figure 1. MultiMark workloads and concurrency constraints for official run (certification). Only the number of contexts (-c) may be changed.*

To measure scaling with MultiMark, specify the number of contexts, *N*:

```
% make TARGET=<target> certify-all XCMD='-cN'
```

The scoring equation is:

MultiMark = 10 x geomean(geomean(rotate1, rotate64), remaining scores)

### 4.1.2 ParallelMark

This mark consolidates the best throughput of workloads with only one work item that each use multiple workers. The calculated throughput factor is 10 times the geometric mean of the iterations per second achieved with the best configuration for each workload (Note: 10 is a multiplication factor). Only one work item may be executed at a time, and multiple workers may be used to take advantage of concurrent hardware resources, `-wN`. All workloads in this mark use a 4MB dataset.

| ParallelMark | -w | -c | Description |
|---|---|---|---|
| iDCT-4M | ✓ | ✗ | Inverse DCT |
| ippktcheck-4M | ✓ | ✗ | IP Packet Header Check |
| ipres-4M | ✓ | ✗ | IP Reassembly |
| md5-4M | ✓ | ✗ | MD5 Checksum |
| rgbcmyk-4M | ✓ | ✗ | RGB to CMYK Conversion |
| rotate-4Ms1 | ✓ | ✗ | Image rotation with one slice |
| rotate-4Ms64 | ✓ | ✗ | Image rotation with 64 slices |
| x264-4Mq | ✓ | ✗ | Encode YUV data to h.264 |

*Figure 2. ParallelMark workloads and concurrency constraints for official run (certification). Only the number of workers (-w) may be changed.*

NOTE: Just to clarify, the context flag, "-c" is valid for all workloads because all workloads can have multiple contexts instantiated, whereas not all workloads support multiple workers ("-w"). However, for ParallelMark, the "-c" **must not be used** as part of the scoring. It is only a worker parallelism benchmark.

To measure scaling with ParallelMark, specify the number of workers, *N*:

```
% make TARGET=<target> certify-all XCMD='-wN'
```

The scoring equation is:

ParallelMark = 10 x geomean(geomean(rotate1, rotate64), remaining scores)

#### 4.1.3 MixMark

MixMark is perhaps the most telling mark, it consolidates the best throughput of workloads with multiple different work items. These workloads are closest to workloads run on actual systems. The calculated throughput factor is 10 times the geometric mean of the iterations per second achieved with the best configuration for each workload (Note: 10 is a multiplication factor). All workloads in this mark use a 4MB dataset.

The number of simultaneous work items, `-cM`, and the number of workers per work item, `-wN`, may be modified for this mark.

| MixMark | -w | -c | Description |
|---|---|---|---|
| 4M-check | ✓ | ✓ | IP Packet Header Check |
| 4M-tcp-mixed | ✓ | ✓ | TCP stack activity |
| 4M-cmykw2 | ✓ | ✓ | RGB to CMYK Conversion |
| 4M-reassembly | ✓ | ✓ | IP Reassembly |
| 4M-rotatew2 | ✓ | ✓ | Image Rotation |
| 4M-x264w2 | ✓ | ✓ | Encode YUV data to h.264 |
| 4M-check-reassembly | ✓ | ✓ | Combination of above |
| 4M-check-reassembly-tcp | ✓ | ✓ | Combination of above |
| 4M-check-reassembly-tcp-cmykw2-rotatew2 | ✓ | ✓ | Combination of above |
| 4M-check-reassembly-tcp-x264w2 | ✓ | ✓ | Combination of above |
| 4M-cmykw2-rotatew2 | ✓ | ✓ | Combination of above |

*Figure 3. MixMark workloads and concurrency constraints for official run (certification). Both the number of contexts (-c) and number of workers (-w) may be changed.*

To measure scaling with ParallelMark, specify the number of workers, *N*, and contexts, *M*:

```
% make TARGET=<target> certify-all XCMD='-wN -cM'
```

The scoring equation is:

MultiMark = 10 x geomean(all scores)

### 4.2 Certification Marks for FPMark

FPMark consists of 8 scoring marks comprised of 55 workloads built from 10 benchmark kernels. Each mark represents the geometric mean of a subset of workloads' performance. The performance, or throughput, of each workload is measured in iterations per second.

#### 4.2.1 FPMark Benchmark Kernels

The following table describes the ten benchmark kernels that form the foundation of the suite. More details can be found in the *datasheet.txt* file located in the benchmark's source directory *benchmarks/fp*.

| Benchmark | Description |
|---|---|
| atan | Calculate atan(x) using a telescoping series. |
| blacks | Black-Scholes simulation. |
| FC_xp1px | Calculates the first *n* fourier coefficients of the function (x+1)^x. |
| fft_radix2 | FFT transform using radix-2 for dataset sizes of powers of 2. |
| horner | Polynomial evaluation using Horner method. |
| linpack | Gaussian elimination with partial pivoting based on LINPACK. |
| loops | A variety of kernels based on the Livermore loops benchmark. |
| lu | Matrix LU decomposition and dot product. |
| nnet | Neural net simulation. |
| ray | Simple ray tracing on a sphere. |

*Figure 4. FPMark suite benchmark kernels.*

#### 4.2.2 FPMark Workloads

The ten kernels are assembled into 53 workloads. These workloads are grouped together into high-level and low-level marks. Low-level marks are grouped according to precision (double or single) and dataset size (small, medium, and large). The following table describes the six low-level marks created and their corresponding workloads.

| FPv1.0 DP Small Dataset | FPv1.1 DP Medium Dataset | FPv1.3 DP Large Dataset |
|---|---|---|
| atan-1k | atan-64k | atan-1M |
| blacks-sml-n500v20 | blacks-mid-n1000v40 | blacks-big-n5000v200 |
| horner-sml-1k | horner-mid-10k | horner-big-100k |
| inner-product-sml-1k | inner-product-mid-10k | inner-product-big-100k |
| linear_alg-sml-50x50 | linear_alg-mid-100x100 | linear_alg-big-1000x1000 |
| loops-all-tiny | loops-all-mid-10k | loops-all-big-100k |
| lu-sml-20x2_50 | lu-mid-200x2_50 | lu-big-2000x2_50 |
| nnet_data1 | nnet_test | |
| radix2-sml-2k | radix2-mid-8k | radix2-big-64k |
| ray-64x48at4s | ray-320x240at8s | ray-1024x768at24s |
| xp1px-sml-c100n20 | xp1px-mid-c1000n200 | xp1px-big-c10000n2000 |

*Figure 5. FPMark Double-Precision Mark workload table. Note, there is no large dataset for the nnet kernel.*

| FPv1.4 SP Small Dataset | FPv1.5 SP Medium Dataset | FPv1.6 SP Large Dataset |
|---|---|---|
| atan-1k-sp | atan-64k-sp | atan-1M-sp |
| blacks-sml-n500v20-sp | blacks-mid-n1000v40-sp | blacks-big-n5000v200-sp |
| horner-sml-1k-sp | horner-mid-10k-sp | horner-big-100k-sp |
| inner-product-sml-1k-sp | inner-product-mid-10k-sp | inner-product-big-100k-sp |
| linear_alg-sml-50x50-sp | linear_alg-mid-100x100-sp | linear_alg-big-1000x1000-sp |
| loops-all-tiny-sp | loops-all-mid-10k-sp | loops-all-big-100k-sp |
| lu-sml-20x2_50-sp | lu-mid-200x2_50-sp | lu-big-2000x2_50-sp |
| nnet-data1-sp | nnet_test-sp | |

*Figure 6. FPMark Single-Precision Mark workload table. Note, there is no large dataset for the nnet kernel, and radix, raytracing, and xp1px do not have a Single-Precision mode.*

The two high-level marks, *FPMark* and *MicroMark*, represent the official EEMBC-endorsed scores for the suite.

*FPMark* is the official EEMBC-endorsed mark for the FPMark suite. It is calculated as the geometric mean of all of the workloads and multiplying by 100. Any device that cannot yield all of the individual scores (e.g., insufficient memory or computation resource) will be unable to obtain an FPMark.

*MicroMark* is the official mark for low-end microcontrollers with limited resources. It is calculated as the geometric mean of the small-dataset, single-precision workloads.

### 4.3 Certification Marks for AutoBench

There are no official Marks for AutoBench. For simple evaluation, the user is encouraged to create a Mark utilizing similar methods applied for MultiBench or FPMark. Otherwise AutoBench can be used as a tool for evaluating the scalability of each of its workloads.

### 4.3.1 AutoBench Benchmark Kernels

The individual benchmark kernels used to derive the workloads are explained in the document found here: http://eembc.org/techlit/datasheets/autobench_db.pdf.

| Kernel | Description |
|---|---|
| a2time01 | Angle to time conversion |
| aifirf01 | Finite impulse response filter |
| bitmnp01 | Bit manipulation |
| canrdr01 | CAN remote data request |
| idctrn01 | Inverse discrete cosine transform |
| iirflt01 | Infinite impulse response filter |
| matrix01 | Matrix arithmetic |
| pntrch01 | Pointer chasing |
| puwmod01 | Pulse width modulation |
| rspeed01 | Road speed calculation |
| tblook01 | Table lookup and interpolation |
| ttsprk01 | Tooth to spark |

*Figure 7. AutoBench 2.0 benchmark kernels*

### 4.3.2 AutoBench Workloads

All of the workloads in this suite contain a between two and four kernels that execute sequentially, for example: in *matrix-tblook* the matrix kernel runs followed by the table lookup kernel. The workloads are provided with two dataset sizes, 4K and 4M:

| Workload |
|---|
| bitmnp-rspeed-puwmod-4K[-4M] |
| matrix-tblook-4K[-4M] |
| puwmod-rspeed-4K[-4M] |
| rspeed-idctrn-canrdr-4K[-4M] |
| rspeed-idctrn-iirflt-4K[-4M] |
| ttsprk-a2time-matrix-4K[-4M] |
| ttsprk-a2time-pntrch-4K[-4M] |
| ttsprk-a2time-pntrch-aifirf-4K[-4M] |
| ttsprk-a2time-pntrch-idctrn-4K[-4M] |
| ttsprk-a2time-pntrch-tblook-4K[-4M] |

*Figure 8. AutoBench 2.0 workloads are combinations of the AutoBench 2.0 kernels with 4kB and 4MB data-sets.*

### 4.4 Certification Marks for CoreMark-Pro

CoreMark-Pro uses a combination of integer and floating-point kernels with a variety of data sizes, as well as the CoreMark benchmark. The number of workers cannot be adjusted for the benchmark, only the number of contexts using `-cN`.

| Benchmark | Description |
|---|---|
| cjpeg-rose7-preset | JPEG compression using 7 workers. |
| core | CoreMark. |
| linear_alg-mid-100x100-sp | Gaussian elimination with partial pivoting on a 100x100 single-precision matrix dataset; single worker. |
| loops-all-mid-10k-sp | Livermore Loops kernel with a 100x100 single-precision matrix; single worker |
| nnet_test | Neural net simulation; single worker |
| parser-125k | XML parser with a 125k dataset; single worker. |
| radix2-big-64k | FFT base-2 transform with 64k dataset; single worker |
| sha-test | SHA256 benchmark |
| zip-test | ZIP compression benchmark |

*Figure 9. CoreMark-Pro workloads.*

Unlike the other benchmarks, the score for CoreMark-Pro is a geometric mean of normalized component scores based on an initial reference platform. The score is automatically computed by the same PERL script referenced earlier, but for clarity, the table below lists the scale factor and reference score (denominator). To compute the mark, divide each measured component score by its reference score, multiply by the scale factor, and take the geometric mean of those numbers, the multiply by 1000.

| Workload | Scale Factor | Reference Score |
|---|---|---|
| cjpeg | 1 | 40.3438 |
| linear | 1 | 38.5624 |
| loops | 1 | 0.87959 |
| nnet | 1 | 1.45853 |
| parser | 1 | 4.81116 |
| radix | 1 | 99.6587 |
| sha | 1 | 48.5201 |
| zip | 1 | 21.3618 |
| core | 10000 | 2855 |

*Figure 10. Scale factor and reference scores used to compute the CoreMark-Pro mark.*

# 5 Results Area

After the certification run completes, the results from each pass reside in the build area cert directory:

```
builds/<TARGET>/<TOOLCHAIN>/cert/<TIMESTAMP>/
       <TARGET>.<TOOLCHAIN>.log
       best/<TARGET>.<TOOLCHAIN>.log
       single/<TARGET>.<TOOLCHAIN>.log
```

Each pass consists of a single verification run, `-v1`, followed by 3 performance runs, `-v0`. The top-level log is the concatenation of the best and single logs, which in turn are concatenations of each individual executable run. Each line in the aggregate log files contains the results of one invocation of the executable, which is just the transpose of this workload's standard output. Here is an example of running a single workload directly (not using make):

```
% cd builds/<TARGET>/<TOOLCHAIN>/bin
% ./iDCT-4M.exe -v0 -c4 -w1
-  Info: Starting Run...
-- Workload:iDCT-4M=1511685754
-- iDCT-4M:time(ns)=600
-- iDCT-4M:contexts=4
-- iDCT-4M:workers=1
-- iDCT-4M:iterations=100
-- iDCT-4M:time(secs)=     0.6
-- iDCT-4M:secs/workload=   0.006
-- iDCT-4M:workloads/sec= 166.667
-- Done:iDCT-4M=1511685754
```

The first option turns off verification mode, the next sets the number of contexts and workers.

Throughput

| # Workers (-w) | # of Contexts (-c) 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| 1 | 56 | 57 | 167 | 225 |
| 2 | 94 | 190 | 209 | 211 |
| 4 | 169 | 196 | 213 | 208 |
| 8 | 181 | 201 | 198 | 189 |

Scaling

| # Workers (-w) | # of Contexts (-c) 1 | 2 | 4 | 8 |
|---|---|---|---|---|
| 1 | 1.0 | 1.0 | 3.0 | 4.0 |
| 2 | 1.7 | 3.4 | 3.8 | 3.8 |
| 4 | 3.0 | 3.5 | 3.8 | 3.7 |
| 8 | 3.3 | 3.6 | 3.5 | 3.4 |

*Figure 11. Example results for an 8-core machine running the iDCT-4M workload.*

These same results could have been obtained by passing `XCMD='-c4 -w4'` to the make command. Make would run the single context reference run and then write the results to the path named *single*, and then would have run the same test but using the supplied XCMD flags, and written those results under *best*.

The aggregate log-files contain a large amount of data and suited more toward spreadsheet analysis. To view a higher-level summary of an aggregate log file, use the provided *util/perl/generate_ summary.pl* script like this:

```
% perl util/perl/generate_summary.pl builds/linux64/gcc64/cert/<date>/*.log
                                     MultiCore SingleCore
Workload Name                         (iter/s)   (iter/s)    Scaling
------------------------------------ ---------- ---------- ----------
4M-check                               1733.70    3012.05       0.58
4M-check-reassembly                     588.24     354.61       1.66
4M-check-reassembly-tcp                 219.30     118.48       1.85
4M-check-reassembly-tcp-cmykw2-rotatew2   74.72      43.80       1.71
4M-check-reassembly-tcp-x264w2             7.70       2.31       3.33
4M-cmykw2                               904.98     261.44       3.46
4M-cmykw2-rotatew2                      118.11      69.28       1.70
4M-reassembly                          1408.45     404.86       3.48
4M-rotatew2                             142.45      94.70       1.50
:
:
```

In this example, the settings used for the best run were different than that of single, and thus the resulting scaling is greater than one, as expected, since this example was run on an 8-core machine with `XCMD='-w2 -c2'`. Note the poor scaling of *4M-check* (0.58x). If we run it by itself with only `-c2` we see a score of ~5924, or scaling of 1.96x. This is one example of how simply increasing the number of workers and contexts doesn't necessarily achieve the best performance, and that the user must experiment to see which combination yields the best results.

## 6 Comparing Mark Results

For suites that generate a mark score, the results are automatically computed and reported in the mark file.

Consider the following MultiBench mark file:

```
% cd builds/<TARGET>/<TOOLCHAIN>/cert/* area>
% cat <TARGET>.<TOOLCHAIN>.mark
:
:
MARK RESULTS TABLE

Mark Name                                    MultiCore SingleCore    Scaling
-------------------------------------------- ---------- ---------- ----------
MultiBench, 1. MultiMark                        4023.32    1200.53       3.35
MultiBench, 2. ParallelMark                     4092.06    1200.29       3.41
MultiBench, 3. MixMark                          2687.99     916.62       2.93
```

These sample marks were collected on a four-core system with the number of contexts set to four. The mark scores are displayed for both the single-core (default) and multi-core runs (in this case, `XCMD='-c4'`). We can see that the scaling approaches the theoretical max of 4.00 for Multi- and ParallelMark, but is 25% lower for MixMark.

## 7 Verification

All workloads either create random data of a specific size or use a pre-determined dataset. In both cases, the output is verified against the input and tests can fail if the results don't match. In the case of floating-point tests, MITH compares the results using a signal-to-noise ratio threshold to account for real-world hardware implementation differences.

## 8 Basic Makefile Options

All MultiBench suites exposes several parameters from the Makefile command-line through the `XCMD='<flags>'` parameter.

1. The number of contexts can be changed with this flag:

   Context flag: `-cN`

2. The number of workers, *N*, can be changed with this flag:

   Number of Concurrent Workers flag: `-wM`

3. The number of iterations for all workloads can be changed with this flag. By default, each workload has its own number of iterations, originally decided by the EEMBC workgroup that developed the benchmark specification. The log file reports the default number of iterations for each workload.

   Iteration flag: `-iN`

4. A value of 0 indicates performance mode, a value of 1 indicates verification mode, which invokes result checking.

   Verification flag: `-v[0|1]`

Other make targets exist, but they are all partial targets of the primary rule *certify-all*, but for the sake of simplicity, it is easier to simply invoke target *certify-all* for each experiment.

# Appendix A System Compatibility

The following platforms, hardware, and toolchains were tested:

| Target | OS Name | OS Version | Toolchain | Compiler Version | CPU | CPU ID | GHz | # logical cores | Container | OK? |
|---|---|---|---|---|---|---|---|---|---|---|
| linux | Ubuntu | 14.04.3 LTS | gcc | 4.8.4-2ubuntu1 | Intel | Xeon E5-2650 | 2.0 | 1 | AWS/i-097594d1 | ✓ |
| linux64 | RHEL | 6.7 Santiago | gcc64 | 4.4.7 2012013 | Intel | Xeon E5-2670 v2 | 2.5 | 2 | AWS/i-51588689 | ✓ |
| linux64 | Ubuntu | 14.04.2 LTS | gcc64 | 4.8.4-2ubuntu1 | Intel | Xeon E5-2676 | 2.4 | 1 | AWS/i-5a9fd99f | ✓ |
| linux64 | Ubuntu | 14.04.3 LTS | gcc64 | 4.8.4-2ubuntu1 | Intel | Xeon E5-2676 v3 | 2.4 | 8 | AWS/i-bf5bc265 | ✓ |
| linux64 | Ubuntu | 14.04 LTS | gcc64 | 4.8.2-19ubuntu1 | ARM | *ThunderX (?)* | 2.5 | 96 | none | ✓ |
| linux64 | Ubuntu | 14.04.1 LTS | gcc64 | 4.8.2-19ubuntu1 | Intel | Core i7-4750HQ | 2.0 | | VMW/OSX | ✓ |
| linux64 | CYGWIN_NT-6.1-WOW | 2.0.4(0.287/5/3) | gcc64 | 4.9.2 | Intel | Core i7-4750HQ | 2.0 | | VMW/OSX | ✓ |
| linux64 | CYGWIN_NT-10.0 | 2.4.1(0.293/5/3) | gcc64 | 4.9.3 | Intel | Core i7-6500U | 2.5 | 4 | WIN10 | ✗ |
| linux64 | CentOS | 5.11 (Final) | gcc64 | 4.1.2 20080704 | Intel | Xeon E5-2676 v3 | 2.4 | 1 | AWS/i-9d27025a | ✓ |
| linux64 | Scientific Linux | 6.4 Carbon | gcc64 | 4.4.7 20120313 | Intel | Xeon E5-2670 v2 | 2.5 | 1 | AWS/i-6d523ab5 | ✓ |

MultiBench requires at least 1GB of RAM; the MultiBench Expansion Pack requires a minimum of 8GB.

## Appendix B Porting

To simplify porting and portability, MITH was written for Linux-based operating systems using GNU-like tool chains. However, it was implemented with an abstraction layer and test harness to facilitate porting to different platforms.

The abstraction layer provides a method to implement thread scheduling, signaling, and affinity. By default, the threading is implemented with POSIX *pthread*, and affinity is deferred to the O/S. The following 12 functions located in the file *mith/al/src/al_smp.c* implement the default MITH threading model:

```
al_mutex_init
al_mutex_lock
al_mutex_trylock
al_mutex_unlock
al_mutex_destroy
al_cond_init
al_cond_signal
al_cond_broadcast
al_cond_wait
al_cond_destroy
al_thread_create
al_join
```

The abstraction layer also contains hooks for implementing processor affinity. Since affinity implementations may vary significantly across hardware and development tools, the effort is largely left to the developer. Please contact EEMBC for paid-support on implementing affinity.

MITH uses a test harness which essentially implements some common standard-library functions and console output. Test harness functions begin with *th_*. For example, the *th_printf* function calls the standard library *printf* for console output and on systems without a console, the user may need to implement *th_printf* as a character output to a UART or any other debug port. The test harness functions are located in *mith/src/th_lib.c*.

In addition to changes to the abstraction layer and test harness, the toolchain may also need modification. The most common toolchain is gcc64, and the make file defining it is located in util/make/gcc64.mak. The Intel® C++ Compiler (*icc*) is also supported through *TOOLCHAIN=icc*. Porting to compilers that support similar switches, e.g., ARM-based cross compilers, is fairly straightforward. For non-GCC compilers this may be a significant effort.

# Appendix C Errata

### Known Warnings

EEMBC mandates the use of aggressive warning reporting using the *-Wall* switch. While significant effort has been spent to reduce the number of compiler warnings, different versions of compilers produce more pedantic warnings than others. EEMBC has tested a number of GCC compilers and not all of the warnings can be dismissed without extensive code modifications. Some of the more pervasive warnings different GCC compiler versions may encounter are listed below, and may be safely ignored.

### GCC 4.1.2

```
benchmarks/networking/tcp/tcp_core.c:440: warning: comparison is always true due to limited range
of data type
benchmarks/video/x264/encoder/analyse.c:29: warning: ignoring #pragma GCC diagnostic
benchmarks/video/x264/encoder/cabac.c:714: warning: ignoring #pragma GCC diagnostic
benchmarks/video/x264/encoder/cabac.c:715: warning: ignoring #pragma GCC diagnostic
benchmarks/video/x264/encoder/cabac.c:807: warning: ignoring #pragma GCC diagnostic
benchmarks/md5/md5test.c:212: warning: dereferencing type-punned pointer will break strict-
aliasing rules
```

### GCC 4.4.7

```
benchmarks/video/x264/common/macroblock_common.c:955: warning: array subscript is below array
bounds
benchmarks/video/x264/common/macroblock_common.c:956: warning: array subscript is below array
bounds
benchmarks/video/x264/encoder/cabac.c:714: warning: expected [error|warning|ignored] after
'#pragma GCC diagnostic'
benchmarks/video/x264/encoder/cabac.c:807: warning: expected [error|warning|ignored] after
'#pragma GCC diagnostic'
```

### Bug Fixes

MultiBench 1.1 fixes a bug from the original 1.0 release. In the x264 encoding algorithm *macroblock.c*, the function *x264_mb_encode_8x8_chroma* contains a bug. Currently none of the suites execute this code, but it may change in the future and the following correction has been implemented.

### Bug #MB001: x264 8x8 Megablock Chroma Encoding Coefficients Incorrect

```
-        for( i = 0; i < 4; i++ )
-            dct4x4[i][0][0] = dct2x2[0][i];

+        for(i=0; i<2; ++i) {
+            for(j=0; j<2; ++j) {
+                dct4x4[ci][0][0] = dct2x2[i][j];
+                ++ci;
+            }
+        }
```

### Sighting #MB002: Conflicting types for 'verify_output'

The variable "verify_output" has a conflicting type.

`benchmarks/video/mp2decode/mp2mith.c:18:21` declares it as `unsigned int` but `mith/include/th_lib.h:462:14` declares it as `e_u32`. If `EE_SIZEOF_LONG` is 4, then `e_u32` is defined as `unsigned long`. This is incorrect as `int` and `long` may have different meanings on 32-bit compilers.

Solution: redefine the variable in be type `e_u32`.

**Sighting #MB003: Not all extended workloads have been cleaned or scrutinized.**

Extended workloads are not fully debugged or supported, and are provided as reference tests. The following workloads generate parameter errors on 32-bit machines, which appears to be related to argument parsing:

```
64M-x264-1worker
64M-x264-2workers
64M-x264-4workers
64M-x264-8workers
x264-64M
x264-90M-1worker
x264-90M-2workers
x264-90M-4workers
x264-base
```

Errors may include "`x264: option '--quiet' is ambiguous`" or "`x264: unrecognized option '--ds=1'`". A potential workaround is to add the line:

```
        nextchar = NULL;
```

After line 163 (#define NONOPTION...) but this has not been thoroughly regressed.

**Sighting #MB003: Incompatible pointer type warnings**

There are a number of pointer-cast type warnings in the x264 library. As they are warnings, and given EEMBC's policy for changing time-critical kernels in released benchmarks (as well as various compiler warning settings) there is currently no plan to resolve them.

**Sighting #MB004: Warnings with x264 kernels used in MultiBench expansion**

Four of the x264 workloads in the MultiBench expansion pack are failing validation. They are under investigation.

## Appendix D MultiBench Expansion Pack

The MultiBench Expansion Pack adds 96 additional workloads of varying dataset size and workload complexity (not included with FPMark or AutoBench). To install the expansion pack, switch to the directory containing the MultiBench original installation, and unpack the archive there. This will add new benchmarks and workloads to the current work area. The command `make … run-all` will run all of the workloads, whereas `make … wrun-<workload_name>` will run just that workload. Since these workloads are not part of a benchmark, the benchmark will not generate a score table.

64M-check-reassembly
64M-check-reassembly-tcp
64M-cmykw2
64M-cmykw2-rotatew2
64M-rotatew2
64M-tcp-mixed
64M-x264-1worker
64M-x264-2workers
*64M-x264-4workers
*64M-x264-8workers
huffde-all
ippktcheck-64M
ippktcheck-64M-1Worker
ippktcheck-64M-2Worker
ippktcheck-8x4M-1Worker
ippktcheck-8x4M-4Worker
ipres-100M10worker
ipres-100M1worker
ipres-100M2worker
ipres-100M4worker
ipres-6M1worker
ipres-6M4worker
ipres-72M
ipres-72M1worker
ipres-72M2worker
md5-128M16worker
md5-128M1worker
md5-128M2worker
md5-128M4worker
md5-1M16worker
md5-1M1worker
md5-1M2worker
md5-1M4worker
md5-32M
md5-32M16worker
md5-32M1worker
md5-32M2worker
md5-32M4worker
mp2decode1
mp2decode2
mpeg2-90Mout-1worker
mpeg2-90Mout-2workers
mpeg2-90Mout-4workers
mpeg2-90Mout-8workers
mpeg2-base
rgbcmyk-12M2workers
rgbcmyk-5x12M
rgbcmyk-5x12M1workers
rgbcmyk-5x12M2workers
rgbcmyk-5x12M4workers
rgbcmyk-5x12M8workers
rotate-16x4Ms1
rotate-16x4Ms1w1
rotate-16x4Ms1w2
rotate-16x4Ms1w32
rotate-16x4Ms1w4
rotate-16x4Ms1w8
rotate-16x4Ms32
rotate-16x4Ms32w1
rotate-16x4Ms32w2
rotate-16x4Ms32w4
rotate-16x4Ms32w8
rotate-16x4Ms4w1
rotate-16x4Ms4w2
rotate-16x4Ms4w4
rotate-16x4Ms4w8
rotate-16x4Ms64
rotate-16x4Ms64w1
rotate-34k-180deg
rotate-34k-270deg
rotate-34k-90deg
rotate-34kX128w1
rotate-34kX16-90deg
rotate-34kX512-90deg
rotate-4M-180deg
rotate-4M-270deg
rotate-4M-90deg
rotate-4Ms32
rotate-4Ms32w1
rotate-4Ms4
rotate-4Ms4w1
rotate-520k-180deg
rotate-520k-270deg
rotate-520k-90deg
rotate-520kX16-90deg
rotate-color1Mp
rotate-color1Mpw1
rotate-color-4M-90deg
rotate-color-4M-90degw1
tcpbase
x264-4M
x264-4Mw1
x264-64M
x264-90M-1worker
*x264-90M-2workers
*x264-90M-4workers
x264-base

* Starred workloads have known bugs which are still under analysis.

## Appendix E Document Revision History

| Version | Date | Notes |
|---|---|---|
| 2.1.1 | 24-08-2018 | Merged in CoreMark-Pro slides; new template; added revision history; renamed from "MultiBench User's Guide". |
| 2.1.2 | 12-09-2019 | More errata, and clarifications on ParallelMark workloads parallelism. |
| 2.1.3 | 01-11-2019 | Errata relating to MultiBench expansion pack x264 workloads. Added two shortcut commands to Appendix D. |
| 2.1.4 | 2019-12-20 | CoreMark-PRO multiplier is 1000, not 100. |