www.eembc.org

MultiBench Algorithms and Workload Datasheets

Embedded Microprocessor Benchmark Consortium

www.eembc.org

TABLE OF CONTENTS

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Introduction

EEMBC is synonymous with industry-standard benchmarks. This consortium first started developing embedded processor benchmarks in 1997. These benchmarks serve as a tool for evaluating embedded processors, compilers, and systems, and help to predict performance in real-world applications.

Evolving with the times, EEMBC has produced a comprehensive suite of benchmarks to help you evaluate the performance of scalable SMP architectures employed within embedded multicore platforms. These benchmarks, called MultiBench, allow you to test:

- Scalability where contexts exceed resources
- Single core versus multiprocessor/multicore
- Memory and I/O bandwidth
- OS scheduling support
- Efficiency of synchronization
- Compiler benchmarking

The key element of MultiBench is the unique Multi-Instance Test Harness (MITH) that provides an abstraction layer to support portability of the benchmarks across an infinite number of platforms. MITH makes it easy to run on any platform, operating system, and tool-chain.

MultiBench is also comprised of a wide-variety of application-focused workloads encompassing the networking, consumer, and office automation domains. Regardless of the application-specific focus of these workloads, their flexibility makes them applicable to a whole host of other applications. And if that’s not enough flexibility, EEMBC’s MultiBench Architect allows you to compose custom workloads for testing specific system characteristics.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Workload Organization

The standard MultiBench distribution includes algorithms/kernels and workloads that primarily encompass three application areas, including networking, consumer, and office automation. The table below shows the general distribution of these workloads.

NOTE: *All three application areas must be used to obtain the MultiBench 'marks' described in the following section.*

	Networking	Consumer	Office Automation
IP Reassembly[1]	X		
Packet Check[1]	X		
TCP[1]	X		
Packet Check + IP Reassembly[1]	X		
Packet Check + IP Reassembly + TCP[1]	X		
JPEG (compression and decompression)		X	
MP3 player		X	
MPEG2 decoder		X	
Huffman decoder		X	
x.264[1]		X	
Filters for RGB to HPG and RGB to YIQ[1]		X	
Md5[1]		X	
Image rotation[1]		X	X
CMYK + rotate[1]			X
Color space conversion from RGB to CMYK[1]			X
Packet Check + IP Reassembly + TCP + x.264[1,2]	O	O	
Packet Check + IP Reassembly + TCP + CMYK +Rotate[1, 3]	O		O

1. Part of official release, all other workloads are beta versions and included at no extra charge.
2. Requires (and included with) purchase of both Networking and Consumer workloads.
3. Requires (and included with) purchase of Networking and Office Automation workloads.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

MultiBench 1.0e: The Official Release Version

EEMBC has created an official release version of MultiBench that will be supported for all certifications. When running these workloads, the user may change the number of work items running in parallel, as well as the affinity of each instance of the work item, to take advantage of available computing resources in the target platform. Datasets for the workloads contained in version 1.0e have been chosen to limit the working set size to 4Mbytes per context.

MultiBench 1.0e, which contains 18 specific workloads, automatically yields a consolidated score for a given platform, thereby generating several different ‘marks’ that can be used to readily analyze a platform’s multicore performance. In general, each mark is based on two figures of merit derived from each workload:

1. Best Throughput Factor- defined in iterations per second, each platform will execute a workload a specific number of times per second. The best configuration is platform dependent.
2. Performance Scaling Factor- this concept defines how well performance scales when more computing resources are brought to bear on the workload. There are several ways of utilizing computing resources in parallel, and MultiBench 1.0e workloads with the MultiBench framework test most of them (the exception being functional decomposition, which will be addressed in the next version of MultiBench). By limiting the available resources to only execute one work item at a time, and comparing the throughput to the best throughput for the workload, we gain an insight into how well the platform scales for that workload.

***NOTE:** All three application areas (Consumer, Networking, and OA) must be used to obtain the MultiBench ‘marks’.*

MultiMark

This mark 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 is using only one worker, and multiple copies of the task can be performed in parallel to take advantage of concurrent hardware resources.

The workloads for this mark are:

1. <rgbcmyk-4Mw1>
2. <ipres-4Mw1>
3. <ippktcheck-4Mw1>
4. <md5-4Mw1>
5. <rotate-4Ms1w1>
6. <rotate-4Ms64w1>
7. <x264-4Mqw1>
8. <iDCT-4Mw1>

```
MultiMark = 10*geomean(geomean(5,6),1,2,3,4,7,8)
```

***NOTE:** Several rotate workloads are included for the MultiMark, the final calculation takes a geomean of the rotate* workloads (to use it as one item for the mark).*

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 using the -w<N> flag.

The workloads for this mark in mini1 subset are:

1. <rgbcmyk-4M>
2. <ipres-4M>
3. <ippktcheck-4M>
4. <md5-4M>
5. <rotate-4Ms1>
6. <rotate-4Ms64>
7. <x264-4Mq>
8. <iDCT-4M>

```
ParallelMark = 10*geomean(geomean(5,6),1,2,3,4,7,8)
```

NOTE: *Several rotate workloads are included for the MultiMark, the final calculation takes a geomean of the rotate* workloads (to use it as one item for the mark).*

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 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).

The number of simultaneous work items (-c<N>) and the number of workers per work item (-<N>) may be modified for this mark.

The workloads for this mark are:

1. <4M-check-reassembly-tcp>
2. <4M-check-reassembly-tcp-cmykw2-rotatew2>
3. <4M-check-reassembly-tcp-x264w2>
4. <4M-check-reassembly>
5. <4M-cmykw2-rotatew2>
6. <4M-rotatew2>
7. <4M-cmykw2>
8. <4M-tcp-mixed>
9. <4M-x264w2>
10. <4M-reassembly>
11. <4M-check>

Scaling

Providing performance scaling information, we also define an associated Scale Factor. To calculate the Scale Factor, first calculate the geometric mean of the throughput for the workloads with only one work item at a time enabled. Then divide the performance mark by that number.

Sample Scores

The following table shows sample scores on an anonymous quad core platform:

	Performance	**Scale**
MultiMark	450.4	1.7
ParallelMark	422.3	0.8
MixMark	182.7	0.7

These sample scores on a platform with 4 cores show some interesting information even without diving into the performance details of specific workloads. For example, consider the fact that the MultiMark scaling factor is 1.7 - rather then a number closer to 4 which you might expect. This factor strongly hints that the system can only use about 1/2 of the computing resources available to it on any particular workload. This may be related to the memory bottlenecks, synchronization efficiency of the platform and operating system, or both.

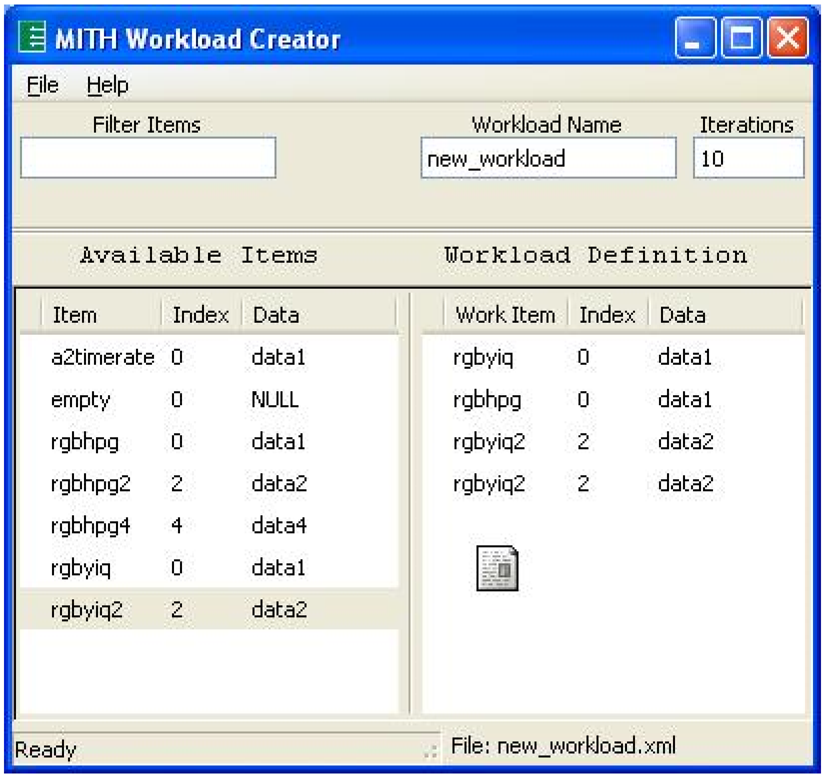
More detailed examination of individual results will yield more answers (some of the answers you can find in MultiBench presentations made at several conferences and available on the EEMBC website). ParallelMark and MixMark scale factors for this system strongly indicate that synchronization overhead is so significant as to erase most of the benefits of using multiple cores to solve a single problem. However, the throughput numbers for the platform indicate strong single core processing power for a machine in that category.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

MultiBench Architect

Although MultiBench comes with a wide variety of pre-defined and tested workloads, you may want to compose custom workloads for testing specific system characteristics. The MultiBench Architect allows you to drag and drop work items to create virtually unlimited new workloads.



Workload Name: new_workload

Iterations: 10

Item	Index	Data
a2timerate	0	data1
empty	0	NULL
rgbhpg	0	data1
rgbhpg2	2	data2
rgbhpg4	4	data4
rgbyiq	0	data1
rgbyiq2	2	data2

Work Item	Index	Data
rgbyiq	0	data1
rgbhpg	0	data1
rgbyiq2	2	data2
rgbyiq2	2	data2

Ready

File: new_workload.xml

Drag and drop available work items into the Workload Definition window to create custom workloads.

Creating a workload

- Drag items from the list to the Workload Definition window.
- Left click to edit item name/dataset/index.
- Use the filter field to search through available items.
- Rename the workload using the “Workload Name” field.
- Save the workload using File/Save or File/SaveAs.
- By default, workloads are saved in workloads/xml). Use workload_parser.pl to create a makefile and source file for the workload.

MultiBench Algorithms

The following pages describe the algorithms and kernels that form the basis for the MultiBench workloads.

Consumer: MD5
Consumer: H.264
General: iDCT
Office Automation: RGB to CMYK Conversion
Consumer and Office Automation: Image Rotation
Networking - IP Packet Check
Networking - IP Reassembly
Networking - Transmission Control Protocol (TCP)

NOTE: *Datasheets for some of the algorithms and kernels listed in the table in the section <Workload Organization> are not included in this MultiBench Databook. As noted above, the workloads derived from these algorithms and kernels are beta versions and included at no extra charge.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Consumer: MD5

Description

A benchmark kernel that calculates MD5 checksum (http://en.wikipedia.org/wiki/MD5) over multiple input buffers. This benchmark kernel is computationally intensive.

Concurrency

The benchmark initiates <W> worker threads to do md5 checksum on <N> buffers of size <B>.

<W>, <N> and <B> are all configurable via the dataset argument to <define_params_md5>.

The benchmark initiates a new worker thread for every buffer, with total number of active threads bounded by <W>.

Verification

Output verification is CRC on the output digest of all input buffers. CRC has been pre-calculated for all predefined input sets.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Consumer: H.264

Description

H.264 video encoding. This benchmark is computationally intensive.

Source

This is a port of the x264 open source coder developed by videolan (http://www.videolan.org/developers/x264.html).

Concurrency

N frames are coded concurrently in N threads.

Verification

Output verification is via a bit accurate comparison to pre-computed encoded output. Pre-computed output is included for multiple predefined input streams and exists under benchmarks/video/x264data

General: iDCT

Inverse Discrete Cosine Transform

Benchmark Description

This EEMBC benchmark simulates an embedded automotive/industrial application performing digital video and graphics applications such as image recognition. The kernel performs an inverse discrete cosine transform (iDCT) on an input data matrix set using 64-bit integer arithmetic.

NOTE: *This benchmark has been adapted from the v1.1 automotive suite with the following changes:*

1. *Removed code duplication.*
2. *Reading data once on initialization from a file.*
3. *An iteration is a pass on the full input dataset, rather than processing of a single value.*
4. *New datasets.*
5. *Enable application of multiple workers to speed up processing for parallel execution.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Office Automation: RGB to CMYK Conversion

Description

Benchmarks digital image processing performance in printers and other digital imaging products

- Explores basic arithmetic and minimum value detection capability
- Provides opportunities for Full Fury benchmark optimization
- Conditional move and multi-byte processing, exercising SIMD and VLIW architectures
- Integer implementation

Application

RGB to CMYK conversion is widely used in color printers. RGB inputs from PC data are converted to CMYK color signals for printing.

Benchmark Details

This benchmark explores the target CPU's ability to perform basic arithmetic and minimum value detection. The R, G, B 8-bit pixel color image input is fed to the following equation: /* calculate complementary colors **/ c = 255 – R; m = 255 – G; y = 255 – B; /** find the black level k **/ K = minimum (c,m,y) /** correct complementary color lever based on k */ C = c – K M = m – K Y = y - K RGB values are in the range of [0:255] CMYK values are in the range of [0:255] The input and output data sizes vary. For example, the 320x240 data for RGB and CMYK is stored sequentially as: R[0], G[0], B[0], R[1], G[1], B[1], R[76799], G[76799], B[76799] C[0], M[0], Y[0], K[0], C[1], M[1], Y[1],K[1] C[76799], M[76799], Y[76799], K[76799].

The pointers are incremented by one to access R, G, B or C, M, Y, K data in this order. If the benchmark score is extrapolated for a larger image, the processing time will be almost linearly proportional to the pixel count (e.g. for a 640 x 480 image, it will be multiplied times 4). The iteration/second score will be the inverse (e.g. for a 640 x 480 image, iterations/sec will be multiplied by .25). There is data dependency in the cycle counts for the minimum value K search, due to branch taken or not taken. If this operation is handled by conditional move, the cycle will be constant.

Parallelism for multiple workers

The image processing can be divided among multiple workers, such that each worker is responsible for X pixels. X = Total image pixels / number of workers.

Since the processing of each pixel is independent, this processing allows scalability with no synchronization, other than at the beginning of the input image (to allocate a piece of the input to each worker), and at the end of processing. In a real work application, this would require that the whole image is available before processing is started.

Verification

The output is verified by computing the CRC of the output image. Precomputed CRC values exist for EEMBC datasets.

Analysis of Computing Resources

A "for loop" calculates the conversion of one set of RGB inputs and CMYK outputs at a time. A set of R, G, B input data is read from the memory by incrementing a read pointer. A set of C, M, Y, K output data is written back to the memory by incrementing a write pointer. There is no complex two-dimensional access. The complementary color calculation and correction are simple subtract calculations without any MAC operation. The minimum value search has two branches for processing each pixel.

If (c<m) { K = (Byte)(c<y ? c:y); } else { K = (Byte)(m<y ? m:y); } This can be a very expensive routine because of the branch penalty.

Full-Fury Optimization

By using the compare and conditional moves, the branch penalty can be avoided. VLIW and SIMD can process multiple bytes of data at a time. For example, a four-way SIMD microprocessor can handle 4 x 8-bit data every cycle.

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

Image Rotation

Description

Benchmark to rotate a binary image (greyscale or color) of arbitrary size by 90/180/270 degrees. This benchmark is memory intensive, with very little computation.

Concurrency

Rotation is done by multiple workers cooperating to process a single image. Each worker thread acquires slices (lines of the input) and writes them to the output buffer.

Potential bottlenecks are related to memory interfaces and synchronization between worker threads.

Synchronization bottlenecks can occur directly due to mutex used to acquire input, or indirectly due to cache coherency protocols enforced on the output image.

Implementation

Image is loaded into a buffer in memory before timing starts. An output buffer is allocated after timing starts.

Verification

Output verification is via CRC of the output.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Networking - IP Packet Check

Description

Simulates a network router to compare memory bus effects. This benchmark focuses on checksum calculations and logical compare operations.

Application

The IP Packet Check benchmark performs a subset (essentially the IP Header Validation) of the network layer forwarding function of the Internet protocol suite as specified in RFC1812, “Requirements for IP Version 4 Routers” which can be found at http://www.faqs.org/rfcs/rfc1812.html. The benchmark provides an indication of the potential performance of a microprocessor in an IP router system.

A TCP/IP router normally examines the IP protocol header as part of the switching process. It generally removes the Link Layer header from a received message, modifies the IP header, and replaces the Link Layer header for retransmission. In this benchmark, the Link Layer header has already been removed and will not be replaced, i.e. all processing is done at Layer 3, on the assumption that lower level functions are handled by hardware or an interrupt service routine.

Detailed Description

The benchmark simulates a router with four network interfaces. It initializes a buffer of programmable size with IP datagrams.

The header is always the minimum 20 bytes and is made up of random characters except in the byte positions to be checked (IP version, checksum, and length). A checksum for the IP header is calculated and stored in each datagram. Errors are introduced in certain headers and an error count is logged. Datagrams are allowed to be aligned on the best natural boundary of the microprocessor and padding is added between them. As a benchmark, the IP packet size is chosen randomly to be either 46 bits (small packets) or 1500 bits (large packets) in size. Packet receipt is simulated by creating a dummy store queue. One timed iteration of the benchmark consists of processing each packet header pointed to by the receive queue and moving the descriptor to a holding queue. Results are reported in iterations per second.

Two descriptor queues are created with a pointer to the next descriptor and a pointer to the datagram header. One queue is called the receive queue (rx_queue in the code) and the other queue is the holding queue.

IP datagrams are often stored like this in actual systems using descriptors that are separate from the datagram. A descriptor has a next member that allows it to be put in a linked list and a pointer to a datagram.

As each datagram is processed by the benchmark algorithm it is removed from the receive queue and placed in the holding queue. Processing consists of: 1. Checking that the packet length is large enough to hold the minimum length legal IP datagram (>=20 bytes). 2. Checking that the IP checksum is correct (a bad packet counter is incremented if the checksum is not correct) 3. Checking that the IP version number is 4 4. Checking that the IP header length field is large enough to hold the minimum length legal IP datagram (20 bytes = 5 words) 5. Checking that the IP total length field is large enough to hold the IP datagram header, whose length is specified in the IP header length field.

A single iteration of the benchmark is complete when the receive queue of packet descriptors is empty. At the end of one iteration, the receive queue and the holding queue are switched allowing the next iteration to execute with a full receive queue.

Analysis of Computing Resources

The IP Packet Check benchmark performs integer math on 16 bit unsigned quantities (the checksum calculation) and shift and logical compare operations (the IP version number and length checks). These operations and accessing the data from memory are primarily what is tested by this benchmark. Though the buffer sizes in memory are large, the checksum and verification process is only over the IP descriptors (first 20 bytes of each packet). The code size is trivial and easily fits in even a small L1 Instruction Cache.

Verification

Verification is done by testing the classification of each packet. Each stage in the RFC is given a different error code, and errors on the predefined packet stream have all been precomputed.

Special Notes

Do not directly compare the results of IP Packet Check benchmark to EEMBC Networking Version 1 or 2 Packet Flow benchmarks. Even though the benchmarks test the same function, the algorithm was changed in IP Packet Check to allow a user specified alignment without impacting the number packets processed, as well as parallel processing with multiple workers.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Networking - IP Reassembly

Description

Simulate a network router reassembling fragmented packets.

Application

The Internet Protocol allows IP fragmentation so that datagrams can be fragmented into pieces small enough to pass over a link with a smaller MTU than the original datagram size.

RFC 791 describes the procedure for IP fragmentation, transmission and reassembly of datagrams. RFC 815 describes a simplified reassembly algorithm which can easily be implemented in hosts. This benchmark kernel implements an algorithm similar to the one described in RFC 815 (http://tools.ietf.org/html/rfc815). All packets are fragments of one or more IP packets terminating at this router.

Detailed Description

This kernel is based on the NetBSD kernel code. The benchmark simulates the arrival of a large number of IP fragments of varying lengths. Internally they are split to buffers of predefined size, so that fragments may require a single buffer, a double buffer or a whole cluster of buffers. The degree of fragmentation, the number of fragments per IP packet, the arrival order of the fragments, and the number of packets being reassembled in parallel is configurable. Each thread in the benchmark kernel accepts an input queue of packets to unfragment. Each queue has been classified prior to reassembly so that the queue contains all fragments for each full IP packet on the queue. When a packet “arrives” it is checked for basic correctness. Its packet ID, source, and destination parameters are compared with those of all the packets waiting for reassembly. If the fragment corresponds to a new packet, a new queue is started to hold the additional fragments that will be required to reassemble the packet. If the fragment belongs to a packet reassembly effort already in progress, then the doubly linked list which forms the reassembly queue is traversed to determine where this fragment belongs in the packet. Each fragment contains offset information indicating its relative position to the start of the packet. The new fragment is then inserted into the linked list at the appropriate position, and because fragment overlap is possible, it may be necessary to trim (or even dequeue) adjacent fragments. A check is subsequently made for complete reassembly. If, with the addition of the current fragment, reassembly is complete, fragment concatenation is undertaken and the reassembled packet is passed up the stack. Since each processing thread is in effect a different target, there is no synchronization or data sharing between the threads.

Analysis of Computing Resources

The benchmark spends most of the time managing queue structures. This makes the code very branchy, where the branches depend on memory contents rather then calculations.

Verification

Verification is done by testing the CRC of each complete packet after it has been reassembled. The CRC has been precomputed previously before the packet has been fragmented.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Networking - Transmission Control Protocol (TCP)

Description

This benchmark kernel captures most frequently used and most processing-intensive portion of RFC793 protocol

- Simulates TCP traffic characteristics in real networks
- De-couples processor speed from any randomness in TCP operation

Application

The ability of an embedded processor to handle Transmission Control Protocol (TCP) layer processing is an important consideration for avoiding bottlenecks in network equipment designs. Unlike ATM and some other network protocols that are mainly processed by network processors, ASICs, or specialized hardware blocks directly attached to general purpose processors, the TCP layer is often processed by the CPUs in general purpose processors. The interest of benchmarking TCP performance on embedded general-purpose processors has increased with the connection of more and more embedded devices to the network. The flexibility of TCP is such that it is used in wireline and wireless applications. The ISO reference model is commonly used when discussing protocol layering. This model depicts the TCP layer as sitting on top of the Internet Protocol (IP) layer and under the application layer. The function of IP is to provide a means of transferring TCP segments over inter-connected networks. IP has unique addressing information for each network element, and data communication is based on routing that provides best effort service to TCP and other transmission control layer protocols like UDP. In contrast to IP, TCP service is a reliable, connection-oriented byte stream service. It typically interfaces with an unreliable network layer protocol. Unlike other connection-oriented protocols that are based on a reliable network layer, TCP has to implement a more complex transmission control scheme to overcome these seemingly contradictory philosophies between protocol layers. The basic operation of TCP can be broken down into the following six areas: 1 - Basic data transfer; 2 – Reliability; 3 - Flow control; 4 – Multiplexing; 5 – Connections; 6 - Precedence and security.

The core of the TCP protocol is to transfer data between two connection endpoints. Like data processing in most of the network protocols, large data blocks are chopped into optimized sizes (as deemed by TCP) and encapsulated in a TCP segment. Communications in TCP involve both data and control operations. Comparing data processing with other protocols, the biggest difference with TCP is a mandatory checksum across the entire segment. This is because TCP provides reliable communication service on top of an unreliable IP layer. For the same reason, TCP requires fairly complex control and signaling to achieve reliability, efficiency, and connection management. Compared with IP, data operations are simpler but are more expensive in terms of performance. The cost associated with the data block size is linear in most of the cases. (For example, computing IP style checksum and memory copy.) The benchmark captures all the costly data manipulations while some of the complex but rarely used control logic can be omitted.

Benchmark Details

This benchmark implementation captures the most frequently used and processing-intensive portion of the protocol described in RFC793. The benchmark measures the data and buffer management performance, which is common and expensive in TCP implementations. Also, because this benchmark targets embedded general-purpose processors, the execution

environment should match code size and memory scale. Typically, execution environments include a reasonably-sized memory and high-performance RTOS with shared kernel and user addressing spaces. The scope of this benchmark does not include measuring overall network performance. EEMBC's TCP benchmark follows these general requirement guidelines. Accurate – the benchmark captures all major TCP operations in terms of processing cost Realistic – the benchmark simulates TCP traffic characteristics in real networks Deterministic – the benchmark de-couples processor speed from any randomness in TCP operation Simplistic – the benchmark implementation allows for a simplified TCP implementation with reasonable assumptions

Application protocols that use bulk transfer contribute 90% of the traffic in terms of number of bytes but represent only about half the packets. The TCP benchmark is designed to be flexible enough to capture processor performance for both transfer types.

Benchmark performance metrics include

1. Complete event-driven TCP state machine, connection management signaling
2. Transient behavior in short TCP conversations
3. Buffer management – Data manipulation in both ingress and egress directions
4. Queue management – Send queue, unacknowledged queues in egress direction
5. Separate re-entrant client-server task with context switching
6. Basic flow control
7. Multiple data stream (phase II)
8. Configurable packet size distribution for different traffic patterns

RFC793 requirements that are not included in benchmark include

1. Real-time timer related – RTT estimation and update, RTO
2. Exception handling, out-of-order delivery, duplicates and lost packets

TCP behavior varies dramatically between different applications. Packet sizes, conversation length, and queue depth can all affect processing in different ways. To cope with different scenarios, the benchmark is configurable.

Analysis of Computing Resources

Each work item based on this kernel simulates network traffic using the following steps: 1. Initiate server task. 2. Insert network channel effect 3. Initiate client task 4. Insert network channel operations of client. This processing is repeated until all client connections are closed. This kernel is fairly branch intensive, and CRC computation over the whole packet is also time consuming.

Verification

Each step of the packet processing, as well as the internal structure modifications can be tracked for verification. Tracking is via a CRC on the control structures used for each connection.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Workload Descriptions

ippktcheck-4M

Description: This workload simulates the activity that may happen when checking packet headers over 4M of data. All work items have a working set size of ~4M.

Workload 1 (ippktcheck-4M) Composition:

- ► 1 instance of ippktcheck, input 4M, variable number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (ippktcheck-4Mw1) Composition:

- ► 1 instance of ippktcheck, input 4M, 1 worker

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 4M - though only the packet headers are processed.
- The input 4M contains 5092 Packets, 1729 of them are corrupt, and the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

ipres-4M Variations

Description: This workload simulates the activity that may happen when doing ip reassembly for packets over 1M of packets data.

Workload 1 (ipres-4M) Composition:

- ► 1 instance of ipres, input 3.62M, variable number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (ipres4Mw1) Composition:

- ► 1 instance of ipres, input 3.62M, 1 worker

Workload 3 (ipres4Mw2) Composition:

- ► 1 instance of ipres, input 3.62M, 2 workers

IP Reassembly kernel details:

- Memory to hold fragments: 1.09 M
- IP fragments: 4454
- Single fragment packets: 3311
- Two fragment packets (linked): 377
- Fragment clusters (more than 2 fragments per packet): 370
- See the ip reassembly datasheet for detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

4M-check-reassembly

Description: This workload simulates part of the activity that may happen when sending 4 greyscale images to a printer over the network in landscape orientation. All work items have a working set size of ~4M.

Workload (4M-check-reassembly) Composition:

- ► 1 instance of ippktcheck, input 4M, 1 worker
- ► 1 instance of ipres, input 3.62M, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 4M - though only the packet headers are processed.
- The input 4M contains 5092 Packets, 1729 of them are corrupt, and the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 1.09 M
- IP fragments: 4454
- Single fragment packets: 3311
- Two fragment packets (linked): 377
- Fragment clusters (more than 2 fragments per packet): 370
- See the ip reassembly datasheet for detailed kernel description.

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

4M-tcp-mixed

Description: This workload simulates TCP stack activity sending 4M of data in TCP connections.

Workload (4M-tcp-mixed) Composition:

► 1 instance of tcp, input 4M, 1 worker

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 4M
- See the TCP datasheet for more detailed kernel description.

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

4M-check-reassembly-tcp

Description: This workload simulates part of the activity that may happen when sending 4 greyscale images to a printer over the network in landscape orientation. All work items have a working set size of ~4M.

Workload (4M-check-reassembly-tcp) Composition:

- 1 instance of tcp, input 4M, 1 worker
- 1 instance of ippktcheck, input 4M, 1 worker
- 1 instance of ipres, input 3.62M, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 4M - though only the packet headers are processed.
- The input 4M contains 5092 Packets, 1729 of them are corrupt, and the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 1.09 M
- IP fragments: 4454
- Single fragment packets: 3311
- Two fragment packets (linked): 377
- Fragment clusters (more than 2 fragments per packet): 370
- See the ip reassembly datasheet for detailed kernel description.

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 4M
- See the TCP datasheet for more detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

4M-cmykw2

Description: This workload simulates color space conversion that can happen in a color printer when switching from RGB to CMYK color space. All work items have a working set size of ~4M.

Workload (4M-cmykw2) Composition:

- ► 1 instance of rgbcmyk, 4M, 2 workers

rgbcmyk:

- Input image: 4 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

4M-rotatew2

Description: This workload simulates activity that can happen in a printer when printing in landscape orientation. In particular the input image is rotated 90deg. All work items have a working set size of ~4M.

Workload (4M-rotatew2) Composition:

- ► 1 instance of rotate, 4M, 2 workers

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

NOTES:

The same images are used for image rotation and color conversion.

The images are copies of the same input image.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

4M-cmykw2-rotatew2

Description: This workload simulates part of the activity that may happen when sending 4 greyscale images to a printer over the network in landscape orientation. In particular, the image is rotated and then converted to cmyk color space. All work items have a working set size of ~4M.

Workload (4M-cmykw2-rotatew2) Composition:

- 1 instance of rotate, 4M, 2 workers
- 1 instance of rgbcmyk, 4M, 2 workers

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

rgbcmyk:

- Input image: 4 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

NOTES:

The same images are used for image rotation and color conversion.

The images are copies of the same input image.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

4M-check-reassembly-tcp-cmykw2-rotatew2

Description: This workload simulates part of the activity that may happen when sending 4 greyscale images to a printer over the network in landscape orientation. All work items have a working set size of ~4M

Workload (4M-check-reassembly-tcp-cmykw2-rotatew2) Composition:

- ► 1 instance of tcp, input 4M, 1 worker
- ► 1 instance of ippktcheck, input 4M, 1 worker
- ► 1 instance of ipres, input 3.62M, 2 workers
- ► 1 instance of rotate, 4M, 2 workers
- ► 1 instance of rgbcmyk, 4M, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 4M - though only the packet headers are processed.
- The input 4M contains 5092 Packets, 1729 of them are corrupt, the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 1.09 M
- IP fragments: 4454
- Single fragment packets: 3311
- Two fragment packets (linked): 377
- Fragment clusters (more than 2 fragments per packet): 370
- See the ip reassembly datasheet for detailed kernel description.

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 4M
- See the TCP datasheet for more detailed kernel description.

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

rgbcmyk:

- Input image: 4 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

NOTES:

The same images are used for image rotation and color conversion.

The images are copies of the same input image.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

x-4M Variations

Description: These workloads encode a stream of images in YUV format to h.264 main profile. All work items have a working set size of ~4M.

Workload 1 (x264-4M) Composition:

- ► 1 instance of h.264 encoding, input 2.5M, flexible number of workers
- ► Input stream size for x.264: marsface - 2.5M input, 47K output

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (x264-4Mw1) Composition:

- ► 1 instance of h.264 encoding, input 2.5M, 1 worker
- ► Input stream size for x.264: marsface - 2.5M input, 47K output

Workload 3 (x264-4Mw2) Composition:

- ► 1 instance of h.264 encoding, input 2.5M, 2 workers
- ► Input stream size for x.264: marsface - 2.5M input, 47K output

Workload 4 (x264-4Mq) Composition:

- ► 1 instance of h.264 encoding, input 2.5M, flexible number of workers
- ► Input stream size for x.264: quad marsface - 2.5M input, 47K output

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 5 (x264-4Mqw1) Composition:

- ► 1 instance of h.264 encoding, input 2.5M, 1 worker
- ► Input stream size for x.264: quad marsface - 2.5M input, 47K output

X.264 details:

- Input parameters
  qp=20
  ref=1
  keyint=30
  scenecut=-1
  bframes=1
  no-b-adapt
- See H.264 for more details about this kernel.

4M-check-reassembly-tcp-x264w2

Description: This workload simulates part of the activity that may happen when sending encoding images coming over the network to h.264 format for storage. All work items have a working set size of ~4M.

Workload (4M-check-reassembly-tcp-x264w2) Composition:

- 1 instance of tcp, input 4M, 1 worker
- 1 instance of ippktcheck, input 4M, 1 worker
- 1 instance of ipres, input 3.62M, 2 workers
- 1 instance of h.264 encoding, input 2.5M, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 4M - though only the packet headers are processed.
- The input 4M contains 5092 Packets, 1729 of them are corrupt, the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 1.09 M
- IP fragments: 4454
- Single fragment packets: 3311
- Two fragment packets (linked): 377
- Fragment clusters (more than 2 fragments per packet): 370
- See the ip reassembly datasheet for detailed kernel description.

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 4M
- See the TCP datasheet for more detailed kernel description.

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

X.264 details:

- Input parameters
  qp=20
  qp ref=1
  qp keyint=30
  qp scenecut=-1
  qp bframes=
  qp no-b-adapt
- See H.264 for more details about this kernel.

Input stream size for x.264: marsface - 2.5M input, 47K output

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

ippktcheck-64M Variations

Description: This workload simulates packet header check (RFC1812) over 64M worth of packets.

Workload 1 (ippktcheck-64M) Composition:

- ► 1 instance of ippktcheck, input 64M, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (ippktcheck-64M-1Worker) Composition:

- ► 1 instance of ippktcheck, input 64M, 1 worker

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 64M - though only the packet headers are processed.
- The input 64M contains 81460 Packets, 27677 of them are corrupt, the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-check-reassembly

Description: This workload contains networking algorithms operating over a ~64M input each.

Workload (64M-check-reassembly) Composition:

- 1 instance of ippktcheck, input 64M, 1 worker
- 1 instance of ipres, input 72M, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 64M - though only the packet headers are processed.
- The input 64M contains 81460 Packets, 27677 of them are corrupt, and the algorithm will classify them accordingly.
  Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 72.72 M
- IP fragments: 297859
- Single fragment packets: 220872
- Two fragment packets (linked): 25252
- Fragment clusters (more than 2 fragments per packet): 24394
- See the ip reassembly datasheet for detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-check-reassembly-tcp

Description: This workload simulates activity that may happen when compressing 6 video streams and sending them over the network. Actual streams to go over the network are about 10M, the rest is considered overhead, bad packets, and other data. In particular, the network algorithms are simulating significant bit corruption.

Workload (64M-check-reassembly-tcp) Composition:

- ► 1 instance of tcp, input 64M, 1 worker
- ► 1 instance of ippktcheck, input 64M, 1 worker
- ► 1 instance of ipres, input 72M, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 64M - though only the packet headers are processed.
- The input 64M contains 81460 Packets, 27677 of them are corrupt, and the algorithm will classify them accordingly.
  Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 72.72 M
- IP fragments: 297859
- Single fragment packets: 220872
- Two fragment packets (linked): 25252
- Fragment clusters (more than 2 fragments per packet): 24394
- See the ip reassembly datasheet for detailed kernel description.

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 64M
- See the TCP datasheet for more detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-cmykw2

Description: This workload does the color space conversion from RGB to CMYK on 4 images of 12M.

Workload (64M-cmykw2) Composition:

► 4 instances of rgbcmyk, 12M, 2 workers

rgbcmyk:

- Input image: 12 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

NOTE: *The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-rotatew2

Description: This workload rotates 4 images 90 deg clockwise. Each image is 12M in size.

Workload (64M-rotatew2) Composition:

- ► 4 instances of rotate, 12M, 2 workers

Image Rotation:

- Input image: 12 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

NOTE: *The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-cmykw2-rotatew2

Description: This workload rotates and color converts 4 images. Each image is 12M in size.

Workload (64M-cmykw2-rotatew2) Composition:

- ► 4 instances of rotate, 12M, 2 workers
- ► 4 instances of rgbcmyk, 12M 2 workers

Image Rotation:

- Input image: 12 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

rgbcmyk:

- Input image: 12 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

NOTES:

The same images are used for image rotation and color conversion.

The images are copies of the same input image.

The 48M of image data actually translate to 64M of network traffic due to overhead and packet corruption in this scenario.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-check-reassembly-tcp-cmykw2-rotatew2

Description: This workload simulates activity that may happen when sending 4 greyscale images to a printer over the network in landscape orientation.

Workload (64M-check-reassembly-tcp-cmykw2-rotatew2) Composition:

- ▶ 1 instance of tcp, input 64M, 1 worker
- ▶ 1 instance of ippktcheck, input 64M, 1 worker
- ▶ 1 instance of ipres, input 72M, 2 workers
- ▶ 4 instances of rotate, 12M, 2 workers
- ▶ 4 instances of rgbcmyk, 12M 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 64M - though only the packet headers are processed.
- The input 64M contains 81460 Packets, 27677 of them are corrupt, the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 72.72 M
- IP fragments: 297859
- Single fragment packets: 220872
- Two fragment packets (linked):   25252
- Fragment clusters (more than 2 fragments per packet): 24394
- See the ip reassembly datasheet for detailed kernel description.

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 64M
- See the TCP datasheet for more detailed kernel description.

Image Rotation:

- Input image: 12 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

rgbcmyk:

- Input image: 12 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

NOTES:

The same images are used for image rotation and color conversion.

The images are copies of the same input image.

The 48M of image data actually translate to 64M of network traffic due to overhead and packet corruption in this scenario.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-x264 Variations

Description: This workload performs h264 encoding of 6 input streams.

Workload 1 (x264-64M) Composition:

- ► 2 instances of x264-data1, flexible number of workers
- ► 2 instances of x264-data2, flexible number of workers
- ► 2 instances of x264-data4, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (64M-x264-1worker) Composition:

- ► 2 instances of x264-data1, 1 worker
- ► 2 instances of x264-data2, 1 worker
- ► 2 instances of x264-data4, 1 worker

Workload 3 (64M-x264-2workers) Composition:

- ► 2 instances of x264-data1, 2 workers
- ► 2 instances of x264-data2, 2 workers
- ► 2 instances of x264-data4, 2 workers

Workload 4 (64M-x264-4workers) Composition:

- ► 2 instances of x264-data1, 4 workers
- ► 2 instances of x264-data2, 4 workers
- ► 2 instances of x264-data4, 4 workers

Workload 5 (64M-x264-8workers) Composition:

- ► 2 instances of x264-data1, 8 workers
- ► 2 instances of x264-data2, 8 workers
- ► 2 instances of x264-data4, 8 workers

X.264 details:

- Input parameters for all streams
  qp=20
  ref=1
  keyint=30
  scenecut=-1
  bframes=1
  no-b-adapt
- See H.264 for more details about this kernel.

Input stream sizes for x.264:

- data1 - 24M input, 2M output
- data2 - 2M input, 47K output
- data4 - 37M input, 2.5M output

Embedded Microprocessor Benchmark Consortium

www.eembc.org

64M-check-reassembly-tcp-h264w2

Description: This workload simulates activity that may happen when compressing 6 video streams and sending them over the network. Actual streams to go over the network are about 10M, the rest is considered overhead, bad packets, and other data. The network algorithms are simulating significant bit corruption.

Workload (64M-check-reassembly-tcp-h264w2) Composition:

- ► 1 instance of tcp, input 64M, 1 worker
- ► 1 instance of ippktcheck, input 64M, 1 worker
- ► 1 instance of ipres, input 72M, 2 workers
- ► 2 instances of x264-data1, 2 workers
- ► 2 instances of x264-data2, 2 workers
- ► 2 instances of x264-data4, 2 workers

Packet Header Check (RFC1812) details:

- The size of the input packet buffer for the packet check algorithm is 64M - though only the packet headers are processed.
- The input 64M contains 81460 Packets, 27677 of them are corrupt, the algorithm will classify them accordingly. Bit corruption is uniformly distributed.
- See the packet header check datasheet for more detailed kernel description.

IP Reassembly kernel details:

- Memory to hold fragments: 72.72 M
- IP fragments: 297859
- Single fragment packets: 220872
- Two fragment packets (linked): 25252
- Fragment clusters (more than 2 fragments per packet): 24394
- See the ip reassembly datasheet for detailed kernel description.

TCP details:

- TCP ring size: 8
- Total input data for TCP transfer: 64M
- See the TCP datasheet for more detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

X.264 details:

- Input parameters for all streams
  qp=20
  ref=1
  keyint=
  scenecut=-1
  bframes=1
  no-b-adapt
- See H.264 for more details about this kernel.

Input stream sizes for x.264:

- data1 - 24M input, 2M output
- data2 - 2M input, 47K output
- data4 - 37M input, 2.5M output

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

iDCT-4M Variations

Description: This workload performs iDCT on buffers. Total size of buffers is 4M.

Workload 1 (iDCT-4M) Composition:

- ► 1 instance of iDCT, 4M input, variable number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (iDCT-4Mw1) Composition:

- ► 1 instance of iDCT, 4M input, one worker

iDCT details:

- See the iDCT datasheet for more details on this kernel.

ipres-72M

Description: This workload performs ip reassembly on fragments whose total size is ~72M.

Workload 1 (ipres-72M) Composition:

- 1 instance of ipres, input 72M, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (ipres-72M1worker) Composition:

- 1 instance of ipres, input 72M, 1 worker

Workload 3 (ipres-72M2 worker) Composition

- 1 instance of ipres, input 72M, 2 workers

IP Reassembly kernel details:

- Memory to hold fragments: 72.72 M
- IP fragments: 297859
- Single fragment packets: 220872
- Two fragment packets (linked): 25252
- Fragment clusters (more than 2 fragments per packet): 24394
- See the ip reassembly datasheet for detailed kernel description.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

md5-32M Variations

Description: This workload performs MD5 checksum on buffers. Total size of buffers is 32M.

Workload 1 (md5-32M) Composition:

- ► 1 instance of md5, 32M input, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (md5-32Mw1) Composition:

- ► 1 instance of md5, 32M input, 1 worker

Workload 3 (md5-32Mw2) Composition:

- ► 1 instance of md5, 32M input, 2 workers

Workload 4 (md5-32Mw4) Composition:

- ► 1 instance of md5, 32M input, 4 workers

MD5 details:

- Number of buffers for digest: 128
- Size of each buffer: 128K
- See MD5 for more details on this kernel.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

md5-4M Variations

Description: This workload performs MD5 checksum on buffers. Total size of buffers is 4M.

Workload 1 (md5-4M) Composition:

- ► 1 instance of md5, 4M input, flexible number of workers to fit the platform

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (md5-4Mw1) Composition:

- ► 1 instance of md5, 4M input, 1 worker

MD5 details:

- Number of buffers for digest: 128
- Size of each buffer: 32K
- See MD5 for more details on this kernel.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rgbcmyk-4M Variations

Description: This workload performs color conversion from RGB to CMYK color space. CMYK colors are used in color printers, and RGB are used for screen displays. All work items have a working set size of ~4M.

Workload 1 (rgbcmyk-4M) Composition:

- ► 1 instance of rgbcmyk, 4M, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rgbcmyk-4Mw1) Composition:

- ► 1 instance of rgbcmyk, 4M, 1 worker

rgbcmyk:

- Input image: 4 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rgbcmyk-5x12M Variations

Description: This workload does the color space conversion from RGB to CMYK on 5 12M images.

Workload 1 (rgbcmyk-5x12M) Composition:

- ► flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rgbcmyk-5x12M1workers) Composition:

- ► 1 worker

Workload 3 (rgbcmyk-5x12M2workers) Composition:

- ► 2 workers

Workload 4 (rgbcmyk-5x12M4workers) Composition:

- ► 4 workers

Workload 5 (rgbcmyk-5x12M8workers) Composition:

- ► 8 workers

rgbcmyk:

- Input image: 12 Mega Pixels, Greyscale
- Operation: Convert from screen RGB to printer CMYK color space.
- See the rgb to cmyk conversion datasheet for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

***NOTE:** The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rotate-16x4Ms1 Variations

Description: This workload rotates 16 4M grayscale images by 90 deg clockwise. Image slice size is 1 and rotation is 1 line at a time.

Workload 1 (rotate-16x4Ms1) Composition:

- ► flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rotate-16x4Ms1w1) Composition:

- ► 1 worker

Workload 3 (rotate-16x4Ms1w2) Composition:

- ► 2 workers

Workload 4 (rotate-16x4Ms1w4) Composition:

- ► 4 workers

Workload 5 (rotate-16x4Ms1w8) Composition:

- ► 8 workers

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.
- 256 unique colors

***NOTE:** The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rotate-16x4Ms32 Variations

Description: This workload rotates 16 4M greyscale images by 90 deg clockwise, 32 lines at a time (slice size of 32).

Workload 1 (rotate-16x4Ms32) Composition:

- ► flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rotate-16x4Ms32w1) Composition:

- ► 1 worker

Workload 3 (rotate-16x4Ms32w2) Composition:

- ► 2 workers

Workload 4 (rotate-16x4Ms32w4) Composition:

- ► 4 workers

Workload 5 (rotate-16x4Ms32w8) Composition:

- ► 8 workers

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.
- 256 unique colors

NOTE: *The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rotate-16x4Ms4 Variations

Description: This workload rotates 16 4M greyscale images by 90 deg clockwise, 4 lines at a time (slice size of 4).

Workload 1 (rotate-16x4Ms4w1) Composition:

- ► 1 worker

Workload 2 (rotate-16x4Ms4w2) Composition:

- ► 2 workers

Workload 3 (rotate-16x4Ms4w4) Composition:

- ► 4 workers

Workload 4 (rotate-16x4Ms4w8) Composition:

- ► 8 workers

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.
- 256 unique colors

NOTE: *The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rotate-16x4Ms64 Variations

Description: This workload rotates 16 images by 90 deg clockwise, 4 lines at a time (slice size of 64).

Workload 1 (rotate-16x4Ms64) Composition:

- ► flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rotate-16x4Ms64w1) Composition:

- ► 1 worker

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.
- 256 unique colors

NOTE: *The images are copies of the same input image.*

rotate-34kx128w1

Description: This workload rotates 128 34k greyscale images by 90 deg clockwise, 64 lines at a time (slice size = 64). Total input size is 4M.

Workload 1 (rotate-34kx128w1) Composition:

- ► 1 worker

Image Rotation:

- Input image: 33823 pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed description.

Image Details:

- 227 x 149 Pixels.
- 8 bits per pixel.
- 146 unique shades of grey.

NOTE: *The images are copies of the same input image.*

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rotate-34kx512-90deg

Description: This workload rotates 512 34k greyscale images by 90 deg clockwise, 64 lines at a time (slice size = 64). Total input size is 17M.

Workload 1 (rotate-34kx512-90deg) Composition:

- 2 workers

Image Rotation:

- Input image: 33823 pixels, Greyscale
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed kernel description.

Image Details:

- 227 x 149 Pixels
- 8 bits per pixel.
- 146 unique colors

NOTE: *The images are copies of the same input image.*

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

rotate-4Msx Variations

Description: This workload rotates greyscale images by 90 deg, variable number of lines. Image size is 4M. All work items have a working set size of ~4M.

Workload 1 (rotate-4Ms1) Composition:

- 1 instance of rotate, 4M, slice size is 1, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rotate-4Ms1w1) Composition:

- 1 instance of rotate, 4M, slice size is 1, 1 worker

Workload 3 (rotate-4Ms32 Composition:

- 1 instance of rotate, 4M, slice size is 32, flexible number of workers

(Run rules: This workload must be run with only one context active (-c1) but number of workers may be changed (-w<N>).)

Workload 4 (rotate-4Ms32w1) Composition:

- 1 instance of rotate, 4M, slice size is 32, 1 worker

Workload 5 (rotate-4Ms4) Composition:

- 1 instance of rotate, 4M, slice size is 4, flexible number of workers

(Run rules: This workload must be run with only one context active (-c1) but number of workers may be changed (-w<N>).)

Workload 6 (rotate-4Ms4w1) Composition:

- 1 instance of rotate, 4M, slice size is 4, 1 worker

Workload 7 (rotate-4Ms64) Composition:

- 1 instance of rotate, 4M, slice size is 64, flexible number of workers

(Run rules: This workload must be run with only one context active (-c1) but number of workers may be changed (-w<N>).)

Workload 8 (rotate-4Ms64w1) Composition:

- 1 instance of rotate, 4M, slice size is 64, 1 worker

Image Rotation:

- Input image: 4 Mega Pixels, Greyscale.
- Operation: 90deg clockwise rotation
- See Image Rotation for more detailed description.

Image Details: 2272 x 1704 Pixels; 4:3 ratio with 8 bits per pixel.

Embedded Microprocessor Benchmark Consortium

www.eembc.org

rotate-color-4M-90deg Variations

Description: This workload rotates 1 4Mpixels color image 90 deg clockwise.

Workload 1 (rotate-color-4M-90deg) Composition:

- ► 1 instance of rotate, 12M input (4 Mpixels at RGB), flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rotate-color-4M-90degw1) Composition:

- ► 1 instance of rotate, 12M input (4 Mpixels at RGB), 1 worker

Image Rotation:

- Input image: 12 Mega Pixels, Greyscale
- Operation: 90deg clockwise rotation
- Slice size: 32
- See Image Rotation for more detailed description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 24 bits per pixel.
- 159185 unique colors

NOTE: *The images are copies of the same input image.*

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

rotate-color1Mp Variations

Description: This workload rotates greyscale images by 90deg. Image size is ~1M pixels in color. All work items have a working set size of ~4M.

Workload 1 (rotate-color1Mp) Composition:

- ► 1 instance of rotate, 1Mp color image, flexible number of workers

(Run with only one context active (-c1) but variable number of workers (-w<N>).)

Workload 2 (rotate-color1Mpw1) Composition:

- ► 1 instance of rotate, 1Mp color image, 1 worker

Image Rotation:

- Input image: 1 Mega Pixels, color.
- Operation: 90deg clockwise rotation, slice size is 1.
- See Image Rotation for more detailed description.

Image Details:

- 2272 x 1704 Pixels
- 4:3 ratio with 8 bits per pixel.

Consumer Workloads

4M-rotatew2.keys
4M-x264w2.keys
64M-rotatew2.keys
64M-x264-1worker.keys
64M-x264-2workers.keys
64M-x264-4workers.keys
64M-x264-8workers.keys
consumer_v2-cjpeg.keys
consumer_v2-djpeg.keys
consumer_v2-mp3player.keys
empty.keys
empty-wld.keys
filters_v2-common.keys
filters_v2-rgbhpg03.keys
filters_v2-rgbyiq03.keys
filters-rgbhpg01.keys
filters-rgbyiq01.keys
huffde.keys
huffde-all.keys
md5.keys
md5-128M16worker.keys
md5-128M1worker.keys
md5-128M2worker.keys
md5-128M4worker.keys
md5-1M16worker.keys
md5-1M1worker.keys
md5-1M2worker.keys
md5-1M4worker.keys
md5-32M.keys
md5-32M16worker.keys
md5-32M1worker.keys
md5-32M2worker.keys
md5-32M4worker.keys
md5-4M.keys
md5-4Mw1.keys
mp2decode1.keys
mp2decode2.keys
mpeg2-90Mout-1worker.keys
mpeg2-90Mout-2workers.keys
mpeg2-90Mout-4workers.keys
mpeg2-90Mout-8workers.keys
mpeg2-base.keys
oa-rotatev2.keys
rotate-16x4Ms1.keys
rotate-16x4Ms1w1.keys
rotate-16x4Ms1w2.keys
rotate-16x4Ms1w32.keys
rotate-16x4Ms1w4.keys
rotate-16x4Ms1w8.keys
rotate-16x4Ms32.keys
rotate-16x4Ms32w1.keys
rotate-16x4Ms32w2.keys
rotate-16x4Ms32w4.keys
rotate-16x4Ms32w8.keys
rotate-16x4Ms4w1.keys
rotate-16x4Ms4w2.keys
rotate-16x4Ms4w4.keys
rotate-16x4Ms4w8.keys
rotate-16x4Ms64.keys
rotate-16x4Ms64w1.keys
rotate-34k-180deg.keys
rotate-34k-270deg.keys
rotate-34k-90deg.keys
rotate-34kX128w1.keys
rotate-34kX16-90deg.keys
rotate-34kX512-90deg.keys
rotate-4M-180deg.keys
rotate-4M-270deg.keys
rotate-4M-90deg.keys
rotate-4Ms1.keys
rotate-4Ms1w1.keys
rotate-4Ms32.keys
rotate-4Ms32w1.keys
rotate-4Ms4.keys
rotate-4Ms4w1.keys
rotate-4Ms64.keys
rotate-4Ms64w1.keys
rotate-520k-180deg.keys
rotate-520k-270deg.keys
rotate-520k-90deg.keys
rotate-520kX16-90deg.keys
rotate-color1Mp.keys
rotate-color1Mpw1.keys
rotate-color-4M-90deg.keys
rotate-color-4M-90degw1.keys
video-mp2decode.keys
video-x264.keys
x264-4M.keys
x264-4Mw1.keys
x264-64M.keys
x264-90M-1worker.keys
x264-90M-2workers.keys
x264-90M-4workers.keys
x264-base.keys

Embedded
Microprocessor
Benchmark
Consortium

www.eembc.org

Networking Workloads

4M-check.keys
4M-check-reassembly.keys
4M-check-reassembly-tcp.keys
4M-reassembly.keys
4M-tcp-mixed.keys
64M-check-reassembly.keys
64M-check-reassembly-tcp.keys
64M-tcp-mixed.keys
empty.keys
empty-wld.keys
ippktcheck-4M.keys
ippktcheck-4Mw1.keys
ippktcheck-64M.keys
ippktcheck-64M-1Worker.keys
ippktcheck-64M-2Worker.keys
ippktcheck-8x4M-1Worker.keys
ippktcheck-8x4M-4Worker.keys
ipres-100M10worker.keys
ipres-100M1worker.keys
ipres-100M2worker.keys
ipres-100M4worker.keys
ipres-4M.keys
ipres-4Mw1.keys
ipres-6M1worker.keys
ipres-6M4worker.keys
ipres-72M.keys
ipres-72M1worker.keys
ipres-72M2worker.keys
md5.keys
md5-128M16worker.keys
md5-128M1worker.keys
md5-128M2worker.keys
md5-128M4worker.keys
md5-1M16worker.keys
md5-1M1worker.keys
md5-1M2worker.keys
md5-1M4worker.keys
md5-32M.keys
md5-32M16worker.keys
md5-32M1worker.keys
md5-32M2worker.keys
md5-32M4worker.keys
md5-4M.keys
md5-4Mw1.keys
networking-ippktcheck.keys
networking-ipres.keys
networking-tcp.keys
tcpbase.keys
tcp-bulk_x1G.keys
tcp-jumbo_x1G.keys
tcp-mixed_x1G.keys

Office Automation Workloads

4M-cmykw2.keys
4M-cmykw2-rotatew2.keys
4M-rotatew2.keys
64M-cmykw2.keys
64M-cmykw2-rotatew2.keys
64M-rotatew2.keys
empty.keys
empty-wld.keys
filters_v2-common.keys
filters_v2-rgbcmyk03.keys
oa-rotatev2.keys
rgbcmyk-12M2workers.keys
rgbcmyk-4M.keys
rgbcmyk-4Mw1.keys
rgbcmyk-5x12M.keys
rgbcmyk-5x12M1workers.keys
rgbcmyk-5x12M2workers.keys
rgbcmyk-5x12M4workers.keys
rgbcmyk-5x12M8workers.keys
rotate-16x4Ms1.keys
rotate-16x4Ms1w1.keys
rotate-16x4Ms1w2.keys
rotate-16x4Ms1w32.keys
rotate-16x4Ms1w4.keys
rotate-16x4Ms1w8.keys
rotate-16x4Ms32.keys
rotate-16x4Ms32w1.keys
rotate-16x4Ms32w2.keys
rotate-16x4Ms32w4.keys
rotate-16x4Ms32w8.keys
rotate-16x4Ms4w1.keys
rotate-16x4Ms4w2.keys
rotate-16x4Ms4w4.keys
rotate-16x4Ms4w8.keys
rotate-16x4Ms64.keys
rotate-16x4Ms64w1.keys
rotate-34k-180deg.keys
rotate-34k-270deg.keys
rotate-34k-90deg.keys
rotate-34kX128w1.keys
rotate-34kX16-90deg.keys
rotate-34kX512-90deg.keys
rotate-4M-180deg.keys
rotate-4M-270deg.keys
rotate-4M-90deg.keys
rotate-4Ms1.keys
rotate-4Ms1w1.keys
rotate-4Ms32.keys
rotate-4Ms32w1.keys
rotate-4Ms4.keys
rotate-4Ms4w1.keys
rotate-4Ms64.keys
rotate-4Ms64w1.keys
rotate-520k-180deg.keys
rotate-520k-270deg.keys
rotate-520k-90deg.keys
rotate-520kX16-90deg.keys
rotate-color1Mp.keys
rotate-color1Mpw1.keys
rotate-color-4M-90deg.keys
rotate-color-4M-90degw1.keys

Networking / Consumer Combined Workloads

4M-check-reassembly-tcp-x264w2
64M-check-reassembly-tcp-h264w2

4M-check-reassembly-tcp-cmykw2-rotatew2
64M-check-reassembly-tcp-cmykw2-rotatew2