Reed Electronics Group

MICROPROCESSOR REPORT

www.MPRonline.com

THE INSIDER'S GUIDE TO MICROPROCESSOR HARDWARE

EEMBC's MultiBench Arrives

CPU Benchmarks: Not Just For 'Benchmarketing' Any More

By Tom R. Halfhill {7/28/08-01}

Imagine a world without measurements or statistical comparisons. Baseball fans wouldn't fail to notice that a .300 hitter is better than a .100 hitter. But would they welcome a trade that sends the .300 hitter to Cleveland for three .100 hitters? System designers and software developers face similar quandaries when making trade-offs with multicore processors. Even if a dual-core processor appears to be better than a single-core processor, how much better is it? Twice as good? Would a quad-core processor be four times better? Are more cores worth the additional cost, design complexity, power consumption, and programming difficulty?

The Embedded Microprocessor Benchmark Consortium (EEMBC) wants to help answer those questions. EEMBC's MultiBench 1.0 is a new benchmark suite for measuring the throughput of multiprocessor systems, including those built with multicore processors. MultiBench is EEMBC's biggest launch since introducing the EnergyBench power-consumption benchmarks two years ago. (See MPR 7/17/06-02, "EEMBC Energizes Benchmarking.")

[Full disclosure: EEMBC was founded and is headed by Markus Levy, a former *Microprocessor Report* technology analyst and a current member of the *MPR* Editorial Board.]

EEMBC's Evolution: More Than Scores

Since its inception in 1997, EEMBC's mission has been to produce benchmarks for embedded processors, not for PC or server CPUS. In keeping with that mission, the MultiBench suite consists of compute-intensive tasks commonly performed by networking equipment, office equipment (especially printers), and consumer-electronics products. Although EEMBC benchmarks may be considered artificial in that they aren't complete applications, they perform functions extracted or adapted from real-world applications.

EEMBC's role has evolved over the years, and MultiBench is another step. Originally, EEMBC was conceived as an independent entity that would create benchmark suites and certify the scores for accuracy, allowing vendors and customers to make valid comparisons among embedded microprocessors. (See MPR 5/1/00-02, "EEMBC Releases First Benchmarks.") EEMBC still serves that role. But, as it turns out, most EEMBC members don't openly publish their scores. Instead, they disclose scores to prospective customers under an NDA or use the benchmarks for internal testing and analysis.

Partly for this reason, *MPR* rarely cites EEMBC scores in articles. EEMBC forbids members to publicly release their scores without EEMBC's independent certification. Another reason for the scarcity of EEMBC scores in *MPR* is they're usually not available for the processors we write about—processors that aren't available yet. And even if more EEMBC scores were available, some engineers remain suspicious of any attempts to distill microprocessor performance into a few numbers. (See MPR 8/30/04-01, "Benchmarking the Benchmarks.")

MultiBench seems to be moving in the same general direction as other EEMBC benchmarks—away from published scores, closer to private testing. Although some EEMBC members have been using early versions of MultiBench for six months, no member is near to releasing certified MultiBench scores. *MPR* suspects that MultiBench will find its niche primarily as an analysis tool for internal testing and closed-door sales pitches, not as a source for widely publicized benchmark comparisons.

That's not all bad. Although published scores make for interesting "benchmarketing," what the industry sorely needs is a technically valid method for evaluating the performance of multicore processors. Within its limitations, MultiBench meets that need.

New Concept: Work Items and Workloads

EEMBC adapted most tasks in MultiBench from existing EEMBC suites, in addition to writing entirely new tasks. Code reuse saved development time and made sense, because multicore embedded processors generally perform the same application-level tasks as single-core embedded processors. Tasks were created or ported by EEMBC members and by the EEMBC Technology Center and its director of software engineering, Shay Gal-On.

Previously, EEMBC referred to the benchmark tasks in each suite as "kernels"—a somewhat confusing term, because they aren't related to operating-system kernels. In EEMBC parlance, a kernel is simply an algorithm or routine that performs a common task found in real-world embedded software. Now EEMBC is using new terminology. MultiBench tasks are called "workloads," which may include one or more "work items." Work items are similar to the kernels of old, because they mate algorithms with sample datasets on which the algorithms operate.

For example, one MultiBench workload is 64M-cmykw2. It has a single work item: a color-conversion routine that transforms four 12-megapixel images from the RGB color space to the CMYK color space. Another single-item MultiBench workload is 64M-rotatew2, an image-rotation routine that operates on four 12-megapixel images, turning each one 90 degrees clockwise. And another MultiBench workload is 64M-cmykw2-rotatew2, which combines the color-conversion and rotation workloads to create a two-item workload. In all, MultiBench 1.0 has 36 workloads, some of which are combinations of work items in other workloads.

There's another twist. Work items may be single threaded or may decompose the data for processing by multiple threads. This variation allows MultiBench to test data-level parallelism within a single work item as well as thread-level parallelism among multiple work items. The grand concept is a little difficult to grasp at first, but it allows for a great variety of possible workloads. A MultiBench workload may consist of a single work item; two or more instances of the same work item, operating on different datasets; two or more different work items; and work items that are single- or multithreaded. Figure 1 illustrates this concept.

EEMBC chose the 36 workloads in MultiBench 1.0 to keep things relatively simple. Actually, EEMBC has ported almost all the kernels in existing EEMBC suites to function as MultiBench workloads. Including all these workloads in MultiBench 1.0 would have overwhelmed benchmark testers. For scoring purposes, if nothing else, it's better to limit the number of workloads to a manageable number. For now, MultiBench departs from the usual EEMBC practice of sorting tasks into application-oriented categories of benchmark suites (Networking, Consumer, Telecom, etc.). MultiBench 1.0 is a unified suite that doesn't specialize by application domain. Specialized suites will come in the future.

Creating Custom Workloads

Although MultiBench 1.0 defines 36 workloads, EEMBC members and MultiBench licensees can create their own custom workloads by selectively choosing work items and changing the parameters of the items. Almost all MultiBench work items can operate on different datasets with adjustable levels of threading. Testers can substitute their own datasets, too.

Custom workloads aren't valid for MultiBench scoring, but they allow testers to create a virtually infinite variety of workloads, even when using the standard EEMBC datasets. Testers can exercise processors and systems to identify strengths and weaknesses or to compare the performance of multicore processors with that of single-core processors. Engineers can test the effectiveness of multithreading at different levels: time-sliced on a single processor or distributed among multiple processors.

To help testers assemble the workloads, EEMBC offers new tools. Figure 2 is a screen shot of the Workload Creator, which lets users choose work items from a list. Items available in the list depend on which workloads the tester has licensed. In addition, testers can build their own work items from scratch and link them into the Workload Creator by implementing a special application programming interface (API). Workload Creator is part of MultiBench Architect, an optional tool. EEMBC's technical documentation explains the function of each work item and shows its adjustable parameters, such as the number of possible threads and the official EEMBC datasets approved for use with the item.

Because MultiBench allows a variable number of work items to run simultaneously, the number of simultaneous threads is also variable. And some multithreaded work items operate on multiple slices of data—potentially, another adjustable parameter. When porting existing kernels to MultiBench, EEMBC made other improvements as well, such as modifying the algorithms to mirror the latest application trends and changing the sample data to reflect the larger datasets crunched by today's systems. In all these ways, MultiBench is more than simply a multiprocessing version of existing EEMBC benchmarks. At every level, EEMBC has updated the tasks to better represent the progress of real-world embedded software.

Inside the MultiBench Suite

Existing benchmark suites from which EEMBC derived the MultiBench workloads include Networking 2.0, Office Automation 2.0, and Digital Entertainment 1.0. As mentioned above, EEMBC has ported almost all the tasks in all the suites—including Automotive 1.1 and Telecom 1.1—but is using only a subset of them in MultiBench 1.0. Future releases of MultiBench will have more workloads. In addition to

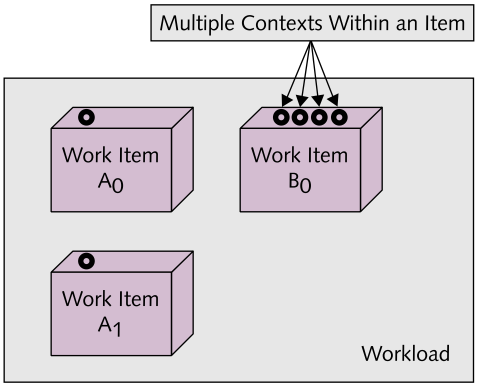
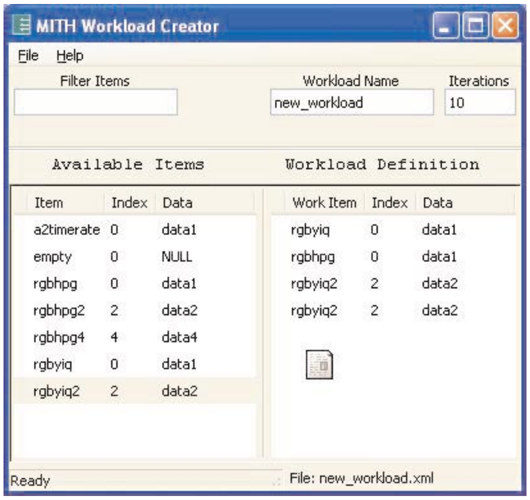


Figure 1. MultiBench 1.0 introduces the concept of work items and workloads instead of kernels. In this figurative example, the workload includes three work items. Items A_0 and A_1 execute the same routine (kernel) on different datasets, and both items are single threaded. Work item B_0 contains a different kernel that decomposes its dataset in a manner that allows four threads to operate on the data simultaneously. Therefore, this combined workload contains three work items testing three levels of parallelism: thread-level parallelism among disparate work items (A_0, A_1, and B_0); thread-level parallelism among two instances of the same work item, operating on different data (A_0 and A_1); and data-level parallelism within a single work item (B_0).



Filter Items

Workload Name: new_workload

Iterations: 10

Available Items

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

Workload Definition

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

Ready

File: new_workload.xml

Figure 2. EEMBC's Workload Creator—part of the optional MultiBench Architect tool—allows testers to pick MultiBench work items from the list on the left and add them to the workload list on the right. Each combination of a routine and a dataset is a work item in the workload. An XML-based definition file encapsulates the work items, links them into the Workload Creator, and allows testers to visualize the components of a workload.

porting tasks from the existing benchmark suites, EEMBC wrote 11 new MultiBench workloads from scratch. Table 1 lists all 36 workloads in MultiBench 1.0.

Although EEMBC adapted most MultiBench workloads from existing kernels in other EEMBC suites, the origin of a workload doesn't necessarily define the scope of its usefulness. Some tasks are applicable to multiple embedded applications that seem unrelated. For instance, the work items for packet checking and reassembly originate from the Networking 2.0 suite, because they perform packet-processing functions common in network routers. But those same work items are relevant to networked printers, which ordinarily would be represented by kernels in the Office Automation suite.

Because EEMBC modified the existing kernels when porting them to MultiBench, the benchmark scores of the old and new versions aren't directly comparable. In other words, it's not valid to compare the score of a particular MultiBench workload with the score of its original kernel before it was ported to MultiBench. To find a baseline of single-core performance for comparisons with multicore performance, testers can run the MultiBench version of a work item as a single thread on a single core or run the workload as multiple time-sliced threads on a single core. Most MultiBench workloads can execute as a single thread on one core, as multiple threads on one core, or as multiple threads on multiple cores.

New API Resembles Posix Threading

EEMBC implemented threading in the MultiBench workloads by creating an API that closely resembles the well-known API for Posix "pthreads." Although the MultiBench API isn't identical to the Posix pthread API, it should be easy to run MultiBench on any Posix-compliant operating system by mapping the MultiBench API calls to corresponding calls in the Posix API.

The MultiBench API has 13 essential calls for threading. It will run on any system software that supports the minimum requirements of thread creation, thread scheduling, thread signaling, mutexes, and thread destruction. Even a roll-your-own substitute for a commercial operating system—a popular alternative in the embedded world—can run MultiBench by implementing those 13 API calls, or by mapping them to another threading API.

Requirements for the thread scheduler are equally flexible. The scheduler can be smart enough to assign different priorities to threads, or so dumb that it juggles multiple threads in simple round-robin fashion. The MultiBench API optionally supports "affinity"—associating a particular thread with a particular processor core.

Thread signaling, another requirement of the MultiBench API, allows a program to awaken an idle thread—a critical feature for priority scheduling. Mutexes provide mutually exclusive access to data, preventing multiple threads from changing the same memory location at the same time. To implement a mutex, the processor must have an atomic test-and-set operation protected from interrupts. Many embedded processors have this capability. For instance, Freescale Semiconductor recently enhanced the Power e500mc processor core by adding an atomic read-modify-write operation. (See the sidebar, "The New, Improved Power e500mc

MultiBench Workloads	Workload Description	Notes	Existing EEMBC Suite
64M-check-reassembly	Check and reassemble IP packets	64MB input (81,460 packets)	Networking 2.0
64M-check-reassembly-tcp	Check and reassemble TCP-IP packets	64MB input (81,460 packets)	Networking 2.0
64M-check-reassembly-tcp-h264w2	Check and reassemble TCP-IP packets, compress six H.264 video streams	Simulates compressing and sending video over network	Networking 2.0
64M-check-reassembly-tcp-cmykw2-rotatew2	Check and reassemble TCP-IP packets, convert 12MP grayscale image RGB - CMYK, rotate image 90° clockwise	Simulates 4 grayscale images sent over network to printer, rotated to landscape	Networking 2.0
64M-tcp-mixed	TCP transfer processing	64MB data, 1 thread	Networking 2.0
ippktcheck-64M-1Worker	IP packet header check (RFC1812)	64MB input, 1 thread	Networking 2.0
ipres-72M1worker	Reassemble 297,859 IP fragments	72MB input, 1 thread	Networking 2.0
ipres-72M2worker	Reassemble 297,859 IP fragments	72MB input, 2 threads	Networking 2.0
64M-cmykw2-rotatew2	Rotate four 12MP images 90° clockwise, convert RGB - CMYK color space	Printer simulation, 64MB data, 2 threads per operation	New workload
64M-rotatew2	Rotate four 12MP images 90° clockwise	2 threads on 4 instances	New workload
64M-cmykw2	Convert four 12MP images RGB - CMYK	2 threads on 4 instances	Digital Entertainment
md5-32M1worker	Calculate MD5 (Message Digest 5) checksum	32MB buffer input, 1 thread	New workload
md5-32M2worker	Calculate MD5 (Message Digest 5) checksum	32MB buffer input, 2 threads	New workload
md5-32M4worker	Calculate MD5 (Message Digest 5) checksum	32MB buffer input, 4 threads	New workload
rotate-color-4M-90deg	Rotate 4MP color image 90° clockwise	12MB data, 2 threads	New workload
rotate-34kX512-90deg	Rotate 512 images 90° clockwise	34KB images, 2 threads	New workload
rgbcmyk-5x12M1workers	Convert 12MP grayscale image RGB - CMYK	Five images, 1 thread each	Digital Entertainment
rgbcmyk-5x12M2workers	Convert 12MP grayscale image RGB - CMYK	Five images, 2 threads each	Digital Entertainment
rgbcmyk-5x12M4workers	Convert 12MP grayscale image RGB - CMYK	Five images, 4 threads each	Digital Entertainment
rgbcmyk-5x12M8workers	Convert 12MP grayscale image RGB - CMYK	Five images, 8 threads each	Digital Entertainment
rotate-16x4Ms1w1	Rotate 16 images 90° clockwise, 1 line/pass	4MB images, 1 thread	Digital Ent. and OA
rotate-16x4Ms1w2	Rotate 16 images 90° clockwise, 1 line/pass	4MB images, 2 threads	Digital Ent. and OA
rotate-16x4Ms1w4	Rotate 16 images 90° clockwise, 1 line/pass	4MB images, 4 threads	Digital Ent. and OA
rotate-16x4Ms1w8	Rotate 16 images 90° clockwise, 1 line/pass	4MB images, 8 threads	Digital Ent. and OA
rotate-16x4Ms32w1	Rotate 16 images 90° clockwise, 32 lines/pass	4MB images, 1 thread	Digital Ent. and OA
rotate-16x4Ms32w2	Rotate 16 images 90° clockwise, 32 lines/pass	4MB images, 2 threads	Digital Ent. and OA
rotate-16x4Ms32w4	Rotate 16 images 90° clockwise, 32 lines/pass	4MB images, 4 threads	Digital Ent. and OA
rotate-16x4Ms32w8	Rotate 16 images 90° clockwise, 32 lines/pass	4MB images, 8 threads	Digital Ent. and OA
rotate-16x4Ms4w1	Rotate 16 images 90° clockwise, 4 lines/pass	4MB images, 1 thread	Digital Ent. and OA
rotate-16x4Ms4w2	Rotate 16 images 90° clockwise, 4 lines/pass	4MB images, 2 threads	Digital Ent. and OA
rotate-16x4Ms4w4	Rotate 16 images 90° clockwise, 4 lines/pass	4MB images, 4 threads	Digital Ent. and OA
rotate-16x4Ms4w8	Rotate 16 images 90° clockwise, 4 lines/pass	4MB images, 8 threads	Digital Ent. and OA
64M-x264-1worker	Encode 6 video streams (64MB total) to H.264	3 streams x 2 instances x 1 thread	New workload
64M-x264-2workers	Encode 6 video streams (64MB total) to H.264	3 streams x 2 instances x 2 threads	New workload
64M-x264-4workers	Encode 6 video streams (64MB total) to H.264	3 streams x 2 instances x 4 threads	New workload
64M-x264-8workers	Encode 6 video streams (64MB total) to H.264	3 streams x 2 instances x 8 threads	New workload

Table 1. EEMBC MultiBench 1.0 workloads and the existing EEMBC benchmark suites (if any) from which they were adapted. Notice that several workloads are based on the same work item but vary the datasets, the manner in which the datasets are manipulated, or the number of simultaneous threads supported. In EEMBC nomenclature, "64M" indicates the total size of the sample data manipulated by a workload (64MB), and "w2" indicates that two threads or "workers" do the processing.

Processor Core," in *MPR 7/7/08-01*, "Freescale's Multicore Makeover.")

Note that the MultiBench API supports symmetric multiprocessing (SMP), not asymmetric multiprocessing (AMP). Although this may be viewed as a limitation, it's not practical for the same benchmark tasks to support both styles of multiprocessing in a meaningful way. EEMBC says future versions of MultiBench may support AMP in some fashion. Until then, MultiBench 1.0 is geared toward homogeneous multicore processors that integrate multiple instances of the same CPU core, not heterogeneous designs that integrate multiple cores having different CPU architectures or microarchitectures. AMP benchmarks would definitely be desirable. EEMBC's domain is embedded processors, which (so far) are more likely to integrate diverse cores than PC or server processors do.

Multiple Methods for Composite Scores

With all MultiBench work items, the raw score is the number of iterations per second, so higher numbers are better. The same is true for the kernels in other EEMBC benchmark suites. For those suites, EEMBC also reports a composite score by calculating the geometric mean of the raw scores. EEMBC names those composite scores after their suites: ConsumerMark, TeleMark, AutoMark, and so on. While *MPR* was preparing this article, EEMBC was finishing the process of defining and naming a composite MultiBench score.

In fact, it looks like MultiBench will have *three* composite scores, all derived from geometric means of raw scores. These composite scores can describe raw throughput (workload iterations per second) and performance scaling (the degree of improvement in throughput when the workload runs on more threads or processors). Performance scaling will be of particular interest to engineers evaluating the efficiency of multicore processors in comparison with processors that are less well endowed.

One composite MultiBench score will be the SingleWorkerMark. It measures the throughput of workloads having only one work item and one worker (thread). Of the 36 workloads in MultiBench 1.0, eight workloads fit this description. One example is 64M-x264-1worker, which encodes six video streams in H.264 format. EEMBC will take the best throughput score (iterations per second) for each of the eight workloads, calculate the geometric mean, and multiply the result by ten. The multiplication factor is arbitrary—it merely makes the SingleWorkerMark score easier to read.

Another composite MultiBench score is the MultiWorkerMark. It measures the throughput of workloads having only one work item with multiple threads. Twenty of the 36 workloads in MultiBench 1.0 fit this description. One example is md5-32M4worker, which calculates Message Digest 5 (MD5) checksums on 32MB of data, using four threads. EEMBC will take the best throughput score for each of the 20 workloads in this subset, calculate the geometric mean, and multiply the result by ten. (Again, the multiplication factor is a convenience.)

The third composite MultiBench score is the MultiItemMark. It measures the throughput of workloads having two or more different work items, which may be single- or multithreaded. Because these workloads perform multiple related tasks, they are closer to real-world programs. Eleven of the 36 workloads in MultiBench 1.0 fit this description. One example is 64M-check-reassembly-tcp-h264, which simulates the task of compressing and sending six video streams over a network. It has four work items that check the packet headers, reassemble fragments of the Internet Protocol packets, perform Transmission Control Protocol (TCP) processing on the packets, and encode the six video streams into H.264 format.

Performance Scaling With MultiBench

To measure performance scaling, EEMBC starts with each composite throughput score: the SingleWorkerMark, MultiWorkerMark, and MultiItemMark. Then, EEMBC takes the geometric mean of workload throughputs when only one work item is enabled at a time. (This number should be greater than the geometric mean of workload throughputs with all work items enabled.) Finally, EEMBC divides the SingleWorkerMark, MultiWorkerMark, and MultiItemMark scores by those numbers. The results are the scale factors. If performance scales linearly, a scale factor rises in step with the additional number of processor cores.

For example, Table 2 shows the results of running MultiBench on two different dual-core processors clocked at 2.0GHz. By comparing the throughput scores of these processors, it's clear that CPU #1 is at least 50% faster than CPU #2. By comparing the scale factors with each processor's own throughput scores, we can see that CPU #1 is slightly better than CPU #2 at scaling its performance. Indeed, CPU #1 comes close to the ideal scale factor of 2.0 for a dual-core processor.

MultiBench Composite	CPU #1	CPU #2
Throughput Performance		
SingleWorkerMark	33.3	24.7
MultiWorkerMark	24.8	13.8
MultiItemMark	11.8	7.2
Dual-Core Scale Factor		
SingleWorkerMark	1.9	1.8
MultiWorkerMark	1.9	1.7
MultiItemMark	1.7	1.5

Table 2. MultiBench composite scores and multicore scale factors for two different dual-core processors. This comparison shows that CPU #1 is significantly faster than CPU #2 when running the MultiBench workloads in these three composite groups. In addition, CPU #1 does a better job of scaling performance when using both its cores. The ideal scale factor for a dual-core processor would be 2.0. (Data source: EEMBC.)

EEMBC is continuing its practice of allowing "out-of-the-box" and "optimized" scores, but the rules have been tightened. To obtain an out-of-the-box score, testers must compile the kernels without modifying the source code, use a publicly available compiler, and disclose the compiler-optimization flags applied. These rules are the same as before. To obtain an optimized or "full-fury" score, testers can modify the source code, but they must disclose the modifications and limit them to regions of source code approved by EEMBC. Previously, the rules didn't restrict the modifications to specific regions of code.

Off limits are regions of source code that verify data input and output. EEMBC wants to ensure that cheaters won't simplify the prescribed datasets or perform less work on the data than EEMBC requires. These rules and others are enforced by the EEMBC Technology Center, which checks and verifies the benchmark results submitted by EEMBC members. Only after the test results pass verification does EEMBC certify the scores. Certification is required before EEMBC members can openly publish scores. Without it, a member can privately disclose the scores to prospective customers under an NDA only. (Certification used to cost a few thousand dollars, but now it's free.)

As before, optimized scores can reflect almost any technique used in real-world software development. Testers can improve the C source code of a kernel, substitute assembly-language code for high-level C code, and use any hardware accelerators that may be part of the processor's design. For instance, some processors have hardware acceleration for video compression and decompression, which should markedly improve performance when running the H.264 kernels. One limitation is that the acceleration logic must be automatically invoked by the processor or callable through an API, because MultiBench 1.0 doesn't support AMP on heterogeneous processors.

Emphasis on Network and Image Processing

As Table 1 shows, the initial workloads that EEMBC chose for MultiBench 1.0 emphasize packet processing, still-image processing, and video processing. Absent are the somewhat mundane tasks found in the Automotive suite, which are also more typical of industrial applications. As with any industry consortium, there was a great deal of discussion in the technical subcommittees about which kernels to include or exclude in MultiBench. The initial emphasis is on higher-level tasks performed by sexier embedded systems, not the lower-level jobs carried out by machine controllers and other humdrum equipment.

There are several reasons for this emphasis. Sophisticated embedded systems are more likely to have multicore processors; many low-level tasks don't lend themselves to multiprocessing; higher-level tasks are becoming more widespread; and the 36 workloads in MultiBench 1.0 already generate a prodigious amount of data to analyze. In addition, EEMBC has a somewhat different objective for MultiBench. Whereas existing EEMBC suites measure performance within particular application domains, MultiBench measures the degree of performance scaling achievable with multiprocessing.

That said, the workloads in MultiBench 1.0 are representative of today's embedded software. Network connectivity, for instance, is becoming a must-have feature in all kinds of products. Packet processing is as important for a networked printer or a voice-over-IP (VoIP) phone as it is for a network router, so the packet-checking routines in MultiBench 1.0 are very relevant. Likewise, growing numbers of embedded systems manipulate still images or video streams. It makes sense for MultiBench to include work items for image rotation, color-space conversions, and H.264 video encoding.

During the selection process, EEMBC ported almost all the kernels in all the EEMBC suites to several different processors. The performance of some workloads scaled to a similar degree on all processors, whereas other workloads behaved quite differently. EEMBC strove for diversity, choosing some workloads from each group. EEMBC decided to limit the initial release of MultiBench to the 36 workloads in Table 1 because it's a manageable list, yet there's enough flexibility to create numerous workloads. Future releases of MultiBench could add most or all the remaining kernels and sort them into the familiar application categories. Meanwhile, all the ported workloads are available to EEMBC members, although they haven't been fully tested and aren't valid for MultiBench scoring and certification.

Less Benchmarketing, More Analysis

Even before EEMBC introduced MultiBench, the number of consortium members releasing certified EEMBC scores had dwindled over the years. It's a shame, because EEMBC benchmarks are far better than the most common substitute—Dhrystone mips. That benchmark was originally created to compare throughput with the DEC VAX 11/780, a late-1970s minicomputer. Measuring the speed of today's microprocessors in Dhrystone mips is like measuring the speed of F-22 fighter jets in furlongs per fortnight. Yet, even *MPR* still cites Dhrystone mips, because often it's the only metric available for prerelease processors, and it's slightly better than comparing clock speeds.

Behind the curtain, EEMBC benchmarks are more popular than they seem. EEMBC members that rarely or never post scores use the suites to test their new processor designs, evaluate compilers, and optimize their program code. Often, prospective customers ask for unreleased EEMBC scores under an NDA so they can make their own comparisons with competing products. Some EEMBC members test their competitors' processors and disclose the scores to customers privately. (EEMBC rules forbid members from publicizing scores that haven't been released by the processor vendor.) One might argue that benchmarketing is no less intense when it's less public. But the reality is that EEMBC is becoming less important as a public benchmark scoreboard and more important as a source for embedded test software. Nonmembers can license the benchmark code, too—EEMBC currently has more than 50 corporate licensees and more than 100 university licensees.

Even EEMBC acknowledges this trend, especially with MultiBench. The benchmarks will help consortium members analyze their new multicore designs and optimize the software that runs on them. To promote this idea, EEMBC is releasing some preliminary MultiBench results, without naming the processors. Table 2 is one example, and Figure 3 is another. The graph in Figure 3 measures performance scaling on a quad-core processor.

The MultiBench results in Figure 3 would be particularly useful for developers writing a multithreaded program or operating system for that processor. More threads aren't always better, at least in this example. Performance is probably impaired by memory bottlenecks and the overhead of synchronizing additional threads. The CPU architect might conclude that hardware multithreading is worth the additional transistors for two threads per core but is of little value beyond that. (Indeed, that is what Intel's CPU architects concluded when designing the new Atom processor; see *MPR 4/7/08-01*, "Intel's Tiny Atom.")

Figure 4 is another example based on preliminary MultiBench tests. It compares two different dual-core processors, again without naming the chips. Note that the graphs in Figures 3 and 4 plot the performance of only four workloads—a small slice of the voluminous data that running the whole MultiBench suite would generate.

Multiprocessing Almost Defies Benchmarking

Chip-level multiprocessing has become the primary path toward higher performance. To remain relevant, EEMBC had to develop multiprocessing benchmarks. It was a daunting task, especially at a time when thousands of programmers are only beginning to grasp the challenge of writing efficient software for multicore processors. Fortunately, EEMBC is a fairly large consortium with about 50 member companies, including the biggest names in the industry.

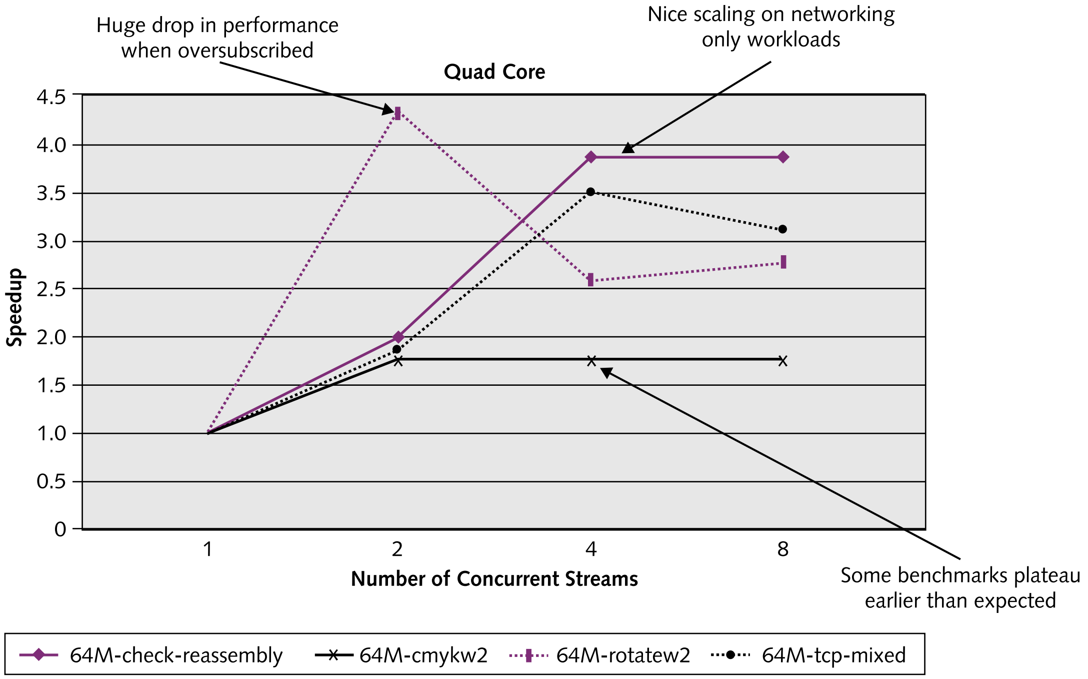


Figure 3. Preliminary MultiBench results on an anonymous quad-core processor. In this test, all four workloads enjoy better performance after doubling the number of concurrent threads. Thereafter, color-space conversion reaches a plateau and image rotation drops sharply, while the packet-processing workloads continue showing improvement up to four threads. (Data source: EEMBC.)

Although MultiBench 1.0 is the product of a committee process, it doesn't look like the usual committee mash-up. It's not perfect, but it's a great start.

We have already noted some limitations of MultiBench. Some of them will be addressed when EEMBC introduces the remaining work items and sorts them into suites representing embedded application domains. Finalizing the rules for composite scores will provide the at-a-glance performance summaries that are the hallmark of benchmarks—without ignoring the raw scores for individual workloads that tell the story in depth. Periodic updates to algorithms and datasets will keep the workloads in step with real-world software.

Other limitations will be harder, or perhaps impractical, to overcome. Right now, MultiBench 1.0 measures SMP

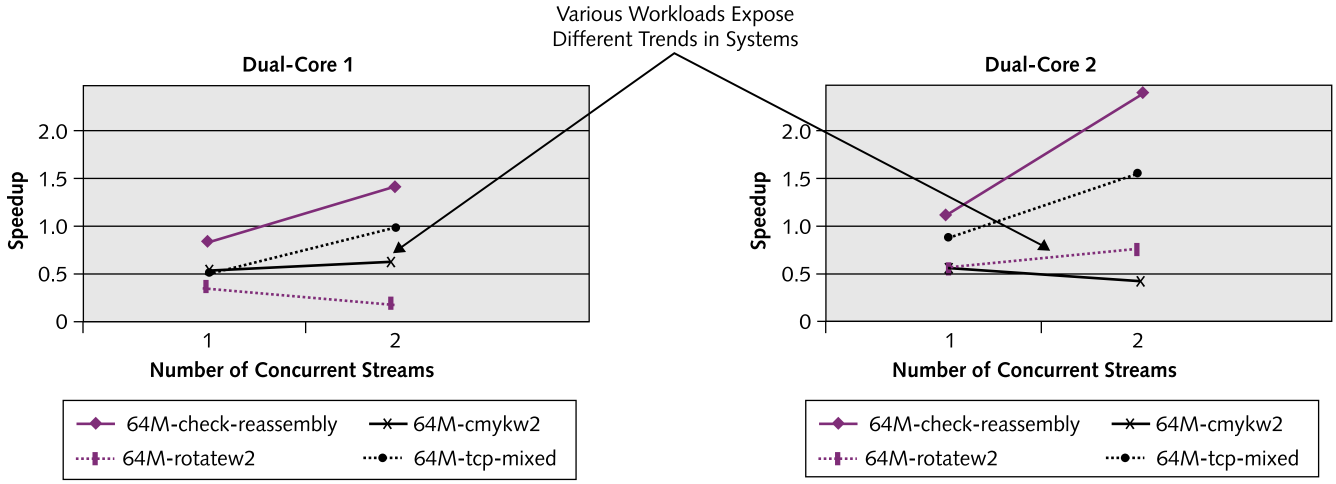


Figure 4. Preliminary MultiBench results comparing two anonymous dual-core processors. In this comparison, color-space conversion actually runs worse when distributed across both cores of one processor, while improving slightly on the other processor. The opposite is true for image rotation. (Data source: EEMBC.)

performance on homogeneous systems. Unfortunately, SMP's shared-memory model and programmer-explicit threading isn't very scalable to large numbers of processors. Massively parallel systems almost always adopt different programming models, sometimes layered on exotic CPU architectures. AMP is as valid as SMP, in some cases—perhaps in most cases, eventually. Heterogeneous multicore processors have numerous advantages, especially for embedded workloads, which tend to be more predictable than general-purpose workloads. There are so many apples and oranges to compare that multiprocessing almost defies straightforward benchmarking. And benchmarking is controversial by nature. (See *MPR 11/8/04-01*, "FPF '04 Benchmarking Panel.")

One intriguing avenue of exploration is the new Multicore Communications API (MCAPI), a project of the Multi-Core Association, another industry consortium. MCAPI is an attempt to create a standardized API for communication and synchronization among closely distributed cores or processors in embedded systems. The first version of the MCAPI specification is available now. Future versions of MultiBench could build on MCAPI to create AMP benchmarks for heterogeneous systems. There's already a tight connection between EEMBC and the MultiCore Association, because both were founded by and are headed by Markus Levy. (See *MPR 9/5/06-01*, "Multicore Multiplies the Challenges.")

Overall, *MPR* thinks MultiBench will prove its value as an independent analysis tool for CPU architects working on new designs and for programmers struggling to write software for those designs. In addition, we look forward to seeing the first MultiBench scores next year, and we encourage EEMBC members to publish scores. The results are sure to be surprising and interesting. ◇

Price & Availability

MultiBench 1.0 is available now to EEMBC members and licensees, including OEMs and educational institutions. Fees depend on the number of workloads licensed. For a single-user license, fees range from $2,000 for the office-automation workloads to $3,500 for the networking workloads. All the workloads can be licensed for $5,500. In addition, benchmarking requires the MultiBench test-harness software, which costs $1,500 for a single-user license. The MultiBench Architect tool, which includes the Workload Creator, is optional and costs $4,000.

EEMBC also sells site licenses for all these products. Groups of workloads range from $3,000 to $6,000 each, or $9,800 for the whole set. The test harness costs $2,500, and MultiBench Architect costs $6,000. Complete testing services are available from the EEMBC Technology Center for additional fees.

For information about EEMBC, visit *www.eembc.org*.

For information about the MultiCore Association, visit *www.multicore-association.org*.