ECL, LLC
6507 Jester Blvd 2222 Francisco Drive
Suite 511 Suite 510-203
Austin, Texas 78750 El Dorado Hills, California 95762
Inquiry@ebenchmarks.com http://www.ebenchmarks.com
512-219-0302

Dhrystone Benchmark

History, Analysis, "Scores" and Recommendations

White Paper

Alan R. Weiss

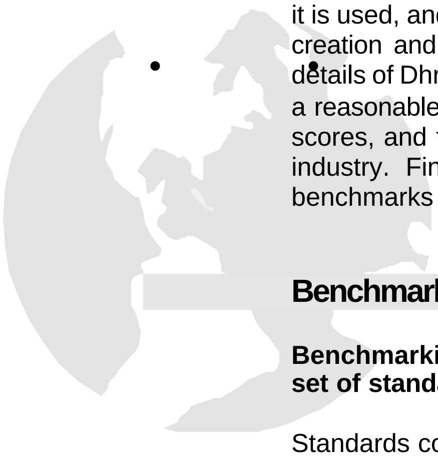
November 1, 2002

Dhrystone Benchmark

Introduction and Disclosure

The EEMBC Certification Laboratories, LLC (ECL) is recognized as the premier benchmarking and certification laboratory[1] in the semiconductor and software industries, and is the authorized certification body for EEMBC. EEMBC (pronounced "embassy") is the industry-standard processor benchmark consortium, and was setup to create reliable application-based benchmarks to measure processor (and compiler) performance.

Despite the growing adoption of EEMBC benchmarks, the Dhrystone benchmark is still misused in the industry. To help people and companies evaluate its usefulness, we decided to analyze Dhrystone for strengths and weaknesses and explain our findings based on real examples. This White Paper will first explain what "benchmarking" is, how it is used, and offer a set of intended uses. Then, we will explain Dhrystone, exploring its creation and evolution and intended purpose. From there, we dive into the technical details of Dhrystone, explaining how it works and what it measures. We then try and distill a reasonable set of run-rules consistent with its creator's intent, report some interesting scores, and then explore how Dhrystone is being used - and misused - by many in the industry. Finally, we compare and contrast Dhrystone with EEMBC's industry-standard benchmarks



Benchmarking: Definition and Purpose

Benchmarking is a method of measuring performance against a standard, or given set of standards.

Standards come about in two ways:

- Common usage over time (i.e. "the standard against one measures")
- Purposeful creation by one or more people

A useful way to characterize benchmarks is whether they are synthetic, or application ("real world") based. A synthetic benchmark is created with the intent to measure one or more features of a system, processor, or compiler. Synthetic benchmarks may try to mimic instruction mixes in real world applications, or they may be artificial. Synthetic benchmarks are useful in debugging specific features, but they cannot be easily related to how that feature will perform in an application. Because they are useful in debugging or isolating specific functionality, synthetic benchmarks tend to be small, though this is not a requirement.

[1] ECL defines ***Certification*** as the process of re-creating the benchmarking environment, verifying the processor and memory bus clock speed, verifying the compiler switches, re-creating the scores, re-building the code to ensure scores are re-creatable, and so on. ECL has over 50 separate steps in its benchmark score certification process.

Application benchmarks, also called "real world" benchmarks, use system- or user-level software code drawn from real algorithms or full applications Application benchmarks are more common in system-level benchmarking and usually have large code and data storage requirements.

A third type of benchmarks, called derived benchmarks (or "algorithm-based benchmarks") is a compromise between synthetic and application. As their name implies, derived benchmarks are created by extracting the key algorithms (software code) and generating realistic data sets from real world applications. This avoids the need to execute an entire application, and the benchmark can be used both for debugging, internal engineering, and for competitive analysis. Derived benchmarks, based on real application code, represent the best of both worlds and are perfectly suited for embedded environments.

What is Dhrystone? Definition, Historical Perspectives, Evolution

Dhrystone is an odd name to the uninitiated. Created in 1984 by Dr. Reinhold P. Weicker, then of Siemens AG, its intention was to measure the performance of computer systems, not embedded processors. Because of the nature of computer systems of that era, Weicker focused on integer performance. As the Whetstone benchmark for floating-point code already existed, Weicker chose the name Dhrystone as its logical counterpart in the integer world. The current version of Dhrystone, Version 2.1 was created in 1988, and remains in its original format today.

Weicker wrote Dhrystone to model what was then viewed as a "typical" application mix of mathematical and other operations. Integer performance predominated, with little or no floating-point calculations, and applications could be contained inside small memory subsystems. Throughout Dhrystone's long history, the benchmark has had to face the following revolutions and evolutions that have changed computer architectures. Unfortunately, Dhrystone doesn't take into account any of the following:

- Reduced Instruction Set Computing (RISC)
- Availability of sophisticated floating point processor units inside main processors
- Single instruction, multiple data (SIMD) vector processors inside the main processors
- Superscalar RISC designs (multiple execution units inside a single processor)
- Very Long Instruction Word (VLIW) processors
- Optimizing compilers
- Large memory subsystems, including processors and systems with L1, L2, and L3 caches
- Real-time operating systems with sophisticated application programming interfaces (API's), multitasking, TCP/IP functionality, and graphical user interfaces
- Large real-time, embedded applications proliferating into practically every area of modern life
- Graphics, multimedia, and communications-intensive applications

Throughout this white paper, it is very important to note Dr. Weicker's long-standing sentiment about his creation:

> *"Although the Dhrystone benchmark that I published in 1984 was useful at the time," said Weicker, "it cannot claim to be useful for modern workloads and CPUs because it is so short, it fits in on-chip caches, and fails to stress the memory system. Also, because it is so short and does not read from an input file, special compiler optimizations can benefit Dhrystone performance more than normal program performance. In embedded computing, EEMBC (www.eembc.org) is collecting larger real-life embedded-computing programs as the basis for benchmarks." Dr. Reinhold P. Weicker, Siemens AG, Vice Chairman of the Spec Open Systems Steering Committee.*
>
> *http://www.einsite.net/ednmag/index.asp?layout=article&articleId=CA46261&st EDN Magazine 10 / 28 / 1999*

Dr. Weicker has long ago gone on to bigger and better things. An important computer scientist, renowned in benchmarking and performance analysis, Weicker has been involved with the SPEC organization (http://www.specbench.org), recognizing the inherent weaknesses endemic with Dhrystone.

Technical Characteristics of Dhrystone

The following table provides a concise summary of Dhrystone's characteristics and corresponding strength or weakness:

Characteristic	*Strength and/Weakness*
Written in C language code	**Strength:** Allows code to be ported to a large number of platforms and architectures.
Very small size	**Strength:** An engineer can quickly master Dhrystone. **Weakness:** A compiler writer, or architect, can quickly defeat Dhrystone and "design to a benchmark." **Weakness:** Minimizes or eliminates stress on memory subsystems and easily fits inside L1 caches. **Weakness:** Cannot hope to mimic the breadth of applications encountered by a processor-based system. **Weakness:** Is based on a single benchmark comprised of three files: dhry_1.c, dhry_2.c, and dhry.h. There is only one set of functions.
Single, easy-to-report score	**Strength:** Reported as a single figure of merit, similar to the 'marks' used by EEMBC, has allowed it to gain industry

	traction. Dhrystone is formally reported as "Dhrystone 2.1 MIPS". **Weakness:** Dhrystone users employ confusing and ambiguous terminology such as DMIPS, DMIPS/MHz, Rounded Dhrystones/second, and Dhrystones/ CPU cycle. Furthermore, a "MIP" is actually 1.75 DEC VAX MIPS.
Synthetic	**Weakness:** Dhrystone only measures a few mathematical and basic operations.
Integer only code	**Strength:** This makes it potentially useful for simple 8- and 16-bit microcontrollers, assuming people don't care about relating anything to real world applications. **Weakness:** Does not measure multiply-accumulate, floating-point, SIMD, or any other type of operations.
Library-dependent performance	**Weakness**: Dhrystone's execution is largely spent in standard C library functions, such as `strcmp()`, `strcpy()`, and `memcpy()`. Compiler vendors generally provide these libraries that are typically optimized and hand-written in assembly language. While you may think you are benchmarking a processor, you are really benchmarking are the compiler writer's optimizations of the C library functions for a particular platform.
No Evolution	**Weakness:** Compiler writers have long ago determined Dhrystone's functionality. The secret to good benchmarks, as SPEC and EEMBC have shown, is to stay ahead of the compiler writers to ensure that the processor and system is benchmarked, not just the compiler.
No Third-Party Certification	**Weakness:** Dhrystone's lack of an official certification process (as defined in Footnote #1) has eliminated this benchmark's credibility. Certification can only come from inherent value, and there is very little value in Dhrystone to modern processors or compilers.
No Source Control	**Weakness:** Dhrystone is available from multiple sources, and while most companies attempt to use Weicker's original source, some servers have "gone dark" as the age of the Web increases. There is great potential that a company, or an individual,

	has modified the code to its advantage. Some companies report Dhrystone 1.1 scores - an even older version of the code.
No Standard Run Rules	**Weakness:**. Due to the lack of a standards organization, Dhrystone's original runtime rules have eroded into a state of confusion, thereby turning it into a performance measurement that is easily circumvented.
No Disclosure of Benchmark Environment	**Weakness:** The benchmarking environment, including processor and memory clock speed, compiler switches, and libraries, are not disclosed nor required.
Inlining or excessive compiler optimization destroys the benchmark	**Weakness**. Instructing the compiler to inline the code, greatly increasing the benchmark's susceptibility to code elimination, typically breaks Dhrystone's apocryphal "rules". The benchmark essentially vanishes and scores get unrealistically good.

Dhrystone Scores: Real World Examples

One of the most important defects in Dhrystone is that it is often unclear what version is being quoted. Furthermore, since there are no "disclosure rules" or independent certification of scores, companies and individuals are free to state, or not state, anything.

For example SuperH publishes a score for Dhrystone 1.1 running on its latest processor core, the SH-5. Dhrystone 1.1 has been obsolete for many years and Dhrystone 1.1 scores are not comparable to Dhrystone 2.1 scores (the current version). But because there is no industry-standard group to manage the process and rules, and ensure a common code base, there is no consistency between vendors. SuperH is moving quickly to supporting EEMBC.

Dhrystone Areas of Optimization – Distilled Run Rules

As Table 1 indicated, Dhrystone is subject to various weaknesses that companies exploit. ECL studied the Dhrystone "rules" as envisioned and published by Dr. Reinhold Weicker to determine if some representative companies have violated the rules.

1. You may not re-write or change the code inside the main "timing loop" – that is, the timed portion of the C source code must not be changed. Subtle changes can influence, sometimes dramatically, the scheduling of certain instructions. Compilers are pattern matchers – if you change the code so that your compiler can isolate Dhrystone's peculiar code pattern, the compiler can select a Dhrystone-optimized code template. Unfortunately, this has limited applicability to real application code.

2. No Inlining Allowed. While you can link in standard C libraries, and indeed you can inline those, you cannot globally inline the Dhrystone code. Unbeknownst to the programmer, some compilers may ignore a directive not to inline, potentially

making it necessary to dump code to assembly language to verify what has occurred.

3. Separate compilation. Dhrystone tried to mimic how real programs are written linking separate modules together. This reflects 1970's style "structured programming" techniques that are still used today. Dhrystone's two C source files and header file must not be combined and compiled as one step. That there are only two C source files means this is not a very difficult barrier for most compilers (and compiler writers, having extensively studied Dhrystone, don't find this terribly "real-world").

4. Because Dhrystone scores are so heavily dependent upon C language functions that copy and compare strings (called `strcmp()` and `strcpy()`), Dhrystone rules allow compilers or assembly programmers to optimize these functions. In fact, most "smart" compilers have these library functions written in assembly language. Another trick is to optimize `strcpy()` by making alignment and fixed-length assumptions for the input strings, but in no case can these functions be optimized in a Dhrystone-specific manner (such as assuming the content, positioning, or length of the strings). In point of fact, this "processor benchmark" can spend between 10% and 20% of its time in these functions!

5. You cannot use post-processing tools (after linkage) to optimize. These illegal optimizations typically fall under the heading of feedback-directed optimizations, and are particularly handy when used with an architecture that has branch prediction and speculative execution

To understand how different vendors use the Dhrystone benchmark ECL measured Dhrystone on cores from ARM Ltd and MIPS Technologies, two leaders in the embedded-processor industry. Both companies cooperated fully with ECL and provided all necessary tools and support.

Analysis of MIPS Code and Score

MIPS Technologies did not violate the Distilled Dhrystone Run Rules. MIPS Technologies did not inline the code, although the compiler does align the data along 64 bit boundaries.

ECL was able to re-create MIPS benchmark environment and obtain exactly the same scores. We hand inspected the source code MIPS provided us, and found that MIPS did not change the code inside the timing loop of Dhrystone, and in porting MIPS did nothing to alter the code.

MIPS used an interesting bit of magic they Gideon's Algorithm. For `strcmp()`, it reads the natural word and compares that to the first word in the string compare. If it gets a match, it assumes that there is a word alignment, and then it picks up the next word, and so on until there is a word that is not aligned along the natural word boundary. Then, of course, it must compare byte by byte. This saves a bit of time, naturally, and accounts for some of the performance differences between MIPS and ARM.

ECL measured Dhrystone on the 5kc core, a single-issue 64-bit processor with an integrated multiply-divide unit. The compiler schedules divisions for this unit and this results in some score improvement. To compare and distill the effect of this multiply-divide unit, we used two additional compilers to see if they supported this feature. The Wind River Diab Data C compiler emitted highly optimized code that showed the architecture to full advantage, including aligning data along natural word (64 bit) boundaries. Another MIPS compiler vendor was less aggressive, and this points out that compiler selection matters - sometimes as much as 10-40%! However, using Dhrystone as a benchmark for compilers is flawed - there simply aren't enough different kinds of instructions in Dhrystone, and as we have seen libraries matter a great deal more than they should.

The 5kc's inclusion of the multiply-divide unit also proves a point about comparing seemingly similar architectures. The ARM 1026-EJS and the MIPS 5kc at first blush appear very similar: both are single-issue machines with similar L1 cache sizes, and so on. The MIPS part is a 64-bit part, and the ARM part is a 32-bit part, but architecturally they are more similar than dissimilar. The multiply-divide, when utilized with suitable instructions from the instruction-set architecture, gives the 5kc a slight performance boost, and so does the effect of the 64-bit fetches.

Note, however, a nearly fatal flaw with using Dhrystone as an embedded benchmark: nowhere is code and data size documented and reported, and nowhere is there a disclosure about the number of gates (transistors, die-size area) required by the processor. In the embedded world, often memory is the most expensive part of a design. Memory requirements have a significant effect on system cost and power consumption.

5kc processor, **40 MHz, TSMC process**	***Column 2*** ***MIPS 32 Bit 5kc using 5kc binaries***	***Column 3*** ***MIPS 32 Bit 5kc using 4kc binaries***
Dhrystones per second	**92889.03451**	**87697.20141**
Dhrystones / MHz	**1.321666368**	**1.247794665**
Rounded Dhrystones / MHz	**1.37**	**1.25**

MIPS 5kc Dhrystone scores, 40 MHz TSMC process part

Table 2 shows a number of performance metrics based on Dhrystone. Column 1 consists of the typical Dhrystone metrics and other derived calculations. Most scores are reported as Dhrystone MIPS/megahertz (abbreviated as DMIPS/MHz) and/or as VAX Dhrystone MIPS (sometimes just called DMIPS).

The number of loops we ran Dhrystone through (in this case, 20,000) had little effect, indicating another Dhrystone weakness: ***it's small size allows it to easily fit inside small L1 caches, therefore, after a few thousand loops the score is constant and scales linearly for clock speed.*** When we increased the loops to 50,000, it had no effect - nor when we decreased it to 5000 loops.

Column 2 indicates that the Dhrystone code was compiled for 64 bits, which is the native word size of the MIPS 5kc. As can be seen by the resulting DMIPS score of 1374 and a rounded DMIPS/MHz score of 1.37, this option gave the best results. We believe that taking advantage of 64 bits had a significant effect on performance.

Column 3 indicates the effects of emitting 32-bit code for the 5kc, resulting in a noticeable negative effect on performance. We generated this by compiling for the MIPS Technologies' 32-bit 4kc core, a different processor, and running the ensuing binary on the 5kc. This procedure validated MIPS Technologies' claim of code compatibility between processors. Performance suffered noticeably with the `strcmp()` function for the 4KC, though, because comparing 32 bit words takes longer than comparing 64 bit words when the strings are the same size.

Analysis of ARM Code and Score

ARM did not violate the Distilled Dhrystone Run Rules. ECL investigated the ARM 1026EJ-S scores using a simulator, as a board with this processor on it was not available at the time of this study.

Previously, we had verified on the ARM and MIPS platforms that using an RTL simulator with Dhrystone was practical from a time perspective, since the benchmark easily fits inside small L1 caches. We were suspicious of everything (being a certification laboratory, it's an occupational hazard), so we investigated everything that went into the set-up of the environment.

ECL was able to re-create ARM's benchmark environment and obtain exactly the same scores. We hand inspected the source code ARM provided us, and found that ARM did not change the code inside the timing loop of Dhrystone, and in porting ARM did nothing to alter the code. ARM ran "out of the box", except for allowed modifications to the `printf()` functionality to accommodate its particular environment (MIPS Technologies did the same).

The ARM10 architecture is 32 bits, not 64 bits, and lacks a multiply-divide unit. A single-issue machine with cache sizes similar to the MIPS 5kc, the Dhrystone benchmark doesn't *measure* the performance, power, and code size attributes of any processor. Dhrystone score *reporting* does not require the inclusion of code and data sizes, and nowhere is there a disclosure about the number of gates (transistors, die-size area) used. Further, this processor includes Jazelle, ARM's Java execution functionality. Dhrystone, of course, doesn't exercise any of that functionality. How could it? When it was written, the word "Java" only meant an island of Indonesia, or a variety of coffee. The world has changed since Dhrystone was written - but it has not.

ECL had to manually calculate ARM's scores since the RTL simulator only produces a cycle count. After running the benchmark for 25 iterations, the system accumulated 11071 cycles. Similarly, after 24 iterations, the system accumulated 10649 cycles. Subtracting the two cycle counts, we derived that 422 cycles are used to run one Dhrystone loop. To calculate Dhrystone MIPS / MHz:

11071 - 10649 = 422 cycles

1 / (422 cycles * 0.001757) = 1.3487 DMIPS/MHz

The ARM processor has a 16-bit instruction mode called Thumb, which ECL did not try to use. Remember, 1 million instructions per second is not the same as a Dhrystone MIP - a VAX VMS 11/1750 actually ran at 1757 DMIPS, and hence the adjustment in the equation.

In Table 3, we round off and compare the results from the ARM 1026EJ-S and the MIPS Technologies 5Kc on that basis.

	MIPS 5KC	*ARM 1026-EJS*
Dhrystones / MHz	1.37	1.35

This data indicates that these processors, per MHz, are practically identical - if all you consider is Dhrystone. As we have seen, this comparison condemns you with a misguided view of the capabilities of each processor.

Dhrystone Fragility: How to Win At Dhrystone The Easy Way

Like all modern computer processor architectures, and the benchmarks that run on them, the concept of "architecture" includes "processor, memory, and compiler." But if you wanted to cheat or win at Dhrystone, here is how you would do it (and many people do exactly this!):

1. Run Dhrystone 1.1, or 2.0, instead of 2.1 - and don't mention the version number.

2. Build special purpose libraries, in assembler, that specifically targets the strings in Dhrystone.

3. Inline the code. If you wanted to be clever about this, have the compiler inline the code, even if you tell it not to. Talk about an "optimizing compiler!"

4. Build a processor that is very good at integer and string functions, has a small L1 cache, has no floating point, Java, SIMD, special-purpose hardware, peripherals.

5. Build a compiler that recognizes the Dhrystone source code (it hasn't changed since 1984, so that shouldn't be too difficult) and just emit the right answer - without doing all those nasty little loops.

A Solution is Found: EEMBC and its Benchmarks

In the 1990s, the new innovations in hardware architecture drove a need for benchmarks based on real applications. RISC became embedded. VLIW became more practical using advanced compiler technology. SIMD moved down from mainframe and supercomputer processors into microprocessors (i.e. MMX, SSE, AltiVec).

If your company is Cisco, 3COM, Nokia, Motorola, Visteon, Delphi, Lexmark, or another large buyer of embedded processors - how do you determine how much processing performance to purchase? If you buy too much performance, your design will run too hot and will consume too much energy (a key factor in portable consumer electronics is battery life), and will cost too much. If you buy too little performance, your design will flounder under stress and full loads, and have no headroom for expansion.

EEMBC, the Embedded Microprocessor Benchmark Consortium, was formed for a number of reasons. EEMBC's goal was to bring real-world, application-based benchmarks to the world. Since one application wouldn't be sufficient, the EEMBC 1.0 code specifies over 35 separate benchmark kernels, divided into 5 application spaces:

- Automotive/Industrial
- Consumer
- Networking
- Office Automation
- Telecommunications

The Mission Statement for EEMBC is:

"EEMBC will work collaboratively to develop a suite of performance benchmarks that will target key applications of embedded systems. These benchmarks will help provide customers an objective means of evaluating processors and controllers."

A problem can occur so long as there is no independent third-party certification, and a canonical benchmark score repository does not exist. EEMBC solves those problems. All scores are available on the EEMBC website, for free. Dhrystone has no such certification, and no canonical main repository of scores exists.

Currently EEMBC has over 180 scores available for free on its website, and has gained a huge amount of support over the years. In 2002 alone, ARC, ARM, Improv Systems, Infineon, Motorola, NEC, SuperH, Tensilica, and Toshiba have published certified scores, with more on the way. Each score has full-disclosure of the environment including the compiler and switch settings. ECL has re-created each score and passed a set of 50 checks to assure that the scores are trustworthy. Furthermore, each score has the associated code and data sizes.

Most importantly, EEMBC is an open process and consortium for members to resolve disputes, express opinions, vote, and include new benchmarks into new versions. It is not static - it moves as the industry moves.

EEMBC 2.0 adds:

- 8/16 Bit Microcontroller Benchmarks (an entire suite)

- Java Benchmarks (an entire suite)
- MP3
- MPEG-2 Decode and MPEG-2 Encode
- MPEG-4 Decode
- Voice Over Internet Protocol (VoIP)
- Additional Networking benchmarks
- Cryptography benchmarks
- Ghostscript

A comparison of EEMBC vs. Dhrystone shows the following:

Attribute	***EEMBC***	***Dhrystone***
Written in C language code	Yes - All code is in ANSI C, except the Java Benchmark Suite	Yes - but it is NOT in ANSI C
Very small size	No - both small and larger kernels and benchmarks - a mixture	Yes - tiny, two .C files
Single, easy-to-report score	Yes - there are aggregates such as AutoMark or TeleMark.	Yes - DMIPS
Multiple kernels / benchmarks	Yes - 35 in Version 1.0, another two-dozen coming in Version 2	No
Synthetic	No – based on application algorithms based	Yes - completely
Related to a real-world machine score	No - no need, since over 100 scores available	Yes - based on ancient VAX 11/750
Integer only code	No - mostly integer, but some floating point, and much that can be re-written for "full-fury" in assembler, SIMD, etc. Good mixture	Yes
Performance too dependent upon libraries	No - profiling suggests that good libraries help, but bad libraries do not hurt as much as with	Yes - terribly sensitive to string functions

	Dhrystone	
Evolution	Yes - EEMBC 1.0 in 1998, 1.1 in 2002, and soon EEMBC 2.0 in 2003	Not since 1988
Third-Party Certification	Yes - EEMBC Certification Laboratories, independent, non-biased, fair	No
Source Control	Yes - strong. All code is available to EEMBC members, backed up by ECL using CVS and source management. A "correct version" always exists	No
Standard Run Rules	Yes - extremely strict, but there is a Fully Fury (optimized) and an Out of the Box set (don't touch the code).	Yes - but open to interpretation, and the lack of certification means some companies can cheat
Disclosure Forms Required	Yes - and the disclosures themselves have to be certified	No
Significant Figure of Merit Varies	No	No
Repository of Official Scores	Yes: http://www.eembc.org	No. Past Usenet repositories are years out of date.
Inlining or Excessive Compiler Optimization destroys the benchmark	No	Yes
Full Fury Mode	Yes - allows any optimization as long as you get the right answer. Helps highlight peripheral performance, and what the theoretical maximum performance of a part would be like	No - Run Rules state no changing the source code

System Level Benchmarking in Embedded Applications

Interestingly, EEMBC has remained and will remain a small-kernel derived and larger applications benchmarking consortium. The members, comprised of semiconductor (processor) developers as well as compiler companies, find the small-kernel approach invaluable for isolating strengths and weaknesses, for high fidelity to their application focuses, flexibility, and usefulness in processor design. EEMBC has replaced SPEC as the benchmark suite useful in design of embedded processors. The larger, Version 2 applications will tax the higher end of the embedded space (32-bit and 64-bit, as well as VLIW and DSP processors) and will demand faster processors and better architectures.

However, there is a need in the embedded community for a system-level, platform-based set of benchmarks. These would be quite portable, running across operating systems, processor types and varieties, and focused on the very high end embedded applications from cellular phones to Personal Digital Assistants (PDA's) to set-top boxes and information appliances (internet appliances, thin clients, and so on). These are all clearly embedded, but they are systems - not simply processors. Desktop PC benchmarks might be made to work in this world, but they have historically been stuck in the x86 / PC rut (and only available on Microsoft operating systems, which is natural since over 90% of PC's are running Microsoft OS's). Moreover, they don't really measure to a fine enough grain the bandwidth issues associated with being wireless, wireline at 56KBps, and cable modem/DSL speeds.

ECL's solution is to work with processor, compiler, RTOS, system-level, and other vendors and manufacturers to create a new set of industry-standard, certified, widely accepted benchmarks for system-level, platform-based solutions. An announcement will be made shortly, and not surprisingly a number of current EEMBC members will join - but not renounce their membership in EEMBC. As they move up-market, generating more of the "solution", these vendors will need ways to benchmark themselves, while also benchmarking the individual components (which EEMBC excels at). Customers, moreover, will benefit by understanding the kind of performance they are buying to integrate, and then sell to end-users.

Conclusion and Recommendations

The economic incentive for ECL would be to find a way to certify Dhrystone scores, and to endorse the scores obtained on the MIPS and ARM platforms, respectively. Unfortunately, we can't do that and pretend that Dhrystone won't mislead a viewer of such a "certification". It no longer makes sense to use Dhrystone as a performance benchmark in the embedded space, or in the PC and server space.

The author of Dhrystone himself, Dr. Weicker, said it best when he said at an event ECL attended, "Dhrystone - why is anyone still using *that*?" The answer is that it has, historically, been the only thing available. With the advent of EEMBC, now over 5 years old and going stronger than ever, growing in popularity and importance, this is no longer true. It is time to do as Dr. Weicker suggests, and put Dhrystone to bed forever.

Both ARM, Ltd. and MIPS Technologies wanted this study to be completed, mostly to verify for themselves that they are doing Dhrystone right, but they are both moving smartly towards using EEMBC scores. Already ARM has published certified scores on the EEMBC website, and MIPS Technologies is in progress. SuperH, the heir to Hitachi for their high-end processors, has published EEMBC scores. Motorola found that not publishing scores was a recipe for disaster, and in the words of one Motorola manager:

> *"We completely underestimated how important EEMBC scores are. We should have done this a long time ago."*
>
> *Chuck Corley, Applications Manager, PowerPC*
>
> *Motorola, Inc.*

Bibliography

[1] Dhrystone Benchmark: Rationale for Version 2 and Measurement Rules published in SIGPLAN Notices 23,8 (Aug. 1988), 49-62]
[2] Understanding Variations in Dhrystone Performance, Reinhold P. Weicker, Siemens AG, AUT E 51, Erlangen, April 1989