Certified Performance Analysis for Embedded Systems Designers

GrinderBench™ for the Java Platform Micro Edition Java ME

WHITE PAPER

May 2003
Updated April 2006

Protagoras, the leading Greek Sophist, was quoted as saying, "Man is the measure of all things," by which he meant that there were no objective standards for judging matters. Most of us in the software and microprocessor industries would beg to differ. The performance of software is critical to users, and measuring that performance is difficult but vital.

This paper documents the work of a multi-company team working within the auspices of the Embedded Microprocessor Benchmark Consortium (EEMBC) to develop an industry-standard GrinderBench™ benchmark for products that contain implementations of Java™ Micro Edition (Java ME™) Profiles. This paper addresses the following questions:

- Why a Java ME benchmark?
- Why EEMBC?
- Who are the beneficiaries of these benchmarks?
- What's inside this benchmark?
- What's next?

Why a Java ME benchmark?

Java ME is the umbrella Java standard designed to cater to a wide variety of mobile devices and other Java-enabled products. Each device category has a specific Java Profile (API) associated with it. With the rise in popularity of Java equipped devices, including PDAs and mobile phones, Java ME has become ubiquitous. Most mobile phones built today are equipped with a Java ME-compatible Java implementation.

The need for a standard methodology for measuring Java ME performance has been evident for some time. Compared to the desktop environment where there are CPU cycles and memory to burn, Java implementations in devices such as mobile phones have a much more difficult job to do. While supporting both control code (including the user interface) and digital signal processing, mobile devices must also closely manage power consumption to prolong battery life. Thus, their memory configurations are more reminiscent of the early PC days. An entire Java implementation (Virtual Machine and Class Libraries) may have less than 256 Kbytes of memory at its disposal. In this environment, Java performance is a matter of more than academic interest. Here, performance in a small envelope counts.

Why EEMBC developed the industry-standard GrinderBench™ Java ME benchmark

When EEMBC began work on GrinderBench, none of the then-available Java benchmarks were usable for Java ME. Benchmarks such as SPECjvm98 and Caffeinemark were created for enterprise and desktop Java and thus assume Java APIs and capabilities (like floating point math) that are not available on the Java ME platform. Even three years after its first public release, GrinderBench remains unique in its focus on providing system designers with measurements of Java ME performance that are useful for making design choices, whether the end product is a cell phone or a set-top box. It is also exceptional in being the product of a democratic process in which a number of different vendors in the Java space were involved. Other benchmarks targeting Java ME have either been proprietary tools, or focused on a specific function, or oriented toward buying choices among consumers.

Who benefits from the GrinderBench benchmark?

Many different types of companies benefit from the ability to measure the performance of Java ME implementations with EEMBC GrinderBench. Processor and bytecode accelerator vendors can use the benchmark code itself to better understand the capabilities of their products. This in turn enables them to "tune up" their architectures. A similar benefit exists for the providers of JVMs. Because the EEMBC benchmarks test real applications in real platforms, an operating system is an important component. Tool vendors can use the benchmarks to ensure that they have provided sufficient software hooks to run a Java implementation efficiently.

Platform OEMs, such as mobile device manufacturers, also derive considerable benefit from these benchmarks. GrinderBench allows them to determine the performance bottlenecks of their platforms and understand how to improve the end user experience. Related to this is the benefit seen by the service providers. GrinderBench allows them to make apples-to-apples comparisons of the various platforms and processing components on which they will be offering their services. The content developers writing applications for these platforms also want to know their minimum and maximum Java performance capabilities so they can take advantage of these to the fullest.

What's inside the GrinderBench benchmarks?

EEMBC designed GrinderBench for the Connected Limited Device Configuration 1.0 (CLDC 1.0), which is the set of Java Classes that underlies that underlies the dominant Java ME MIDP profile. The GrinderBench development team also chose to focus on testing the execution of code delivered to a device as standard Java class files, this being the most reliable indication of a Java ME implementation's actual processing power.

Components of the EEMBC GrinderBench benchmark suite were chosen to meet the following criteria:

1. *Includes code representative of real-world CLDC 1.0 applications*
   The programs can be legitimately applied to real Java ME-based applications. For example, it's inconceivable to consider running

numerical simulations or large database manipulations on a mobile phone. By contrast, games are natural fit in this environment.

2. *Avoids synthetic benchmarks to whatever extent possible*
   Although synthetic tests may help focus on specific attributes of a platform, synthetic tests are typically not representative of the real world. If great care is not exercised, these tests can be so narrowly constructed so as to be unrealistic in their performance profile.
3. *Broadly exercises the Java API*
   The GrinderBench suite includes software that goes beyond a trivial rewrite of a C program to looks and behave like Java in terms of software style. Furthermore, the benchmarks exercise a significant part of the CLDC 1.0 API.
4. *Resistant to manipulation*
   The included tests are complex enough to be realistic and avoid characteristics that lend themselves to cheating by using selective optimization tricks.
5. *Composed of five component tests*
   A suite of five widely different component tests assures that the benchmark suite is broadly based and well balanced.

In the process of meeting these criteria, many submissions for inclusion in the benchmark suite did not urvive the scrutiny of the GrinderBench development team. It turned out that some of potential benchmarks spent more than 90% of their execution time in a few lines of code. Generally, this type of code is bad for a benchmark because it can tempt software vendors to create special versions of their JVMs that can detect these critical lines of code and replace the normal Java byte codes with highly optimized, possibly native, code.

The Chosen Few

After careful consideration and extensive profiling, EEMBC chose to include the following components in its GrinderBench benchmark suite:

- A PNG image decoder
- A chess game
- An XML parser
- A cryptographic package
- A benchmark that exercises thread switching and synchronization (ParallelBench)

Taking photographs with a mobile phone is becoming one of the most popular activities next to text messaging. PNG images are the de facto standard for storing and transmitting pictures in the mobile phone arena, and thus a PNG decoder is a natural fit for inclusion. It meets the relevance standard and, given an image of reasonable size, it is of sufficient code and execution complexity to make it a serious test.

Games are the most common class of application on consumer mobile devices today. Chess is a good choice because it is a game of significant complexity. Profiling demonstrated that it was not susceptible to easy manipulation. Further, the execution time could be adjusted through game play parameters.

XML, a standard for self-describing data files, is sweeping the entire industry from the enterprise server down to the mobile handset. Many future applications on mobile devices will need to support a subset of XML to communicate with web application services, and thus an XML parsing benchmark was developed. This benchmark, based on the public domain kXML parser, exercises both the DOM and SAX XML APIs. It does a SAX-based parse of an input source and accumulates counts of the various items it encounters. These statistics are then output for validation. It uses DOM to build a uniform XML tree of arbitrary width and depth and then optionally outputs this data structure for validation. It performs DOM-based search for a node having a particular name as well as a DOM-based search for any text nodes having a particular value.

Cryptographic functions are essential on mobile devices to achieve the security expected for applications such as bank and stock transactions. The benchmark includes a crypto test based on a public domain crypto suite.

All of the above benchmarks have the attribute of supporting output verification. Given a known, though still changeable, input data vector (an input image in the case of the PNG decoder for example), the benchmark generates an output vector that can be tested against a reference result. This helps prevent vendors from “baking in” expected results.

Time and Space

CLDC 1.0 provides a milliseconds-accurate timer. The benchmark components will use a self-timing mechanism. The benchmark is designed so that the amount of time spent in the test dwarfs any inaccuracy in the Java-based timers.

Memory address space considerations are also critically important. When looking at performance, it is vital to remember that execution speed almost always trades off against memory usage. In other words, if you are willing to use more memory, you can make your software run faster. A simple illustration is a just-in-time (JIT) compiler that turns compact byte code into less compact, but faster-to-execute, native code. Hence, EEMBC requires that peak memory reporting be done as part of the benchmark whenever possible. Despite our desire to require this in all situations, memory measurements are impractical on some platforms. Platforms supporting memory measurements include many of the PDAs. On the other hand, it is more difficult to measure memory utilization on a mobile phone.

Another memory space consideration is ensuring that the benchmark fits on devices with a very limited amount of memory. Many phones do not permit MIDlet files greater than 64K in size, for example. For that reason, the benchmark is structured so that the individual components can be downloaded (not just the entire multi-test benchmark). Also, the individual MIDlet sizes for these components will not exceed 64 Kbytes in size.

The Execution Environment

During the benchmark development process, the team realized that the CLDC 1.0 benchmark could not be used on a platform that many would like to see tested — mobile phones. The problem was that most phones are equipped with the MIDP Java ME environment and can only accept Java code packaged as MIDlets.

To accommodate this testing modality, the benchmark development team created two versions of the benchmark — one that was pure CLDC 1.0 and one that was the same except for having a MIDP 1.0 wrapper. Despite the MIDP 1.0 wrapper, the benchmark is still a CLDC 1.0 benchmark because it only tests the CLDC 1.0 API and not the MIDP API.

What's Next?

With the CLDC 1.0 benchmark complete, EEMBC is looking down the road to the next big challenge — creating a benchmark that tests the MIDP API. But in the meantime, EEMBC encourages all those reading this White Paper to ask providers of Java components (processors, bytecode accelerators, JVMs, operating systems, etc.) to publish their benchmark scores to help the industry analyze the performance differences. In this way, users can help EEMBC to create a competitive environment.