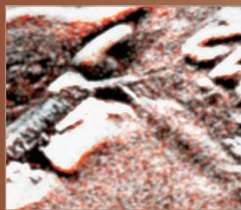


# Measuring Multicore Performance

Shay Gal-On and Markus Levy, EEMBC



Multicore benchmarking techniques must break the mold of traditional methods.

**A**s an astute reader of *Computer*, you probably don't need the detailed explanation of why most of the industry has so markedly shifted to multicore technology. But in all honesty, many embedded system designers are still struggling to determine whether multicore really buys them anything in terms of performance.

Resolving this quandary requires a thorough understanding of the target application, the characteristics of multicore processors that could be used, and the amount of time that must be invested to make the transition.

Having reliable performance information provides a good starting point for analyzing these factors, but "reliable" is the operative word. In other words, it's imperative to pick the right types of benchmarks to accurately predict the performance of the multicore processor once it's in the final product.

## MULTITUDES OF MULTICORE

Amazingly, many people—engineers included—think that a specific x86 manufacturer invented multicore. While the x86 has garnered the biggest spotlight, it only represents a fraction of multicore-enabled devices.

Focusing on the multicore processors with characteristics more or less similar to an x86, there's a plethora of general-purpose, shared-memory, symmetric multiprocessing (SMP)-featured products from companies such as ARM, Freescale Semiconductor, IBM, and MIPS Technologies.

Beyond this, countless vendors are building multicore products in the form of application-specific systems on a chip. SoCs can be as simple as a processor with a general computing core plus a digital signal processing core. They might have any number of cores, ranging in complexity from single-function hardware acceler-

ators to full-fledged processors.

Further, SoCs could be based on shared- or distributed-memory architectures. They could consist of either homogeneous or heterogeneous cores. And they could employ a variety of interconnect technologies.

Multicore technologies are highly differentiated, so multicore benchmarks need to be highly differentiating as well.

## TRADITIONAL BENCHMARKING METHODS

Before elaborating on different multicore benchmarking methods, it's useful to provide a brief historical perspective.

During the past two decades, benchmarks that only exercised a processor core's internal workings were sufficient. In fact, these benchmarks are still valid depending on the processor characteristics to be ascertained.

Most of the EEMBC first-generation benchmarks fall into this category. While still quite popular, they predominantly exercise the processor core and have little interaction with the external memory. For example, they test functions and features such as pipelines, branch prediction units, instruction sets, and caches.

Although these benchmarks can run on top of most operating systems, they're designed to run bare-metal. Performance is measured in iterations per second, where an iteration is the sequential execution of the benchmark kernel.

The compiler also plays a big role in this type of benchmarking. In fact, we've seen as much as 70 percent performance difference depending on the compiler used to generate the benchmark results.

EEMBC's second-generation benchmarks go a step further, providing significantly larger code and datasets to ensure that even the most robust memory and cache hierarchies are tested.

Due to the popularity of and familiarity with the first- and

second-generation benchmarks, many entities—including processor vendors, system developers, and academic research groups—want to continue using them to measure multicore processor performance.

In theory, multiple instantiations of each benchmark can be launched simultaneously. Some might recognize this method as similar to a SPECrate, which measures a system's capacity for processing jobs of a specified type in a given amount of time. The Standard Performance Evaluation Corporation notes that this “metric is used the same for multi-processor systems and for uni-processors” ([www.spec.org/spec/glossary](http://www.spec.org/spec/glossary)).

In fact, even in a multicore system, it isn't possible to guarantee that the system is utilizing more than one core unless it's employing some form of processor affinity. In other words, without programmer intervention, the platform's scheduler will assume control over execution of the individual benchmark instantiations.

The question is whether sequential code really works for this purpose.

### MULTICORE BENCHMARK CRITERIA

To answer this question, it's necessary to understand the important multicore performance characteristics. At the highest level, benchmarks for multicore architectures should be either computationally or memory intensive, or some combination of both.

#### Memory bandwidth

Regardless of the type of multicore architecture, memory bandwidth is a key factor in performance. A multicore processor's memory bandwidth, as with any other processor, depends on the memory subsystem's design. In turn, the memory subsystem depends on the underlying multicore architecture.

Shared memory, typically accessed through a bus and controlled by some type of locking mechanism to avoid simultaneous access by multiple cores, provides a straightforward programming model because each processor can directly access the memory. Typically associated with homogeneous multicore systems, shared memory also facilitates programming with traditional languages because it allows passing data by reference, without actually moving the data.

**At the highest level, benchmarks for multicore architectures should be either computationally or memory intensive, or some combination of both.**

For cores with individual caches, there must be a coherency mechanism between the caches. The ease of use of this architecture can lead to performance bottlenecks due to competition between multiple cores accessing the same memory locations.

In a distributed memory system, which is more common within an SoC, each processor can access its own local memory but doesn't have to share it with other cores, even though there might also be a global memory address space across them.

When one core requires that data from another core or cores must synchronize, the system must physically move data or the control code must switch to run on a different core. Even though each core has its own local memory, there might still be memory bottlenecks depending on how data moves on or off the chip itself.

#### Scalability

Another important benchmark criterion is scalability, in which the processor incurs performance pen-

alties when it oversubscribes computing resources.

In familiar terms, assume that an application program consists of a varying number of threads—it's not unreasonable to have hundreds of threads in a relatively complex program. If the number of threads exactly matches the number of processor cores, performance could scale linearly assuming no limitations on memory bandwidth.

However, realistically the number of threads will exceed the number of cores, and performance will depend on other factors such as cache utilization, memory and I/O bandwidth, intercore communications, OS scheduling support, and synchronization efficiency.

#### Example

So does sequential code work for benchmarking multicore processors? The answer is yes with respect to the cumulative throughput of each individual core. In this case, memory bandwidth and computation can be evaluated.

In fact, we're aware of at least one example of this. A telecom equipment manufacturer transitioned its application from a multiprocessor to a multicore system. Merely switching from a system in which each processor had its own memory subsystem to one in which the two cores shared the memory subsystem resulted in a critical performance reduction, regardless of how much inherent parallelism the code possessed.

This is the type of information we must be able to derive using benchmarks before going through the massive porting effort required to switch to a new platform.

### SMP-BASED MULTICORE BENCHMARKS

Executing multiple copies of sequential code doesn't account for one of the most important potential benefits of multicore: using parallelism to improve the performance of individual tasks rather than

improving overall throughput. One example, assuming a dual-core processor, would be using both cores to load one web page twice as fast. Alternatively, you could parallelize the work by using both cores to load two web pages (one per core) in the same amount of time it would take to load one.

Accomplishing that would require benchmarks that utilize task decomposition, functional decomposition, or data decomposition. These methods could more comprehensively exercise all of the major multicore benchmark criteria. Of course, the devil's in the details.

Further, for a benchmark to be relevant for multiple cores and produce comparable results, it must be able to execute the same amount of work regardless of the number of contexts used, and it must be able to show the performance improvement (or degradation) that results from the number of contexts used. The benchmark must be able to utilize any number of computation contexts because it will be used across many different platforms.

### MULTIBENCH

Primarily addressing the embedded market, EEMBC has implemented MultiBench, an extensive suite of multicore benchmarks that utilizes an API abstraction to more easily support SMP architectures ([www.eembc.org/press/pressrelease/223001\\_M30\\_EEMBC.pdf](http://www.eembc.org/press/pressrelease/223001_M30_EEMBC.pdf)).

Given the variety of architectures in the embedded industry, flexibility is key. Hence, when porting to a new platform, only 13 API calls are needed to allow the framework and all benchmarks to run and to exploit parallel execution.

Moreover, many systems support the Portable Operating System Interface (Posix) threads interface, and EEMBC carefully chose the API abstraction such that there is a direct mapping to the more complex Pthreads inter-

face. This means that if the system already supports Pthreads, no porting is necessary.

The multicore benchmarks are delivered as a set of workloads, each comprising one or more work items. Although it's easier said than done, users can select from this list the workloads that most closely resemble their application.

### BEYOND SMP

It's important to note that the MultiBench tests are oriented toward general-purpose processors based on an SMP architecture. Although MultiBench is significantly more parallel than the first- and second-generation benchmarks, it still doesn't support the heterogeneous cores found in SoCs. This calls for an entirely different benchmarking strategy.

**Heterogeneous cores  
require careful analysis and  
workload partitioning.**

Unlike the SMP benchmarks, which can essentially consist of orthogonal threads, heterogeneous cores require careful analysis and workload partitioning. Benchmarking these devices in a relevant manner with portable code is a huge challenge because each part of the system might use a different tool chain, and communication between different parts of the system isn't standardized.

There are several options for benchmarking such systems. Extending the existing MultiBench framework to support heterogeneous systems requires some standard to allow the benchmarks to be portable. For example, the Multicore Communications API from the Multicore Association ([www.multicore-association.org/home.php](http://www.multicore-association.org/home.php)) provides a standardized framework for partitioning benchmark

code into blocks that use MCAPI to communicate.

### APPLICATION-SPECIFIC STANDARD BENCHMARKS

While the demand for SMP-based benchmarks and for those that support heterogeneous cores is growing almost exponentially, a move to *application-specific standard benchmarks* is afoot. Also known as scenario-oriented benchmarks, ASSBs are designed to take into account more system-level features.

### Black-box benchmarks

We view ASSBs as black-box benchmarks. From the simplest perspective, they specify the input, expected output, and interface points. In other words, it doesn't matter what's inside the system as long as the benchmark can handle the input and deliver the expected output.

Unlike complete off-the-shelf systems, many embedded systems are tested using evaluation boards or similar reproduction platforms. The problem is that developers can tinker with these platforms in subtle ways to indicate better performance than a system will achieve as a commercial product.

For example, suppose you were evaluating the performance of an SoC running the Session Initiation Protocol. SIP is used to set up and tear down multimedia communication sessions such as voice and video calls over the Internet.

In normal operation, a commercial product should be able to receive and handle packet streams consisting of mostly valid, but some invalid, packets. However, in an evaluation platform, a performance advantage could hypothetically be obtained by leaving out the code that processes invalid packets. Therefore, the ASSB must also inject and test for bad packets.

### Challenges

Although ASSBs have been described as the way forward in

computer benchmarking (S.M. Pieper et al., "A New Era of Performance Evaluation," *Computer*, Sept. 2007, pp. 23-30), they don't come without challenges.

First, creating an industry standard, at least within the EEMBC domain, requires a consensus among the members to select the right mix of scenarios to test from among an almost infinite number of possibilities.

Second, getting an ASSB to run properly on an evaluation platform requires the assembly of many system-level components, including both hardware and software. In many cases, the company running the ASSB might not have all the components at its disposal. For example, a processor vendor could have all the supporting hardware to run a SIP benchmark but be

missing various components of the software stack.

Another challenge of ASSBs lies in analyzing the results. While an embedded platform is the sum of its components, running an ASSB makes it difficult to isolate any particular component. In other words, assuming that an ASSB requires the hardware (SoC, memory system, I/O) and the software (OS, application, runtime stacks), finding bottlenecks can be as difficult as agreeing on a performance metric.

**R**egardless of the architecture, measuring multicore performance requires new ways of benchmarking. Coming up with the benchmarks is only half of the challenge; the other half is interpreting the results. Of course, that's where the fun really begins. ■

*Shay Gal-On is director of software engineering for EEMBC and leader of the EEMBC Technology Center. Contact him through [www.eembc.org/contact](http://www.eembc.org/contact).*

*Markus Levy is president of EEMBC and the Multicore Association, as well as chairman of the Multicore Expo. Contact him through [www.eembc.org/contact](http://www.eembc.org/contact).*

**Editor: Tom Conte, College of Computing, Georgia Institute of Technology; [conte@cc.gatech.edu](mailto:conte@cc.gatech.edu).**

# CALL FOR PARTICIPATION



**CTS 2009**



## The 2009 International Symposium on Collaboration Technologies and Systems

**May 18 – 22, 2009**

**The Westin Baltimore Washington International Airport Hotel  
Baltimore, Maryland, USA**

<http://cisedu.us/cis/cts/09/main/callForPapers.jsp>



In cooperation with the ACM, IEEE, IFIP