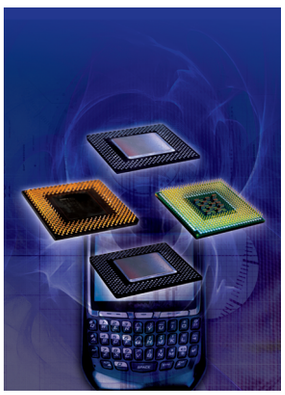
EMBEDDED MULTICORE PROCESSORS AND SYSTEMS



Markus Levy
EEMBC and Multicore Association

Thomas M. Conte
Georgia Institute of Technology

At the highest level, embedded multicore can be defined as a technology, a methodology, and a business and research opportunity. Building a multicore-enabled embedded system requires system developers to leverage a combination of these factors. Furthermore, unlike desktop PC or server applications, multicore devices used in embedded systems are as diverse as the stars in the universe. Well, maybe not quite that diverse, but the point is that there are many potential approaches to solving the numerous multicore-related issues. The articles in this special issue of *IEEE Micro* on embedded multicore processors and systems represent a small sampling of these issues.

Embedded Multicore: The Technology

From the technology perspective, embedded multicore is represented by many different devices and architectures. These processors include dual-, quad-, and eight-core processors based on symmetric multiprocessing or shared-cache architectures. They also include manycore processors that implement a form of shared memory, distributed memory, or a combination of the two. Although the x86 is probably the most popular architecture represented here, it's really just the tip of the iceberg for processors targeted at the embedded industry.

Other examples in this arena include processors such as the Freescale 8641D (dual core), the ARM Cortex-A9 MPCore (supporting 2 to 4 cores), Plurality's Hypercore processor (capable of supporting 16 to 256 cores), and Tilera's TilePro64 processor (64 cores). One of the articles in this special issue, Thomas Berg's article, ''Maintaining I/O Data Coherence in Embedded Multicore Systems,'' is centered on another of these processors—the MIPS32 1004K. This processor helps solve the communication problem of how to pass data between the I/O peripheral devices and on-chip CPUs. Solutions can use software, hardware, or hybrid mechanisms. Specifically, the article discusses techniques for maintaining I/O coherence and presents the tradeoffs of each approach.

Regardless of the number of cores, processor designers must resolve the bottlenecks associated with shared memory and other shared system resources. System designers are actually experiencing performance drops when moving their single-core embedded multiprocessor designs to a multicore processor. Extensive efforts are underway to help understand and resolve these performance-related issues, which also include issues associated with oversubscription, synchronization overhead, and intercore communications.[1]

Embedded multicore technology also includes systems on chip (SoCs) with an almost unlimited combination of homogeneous and heterogeneous processors designed to tackle specific applications. Many SoCs are proprietary and deeply embedded in end-user devices. However, SoC products such as Cavium Network's Octeon CN38XX, Freescale's QorIQ P4080, and Texas Instrument's OMAP3503 are commercially

0272-1732/09/$25.00 © 2009 IEEE Published by the IEEE Computer Society

available for a wide variety of embedded applications.

Regarding the intercore communications performance aspect of multicore, a significant amount of effort is going toward optimizing the networks-on-chip (NoCs) technology used in the design of multicore devices. Vendors (Silistix and Sonics) and many research projects are devoted to this glue that holds systems together. As a matter of fact, quite a few articles on this technology were submitted for this issue. We found the article "Double-Data-Rate, Wave-Pipelined Interconnect for Asynchronous NoCs," by Jiang Xu, Wayne Wolf, and Wei Zhang, particularly interesting because it considers many of the practical aspects of NoC design. In addition to throughput, the article also discusses power consumption and silicon area costs.

Embedded Multicore: The Methodology

All multicore technologies aim to either exploit concurrency, increase compute density, handle partitioned workloads, or achieve some combination of these objectives. And regardless of the intended goals, most of the embedded industry has been challenged (as have the PC and server industries) with developing new approaches and tools to let system developers optimize their multicore-targeted programs. In general, the industry is approaching these optimal solutions from the perspective of starting from scratch—or in an attempt to migrate decades' worth of legacy applications to multicore technology.

CriticalBlue and PolyCore Software offer interesting examples of this approach to new methodologies. CriticalBlue recently released Prism, a tool that lets software engineers take their existing sequential code and, without changing it, explore and analyze opportunities for concurrency, implement parallel structures, and verify efficient and safe operation. Prism works from simulated execution traces and can identify performance bottlenecks caused by long periods of serialization or multiple threads of activity waiting for a resource to become unlocked. Additionally, Prism considers scenarios with any number of cores, so it could also check how the performance scales as more cores are added to the system.

Jean-Yves Mignolet and colleagues describe a similar methodology in their article, "MPA: Parallelizing an Application onto a Multicore Platform Made Easy." Relying on programmer input provided through a separate file, the MPSoC parallelization assist (MPA) tool performs the partitioning, inserting communication and synchronization as required to respect the dependencies in the sequential application. It also helps to verify that the parallel code is correct by construction, avoiding common problems with parallel programming models such as race conditions, deadlock, livelock, and starvation.

Approaching the problem from a different perspective, PolyCore Software's Poly-Mapper tool lets multicore application developers create, configure, and reconfigure multicore communications topologies, allowing them to focus on partitioning and distributing the application across multiple cores. The topology manager uses a graphical interface to help developers lay out the nodes and links, after which Poly-Mapper builds and validates the topology and generates an XML-based topology map.

Another PolyCore tool, Poly-Generator, processes the topology map and generates an optimized C-based topology definition. Poly-Messenger, a runtime library, is an interprocessor-communication framework that abstracts application software from the underlying interconnect networks, operating systems, and network topology (bus, star, mesh, or hybrid). The PolyCore platform thus supports portability and reuse of an organization's software investment. Poly-Messenger provides an implementation of the Multicore Association's Multicore Communications API, which is the subject of another article in this issue—"Software Standards for the Multicore Era," by Jim Holt and colleagues. MCAPI lets application software developers program to one API that is implemented by real-time operating systems and middleware software across multiple multicore architectures.

Multicore: The Business and Research Opportunity

As is true for any industry, the embedded industry has its share of vendors who are exploiting the multicore buzz and simply putting a new package on an old product and referring to it as "multicore ready." However, new multicore technologies and

methodologies are providing industry and research organizations with many interesting opportunities. For example, the companies we've mentioned here represent a mixture of well-established and start-up companies that are, in fact, concocting novel approaches to help propagate the adoption of multicore technologies. Furthermore, the articles in this special issue represent a fraction of the different perspectives and approaches that are becoming available. Other important multicore-related topics include load balancing, operating system support for symmetric and asymmetric multiprocessing, debugging issues, resource management, and hypervisors and virtual machine monitors. But rest assured that these topics and more are critical to the evolution of embedded multicore technology. MICRO

Reference

1. S. Gal-On and M. Levy, ''Measuring Multicore Performance,'' *Computer,* Nov. 2008, pp. 99-102.

Markus Levy is president of the Embedded Microprocessor Benchmark Consortium and the Multicore Association, and chair of Multicore Expo. His research interests include embeddded processor performance and energy analysis and standards to support multicore system development. Levy has a BS in electrical engineering from San Francisco State University. He is the coauthor of *Designing with Flash Memory* (Annabooks, 1993) and has several patents related to flash memory architecture.

Thomas M. Conte is a professor in the College of Computing at Georgia Institute of Technology. His research is in the areas of manycore/multicore architectures, microprocessor architectures, compiler code generation, architectural performance evaluation, and embedded computer systems. Conte has a PhD in electrical engineering from the University of Illinois at Urbana, Champaign. He is an associate editor of *ACM Transactions on Embedded Computer Systems, ACM Transactions on Architecture and Compiler Optimization, Computer*, and *IEEE Micro*.

Direct questions and comments about this article to Markus Levy, 4354 Town Center Blvd. #114-200, El Dorado Hills, CA, 95762; markus.levy@eembc.org.