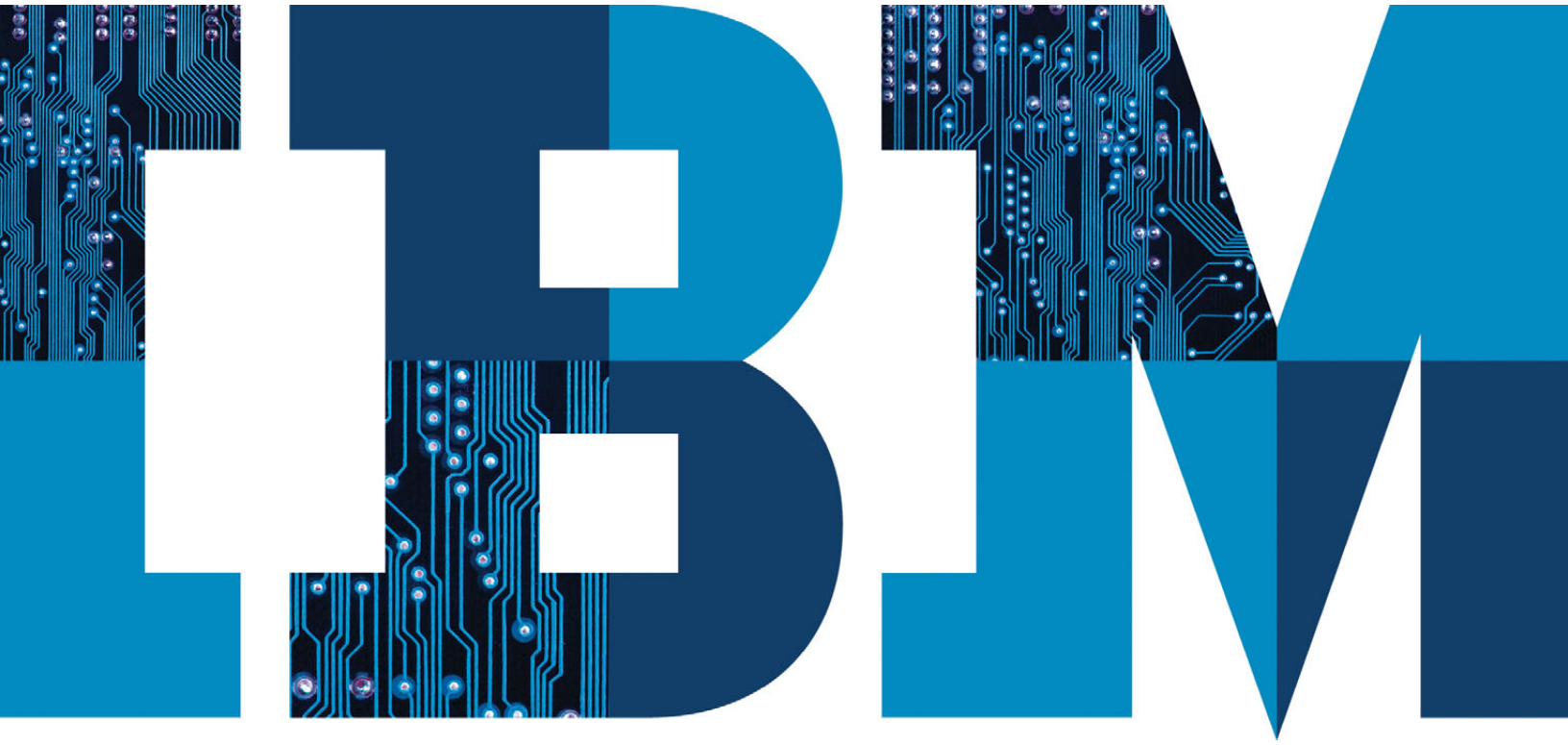
Cutting-edge multicore development techniques for the next wave of electronics products



IBM

Introduction

Miniaturization continues to drive the electronics industry. Device form factors and their underlying components are shrinking while capabilities and performance are increasing.

Devices are also becoming elements of a highly interconnected ecosystem as a result of added communications links. Electronics companies are adopting multicore processors, a processing system composed of two or more independent cores,[1] to deliver these advances in their next-generation products.

As multicore processors become more pervasive in electronics products—PCs, mobile phones, gaming systems, network equipment, industrial control systems, and now even medical devices—the demands of developing software for these systems has become a primary concern in the electronics industry. The advantages of multicore architectures are many, such as higher performance, lower power consumption, lower cost, and more flexibility, but can be realized only if the corresponding software is developed to unlock these benefits. Many software developers in today's electronics industry lack the skills to write software optimized for multicore. Furthermore, multicore architectures can become exponentially complex as the number of cores increases from two to four to thirty-two different cores; that is, traditional means of development no longer scale. The only way to handle the complexity is through automation and with a higher-level language that allows software deployment to be automated to different cores. Since time to market is a critical differentiator in the electronics industry, meeting market deadlines means utilizing an organization's existing software code. This is mostly single-core source code, however, and not optimized for multicore technology.

This IBM® Rational® white paper discusses cutting-edge software development methods that enable your development teams to take advantage of new multicore technology, as well as an automated way to reuse single-core software for multicore systems, enabling you to both outperform the industry and drastically save time and money through the reuse of existing systems.

Promise of multicore

The tide has turned in the drive to increase performance and reduce power consumption and cost in computing platforms used for building today's advanced electronic products. Traditionally processors were pushed to deliver more performance by increasing the number of transistors on a die and increasing the clock rate, ever challenging Moore's law.[2] This will continue to happen, but the path to true leaps in performance and power management lies with innovation around multicore technologies.

Multicore-based systems deliver reduced power consumption and typically operate at slower clock speeds, which can significantly improve battery life and reduce the amount of heat production. Cooler operating temperatures mean that organizations can use quieter and more power-efficient fanless cooling systems. Products with smaller form factors are possible, reducing the number of distinct processors in favor of a reduced number of multicore processors. Further, multicore operating environments hold the promise of true multitasking and much better application performance than is possible with single-core-based environments, which can reach a CPU utilization threshold quickly. True parallelism is now possible when handling an increasing load of compute-intensive applications. Additionally, load balancing and separation of system functionality with multiple cores can lead to improved system robustness and security.

From personal computers and workstations to high-end smart phones and portable devices, consumer demand for ever-improving performance, battery life and advanced features has led device manufacturers to build and deliver products that use multicore technologies. The next generation of personal electronics and smarter products will rely on multicore environments to deliver even greater connectivity, responsiveness, usability and productivity-yielding applications to an increasingly technology-savvy clientele. Software will be the critical factor in delivering on the promise of multicore, and success or failure clearly rests on the embedded software team.

Challenges with multicore

The promise of multicore technologies comes at a significant cost and risk to today's product development community. Multicore environments introduce even greater complexity into the design and delivery of products and the software-intensive applications that drive them. Choices must be made about the distribution of resources and functionality, affecting the product architecture. For system designers, decisions about the actual number of processors, number of cores required, operating system (symmetric or asymmetric) characteristics and required middleware are more difficult. On the software front, designers must take into account new issues about application partitioning, intercore and intertask communication, and scalability of the design in a multicore-based system. The features of the operating system(s) and the potential middleware in use further the complexity of the overall software design.

Organizations moving to multicore will require changes in the skill set of their software design teams. Parallelism must become the norm, and the way in which their applications are built must be altered to take full advantage of the parallel architecture that is available in a multicore operating environment. Considerations about increased use of multiple tasks, intertask communication mechanisms and task-to-core allocation all become critical factors in successful application development. Reuse increases in importance for tasks and components of the software-intensive systems.

Existing applications must be refactored to run on the new multicore architecture to take advantage of the underlying performance improvements of the parallel architecture and, more important, to ensure that they are able to function properly. What ran correctly and safely on a single-core system, even in a multitasking operating system, might exhibit erroneous and even unsafe behavior in the parallel environment offered by a multicore system.

Debug and test become even more critical when deploying software to a multicore system. Traditional code-level debug and late-cycle testing techniques often do not scale to multicore environments. New techniques and test environments are needed to ensure that software is operating correctly and providing the required level of functionality.

IBM Rational methods for developing for multicore

Three methods are recommended for effective multicore development:

- Perform trade-off studies to assess alternatives
- Use existing software to accelerate development
- Automate software generation to improve quality

Perform trade-off studies to assess alternatives

Simply taking an existing system and moving it to a multicore processor can actually cause it to run at the same speed or even slower.[3] This is because the actual cores typically are slower on an individual basis than on the original processor. This is especially true because shared data can slow down the

communication. The whole application might even run at the speed of a single core. Performing trade-off studies is critical to assessing alternatives and ensuring that the desired improvements in performance, speed and resource utilization will be realized.

Modeling your product's software architecture and creating different options for it to run on multiple cores is critical to a successful implementation. Existing tasks are mapped to the cores and additional tasks are added, if needed. Further, modeling enables you to gain an understanding of how to optimize communications between the tasks and processes that your applications involve.

Another key method is to simulate the model you built to verify that everything works as expected. Without such simulation, all you have is an expectation that the system works.

As you make your trade-offs, the model enables you to document the reasons why you made specific decisions. Another important aspect is to map your design to the original requirements that were agreed upon. Understanding the real-time aspects of the requirements helps you to make decisions about what tasks should map to each core. For example, you might not want the user interface of your product's software on the same cores as the real-time or critical control components.

Trade-off studies are important for understanding the options of what part of your software should go on which cores. Without it, you simply cannot assess the best alternatives and you cannot be sure whether your application is running optimally on the cores.

Use existing software to accelerate development

Your existing product might be running successfully with key application software already. It runs well in the existing single-core operating environment and it might even function properly in the new multicore environment. You want to retain its value by reusing as much as you can, as you work to ensure that it will perform better and scale to meet the demands in the new environment.

Development for multicore anchored by a model-driven approach allows for significant reuse opportunities for your existing software. From the simple visualization of the structure and relationships of the software and its component parts to refactoring the code to optimize for deployment to a multicore environment, to integration and extension into a completely new design and application architecture, the right model-driven tooling can greatly accelerate your ability to reuse the software you have come to rely on.

Visualization of your existing application software yields immediate gains by increasing your understanding of how the software has been architected and how the individual components are related. It enables you to identify the optimal and suboptimal aspects of your design and generate documentation of the current design to aid in the analysis of how best to use the code in the new design, even without having to change it. This visualized code can be targeted at the new multicore environment and you can perform trade-off studies to determine how best to proceed with the application, either using it as is or refactoring it.

If you decide that the existing software does not perform as well in the new environment or is poorly architected for the multicore environment, refactoring is a valuable option. By importing the code as model elements, it can be refactored and regenerated to support a much better application architecture that is tailored for multicore. Modeling and automated generation of the code make refactoring existing code a cost-effective alternative to rewriting the application.

Finally, existing software components can be used in new designs and applications that are built for the products that target a multicore environment. Again, visualization, refactoring and automated generation capabilities in a strong model-driven development tool greatly enhance your ability to make these reuse decisions. New applications can be written and delivered for products, and limit the need to design from scratch. Further, the components being reused have a track record of success in earlier products so their behavioral aspects are understood and tested, limiting the new test-and-debug effort to their performance as part of the new application.

In moving to new hardware, reuse of legacy code is always critical because it is too time consuming to write everything from scratch. Additionally, having code that has already been tested is critical for an on-time project completion. Graphically viewing the reusable code helps you understand how it will fit with the functionality you are adding as you switch to a new multicore hardware platform.

Automate software generation to improve quality

Because multicore inherently provides a highly parallel environment, communications between tasks within an application and between applications becomes a critical factor in the successful deployment of these applications. Tasks must communicate, in most situations with different mechanisms as they switch from being on the same core to different cores. This heavy reliance on efficient communications further increases the importance of having good interfaces between the tasks and using a modeling environment to enable automatic generation of the supporting mechanisms.

Fully understanding, specifying and exploring alternatives involving the switch to target different hardware and communication protocols necessitates the use of models and specifically automatic code generation. This allows for the visualization of design and systematic replication of this design in the generated code. When moving a task from one core to another, the modeled change can then be regenerated and reflected in code that is specific for the new core as needed, even if it uses a different operating system or has to use a different protocol to communicate to tasks on the original core. Typically this would require hand-coding the change, but with a modeled application this change would occur by simple assignment of the options with the automatic generation capabilities of a model-driven development tool taking care of the rest of conversion. In every phase of development, modeling and code generation are critical to helping developers test and deploy their multicore applications. This is especially true in time-sensitive markets such as the electronics industry.

Conclusion

In its quest to bring innovative and differentiated products to market in shorter time frames, the electronics industry as a whole will increasingly adopt multicore processing in order to reap the benefits of higher performance, lower power consumption, longer battery life, lower cost and increased flexibility. But these benefits cannot be realized unless the corresponding software is architected for multicore. Several cutting-edge methods can be used to simplify and accelerate software development for multicore. By using model-driven development, software design teams can perform multicore design trade-off studies, visualize and refactor code leading to greater reuse, and generate code automatically depending on the hardware configuration and communication protocols in use.

For more information

To learn more about multicore development techniques, contact your IBM representative or business partner or visit: **ibm.com**/software/rational/info/multicore/

Additionally, financing solutions from IBM Global Financing can enable effective cash management, protection from technology obsolescence, improved total cost of ownership and return on investment. Also, our Global Asset Recovery Services help address environmental concerns with new, more energy-efficient solutions. For more information on IBM Global Financing, visit: **ibm.com**/financing

© Copyright IBM Corporation 2010

IBM Corporation
Software Group
Route 100
Somers, NY 10589 U.S.A.

Produced in the United States of America
March 2010
All Rights Reserved

IBM, the IBM logo, ibm.com, and Rational are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. If these and other IBM trademarked terms are marked on their first occurrence in this information with a trademark symbol (® or ™), these symbols indicate U.S. registered or common law trademarks owned by IBM at the time this information was published. Such trademarks may also be registered or common law trademarks in other countries. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at **ibm.com**/legal/copytrade.shtml

Other company, product, or service names may be trademarks or service marks of others.

References in this publication to IBM products or services do not imply that IBM intends to make them available in all countries in which IBM operates.

The information contained in this documentation is provided for informational purposes only. While efforts were made to verify the completeness and accuracy of the information contained in this documentation, it is provided "as is" without warranty of any kind, express or implied. In addition, this information is based on IBM's current product plans and strategy, which are subject to change by IBM without notice. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, this documentation or any other documentation. Nothing contained in this documentation is intended to, nor shall have the effect of, creating any warranties or representations from IBM (or its suppliers or licensors), or altering the terms and conditions of the applicable license agreement governing the use of IBM software.

[1] http://en.wikipedia.org/wiki/Multicore_processor

[2] http://en.wikipedia.org/wiki/Moore's_law

[3] http://www.forbes.com/2009/11/23/google-microsoft-programming-technology-cio-network-multicore-hardware.html

Please Recycle

RAW14209-USEN-00