Enhancing software development
To support your business objectives

Rational. software

Model driven development—simplifying multicore systems deployment.

Highlights

- ***Allows developers to visualize and implement parallelism to verify expected performance gains***
- ***Automates execution of systems models to speed trade-off studies***
- ***Simplifies mapping of application components from existing legacy systems to multicore devices***

Adopting multicore technology can accelerate any green initiative. In addition to being cost-effective, multicore systems have a smaller carbon footprint and potentially lower greenhouse gas emissions than single-core systems. Their reduced power consumption and slower clocking speeds can significantly improve battery life, which translates to fewer disposals into landfills. Further, the cooler operating temperatures mean that organizations can use quieter and more power-efficient fan-less cooling systems.

However, implementing multicore systems is not a simple case of breaking and redeploying the single-core application onto multiple processors. Requirements and architectural analysis gain paramount importance when developing for a parallel architecture. Abstract-level models facilitate architectural and design studies that would otherwise take much longer to perform in source code. In addition, the run-time platform has to support true parallelism as opposed to the concurrency achieved through commonly used sequential multitasking.

Unique challenges of multicore software development

Creating software for multicore devices opens a unique set of challenges. These challenges often require novel approaches throughout the development cycle, from initial requirements gathering to final testing and validation of the parallel system. While architects and developers customarily work with concurrent applications, true parallelism in multicore systems mandates rethinking task design, inter-task messaging and load distribution, as well as parallel testing to validate the completed systems. These milestones may be unchartered territory for single-core developers.

With conventional development, concurrency or parallelism is not finalized until later in the development cycle, during the optimization stage. Such a delay in finalization is logical because concurrency in single-core systems is implemented by the multitasking runtime platform. However, with multicore systems, parallelism is not a logical abstraction but a physical reality. Therefore, this architectural difference must be considered early in the development cycle because it can affect both the application and platform architectures.

For example, consider a company developing a multicore handheld device to stream video and receive satellite radio simultaneously. The first steps are deciding how many cores to specify for the product and selecting the clock speed for each core. Assume two 32-bit cores are required, running at 1GHz each. Additionally, two spare cores are needed for offloading peak usage. Validating the minimum, average and peak usage scenarios plays a vital role in the development cycle. Performing a use case analysis could reveal spare cores or excess clock speeds. Both factors can affect the total product cost. More importantly, decisions about how to deal with these factors can also affect operational quality issues, such as the useful life and environmental impact of the device.

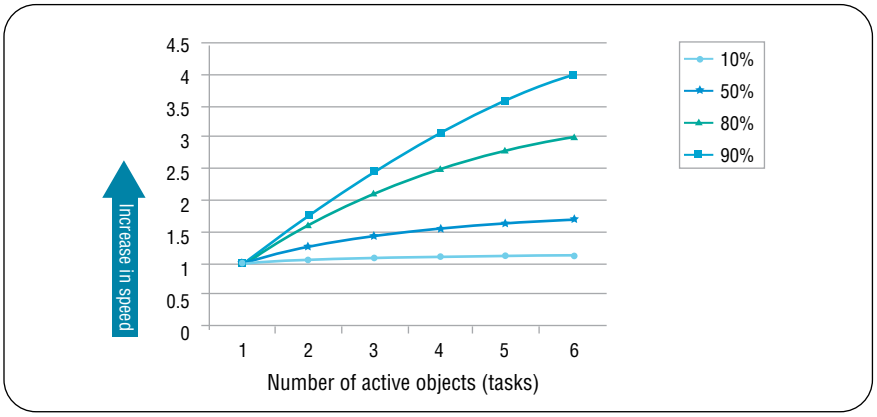

Figure 1: Applying Amdahl's law to predict the extent of speed gain as a function of parallel threads in a system

Simplify multicore development using models

Model driven development solutions from IBM can help organizations migrate successfully to multicore systems. By enabling a higher level of system abstraction, models allow developers to visualize and implement parallelism.

Helping to achieve predicted performance gains

One of the major motivations for deployment of multicore technology is to improve the underlying performance of the server and client machines. Most single-core applications today, including embedded and real-time systems, are implemented as concurrent tasks. Once the logically parallel tasks are executed on a physically parallel platform, significant performance gains can result (see figure 1).

In a model-based system, a collection of active objects represents these system tasks. Amdahl's law predicts that in a system with six active objects, a four-fold performance gain can be achieved when 90 percent of the objects are parallelized. An example of a model-defined concurrent system is provided in figure 2 using a UML class diagram. The diagram is used for automated code generation.

When the source code for multicore systems is generated from a model driven approach, it is possible to arbitrarily assign tasks to different cores and execute the system virtually to achieve the performance gains predicted by Amdahl's law. This quick form of trade-off study becomes a vital component in the multicore development workflow.

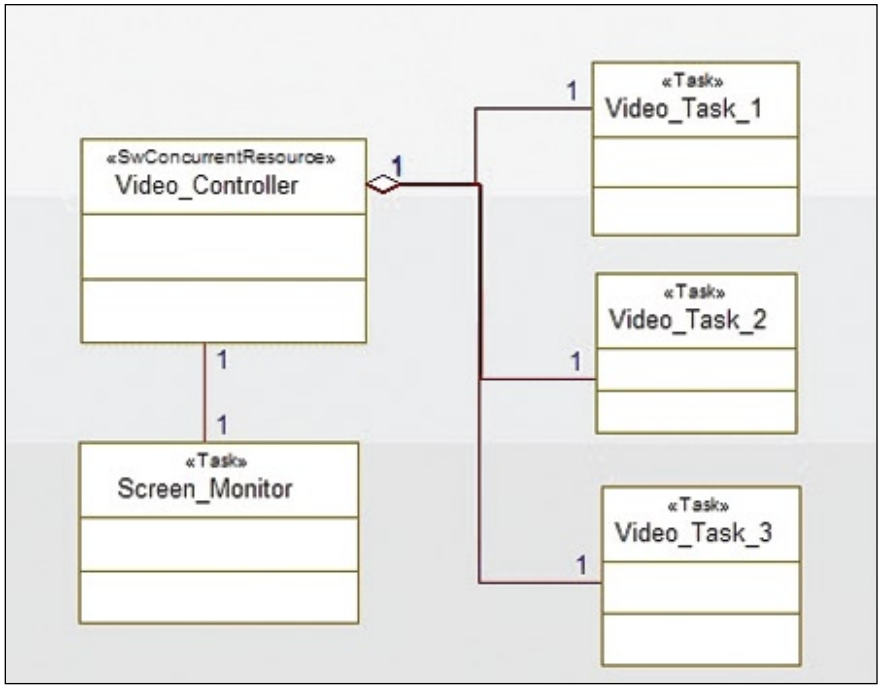

Figure 2: Concurrent task structure of a video player modeled in IBM Rational Rhapsody® software

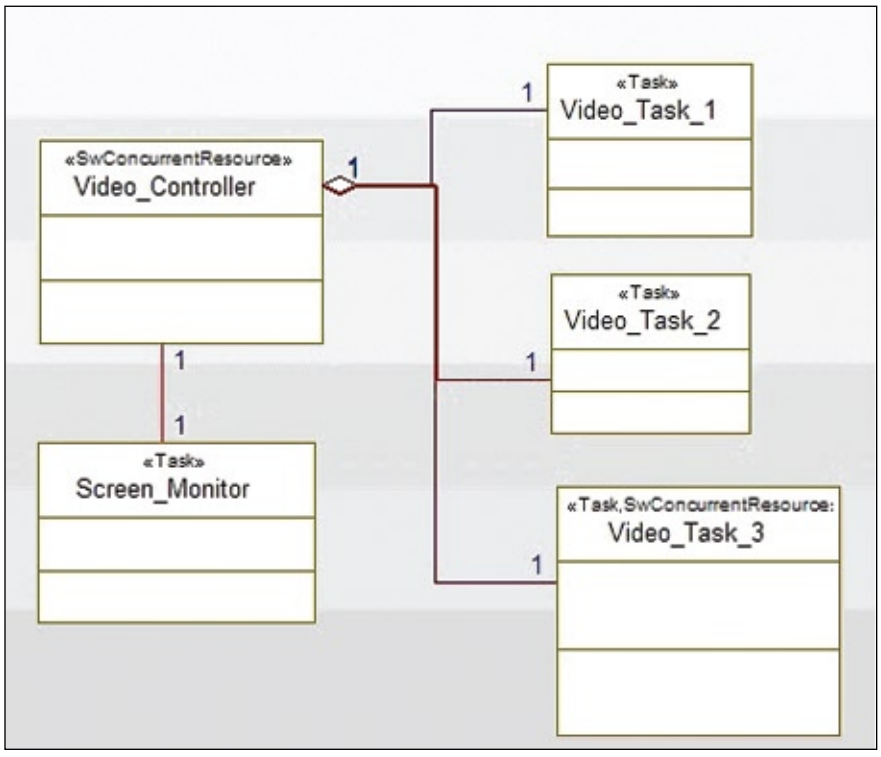

Figure 3: A legacy, single-core system redeployed on a multicore platform

Automating tradeoff analysis

A model-based trade-off study is only possible when the underlying modeling tool is capable of executing the modeled system. To illustrate this, the architecture and behavioral semantics of the system shown in figure 2, once captured, must be able to compile and execute before generating the runtime data needed for trade-off study. Therefore, model execution capability or automation provided by the modeling tool is critical for realizing the full value of using a model driven approach.

Evolving legacy systems with ease

Rarely does a new project start with a clean canvas. In case of an organization standardized on a model driven approach, a new project typically borrows components from existing models. If modeled components are largely platform independent, as is the case with unified modeling language (UML) 2.0–based components, mapping existing components to a multicore platform is a trivial task. In figure 3, the original legacy system is redeployed on a multicore system by simply specifying "Video_Task_3" to run on a parallel platform, leaving intact the overall functionality of the system. In other words, core awareness can be introduced into legacy systems without drastically altering the overall architecture and the design of the application, and, at the same time, clearly capturing the new architecture. Through a simple change in the annotation of the class, followed by studying the execution of the complete system as a simulation, the refactored model can be quickly verified and incrementally evolve into a multicore system.

Enabling successful multicore development

Successfully migrating to multicore development requires modification of the software development workflow. Physical parallelism provided by the hardware platform affects all levels of the underlying system. Through modeling, parallelism can be easily abstracted, making it easy to refactor a legacy, single-core application into its multicore counterpart. At the same time, through model execution and subsequent trade-off study, it is also possible to optimize core usage.

For more information

To learn more about how the IBM Rational® solution for model driven development can simplify multicore systems deployment, contact your IBM representative or IBM Business Partner, or visit:

ibm.com/software/awdtools/rhapsody/

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

Produced in the United States of America
October 2009

RAS14039-USEN-00