Improving Java applications on multicore architectures

Mots clés :
- Directeur de thèse : gilles MULLER
- Co-encadrant(s) :
- Unité de recherche : Laboratoire d'informatique de Paris 6
- École doctorale : École Doctorale Informatique, Télécommunications, Électronique de Paris
- Domaine scientifique principal: Divers

Résumé du projet de recherche (Langue 1)
Multicore hardware is today a reality in all kind of computing architectures ranging from powerful servers to desktop environments and even embedded systems. Nevertheless, current systems and applications are unable to fully exploit these new architectures: performance is stalling, even though the number of cores is increasing. Taking advantage of multicore hardware is thus one of the most important scientific challenges in the systems domain. Furthermore, addressing this challenge has a crucial economic impact, as highly multicore machines will soon be cheaper and more energy efficient than a cluster of machines with one or only a few cores, but only if the highly multicore machines can be used effectively. Nevertheless, legacy Java applications are notoriously ill-adapted to multicore architectures. The only concurrency abstraction provided by Java is the synchronized block, which encourages the use of coarse-grained synchronization. Applications cannot be fine tuned for execution on a specific multicore configuration, taking into account, e.g., cache behavior, because such features are hidden by the Java Virtual Machine (JVM). Finally, the training and experience of Java developers is typically more oriented towards aspects of high-level software structuring, and less towards low-level synchronization issues. Nevertheless, legacy Java applications cannot be ignored. The volume of such code and the trend of providing more processing power by increasing the number of cores implies that it is essential to enable legacy Java applications to run efficiently on multicore hardware. The problem of inadequate support for concurrency has previously been considered in the context of systems software, such as web servers implemented in languages such as C and C++. In this context, the source of inefficiency has been found to be the unpredictability of the operating system process scheduler in its management of concurrency. Because this scheduler is unaware of the pattern of shared data between threads, it may preempt a thread while the thread is holding a lock on shared data, thus effectively holding up all of the other threads that access the data, and it may place threads that share data on different cores, causing cache misses as the data migrates from one core to the other. These problems are exacerbated in the case of coarse-grained parallelism, where the degree of concurrency is already reduced. Coarse-grained parallelism was typically found in the considered web servers, and is typically found in legacy Java programs, for ease of programming. In the context of C and C++ web servers, the use of an event-based programming model was found to provide a significant performance improvement. This model lifts the problem of scheduling from the operating system to the application level. Specifically, an application is designed as a set of handlers reacting to events. Scheduling of events is entirely controlled by an application-level event engine. This provides predictability, because the event engine controls when, if at all, handler execution is preempted, and flexibility, because the event engine can control the mapping of handlers to cores, to achieve any data locality effect that is desired. Nevertheless, despite the advantages of the event-based programming model, it has so far not seen wide use, due to the difficulty of manually restructuring legacy code to use events.

Résumé du projet de recherche (Langue 2)
In the context of this PhD, we want to investigate how to improve the performance of legacy Java applications on multicore architectures using the event-based model. In contrast to C and C++ web servers, which run directly on the hardware, Java code runs on top of a virtual machine. Our key observation is then that events can be introduced at the virtual machine level, with no modification to the legacy application. Because this proposal will recognize the beginning of a synchronized block as the start of an event, and the body of the synchronized block as the event handler. The event engine in the Java Virtual Machine can then ensure that blocks that synchronize on the same value are executed on the same core and without preemption, ensuring that there is no conflict in their access to shared data. The event engine can thus schedule the associated event handlers. As a result of this PhD, we hope to improve the performance of legacy Java applications fully automatically on multicore architectures.
Deep understanding of language-based virtual machines, such as Java. Knowledge of OSes is a plus.