Software Architectures for Flexible Task-Oriented Program Execution on Multicore Systems

Thomas Rauber and Gudula Rünger

Abstract. The article addresses the challenges of software development for current and future parallel hardware architectures which will be dominated by multicore and manycore architectures in the near future. This will have the following effects: In several years desktop computers will provide many computing resources with more than 100 cores per processor. Using these multicore processors for cluster systems will create systems with thousands of cores and a deep memory hierarchy. A new generation of programming methodologies is needed for all software products to efficiently exploit the tremendous parallelism of these hardware platforms.

The article aims at the development of a parallel programming methodology exploiting a two-level task-based view of application software for the effective use of large multicore or cluster systems. Task-based programming models with static or dynamic task creation are discussed and suitable software architectures for designing such systems are presented.

1 Introduction

Future cluster and multicore systems will soon offer ubiquitous parallelism not only for high-performance computing but for all software developers and application areas. However, the mainstream programming model is still sequential and von Neumann oriented, and sophisticated programming techniques are required to access the parallel resources available. Therefore, a change in programming and software development is imperative for making the capabilities of the new architectures available to programmers and users of all kinds of software systems.

Thomas Rauber
University Bayreuth
e-mail: rauber@uni-bayreuth.de

Gudula Rünger
Chemnitz University of Technology
e-mail: ruenger@informatik.tu-chemnitz.de

The article aims at the software development for future parallel systems with a large number of compute units (cores). Although many parallel programming models, including MPI, OpenMP, and Pthreads, have been developed in the past (especially targeting HPC and scientific computing), none of them seems to be appropriate for main stream software development for large-scale complex industrial systems since the level of abstraction provided is too low-level [22]. The trend in hardware technology towards large multicore systems requires a higher level of abstraction for software development to reach productivity and scalability. To provide such programming abstractions for parallel executions is a key challenge of future software development.

In this article, we propose a parallel programming methodology exploiting task-based views of software products, so that future multicore and cluster systems can be used efficiently and productively without requiring too much effort by the programmer. The main goal is to deliver a hybrid, flexible and abstract parallel programming model at a high level of abstraction (in contrast to low-level state-of-the-art models like MPI or Pthreads) and a corresponding programming environment. A main feature of the approach is a decoupling of the specification of a parallel algorithm from the actual execution of the parallel work units identified by the specification on a given parallel system. This allows the programmer to concentrate on the characteristics of his algorithm and relieves him from low-level details of the parallel execution. In particular, we propose to extend the standard model of a task-based execution to multi-threaded tasks that can be executed by multiple cores and in parallel with other multi-threaded tasks of the same application program.

The use of an appropriate specification mechanism allows the expression of algorithms from different application areas on an abstract level and relieves the programmer from an explicit mapping of computations to processors as well as from the specification of explicit data exchanges. This facilitates the development of parallel programs significantly and, thus, makes parallel computing resources available to a large community of software developers. A coordination language provides support for the specification of tasks and for expressing dependencies between tasks.

In the following sections, we present an overview of a new approach for task-oriented programming in Section 2, a description of the corresponding software architectures for task execution in Section 3, and a runtime evaluation in Section 4. Section 5 discusses related work and Section 6 concludes.

2 Programming Models with Tasks

In this section, we give a short overview of task-based programming, which we propose to use for multicore systems with a large number of cores.

Task-based programming is based on a decomposition of the application program into tasks that can cooperate with each other. Task creation can be determined statically at compile time or at program start, but can also evolve dynamically during program execution. An important property is that, for an arbitrary execution platform, task creation is separated from task execution, i.e., tasks do not need to