Foreword

Parallel programming is the mainstream. From laptops to the largest supercomputers in the world, and from cell phones to high-end medical devices, modern computing systems incorporate architectural parallelism in order to provide high performance for their applications. When multicore computers became the norm, parallel computing also became available to almost everyone with a laptop. Yet in order to benefit from this architectural parallelism, application programs must typically be adapted to express their inherent concurrency. Whether creating a new parallel application or introducing parallelism into an existing one, today's application developer may choose a new, high-level parallel language, exploit a parallel extension of an existing sequential language, or make use of library calls to meet this need. Never before has there been so much active development of parallel languages and experimentation with fresh approaches to obtaining high performance, portability, and performance portability across parallel platforms.

Much has been written about parallel programming language features, the algorithms that exploit them, and the experiences of application developers who deploy them. Less has been written to explain how they are implemented and what goes into the creation of a high-quality compiler and runtime system for a parallel programming interface. A compiler that translates programs that are created using a high-level parallel programming language requires the support of a runtime system that manages the compute resources during execution, assigning work to them, implementing the necessary synchronization, and more. Since the adoption of a high-level programming interface depends to a large extent on the quality of the user experience, each of these must be well designed and their interactions carefully engineered.

Developing a runtime system that provides consistent, high performance to applications that are executed on a multicore platform is about very much more than meeting basic implementation needs. Rather, the components of a runtime system must be highly efficient and carefully crafted. The implementers must select scalable algorithms for key functionality, such as barriers, and implement these and other algorithms in a manner that avoids potential performance problems during execution, such as the false sharing of cached data. They must provide sensible strategies for handling idle threads, help to optimize the performance of parallel loops, and devise methods for using architectural features where possible.

This book focuses on the practice of creating high-performance parallel runtimes. Given the importance of a well-engineered runtime for parallel programming models, it is long overdue. The authors explore in depth many of the design and implementation decisions that go into the creation of a state-of-the-art runtime system for executing parallel application code, provide numerous examples, and discuss several real-world compilers and runtimes. The book describes the salient details of current computer architectures so that the parallel programming requirements and subsequent

https://doi.org/10.1515/9783110632729-201

techniques are well motivated. We learn, for instance, about the details of modern multicore cache management strategies and their performance implications, as well as how atomics and transactional memory may be used to support synchronization.

The authors address general design considerations, but also discuss the implications of contemporary architectures and parallel programming features for the runtime developer. They explain in some detail how the compiler and runtime collaborate to implement the OpenMP* standard and Intel* Threading Building Blocks (TBB). While these serve admirably to illustrate the interactions between compiler and runtime, the text goes much further than that. It gives a short introduction to the workings of a compiler and describes how it can implement lambda functions that simplify the code for creating TBB tasks. OpenMP has a number of different constructs by means of which an application developer may express parallelism in a code. The authors give an overview of its implementation that is also very much more than an illustration of the role of runtime systems. Their detailed explanation of the compilation steps that precede execution, in addition to details of the runtime components, will allow the book to serve as an excellent introduction to the implementation of these and other high-level parallel programming language features.

There are a variety of different synchronization mechanisms and algorithms for implementing them. Here, we are treated to a comprehensive and practical discussion of hardware support for synchronization, popular synchronization constructs, and strategies for their implementation. There is a treatise on the extraordinary challenges of implementing locks, and a discussion of the perennial problem of what to do with threads that are waiting.

Tasks have grown tremendously in popularity as a means for specifying parallelism. They are inherently dynamic, in the sense that their creation is decoupled from their execution. In contrast to many other language constructs, the runtime oversees all aspects of the execution of tasks, including handling any requirements of their relative ordering and enqueuing them for execution. The discussion of their runtime support here goes far beyond the design of a task pool (or queue) and considers task dependences, scheduling, task stealing, and additional synchronizations.

As our platforms grow in heterogeneity, as well as in scale, the role of the runtime in the implementation is becoming increasingly important. Far from being "merely" a support infrastructure for the compiler, today's parallel runtime must actively manage the workload across multiple NUMA domains, potentially making decisions on where a particular piece of code should be executed or when to attempt to steal a task. Future runtimes are expected to be even more powerful, potentially adapting details of a code's execution to address changes in the execution environment or in the computational needs of the program itself.

I believe that this book will be of great value to systems researchers and practitioners alike. It provides a wealth of information on all the major responsibilities of a modern runtime system, with practical implementation details and potential pitfalls clearly explained. Code for the OpenMP runtime is also available for the reader to try

out and experiment with. There is sufficient background on multicore platforms to ensure that newcomers can understand the contents. Indeed, it is also an excellent introduction to the practical workings of such systems. All of this comes in a well-written, entertaining, and up-to-date book, with many examples and illustrations. Despite the depth with which many topics are treated, the narrative remains accessible throughout. The authors are especially well qualified to write on this topic. Both active practitioners, each of them has accumulated extensive experience in the design and implementation of parallel programming libraries, languages, and of course, parallel runtimes. They have a deep appreciation for the challenges and pitfalls that may present themselves in the context of exploiting parallel systems, and for the engineering aspects of any real-world software development effort.

I have been involved in the development of compiler support for OpenMP almost since it was first introduced to the public, and am keenly aware of the importance of a carefully crafted runtime system. A crucial element in any OpenMP implementation is a quality runtime system whose creators have successfully overcome the myriad performance challenges posed by multicore platforms. This requires engineering of the highest order, led by a deep understanding of the nature of the task and its inherent challenges along with some solid solution strategies. This understanding, and the techniques to get the job done, is what this book provides.

I highly recommend it!

Barbara Chapman
Stony Brook University
Brookhaven National Laboratory