Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

For further volumes:
http://www.springer.com/series/7592

Yosi Ben-Asher

Multicore Programming Using the ParC Language

Springer

Yosi Ben-Asher
Department of Computer Science
University of Haifa
Haifa, Israel

ISSN 1863-7310 Undergraduate Topics in Computer Science
ISBN 978-1-4471-2163-3 ISBN 978-1-4471-2164-0 (eBook)
DOI 10.1007/978-1-4471-2164-0
Springer London Heidelberg New York Dordrecht

Library of Congress Control Number: 2012940236

Preface

This book discusses principles of practical parallel programming using shared memory on multicore machines. It uses a simple yet powerful parallel dialect of C called ParC as the basic programming language. The book contains a mixture of research directions combined with elementary material and basic concepts of parallel processing. As such it is intended to serve both as a text-book for a course on shared memory programming and as a way to improve the research background of specialists working on practical implementations of parallel programming systems over multicore machines. The objective of this book is to provide a firm basis for the "delicate art" of creating efficient parallel programs. The students can exercise parallel programming using a simulation software, which is portable on PC/Unix Multicore computers. Hence no special hardware is needed in order to gain experience in parallel programming using this book. Apart from the simulator *ParC* can be directly executed on Multicore machines. In the first four chapters elementary and advance concepts of parallel programming are carefully introduced via *ParC* examples. There is a special effort to present parallel programs as partial order over a set of assignments. Partial orders thus form the basis for determining elementary concepts such as the execution time of parallel programs scheduling, atomicity and threads.

The remaining chapters cover issues in parallel operating systems and compilation techniques that are relevant for shared memory and multicore machines. This reflects our belief that knowing how a parallel system works is necessary to master parallel programming. It describes the principles of scheduling parallel programs and the way ParC's constructs are implemented. A separate chapter presents a set of ParC programs that forms a benchmark for measuring basic constants of the underlying system, e.g., the cost of creating a thread. A set of relevant compilation techniques of ParC programs is described. A special treatment is given to stack management, showing that essentially the stacks of different threads can be mapped to the regular stack of each core.

This book targets multicore machines which are now available as regular personal computers, laptops and servers. The shared memory in multicore machines is implemented using complex algorithms maintaining cache-consistency of the dif-

ferent cores. The hidden costs involved with the use of cache-consistency can easily damage the expected benefit of using shared memory parallel programming in multicore machines. Thus we carefully study the issue of cache consistency in multicore machines and consider several programming techniques to optimize the cache usage in ParC programs.

The usefulness of ParC stems from two sources. On the one hand, it is similar in flavor to the pseudo-languages that are used by theoreticians for describing their parallel algorithms. Indeed, parallel algorithms can be directly coded in ParC. On the other hand, sequential code segments can be easily parallelized in ParC. These two directions of coding parallel algorithms and parallelizing sequential code form the basis for developing practical parallel programs and are thus widely studied in this book. The ParC language promotes an understanding of the problems hidden in parallel code, and allows various approaches for their solutions. Though there are several popular ways of programming multicore machines such as OpenMP and Java-multithreading, we prefer ParC for learning parallel programming:

- It is significantly simpler than OpenMP and is naturally learned and used by C programmers.
- In some sense, it is more powerful than OpenMP and Java due to its extended scoping rules and support for optimizing locality of shared memory references.

As parallel hardware becomes more and more popular the demand to develop methodologies for teaching parallel programming becomes more evident. This book promotes the idea that parallel programming is best taught through the issue of **efficiency**. The claim is, that parallel programming capability is best gained through a systematic learning of programming techniques used to derive better efficient versions of parallel programs. Rather than surveying, parallel architectures, programming stiles and a verity of applications, as is the case in many of the existing books in this field.

Finally, the only motive in developing and using parallel machines and languages is to speedup performances. This calls for a highly skilled careful programmer, who can create efficient programs, which exploit all potential advantages of the parallel hardware. A parallel algorithm can be programmed in many ways, yielding different performances. Lack of experience and careless programming can easily lead to extreme situations, namely when the performances of a parallel program executed on a given parallel machine, are actually worse than its sequential version.

In this book the method used to evaluate the expected performances of programs is **analytical**. The source code of the program is analyzed via a formula to determine the execution time of a program based on derived upper and a lower bounds. Next a speedup formula is derived and is separated to the different factors that may block the program from achieving a good speedup. Following this procedure it is possible to identify bottlenecks areas in the program and correct them. We shortly consider a second method to locate limiting factors that is based on the ParC simulator. The program is executed on a simulator, and its time measurements are used to spot performance hot-spots. Two types of time statistics for a parallel execution of a program are defined. Ideal-times are execution times that reflect optimal execution

of the program without any practical considerations (such as number of cores and overheads), while the Effective-times reflect limiting factors. An efficient program is obtained (using a simulator) when the gap between the ideal times and the effective times is reduced.

Acknowledgements

ParC was first developed by Yosi Ben-Asher and Marc Snir at NYU the first implementation run on the Ultra Computer at NYU. At a later stage *ParC* was implemented on the Makbilan research project led by Larry Rudolph in the Hebrew University. The compiler and simulator were written by Yosi Ben-Asher. A new run-time environment for *ParC* on the Makbilan was implemented by Dror Feitelson, Moshe Ben Ezra, and Lior Picherski. Gudy Habber modified the *ParC* simulator from SUN workstation to PC-DOS systems.

Contents