Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency

Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency

Kunle Olukotun, Lance Hammond, and James Laudon

ISBN: 978-3-031-00592-3 paperback
ISBN: 978-3-031-01720-9 ebook

DOI 10.1007/978-3-031-01720-9

A Publication in the Springer series
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #3

Lecture #3
Series Editor: Mark D. Hill, University of Wisconsin

Library of Congress Cataloging-in-Publication Data

Series ISSN: 1935-3235 print
Series ISSN: 1935-3243 electronic

First Edition
10 9 8 7 6 5 4 3 2 1

Synthesis Lectures on Computer Architecture

Editor
Mark D. Hill, ***University of Wisconsin, Madison***

Synthesis Lectures on Computer Architecture publishes 50- to 150 page publications on topics pertaining to the science and art of designing, analyzing, selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals.

Chip Mutiprocessor Architecture: Techniques to Improve Throughput and Latency
Kunle Olukotun, Lance Hammond, James Laudon
2007

Transactional Memory
James R. Larus, Ravi Rajwar
2007

Quantum Computing for Computer Architects
Tzvetan S. Metodi, Frederic T. Chong
2006

Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency

Kunle Olukotun
Stanford University

Lance Hammond
Stanford University

James Laudon
Sun Microsystems

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #3

ABSTRACT

Chip multiprocessors — also called multi-core microprocessors or CMPs for short — are now the only way to build high-performance microprocessors, for a variety of reasons. Large uniprocessors are no longer scaling in performance, because it is only possible to extract a limited amount of parallelism from a typical instruction stream using conventional superscalar instruction issue techniques. In addition, one cannot simply ratchet up the clock speed on today's processors, or the power dissipation will become prohibitive in all but water-cooled systems. Compounding these problems is the simple fact that with the immense numbers of transistors available on today's microprocessor chips, it is too costly to design and debug ever-larger processors every year or two.

CMPs avoid these problems by filling up a processor die with multiple, relatively simpler processor cores instead of just one huge core. The exact size of a CMP's cores can vary from very simple pipelines to moderately complex superscalar processors, but once a core has been selected the CMP's performance can easily scale across silicon process generations simply by stamping down more copies of the hard-to-design, high-speed processor core in each successive chip generation. In addition, parallel code execution, obtained by spreading multiple threads of execution across the various cores, can achieve significantly higher performance than would be possible using only a single core. While parallel threads are already common in many useful workloads, there are still important workloads that are hard to divide into parallel threads. The low inter-processor communication latency between the cores in a CMP helps make a much wider range of applications viable candidates for parallel execution than was possible with conventional, multi-chip multiprocessors; nevertheless, limited parallelism in key applications is the main factor limiting acceptance of CMPs in some types of systems.

After a discussion of the basic pros and cons of CMPs when they are compared with conventional uniprocessors, this book examines how CMPs can best be designed to handle two radically different kinds of workloads that are likely to be used with a CMP: highly parallel, *throughput-sensitive* applications at one end of the spectrum, and less parallel, *latency-sensitive* applications at the other. Throughput-sensitive applications, such as server workloads that handle many independent transactions at once, require careful balancing of all parts of a CMP that can limit throughput, such as the individual cores, on-chip cache memory, and off-chip memory interfaces. Several studies and example systems, such as the Sun Niagara, that examine the necessary tradeoffs are presented here. In contrast, latency-sensitive applications — many desktop applications fall into this category — require a focus on reducing inter-core communication latency and applying techniques to help programmers divide their programs into multiple threads as easily as possible. This book discusses many techniques that can be used in CMPs to simplify parallel programming, with an emphasis on research

directions proposed at Stanford University. To illustrate the advantages possible with a CMP using a couple of solid examples, extra focus is given to *thread-level speculation* (TLS), a way to automatically break up nominally sequential applications into parallel threads on a CMP, and *transactional memory*. This model can greatly simplify manual parallel programming by using hardware — instead of conventional software locks — to enforce atomic code execution of blocks of instructions, a technique that makes parallel coding much less error-prone.

KEYWORDS

Basic Terms: chip multiprocessors (CMPs), multi-core microprocessors, microprocessor power, parallel processing, threaded execution

Application Classes: throughput-sensitive applications, server applications, latency-sensitive applications, desktop applications, SPEC benchmarks, Java applications

Technologies: thread-level speculation (TLS), JRPM virtual machine, tracer for extracting speculative threads (TEST), transactional memory, transactional coherency and consistency (TCC), transactional lock removal (TLR)

System Names: DEC Piranha, Sun Niagara, Sun Niagara 2, Stanford Hydra

Contents