

COMP8320: Multicore Computing: Principles and Practices

(6 units) Group D

Second Semester

There will be one two-hour lecture per week (20 lectures in total), 4-6 two-hour tutorials (based on selected readings), and 6 two-hour practical laboratory sessions.

Prerequisites

The course is normally restricted to those students enrolled in the Master of Computing and honours students. Assumed knowledge is equivalent to having done the equivalent of an introductory course on computer architecture, a course on concurrency, and intermediate programming and data structure courses.

Syllabus

This course is a practical introduction to large-scale multicore computing. It covers the principles and practices of contemporary and emerging multicore computers, with an emphasis on their impact upon software engineering practice. It also has an emphasis on the state-of-the-art of research which is driving the rapid evolution of these systems.

Topics include multicore processor architecture, operating system considerations, programming language support with explicit and implicit parallelism, fast synchronization and concurrent data structure access, programming techniques and scalability/reliability considerations.

The central principles and techniques will be grounded in an in-depth study of a contemporary multicore system and programming paradigm. Their underlying design decisions will be analysed, and advanced topics illustrated by this system and paradigm will be studied.

Description

Multicore Computing involves combining an understanding modern computer architecture, emerging parallel programming paradigms, key concurrent algorithms and software engineering principles in order to take advantage of the tremendous processing power and efficiencies of this emerging computer technology.

The course will cover the following topics:

- principles of multicore computer architecture: CPU design for multi-core, heterogeneity, hardware threading, memory hierarchy coherency and consistency, power saving considerations, transactional memory support.
- overview of programming paradigms for multicore, including low-level (e.g. Posix threads), annotation-based (e.g. OpenMP), object-based (e.g. Intel TBB), implicit parallelism (e.g. Fortress), functional (e.g. Erlang). Considerations of programmability and performance tradeoffs; transactional memory support.
- synchronization methods; blocking and busy-wait; concurrent linked lists, queues, stacks, and hashing.

- operating system issues: scheduling and work distribution.
- approaches to parallel application design; testing and logical fault tolerance for parallel programs; scalability concerns.
- special advanced topics may include virtualization support for multicore computing, support for modern programming language implementation, special processor cores including graphics processors.

For the practical aspects of the course, students will be given access to state-of-the-art multicore system, such as an UltraSPARC T2 processor.

The course will mainly be based on the texts *The Art of Multiprocessor Programming*, Maurice Herlihy and Nir Shavit Morgan Kaufman, ISBN-13: 978-0-12-370591-4, 2008; and *Professional Multicore Programming: Design and Implementation for C++ Developers*, Cameron Hughes and Tracey Hughes, Wiley, ISBN: 978-0-470-28962-4, 2008.

Rationale

Large-scale multicore processing has emerged as the disruptive technology of the mid 2000's, as it is seen as the only viable way of increasing single processing chip performance, and in particular, with manageable energy consumption. It is a key enabling technology of the international High Productivity Computing Systems initiative. It is having a profound effect on software engineering practice, where new applications needing to be written, and old applications needing to be re-written, to expose sufficient parallelism to be utilized on multicore systems. It is essential that the practice of IT professionals and in particular software engineers be upgraded to reflect these trends.

This requires in turn an understanding of architecture, programming paradigms, key concurrent algorithms and data structures, and programming practices.

Objectives

Students will have an understanding of the issues involved in the design of hardware and programming languages for multicore systems, and be able to employ algorithms and data structures for applications that are efficient on large-scale systems. They will be proficient in at least two programming languages used on multicore systems, and will be able to evaluate their program's reliability and scalability. Students will have an understanding of the research issues driving multicore technology, and be able to assimilate and understand the impact of current literature.

Ideas

Topics

See Description.

Proposed Assessment

A combination of written examination, exercises, and a seminar presentation.

The mark reported for a student will reflect that student's performance in the course and will be based on:

- Two assignments (programming-based) [worth 15% each]
- A presentation of a research paper [20%]
- A final written examination [50%]

Recommended Reading

- Maurice Herlihy and Nir Shavit. *The Art of Multiprocessor Programming*. Morgan Kaufmann, 2008.
- Cameron Hughes and Tracey Hughes. *Professional Multicore Programming: Design and Implementation for C++ Developers*. Wiley, 2008.

Peter Strazdins 2008-07-18