Module 4: Concurrent and Parallel Programming

Stage				1			
Semester				1			
Module Title				Concurrent and Parallel Programming			
Module Number				4			
Module Status				Mandatory			
Module ECTS Credits				10			
Module NFQ level				9			
Pre-Requisite Module Titles				None			
Co-Requisite Module Titles				None			
Capstone Module?				No			
List of Module Teaching Personnel				Mr Tony Mullins			
Contact Hours				**Non-Contact Hours**			**Total Effort (hours)**
72				**128**			**200**
Lecture	**Practical**	**Tutorial**	**Seminar**	**Assignment**	**Placement**	**Independent Work**	
36		36		64		64	
Allocation of Marks (Within the Module)							
	Continuous Assessment	**Project**	**Practical**	**Final Examination**			**Total**
Percentage Contribution	**50**			**50**			**100**

Intended Module Learning Outcomes

On successful completion of this module the learner will be able to:

1. Design and implement concurrent and parallel algorithms
2. Use a parallel model to map algorithms to an underlying architecture , e.g. C, C++, Ada tasks, Java threads, OpenMP, Posix thread, Scala actors(agents), f# agents(actors);
3. Demonstrate a critical understanding of multi processor and multi core architectures;
4. Solve problems requiring both semaphores and events as part of the solution;
5. Analyse and document the difference between state based concurrent systems and concurrent systems based on immutable state;
6. Design solutions to concurrent & parallel problems using actor based message passing;
7. Implement different server architectures (topologies), e.g. client server, peer-to-peer, agent systems, grid architectures;
8. Use parallelism to optimise performance of algorithms

Module Objectives

The future of microprocessor development is based around multiprocessor multicore architectures that will deliver the performance required for future application demands. The difficulty for software developers is to write programs that harness the power of these new architectures. As a result the fundamental aim of this module is to teach the learner how to write software for these machines.

The general module aims are to provide the learner with an understanding of the need for, and advantages of, concurrent and parallel systems; to master a new programming paradigm that is different from that of the single threaded one; a description of how processes and threads are managed in multiprocessor, multi core machines. The learner will achieve an understanding and mastery of the many classical problems arising with concurrent and parallel tasks; an awareness of the need for such issues as fairness, process synchronisation, deadlock avoidance, etc.; the ability to write concurrent and parallel programs to solve real world problems; an understanding of multi-core architectures and their significance for the implementation of parallel systems; a mastery of notations to express solutions to parallel problems.

Module Curriculum

- **Fundamentals**
  Multi core, multiprocessor systems / Hardware Architectures / Motivation for concurrency and parallelism / simple examples / advantages / disadvantages / Process versus threads / priority of processes / process creation and destruction (Fork in Unix, Task in Ada, thread in java) / processes sharing memory / dynamic process creation / distributed memory / facilities for concurrency and parallelism provided by programming languages and operating systems; C, C++, Ada tasks, Java threads, OpenMP, MPI, Erlang, Posix threads, Scala, F#, Windows, Linux and Unix.

- **Resource Sharing**
  Mutual exclusion / semaphores / fairness / deadlock / starvation / monitors / protected objects / condition variables / various kinds of shareable resources, e.g. memory, files, printers, etc / degrees of sharing, e.g. grab whole file or grab a single record / deadlock prevention. Classic problems: readers/writers, producer/consumer, bounded buffer. / General problems requiring concurrent and parallel solutions e.g. lift control, telephone exchange, etc. / Strategies for allocating resources / fairness / resource allocation algorithms. / Necessity for scheduling algorithms / thread priority / resource allocation problems.

- **Parallelism**
  Task parallelism, data parallelism / Parallel algorithms from many areas, including matrix algorithms, graph algorithms, solution to linear equations, searching and sorting, image processing, encryption / Parallel models – mapping parallel algorithms to underlying architecture using e.g. C, C++, Ada tasks, Java threads, OpenMP, Erlang, Posix threads, Scala, F#.

- **Function Programming and Concurrency**
  Message passing systems based on Actors (Scala, F# and Erlang) / Avoiding race conditions with the use of immutable state / Communication protocols for actors / Programming with Actors.

- **Communicating Processes (processes without shared memory)**
  Distributed memory model; Pipes; channels; message passing; remote procedure call; process identities; multi-casting – broadcast to multiple processes.

- **Server Architectures**
  Client server architecture, peer-to-peer architecture, grid computing. Deploying services over these architectures. Developing an Agent based architecture.

Reading Lists and other learning materials

Recommended Reading

Mullins, 2010, *Concurrent and Parallel Programming*, Griffith College Dublin

Doug, 2000, *Concurrent Programming in Java*, Sun

Goetz et al, 2006, *Java Concurrency in Practice*, Addison Wesley

Lester, 2006, *The Art of Parallel Programming*, 1st World Publishing

Herlihy, Shavit, 2007, *The Art of Multi Processor Programming*, Morgan Kaufmann

Quinn, 2004, *Parallel Programming in C with MPI and OpenMP*, McGraw Hill

Secondary Reading

Mullins, 2012, *Programming Paradigms with Scala*, Griffith College Dublin

Odersky, Spoon, Venners, 2010, *Programming in Scala*, Artima Press

Chandra, Mennon, et al, 2000, *Parallel Programming in OpenMP*, Morgan Kaufmann

Additional reading as recommended by lecturer, appropriate to topic and to each learner's area of research.

Module Learning Environment

Lectures are carried out in class rooms / lecture halls in the College. Lab tutorials are carried out in computer labs. All labs have the software required to deliver the programme.

Library

All learners have access to an extensive range of physical and electronic (remotely accessible) library resources. The library monitors and updates its resources on an on-going basis, in line with the College's Library Acquisition Policy. Lecturers update reading lists for this course on an annual basis as is the norm with all courses run by Griffith College.

Module Teaching and Learning Strategy

The module is taught using a combination of lectures, demonstrations and tutorials. The demonstrations and tutorials focus on getting learners up to standard in practical application development. The lectures supply the necessary theoretical background which informs the practice in tutorial. There is an emphasis on continuing practical development with learners receiving focussed formative feedback throughout.

Module Assessment Strategy

This module is 50% continuous assessment and the other 50% is an examination. The full breakdown of module assessment is described in the following table

Element No.	Weighting	Type	Description	Learning Outcomes Assessed
1	10%	Assignment	Programming problems that involve use of threads and parallel processing	1,2
2	10%	Assignment	Programming problems that involve race conditions and that require the use of condition variables to solve	1,2
3	10%	Assignment	Programming problems that may be solved using semaphores	1,2,4
4	15%	Assignment	Distributed processing and client server architecture	1,2,7,8
5	5%	Assignment	Concurrent systems based on immutable state and solution to problems using actor based model of concurrency	1,2,5,6
6	50%	Examination	Questions on the exam paper are drawn from all aspects of the course. The paper consists of 6 questions and learners are required to answer any 5. All questions carry equal marks.	2,3,4,5,6,8