

F21DP2 - Distributed and Parallel Technology

Greg Michaelson

Phil Trinder

Multithreaded Computation

1

Reasons for Multithreaded Programming

Programs execute in multiple threads or on multiple processors for different purposes:

- **Concurrency:** stateful interactions at a single location. Some applications are naturally modelled as cooperating processes, e.g. GUI constructed as cooperating threads, where a thread manages each device: keyboard, screen, etc.
- **Parallelism:** aims to reduce execution time or improve throughput, e.g. render an animated film using a different machine for each sequence.
- **Distribution:** stateful interactions at multiple locations, e.g. multiple users in remote locations interact in a shared-world.
- **Mobility:** computations move from location to location, e.g. personalised web search agent.

2

Multithreaded Computation is Hard

A multithreaded program poses all the challenges of sequential **Computation:** i.e. **what** to compute. A correct and efficient algorithm must be constructed.

A multithreaded program must also specify a correct and effective strategy for **Coordination:** i.e. **how** computations should be arranged on the locations.

3

Coordination Aspects

- **Partitioning:** determining what parts of the computation should be separately evaluated, e.g. a thread to render each frame of a film.
- **Placement:** determining where threads should be executed, e.g. allocate thread to least busy location.
- **Communication:** what data to send, e.g. maintain at least 1 frame on each location, ready for processing when the current frame has been rendered.
- **Synchronisation:** ensuring threads can cooperate without interference, e.g. if threads representing 2 players compete to get a single resource then only one succeeds.
- **Scheduling:** determining which thread to execute next in each location, e.g. round robin scheduling
- etc.

4

Coordination Levels

You have used many notations for specifying, designing & constructing computations but relatively few for coordination.

This module will study parallel, distributed and mobile coordination

Computations can be written in languages with different levels of abstraction, e.g.

Low-level	Mid-level	High-Level
←----->		
Assembler	Java	SML, Haskell Prolog

Likewise coordination can be written in languages with different levels of abstraction, e.g.

Low-level	Mid-level	High-Level
←----->		
Sockets, Semaphores	Comm Lib. e.g. MPI	GpH, PMLS HPF

5

Distribution Topics

The module will cover existing distribution technologies:

- Review of low-level technologies, e.g. sockets
- Remote evaluation, e.g. RPC/RMI
- Object-based systems, e.g. CORBA
- Fault tolerance, e.g. exceptions

The module will also cover emerging distribution technologies:

- Mobile/Global Computation
- Grid Computing

6

Haskell as a Computation Language

The module uses the functional language Haskell as a high-level computation language, as several variants are available with high-level coordination extensions:

- Glasgow parallel Haskell (GpH)
- mHaskell, for mobility
- GridGpH, for computational Grid programming

7