A Work-Stealing Scheduling Technique Applied to Computing the Mandelbrot Set.

Martin J Hawes
The University of Hertfordshire

February 1, 2013

0.1 Abstract

This is the abstract. Lets cite something (Person 1998, p. 201).

This is another citation which should look a bit nicer (McGuiness 2006).

hello reference of fig **??**

0.2 Acknowledgements

Contents

Chapter 1

Introduction

Chapter 2

Background Research

2.1 Scheduling Techniques for Multi-Threaded Computations

This section briefly describes the problems associated with scheduling Multi-Threaded Algorithms and the major paradigms that have surfaced.

In order to efficiently utilize a parallel computer architecture it is desirable to minimize the amount of time a processor spends idle or performing other logistical tasks, I.e not doing work. When a computation's *concurrent sub-tasks* or *threads* incur a regular cost in processor time, each processor can simply have the same amount of work assigned to them. When the computation has more irregular or dynamically growing sub-tasks a problem arises that results in processors becoming idle while others still remain working. The solution to this problem is refereed to as *load balancing* and can be described as a form of dynamic scheduling that ensures each processor spends approximately the same amount of time working. This means processors generally spend less time idle, however have to deal with scheduling overheads as a trade-off.

When considering the scheduling of multi-threaded computations, two major load balancing techniques have been used. These are *work-sharing* and *work-stealing*.

- **Work-Sharing:** When scheduling using a work-sharing approach, a thread that creates new threads attempts to migrate these to another underutilised processor at creation time.
- **Work-Stealing:** In work-stealing a thread that is starved of work, attempts to "steal" work from other processors.

Both techniques intend to promote balanced work-load across all processors, however in Work-Stealing the frequency of thread migrations is lower. When all processors have a high work-load and no need to "steal" this becomes useful because threads need not get migrated at all. With work-sharing thread migration occurs each time a new thread is spawned (Blumofe & Leiserson 1994, p. 2). This also suggests that work-stealing promotes better locality and grouping of sub-tasks, as spawned threads stay with the same processor until stolen.

2.2 The Work-Stealing Technique - Described in Depth

2.3 The Mandelbrot Set

2.3.1 Fractals

2.3.2 Julia Sets

2.3.3 Complex Numbers

2.3.4 Computing the Mandelbrot Set

2.4 Programming Tools

Chapter 3

Main Chapters

3.1 Design of the Algorithm

3.2 The Implementation

3.2.1 An Algorithm to Compute the Mandelbrot Set

3.2.2 A Concurrent Algorithm to Compute the Mandelbrot Set

3.3 Features for Demonstration

3.4 Validation and Verification

Chapter 4

Discussion and Evaluation

4.1 Analysis of the Algorithm

4.2 Reflection on Project

Chapter 5

Resources

Bibliography

R. D. Blumofe & C. E. Leiserson (1994). ‘Scheduling Multithreaded Computations by Work Stealing’. Tech. rep., MIT Laboratory for Computer Science.

J. McGuiness (2006). ‘Something to do with this or that? who knows...’. Master’s thesis, University of Hertfordshire.

P. Person (1998). ‘Big fancy title line’. *the journey* .

F. Turner (2001). ‘Something to do with this or that? who knows...’. Master’s thesis, University of France.

Chapter 6

Appendices