CONTENTS

1. INTRODUCTION
   1.1 Parallel Processing: Functional Language Approach
   1.2 Motivation
   1.3 Parallelism
   1.4 Organisation of the thesis

2. REVIEW OF FUNCTIONAL PROGRAMMING
   2.1 Introduction
   2.2 Functional Programming and Lambda Calculus
      2.2.1 Functional Programming
      2.2.2 Lambda Calculus
   2.3 Haskell
      2.3.1 Formal semantics
   2.4 Reduction Machine
      2.4.1 Lambda calculus as intermediate language
      2.4.2 Other forms of Intermediate language
   2.5 Parallel Functional Programming

3. LAZINESS AND PARALLEL DATA STRUCTURES.
   3.1 Introduction
      3.1.1 Parallel paradigms
      3.1.2 Map function
   3.2 An introduction to the 'aim of evaluation'
      3.2.1 The print-eval loop:
      3.2.2 The aim as an abstraction of a SIMD activity mark
      3.2.3 An operational View of the aim of evaluation
      3.2.4 Data-Parallel non-strict glue.
3.3 PODS

3.4 POD Comprehensions

3.5 The Semantics of POD Comprehensions: Primitive operations

3.5.1 ⊥ and the representation of PODs

3.5.2 map

3.5.3 Indices

3.5.4 fetching

3.5.5 Sending

3.6 The Semantics of POD comprehensions: desugaring,

3.6.1 Drawn – from

3.6.2 Indexed-from

3.6.3 Filtering

3.6.4 Left-hand side

3.7 Conclusion.

4. PARALLEL ALGORITHMS

4.1 Deriving a parallel Scan function.

4.1.1 Sequential: The partial sums of list.

4.1.2 Parallel : a first attempt

4.1.3 Parallel : a lazier attempt

4.1.4 Higher order glue and fold.

4.2 First order linear recurrence

4.3 Packing

4.4 Radix Sorting

4.4.1 The radix sorting algorithm

4.4.2 The “items before”: imperative

4.4.3 The “items before” : functional

4.4.4 The Radix part of the sort

4.4.5 Bit twiddling in a functional language
5. ALGORITHMS FOR SIMD ARCHITECTURES

5.1 Program identifier for Vectorization
   5.1.1 Vector form
   5.1.2 Basic rules for expressions
   5.1.3 Mapping through the "fire break" of a local binding
   5.1.4 Letrec: tidying-up Vectorisation
   5.1.5 Vectorising top level bindings
   5.1.6 Vectoring sequential objects to infinite PODs
   5.1.7 Conversion
   5.1.8 Mapping through a primitive case expression

5.2 Vectorising algebraic data types
   5.2.1 Care closures: a naive implementation
   5.2.2 The "inside-out" transformation
   5.2.3 Flattening a sum of product type
   5.2.4 Vectorising a flattened sum-of-product type
   5.2.5 The program identity for constructors
   5.2.6 An introduction to merging
   5.2.7 Case analysis

5.3 Higher-Order Vectorisation

5.4 Programs that fail Vectorisation

5.5 Summary

6. PFPM – AN ABSTRACT MACHINE

6.1 Introduction

6.2 The PFPM System
   6.2.1 The intermediate code FC
   6.2.2 Constants (cv)