

**THESIS TITLE**

*by*

FName LName

Submitted in Fulfilment of the Requirements for the Degree of  
Doctor of Philosophy



UNIVERSITY OF GLASGOW  
COLLEGE OF SCIENCE AND ENGINEERING  
SCHOOL OF COMPUTING SCIENCE

September 2014

The copyright in this thesis is owned by the author. Any quotation from the thesis or use of any of the information contained in it must acknowledge this thesis as the source of the quotation or information.

# Abstract

Write abstract here..

# Contents

<b>1</b>	<b>Introduction</b>	<b>9</b>
1.1	Context . . . . .	9
1.2	Thesis Statement . . . . .	9
1.3	Contributions . . . . .	9
1.4	Authorship and Publications . . . . .	9
1.5	Thesis Structure . . . . .	9
<b>2</b>	<b>Background</b>	<b>10</b>
2.1	Introduction . . . . .	10
2.2	Parallel Architectures . . . . .	10
2.3	Summary . . . . .	10
<b>3</b>	<b>Conclusion and Future Work</b>	<b>11</b>
3.1	Summary . . . . .	11
3.2	Limitations . . . . .	11
3.3	Future Work . . . . .	11
<b>A</b>	<b>Benchmarks</b>	<b>12</b>
A.1	Sum Euler . . . . .	13
	<b>Glossary</b>	<b>15</b>
	<b>Bibliography</b>	<b>16</b>

# List of Tables

# List of Figures

2.1 Shared Memory SMP Architecture . . . . .	10
--	----

# Dedication

To my Mother and Father.

# Acknowledgements

Write here...

# Declaration

I declare that, except where explicit reference is made to the contribution of others, that this dissertation is the result of my own work and has not been submitted for any other degree at the University of Glasgow or any other institution.

Write your name here



# Chapter 1

## Introduction

### 1.1 Context

How to use Macros. The Glasgow Haskell Compiler (GHC) is write.

How to cite. According to Akhter and Roberts [1] write. According to Shende et al. [2] write.

How to use glossaries. The number of Processing Elements (PEs) write. The Haskell on a Shared Memory Multiprocessor (GHC-SMP) is write.

### 1.2 Thesis Statement

### 1.3 Contributions

1. **Write..** detail here.
2. **Write..** detail here.

### 1.4 Authorship and Publications

### 1.5 Thesis Structure

The structure of this thesis is as follows:

**Chapter 2** gives....

# Chapter 2

## Background

### 2.1 Introduction

### 2.2 Parallel Architectures

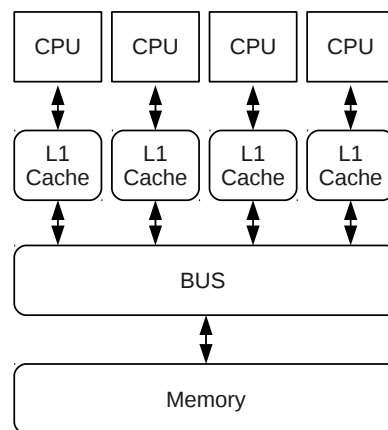


Figure 2.1: Shared Memory SMP Architecture

### 2.3 Summary

# **Chapter 3**

## **Conclusion and Future Work**

### **3.1 Summary**

### **3.2 Limitations**

### **3.3 Future Work**

# Appendix A

## Benchmarks

Listing A.1: Fibonacci Benchmark Implementation.

---

```
1
2 — / sequential Fibonacci
3
4 fib :: Int -> Integer
5 fib n | n <= 1    = 1
6       | otherwise = fib (n-1) + fib (n-2)
7
8
9 — / parallel Fibonacci; shared memory
10
11 par_fib :: Int -> Int -> Par Integer
12 par_fib seqThreshold n
13   | n <= k    = force $ fib n
14   | otherwise = do v <- new
15                   let job = par_fib seqThreshold (n - 1) >>=
16                       force >>=
17                       put v
18                   fork job
19                   y <- par_fib seqThreshold (n - 2)
20                   x <- get v
21                   force $ x + y
22   where k = max 1 seqThreshold
23
24 — / parallel Fibonacci; distributed memory
25
26 dist_fib :: Int -> Int -> Int -> Par Integer
27 dist_fib seqThreshold parThreshold n
28   | n <= k    = force $ fib n
29   | n <= l    = par_fib seqThreshold n
30   | otherwise = do
31     v <- new
32     gv <- glob v
33     spark $(mkClosure [| dist_fib_abs (seqThreshold, parThreshold, n, gv) |])
34     y <- dist_fib seqThreshold parThreshold (n - 2)
35     clo_x <- get v
36     force $ unClosure clo_x + y
37   where k = max 1 seqThreshold
38         l = parThreshold
39
40 dist_fib_abs :: (Int, Int, Int, GVar (Closure Integer)) -> Thunk (Par ())
41 dist_fib_abs (seqThreshold, parThreshold, n, gv) =
42   Thunk $ dist_fib seqThreshold parThreshold (n - 1) >>=
43     force >>=
44     rput gv . toClosure
45
46 — / parallel Fibonacci; distributed memory; using sparking d-n-c skeleton
47
48 spark_skel_fib :: Int -> Int -> Par Integer
49 spark_skel_fib seqThreshold n = unClosure <$> skel (toClosure n)
50   where
51     skel = parDivideAndConquer
52           $(mkClosure [| dnc_trivial_abs (seqThreshold) |])
```

```

53      $(mkClosure [| dnc_decompose |])
54      $(mkClosure [| dnc_combine |])
55      $(mkClosure [| dnc_f |])
56
57 dnc_trivial_abs :: Int -> Thunk (Closure Int -> Bool)
58 dnc_trivial_abs (seqThreshold) =
59   Thunk $ \ clo_n -> unClosure clo_n <= max 1 seqThreshold
60
61 dnc_decompose =
62   \ clo_n -> let n = unClosure clo_n in [toClosure (n-1), toClosure (n-2)]
63
64 dnc_combine =
65   \ - clos -> toClosure $ sum $ map unClosure clos
66
67 dnc_f =
68   \ clo_n -> toClosure <$> (force $ fib $ unClosure clo_n)
69
70 — / parallel Fibonacci; distributed memory; using pushing d-n-c skeleton
71
72 push_skel_fib :: [Node] -> Int -> Int -> Par Integer
73 push_skel_fib nodes seqThreshold n = unClosure <$> skel (toClosure n)
74   where
75     skel = pushDivideAndConquer
76           nodes
77           $(mkClosure [| dnc_trivial_abs (seqThreshold) |])
78           $(mkClosure [| dnc_decompose |])
79           $(mkClosure [| dnc_combine |])
80           $(mkClosure [| dnc_f |])

```

---

## A.1 Sum Euler

Listing A.2: Sum Euler Benchmark Implementation.

---

```

1 — / Euler's totient function (for positive integers)
2
3 totient :: Int -> Integer
4 totient n = toInteger $ length $ filter (\ k -> gcd n k == 1) [1 .. n]
5
6 — / sequential sum of totients
7
8 sum_totient :: [Int] -> Integer
9 sum_totient = sum . map totient
10
11 — / parallel sum of totients; shared memory
12
13 par_sum_totient_chunked :: Int -> Int -> Int -> Par Integer
14 par_sum_totient_chunked lower upper chunksize =
15   sum <$> (mapM get << (mapM fork_sum_euler $ chunked_list))
16   where
17     chunked_list = chunk chunksize [upper, upper - 1 .. lower] :: [[Int]]
18
19
20 par_sum_totient_sliced :: Int -> Int -> Int -> Par Integer
21 par_sum_totient_sliced lower upper slices =
22   sum <$> (mapM get << (mapM fork_sum_euler $ sliced_list))
23   where
24     sliced_list = slice slices [upper, upper - 1 .. lower] :: [[Int]]
25
26
27 fork_sum_euler :: [Int] -> Par (IVar Integer)
28 fork_sum_euler xs = do v <- new
29   fork $ force (sum_totient xs) >>= put v
30   return v
31
32 — / parallel sum of totients; distributed memory
33
34 dist_sum_totient_chunked :: Int -> Int -> Int -> Par Integer
35 dist_sum_totient_chunked lower upper chunksize = do
36   sum <$> (mapM get.and.unClosure << (mapM spark_sum_euler $ chunked_list))
37   where

```

```

38     chunked_list = chunk chunksize [upper, upper - 1 .. lower] :: [[Int]]
39
40
41 dist_sum_totient_sliced :: Int -> Int -> Int -> Par Integer
42 dist_sum_totient_sliced lower upper slices = do
43   sum <$> (mapM get_and_unClosure =<< (mapM spark_sum_euler $ sliced_list))
44   where
45     sliced_list = slice slices [upper, upper - 1 .. lower] :: [[Int]]
46
47
48 spark_sum_euler :: [Int] -> Par (IVar (Closure Integer))
49 spark_sum_euler xs = do
50   v <- new
51   gv <- glob v
52   spark $(mkClosure [| spark_sum_euler_abs (xs, gv) |])
53   return v
54
55 spark_sum_euler_abs :: ([Int], GIVar (Closure Integer)) -> Thunk (Par ())
56 spark_sum_euler_abs (xs, gv) =
57   Thunk $ force (sum_totient xs) >>= rput gv . toClosure
58
59 get_and_unClosure :: IVar (Closure a) -> Par a
60 get_and_unClosure = return . unClosure <=< get
61
62 — / parallel sum of totients; distributed memory (using plain task farm)
63
64 farm_sum_totient_chunked :: Int -> Int -> Int -> Par Integer
65 farm_sum_totient_chunked lower upper chunksize =
66   sum <$> parMapNF $(mkClosure [| sum_totient |]) chunked_list
67   where
68     chunked_list = chunk chunksize [upper, upper - 1 .. lower] :: [[Int]]
69
70
71 farm_sum_totient_sliced :: Int -> Int -> Int -> Par Integer
72 farm_sum_totient_sliced lower upper slices =
73   sum <$> parMapNF $(mkClosure [| sum_totient |]) sliced_list
74   where
75     sliced_list = slice slices [upper, upper - 1 .. lower] :: [[Int]]
76
77 — / parallel sum of totients; distributed memory (chunking/slicing task farms)
78
79 chunkfarm_sum_totient :: Int -> Int -> Int -> Par Integer
80 chunkfarm_sum_totient lower upper chunksize =
81   sum <$> parMapChunkedNF chunksize $(mkClosure [| totient |]) list
82   where
83     list = [upper, upper - 1 .. lower] :: [Int]
84
85
86 slicefarm_sum_totient :: Int -> Int -> Int -> Par Integer
87 slicefarm_sum_totient lower upper slices =
88   sum <$> parMapSlicedNF slices $(mkClosure [| totient |]) list
89   where
90     list = [upper, upper - 1 .. lower] :: [Int]

```

---

# Glossary

**GHC-SMP** Haskell on a Shared Memory Multiprocessor.

**PE** Processing Element.

# Bibliography

- [1] S. Akhter and J. Roberts. *Multi-Core Programming: Increasing Performance Through Software Multi-threading*. Richard Bowles, 2006.
- [2] S. Shende, A. D. Malony, J. Cuny, P. Beckman, S. Karmesin, and K. Lindlan. Portable Profiling and Tracing for Parallel, Scientific Applications Using C++. In *Proceedings of the SIGMETRICS Symposium on Parallel and Distributed Tools*, SPDT '98, pages 134–145, New York, NY, USA, 1998. ACM.