

PhD thesis: Data-parallel lazy functional programming

Hill, Jonathan

For additional information about this publication click this link.

<http://qmro.qmul.ac.uk/jspui/handle/123456789/4630>

Information about this research object was correct at the time of download; we occasionally make corrections to records, please therefore check the published record when citing. For more information contact scholarlycommunications@qmul.ac.uk

Phd thesis: *Data-parallel lazy functional programming*

701

Jonathan M. D. Hill (email Jonathan.Hill@comlab.ox.ac.uk)
Department of Computer Science
Queen Mary and Westfield College
University of London

September 1994

Supervisors: Richard Bornat, Keith Clarke, Heather Liddell, Peter Flanders (Cambridge Parallel Processing).

Contents:

- [Abstract](#)
- [Online copies of the thesis](#)

Abstract:

The essence of data-parallelism is a constant complexity map function. A data-parallel interpretation of map is the application of a function to every element of a parallel data structure at the same time. This model is at odds with a version of map over lists. Although list map can be interpreted as applying a function to every element of a list, in a non-strict functional language the function applications only occur to those elements of the list required by a subsequent computation. This thesis reconciles these opposing views of map using a three-tiered model: (1) a non-strict data-parallel evaluation mechanism based upon "aims" is used that combines the "only evaluate what is required" philosophy of non-strict evaluation, with the "evaluate everything synchronously, and in parallel" mechanism of a data-parallel paradigm; (2) program transformations inspired by the map distributivity law are used to vectorise functional programs that contain map; (3) the resulting vectorised programs are compiled into machine code that mimics an abstract machine based upon the Spineless Tagless G-machine. The novel features of this machine are that it incorporates the "aim" mechanism of data-parallel evaluation, and case analysis of algebraic data-types is vectorised by performing tag-checking in parallel.

As well as describing how to implement a data-parallel non-strict function language efficiently, this thesis also describes extensions to the non-strict functional language Haskell that enable data-parallel algorithms to be expressed. We describe s, unbounded parallel data structures that share many of the characteristics of Haskell arrays. We present comprehensions, a framework within which communication and parallel operations on s can be expressed. Development of the higher order parallel map, fold, and scan is presented, a trio of functions that is fundamental to a data-parallel paradigm. Unlike other parallel implementations of these functions, particular attention is given to their non-strict nature.

Online copies of the thesis:

1. [LaTeX_{HTML} copy of the abstract, introduction, and conclusions to the thesis](#)
2. [A single gzipped Postscript file of the thesis](#) (300dpi, 519 Kbytes)
3. [A single compressed Postscript file of the thesis](#) (300dpi, 662 Kbytes)
4. [A single gzipped Postscript file of the thesis](#) (600dpi, 618 Kbytes)
5. Gzipped Postscript files split by chapter:
 - [Cover page](#) and table of contents
 - [Chapter 1](#): Introduction
 - Parallel graph reduction

- Data parallelism
 - Liberation from the Von-Neumann bottleneck
 - Overview of the thesis
 - **Chapter 2:** The *aim* is laziness in a data-parallel language
 - An introduction to the aim of evaluation (contrast with print eval loop, activity mask of a SIMD machine, an operational view of the aim, and data-parallel non-strict glue).
 - PODs and POD comprehensions
 - Comparisons
 - The semantics of POD comprehensions
 - **Chapter 3:** A denotational semantics for data-parallelism
 - **Chapter 4:** Fundamental parallel algorithms (scan, fold, linear recurrence, packing, sorting, segmented scan, sending with collisions).
 - **Chapter 5:** Vectorising a non-strict language
 - **Chapter 6:** A data-parallel STG-machine
 - **Chapter 7:** Example programs in DPHaskell
 - Parallel LL(1) parsing
 - Word searching
 - **Chapter 8:** Implementation and results
 - **Chapter 9:** The vectorisation monad
 - **Chapter 10:** Conclusions and further work
 - **Appendix A:** Static semantics
 - **Appendix B:** An introduction to Monads and Category theory
 - **References and index**
-

Jon Hill