UNSW AUSTRALIA

GRADUATE RESEARCH SCHOOL

EXAMINER'S REPORT FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

STRICTLY CONFIDENTIAL

Name of Candidate: Trevor McDonell
School: School of Computer Science and Engineering
Title of Thesis: Optimising purely functional GPU programs
Report Due Date: 28 October 2014

1	**After examination of the thesis (and supporting papers) I recommend that:** **(Please circle the letter next to the appropriate recommendation)**

A. The thesis merits the award of the degree.

This recommendation is appropriate if the thesis contains no faults that are apparent to the examiner. It is also appropriate where errors and omissions of an editorial nature are minor and, if left uncorrected, will not alter the conclusion that "the thesis merits the award of the degree".

or

(B) **The thesis merits the award of the degree subject to minor corrections as listed being made to the satisfaction of the Head of School.**

The errors and omissions, which extend beyond those of an editorial nature, must be corrected if the thesis is to merit the award of the degree. The corrections are minor in that they do not change the structure or the conclusions of the relevant chapters of the thesis.

or

C. The thesis requires further work on matters detailed in my report. Should performance in this further work be to the satisfaction of the Faculty Higher Degree Committee, the thesis would merit the award of the degree.

The further work required should be sufficiently straightforward such that the examiner is happy to delegate approval of the revised thesis to the Higher Degree Committee. Examples of further work in this category could include: discussion and consideration of published work that is relevant to the conclusions of the thesis; consideration of alternative hypotheses that should reasonably be suggested by the candidate; presentation of additional experimental data that could be expected to be in the possession of the candidate; clearer specification of how the presented results/conclusions were arrived at.

or

D. The thesis does not merit the award of the degree in its present form and further work as described in my report is required. The revised thesis should be subject to re-examination.

Please indicate whether you are willing to review the resubmitted thesis Yes [ ] No [ ]

The further work involves a major revision of the thesis on the same topic. The examiner is assumed to be satisfied with the candidate's capability and demonstrated competence for this further work. The comments and suggestions in the detailed report should be clear and helpful to the candidate. As the thesis is to be revised along the lines suggested by the examiners, it would normally be re-examined by the same examiners. Examples of further work in this category could include: further analyses or experiments where the scientific method as presented in the thesis has significant flaws; performance of additional experiments that are deemed vital to the conclusions drawn in the thesis.

Or

E. The thesis does not merit the award of the degree and does not demonstrate sufficient ability by

the candidate for a resubmitted thesis to achieve this merit.

The examiner should provide the basis of this recommendation in the detailed report.

2	**I agree to my name being released to the candidate**	**Yes** [X] **No** [ ]

3	**An Evaluation of the merits of the thesis is given overleaf (Attach additional pages if required)**

The examiner is requested to state concisely the grounds on which the recommendation is based, indicating where appropriate the strengths and weaknesses of the thesis. To assist the candidate with future work, this report may be released after a decision has been taken to award or not to award the degree.

[redacted]

Report on Trevor McDonnell's Ph.D. Dissertation

This document presents comments on Trevor McDonnell's Ph.D. Dissertation, which is titled "Optimising Purely Functional GPU Programs." The comments are meant as suggestions for improvement, such as adding missing citations to related work and clarifying parts that are hard to understand. I encountered a few minor grammatical errors in my reading, but I have not included those in the report.

Overall comments

Overall this dissertation is a great piece of work that makes a significant contribution to the field of Parallel Programming Languages. It provides a end-to-end description of the Accelerate embedded DSL targeted at Nvidia GPUs. Trevor's Ph.D. research breaks new ground in a number of distinct areas as evidenced by the number of publications that have already been produced from his work. I want to particularly call out Chapter 6, which is the experimental evaluation of Accelerate. I think that the experiments described in this chapter are very thorough and well presented.

As detailed below, there are a number of minor corrections and improvements that I think can be made to the dissertation, but these do not detract from its quality and significance.

Chapter 1 — Introduction

No comments.

Chapter 2 — Basics

Describing GPUs as SIMD architectures (page 7) is not accurate and obscures the important issue of divergent control flow in the GPU execution model.

There is a very detailed and comprehensive technical report on Parallel Scans for GPUs that should be referenced. The reference is

> "Parallel Scan for Stream Architectures," by Duane Merrill and Andrew Grimshaw. Technical Report CS2009-14, Department of Computer Science, University of Virginia. December 2009.

The discussion of Segmented Scan (page 12) lacks context. The concept and representation of segmented sequences has not been introduced. Also, there is no discussion of segmented reductions, which are also part of the NDP toolset.

Chapter 3 — Embedded array computations

Change "vector library" to "Haskell vector library."

Perhaps “task parallel” is a more standard term than “control parallel” (page 21)?

Should “rank polymorphic” be “shape polymorphic” (page 22)?

Does accelerate's fold operation really reorder elements? Usually, associativity is sufficient for parallel array computations, since the array indices impose an order on the elements.

The operations in Listing 3.2 deserve more explanation. For example the semantics of backpermute should be defined, since it is used in a later example, and it is not a widely used operation outside of DP programming.

The PreOpenExp type is used on page 31 without explanation (the "Pre" types are not described until the next chapter).

A mention of, or comparison to, the work on commutative associative rewriting (some of it even published in JFP!) would be appropriate on page 33.

Overall, Section 3.2 could use some examples and an overview of the implementation architecture. It is great that there is a lot of detail about the implementation, but it is hard to see the big picture from just a lot code snippets from the implementation, since it is not clear how they fit together.

Chapter 4 — Accelerated array code

Given the importance of quasiquotation and Template Haskell to the implementation, some additional explanation of this infrastructure is in order. For example, does this mechanism provide a term representation of the CUDA code? Is the splicing of code into templates handled by Accelerate or by Template Haskell?

It is nice to see examples of the templates presented, but it would be even better to see one or two examples of the expanded templates (i.e., the final CUDA code) as well.

The example on page 53 uses backpermute; as mentioned above, it would be good to explain this operation so that the example makes sense.

Also on page 53: “patters” should be “patterns”.

Because of the lack of details about how skeletons are instantiated, I do not understand how generated variables are named. A common technique is to use unique stamps (gensym) to avoid accidental name capture, but that would seem to interfere with the hashing of the kernels so that they can be reused. I would like to see this question addressed.

Is there any empirical evaluation of the runtime optimization techniques? For example, how common is sharing of kernels across multiple Accelerator expressions? Do you have a measurement of the performance benefits of loading from the persistent cache (vs compiling from scratch)? I assume that it saves significant time, but how does it compare to the execution time of the program?

Does Accelerate provide any sort of explicit deallocation of device memory? Relying on finalization for timely resource recovery is known to be problematic, since GC efficiency

often requires postponing identifying dead objects.

Is Accelerate able to update in place, or is the only reuse of device memory tied to the host-side object being deallocated?

In Section 4.5.5, you might cite the work on piecewise NDP as a way to reduce memory pressure (e.g., Keller's work on distributed types, Pfannenstiel's work on piecewise NDP, and the more recent work by Madsen and Filinski).

It seems to me that Section 4.6.2 might better be part of Chapter 3.

Can you provide more details about your occupancy calculation (Section 4.6.4) and how you use that information in specializing kernels and specifying the runtime block sizes *etc*?

On page 76 (footnote 13) "user user" should be "user."

I would have included Ct, RapidMind, etc. as Embedded DSLs; I do not see why they would be classified as libraries, when Accelerate, Nikola, etc. are classified as EDSLs.

The system that you call "NDP2GPU" is actually called "Nesl/GPU" in the paper (The "NDP2GPU" name was used for an earlier version of the project). Also, Zhang & Mueller independently on their own GPU version of Nesl called CuNesl (ICPP 2012).

Chapter 5 — Optimising embedded array programs

Is Figure 5 in color (it is black-and-white in my copy)? The labels on the nodes are small and very faint, making it hard to follow the details of the figure.

On page 83: I'm curious why applying CSE to the terms would not be a more effective way to recover sharing (and to identify possible additional sharing)?

On page 84: you state that Phase 1 is O(n); does this means your occurrence map has a constant-time insert/lookup?

The use of the standard terms "Producer" and "Consumer" in Chapter 5 is very non-standard. I understand the desire to distinguish between fusing two maps and fusing a map with a reduction, I think that it would be better to invent new terminology instead of redefining existing terminology.

On page 104: is the text "⟨ **common subexpression elimination** ⟩" meant to be included in the code example? Also "can not" should be "cannot." The description of the limitations of your implementation of CSE are unclear; does it mean that you only eliminate common subexpressions where the defining subexpression is already let-bound?

Have you evaluated the cost and/or benefit of the various optimization techniques?

How do the techniques used for fusing loops in imperative programming languages that are described in Section 5.4.2 compare with the techniques used by Accelerate?

Again, "NDP2GPU" should be "Nesl/GPU."

Chapter 6 — Results

Can you include a summary of the benchmark programs at the beginning of Chapter 6?

Section 6.1.3 is a natural place to include an evaluation of the benefits of the persistent cache. What is the overhead if you load binary kernels from disk instead of compiling them using **nvcc**.

There is an inconsistency between Sections 3.1.1 and 6.2.3; in the former you require the operator of a fold to be associative and commutative, while in the latter, you only require associativity (the weaker requirement is okay for arrays).

With respect to the issue of unbalanced loads for applications such as Mandelbrot, I think the work on persistent threads is more applicable than work stealing. See, for example, by Gupta, Stuart and Owens.

> "A Study of Persistent Threads Style GPU Programming for GPGPU Workloads," by Gupta, Stuart and Owens. Innovative Parallel Computing, May 2012.

Chapter 7 — Conclusion

No comments.