[redacted]

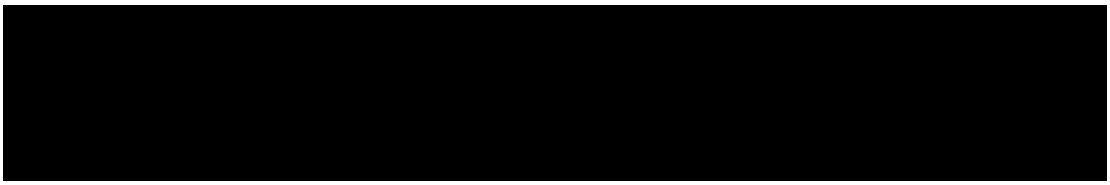
Examiner's report on the thesis "Optimising Purely Functional GPU Programs" by Trevor L. McDonell, School of Computer Science and Engineering, UNSW

Trevor L. McDonell's thesis represents a significant contribution to knowledge in the area of programming of graphical processing units (GPUs). The thesis builds on earlier work by Lee and others on the Accelerate system for programming GPUs from the functional programming language Haskell. McDonell's contribution to the work is both clearly documented and considerable. In my opinion, the amount and quality of the work is well above the bar for the award of a doctorate. Indeed, I consider this to be one of the best theses that I have ever read.

The work is timely in that the question of how to program GPUs and other manycore architectures is a pressing one. Current approaches usually demand programming directly in CUDA or other very low level languages, and this is difficult, expensive and time consuming. McDonell's work has clearly demonstrated that programming these architectures from Haskell is a viable alternative; in particular, his work identified and fixed performance-related problems in earlier work on Accelerate. This demanded the building of a sophisticated backend with dynamic code generation based on algorithmic skeletons, as well as the addition of a new approach to fusing sub-computations, so that fewer kernel calls and expensive host-device transfers of data are needed. Much of the work in the thesis is rather technical. The building of a sophisticated code-generating backend of necessity demands that attention be paid to many small details, and describing such work is not an easy task. But McDonell manages to write an engaging and readable thesis, in which the various steps to building up the whole system are carefully explained. Code is included to keep the presentation concrete, and it is explained both clearly and concisely. Often, the code is beautiful! In general, the quality of the exposition is excellent. The thesis was a pleasure to read.

The work presented has been published in a paper at a workshop (DAMP'11) and at the International Conference on Functional Programming (ICFP'13), the main conference in the area of functional programming. The latter paper has McDonell as first author, and he presented the paper at the conference in a very well-received presentation (which I saw). I note that ICFP is a highly competitive conference and

[redacted]



that an ICFP publication ranks above journal papers in our field. Thus, the work of the thesis has been presented at a top conference, which is a strong indication that it is a significant contribution to knowledge. In addition, a recent book on Parallel Functional Programming by S. Marlow included a chapter on Accelerate, indicating both that the system has reached a certain level of maturity and usability and that it is attracting attention from other researchers and practitioners.

The work presented is highly original. I don't know of any comparable skeleton-based system for GPU programming. The presentation includes ample evidence that the author is familiar with related work over an impressive breadth of topics, and that he has made careful judgements when choosing his own approaches to the problem of how to combine a high level of abstraction for the programmer with the generation of high performance code. The new fusion method presented is elegant. It will very likely prove to be of use in other settings than that described in the thesis.

It is very clear from the thesis that the author is extremely knowledgeable about related work. He shows mastery of a wide range of techniques used in the implementation of embedded domain specific languages, and in associated backends and runtime systems. Given this, I found the discussion of related work in some sections of the thesis to be disappointingly brief, sometimes verging on desultory. Often, important related work is mentioned only in a sentence or two of fact, but without the additional information that would really allow the reader to understand its relation to the presented methods. An example is the reference to work on Delite and Lightweight Modular Staging on page 78. This work, which (at least in the form of OptiML) also targets GPUs using a staged language is closely related to that of the thesis and also emphasizes fusion techniques. An excellent thesis would have been made even better had the author included his technical *judgement* of related work such as this, along with further explanation of how it relates to the presented research. Scientific papers tend to hold themselves to facts and to have constraints on length so that related work is only briefly discussed, but a PhD thesis can benefit from the inclusion of a deeper discussion of some selected related work (at least in the case of a very strong thesis such as this one).

The author's evaluation of his final system through benchmarks is very well done. Initial small benchmarks are well chosen to highlight the usefulness of particular optimisations. In addition, good decisions are made when choosing what to compare to (so that the comparisons give useful information and don't fall into the trap of comparing only with the author's own earlier work). The chapter on benchmarks

would have been improved by a deeper discussion at the end, which returned to the question of what is gained by the various optimization methods presented, most particularly the new fusion method. I would have liked to see a listing of the various techniques being studied and a discussion, for each technique, of how it fared in the set of benchmarks. This information is there in the chapter, but it is spread out, and it is hard for the reader to gain an overview (and I have no doubt that the author of the thesis could provide much insight in such a summary). When I read this chapter, I am left wondering about the extent to which fusion helps, for example in the examples with most kernels (fluid flow and Canny). A summary of the kind that I suggest could clarify this. (A separate document suggests some small changes to the thesis. However, I note that I consider that the thesis clearly deserves to pass even in its current form.)

In similar vein, there are some places in the thesis where future work is proposed only very briefly and where I would be very interested to read some more details about the candidate's ideas for how to proceed. For example, a cost model for making various decisions about how to generate good code is mentioned more than once. A little more detail on how this might be done, perhaps including some references to related work, would have been enlightening.

I would be grateful for a hard copy of the final (possibly slightly revised) thesis as my current copy has fallen apart!

In summary, Trevor L. McDonell has made an original and significant contribution to knowledge in the area of programming of GPUs, a new and important topic. He has shown a very good understanding of the literature and has explained and documented his own choices in exemplary fashion. The work has been published in the main conference in our field and it is already of benefit to a wider research community.
I conclude that the thesis very clearly **merits the award of the degree of Doctor of Philosophy**.

Thanks for a great thesis and a very enjoyable read.

Typos /small errors that should be fixed

p 8

Each step S does N/S^2 independent operations

should be N / 2^S

p 135

Listing 6.7 doesn't seem to be complete, in that I can't

see how the different computations are finally composed??

p 155

I can't make sense of

(1 - xy)^1 x -> x (1 - yx)^(-1)

The references need some work to make them more complete and useful.

For example:

Several tech report references are incomplete (e.g. 11,24,47).

ref 30 just says page 21. Omit or put in the range.

I don't know why several refernces say "ACM Request Permissions"

ref 71 fusio -> fusion

Simon Peyton Jones should not become Simon P. Jones but
should be under Peyton (as in refs 89 and 90).

Questions that arose in my mind while reading the thesis

In our experiments with both CUDA and Obsidian, we have found that it is
often necessary (for decent performance) to introduce "sequentialness" into kernels, moving
away from one element per thread. I believe that you have also done this, at
least in the implementation of fold (judging from some somewhat vague comments you
made in the thesis). But you don't seem (in the thesis) to explicitly provide (say)
small sequential functions that read sub-parts
of an array and/or produce subparts of an output array. I see on github that there
is mention of szipWith, sfoldl and similar but you didn't mention
them in the thesis (or I missed the mention). And your implementation
of the map skeleton seems to use one element per thread. Is that
correct? Could you comment somewhere on this question and how you
think adding sequentialness is best done (if indeed you agree
that it is important at all)? I presume that you want to protect
the user from having to worry about this?? Perhaps your proposed
cost model is going to provide the answer?

I would guess that the array of structs to struct of arrays transformation is very important.
Is this demonstrated in any of the benchmarks? (If it is, I missed it.)

I am guessing that none of your benchmarks is actually very representative of
the ideal use of Accelerate (which I imagine to be acceleration of parts of
"ordinary" Haskell programs, which would then run sometimes in multicore
and occasionally on GPU). Could you comment on this? What is really
your killer app?

Suggestions or comments(minor)

p 6
expressing problems that can be expressed as data parallel computations ...
perhaps expressing -> executing

p 10
The mention of shape (Z :. 1 :. n) may be confusing (not yet introduced).

p 11
It would be a good idea to make clear the different between inclusive
and exclusive scan

p 11
I have difficulty matching the picture in Fig. 2.3 with the written
description of how scan is implemented. The picture seems to be of
a reduce-then-scan type implementation (in the Merrill terminology)

but you describe an implementation in which you start with scans (but
throw away all but the last of the resulting sub-arrays (I believe)).
The explanation for why you need these initial scans (without keeping
the entire scan results) is very unclear and should be improved.
Why could you not have directional reduce operations too?

I would expect a reference to Merrill and Grimshaw's tech report
about scan on GPU:
http://www.cs.virginia.edu/~dgm4d/papers/ParallelScanForStreamArchitecturesTR.pdf
It is only a tech report but is very informative and probably a useful
reference for readers of your thesis.
(I think that some of the same material used to appear on the CUB pages at NVIDIA,
including some nice pictures of scan decompositions,
but I can't see it now, so I would guess that referring to the tech. report still
makes sense.)

p 12
The mention of array fusion and delayed arrays here may be difficult to
follow for readers not already familiar with these ideas.

The discussion of two possible options in the sub-section called Interaction with array fusion
is not at all clear or conclusive. It is hard for the reader to figure out what the current
implementation does (especially since he does not yet know
what producer/consumer fusion is). The final sentence "Currently the user has no control ..."
is uninformative.

p 13

Mention of backpermute in the explanation of permute assumes that the reader already knows what backpermute is.

p 19

I might have expected mention of the fact that combinations of deep and shallow embeddings are possible (e.g. ala Svenningsson and Axelsson). Indeed, you have done this yourself, for example in how you deal with filter.

p 23

I don't understand why implementing a tree reduction demands commutativeness of the operator? Is it because you want the reduction to actually combine the elements in a way that is different from Figure 2.2 (for example to improve the access pattern)? If so, I think this should be made clearer (perhaps in the previous chapter).

p 38

You say "None of these embedded languages encode the types of the source program into the embedded

language terms to the degree with which Accelerate does" and I don't doubt this.
But could you please be more precise (somewhere) about the advantages of keeping the types around for longer. (I think the advantages get mentioned in a couple of places (e.g. on p. 87), but they are not gathered (I believe) and I think that would be good.)
Also encode -> encodes (after use of none).

Recent versions of Obsidian are very much less restrictive than the original 2008 paper, so please consider referring to Svensson's thesis (2013). (Of course I have a conflict of interest here.)

p 62-63

Very cool association of host and device arrays and use of nursery! (just a comment)

p 86

"Since filter-like operations are not part of the core language, we do not need to consider them further." Fair enough! But the reader would still like to know how filter operations turn out (performance-wise) in practice. (I don't remember them being mentioned in the benchmark sections.) Perhaps they have just not been much exercised (or needed)?

p 89

When discussing delayed arrays, I would expect a reference to Pan (Elliot et al, JFP 13(2), 2003). (You point out more than once that the idea is well known, and I think this is a good, early reference.)
I am fine with calling them Repa-style delayed arrays (as on p. 109) if you wish, but they were in Obsidian from the start too (which was 2008) and Pan was the inspiration.
Apparently the idea dates back to early APL implementations, see Guibas and Wyatt from POPL'78, so including that reference may make sense too. (I can't get hold of it right now to check that.)

p 98

.. as the delayed array constructors always refers -> ... refer

p 99

with when -> when

p 103

return the number of uses -> returns the number of uses

p 106

No explicit checks for underflow or overflow is made -> ... are made ...

p 109

and helps to compile -> and help to compile

p 100

suitable execution on -> suitable for execution on

p 115

recommend -> recommended

p 116

.. that the complexity of the optimisations are not exponential -> ... is ...

p 116

It is not clear (here) Nodes means in the graph in Fig 6.1.

p 120

I don't fully follow the discussion about the use of Int and

64 bit machines. Would it have been very difficult to include

Int32 as an explicit type? Or do you think that is asking too

much of the user?

In any case, I am really curious to know int32 in the code you

showed would have on speed (if necessary just by doing a hand edit...)

Would you catch up with CUBLAS then?

(I feel that my curiosity is not being satisfied here.)

p 122

"in parallel as map operation" reads oddly

p 130-131 Listing 6.4

I am unable to tell exactly why this code uses shared
memory on the GPU. Is it because of the 2-step nature of fold's
skeleton implementation? Please clarify.

p 131

Measurement in Sec. 6.6 is only of core sparse matrix multiply (which is fine).
But I am still curious about host-dev and dev-host transfers. How much time do
they take both in Accelerate and in CUSP and CUSPARSE?

p 137

Please clarify why the sequence of stencil operations doesn't fuse.
Because they are all consumers?

p150

Discussion too brief here (see report). I would have liked to see a summary
of the effects of your innovations (sharing, fusion, even iterate etc.).
I realise that the info is here, but it is hard for the reader to grasp. Having you summarise it and give
your opinions on the results and prospects for further improvement would
be really interesting.

p 153

breath -> breadth