Study Guide (Version 2)
Parallel Functional Programming
Chalmers

Mary Sheeran

May 26, 2013

Abstract

This brief guide aims to help you structure your study of parallel functional programming, relying on the material linked to on the course home page. The course covered quite a few topics and had many guest lectures. Here, we try to point to the important topics and stress the important ideas, not only for the immediate task of passing the exam, but also for your longer term learning and enjoyment. Dislaimer: This is an experiment! I can't guarantee that I have not forgotten something important. Feedback and tips are welcome! I have linked to a couple of your submitted tutorials and I may add more. So I will just have to keep tweaking this document. Good luck!

The single sentence that has been added to Version 1 is highlighted in red.

1 Introduction

The first thing to note is that we (the examiners) are, believe it or not, actually serious about the course Aim, Syllabus and Learning Outcomes! They are on the main course page and are reproduced here:

1.1 Aim

The aim of the course is to introduce the principles and practice of parallel programming in a functional programming language. By parallel programming, we mean programming using multiple hardware cores or processors in order to gain speed. The course covers approaches to parallel functional programming in both Haskell and Erlang. It covers current research on these topics, and relies heavily on scientific papers as its source materials.

1.2 Syllabus

- The course covers the principles and practice of parallel programming in both Haskell and Erlang.
- Advantages of functional approaches to parallelism: immutability, absence of data races, determinism.
- Profiling parallel functional programs: granularity, bottlenecks, locality, data-dependencies.
- Parallel functional algorithms: divide-and-conquer.
- Approaches to expressing parallelism in Haskell: the Eval monad, the Par monad, parallel strategies, skeletons, data parallelism.
- Functional approaches to GPU programming
- Parallelisation and distribution for Erlang. Scalability. Handling errors in a massively parallel system.
- Case studies of industrial parallel functional programming, such as map-reduce and scalable no-SQL databases.
- Guest lectures by leading researchers and practitioners.

1.3 Learning Outcomes

- Knowledge and understanding
  - Distinguish between concurrency and parallelism.
  - Give an overview of approaches to parallelism in functional programming languages in the scientific literature.

  Skills and abilities
- - Write, modify and test parallel functional programs, to run on a variety of architectures such as shared memory multiprocessors, networks of commodity servers, and GPUs.
  - Interpret parallelism profiles and address bottlenecks.
- Judgement and approach
  - Identify when using a functional language may be appropriate for solving a parallel programming problem.
  - Select an appropriate form of parallel functional programming for a given problem, and explain the choice.

2 Last year's exam

First, reassure yourself by looking at last year's exam. As you can see, it does not contain surprises or tricky questions. Also, it reflects the above learning outcomes pretty closely!

This year, there will be a question or part question about Obsidian (which was not covered in the 2012 version of the course). Oh, and cache complexity is also a new topic for this year (see section 3.6).

3 A possible study plan

3.1 Motivation, determinism, purity etc.

Look through the slides of lecture 1 the first lecture, which motivates the whole course and the advantages of choosing a purely functional approach. John makes the importance of immutable data pretty clear :) Note how important it is to control task granularity. The idea of introducing a depth parameter to control granularity appears first in this lecture and then a few times later on. You probably got told about this by Nick in his comments on the first lab too. At this point, it would be a good idea to look at your first lab and the comments you got on it, and think about the control of granularity. The first three sections of Simon Marlow's lecture notes also fit well here. Note the discussion of data dependencies at the end of section 3. Also, Marlow distinguishes between concurrency and parallelism in exactly the way that we want to.

3.2 Threadscope and the like

Remind yourself, too, about how to use +RTS -s and Threadscope to investigate the effects of getting the granularity wrong or of having sequential blocks that ruin your parallel performance (see Amdahl's law at the end of lecture 1). Andres Löh's slides from 2012 give a useful summary of the most usual pitfalls in parallelising functional code and of how to investigate problems (even without being able to see the Threadscope Demo that he did live in 2012).

3.3 Haskell's palette of methods

Next, you need to consider the sequence of approaches to deterministic parallel (multicore) programming in Haskell (par and pseq, strategies, the Eval and Par Monads, and Repa). Look at the slides for lecture 2, lecture 3 and lecture 9. Read the papers (the two strategies papers, the Par monad paper and two Repa papers), paying attention to the advantages and disadvantages of the various approaches. Simon Marlow's notes are again very useful (sections 4 to 6). This tutorial about Repa by Eddeland and Söderlund from this year's class captures Repa nicely. Students in the class have also contributed informative tutorials about the par monad and about strategies. Data Parallel Haskell was also mentioned in lecture 9. For now, we consider that it is not quite ready for the big time, and this is confirmed by this (long) tutorial by Lypai, Katelaan and Blöndal. Maybe next year...

Look at your lab work that uses the various approaches. How would you argue for any of these approaches if asked to advise on their use in a real project? And what are the downsides of these approaches, both those stated in the papers and those that you have experienced in practice?

Obsidian is a bit different, both because it is local and very much under development and because it is for GPU programming. Look at the slides for lectures 5 and 6. It is the user's perspective that we want you to concentrate on. What can the user control? Again, look back at your lab work and Joel's comments.

3.4 Parallel Patterns, Skeletons, Divide and Conquer

The invited lectures by Hammond and Berthold introduce the idea of parallel patterns or skeletons. Hammond argues for a lightweight approach built on `par` and `pseq`, while Berthold presents Eden, with its "pragmatically impure" notion of explicit processes for coordination. Berthold presents the idea of skeletons as an abstraction of algorithmic structure as higher order functions, with embedded workers contributed by the application programmer and a hidden parallel library implementation done by the system programmer. Divide and Conquer features heavily in a variety of approaches (see for example the "Better Strategies" paper from Haskell'10 as well as the above invited lectures). Jean-Philippe's lecture on parallel and incremental parsing was also a lovely use of Divide and Conquer! Study this pattern, particularly if you did not make the choice to use it in Lab A. And parMap is ubiquitous, coming up nearly everywhere! It is probably the most commonly used pattern (even in the real world, as Lennart Augustsson told us).

3.5 Data Parallelism

The distinction between task and data parallelism came up several times, and lectures 8 and 9 were devoted to data parallelism. This includes DPH and Repa (mentioned before). This section of the course is mainly devoted to the ideas of Blelloch, which you will also have explored in Lab C. Read as much as you can about Blelloch's notions of *work* and *depth*, making use of the links that appear in the notes for lecture 8. Maybe watch Blelloch's invited talk from ICFP'10? You can't read Blelloch's work without coming across parallel scan and why it is important. Be prepared to analyse small example algorithms for work and depth (as you did in Lab C). Besides work and depth, the other important idea to note is the distinction between flat and nested data parallelism, which comes up both in Blelloch's work and (for instance) in the arguments for why DPH is interesting (and also difficult to make work).

3.6 Caches

You have just seen Nick's lecture, viewing cache complexity as a further step beyond the simple notions of work and depth. Nick's advice is to read Prokop's MSc thesis (the second ref. on the final slide of the lecture). Nick says it is an easier read with more examples than the classic papers about Cache-oblivious algorithms by Frigo and others (including Prokop). So, first read section 7 of Prokop's thesis, and then sections 2 and 3.

3.7 Erlang, Fault Tolerance

And then there is the Erlang part of the course, about which I know less than I should! But here the revision strategy seems to be less complicated. Look at the slides from John's lectures and from the invited lecturers. Probably your lab work in Erlang is very fresh in your mind... What distinguishes the Erlang approaches from those that you saw in the Haskell part of the course?

3.8 Your opinion

It is important to think about your experiences with programming using the various approaches presented in the course. We expect you to have personal opinions based both on your own experience and on what you have read.

In order to try to improve the course for next year, we have added extra questions to the usual Chalmers questionnaire (working with the class reps). Please please fill in the questionnaire. Or if you prefer, just send a mail to Mary, Nick or John with ideas and concrete suggestions.

3.9 What Next?

The course is hopefully about more than just passing the exam. To keep up with developments in topics related to the course, look for the following conferences and workshops in the ACM Digital Library: ICFP and the Haskell Symposium (which should be familiar by now), the Erlang Workshop, DAMP (Declarative Aspects of Multicore Programs) and CUFP (Commercial Users of Functional Programming). There is also a new arrival on the scene, FHPC (Functional High Performance Computing). On the Erlang side, both the Erlang User Conference (usually in Stockholm) and Erlang Factory are good sources of tips if you want to work with Erlang.

If all of this makes you wonder, even vaguely, about a Masters Thesis project in this area, talk to Mary or John. We can set you up with a project and supervisor tailored to your interests. We may also be able to help with finding PFP related projects in industry. If the course has made you interested in research, we will be happy to talk to you about that too, and to help you make contacts within the department or in other places. We are very well connected. Don't hesitate to contact us!