CHALMERS UNIVERSITY OF TECHNOLOGY

DAT280 Parallel Functional Programming Lab C

Johan Swetzén, 890916-5576
swetzen@student.chalmers.se
Sebastian Lagerman, 890608-5017
seblag@student.chalmers.se

May 11, 2014

INTRODUCTION

We found it very difficult to implement the stock market problem in repa. We did not manage to get anyway near the speeds of the sequential version, and there are many reasons that could have influenced that. Figure 1.1 illustrates how incredibly bad the parallel version was. We talk about some of the reasons for that in the last section.

SEQUENIAL VERSION

A sequential version that takes only one pass through the list of values was trivial to write. It seemed like we would be able to implement it using fold and so we did, using foldl. When we realised that the fold operator needed to be associative in order to use it with foldAllP in repa,

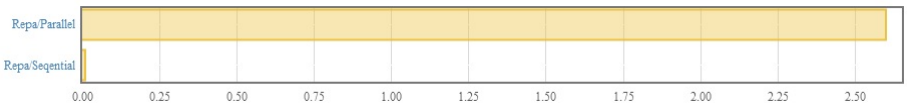

Figure 1.1: Our results of the Repa functionality run on a 8 core machine.

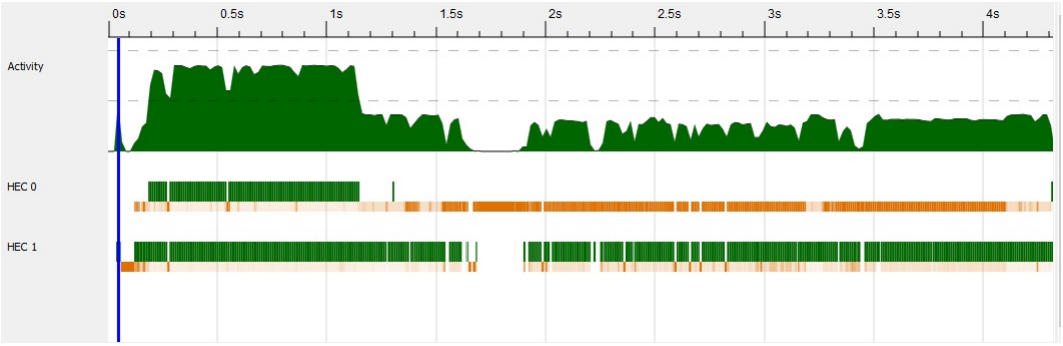

Figure 1.2: The results of our parallel stock market problem is not impressing

we tried to do that but never got it working. The problem is that the fold does pretty much the same thing as the first loop we wrote, except that it keeps track of all the values in tuples that are in the list from the beginning. This is to match repa, which does not allow the operation to foldAllP to return a different value than it has taken in. Our first loop goes through the values from start to end, and that causes problems when we try to go either from right to left (using foldr) or even worse: without a known order as is the case with repa. Folding was a dead end. After a while, we found a way to solve the problem using Blelloch's pre-scan function, and that seemed better suited to implement using repa. The sequential version was fairly straightforward, although it is more difficult to understand than our first version.

PARALLEL VERSION

Implementing a parallel version of the (non-functional) fold solution was very straightforward. The folding operation was the same so apart from a different fold function, only the initial state and the list needed to be changed. In the end it did not work out, however.
Blelloch's pre-scan was more successful in that it actually produced a correct answer. The zipWith and interleave functions from repa could be used, and should in theory produce some parallel work, but in the end we mostly got a lot of garbage collection. This can bee seen in figure 1.2, where there is some parallel work at the beginning but then only parallel garbage collection.

POTENTIAL CAUSES FOR OUR BAD RESULTS

One of the problems with our function was that we never managed to implement a depth value, because our sequential function uses arrays while the parallel one uses a repa array. We found it difficult to change from a normal array to a repa array after the sequential function had returned it's value.
An important factor when writing repa code is that the arrays need to be evaluated at the right time, so appropriate use of computeP and computeS is important. We could not figure out how to do this properly, but we are not sure about how big an impact that made. An

even bigger problem could be if we have introduced nested parallelism by the use of traverse in combination with zipWith. This could be an appropriate place for computeS, but we did not have the time to test that. The function splits and zips the array repeatedly so that it goes down to a depth of $logn$ before it ends up with a single value, so either telling repa to compute sequentially or adding a depth seems important.
We also ended up using only delayed arrays, which was the way we got it to work. Using unboxed arrays might have made a difference, and going from delayed to unboxed at the right time is potentially a large factor in making a repa program perform well.
Another problem we ran into was that neither of our computers (a mac and a windows pc) were able to install llvm which is important for optimising repa code correctly. We cannot tell if this would have made a big impact, but most likely it played some part.