CHALMERS TEKNISKA HÖGSKOLA
Dept. of Computer Science and Engineering

8:30–12:30, Saturday, May 30, 2015.
Parallel Functional Programming
DAT280, DIT261

Exam in Parallel Functional Programming

8:30–12:30, Saturday, May 30, 2015.
Lecturers:
John Hughes, tel 031 681454
Mary Sheeran, tel 031 681454

Permitted aids:
Up to two pages (on one A4 sheet of paper) of pre-written notes. These notes may be typed or hand-written. This summary sheet must be handed in with the exam.
There are 8 questions and 61 points in total.
24 points are required to pass (grade 3), 36 points are required for grade 4, and 48 points for grade 5.

1. **Parallel Functional Programming** **8 points**

(a) Why are functional languages particularly well-suited to parallel programming? **1 points**

(b) What is the main advantage of the *Strategies* approach to parallel programming in Haskell? **1 points**

(c) "After parallelization, any program should be able to run N times faster on N cores." Is this true or false? Explain your answer briefly (for example, with reference to *Amdahl's Law*)c. **1 points**

(d) In the `Repa` approach to parallel programming in Haskell, how does the programmer specify the desired parallelism in his program? **1 points**

(e) Often in writing parallel functional programs one needs to control the size of tasks that are to be run in parallel. Explain one approach to doing this. **1 points**

(f) Haskell and Erlang both use *garbage collection* to recycle memory, but they work rather differently. What aspect of garbage collection may cause a problem in real-time systems, and how does Erlang's VM design mitigate that problem? **1 points**

(g) What is the effect of linking two Erlang processes? **1 points**

(h) What is the purpose of a supervisor? **1 points**

2. **Parallel Sorting** **8 points**

(a) Read this Haskell definition of quicksort:

```
qsort [] = []
qsort (x:xs) =
     qsort [y | y <- xs, y<x]
  ++ [x]
  ++ qsort [y | y <- xs, y>=x]
```

Using `par` and `pseq`, write a parallel version of `qsort`. Ensure that the task granularity is not so fine that the overheads of parallelism dominate the run time. **4 points**

(b) Read this Erlang definition of quicksort (which is simply a translation of the Haskell version):

```
qsort([]) -> [];
qsort([X|Xs]) -> qsort([Y || Y <- Xs, Y<X])
 ++ [X]
 ++ qsort([Y || Y <- Xs, Y>=X]).
```

Using `spawn_link`, `self`, and message passing, write a parallel Erlang version of `qsort`. As above, ensure that the task granularity is not so fine that the overheads of parallelism dominate the run-time. **4 points**

3. **The Par monad** **8 points**

(a) Briefly explain how Marlow's Par Monad builds upon Claessen's Poor Man's Concurrency Monad. **2 points**

(b) Write a parallel divide-and-conquer higher-order function in Haskell for use in the Par monad. If you wish, you may use

```
parMapM :: NFData b => (a -> Par b) -> [a] -> Par [b]
```

4 points

(c) Define merge sort using your divide-and-conquer function. **2 points**

4. **Work and Depth** **7 points**

(a) In Blelloch's cost model for data parallel programming, what do the notions of *work* and *depth* (or *span*) express? How does expected running time relate to work, depth and number of processors? **2 points**

(b) The following is Blelloch's Fast Fourier Transform (FFT) function (for inputs whose length is a power of two), first a general version and then one specialised to complex numbers:

```
function fft(a,w,add,mult) =
if #a == 1 then a
else
  let r = {fft(b, even_elts(w), add, mult):
           b in [even_elts(a),odd_elts(a)]}
  in {add(a, mult(b, w)):
      a in r[0] ++ r[0];
      b in r[1] ++ r[1];
      w in w};

function complex_fft(a) =
let
    c = 2.*pi/float(#a);
    w = {cos(c*float(i)),sin(c*float(i)) : i in [0:#a]};
    add = ((ar,ai),(br,bi)) => (ar+br,ai+bi);
    mult = ((ar,ai),(br,bi)) => (ar*br-ai*bi,ar*bi+ai*br);
in fft(a,w,add,mult);
```

The `w` input is a list of primitive nth roots of unity (the so-called twiddle factors). What are the work and depth for this FFT function, in terms of n, the length of the input sequence? Briefly explain your reasoning. **2 points**

(c) Consider the following variant of the stock market problem: given the price of a stock at each day for n days, determine the biggest profit you can make by buying one day and selling on a later day. A simple sequential (serial) solution requires O(n) work for an input sequence of length n. In NESL, the problem can be solved as follows:

```
function stock(a) =
  max_val({x - y : x in a; y in min_scan(a)});
```

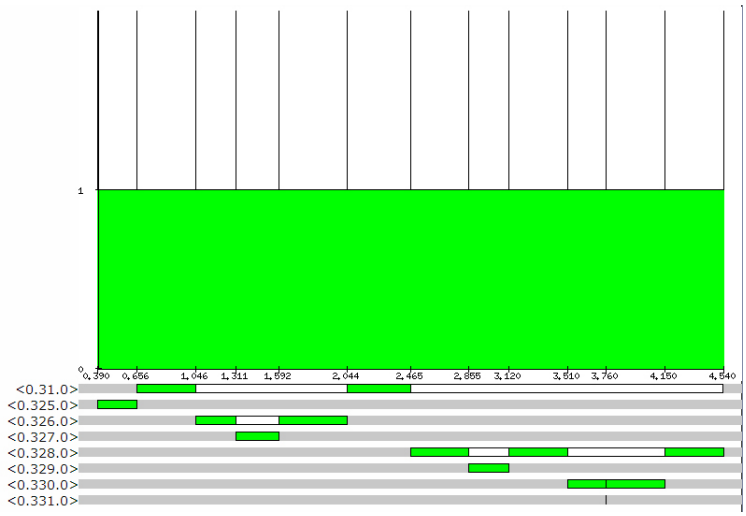
It uses `min_scan` (a parallel scan) and `max_val`, which is a parallel fold. Is the work of this parallel solution still $O(n)$? Explain your answer. What is the depth of the solution? **2 points**

(d) Explain the difference between flat and nested data parallel computations. **1 points**

5. **Parallel Reduce** Read the following definition of a parallel reduce function. **12 points**

```
reduce(_,[X]) ->
    X;
reduce(F,[X,Y]) ->
    F(X,Y);
reduce(F,L) ->
    {L1,L2} = lists:split(length(L) div 2,L),
    Parent = self(),
    Y = reduce(F,L2),
    Pid = spawn_link(fun() -> Parent ! {self(),reduce(F,L1)} end),
    receive
{Pid,X} ->
    F(X,Y)
    end.
```

Profile a call of `reduce` with Percept on a list of 15 elements resulted in the following graph:

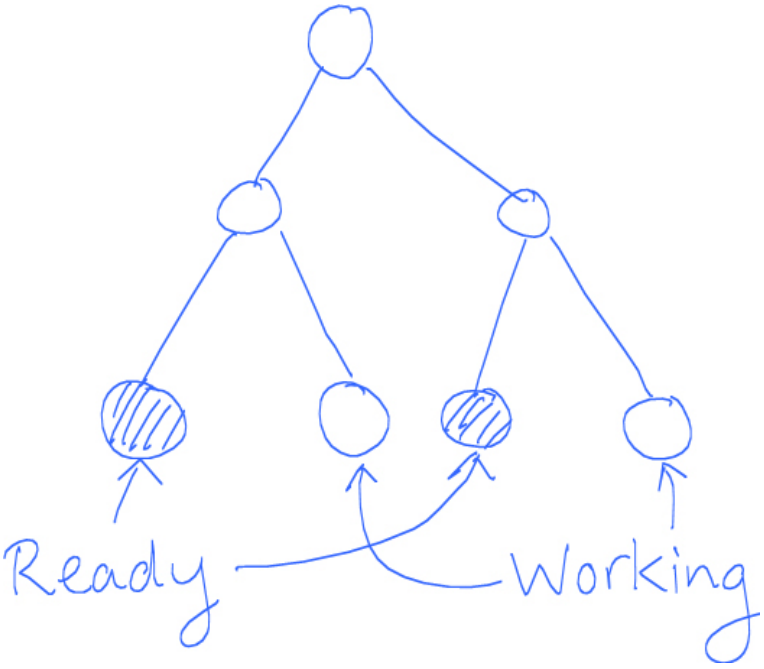


(a) How many processing cores can this version of `reduce` make good use of? **1 points**

(b) Why can't this program take advantage of a larger number of cores? **1 points**

(c) Make a small fix to the code above so that it makes better use of parallelism. You need not copy the entire definition into your answer: just write the modified lines and explain clearly where in the code they should be placed. **1 points**

(d) A weakness of the code above (even after your fix) is that, if recursive calls to `reduce` take widely differing amounts of time, then cores may remain idle even though there is work that could be done. For example, in the situation in this diagram



then two recursive calls are ready, while two more are currently being evaluated. The code above will wait for each busy call to terminate before combining its result with its neighbour; however, a smarter implementation could begin to combine the two available results already to use more parallelism. (Of course, this changes the *order* in which results are combined, but we will assume this does not affect the final result). Write a new version of `reduce` which uses this idea to combine the results of recursive calls as soon as any two are available. **4 points**

(e) Recall that `spawn_link(Node,Fun)` spawns a process that calls `Fun()` on the Erlang node `Node`. In a distributed system, we might want to run reduce jobs on different nodes in the network to share the workload. Write a *load balancing* server, which accepts requests of the form `{call,Pid,F}`, calls `F()` on one of the nodes of the network, and then sends the result back to `Pid` in a message of the form `{result,Res}` (where `Res` is the value that `F()` returned). You should ensure that you can make use of all nodes in your network, but that each node has at most one job to execute at a time. **4 points**

(f) What modification would you make to your `reduce` function to make use of the load balancer you wrote in question 5e? **1 points**

6. **Map-Reduce** **7 points**

(a) `map_reduce` takes a mapper function, a reducer function, and input data as parameters. Consider a naïve version in which the input data is represented as a list. If `map_reduce` were defined in Haskell, what would its type be? You need not include any class constraints, such as `Eq a`, in the type that you give. **1 points**

(b) Suppose the input data to `map_reduce` consists of pairs of a page number and a list of words, such as

```
[{1,["hello","clouds"]},{2,["hello","sky"]}]
```

Write a mapper and a reducer function to convert this to an index of words and page numbers... in this example,

```
[{"clouds",[1]},{"hello",[1,2]},{"sky",[2]}]
```

2 points

(c) Given the same input data, write a mapper and a reducer function to associate each word with its total number of occurrences. Recall that a word may occur several times on the same page. **2 points**

(d) In Google's implementation of Map-Reduce:

i. Explain the difference between local and replicated files.

ii. Which kind of file is used to hold the output of a map job?

iii. Which kind of file is used to hold the output of a reduce job?

iv. What actions are taken by the master node if a worker node crashes?

2 points

7. **Distributed systems and databases** **7 points**

(a) What is the *circuit breaker* pattern in distributed system design, and when should it be used? **1 points**

(b) What is a *network partition*? **1 points**

(c) Brewer's *CAP-theorem* is well known in the context of distributed databases.

i. What do C, A and P stand for? Explain each term briefly (in one sentence). **1 points**

ii. What does the theorem say? **1 points**

iii. Give an example in which a distributed database would be forced to make the choice that the CAP-theorem forces on us.

Distributed key-value stores like Dynamo and Riak use *vector clocks* (or *version vectors*) to track versions of each key-value pair. Suppose we fetch the value of a key K from such a key-value store, and get the value `[1]` with vector clock `[{a,1},{b,1}]`, where `a` and `b` are the names of nodes in the database cluster.

Explain what this vector clock tells us about the history of the key K. **1 points**

Now support that while serving a later request to read the value of key K, the database receives the following three values from three replicas of the key on different nodes:

- `[1]`, with vector clock `[{a,1},{b,1}]`,
- `[2]`, with vector clock `[{a,2},{b,1}]`,
- and `[3]`, with vector clock `[{a,1},{b,2}]`.

Which value(s) will be returned to the client as a result of the call? **1 points**

(d) Assuming that the list values in the question above are intended to represent sets, suggest a way for a client to resolve conflicts. What would your method lead to in the example above? **1 points**

8. **Different approaches to parallel functional programming** **4 points**

Imagine that you work at a company that builds applications that are highly data parallel, targetting several different parallel architectures. The possibility of using Haskell is being discussed, as is the possibility of using Single Assignment C (SAC). Your manager asks for your opinion. Pick either a Haskell library (such as Repa) or SAC, whichever one you would advocate. Write a description of your chosen approach, listing its pros and cons.