CS 457/557: Functional Languages

An Introduction to Control.Parallel

Mark P Jones
Portland State University

A Silly, Slow Program:

```
> fib 0 = 0
> fib 1 = 1
> fib n = fib (n-1) + fib (n-2)

> nfib 0 = 1
> nfib 1 = 1
> nfib n = 1 + nfib (n-1) + nfib (n-2)

> diffib n = nfib n - fib n

> main = print (diffib 38)
```


Why is it Slow?

```
prompt$ ghc --make par.lhs -o par
prompt$ ./par +RTS -s
87403802
prompt$ cat par.stat
16,759,034,836 bytes allocated in the heap
 11,625,744 bytes copied during GC (scavenged)
  2,884,616 bytes copied during GC (not scavenged)
     24,576 bytes maximum residency (2 sample(s))
…
  INIT  time    0.00s  (  0.03s elapsed)
  MUT   time   17.05s  ( 17.25s elapsed)
  GC    time    0.21s  (  0.28s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   17.26s  ( 17.56s elapsed)
…
prompt$
```


Introducing Control.Parallel:

`par  :: a -> b -> b`

`par x y` is semantically just `y`, but hints to the compiler that it might be useful to start evaluating `x`

`pseq :: a -> b -> b`

`pseq x y` is semantically just `y`, but will evaluate `x` before returning a result

A Silly, Parallel Program:

```
> fib 0 = …

> nfib 0 = …

> diffib n = let l = nfib n
>                r = fib n
>            in par l (l - r)

> main = print (diffib 38)
```


Does this Run Better?

```
prompt$ ghc --make -threaded par1a.lhs -o par1a
prompt$ ./par1a +RTS -s
87403802
prompt$ cat par1a.stat
16,759,034,836 bytes allocated in the heap
 11,625,760 bytes copied during GC (scavenged)
  2,884,616 bytes copied during GC (not scavenged)
     24,576 bytes maximum residency (2 sample(s))
…
  INIT  time    0.00s  (  0.00s elapsed)
  MUT   time   16.43s  ( 16.63s elapsed)
  GC    time    0.21s  (  0.28s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   16.64s  ( 16.91s elapsed)
…
prompt$
```

On Multiple Cores:

```
prompt$ ./par1a +RTS -s -N2
87403802
prompt$ cat par1a.stat
16,759,034,636 bytes allocated in the heap
 11,618,096 bytes copied during GC (scavenged)
  2,878,584 bytes copied during GC (not scavenged)
     24,576 bytes maximum residency (2 sample(s))

  INIT  time    0.00s  (  0.00s elapsed)
  MUT   time   16.47s  ( 16.89s elapsed)
  GC    time    0.25s  (  0.33s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   16.73s  ( 17.23s elapsed)

prompt$
```

7

A Different, Silly Program:

```
> fib 0 = …

> nfib 0 = …

> diffib n = let l = nfib n
>                r = fib n
>            in par r (l - r)

> main = print (diffib 38)
```

8

At Last, a Speedup!

```
prompt$ ghc --make -threaded par1b.lhs -o par1b
prompt$ ./par1b +RTS -s -N2 ; cat par1b.stat
87403802
16,759,227,260 bytes allocated in the heap
 12,463,976 bytes copied during GC (scavenged)
  3,158,992 bytes copied during GC (not scavenged)
     28,672 bytes maximum residency (2 sample(s))
…
  INIT  time    0.00s  (  0.00s elapsed)
  MUT   time   16.54s  (  9.30s elapsed)
  GC    time    0.21s  (  0.25s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   16.75s  (  9.56s elapsed)
…
prompt$
```

9

A More Robust, Silly Program:

```
> fib 0 = …

> nfib 0 = …

> diffib n = let l = nfib n
>                r = fib n
>            in par l (pseq r (l - r))

> main = print (diffib 38)
```

10

A More Robust, Silly Program:

```
> fib 0 = …

> nfib 0 = …

> diffib n = let l = nfib n
>                r = fib n
>            in l `par` r `pseq` (l-r)

> main = print (diffib 38)
```

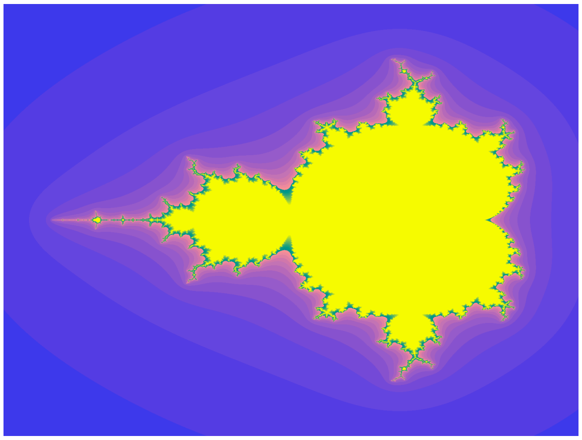
11

Consistent Speedup!

```
prompt$ ./par2 +RTS -s -N2 ; cat par2.stat
87403802
16,759,225,356 bytes allocated in the heap
 12,494,824 bytes copied during GC (scavenged)
  3,179,576 bytes copied during GC (not scavenged)
     24,576 bytes maximum residency (2 sample(s))
...
  INIT  time    0.00s  (  0.03s elapsed)
  MUT   time   16.53s  (  9.66s elapsed)
  GC    time    0.21s  (  0.27s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   16.74s  (  9.95s elapsed)
...
prompt$
```

12

Back to Fractals:



Leveraging Parallelism:

```
> sample :: Grid Point
>           -> Image color
>           -> Grid color
> sample points image
>         = map (map image) points
```

This looks like a good candidate for parallelization …

But how?

Control.Parallel.Strategies:

```
type Strategy a = a -> ()
```

The result of a strategy is always `()`, except that it may do some work to evaluate the argument first

```
using :: a -> Strategy a -> a
```

``e `using` s`` is semantically just the same as `e`, except that it applies the strategy `s`

Control.Parallel.Strategies:

```
class NFData a where
  rnf :: Strategy a
```

`rnf` is a strategy for reducing values to normal form

```
instance NFData Int where …
instance NFData Bool where …
instance NFData a => NFData [a]
  where …
…
```


Control.Parallel.Strategies:

```
parList
    :: Strategy a -> Strategy [a]
```

Evaluate a list in parallel, using the argument strategy for each element.

```
parMap :: Strategy b ->
            (a -> b) -> [a] -> [b]
parMap s f xs
        = map f xs `using` parList s
```


Adopting a Strategy:

```
> sample :: NFData color
>           => Grid Point
>           -> Image color
>           -> Grid color
> sample points image
>        = parMap rnf (map image) points
```

(also need to add an `NFData color` context to the type of `draw`)

Adopting a Strategy:

```
> sample :: NFData color
>           => Grid Point
>           -> Image color
>           -> Grid color
> sample points image
>         = map (map image) points
>             `using` parList rnf
```

(also need to add an `NFData color` context to the type of `draw`)

Adopting a Strategy:

```
> sample :: Grid Point
>           -> Image color
>           -> Grid color
> sample points image
>         = map (map image) points

> draw pal grid render
>  = render (sample grid (fracImage pal)
>            `using` parList rnf)
```


Before:

```
prompt$ ghc --make -threaded parfrac.lhs -o parfrac
prompt$ ./parfrac +RTS -s -N1 ; cat parfrac.stat
8,746,623,328 bytes allocated in the heap
113,302,744 bytes copied during GC (scavenged)
 14,617,944 bytes copied during GC (not scavenged)
    192,512 bytes maximum residency (120 sample(s))
...
  INIT  time    0.00s  (  0.00s elapsed)
  MUT   time    8.38s  (  8.88s elapsed)
  GC    time    0.65s  (  0.69s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time    9.03s  (  9.57s elapsed)
...
prompt$
```


After:

```
prompt$ ghc --make -threaded parfrac.lhs -o parfrac
prompt$ ./parfrac +RTS -s -N1 ; cat parfrac.stat
8,863,473,948 bytes allocated in the heap
180,756,008 bytes copied during GC (scavenged)
 14,648,536 bytes copied during GC (not scavenged)
    352,256 bytes maximum residency (195 sample(s))
...
  INIT  time    0.00s  (  0.00s elapsed)
  MUT   time    9.04s  (  9.56s elapsed)
  GC    time    1.05s  (  1.10s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   10.08s  ( 10.66s elapsed)
...
prompt$
```


After (-N2):

```
prompt$ ghc --make -threaded parfrac.lhs -o parfrac
prompt$ ./parfrac +RTS -s -N2 ; cat parfrac.stat
9,593,542,412 bytes allocated in the heap
355,170,160 bytes copied during GC (scavenged)
 14,272,640 bytes copied during GC (not scavenged)
  1,351,680 bytes maximum residency (335 sample(s))
...
  INIT  time    0.00s  (  0.00s elapsed)
  MUT   time    9.72s  (  5.57s elapsed)
  GC    time    1.71s  (  1.77s elapsed)
  EXIT  time    0.00s  (  0.00s elapsed)
  Total time   11.43s  (  7.34s elapsed)
...
prompt$
```


Conclusions:

- Control.Parallel provides simple mechanisms that can be used to annotate code with hints for parallel execution (and potential speedup on multiprocessor/multicore machines)
- Experimentation may be required to determine best uses for annotations
- Algorithm + Strategy = Parallelism
- Further reading: RWH Chapter 24