Flattening and Replication in Data Parallel Haskell

Ben Lippmeier
University of New South Wales
FP-SYD 2011/05/19

Flat vs Nested Data Parallelism

- Flat Parallelism: Worker function is sequential.

```
thingo xs
  = mapP (\x. x + 1) xs
```

- Nested Parallelism: Worker function is parallel.

```
thingo xss
  = mapP (\xs. zipWithP xs ys) xss
```

- The Flattening / Vectorisation transform converts nested parallelism into flat parallelism.

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys
```

```
fL :: Array Int -> Array Int
fL xs = xs +L (replicate n 1)
 where n = length xs

g :: Array Int -> Array Int
g ys = fL ys
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys
```

```
fL :: Array Int -> Array Int
fL xs = zipWithP (+) xs (replicate n 1)
 where n = length xs

g :: Array Int -> Array Int
g ys = fL ys
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys
```

```
fL :: Array Int -> Array Int
fL xs = mapP (\x. x + 1) xs


g :: Array Int -> Array Int
g ys = fL ys
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys

h :: Array (Array Int) -> Array (Array Int)
h zss = mapP g zss
```

```
g :: Array Int -> Array Int
g ys = fL ys

gL :: Array (Array Int) -> Array (Array Int)
gL yss = fLL yss
```

The Flattening Transform

```
f :: Int -> Int
f x = x + 1

g :: Array Int -> Array Int
g ys = mapP f ys

h :: Array (Array Int) -> Array (Array Int)
h zss = mapP g zss
```

```
g :: Array Int -> Array Int
g ys = fL ys

gL :: Array (Array Int) -> Array (Array Int)
gL yss = unconcatP yss (fL (concatP yss))
```

Replicating Scalars in Uncomfortable

```
 f :: Int -> Int
 f x = x + 1

fL :: Array Int -> Array Int
fL xs = xs +L (replicate n 1)
 where n = length xs
```

Replicating Arrays is Death

```
ys :: Array Int
ys = ...

f :: Int -> Int
f i = ys ! i

g :: Array Int -> Array Int
g xs = mapP f xs
```

```
fL :: Array Int -> Array Int
fL is = replicate n ys !L is
 where n = length is

g :: Array Int -> Array Int
g xs = fL xs
```

Replicating Arrays is Death

```
ys :: Array Int
ys = ...

f :: Int -> Int
f i = ys ! i

g :: Array Int -> Array Int
g xs = mapP f xs
```

```
fL :: Array Int -> Array Int
fL is = replicate n ys !L is
 where n = length is

g :: Array Int -> Array Int
g xs = fL xs
```

Replicating Scalars in Uncomfortable

```
 f :: Int -> Int
 f x = x + 1

fL :: Array Int -> Array Int
fL xs = xs +L (replicate n 1)
 where n = length xs
```

Distribution, not Replication

```
 f :: Int -> Int
 f x = x + 1

fL :: Array Int -> Array Int
fL xs = xs +L (distribute 1)
```