Distributed and Parallel Technology

Algorithmic Skeletons

Hans-Wolfgang Loidl

Heriot-Watt University

[0]Based on earlier versions by Greg Michaelson and Patrick Maier

Part I: More on Skeletons

Skeletons Are Parallel Higher-Order Functions

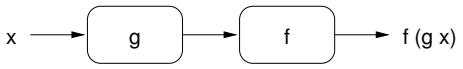
Observations:

- A *skeleton* (or any other template) is essentially a higher-order function (HOF), ie. a function taking functions as arguments.
  - Sequential code parameters are functional arguments.
- Skeleton implementation is parallelisation of HOF.
- Many well-known HOFs have parallel implementations.
  - Thinking in terms of higher-order functions (rather than explicit recursion) helps in discovering parallelism.

Consequences:

- Skeletons can be combined (by function composition).
- Skeletons can be nested (by passing skeletons as arguments).

Skeletons Are PHOFs — Pipeline



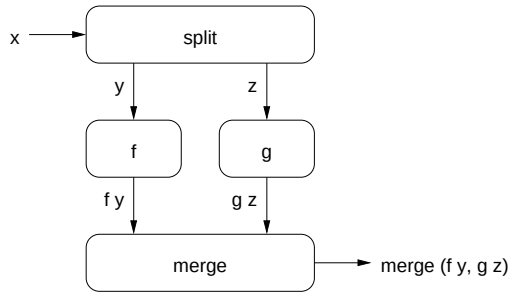
Code (parallel implementation in red)

```
pipe2 :: (b -> c) -> (a -> b) -> a -> c
pipe2 f g x = let y = g x in
              y `par` f y
```

Notes:

- `pipe2` is also known as *function composition*.
- In Haskell, sequential function composition is written as `.` (read “dot”).

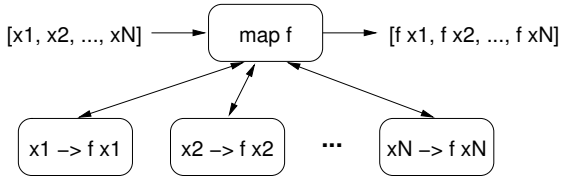
Skeletons Are PHOFs — Parallel Tasks



Code (parallel implementation in red)

```
task2 :: (a -> (b,c)) -> (d -> e -> f) -> (b -> d) -> (c -> e) -> a -> f
task2 split merge f g x = let (y,z) = split x
                              fy = f y
                              gz = g z in
                          fy `par` gz `pseq` merge fy gz
```


Skeletons Are PHOFs — Task Farm



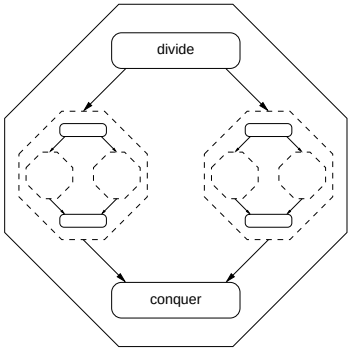
Code (parallel implementation in red)

```
farm :: (a -> b) -> [a] -> [b]
farm f []     = []
farm f (x:xs) = let fx = f x in
                fx `par` fx : (farm f xs)
```

Notes:

- `farm` is also known as *parallel map*.
  - Map functions exist for many data types (not just lists).
- Missing in implementation: strategy to force eval of lazy list.
- Strategies also useful to increase granularity (by chunking).

Skeletons Are PHOFs — Divide & Conquer



Code (parallel implementation in red)

```
dnc :: (a -> (a,a)) -> (b -> b -> b) -> (a -> Bool) -> (a -> b) -> a -> b
dnc div conq atomic f x | atomic x  = f x
                        | otherwise = let (l0,r0) = div x
                                          l = dnc div conq atomic f l0
                                          r = dnc div conq atomic f r0 in
                                      l `par` r `pseq` conq l r
```


Skeletons Are PHOFs — Divide & Conquer

Notes:

- Divide & Conquer is a generalised *parallel fold*.
  - Folds exist for many data types (not just lists).
- Missing in impl: strategies to force eval and improve granularity.

Aside: folding/reducing lists

```
fold :: (a -> a -> a) -> a -> [a] -> a
-- fold f e [x1,x2,...,xn] == e `f` x1 `f` x2 ... `f` xn, provided that
-- (1) f is associative, and
-- (2) e is an identity for f.

-- Tail-recursive sequential implementation:
fold f e []     = e
fold f e (x:xs) = fold f (e `f` x) xs

-- Parallel implementation as instance of divide & conquer:
fold f e = dnc split f atomic evalAtom where
    split xs = splitAt (length xs `div` 2) xs
    atomic []  = True
    atomic [_] = True
    atomic _   = False
    evalAtom []  = e
    evalAtom [x] = x
```

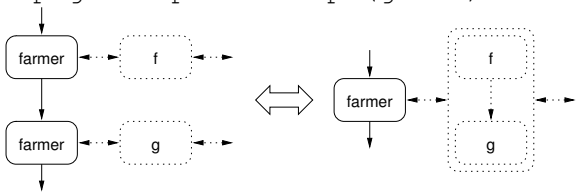
Program Transformations

Observation:

- HOFs can be transformed into other HOFs with provably equivalent (sequential) semantics.

Example: Pipeline of farms vs. farm of pipelines

- `map g . map f == map (g . f)`



- use `map g . map f` (pipe of farms) if ratio comp/comm high
- use `map (g . f)` (farm of pipes) if ratio comp/comm low
- More transformations in
  - G. Michaelson, N. Scaife. *Skeleton Realisations from Functional Prototypes*, Chap. 5 in S. Gorlatch and F. Rabhi (Eds), *Patterns and Skeletons for Parallel and Distributed Computing*, Springer, 2002

Program Development with Functional Skeletons

Programming Methodology:

1. Write seq code using HOFs with known equivalent skeleton.
2. Measure sequential processing cost of functions passed to HOFs.
3. Evaluate skeleton cost model.
4. If no useful parallelism, transform program and go back to 3.
5. Replace HOFs that display useful parallelism with their skeletons.

Tool support:

- Compilers can automate some steps (see Michaelson/Scaife)
  - Only for small, pre-selected set of skeletons
- Example: PMLS (developed by Greg Michaelson et al.)
  - Skeletons: map/fold (arbitrarily nested)
  - Automates steps 2-5.
    - Step 2: automatic profiling
    - Step 4: rule-driven program transformation + synthesis of HOFs
    - Step 5: map/fold skeletons implemented in C+MPI