A Data Flow Approach for Higher Order Functions in Numerical Applications

Ralf Ebner

Technische Universität München

(ebner@informatik.tu-muenchen.de)

Numerical Applications

Modern numerical applications

- use recursion and hierarchical data structures
- with "fat" tree nodes and leaves (matrices)
- run on networks of workstations (intranets)
- are based on numerical libraries (solvers, preconditioners, ..., often in FORTRAN)

Therefore: try to avoid or reduce expensive communication in a parallel implementation

$\Rightarrow$ keep data local

$\Rightarrow$ choose direct channels

FASAN

"**F**unctions **A**pplied to **S**treams

in **A**lgorithms of **N**umerical Computing "

Requirements:
- **comfortable**
- **reusable**
- **efficient**
- **portable**

Concepts of FASAN:

- functional layer for parallel configuration of sequential modules
- linking with functions written in procedural languages (C, Fortran, ...) or libraries
- data flow computation model, streams of data
- efficient implementation of tree-like data structures
- built upon portable parallel system: PVM

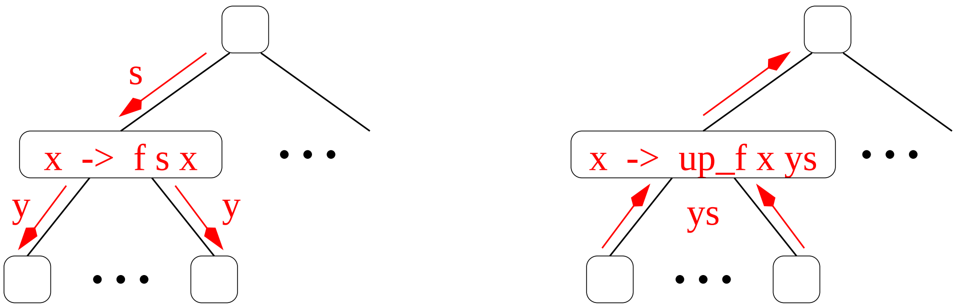
Higher Order Functions on Trees

(SML Version)

```
datatype α Tree = Leaf of α
                | Node of α * (α Tree) list;

fun mapTree
  f:(α→α) (Leaf x) = Leaf (f x)
| f  (Node x sons)
   = Node (f x) map (mapTree f) sons;

fun mapDown
  f:(α→α→α) s:α (Leaf x) = Leaf (f s x)
| f s (Node x sons)
   = let val y =  f s x
         val ys = map (mapDown f y) sons)  in
     Node y ys end;
```




```
fun mapUp
  leaf_f:(α→α) up_f:(α * (α List)→α) (Leaf x)
   = Leaf (leaf_f x)
| leaf_f  up_f  (Node x sons)
   = let val ys = map (mapUp leaf_f up_f) sons in
     Node (up_f x ys) ys end;
```

V-cycle

Top-down and bottom-up tree traversal combined

Typical multi-level numerical algorithms:

- Multigrid, AMG
- Recursive substructuring
- Sparse grid algorithms

Short solution (combining skeletons):

```
fun mapUpDown down_f leaf_f up_f s t
  = mapUp leaf_f up_f (mapDown down_f s t)
```

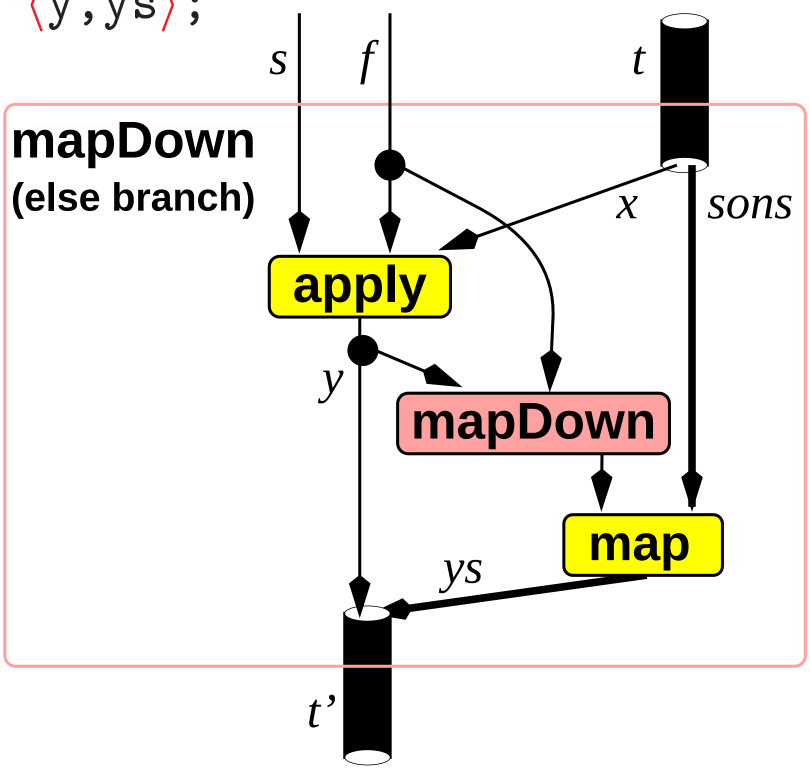
More efficient solution (“explicit”):

```
fun mapUpDown
  down_f leaf_f up_f s (Leaf x) = Leaf (leaf_f x)
| down_f leaf_f up_f s (Node x sons)
  = let
    val u  = down_f s x
    val ys = map (mapDownUp down_f leaf_f up_f u)
                sons
    val y  = up_f u ys in
    Node y ys end;
```

Higher Order Functions on Trees

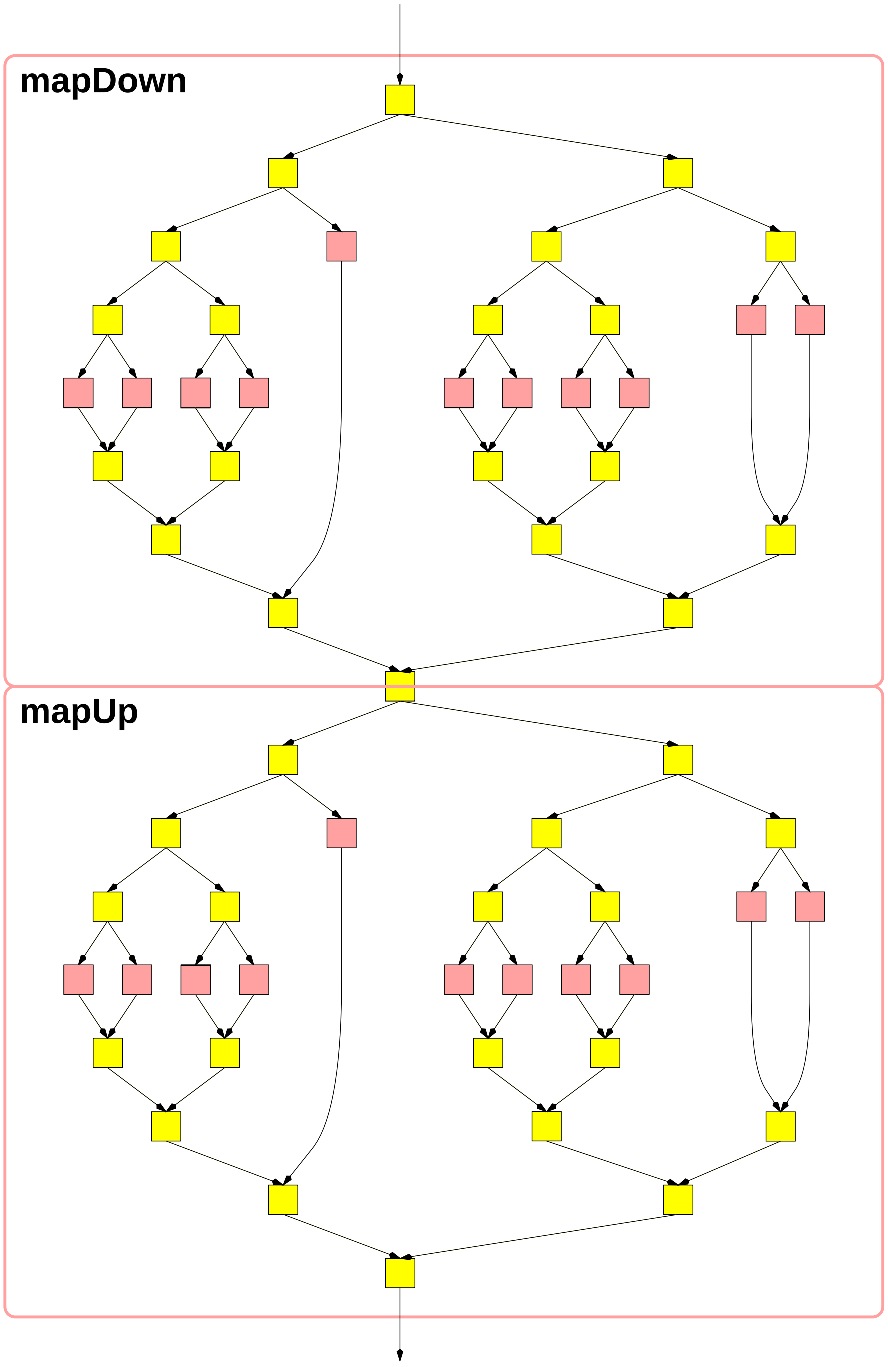
A FASAN function and its corresponding data flow graph

```
function mapDown (f, s, t): t’
begin
  if isLeaf(t) then
    t’ := f(s,t);
  else
    ⟨x,sons⟩ := t;
    y := f(s,x);
    ys := map (mapDown(f,y,_), sons);
    t’ := ⟨y,ys⟩;
  fi
end
```



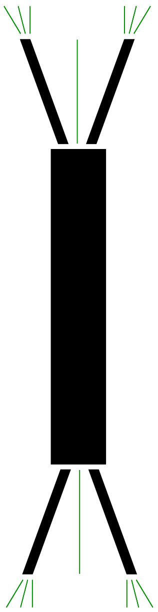
Data Flow Graph

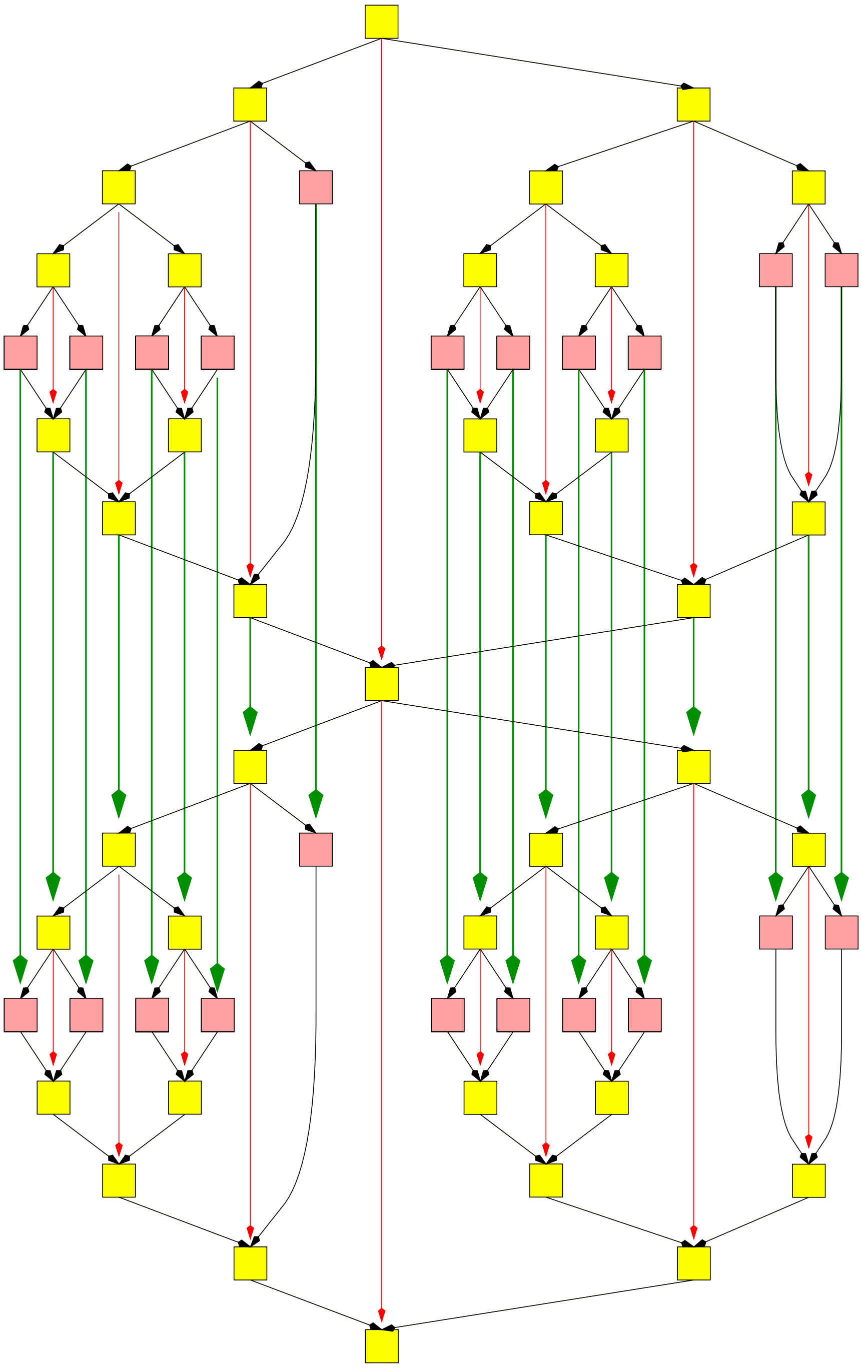
Unfolding of recursive calls



Unfolded Wrapper Streams

Direct asynchronous communication channels reduce the overhead of combined skeletons





Conclusions

FASAN's advantages with respect to higher order functions:

Practicability:

- "imperative" syntax
- functions and closures as stream elements
- separation of the algorithm from parallel execution

Efficiency in combining tree skeletons through *wrapper streams*:

- direct communication channels
- data locality and *prefetch* behaviour
- only one request for all stream elements