Parallel Functional Programs Translated into Java

Ralf Ebner

Technische Universität München

ebner@in.tum.de

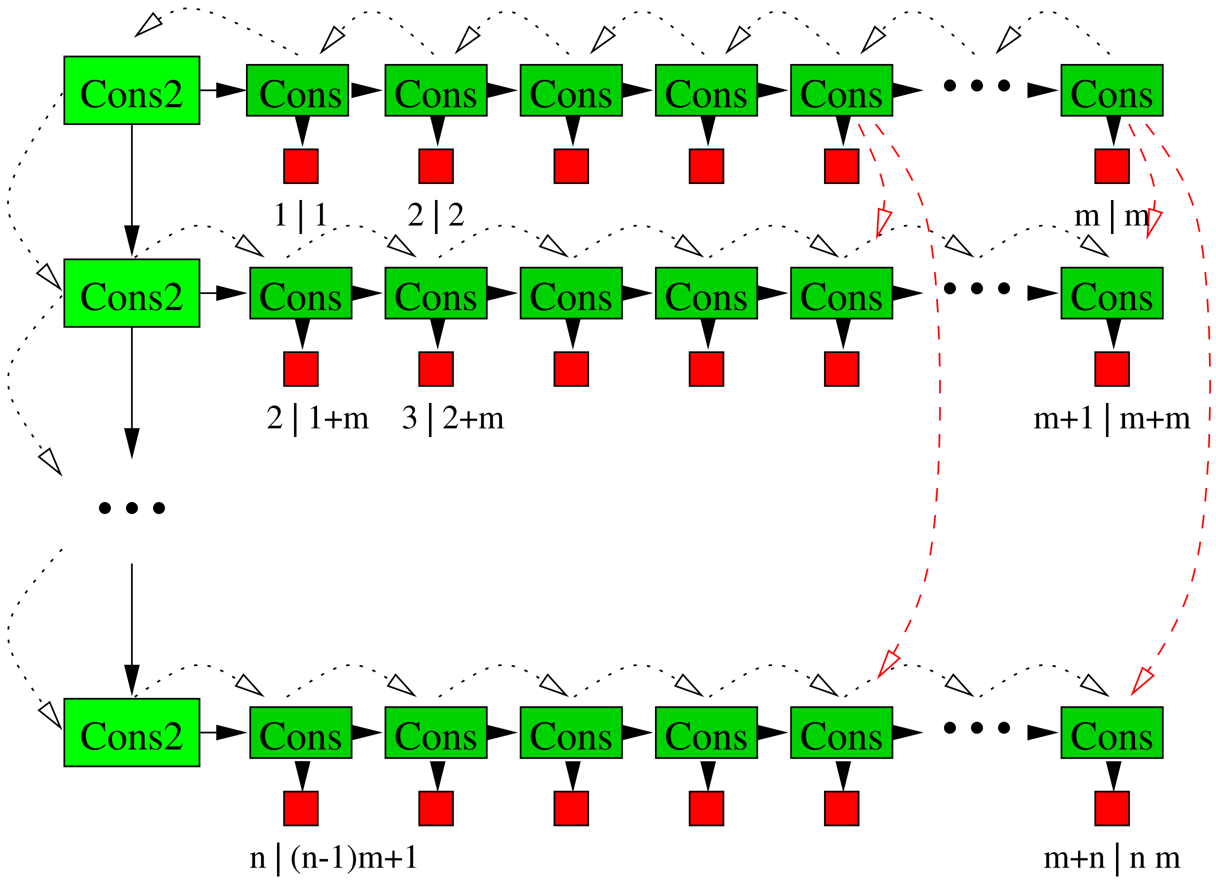
http://www.in.tum.de/∼ebner

Overview

1. Introduction
2. FASAN Language and Dataflow Concept
3. Implementation on Java
4. Examples: Barnes-Hut, Rec. Substructuring
5. Summary

Introduction: a Pipeline of Lists

```
type Area = Empty() | Cons2(first:Line; rest:Area);
function wave (prec:Line, ar:Area) -> ar':Area
{
  if not isEmpty(ar) then
    line = step (prec, ar.first);
    ar' = wave (line, ar.rest);
  else ... fi
}
```



Steps to Distributed Application

- Sequential algorithm
- Partitioning into parallel tasks
- Coordination: synchronization,communication
- Load distribution

FASAN

Functions **A**pplied to **S**ubstructured data
in **A**lgorithms of **N**umerical computing

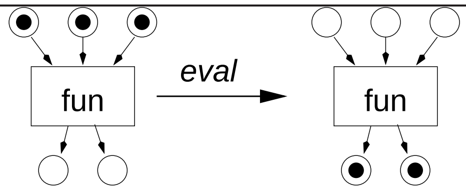
Concepts and Goals:

- Application range: adaptive algorithms on hierarchical data structures
- Functional language: single assignment semantics
  $\hookrightarrow$ automated parallelization
- Distributed tree data structures
  $\hookrightarrow$ data locality, direct communication routes
- Portability and platform independence
- Coordination language based on host language Java (and also C,C++,Fortran)

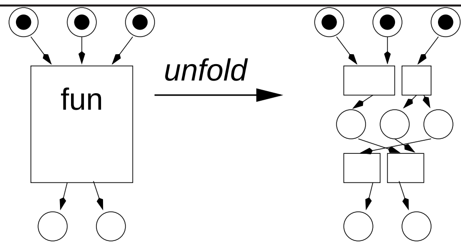
Data flow semantics

FASAN program is mapped to data flow graph, basically a dynamic acyclic Petri Net

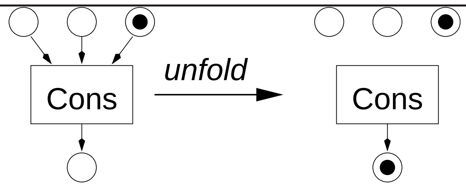
External functions are evaluated if all input data is available.



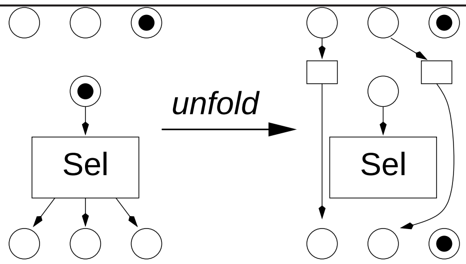
FASAN functions are unfolded as subgraphs, results are available independently.



Constructors are unfolded with only one argument.



Selectors draw direct communication channels.



Conditional expressions become function nodes.
Partitioning of the graph yields parallel application.

Comparison

OOP	FP

- automatic memory management
- polymorph type system

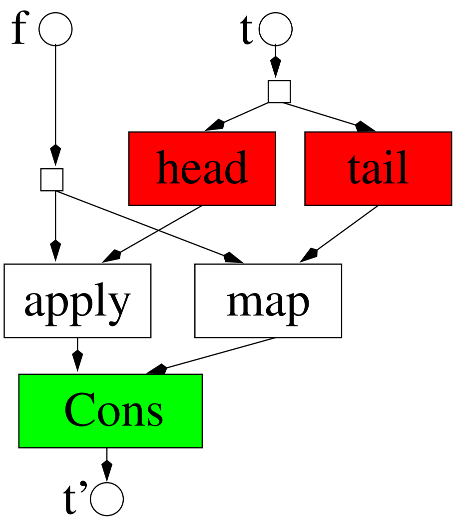
OOP	FP
• state within objects	• referential transparency
• classes are "first class"	• functions are "first class"
• inheritance, dynamic binding...	• evaluation strategies...

- implementation of abstract data types

FASAN Concepts	Support in Java
FASAN compiler	generated with JavaCC
external functions	`java.jni` (java native interface), flexible and standardized, fixed size of basic types
parallel function calls	`java.lang.Thread`
remote function calls	`java.rmi` (remote method invocation) with automatic distributed garbage collection
data types and result parameters are wrapper objects	`Remote` objects serve as: global references, monitors (`sychronized`) that block until content is available from producer thread, and keys in object caches
communication: arguments+results of remote calls	object serialization

Linear Recursion

```
type Object extern Java;
type List = Nil() | Cons (head:Object; tail:List);

function map (f:Object->Object; t:List)
            -> t':List
{ if isNil(t) then
    t' := Nil();
  else
    y = f(t.head)
      on getFreeHost();
    t' := Cons(y, map(f,t.tail));
  fi
}
```



Linear Recursion: Load Distribution

Barnes-Hut Algorithm:
calculation of accelerations for each particle

```
function accelL (t:BarnesHutTree; len:double;
                 mpL:MassPntList) -> acL:AccelList
  { acL = map (accel(t,len,_), mpL) }
```

- MassPntList contains array of MassPnts in each node, otherwise granularity gets too fine:
  ```
  type MassPntList = MEmpty()
       | MCons (head:MassPnt[]; tail:MassPntList);
  ```
- BarnesHutTree gets serialized and sent to target host in advance, and
- is saved in local object cache of target host

Location functions:
select host from queue of free hosts

```
int getFreeHost() {
    int p = queue.pop();
    if (load(p) < MAX_LOAD) queue.append(p);
    return p;
}
void setFreeHost (int p) {
    if (load[s] < MAX_LOAD) queue.append (s);
}
```

Barnes-Hut Algorithm

first-order version:

```
type MassPntList = MEmpty()
                 | MCons (Mfirst:MassPnt[]; Mrest:MassPntList);
type AccelList = AEmpty()
               | ACons (Afirst:Accel[]; Arest:AccelList);

function accelL (t:TreeWrapper; len:double; mpL:MassPntList)
         -> acL:AccelList
{   if isMEmpty(mpL) then
       acL = AEmpty();
    else
       a = accel (t, len, mpL.Mfirst) on getFreeHost();
       r = accelL (t, len, mpL.Mrest);
       setFreeHost (a.host);
       acL = ACons(a.result, r);
    fi
}
```

Runtime results

NOW of Sun Ultra-60's, JDK-1.2, 5000 particles

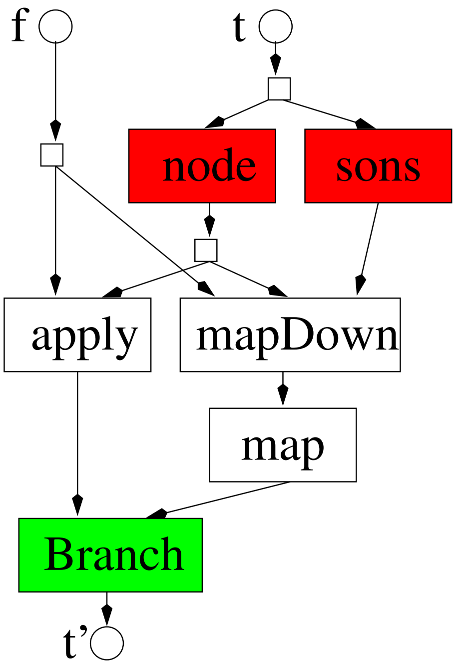
# processors	1	2	3	4
runtime	125.79	97.20	89.99	83.39

# processors	5	6	7	8
runtime	96.44	104.22	111.54	115.79

- One MassPnt per list node is too fine grain.
- Serialization of tree is costly.
- JIT increases the need for big subproblems (list elements).
- RMI (client-server concept) is not really suitable for High Performance Computing yet.

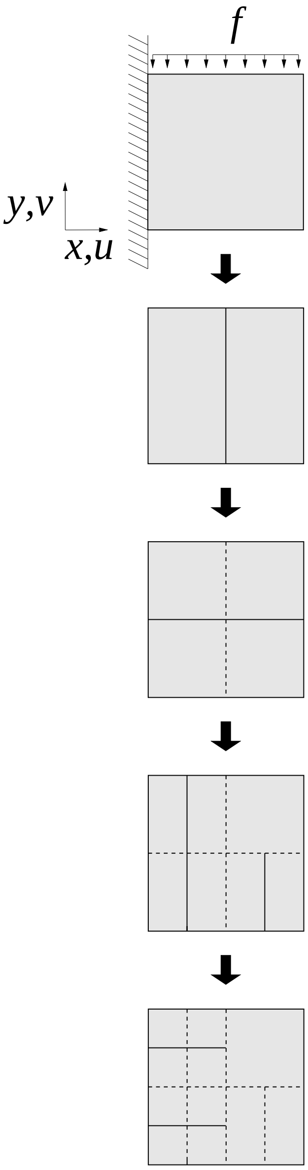
Cascadic recursion, Divide&Conquer

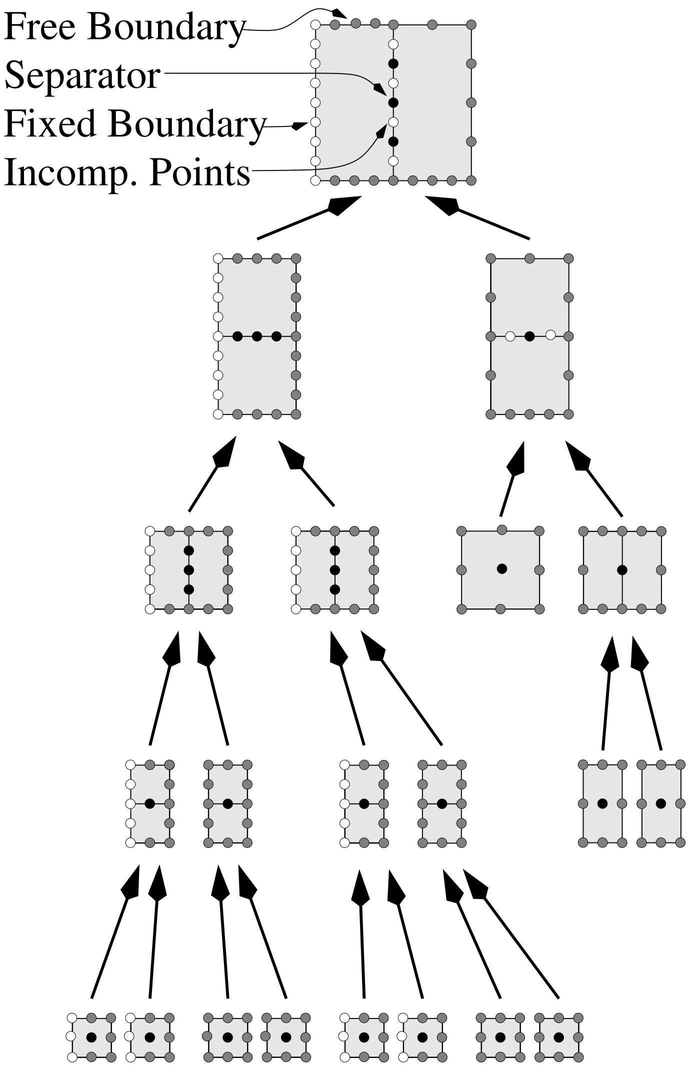
```
type Tree = Leaf (val:Object)
          | Branch (node:Object; sons:List);

function mapDown (f:(Object,Object)->Object;
                  s:Object; t:Tree)
               -> t':Tree
{ if isLeaf(t) then
    t' := Leaf (f(s, t.val));
  else
    y := f (s, t.node);
    ys := map (mapDown(f,y,_),
               t.sons);
    t' := Branch(y,ys);
  fi
}
```



Cascadic Recursion on Trees

Example: Recursive Finite Element Simulation (ARESO)

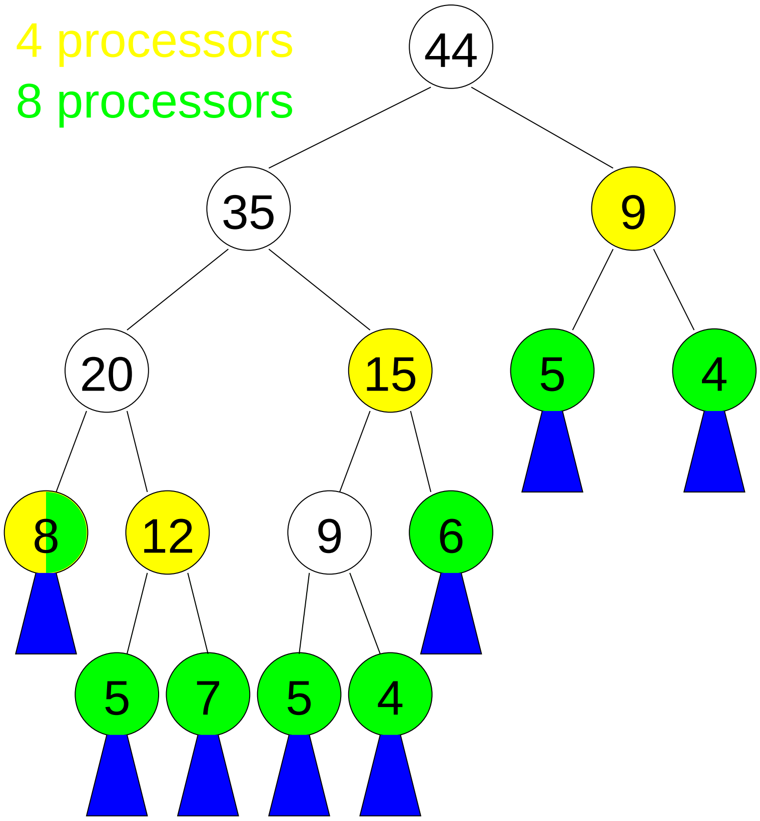




Cascadic Recursion: Load distribution

Algorithm HF (heaviest problem first):

- each node of bisection tree contains the "weight" of its subtree
- get heaviest substrees of the top part of the tree



Theoretical performance guaranty:

If the weight of every subtree is at least an α-fraction of the weight of its parent, the runtime is at most $\lfloor 1/\alpha \rfloor \cdot (1-\alpha)^{\lfloor 1/\alpha \rfloor - 2}$ greater than theoretical optimum.

Summary

FASAN as coordination language on top of Java:

+ higher abstraction: threads and RMI are transparent
  - ↪ less programming effort
  - ↪ no deadlocks or race conditions
+ wrapper for distributed data structures act as global references whose contents are copied only when dereferencing takes place
  - ↪ data locality
  - ↪ direct communication
+ increased parallel potential through use of pipelines
+ easy integration of load balancing strategies

Future work

... on Java:

- native threads (supported in JDK1.2)
- RMI optimization for High Perfomance Computing

... on FASAN:

- predefined load distribution schemes
- further automatization (location annotations, tail recursions)
- replacing RMI with MPI for Java (when available)