Functional Programs on Clusters*

Viktória Zsók, Zoltán Horváth, Zoltán Varga

Department of Programming Languages and Compilers
Faculty of Informatics
University Eötvös Loránd, Budapest, Hungary
E-mail: hz@inf.elte.hu

*Supported by OTKA T037742

Overview

- Clean-CORBA, Haskell-CORBA interfaces
- Connecting Clean clients (pure functional components expressing computation) via CORBA channel objects, communication layer for cluster computing
- high-level, reliable, type-safe, asynchronous communication,

  distribution transparency
- Computations implemented as process networks
- Example: pipeline
- Interconnection of components written in object-oriented and functional programming languages

Parallelism and functional programming

- composition of functions is an associative operation
- evaluation (rewriting) of expressions can be done in parallel
- implicit parallelism - control of evaluation degree
- explicit parallelism and communication

Approaches

- TCP - Clean OIO, Distributed Haskell with Ports (parallel elementwise processing on cluster)
- agents (dynamics in Clean, JoCaml)
- abstract communication layer based on a middleware (CORBA)

Mapping of IDL to Clean

- IDL module mapping: the names of the modules are included in the identifiers
- mapping of simple and composed types: integer-Int, real-Real, enumeration-algebraic type, array-array, structures-record, sequences-list, union-algebraic with data constructor for discr. values
- mapping of interfaces: abstract Clean types hiding object references, conversion supported by IDL compiler (narrow,widen)
- operations are mapped to functions (side effects, `World` argument as environment of a Clean program, result: `ResultOrException`).

- IDL compiler: interfaces are generated both for clients and for servers.

- Server side mapping: simplified version of the Object IO framework.

- servant generated as a record type with one field for each IDL operation in the interface

- programmer: creates an instance and registers it

- For communication through TCP ports and for IP identification the services of MICO Binder are used.

Channel object - initialization and registration

```
Start w
   # (orb,_,w) = CORBA_ORB_init args w            // init ORB
   = CORBA_Server_run orb Void ServerInit w      // init server
where
 ServerInit ps w
  # (obj, ps, w) = Channel__servant_open ps servant w  // reg.servant
  # w
    = WriteIORToFile (CORBA_Server_get_orb ps) obj "channel.ior" w
  = (ps, w)
```

Channel object - servant

```
servant = { Channel__servant |
                      ls                = messages,
                      impl_send         = my_send,
                      impl_receive      = my_receive,
                      }
       my_send (ls, ps) what w
               = ((ls ++ [what], ps), Result Void , w)
       my_receive ([x:xs], ps) w
               = ((xs, ps), Result x, w)
```

The pipeline skeleton

- linear process network for calculating a composite function

- pipeline element (CORBA client) calculates a component function and sends intermediate results to its immediate successor via channel object.

- Receiving data by polling

```
receivef obj w
    # (Result l, w) = Channel_empty obj w
    | l
       // delay
       = receivef obj w
    = Channel_receive obj w
```

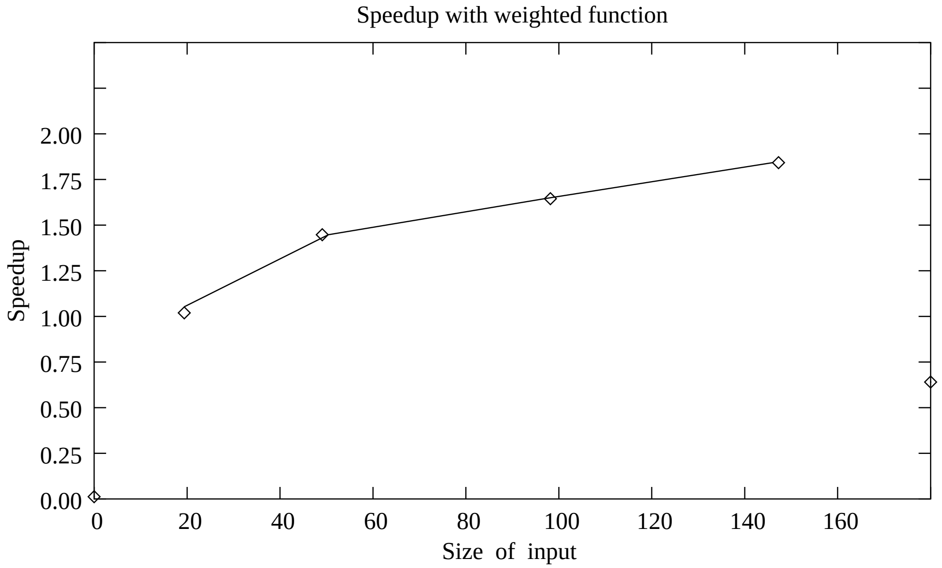
- in case of distributed applications for some expressions a strict evaluation should be enforced (system clock)

Performance measurement

- The cost of the communication via the CORBA server objects is relatively high compared to the cost of this simple computation.

- If we slow down the computation of component functions simulating a most complex computation (we apply a weighted function)

Speedup with different number of input data



Conclusions and further work

- Clean-CORBA interface (an other application: testing objects from Clean) to do: support for distributing and starting processes

- abstract communication layer, process networks, to add: skeleton library (farm, mesh, torus, pipeline, pewp, etc.) in form of higher order functions

- to do: type independent, type safe channel object (using Any and TC, and Clean polymorhism and generics), substitution of polling by data driven method, safe code exchange (expressions, functions, as like as dynamics), code migration, agents

- application may consist of components written in several programming languages (e.g. C++ channel object exists)