pony – The occam-π Network Environment

Mario Schweigler and Adam Sampson
Computing Laboratory, University of Kent
Canterbury, UK

Contents

What is pony?

- Network environment for occam-π
  (Name: anagram of [o]ccam, [p]i, [n]etwork, [y])
  (Formerly known as 'KRoC.net')
- Comes with the KRoC distribution
- Allows a unified approach for inter- and intra-processor concurrency

The Need for a Unified Concurrency Model

- It should be possible to distribute applications across several processors (or 'back' to a single processor) without having to change the components
- This transparency should not damage the performance
  - The underlying system should apply the overheads of networking only if needed in a concrete situation – this should be determined dynamically at runtime

pony Applications

- A pony *application* consists of several *nodes*
- There are *master* and *slave* nodes – each application has one master node and any number of slave nodes
- Applications are administrated by an *Application Name Server (ANS)* which stores the network location of a master for a given application-name and allows slave nodes to 'look up' the master

Network-channel-types

- The basic communication primitive between occam-π processes in pony are channel-types
- A *network-channel-type (NCT)* is the networked version of an occam-π channel-type
- It is transparent to the programmer whether a given channel-type is an intra-processor channel-type or an NCT

Network-channel-types

- NCTs have the same semantics and usage as normal channel-types
  - Bundle of channels
  - Two ends, each of which may be shared
  - Ends are mobile
- Ends may reside on different nodes, and may be moved to other nodes

Transparency in pony

- Semantic transparency
  - All occam-π **`PROTOCOL`**s can be communicated over NCTs
  - Semantics of communicated items are preserved, including **`MOBILE`** semantics
  - Handling of NCTs is transparent to the programmer
    - NCT-ends may be ***shared***, like any other channel-type-end
    - NCT-ends are ***mobile***, like any other channel-type-end, and can be communicated across distributed applications in the same way as between processes on the same node

Transparency in pony

- Pragmatic transparency
  - pony infrastructure is set up dynamically when needed
  - If sender and receiver are on the same node, communication only involves 'traditional' channel-word access
  - If they are on different nodes, communication goes through the pony infrastructure (and the network)

Using pony

- There are a number of public processes for:
  - Starting pony
  - Explicit allocation of NCT-ends
  - Shutting down pony
  - Error-handling
  - Message-handling

Starting pony

- Startup process
  - Starts the pony environment on a node
- Returns:
  - A network-handle
    - Used for allocation and shutdown
  - An error-handle if required
    - Used for error-handling
  - A message-handle if required
    - Used for message-handling

Starting pony

- Startup process contacts the ANS
- If our node is the master, its location is stored by the ANS
- If our node is a slave
  - Startup process gets location of master from ANS
  - Connects to master
- On success, startup process starts the pony environment and returns the requested handles
- Otherwise returns error

Explicit Allocation of NCT-ends

- pony's allocation process returns an end of an NCT
  - The ends of an NCT may be allocated on different nodes at any given time
- Allocation process communicates with pony environment via network-handle (given as a parameter)
- An *explicitly* allocated NCT is identified by a unique NCT-name stored by the master node

Explicit Allocation of NCT-ends

- If parameters are valid, allocation process allocates and returns the NCT-end
- Allocation process returns error if there is a mismatch with previously allocated ends of the same name, regarding:
  - Type of the channel-type
  - Shared/unshared properties of its ends

Implicit Allocation by Moving

- Any channel-type-end, including NCT-ends, may be moved along a channel
- If an end of a locally declared channel-type is moved to another node (along a channel of an NCT) for the first time, this channel-type is *implicitly* allocated by the pony environment
  - That channel-type automatically becomes an NCT
  - Programmer does not need to take any action himself
  - Does not even need to be aware whether the end is sent to a process on the same or on a remote node

Shutting down pony

- Shutdown process communicates with pony environment via network-handle (given as a parameter)
- Must be called after all activity on (possibly) networked channel-types has ceased
- If node is master, it notifies the ANS about the shutdown
- pony environment shuts down its internal components

Error-handling and Message-handling

- Error-handling used for the detection of networking errors during the operation of pony applications
- Message-handling used for outputting status and error messages
- Not discussed in the paper; see Mario's PhD thesis for details

Configuration

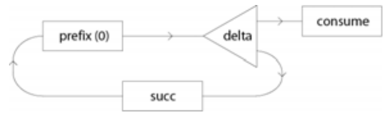
- Done via simple configuration files
- Used to configure
  - Network location of a node
  - Network location of the ANS
- All settings may be omitted
  - In this case either defaults are used or the correct setting is detected automatically

Implementation of pony

- Brief overview of pony's internal components can be found in the paper
- For a detailed discussion, see Mario's thesis

'`commstime`' Example

- The classical '`commstime`' benchmark



Non-distributed '`commstime`'

```
PROC commstime (CHAN BYTE key?, scr!, err!)
  BOOL use.seq.delta:
  INT num.loops:
  SEQ
    ...  Find out whether to use the sequential or the parallel delta
    ...  Find out the number of loops
    -- Channels between the processes
    CHAN INT a, b, c, d:
    -- Run sub-processes in parallel
    PAR
      prefix (0, b?, a!)
      IF
        use.seq.delta
          -- Sequential delta
          seq.delta (a?, c!, d!)
        TRUE
          -- Parallel delta
          delta (a?, c!, d!)
      succ (c?, b!)
      -- Monitoring process
      consume (num.loops, d?, scr!)
:
```


Distributed '`commstime`'

- Each sub-process runs on a separate node
- Channels between the processes become NCTs containing a single `INT` channel
- Used for benchmarking pony (see below)

Distributed '`commstime`' – The Channel-type Declaration

```
-- Channel-type with one INT channel
CHAN TYPE INT.CT
  MOBILE RECORD
    CHAN INT chan?:
:
```


Distributed '`commstime`' – The '`prefix`' Node (1)

```
PROC commstime.prefix (CHAN BYTE key?, scr!, err!)
  -- Network-handle
  PONY.NETHANDLE! net.handle:
  -- NCT-end variables
  INT.CT? b.svr:
  INT.CT! a.cli:
  -- Other variables
  INT own.node.id, result:
  SEQ
    -- Start pony
    pony.startup.unh (PONYC.NETTYPE.TCPIP, "", "commstime",
                      "", PONYC.NODETYPE.SLAVE,
                      own.node.id, net.handle, result)
    ASSERT (result = PONYC.RESULT.STARTUP.OK)
```

Distributed ‘**commstime**’ – The ‘**prefix**’ Node (2)

```
  -- Allocate NCT-ends
  pony.alloc.us (net.handle, "b", PONYC.SHARETYPE.UNSHARED,
                 b.svr, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  pony.alloc.uc (net.handle, "a", PONYC.SHARETYPE.UNSHARED,
                 a.cli, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  -- Start sub-process
  prefix (0, b.svr[chan], a.cli[chan])
  -- No shutdown of pony here
  --   because the sub-process that was started is running infinitely
:
```


Distributed ‘**commstime**’ – The ‘**delta**’ Node (1)

```
PROC commstime.delta (CHAN BYTE key?, scr!, err!)
  -- Network-handle
  PONY.NETHANDLE! net.handle:
  -- NCT-end variables
  INT.CT? a.svr:
  INT.CT! c.cli, d.cli:
  -- Other variables
  BOOL use.seq.delta:
  INT own.node.id, result:
  SEQ
    ...  Find out whether to use the sequential or the parallel delta
    -- Start pony
    pony.startup.unh (PONYC.NETTYPE.TCPIP, "", "commstime",
                      "", PONYC.NODETYPE.SLAVE,
                      own.node.id, net.handle, result)
    ASSERT (result = PONYC.RESULT.STARTUP.OK)
```


Distributed ‘**commstime**’ – The ‘**delta**’ Node (2)

```
  -- Allocate NCT-ends
  pony.alloc.us (net.handle, "a", PONYC.SHARETYPE.UNSHARED,
                 a.svr, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  pony.alloc.uc (net.handle, "c", PONYC.SHARETYPE.UNSHARED,
                 c.cli, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  pony.alloc.uc (net.handle, "d", PONYC.SHARETYPE.UNSHARED,
                 d.cli, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  -- Start sub-process
  IF
    use.seq.delta
      -- Sequential delta
      seq.delta (a.svr[chan], c.cli[chan], d.cli[chan])
    TRUE
      -- Parallel delta
      delta (a.svr[chan], c.cli[chan], d.cli[chan])
  -- No shutdown of pony here
  --   because the sub-process that was started is running infinitely
:
```


Distributed ‘**commstime**’ – The ‘**succ**’ Node (1)

```
PROC commstime.succ (CHAN BYTE key?, scr!, err!)
  -- Network-handle
  PONY.NETHANDLE! net.handle:
  -- NCT-end variables
  INT.CT? c.svr:
  INT.CT! b.cli:
  -- Other variables
  INT own.node.id, result:
  SEQ
    -- Start pony
    pony.startup.unh (PONYC.NETTYPE.TCPIP, "", "commstime",
                      "", PONYC.NODETYPE.SLAVE,
                      own.node.id, net.handle, result)
    ASSERT (result = PONYC.RESULT.STARTUP.OK)
```


Distributed ‘**commstime**’ – The ‘**succ**’ Node (2)

```
  -- Allocate NCT-ends
  pony.alloc.us (net.handle, "c", PONYC.SHARETYPE.UNSHARED,
                 c.svr, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  pony.alloc.uc (net.handle, "b", PONYC.SHARETYPE.UNSHARED,
                 b.cli, result)
  ASSERT (result = PONYC.RESULT.ALLOC.OK)
  -- Start sub-process
  succ (c.svr[chan], b.cli[chan])
  -- No shutdown of pony here
  --   because the sub-process that was started is running infinitely
:
```


Distributed ‘**commstime**’ – The ‘**consume**’ Node (1)

```
PROC commstime.consume (CHAN BYTE key?, scr!, err!)
  -- Network-handle
  PONY.NETHANDLE! net.handle:
  -- NCT-end variable
  INT.CT? d.svr:
  -- Other variables
  INT num.loops, own.node.id, result:
  SEQ
    ...  Find out the number of loops
    -- Start pony
    pony.startup.unh (PONYC.NETTYPE.TCPIP, "", "commstime",
                      "", PONYC.NODETYPE.MASTER,
                      own.node.id, net.handle, result)
    ASSERT (result = PONYC.RESULT.STARTUP.OK)
```

Distributed ‘**commstime**’ – The ‘**consume**’ Node (2)

```
-- Allocate NCT-end
pony.alloc.us (net.handle, "d", PONYC.SHARETYPE.UNSHARED,
               d.svr, result)
ASSERT (result = PONYC.RESULT.ALLOC.OK)
-- Start sub-process (monitoring process)
consume (num.loops, d.svr[chan], scr!)
-- No shutdown of pony here
--   because the sub-process that was started is running infinitely
:
```

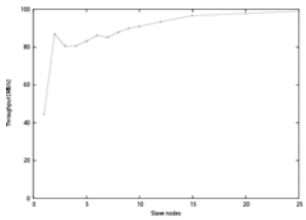

Benchmarking pony

- Measure the impact of using pony
  - vs. non-distributed programs
  - vs. hand-written networking
- Benchmarks conducted on TUNA cluster
  - 30 PCs, 3.2 GHz Intel Pentium IV
  - Dedicated gigabit Ethernet network
- Code of the benchmarks is in the KRoC distribution

Communication Time

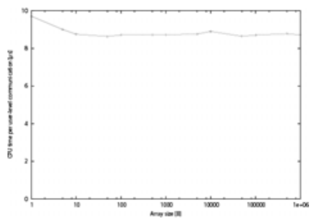
- The ‘**commstime**’ benchmark (with sequential ‘**delta**’)
  - With normal channels: 19 ns
    - One occam context switch
  - With pony channels: 66 μs
    - Many context switches and two network round trips
- Still 15,000 communications per second

Throughput



- Slaves send 100 KB messages as fast as possible to collector

Overhead

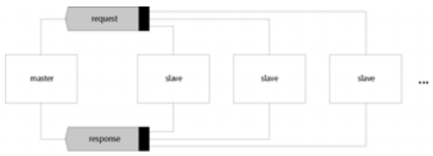


- 100 KB messages;
  < 2% network traffic overhead

A Simple Application

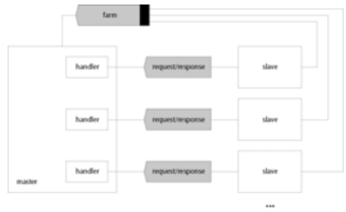
- Rendering the Mandelbrot set via farming
- Master generates requests and collects results
- 25 slaves; buffer requests, use C maths code
- pony support: ~30 lines of self-contained code

Shared Approach



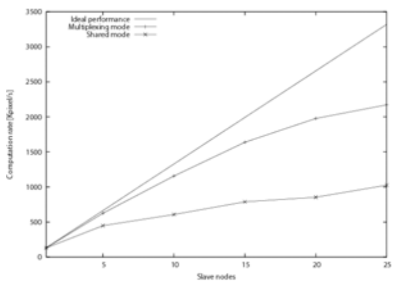
- Shared request/response channel-types
- 25 nodes: 30% CPU utilisation – not very good
- Problem: claiming the shared channels

Multiplexing Approach



- One channel-type per slave
- 25 nodes: 85% CPU utilisation

Farming Performance



Conclusions

- pony and occam-π provide a unified concurrency model for inter- and intra-processor applications
  - Achieving semantic and pragmatic transparency
- Simple handling of pony for the occam-π programmer
  - Minimum number of public processes for basic operations (startup, explicit allocation, etc.)
  - Runtime operation handled automatically and transparently by the pony environment
  - Simple (mostly automatic) configuration
- Initial performance measurements are encouraging

Future Work

- Adding support for new occam-π features
  - Mobile processes
  - Mobile barriers
  - … and anything else that comes up, to keep pony transparent to occam-π
- Integrating pony into RMoX
- Supporting the Transterpreter (and other platforms)
- Supporting network-types other than TCP/IP (e.g. for robotics with the Transterpreter)

Future Work

- Security model (encryption)
- Simplified startup and configuration on clusters
- ‘Tuning’ work to further enhance performance
- And… this implementation really works – could use the same design patterns to add full transparency to JCSP.net and friends!