PyCSP – Communicating Sequential Processes for Python

John Markus Bjørndalen[1]
Brian Vinter[2]
Otto Anshus[1]

[1]Department of Computer Science, University of Tromsø, Norway
[2]Department of Computer Science, University of Copenhagen, Denmark

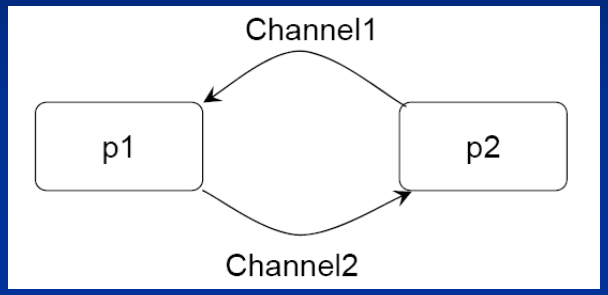
Why PyCSP

- Internal research projects
  - Simple prototyping, especially in projects that already use Python
  - Want to use CSP from Python
- eScience
  - Python
    - Script and integration language
    - Prototyping
    - Easy to learn, readable code
    - Plenty of tools and libraries
  - CSP
    - Simpler than message passing and shared memory?
- Teaching
  - eScience
  - CS students
    - Show them CSP *and* the implementation in a few lectures
    - and let them tinker with it

Some goals

- Simple, short, and readable source code
  - Should be easy to walk students through the code
- Pure python code
  - Portable implementation that does not depend on compiling extra libraries
- Reasonable performance

Simple PyCSP program




```
~/pycsp/pycsp-0-1/test> python2.5 simple.py
P1, read from input channel: 0
P2, read from input channel: 0
P1, read from input channel: 1
P2, read from input channel: 1
P1, read from input channel: 2
P2, read from input channel: 2
P1, read from input channel: 3
P2, read from input channel: 3
....
```

```
import time
from pycsp import *

def P1(cin, cout):
    while True:
        v = cin()
        print "P1, read from input channel:", v
        time.sleep(1)
        cout(v)

def P2(cin, cout):
    i = 0
    while True:
        cout(i)
        v = cin()
        print "P2, read from input channel:", v
        i += 1

chan1 = One2OneChannel()
chan2 = One2OneChannel()

Parallel(Process(P1, chan1.read, chan2.write),
         Process(P2, chan2.read, chan1.write))
```

Simplifying Process Syntax using Python Descriptors

```
# Old code
def TestProc(n):
    print "This is test proc", n

Sequence(Process(TestProc, 1),
         Process(TestProc, 2),
         Process(TestProc, 3))
```

```
def process(func):
    "Decorator for creating process functions"
    def _call(*args, **kwargs):
        return Process(func, *args, **kwargs)
    return _call
```

-Tags the function as a PyCSP process

```
# New code
@process
def TestProc2(n):
    print "This is test proc", n

Sequence(TestProc2(1),
         TestProc2(2),
         TestProc2(3))
```

Parallel and Sequence

```
 1 class Parallel:
 2     def __init__(self, *processes):
 3         self.procs = processes
 4         # run, then sync with them.
 5         for p in self.procs:
 6             p.start()
 7         for p in self.procs:
 8             p.join()
 9
10 class Sequence:
11     def __init__(self, *processes):
12         self.procs = processes
13         for p in self.procs:
14             p.run()
```

Alternative Example

```
final Skip sg = new Skip();
final Guard[] guards = {in1, in2, sg};        // prioritised order
final int IN1 = 0, IN2 = 1, SG = 2;           // index into guards

final Alternative alt = new Alternative (guards);

switch (alt.priSelect()) {
  case IN1:
    x1 = in1.read();
    break;
  case IN2:
    x2 = in2.read();
    break;
  case SG:
    break;
}
```

JCSP

PyCSP – returns the guard, not the guard index

```
1 # assuming that we already have two channel inputs: in1, and in2
2 sg = Skip()
3 alt = Alternative(in1, in2, sg)
4 ret = alt.priSelect()
5 if ret != sg:
6     # Alt did not return the skip guard
7     print "Reading from the selected channel:", ret()
```

Commstime

```
1  @process
2  def Consumer(cin):
3      "Commstime consumer process"
4      N = 5000
5      ts = time.time
6      t1 = ts()
7      cin()
8      t1 = ts()
9      for i in range(N):
10         cin()
11     t2 = ts()
12     dt = t2-t1
13     tchan = dt / (4 * N)
14     print "DT = %f.\nTime per ch : %f/(4*%d) = %f s = %f us" % \
15           (dt, dt, N, tchan, tchan * 1000000)
16     print "consumer done, posioning channel"
17     poisonChannel(cin)
18
19 def CommsTimeBM():
20     # Create channels
21     a = One2OneChannel("a")
22     b = One2OneChannel("b")
23     c = One2OneChannel("c")
24     d = One2OneChannel("d")
25
26     print "Running commstime test"
27     Parallel(Prefix(c.read, a.write, prefixItem = 0),
28              Delta2(a.read, b.write, d.write),
29              Successor(b.read, c.write),
30              Consumer(d.read))
```

Current state

- Implemented:
  - Channels: One2One, One2Any, Any2One, Any2Any, BlackHole
  - Channel Poison, and poison propagation
  - Processes
  - Parallel and Sequence constructs
  - Alternative
  - Guards: Guard, Skip, and input channels
  - Some components based on JCSP.plugNplay library
- Todo
  - Network support
  - More from plugNplay and core

Early Experiences

- University of Copenhagen, department of Computer Science
  students in Extreme Multiprogramming course:
  - Offered occam, C++CSP, and JCSP
  - Several opted for PyCSP
  - Informal look-over seems to indicate that the solutions using PyCSP were shorter and easier to understand than solutions using statically typed languages

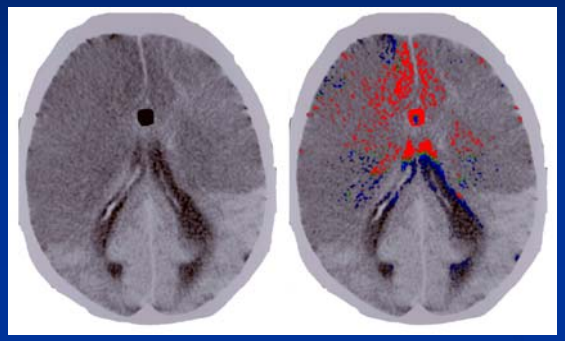
Performance evaluation

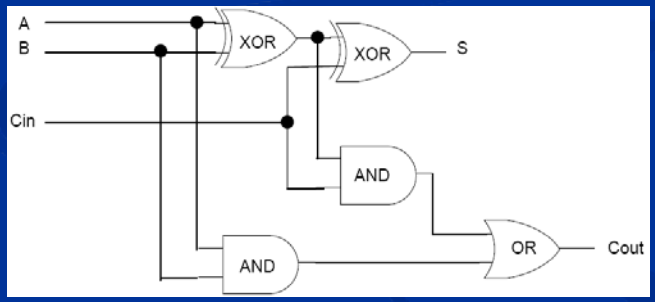
Implementation	Optimization	min	max	avg
AMD, PyCSP		74.78μs	88.40μs	84.81μs
AMD, PyCSP	Psyco	48.15μs	54.91μs	52.67μs
R360, PyCSP		141.67μs	142.51μs	142.09μs
R360, PyCSP	Psyco	89.50μs	91.57μs	90.37μs
R370, PyCSP		128.14μs	129.12μs	128.61μs
Qtek mobile phone, PyCSP		6500μs	6500μs	6500μs
AMD, JCSP, w/SeqDelta		6μs	9μs	8.1μs

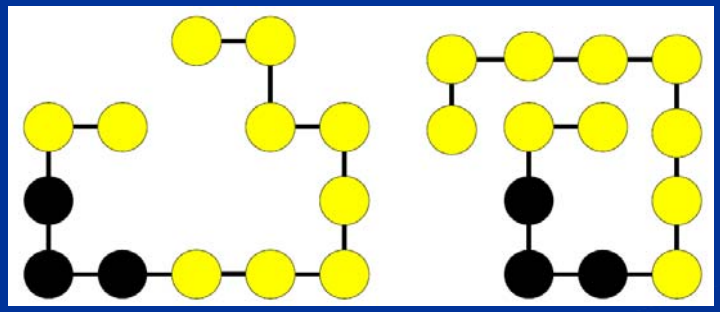
Table 1 Commstime results

Application examples in the paper

- Radiation planning
- Circuit design
- Protein folding
- Commstime







Conclusions

- Python CSP library using Python 2.5+
- Attempt to keep the code simple, short and readable
- Early experiences using PyCSP for teaching looks promising
- Available from http://www.cs.uit.no/~johnm/code/PyCSP/
- Recent:
  - Alternative process syntax
  - Network support through Pyro (channel poision does not work properly yet)