Scripting HPX

Cutting away complexity

Steven R. Brandt (sbrandt@cct.lsu.edu)

C++ vs. Scripting

C++

- type checked
- long compile times
- complex syntax
- fast execution

Lua

- not type checked
- no compile times
- simple syntax
- slow execution

What is Lua?

- A scripting language
- Lua means "moon" in Portugese, it's not an acronym
- Lua is a fast
- Lua is lightweight (about 20k lines of c code)
- Lua is portable
- Lua is embeddable
- Lua is free

Where is Lua useful?

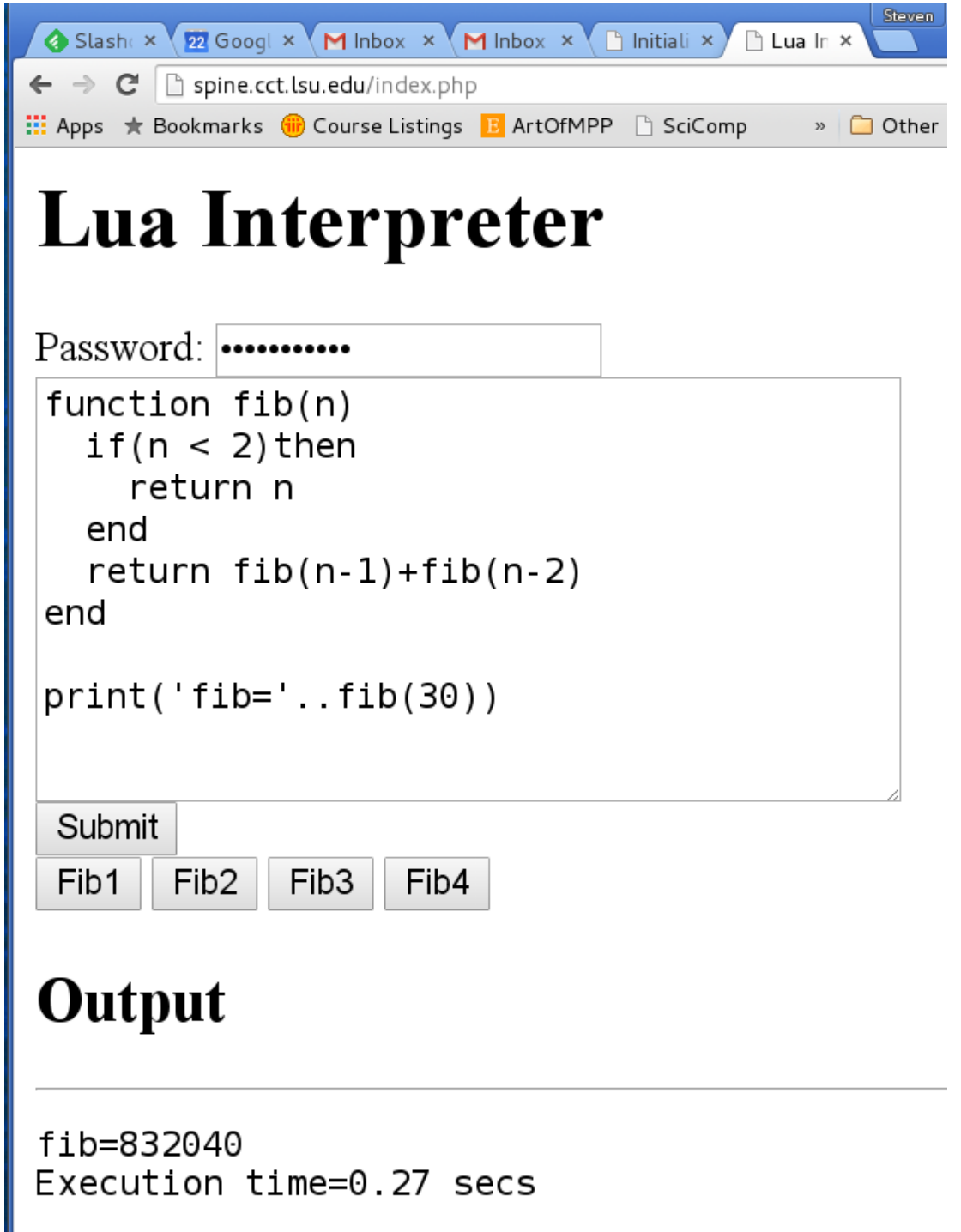
- Initialization / parameter files to start more complex programs
- For enabling on-the-fly runtime changes in a complex program
- scheduling calls to routines in a more complex program
- a teaching tool for library concepts

Basic Lua - Hello, World

```
--two dashes are a comment
print("Hello, world")
```

The Lua Interpreter

- Web page: http://melete.cct.lsu.edu
- Password: SCLuaHPX
- Input your code and go!
- Try the "hello world" program right now.



STE||AR GROUP

Basic Lua - Loops & Arrays

```
a = {5,3,9,10} --declare an array, a "table" in Lua speak
print(a[100]) --prints nil, surprise!
for i=1,#a do --#a gets the length of the array
   print(i,a[i]) --array index starts at one
end --end of block

for i,v in ipairs(a) do --this loop does the same thing
   print(i,v)
end
```

Basic Lua - Loops & if

```
a = {5,3,9,10}
i = 1
while i <= #a do --yet another way to loop
   print(i,a[i])
   if(i == #a)then --a basic if loop
      print('End')
   end --all blocks end this way
   i = i + 1 --no ++ or +=
end
```

Basic Lua - If, Elseif, Else

```
a = 3
if a==4 then
   print('Equal to 4')
elseif a ~= 3 then -- ~= means not equal
   print('Not equal to 3')
else
   print('Else')
end
```

Basic Lua - Functions

```
function foo(a,b) --declare a function
    print(a,b) --built-in print function
    return a+b --return a value, nil if omitted
end --end of a block
```

Basic Lua - Functions

```
function foo(a,b) --declare a function
    print(a,b) --built-in print function
    local c = a+b --local variable
    d = a-b --global variable
end --end of a block
foo(10,2)
print(c) -- prints nil
print(d) -- prints 8
```

Lua with HPX!

```
function after(arg1,f)
   print(arg1,f:Get())
end

-- create a future to a print statement
f = async('print','Hello, world')
f:Get() --get the future, note that it's Get() not get()
f:Then(after,arg):Get() --Then called when the future is ready
```

Lua with HPX!

```
--pr is a global function
function pr(a,b)
   print(a,b)
   return a+b
end
--globals are unreliable unless they are registered
HPX_PLAIN_ACTION('pr')
f = async('pr',3,9) --create a future
print(f:Get()) --print the result
```

Lua+HPX - Composing

```
--Create a bunch of futures
tbl = {}
for i=1,n do
  table.insert(tbl,
    async(work,i))
end
```

```
--Repeatedly call wait any
for i=1,n do
  wr = when_any(a):Get()
  table.remove(tbl,wr.index)
  if(wr.index ~= 1)then
    io.write('index='..wr.index..'\n')
    io.write(value='..wr.futures[wr.index]:Get()..'\n')
  end
end
```

Lua+HPX - Composing

```
--Create a bunch of futures
tbl = {}
for i=1,n do
  table.insert(tbl,
    async(work,i))     --wait for all work to finish
end                    wait_all(tbl)
```

Lua+HPX - Composing

```
--Create a bunch of futures
tbl = {}
for i=1,n do
  table.insert(tbl,
    async(work,i))
end
```

```
--function walk
function walk(arg1,futures)
      for i,v in pairs(futures:Get()) do
          print('result',i,v:Get())
      end
end
--wait for all work to finish
when_all(tbl):Then(walk,"arg 1")
```

Lua - Fibonacci

```
function fib(n)
  if(n < 2)then
    return n
  end
  return fib(n-1)+fib(n-2)
end

print('fib='..fib(30))
```

Lua+HPX - Fibonacci, parallel

```
function fib(n)
  if(n < 2)then
    return n
  end
  local f1 = async('fib',n-1)
  local f2 = async('fib',n-2)
  return f1:Get()+f2:Get()
end
HPX_PLAIN_ACTION('fib')
--too slow for 30
print('fib='..fib(22))
```

Lua+HPX - Fibonacci, parallel

```
function fib(n)
  if n < 2 then return n; end
  if (n > 21) then
    local n1 = async('fib',n-1)
    return
        async(unwrapped(
          'fadd',n1,fibs(n-2)))
  else
    return fibs(n-1) + fibs(n-2)
  end
end
```

```
function fadd(a,b)
  return a+b; end

function fibs(n)
  if n < 2 then return n; end
  return fibs(n-1)+fibs(n-2); end

HPX_PLAIN_ACTION(
  'fadd','fib','fibs')
print(async('fib',30):Get())
```

Lua+HPX - Closures

```
do
   local a=10
   f = async(function()
      print('a=',a)
      --prints a= 10
   end)
   f:Get()
end
```

```
--[[
Multi-line comment:

When closures are passed into
HPX they pass by value, not
reference, so you can't do
all the same things with them
that you can do with regular
closures
--]]
```

Lua+HPX - Parallel Loops

```
--serial code
local lo,hi
lo = 1
hi = 1000
for i=lo,hi do
   do_work(i)
end
```

```
--ready for parallelism
--gr = grain size
local lo,hi,gr
lo = 1
hi = 1000
gr = 4
for i0=lo,hi,gr do
  local ihi =
    math.min(hi,i0+gr-1)
  for i=i0,ihi do
    do_work(i)
  end
end
```

```
--parallelism added
local lo,hi,gr,fs
lo = 1
hi = 1000
gr = 4
fs = {}
for i0=lo,hi,gr do
  local ihi =
    math.min(hi,i0+gr-1)
  fs[#fs+1] =
    async(function()
      for i=i0,ihi do
        do_work(i)
      end
    end)
  wait_all(fs)
end
```

Lua+HPX - Parallel Loops

```
--serial code
local lo,hi
lo = 1
hi = 1000
for i=lo,hi do
   do_work(i)
end
```

```
--parallel code
local lo,hi
lo = 1
hi = 1000
for_each(lo,hi,
function(i)
   do_work(i)
end)
```

```
--parallel code
grain_size=10
local lo,hi
lo = 1
hi = 1000
for_each(lo,hi,
function(i)
   do_work(i)
end,grain_size)
```

Lua+HPX - Helpers

```
--identify an HPX/Lua object
print(obj:Name())
```

```
--create a table
tbl = table_t.new()
--create a vector of numbers
vec = vector_t.new()
--these special table
--and vector objects
--pass between threads
--by reference, not
--value
```

```
--globals within a locality
globals.x = 3

function globals.add(a,b)
  return a+b
end
```

Lua+HPX - Exercises

1) Quicksort - Insert futures to parallelize a quicksort function.

2) Matmul - Insert futures to parallelize a matrix multiply function.

3) 1d_stencil - Parallelize the main loop for a heat equation evolution

4) Integ - Parallelize the numerical integrator.

Image from Wikipedia

Lua+HPX - Distributed

```
--find the current locality
here = find_here()
--execute something on 'here'
f = async(here,'print','hello')
f:Get()
remotes = find_remote_localities()
for i,r in ipairs(remotes) do
  -- .. does string concatenation
  local str = 'loc='..r
  async(r,'print',str) --remote!
end
```

Lua+HPX - Components

```
--create at a locality
loc = find_here()
c = component.new(loc)
--set a value on the component
fut = c:Set("field",3)
print(fut:Get()) --returns nil
--get a value from the component
print(c:Get("field"):Get())
```

```
--call a component method like this
c:Set("hello",function(self)
    print("hello") end)
c:Call("hello")
--or do it this way...
c:Set("x",1)
c:Call(
    function(self) print("x",self.x) end)
```

Lua+HPX - Performance Counters

```
--Simple example of counters
--First, discover the counters
d = hpx.discover_counter_types()
counter = nil

--Search for the specific counter
--we are interested in
for i,c in ipairs(d) do
  if string.find(c["fullname_"],"uptime") then
    print(i,c["fullname_"])
    counter = hpx.get_counter(c)
  end
end
```

Lua+HPX - Performance Counters

```
--Get the counter value
value = get_value(counter)
t1 = value["time_"]
print("time=",t1)
--Do some work
os.execute("sleep 1")
--Get the counter value again
value = get_value(counter)
t2 = value["time_"]
print("time=",t2)
--Compute the elapsed time
print("delt=",(t2-t1)*1e-3)
```

Lua+HPX - Transposition

```
function transpose(inp,outp)
  local i,j,n
  n = #inp
  for i=1,n do
    for j=1,n do
      outp[i][j] = inp[j][i]
    end
  end
end
```

Lua+HPX - Transposition by Block

00	10	20	30	40	50	60	70	80
01	11	21	31	41	51	61	71	81
02	12	22	32	42	52	62	72	82
03	13	23	33	43	53	63	73	83
04	14	24	34	44	54	64	74	84
05	15	25	35	45	55	65	75	85
06	16	26	36	46	56	66	76	86
07	17	27	37	47	57	67	77	87
08	18	28	38	48	58	68	78	88

00	01	02	03	04	05	06	07	08
10	11	12	13	14	15	16	17	18
20	21	22	23	24	25	26	27	28
30	31	32	33	34	35	36	37	38
40	41	42	43	44	45	46	47	48
50	51	52	53	54	55	56	57	58
60	61	62	63	64	65	66	67	68
70	71	72	73	74	75	76	77	78
80	81	82	83	84	85	86	87	88

Lua+HPX - Transposition by Block

```
function transpose(inp,outp,block_count,block_size)
  local i,j,ib,jb
  for ib=1,block_count do
    for jb=1,block_count do
      for i=1,block_size do
        for j=1,block_size do
          outp[ib][jb][i][j] = inp[jb][ib][j][i]
        end
      end
    end
  end
end
```

Lua+HPX - Transposition by Block

```
function create_matrix(block_count,block_size)
  local i,ib,jb,m,mb
  m = table_t.new()
  for ib=1,block_count do
    m[ib] = table_t.new()
    for jb=1,block_count do
      mb = table_t.new()
      m[ib][jb] = mb
      for i=1,block_size do
        mb[i] = vector_t.new()
        mb[i][block_size] = 0
      end; end; end; return m; end
```

Lua+HPX - In Parallel Transposition by Block

```
function transpose(inp,outp,block_count,block_size)
  local i,j,ib,jb,v,f
  v={}
  for ib=1,block_count do
    for jb=1,block_count do
      f = async(transpose_s,inp[jb][ib],
        outp[ib][jb],block_size)
      v[#v+1] = f
    end
  end
  wait_all(f)
end
```

```
function
transpose_s(inp,outp,block_size)
  local i,j
  for i=1,block_size do
    for j=1,block_size do
      outp[i][j] = inp[j][i]
    end
  end
end
```

Lua+HPX - Distributed Transposition by Block

```
function create_matrix(block_count,block_size)
  local i,ib,jb,m,mb,all,loc
  all = find_all_localities()
  m = table_t.new()
  for ib=1,block_count do
    m[ib] = table_t.new()
    for jb=1,block_count do
      loc = all[(ib+jb) % #all + 1]
      sub = component.new(loc)
      sub:Call(create_matrix_s,
          block_size)
      m[ib][jb] = sub
    end;  end ;  return m;end
```

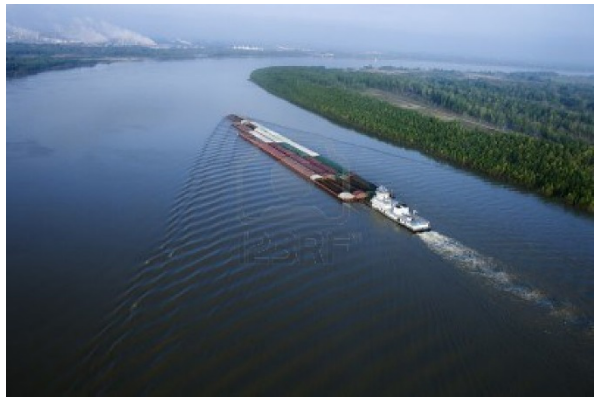
```
function
create_matrix_s(self,block_size)
  mb = table_t.new()
  for i=1,block_size do
    mb[i] = vector_t.new()
    mb[i][block_size] = 0
  end
  self.mb = mb
end
```

Lua+HPX - Distributed Transposition by Block

```
function transpose(inp,outp,block_count,block_size)
  local i,j,ib,jb,v,f
  v={}
  for ib=1,block_count do
    for jb=1,block_count do
      f = outp[ib][jb]:Set("mb",
        inp[jb][ib]:Call(transpose_s,block_size))
      v[#v+1] = f
    end
  end
  wait_all(f)
end
```

```
function
transpose_s(self,block_size)
  local i,j,m,inp
  inp = self.mb;
  m = table_t.new()
  for i=1,block_size do
    m[i] = vector_t.new()
    for j=1,block_size do
      m[i][j] = inp[j][i]
    end
  end
  return m
end
```

LSU CENTER FOR COMPUTATION & TECHNOLOGY



STE||AR GROUP