Julia on HPC Platform

DJ Choi
SDSC
Feb 21, 2017

Julia is

- An open source programming Language for Scientific applications like Matlab, Python+SciPy, R (Statistical application).
- Has many packages/modules on mathematics, bio science, chemistry, deep learning, bigdata, visualization, etc.
  - There are Julia >1,000 registered packages: HDF, MPI..
- Easier installation and maintenance of the packages using Pkg commands:
  - Example: Pkg.add(“HDF”), Pkg.status, Pkg.rm, Pkg. update

Starting to Use

- Module load julia
  - export MODULEPATH=/home/dchoi/apps/modulefiles:$MODULEPATH
  - module load julia
- Interactive run:
  - Julia
- Script run:
  - julia test.jl

Simple Use

- Julia> 3+4
- Julia> 7
-
- Julia> for i=1:2

  Print i

  end
- Julia> if var>5

  println(“greater than 5”)

  else

  println(“less than 5”)

  end

Vector and Matrix

```
julia> a=[1 2 3 4]
1×4 Array{Int64,2}:
 1  2  3  4

julia> b=[1 2 3 4
          5 6 7 8
          1 2 3 4
          5 6 7 8]
4×4 Array{Int64,2}:
 1  2  3  4
 5  6  7  8
 1  2  3  4
 5  6  7  8
```

Variable Type

```
julia> a=(b/c)::Float64

julia> type food
            bread
            drink
          end
Julia> mycar=food(“Italian”,”Orange”)
```

Function

```
Julia> function f(x,y)
        2x+y
         end

Julia> a=f(1,2)
```

Module Example

```
module mymodule
        export foo
        function foo()
            println("using mymodule")
        end
        end
mymodule

julia> using mymodule
```

Unicode

- Program close to Math. expression.
- Input Ctrl+v u+number in vi

```
∑(x) = sum(x)
√(x) = sqrt(x)
function normalize(x)
    return x ./ √(∑(x.^2))
end
```

Using a Package

Solve

X”(t)+x(t)=0

t: 0<t<4*pi
X(0), x’(0)=[0,0.1]

x1=x
x2=x’
x1’=x2
x2’=x”=-x1

y’={x1,x2}’= 0 1 {x1 x2}
-1 0

```
Julia>using ODE

function f(t, y)
        (x1, x2) = y

        x1_prime = x2
        x2_prime = -x1

        [x1_prime; x2_prime]
    end
julia> const starty = [0.0; 0.1]

julia> time=0:0.1:4*pi

julia> t, y = ode45(f, starty, time);

julia> x = map(y -> y[1], y)
```

Package DifferentialEquations

- Module for solving differential equations in Julia: Discrete equations, ordinary, stochastic ordinary, algebraic, delay and partial differential equations like simple finite element method.
- Developed by Christopher Rackauckas at University of California, Irvine.
- To install do Pkg.add(“DifferentialEquations”) at julia prompt
- Or available at https://github.com/JuliaDiffEq/DifferentialEquations.jl

Parallel operation in JULIA

- Julia’s parallel implementation: Julia provides a multiprocessing environment
- Using MPI Package: uses system installed MPI

Setting Number of Processors

- julia -p 2

```
julia> np=nprocs()
3

julia> addprocs(3)   ! rmprocs()
3-element Array{Int64,1}:
 4
 5
 6

julia> np=nprocs()
6
```

Quick Test using Julia's Implementation

- [dchoi@comet-15-56 ~]$ julia -p 1
- julia> @time nheads=@parallel (+) for i=1:20000000000
  Int(rand(Bool))
- end
  39.446589 seconds (413.24 k allocations: 17.373 MB, 0.14% gc time)
  9999938807
-

- [dchoi@comet-15-56 ~]$ julia -p 4
- julia> @time nheads=@parallel (+) for i=1:20000000000
  Int(rand(Bool))
  end
  10.900994 seconds (395.73 k allocations: 16.489 MB, 0.53% gc time)
  10000018946

Julia’s Parallel Implementation

```
@everywhere function count_heads(n)
    c::Int = 0
    for i=1:n
        c += rand(Bool)
    end
    c
end

tic()
a = @spawn count_heads(10000000000)
b = @spawn count_heads(10000000000)
fetch(a)+fetch(b)
Toc()

julia> toc()
elapsed time: 21.018663023 seconds
```

MPI Package Module Build on Comet

- Available from https://github.com/JuliaParallel/MPI.jl In
- Installed on top of the MVAPICH2 in COMET system

```
Pkg.add(“MPI”)
Pkg.build(“MPI”)

[dchoi@comet-ln3 ~]$ cat .juliarc.jl
ENV["JULIA_MPI_C_LIBRARIES"] = "-L/opt/mvapich2/intel/ib/lib -lmpicxx -lmpi"
ENV["JULIA_MPI_C_INCLUDE_PATH"] = "-I/opt/mvapich2/intel/ib/include"
ENV["JULIA_MPI_Fortran_LIBRARIES"] = "-L/opt/mvapich2/intel/ib/lib -lmpifort -lmpi"
ENV["JULIA_MPI_Fortran_INCLUDE_PATH"] = "-I/opt/mvapich2/intel/ib/include"
ENV["JULIA_MPI_FORTRAN_INCLUDE_PATH"] = "-I/opt/mvapich2/intel/ib/include"

mpirun -np 4 julia  hello.jl
```

MPI Example

```
import MPI

function do_hello()
    comm = MPI.COMM_WORLD
    println("Hello world, I am $(MPI.Comm_rank(comm)) of $(MPI.Comm_size(comm))")
    MPI.Barrier(comm)
end

function main()
    MPI.Init()
    do_hello()
    MPI.Finalize()
end

main()
```

```
mpirun -np 4 julia  hello.jl
Hello world, I am 0 of 4
Hello world, I am 1 of 4
Hello world, I am 2 of 4
Hello world, I am 3 of 4
```

Some MPI Functions

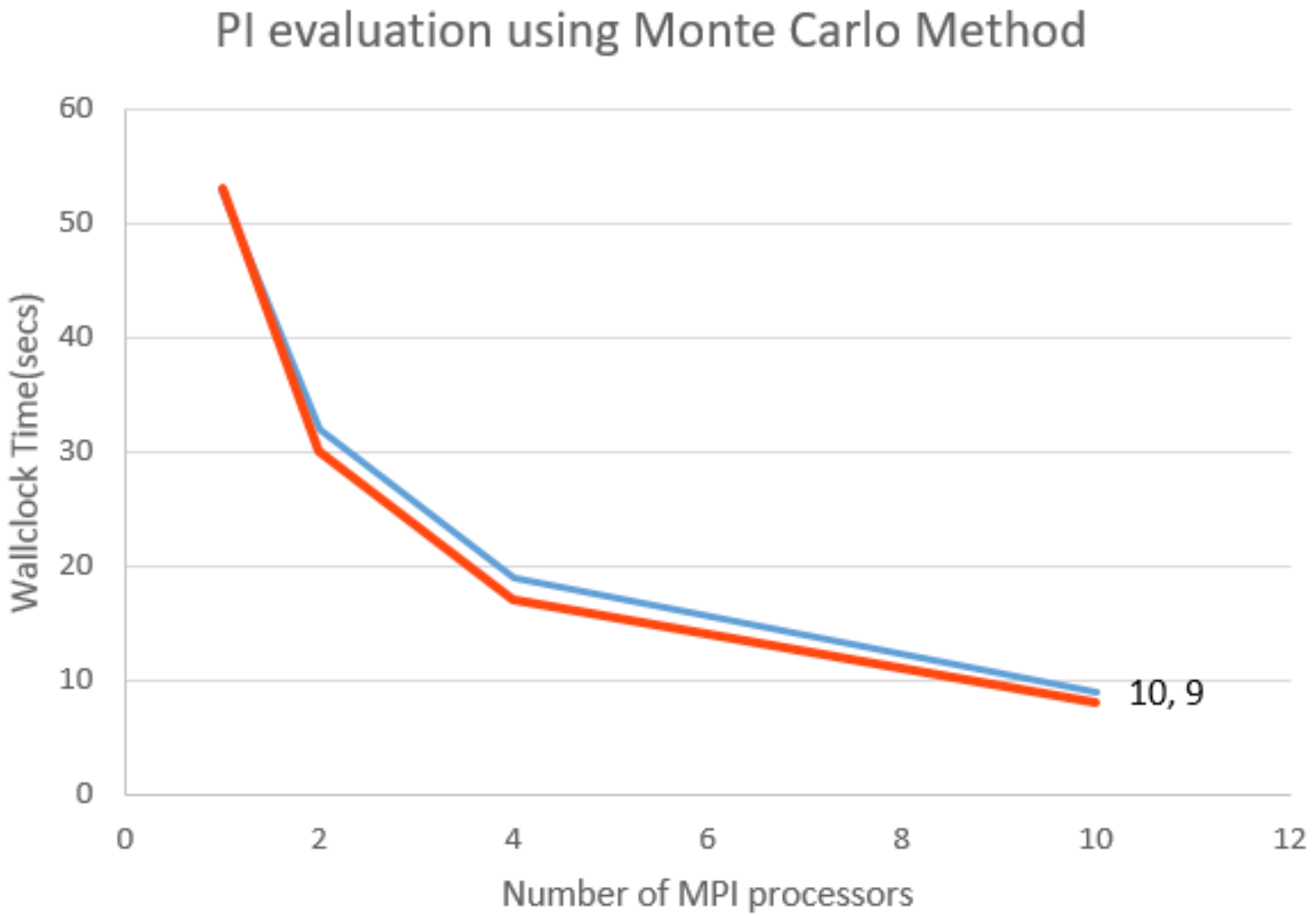
Fortran	Julia
MPI_Init	MPI.Init
MPI_Send	MPI.Send
MPI_Recv	MPI.Recv!

Another Example: PI Computation using Monte Carlo Method

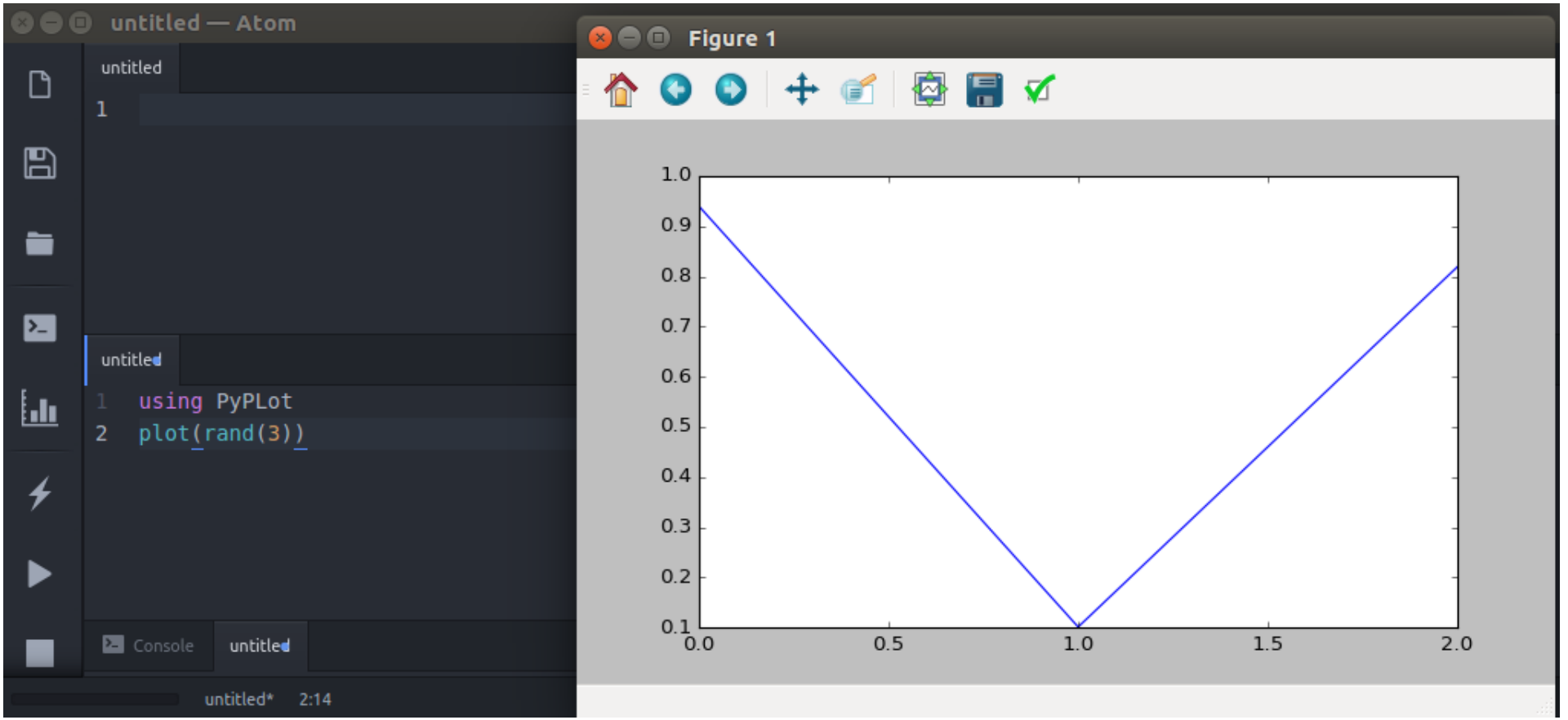
```
MPI.Init()
n_evals = 10^8 # desired number of MC reps
n_returns = 1
batchsize = 10^5
montecarlo(pi_wrapper, pi_monitor,
          MPI.COMM_WORLD, n_evals, n_returns, batchsize)
MPI.Finalize()
```

```
Function montecarlo(mc_eval::Function...)
if rank > 0
          ...
        MPI.Send(contrib, 0, 0, comm)
else
          ...
        MPI.Recv!(contrib, MPI.ANY_SOURCE, 0, comm)
          ...
end
```

Performance



Julia Development with ATOM IDE



Installation and Maintenance Issue

- Installing and maintaining Julia and its packages involve complex issues: OS type, version, library environment, Julia versions, package versions, their dependencies.
- Virtual environment like JetStream and singularity use.

Julia with Singularity

- Singularity image started with debian-julia.def from Andrea Zonca at SDSC

```
julia -e 'Pkg.add("Gadfly")' && julia -e 'Pkg.add("RDatasets")' && julia --startup-file=yes -e 'Pkg.add("HDF5")' && julia -e 'Pkg.add("MPI")'
julia -e 'using Gadfly' && julia -e 'using RDatasets'&& julia -e 'using HDF5'
julia -e 'Pkg.build("MPI")' && julia -e 'using MPI'
```

Run command in Comet system

```
ibrun -v singularity exec juliaC.img /usr/local/julia/bin/julia /usr/bin/pi.jl
```

Summary

- Rich packages covering many computational areas.
- Good potential to improve parallel performance and reliability on HPC platforms
- Good programming tool for computational scientific research.
- Future work on scalability study, accelerator, clound/virtual environment use and big data applications with Julia.

Reference

- http://docs.julialang.org/
- https://github.com/JuliaDiffEq/DifferentialEquations.jl
- https://github.com/JuliaParallel/MPI.jl
- https://github.com/pluskid/Mocha.jl
- https://github.com/pcmoritz/Strada.jl
- http://julialan.org
- https://github.com/zonca/singularity-comet