

Native distributed and MPI parallelism in the high-level language Julia for quantum Monte Carlo

Mingrui Yang^{1,2,3}, Dr Elke Pahl^{3,4,5} and Prof. Joachim Brand^{1,2}

¹ New Zealand Institute for Advanced Study and Centre for Theoretical Chemistry and Physics,
Massey University, Auckland, New Zealand

² Dodd-Walls Centre for Photonic and Quantum Technologies, New Zealand

³ MacDiarmid Institute for Advanced Materials and Nanotechnology, New Zealand

⁴ Department of Physics, University of Auckland, Auckland, New Zealand

⁵ School of Natural and Computational Sciences and Centre for Theoretical Chemistry and Physics,
Massey University, Auckland, New Zealand



DODD-WALLS CENTRE
for Photonic and Quantum Technologies



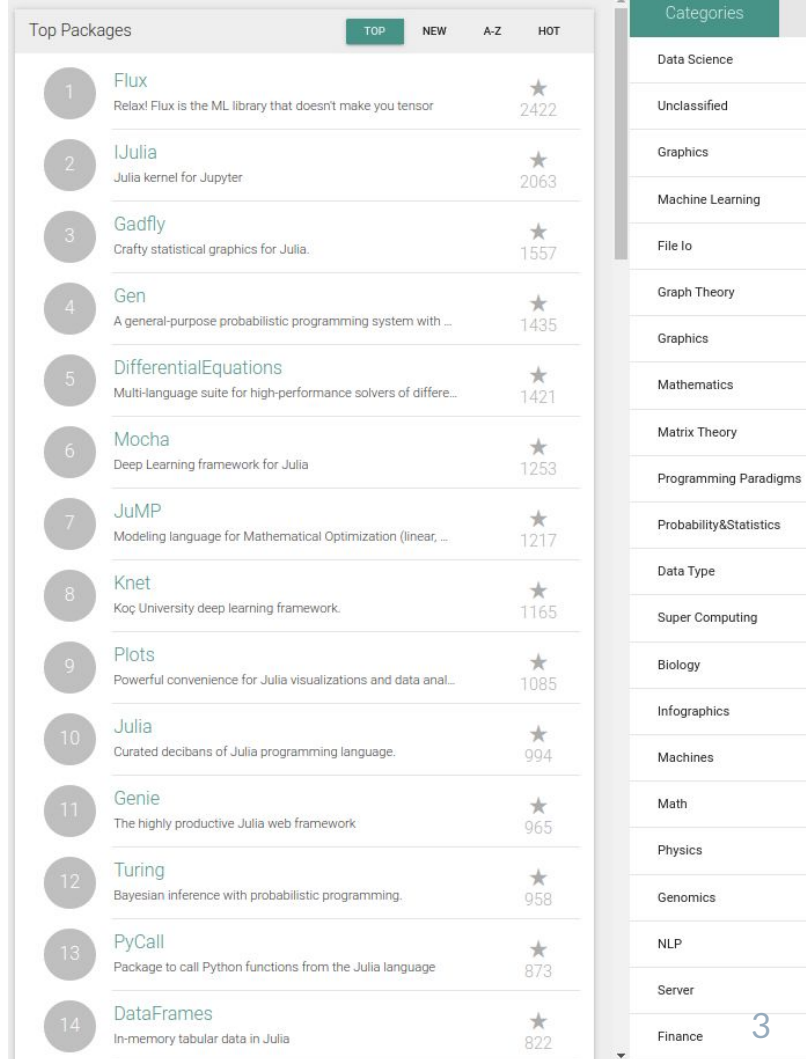
Outline

1. Julia on HPC
2. Paradigms of parallelism
3. Monte Carlo algorithm
4. Profiling the parallel Julia code on NeSI
(NeSI consultancy project)

julia: the high level programming language

Key features:

- ▶ It's new: released in 2012; v1.0 in 2018
- ▶ High level language: fewer lines of code
- ▶ High performance: just-in-time (JIT) compiler
- ▶ 4225 (today) packages available in Julia =>
- ▶ Rapidly growing community
- ▶ Open source (it is free!)
- ▶ **Designed for parallel computing**



The screenshot displays the Julia Package Registry interface. At the top, there are tabs for 'TOP', 'NEW', 'A-Z', and 'HOT'. Below this is a list of 14 top packages, each with a rank in a circle, the package name, a brief description, a star icon, and a star count. To the right, a vertical sidebar lists various categories, with 'Finance' highlighted at the bottom and a large number '3' next to it.

Rank	Package Name	Description	Star Count
1	Flux	Relax! Flux is the ML library that doesn't make you tensor	2422
2	IJulia	Julia kernel for Jupyter	2063
3	Gadfly	Crafty statistical graphics for Julia.	1557
4	Gen	A general-purpose probabilistic programming system with ...	1435
5	DifferentialEquations	Multi-language suite for high-performance solvers of differe...	1421
6	Mocha	Deep Learning framework for Julia	1253
7	JuMP	Modeling language for Mathematical Optimization (linear, ...)	1217
8	Knet	Koç University deep learning framework.	1165
9	Plots	Powerful convenience for Julia visualizations and data anal...	1085
10	Julia	Curated decibans of Julia programming language.	994
11	Genie	The highly productive Julia web framework	965
12	Turing	Bayesian inference with probabilistic programming.	958
13	PyCall	Package to call Python functions from the Julia language	873
14	DataFrames	In-memory tabular data in Julia	822

Categories: Data Science, Unclassified, Graphics, Machine Learning, File Io, Graph Theory, Mathematics, Matrix Theory, Programming Paradigms, Probability&Statistics, Data Type, Super Computing, Biology, Infographics, Machines, Math, Physics, Genomics, NLP, Server, Finance 3

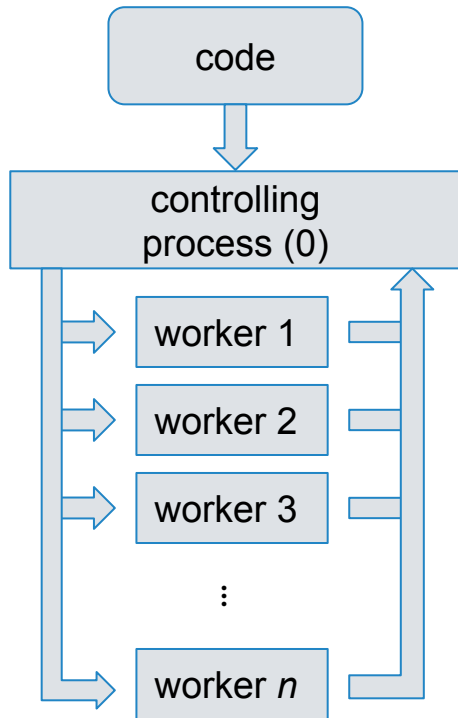
Parallel computing with

Options:

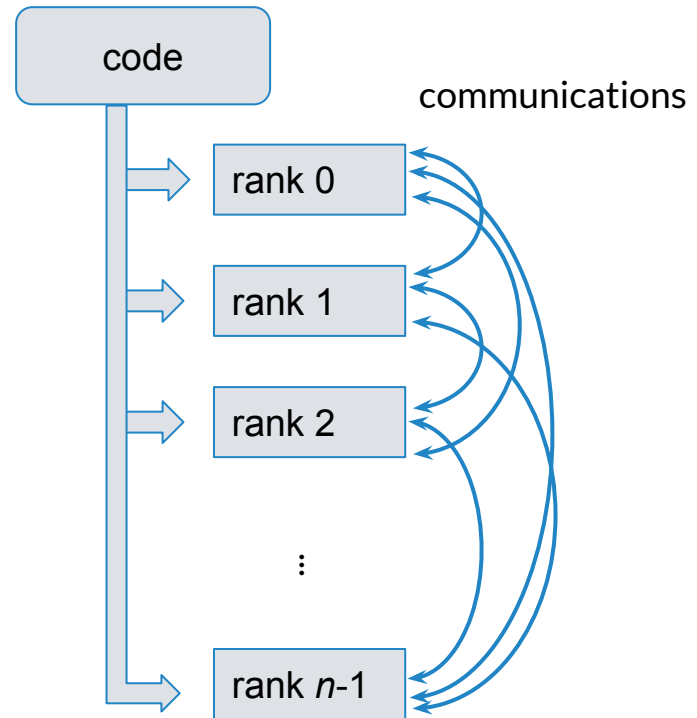
- ▶ **Distributed Computing (built-in with Distributed.jl, Julia standard library)**
(distributed memory, works across nodes)
- ▶ **Multi-Threading (built-in with @threads macro, available in Julia v1.3 or higher)**
(shared memory, works only within a single node)
- ▶ **Message Passing Interface (with MPI.jl wrapper of external C MPI library)**
(distributed memory, works across nodes with fast hardware communication)

Paradigms: Distributed vs MPI (for n processors)

Distributed.jl (built-in Julia package):



MPI.jl (with external MPI library):

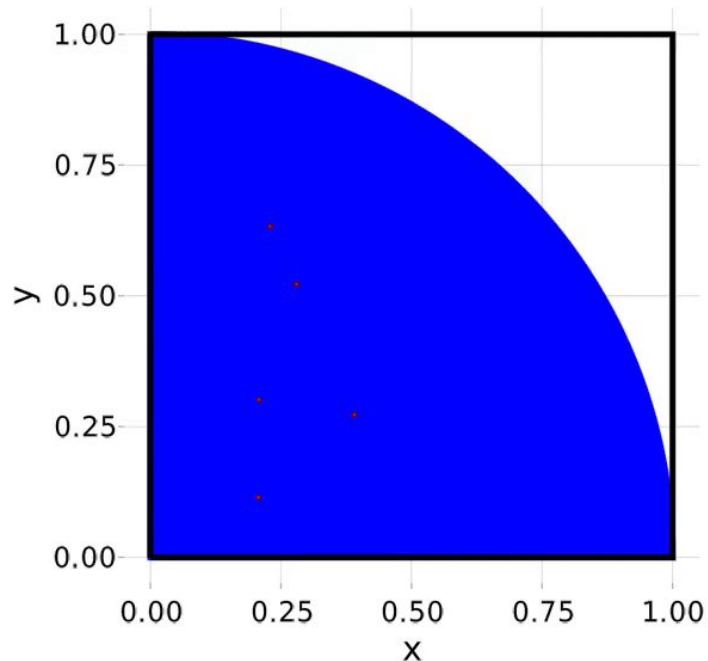


Running Julia on HPC

- ▶ Julia is available as a module on NeSI's Mahuika & Maui, alternatively you can use the binary version
- ▶ For Distributed, use `--ntasks=1` and `--cpus-per-task=n` for slurm, and use `julia -p n <your_julia_code>.jl` to run your code
- ▶ For MPI, use `--ntasks=n` and `--cpus-per-task=1` in slurm, and use `srun julia <your_julia_code>.jl` to run your code
A MPI module may also need to be loaded depends on the Julia build

Monte Carlo methods

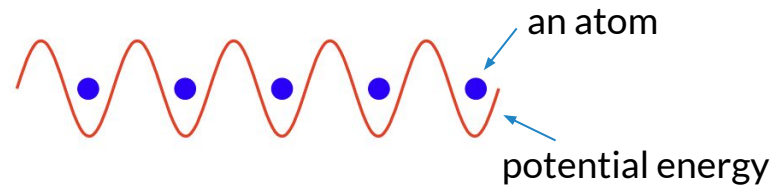
Estimating the value of π with random numbers:



$$\text{value of } \pi = 4 \frac{(\text{area the } \color{blue}{\text{quarter-circle}})}{(\text{area the } \square)} \approx 4 \frac{(\# \text{ of dots inside the } \color{blue}{\text{quarter-circle}})}{(\text{total } \# \text{ of dots})}$$

Monte Carlo sampling becomes very efficient for high-dimensional problems

Ultracold atom physics: experimentalists “put” particles into washboard-like potential:



The wave function:

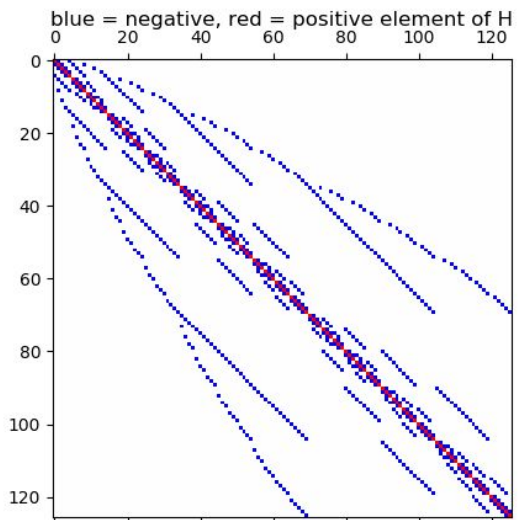
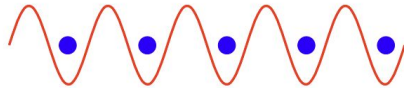
$$\Psi = \sum_i c_i |D_i\rangle$$

a coefficient a configuration

Monte Carlo methods

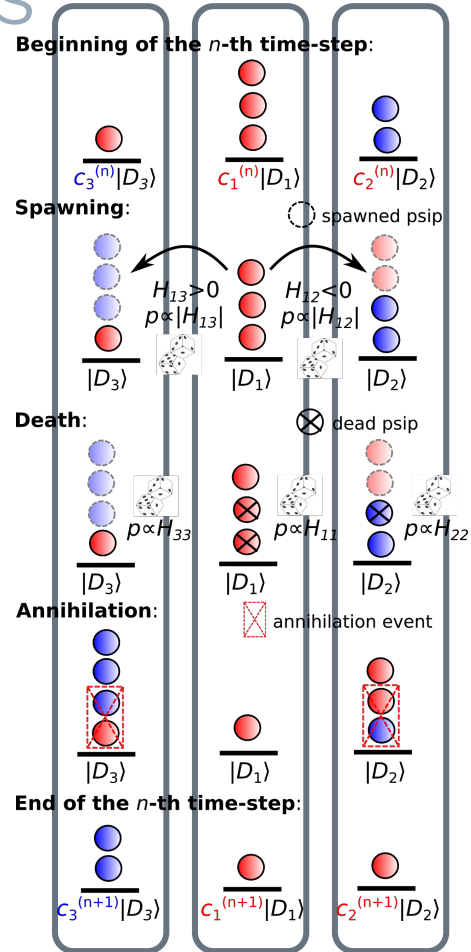
Solve it as a (sparse) matrix eigenvalue problem

Schrödinger equation $\mathbf{H}\Psi = E\Psi$



Matrix dimension can get very large $\gg 10^{100}$!!!

Full Configuration Interaction quantum Monte Carlo (FCIQMC) walker dynamics:



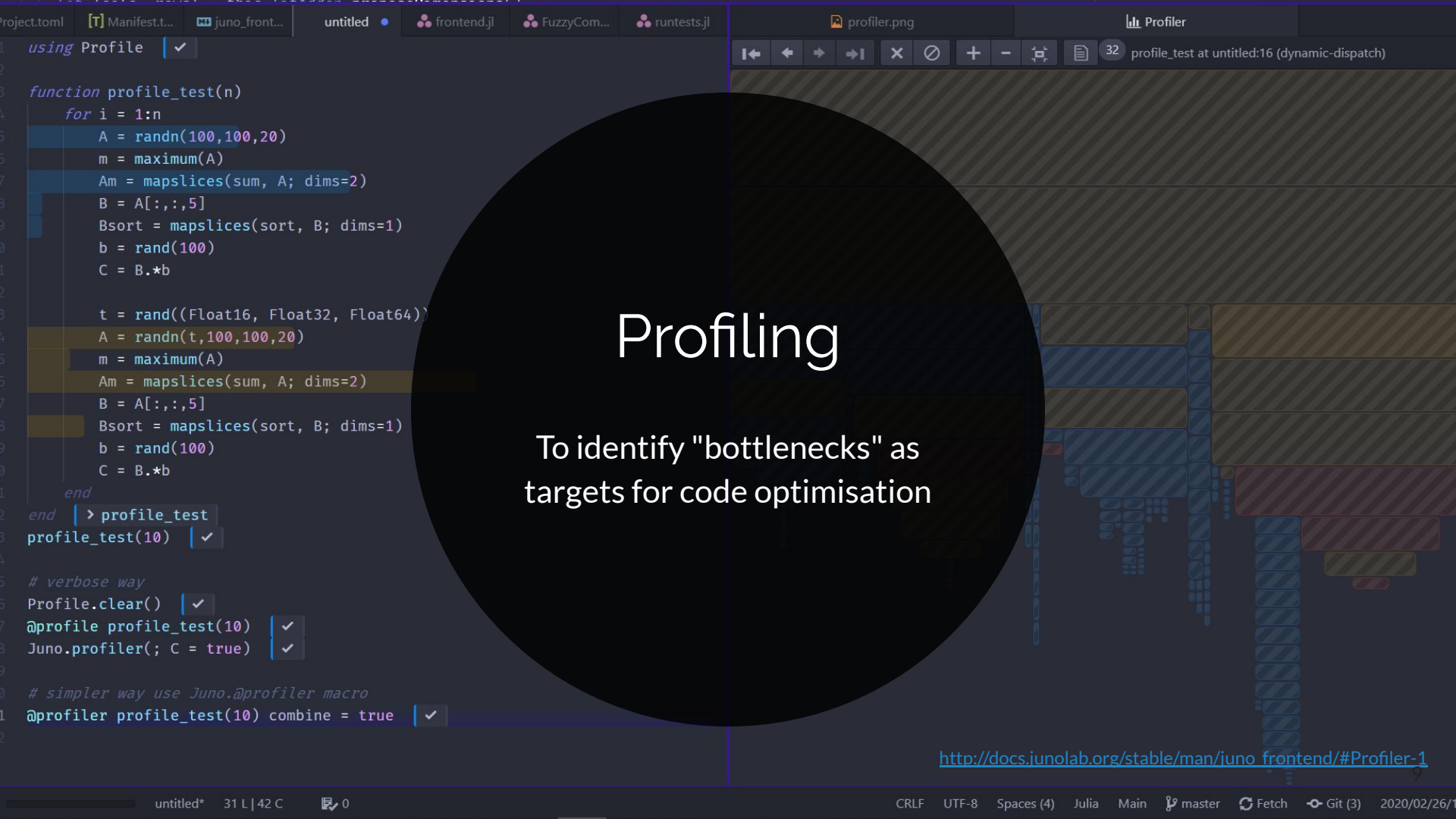
The wave function:

$$\Psi = \sum_i c_i |D_i\rangle$$

an integer \uparrow a bit "address" 01011011100...

Typically 10^6 to 10^9 walkers are used
Need **parallelism** and **HPC!**

- $|D_i\rangle$: configuration
- c_i : coefficient
- : (+ve) walker
- : (-ve) walker



using Profile

```
function profile_test(n)
    for i = 1:n
        A = randn(100,100,20)
        m = maximum(A)
        Am = mapslices(sum, A; dims=2)
        B = A[:, :, 5]
        Bsort = mapslices(sort, B; dims=1)
        b = rand(100)
        C = B.*b

        t = rand((Float16, Float32, Float64))
        A = randn(t,100,100,20)
        m = maximum(A)
        Am = mapslices(sum, A; dims=2)
        B = A[:, :, 5]
        Bsort = mapslices(sort, B; dims=1)
        b = rand(100)
        C = B.*b
    end
end

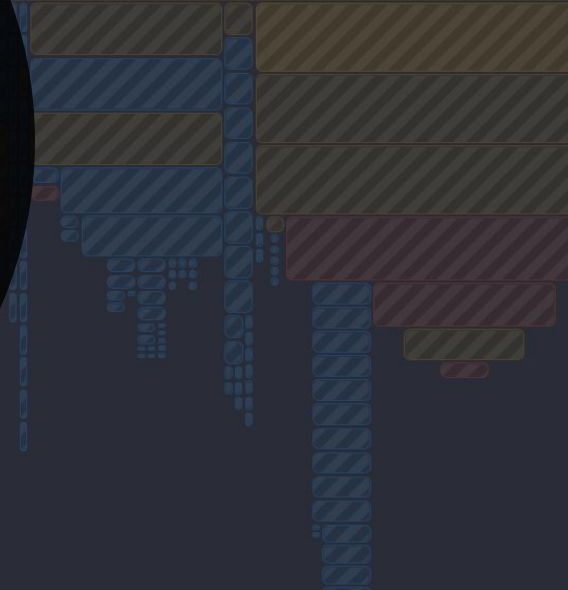
> profile_test
profile_test(10)

# verbose way
Profile.clear()
@profile profile_test(10)
Juno.profiler(; C = true)

# simpler way use Juno.@profiler macro
@profiler profile_test(10) combine = true
```

profiler.png Profiler profile_test at untitled:16 (dynamic-dispatch)

Profiling
To identify "bottlenecks" as
targets for code optimisation



http://docs.iunolab.org/stable/man/juno_frontend/#Profiler-1

```
untitled • frontend.jl FuzzyCom... runtests.jl
using Profile

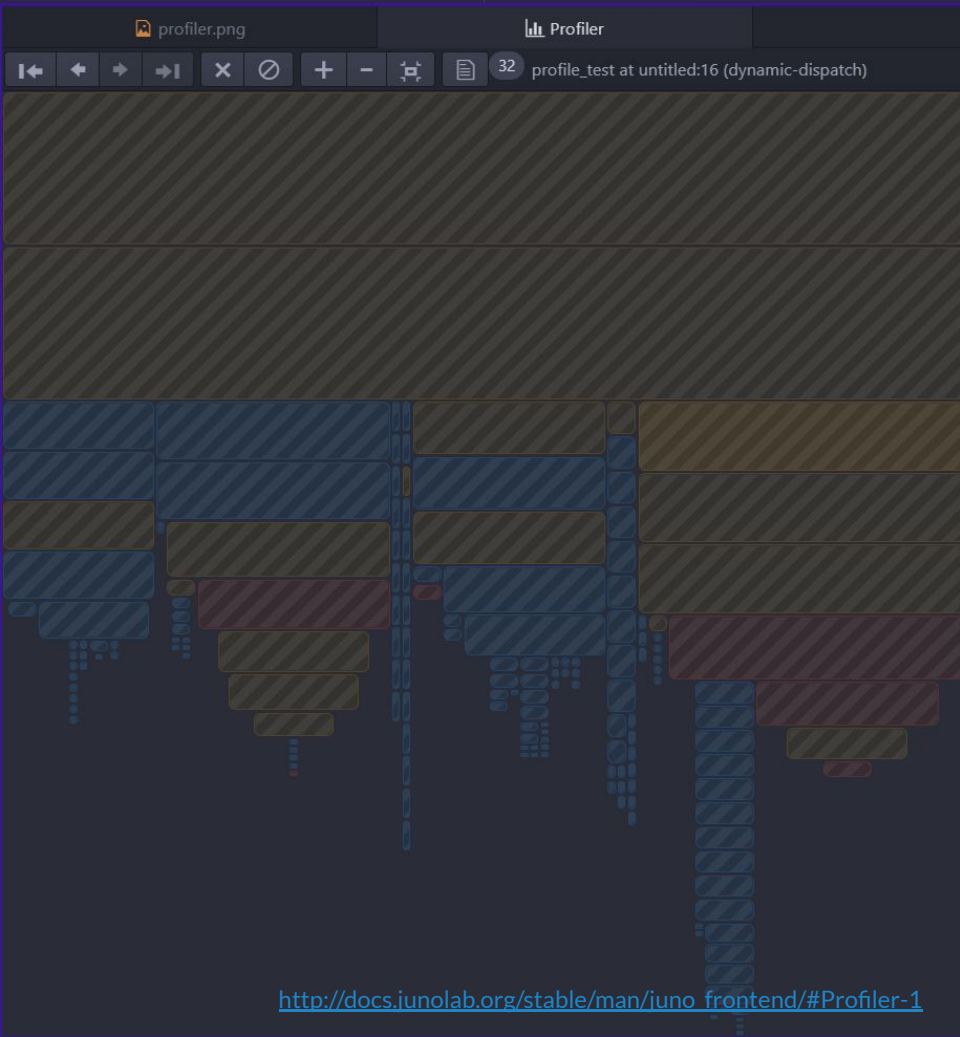
function profile_test(n)
    for i = 1:n
        A = randn(100,100,20)
        m = maximum(A)
        Am = mapslices(sum, A; dims=2)
        B = A[:, :, 5]
        Bsort = mapslices(sort, B; dims=1)
        b = rand(100)
        C = B.*b

        t = rand((Float16, Float32, Float64))
        A = randn(t,100,100,20)
        m = maximum(A)
        Am = mapslices(sum, A; dims=2)
        B = A[:, :, 5]
        Bsort = mapslices(sort, B; dims=1)
        b = rand(100)
        C = B.*b
    end
end

> profile_test
profile_test(10)

# verbose way
Profile.clear()
@profile profile_test(10)
Juno.profiler(; C = true)

# simpler way use Juno.@profiler macro
@profiler profile_test(10) combine = true
```



http://docs.iunolab.org/stable/man/juno_frontend/#Profiler-1

Profiling parallel Julia code (NeSI consultancy project)



With Chris Scott, Alexander Pletzer and Wolfgang Hayek

Challenges:

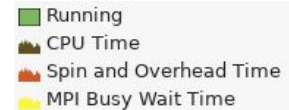
- ▷ Julia is new - limited guidance
- ▷ Some commercial tools do not support Julia yet
- ▷ Combined with parallelisation makes it harder for Julia native profiler
- ▷ Collect and visualise data on HPC (NeSI)

Outcomes:

- ▷ Slurm profiling - limited insight
- ▷ Intel® VTune™ Profiler works well for Julia with parallelism
- ▷ VTune's GUI is informative and fairly easy to navigate
- ▷ Need a special build for Julia to use VTune (won't work with binary version) - available now (v1.2; v1.4)

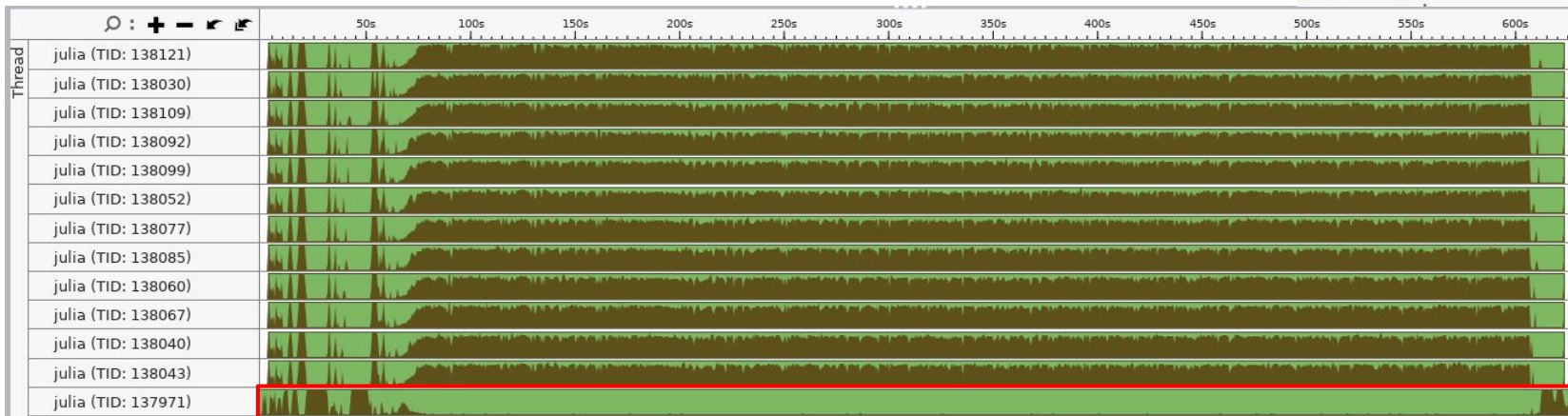
VTune - CPU Utilisation

12 CPUs
Hyperthreading off



Distributed.jl:

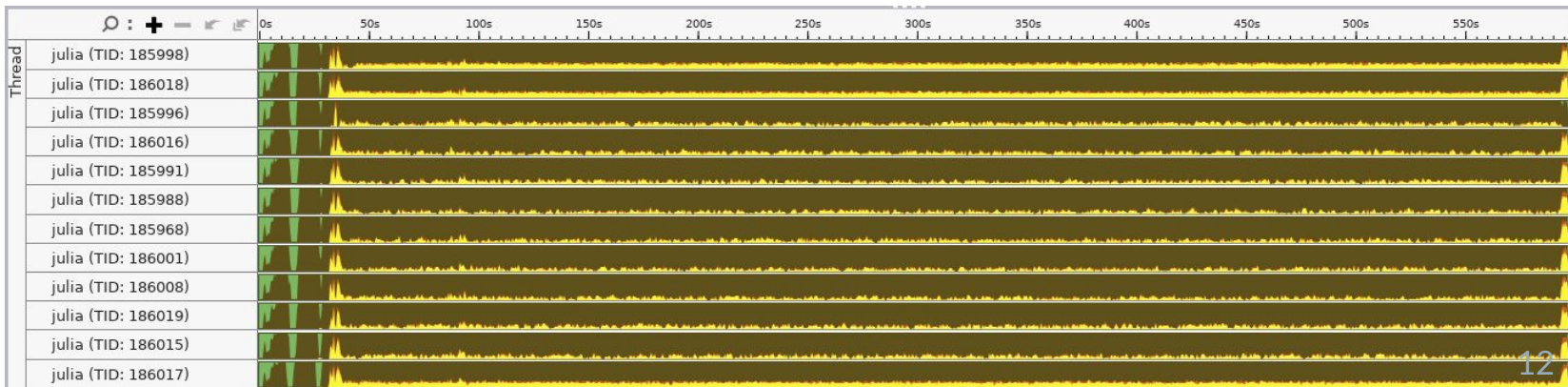
13 processes



← the controlling process

MPI.jl:

12 ranks



What's learned

Distributed:

- ▷ Small scale computation (small number of walkers)
- ▷ Single node (slower across nodes)
- ▷ Good for serial file IO
- ▷ Pure Julia code and no hardware dependency

MPI:

- ▷ Large scale (large number of walker with balanced load)
- ▷ Multiple nodes (fast InfiniBand interconnection)
- ▷ File IO in parallel (HDF5)
- ▷ Coding: more low-level thinking required

Profiling + NeSI consultancy:

- ▷ Optimised code (20x faster)
- ▷ <https://github.com/joachimbrand/Rimu.jl>
- ▷ <https://www.nesi.org.nz/services/consultancy>



Summary

- ▶ Julia is modern and powerful
- ▶ Multiple parallelisms available
- ▶ Many new things to explore
- ▶ NeSI consultancy is very helpful

Acknowledgement



Brand group and CTCP
members at Massey



Chris Scott, Alexander Pletzer
and Wolfgang Hayek

Thank you!