A Julia Package to Analyse Large-Scale Data from RAMSES

Manuel Behrendt

19.04.2023 - RUM2023 - Oxford

LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

ORIGINS Excellence Cluster

MPE

MAX PLANCK INSTITUTE FOR EXTRATERRESTRIAL PHYSICS

MERA in a nutshell

- public since 2020

- written in pure Julia language
  -> very readable code
  -> runs on multiple platforms
  -> fast and interactive: “just-ahead-of-time” (jit + aot) -> optimized machine code
  -> easy to install (package manager)
  -> reproducibility (Julia environment)

- database framework

- many helper functions -> to realise individual analysis purpose (“no black boxes”)

- fast and memory lightweight data reading/saving/handling (leaf-cells; compressed files)

- Many examples, extensive documentation (online and interactive)

- well tested code (compilation, unit-tests, user experiences)

MERA

Simulation Overview
InfoType
passed to all functions and types

Hydro Data → database
Particle Data → database
Gravity Data → database
Clump Data → database
……. → database

multiple dispatch

- subregions/shellregions
- get (derived) quantities (in arrays)
  - filter/masking
- projection
- many other helper functions…
- pre-calculated units
- timestamps for heavy functions
- progress bar
- transparent memory consumption
- interactive function help
- …

I/O

- RAMSES file reader (leaf-cells)
- MERA file writer/reader

reading/writing

information about the simulation **timestamp**

```
gas = gethydro(info);
```

```
[Mera]: Get hydro data: 2023-04-14T19:35:14.581

Key vars=(:level, :cx, :cy, :cz)
Using var(s)=(1, 2, 3, 4, 5, 6, 7) = (:rho, :vx, :vy, :vz, :p, :var6, :var7)

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]

Reading data...
Progress: 100%|████████████████████████████████████████| Time: 0:00:23
Memory used for data table :2.3210865957662463 GB
------------------------------------------------------------
```

loaded memory

progressbar (switchable)

verbose mode (switchable)

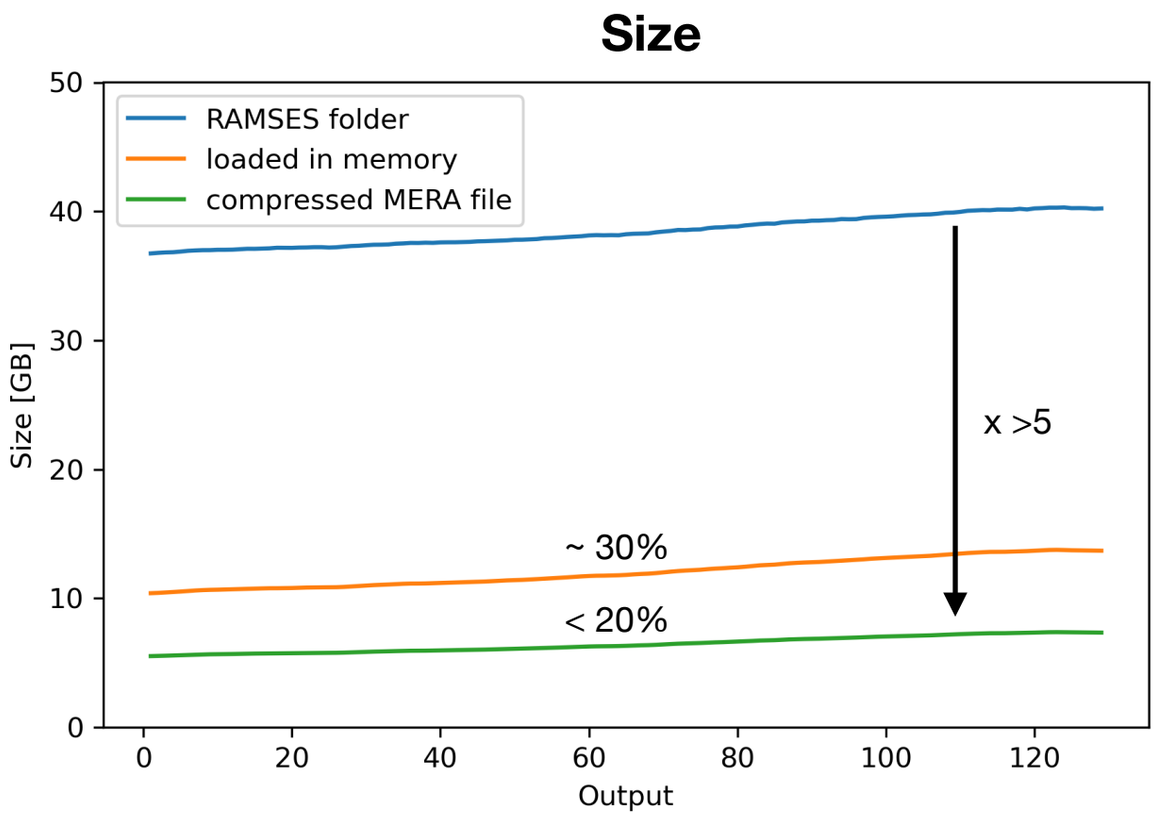
select: variables **select max. level** **physical ranges**

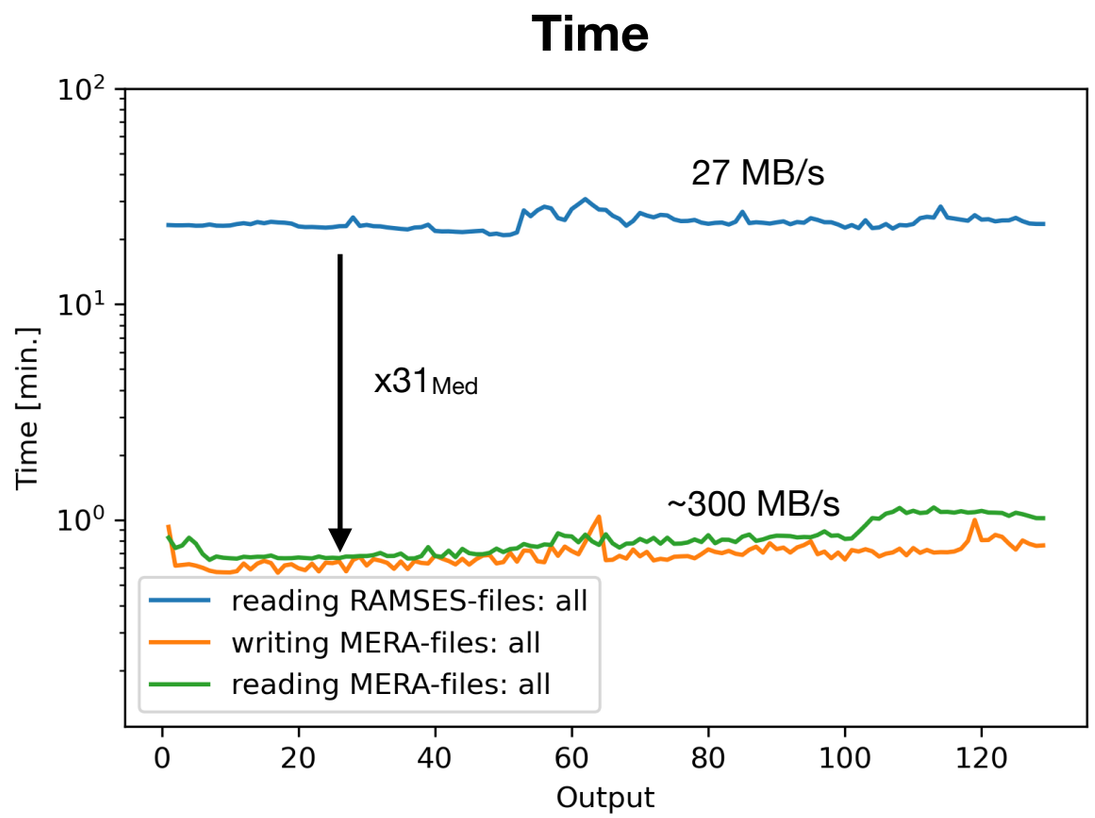
```
gas = gethydro(info, [:rho,:p], lmax=10 ,
                xrange=[-20,20],
                yrange=[-20,20],
                zrange=[-2,2],
                center=[:boxcenter],
                range_unit=:kpc);
```

```
[Mera]: Get hydro data: 2023-04-14T19:33:43.501

Key vars=(:level, :cx, :cy, :cz)
Using var(s)=(1, 5) = (:rho, :p)

center: [0.5, 0.5, 0.5] ==> [24.0 [kpc] :: 24.0 [kpc] :: 24.0 [kpc]]

domain:
xmin::xmax: 0.0833333 :: 0.9166667      ==> 4.0 [kpc] :: 44.0 [kpc]
ymin::ymax: 0.0833333 :: 0.9166667      ==> 4.0 [kpc] :: 44.0 [kpc]
zmin::zmax: 0.4583333 :: 0.5416667      ==> 22.0 [kpc] :: 26.0 [kpc]

Reading data...
Progress: 100%|████████████████████████████████████████| Time: 0:00:17
Memory used for data table :461.47086906433105 MB
------------------------------------------------------------
```

reading/writing

full box - all variables *simulation example*





Simulation with
- 5120 cores
- 7e7-1e8 cells
- ~1e6 particles

hydro: 7 vars (float)
gravity: 4 vars (float)
particles: 5 vars (float)

used compression:
LZ4 is a lossless data compression algorithm

comparison:
zlib writing 3x slower; but similar compression or reading rate

We will work on improving:
- reading speed
- reducing disk space
- ....

database

hydro:

every row == cell *selected variables*

Table with 28320979 rows, 8 columns:

level	**cx**	**cy**	**cz**	rho	vx	vy	vz
6	1	1	1	3.18647e-9	-1.25532	-1.2941	-1.25406
6	1	1	2	3.58591e-9	-1.23262	-1.2677	-1.15465
6	1	1	3	3.906e-9	-1.2075	-1.24194	-1.10506
6	1	1	4	4.27441e-9	-1.16462	-1.19806	-1.06711
6	1	1	5	4.61042e-9	-1.10493	-1.1418	-1.02841
6	1	1	6	4.83977e-9	-1.02686	-1.07472	-0.974529
6	1	1	7	4.974e-9	-0.948004	-1.0109	-0.90304
6	1	1	8	5.08112e-9	-0.879731	-0.960085	-0.825048
6	1	1	9	5.20596e-9	-0.824484	-0.924039	-0.741562
6	1	1	10	5.38372e-9	-0.782768	-0.901728	-0.655418
6	1	1	11	5.67209e-9	-0.754141	-0.891348	-0.569161
6	1	1	12	6.14423e-9	-0.737723	-0.88947	-0.485604
⋮							
10	814	493	514	0.000321702	0.268398	3.15405	0.0687598
10	814	494	509	1.42963e-6	0.00398492	2.66497	0.0523603
10	814	494	510	1.4351e-6	0.00496945	2.66617	0.0724312
10	814	494	511	0.00029515	0.303842	3.16557	0.0942068
10	814	494	512	0.000395273	0.305647	3.16567	0.0847084
10	814	494	513	0.000321133	0.266079	3.15406	0.0728146
10	814	494	514	0.000319678	0.26508	3.15377	0.0712659
10	814	495	511	0.00024646	0.289612	3.16227	0.101586
10	814	495	512	0.000269009	0.290753	3.16253	0.0978822
10	814	496	511	0.000235329	0.285209	3.16122	0.103397
10	814	496	512	0.000242422	0.285463	3.16132	0.10209

particles:

every row == particle *selected variables*

Table with 544515 rows, 8 columns:

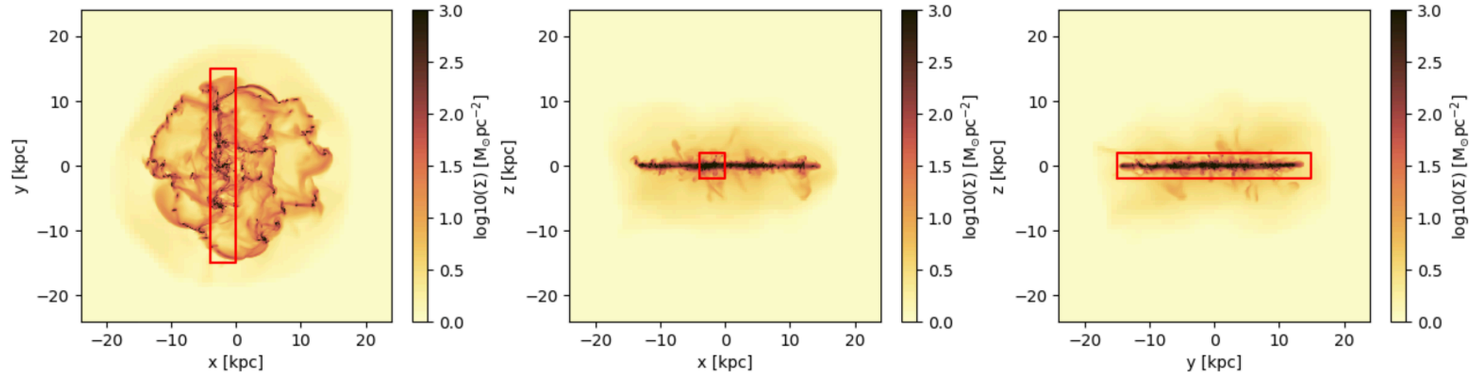
x	**y**	**z**	**id**	**family**	vx	vy	vz
9.17918	22.4404	24.0107	128710	2	0.670852	-2.99998	5.30744e-5
9.23642	21.5559	24.0144	126838	2	0.810008	-2.94186	0.00176287
9.35638	20.7472	24.0475	114721	2	0.93776	-2.88701	-0.000553183
9.39529	21.1854	24.0155	113513	2	0.870351	-2.94154	8.8261e-5
9.42686	20.9697	24.0162	120213	2	0.899373	-2.92516	0.000152925
9.42691	22.2181	24.0137	125689	2	0.717235	-3.01936	0.000285832
9.48834	22.0913	24.0137	126716	2	0.739564	-3.01743	0.000334681
9.5262	20.652	24.0179	115550	2	0.946747	-2.90938	6.47526e-6
9.60376	21.2814	24.0155	116996	2	0.893236	-2.97729	0.00211541
9.6162	20.6243	24.0506	125003	2	0.996445	-2.90176	-0.00160211
9.62155	20.6248	24.0173	112096	2	0.960817	-2.92635	-0.000106305
9.62252	24.4396	24.0206	136641	2	0.239579	-3.13802	-0.00168714
⋮							
38.1381	24.0197	24.0189	130306	2	-0.309088	3.16317	-0.00102494
38.1389	23.7861	24.0209	132099	2	-0.261858	3.17489	-0.00130051
38.1551	24.7717	24.0193	134582	2	-0.47149	3.13718	-0.00125223
38.1637	22.4321	24.0217	139043	2	0.0104468	3.17814	-0.00337595
38.1704	24.9575	24.0096	142954	2	-0.295343	3.08533	0.0012649
38.1777	25.6817	24.0189	126828	2	-0.693701	3.08456	-0.000476848
38.1831	22.3353	24.0217	136860	2	0.0527758	3.16731	-0.00153629
38.212	25.439	24.0163	131937	2	-0.627377	3.08851	0.00089497
38.2518	23.4388	24.0203	135516	2	-0.208999	3.16471	-0.00241952
38.2591	27.1451	24.0267	116411	2	-1.05094	2.98651	-0.00172146
38.2859	24.796	24.0143	132016	2	-0.48386	3.10356	0.000202783

easily apply: filter, grouping, reduce-operations, transform, insertcol, …

subregions

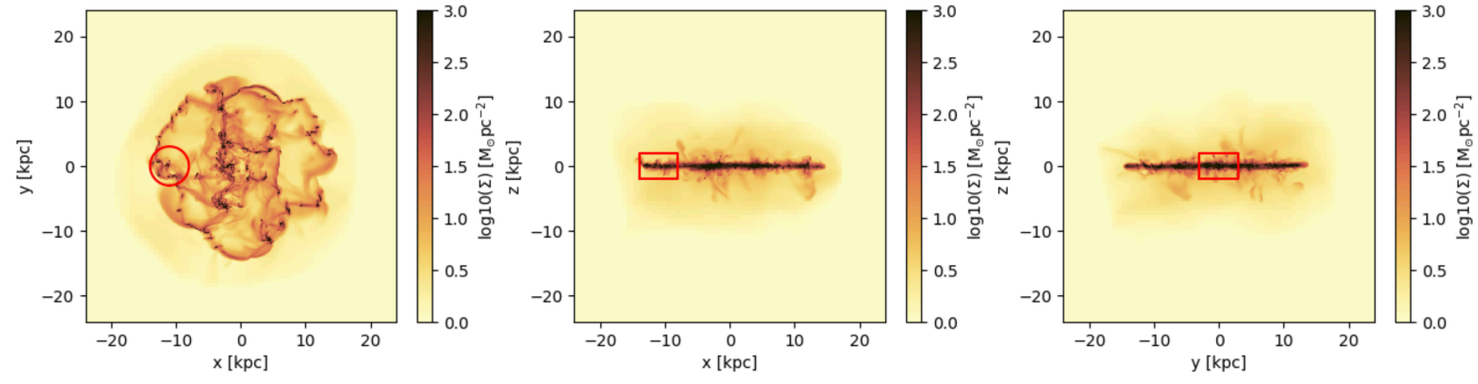
applies to all datatypes (multiple dispatch)

```
gas_subregion = subregion( gas, :cuboid, xrange=[-4., 0.], yrange=[-15., 15.], zrange=[-2., 2.], center=[:boxcenter], range_unit=:kpc)
```



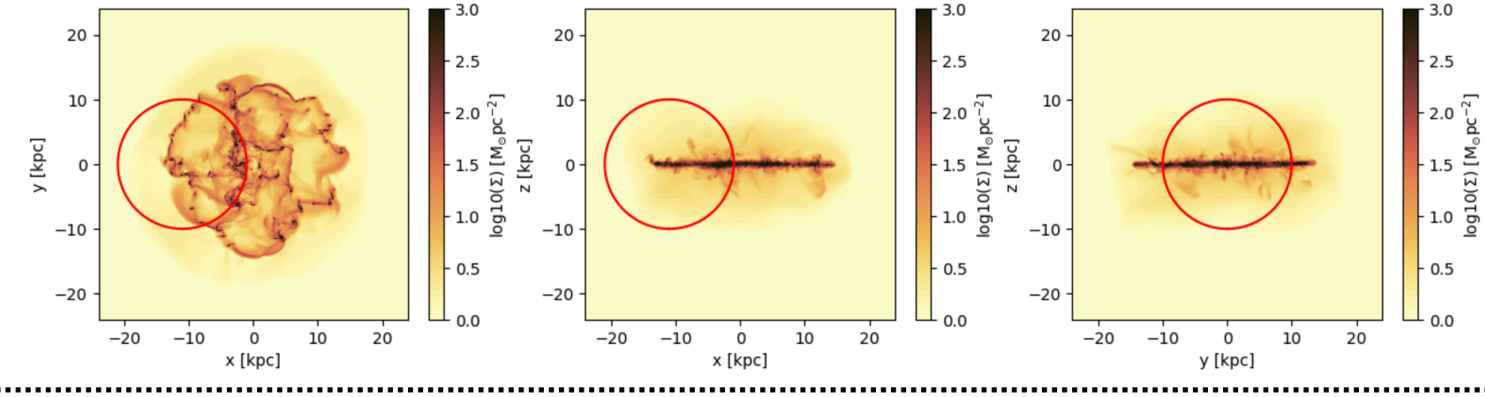
cuboid

```
gas_subregion = subregion( gas, :cylinder,radius=3., height=2., range_unit=:kpc, center=[13., :bc, :bc]); # direction=:z, by default
```



cylinder

```
gas_subregion = subregion( gas, :sphere, radius=10., range_unit=:kpc, center=[13.,:bc,:bc])
```



sphere

get quantities

applies to all datatypes (multiple dispatch)

hydro:

```
# returns an array
mass = getvar(gas, :mass, :Msol)
T    = getvar(gas, :T, :K)
```

```
28320979-element Vector{Float64}:
 226363.43149536985
 253604.64779025855
 275500.87566792965
 307439.06675463304
 350953.21192328085
 396349.67619611276
 438876.2803421858
 476742.09180423705
 508574.1930953268
 531291.9427526356
 540028.3811619072
 529909.430604811
 506089.5601163978
      ⋮
```

particles:

```
# returns an array
mass = getvar(particles, :mass, :Msun)
age  = getvar(particles, :age, :Myr)
```

```
544515-element Vector{Float64}:
 313.6665910729627
 315.93764338919476
 327.87184212012204
 328.789881085896
 323.3416458761294
 317.3549079101028
 316.08708317528027
 327.1934191285363
 326.09927163699354
 318.176251838094
 329.86934302651974
 305.003428103738
 308.64754681491524
   ⋮
```

other datatypes:

.....

request multiple quantities:

```
quantities = getvar(gas, [:mass, :r_cylinder, :vϕ_cylinder],
                         [:Msol, :kpc, :km_s],
                          center=[12.,12.,24.], center_unit=:kpc) # returns a Dict with arrays
```

```
Dict{Any, Any} with 3 entries:
  :r_cylinder  => [15.9099, 15.9099, 15.9099, 15.9099, 15.9099, 15.9099, 15.909…
  :vϕ_cylinder => [1.79831, 1.62656, 1.59698, 1.55068, 1.70983, 2.2191, 2.91637…
  :mass        => [1.34392, 1.51239, 1.64739, 1.80277, 1.94448, 2.04121, 2.0978…
```

many other predefined quantities!

masking

```jl
# create single masks
mask_T    = getvar(gas, :T, :K) .> 1e3 #K
mask_nH   = getvar(gas, :rho, :nH) .< 100. #cm-3

# combine multiple masks
mask_tot = mask_T .* mask_nH # BitVector
```

```jl
# prepare derived quantities
T   = getvar(gas, :T, :K)
nH   = getvar(gas, :rho, :nH)

# create individual masks
mask_Tlower = T .> 100. # K
mask_Tupper = T .< 1e3  # K

mask_nH_lower = nH .> 1e-2 # cm-3
mask_nH_upper = nH .< 100. # cm-3

# combine multiple masks
mask_tot = mask_Tlower .* mask_Tupper .* mask_nH_lower .* mask_nH_upper # BitVector
```

pass to any function to consider only masked data in calculations:
total mass, projection, getvar, center-of-mass, your own functions, etc.

projection

function

hydro:

```
p = projection(gas, :sd, :Msun_pc2);
```

```
[Mera]: 2023-04-15T13:47:57.596

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]

Selected var(s)=(:sd,)
Weighting      = :mass

Effective resolution: 1024^2
Map size: 1024 x 1024
Pixel size: 46.875 [pc]
Simulation min.: 46.875 [pc]

Progress: 100%|████████████████████████████████████████| Time: 0:00:04
```

particles:

```
pp = projection(part, :sd, :Msun_pc2);
```

```
[Mera]: 2023-04-15T13:49:21.198

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]

Effective resolution: 1024^2
Pixel size: 46.875 [pc]
Simulation min.: 46.875 [pc]
```

```
p = projection(gas, [:sd, :T], [:Msun_pc2, :K], pxsize=[100.,:pc], mask=mask_tot);
```

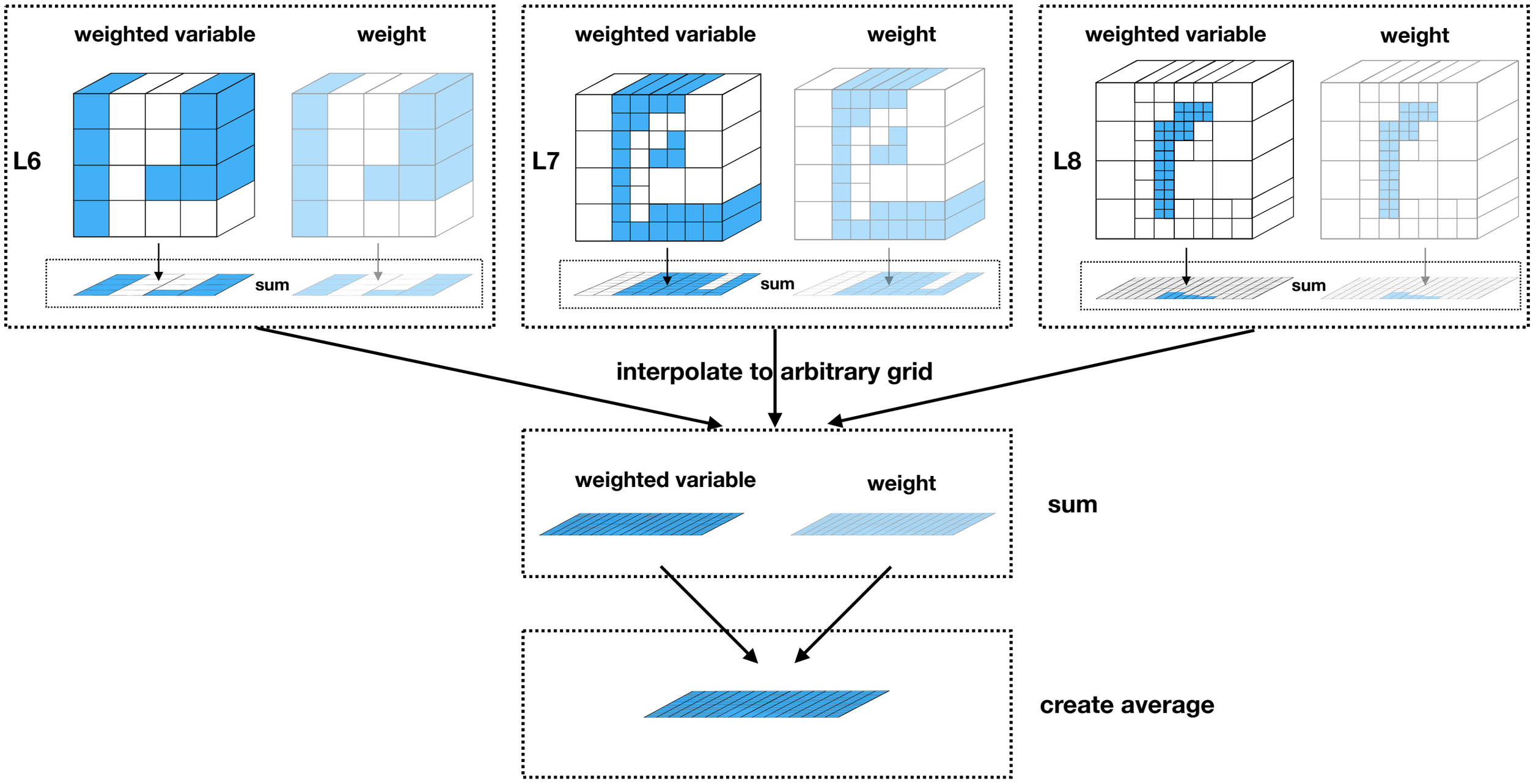
```
[Mera]: 2023-04-15T13:53:07.321

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 48.0 [kpc]

Selected var(s)=(:sd, :T)
Weighting      = :mass

Effective resolution: 481^2
Map size: 481 x 481
Pixel size: 99.792 [pc]
Simulation min.: 46.875 [pc]

:mask provided by function
```

request:

- multiple quantities
- physical size of grid
- pass a mask

MERA projection

illustration



projection

speed

```
gas = loaddata(250, fpath, :hydro);
```

```
[Mera]: 2023-04-17T17:21:55.182

Open Mera-file output_00250.jld2:

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]

Memory used for data table :7.403292601928115 GB
```

~7.4 GB

```
Ncells = length(gas.data)
```

```
90323997
```

~9x10^7 cells

one quantity

```
p = projection(gas, :sd, :Msun_pc2);
```

```
[Mera]: 2023-04-17T17:25:21.505

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]

Selected var(s)=(:sd,)
Weighting      = :mass

Effective resolution: 16384^2
Map size: 16384 x 16384
Pixel size: 12.207 [pc]
Simulation min.: 12.207 [pc]

Progress: 100%|                                        | Time: 0:02:00
```

```
p = projection(gas, :sd, :Msun_pc2, pxsize=[100.,:pc]);
```

```
[Mera]: 2023-04-17T17:27:40.789

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]

Selected var(s)=(:sd,)
Weighting      = :mass

Effective resolution: 2001^2
Map size: 2001 x 2001
Pixel size: 99.95 [pc]
Simulation min.: 12.207 [pc]

Progress: 100%|                                        | Time: 0:00:23
```

note:

- faster, since integrated mass (surface density) is calculated only once and used for the mass-weighted average for all projected variables

- particle projections are generally very fast if no AMR structure is considered

several quantities

```
p = projection(gas, [:sd, :T, :vr_cylinder], [:Msun_pc2, :K, :km_s], center=[:bc]);
```

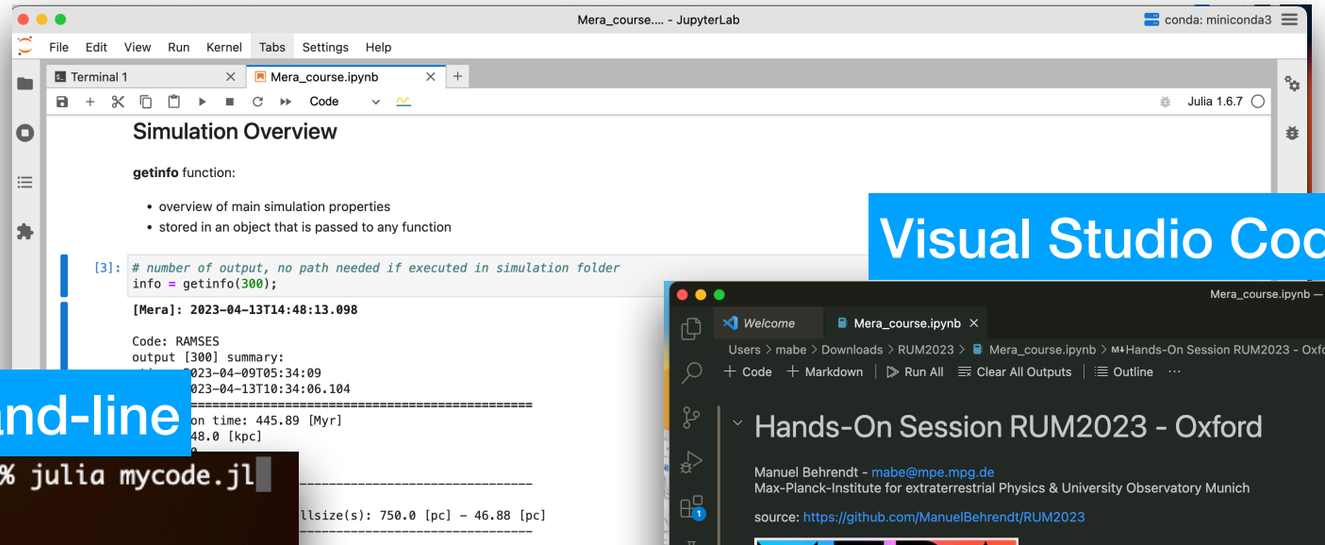
```
[Mera]: 2023-04-17T17:29:25.990

center: [0.5, 0.5, 0.5] ==> [100.0 [kpc] :: 100.0 [kpc] :: 100.0 [kpc]]

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]

Selected var(s)=(:sd, :T, :vr_cylinder)
Weighting      = :mass

Effective resolution: 16384^2
Map size: 16384 x 16384
Pixel size: 12.207 [pc]
Simulation min.: 12.207 [pc]

Progress: 100%|                                        | Time: 0:04:04
```

```
p = projection(gas, [:sd, :T, :vr_cylinder], [:Msun_pc2, :K, :km_s], center=[:bc],
               pxsize=[100.,:pc]);
```

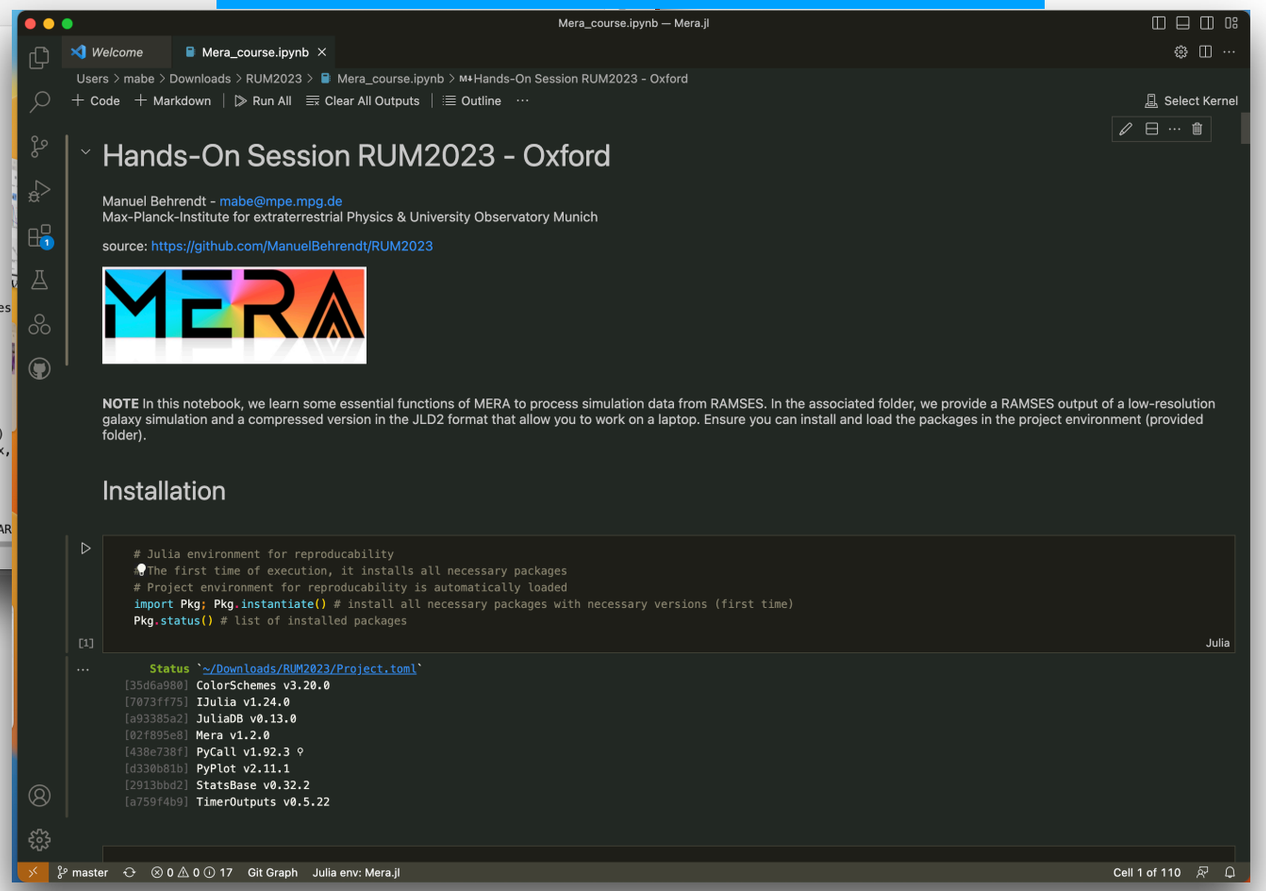
```
[Mera]: 2023-04-17T17:34:49.324

center: [0.5, 0.5, 0.5] ==> [100.0 [kpc] :: 100.0 [kpc] :: 100.0 [kpc]]

domain:
xmin::xmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
ymin::ymax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]
zmin::zmax: 0.0 :: 1.0          ==> 0.0 [kpc] :: 200.0 [kpc]

Selected var(s)=(:sd, :T, :vr_cylinder)
Weighting      = :mass

Effective resolution: 2001^2
Map size: 2001 x 2001
Pixel size: 99.95 [pc]
Simulation min.: 12.207 [pc]

Progress: 100%|                                        | Time: 0:00:55
```

MERA interactive/script

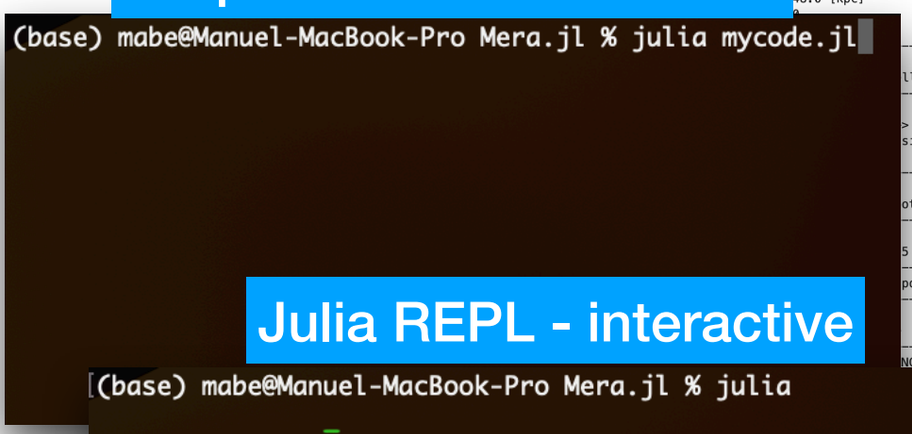
Jupyter Notebook - interactive



Visual Studio Code - interactive



script from command-line



Julia REPL - interactive

documentation

interactive help

```
?gethydro
```

```
search: gethydro
```

Read the leaf-cells of the hydro-data:

- select variables
- limit to a maximum level
- limit to a spatial range
- set a minimum density or sound speed
- check for negative values in density and thermal pressu
- print the name of each data-file before reading it
- toggle verbose mode
- toggle progress bar
- pass a struct with arguments (myargs)

```
gethydro(   dataobject::InfoType;
            lmax::Real=dataobject.levelmax,
            vars::Array{Symbol,1}=[:all],
            xrange::Array{<:Any,1}=[missing, 
            yrange::Array{<:Any,1}=[missing, 
            zrange::Array{<:Any,1}=[missing, 
            center::Array{<:Any,1}=[0., 0., 0
            range_unit::Symbol=:standard,
            smallr::Real=0.,
            smallc::Real=0.,
            check_negvalues::Bool=false,
            print_filenames::Bool=false,
            verbose::Bool=true,
            show_progress::Bool=true,
            myargs::ArgumentsType=ArgumentsTy
```

Returns an object of type HydroDataType, contain ScaleType and summary of the InfoType

```
return HydroDataType()
```

arguments explained

```
# get an overview of the returned fields:
# e.g.:
julia> info = getinfo(100)
julia> gas  = gethydro(info)
julia> viewfields(gas)
#or:
julia> fieldnames(gas)
```

Arguments

Required:

- **`dataobject`** : needs to be of type: "InfoType", c

Predefined/Optional Keywords:

- **`lmax`** : the maximum level to be read from the da
- **`var(s)`** : the selected hydro variables in arbitrar
- **`xrange`** : the range between [xmin, xmax] in unit xmin=xmax=0. is converted to maximum possible
- **`yrange`** : the range between [ymin, ymax] in unit ymin=ymax=0. is converted to maximum possible
- **`zrange`** : the range between [zmin, zmax] in unit zmin=zmax=0. is converted to maximum possible
- **`range_unit`** : the units of the given ranges: :sta defined length-scales viewfields(info.scale)
- **`center`** : in units given by argument `range_un` :bc, :bc], etc..
- **`smallr`** : set lower limit for density; zero means
- **`smallc`** : set lower limit for thermal pressure; ze
- **`check_negvalues`** : check loaded data of "rho"
- **`print_filenames`** : print on screen the current
- **`verbose`** : print timestamp, selected vars and ra
- **`show_progress`** : print progress bar on screen
- **`myargs`** : pass a struct of ArgumentsType to pas center, range*unit, verbose, show*progress

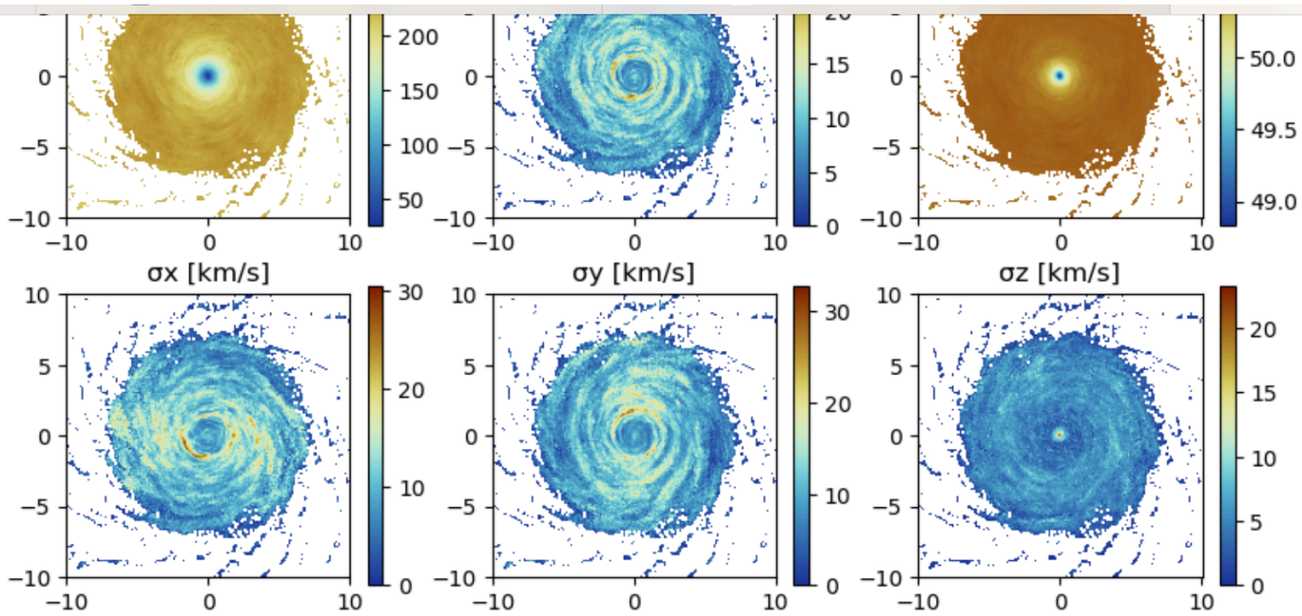
short examples

Examples

```
# read simulation information
julia> info = getinfo(420)

# Example 1:
# read hydro data of all variables, full-box, all levels
julia> gas = gethydro(info)

# Example 2:
# read hydro data of all variables up to level 8
# data range 20x20x4 kpc; ranges are given in kpc relative to the box (here: 48 kpc) center at 24 kpc
julia> gas = gethydro(    info,
                          lmax=8,
                          xrange=[-10.,10.],
                          yrange=[-10.,10.],
                          zrange=[-2.,2.],
                          center=[24., 24., 24.],
                          range_unit=:kpc )

# Example 3:
# give the center of the box by simply passing: center = [:bc] or center = [:boxcenter]
# this is equivalent to center=[24.,24.,24.] in Example 2
# the following combination is also possible: e.g. center=[:bc, 12., 34.], etc.
julia> gas = gethydro(    info,
                          lmax=8,
                          xrange=[-10.,10.],
                          yrange=[-10.,10.],
                          zrange=[-2.,2.],
                          center=[33., bc:, 10.],
                          range_unit=:kpc )

# Example 4:
# read hydro data of the variables density and the thermal pressure, full-box, all levels
julia> gas = gethydro( info, [:rho, :p] ) # use array for the variables

# Example 5:
# read hydro data of the single variable density, full-box, all levels
```

MERA documentation

online doc



Use quantities in cylindrical coordinates:

Face-on disc (z-direction)

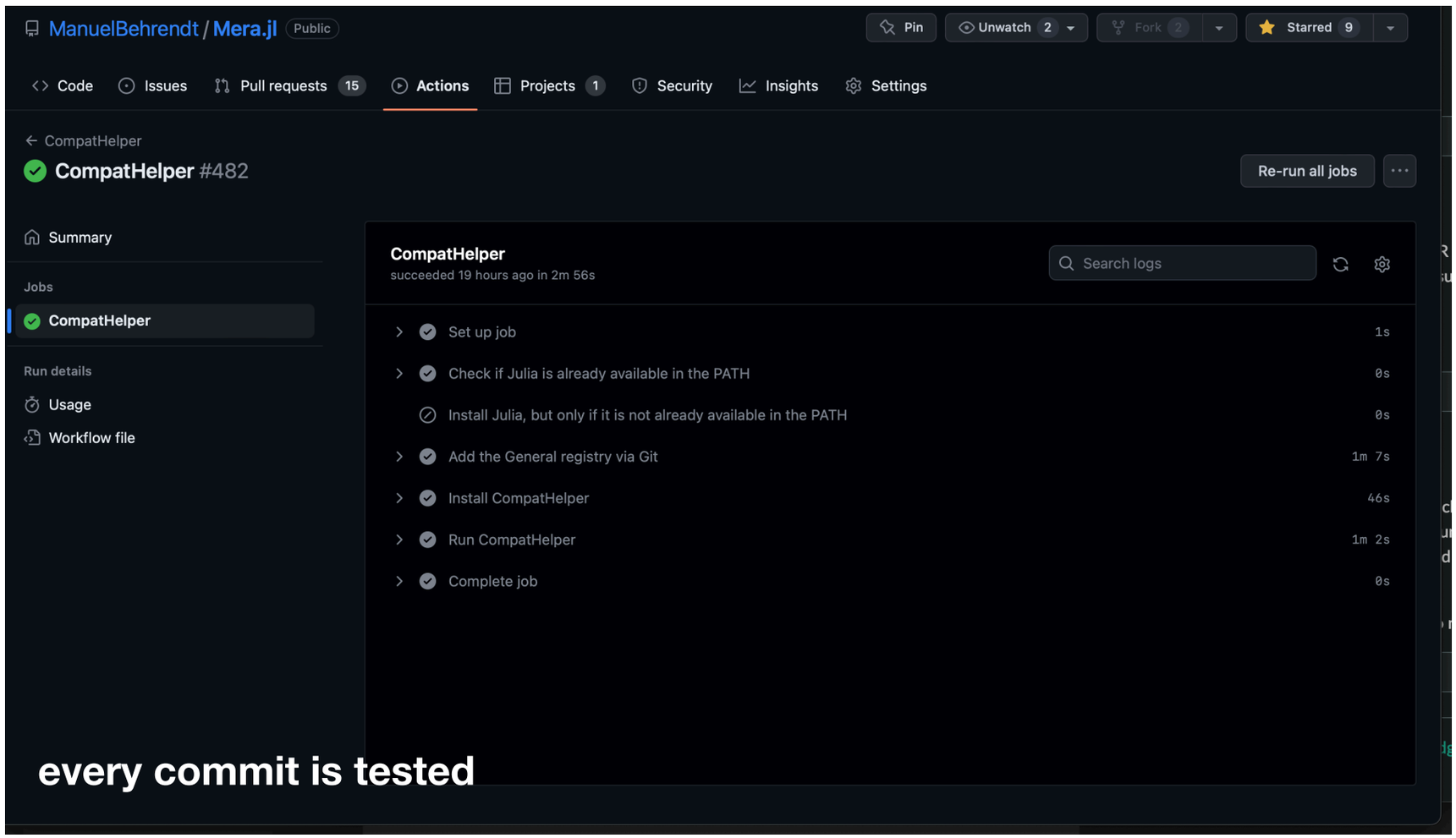
For the cylindrical or spherical components of a quantity, the center of the coordinate system is used (keywords: data*center = center default) and can be given with the keyword "data*center" and its units with "data*centerunit". Additionally, the quantities that are based on cartesian coordinates can be given.

```
proj_z = projection(particles, [:v, :σ, :σx, :σy, :ϕ, :r_cylinder, :vr_cylinder, :vϕ_cylinder, :
                    units=[:km_s,:km_s,:km_s, :km_s, :standard, :kpc, :km_s, :km_s, :km_s, :km_s
                    xrange=[-10.,10.], yrange=[-10.,10.], zrange=[-2.,2.],
                    center=[:boxcenter], range_unit=:kpc,
                    data_center=[24.,24.,24.], data_center_unit=:kpc);
```

```
[Mera]: 2023-04-10T13:50:46.517

center: [0.5, 0.5, 0.5] ==> [24.0 [kpc] :: 24.0 [kpc] :: 24.0 [kpc]]

domain:
```

compiling/unit tests

compiling



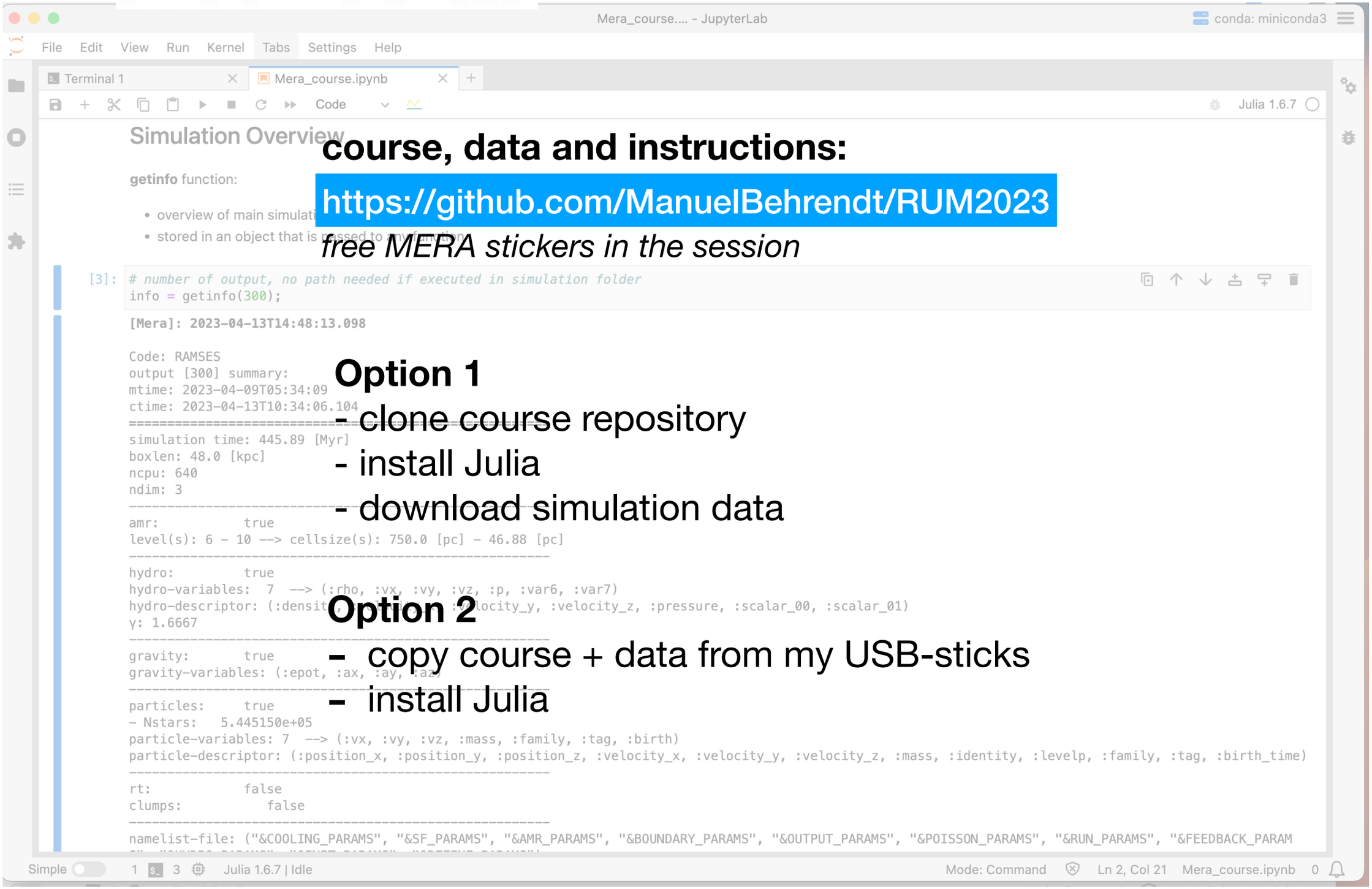
unit tests

- currently offline
- will be soon deployed on GitHub automatically with good code coverage

-> submit to JOSS journal

hands-on

session



course, data and instructions:

https://github.com/ManuelBehrendt/RUM2023

free MERA stickers in the session

Option 1

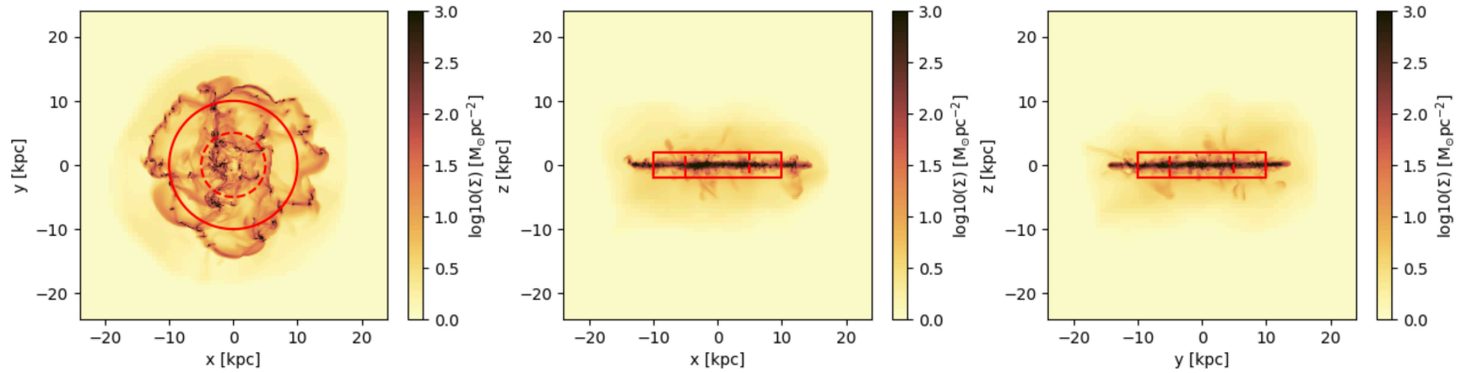
- clone course repository
- install Julia
- download simulation data

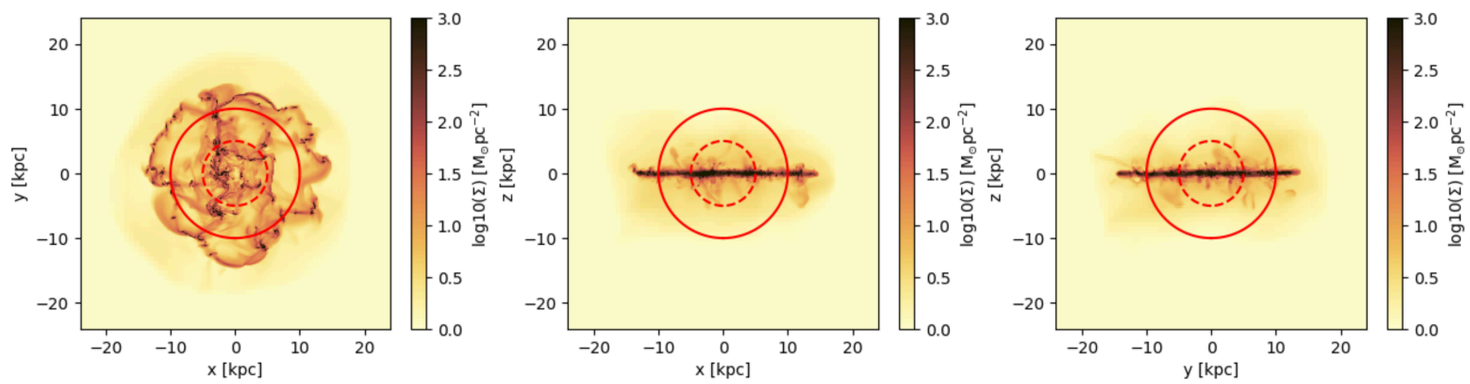
Option 2

- copy course + data from my USB-sticks
- install Julia

shellregions

applies to all datatypes (multiple dispatch)





compression

Benchmarks

The benchmark uses lzbench, from @inikep compiled with GCC v8.2.0 on Linux 64-bits (Ubuntu 4.18.0-17). The reference system uses a Core i7-9700K CPU @ 4.9GHz (w/ turbo boost). Benchmark evaluates the compression of reference Silesia Corpus in single-thread mode.

Compressor	Ratio	Compression	Decompression
memcpy	1.000	13700 MB/s	13700 MB/s
LZ4 default (v1.9.0)	**2.101**	**780 MB/s**	**4970 MB/s**
LZO 2.09	2.108	670 MB/s	860 MB/s
QuickLZ 1.5.0	2.238	575 MB/s	780 MB/s
Snappy 1.1.4	2.091	565 MB/s	1950 MB/s
Zstandard 1.4.0 -1	2.883	515 MB/s	1380 MB/s
LZF v3.6	2.073	415 MB/s	910 MB/s
zlib deflate 1.2.11 -1	2.730	100 MB/s	415 MB/s
LZ4 HC -9 (v1.9.0)	**2.721**	41 MB/s	**4900 MB/s**
zlib deflate 1.2.11 -6	3.099	36 MB/s	445 MB/s

https://github.com/lz4/lz4

LZ4 is lossless compression algorithm, providing compression speed > 500 MB/s per core (>0.15 Byte cycle). It features an extremely fast decoder, with speed in multiple GB/s per core (~1 Byte/cycle)

Benchmarks

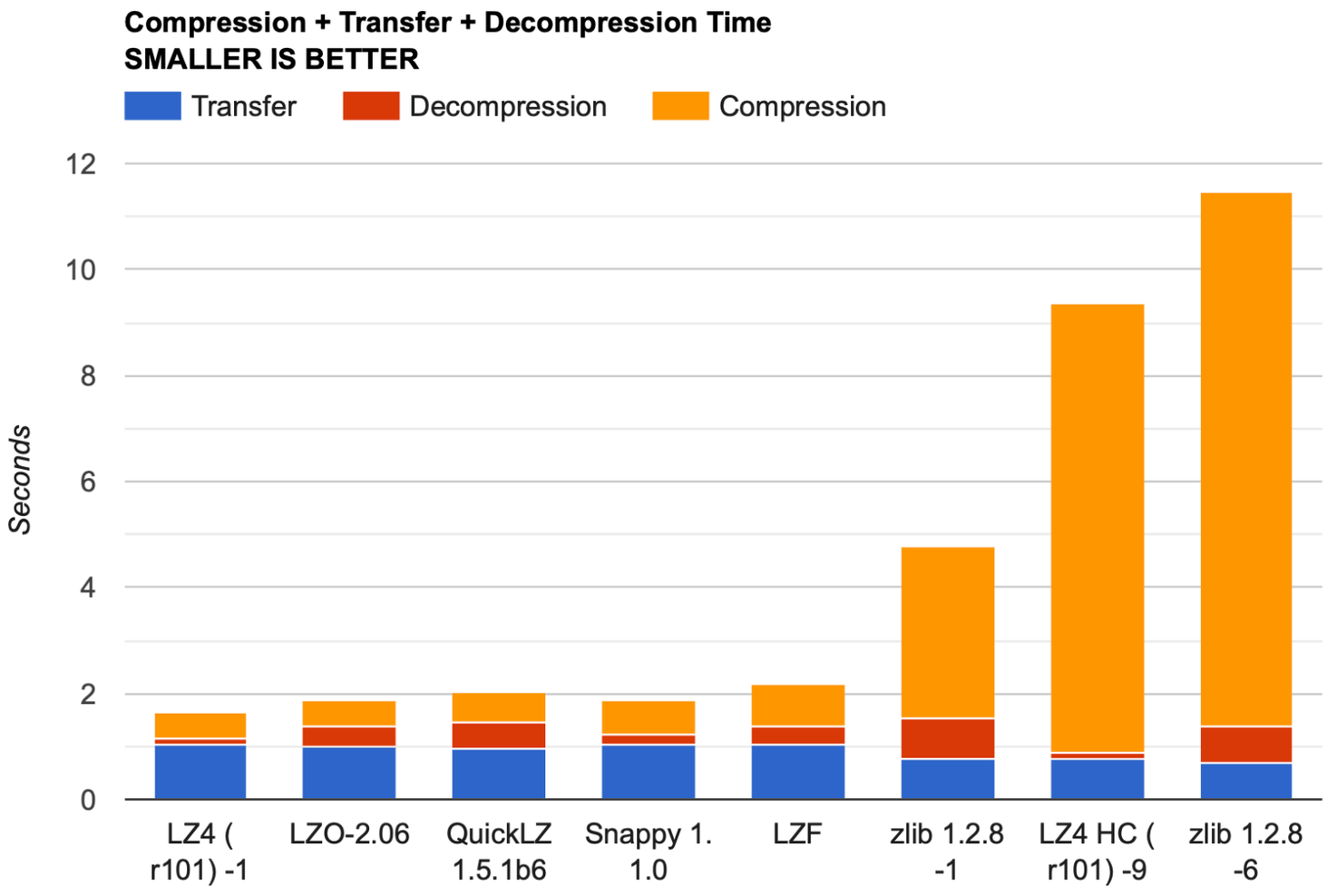
The benchmark uses the Open-Source Benchmark program by m^2 (v0.14.2) compiled with GCC v4.6.1 on Linux Ubuntu 64-bits v11.10, The reference system uses a Core i5-3340M @2.7GHz. Benchmark evaluates the compression of reference Silesia Corpus in single-thread mode.

TRANSFER + DECOMPRESSION TIME @1000MBIT/S | COMPRESSION + TRANSFER + DECOMPRESSION TIME @1000MBIT/S

Compression + Transfer + Decompression Time @1000Mbit/s (without I/O overwrapping, multithreading)

This benchmark simulates "dynamic content transfer" scenario such as filesystems, caches, network packets, IPC/RPCs, and databases.

In this case, both compression and decompression times are important. You can observe "Fast compression algorithms" are better than traditional algorithms such as DEFLATE (zlib).



https://lz4.github.io/lz4/

Benchmarks

The benchmark uses the Open-Source Benchmark program by m^2 (v0.14.2) compiled with GCC v4.6.1 on Linux Ubuntu 64-bits v11.10, The reference system uses a Core i5-3340M @2.7GHz. Benchmark evaluates the compression of reference Silesia Corpus in single-thread mode.

TRANSFER + DECOMPRESSION TIME @1000MBIT/S | COMPRESSION + TRANSFER + DECOMPRESSION TIME @1000MBIT/S

Transfer + Decompression Time @1000Mbit/s (without I/O overwrapping, multithreading)

This benchmark simulates simple "static content transfer" scenario such as OS Kernel compression or video game's static assets (text/images/tables/scripts/etc) which loading from Flash Memory / HDD / SSD.

In this case, compression time is completely ignored. Because only content developers compress the data at once and usually they don't care about its computational cost. But they always care enduser's experience a.k.a. "loading time" and bandwidth.

Please pay attention to "LZ4HC -9" which is quite faster than other methods.