The DISPATCH Code Framework

Davos, February 2017

Åke Nordlund, Jon Ramsey, Michael Küffmeier & Andrius Popovas

Niels Bohr Institute / STARPLAN, University of Copenhagen

OOP, Zeus solver, curvi-linear mesh, …

non-ideal MHD

ray-based radiative transfer

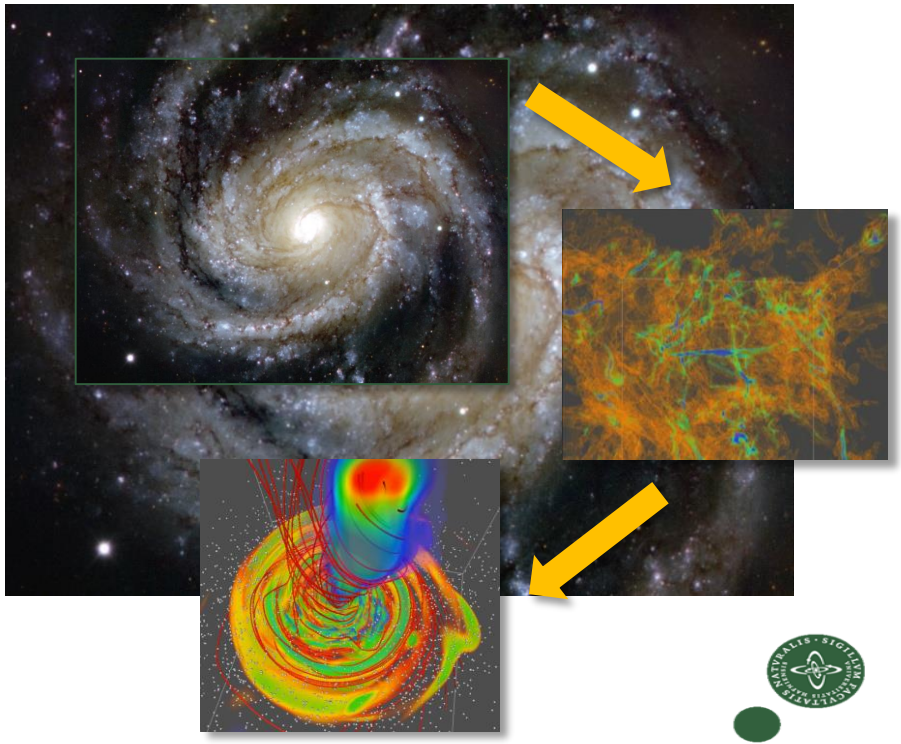
DISPATCH: Exa-scale features with immediate benefits

- Basic **concepts and ideas**
  - Moving patches
  - Local time steps
  - Task based scheduling
- **Object hierarchy**
  - Patches, solvers, experiments
  - Scenes, components
- **Code components**
  - OpenMP tasks, local neighborhood MPI, load balancing
- **Application examples**
  - Supersonic turbulence, solar active regions, chondrule accretion

Archetype Problem: Inspiring the right mind set & choices

Imagine: a ***full galaxy*** simulation, down to individual star and planet formation

- ***Exascale ⇒ simulating huge systems !***
  - Pointless to apply millions of cores to a single, relatively simple system
  - Unavoidably: ***many semi-autonomous hierarchical regions of space***
    - *GMCs, MCs, accretion disks, planets, …*
- This point of view has ***decisive implications*** for the choice of mechanisms in the code
  - A distant GMC has very few properties of interest to a local GMC: Essentially only its **mass**, **position** and **light output**
  - For the same reason: Solving the Poisson equation does ***NOT*** imply a need for **exact** synchronism across huge scale ranges

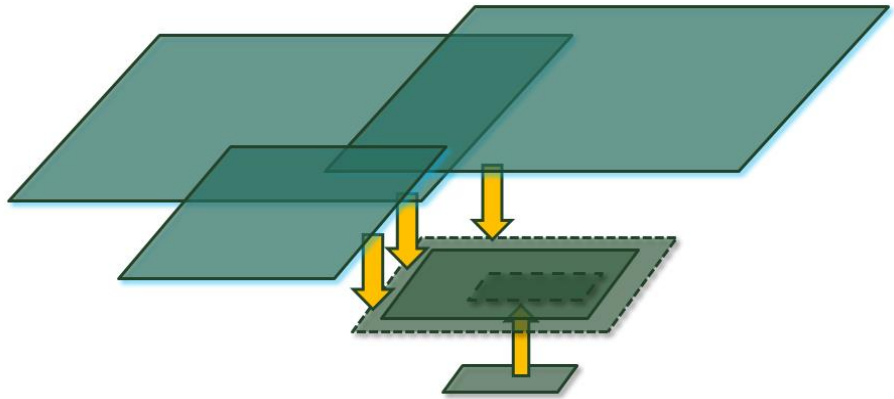


The DISPATCH name

The keys to the scaling properties are encoded in the name:

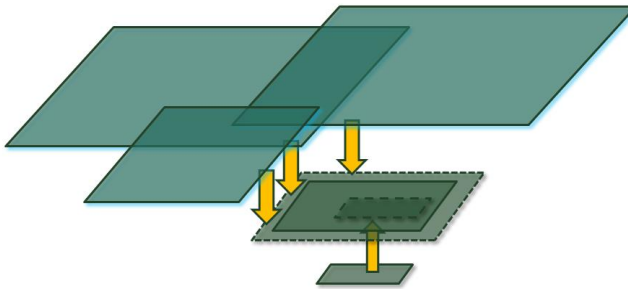
- Using a **DISPATCH**er to spawn **semi-independent OpenMP processes**, which are ***not*** forced to be locked together in time, ***nor*** are they forced to exist in grid-locked arrangement in space!
- The result is a number of **DIS**connected **PATCH**es, each of which has its own "task" definition, which is responsible for its evolution. **Patches interact via**

1. **Downloading** of guard zone data from neighboring patches
2. **Uploading** of interior data, from patches that have higher quality data
3. **Retaining** several **time slices** for interpolation in time

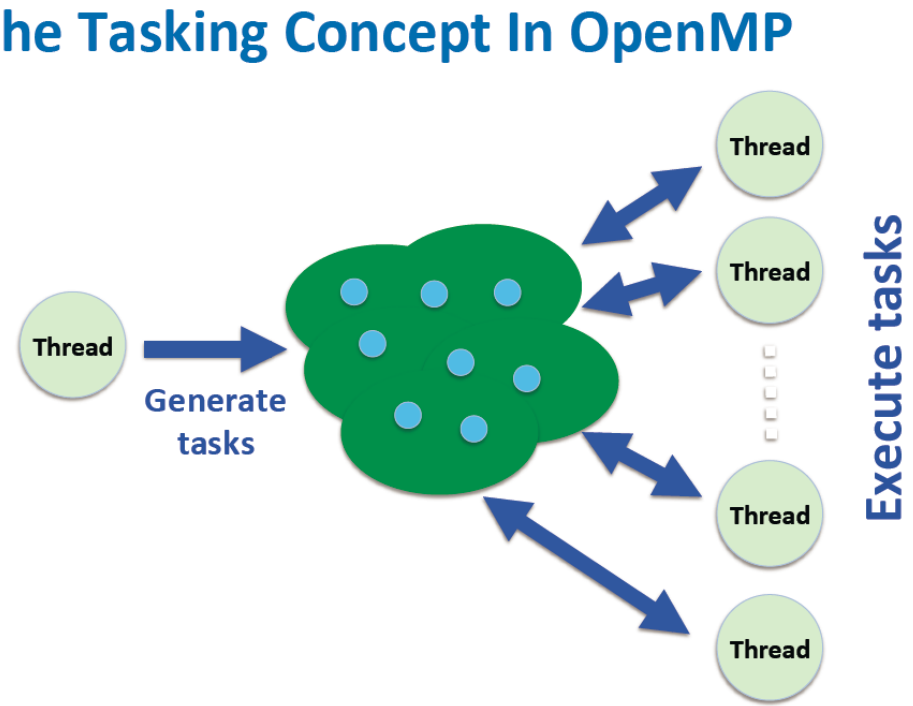


DISPATCH breaks with traditions, to achieve ~unlimited scaling:

- Allow **asynchronous evolution** of sub-domains (patches)
- Allow **moving patches** – small Cartesian meshes with bulk motion
- Allow **local time steps**; determined independently for each patch
- Use **task-based scheduling**, via OpenMP inside nodes
- Use **neighborhood-limited MPI** between nodes
- Use **any preferred solver inside** patches, balancing speed against quality and guard zone requirements
  - Can include **Multiple-Domain-Multiple-Physics**
    - e.g. PIC codes for kinetic simulations inside MHD
    - dust+gas dynamics
    - …



Task based OpenMP on each node



Object task hierarchy, task_lists, components, scenes

Task hierarchy: basic "task"

```
type, public:: task_t
  integer:: id, status, …
  real(8), pointer:: position(:,:)
  real(8), pointer:: t(:), dt(:)
  integer, pointer:: iit(:)
  ...
contains
  procedure:: init
  procedure:: update
  ...
end type
```

Basic "list": triply linked list

```
type, public:: link_t
 type(link_t), pointer:: next, prev
 type(link_t), pointer:: next_time
 type(link_t), pointer:: nbor
 class(task_t), pointer:: task
 ...
end type

type, public:: list_t
  type(link_t), pointer:: head, tail
  type(link_t), pointer:: queue
  ...
contains
  procedure:: append
  procedure:: remove
  procedure:: add_nbors
  procedure:: queue_by_time
  ...
end type
```


Object task hierarchy, task_lists, components, scenes

Task hierarchy: basic “task”

“patch” adds mesh, memory, …

“solver” adds specific method + params

```
type, public, extends(patch_t):: mhd_t
  real:: courant_number
  type(timestep_t):: timestep
  ...
contains
  procedure:: init
  procedure:: update
  ...
end type
```

“task_list”: heart of dispatcher

```
type, public, extends(list_t):: task_list
  ...
contains
  procedure:: execute
  procedure:: update
  ...
end type

SUBROUTINE execute (self)
  class(task_list_t):: self
  ...
  !$omp parallel
  do while (self%n > 0)
    call self%update
  end do
  !$omp end parallel
END SUBROUTINE
```

Object task hierarchy, task_lists, components, scenes

Task hierarchy: basic “task”

“patch” adds mesh, memory, …

“solver” adds specific method + params

“experiment” adds specific IC, BC,

```
type, public, extends(patch_t):: experiment_t
  ...
contains
  procedure:: init
  procedure:: update
end type

SUBROUTINE init (self)
  USE initial_mod
  call self%patch_t%init
  call initial_t%init
  ...
END SUBROUTINE init
```

“cartesian”: builds task list

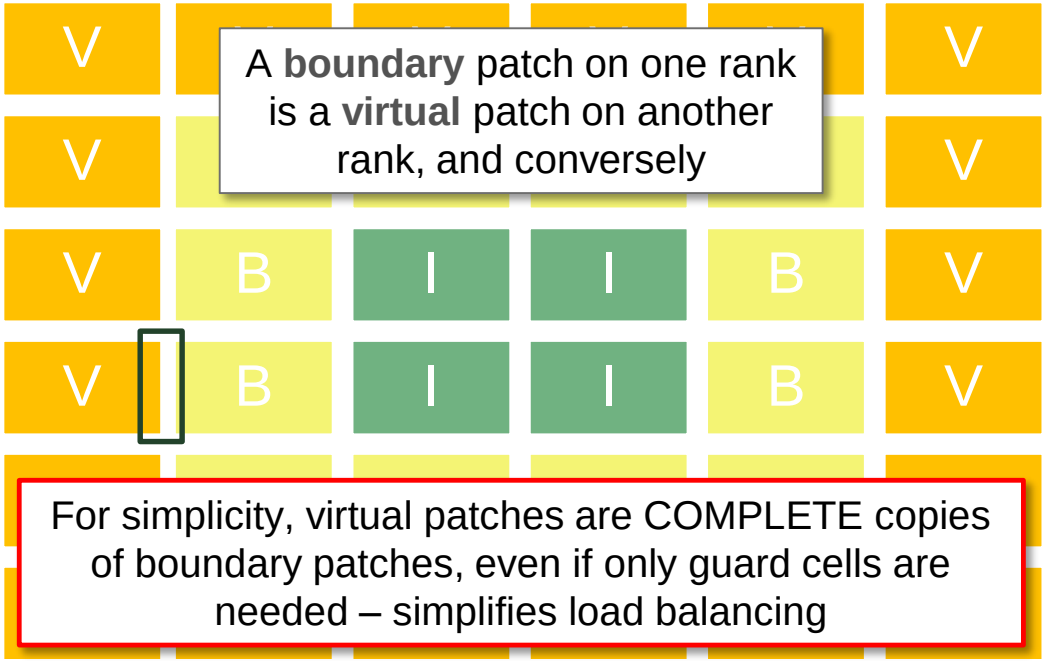
“scenes, components”:

```
USE galaxy
USE gmc
USE accretion_disk
USE star
USE planet

SUBROUTINE init (self)
  class(scene_t):: self
  ...
  call galaxy%init
  self%task_list => galaxy%task_list
  do i=1,galaxy%n_gmc
    call gmc%init
    self%task_list%append_list%gmc%task_list
  end do
  ...
END SUBROUTINE
```

```
  end do
  end do
  end do
END SUBROUTINE
```

Interior, Boundary, and Virtual patches



Sender & receiver side MPI

Send buffers to virtual patch owners:

```
call copy_to_buffer (patch, mesg%buffer)
...
nbor => link%nbor
do while (associated(nbor))
   rank = nbor%task%rank
   if (rank /= done) then
      call MPI_ISEND (mesg%buffer, nwords, MPI_REAL, rank, …)
      call add_mesg (send_list, mesg)
      done => rank
   end if
   nbor => nbor%next
end do
```

Sender & receiver side MPI

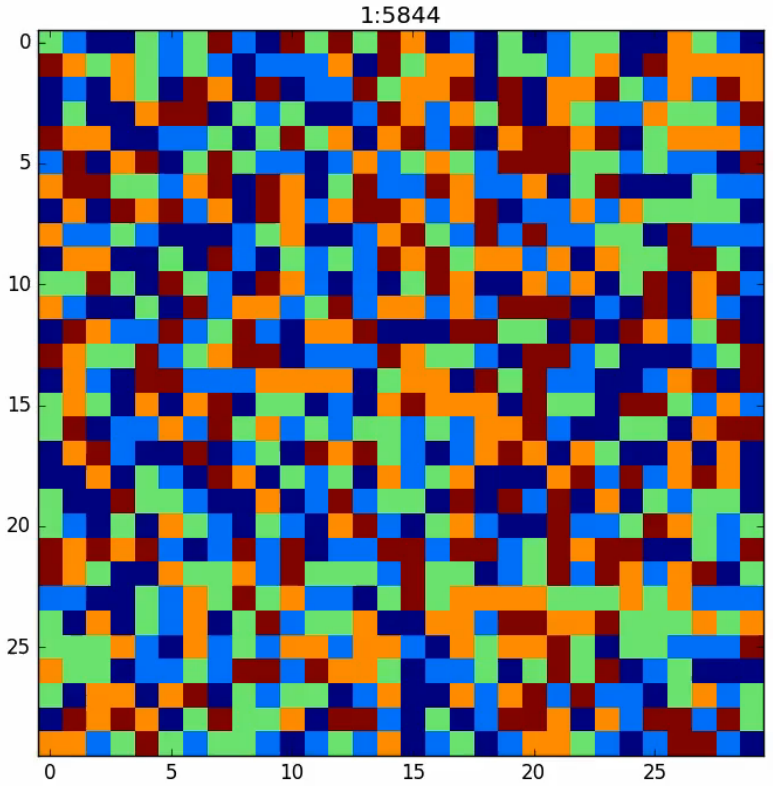
Gather incoming messages into a recv_list:

```
call MPI_IMPROBE (MPI_ANY_SOURCE, MPI_ANY_TAG, flag, msg, …)
do while (flag)
   call MPI_GET_COUNT (stat, MPI_REAL, nbuf, ierr)
   allocate (mesg)
   ...
   call MPI_IMRECV (mesg%buffer, nbuf, MPI_REAL, msg, req, …)
   ...
   call add_mesg (recv_list, mesg)
   call MPI_IMPROBE (MPI_ANY_SOURCE, MPI_ANY_TAG, flag, msg, …)
end do
```

Load & communication balance

The Python animation shows the result of ***simultaneously*** running:

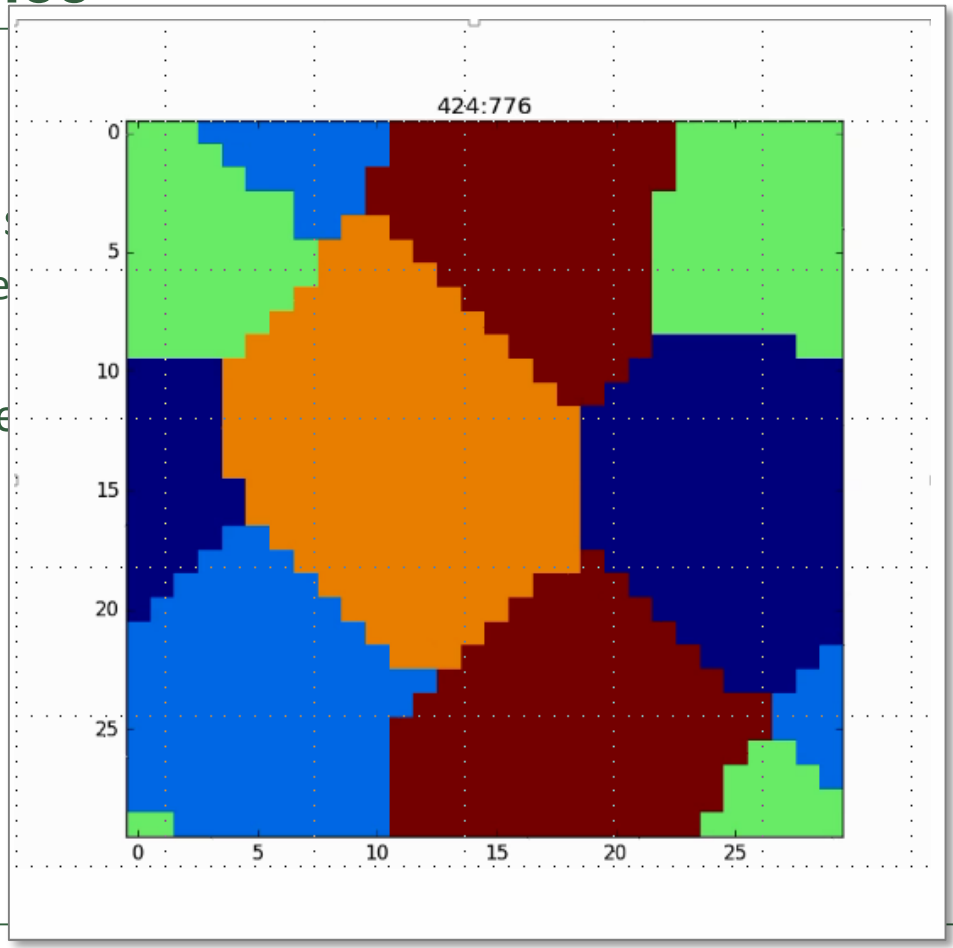
- 1st procedure reducing communications
  - Interchanging "boundary" and "virtual" roles
  - ***temporarily*** introduces a small imbalance
- 2nd procedure evens out the load, while avoiding to increase communications
  - Operates independent of the 1st procedure
  - Uses only local information
  - Swap "boundary" and "virtual" roles
  - Both procedures are ***entirely local***, requiring ***no global information***



Load & communication balance

The Python animation shows the result of ***simultaneously*** running:

- 1st procedure reducing communication
  - Interchanging “boundary” and “virtual” role
  - ***temporarily*** introduces a small imbalance
- 2nd procedure evens out the load, while avoiding to increase communications
  - Operates independent of the 1st procedure
  - Uses only local information
  - Swap “boundary” and “virtual” roles
  - Both procedures are ***entirely local***, requiring ***no global information***



Recent developments

- Ray-tracing radiative energy transfer is working
  - Several task structures are being explored
  - Total cost when along axes & diags ~10 ns/pt/angle/opacity-bin
- Prototype coupling to external code runs
  - One file on DISPATCH side, one file on external code side, one interface file
  - Aim: to be used e.g. with BIFROST (solar chromosphere + corona) and PIC code
  - Time step advantage: factor 30-100 (!)
- OpenMP optimized:
  - Removing all critical regions using ‘dispatcher’

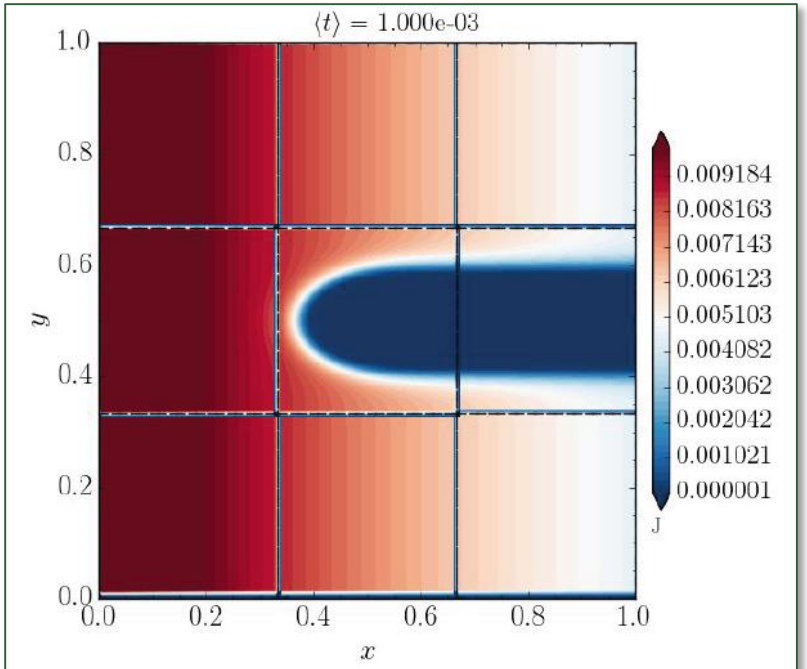


Figure 3: Results from the DISPATCH ‘bundle chain’ radiative transfer algorithm for the “shadow” benchmark.

Scaling tests on KNL and Hazel Hen / HLRS

- Using single-node KNL at UCPH:
  - flat mode: 0.98 μs/pt flat
  - 4-corner mode: 1.05 μs/pt flat

- MPI scaling tested at HLRS
  - Hazel Hen (CRAY XC40)
  - Haswell nodes (2x12 cores)
  - 256 nodes x 4 ranks / node

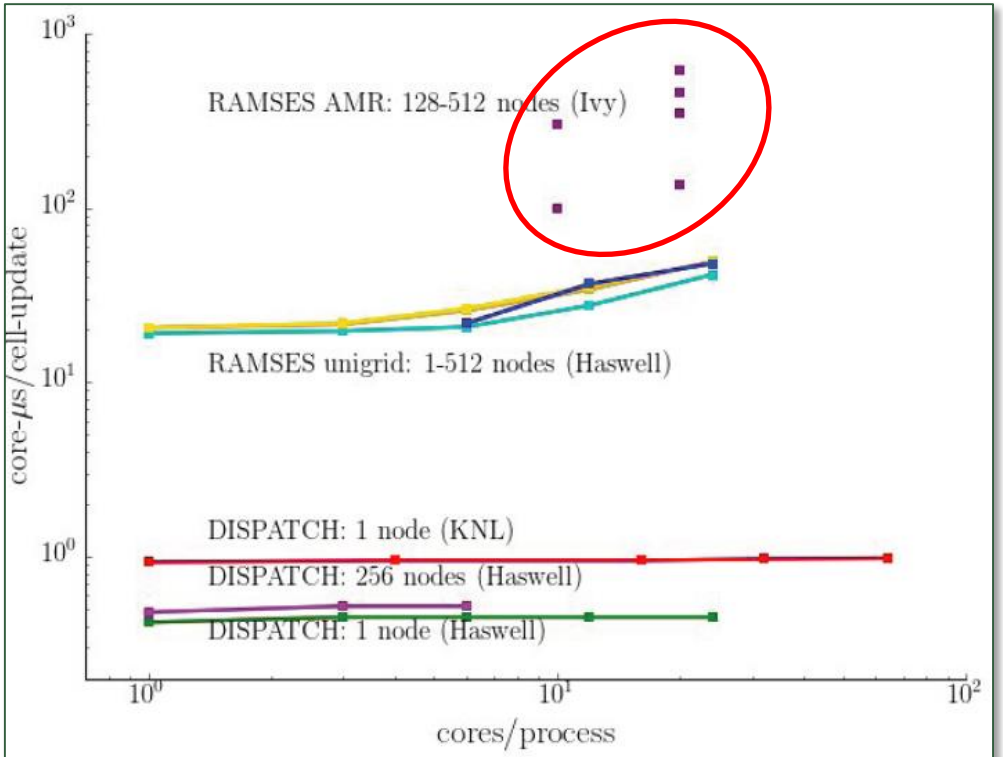


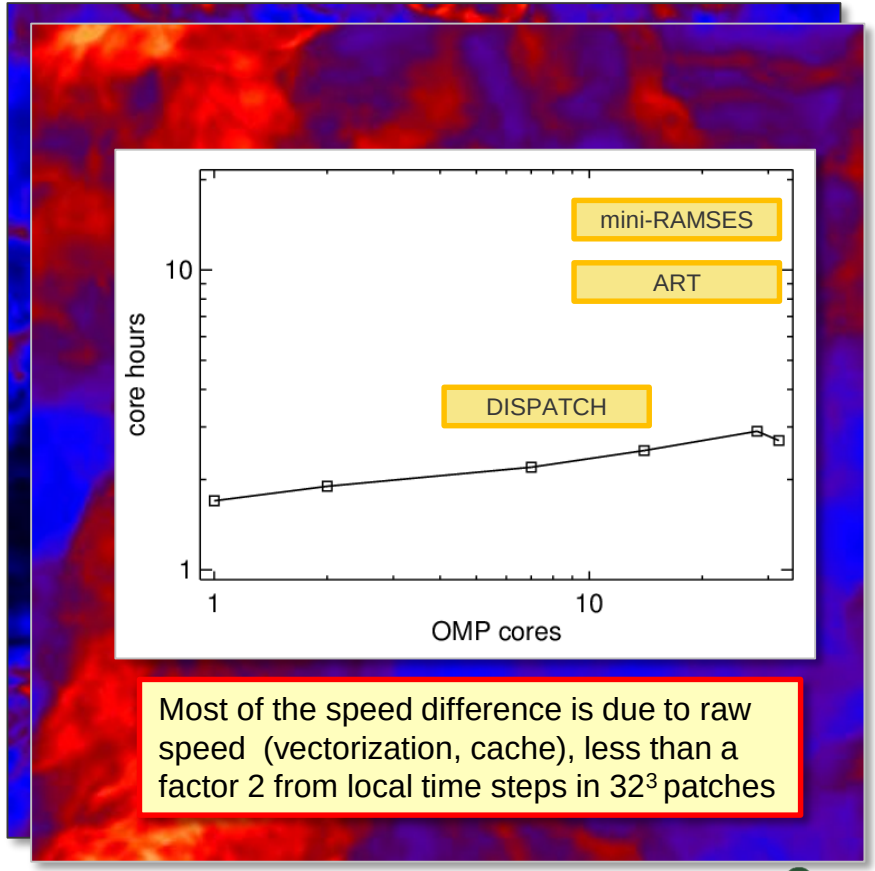
Figure 2: Strong and weak scaling for DISPATCH with the RAMSES HLLC solver, as compared to the version of RAMSES with HLLD used in the zoom-in simulatons of [5].

Example 1: Decaying supersonic HD-turbulence

This is one of the benchmarks from the KITP code comparison (Kritsuk et al, 2011)

- Pure HD – no magnetic field
- Initial RMS Mach number: ~10
- Resolution: 256^3
- To be run from t=0.02 to t=0.20

We did this, **all using RAMSES/HLLC**, in the Chicago/Copenhagen/Zuric collaboration, using (mini-)RAMSES, ART, and DISPATCH



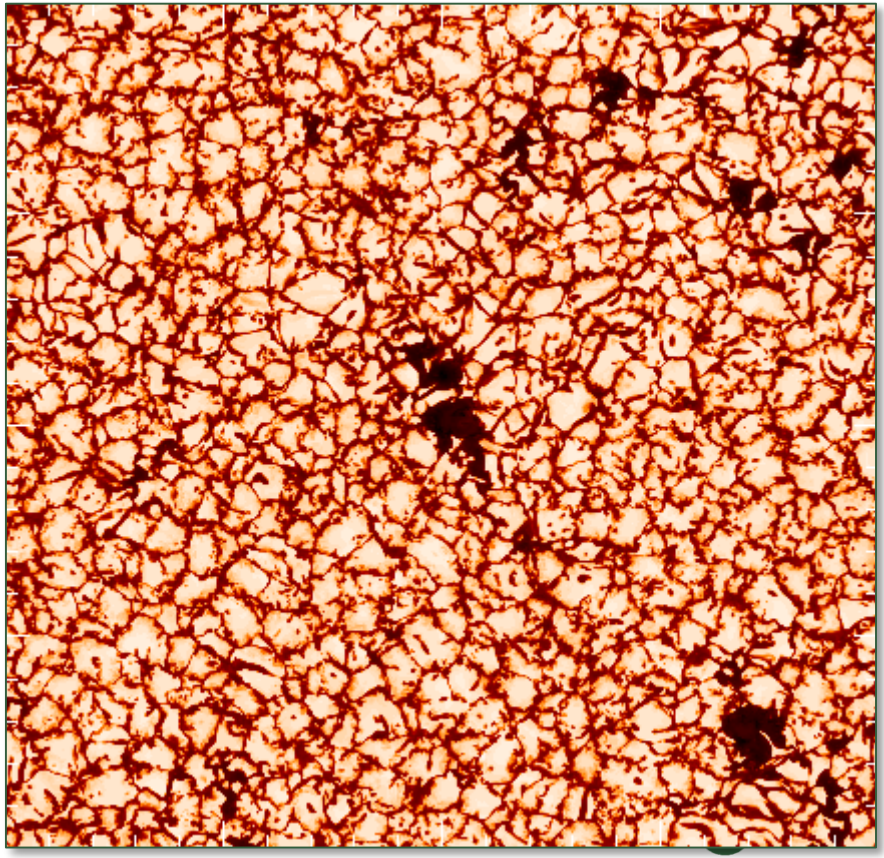
Example 2: Solar active regions

This is a **unigrid** example, with **2016 x 500 x 2016** grid point – to be expanded into the chromosphere and corona with up to 4032 x 1000 x 4032 grid points.

The **fast mode speed is extremely intermittent** in solar active regions, and hence the update cost is **concentrated to a few spots horizontally**, and there the update cost is **large only at the very surface**.

Hence a very large cost reduction can be achieved with local time steps ; in the cases shown here, the local **time step advantages is in the range 30-100**, depending on the size of sunspots.

[This has previously been handled to some extent by artificially & inconsistently limiting the fast mode speed]



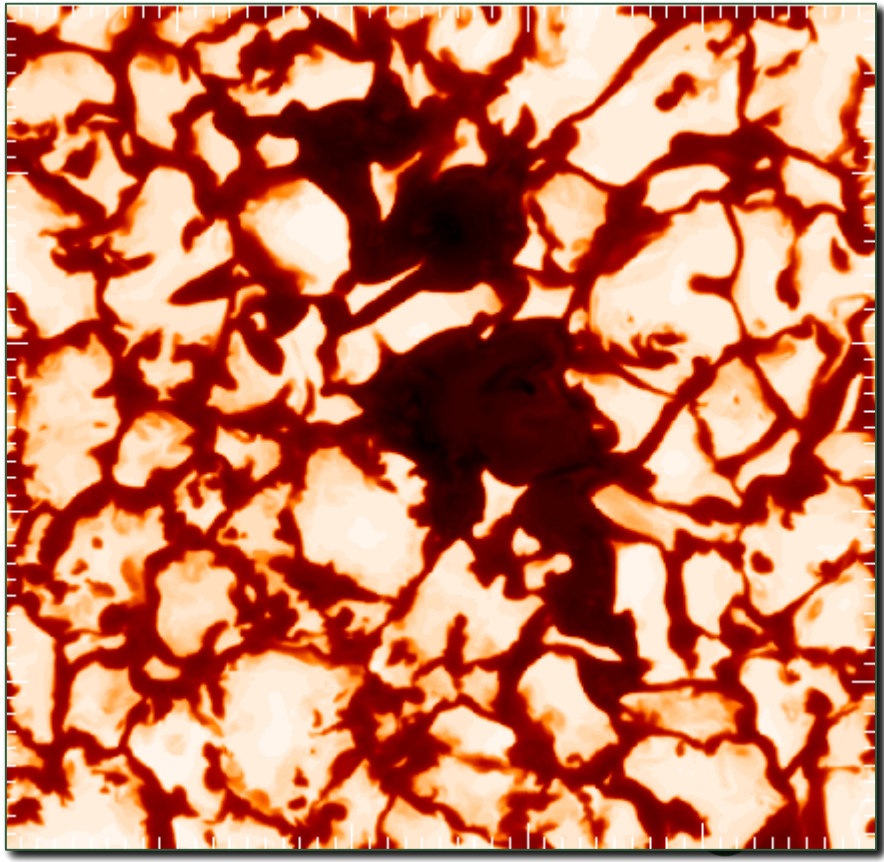
Example 2: Solar active regions

This is a **unigrid** example, with **2016 x 500 x 2016** grid point – to be expanded into the chromosphere and corona with up to 4032 x 1000 x 4032 grid points.

The **fast mode speed is extremely intermittent** in solar active regions, and hence the update cost is **concentrated to a few spots horizontally**, and there the update cost is **large only at the very surface**.

Hence a very large cost reduction can be achieved with local time steps ; in the cases shown here, the local **time step advantages is in the range 30-100**, depending on the size of sunspots.

[This has previously been handled to some extent by artificially & inconsistently limiting the fast mode speed]



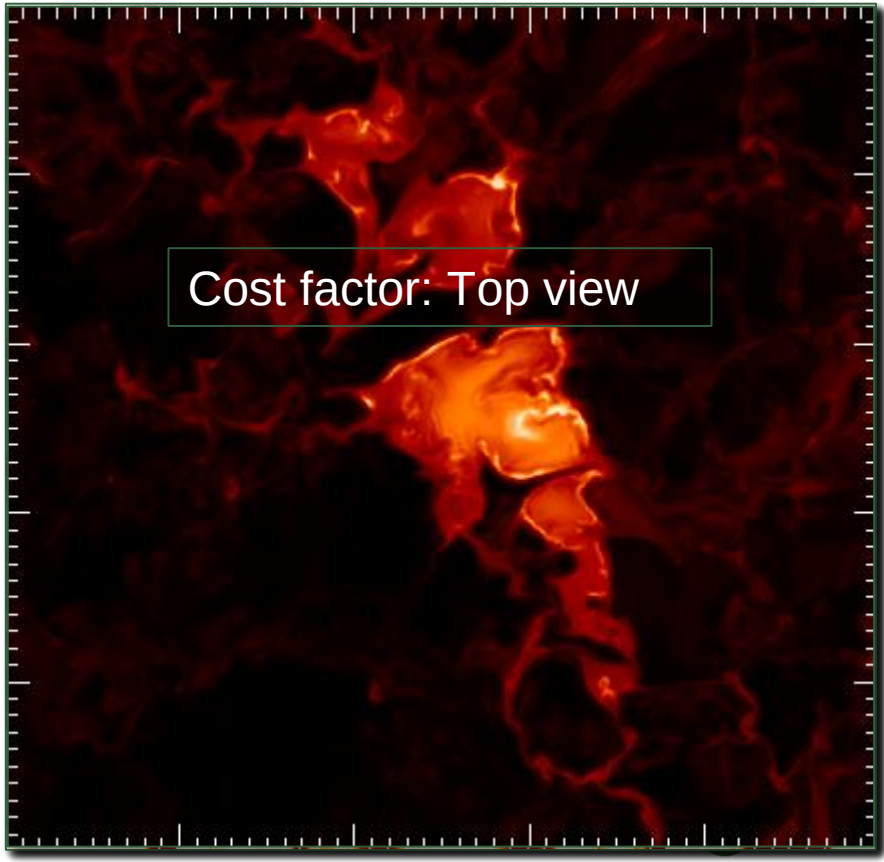
Example 2: Solar active regions

This is a **unigrid** example, with **2016 x 500 x 2016** grid point – to be expanded into the chromosphere and corona with up to 4032 x 1000 x 4032 grid points.

The **fast mode speed is extremely intermittent** in solar active regions, and hence the update cost is **concentrated to a few spots horizontally**, and there the update cost is **large only at the very surface**.

Hence a very large cost reduction can be achieved with local time steps ; in the cases shown here, the local **time step advantages is in the range 30-100**, depending on the size of sunspots.

[This has previously been handled to some extent by artificially & inconsistently limiting the fast mode speed]



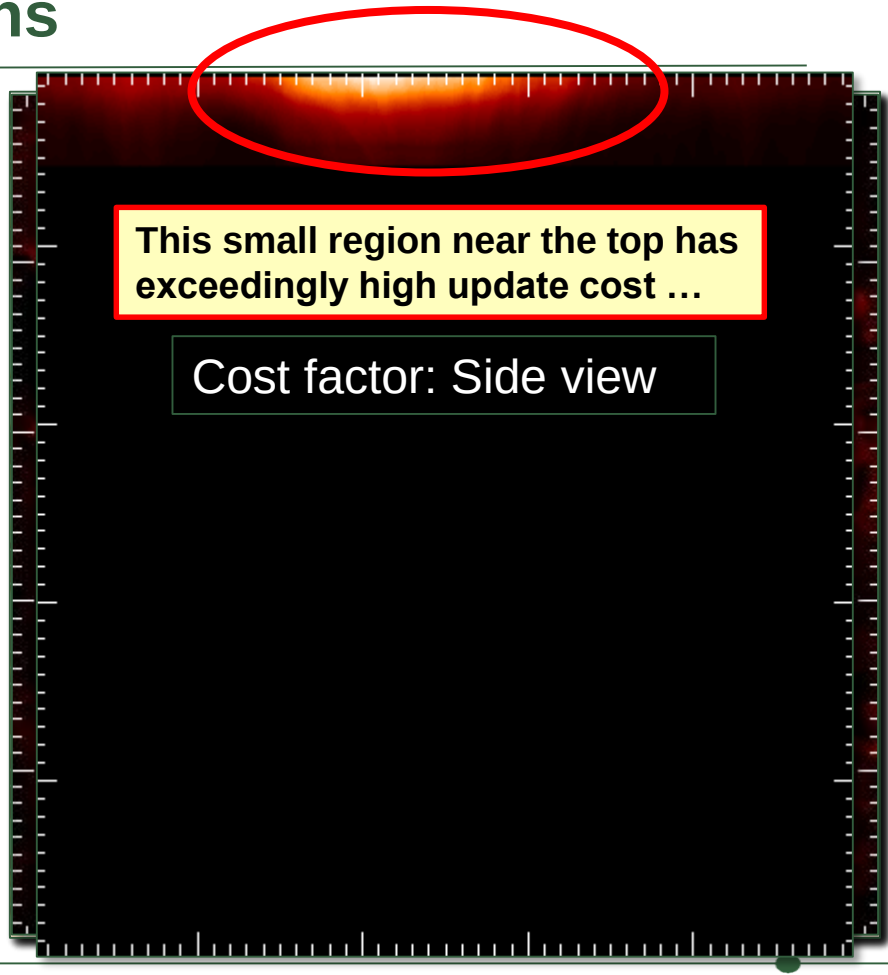
Example 2: Solar active regions

This is a **unigrid** example, with **2016 x 500 x 2016** grid point – to be expanded into the chromosphere and corona with up to 4032 x 1000 x 4032 grid points.

The **fast mode speed is extremely intermittent** in solar active regions, and hence the update cost is **concentrated to a few spots horizontally**, and there the update cost is **large only at the very surface**.

Hence a very large cost reduction can be achieved with local time steps ; in the cases shown here, the local **time step advantages is in the range 30-100**, depending on the size of sunspots.

[This has previously been handled to some extent by artificially & inconsistently limiting the fast mode speed]



Example 3: Accretion disks and chondrule accretion

For two separate reasons the **local time step advantage** is large also **in accretion disks**:

1. Low density in magnetically dominated regions (e.g. jets) give **large fast mode speeds**
2. **Kepler motions in disks** are supersonic when seen from fixed mesh; cf. **FARGO** method

For details on zoom-in simulations, cf. Küffmeier et al, arXiv 1611,10360

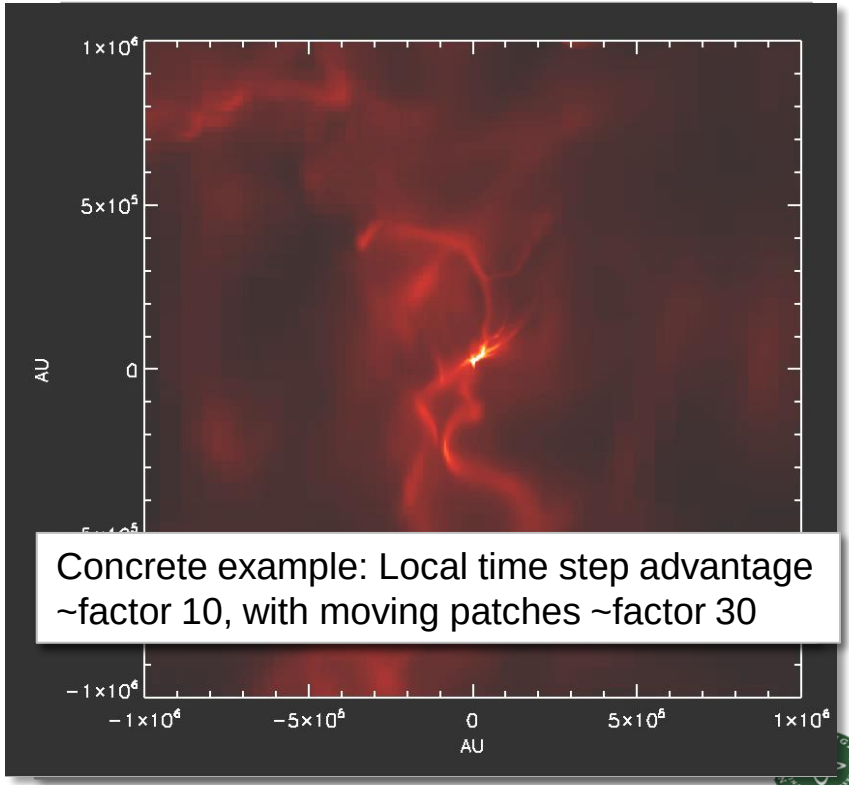
DISPATCH can read RAMSES snapshots, merging neighboring octs into larger patches

Example 3: Accretion disks and chondrule accretion

For two separate reasons the **local time step advantage** is large also **in accretion disks**:

1. Low density in magnetically dominated regions (e.g. jets) give **large fast mode speeds**
2. **Kepler motions in disks** are supersonic when seen from fixed mesh; cf. **FARGO** method

For details on zoom-in simulations, cf. Küffmeier et al, arXiv 1611,10360

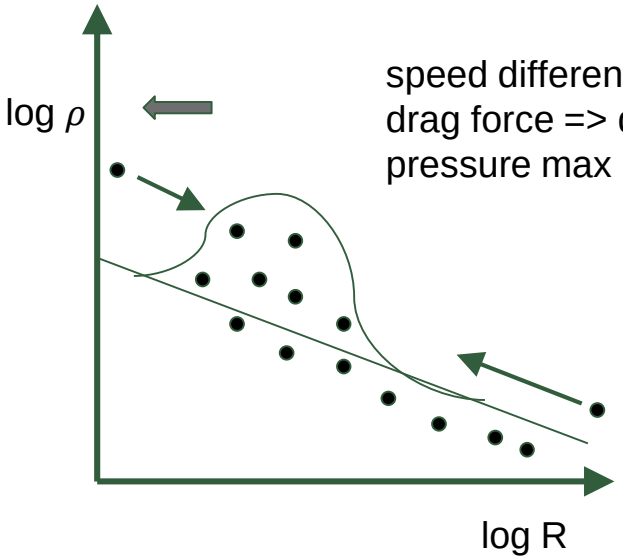


Concrete example: Local time step advantage ~factor 10, with moving patches ~factor 30

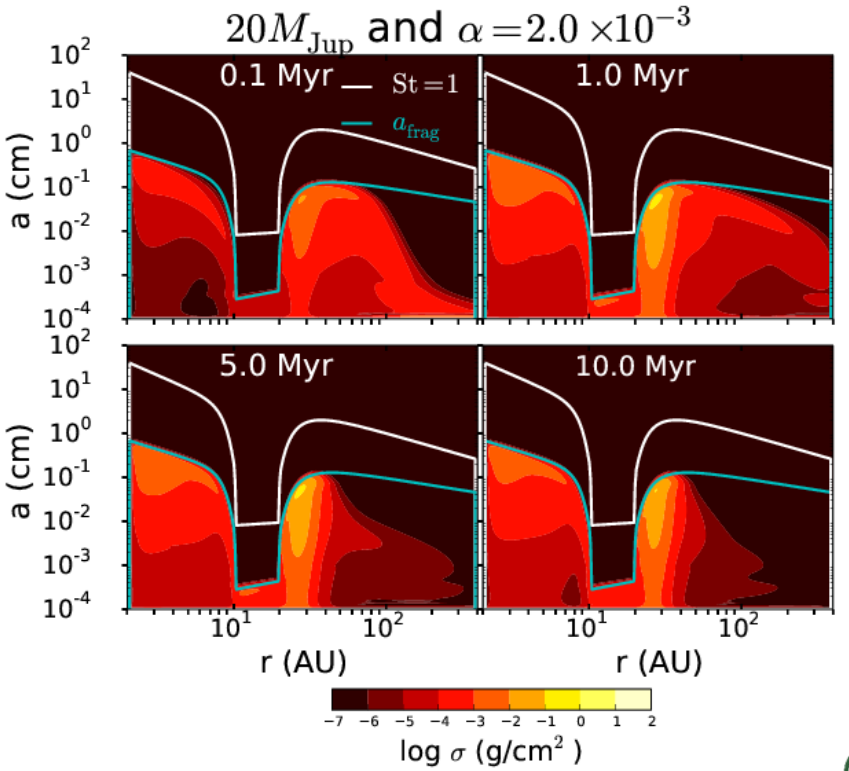
Gas + dust dynamics

Pressure traps

Inward pressure gradient => faster than Kepler
Outward pressure gradient => slower than Kepler



speed difference =>
drag force => drift toward
pressure max => trap

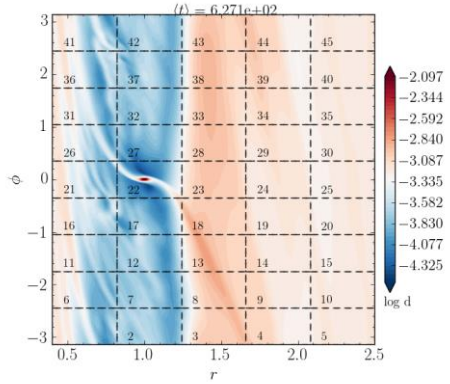


Pinilla, Birnstiel, & Walsh (2015)

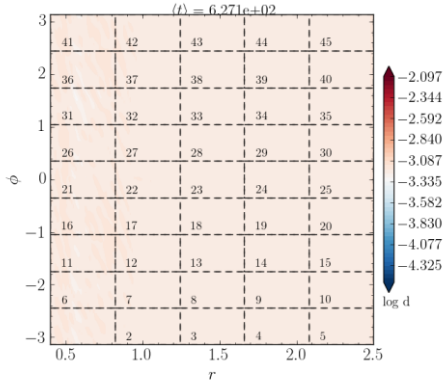
Modeling global disks and pebble accretion with DISPATCH

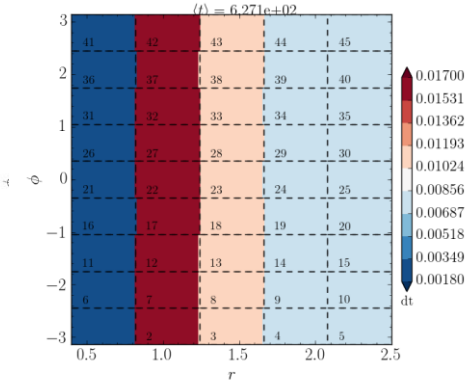
Set up analogous to the de Val-Borro et al. (2004) benchmark:

1 M_J



1 M_E





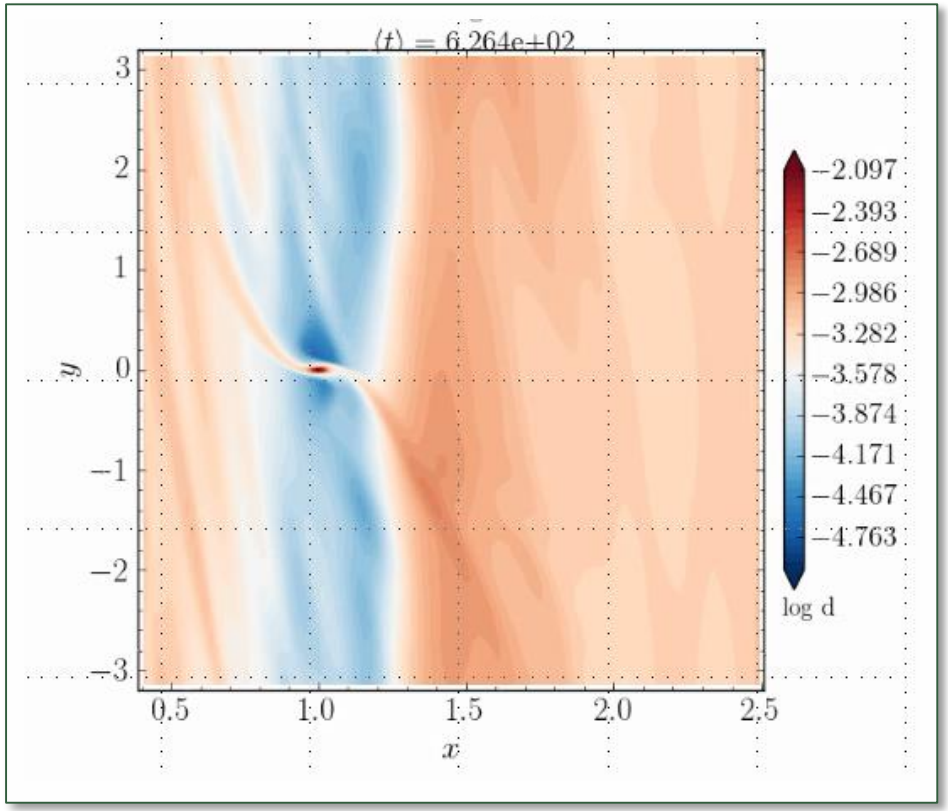
To be connected to “pebble atmosphere”: pebble accretion through a hydrostatic atmosphere

Jupiter size planet

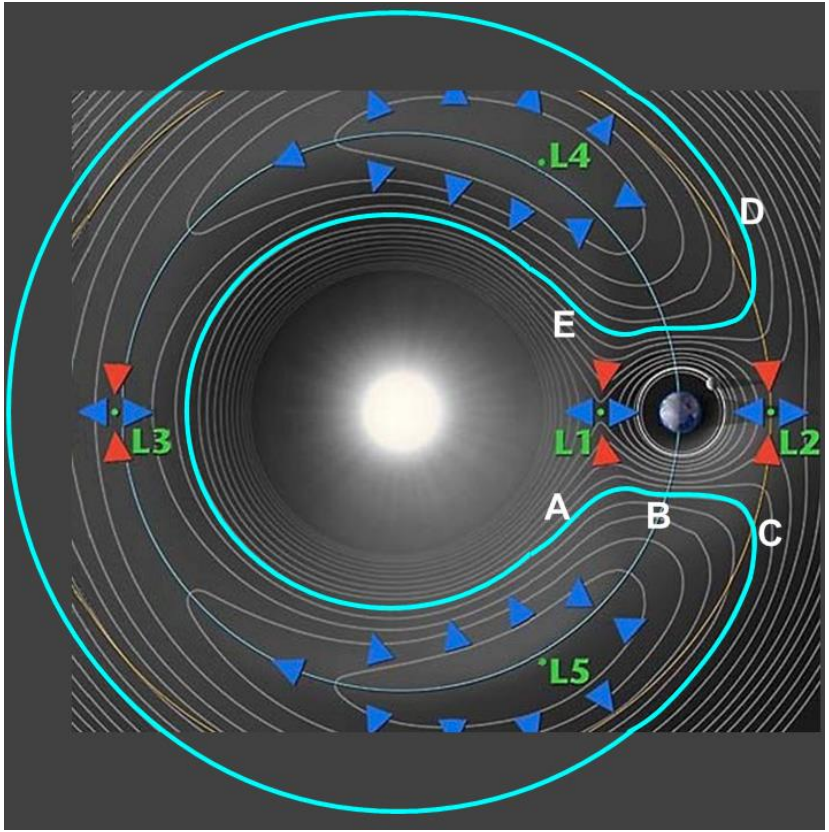
Here's the standard case, with a Jupiter size planet, evolved with the new DISPATCH code framework.

Resolution: 384 x 128

Core-time: a few hours



Near-planet dynamics: Horse-shoe orbits



Kepler rotation speed is faster inside, slower outside.

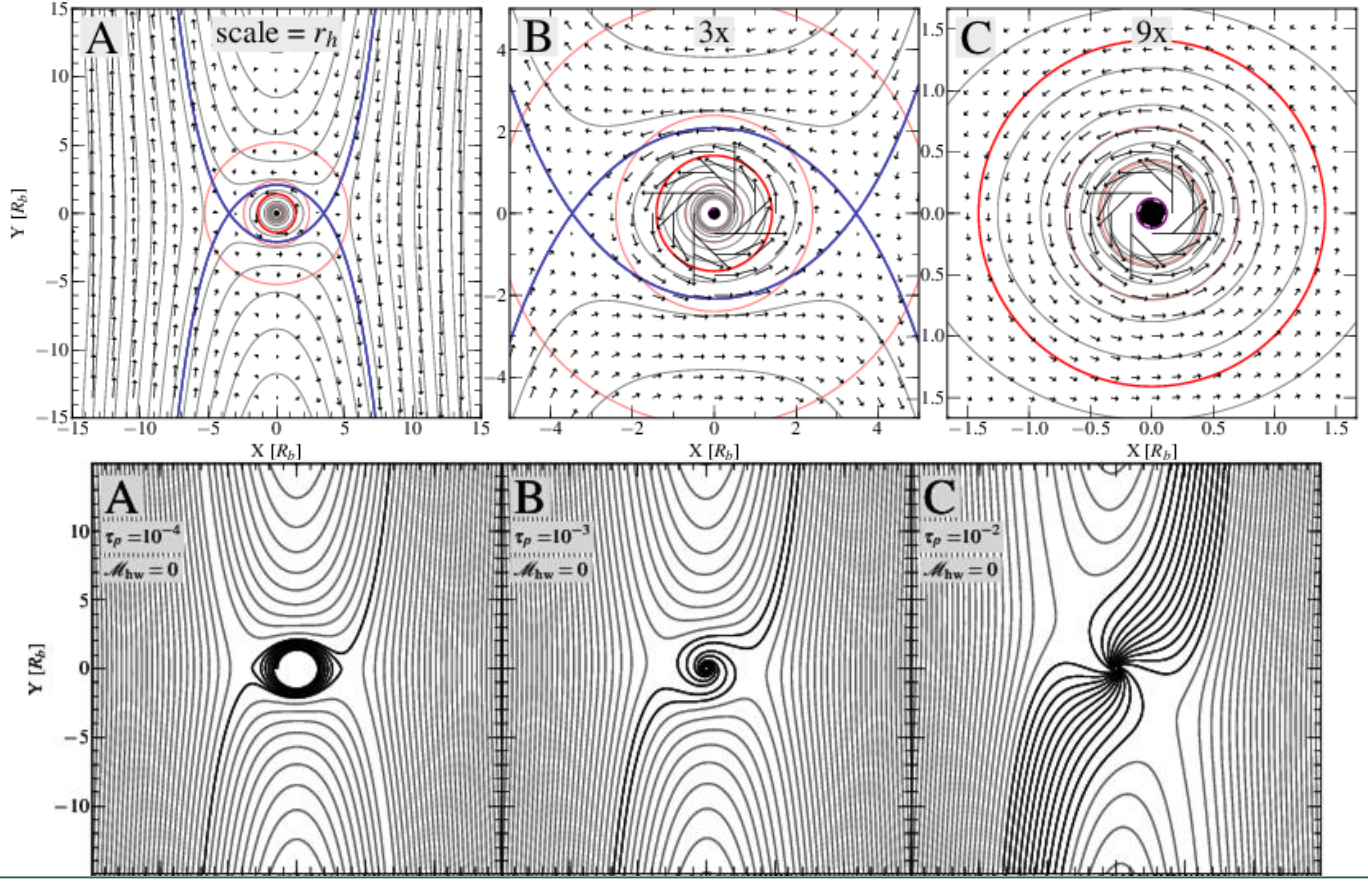
The slower outside flow is deflected towards the planet, but inward motion => acceleration in the direction of rotation, and it ends up in the faster inside stream.

When the slower flow approaches, the opposite thing happens; the pull of the planet => outward deflection => deceleration => ends up in the slower outside flow.

Janus and Epimetheus are a good “local” example of horseshoe orbits.

Image credit:
Wikipedia

Near-planet dynamics: Opens up for particle accretion



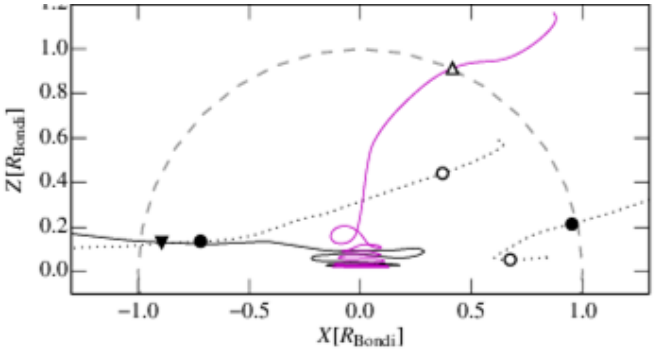
Ormel (2013)

Near-planet dynamics: Three-dimensional structure

L1 and L2 are points in the midplane where the ***radial force vanishes***:

radial-force = pull-of-planet + pull-of-Sun + centripetal force = 0

Ormel, Shi, & Kuiper (2014)



But ***vertically***:

vertical-force = pull-of-planet + pull-of-Sun

with ***both pulling towards the planet*** in the midplane!

fundamental ***asymmetry*** => ***systematic flows***

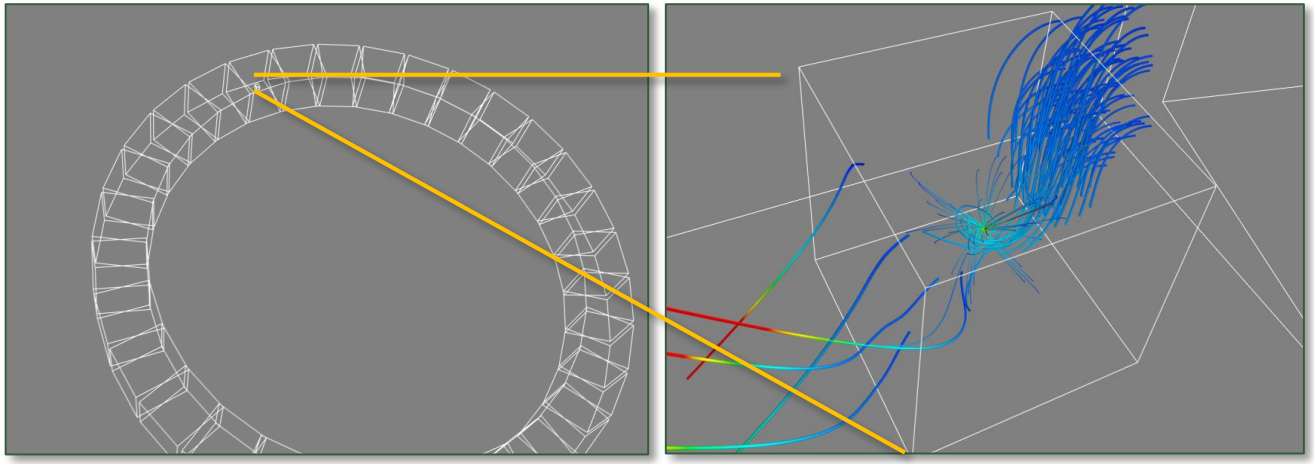
Inflow in polar regions, outflows in equatorial regions, pebbles can `drop out'

Rubik's Cube Hierarchies

Aim: ***study the accretion of chondrules*** onto planet embryos:

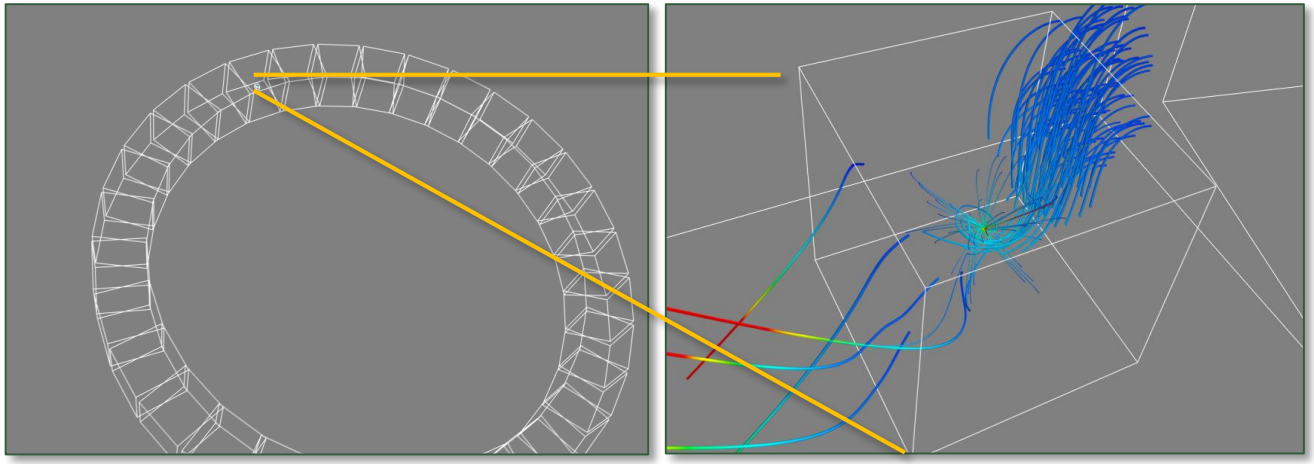
- Patch hierarchy
  - o Co-moving with Keplerian motion
  - o Resolving from 0.03-4000 R_{Earth} in 7 levels
  - o With pseudo-atmospheres represented by hierarchical meshes:
    - ▪Rubics Cube (3x3x3 = 27 patches – refining the innermost one recursively by factor 3)
    - ▪Using an efficient acceleration mechanism; allows relaxing ~1 orbit of evolution in ~ 2 core-days

Pebble accretion onto embryo with proto-atmosphere



We model an entire annulus, with a vertical size ~ disk scale height, using a number of quasi-cubical volume strung together around the disk, with one (or more) of the volumes containing a hierarchy of 3x3x3 subvolumes (Rubik's cubes), down to where the central cube contains the planet embryo.

Pebble accretion onto embryo with proto-atmosphere



We model an entire annulus with a vertical size & disk scale height using a n
one
(Ru

The chondrule accretion rate depends on number of factors that we can extract from our simulations, such as the production rate, the typical flow paths, the fate of chondrules “raining down” at large radii, etc.

We aim to build a plausible planet formation scenario, which both depends on and influences the transport of chondrules in the disk(s).

Summary, DISPATCH code framework (cf. arXiv later this week)

- Object orientation allows:
  - **Multiple-physics** + construction of hierarchical **scenes** = coupled simulations
  - Modular **built-in solvers**: HLLC, HLLD, Zeus, Stagger, …
  - Simple coupling to **external solvers**: PIC, Bifrost, …
- Large **gains in computing time** arising from
  - **Vectorization** and cache efficiency ⇒ 1 core-μs/cell-update on KNL
  - **Local time steps** ⇒ factor 5-50 speed-up in realistic cases
  - **Moving mesh** ⇒ additional speed-up, especially in disks
  - Task based **OpenMP scheduling** ⇒ flat KNL performance per core
  - **Nearest neighbor MPI** communication ⇒ in principle unlimited scaling

Summary, specific example cases

- ***Decaying turbulence*** (KITP benchmark, RAMSES/HLLC)
  - Cost reduction from **optimal patch size** (~factor 3)
  - Cost reduction from **local time steps** (~factor 1.5)
- Solar and stellar ***active regions*** (unigrid)
  - Cost reductions from **local time stepping** (~factor 30-100)
  - Cost reduction from **reduced cadence** radiative transfer
- Zoom-in simulations of ***star and planet formation*** (AMR, FMR)
  - Large cost reduction (~factor 100) from **optimal patch size**
  - Additional cost reduction (~factor 10) from **local time steps**
  - FARGO-type cost reduction (~factor 3) from **moving mesh**