cuRAMSES-kjhan

Technical Reference Manual

K-Section Ordering, Morton Key Octree,
Memory-Based Load Balancing, and
Performance Optimizations

Jaehyun Han

February 2026

Based on RAMSES (R. Teyssier)

`git@github.com:kjhan0606/cuRAMSES-kjhan.git`

Contents

Code Modifications

1 K-Section Ordering

1.1 Overview

The **k-section ordering** replaces the standard Hilbert curve domain decomposition with a hierarchical spatial bisection tree. Unlike the Hilbert curve approach that relies on a 1D space-filling curve, the k-section ordering partitions the 3D domain using recursive bisection along each coordinate axis, producing a balanced k-ary tree of spatial subdomains.

Key Advantage

The k-section tree enables **hierarchical MPI exchange** where communication follows the tree structure, achieving $O(\sum k_l)$ messages per exchange instead of $O(N_{\text{cpu}})$ all-to-all communication.

1.2 Hierarchical MPI Exchange

Two exchange routines are provided in `patch/cuda/ksection.f90`:

- `ksection_exchange_dp` — exclusive exchange (each item → exactly 1 destination CPU)
- `ksection_exchange_dp_overlap` — overlap exchange (items routed by spatial bounding box)

1.2.1 Exclusive Exchange

Each data item has a known destination CPU. The algorithm performs level-by-level correspondent exchange through the k-section tree, using MPI tags 100–300+level.

1.2.2 Overlap Exchange

Items have spatial extent (bounding box) and may need to reach multiple CPUs. The tree walk determines all destination CPUs whose spatial domain overlaps the item's bounding box. MPI tags 400–500+level are used.

1.2.3 Periodic Boundary Conditions

The optional `periodic=.true.` parameter enables handling of items that wrap around the domain boundary $[0, \text{scale}]$. For each wrapping dimension, $2^{n_{\text{wrap}}} - 1$ shifted copies are generated via bitmask subset enumeration before the tree walk.

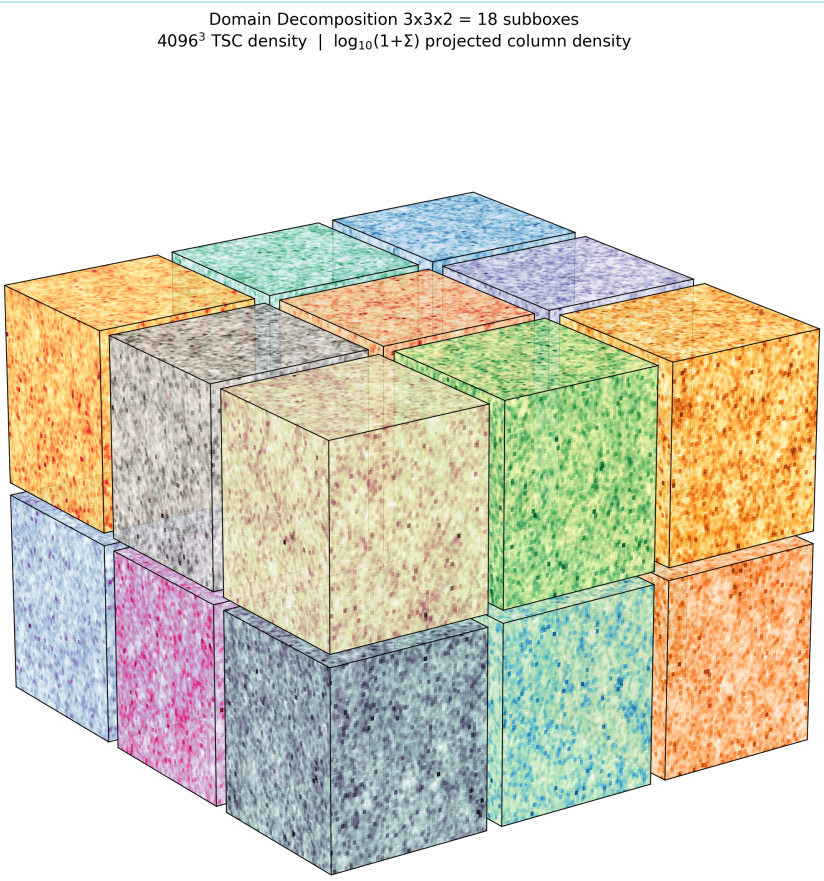

Figure 1.1: Example of k-section domain decomposition with a $3 \times 2 \times 2 = 12$ uniform partition. Each subbox surface shows the projected column density $\log_{10}(1+\Sigma)$ rendered with a distinct sequential colormap. The density field is computed on a 512^3 grid from $\sim$18.6 million dark matter particles.

1.3 Tree Navigation

The arrays `ksec_cpumin/cpumax` and `ksec_cpu_path` (set in `build_ksection` and at restart via `rebuild_ksec_cpuranges`) enable each CPU to navigate the tree efficiently.

1.4 Verification

The exchange routines are tested in `test_ksection_exchange` with 4 test cases:

1. Exclusive point exchange
2. Overlap point exchange
3. Full overlap exchange
4. Periodic overlap exchange (20% radius items)

All tests pass.

2 Ghost Zone Exchange via K-Section

2.1 AMR Ghost Zones

The standard RAMSES ghost zone exchange uses `MPI_ISEND/IRECV` with all-to-all communication patterns. This was replaced with k-section tree-routed exchange for the `ordering='ksection'` mode.

2.1.1 Modified File

`patch/cuda/virtual_boundaries.kjhan.f90`

2.1.2 Subroutines

Four k-section variants were implemented:

Subroutine	Direction	Data Type
`make_virtual_fine_dp_ksec`	Forward	`real(dp)`
`make_virtual_fine_int_ksec`	Forward	`integer`
`make_virtual_reverse_dp_ksec`	Reverse (+= accumulate)	`real(dp)`
`make_virtual_reverse_int_ksec`	Reverse (+= accumulate)	`integer`

2.1.3 Data Packing

Each emission grid is packed as:

$$\texttt{sendbuf}(1{:}\text{twotondim}+2,\ i) = [\underbrace{c_1, c_2, \ldots, c_8}_{\text{cell data}},\ \underbrace{\text{myid}}_{\text{sender}},\ \underbrace{i}_{\text{index}}]$$

The metadata (sender ID, emission index) allows the receiver to scatter data to the correct reception grid without requiring a priori knowledge of the communication pattern.

2.1.4 Dispatch

Dispatch is automatic via:

```
if(ordering=='ksection') then
   call make_virtual_fine_dp_ksec(xx, ilevel)
   return
end if
```

2.1.5 Bulk Exchange

Four bulk variants exchange all columns of a 2D array (e.g., `uold`, `f`, `unew`) in a single `ksection_exchange_dp` call:

Subroutine	Description
`make_virtual_fine_dp_bulk`	Forward bulk exchange
`make_virtual_fine_dp_bulk_ksec`	Forward bulk (ksection impl.)
`make_virtual_reverse_dp_bulk`	Reverse bulk exchange
`make_virtual_reverse_dp_bulk_ksec`	Reverse bulk (ksection impl.)

Buffer layout for `ncols` variables:

$$\texttt{sendbuf}\big((v-1)\cdot 2^3 + j,\ \text{idx}\big) = \texttt{xx}(\text{icell},\ v) \quad v = 1\ldots\text{ncols},\ j = 1\ldots 8$$

plus 2 metadata words (sender ID, index). This reduces MPI exchanges from N_{var} per level to 1 per array.

2.2 build_comm via K-Section

The `build_comm` subroutine's `MPI_ALLTOALL` + `MPI_ISEND/IRECV` pattern was also replaced with `ksection_exchange_dp`.

2.3 MPI_ALLTOALL Replacements

Additional `MPI_ALLTOALL` calls in the following files were replaced with k-section exchange:

- `particle_tree.kjhan.f90`
- `init_part.f90`
- `multigrid_fine_commons.f90` (in `build_parent_comms_mg`)

2.4 Pre-Allocated Buffer Pool

Per-level small arrays (child count, peer list, MPI requests) were converted to `save` variables to eliminate $\sim$100 allocations/deallocations per call. The `peer_recv` buffer uses grow-only capacity, and the first dimension must exactly match `nprops` for MPI stride correctness.

3 Multigrid Poisson K-Section Communication

3.1 Motivation

The multigrid Poisson solver (`poisson-mg`) accounts for 29–41% of total runtime. The original MPI communication used `MPI_ISEND/IRECV` with all-to-all patterns across all CPUs.

3.2 Implementation

Four k-section variants were added to `patch/cuda/multigrid_fine_commons.f90`:

Subroutine	Direction	Data Type
`make_virtual_mg_dp_ksec`	Forward	`real(dp)`
`make_virtual_mg_int_ksec`	Forward	`integer`
`make_reverse_mg_dp_ksec`	Reverse (+= accumulate)	`real(dp)`
`make_reverse_mg_int_ksec`	Reverse (+= accumulate)	`integer`

3.2.1 Key Differences from AMR Exchange

The MG communication uses different data structures from the standard AMR exchange:

Aspect	AMR	Multigrid
Active grids	`active(ilevel)`	`active_mg(myid,ilevel)`
Reception	`reception(icpu,ilevel)`	`active_mg(icpu,ilevel)`
Emission	`emission(icpu,ilevel)`	`emission_mg(icpu,ilevel)`
Data arrays	`xx(igrid+iskip)`	`active_mg%u(icell,ivar)`
Indexing	Global grid index	Local offset in active_mg

3.2.2 Forward Exchange

1. Pack: for each `emission_mg(icpu)%igrid(i)`, gather `twotondim` values from `active_mg(myid)%u` + metadata (sender_id, index)
2. Exchange via `ksection_exchange_dp`
3. Scatter: use metadata to write into `active_mg(sender)%u(ridx + step, ivar)`

3.2.3 Reverse Exchange (Accumulation)

1. Pack: for each remote `active_mg(icpu)%u(i + step, ivar)` + metadata
2. Exchange via `ksection_exchange_dp`
3. Accumulate: `active_mg(myid)%u(emission_mg(sender)%igrid(ridx) + step, ivar) += recvbuf`

4 Morton Key Octree

4.1 Overview

The `nbor` array (6 neighbor pointers per grid) was replaced with a Morton key hash table for $O(1)$ neighbor lookup. This saves $6 \times 8 \times N_{\text{gridmax}}$ bytes of memory.

4.1.1 Modified Files

- `patch/oct_tree/morton_keys.f90` — Morton key computation
- `patch/oct_tree/morton_hash.f90` — Hash table and helper functions
- `patch/oct_tree/morton_init.f90` — Initialization and verification
- `patch/oct_tree/refine_utils.f90` — Hash table maintenance
- `patch/oct_tree/nbors_utils.kjhan.f90` — Neighbor lookup functions

4.2 Morton Key

A 64-bit Morton key is computed by interleaving the 3D integer coordinates (21 bits per dimension):

$$\text{key} = \text{interleave}\left(\lfloor x_g \cdot 2^{l-1} \rfloor,\ \lfloor y_g \cdot 2^{l-1} \rfloor,\ \lfloor z_g \cdot 2^{l-1} \rfloor\right)$$

4.3 Hash Table

A per-level open-addressing hash table with linear probing and power-of-2 capacity maps Morton keys to grid indices. The table is maintained in `make_grid_coarse/fine` and `kill_grid`, with a full rebuild after each time step.

4.4 Neighbor Lookup

Two helper functions in the `morton_hash` module:

- `morton_nbor_grid(igrid, ilevel, j)` — returns `son(nbor(igrid,j))` equivalent
- `morton_nbor_cell(igrid, ilevel, j)` — returns `nbor(igrid,j)` equivalent

Direction convention: $j = 1{:}{-x},\ 2{:}{+x},\ 3{:}{-y},\ 4{:}{+y},\ 5{:}{-z},\ 6{:}{+z}$.

4.5 nbor Array Removal (Phase 4)

The `nbor` array is allocated as `allocate(nbor(1:1,1:1))` (minimum size to avoid compilation errors). All code paths use Morton lookup exclusively.

5 Memory-Based Load Balancing

5.1 Motivation

Standard RAMSES load balancing distributes cells evenly across CPUs, but cells with many particles consume significantly more memory. Memory-based balancing weights each cell by its memory footprint.

5.2 Cost Function

$$\text{cell\_cost} = \frac{\texttt{mem\_weight\_grid}}{\text{twotondim}} + \text{numbp(igrid)} \times \frac{\texttt{mem\_weight\_part}}{\text{twotondim}}$$

where:

- `mem_weight_grid` = 270 (default) — memory per grid in dp-equivalents
- `mem_weight_part` = 12 (default) — memory per particle in dp-equivalents

5.3 Implementation Details

- All histogram variables use 64-bit integers (`integer(i8b)`) with `MPI_INTEGER8`
- `numbp` is synchronized for virtual/reception grids before cost computation, then restored afterwards
- The `numbp` restore uses a save/restore pattern to avoid breaking the particle tree

5.4 Parameters

Controlled by three namelist parameters in `&RUN_PARAMS`:

- `memory_balance = .true.` — enable memory-based balancing
- `mem_weight_grid = 270` — grid memory weight
- `mem_weight_part = 12` — particle memory weight

6 Memory Savings: Large Array Optimization

6.1 Overview

Several large arrays were eliminated or converted to on-demand allocation to reduce steady-state memory usage by ∼960 MB (for `ngridmax`=5M).

Array	Strategy	Savings
`hilbert_key`	`allocate(1:1)` for ksection	∼640 MB
`bisec_ind_cell` + `cell_level`	On-demand alloc/dealloc	∼320 MB
`defrag_map`	Local scratch during defrag	minor
`nbor`	`allocate(1:1,1:1)` (Morton)	∼240 MB

7 IC Reading with Stream Access

7.1 Motivation

Sequential Fortran I/O requires reading all preceding planes to reach a target plane, which is $O(n^2)$ for large files. Stream access enables direct byte-offset seeks.

7.2 Implementation

Fortran 2003 `ACCESS='STREAM'` is used with computed byte offsets:

```
hdr_bytes = 52 + (i3-1)*plane_bytes + 5
plane_bytes = n1*n2*4 + 8  ! data + 2 record markers
```

Applied to hydro IC (deltab, velocity, temperature), particle velocity and position files. Only for `multiple=.false.` mode.

7.2.1 Modified Files

- `patch/Horizon5-master-2/init_flow_fine.f90`
- `init_part.f90`

8 Load Balance Profiling and Tuning

8.1 Internal Timing

Detailed timing breakdown was added to `load_balance` in `patch/cuda/load_balance.kjhan.f90`:

Section	Description	Typical (s/step)
`numbp_sync`	MPI sync of numbp for virtual grids	0.8–1.0
`cmp_new_cpu_map`	Build ksection + compute new map	0.4–0.6
`expand_pass`	build_comm + make_virtual loop	0.8–1.5
`grid_migration`	Linked-list reconnection	< 0.01
`allreduce+cpumap_update`	MPI_ALLREDUCE $\times 4$ + cpu_map	2.3–3.3
`shrink_pass`	flag_fine + build_comm loop	0.4–1.0

8.2 nremap Tuning

The `nremap` parameter controls load balancing frequency (every N coarse steps). Testing with 200M particles on 12 ranks showed:

nremap	Total (s)	Loadbal (s)	Speedup	Note
1	303.8	64.4 (21.2%)	—	Baseline
3	269.9	24.7 (9.1%)	1.13×	
5	249.8	15.7 (6.3%)	**1.22×**	**Optimal**
10	258.6	11.6 (4.5%)	1.17×	Imbalance grows

Default Setting

`nremap=5` is set as the default. All four configurations produce **bit-identical** results: `econs=3.77E-03`, `epot=-1.88E-06`, `ekin=1.23E-06` at step 10.

8.3 Min/Max Memory Reporting

The `writemem_minmax` subroutine prints per-step min/max memory usage across all MPI ranks.

II

Namelist Reference

9 RUN_PARAMS

Runtime control parameters.

Parameter	Type	Default	Description
`cosmo`	logical	`.false.`	Enable cosmological simulation
`pic`	logical	`.false.`	Enable Particle-In-Cell
`poisson`	logical	`.false.`	Enable Poisson gravity solver
`hydro`	logical	`.false.`	Enable hydrodynamics
`rt`	logical	`.false.`	Enable radiative transfer
`sink`	logical	`.false.`	Enable sink particles
`verbose`	logical	`.false.`	Verbose output
`debug`	logical	`.false.`	Debug mode
`nrestart`	integer	0	Restart file number (0 = new run)
`nstepmax`	integer	1000000	Maximum number of coarse steps
`ncontrol`	integer	1	Frequency of control variable output
`nsubcycle`	integer[]	2	Subcycling factor per level
`nremap`	integer	**5**	Load balancing frequency (0=never)
`ordering`	char	`hilbert`	Domain decomposition: `hilbert`, `bisection`, `ksection`
`static`	logical	`.false.`	Static (no refinement) mode
`overload`	integer	1	MPI overload factor
`cost_weighting`	logical	`.true.`	CPU time-based cost weighting
Memory-based load balancing (new)			
`memory_balance`	logical	`.false.`	Enable memory-weighted balancing
`mem_weight_grid`	integer	270	Memory per grid (dp-equivalents)
`mem_weight_part`	integer	12	Memory per particle (dp-equivalents)

10 AMR_PARAMS

Adaptive Mesh Refinement grid parameters.

Parameter	Type	Default	Description
`levelmin`	integer	1	Minimum (uniform) refinement level
`levelmax`	integer	1	Maximum refinement level
`ngridmax`	integer8	0	Max grids per CPU (0 = auto)
`ngridtot`	integer8	0	Total grids for auto-computation
`npartmax`	integer	0	Max particles per CPU (0 = auto)
`nparttot`	integer	0	Total particles for auto-computation
`nexpand`	integer[]	1	Mesh expansion layers per level
`boxlen`	real(dp)	1.0	Box side length in code units

11 OUTPUT_PARAMS

Output control parameters.

Parameter	Type	Default	Description
`noutput`	integer	1	Number of scheduled outputs
`foutput`	integer	1000000	Output every N steps
`fbackup`	integer	1000000	Backup every N steps
`aout`	real[]	1.1	Output expansion factors (cosmo)
`tout`	real[]	0.0	Output times (non-cosmo)
`output_mode`	integer	0	Hi-res output mode
`gadget_output`	logical	`.false.`	Write Gadget-format snapshots
`walltime_hrs`	real(dp)	-1	Job walltime (hours, < 0 = ignore)
`minutes_dump`	real(dp)	1.0	Dump this many minutes before walltime

12 INIT_PARAMS

Initial conditions parameters.

Parameter	Type	Default	Description
`filetype`	char(20)	`ascii`	IC file format: `ascii`, `grafic`, `gadget`
`initfile`	char[]	`' '`	IC file path per level
`multiple`	logical	`.false.`	Multiple IC files per rank
`nregion`	integer	0	Number of IC regions

13 REFINE_PARAMS

Refinement criteria parameters.

Parameter	Type	Default	Description
`m_refine`	real[]	-1	Lagrangian mass threshold per level
`ivar_refine`	integer	-1	Variable index for gradient refinement
`var_cut_refine`	real(dp)	-1	Variable threshold
`mass_cut_refine`	real(dp)	-1	Particle mass threshold
`interpol_var`	integer	0	Interpolation variable (0=conservative, 1=primitive)
`interpol_type`	integer	1	Interpolation type (0=MinMod, 1=MonCen)
`sink_refine`	logical	`.false.`	Fully refine around sinks
`jeans_ncells`	real(dp)	-1	Jeans length in cells (>0 enables polytropic EOS)

14 HYDRO_PARAMS

Hydrodynamics solver parameters.

Parameter	Type	Default	Description
`gamma`	real(dp)	$\frac{5}{3}$	Adiabatic index γ
`courant_factor`	real(dp)	0.8	Courant–Friedrichs–Lewy number
`scheme`	char(20)	`muscl`	Hydro scheme (`muscl`)
`slope_type`	integer	1	Slope limiter (1=MinMod, 2=MonCen, 3=unlimited)
`pressure_fix`	logical	`.false.`	Pressure floor for strong shocks
`beta_fix`	real(dp)	0.0	Pressure fix strength
`isothermal`	logical	`.false.`	Isothermal mode

15 POISSON_PARAMS

Gravity and Poisson solver parameters.

Parameter	Type	Default	Description
`epsilon`	real(dp)	10^{-4}	Multigrid convergence criterion
`gravity_type`	integer	0	Gravity type (0=self-gravity, >0=analytic)
`cg_levelmin`	integer	999	Min level for CG solver fallback
`cic_levelmax`	integer	0	Max level for CIC particle interpolation

16 PHYSICS_PARAMS

Subgrid physics parameters (star formation, feedback, cooling).

Parameter	Type	Default	Description
`cooling`	logical	`.false.`	Enable radiative cooling
`metal`	logical	`.false.`	Enable metal tracking
`haardt_madau`	logical	`.false.`	UV background
`z_reion`	real(dp)	8.5	Reionization redshift
Star formation			
`n_star`	real(dp)	0.1	SF density threshold (H/cc)
`t_star`	real(dp)	0.0	SF timescale (Gyr)
`eps_star`	real(dp)	0.0	SF efficiency
`T2_star`	real(dp)	0.0	ISM polytropic temperature
`g_star`	real(dp)	1.6	ISM polytropic index
`sf_birth_properties`	logical	`.true.`	Output stellar birth properties
Cosmology (read from IC header or namelist)			
`omega_b`	real(dp)	0.0	Baryon density Ω_b
`omega_m`	real(dp)	1.0	Matter density Ω_m
`omega_l`	real(dp)	0.0	Dark energy Ω_Λ
`h0`	real(dp)	1.0	Hubble constant H_0 (km/s/Mpc)
Feedback			
`f_ek`	real(dp)	1.0	SN kinetic energy fraction
`rbubble`	real(dp)	0.0	SN superbubble radius (pc)
`yieldtable-filename`	char	—	Yield table file path

17 Example Namelist

17.1 Cosmological Simulation with Memory Balancing

cosmo_ksection_membal.nml

```
&RUN_PARAMS
cosmo=.true.
pic=.true.
poisson=.true.
hydro=.true.
nrestart=0
nremap=5
nsubcycle=1,1,2
ncontrol=1
nstepmax=10
ordering='ksection'
memory_balance=.true.
/

&OUTPUT_PARAMS
noutput=1
aout=1.0
/

&INIT_PARAMS
filetype='grafic'
initfile(1)='/path/to/ics/level_008'
/

&AMR_PARAMS
levelmin=8
levelmax=10
nexpand=1
ngridtot=40000000
nparttot=200000000
/

&REFINE_PARAMS
m_refine=3*8.,
ivar_refine=0
interpol_var=1
interpol_type=0
/
```

```
&HYDRO_PARAMS
gamma=1.6666667
courant_factor=0.8
scheme='muscl'
slope_type=1
/

&POISSON_PARAMS
/

&PHYSICS_PARAMS
sf_birth_properties=.false.
yieldtablefilename='yield_table.asc'
/
```

Build and Testing

18 Build Instructions

18.1 Prerequisites

- Intel Fortran Compiler (`ifx`) with MPI support (`mpiifx`)
- OpenMP support (`-qopenmp`)

18.2 Build

Build Commands

```
cd bin
make clean
make
```

The binary is produced as `bin/ramses_final3d`.

18.3 Compile-Time Options

Key preprocessor flags set in the Makefile:

Flag	Meaning
`-DNDIM=3`	Three-dimensional simulation
`-DNVECTOR=32`	Vector length for grid sweeps
`-DLONGINT`	64-bit integer grid indices
`-DQUADHILBERT`	Quad-precision Hilbert keys
`-DNVAR=11`	Number of hydro variables
`-DNPRE=8`	Passive scalar variables

19 Verification Tests

19.1 Test Configuration

Setting	Value
IC	MUSIC `level_008` (GRAFIC2 format)
MPI ranks	12
Levels	8–10
Steps	10
Ordering	ksection

19.2 Reference Values (nremap=5)

Test	nparttot	econs	epot	ekin	mcons
membal	200M	3.77E-03	−1.88E-06	1.23E-06	−1.84E-16
nomembal	80M	3.77E-03	−1.88E-06	1.23E-06	−1.84E-16

19.3 Running Tests

Membal Test

```
cd test_ksection/run_cosmo_membal
cp ../../bin/ramses_final3d .
mpirun -np 12 ./ramses_final3d \
    ../cosmo_ksection_membal.nml
```

Verify that at step 10: `econs=3.77E-03`, `epot=-1.88E-06`, `ekin=1.23E-06`.

A Modified Files Summary

File	Modifications
patch/cuda/	
`amr_parameters.jaehyun.f90`	Memory balance params, nremap=5 default
`amr_commons.kjhan.f90`	grid_level array, communicator type
`read_params.jaehyun.f90`	Read new namelist params
`bisection.f90`	nc_in parameter, 64-bit histograms
`ksection.f90`	K-section tree + exchange routines
`load_balance.kjhan.f90`	numbp sync, bulk exchange, internal timing
`virtual_boundaries.kjhan.f90`	Ksec ghost exchange + bulk + build_comm
`multigrid_fine_commons.f90`	MG ksection communication
`init_amr.f90`	Memory savings, Morton init
`update_time.f90`	Memory reporting
`adaptive_loop.jaehyun.f90`	writemem_minmax, Morton rebuild, bulk exchange
patch/oct_tree/	
`morton_keys.f90`	Morton key computation
`morton_hash.f90`	Hash table + neighbor helpers
`morton_init.f90`	Build and verify
`refine_utils.f90`	Hash maintenance in grid ops
`nbors_utils.kjhan.f90`	Morton-based neighbor lookup
patch/Horizon5-master-2/	
`init_flow_fine.f90`	Stream access IC reading
`init_part.f90`	Stream access particle IC
`particle_tree.kjhan.f90`	MPI_ALLTOALL $\rightarrow$ ksection
`amr_step.jaehyun.f90`	Bulk virtual exchange