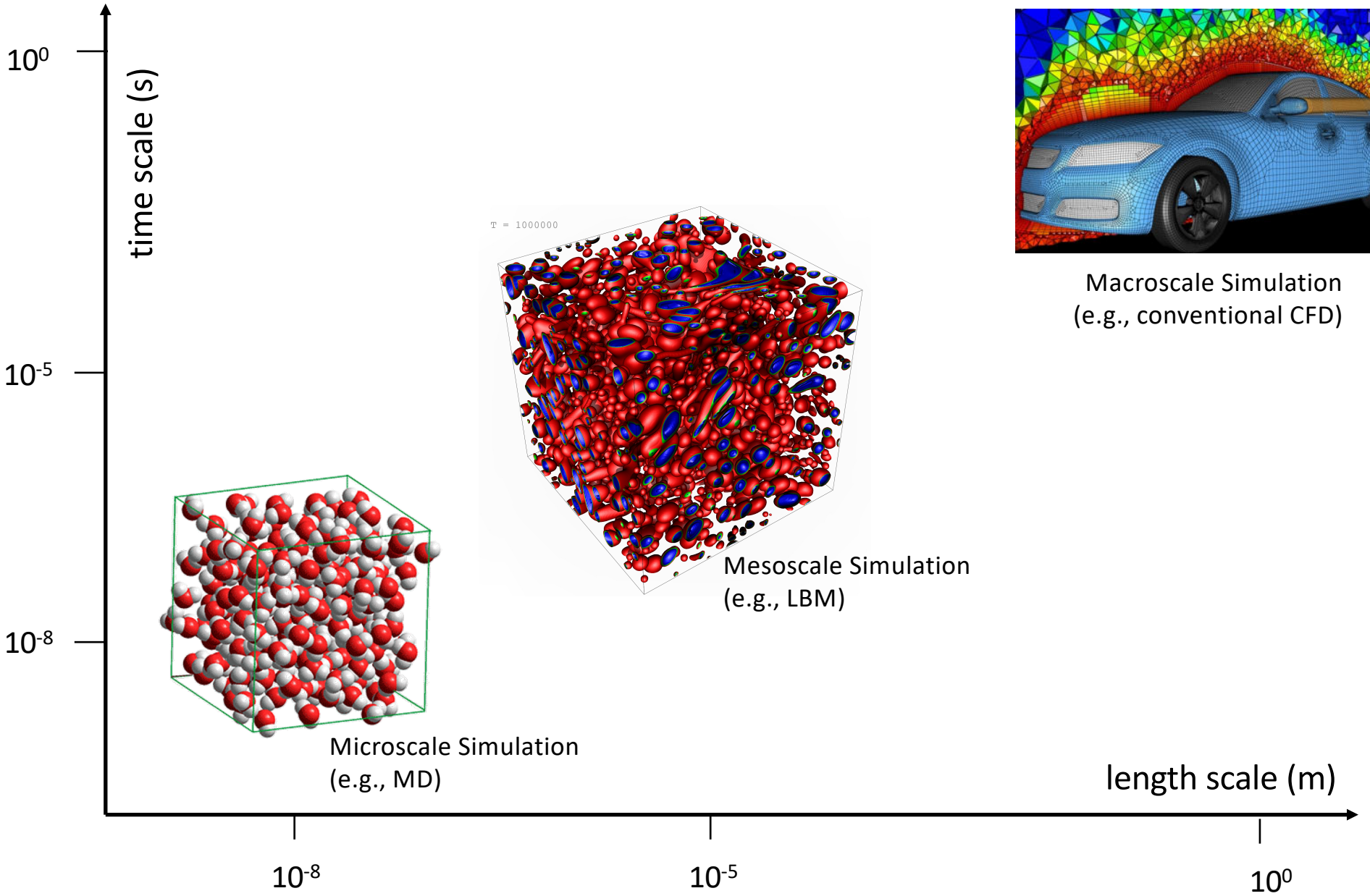
HPC Best Practices: LBM

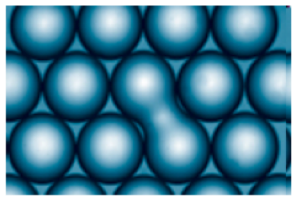
Ivan Girotto – igirotto@ictp.it
International Centre for Theoretical Physics (ICTP)

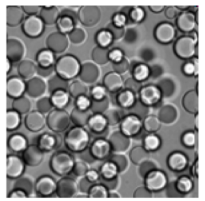
Relevant Scales in Fluid Dynamics

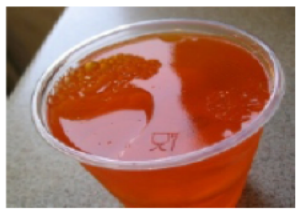


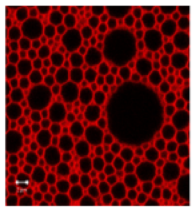
Multicomponent Fluid Emulsions

Solid or Fluid ?!







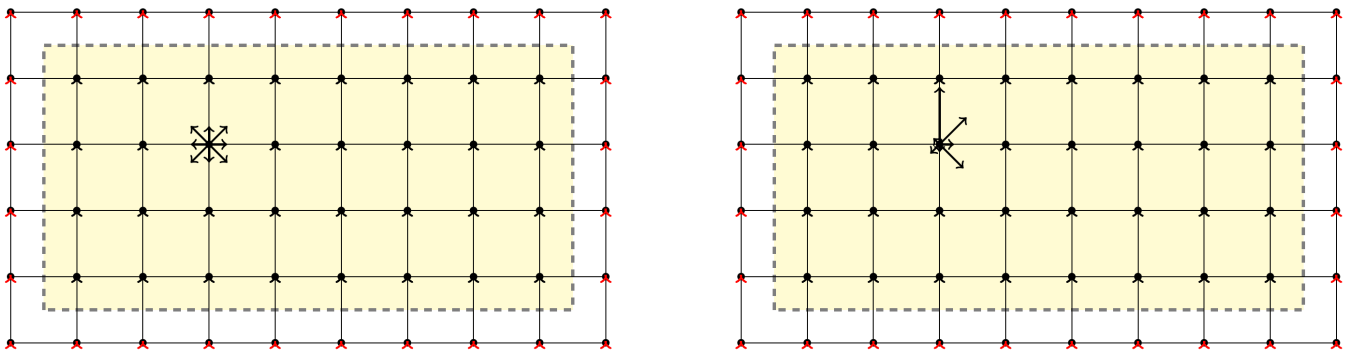


Colloidal Gels and Dense Emulsions
Colloidal Suspensions
Foams
Granular Materials & pastes
...and many others

The Lattice Boltzmann scheme

$$f_i(\boldsymbol{x} + \boldsymbol{c}_i \Delta t, t + \Delta t) - f_i(\boldsymbol{x}, t) = -\frac{1}{\tau}\left[f_i(\mathbf{x}, t) - f_i^{\mathrm{eq}}(\mathbf{x}, t)\right]$$

- f_i probability distribution function for LBM populations
- $i = 0, \ldots, N$ indexes the population streaming with velocity c_i
- The Lattice Boltzmann scheme is composed by two steps:
  - Streaming: involves only memory-to-memory copies
  - Collision: involves only (local) floating point operations

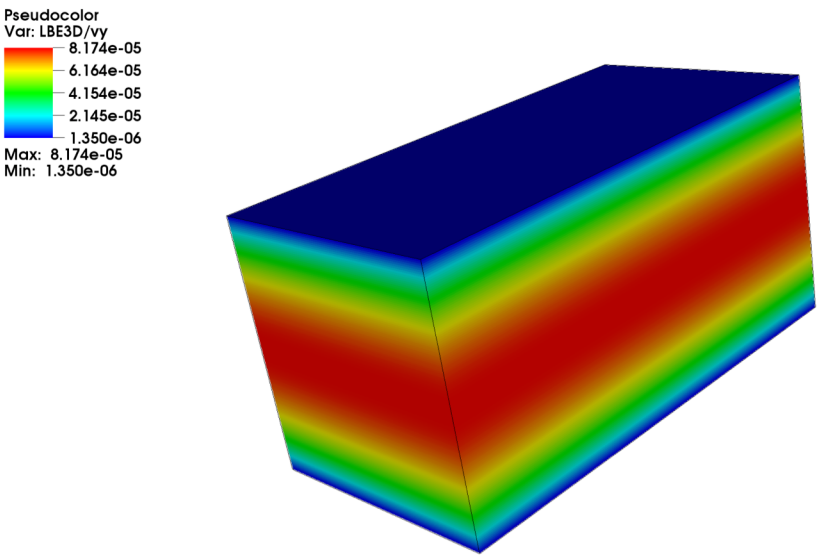


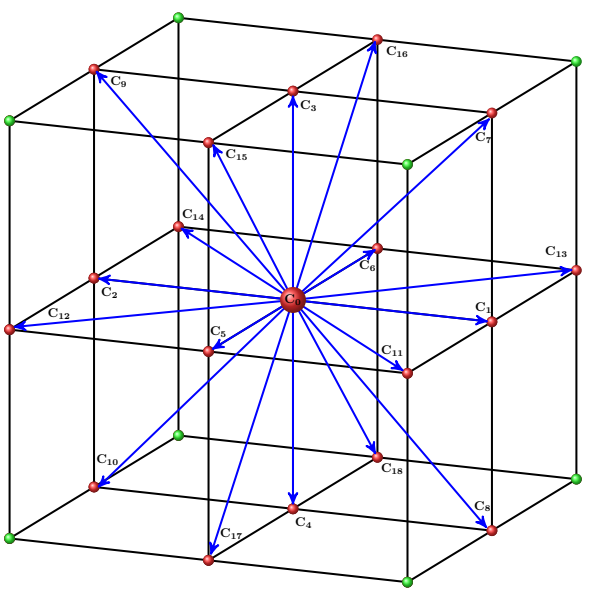
LBE3D: Application Description

- Nowadays commonly applied in several science and engineering fields to accurately model single-, multi-phase and multi-component flows

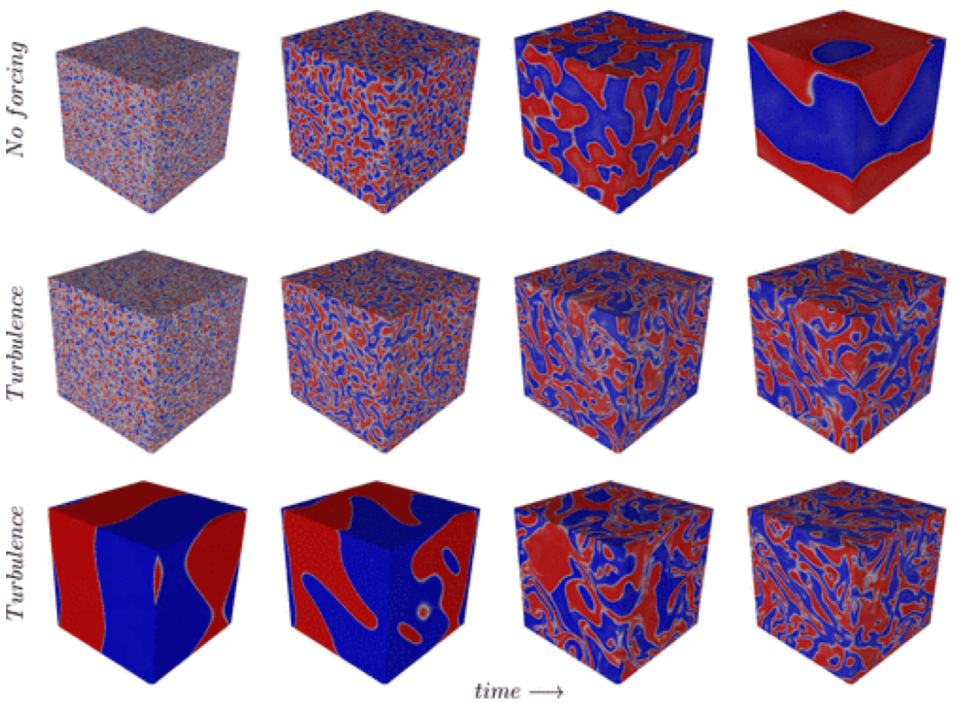
- D3Q19 LB stencil, a 3-dimensional model with a set of 19 population elements corresponding to pseudo-particles moving one lattice point away along all 19 possible directions, at each time step

- The LBE3D code is developed and used mainly by the group of Prof. Toschi @ TU/e





Multicomponent Emulsion



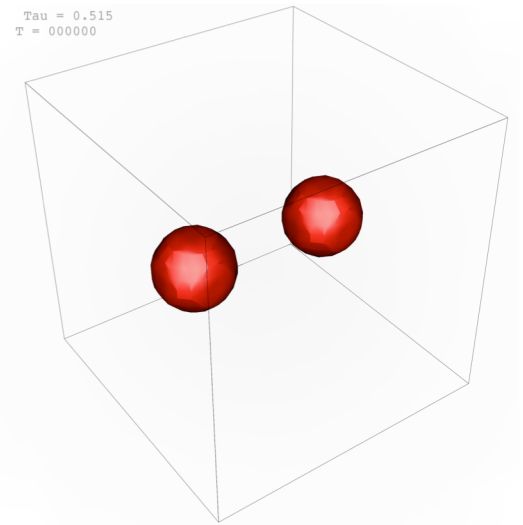
- Multicomponent fluid are extremely common in industrial as well as natural processes
- Complex fluid that can display a surprisingly rich phenomenology, typical of soft glass material

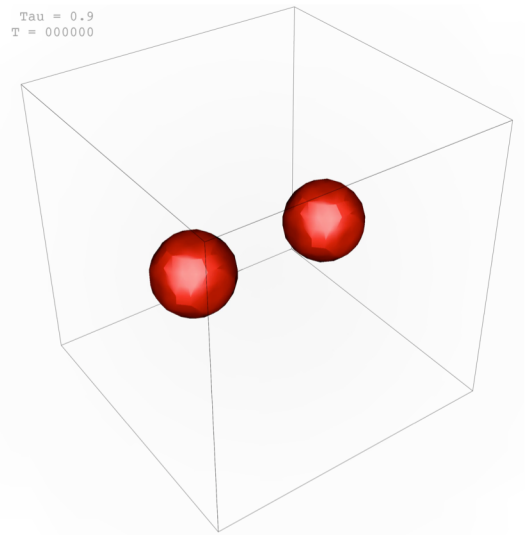
$$f_{\sigma a}(\boldsymbol{x}+\boldsymbol{c}_a,\boldsymbol{c}_a;t+1)-f_{\sigma a}(\boldsymbol{x},\boldsymbol{c}_a;t)=-\frac{1}{\tau_{LB,\sigma}}\left(f_{\sigma a}-f_{\sigma a}^{(eq)}\right)(\boldsymbol{x},\boldsymbol{c}_a;t)+F_{\sigma a}(\boldsymbol{x},\boldsymbol{c}_a;t)$$

$$f_{\sigma a}^{(eq)}=w_a\rho_\sigma\left(1+\frac{\boldsymbol{u}\cdot\boldsymbol{c}_a}{c_s^2}+\frac{\boldsymbol{uu}:\left(\boldsymbol{c}_a\boldsymbol{c}_a-c_s^2I\right)}{2c_s^4}\right)$$

The Computational Challenge

- Several millions of time steps at 256^3, 512^3 and 1024^3 scaling for few hundreds to tens of thousands cores
- Study the dynamic of the fluids as well as the final morphology applying various level of turbulent forcing
- Study several level of viscosities by creating different volume ratio between the two components
- Staring from the previous project we want to find the correct parametrization to guarantee the correct coupling parameters at various viscosity levels





TurEmu - The physics of (turbulent) emulsions

- 50 Million CPU-h on MARCONI - A2 (KNL) - ~ 20PFlop/s Lenovo NeXtScale interconnected with Intel OmniPath Cluster

- 66 Million CPU-h on MareNostrum ~ 13.7PFlop/s equipped with Lenovo ThinkSystem SD530 compute nodes. Each node has two Intel Xeon Platinum chips. The system is interconnected with Intel OmniPath Cluster

The code

- It a stencil code that at every time step requires communication between the neighbor processes
- It implements a flexible distribution of processes among all directions of the 3D domain
- It implements sequential as well as parallel I/O using HDF5 format for forces, velocities, densities (multicomponent)
  - The I/O frequency is a runtime parameter, different files are created for each time step (i.e., density_t.[step number].h5)
- All statistics are logged on text file
  - Serialized logging from the I/O root process
- It implements a system of check point restart

Grid of Processes (Virtual Topology) /1

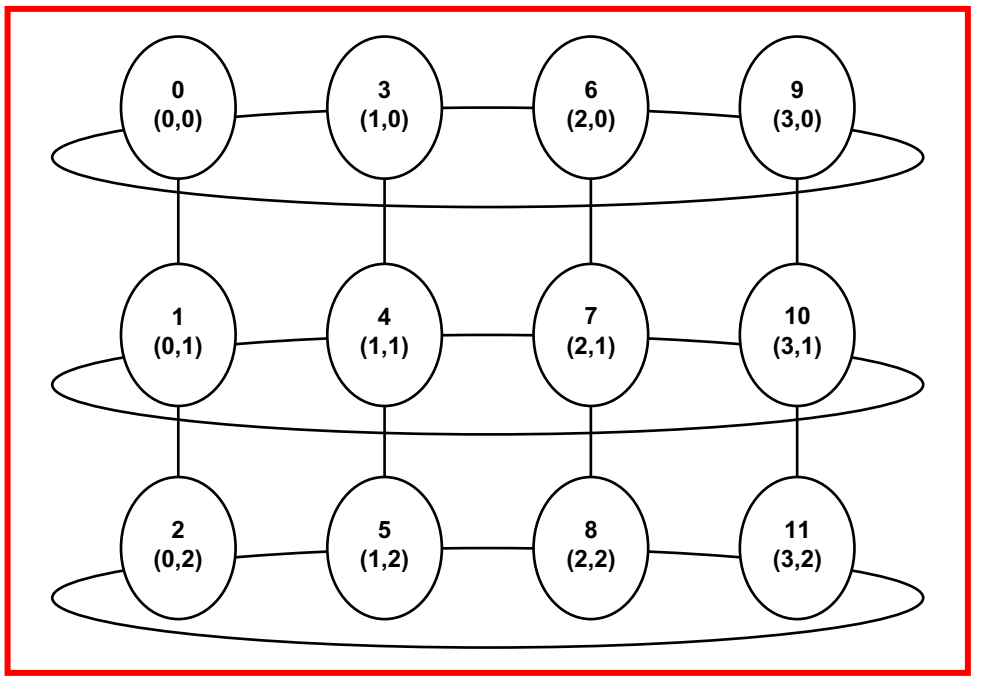
- The MPI library makes available interfaces to create a cartesian grid of processes (extremely convenient)
- The creation of a topologies produces new communicators
  - Communication among the processors grid coordinates
- Interfaces are provided to compute process ranks, based on the topology naming scheme and vice versa
- Features:
  - each process is connected to its neighbor in a virtual grid
  - boundaries can be cyclic, or not
  - processes are identified by Cartesian coordinates
  - compute process neighbor rank in the topology among different directions

Grid of Processes (Virtual Topology) /1

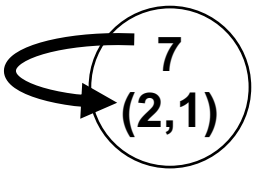
```
MPI_Dims_create(int nnodes, int ndims, int *dims)
```

```
MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims,
                int *periods, int reorder, MPI_Comm  *comm_cart)
```

comm_old = MPI_COMM_WORLD
ndims = 2
dims = (4, 3)
periods = (1/.true., 0/.false.)
reorder = *see next slide*

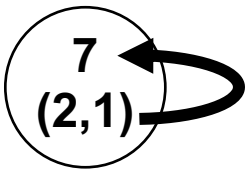


Grid of Processes (Virtual Topology) /2



- Mapping ranks to process grid coordinates

```
MPI_Cart_coords( MPI_Comm comm_cart, int rank,
                 int maxdims, int *coords)
```



- Mapping process grid coordinates to ranks

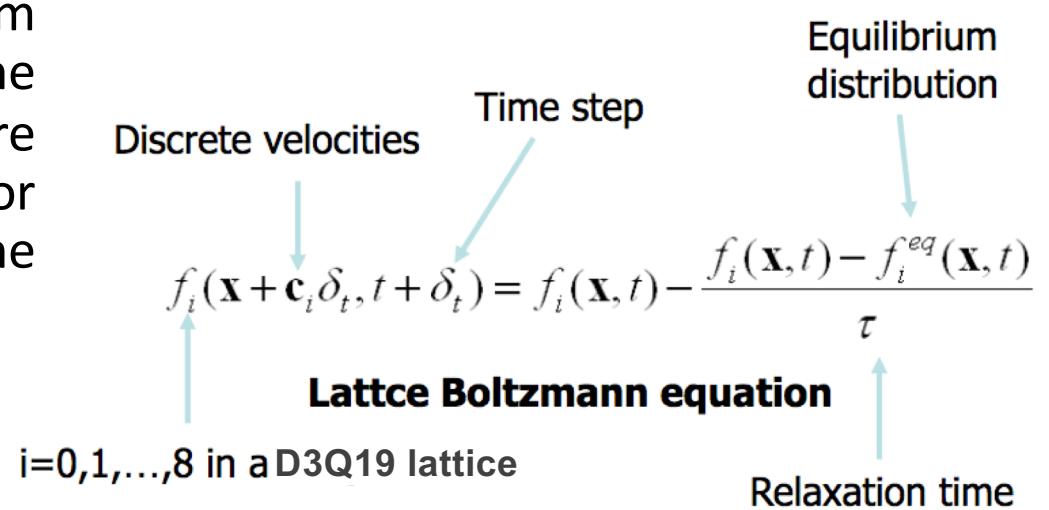
```
MPI_Cart_rank(MPI_Comm comm_cart, int *coords,
              int *rank)
```

The Intel ® Xeon Phi™ CPU 7230 Processor

- 64-cores (4-way SMT) @ 1.30 GHz, supporting AVX-512
  - 3 TFlop/s DP
- 98 GB DDR4
  - peak raw bandwidth of 115.2 GB/s
- 16 GB on chip MCDRAM
  - peak raw bandwidth of more than 450 GB/s
- Quadrant cluster configuration
  - 64-cores available are divided in four quadrants, each directly connected to one MCDRAM bank
- MCDRAM configurable at boot time: Flat, Cache or Hybrid
- Intel Compiler and Intel MPI
  - ”KMP_AFFINITY=compact” and ”I_MPI_PIN_DOMAIN=socket”
  - `numactl -m 1 mpirun ./lbe3d`

Lattice Boltzmann Method (LBM)

- Populations are first moved from lattice-site to lattice-site applying the propagate operator, and then are modified through a collisional operator changing their values according to the local equilibrium condition.




```
1:  for all time step do
2:      < Set boundary conditions >
3:      for all lattice site do
4:              < Move >
5:      for all lattice site do
6:              < Hydrovar >
7:      for all lattice site do
8:              < Equili >
9:      for all lattice site do
10:             < Collis >
11:     end for
12: end for
```

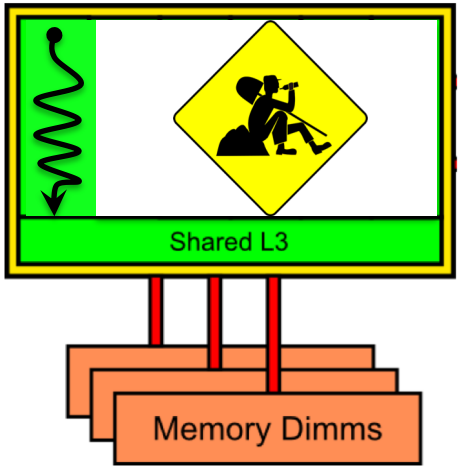
```
1: for all time step do
2:      < Set boundary conditions >
3:      for all lattice site do
4:              < Move >
5:      for all lattice site do
6:              < COLLIDE_FUSED >
7:      end for
8: end for
```

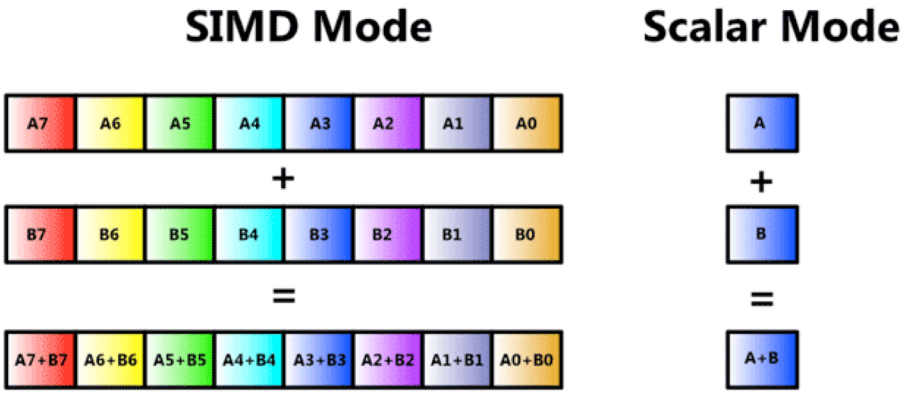
```
1: for all time step do
2:      < Set boundary conditions >
3:      for all lattice site do
4:              < FULLY_FUSED>
6:      end for
7: end for
```

Loop compression and better data locality!!

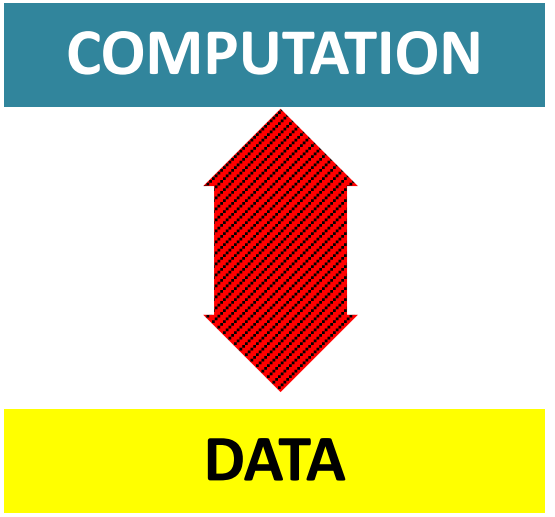
How fast is a CPU?

- CPU power is measured in number of floating point operations x second (FLOPs)
  - FLOPS = #cores x clock x $\frac{\text{FLOP}}{\text{cycle}}$



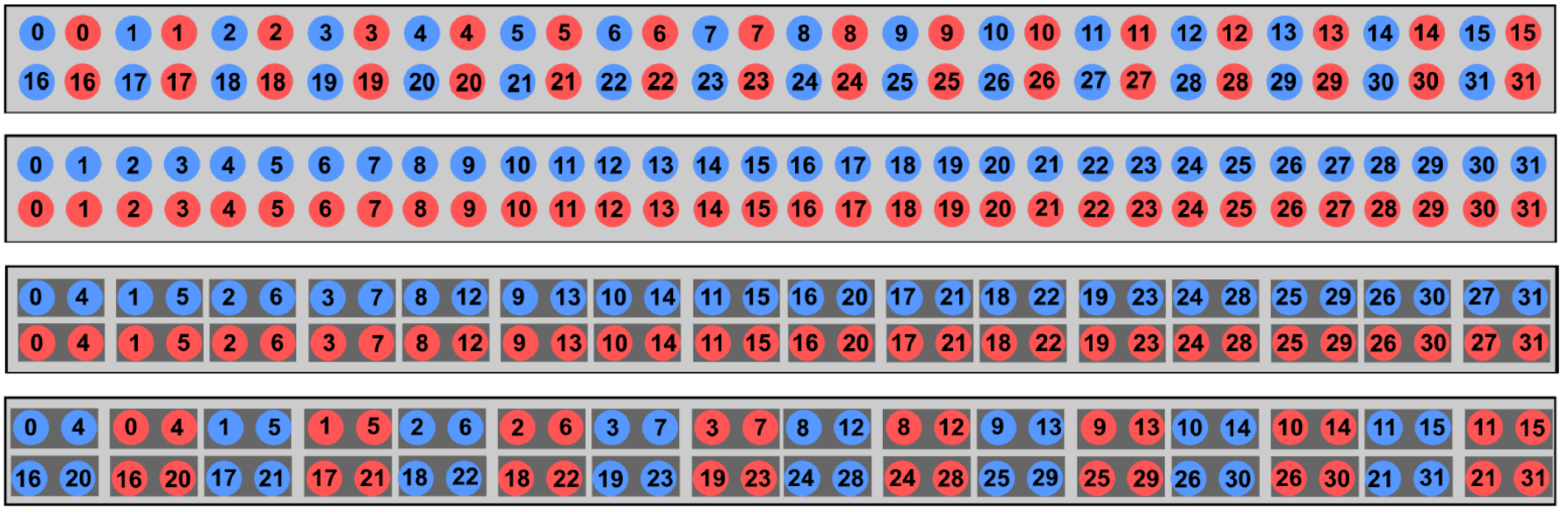


#cores	Vector Length	Freq. (GHz)	GFLOPs
1	1	1.0	1
1	16	1.0	16
10	1	1.0	10
10	16	1.0	160



Data Structure

Lattice 4 $\times$ 8 with two (blu and red) population per site.



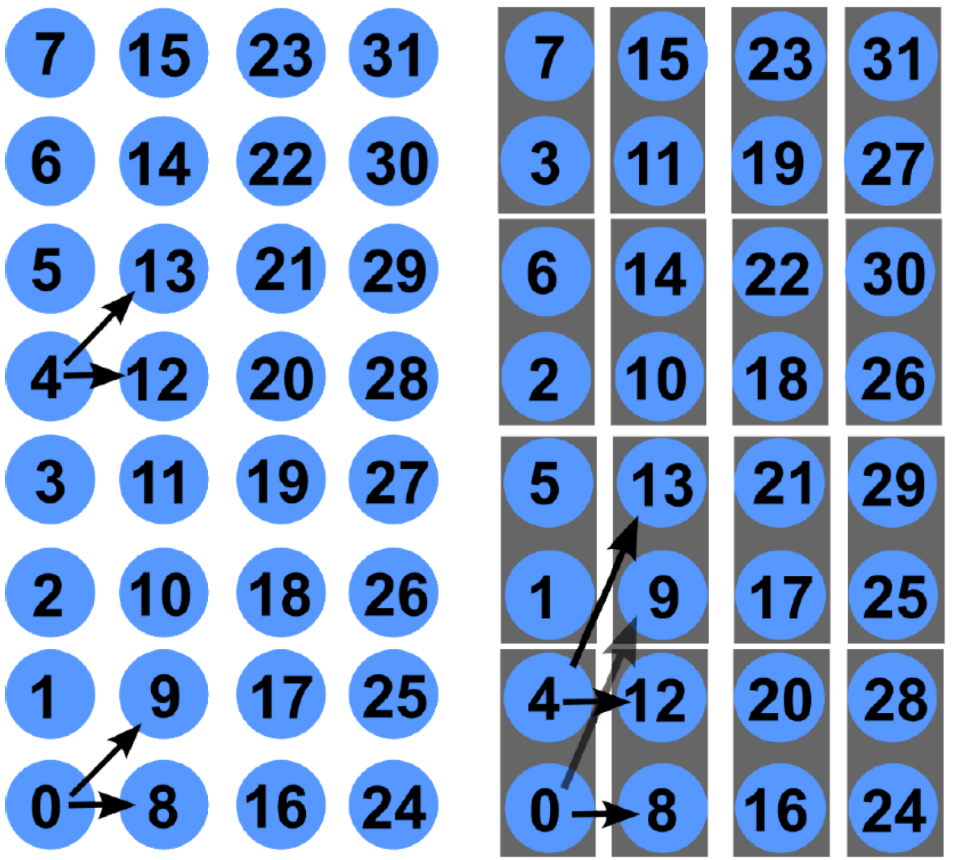
Top to bottom

- Array of Structures (AoS)
- Structure of Arrays (SoA)
- Clustered Structure of Arrays (CSoA)
- Clustered Array of Structure of Arrays (CAoSoA)

* Calore E., Gabbana A., Schifano S.F., Tripiccione R. (2018) Early Experience on Using Knights Landing Processors for Lattice Boltzmann Applications. Parallel Processing and Applied Mathematics. PPAM 2017. Lecture Notes in Computer Science, vol 10777. Springer, Cham

SoA vs CSoA

- Lattice 4×8
- machine vector size of 2-doubles:
- memory alignement is 8 Bytes
- process two sites in parallel
- $0 \rightarrow 8$ has read and write aligned
- $0 \rightarrow 9$ has read aligned and write mis-aligned
- $(0, 4) \rightarrow (8, 12)$ has read and write aligned
- $(0, 4) \rightarrow (9, 13)$ has read and write aligned
- clusters close to borders need special handling



Code analysis AoS Vs CSoA

```
void move_aos ( pop_type *  const __RESTRICT__ nxt,
                const pop_type * const __RESTRICT__ prv )
{
 int i, j, k, pp, idx0, idx0_offset;

 profile_on ( __move_aos__ );

#pragma omp parallel for private( i, j, k, idx0, idx0_offset, pp )
 for ( i = 1; i <= NX; i++ ) {
  for ( j = 1; j <= NY; j++ ) {
   for ( k = 1; k <= NZ; k++ ) {

     idx0 = IDX (i, j, k);
     for ( pp = 0; pp < NPOP; pp++ ) {

       idx0_offset = idx0 + offset_idx[ pp ];
       nxt[ idx0 ].p[ pp ] = prv[ idx0_offset ].p[ pp ];

     }
    }
  }
 }
  profile_off ( __move_aos__ );
}
```

```
# define for_each_element_v_nontemporal(_k)    \
 _Pragma("unroll")                    \
 _Pragma("vector aligned nontemporal")       \
 for(_k = 0; _k < VL; _k++)
```

```
__INLINE__ void vpopcpy_nt ( poptype * const __RESTRICT__ _pp,
                             const poptype * const __RESTRICT__ _qq )
{
  int k_vl;
  for_each_element_v_nontemporal( k_vl )  _pp[ k_vl ] = _qq[ k_vl ];
}

void move_csoa ( pop_type_csoa * const __RESTRICT__ nxt,
                 const pop_type_csoa * const __RESTRICT__ prv )
{
 int i, j, k, pp, vidx0, vidx0_offset;

 profile_on ( __move_csoa__ );

#pragma omp parallel for private( i, j, k, pp, vidx0, vidx0_offset )
 for( i = 1; i <= NX; i++ ){
   for( j = 1; j <= NY; j++ ){

     for (pp = 0; pp < NPOP; pp++) {

       for( k = 1; k <= NZOVL; k++ ){

         vidx0 = IDX_CLUSTER( i, j, k );
         vidx0_offset = vidx0 + offset_idx[ pp ];
         vpopcpy_nt( nxt->p[ pp ][ vidx0 ].c, prv->p[ pp ][vidx0_offset].c );
       }
      }
   }
 }
}profile_off (__move_csoa__);
```

Vector report analysis AoS

```
mpiicc  -DAOS -DARCH -DARCH_SKL [... ] -O3 -fp-model=precise -xCOMMON-AVX512 -D__VEC_WIDTH__=512 -D__INTEL -restrict -qopt-report-routine=move_aos -qopt-report-phase=vec -DH5Tarray_create_vers=2 -DH5Eget_auto_vers=2 -DH5Eset_auto_vers=2 -DH5Acreate_vers=2 -DH5Gcreate_vers=2 -DH5Dcreate_vers=2 -DH5Dopen_vers=2 -DH5Gopen_vers=2  -qopenmp -c lbe_performance.c -qopt-report=2

[...]

Begin optimization report for: move_aos(pop_type *const __restrict__, const pop_type *const __restrict__)

   Report from: Vector optimizations [vec]


LOOP BEGIN at lbe_performance.c(56,3)
  remark #15344: loop was not vectorized: vector dependence prevents vectorization. First dependence is shown below. Use level 5 report for\
 details
  remark #15346: vector dependence: assumed OUTPUT dependence between nxt->p[idx0][pp] (64:4) and nxt->p[idx0][pp] (64:4)

  LOOP BEGIN at lbe_performance.c(57,5)
    remark #15335: loop was not vectorized: vectorization possible but seems inefficient.

    LOOP BEGIN at lbe_performance.c(58,7)
      remark #15335: loop was not vectorized: vectorization possible but seems inefficient.
      LOOP BEGIN at lbe_performance.c(61,2)
        remark #15335: loop was not vectorized: vectorization possible but seems inefficient.
      LOOP END
    LOOP END
  LOOP END
LOOP END
```

Vector report analysis CSoA

```
Begin optimization report for: move_csoa(pop_type_csoa *const __restrict__, const pop_type_csoa *const __restrict__)
LOOP BEGIN at lbe_performance.c(1023,3)
  remark #15542: loop was not vectorized: inner loop was already vectorized
  LOOP BEGIN at lbe_performance.c(1024,7)
    remark #15542: loop was not vectorized: inner loop was already vectorized
    LOOP BEGIN at lbe_performance.c(1026,2)
      remark #15542: loop was not vectorized: inner loop was already vectorized
      LOOP BEGIN at lbe_performance.c(1028,4)
        remark #15542: loop was not vectorized: inner loop was already vectorized

        LOOP BEGIN at lbe_performance.c(993,3) inlined into lbe_performance.c(1033,6)
          remark #15388: vectorization support: reference _pp[k_vl] has aligned access   [ lbe_performance.c(994,5) ]
          remark #15388: vectorization support: reference _qq[k_vl] has aligned access   [ lbe_performance.c(994,19) ]
          remark #15412: vectorization support: streaming store was generated for _pp[k_vl]   [ lbe_performance.c(994,5) ]
          remark #15305: vectorization support: vector length 8
          remark #15427: loop was completely unrolled
          remark #15300: LOOP WAS VECTORIZED
          remark #15448: unmasked aligned unit stride loads: 1
          remark #15449: unmasked aligned unit stride stores: 1
          remark #15467: unmasked aligned streaming stores: 1
          remark #15475: --- begin vector cost summary ---
          remark #15476: scalar cost: 5
          remark #15477: vector cost: 0.370
          remark #15478: estimated potential speedup: 13.330
          remark #15488: --- end vector cost summary ---
        LOOP END
      LOOP END

      LOOP BEGIN at lbe_performance.c(1028,4)
      <Remainder>
      LOOP END
    LOOP END
  LOOP END
LOOP END
```

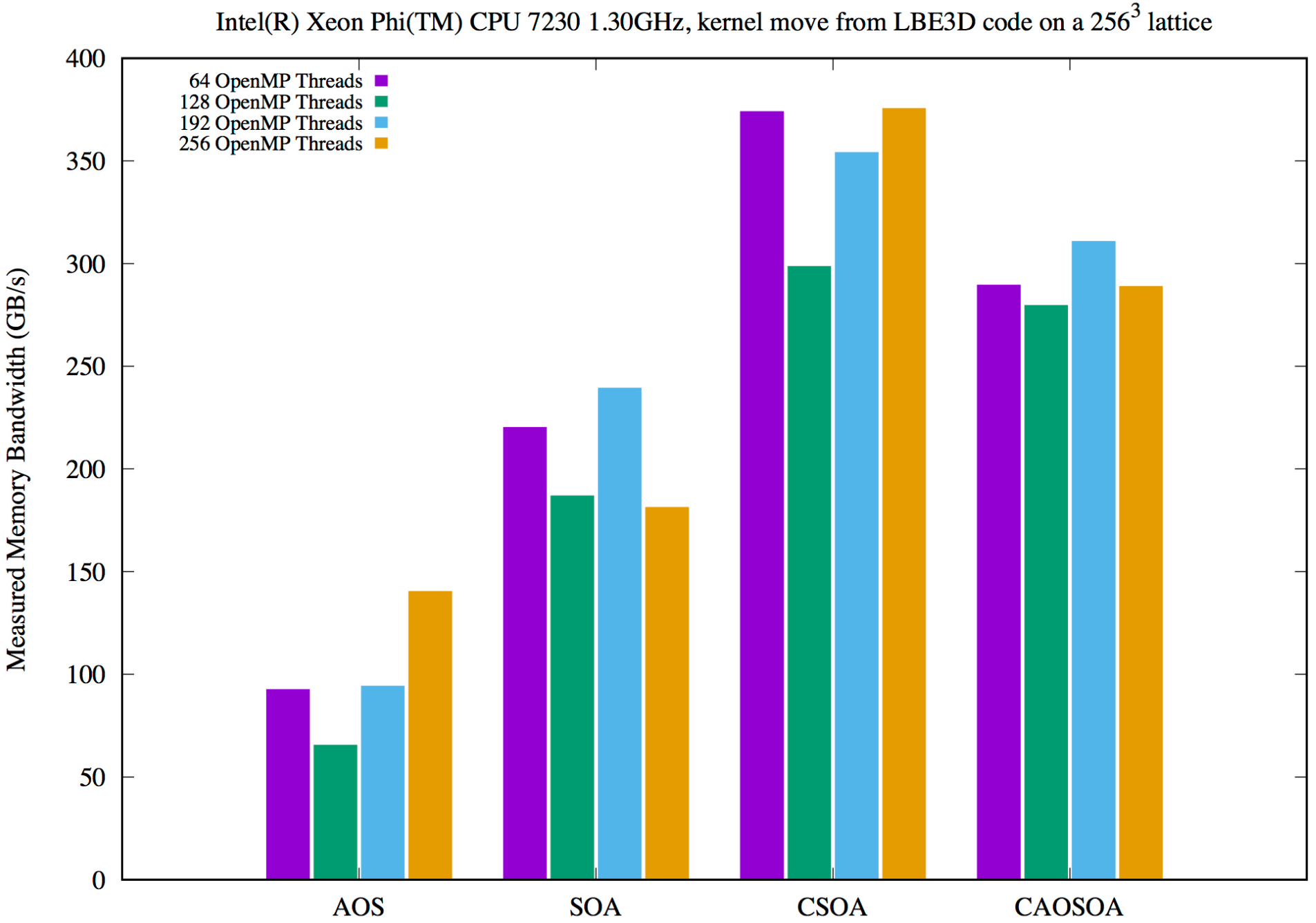
Vectorization Measurement by Vtune

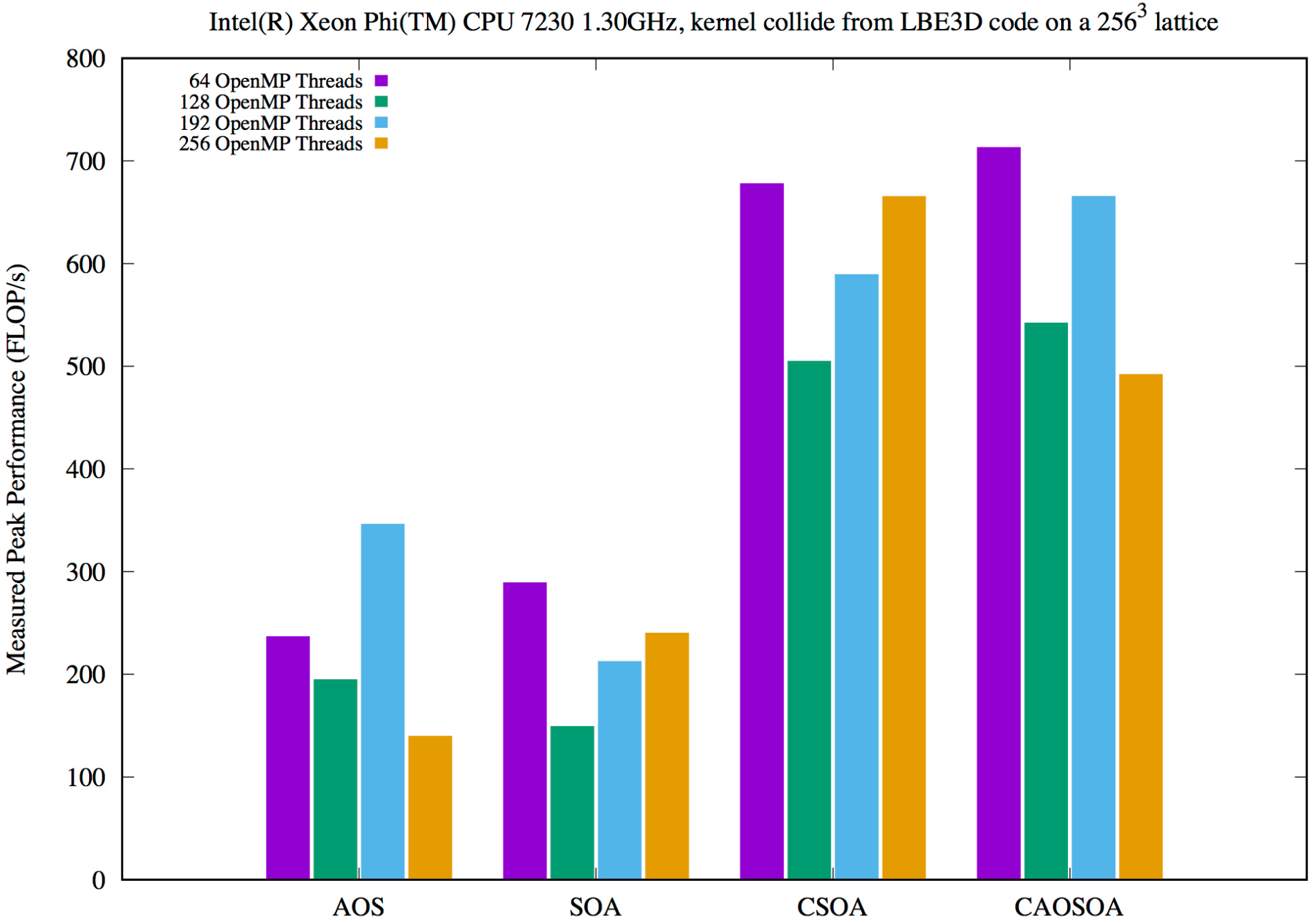
TABLE I. VPU USAGE MEASURED BY THE INTEL® VTUNE FOR FUSED LBE3D

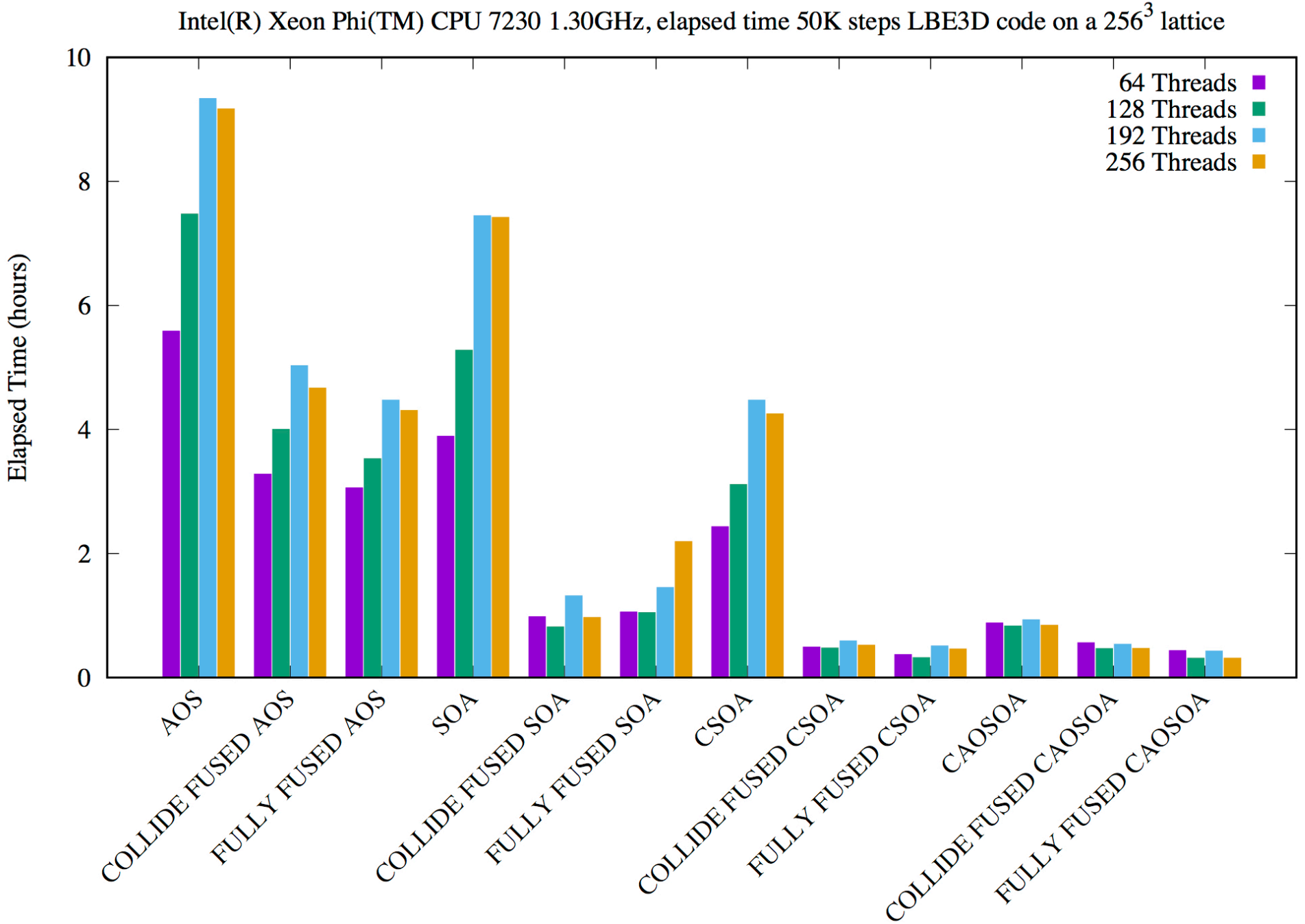
data layouts	*Vector VPU Intensity*
AoS	20%
SoA	20%
CSoA	100%
CAoSoA	100%

- Vtune's counters UOPS RETIRED.SCALAR SIMD and UOPS RETIRED.PACKED SIMD
- The final value is given by the ratio between the number of vector operations the core performed (PACKAED SIMD) to the sum of all operations (SCALAR SIMD and PACKED SIMD)

* J. Jeffers, J. Reinders, and A. Sodani, Intel Xeon Phi Processor High Performance Programming. Morgan Kaufmann, Jun. 2016, ISBN: 978-0-12-809194-4.


Intel(R) Xeon Phi(TM) CPU 7230 1.30GHz, kernel move from LBE3D code on a 256^3 lattice
Measured Memory Bandwidth (GB/s)
0
50
100
150
200
250
300
350
400
64 OpenMP Threads
128 OpenMP Threads
192 OpenMP Threads
256 OpenMP Threads
AOS
SOA
CSOA
CAOSOA


Intel(R) Xeon Phi(TM) CPU 7230 1.30GHz, kernel collide from LBE3D code on a 256^3 lattice
64 OpenMP Threads
128 OpenMP Threads
192 OpenMP Threads
256 OpenMP Threads
Measured Peak Performance (FLOP/s)
0
100
200
300
400
500
600
700
800
AOS
SOA
CSOA
CAOSOA

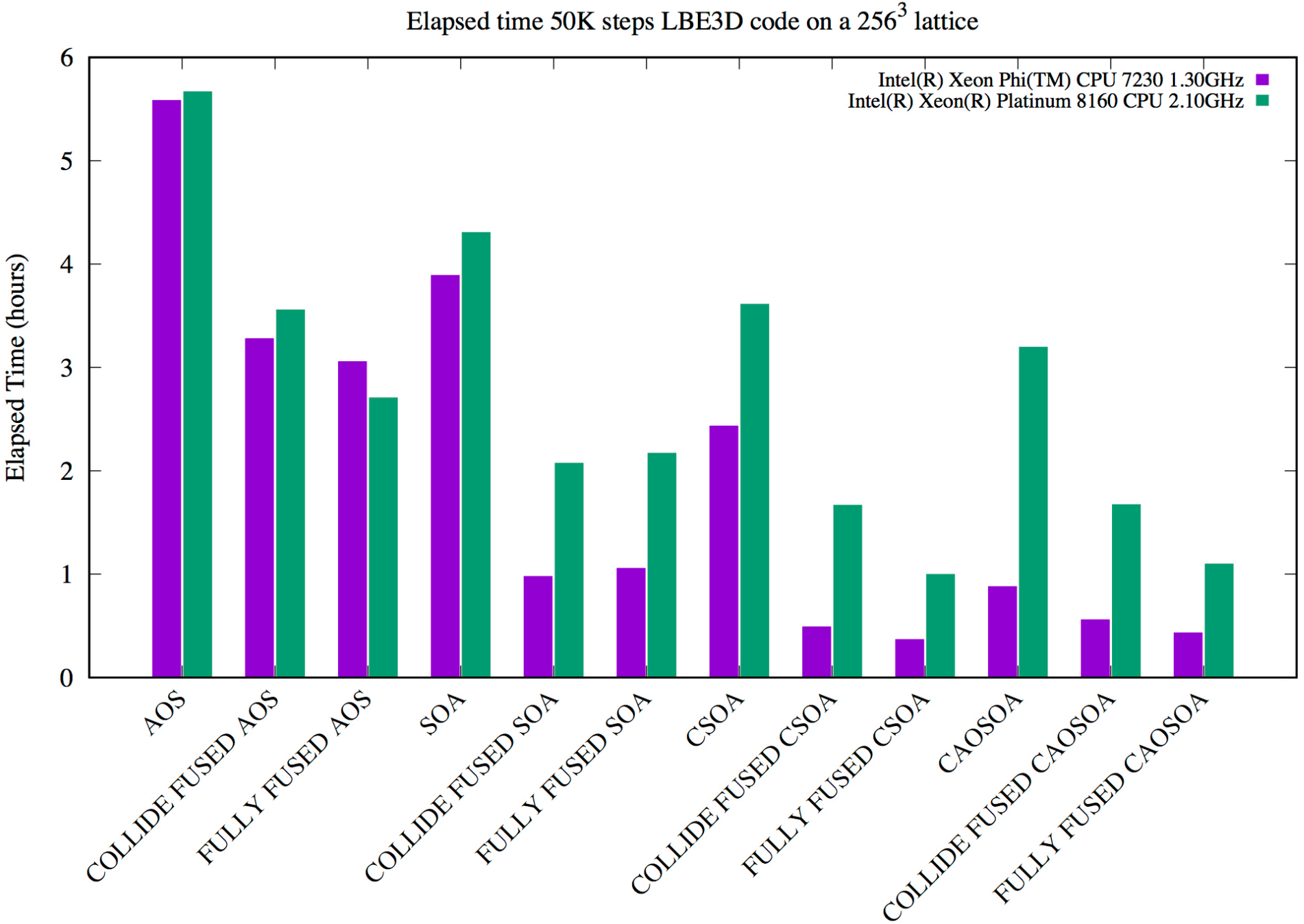

Intel(R) Xeon Phi(TM) CPU 7230 1.30GHz, elapsed time 50K steps LBE3D code on a 256^3 lattice
Elapsed Time (hours)
0
2
4
6
8
10
64 Threads
128 Threads
192 Threads
256 Threads
AOS
COLLIDE FUSED AOS
FULLY FUSED AOS
SOA
COLLIDE FUSED SOA
FULLY FUSED SOA
CSOA
COLLIDE FUSED CSOA
FULLY FUSED CSOA
CAOSOA
COLLIDE FUSED CAOSOA
FULLY FUSED CAOSOA

Ghost cells exchange optimization

```
void pack_pop_csoa_X_bottom( const pop_type_csoa * const __RESTRICT__ f, vpoptype * const
__RESTRICT__ buffer_send )
{

#pragma omp parallel
  {
    size_t j, k, src, dest, pp;
    size_t count = 0;

    for ( pp = 0; pp < NPOP; pp++ ) {

      if( cx[pp] < 0 ){

#pragma omp for
        for( j = 0; j < NYP2 ; j++ ){
          for( k = 0; k < NZP2OVL; k++ ){

            src  = IDX_CLUSTER( 1, j, k );
            dest = count + ( j * NZP2OVL ) + k;

            vpopcpy_nt( buffer_send[ dest  ].c, f->p[ pp ][ src ].c );
          }
        }

        count += NZP2OVL * NYP2;
      }
    }
  }
}
```

Ghost cells exchange optimization

```
// CSOA Halos exchange along X

  MPI_Comm_size( MPI_COMM_ALONG_X, &size_comm );

  if( size_comm > 1 ){


    pack_pop_csoa_X_up( f_out, buffer_send );

    MPI_Sendrecv(  buffer_send,  5 * NYP2 * NZP2OVL * VL, MPI_DOUBLE, pxp, 120,
                   buffer_recv, 5 * NYP2 * NZP2OVL * VL, MPI_DOUBLE, pxm, 120,
                   MPI_COMM_ALONG_X, &status1);

    unpack_pop_csoa_X_up( f_out, buffer_recv );


    pack_pop_csoa_X_bottom( f_out, buffer_send );

    MPI_Sendrecv(  buffer_send,  5 * NYP2 * NZP2OVL * VL, MPI_DOUBLE, pxm, 121,
                   buffer_recv, 5 * NYP2 * NZP2OVL * VL, MPI_DOUBLE, pxp, 121,
                   MPI_COMM_ALONG_X, &status1);

    unpack_pop_csoa_X_bottom( f_out, buffer_recv );

  }
```

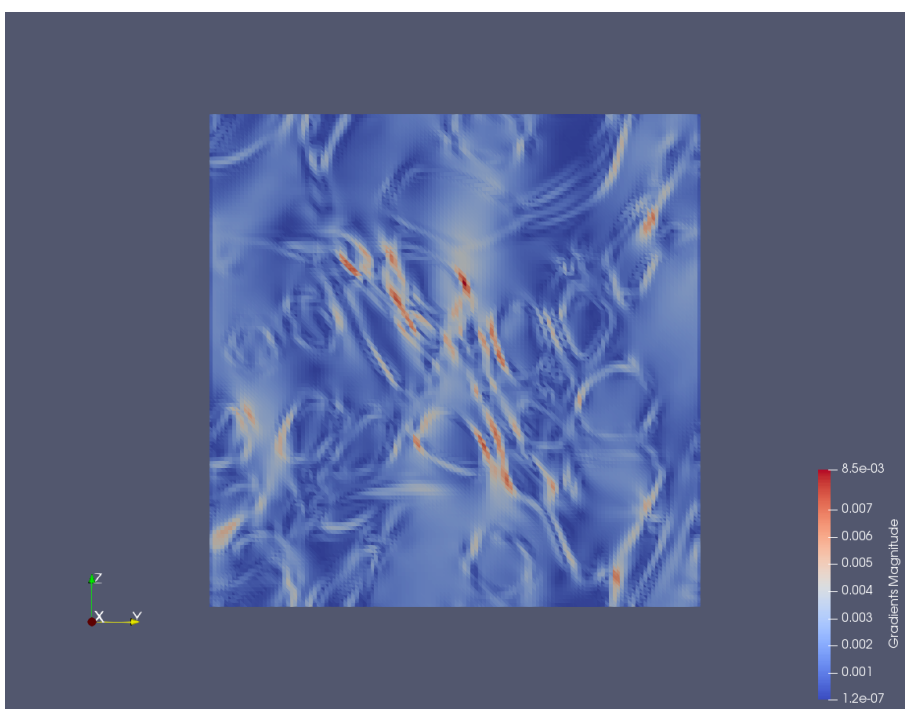

Elapsed time 50K steps LBE3D code on a 256^3 lattice
Elapsed Time (hours)
0
1
2
3
4
5
6
Intel(R) Xeon Phi(TM) CPU 7230 1.30GHz
Intel(R) Xeon(R) Platinum 8160 CPU 2.10GHz
AOS
COLLIDE FUSED AOS
FULLY FUSED AOS
SOA
COLLIDE FUSED SOA
FULLY FUSED SOA
CSOA
COLLIDE FUSED CSOA
FULLY FUSED CSOA
CAOSOA
COLLIDE FUSED CAOSOA
FULLY FUSED CAOSOA

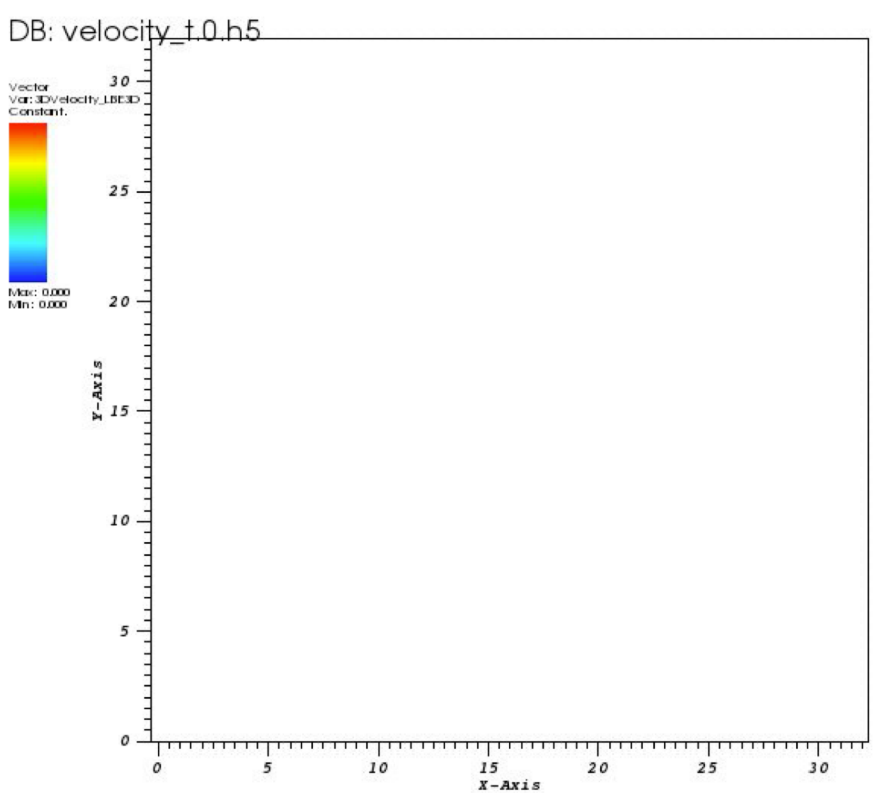
Possible Impacts

- The large produced database will allow to explore the process of turbulent emulsification in great details
- The simulation project is expected to allow to explore, for the first time ever in 3d, the statistical physics of avalanche
- Development of phenomenological macroscopic models for the behavior of complex fluid, potentially of great relevance for industrial processes
- Development of new tools and experience for LB simulation at the Exascale

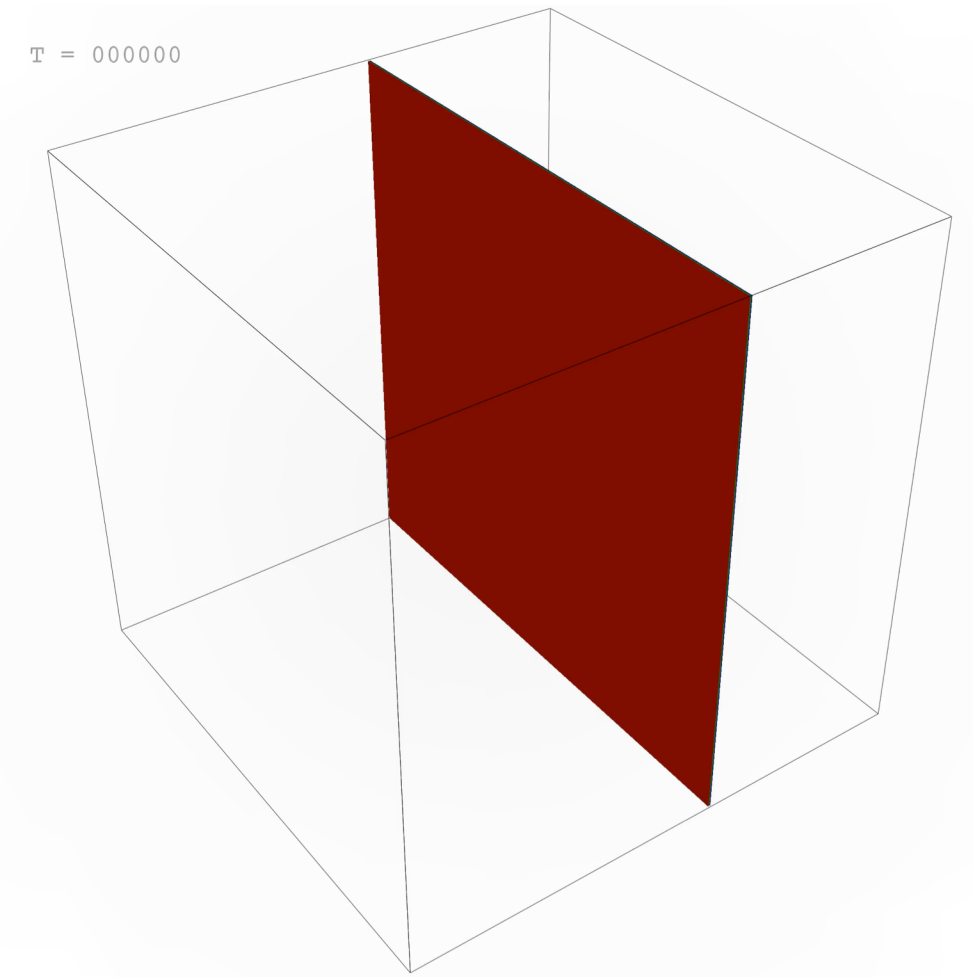
The I/O Challenge

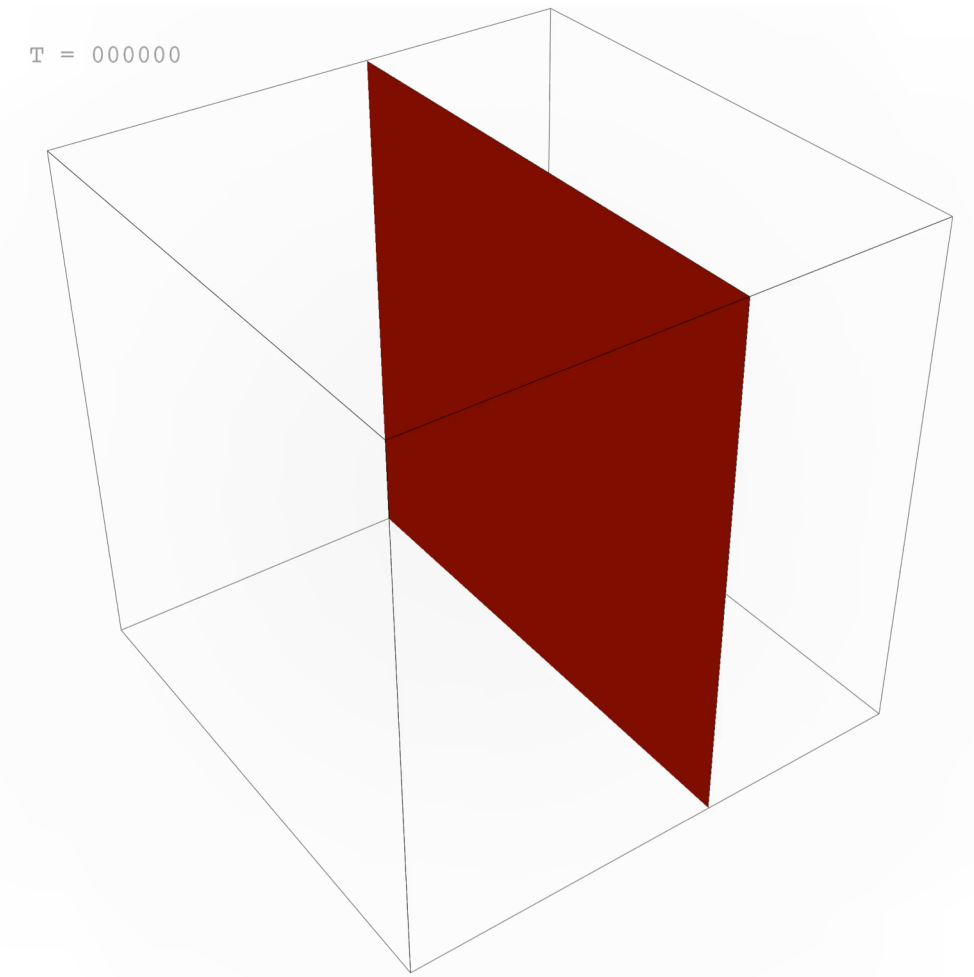
- Up to 1024^3 lattices are needed for this simulation
- At this scale every step saved into disk of a single physical quantity is of about 8.5GBs (about ~45 GBs x I/O step if only considering densities and velocities)
- Lattice files (multicomponent) for check point restart are about 350GBs
- Densities files are essential to make high-resolution visualization of the system. Velocities as well density files are to make velocities spectrums, statistical analysis and understand the physical properties of the sy





The First Simulations

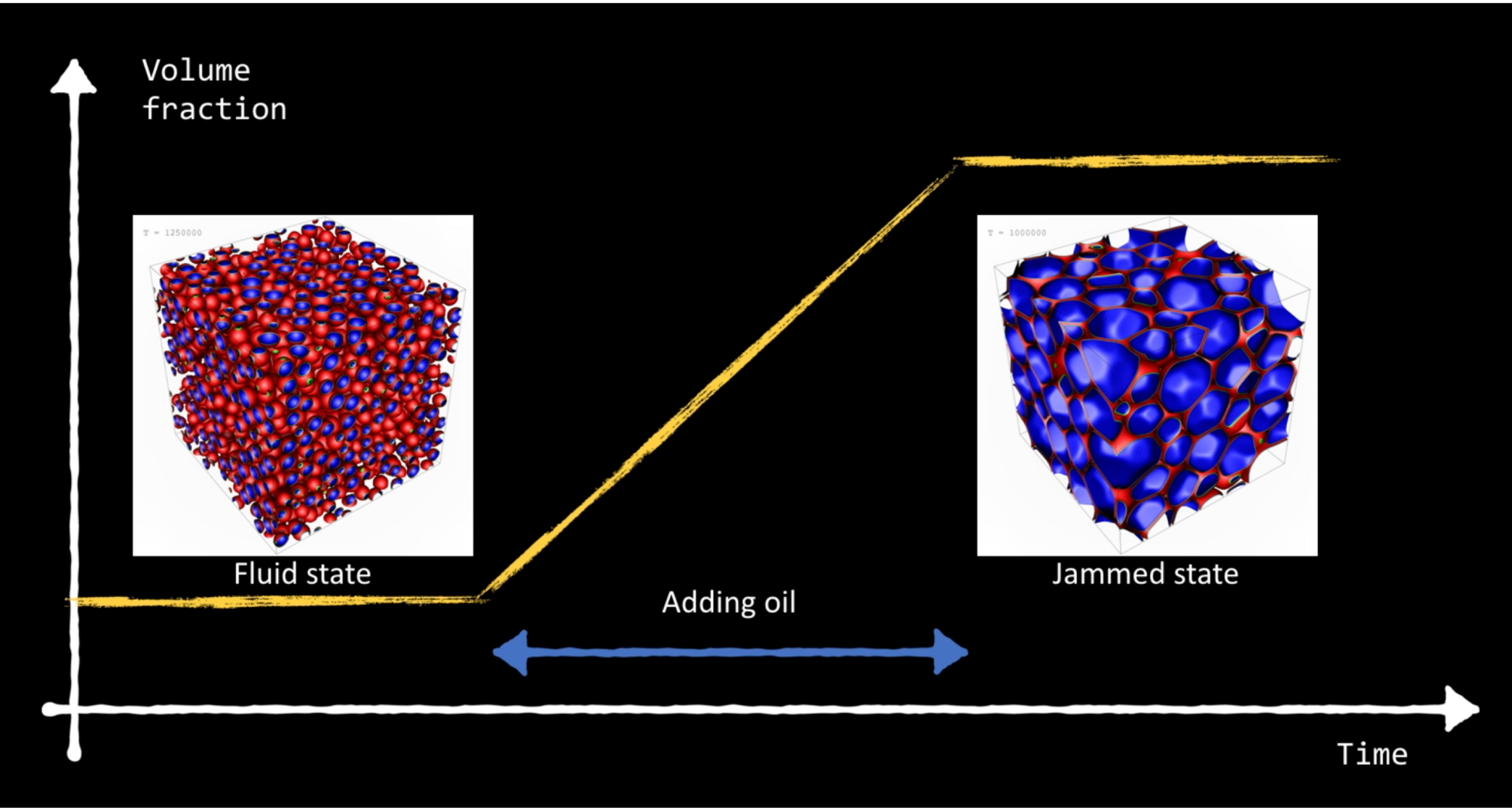




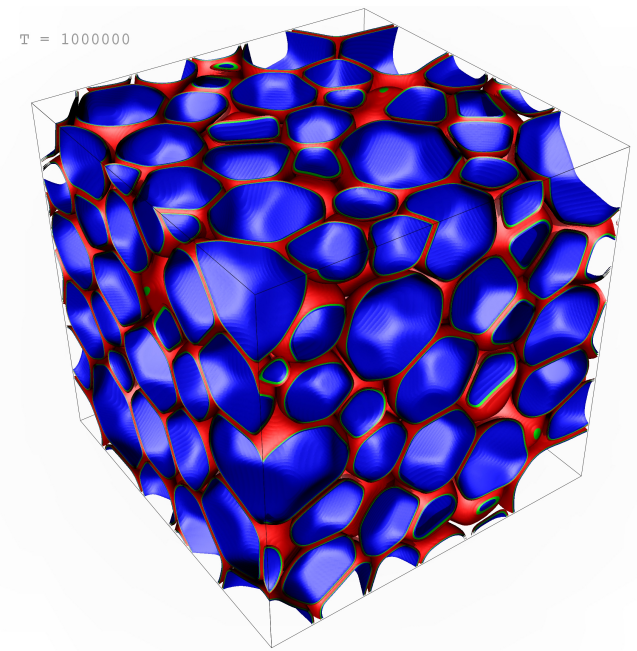
The Rendering Challenge

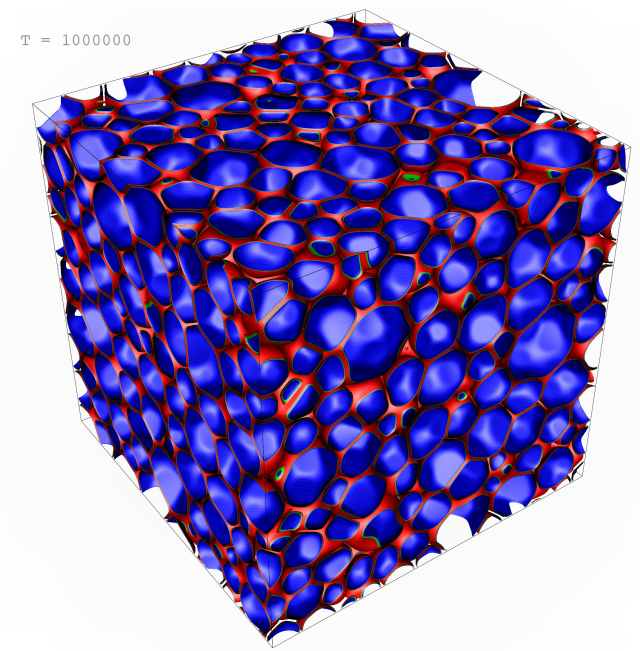
- Every picture is generated using visit (via a python script)
- We need to load hundreds of file of several GBs to create images of about tens of MBs
- Already at 1024^3 each single files takes about 20 minutes to be rendered on Marconi
- The rendering of all the pictures is performed in parallel at once on hundreds of nodes (task farming)
- However, we would need to adopt special approaches for parallel distributed rendering 2048^3 file
- And a really large memory node is needed to build the video

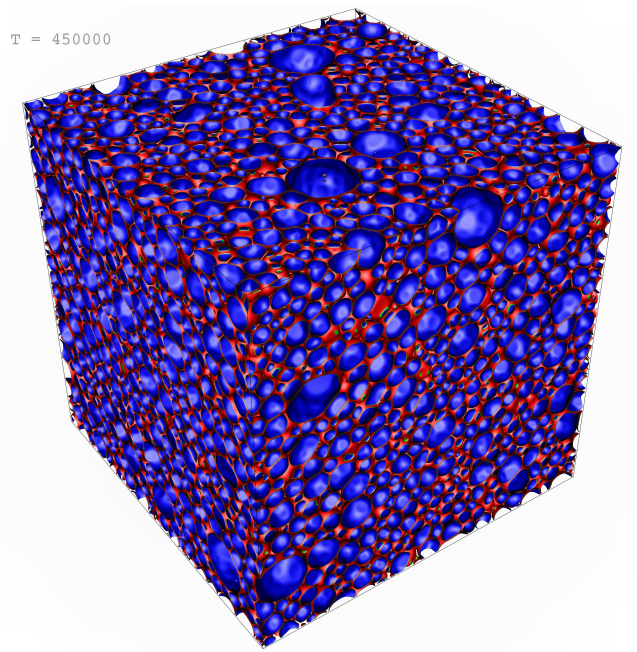
How to make turbulent dense emulsions?!



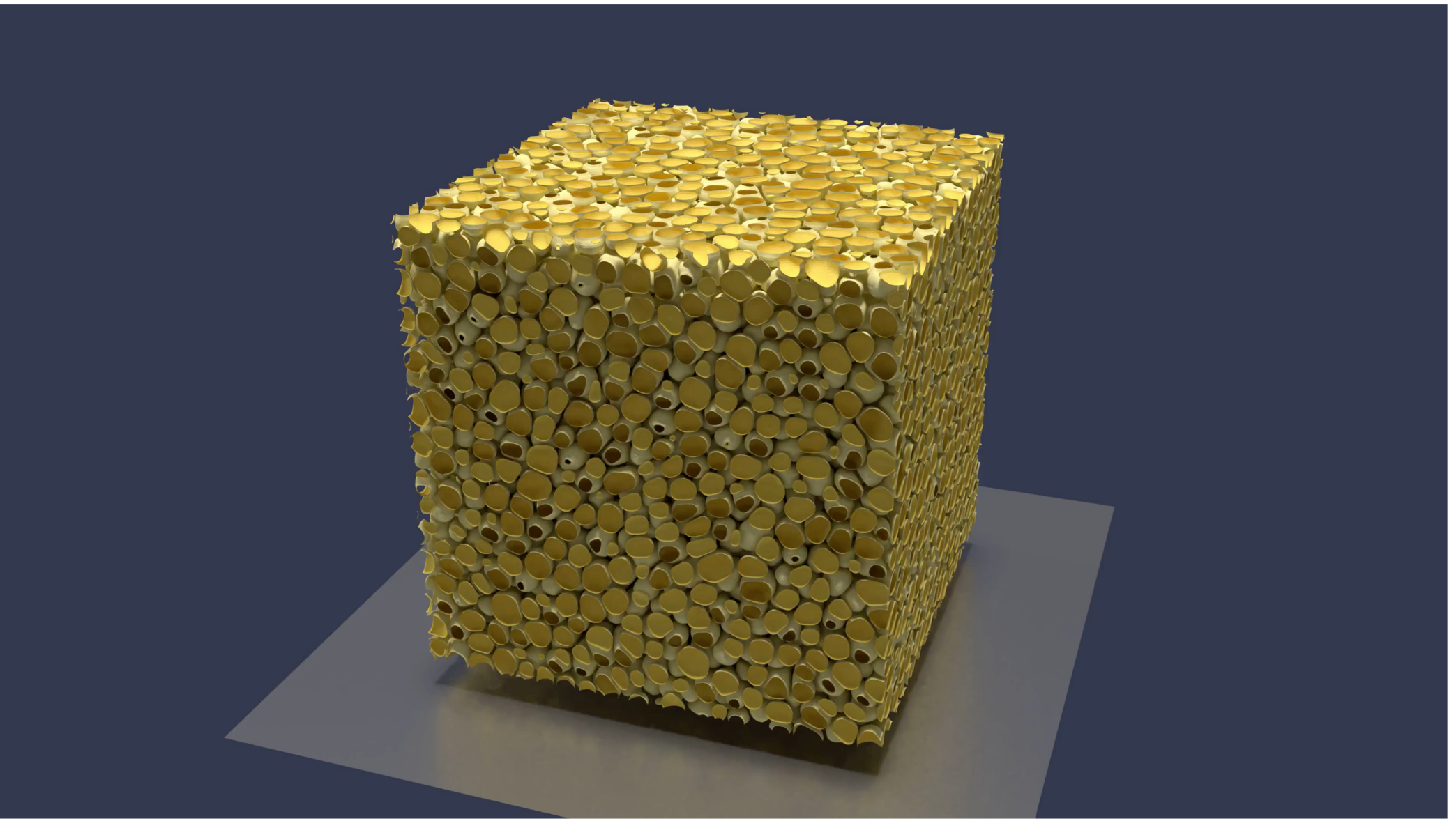
How to make turbulent dense emulsions?!







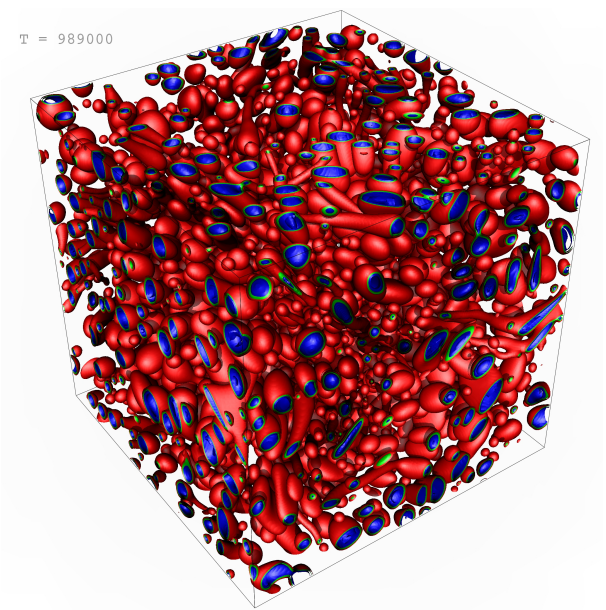
How to avoid phase inversion?

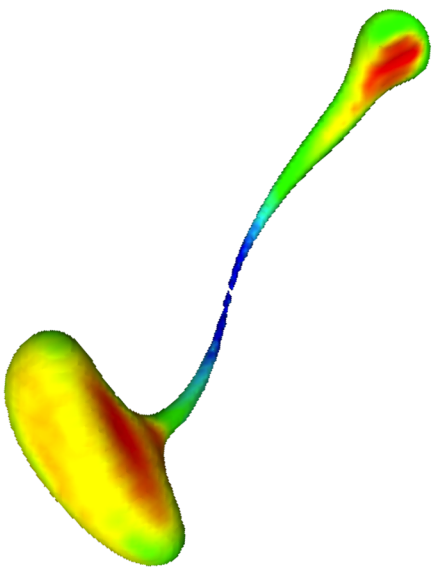


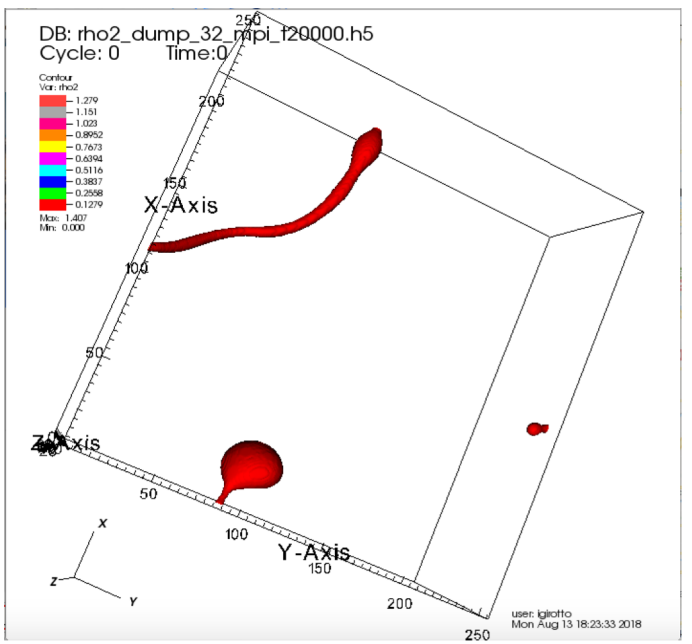
Physical Statistics: the opportunity

- We aim to track all simulations droplets every given timesteps to collect a raw database of physical statistics of what is happening during the emulsion (only post run analysis is not possible at this scale)

- Droplets could be really big, of unpredictable shape and, moreover, distributed across the grid of processes







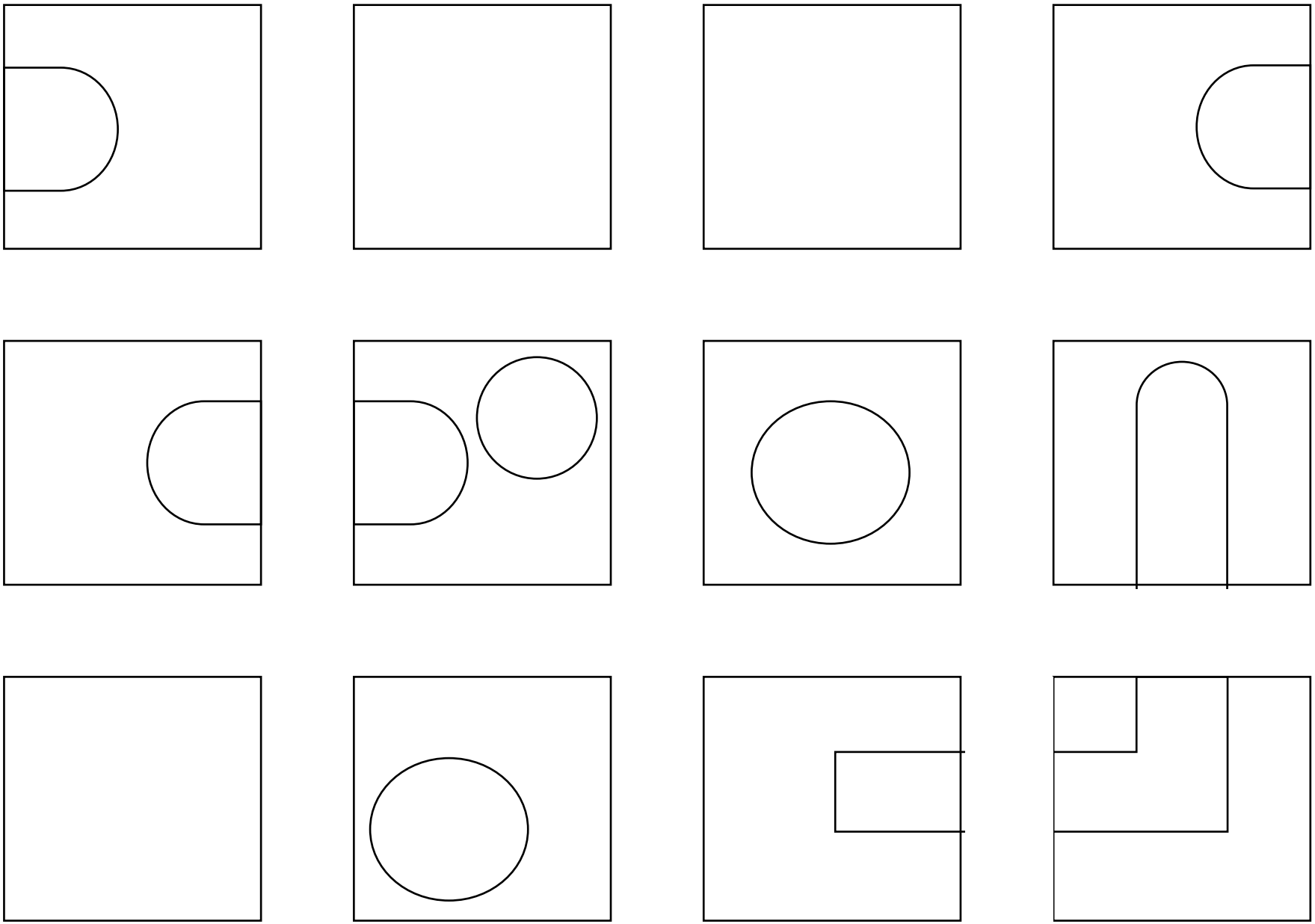
Physical Statistics: the challenge

- identify all droplets' chucks in a density grid
- identify whether a chunk is a single droplet or a chunk of a droplet composed of multiple chunks
- droplet chunks can be spread among the processes
- PBC are applied

Process (X, Y, Z)

Process (X^1, Y^1, Z^1)

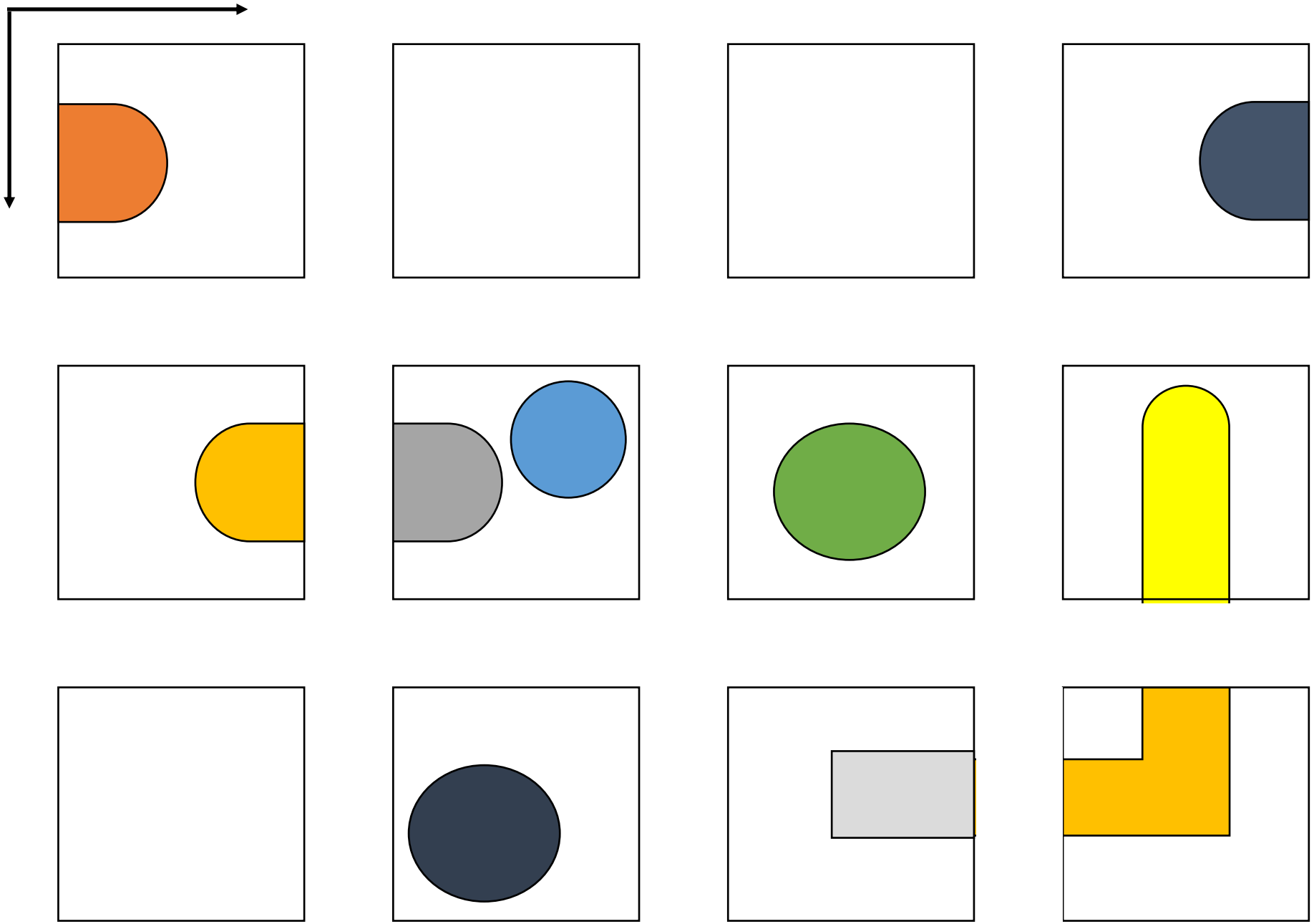
Parallel Flood Fill /1



Parallel Flood Fill Description /1

```
Set <- density lattice

//First Phase: all droplets point are somehow colored
For all element ∈ Set do
      If element >= threshold then
            If( check_all_neighbors_colors == 0) Then
                  color <- create_new_color
                  flag[ element ] <- color
                  flag_all_neighbors_above_threshold
            Else
                  color <- find_minimum_colorId_among_neighbors
                  flag[ element ] <- color
                  flag_all_neighbors_above_threshold(color)

End For
//assign to all distributed colors a unique and progressive Id
Parallel_Global_Color_Naming
```

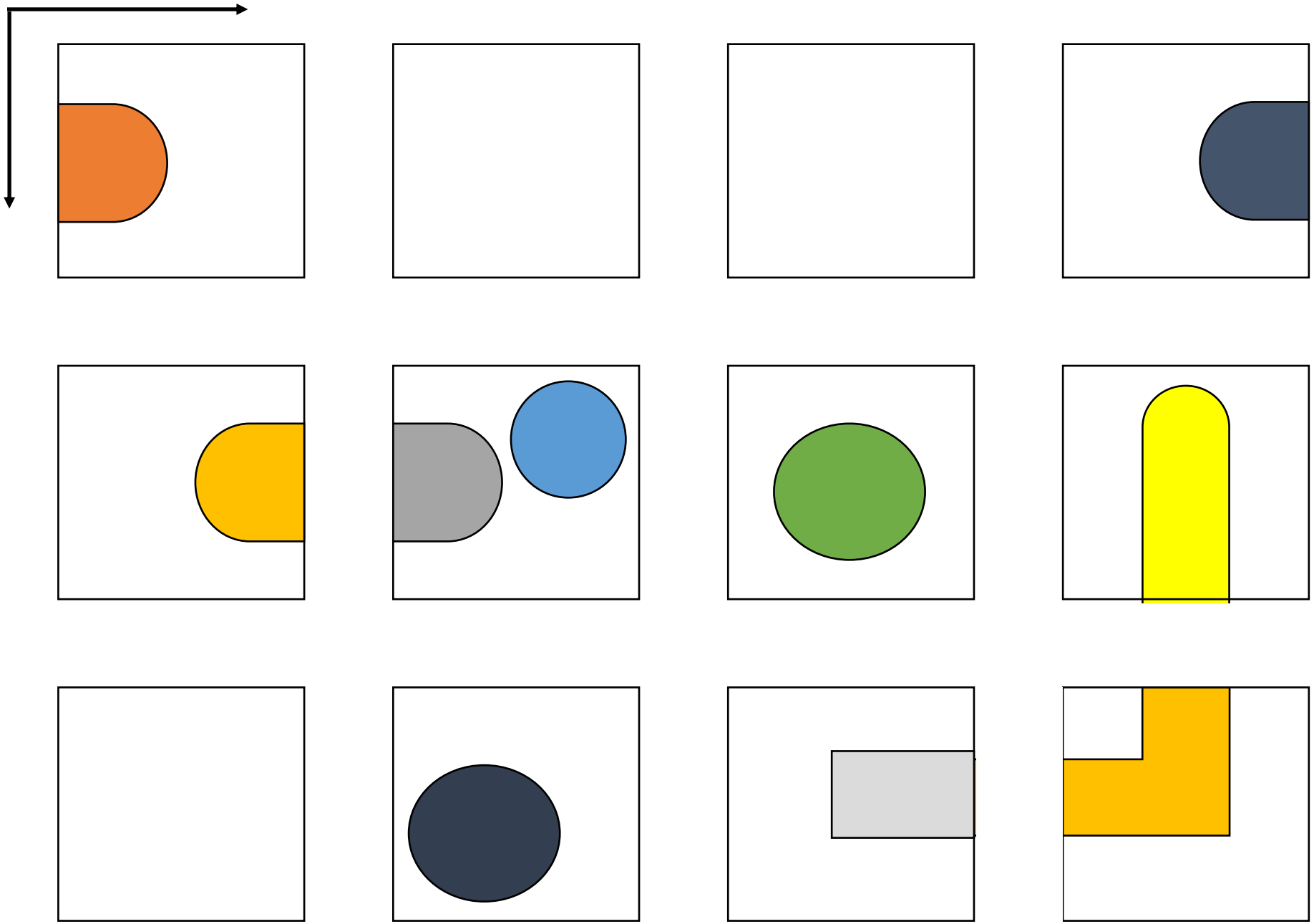
Parallel Flood Fill /2



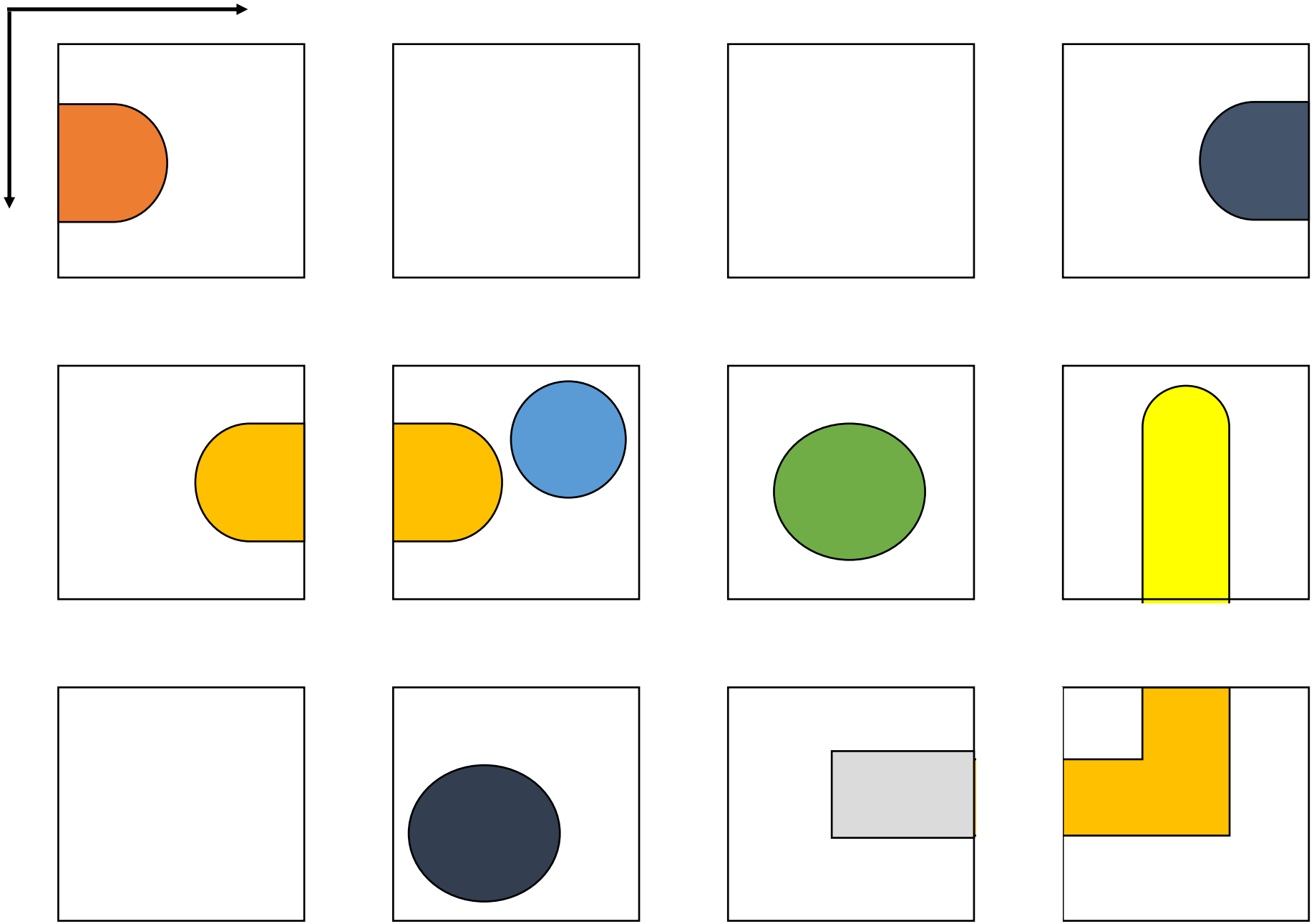
Parallel Flood Fill Description /2

```
//Second Phase: propagate the colors among the processes
Set <- flag // colors lattice

Do      check <- 0
    For all element ∈ flag do
            If ( element is a color ) Then
                    color <- find_minimum_colorId_among_neighbors
                    flag_all_neighbors_elements(color)
                    check <- 1
        End For
        For all element ∈ flag do in reverse order
            If ( element is a color ) Then
                    color <- find_minimum_colorId_among_neighbors
                    flag_all_neighbors_elements(color)
                    check <- 1
        End For
        All_reduce_check_value_among_processes
While( check != 0)
```

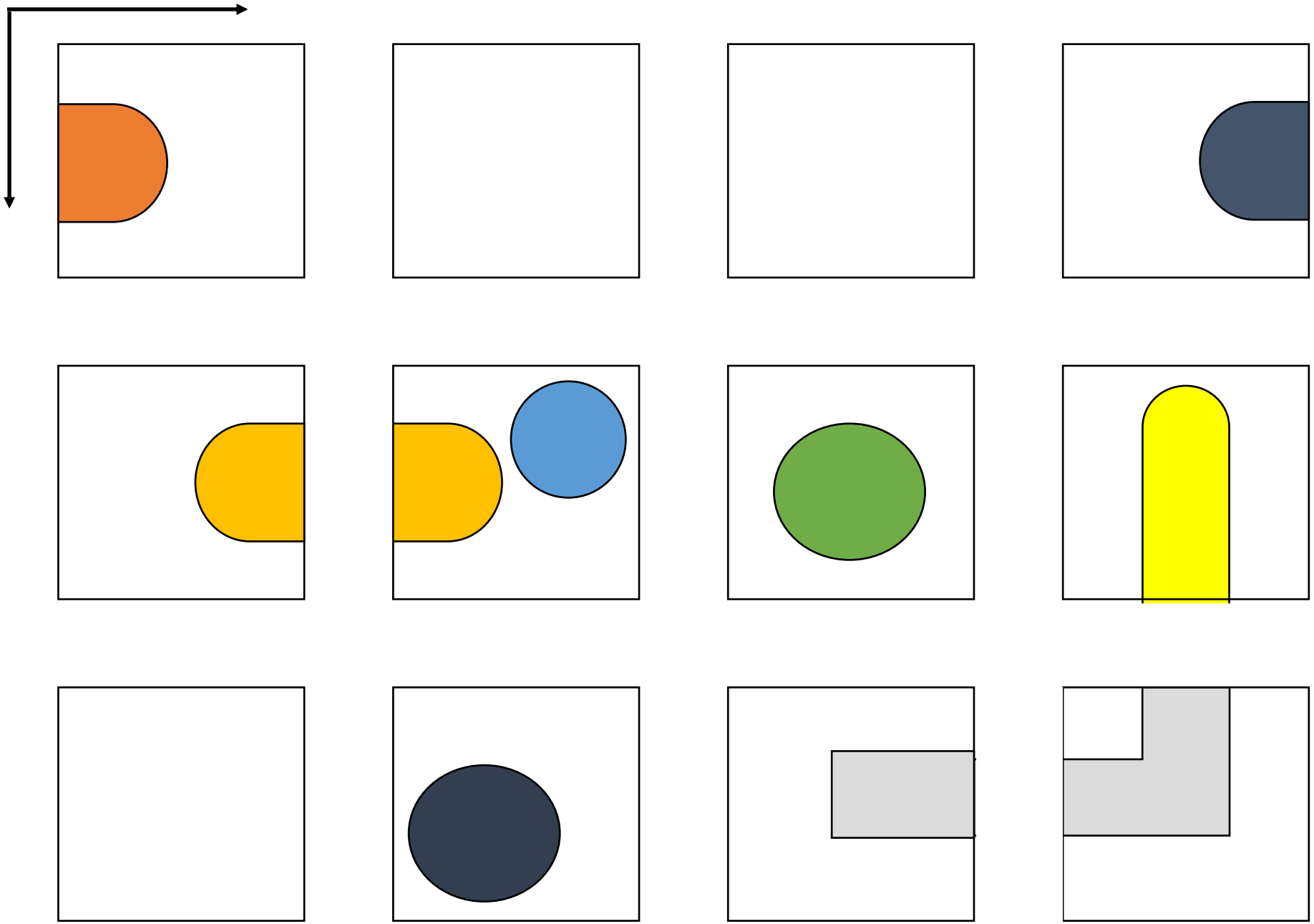
Parallel Flood Fill /2



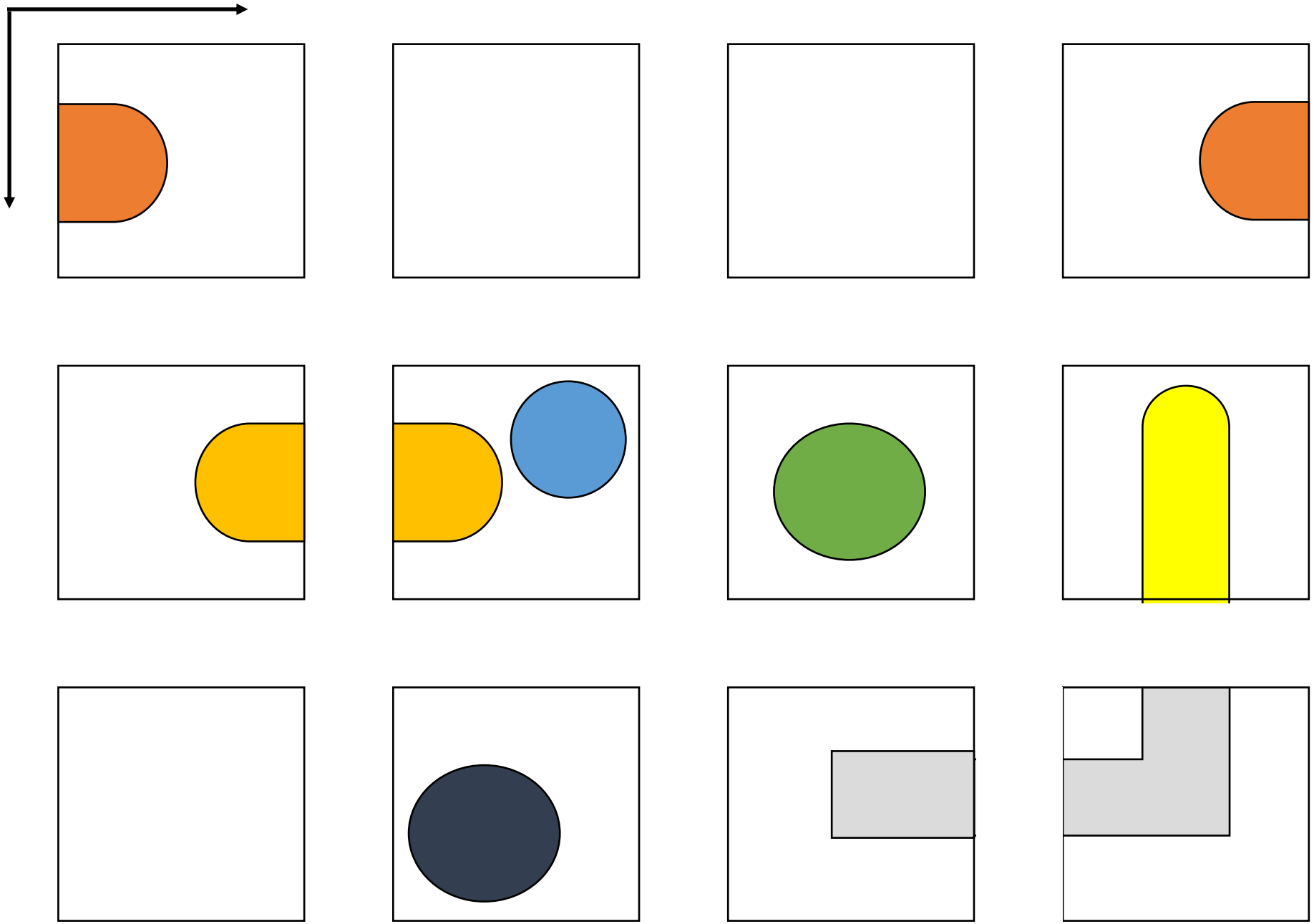
Parallel Flood Fill /3



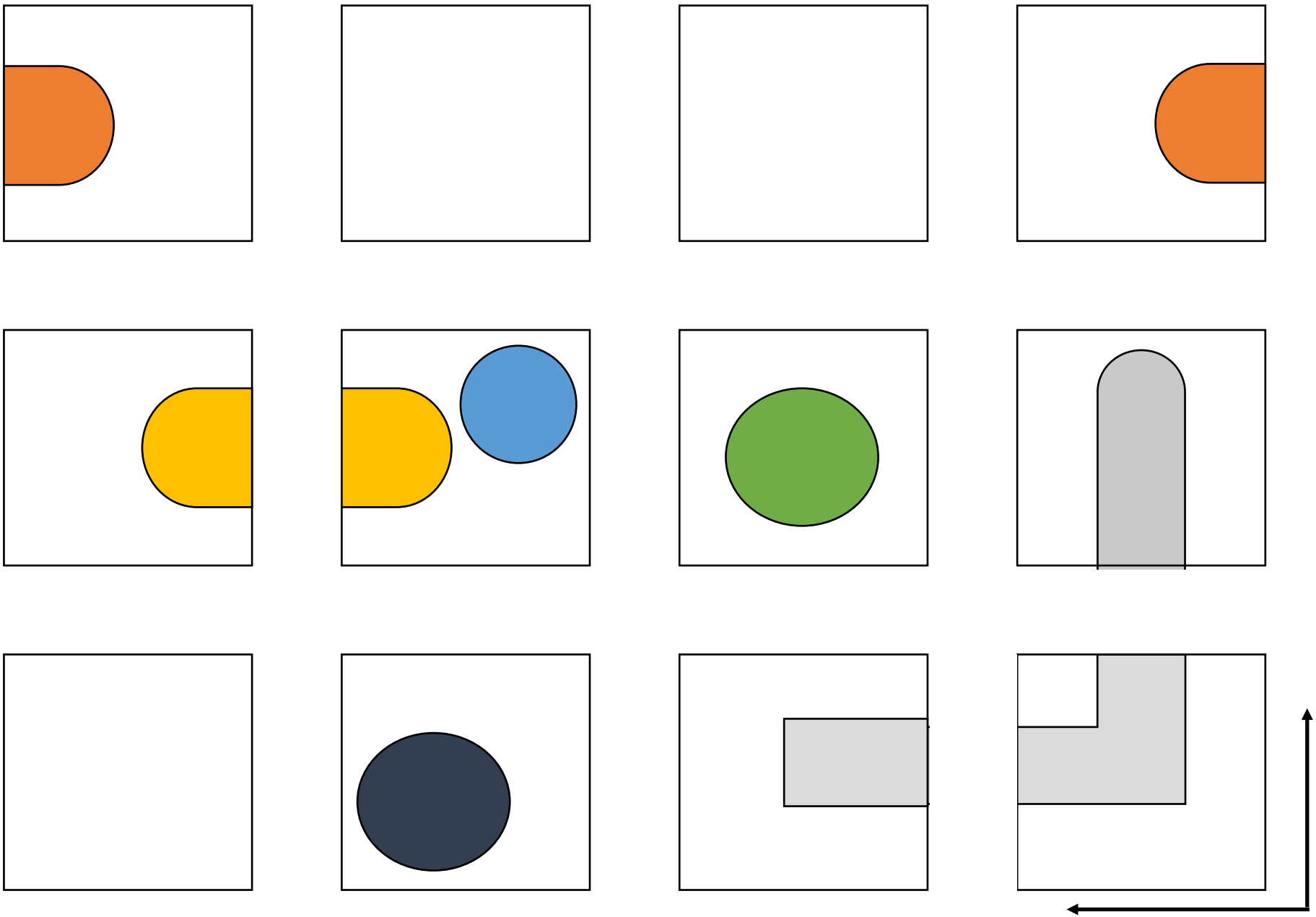
Parallel Flood Fill /4



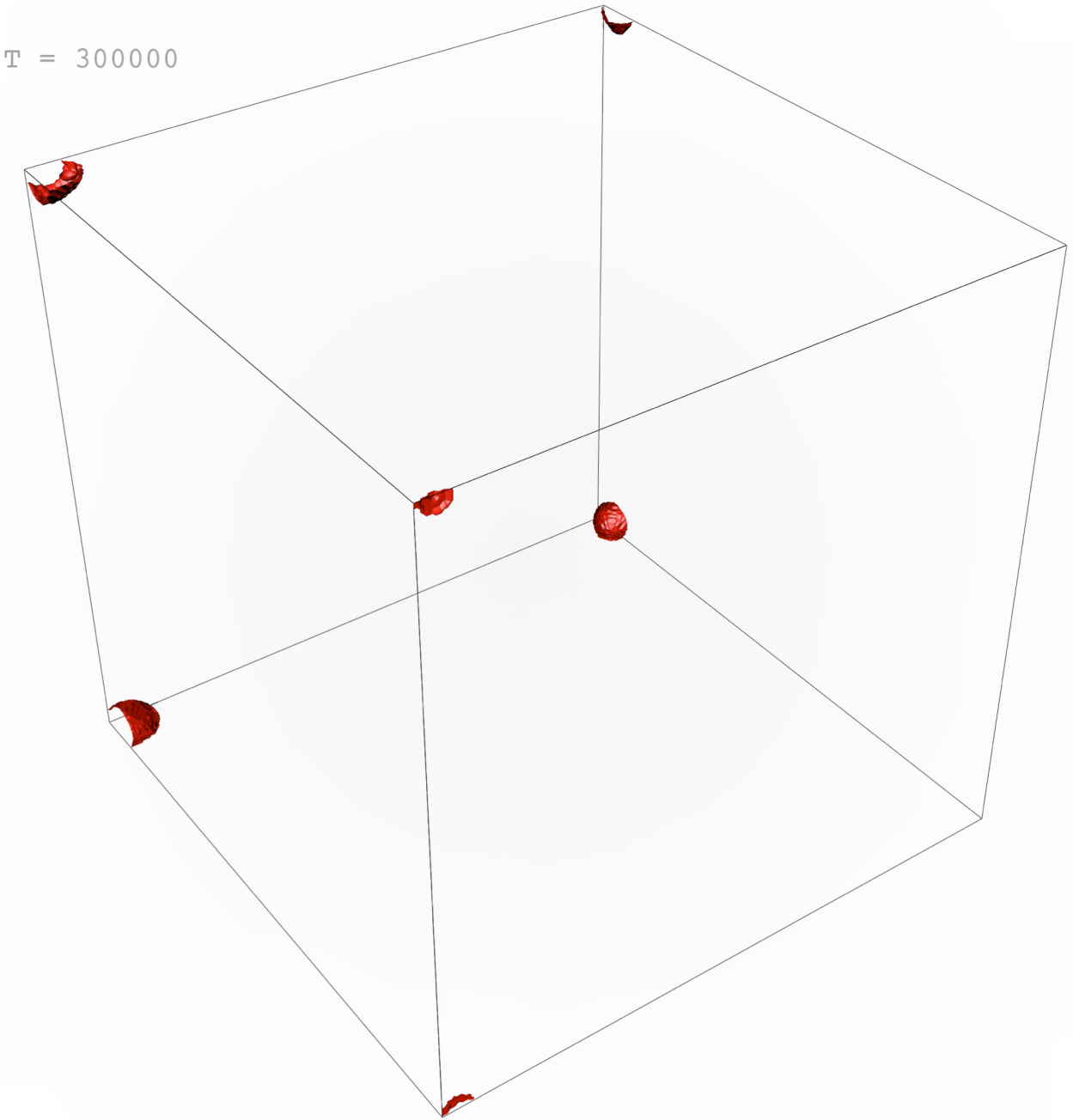
Parallel Flood Fill /5



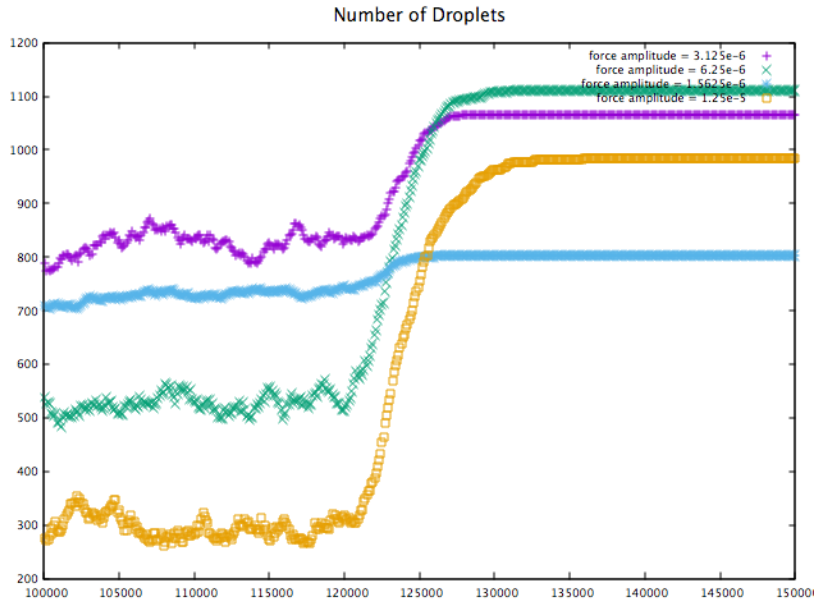
Parallel Flood Fill /6

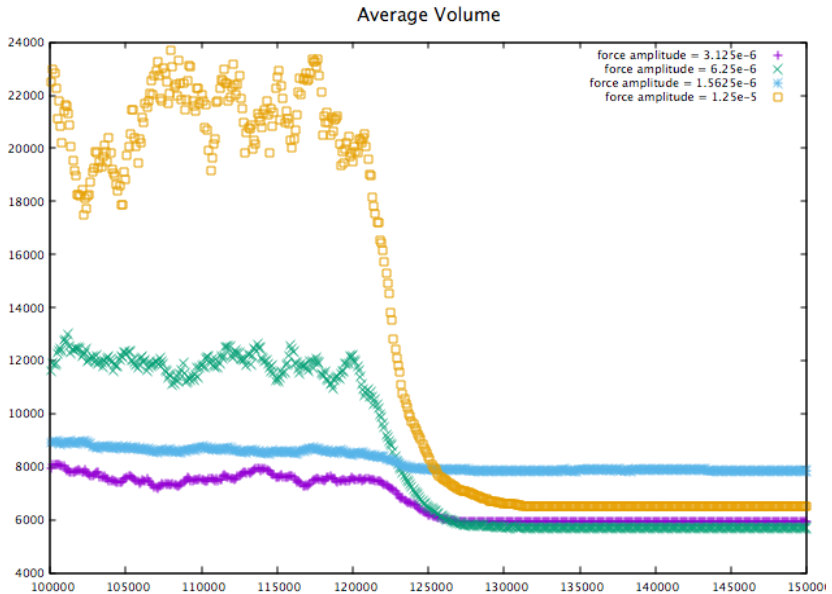


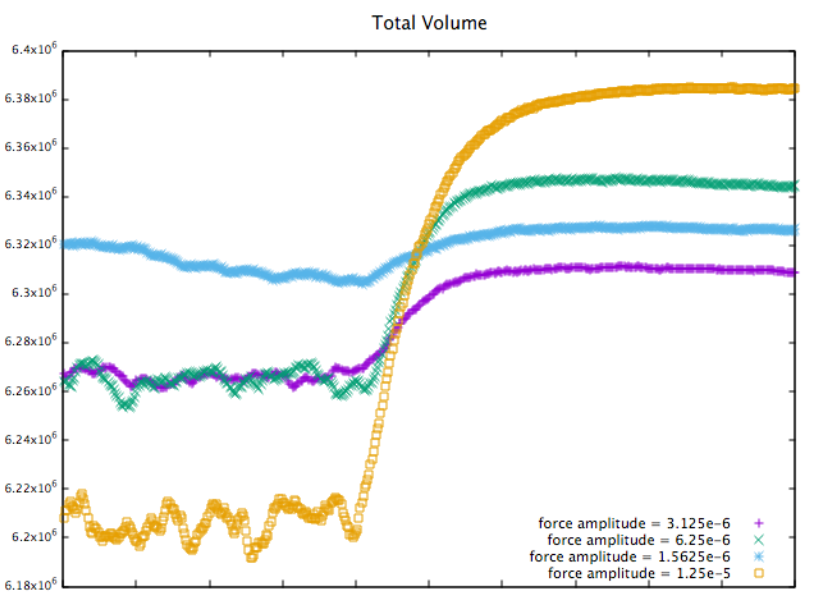
Physical Statistics

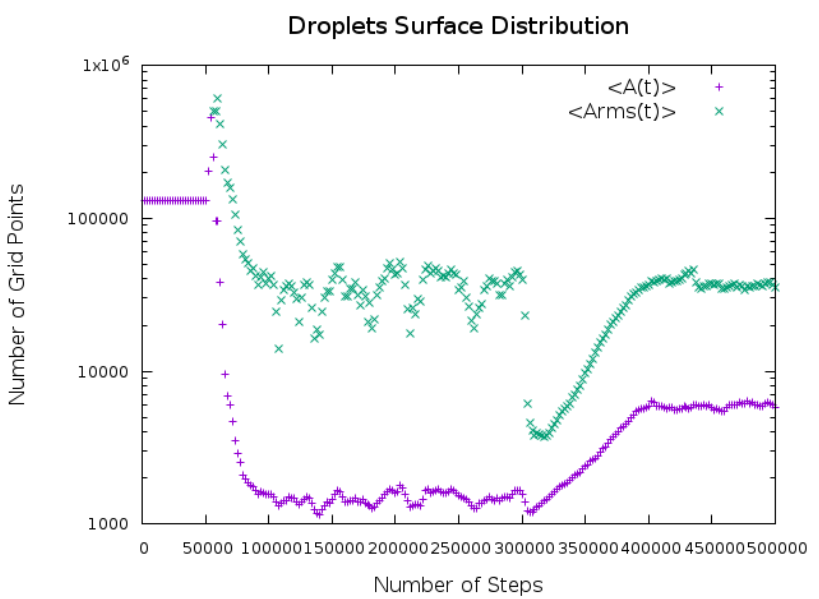


… and more

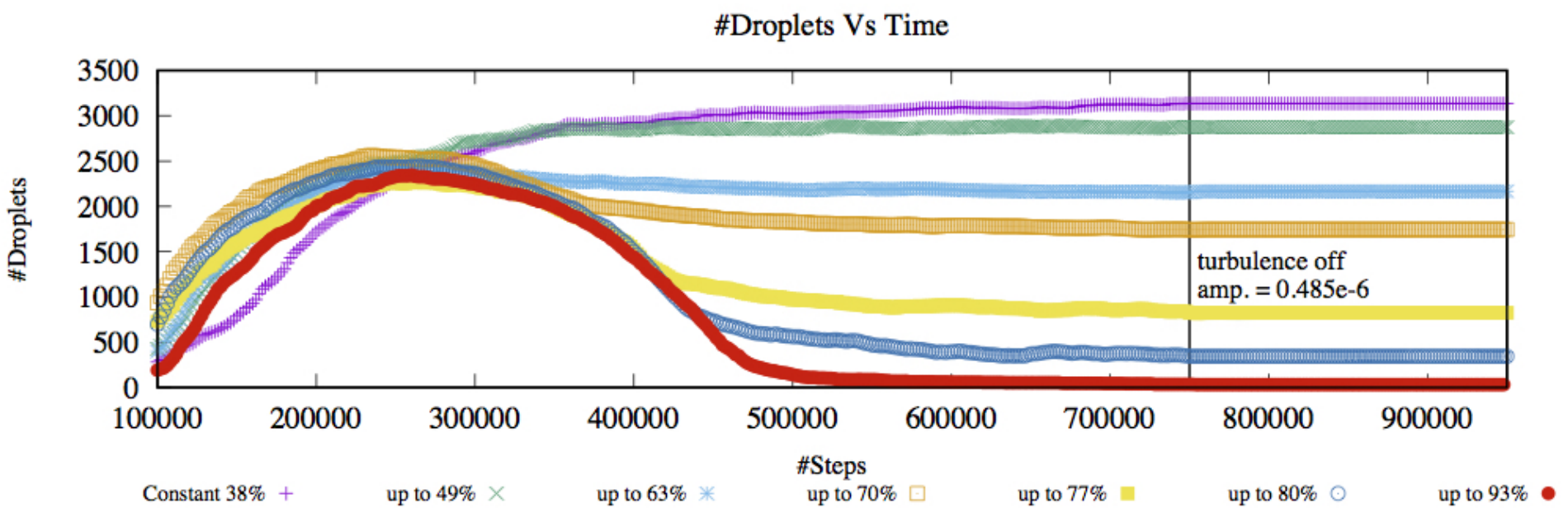

Number of Droplets
force amplitude = 3.125e-6
force amplitude = 6.25e-6
force amplitude = 1.5625e-6
force amplitude = 1.25e-5

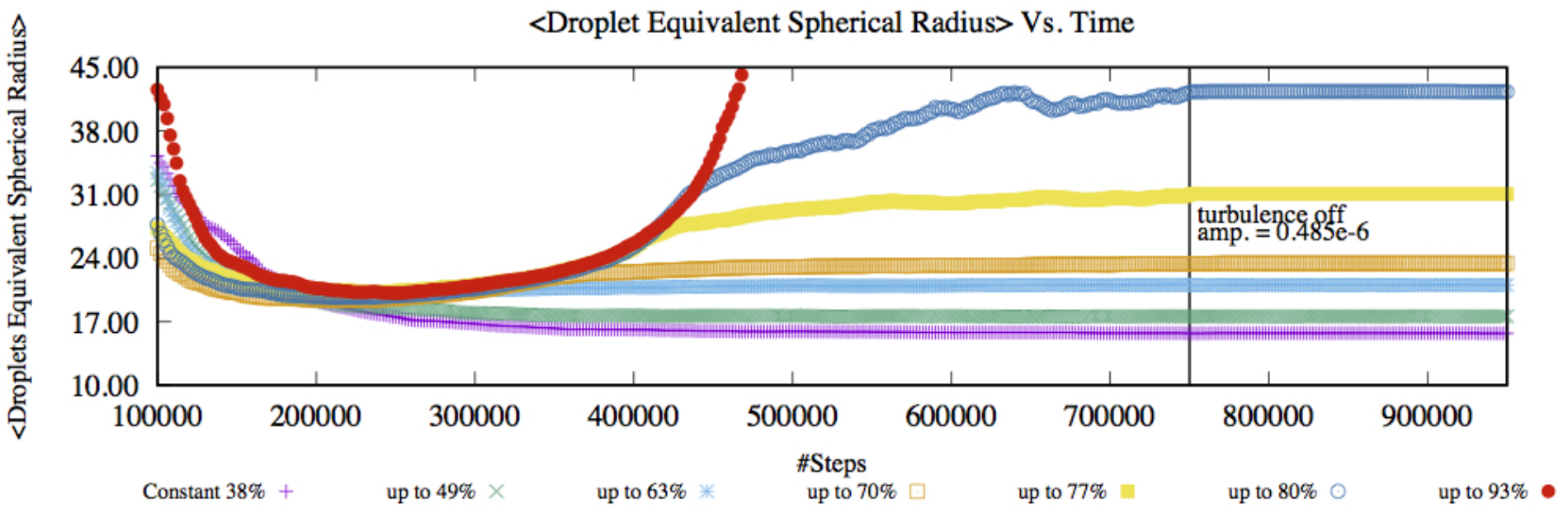

Average Volume
force amplitude = 3.125e-6
force amplitude = 6.25e-6
force amplitude = 1.5625e-6
force amplitude = 1.25e-5


Total Volume
force amplitude = 3.125e-6
force amplitude = 6.25e-6
force amplitude = 1.5625e-6
force amplitude = 1.25e-5


Droplets Surface Distribution
<A(t)>
<Arms(t)>
Number of Grid Points
Number of Steps

… and more


#Droplets Vs Time
#Droplets
3500
3000
2500
2000
1500
1000
500
0
100000
200000
300000
400000
500000
600000
700000
800000
900000
#Steps
turbulence off
amp. = 0.485e-6
Constant 38%
up to 49%
up to 63%
up to 70%
up to 77%
up to 80%
up to 93%


<Droplet Equivalent Spherical Radius> Vs. Time
<Droplets Equivalent Spherical Radius>
45.00
38.00
31.00
24.00
17.00
10.00
100000
200000
300000
400000
500000
600000
700000
800000
900000
#Steps
turbulence off
amp. = 0.485e-6
Constant 38%
up to 49%
up to 63%
up to 70%
up to 77%
up to 80%
up to 93%

The Framework

User Features

- Scaling and Efficiency
- Steering*
- Statistics
- Data Analysis
- Performance Profiling
- Visualization

Robustness & Sustainability

- Dynamic Documentation*
- Modularization of the Code Development
- Usability
- CI
- Portability

Reproducibility

- Binary Reproducibility
- Workflow Reproducibility
- RDM

Reproductivity

- Code development is based on Git
- Every time the code is compiled a unique id is generated and associated to the (commit hash), including userId
- Platform information as well as compiling information are sent to the a web service via a json file format
- Also the code modules composition is preset by the user and sent to the web service
- Every time the generated binary is executed a runId is associated to the run together with input the input file
- All output data and statistics are sent to the web server when the simulation ends

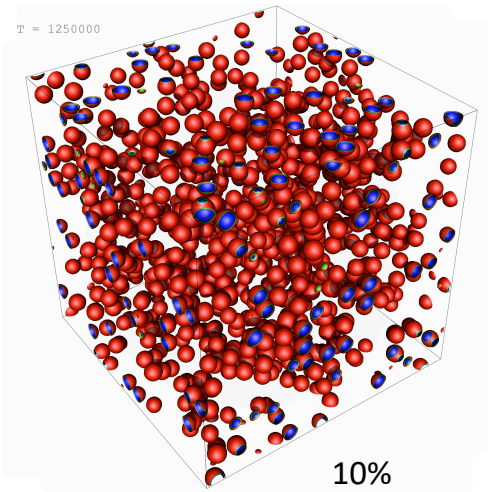
Robustness & Sustainability

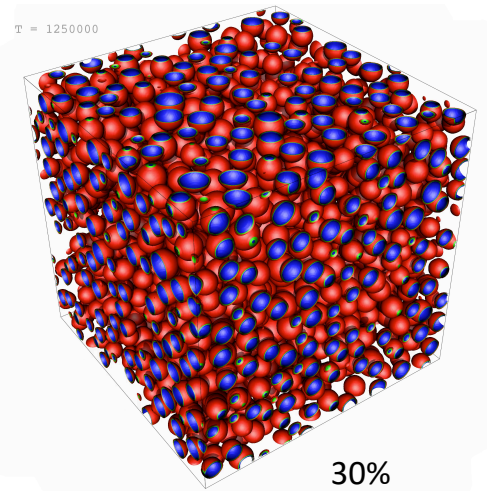
- The code is composed by several modules that the user can set at compile time
- Development is managed via e GitLab private server
- Every push to the remote repository executes a first level series of tests (CI)
- Every merge requested is accepted only if the new development passes a second level series of tests
- All testing and test logging is managed by the Cdash software framework

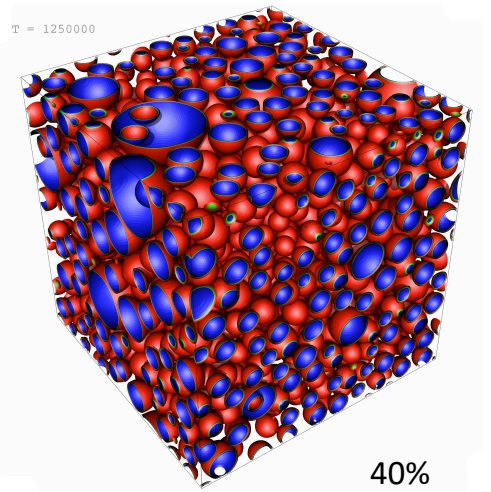
Features

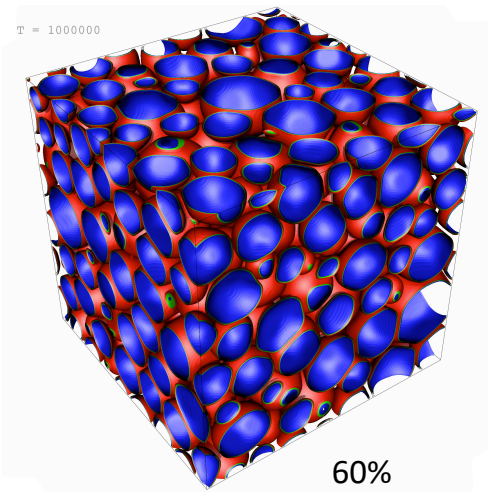
- The code implements a 3D data distribution schema
- It is a hybrid MPI+OpenMP code
- I/O fully managed by HDF5, with both parallel and serial interfaces (data are sequentially sent to 0)
- Large number of statistics (up to the per-droplet level) are implemented, with the objectives of making most of those on-line available at runtime (i.e., posting issues on GitLab)
- The code implements an ad-hoc (per subroutine) profiling system
- There is the plan to implement runtime tuning of the application (steering)

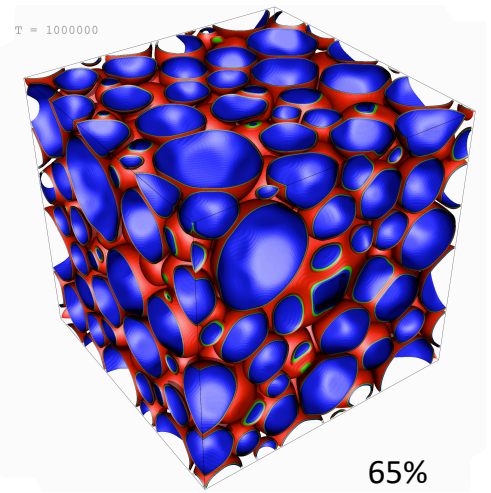
Rheology

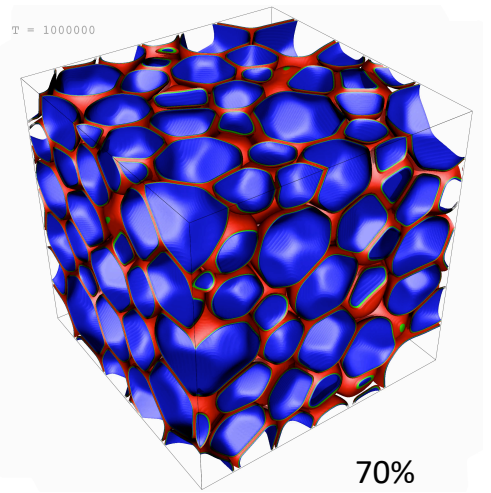

T = 1250000
10%


T = 1250000
30%


T = 1250000
40%


T = 1000000
60%


T = 1000000
65%


T = 1000000
70%

Thanks For Your Attention!

- Federico Toschi
- Fabio Schifano
- Gianluca di Stasio
- Karun P.N. Datadien
- Alessandro Corbetta

UNIMORE
UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA

TU/e Technische Universiteit Eindhoven University of Technology

The Abdus Salam
International Centre
for Theoretical Physics