PARAVT – PARALLEL VORONOI TESSELLATION

Roberto González
regonzar@astro.puc.cl
https://github.com/regonzar/paravt
January 2016

Contents

1 Introduction

PARAVT is new parallel code to compute Voronoi tessellations(VT hereafter) in large data-sets. It is designed for Linux platforms from small multicore computers up to large computer clusters.

The code is focused for astrophysical purposes where Voronoi densities and neighbors data are widely used.

Particles are transferred to different processors under MPI language[1], following a domain decomposition and taking into account consistent boundary computation between tasks. It also works with periodic conditions. VT is computed using Qhull libraries[2].

2 Features

- MPI parallel code which makes possible VT computation of large data-sets taking advantage of multicore and cluster architectures.
- VT of particles at boundaries of tasks sub-volumes are properly computed and neighbors re-indexed as a single volume.
- Computes neighbors list for each particle.
- Computes Voronoi density and cell volumes for each particle.
- Optionally can computes average density on a regular grid.
- Periodic conditions and multimass particles.
- ASCII and Binary I/O format.
- Gadget and Rockstar input formats, specially for N-body cosmological analysis.
- Raw Qhull output, memory and precision management, VTK grid output for quick visualization.
- User guide.

[1] *http://www.mpi-forum.org/*
[2] *http://www.qhull.org*

3 Download

Latest version of the code can be downloaded from,
https://github.com/regonzar/paravt

4 Compile and Run

This code should be compatible with most Linux distribution. It requires any MPI implementation and run with at least 2 MPI tasks. Can not run on a single task or compiled in a serial version (use Qhull directly to do that). Code was tested with OpenMPI-1.6.5+ and Cent-OS 7 Linux distribution. Qhull library is self contained and is compiled with the code.

Before compilation, you should check in next section the code parameters from *config.h*. Always after changing parameters you should use "make clean" command.

To compile, just type "make" command inside code directory. If compilation fail, edit MAKEFILE and set CC, INC, and LIBS variables accordingly the custom location of your MPI implementation. Code runs with:

```
> mpirun -np <Ntask> ./paravt <datafile> <format>

    where:

<Ntask> = Number of MPI tasks
<datafile> = input file name
<format> = input file format:
    0=ASCII, 1=Binary, 2=Gadget, 3=Rockstar
```

In general, total memory consumption is up to $\sim 3Gb$ per million particles with masses and with double precision.

5 Configuration

In *config.h* file you can configure several parameters, they are in the form of **#define PARAMETER**, some of them require a value and other just need to be defined. To undefine or unset a given parameter just comment that line. Remember to "make clean" after a change of parameters.

- DOMAINTYPE: Domain decomposition strategy. DOMAINTYPE 0 split volume in rectangular blocks where Ntask must be a power of 2. DO-

MAINTYPE 1 split volume along the larger axis in Ntask blocks where Ntask can be any number. For cubic, rectangular volumes and/or periodic conditions DOMAINTYPE 0 is should be used, and for irregular volumes DOMAINTYPE 1 is recommended. PERIODIC option is disabled for DOMAINTYPE 1. (Default 0).

- BOX: Set if volume is a cube, input format will just read LBOX as the cube side. DOMAINTYPE 0 should be used. Gadget and Rockstar format works only with BOX enabled.
- PERIODIC: Periodic conditions. Works only with BOX enabled.
- MULTIMASS: Enable particle mass. Read an additional column of data with particle masses. Rockstar format requires MULTIMASS, and must be disabled for GADGET format.
- BUFFERSIZE: number of particles for communication buffer. A larger value requires more temporary memory. Default (100000).
- BORDERFACTOR: Thickness of boundary particles layer around each task subvolume, in terms of the inter-particle distance. This parameter should be larger than 1.0 to ensure accurate VT computation at boundaries, and be smaller than ~ 5.0 to avoid an unnecessary excess of repeated temporary particles. Default (2.0)
- DIM: Number of dimensions. Works only for 3 dimensions for the moment, two dimensions case will be enabled for future versions. Default (3)
- NFAC: Guess of the mean number of vertices of a facet. You should not change this parameter, but you can try to reduce up to ~ 9 if you want to save a little bit of memory. Safe values are $15 - 20$. (default 15).
- WRITEASCII: Output files will be written in ASCII format, otherwise it will be binary.
- WRITEDENSITY: Enable output of density(.den) and volume(.vol) files.
- WRITENEIGHBORS: Enable output of neighbors(.nb .nb2) files.
- WRITEVTSTRUCT: Enable raw Qhull output for each task(.VT). Output in this case is only available in ASCII format and produces very large files compared to the other outputs.

- COMPUTEGRID: Compute average Voronoi density in a regular grid. Grid size is defined by GRIDSIZE parameter and must be divisible by domain decomposition. i.e. grid size of 256 can be split in 8 tasks with grid portions of $128 \times 128 \times 128$.
- GADGET_PTYPE: If you read a gadget file, it set the particle type to use. (0-5).
- QHULLOPTIONS: Qhull additional parameters, QJ add small random noise to avoid repeated and co-planar particles. Qbb is precise but fail if repeated coordinates found. (default "Qbb").
- LOWMEMORY: Reduces total memory consumption by 1/3, but doubles execution time.
- VERBOSE: Write additional information on screen.

Qhull precision can be configured in *qhullsrc_r/user_r.h* by changing,

```
#define REALfloat 0     default 'double' type

#define REALfloat 1     'float' type
```

Using float data type instead double does not improve speed, but in most cases reduce memory consumption.

6 Input Formats

Here is the description of input file formats options:

6.1 ASCII

This is the simplest way for input data, format option is 0.

We show two examples:

Case 1: cube of side $LBOX$, and with NP particles with coordinates X, Y, Z and mass M. BOX and MULTIMASS options enabled.

```
NP LBOX
X(0) Y(0) Z(0) M(0)
.
.
.
X(NP−1) Y(NP−1) Z(NP−1) M(NP−1)
```

Case 2: rectangular volume with sides LX, LY, and LZ, with NP particles with the same mass. BOX and MULTIMASS options disabled.

```
NP LX LY LZ
X(0) Y(0) Z(0)
.
.
.
X(NP−1) Y(NP−1) Z(NP−1)
```

6.2 Binary

This is format option 1. Binary format is faster and save space. Case 1 from previous example should be:

```
<NP><LBOX><X(0)><Y(0)><Z(0)><M(0)>...
...<X(NP−1)><Y(NP−1)><Z(NP−1)><M(NP−1)>
```

where <NP> is a 4 bytes integer, <LBOX> a 4 bytes float, and coordinates are also 4 bytes floats.

6.3 Gadget

With format option 2, code reads Gadget[3] single or multiple files binary format. It can process one particle type defined by GADGET_PTYPE parameter. MULTIMASS must be disabled and BOX enabled.

6.4 Rockstar

With format option 3, code reads Rockstar[4] halo format. MULTIMASS and BOX must be enabled. It reads X, Y, Z, and MVIR columns.

[3] *http://wwwmpa.mpa-garching.mpg.de/gadget*

[4] *https://bitbucket.org/gfcstanford/rockstar*

7 Output Files

Output files may be enabled/disabled in *config.h*, and their names are the same as the input file plus an extension.

7.1 Density file(.den)

WRITEDENSITY must be enabled. Particles density is written. Format depends on WRITEASCII parameter.

If ASCII, file has a single column, where first element is the number of particles and following lines are the Voronoi density of each particle, written in the same order as input file.

```
NP
dens(0)
.
.
.
dens(NP-1)
```

If binary format, first element is a 4 bytes integer with the number of particles, then follows an array of 4 bytes floats with densities.

```
<NP><dens(0)>...<dens(NP-1)>
```

7.2 Volumes file(.vol)

This file is written in addition to densities, with identical format, but vornoi cell volumes for each particles are written instead.

7.3 Neighbors files(.nb .nb2)

WRITENEIGHBORS must be enabled, two files are written, one contains the number of neighbors and a neighbor index, and the other file an array of neighbors, with both files you can know the indices of the neighbors for each particle. Format depends on WRITEASCII parameter.

In ASCII we have the following format for the first file (.nb),

```
NP
Nneig(0)  nindex(0)
.
.
.
.
Nneig(NP−1)  nindex(NP−1)
```

and the second file (.nb2) hast the format,

```
NneigTotal
id(0)
.
.
.
.
id(NneigTotal−1)
```

where we have that particle with index i, has Nneig(i) neighbors, and the indices of their neighbors are,
id[nindex(i)], id[nindex(i)+1], ... id[nindex(i)+Nneig(i)-1]
indices correspond to values in the range [0..NP-1] from input particles.

In binary format the structure is the same where all numbers are 4 bytes integers (.nb),

```
<NP><Nneig(0)>...<Nneig(NP−1)><nindex(0)>...<nindex(NP−1)>
```

and (.nb2) file,

```
<NneigTotal><id(0)><id(1)>...<id(NneigTotal−1)>
```

7.4 VT structure files(.VT)

WRITEVTSTRUCT must be enabled, an ASCII file is generated per Task with extension (.VT.Task). File format first include the indices of particles in given Task, then follows Qhull raw output with vertices and facets.

```
NPTask  NPboundary
ind(0)
ind(1)
.
.
.
ind(NPTask−1)
.
.
.
ind(NPTask+NPboundary−1)
Ndim
NV
XV(0)  YV(0)  ZV(0)
.
.
.
XV(NV−1)  YV(NV−1)  ZV(NV−1)
Nfacet
len(0)  id1  id2  iv(0)  iv(1)  ...  iv(len(0)−4)
.
.
.
len(Nfacet−1)  id1  id2  iv(0)  iv(1)  ...  iv(len(Nfacet−1)−4)
```

where indices *ind* correspond to input particles order, the last NPboundary particles in the list are boundary particles which should not be considered because they are in the domain of other task. Then follow number of dimensions Ndim=3, the number of vertices NV, and three column coordinates list with NV lines. After vertices follows Nfacet, it is the number of facets, then follows a list of variable width with facet information, where $len-3$ is the number of vertices forming the given facet. $id1$ and $id2$ are the indices of the two particles sharing the facet in current file, then their absolute index(from input file) are $ind[id1]$ and $ind[id2]$. Finally follows the vertices indices of the facet. Notices that iv values ranges from (1 to NV) then for arrays starting from 0 index remember to shift by -1 (C, C++ types).

7.5 Grid files(.grid .vtk)

COMPUTEGRID must be enabled. VT density averaged on a regular grid of side size GRIDSIZE is generated. Two files are produced, a binary file (.grid) where first element is a 4 byte integer with GRIDSIZE followed by GRIDSIZE*GRIDSIZE*GRIDSIZE 4 bytes floats with grid values. The other file (.vtk) is just the same grid in VTK format[5] for visualization. i.e. volume rendering using VisIt[6].

8 Next Features/Updates

Several new features will be included in next versions:

- Density gradient vectors.
- Domain decomposition for irregular volume shapes. i.e. Mock catalogs.
- Grid output for T-web V-web cosmicweb metrics.
- HDF5 input format.
- Other cosmological codes file formats (RAMSES, ART, etc).
- Extension to 2D.
- Improve sequential execution in LOWMEMORY mode to limit memory usage in very large data-sets.

Questions, comments, or feature requests are welcome.

Roberto Gonzalez
regonzar@astro.puc.cl

[5] *http://www.vtk.org*
[6] *https://visit.llnl.gov*