Deterministic schemes for the numerical solution of the Boltzmann equation

E.A. Malkov

Computational Aerodynamics Lab
ITAM SB RAS
Novosibirsk, RUSSIA

Workshop on Non-equilibrium Flow Phenomena
in Honor of Mikhail Ivanov's 70th Birthday

Boltzmann transport equation

$$\underbrace{\frac{\partial f}{\partial t} + \mathbf{v}\frac{\partial f}{\partial \mathbf{r}} + \mathbf{F}\frac{\partial f}{\partial \mathbf{v}}}_{\text{convective term}} = \underbrace{St(f,f)}_{\text{stoss term}}$$

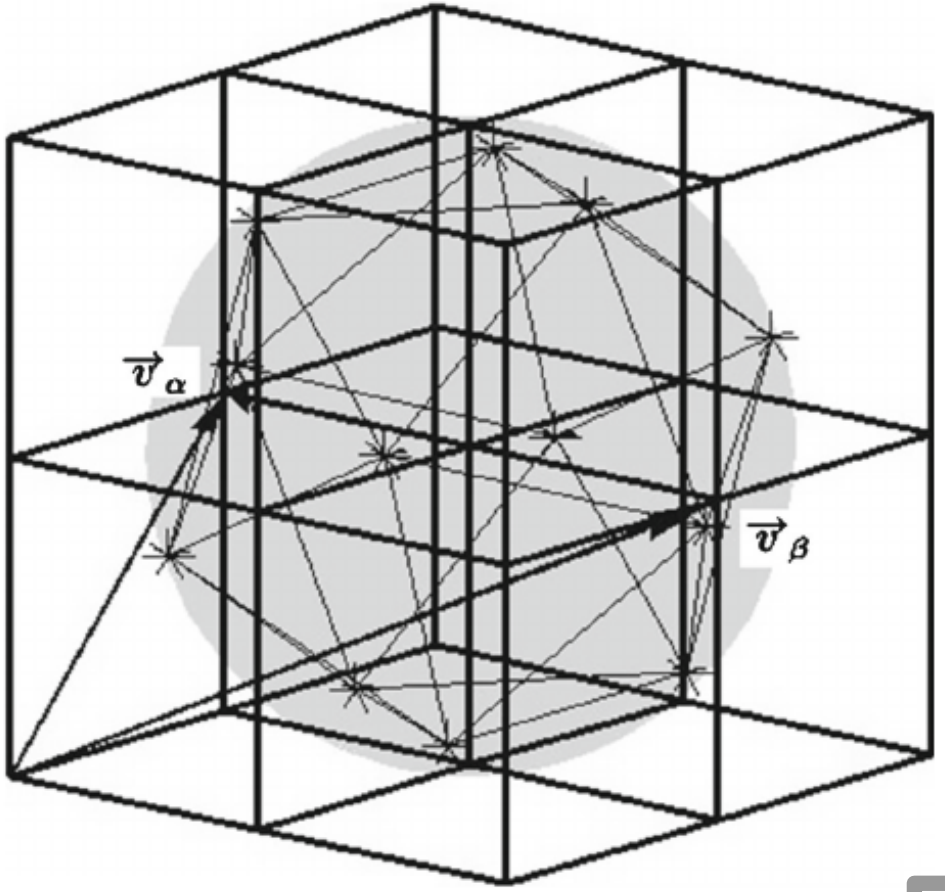
$$\underbrace{St(f,f)(\mathbf{v}) = \int\limits_{R^3} d^3v_1 \int\limits_{S^2} d^2n(f'f'_1 - ff_1)|\mathbf{V}|\sigma(V, \mathbf{V}\cdot\mathbf{n}),}_{\textbf{the collision integral}}$$

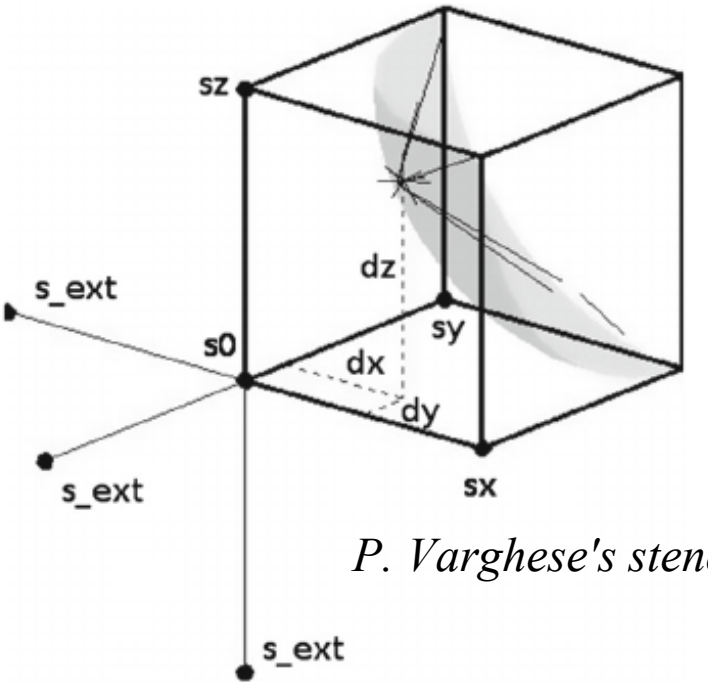
where

$$f = f(t,\mathbf{r},\mathbf{v}), f_1 = f(t,\mathbf{r},\mathbf{v}_1),$$
$$f' = f(t,\mathbf{r},\mathbf{v}'), f'_1 = f(t,\mathbf{r},\mathbf{v}'_1),$$

$$\mathbf{v}' = \frac{\mathbf{v}+\mathbf{v}_1}{2} + \frac{|\mathbf{V}|}{2}\mathbf{n},$$
$$\mathbf{v}'_1 = \frac{\mathbf{v}+\mathbf{v}_1}{2} - \frac{|\mathbf{V}|}{2}\mathbf{n}.$$

grid approximation





P. Varghese's stencil

$$\mathbf{v}_{\alpha\prime} = \frac{\mathbf{v}_\alpha + \mathbf{v}_\beta}{2} + \frac{|\mathbf{V}|}{2}\mathbf{n}_{\alpha\prime},$$

$$\mathbf{v}_{-\alpha\prime} = \frac{\mathbf{v}_\alpha + \mathbf{v}_\beta}{2} - \frac{|\mathbf{V}|}{2}\mathbf{n}_{\alpha\prime}.$$

Distribution of *dm* (inverse interpolation)
from
the quadrature nodes on sphere
to
the calculation grid nodes,
mass, energy and impulse are saved.

$$dm = f(\mathbf{v}_\alpha)f(\mathbf{v}_\beta)w(\alpha,\beta;\alpha',-\alpha')\ \ f(\mathbf{v}_\alpha)-=dm,\ f(\mathbf{v}_\beta)-=dm,\ \ f(\mathbf{v}_{\alpha'})+=dm,\ f(\mathbf{v}_{-\alpha'})+=dm$$

$\forall(\alpha,\beta) \in \{pairs\,of\,nodes\,of\,the\,calculation\,grid\ \ in\,the\,velocity\,space\}$

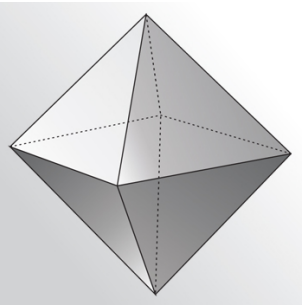
$\forall(\alpha,-\alpha) \in \{pairs\,of\,the\,quadrature\,nodes\,on\,sphere\ \ in\,the\,velocity\,space\}$

what one has to calculate

Quadrature on sphere: invariant with respect to groups of rotation of octahedron with inversion, a fifth-order of algebraic accuracy (the quadrature yields an exact value of the integral for all spherical polynomials with the power of 5 or lower) with 14 nodes.

Popov, A.S.: Sib. Zh. Vych. Mat. 8(2), 143–148 (2005)

Nodes	Weights/4π	Comments
$(\pm 1, 0, 0)$	1/15	Octahedron vertices
$(0, \pm 1, 0)$		
$(0, 0, \pm 1)$		
$(\pm 1/\sqrt{3}, \pm 1/\sqrt{3}, \pm 1/\sqrt{3})$	3/40	Face centers



http://mnogogranniki.ru/vidy-mnogogrannikov/8-vidy/79-oktajedr.html

```
struct SphereGrid{
 float val_knots[7*3];
 float weights[7];
 int num_of_knots;
}sgrid;
```

Parameters of quadrature on spheres for all pairs of nodes (different relative velocities) in the calculation grid are obtained by homotheties and translations from ***sgrid***.

what one has to calculate yet

All parameters of distribution (or inverse interpolation) are calculated once and are stored in data structure:

```
typedef struct __align__(16) tag_InterpolationParams{
 int icell,jcell,kcell;          //a nearest node
 int is,js,ks;   //auxiliary parameters
 int icell1,jcell1,kcell1; //a nearest node for opposite quadrature  node on sphere
 int is1,js1,ks1; //auxiliary parameters
 REAL s0,sx,sy,sz,s_ext;   //interpolation coefficients
 REAL weight;  //the differential scattering cross section values multiplied by quadrature
weghts
}InterpolationParams;
```

here REAL is defined by

```
#ifdef
COMPUTE_CAPABILITY_20
#define REAL double
#else
#define REAL float
#endif
```

parallelization of the algorithm of the collision integral calculation

Let *df* and *st* be arrays of length $Nvx \times Nvy \times Nvz \times Nx \times Ny \times Nz$ storing the values of the distribution function and the collision integral in the computational grid nodes in the phase space . The index of the grid node in the physical space is $n = nx + ny \cdot Nx + nz \cdot Nx \cdot Ny$, and the index of the grid node in the velocity space is $nv = nvz + nvy \cdot Nvz + nvx \cdot Nvz \cdot Nvy$

Using these notations, we write **the main actions to be parallelized**:

$$for\ \ all\ \ n \in G_R^3 \subset R^3$$

$$for\ \ all\ \ (nv_1, nv_2) \in G_V^3 \times G_V^3 \subset R^3 \times R^3 \quad do$$

$$\Delta m = df[n + nv_1 * Nx * Ny * Nz] * df[n + nv_2 * Nx * Ny * Nz];$$

$$for\ \ all\ \ \{\alpha_1, \alpha_2\} \subset G_V^2 \subset S^2 \quad do$$

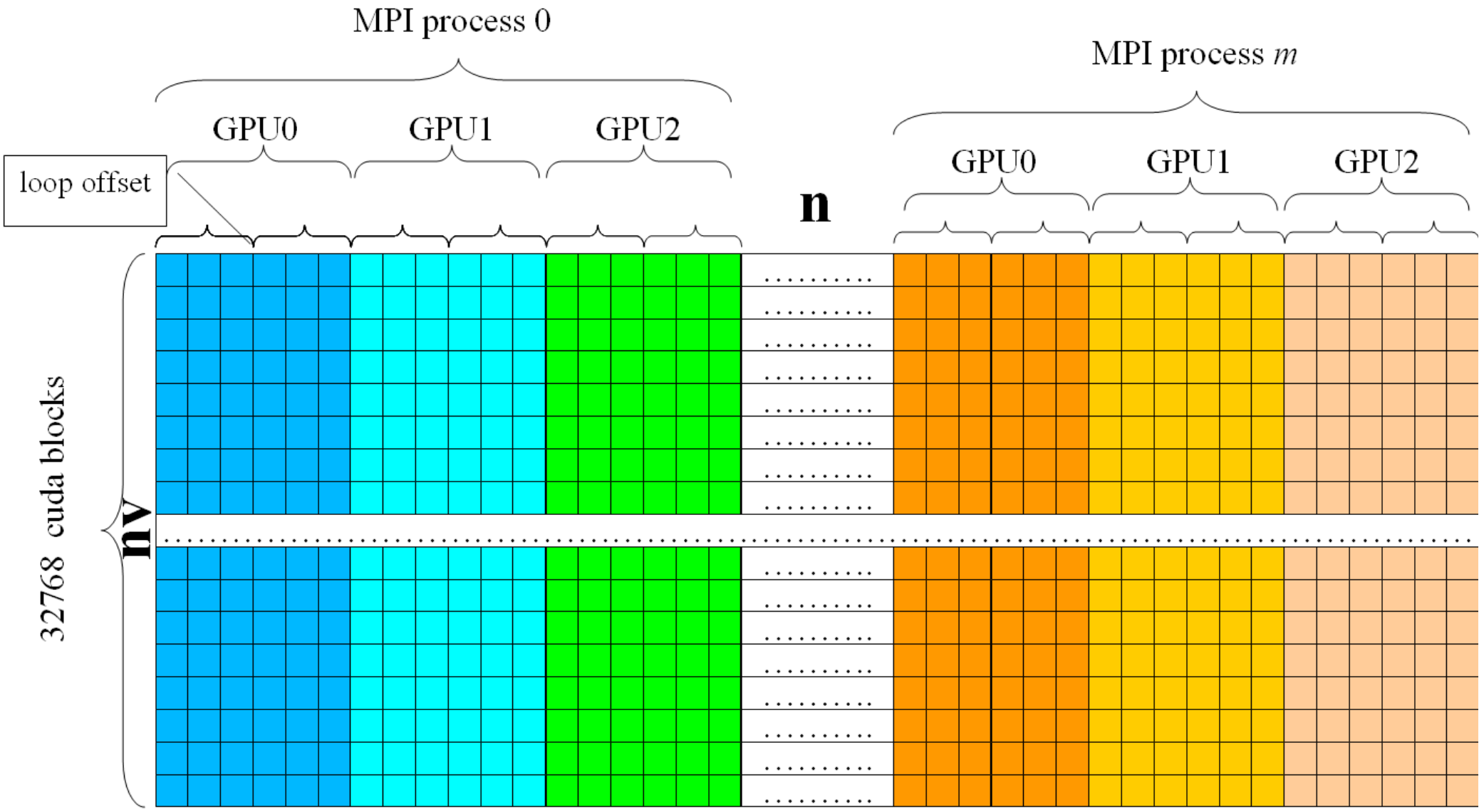
$$for\ \ p = 0\ to\ num\_of\_stencil\_points \quad do$$

$$st[n + nv_{p,\alpha_1} * Nx * Ny * Nz] += \Delta m * s_{p,\alpha_1};$$

$$st[n + nv_{p,\alpha_2} * Nx * Ny * Nz] += \Delta m * s_{p,\alpha_2};$$

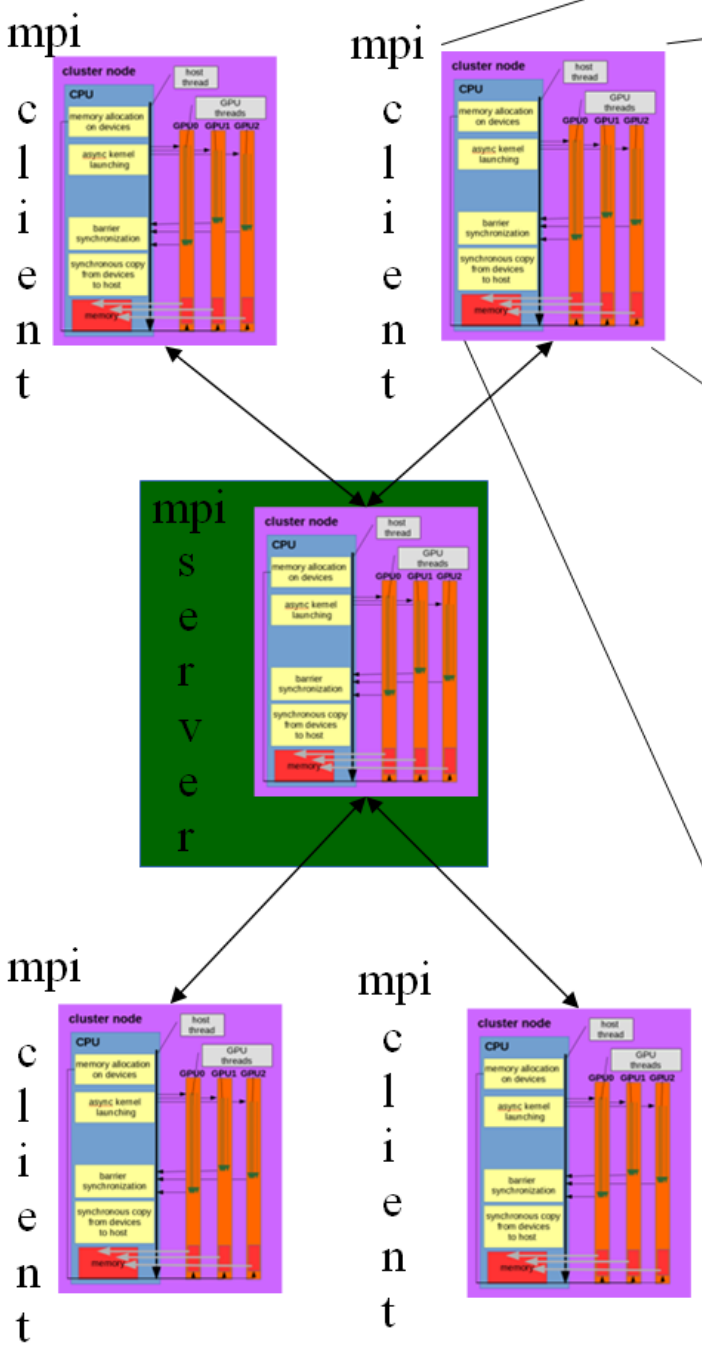
$$end\ do;\ end\ \ do;\ \ end\ \ do;$$

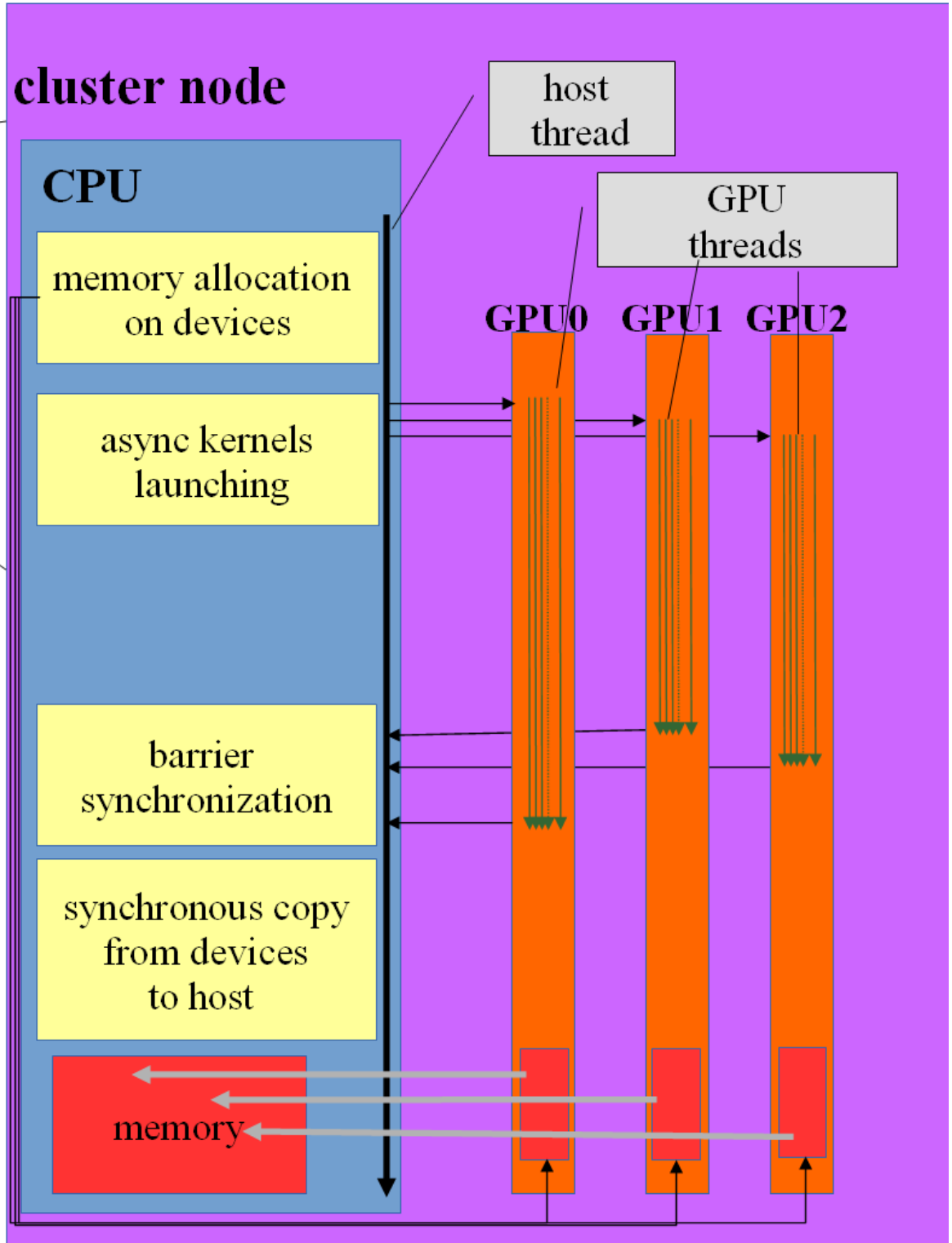
decomposition



Typical value of *loop offset* – from 32 to 512, correspond to number of cuda threads in a cuda block.

select=any:ncpus=1:
mpiprocs=1:gpus=1..3





memory allocation

on host: allocation memory for arrays of pointers to store pointers initialized on GPU

```
InterpolationParams**  d_InterParams=( InterpolationParams**)calloc(NG,
                                                    sizeof(InterpolationParams*) );
float**  Df_device=(float**)calloc(NG, sizeof(REAL*));
float**  St_device=(float**)calloc(NG, sizeof(REAL*));
```

on devices: allocation memory to store values of grid approximations of functions

```
 for(int idev=0;idev< NG; idev++){
   cudaSetDevice(assigned_devices[idev]);
        cudaMalloc((void **) &d_InterParams[idev], sizeOfInterParams );
        cudaMalloc((void **) &Df_device[idev], size_of_Df/NG);
        cudaMalloc((void **) &St_device[idev], size_of_Df/NG);
}
```

InterpolationParams – structure to store interpolation coefficients and coordinates of the stencil points.
NG – number of GPUs.
Df_device[idev], St_device[idev] – parts of the distribution function and collisional integral grid approximations destined to different GPUs.
idev – GPU's number.
assigned_devices[idev] – GPU's identifier.

async kernel launching and barrier synchronization

allocation memory for CUDA Event objects

```
//cudaEvent_t  *mdEventStart=(cudaEvent_t*)calloc(NG, sizeof(cudaEvent_t));
cudaEvent_t  *mdEventStop=(cudaEvent_t*)calloc(NG, sizeof(cudaEvent_t));
```

creating events on GPUs.

```
for(int idev=0;idev< NG; idev++){
        cudaSetDevice(assigned_devices[idev]);
                //cudaEventCreate(&mdEventStart[idev]);
                cudaEventCreate(&mdEventStop[idev]);
}
```

recording event in default stream on GPUs just after calling a kernel

```
for(int idev=0;idev< NG; idev++){
        cudaSetDevice(assigned_devices[idev]);
            //cudaEventRecord( mdEventStart[idev], 0 );
            gStossCalc<<<dim3(nvx, nvy,nvz), dim3(SPACE_CELL_OFFSET)>>>
                    (d_InterParams[idev], Df_device[idev], St_device[idev],  N/NG,  l_offset);
            cudaEventRecord( mdEventStop[idev], 0);
}
```

synchronization kernel calls

```
for(int idev=0;idev< NG; idev++){
     cudaSetDevice(assigned_devices[idev]);
                cudaEventSynchronize(mdEventStop[idev]);
}
```

synchronous copy from devices to host

```
for(int idev=0;idev< NG; idev++){
  cudaSetDevice(assigned_devices[idev]);
       cudaMemcpy(St+idev*nvx*nvy*nvz*N/NG, St_device[idev],
            size_of_Df/NG, cudaMemcpyDeviceToHost);
}
```

(one gpu) x (six loops)

```
NG=1   gCalculateParams        Elapsed time: 1.55024
gClearStoss        gClearStoss        Elapsed time: 0.213664
gTransposeDf       Elapsed time: 5.2569
gStossCalc         Elapsed time: 246532
gTransposeDfInverse        Elapsed time: 7.51043
gClearStoss        gClearStoss        Elapsed time: 0.217504
gTransposeDf       Elapsed time: 5.25312
gStossCalc         Elapsed time: 246536
gTransposeDfInverse        Elapsed time: 7.53414
gClearStoss        gClearStoss        Elapsed time: 0.217088
gTransposeDf       Elapsed time: 5.25322
gStossCalc         Elapsed time: 246535
gTransposeDfInverse        Elapsed time: 7.51981
gClearStoss        gClearStoss        Elapsed time: 0.216192
gTransposeDf       Elapsed time: 5.25834
gStossCalc         Elapsed time: 246533
gTransposeDfInverse        Elapsed time: 7.46163
gClearStoss        gClearStoss        Elapsed time: 0.216544
gTransposeDf       Elapsed time: 5.25523
gStossCalc         Elapsed time: 246538
gTransposeDfInverse        Elapsed time: 7.49238
gClearStoss        gClearStoss        Elapsed time: 0.215904
gTransposeDf       Elapsed time: 5.24982
gStossCalc         Elapsed time: 246536
gTransposeDfInverse        Elapsed time: 7.52288
real    24m47.123s
user    24m45.149s
sys     0m0.944s
```

25,165,824
grid nodes
in phase space

every loop handles
4,194,304 grid nodes
in phase space

(two gpu) x (three loops)

```
NG=2  gCalculateParams      Elapsed time: 1.53939
NG=2  gCalculateParams      Elapsed time: 1.49997
gClearStoss         gClearStoss         Elapsed time: 0.211264
gClearStoss         gClearStoss         Elapsed time: 0.19552
gTransposeDf        Elapsed time: 5.2217
gTransposeDf        Elapsed time: 5.23258
gStossCalc          Elapsed time: 248925
gStossCalc          Elapsed time: 248926
gTransposeDfInverse         Elapsed time: 7.48387
gTransposeDfInverse         Elapsed time: 7.48378
gClearStoss         gClearStoss         Elapsed time: 0.200736
gClearStoss         gClearStoss         Elapsed time: 0.209472
gTransposeDf        Elapsed time: 5.2145
gTransposeDf        Elapsed time: 5.22477
gStossCalc          Elapsed time: 248722
gStossCalc          Elapsed time: 248723
gTransposeDfInverse         Elapsed time: 7.51037
gTransposeDfInverse         Elapsed time: 7.51152
gClearStoss         gClearStoss         Elapsed time: 0.199392
gClearStoss         gClearStoss         Elapsed time: 0.208352
gTransposeDf        Elapsed time: 5.21789
gTransposeDf        Elapsed time: 5.22787
gStossCalc          Elapsed time: 248926
gStossCalc          Elapsed time: 248926
gTransposeDfInverse         Elapsed time: 7.51379
gTransposeDfInverse         Elapsed time: 7.51504
real    12m35.874s
user    12m33.123s
sys     0m1.628s
```

25,165,824 grid nodes in phase space

every loop handles ***4,194,304*** grid nodes in phase space

two-times speedup vs. 1 GPU

(three gpu) x (two loops)

```
NG=3  gCalculateParams      Elapsed time: 1.47718
NG=3  gCalculateParams      Elapsed time: 1.53101
NG=3  gCalculateParams      Elapsed time: 1.5304
gClearStoss     gClearStoss     Elapsed time: 0.140192
gClearStoss     gClearStoss     Elapsed time: 0.155264
gClearStoss     gClearStoss     Elapsed time: 0.124384
gTransposeDf    Elapsed time: 4.94176
gTransposeDf    Elapsed time: 5.1377
gTransposeDf    Elapsed time: 5.14787
gStossCalc      Elapsed time: 164576
gStossCalc      Elapsed time: 164589
gStossCalc      Elapsed time: 164590
gTransposeDfInverse     Elapsed time: 7.12509
gTransposeDfInverse     Elapsed time: 7.43171
gTransposeDfInverse     Elapsed time: 7.44042
gClearStoss     gClearStoss     Elapsed time: 0.128416
gClearStoss     gClearStoss     Elapsed time: 0.13792
gClearStoss     gClearStoss     Elapsed time: 0.147904
gTransposeDf    Elapsed time: 4.93661
gTransposeDf    Elapsed time: 5.13046
gTransposeDf    Elapsed time: 5.13981
gStossCalc      Elapsed time: 164575
gStossCalc      Elapsed time: 164590
gStossCalc      Elapsed time: 164591
gTransposeDfInverse     Elapsed time: 7.11542
gTransposeDfInverse     Elapsed time: 7.44173
gTransposeDfInverse     Elapsed time: 7.4505
real    5m38.815s
user    5m36.017s
sys     0m1.800s
```

25,165,824
grid nodes
in phase space

every loop handles
4,194,304 grid nodes
in phase space

superlinear speedup:

aprox 5-times (not 3 !) speedup
vs. 1 GPU

the former estimation of speedup

Parallel computations with one spatial cell ensure a **12-fold speedup as compared with sequential computations on a processor with a frequency of 2.66 GHz.** The results of relaxation computations with different numbers of spatial cells (threads in a block) performed on the GPGPU Tesla M2090 are summarized in the table:

Number of threads	Execution time (ms)	Speedup
1	676	1
2	523	2.59
4	462	5.85
8	416	13
16	349	30.99
32	523	41.36
64	1045	41.4

conclusion

The use of CUDA streams for asynchronous launching of kerns for calculations of the collision integral on different GPU is very effective. It is possible to reach superlinear speedup, having set smart configuration for cuda threads. So, 3 GPUs on the same node can give 5 times speedup. Taking into account the former estimations, one can estimate the total speedup due only **one node of hybride cluster** as **12x41x5** … and even more.