BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasarkan pembahasan sebelumnya di dapatkan Kesimpulan sebagai berikut :

a. Program visualisasi fluida 1 fase 2 fase dengan model *lattice* bolztman berhasil di lakukan.
b. Algoritma pemrograman komputasi paralel untuk visualisasi fluida berhasil di kembangkan.
c. Dengan Komputasi paralel GPU CUDA pada kasus pertama pada nilai *pixel* 320x112 lebih cepat 1 kali di bandingkan tanpa komputasi paralel. Pada jumlah *lattice* 960x339 lebih cepat 25 kali.
d. Dengan Komputasi paralel GPU CUDA pada kasus kedua pada nilai *pixel* 640x640 lebih cepat 115 kali di bandingkan tanpa komputasi paralel.
e. Dengan Komputasi paralel GPU CUDA pada kasus ketiga pada nilai *pixel* 960x339 lebih cepat 20 kali di bandingkan tanpa komputasi paralel.

5.2. Saran

Beberapa saran untuk pembaca yang akan melanjutkan penelitian ini :

1. Pengembangan kode program visualisasi 3 dimensi.
2. Pengembangan kode program untuk simulasi fluida multi component, seperti minyak dan air.

3. Kode program Perbandingan dengan menggunakan metode yang berbeda selain metode Lattice Boltzmann.
4. Pengembangan kode program yang mampu menerapkan nilai *lattice* tanpa nilai batas tertentu pada *hardware* yang di gunakan.
5. Untuk kasus fluida multi fase perlu di tambahkan *improvement*, untuk GPU karena hasil yang di capai belum sesuai.

DAFTAR PUSTAKA

Brodtkorb, S.A., 2011. Efficient Shallow Water Simulations on GPUs: Implementation, Visualization, Verification, and Validation. *computer & fluid*, 30 april. p.125.

Dharmawan, A., Fitriani, D. & Susanto, K., 2011. Simulasi Lattice Boltzmann Untuk Menentukan Konsentrasi Polarisasi Pada Solid Oxide Fuel Cell. *Material dan Energi Indonesia*, 01(01), pp.47-57.

Gebaly, F., 2011. *Algorithms and Parallel Computing*. Canada: John Wiley & Sons, Inc.

Hansen, C.D. & Johnson, C.R., 2005. *The Visualization handbook*. USA: Elsevier Inc.

Huang, H., Li, Z., Liu, S. & Lu, X.-y., 2008. Shan-and-Chen-type multiphase Lattice Boltzmann study of viscous coupling effects for two-phase flow in porous media. *INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN FLUIDS*.

Huang, H., Wang, L. & Lu, X.-y., 2010. Evaluation of three Lattice Boltzmann models for multiphase flows in porous media. *elsevier*, 61, p.3606–3617.

Janicke, & Scheuermann,, 2010. Measuring Complexity in Lagrangian and Eulerian Flow Descriptions. *COMPUTER GRAPHICS forum*, 29(6), pp.1783-94.

kanginan, M., 2007. mekanika fluida. In e. Widi janto & s.n. andi ni, eds. *SERIBUPENA FISIKA SMA KELAS XI JILID2*. malang: penerbit Erlangga. pp.158-59.

Kirk, & Hwu, W.-m., 2010. *Programming Massively Parallel Processors A Hands-on Approach*. Burlington, USA: Elsevier Inc.

kjk, 2012. *openLBMflow*. [Online] Available at: http://www.lbmflow.com/ [Accessed 3 maret 2013].

Kuzmin, A., 2009. *Multiphase Simulations With* Lattice Boltzmann *Sceme*. Thesis. CALGARY, ALBERTA: Alexandr Kuzmin 2009 UNIVERSITY OF CALGARY.

Mohamad, A., 2011. Lattice Boltzmann *Method Fundamentals and Engineering Applications with Computer Codes*. london: Springer-Verlag.

Munoz, A.J., 2010. *Three-Dimensional Tsunami Modeling Using Gpu-Sphysics*. Thesis. texas: Texas A&M University.

Munson, R., Young, D.F. & Okiishi, T.H., 2004. *Mekanika Fluida*. 4th ed. USA: Pernerbit Erlangga.

Myers, B.A., 1990. Toxonomies of visual programing and program visualization. *Journal of Visual Languages and Computing*, pp.97-123.

Nvidia, 2012. *CUDA*. [Online] Available at: http://developer.nvidia.com/what-cuda [Accessed 30 april 2012].

nvidia, 2012. *CUDA C Programming Guide v.4.2*. Canada: Santa CLara.

nvidia, 2013. *CUDA Community Showcase*. [Online] Available at: http://www.nvidia.com/object/cuda_showcase_html.html [Accessed 1 April 2013].

Padua, D., 2011. *Encyclopedi a of Parallel Computing*. london: Springer.

Prmigiani, A., 2011. Lattice Boltzmann *calculations of reactive multiphase flows in porous media*. Thesis. italia: http://archive-ouverte.unige.ch Universit´e de Gen`eve.

Pullan, G., 2008. *A 2D Lattice Botlzmann flow solver demo*. [Online] Available at: http://www.many-core.group.cam.ac.uk/projects/LBdemo.shtml [Accessed 24 September 2011].

Rohmadani,, Ramadi janti, & M., R., 2010. *Visualisasi Gerakan Air 3 Dimensi Menggunakan Metode* Lattice Boltzmann. Skripsi. surabaya: Institut Teknologi Sepuluh Nopember Institut Teknologi Sepuluh Nopember.

Schreiber, M., 2010. *GPU based simulation adn visulalization of fluids with free surfaces*. german: Universitas Munchen.

Shan, X. & Chen, H., 1993. Lattice Boltzmann model for simulating flows with multiple phases and components. *Physical Review E*, 47(3), p.815–1819.

Shreiner, D., 2010. *OpenGL Programming Guide The Official Guide*. Boston: Pearson Education Inc.

Strzodka, R., Doggett, M. & Kolb, A., 2005. Cientific Computation for Simulationson Programmable Graphics Hardware. *Simulation Modelling Practice and Theory*, pp.667-81.

Sukop, M.C. & Thorne, Jr., D.T., 2007. Lattice Boltzmann *Modeling*. berlin: Springer.

Thurey, N., 2007. *Physically based Animation of Free Surface*. Doktor-Ingenieur. Erlangen,: University of Erlangen-Nuremberg.

Tomas, G. & Ueberhuber, c.W., 1994. *visualization scientific parallel programs*. german: Springer-Verlag.

Wehner, M., 2009. A real cloud computer. *IEEE Spectrum*, p.24 – 29.

Weiskopf, D., 2006. *GPU-Based Interactive Visualization techniques*. canada: Springer.

Wijaya, D., 2004. *SPECIAL EFFECTS HISTORY AND TECHNIQUES*. [Online] Available at: www.escaeva.com [Accessed 12 September 2011].

Zhang, H. et al., 2008. Modeling and Visualization of Tsunamis. *Pure appl. geophys.*, 2 april. pp.475-96.

LAMPIRAN

Tabel hasil uji pengambilan rata waktu untuk kasus pertama pada PC

Jumlah lattice 320x112

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	21.07	41.34	83.53
2	21.16	41.55	83.33
3	21.21	42.07	83.12
rata rata	21.15	41.65	83.33

Jumlah lattice 640X224

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	196.29	390.48	769.13
2	193.29	385.34	783.85
3	195.83	385.63	772.92
rata rata	195.14	387.15	775.30

Jumlah lattice 960x339

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	435.05	862.51	1761.20
2	439.30	878.13	1764.34
3	437.03	882.53	1755.18
rata rata	437.13	874.39	1760.24

Tabel hasil uji pengambilan rata waktu untuk kasus pertama pada GPU

Jumlah lattice 320x112

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.08	33.66	67.01
2	16.99	33.67	67.01
3	17.01	33.66	67.01
rata rata	17.03	33.66	67.01

Jumlah lattice 640X224

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.05	33.68	68.29
2	17.00	33.68	67.01
3	17.00	33.68	67.02
rata rata	17.02	33.68	67.44

Jumlah lattice 960x339

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.04	33.68	68.10
2	17.02	33.69	67.16
3	17.04	33.70	67.03
rata rata	17.03	33.69	67.43

Setting

Jumlah lattice X	:	320	**Jumlah lattice X**	:	640	**Jumlah lattice X**	:	960
Jumlah lattice Y	:	112	**Jumlah lattice Y**	:	224	**Jumlah lattice Y**	:	339

POSISI FLUID			**POSISI FLUID**			**POSISI FLUID**		
Radius	:	25	**Radius**	:	25	**Radius**	:	25
Posisi X	:	150	**Posisi X**	:	150	**Posisi X**	:	150
Posisi Y	:	80	**Posisi Y**	:	80	**Posisi Y**	:	80

Lampiran 2

Tabel hasil uji pengambilan rata waktu untuk kasus Kedua pada PC

160*160

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	21.29	42.17	83.93
2	20.04	41.27	82.14
3	21.15	41.86	83.22
rata rata	20.83	41.77	83.10

320*320

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	205.60	407.41	813.81
2	212.12	406.31	811.91
3	199.10	403.14	805.41
rata rata	205.61	405.62	810.38

640*640

percobaan	iterasi		
	500	700	1000
	waktu /sc	waktu /sc	waktu /sc
1	1010.97	1415.11	2021.16
2	1011.99	1416.10	2022.23
3	1009.13	1412.27	2017.83
rata rata	1010.70	1414.49	2020.41

Tabel hasil uji pengambilan rata waktu untuk kasus Kedua pada GPU

160*160

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.60	34.43	69.02
2	17.00	33.75	67.11
3	17.02	33.74	67.08
rata rata	17.21	33.97	67.74

320*320

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.30	33.98	67.37
2	18.01	35.66	70.57
3	17.90	35.40	70.59
rata rata	17.74	35.01	69.51

640*640

percobaan	iterasi		
	500	700	1000
	waktu /sc	waktu /sc	waktu /sc
1	8.71	12.05	17.05
2	8.74	12.07	17.07
3	8.78	12.20	17.20
rata rata	8.74	12.11	17.11

Jumlah lattice X	:	**160**	Jumlah lattice X	:	**320**	Jumlah lattice X	:	**960**
Jumlah lattice Y	:	160	**Jumlah lattice Y**	:	320	**Jumlah lattice Y**	:	339
POSISI FLUID 1 DAN FLUID 2			**POSISI FLUID 1 DAN FLUID 2**			**POSISI FLUID 1 DAN FLUID 2**		
Radius	:	20, 70	**Radius**	:	50,70	**Radius**	:	50,70
Posisi X	:	120, Y	**Posisi X**	:	150,Y	**Posisi X**	:	150,Y
Posisi Y	:	80, 30	**Posisi Y**	:	150,30	**Posisi Y**	:	150,30

Tabel hasil uji pengambilan rata waktu untuk kasus ketiga pada PC

Jumlah Lattice 960x336

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	350.11	699.11	1397.45
2	357.82	715.66	1430.71
3	349.64	698.77	1397.03
rata rata	352.52	704.51	1408.40

Jumlah Lattice 640x220

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	158.75	316.9	633.4
2	160.40	320.25	640.4
3	158.19	316.27	632.09
rata rata	159.11	317.81	635.30

Jumlah Lattice 320x112

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.14	33.81	67.15
2	17.17	33.02	67.16
3	17.2	33.89	67.23
rata rata	17.17	33.57	67.18

Tabel hasil uji pengambilan rata waktu untuk kasus ketiga pada GPU

Jumlah Lattice 320x112

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.61	34.29	67.63
2	17.00	33.67	67.61
3	17.59	34.5	68.79
rata rata	17.40	34.15	68.01

Jumlah Lattice 640x224

percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	17.06	74.35	147.32
2	24.30	43.34	100.77
3	17.07	33.74	67.09
rata rata	19.48	50.48	105.06

Jumlah Lattice 960x336

Percobaan	iterasi		
	1000	2000	4000
	waktu /sc	waktu /sc	waktu /sc
1	71.04	33.74	67.08
2	17.02	33.69	67.09
3	17.03	33.7	67.04
rata rata	35.03	33.71	67.07

Jumlah lattice X	:	320	**Jumlah lattice X**	:	640	**Jumlah lattice X**	:	960
Jumlah lattice Y	:	112	**Jumlah lattice Y**	:	224	**Jumlah lattice Y**	:	336
POSISI solid atas			**POSISI bawah**			**POSISI lingkaran**		
Radius	:	100	**Radius**	:	100	**Radius**	:	20
Posisi X	:	X	**Posisi X**	:	X	**Posisi X**	:	X/2
Posisi Y	:	0	**Posisi Y**	:	Y-1	**Posisi Y**	:	Y-50

Kode program untuk GPU

Kondisi dua fase pada kasus satu dan dua

```
///////////////////////////////////////////////////
// multiphase arifiayanto hadinegoro – JUNI 2013
// CUDA version
// Graham Pullan-Oct 2008
// D2Q9
//
//      f6  f2  f5
//      \   |   /
//       \  |  /
//        \ | /
//   f3---|--- f1
//        /|\
//       / | \        and f0 for the rest (zero) velocity
//      /  |  \
//   f7   f4   f8
//
/////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include <cutil.h>
#include <cuda_runtime_api.h>
#include <cuda_gl_interop.h>

#define TILE_I 16
#define TILE_J 8
#define I2D(ni,i,j) (((ni)*(j))+i)
// OpenGL pixel buffer object and texture //
GLuint gl_PBO,gl_Tex;

// arrays on host //
float *f0,*f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8,*plot;
int *solid;
unsigned int *cmap_rgba,*plot_rgba; // rgba arrays for plotting
float *ux,*uy,*rho,*phi;

// arrays on device //
float *f0_data,*f1_data,*f2_data,*f3_data,*f4_data;
float *f5_data,*f6_data,*f7_data,*f8_data,*plot_data;
int *solid_data;
unsigned int *cmap_rgba_data,*plot_rgba_data;
float *ux_data,*uy_data,*rho_data,*phi_data;
float *tmpf0,*tmpf1,*tmpf2,*tmpf3,*tmpf4,*tmpf5,*tmpf6,*tmpf7,*tmpf8; //
      yang aku tambahkan
// textures on device //
```

```
texture<float,2> f1_tex,f2_tex,f3_tex,f4_tex,
f5_tex,f6_tex,f7_tex,f8_tex;

// CUDA special format arrays on device //
cudaArray *f1_array,*f2_array,*f3_array,*f4_array;
cudaArray *f5_array,*f6_array,*f7_array,*f8_array;

// scalars //
float tau,faceq1,faceq2,faceq3;
float vxin,roout;
float width,height;
float minvar,maxvar;

int ni,nj,i,j,i0,i1;
int nsolid,nstep,nsteps,ncol;
int ipos_old,jpos_old,draw_solid_flag,i_steps;

size_t pitch;
int ni_=320;
int nj_=112;
float M_PI=3.14159265358979;
float rhoh=1.6429; //high density fluid (
float rhol=0.0734; //lowdensity fluid
float ifaceW=3.0;//interface width
float G=-6.0;//interparticular interaction potential
float body_force=1.0e-4; //NILAI GRAVITASI
float body_force_dir=0;//arah gravitasi (0=down 90=right 180=top
      270=left)
float body_force_x,body_force_y;
int dr=25;                          //droplet radius
int dx=150;//150; //droplet position x (for multiphase model)
int dy=40;//80; //droplet position y (for multiphase model)
int bot,top,lef,rig,fro,bac;
float ux2,uy2,uz2,uxy,uxz,uyz,uxy2,uxz2,uxyz2;
float dx2,dy2,dx3,dy3,radius,radius2,tmp,tmp2,tmp1;
int iterasi=4000;
unsigned int Timer=0;
int iter1=1000;
int iter2=2000;
//
// OpenGL function prototypes
//
void display(void);
void resize(int w,int h);
//
// CUDA kernel prototypes
//
__global__ void stream_kernel (int pitch,float *f1_data,float *f2_data,
 float *f3_data,float *f4_data,float *f5_data,float *f6_data,
 float *f7_data,float *f8_data, int *solid_data  );

__global__ void collide_kernel (int pitch,float tau,float faceq1,float
      faceq2,float faceq3, float *f0_data,float *f1_data,float *f2_data,
```

```
	oat *f3_data,float *f4_data,float *f5_data,float *f6_data, float
	*f7_data,float *f8_data,float *plot_data, float *ux_data,float
	*uy_data,float *rho_data,float *phi_data, float body_force_x,float
	body_force_y,float G,float rhoh, int ni,int nj);

__global__ void apply_Periodic_BC_kernel (int ni,int nj,int pitch, float
	*f2_data,float *f4_data,float *f5_data, float *f6_data,float
	*f7_data,float *f8_data);

__global__ void apply_BCs_kernel (int ni,int nj,int pitch,float
	vxin,float roout, float faceq2,float faceq3, float *f0_data,float
	*f1_data,float *f2_data, float *f3_data,float *f4_data,float
	*f5_data, float *f6_data,float *f7_data,float
	*f8_data,int*solid_data);

__global__ void get_rgba_kernel (int pitch,int ncol,float minvar,float
	maxvar, float *plot_data, unsigned int *plot_rgba_data,unsigned int
	*cmap_rgba_data, int *solid_data);

//
// CUDA kernel C wrappers
//
void stream(void);
void collide(void);
void apply_Periodic_BC(void);
void apply_BCs(void);
void get_rgba(void);
void garis_batas(void);
void droplet(void);

void inisial(void);
///
int main(int argc,char **argv)
{
int totpoints,i;
float rcol,gcol,bcol,vx,vy;

FILE *fp_col;
cudaChannelFormatDesc desc;

// The following parameters are usually read from a file,but
// hard code them for the demo:
ni=ni_;//320*3;
nj=nj_;//112*3;
vxin=0.00;
roout=1.0;
tau=1.0;
	vx=0.0;
	vy=0.0;
	//i_steps=0;
// End of parameter list
totpoints=ni*nj;
// Write parameters to screen
```

```
printf ("ni=%d\n",ni);
printf ("nj=%d\n",nj);
printf ("total pixel=%d\n",(ni_*nj_));
printf ("roout=%f\n",roout);
printf ("tau=%f\n",tau);
//
// allocate memory on host
//
f0=(float *)malloc(ni*nj*sizeof(float));
f1=(float *)malloc(ni*nj*sizeof(float));
f2=(float *)malloc(ni*nj*sizeof(float));
f3=(float *)malloc(ni*nj*sizeof(float));
f4=(float *)malloc(ni*nj*sizeof(float));
f5=(float *)malloc(ni*nj*sizeof(float));
f6=(float *)malloc(ni*nj*sizeof(float));
f7=(float *)malloc(ni*nj*sizeof(float));
f8=(float *)malloc(ni*nj*sizeof(float));
plot=(float *)malloc(ni*nj*sizeof(float));

      rho=(float *)malloc(ni*nj*sizeof(float));
      phi=(float *)malloc(ni*nj*sizeof(float));
      ux =(float *)malloc(ni*nj*sizeof(float));
      uy =(float *)malloc(ni*nj*sizeof(float));

solid=(int *)malloc(ni*nj*sizeof(int));

plot_rgba=(unsigned int*)malloc(ni*nj*sizeof(unsigned int));

//
// allocate memory on device
//
CUDA_SAFE_CALL(cudaMallocPitch((void
      **)&f0_data,&pitch,sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f1_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f2_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f3_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f4_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f5_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f6_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f7_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&f8_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&plot_data,&pitch,
      sizeof(float)*ni,nj));
      /// aku tambahin
```

```
CUDA_SAFE_CALL(cudaMallocPitch((void **)&rho_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&phi_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&ux_data,&pitch,
      sizeof(float)*ni,nj));
CUDA_SAFE_CALL(cudaMallocPitch((void **)&uy_data,&pitch,
      sizeof(float)*ni,nj));

CUDA_SAFE_CALL(cudaMallocPitch((void **)&solid_data,&pitch,
      sizeof(int)*ni,nj));

CUDA_SAFE_CALL(cudaMallocPitch((void **)&plot_data,&pitch,
      sizeof(float)*ni,nj));

desc=cudaCreateChannelDesc<float>();
CUDA_SAFE_CALL(cudaMallocArray(&f1_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f2_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f3_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f4_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f5_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f6_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f7_array,&desc,ni,nj));
CUDA_SAFE_CALL(cudaMallocArray(&f8_array,&desc,ni,nj));
//
// Some factors used in equilibrium f's
//
faceq1=4.f/9.f;
faceq2=1.f/9.f;
faceq3=1.f/36.f;

      // aku tambahin
            body_force_x= body_force*sin(body_force_dir/(180.0/M_PI));
            body_force_y=-body_force*cos(body_force_dir/(180.0/M_PI));

            for (i=0; i<totpoints; i++) {

            if (solid[i]!=1)
            {rho[i]= 0.2*(rhoh-rhol)+rhol;}
            if (solid[i]=1)
            {
            rho[i]=rhol;
            }}
            for (j=0; j<nj; j++) {
            for (i=0; i<ni; i++) {
      i0=I2D(ni,i,j);
       solid[i0]=1;
      //bola
      dx2=((float)(i-dx))*((float)(i-dx));
dy2=((float)(j-dy))*((float)(j-dy));
      // dataran
      dx3=ni;
      dy3=((float)(j-30))*((float)(j-30));
```

```
      radius2 =sqrtf(dx3+dy3 );
      radius =sqrtf(dx2+dy2 );
tmp1=0.5*( (rhoh+rhol)-(rhoh-rhol)*tanh ((radius-dr) / ifaceW*2.0) );
tmp2 =      0;//0.5*( (rhoh+rhol) -(rhoh-rhol)*tanh (radius2-49 /
      ifaceW*2.0) );
tmp=tmp1+tmp2;
if (tmp > rhol) rho[i0]=tmp;
      }}
// Initialise f's
            for (i=0; i<totpoints; i++) {
                  solid[i0]=1;
                  ux[i]=vx;
                  uy[i]=vy;
f0[i]=faceq1*rho[i]*(1.f+1.5f*(vx*vx+vy*vy));
f1[i]=faceq2*rho[i]*(1.f+3.f*vx+4.5f*vx*vx-1.5f*(vx*vx+vy*vy));
f2[i]=faceq2*rho[i]*(1.f+3.f*vy+4.5f*vy*vy-1.5f*(vx*vx+vy*vy));
f3[i]=faceq2*rho[i]*(1.f+3.f*vx+4.5f*vx*vx-1.5f*(vx*vx+vy*vy));
f4[i]=faceq2*rho[i]*(1.f+3.f*vy+4.5f*vy*vy-1.5f*(vx*vx+vy*vy));
f5[i]=faceq3*rho[i]*(1.f+3.f*(vx+vy) + 4.5f*(vx+vy) *(vx+vy)-
      1.5f*(vx*vx+vy*vy));
f6[i]=faceq3*rho[i]*(1.f+3.f*(-vx+vy)+4.5f*(-vx+vy)*(-vx+vy)-
      1.5f*(vx*vx+vy*vy));
f7[i]=faceq3*rho[i]*(1.f+3.f*(-vx-vy)+4.5f*(-vx-vy)*(-vx-vy)-
      1.5f*(vx*vx+vy*vy));
f8[i]=faceq3*rho[i]*(1.f+3.f*(vx-vy) + 4.5f*(vx-vy) *(vx-vy)-
      1.5f*(vx*vx+vy*vy));
      plot[i]=rho[i];
      //solid[i]=1;
}

//
// Read in colourmap data for OpenGL display
//
fp_col=fopen("cmap.dat","r");
if (fp_col==NULL) {
      printf("Error: can't open cmap.dat \n");
      return 1;
}
fscanf (fp_col,"%d",&ncol);
cmap_rgba=(unsigned int *)malloc(ncol*sizeof(unsigned int));
CUDA_SAFE_CALL(cudaMalloc((void **)&cmap_rgba_data,
 sizeof(unsigned int)*ncol));

for (i=0;i<ncol;i++){
      fscanf(fp_col,"%f%f%f",&rcol,&gcol,&bcol);
      cmap_rgba[i]=((int)(255.0f) << 24) | // convert colourmap to int
      ((int)(bcol*255.0f) << 16) |
      ((int)(gcol*255.0f) <<8) |
      ((int)(rcol*255.0f) <<0);
}
fclose(fp_col);
            // aku tambahin
```

```
	for (i=0; i<ni; i++) {
		i0 =I2D(ni,i,0);
		i1 =I2D(ni,i,nj-1);
solid[i0]=0;
 solid[i1]=0;
}
	for (j=0; j<nj; j++) {
		i0 =I2D(ni,0,j);
		i1 =I2D(ni,ni-1,j);
solid[i0]=0;
solid[i1]=0;
}

	/////////////

//
// Transfer initial data to device
//
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f0_data,pitch,(void
	*)f0,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f1_data,pitch,(void
	*)f1,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f2_data,pitch,(void
	*)f2,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f3_data,pitch,(void
	*)f3,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f4_data,pitch,(void
	*)f4,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f5_data,pitch,(void
	*)f5,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f6_data,pitch,(void
	*)f6,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f7_data,pitch,(void
	*)f7,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)f8_data,pitch,(void
	*)f8,sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)plot_data,pitch,(void *)plot,
	sizeof(float)*ni,sizeof(float)*ni,nj,cudaMemcpyHostToDevice));
CUDA_SAFE_CALL(cudaMemcpy2D((void *)solid_data,pitch,(void
	*)solid,sizeof(int)*ni,sizeof(int)*ni,nj,cudaMemcpyHostToDevice));

	// aku tambahin
	CUDA_SAFE_CALL(cudaMemcpy2D((void *)rho_data,pitch,(void *)rho,
sizeof(int)*ni,sizeof(int)*ni,nj,
cudaMemcpyHostToDevice));
	CUDA_SAFE_CALL(cudaMemcpy2D((void *)phi_data,pitch,(void *)phi,
sizeof(int)*ni,sizeof(int)*ni,nj,
cudaMemcpyHostToDevice));
	CUDA_SAFE_CALL(cudaMemcpy2D((void *)ux_data,pitch,(void *)ux,
sizeof(int)*ni,sizeof(int)*ni,nj,
cudaMemcpyHostToDevice));
	CUDA_SAFE_CALL(cudaMemcpy2D((void *)uy_data,pitch,(void *)uy,
sizeof(int)*ni,sizeof(int)*ni,nj,
```

```
cudaMemcpyHostToDevice));

CUDA_SAFE_CALL(cudaMemcpy((void *)cmap_rgba_data,
 (void *)cmap_rgba,sizeof(unsignedint)*ncol,cudaMemcpyHostToDevice));
      //garis_batas();
      // Set up CUDA Timer
      cutCreateTimer(&Timer);
      cutResetTimer(Timer);
      cutStartTimer(Timer);

//
// Iinitialise OpenGL display-use glut
//
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(ni,nj); // Window of ni x nj pixels
glutInitWindowPosition(50,500); // Window position
glutCreateWindow("CUDA 2D LB" ); // Window title

printf("Loading extensions: %s\n",glewGetErrorString(glewInit()));
if(!glewIsSupported(
"GL_VERSION_2_0 "
"GL_ARB_pixel_buffer_object "
"GL_EXT_framebuffer_object "
)){
fprintf(stderr,"ERROR: Support for necessary OpenGL extensions
      missing.");
fflush(stderr);
return 1;
}

// Set up view
glClearColor(0.0,0.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0,ni,0.,nj,-200.0,200.0);

// Create texture and bind to gl_Tex
glEnable(GL_TEXTURE_2D);
glGenTextures(1,&gl_Tex); // Generate 2D texture
glBindTexture(GL_TEXTURE_2D,gl_Tex);// bind to gl_Tex
// texture properties:
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA8,ni,nj,0,
 GL_RGBA,GL_UNSIGNED_BYTE,NULL);
printf("Texture created.\n");

// Create pixel buffer object and bind to gl_PBO
glGenBuffers(1,&gl_PBO);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB,gl_PBO);
```

```
glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB,pitch*nj,NULL,GL_STREAM_COPY);
CUDA_SAFE_CALL( cudaGLRegisterBufferObject(gl_PBO) );
printf("Buffer created.\n");
printf("Starting GLUT main loop...\n");
glutDisplayFunc(display);
glutReshapeFunc(resize);
glutIdleFunc(display);
glutMainLoop();
return 0;
}
////
__global__ void stream_kernel (int pitch,float *f1_data,float *f2_data,
                float *f3_data,float *f4_data,float *f5_data, float
                *f6_data,float *f7_data,float *f8_data, int *solid_data)
// CUDA kernel

{
int i,j,i2d;

i=blockIdx.x*TILE_I+threadIdx.x;
j=blockIdx.y*TILE_J+threadIdx.y;
i2d=i+j*pitch/sizeof(float);
// look up the adjacent f's needed for streaming using textures
// i.e. gather from textures,write to device memory: f1_data,etc
f1_data[i2d]=tex2D(f1_tex,(float) (i-1),(float) j);
f2_data[i2d]=tex2D(f2_tex,(float) i,(float) (j-1));
f3_data[i2d]=tex2D(f3_tex,(float) (i+1),(float) j);
f4_data[i2d]=tex2D(f4_tex,(float) i,(float) (j+1));
f5_data[i2d]=tex2D(f5_tex,(float) (i-1),(float) (j-1));
f6_data[i2d]=tex2D(f6_tex,(float) (i+1),(float) (j-1));
f7_data[i2d]=tex2D(f7_tex,(float) (i+1),(float) (j+1));
f8_data[i2d]=tex2D(f8_tex,(float) (i-1),(float) (j+1));
      }
void stream(void)
// C wrapper
{
// Device-to-device mem-copies to transfer data from linear memory
      (f1_data)
// to CUDA format memory (f1_array) so we can use these in textures
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f1_array,0,0,(void *)f1_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f2_array,0,0,(void *)f2_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f3_array,0,0,(void *)f3_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f4_array,0,0,(void *)f4_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f5_array,0,0,(void *)f5_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f6_array,0,0,(void *)f6_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f7_array,0,0,(void *)f7_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
```

```
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f8_array,0,0,(void *)f8_data,pitch,
                  sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));


// Tell CUDA that we want to use f1_array etc as textures. Also
// define what type of interpolation we want (nearest point)
f1_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f1_tex,f1_array));

f2_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f2_tex,f2_array));

f3_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f3_tex,f3_array));

f4_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f4_tex,f4_array));

f5_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f5_tex,f5_array));

f6_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f6_tex,f6_array));

f7_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f7_tex,f7_array));

f8_tex.filterMode=cudaFilterModePoint;
CUDA_SAFE_CALL(cudaBindTextureToArray(f8_tex,f8_array));

dim3 grid=dim3(ni/TILE_I,nj/TILE_J);
dim3 block=dim3(TILE_I,TILE_J);

stream_kernel<<<grid,block>>>(pitch,f1_data,f2_data,f3_data,f4_data,
 f5_data,f6_data,f7_data,f8_data,solid_data);

CUT_CHECK_ERROR("stream failed.");

CUDA_SAFE_CALL(cudaUnbindTexture(f1_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f2_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f3_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f4_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f5_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f6_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f7_tex));
CUDA_SAFE_CALL(cudaUnbindTexture(f8_tex));
}

/////////////////////
__global__ void collide_kernel (int pitch,float tau,float faceq1,float
                faceq2,float faceq3, float *f0_data,float *f1_data,float
                *f2_data, float *f3_data,float *f4_data,float
                *f5_data,float *f6_data, float *f7_data,float
```

```
                  *f8_data,float *plot_data, float *ux_data,
                  float*uy_data,float *rho_data,float *phi_data, float
                  body_force_x,float body_force_y,float G,float rhoh, int
                  ni,int nj)
// CUDA kernel

{
int i,j,i2d;
float ro,vx,vy,v_sq_term;
      //float rtau,rtau1;
float f0now,f1now,f2now,f3now,f4now,f5now,f6now,f7now,f8now;
float f0eq,f1eq,f2eq,f3eq,f4eq,f5eq,f6eq,f7eq,f8eq;
float grad_phi_x,grad_phi_y;

      int im1,ip1,jm1,jp1;
      float ux2,uy2,uxy,uxy2;


i=blockIdx.x*TILE_I+threadIdx.x;
j=blockIdx.y*TILE_J+threadIdx.y;
i2d=i+j*pitch/sizeof(float);
// Read all f's and store in registers
f0now=f0_data[i2d];
f1now=f1_data[i2d];
f2now=f2_data[i2d];
f3now=f3_data[i2d];
f4now=f4_data[i2d];
f5now=f5_data[i2d];
f6now=f6_data[i2d];
f7now=f7_data[i2d];
f8now=f8_data[i2d];
// Macroscopic flow props:
ro= f0now+f1now+f2now+f3now+f4now+f5now+f6now+f7now+f8now;
vx=(f1now-f3now+f5now-f6now-f7now+f8now)/ro;
vy=(f2now-f4now+f5now+f6now-f7now-f8now)/ro;
      // aku tambahin
            ux_data[i2d]=vx;
            uy_data[i2d]=vy;

      ux_data[i2d] += tau*body_force_x;
uy_data[i2d] += tau*body_force_y;

      rho_data[i2d]=ro;

      phi_data[i2d]=1.0-exp(-rho_data[i2d]);
// Set plotting variable to velocity magnitude
plot_data[i2d] =sqrt(rho_data[i2d]);
      ////////////////
      jm1=j-1;
      jp1=j+1;
      if (j==0) jm1=0;
      if (j==(nj-1)) jp1=nj-1;
      im1=i-1;
```

```
        ip1=i+1;
       if (i==0) im1=0;
       if (i==(ni-1)) ip1=ni-1;

grad_phi_x=faceq2*(phi_data[ip1+j*pitch/sizeof(float)]-
      phi_data[im1+j*pitch/sizeof(float)]);
grad_phi_y=faceq2*(phi_data[i+jp1*pitch/sizeof(float)]-
      phi_data[i+jm1*pitch/sizeof(float)]);//top dan bot

grad_phi_x += faceq3*(phi_data[ip1+jp1*pitch/sizeof(float)]-
      phi_data[im1+jp1*pitch/sizeof(float)]+phi_data[ip1+jm1*pitch/sizeof
      (float)]-phi_data[im1+jm1*pitch/sizeof(float)]);
grad_phi_y +=
      faceq3*(phi_data[ip1+jp1*pitch/sizeof(float)]+phi_data[im1+jp1*pitc
      h/sizeof(float)]-phi_data[im1+jm1*pitch/sizeof(float)]-
      phi_data[ip1+jm1*pitch/sizeof(float)]);
      ux_data[i2d] += tau*(-G*phi_data[i2d]*grad_phi_x)/rho_data[i2d];
uy_data[i2d] += tau*(-G*phi_data[i2d]*grad_phi_y)/rho_data[i2d];

vx=ux_data[i2d] ;
      vy=uy_data[i2d] ;
// Calculate equilibrium f's
            ux2         = vx*vx;
            uy2         = vy*vy;
            uxy2        = ux2+uy2;
            uxy         = 1.3*vx*vy;
       v_sq_term=1.5f*(vx*vx+vy*vy);
      // Evaluate the local equilibrium f values in all directions
f0eq=(rho_data[i2d]*faceq1*(1.f-v_sq_term)) ;
f1eq=(rho_data[i2d]*faceq2*(1.f+3.f*vx+ 4.5f*ux2- v_sq_term)) ;
f2eq=(rho_data[i2d]*faceq2*(1.f+3.f*vy+ 4.5f*uy2- v_sq_term)) ;//
f3eq=(rho_data[i2d]*faceq2*(1.f-3.f*vx+ 4.5f*ux2- v_sq_term)) ; //
f4eq=(rho_data[i2d]*faceq2*(1.f-3.f*vy+ 4.5f*uy2- v_sq_term)) ;//
f5eq=(rho_data[i2d]*faceq3*(1.f+3.f* (+vx+vy) + 4.5f*(uxy2+uxy)-
      v_sq_term)) ;     //
f6eq=(rho_data[i2d]*faceq3*(1.f+3.f* (-vx+vy) + 4.5f* (uxy2-uxy) -
      v_sq_term)) ;//
f7eq=(rho_data[i2d]*faceq3*(1.f+3.f* (-vx-vy) + 4.5f* (uxy2+uxy) -
      v_sq_term)) ;//
f8eq=(rho_data[i2d]*faceq3*(1.f+3.f* (+vx-vy) + 4.5f* (uxy2-uxy) -
      v_sq_term)) ;
       // Do collisions
      f0_data[i2d]=f0_data[i2d]-(f0_data[i2d]-f0eq)/tau;
      f1_data[i2d]=f1_data[i2d]-(f1_data[i2d]-f1eq)/tau;
      f2_data[i2d]=f2_data[i2d]-(f2_data[i2d]-f2eq)/tau;
      f3_data[i2d]=f3_data[i2d]-(f3_data[i2d]-f3eq)/tau;
      f4_data[i2d]=f4_data[i2d]-(f4_data[i2d]-f4eq)/tau;
      f5_data[i2d]=f5_data[i2d]-(f5_data[i2d]-f5eq)/tau;
      f6_data[i2d]=f6_data[i2d]-(f6_data[i2d]-f6eq)/tau;
      f7_data[i2d]=f7_data[i2d]-(f7_data[i2d]-f7eq)/tau;
      f8_data[i2d]=f8_data[i2d]-(f8_data[i2d]-f8eq)/tau;

      }
```

```
void collide(void)
// C wrapper
{
dim3 grid=dim3(ni/TILE_I,nj/TILE_J);
dim3 block=dim3(TILE_I,TILE_J);
collide_kernel<<<grid,block>>>(pitch,tau,faceq1,faceq2,faceq3,
                  f0_data,f1_data,f2_data,f3_data,f4_data,f5_data,f6_data
                  ,f7_data,f8_data,plot_data,ux_data,uy_data,rho_data,phi
                  _data,body_force_x,body_force_y,G ,rhoh, ni,nj);

CUT_CHECK_ERROR("collide failed.");
}

/////
__global__ void apply_BCs_kernel (int ni,int nj,int pitch,float
      vxin,float roout,
 float faceq2,float faceq3,
float *f0_data,float *f1_data,float *f2_data,
      float *f3_data,float *f4_data,float *f5_data,
 float *f6_data,float *f7_data,float *f8_data, int*solid_data)
// CUDA kernel all BC's apart from periodic boundaries:
{
int i,j,i2d;// i2d2;
//float v_sq_term;
float f1old,f2old,f3old,f4old,f5old,f6old,f7old,f8old;

i=blockIdx.x*TILE_I+threadIdx.x;
j=blockIdx.y*TILE_J+threadIdx.y;
i2d=i+j*pitch/sizeof(float);
// Solid BC: "bounce-back"
if (solid_data[i2d] == 0) {
f1old=f1_data[i2d];
f2old=f2_data[i2d];
f3old=f3_data[i2d];
f4old=f4_data[i2d];
f5old=f5_data[i2d];
f6old=f6_data[i2d];
f7old=f7_data[i2d];
f8old=f8_data[i2d];

f1_data[i2d]=f3old;
f2_data[i2d]=f4old;
f3_data[i2d]=f1old;
f4_data[i2d]=f2old;
f5_data[i2d]=f7old;
f6_data[i2d]=f8old;
f7_data[i2d]=f5old;
f8_data[i2d]=f6old;
}

}
```

```
void apply_BCs(void)
// C wrapper
{
dim3 grid=dim3(ni/TILE_I,nj/TILE_J);
dim3 block=dim3(TILE_I,TILE_J);

apply_BCs_kernel<<<grid,block>>>(ni,nj,pitch,vxin,roout,faceq2,faceq3,
      f0_data,f1_data,f2_data,
      f3_data,f4_data,f5_data,f6_data,f7_data,f8_data, solid_data);

CUT_CHECK_ERROR("apply_BCs failed.");
}
/////

__global__ void get_rgba_kernel (int pitch,int ncol,float minvar,float
      maxvar, float *plot_data, unsigned int *plot_rgba_data, unsigned
      int *cmap_rgba_data, int *solid_data)

// CUDA kernel to fill plot_rgba_data array for plotting

{
int i,j,i2d,icol;
float frac;
i=blockIdx.x*TILE_I+threadIdx.x;
j=blockIdx.y*TILE_J+threadIdx.y;
i2d=i+j*pitch/sizeof(float);
frac=(plot_data[i2d]-minvar)/(maxvar-minvar);
icol=(int)(frac*(float)ncol);
plot_rgba_data[i2d]=solid_data[i2d]*cmap_rgba_data[icol];
}
void get_rgba(void)
// C wrapper
{
dim3 grid=dim3(ni/TILE_I,nj/TILE_J);
dim3 block=dim3(TILE_I,TILE_J);
get_rgba_kernel<<<grid,block>>>(pitch,ncol,minvar,maxvar,
                         plot_data,plot_rgba_data,cmap_rgba_data,
 solid_data);
CUT_CHECK_ERROR("get_rgba failed.");
}
////////

void display(void)
// This function is called automatically,over and over again, by GLUT
{
// Set upper and lower limits for plotting
minvar=rhol;
maxvar=rhoh;
      // aku tambahin
            i_steps++;
            apply_BCs();
            collide();
            stream();
```

```
      // For plotting,map the plot_rgba_data array to the
// gl_PBO pixel buffer
CUDA_SAFE_CALL(cudaGLMapBufferObject((void**)&plot_rgba_data,gl_PBO));

// Fill the plot_rgba_data array (and the pixel buffer)
get_rgba();
CUDA_SAFE_CALL(cudaGLUnmapBufferObject(gl_PBO));

// Copy the pixel buffer to the texture,ready to display
 glTexSubImage2D(GL_TEXTURE_2D,0,0,0,ni,nj,GL_RGBA,GL_UNSIGNED_BYTE,0);
// Render one quad to the screen and colour it using our texture
// i.e. plot our plotvar data to the screen
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_QUADS);
glTexCoord2f (0.0,0.0);
glVertex3f (0.0,0.0,0.0);
glTexCoord2f (1.0,0.0);
glVertex3f (ni,0.0,0.0);
glTexCoord2f (1.0,1.0);
glVertex3f (ni,nj,0.0);
glTexCoord2f (0.0,1.0);
glVertex3f (0.0,nj,0.0);
glEnd();
glutSwapBuffers();
      //TIMER
      if (i_steps%iterasi == iter1) {

      printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f perSecond
      \n",i_steps,0.001*cutGetTimerValue(Timer));}
      if (i_steps%iterasi == iter2) {
      printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f perSecond
      \n",i_steps,0.001*cutGetTimerValue(Timer));}
      if (i_steps%iterasi == 0) {
      printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f perSecond
      \n",i_steps,0.001*cutGetTimerValue(Timer));
      system("PAUSE");
      exit(0);}
}
/////
void resize(int w,int h)
// GLUT resize callback to allow us to change the window size
{
 width=w;
 height=h;
 glViewport (0,0,w,h);
 glMatrixMode (GL_PROJECTION);
 glLoadIdentity ();
 glOrtho (0.,ni,0.,nj,-200. ,200.);
 glMatrixMode (GL_MODELVIEW);
 glLoadIdentity ();
}
```

Kode program untuk GPU

Kondisi satu fase pada kasus tiga

```
///////////////////
// arifiyanto hadinegoro juni 2013
// single phase code
// edit from kode # Graham Pullan-Oct 2008 #
//
//      f6  f2   f5
//        \  |  /
//         \ | /
//          \|/
//      f3---|--- f1
//          /|\
//         / | \       and f0 for the rest (zero) velocity
//        /  |  \
//      f7  f4   f8
//
/////////////////////////////////////////////////////////////////////////
#include <stdio.h>
#include <stdlib.h>
#include <GL/glew.h>
#include <GL/glut.h>
#include <cutil.h>
#include <cuda_runtime_api.h>
#include <cuda_gl_interop.h>

#define TILE_I 16
#define TILE_J 8
#define I2D(ni,i,j) (((ni)*(j))+i)

///

// OpenGL pixel buffer object and texture //
GLuint gl_PBO,gl_Tex;
// arrays on host //
float *f0,*f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8,*plot;
int *solid;
unsigned int *cmap_rgba,*plot_rgba; // rgba arrays for plotting

// arrays on device //
float *f0_data,*f1_data,*f2_data,*f3_data,*f4_data;
float *f5_data,*f6_data,*f7_data,*f8_data,*plot_data;
int *solid_data;
unsigned int *cmap_rgba_data,*plot_rgba_data;
// textures on device //
texture<float,2> f1_tex,f2_tex,f3_tex,f4_tex,
                  f5_tex,f6_tex,f7_tex,f8_tex;

// CUDA special format arrays on device //
```

```
cudaArray *f1_array,*f2_array,*f3_array,*f4_array;
cudaArray *f5_array,*f6_array,*f7_array,*f8_array;

// scalars //
float tau,faceq1,faceq2,faceq3;
float vxin,roout;
float width,height;
float minvar,maxvar;

int ni,nj,i,j,i0;
int nsolid,nstep,nsteps,ncol;
int ipos_old,jpos_old,draw_solid_flag,i_steps;;

size_t pitch;

int iterasi = 4000;
unsigned int Timer = 0;
int iter1 = 1000;
int iter2 = 2000;
///////////////////////////////
// OpenGL function prototypes
//
void display(void);
void resize(int w,int h);
// CUDA kernel prototypes
//
__global__ void stream_kernel (int pitch,float *f1_data,float
             *f2_data,float *f3_data,float *f4_data,float *f5_data,float
             *f6_data, float *f7_data,float *f8_data);

__global__ void collide_kernel (int pitch,float tau,float faceq1,float
             faceq2,float faceq3, float *f0_data,float *f1_data,float
             *f2_data, float *f3_data,float *f4_data,float *f5_data,float
             *f6_data, float *f7_data,float *f8_data,float *plot_data);

__global__ void apply_Periodic_BC_kernel (int ni,int nj,int pitch, float
             *f2_data,float *f4_data,float *f5_data, float *f6_data,float
             *f7_data,float *f8_data);

__global__ void apply_BCs_kernel (int ni,int nj,int pitch,float
             vxin,float roout, float faceq2,float faceq3, float
             *f0_data,float *f1_data,float *f2_data, float *f3_data,float
             *f4_data,float *f5_data, float *f6_data,float *f7_data,float
             *f8_data,int* solid_data);

__global__ void get_rgba_kernel (int pitch,int ncol,float minvar,float
             maxvar,float *plot_data, unsigned int *plot_rgba_data,
             unsigned int *cmap_rgba_data, int *solid_data);
// CUDA kernel C wrappers
//
void stream(void);
void collide(void);
```

```
void apply_Periodic_BC(void);
void apply_BCs(void);
void get_rgba(void);

///////////////////
int main(int argc,char **argv)
{
    int totpoints,i;
    float rcol,gcol,bcol,R;

    FILE *fp_col;
    cudaChannelFormatDesc desc;

    // The following parameters are usually read from a file,but
    // hard code them for the demo:
    ni=320*3;
    nj=112*3;
    vxin=0.05;
    roout=1.0;
    tau=0.51;
    // End of parameter list
    // Write parameters to screen
    printf ("ni = %d\n",ni);
    printf ("nj = %d\n",nj);
    printf ("vxin = %f\n",vxin);
    printf ("roout = %f\n",roout);
    printf ("tau = %f\n",tau);

    totpoints=ni*nj;
    //
    // allocate memory on host
    //
    f0 = (float *)malloc(ni*nj*sizeof(float));
    f1 = (float *)malloc(ni*nj*sizeof(float));
    f2 = (float *)malloc(ni*nj*sizeof(float));
    f3 = (float *)malloc(ni*nj*sizeof(float));
    f4 = (float *)malloc(ni*nj*sizeof(float));
    f5 = (float *)malloc(ni*nj*sizeof(float));
    f6 = (float *)malloc(ni*nj*sizeof(float));
    f7 = (float *)malloc(ni*nj*sizeof(float));
    f8 = (float *)malloc(ni*nj*sizeof(float));
    plot = (float *)malloc(ni*nj*sizeof(float));

    solid = (int *)malloc(ni*nj*sizeof(int));

    plot_rgba = (unsigned int*)malloc(ni*nj*sizeof(unsigned int));

    //
    // allocate memory on device
    //
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f0_data,&pitch,
                                   sizeof(float)*ni,nj));
```

```
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f1_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f2_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f3_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f4_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f5_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f6_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f7_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&f8_data,&pitch,
                                   sizeof(float)*ni,nj));
    CUDA_SAFE_CALL(cudaMallocPitch((void **)&plot_data,&pitch,
                                   sizeof(float)*ni,nj));


    CUDA_SAFE_CALL(cudaMallocPitch((void **)&solid_data,&pitch,
                                   sizeof(int)*ni,nj));

    desc = cudaCreateChannelDesc<float>();
    CUDA_SAFE_CALL(cudaMallocArray(&f1_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f2_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f3_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f4_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f5_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f6_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f7_array,&desc,ni,nj));
    CUDA_SAFE_CALL(cudaMallocArray(&f8_array,&desc,ni,nj));

    //
    // Some factors used in equilibrium f's
    //
    faceq1 = 4.f/9.f;
    faceq2 = 1.f/9.f;
    faceq3 = 1.f/36.f;

    //
    // Initialise f's
    //
    for (i=0; i<totpoints; i++) {
f0[i] = faceq1*roout*(1.f         -1.5f*vxin*vxin);
f1[i] = faceq2*roout*(1.f+3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f2[i] = faceq2*roout*(1.f                             -1.5f*vxin*vxin);
f3[i] = faceq2*roout*(1.f-3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f4[i] = faceq2*roout*(1.f                             -1.5f*vxin*vxin);
f5[i] = faceq3*roout*(1.f+3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f6[i] = faceq3*roout*(1.f-3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f7[i] = faceq3*roout*(1.f-3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
```

```
f8[i] = faceq3*roout*(1.f+3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
plot[i] = vxin;
solid[i] = 1;
    }
    //
    // Read in colourmap data for OpenGL display
    //
    fp_col = fopen("cmap.dat","r");
    if (fp_col==NULL) {
            printf("Error: can't open cmap.dat \n");
            return 1;
    }

    fscanf (fp_col,"%d",&ncol);
    cmap_rgba = (unsigned int *)malloc(ncol*sizeof(unsigned int));
    CUDA_SAFE_CALL(cudaMalloc((void **)&cmap_rgba_data,
                                    sizeof(unsigned int)*ncol));

    for (i=0;i<ncol;i++){
            fscanf(fp_col,"%f%f%f",&rcol,&gcol,&bcol);
            cmap_rgba[i]=((int)(255.0f) << 24) | // convert colourmap to
            int
                ((int)(bcol*255.0f) << 16) |
                ((int)(gcol*255.0f) <<  8) |
                ((int)(rcol*255.0f) <<  0);
    }
    fclose(fp_col);

            /////////////////////////

            ///bola
            for (i=0; i<ni; i++){
            for(j=0; j<nj; j++){

                        R=sqrtf(((float)i-(ni/2))*((float)i-
            (ni/2))+((float)j-50.f)*((float)j-50.f));

                        if(R<=20.f){
                        i0=I2D(ni,i,j);
                        solid[i0]=0;

                        }
            }}

//garis bawah
            for (i=0; i<=ni; i++){
                 for(j=0; j<=10; j++){
                        i0=I2D(ni,i,j);
                        solid[i0]=0;
                 }
            }
// garis atas
```

```
            for (i=0; i<=ni; i++){
                for(j=300; j<=nj-1; j++){
                    i0=I2D(ni,i,j);
                    solid[i0]=0;
                }
            }
//kotak awal
            for (i=0; i<=50; i++){
                for(j=10; j<=50; j++){
                    i0=I2D(ni,i,j);
                    solid[i0]=0;
                }
            }
/////////////////////
   //
   // Transfer initial data to device
   //
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f0_data,pitch,(void *)f0,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f1_data,pitch,(void *)f1,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f2_data,pitch,(void *)f2,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f3_data,pitch,(void *)f3,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f4_data,pitch,(void *)f4,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f5_data,pitch,(void *)f5,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f6_data,pitch,(void *)f6,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f7_data,pitch,(void *)f7,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)f8_data,pitch,(void *)f8,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)plot_data,pitch,(void *)plot,
                               sizeof(float)*ni,sizeof(float)*ni,nj,
                               cudaMemcpyHostToDevice));
   CUDA_SAFE_CALL(cudaMemcpy2D((void *)solid_data,pitch,(void *)solid,
                               sizeof(int)*ni,sizeof(int)*ni,nj,
                               cudaMemcpyHostToDevice));
```

```
CUDA_SAFE_CALL(cudaMemcpy((void *)cmap_rgba_data, (void
         *)cmap_rgba,sizeof(unsigned int)*ncol,
         cudaMemcpyHostToDevice));

         // Set up CUDA Timer
         cutCreateTimer(&Timer);
         cutResetTimer(Timer);
         cutStartTimer(Timer);
         //
// Iinitialise OpenGL display-use glut
//
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowSize(ni,nj);                   // Window of ni x nj
         pixels
glutInitWindowPosition(50,50);               // Window position
glutCreateWindow("CUDA 2D LB");               // Window title

printf("Loading extensions: %s\n",glewGetErrorString(glewInit()));
if(!glewIsSupported(
                    "GL_VERSION_2_0 "
                    "GL_ARB_pixel_buffer_object "
                    "GL_EXT_framebuffer_object "
                    )){
    fprintf(stderr,"ERROR: Support for necessary OpenGL extensions
         missing.");
    fflush(stderr);
    return 1;
}

// Set up view
glClearColor(0.0,0.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0,ni,0.,nj,-200.0,200.0);

// Create texture and bind to gl_Tex
glEnable(GL_TEXTURE_2D);
glGenTextures(1,&gl_Tex);                     // Generate 2D texture
glBindTexture(GL_TEXTURE_2D,gl_Tex);          // bind to gl_Tex
// texture properties:
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA8,ni,nj,0,
             GL_RGBA,GL_UNSIGNED_BYTE,NULL);
printf("Texture created.\n");

// Create pixel buffer object and bind to gl_PBO
glGenBuffers(1,&gl_PBO);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB,gl_PBO);
```

```
            glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB,pitch*nj,NULL,GL_STR
            EAM_COPY);
    CUDA_SAFE_CALL( cudaGLRegisterBufferObject(gl_PBO) );
    printf("Buffer created.\n");
    printf("Starting GLUT main loop...\n");
    glutDisplayFunc(display);
    glutReshapeFunc(resize);
    glutIdleFunc(display);
    glutMainLoop();
    return 0;
}

///////////////////
__global__ void stream_kernel (int pitch,float *f1_data,float *f2_data,
                              float *f3_data,float *f4_data,float
            *f5_data,float *f6_data,float *f7_data,float *f8_data)
// CUDA kernel

{
    int i,j,i2d;

    i = blockIdx.x*TILE_I+threadIdx.x;
    j = blockIdx.y*TILE_J+threadIdx.y;

    i2d = i+j*pitch/sizeof(float);

    // look up the adjacent f's needed for streaming using textures
    // i.e. gather from textures,write to device memory: f1_data,etc
    f1_data[i2d] = tex2D(f1_tex,(float) (i-1)  ,(float) j);
    f2_data[i2d] = tex2D(f2_tex,(float) i      ,(float) (j-1));
    f3_data[i2d] = tex2D(f3_tex,(float) (i+1)  ,(float) j);
    f4_data[i2d] = tex2D(f4_tex,(float) i      ,(float) (j+1));
    f5_data[i2d] = tex2D(f5_tex,(float) (i-1)  ,(float) (j-1));
    f6_data[i2d] = tex2D(f6_tex,(float) (i+1)  ,(float) (j-1));
    f7_data[i2d] = tex2D(f7_tex,(float) (i+1)  ,(float) (j+1));
    f8_data[i2d] = tex2D(f8_tex,(float) (i-1)  ,(float) (j+1));
}

void stream(void)
// C wrapper
{
// Device-to-device mem-copies to transfer data from linear memory
            (f1_data)
// to CUDA format memory (f1_array) so we can use these in textures
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f1_array,0,0,(void *)f1_data,pitch,
            sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f2_array,0,0,(void*)f2_data,pitch,size
            of(float)*ni,nj,cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f3_array,0,0,(void*)f3_data,pitch,size
            of(float)*ni,nj,cudaMemcpyDeviceToDevice));
```

```
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f4_array,0,0,(void *)f4_data,pitch,
                sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f5_array,0,0,(void *)f5_data,pitch,
                sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f6_array,0,0,(void *)f6_data,pitch,
                sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f7_array,0,0,(void *)f7_data,pitch,
                sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));
CUDA_SAFE_CALL(cudaMemcpy2DToArray(f8_array,0,0,(void *)f8_data,pitch,
                sizeof(float)*ni,nj, cudaMemcpyDeviceToDevice));

    // Tell CUDA that we want to use f1_array etc as textures. Also
    // define what type of interpolation we want (nearest point)
    f1_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f1_tex,f1_array));

    f2_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f2_tex,f2_array));

    f3_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f3_tex,f3_array));

    f4_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f4_tex,f4_array));

    f5_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f5_tex,f5_array));

    f6_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f6_tex,f6_array));

    f7_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f7_tex,f7_array));

    f8_tex.filterMode = cudaFilterModePoint;
    CUDA_SAFE_CALL(cudaBindTextureToArray(f8_tex,f8_array));

    dim3 grid = dim3(ni/TILE_I,nj/TILE_J);
    dim3 block = dim3(TILE_I,TILE_J);

    stream_kernel<<<grid,block>>>(pitch,f1_data,f2_data,f3_data,f4_data,
                                  f5_data,f6_data,f7_data,f8_data);

    CUT_CHECK_ERROR("stream failed.");

    CUDA_SAFE_CALL(cudaUnbindTexture(f1_tex));
    CUDA_SAFE_CALL(cudaUnbindTexture(f2_tex));
    CUDA_SAFE_CALL(cudaUnbindTexture(f3_tex));
    CUDA_SAFE_CALL(cudaUnbindTexture(f4_tex));
    CUDA_SAFE_CALL(cudaUnbindTexture(f5_tex));
    CUDA_SAFE_CALL(cudaUnbindTexture(f6_tex));
    CUDA_SAFE_CALL(cudaUnbindTexture(f7_tex));
```

```
    CUDA_SAFE_CALL(cudaUnbindTexture(f8_tex));
}

///
__global__ void collide_kernel (int pitch,float tau,float faceq1,float
             faceq2,float faceq3, float *f0_data,float *f1_data,float
             *f2_data, float *f3_data,float *f4_data,float *f5_data,float
             *f6_data, float *f7_data,float *f8_data,float *plot_data)
// CUDA kernel
{
    int i,j,i2d;
    float ro,vx,vy,v_sq_term,rtau,rtau1;
    float f0now,f1now,f2now,f3now,f4now,f5now,f6now,f7now,f8now;
    float f0eq,f1eq,f2eq,f3eq,f4eq,f5eq,f6eq,f7eq,f8eq;

    i = blockIdx.x*TILE_I+threadIdx.x;
    j = blockIdx.y*TILE_J+threadIdx.y;

    i2d = i+j*pitch/sizeof(float);

    rtau = 1.f/tau;
    rtau1 = 1.f-rtau;

    // Read all f's and store in registers
    f0now = f0_data[i2d];
    f1now = f1_data[i2d];
    f2now = f2_data[i2d];
    f3now = f3_data[i2d];
    f4now = f4_data[i2d];
    f5now = f5_data[i2d];
    f6now = f6_data[i2d];
    f7now = f7_data[i2d];
    f8now = f8_data[i2d];

    // Macroscopic flow props:
    ro =  f0now+f1now+f2now+f3now+f4now+f5now+f6now+f7now+f8now;
    vx = (f1now-f3now+f5now-f6now-f7now+f8now)/ro;
    vy = (f2now-f4now+f5now+f6now-f7now-f8now)/ro;

    // Set plotting variable to velocity magnitude
    plot_data[i2d] = sqrtf(vx*vx+vy*vy);

    // Calculate equilibrium f's
    v_sq_term = 1.5f*(vx*vx+vy*vy);
    f0eq = ro*faceq1*(1.f-v_sq_term);
    f1eq = ro*faceq2*(1.f+3.f*vx+4.5f*vx*vx-v_sq_term);
    f2eq = ro*faceq2*(1.f+3.f*vy+4.5f*vy*vy-v_sq_term);
    f3eq = ro*faceq2*(1.f-3.f*vx+4.5f*vx*vx-v_sq_term);
    f4eq = ro*faceq2*(1.f-3.f*vy+4.5f*vy*vy-v_sq_term);
    f5eq = ro*faceq3*(1.f+3.f*(vx+vy)+4.5f*(vx+vy)*(vx+vy)-v_sq_term);
    f6eq = ro*faceq3*(1.f+3.f*(-vx+vy)+4.5f*(-vx+vy)*(-vx+vy)-v_sq_term);
    f7eq = ro*faceq3*(1.f+3.f*(-vx-vy)+4.5f*(-vx-vy)*(-vx-vy)-v_sq_term);
```

```
    f8eq = ro*faceq3*(1.f+3.f*(vx-vy)+4.5f*(vx-vy)*(vx-vy)-v_sq_term);

    // Do collisions
    f0_data[i2d] = rtau1*f0now+rtau*f0eq;
    f1_data[i2d] = rtau1*f1now+rtau*f1eq;
    f2_data[i2d] = rtau1*f2now+rtau*f2eq;
    f3_data[i2d] = rtau1*f3now+rtau*f3eq;
    f4_data[i2d] = rtau1*f4now+rtau*f4eq;
    f5_data[i2d] = rtau1*f5now+rtau*f5eq;
    f6_data[i2d] = rtau1*f6now+rtau*f6eq;
    f7_data[i2d] = rtau1*f7now+rtau*f7eq;
    f8_data[i2d] = rtau1*f8now+rtau*f8eq;
}

void collide(void)
// C wrapper
{
    dim3 grid = dim3(ni/TILE_I,nj/TILE_J);
    dim3 block = dim3(TILE_I,TILE_J);

    collide_kernel<<<grid,block>>>(pitch,tau,faceq1,faceq2,faceq3,
             f0_data,f1_data,f2_data,f3_data,f4_data,
             f5_data,f6_data,f7_data,f8_data,plot_data);

    CUT_CHECK_ERROR("collide failed.");
}

///////////////////
__global__ void apply_BCs_kernel (int ni,int nj,int pitch,float
             vxin,float roout, float faceq2,float faceq3, float
             *f0_data,float *f1_data,float *f2_data, float *f3_data,float
             *f4_data,float *f5_data, float *f6_data,float *f7_data,float
             *f8_data, int* solid_data)

// CUDA kernel all BC's apart from periodic boundaries:

{
    int i,j,i2d;
    float f1old,f2old,f3old,f4old,f5old,f6old,f7old,f8old;

    i = blockIdx.x*TILE_I+threadIdx.x;
    j = blockIdx.y*TILE_J+threadIdx.y;

    i2d = i+j*pitch/sizeof(float);

    // Solid BC: "bounce-back"
    if (solid_data[i2d] == 0) {
      f1old = f1_data[i2d];
      f2old = f2_data[i2d];
      f3old = f3_data[i2d];
      f4old = f4_data[i2d];
      f5old = f5_data[i2d];
```

```
        f6old = f6_data[i2d];
        f7old = f7_data[i2d];
        f8old = f8_data[i2d];

        f1_data[i2d] = f3old;
        f2_data[i2d] = f4old;
        f3_data[i2d] = f1old;
        f4_data[i2d] = f2old;
        f5_data[i2d] = f7old;
        f6_data[i2d] = f8old;
        f7_data[i2d] = f5old;
        f8_data[i2d] = f6old;
    }
}

void apply_BCs(void)
// C wrapper
{
    dim3 grid = dim3(ni/TILE_I,nj/TILE_J);
    dim3 block = dim3(TILE_I,TILE_J);

   apply_BCs_kernel<<<grid,block>>>(ni,nj,pitch,vxin,roout,faceq2,faceq3,
             f0_data,f1_data,f2_data, f3_data,f4_data,f5_data,
             f6_data,f7_data,f8_data,solid_data);
        CUT_CHECK_ERROR("apply_BCs failed.");
}

///////////////////

__global__ void apply_Periodic_BC_kernel (int ni,int nj,int pitch, float
             *f2_data,float *f4_data,float *f5_data, float *f6_data,float
             *f7_data,float *f8_data)
// CUDA kernel
{
    int i,j,i2d,i2d2;

    i = blockIdx.x*TILE_I+threadIdx.x;
    j = blockIdx.y*TILE_J+threadIdx.y;

    i2d = i+j*pitch/sizeof(float);

    if (j == 0 ) {
        i2d2 = i+(nj-1)*pitch/sizeof(float);
        f2_data[i2d] = f2_data[i2d2];
        f5_data[i2d] = f5_data[i2d2];
        f6_data[i2d] = f6_data[i2d2];
    }
    if (j == (nj-1)) {
        i2d2 = i;
        f4_data[i2d] = f4_data[i2d2];
        f7_data[i2d] = f7_data[i2d2];
        f8_data[i2d] = f8_data[i2d2];
```

```
    }
}

// C wrapper

void apply_Periodic_BC(void)
{
    dim3 grid = dim3(ni/TILE_I,nj/TILE_J);
    dim3 block = dim3(TILE_I,TILE_J);

    apply_Periodic_BC_kernel<<<grid,block>>>(ni,nj,pitch,
                                             f2_data,f4_data,f5_data,
                                             f6_data,f7_data,f8_data);

    CUT_CHECK_ERROR("apply_Periodic_BC failed.");
}

///////////////////////

__global__ void get_rgba_kernel (int pitch,int ncol,float minvar,float
             maxvar, float *plot_data, unsigned int *plot_rgba_data,
             unsigned int *cmap_rgba_data, int *solid_data)

// CUDA kernel to fill plot_rgba_data array for plotting

{
    int i,j,i2d,icol;
    float frac;

    i = blockIdx.x*TILE_I+threadIdx.x;
    j = blockIdx.y*TILE_J+threadIdx.y;

    i2d = i+j*pitch/sizeof(float);

    frac = (plot_data[i2d]-minvar)/(maxvar-minvar);
    icol = (int)(frac*(float)ncol);
    plot_rgba_data[i2d] = solid_data[i2d]*cmap_rgba_data[icol];
}

void get_rgba(void)
// C wrapper
{
    dim3 grid = dim3(ni/TILE_I,nj/TILE_J);
    dim3 block = dim3(TILE_I,TILE_J);

    get_rgba_kernel<<<grid,block>>>(pitch,ncol,minvar,maxvar,
             plot_data,plot_rgba_data,cmap_rgba_data, solid_data);

    CUT_CHECK_ERROR("get_rgba failed.");
}
///////////////////
void display(void)
```

```
// This function is called automatically,over and over again, by GLUT
{
    // Set upper and lower limits for plotting
    minvar=0.;
    maxvar=0.2;

    // Do one Lattice Boltzmann step: stream,BC,collide:
             i_steps++;
    stream();
    apply_Periodic_BC();
    apply_BCs();
    collide();

    // For plotting,map the plot_rgba_data array to the
    // gl_PBO pixel buffer

             CUDA_SAFE_CALL(cudaGLMapBufferObject((void**)&plot_rgba_data
             ,gl_PBO));

    // Fill the plot_rgba_data array (and the pixel buffer)
    get_rgba();
    CUDA_SAFE_CALL(cudaGLUnmapBufferObject(gl_PBO));

    // Copy the pixel buffer to the texture,ready to display

             glTexSubImage2D(GL_TEXTURE_2D,0,0,0,ni,nj,GL_RGBA,GL_UNSIGNE
             D_BYTE,0);

    // Render one quad to the screen and colour it using our texture
    // i.e. plot our plotvar data to the screen
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
    glTexCoord2f (0.0,0.0);
    glVertex3f (0.0,0.0,0.0);
    glTexCoord2f (1.0,0.0);
    glVertex3f (ni,0.0,0.0);
    glTexCoord2f (1.0,1.0);
    glVertex3f (ni,nj,0.0);
    glTexCoord2f (0.0,1.0);
    glVertex3f (0.0,nj,0.0);
    glEnd();
    glutSwapBuffers();

             if (i_steps%iterasi == 1) {

             printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f
             perSecond \n",i_steps,0.001*cutGetTimerValue(Timer));
             }
             if (i_steps%iterasi == iter1) {
             printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f
             perSecond \n",i_steps,0.001*cutGetTimerValue(Timer));
```

```
		}
		if (i_steps%iterasi == iter2) {
		printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f
		perSecond \n",i_steps,0.001*cutGetTimerValue(Timer));

		}
		if (i_steps%iterasi == 0) {
		printf("\n Iteration: %6d ; Elapsed Time (s): %4.2f
		perSecond \n",i_steps,0.001*cutGetTimerValue(Timer));
		system("PAUSE");
		//exit(0);}
}
/////////////////////
void resize(int w,int h)
// GLUT resize callback to allow us to change the window size
{
   width = w;
   height = h;
   glViewport (0,0,w,h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho (0.,ni,0.,nj,-200. ,200.);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
}
```

Kode program untuk CPU

Kondisi dua fase pada kasus satu dan dua

```
#include<stdio.h>
#include<stdarg.h>
#include<stdlib.h>
#include<math.h>
#include<time.h>
#include<GL/glew.h>
#include<GL/glut.h>

#defineWINDOW_TITTLE_PREFIX"LB2Dmultiphase"
#defineI2D(ni,i,j)(((ni)*(i))+j)

///

//OpenGLpixelbufferobjectandtexture//
GLuintgl_PBO,gl_Tex;


//arrays//
float*f0,*f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8;
float*tmpf0,*tmpf1,*tmpf2,*tmpf3,*tmpf4,*tmpf5,*tmpf6,*tmpf7,*tmpf8;
float*cmap,*plotvar;
float*ux,*uy,*rho,*phi;
int*solid;
unsignedint*cmap_rgba,*plot_rgba;//rgbaarraysforplotting

//scalars//
floattau,faceq1,faceq2,faceq3;
floatvxin,roout;
floatwidth,height;
intni,nj;
intncol;
intipos_old,jpos_old,draw_solid_flag,i_steps;

floatM_PI=3.14159265358979;
doublerhoh=2.6429;//highdensityfluid(rhoh=1forsinglephase)
doublerhol=0.0734;//lowdensityfluid(rhoh=1forsinglephase)
doubleifaceW=3.0;//interfacewidth
doubleG=-6.0;//interparticularinteractionpotential(G=0forsinglephase)
doublebody_force=1.0e-4;//gravityforce
doublebody_force_dir=270;//gravitydirection(0=down90=right180=top270=left)
floatbody_force_x,body_force_y;
intdr=25;    //dropletradius
intdx=150;//dropletpositionx(formultiphasemodel)
intdy=80;//dropletpositiony(formultiphasemodel)
intbot,top,lef,rig,fro,bac;
floatux2,uy2,uz2,uxy,uxz,uyz,uxy2,uxz2,uxyz2;
clock_tmulai,berhenti;
intiterasi=3000;
intiter1=10;
intiter2=20;
```

```
////
//OpenGLfunctionprototypes
//
voiddisplay(void);
voidresize(intw,inth);
voidmouse(intbutton,intstate,intx,inty);
voidmouse_motion(intx,inty);
voidCalculateFrameRate();

//
//LatticeBoltzmannfunctionprototypes
//
voidstream(void);
voidcollide(void);
voidsolid_BC(void);
voidper_BC(void);
voidin_BC(void);
voidex_BC_crude(void);
voidapply_BCs(void);

unsignedintget_col(floatmin,floatmax,floatval);

//
intmain(intargc,char**argv)
{
intarray_size_2d,totpoints,i,j,i0,i1;
floatrcol,gcol,bcol,tmp,tmp2,tmp1;
 floatdx2,dy2,dx3,dy3,radius,radius2,vx,vy;

FILE*fp_col;
 //start=clock();
//Thefollowingparametersareusuallyreadfromafile,but
//hardcodethemforthedemo:
ni=200;//320;
nj=200;//112;
vxin=0.0;
roout=1.0;
tau=1.0;
 vx=0.0;
 vy=0.0;
  //Endofparameterlist
//Writeparameterstoscreen
 mulai=clock();
 printf("ni=%d\n",ni);
printf("nj=%d\n",nj);
printf("vxin=%f\n",vxin);
printf("roout=%f\n",roout);
printf("tau=%f\n",tau);


totpoints=ni*nj;
array_size_2d=ni*nj*sizeof(float);

//Allocatememoryforarrays

f0=malloc(array_size_2d);
```

```
f1=malloc(array_size_2d);
f2=malloc(array_size_2d);
f3=malloc(array_size_2d);
f4=malloc(array_size_2d);
f5=malloc(array_size_2d);
f6=malloc(array_size_2d);
f7=malloc(array_size_2d);
f8=malloc(array_size_2d);

tmpf0=malloc(array_size_2d);
tmpf1=malloc(array_size_2d);
tmpf2=malloc(array_size_2d);
tmpf3=malloc(array_size_2d);
tmpf4=malloc(array_size_2d);
tmpf5=malloc(array_size_2d);
tmpf6=malloc(array_size_2d);
tmpf7=malloc(array_size_2d);
tmpf8=malloc(array_size_2d);

rho=malloc(array_size_2d);
 phi=malloc(array_size_2d);
 ux=malloc(array_size_2d);
 uy=malloc(array_size_2d);
plotvar=malloc(array_size_2d);

plot_rgba=malloc(ni*nj*sizeof(unsignedint));

solid=malloc(ni*nj*sizeof(int));

//
//Somefactorsusedtocalculatethef_equilibriumvalues
//
faceq1=4.f/9.f;
faceq2=1.f/9.f;
faceq3=1.f/36.f;

body_force_x=body_force*sin(body_force_dir/(180.0/M_PI));
body_force_y=-body_force*cos(body_force_dir/(180.0/M_PI));

  for(i=0;i<totpoints;i++){

  if(solid[i]!=1)
  {rho[i]=0.2*(rhoh-rhol)+rhol;}
  if(solid[i]=1)
  {
  rho[i]=rhol;
  }

  }

 for(j=0;j<nj;j++){
 for(i=0;i<ni;i++){
 i0=I2D(ni,i,j);
 solid[i0]=1;
  //rho[i0]=rhol;
//bola
     dx2=((float)(i-dx))*((float)(i-dx));
```

```
dy2=((float)(j-dy))*((float)(j-dy));
    //dataran
    dx3=ni;
    dy3=((float)(j-30))*((float)(j-30));
    radius2=sqrtf(dx3+dy3);
    radius=sqrtf(dx2+dy2);
tmp1=0.5*((rhoh+rhol)-(rhoh-rhol)*tanh((radius-dr)/ifaceW*2.0));
     tmp2=0
tmp=tmp1+tmp2;
if(tmp>rhol)rho[i0]=tmp;

 }
}

//Initialisef'sbysettingthemtothef_equilibirumvaluesassuming
//thatthewholedomainisatvelocityvx=vxinvy=0anddensityro=roout
for(i=0;i<totpoints;i++){
 solid[i]=1;
 ux[i]=vx;
 uy[i]=vy;
f0[i]=faceq1*rho[i]*(1.f+1.5f*(vx*vx+vy*vy));
f1[i]=faceq2*rho[i]*(1.f+3.f*vx+4.5f*vx*vx-1.5f*(vx*vx+vy*vy));
f2[i]=faceq2*rho[i]*(1.f+3.f*vy+4.5f*vy*vy-1.5f*(vx*vx+vy*vy));
f3[i]=faceq2*rho[i]*(1.f+3.f*vx+4.5f*vx*vx-1.5f*(vx*vx+vy*vy));
f4[i]=faceq2*rho[i]*(1.f+3.f*vy+4.5f*vy*vy-1.5f*(vx*vx+vy*vy));
f5[i]=faceq3*rho[i]*(1.f+3.f*(vx+vy)+4.5f*(vx+vy)*(vx+vy)-.5f*(vx*vx+vy*vy));
f6[i]=faceq3*rho[i]*(1.f+3.f*(-vx+vy)+4.5f*(-vx+vy)*(-vx+vy)-
1.5f*(vx*vx+vy*vy));
f7[i]=faceq3*rho[i]*(1.f+3.f*(-vx-vy)+4.5f*(-vx-vy)*(-vx-vy)-
1.5f*(vx*vx+vy*vy));
f8[i]=faceq3*rho[i]*(1.f+3.f*(vx-vy)+4.5f*(vx-vy)*(vx-vy)-
1.5f*(vx*vx+vy*vy));
 plotvar[i]=rho[i];

}

 for(i=0;i<ni;i++){
  i0=I2D(ni,i,0);
  i1=I2D(ni,i,nj-1);
}
 //garisbawah
//ReadincolourmapdataforOpenGLdisplay
//
 fp_col=fopen("cmap.dat","r");
if(fp_col==NULL){
 printf("Error:can'topencmap.dat\n");
 return1;
}
//allocatememoryforcolourmap(storedasalineararrayofint's)
fscanf(fp_col,"%d",&ncol);
cmap_rgba=(unsignedint*)malloc(ncol*sizeof(unsignedint));
//readcolourmapandstoreasint's
for(i=0;i<ncol;i++){
 fscanf(fp_col,"%f%f%f",&rcol,&gcol,&bcol);
 cmap_rgba[i]=((int)(255.0f)<<24)|//convertcolourmaptoint
 ((int)(bcol*255.0f)<<16)|
 ((int)(gcol*255.0f)<<8)|
```

```
 ((int)(rcol*255.0f)<<0);
}
fclose(fp_col);
//IinitialiseOpenGLdisplay-useglut
//
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB);
glutInitWindowSize(ni,nj);//Windowofnixnjpixels
glutInitWindowPosition(50,450);//position
glutCreateWindow("2DLBMpsc");//title

//CheckforOpenGLextensionsupport
printf("Loadingextensions:%s\n",glewGetErrorString(glewInit()));
if(!glewIsSupported(
"GL_VERSION_2_0"
"GL_ARB_pixel_buffer_object"
"GL_EXT_framebuffer_object"
)){
fprintf(stderr,"ERROR:SupportfornecessaryOpenGLextensionsmissing.");
fflush(stderr);
return;
}

//Setupview
glClearColor(0.0,0.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0,ni,0.,nj,-200.0,200.0);

//Createtexturewhichweusetodisplaytheresultandbindtogl_Tex
glEnable(GL_TEXTURE_2D);
glGenTextures(1,&gl_Tex);//Generate2Dtexture
glBindTexture(GL_TEXTURE_2D,gl_Tex);//bindtogl_Tex
//textureproperties:
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D,GL_TEXTURE_MIN_FILTER,GL_LINEAR);
glTexImage2D(GL_TEXTURE_2D,0,GL_RGBA8,ni,nj,0,
GL_RGBA,GL_UNSIGNED_BYTE,NULL);

//Createpixelbufferobjectandbindtogl_PBO.Westorethedatawewantto
//plotinmemoryonthegraphicscard-ina"pixelbuffer".Wecanthen
//copythistothetexturedefinedaboveandsendittothescreen
glGenBuffers(1,&gl_PBO);
glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB,gl_PBO);
printf("Buffercreated.\n");

//Setthecall-backfunctionsandstarttheglutloop
printf("StartingGLUTmainloop...\n");
 glutDisplayFunc(display);
glutReshapeFunc(resize);
 glutIdleFunc(display);
 glutMainLoop();

return0;
```

```
}

//////////////

voidstream(void)

//Movethefvaluesonegridspacinginthedirectionsthattheyarepointing
//i.e.f1iscopiedonelocationtotheright,etc.

{
inti,j,im1,ip1,jm1,jp1,i0;

//Initiallythef'saremovedtotemporaryarrays
for(j=0;j<nj;j++){
 jm1=j-1;
 jp1=j+1;
 if(j==0)jm1=0;
 if(j==(nj-1))jp1=nj-1;
 for(i=1;i<ni;i++){
 i0=I2D(ni,i,j);
 im1=i-1;
 ip1=i+1;
 if(i==0)im1=0;
 if(i==(ni-1))ip1=ni-1;
 tmpf1[i0]=f1[I2D(ni,im1,j)];
 tmpf2[i0]=f2[I2D(ni,i,jm1)];
 tmpf3[i0]=f3[I2D(ni,ip1,j)];
 tmpf4[i0]=f4[I2D(ni,i,jp1)];
 tmpf5[i0]=f5[I2D(ni,im1,jm1)];
 tmpf6[i0]=f6[I2D(ni,ip1,jm1)];
 tmpf7[i0]=f7[I2D(ni,ip1,jp1)];
 tmpf8[i0]=f8[I2D(ni,im1,jp1)];
 }
}

//Nowthetemporaryarraysarecopiedtothemainfarrays
for(j=0;j<nj;j++){
 for(i=1;i<ni;i++){
 i0=I2D(ni,i,j);
 f1[i0]=tmpf1[i0];
 f2[i0]=tmpf2[i0];
 f3[i0]=tmpf3[i0];
 f4[i0]=tmpf4[i0];
 f5[i0]=tmpf5[i0];
 f6[i0]=tmpf6[i0];
 f7[i0]=tmpf7[i0];
 f8[i0]=tmpf8[i0];
 }
}
 solid[i0]=1;
}


///

voidcollide(void)
```

```
//Collisionsbetweentheparticlesaremodeledhere.Weusetheverysimplest
//modelwhichassumesthef'schangetowardthelocalequlibriumvalue(based
//ondensityandvelocityatthatpoint)overafixedtimescale,tau

{
inti,j,i0,im1,ip1,jm1,jp1;
floatro,rovx,rovy,vx,vy,v_sq_term,grad_phi_x,grad_phi_y;
floatf0eq,f1eq,f2eq,f3eq,f4eq,f5eq,f6eq,f7eq,f8eq;
floatrtau,rtau1;


//Someusefulconstants
rtau=1.f/tau;
rtau1=1.f-rtau;

for(j=0;j<nj;j++){
 for(i=0;i<ni;i++){

 i0=I2D(ni,i,j);

 //Dothesummationsneededtoevaluatethedensityandcomponentsofvelocity
 ro=f0[i0]+f1[i0]+f2[i0]+f3[i0]+f4[i0]+f5[i0]+f6[i0]+f7[i0]+f8[i0];
 rovx=f1[i0]-f3[i0]+f5[i0]-f6[i0]-f7[i0]+f8[i0];
 rovy=f2[i0]-f4[i0]+f5[i0]+f6[i0]-f7[i0]-f8[i0];
 ux[i0]=rovx/ro;
 uy[i0]=rovy/ro;

  ux[i0]+=tau*body_force_x;
uy[i0]+=tau*body_force_y;

rho[i0]=ro;
 //Alsoloadthevelocitymagnitudeintoplotvar-thisiswhatwewill
 //displayusingOpenGLlater

phi[i0]=1.0-exp(-rho[i0]);

plotvar[i0]=sqrt(ux[i0]*ux[i0]+uy[i0]*uy[i0]);//(sqrt(ro+G*phi[i0]*phi[i0]))*
rho1;//sqrt(vx*vx+vy*vy);
  }
}

for(j=0;j<nj;j++){
 jm1=j-1;
 jp1=j+1;
 if(j==0)jm1=0;
 if(j==(nj-1))jp1=nj-1;
 for(i=1;i<ni;i++){
 //i0=I2D(ni,i,j);
 im1=i-1;
 ip1=i+1;
 if(i==0)im1=0;
 if(i==(ni-1))ip1=ni-1;
 i0=I2D(ni,i,j);


grad_phi_x=faceq2*(phi[I2D(ni,ip1,j)]-phi[I2D(ni,im1,j)]);
grad_phi_y=faceq2*(phi[I2D(ni,i,jp1)]-phi[I2D(ni,i,jm1)]);
```

```
grad_phi_x+=faceq3*(phi[I2D(ni,ip1,jp1)]-
phi[I2D(ni,im1,jp1)]+phi[I2D(ni,ip1,jm1)]-phi[I2D(ni,im1,jm1)]);
grad_phi_y+=faceq3*(phi[I2D(ni,ip1,jp1)]+phi[I2D(ni,im1,jp1)]-
phi[I2D(ni,im1,jm1)]-phi[I2D(ni,ip1,jm1)]);

//interparticulepotentialinequilibriumvelocity
ux[i0]+=tau*(-G*phi[i0]*grad_phi_x)/rho[i0];
uy[i0]+=tau*(-G*phi[i0]*grad_phi_y)/rho[i0];

vx=ux[i0];
vy=uy[i0];

  ux2  =vx*vx;
  uy2  =vy*vy;
  uxy2 =ux2+uy2;
  uxy  =1.3*vx*vy;
  v_sq_term=1.5f*(vx*vx+vy*vy);
 //Evaluatethelocalequilibriumfvaluesinalldirections
 f0eq=(rho[i0]*faceq1*(1.f-v_sq_term));
 f1eq=(rho[i0]*faceq2*(1.f+3.f*vx+4.5f*ux2 -v_sq_term));
 f2eq=(rho[i0]*faceq2*(1.f+3.f*vy+4.5f*uy2-v_sq_term));//
 f3eq=(rho[i0]*faceq2*(1.f-3.f*vx+4.5f*ux2-v_sq_term));//
 f4eq=(rho[i0]*faceq2*(1.f-3.f*vy+4.5f*uy2-v_sq_term));//
 f5eq=(rho[i0]*faceq3*(1.f+3.f*(+vx+vy) +4.5f*(uxy2+uxy)-v_sq_term)); //
 f6eq=(rho[i0]*faceq3*(1.f+3.f*(-vx+vy) +4.5f*(uxy2-uxy)-v_sq_term));//
 f7eq=(rho[i0]*faceq3*(1.f+3.f*(-vx-vy) +4.5f*(uxy2+uxy)-v_sq_term));//
 f8eq=(rho[i0]*faceq3*(1.f+3.f*(+vx-vy) +4.5f*(uxy2-uxy)-v_sq_term));

 //Simulatecollisionsby"relaxing"towardthelocalequilibrium
 f0[i0]=f0[i0]-(f0[i0]-f0eq)/tau;
 f1[i0]=f1[i0]-(f1[i0]-f1eq)/tau;
 f2[i0]=f2[i0]-(f2[i0]-f2eq)/tau;
 f3[i0]=f3[i0]-(f3[i0]-f3eq)/tau;
 f4[i0]=f4[i0]-(f4[i0]-f4eq)/tau;
 f5[i0]=f5[i0]-(f5[i0]-f5eq)/tau;
 f6[i0]=f6[i0]-(f6[i0]-f6eq)/tau;
 f7[i0]=f7[i0]-(f7[i0]-f7eq)/tau;
 f8[i0]=f8[i0]-(f8[i0]-f8eq)/tau;
  }
}
}

///
voidsolid_BC(void)

//Thisistheboundaryconditionforasolidnode.Allthef'sarereversed-
//thisisknownas"bounce-back"

{
inti,j,i0;
floatf1old,f2old,f3old,f4old,f5old,f6old,f7old,f8old;

for(j=0;j<nj;j++){
 for(i=0;i<ni;i++){
 i0=I2D(ni,i,j);
 if(solid[i0]==0){
```

```
   //rho[i0]=0.09*(rhoh-rhol)+rhol;
  f1old=f1[i0];
  f2old=f2[i0];
  f3old=f3[i0];
  f4old=f4[i0];
  f5old=f5[i0];
  f6old=f6[i0];
  f7old=f7[i0];
  f8old=f8[i0];
//noslipboundary
  f1[i0]=f3old;
  f2[i0]=f4old;
  f3[i0]=f1old;
  f4[i0]=f2old;
  f5[i0]=f7old;
  f6[i0]=f8old;
  f7[i0]=f5old;
  f8[i0]=f6old;
 }
 }
}
}

//////
//////

voidper_BC(void)

//Allthef'sleavingthebottomofthedomain(j=0)enteratthetop(j=nj-1),
//andvice-verse

{
inti0,i1,j;

for(j=0;j<nj;j++){
 i0=I2D(ni,0,j);
 i1=I2D(ni,ni-1,j);
 f1[i0]=f1[i1];
 f5[i0]=f5[i1];
 f8[i0]=f8[i1];

 f3[i1]=f3[i0];
 f6[i1]=f6[i0];
 f7[i1]=f7[i0];

 }

}

///

voidapply_BCs(void)

//JustcallstheindividualBCfunctions

{
solid_BC();
```

```
 //per_BC();
}

///

voiddisplay(void)

//Thisfunctioniscalledautomatically,overandoveragain,byGLUT

{

inti,j,i0,icol,isol;
floatminvar,maxvar,frac;
 floatstart,stop;
 start=mulai;

 minvar=rhol;
maxvar=rhoh;


//dooneLatticeBoltzmannstep:stream,BC,collide:
 apply_BCs();
 collide();
 stream();
   ++i_steps;

//converttheplotvararrayintoanarrayofcolorstoplot
//ifthemeshpointissolid,makeitblack
for(j=0;j<nj;j++){
 for(i=0;i<ni;i++){
 i0=I2D(ni,i,j);
 frac=(plotvar[i0]-minvar)/(maxvar-minvar);
 icol=frac*ncol;
 isol=(int)solid[i0];
 plot_rgba[i0]=isol*cmap_rgba[icol];
 }
}
//Fillthepixelbufferwiththeplot_rgbaarray
glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB,ni*nj*sizeof(unsignedint),
  (void**)plot_rgba,GL_STREAM_COPY);

//Copythepixelbuffertothetexture,readytodisplay
glTexSubImage2D(GL_TEXTURE_2D,0,0,0,ni,nj,GL_RGBA,GL_UNSIGNED_BYTE,0);

//Renderonequadtothescreenandcolouritusingourtexture
//i.e.plotourplotvardatatothescreen
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_QUADS);
glTexCoord2f(0.0,0.0);
glVertex3f(0.0,0.0,0.0);
glTexCoord2f(1.0,0.0);
glVertex3f(ni,0.0,0.0);
glTexCoord2f(1.0,1.0);
glVertex3f(ni,nj,0.0);
glTexCoord2f(0.0,1.0);
glVertex3f(0.0,nj,0.0);
glEnd();
```

```
 glutSwapBuffers();
 //untuktimersaja
 if(i_steps%iterasi==iter1){
 stop=clock();
 printf("\nIteration:%6d;ElapsedTime(s):%4.2fperSecond\n",i_steps,(stop-
start)/CLOCKS_PER_SEC);
}
 if(i_steps%iterasi==iter2){
 stop=clock();
 printf("\nIteration:%6d;ElapsedTime(s):%4.2fperSecond\n",i_steps,(stop-
start)/CLOCKS_PER_SEC);
}
 if(i_steps%iterasi==0){
 stop=clock();
 printf("\nIteration:%6d;ElapsedTime(s):%4.2fperSecond\n",i_steps,(stop-
start)/CLOCKS_PER_SEC);
 system("PAUSE");
 exit(1);
 }

}

///
voidresize(intw,inth)
//GLUTresizecallbacktoallowustochangethewindowsize
{
width=w;
height=h;
glViewport(0,0,w,h);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
glOrtho(0.,ni,0.,nj,-200.,200.);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
}
```

Kode program untuk CPU

Kondisi satu fase pada kasus tiga

```
////////////////////////////////////////////////////////////////////
// arifiyanto hadinegoro
// single phase code
// edit from kode # Graham Pullan-Oct 2008 #
//
//       f6  f2   f5
//         \  |  /
//          \ | /
//           \|/
//       f3---|--- f1
//           /|\
//          / | \       dan f0 untuk area diam (zero)kecepatan
//         /  |  \
//       f7  f4   f8
////////////////////////////////////////////////////////////////////

#include <stdio.h>
#include <stdarg.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <GL/glew.h>
#include <GL/glut.h>

#define I2D(ni,i,j)(((ni)*(j))+i)
const double PI=3.141592653589793;

////////////////////////////////////////////////////////////////////

// OpenGL pixel buffer object and texture //
GLuint gl_PBO, gl_Tex;
// arrays //
float*f0,*f1,*f2,*f3,*f4,*f5,*f6,*f7,*f8;
float*tmpf0,*tmpf1,*tmpf2,*tmpf3,*tmpf4,*tmpf5,*tmpf6,*tmpf7,*tmpf8;
float*cmap,*plotvar;
int*solid;
unsigned int*cmap_rgba,*plot_rgba;  //rgba arrays for plotting

// scalars //
float tau,faceq1,faceq2,faceq3;
float vxin, roout;
float width, height;
int ni,nj,i,j,i0;
int ncol;
int ipos_old,jpos_old, draw_solid_flag;
int i_steps;
```

```
clock_t mulai, berhenti;
int iterasi=4000;
int iter1=1000;
int iter2=2000;

/////////////
// OpenGL function prototypes
//
void display(void);
void resize(int w, int h);

//
// Lattice Boltzmann function prototypes
//
void stream(void);
void collide(void);
void solid_BC(void);
void in_BC(void);
void apply_BCs(void);

unsigned int get_col(float min, float max, float val);
////////
int main(int argc, char**argv)
{
    int array_size_2d,totpoints,i;
    float rcol,gcol,bcol, R;

    FILE*fp_col;

    // The following parameters are usually read from a file, but
    // hard code them for the demo:
    ni=320;
    nj=112;
    vxin=0.04;
    roout=1.0;
    tau=0.51;
    // End of parameter list
      mulai=clock();
    // Write parameters to screen
    printf ("ni=%d\n", ni);
    printf ("nj=%d\n", nj);
    printf ("vxin=%f\n", vxin);
    printf ("roout=%f\n", roout);
    printf ("tau=%f\n", tau);

    totpoints=ni*nj;
    array_size_2d=ni*nj*sizeof(float);

    // Allocate memory for arrays

    f0=malloc(array_size_2d);
    f1=malloc(array_size_2d);
```

```
    f2=malloc(array_size_2d);
    f3=malloc(array_size_2d);
    f4=malloc(array_size_2d);
    f5=malloc(array_size_2d);
    f6=malloc(array_size_2d);
    f7=malloc(array_size_2d);
    f8=malloc(array_size_2d);

    tmpf0=malloc(array_size_2d);
    tmpf1=malloc(array_size_2d);
    tmpf2=malloc(array_size_2d);
    tmpf3=malloc(array_size_2d);
    tmpf4=malloc(array_size_2d);
    tmpf5=malloc(array_size_2d);
    tmpf6=malloc(array_size_2d);
    tmpf7=malloc(array_size_2d);
    tmpf8=malloc(array_size_2d);

    plotvar=malloc(array_size_2d);

    plot_rgba=malloc(ni*nj*sizeof(unsigned int));

    solid=malloc(ni*nj*sizeof(int));

    // Some factors used to calculate the f_equilibrium values

    faceq1=4.f/9.f;
    faceq2=1.f/9.f;
    faceq3=1.f/36.f;
//Initialise f's by setting them to the f_equilibirum values
assuming that the whole domain is at velocity vx=vxin vy=0 and
density ro=roout

    for (i=0; i<totpoints; i++){
f0[i]=faceq1*roout*(1.f                            -1.5f*vxin*vxin);
f1[i]=faceq2*roout*(1.f+3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f2[i]=faceq2*roout*(1.f                            -1.5f*vxin*vxin);
f3[i]=faceq2*roout*(1.f-3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f4[i]=faceq2*roout*(1.f                            -1.5f*vxin*vxin);
f5[i]=faceq3*roout*(1.f+3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f6[i]=faceq3*roout*(1.f-3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f7[i]=faceq3*roout*(1.f-3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
f8[i]=faceq3*roout*(1.f+3.f*vxin+4.5f*vxin*vxin-1.5f*vxin*vxin);
plotvar[i]=vxin;
solid[i]=1;
}

    //
    // Read in colourmap data for OpenGL display
    //
    fp_col=fopen("cmap.dat","r");
    if (fp_col==NULL){
```

```
        printf("Error: can't open cmap.dat \n");
        return 1;
      }
      // allocate memory for colourmap (stored as a linear array of
int's)
      fscanf (fp_col, "%d", &ncol);
      cmap_rgba=(unsigned int*)malloc(ncol*sizeof(unsigned int));
      // read colourmap and store as int's
      for (i=0;i<ncol;i++){
        fscanf(fp_col, "%f%f%f", &rcol, &gcol, &bcol);
        cmap_rgba[i]=((int)(255.0f)<< 24)| // convert colourmap to int
            ((int)(bcol*255.0f)<< 16)|
            ((int)(gcol*255.0f)<<  8)|
            ((int)(rcol*255.0f)<<  0);
      }
      fclose(fp_col);

        /////////////////////////
        ///bola
        for (i=0; i<ni; i++){
        for(j=0; j<nj; j++){
R=sqrtf(((float)i-(ni/2))*((float)i-(ni/2))+((float)j-
50.f)*((float)j-50.f));
                    if(R<=20.f){
                    i0=I2D(ni,i,j);
                    solid[i0]=0;
              }
        }}

//garis bawah
        for (i=0; i<=ni; i++){
              for(j=0; j<=10; j++){
                    i0=I2D(ni,i,j);
                    solid[i0]=0;
              }
        }
// garis atas
        for (i=0; i<=ni; i++){
              for(j=100; j<=nj-1; j++){
                    i0=I2D(ni,i,j);
                    solid[i0]=0;
              }
        }
//kotak awal
        for (i=0; i<=50; i++){
              for(j=10; j<=50; j++){
                    i0=I2D(ni,i,j);
                    solid[i0]=0;}}

        ////////////////////
      //
      // Iinitialise OpenGL display-use glut
```

```
    //
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(ni, nj);      // Window of ni x nj pixels
    glutInitWindowPosition(50, 50);  // position
    glutCreateWindow("2D LB");       // title

    // Check for OpenGL extension support
    printf("Loading extensions: %s\n",
glewGetErrorString(glewInit()));
    if(!glewIsSupported(
                        "GL_VERSION_2_0 "
                        "GL_ARB_pixel_buffer_object "
                        "GL_EXT_framebuffer_object "
                        )){
        fprintf(stderr, "ERROR: Support for necessary OpenGL
extensions missing.");
        fflush(stderr);
        return;
    }

    // Set up view
    glClearColor(0.0, 0.0, 0.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho(0,ni,0.,nj, -200.0, 200.0);
    // Create texture which we use to display the result and bind to
gl_Tex
    glEnable(GL_TEXTURE_2D);
    glGenTextures(1, &gl_Tex);                     // Generate 2D
texture
    glBindTexture(GL_TEXTURE_2D, gl_Tex);          // bind to gl_Tex
    // texture properties:
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_LINEAR);
    glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_LINEAR);
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, ni, nj, 0,
                 GL_RGBA, GL_UNSIGNED_BYTE, NULL);

    // Create pixel buffer object and bind to gl_PBO. We store the
data we want to
    // plot in memory on the graphics card-in a "pixel buffer". We
can then
    // copy this to the texture defined above and send it to the
screen
    glGenBuffers(1, &gl_PBO);
    glBindBuffer(GL_PIXEL_UNPACK_BUFFER_ARB, gl_PBO);
    printf("Buffer created.\n");
```

```
    // Set the call-back functions and start the glut loop
    printf("Starting GLUT main loop...\n");
    glutDisplayFunc(display);
    glutReshapeFunc(resize);
    glutIdleFunc(display);
    glutMainLoop();

    return 0;
}

////////////

void stream(void)

// Move the f values one grid spacing in the directions that they
are pointing
// i.e. f1 is copied one location to the right, etc.

{
    int i,j,im1,ip1,jm1,jp1,i0;

    // Initially the f's are moved to temporary arrays
    for (j=0; j<nj; j++){
      jm1=j-1;
      jp1=j+1;
      if (j==0)jm1=0;
      if (j==(nj-1))jp1=nj-1;
      for (i=1; i<ni; i++){
          i0 =I2D(ni,i,j);
          im1=i-1;
          ip1=i+1;
          if (i==0)im1=0;
          if (i==(ni-1))ip1=ni-1;
          tmpf1[i0]=f1[I2D(ni,im1,j)];
          tmpf2[i0]=f2[I2D(ni,i,jm1)];
          tmpf3[i0]=f3[I2D(ni,ip1,j)];
          tmpf4[i0]=f4[I2D(ni,i,jp1)];
          tmpf5[i0]=f5[I2D(ni,im1,jm1)];
          tmpf6[i0]=f6[I2D(ni,ip1,jm1)];
          tmpf7[i0]=f7[I2D(ni,ip1,jp1)];
          tmpf8[i0]=f8[I2D(ni,im1,jp1)];
      }
    }

    // Now the temporary arrays are copied to the main f arrays
    for (j=0; j<nj; j++){
      for (i=1; i<ni; i++){
          i0=I2D(ni,i,j);
          f1[i0]=tmpf1[i0];
          f2[i0]=tmpf2[i0];
          f3[i0]=tmpf3[i0];
          f4[i0]=tmpf4[i0];
```

```
          f5[i0]=tmpf5[i0];
          f6[i0]=tmpf6[i0];
          f7[i0]=tmpf7[i0];
          f8[i0]=tmpf8[i0];
      }
    }
}

///////////////////
void collide(void)
// Collisions between the particles are modeled here. We use the
very simplest model which assumes the f's change toward the local
equlibrium value (base on density and velocity at that point)over a
fixed timescale, tau

{
    int i,j,i0;
    float ro, rovx, rovy, vx, vy, v_sq_term;
    float f0eq, f1eq, f2eq, f3eq, f4eq, f5eq, f6eq, f7eq, f8eq;
    float rtau, rtau1;


    // Some useful constants
    rtau=1.f/tau;
    rtau1=1.f-rtau;

    for (j=0; j<nj; j++){
      for (i=0; i<ni; i++){

          i0=I2D(ni,i,j);

          // Do the summations needed to evaluate the density and
components of velocity
ro=f0[i0]+f1[i0]+f2[i0]+f3[i0]+f4[i0]+f5[i0]+f6[i0]+f7[i0]+f8[i0];
      rovx=f1[i0]-f3[i0]+f5[i0]-f6[i0]-f7[i0]+f8[i0];
      rovy=f2[i0]-f4[i0]+f5[i0]+f6[i0]-f7[i0]-f8[i0];
          vx=rovx/ro;
          vy=rovy/ro;
    // Also load the velocity magnitude into plotvar-this is what we
will
          // display using OpenGL later
          plotvar[i0]=sqrt(vx*vx+vy*vy);

          v_sq_term=1.5f*(vx*vx+vy*vy);

    // Evaluate the local equilibrium f values in all directions
f0eq=ro*faceq1*(1.f-v_sq_term);
f1eq=ro*faceq2*(1.f+3.f*vx+4.5f*vx*vx-v_sq_term);
f2eq=ro*faceq2*(1.f+3.f*vy+4.5f*vy*vy-v_sq_term);
f3eq=ro*faceq2*(1.f-3.f*vx+4.5f*vx*vx-v_sq_term);
f4eq=ro*faceq2*(1.f-3.f*vy+4.5f*vy*vy-v_sq_term);
f5eq=ro*faceq3*(1.f+3.f*(vx+vy)+4.5f*(vx+vy)*(vx+vy)-v_sq_term);
```

```
f6eq=ro*faceq3*(1.f+3.f*(-vx+vy)+4.5f*(-vx+vy)*(-vx+vy)-v_sq_term);
f7eq=ro*faceq3*(1.f+3.f*(-vx-vy)+4.5f*(-vx-vy)*(-vx-vy)-v_sq_term);
f8eq=ro*faceq3*(1.f+3.f*(vx-vy)+4.5f*(vx-vy)*(vx-vy)-v_sq_term);

          // Simulate collisions by "relaxing" toward the local
equilibrium
          f0[i0]=rtau1*f0[i0]+rtau*f0eq;
          f1[i0]=rtau1*f1[i0]+rtau*f1eq;
          f2[i0]=rtau1*f2[i0]+rtau*f2eq;
          f3[i0]=rtau1*f3[i0]+rtau*f3eq;
          f4[i0]=rtau1*f4[i0]+rtau*f4eq;
          f5[i0]=rtau1*f5[i0]+rtau*f5eq;
          f6[i0]=rtau1*f6[i0]+rtau*f6eq;
          f7[i0]=rtau1*f7[i0]+rtau*f7eq;
          f8[i0]=rtau1*f8[i0]+rtau*f8eq;
      }
    }
}

//
void solid_BC(void)

// This is the boundary condition for a solid node. All the f's are
reversed -
// this is known as "bounce-back"

{
    int i,j,i0;
    float f1old,f2old,f3old,f4old,f5old,f6old,f7old,f8old;

    for (j=0;j<nj;j++){
      for (i=0;i<ni;i++){
          i0=I2D(ni,i,j);
          if (solid[i0]==0){
            f1old=f1[i0];
            f2old=f2[i0];
            f3old=f3[i0];
            f4old=f4[i0];
            f5old=f5[i0];
            f6old=f6[i0];
            f7old=f7[i0];
            f8old=f8[i0];

            f1[i0]=f3old;
            f2[i0]=f4old;
            f3[i0]=f1old;
            f4[i0]=f2old;
            f5[i0]=f7old;
            f6[i0]=f8old;
            f7[i0]=f5old;
            f8[i0]=f6old;
          }
```

```
      }
    }
}


void in_BC(void)

// This inlet BC is extremely crude but is very stable
// We set the incoming f values to the equilibirum values assuming:
// ro=roout; vx=vxin; vy=0

{
    int i0, j;
    float f1new, f5new, f8new, vx_term;

    vx_term=1.f+3.f*vxin +3.f*vxin*vxin;
    f1new=roout*faceq2*vx_term;
    f5new=roout*faceq3*vx_term;
    f8new=f5new;

    for (j=0; j<nj; j++){
      i0=I2D(ni,0,j);
      f1[i0]=f1new;
      f5[i0]=f5new;
      f8[i0]=f8new;
    }

}
void apply_BCs(void)
// Just calls the individual BC functions
{

    solid_BC();
    in_BC();

}
///////
void display(void)

// This function is called automatically, over and over again,  by
GLUT

{
    int i,j,i0,icol,isol;
    float minvar,maxvar,frac;
      float start, stop;
      start=mulai;

    // set upper and lower limits for plotting
    minvar=0.0;
    maxvar=0.2;
```

```
    // do one Lattice Boltzmann step: stream, BC, collide:
      i_steps++;
    stream();
    apply_BCs();
    collide();

    // convert the plotvar array into an array of colors to plot
    // if the mesh point is solid, make it black
    for (j=0;j<nj;j++){
      for (i=0;i<ni;i++){
          i0=I2D(ni,i,j);
          frac=(plotvar[i0]-minvar)/(maxvar-minvar);
          icol=frac*ncol;
          isol=(int)solid[i0];
          plot_rgba[i0]=isol*cmap_rgba[icol];
      }
    }

    // Fill the pixel buffer with the plot_rgba array
    glBufferData(GL_PIXEL_UNPACK_BUFFER_ARB,ni*nj*sizeof(unsigned
int),
            (void**)plot_rgba,GL_STREAM_COPY);

    // Copy the pixel buffer to the texture, ready to display

glTexSubImage2D(GL_TEXTURE_2D,0,0,0,ni,nj,GL_RGBA,GL_UNSIGNED_BYTE,0
);

    // Render one quad to the screen and colour it using our texture
    // i.e. plot our plotvar data to the screen
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_QUADS);
    glTexCoord2f (0.0, 0.0);
    glVertex3f (0.0, 0.0, 0.0);
    glTexCoord2f (1.0, 0.0);
    glVertex3f (ni, 0.0, 0.0);
    glTexCoord2f (1.0, 1.0);
    glVertex3f (ni, nj, 0.0);
    glTexCoord2f (0.0, 1.0);
    glVertex3f (0.0, nj, 0.0);
    glEnd();

      glutSwapBuffers();
      // untuk timer saja
if (i_steps%iterasi == 1){
stop= clock();
printf("\n Iteration: %6d; Elapsed Time (s): %4.2f perSecond \n",
i_steps, (stop-start)/CLOCKS_PER_SEC);
}
if (i_steps%iterasi == iter1){
stop= clock();
```

```
printf("\n Iteration: %6d; Elapsed Time (s): %4.2f perSecond \n",
i_steps, (stop-start)/CLOCKS_PER_SEC);
}
if (i_steps%iterasi == iter2){
stop= clock();
printf("\n Iteration: %6d; Elapsed Time (s): %4.2f perSecond \n",
i_steps, (stop-start)/CLOCKS_PER_SEC);
}
if (i_steps%iterasi == 0){
stop= clock();
printf("\n Iteration: %6d; Elapsed Time (s): %4.2f perSecond \n",
i_steps, (stop-start)/CLOCKS_PER_SEC);
system("PAUSE");
exit(1);
}
}


void resize(int w, int h)
// GLUT resize callback to allow us to change the window size
{
   width=w;
   height=h;
   glViewport (0, 0, w, h);
   glMatrixMode (GL_PROJECTION);
   glLoadIdentity ();
   glOrtho (0., ni, 0., nj, -200. ,200.);
   glMatrixMode (GL_MODELVIEW);
   glLoadIdentity ();
}
```