Eleventh International Conference on CFD in the Minerals and Process Industries
CSIRO, Melbourne, Australia
7-9 December 2015

COMPUTATIONAL FLUID DYNAMICS AND GPUS

Tomasz BEDNARZ[1,2*], Steven PSALTIS[1,2], John A TAYLOR[3], Maciej MATYKA[4] and Ian TURNER[1,2]

[1] School of Mathematical Sciences, Queensland University of Technology, Brisbane, AUSTRALIA
*Corresponding author, E-mail address: tomasz.bednarz@qut.edu.au

[2]ARC Centre of Excellence for Mathematical and Statistical Frontiers

[3] CSIRO, Canberra, AUSTRALIA

[4] Faculty of Physics and Astronomy, University of Wroclaw, Wroclaw, POLAND

ABSTRACT

Computational Fluid Dynamics (CFD) simulations are aimed to reconstruct the reality of fluid motion and behaviour as accurately as possible, to better understand the natural phenomena under specified conditions. Ideally, computational models would need to cover different scales and geometric configurations, and the classic CFD solvers most often require long computational times to satisfy the convergence criteria. With the advent of heterogeneous compute platforms (including Graphics Processing Units GPUs), CFD algorithms can now be implemented to give results in near real-time. The current paper briefly reviews and demonstrates in a general way, two methods able to harness the power of GPUs, to speed up numerical simulations of fluid flows for industrial applications. These include the Highly Simplified Marker and Cell Method (HSMAC), and Lattice-Boltzmann Method (LBM) implemented on GPUs using OpenCL. This paper describes general capabilities for compute and graphics, and method presented could be used to simulate various cases with a specific boundary conditions.

NOMENCLATURE

C_x	coefficient X (1-5) depending on a specific case
g	gravitational acceleration
f_i	pocket distribution
l	length of cube
P	pressure
Pr	Prandl number (ratio of momentum and thermal diffusivities)
Ra	Rayleigh number (associated with buoyancy driven flow, describes strength of convection)
t	time, time-step
U	velocity
α	thermal diffusivity
$\Delta\theta$	maximum temperature difference
ρ	density
β	thermal expansion coefficient
ν	kinematic viscosity

INTRODUCTION

Experiments with fluids are usually very expensive and many times not feasible, or require work in harmful environments, etc. Therefore, in such cases, it is advantageous to replace them with numerical modelling whenever possible. The Computational Fluid Dynamics (CFD) that aims to reconstruct the reality of fluid motion and behaviour as accurately as possible in order to better understand the natural phenomena under specified conditions. Ideally, general solutions can also cover different scales and geometric configurations and should be in agreement with equivalent experimental results.
Unfortunately, due to expensive algorithms, classical CFD codes most often require long computational times to satisfy the convergence criteria. With the advent of high-performance GPUs with massively parallel multi-threaded architectures, basic CFD algorithms can now be implemented to give results in near real-time. The algorithms used to solve these problems utilize GPU and in this case OpenCL. Presented results demonstrate that GPUs can be successfully used to accelerate fluid simulations. We have seen significant gains in productivity and opportunity as a result of leveraging GPUs, being able to tackle computational problems in which execution time was previously infeasible. OpenCL was chosen for generality of running simulations on various platforms compared to CUDA which is just locking users to a specific platform. The codes could be easily ported to CUDA without difficulty.

HSMAC FOR THERMAL FLUIDS

Model Equations

In order to solve general incompressible thermo-fluids problems in the Cartesian coordinate system, the Navier-Stokes and energy equations can be defined as follows:

$$\vec{\nabla}\cdot\vec{U} = 0 \tag{1}$$

$$\frac{\mathrm{D}\vec{U}}{\mathrm{D}t} = -C_1\vec{\nabla}P + C_2\nabla^2\vec{U} + C_3(\theta - C_4) \tag{2}$$

$$\frac{\mathrm{D}\theta}{\mathrm{D}t} = C_5\nabla^2\theta \tag{3}$$

For generality, the equations presented are in non-dimensional form. The same methodology can be applied to simulate a wide range of different fluid flows: pipe flows, forced convection, magneto-thermal convection, scaling analysis, exchange flows in reservoir models, mixing, fountain flows, bubble flow, step flows, heat exchangers, ventilation problems, etc. For instance, for square cavity heated from one vertical wall and cooled from the opposite one with top and bottom walls kept adiabatic, the coefficients for the dimensionless solution are defined as follows:

$$C_1 = 1;\ C_2 = \mathrm{Pr} = \frac{\nu}{\alpha};\ C_3 = PrRa = \frac{\nu}{\alpha}\cdot\frac{g\beta(\Delta\theta)l^3}{\alpha\nu}$$
$$C_4 = 0;\ C_5 = 1 \tag{4}$$

Numerical Approach

These equations are approximated with finite difference equations and the HSMAC (Highly Simplified Marker and Cell) method (Bednarz et al. 2005-2010, Hirt 1975, A sample CPU based code is available online under https://bitbucket.org/tomaszbednarz/hsmac-bfc-2d) that is used to iterate mutually the pressure and velocity fields on a staggered mesh/grid allocation system, see Figure 1.

The inertial terms in the momentum equations are approximated using a third-order upwind UTOPIA scheme (Tagawa 1996). The absolute convergence criteria for the numerical solutions are specified based on the residual sums of all conserved quantities. If the residual sum is less than 10^{-6} for each conserved quantity, the equations are deemed to have converged at a specific time step. The time-step is chosen to ensure numerical stability according to the CFL condition. The numerical methods used in this work for simulation of natural convection have been widely verified by co-workers, by both numerical and experimental investigations for closely related problems (Bednarz et al. 2005-2010). In this paper we focused on 2D simulation, for simplicity, but also as the tested convections modes are accurately approximated using 2D geometry.

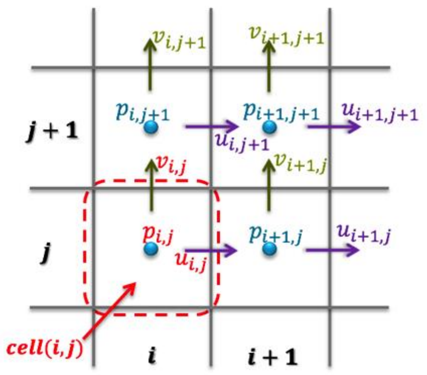


Figure 1: Staggered mesh grid allocation (notice location of pressure, and velocity components for a cell).

OpenCL Implementation

The numerical code is ported from the CPU to GPU version using the OpenCL API (https://www.khronos.org). This was motivated by the need for improving computation speed, as in some cases, e.g. computation of single case of boundary layer evolution (Bednarz et al. 2009) could take up to 12 hours on grid size 256x256. Therefore, all critical parts of the previously available CPU code are re-implemented in several OpenCL kernels that could be executed by thousands of simultaneous threads by a GPU. **Figure** 2 shows the flow chart of our HSMAC OpenCL solver. As seen, the initialization part includes: reading initial configuration files describing geometry and parameters of the problem to be computed, allocating memory for all field variables (pressure, velocity components, temperatures and spatial coordinates), preparing boundary condition flags (to mark regions where the boundaries are located and their type). Once that's done, OpenCL is initialized, the proper compute device is attached to its context and the CL program is compiled. The device memory buffers are also created and filled with initial data.

The solver / runtime execution of all OpenCL kernels is controlled by the host device (CPU). The flow chart shows two main loops: the outer loop, which is responsible for general time-step iterations and the inner loop used for the mutual iteration of velocity and pressure fields to satisfy the continuity equation, before solving each energy equation time-step (Hirt 1975). For instance, calculation of the pressure and velocity correction is done by execution of the kernel `krnl_pressure_velocity_correction`, calculation of the energy equation by `krnl_energy`, etc. In addition the interoperability feature of OpenCL with OpenGL can be used to visualize the results while they are still under computation. After reaching the final time step, all results can be saved on disk for further analysis and visualisation purposes.

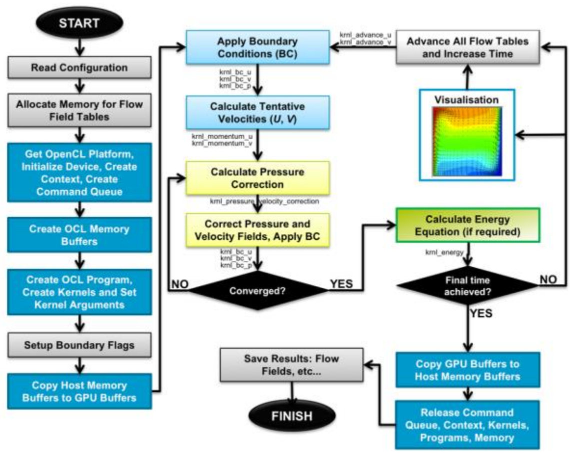
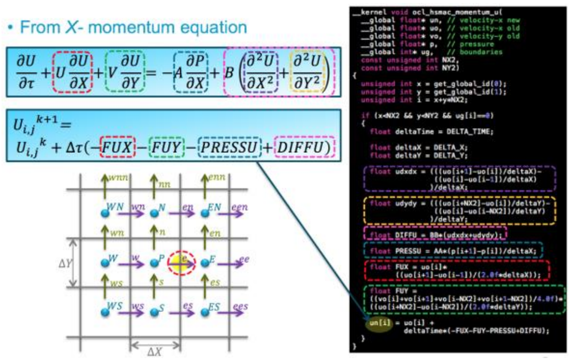

START
Read Configuration
Allocate Memory for Flow Field Tables
Get OpenCL Platform, Initialize Device, Create Context, Create Command Queue
Create OCL Memory Buffers
Create OCL Program, Create Kernels and Set Kernel Arguments
Setup Boundary Flags
Copy Host Memory Buffers to GPU Buffers
Apply Boundary Conditions (BC)
krnl_bc_u
krnl_bc_v
krnl_bc_p
Calculate Tentative Velocities (U, V)
krnl_momentum_u
krnl_momentum_v
Calculate Pressure Correction
krnl_pressure_velocity_correction
Correct Pressure and Velocity Fields, Apply BC
krnl_bc_u
krnl_bc_v
krnl_bc_p
NO
Converged?
YES
krnl_advance_u
krnl_advance_v
Advance All Flow Tables and Increase Time
Visualisation
Calculate Energy Equation (if required)
krnl_energy
Final time achieved?
NO
YES
Copy GPU Buffers to Host Memory Buffers
Release Command Queue, Context, Kernels, Programs, Memory
Save Results: Flow Fields, etc...
FINISH

Figure 2: Flow chart of OpenCL based HSMAC solver.

Figure 3 shows a simplified code snippet of a sample OpenCL kernel being used to calculate the horizontal component of the tentative velocity from the momentum equation. Please note, that all derivatives for the horizontal velocity are calculated at the middle of the right cell edge. For simplicity of presentation, convection acceleration is calculated using central approach. In real simulations, as mentioned UTOPIA was used for accuracy and stability.


• From X- momentum equation
$\frac{\partial U}{\partial \tau} + U\frac{\partial U}{\partial X} + V\frac{\partial U}{\partial Y} = -A\frac{\partial P}{\partial X} + B\left(\frac{\partial^2 U}{\partial X^2} + \frac{\partial^2 U}{\partial Y^2}\right)$
$U_{i,j}^{k+1} = U_{i,j}^{k} + \Delta\tau(-FUX - FUY - PRESSU + DIFFU)$
wnn
nn
enn
WN
wn
N
en
EN
een
wn
n
en
ΔY
W
w
P
e
E
ee
ws
s
es
WS
ws
S
es
ES
ees
ΔX

```
__kernel void ocl_hsmac_momentum_u(
  __global float* un, // velocity-x new
  __global float* uo, // velocity-x old
  __global float* vo, // velocity-y old
  __global float* p,  // pressure
  __global int* ug,   // boundaries
  const unsigned int NX2,
  const unsigned int NY2)
{
  unsigned int x = get_global_id(0);
  unsigned int y = get_global_id(1);
  unsigned int i = x+y*NX2;

  if (x<NX2 && y<NY2 && ug[i]==0)
  {
    float deltaTime = DELTA_TIME;

    float deltaX = DELTA_X;
    float deltaY = DELTA_Y;

    float udxdx = (((uo[i+1]-uo[i])/deltaX)-
                   ((uo[i]-uo[i-1])/deltaX)
                  )/deltaX;

    float udydy = (((uo[i+NX2]-uo[i])/deltaY)-
                   ((uo[i]-uo[i-NX2])/deltaY)
                  )/deltaY;

    float DIFFU = BB*(udxdx+udydy);

    float PRESSU = AA*(p[i+1]-p[i])/deltaX;

    float FUX = uo[i]*
          ((uo[i+1]-uo[i-1])/(2.0f*deltaX));

    float FUY =
    ((vo[i]+vo[i+1]+vo[i-NX2]+vo[i+1-NX2])/4.0f)*
    ((uo[i+NX2]-uo[i-NX2])/(2.0f*deltaY));

    un[i] = uo[i] +
          deltaTime*(-FUX-FUY-PRESSU+DIFFU);
  }
}
```

Figure 3: OpenCL code snippet of horizontal component of the momentum equation.

These results showed speed-ups that can be achieved using GPU compute devices. Tests comparing NVIDIA S2050 to Intel Core i7 showed an average 200-290X speedup of GPU/CPU. These results give an indication of what can be achieved when using OpenCL with today's GPUs for solving explicitly the Navier-Stokes equation for natural convection flows, i.e. obtaining converged results in a few minutes instead of 10-12 hours. Please also refer to earlier published papers, that focus on applications for natural convection flows (including magnetic convection), and scaling analysis that verifies the correctness of the HSMAC method for fluid simulations (Bednarz, 2009).

LATTICE-BOLTZMANN METHOD

Method

The Lattice-Boltzmann Method (LBM) is a newer class of computational fluid dynamics schemes that simulates fluid flow by solving a discretised Boltzmann equation in conjunction with particle collision models. Therefore, the method is dissimilar to other conventional CFD methods such as FDM presented earlier, in that it models fluids as fictitious, discrete particles versus a continuous medium, algorithmically performs particle streaming, and accounts for particle collision and boundary operators within a given domain. The HSMAC method presented above numerically solve the conservation equations of particular macroscopic quantities such as mass, momentum and energy. The LBM however, simulates fluid flow by tracking the collective kinetic behaviour of microscopic particles at each node within the lattice, then quantitatively accumulates "the behaviour" to obtain the averaged macroscopic properties. The collective behaviour of microscopic particles is represented by a distribution function, which indicates the most probable number of particles in a spatial unit volume at a specified position in time with given velocity.

Due to its local character the basic LBM algorithm, it is very attractive for multi-core processors. The LBM has been extensively used in a non real-time GPU applications on single and multi-GPU systems using Nvidia only CUDA API (J. Tölke, 2010) as well as OpenCL (McIntosh-Smith et al. 2014) and OpenACC (Calore et al., 2015). Here, we use LBM to get real-time fluid flow with an interactive visualisation on large 3-m hemispherical dome display with modern OpenGL using portable, vendor independent OpenCL implementation.

For sake of simplicity of the prototype used for real-time tests, in our experiments, the D2Q9 model was used: each node builds nine interactions (Figure 4). Each node is associated with various attributes, such as packet distribution f_i, density ρ, and velocity u. The balance is represented by the following equation:

$$f_i(x+c_i,t+1) - f_i(x,t) = -\frac{1}{t}\left(f_i(x,t) - f_i^{eq}(x,t)\right) \qquad (5)$$

where the left hand side of the equation represents the transport component, and right hand side collision. As the real challenge is three-dimensional flow we note here that the following discussion on implementation is valid in 3D case. The only modification required is basic extension of all fields into third dimension while the fundamental algorithm of LBM remains the same.

Densities and velocities are derived from packet distributions, please refer to Chen and Doolean, (1993) for full details.

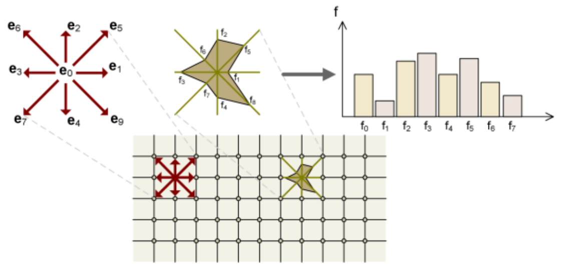


Figure 4: The D2Q9 model, and corresponding packet distribution function.

The code snippet below shows a basic CPU implementation for calculating density and velocity components, operating on two dimensional arrays:

```
for (int i=0; i<L; i++)
     for (int j=0; j<L; j++)
     {
          idx =  i+j*L;
          U[idx]=V[idx]=R[idx]=0;
          for (int k=0; k<9; k++)
          {
               tmp = df[c][idx][k];
               R[idx] = R[idx] + tmp;
               U[idx] = U[idx] + tmp*ex[k];
               V[idx] = V[idx] + tmp*ey[k];
          }
          U[idx] = U[idx]/R[idx] + fx;
          V[idx] = V[idx]/R[idx];
          //
          // streaming + collision here
          // ...
}
```

and also transport and collision:

```
// transport + collistion
for (int k=0; k<9; k++)
{
     int ip = (i+ex[k]+L)%(L);
     int jp = (j+ey[k]+L)%(L);
     tmp = ex[k]*U[idx] + ey[k]*V[idx];
     feq = w[k]*rho*(1-1.5*(U[idx]*U[idx]+V[idx]*V[idx])+
          3*tmp + 4.5 *tmp*tmp);
     if(FLAG[ip+jp*L] == 1)
          df[1-c][idx][inv[k]] = (1-omega)*df[c][idx][k]+
               omega*feq
     else
          df[1-c][ip+jp*L][k] = (1-omega)*df[c][idx][k]+
               omega*feq;
}
```

GPU Implementation

The main CPU code for solving basic flows using LBM is very concise and simple, and can be extended easily to simulate, for instance, turbulence. The calculation for each node can be very easily parallelised and can utilise two-dimensional GPU architecture memory buffers effectively. The code was initially ported to OpenGL compute shaders (http://bit.ly/1KbTdwe), but eventually migrated to OpenCL.

The OpenCL implementation main kernel call is listed below, where basic LBM algorithm is placed below the computation of "idx" node index:

```
kernel void lbm(
     global float *U,
     global float *V,
     global float *R,
     global int *F,
     global float *f0,
     global float *f1)
{
     int i = get_global_id(0);
     int j = get_global_id(1);

     int idx = i+j*NX;

     ...
}
```

LBM RESULTS

For visualisation, the high level C/C++ library openFrameworks was used for all low-level graphics maintenance such as opening the window, drawing primitives, and displaying on-screen information. Mass-less particles were placed into the fluid flow profiles, as seen in Figure 5.

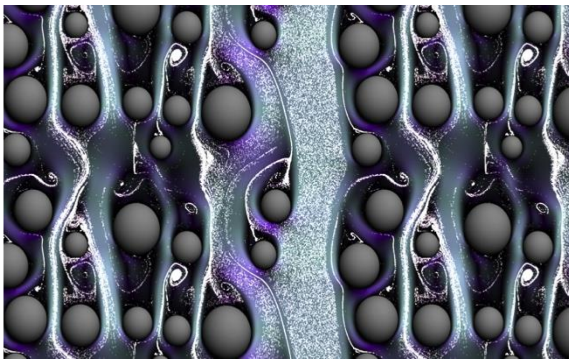


Figure 5: Real-time LBM, runs 10FPS on GTX750M, domain size 1920x1080.

Hemispherical Dome Visualisation

The OpenCL LBM code was integrated with a C++ dome twisting code, to display a full screen simulation on the hemispherical dome, as seen in Figure 6.

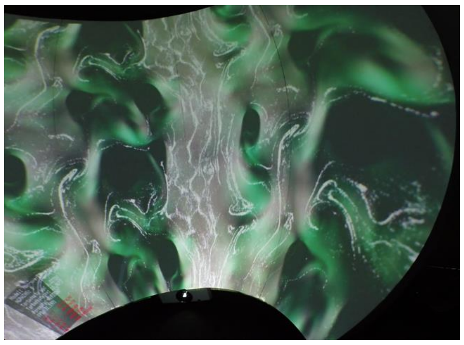


Figure 6: Visualisation of fluid flow on the 3-m hemispherical dome.

We found the overall speed of the LBM satisfactory for real-time applications. One should, however, keep in mind the limitations of the scheme itself as the method is nearly incompressible which may limit its usage in some specific applications. Also, an accuracy of the boundaries depends on the way they are handled which has to be considered if accuracy of the solution is an issue (Latt et. al. 2008).

CONCLUSION

The present paper briefly outlined the usefulness of GPUs to solve fluid flows problems. The numerical simulations were fast enough to be useful for real-time interactive visualisations.

REFERENCES

BEDNARZ, T., TAGAWA, T., KANEDA, M., OZOE, H. and SZMYD, J.S., (2005) “The convection of air in a cubic enclosure with an electric coil inclined in general orientations”, *Fluid Dynamics Research*, **36**, 91-106.

BEDNARZ, T., LIN, W., PATTERSON, J.C., LEI, C. and ARMFIELD, S.W., (2009) “Scaling for unsteady thermomagnetic convection boundary layer of paramagnetic fluids of Pr>1 in microgravity conditions”, *Int. J. of Heat and Fluid Flow*, **30**, 1157-1170.

BEDNARZ, T., CARIS, C., and TAYLOR, J., (2010), “A practical introduction to Computational Fluid Dynamics on GPUs”, *Proc. GPU Technology Conference*, San Jose, USA, September 20-23.

BOTELLA, O., and PEYRET, R., (1998), “Benchmark spectral results on the lid-driven cavity flow”, *Computers & Fluids*, **27**, 421-433.

CALORE, E., KRAUS, J., SCHIFANO, S.F., and , TRIPICCIONE, R., (2015) “Accelerating Lattice Boltzmann Applications with OpenACC”, Euro-Par 2015: Parallel Processing, Lecture Notes in Computer Science, **9322**, 613-624.

CHEN, S., and DOOLEAN, G.D., (1998), “Lattice Boltzmann Method for Fluid Flows”, *Annual Review of Fluid Mechanics*, **30**, 329-264.

DE VAHL DAVIS, G., (1983), “Natural convection of air in a square cavity: a bench mark numerical solution”, *Int. J. of Numerical Methods in Fluids*, **3**, 249-264.

HIRT, C.W., NICHOLS, B.D., and ROMERO, N., (1975), “A numerical solution algorithm for transient fluid flow”, *Los Alamos Scientific Laboratory*, LA-5852.

LATT, J. ,CHOPARD, B., MALASPINAS, O., DEVILLE, M., and MICHLER, A., (2008) "Straight velocity boundaries in the lattice Boltzmann method", *Phys. Rev. E*, **77**, 056703.

MCINTOSH-SMITH, S., and CURRAN, D., 2013 and 2014, “Evaluation of a performance portable Lattice Boltzmann code using OpenCL”, IWOCL '14 Proceedings of the International Workshop on OpenCL 2013 & 2014.

TAGAWA, T., and OZOE, H., (1996), “Effect of Prandtl number and computational schemes on the oscillatory natural convection in an enclosure”, *Numerical Heat Transfer A*, **30**, 271-282.

TÖLKE, J., (2010), “Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA”, *Comput. Visual Sci.*, **13**, 29–39.