Optimized CUDA Implementation of a Navier-Stokes Based Flow Solver for the 2D Lid Driven Cavity

Michael Bader[1], Hans-Joachim Bungartz[1], Dheevatsa Mudigere [1, 2], Srihari Narasimhan[2], Babu Narayanan[2]

[1] Chair for Scientific Computing, Department of Informatics, Technische Universität München, Munich, Germany.
[2] Computing and Decision Sciences Lab, GE Global Research, JFWTC, Bangalore, India.

imagination at work

Problem

• Novel and near-optimal GPU implementation of a basic Navier-Stokes based 2D flow solver for incompressible, laminar, non-stationary flow.

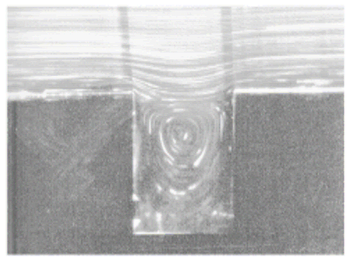
• 2D lid-driven cavity - classic CFD problem.

•Navier-Stokes equations – Mathematical model to describe non-stationary incompressible viscous fluids.

$$\frac{\partial u}{\partial t}+\frac{\partial p}{\partial x}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 u}{\partial x^2}+\frac{\partial^2 u}{\partial y^2}\right)-\frac{\partial\left(u^2\right)}{\partial x}-\frac{\partial(uv)}{\partial y}+g_x, \quad (1)$$

$$\frac{\partial v}{\partial t}+\frac{\partial p}{\partial y}=\frac{1}{\mathrm{Re}}\left(\frac{\partial^2 v}{\partial x^2}+\frac{\partial^2 v}{\partial y^2}\right)-\frac{\partial\left(v^2\right)}{\partial v}-\frac{\partial(uv)}{\partial x}+g_y, \quad (2)$$

$$\frac{\partial u}{\partial x}+\frac{\partial v}{\partial y}=0 \quad (3)$$



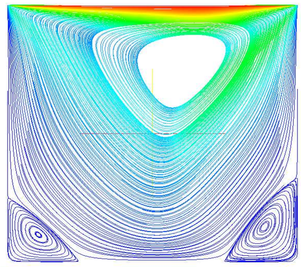


Fig.1: 512x512 steady-state for Re=100

Method

•Solved by the projection method - decouples velocity and pressure calculations

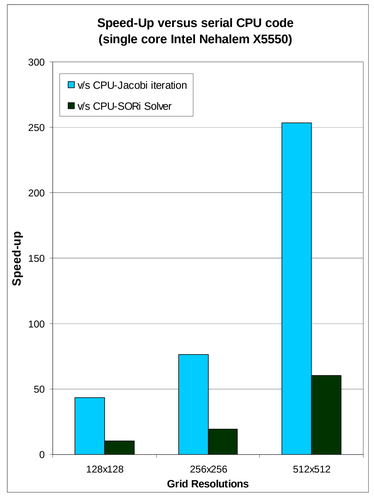
• Solve pressure equation using intermediate velocities at each time step

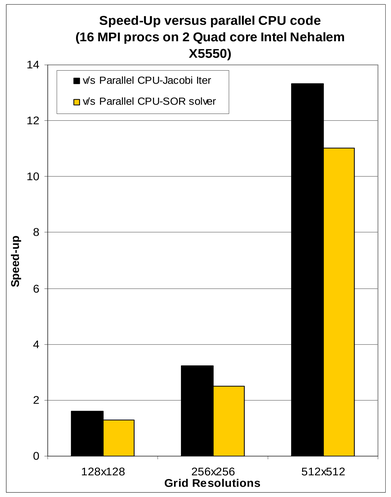
•Finite difference discretization scheme, with a **staggered grid**

• Explicit Euler scheme for the (outer) time stepping loop

• **Jacobi Iteration** (GPU) and **SOR solver** (CPU) for solving the discretized Poisson pressure equation

Results





Grid-Resolution	Intel Nehalem X5550 (s)		Tesla c1060			Speed-up v/s CPU	
	Serial	16 MPI procs	Time (s)	GB/s	GFLOPS	Serial	16 MPI procs
128 x 128	1241.1	45.8	28.6	20.4	34.8	**43.3x**	**1.6x**
256 x 256	23521.0	989.7	307.1	34.9	60.7	**76.5x**	**3.2x**
512 x 512	577510.0	30379.3	2280.8	55.7	110.3	**253.2x**	**13.3x**

• GPU implementation (NVIDIA Tesla C1060) compared with both serial (Single core Intel Nehalem X5550) and parallel (16 MPI procs on 2 Quad core Intel Nehalem X5550) CPU codes

• Compared also with the CPU implementations with the SOR solver

• Successive Over Relaxation (SOR) solver – iterative solver for PDEs, uses available updated values in each iteration, GPU unfriendly, faster convergence

Future Work

• Promising results encourage further extension into a comprehensive, generic flow solver on the GPU, capable of catering to real world applications.

• Further optimizing the GPU implementation – reducing synchronization overheads, more efficient use of available bandwidth...targeting maximum performance.

GPU Implementation

• Five separate kernels – because of global synchronization requirements

• Boundary conditions, implicitly handled within the computation kernels itself

```
/* Time stepping loop */
for (t = 0; t < t_end; t+=dt)
{
      /* Kernel for partial velocities calculation (along with boundary setting for F&G) */
      calculateFG <<<grid, threads>>> (d_F, d_G, d_U, d_V, nx, alpha, dx, dy, dt, Re, GX, GY);

      /* Kernel for RHS calculation */
      interpolateFG <<<grid, threads>>> (d_R, d_F, d_G, nx, dx, dy, dt);

      /* Inner Jacobi solver loop (end_cond: res and itermax) */
      while ((it<itermax && res > eps))
      {
            /* kernel solver calculation */
            calculateP <<<grid, threads>>> (d_P, d_R, nx, dx, dy);

            /* Kernel for residual calculation(?? reduction operation needed)*/
            calculateRES <<<grid, threads>>> (d_r, d_P, d_R, nx, dx, dy);

            it++;
      }

      /* Kernel for updating actual velocites U & V (along with boundary setting for U&V) */
      calculateUV <<<grid, threads>>> (d_U, d_V, d_F, d_G, d_P, nx, dx, dy, dt);
}
```

• 2D CUDA grid, block size : 32x32, threads : 32x8

• Each thread handles 4 elements – *Vector computing model.*

• Each block loads 34x34 elements (incl. apron values/ghost layers)

• Specific threads designated with extra work, access to apron values uncoalesced

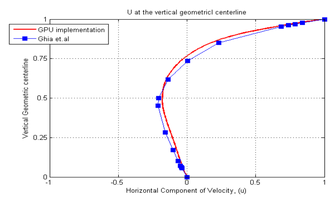
• Coalescence maintained for 90% of data accesses

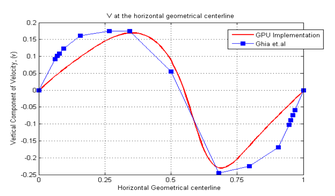
• Kernels use shared memory, user-managed cache

• Every kernel performs a stencil-type computation

• Stencil kernel – generic implementation, templatized and can incorporate standard stencils as *functor objects*,

• Part of in-house GPU data rearrangement library - optimized GPU kernels for basic data rearrangement operations





• Validation of simulation results at the geometric centre of the cavity