Numerical Simulation of Two-phase Flows on a GPU

F. Pereira and A. Rahunanthan

Department of Mathematics and School of Energy Resources, University of Wyoming, Laramie, WY 82071

Abstract. We consider a model for two-phase (water and oil), immiscible and incompressible displacement in heterogeneous porous media. In such a model, high-resolution, non-oscillatory schemes are efficient numerical schemes for solving hyperbolic conservation laws. In this paper we describe a GPU parallelization of such central schemes for solving hyperbolic conservation laws arising in the simulation of two-phase flows in three space dimensions.

1 Introduction

In this paper we discuss an implementation of central schemes for the approximation of the hyperbolic conservation law arising in two-phase, three-dimensional flows in heterogeneous porous media on a relatively inexpensive GPU which supports double precision calculations.

A Graphics Processor Unit (GPU) that is traditionally designed for graphics rendering evolved into highly parallel, multi-threaded, many core processor with tremendous computational horsepower and very high memory bandwidth. This influenced computing to evolve from "central processing" on the CPU to "co-processing" on the CPU and GPU. In November 2006, to enable this new computing paradigm, NVIDIA introduced the Compute Unified Device Architecture (CUDA). Since then, high performance parallel computing with CUDA has been attracting various researchers in several disciplines, such as computational fluid dynamics [6, 25, 17, 24], molecular dynamics [3, 15, 26], computational biology [21], linear algebra[4, 7], weather forecasting [16] and artificial intelligence [5], where most of them used GPUs which support only single precision calculations.

Traditional simulations of multiphase flows rely on ad-hoc upscaling techniques along with coarse grid simulations of the up-scaled models. The models for multiphase flows are defined at the lab scale (a few centimeters) while simulations of interest to important problems, such as the migration, trapping and possible leakage of CO_2 plumes in the subsurface, enhanced oil recovery, production of gas from unconventional resources, have to be performed in the field scale (a few kilometers). Such ad-hoc techniques, frequently developed and tested for some flow regimes for single or two-phase problems, may produce serious errors when applied to more complex compositional flows. Although upscaled solutions may be important in some areas, such as in identifying trends of flow patterns in oil reservoir simulation, there are important practical problems where the fine scale details of the numerical simulations should be numerically captured. We mention, for example, the simulation of leakage of injected CO_2 from brine aquifers: the fine scale preferential paths for flow that may lead to leakage cannot be captured by coarse grid, upscaled simulations. The new CPU/GPU system can be effectively used for these problems.

Although implicit schemes for transport have been investigated for many years for application in porous media flows, where predictions are needed for long periods of time, the new CPU/GPU system makes explicit schemes a very competitive option for the numerical solution of these problems. The explicit schemes are naturally parallelizable in these systems: a geometrical domain decomposition divides the original domain into subdomains such that the hyperbolic problems can be solved in parallel using the GPUs.

Here, we consider a model for two-phase, incompressible, immiscible displacement in heterogeneous porous media. In such a model, the highly nonlinear equations are of very practical importance [8]. The conventional theoretical description of two-phase flows in a porous medium, in the limit of vanishing capillary pressure, is via Darcy's law coupled to the Buckley-Leverett equation. We refer the two phases, water and oil, by the subscripts w and o, respectively. We also assume that the two fluid phases saturate the pores. With no sources or sinks, and neglecting the effects of gravity, these equations become

$$\nabla \cdot \boldsymbol{v}_s = 0, \quad \text{where} \;\; \boldsymbol{v}_s = -\lambda(s) K(\boldsymbol{x}) \nabla p, \tag{1}$$

and

$$\frac{\partial s}{\partial t} + \nabla \cdot (f(s)\boldsymbol{v}) = 0. \tag{2}$$

Here, $\boldsymbol{v}_s$ is the total seepage velocity, and $\boldsymbol{v} = \boldsymbol{v}_s/\phi$, where ϕ is the porosity which is assumed to be a constant. Furthermore, s is the water saturation, $K(x)$ is the absolute permeability which we assume to be a scalar, and p is the pressure. The functions, $\lambda(s)$ and $f(s)$ represent the total mobility and the fractional volumetric flow of water, respectively.

The equations, (1) and (2), which are coupled together by the seepage velocity $\boldsymbol{v}$ are solved by using an operator splitting technique. For efficiency, the time steps for solving the velocity-pressure equation (1), can be much longer than the time steps for solving the saturation equation (2). Here we focus on the efficient solution of the scalar hyperbolic conservation law (2) using a GPU.

In [24, 17, 23], the three-dimensional domain is represented by several two-dimensional slices. This two-dimensional mapping translates to efficient data transfer between the host (CPU) and the device (GPU). In this paper, the three-dimensional domain is represented by a one-dimensional array. Although, we solve a coupled hyperbolic equation using central schemes which are extensions of Kurganov-Tadmor [14] central scheme, the operator splitting that we employ enables us to solve the hyperbolic equation continuously for several time steps. We thus reduce the data transfer between CPU and GPU drastically.

This paper is structured as follows. In section 2, we discuss numerical approximation of two-phase flows using an operator splitting technique and discuss the central schemes for two-phase flows in heterogeneous porous media. In section 3, we discuss the implementation of central schemes on a GPU. In section 4, we apply the central schemes for a rectangular parallelepiped, heterogeneous reservoir and present the numerical results. Section 5 contains the conclusion.

2 Numerical approximation of two-phase flows

2.1 Operator splitting for two-phase flow

We employ an operator splitting technique to compute the solutions of the saturation and velocity-pressure equations, (1) and (2), which are coupled together by the seepage velocity $\boldsymbol{v}$. This technique is computationally efficient in producing accurate numerical solutions for two-phase flows [10].

Typically, for computational efficiency, larger time steps are used to solve the pressure equation (1). The splitting technique allows time steps used in the velocity-pressure computation that are longer than the steps allowed under an appropriate CFL condition in the saturation computation. We thus introduce a variable time step, Δt_s, for the saturation computation and a longer time step, Δt_p, for the velocity-pressure computation. The pressure and thus the seepage velocity are approximated at times $t^m = m\Delta t_p$, where $m = 0, 1, \ldots$; the saturation is approximated at times $t_m^k = t^m + k\Delta t_s$ for $t^m \leq t_m^k \leq t^{m+1}$. We remark that we must specify the water saturation at $t = 0$. For further details, we refer the reader to [10, 1, 2].

For the velocity-pressure computation, we use a hybridized mixed finite element discretization equivalent to cell-centered finite differences [10], which effectively treats the rapidly changing permeabilities that arise from stochastic geology and produces accurate velocity fields. The resulting algebraic system can be solved by a preconditioned conjugate gradient procedure or by a multi-grid technique.

2.2 Second-order semi-discrete central schemes for the saturation equation

In this section we discuss high-resolution central schemes which are extensions of the Kurganov-Tadmor central scheme [14], for solving the scalar hyperbolic conservation law (2).

The scalar hyperbolic conservation law (2) can be written as

$$\frac{\partial s}{\partial t} + \frac{\partial}{\partial x}({}^{x}v f(s)) + \frac{\partial}{\partial y}({}^{y}v f(s)) + \frac{\partial}{\partial z}({}^{z}v f(s)) = 0, \tag{3}$$

where ${}^{x}v = {}^{x}v(x, y, z)$, ${}^{y}v = {}^{y}v(x, y, z)$, and ${}^{z}v = {}^{z}v(x, y, z)$ denote the x, y and z components of the seepage velocity $\boldsymbol{v}$, respectively.

Define $x_i := i\Delta x$, $y_j := j\Delta y$, $z_k := k\Delta z$, $x_{i\pm\frac{1}{2}} := x_i \pm \frac{\Delta x}{2}$, $y_{j\pm\frac{1}{2}} := y_j \pm \frac{\Delta y}{2}$, and $x_{k\pm\frac{1}{2}} := z_k \pm \frac{\Delta z}{2}$. Assume that we have already computed an approximation to the solution at time level $t = t^n$ of the form,

$$\tilde{s}(x, t^n) := \sum_{i,j,k} p_{i,j,k}^n(x, y, z)\chi_{i,j,k}(x, y, z), \tag{4}$$

where

$$p_{i,j,k}(x, y, z) = \bar{s}_{i,j,k} + (s_x)_{i,j,k}(x - x_i) + (s_y)_{i,j,k}(y - y_j) + (s_z)_{i,j,k}(z - z_k), \tag{5}$$

with slopes approximated by MinMod limiter, $\chi_{i,j,k}$ is the characteristic function of the corresponding region, and $\bar{s}_{i,j,k}$ is the cell average defined by

$$\bar{s}_{i,j,k} := \frac{1}{\Delta x \Delta y \Delta z} \int_{x_{i-\frac{1}{2}}}^{x_{i+\frac{1}{2}}} \int_{y_{j-\frac{1}{2}}}^{y_{j+\frac{1}{2}}} \int_{z_{k-\frac{1}{2}}}^{z_{k+\frac{1}{2}}} p^n_{i,j,k}(x,y,z)\,dx\,dy\,dz. \tag{6}$$

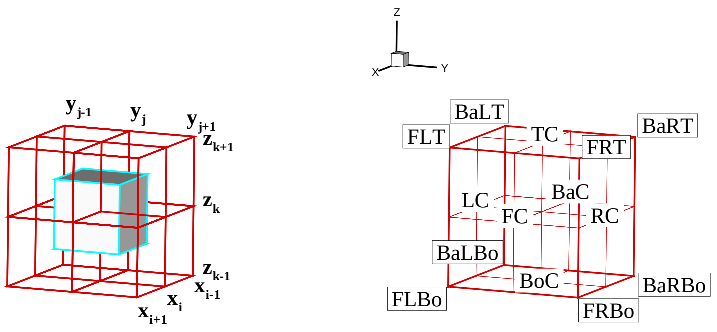

Fig. 1. *Left*: The non-staggered cell $[x_{i-\frac{1}{2}}, x_{i+\frac{1}{2}}] \times [y_{j-\frac{1}{2}}, y_{j+\frac{1}{2}}] \times [z_{k-\frac{1}{2}}, z_{k+\frac{1}{2}}]$, for central differencing in three-space dimensions. *Right*: The reconstructing points in a magnified view of the shaded cell that is on the left.

We denote the cell interface value at the faces

$$\begin{aligned} s^{FC}_{i,j,k} &:= p^n_{i,j,k}(x_{i+\frac{1}{2}}, y_j, z_k), & s^{BaC}_{i,j,k} &:= p^n_{i,j,k}(x_{i-\frac{1}{2}}, y_j, z_k), \\ s^{RC}_{i,j,k} &:= p^n_{i,j,k}(x_i, y_{j+\frac{1}{2}}, z_k), & s^{LC}_{i,j,k} &:= p^n_{i,j,k}(x_i, y_{j-\frac{1}{2}}, z_k), \\ s^{TC}_{i,j,k} &:= p^n_{i,j,k}(x_i, y_j, z_{k+\frac{1}{2}}), & s^{BoC}_{i,j,k} &:= p^n_{i,j,k}(x_i, y_j, z_{k-\frac{1}{2}}), \end{aligned} \tag{7}$$

where FC, BaC, RC, LC, TC and BoC denote Front, Back, Right, Left, Top and Bottom Centers respectively, and the cell interface value at the vertices

$$\begin{aligned} s^{FRT}_{i,j,k} &:= p^n_{i,j,k}(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}, z_{k+\frac{1}{2}}), & s^{BaRT}_{i,j,k} &:= p^n_{i,j,k}(x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}}, z_{k+\frac{1}{2}}), \\ s^{FRBo}_{i,j,k} &:= p^n_{i,j,k}(x_{i+\frac{1}{2}}, y_{j+\frac{1}{2}}, z_{k-\frac{1}{2}}), & s^{BaRBo}_{i,j,k} &:= p^n_{i,j,k}(x_{i-\frac{1}{2}}, y_{j+\frac{1}{2}}, z_{k-\frac{1}{2}}), \\ s^{FLT}_{i,j,k} &:= p^n_{i,j,k}(x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}}, z_{k+\frac{1}{2}}), & s^{BaLT}_{i,j,k} &:= p^n_{i,j,k}(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}}, z_{k+\frac{1}{2}}), \\ s^{FLBo}_{i,j,k} &:= p^n_{i,j,k}(x_{i+\frac{1}{2}}, y_{j-\frac{1}{2}}, z_{k-\frac{1}{2}}), & s^{BaLBo}_{i,j,k} &:= p^n_{i,j,k}(x_{i-\frac{1}{2}}, y_{j-\frac{1}{2}}, z_{k-\frac{1}{2}}), \end{aligned} \tag{8}$$

where the letters F, Ba, R, L, T and Bo in the superscripts denote Front, Back, Right, Left, Top and Bottom, respectively (see Fig. 1). We compute the maximum local speeds of propagation of the discontinuities by

$$\begin{aligned} a^x_{i+\frac{1}{2},j,k} &:= \max\left\{\rho\left(\frac{\partial f}{\partial s}(s^{BaC}_{i+1,j,k})\right), \rho\left(\frac{\partial f}{\partial s}(s^{FC}_{i,j,k})\right)\right\}, \\ a^y_{i,j+\frac{1}{2},k} &:= \max\left\{\rho\left(\frac{\partial g}{\partial s}(s^{LC}_{i,j+1,k})\right), \rho\left(\frac{\partial g}{\partial s}(s^{RC}_{i,j,k})\right)\right\}, \\ a^z_{i,j,k+\frac{1}{2}} &:= \max\left\{\rho\left(\frac{\partial h}{\partial s}(s^{BoC}_{i,j,k+1})\right), \rho\left(\frac{\partial h}{\partial s}(s^{TC}_{i,j,k})\right)\right\}. \end{aligned} \tag{9}$$

Dimension-by-dimension approach: As in [13], the one-dimensional numerical flux is straightforwardly applied in all coordinate directions. Thus, we can now write the corresponding semi-discrete form as

$$\begin{aligned} \frac{d}{dt}\bar{u}_{i,j,k}(t) = &- \frac{H^x_{i+\frac{1}{2},j,k} - H^x_{i-\frac{1}{2},j,k}}{\Delta x} \\ &- \frac{H^y_{i,j+\frac{1}{2},k} - H^y_{i,j-\frac{1}{2},k}}{\Delta y} - \frac{H^z_{i,j,k+\frac{1}{2}} - H^z_{i,j,k-\frac{1}{2}}}{\Delta z}, \end{aligned} \tag{10}$$

with the numerical flux

$$\begin{aligned} H^x_{i+\frac{1}{2},j,k} = &\frac{1}{2}[{}^x v^{FC}_{i,j,k}\{f(s^{BaC}_{i+1,j,k}) + f(s^{FC}_{i,j,k})\}] \\ &- \frac{a^x_{i+\frac{1}{2},j,k}}{2}(s^{BaC}_{i+1,j,k} - s^{FC}_{i,j,k}). \end{aligned} \tag{11}$$

When solving for the saturation in time, the velocity ${}^x v^{FC}_{i,j,k}$ is given by the solution of the velocity-pressure equation. Recall that the solution of (1) is approximated by the lowest order Raviart-Thomas mixed finite element method [10]. Therefore, the velocity ${}^x v^{FC}_{i,j,k}$ in the semi-discrete scheme is obtained directly from the Raviart-Thomas space on the cell faces.

Genuinely multi-dimensional approach: A new second-order, semi-discrete central scheme for the approximation of two-phase flows in three space dimensions is presented in [20]. The corresponding semi-discrete scheme can be written as

$$\begin{aligned} \frac{d}{dt}\bar{u}_{i,j,k}(t) = &- \frac{H^x_{i+\frac{1}{2},j,k} - H^x_{i-\frac{1}{2},j,k}}{\Delta x} \\ &- \frac{H^y_{i,j+\frac{1}{2},k} - H^y_{i,j-\frac{1}{2},k}}{\Delta y} - \frac{H^z_{i,j,k+\frac{1}{2}} - H^z_{i,j,k-\frac{1}{2}}}{\Delta z}, \end{aligned} \tag{12}$$

with the numerical flux

$$\begin{aligned} H^x_{i+\frac{1}{2},j,k} = &\frac{1}{8}[{}^x v^{FRT}_{i,j,k}\{f(s^{BaRT}_{i+1,j,k}) + f(s^{FRT}_{i,j,k})\} + {}^x v^{FRBo}_{i,j,k}\{f(s^{BaRBo}_{i+1,j,k}) + f(s^{FRBo}_{i,j,k})\} \\ &+ {}^x v^{FLT}_{i,j,k}\{f(s^{BaLT}_{i+1,j,k}) + f(s^{FLT}_{i,j,k})\} + {}^x v^{FLBo}_{i,j,k}\{f(s^{BaLBo}_{i+1,j,k}) + f(s^{FLBo}_{i,j,k})\}] \\ &- \frac{a^x_{i+\frac{1}{2},j,k}}{2}(s^{BaC}_{i+1,j,k} - s^{FC}_{i,j,k}). \end{aligned} \tag{13}$$

Now, we need to have the seepage velocity at the vertices. However, the seepage velocity $\boldsymbol{v}$, which is defined in the lowest order Raviart-Thomas space, has the components $v_{FC}, v_{BaC}, v_{RC}, v_{LC}, v_{TC}$ and v_{BoC} on the faces of a rectangular parallelepiped (the components of $\boldsymbol{v}$ are pointing outward). Therefore, we use a bilinear interpolation, which preserves the null divergence necessary for the incompressible flows, to compute the seepage velocity at the vertices. For instance, to compute the seepage velocity ${}^{x}v_{i,j,k}^{FRT}$, we have to use the eight cells which share the vertex FRT of the cell, ijk. Thus, we define,

$$\begin{aligned} {}^{x}v_{i,j,k}^{FRT} = \frac{1}{16} \big[& (v_{i,j,k}^{FC} - v_{i,j,k}^{BaC}) + (v_{i+1,j,k}^{FC} - v_{i+1,j,k}^{BaC}) + (v_{i,j+1,k}^{FC} - v_{i,j+1,k}^{BaC}) \\ & + (v_{i,j,k+1}^{FC} - v_{i,j,k+1}^{BaC}) + (v_{i+1,j+1,k}^{FC} - v_{i+1,j+1,k}^{BaC}) + (v_{i+1,j,k+1}^{FC} - v_{i+1,j,k+1}^{BaC}) \\ & + (v_{i,j+1,k+1}^{FC} - v_{i,j+1,k+1}^{BaC}) + (v_{i+1,j+1,k+1}^{FC} - v_{i+1,j+1,k+1}^{BaC}) \big] . \end{aligned} \tag{14}$$

2.3 Time marching for the semi-discrete central schemes

The time integration adopted for solving equations (10) and (12) is the second-order, SSP Runge-Kutta scheme [22],

$$\begin{aligned} s^{(1)} &= s^n + \Delta t L(s^n), \\ s^{n+1} &= \frac{1}{2}s^n + \frac{1}{2}s^{(1)} + \frac{1}{2}\Delta t L(s^{(1)}), \end{aligned} \tag{15}$$

where the superscripts n and $n+1$ denote time level t and $t+1$, respectively and $L(s)$ denotes the right-hand side of the semi-discrete schemes (10) and (12).

3 GPU implementation of central schemes using CUDA

In this section, we discuss some strategies for GPU implementation of the central schemes for solving the saturation equation (2), using CUDA.

CUDA supports several types of memory that can be used by a programmer to achieve high execution speeds in his/her kernels. We here summarize the CUDA device memories that will be used in the following discussions. In order to execute CUDA kernel on a device, the programmer needs to allocate memory on the device and transfer pertinent data from the host memory to the allocated device memory. For this purpose, the programmer can use global memory or constant memory. The global memory is accessible from either the host or device and has the lifetime of the application. The constant memory supports short-latency, high-bandwidth, read only access by the device when all threads simultaneously access the same location. Registers and shared memory are on-chip memories. Variables that reside in these types of memory can be accessed at very high speed in a highly parallel manner. Local memory which resides in the global memory is only accessible by the thread and has the lifetime of the thread [12].

Let n_x, n_y and n_z be the number of computational elements in the x, y and z directions for a three-dimensional flow domain, respectively. The three-dimensional domain $n_x \times n_y \times n_z$ is represented by a one-dimensional array of size $n_x n_y n_z$. The element (i, j, k) in three-dimensional domain is denoted by $n_x n_y \cdot k + n_x \cdot j + i$ in the one-dimensional array. For any element in the one-dimensional array, we can always find the corresponding i, j, k of the element. Identifying i, j, k of an element in the one-dimensional array requires two integer divisions and a modulo operation. Though, these operations are costly, they can be replaced with relatively inexpensive bitwise operations if the denominator is a power of two [19].

Now, we solve the saturation equation (2) using the central schemes, where the seepage velocity is obtained by solving the pressure-velocity equation (1). The following algorithm outlines the steps in solving the saturation equation using the time marching scheme (15).

```
Reconstruct cell interface values using $s^n$
Compute $L(s^n)$ using computed cell interface values
Launch CUDA kernel to compute $s^{(1)}$

Reconstruct cell interface values using $s^{(1)}$
Compute $L(s^{(1)})$ using computed cell interface values
Launch CUDA kernel to compute $s^{n+1}$
```

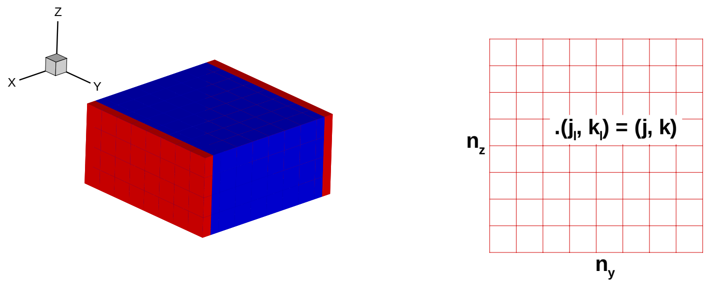

Fig. 2. *Left*: Side 1 and Side 2 in the three-dimensional domain. *Right*: Side 1 mapping into the three-dimensional domain

Since we have different boundary conditions to be imposed on sides, we have to treat each side separately. This helps to reduce the branches (if loops) and thus improve the CUDA performance. Therefore, we define the following computational domains.

- Side 1: $i = 0$, $j = 0..n_y - 1$ and $k = 0..n_z - 1$
- Side 2: $i = n_x - 1$, $j = 0..n_y - 1$ and $k = 0..n_z - 1$
- Side 3: $i = 1..n_x - 2$, $j = 0$ and $k = 0..n_z - 1$
- Side 4: $i = 1..n_x - 2$, $j = n_y - 1$ and $k = 0..n_z - 1$
- Side 5: $i = 1..n_x - 2$, $j = 1..n_y - 2$ and $k = 0$
- Side 5: $i = 1..n_x - 2$, $j = 1..n_y - 2$ and $k = n_z - 1$
- Interior: $i = 1..n_x - 2$, $j = 1..n_y - 2$ and $k = 1..n_z - 2$

If we consider the dimension-by-dimension approach, each cell has six cell interface values to be stored. In order to optimize global memory access [18], we use structure of arrays (SOA) for storing these cell interface values. For the dimension-by-dimension approach, we define arrays to hold the cell interface values FC, BaC, RC, LC, TC and BoC, and the pointers to these arrays are kept in a structure. Since CUDA kernels do not support very long argument lists, references to these arrays are copied to constant memory. It also helps to save some shared memory [18]. When we launch the kernel for reconstructing cell interface values, we launch separately for each computational domain. However, in order to launch separately for each computational domain, we have to keep track of elements in each computational domain. One approach to do this is to keep the elements of each computational domain in an array in the global device memory. However, in this approach we increase unnecessary device memory fetching and decrease the global device memory available for storing cell interface values and values of $L(u)$. Another approach is to identify elements of each computational domain at runtime. Since we work on a structured, three-dimensional domain, we can easily identify i, j, k of any element. For side 1 (see Fig. 2), we can find the global index (i, j, k) from the local index as shown below:

$$\begin{aligned} \text{local index} &= n_y k_l + j_l, \\ i = 0, \;\; j &= j_l, \;\; k = k_l, \\ \text{global index} &= n_x n_y k + n_x j, \end{aligned} \tag{16}$$

In CUDA kernel, local index is equal to $blockIdx.x * blockDim.x + threadIdx.x$. Similarly, we can find the global indexes in each computational domain.

The right-hand side (RHS) of the semi-discrete scheme can be written as

$$\begin{aligned} L(s) = &- \frac{H^x_{i+\frac{1}{2},j,k}}{\Delta x} + \frac{H^x_{i-\frac{1}{2},j,k}}{\Delta x} \\ &- \frac{H^y_{i,j+\frac{1}{2},k}}{\Delta y} + \frac{H^y_{i,j-\frac{1}{2},k}}{\Delta y} - \frac{H^z_{i,j,k+\frac{1}{2}}}{\Delta z} + \frac{H^z_{i,j,k-\frac{1}{2}}}{\Delta z}. \end{aligned} \tag{17}$$

To compute the $L(s)$, we consider each computational domain separately as we discussed earlier. For each computational domain, we can compute $L(s)$ by considering $H^x_{i+\frac{1}{2},j,k}$, $H^x_{i-\frac{1}{2},j,k}$, $H^y_{i,j+\frac{1}{2},k}$, $H^y_{i,j-\frac{1}{2},k}$, $H^z_{i,j,k+\frac{1}{2}}$ and $H^z_{i,j,k-\frac{1}{2}}$ in a single kernel. The new kernel now uses more registers and shared memory; even some of the variables are spilled into local memory. Therefore, we are in need to break this kernel into smaller kernels. To increase the CUDA performance further, we split this computation into six CUDA kernels [12]. That is, the first CUDA kernel updates $L(s)$ with $-\frac{H^x_{i+\frac{1}{2},j,k}}{\Delta x}$ and so on. This will restrict the use of shared memory. Anyhow, the shared memory is very limited to store double precision interface values.

3.1 Shared memory implementation

We can use shared memory implementation for reconstructing cell interface values. However, we cannot use it for computing $L(s)$ as we update $L(s)$ of a computational domain using separate six kernels.

The shared memory implementation will be effective only for the elements in the interior region. Since we launch kernels based on computational domain, we can easily modify the kernel for the elements in the interior region to accommodate shared memory implementation. We consider a cube $n_c \times n_c \times n_c$, where n_c is the number of elements in each side of the cube. Here, to compute cell interface values of $(n_c - 2) \times (n_c - 2) \times (n_c - 2)$ elements, the block threads copy the values of s for the elements in the cube from the global memory to the shared memory. Then, the cell interface values are computed using data from the shared memory. Finally the cell interface values are written back to the global memory.

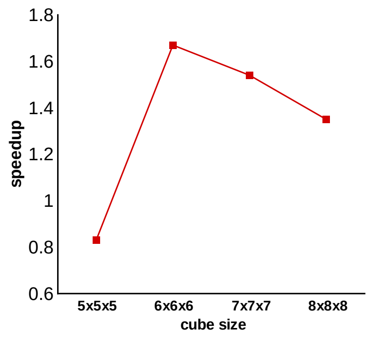

Fig. 3. Kernel speedup when using shared memory over global memory for the computational domain 62x62x62

Fig. 3 shows the performance of the shared memory implementation against n_c. We do not see any drastic improvement in the performance as we increase the cube size. Also, the shared memory available is only 16KB for a multiprocessor and it is often a very limiting factor for double precision computations.

Since we cannot effectively implement shared memory for computing $L(s)$ and arithmetic intensity for computing $L(s)$ is very much higher than computing cell interface values, the overall performance is not improved by the use of shared memory in GPU implementation for the central schemes.

4 Numerical experiments

In this section, we present the results of numerical simulation of three-dimensional, two-phase flows associated with a flooding problem. For the saturation calculation, we use the second-order, semi-discrete central schemes (10) and (12).

In all simulations, the reservoir contains initially 79% oil and 21% water. Water is injected uniformly into the reservoir at a constant rate of one pore-volume every five years. For the heterogeneous reservoir studies, we consider a scalar, heterogeneous absolute permeability field taken to be the logarithmic of a realization of a Gaussian random fractal field [11, 9]. The spatially variable permeability field is defined on a $32 \times 32 \times 4$ grid with the coefficient of variation (C_v) 0.5, where the C_v is defined as $(standard\ deviation/mean)$. The fractional volumetric flow and the total mobility are given by

$$f(s) = \frac{k_{rw}(s)/\mu_w}{\lambda(s)}, \;\; \lambda(s) = \frac{k_{rw}}{\mu_w} + \frac{k_{ro}}{\mu_o}, \tag{18}$$

where

$$k_{ro}(s) = \left\{ 1 - \frac{s}{(1 - s_{ro})} \right\}^2, \;\; k_{rw}(s) = \left\{ \frac{s - s_{rw}}{1 - s_{rw}} \right\}^2. \tag{19}$$

Furthermore, we use the following data in all flow simulations.

Viscosity	$\mu_w = 0.5$ cP $\mu_o = 10$ cP
Porosity	$\phi = 0.2$
Residual saturations	$s_{ro} = 0.15 \quad s_{rw} = 0.2$

We consider a three-dimensional flow in a rectangular parallelepiped, heterogeneous reservoir of size $64m \times 64m \times 8m$. The boundary conditions, and the injection and production specifications for the two-phase flow equations, (1) and (2), are given below. The injection is made along the face, $\{(x, y, z)|x = 0, 0 \leq y \leq 64m$ and $0 \leq z \leq 8m\}$ of the reservoir, and the production is taken along the face, $\{(x, y, z)|x = 64m,\ 0 \leq y \leq 64m$ and $0 \leq z \leq 8m\}$ of the reservoir; no flow is allowed along the remaining faces. The time step Δt_p is kept at 10 days and Δt_s is computed using CFL condition.

Fig. 4 and Fig. 5 show the mesh refinement study of the problem. Clearly, as the mesh is refined, a better resolution of the fingering instabilities is observed.

For comparing speedup of GPU, we used NVIDIA GeForce 295 GPU with Intel Xeon (E5405) 2.00GHz and Intel Core i7 (965) 3.20GHz processors. GTX 295 graphics card (compute capability 1.3) has 240 CUDA cores and 0.9GB global device memory per GPU. CPU version and GPU version of the codes adopt strategies that best suited for their optimal performance. Fig. 6 shows speedup curves of GTX 295 GPU with Intel processors.

5 Conclusion and future work

We have presented an implementation of the central schemes on a GPU for solving the hyperbolic equation arising in two-phase flows. By using a single GPU we get a factor of 50-65 speedup compared to an Intel Xeon processor. This GPU implementation can be very effective in solving medium-scale, three-dimensional problems in flow through porous media applications.

The explicit schemes are naturally parallelizable in multi-GPUs: a geometrical domain decomposition may be applied to divide the original domain into subdomains such that the hyperbolic problems can be solved in parallel using multi-GPUs. The authors are investigating adequate strategies for the communication between sub domains (needed because of the explicit nature of the numerical schemes), including the possibility of using an overlapping domain decomposition procedure.

The GPU code based on CUDA is currently being adapted to OpenCL. The implementation in OpenCL and its performance study will appear elsewhere.

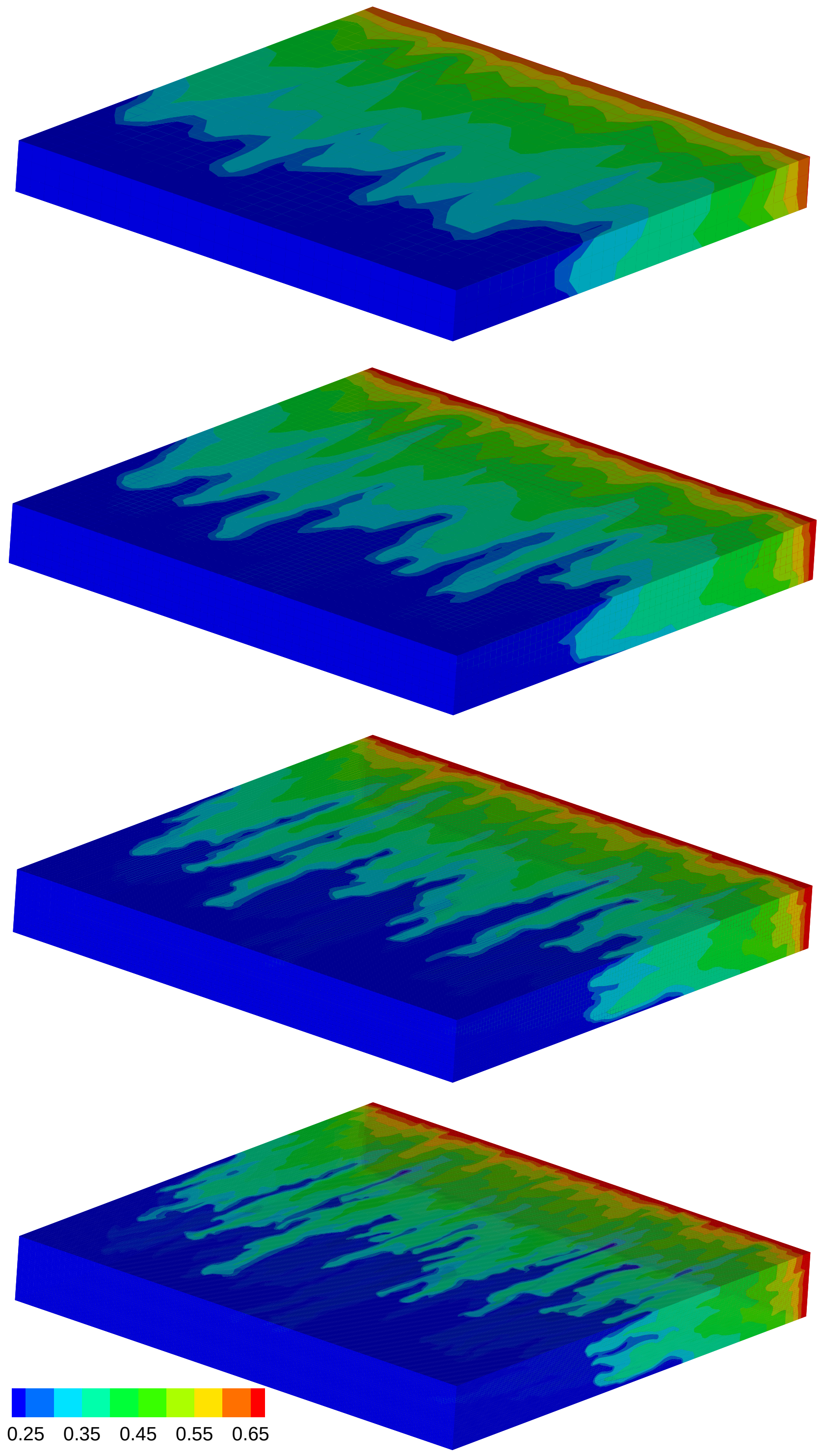

Fig. 4. Dimension-by-dimension approach: Water saturation plots for two-phase flow in a three-dimensional heterogeneous reservoir of dimension $64m \times 64m \times 8m$. *Top to bottom*: $32 \times 32 \times 4$ grid, $64 \times 64 \times 8$ grid, $128 \times 128 \times 16$ grid and $256 \times 256 \times 32$ grid, respectively.

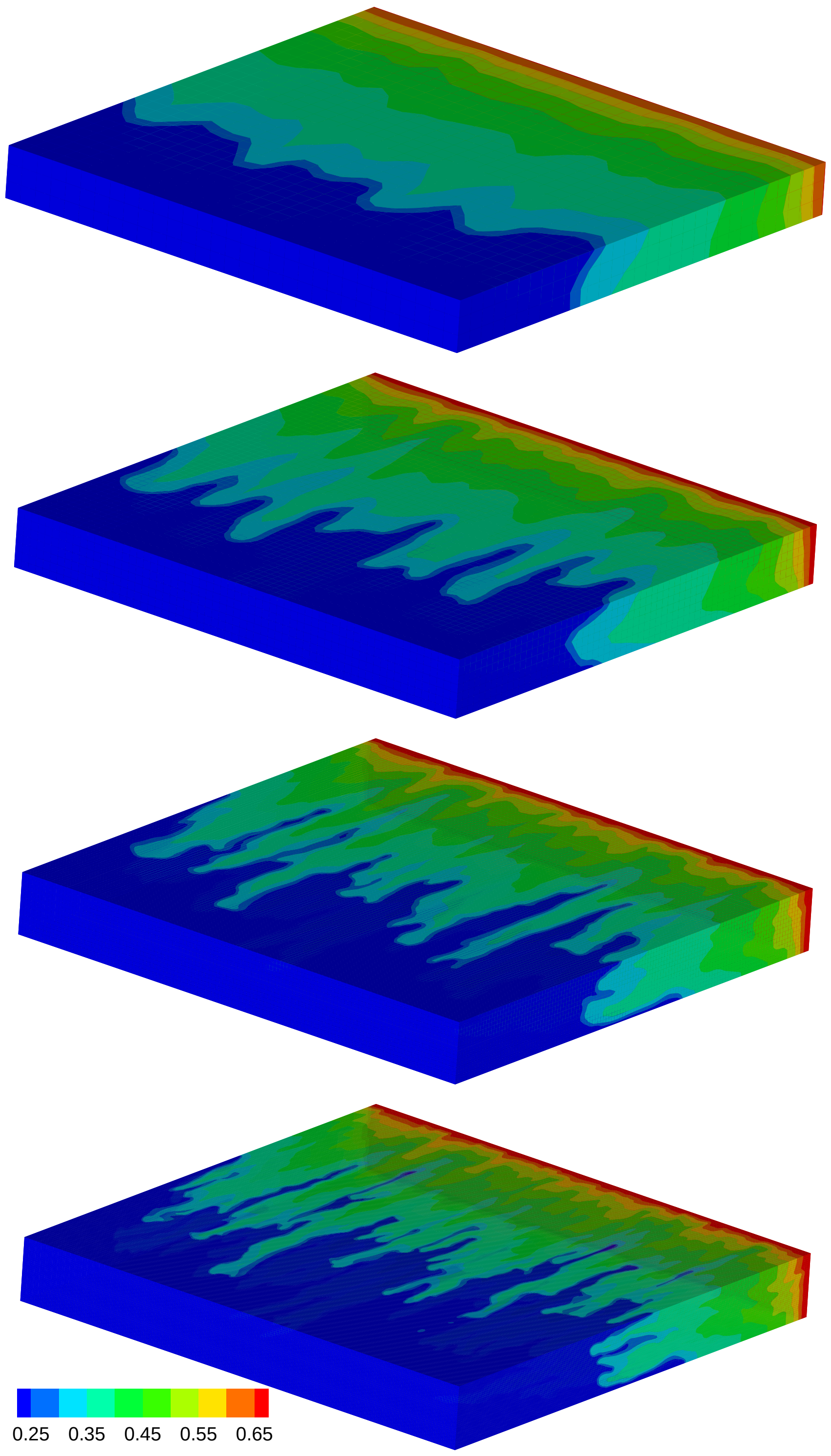

Fig. 5. Genuinely multi-dimensional approach: Water saturation plots for two-phase flow in a three-dimensional heterogeneous reservoir of dimension $64m \times 64m \times 8m$. *Top to bottom*: $32 \times 32 \times 4$ grid, $64 \times 64 \times 8$ grid, $128 \times 128 \times 16$ grid and $256 \times 256 \times 32$ grid, respectively.

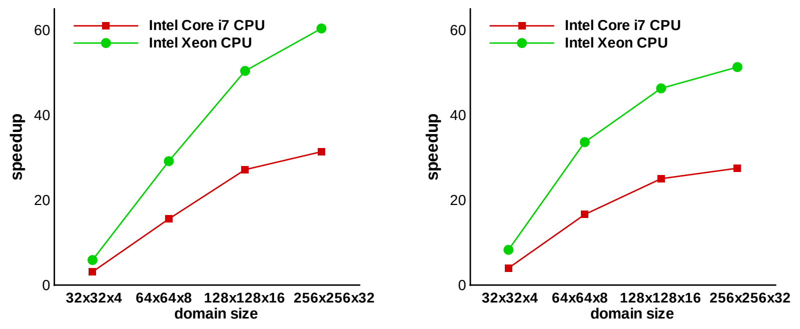

Fig. 6. Single GPU speedup relative to a single CPU core. *Left*: Dimension-by-dimension approach. *Right*: Genuinely multi-dimensional approach

Acknowledgments: We thank Manfred Liebmann of the Institute for Mathematics and Scientific Computing, University of Graz for allowing us to use his preconditioned conjugate gradient code with algebraic multigrid preconditioner (PCG-AMG) for solving the elliptic problem.

References

1. Abreu, E., Douglas Jr., J., Furtado, F., Pereira, F.: Operator splitting based on physics for flow in porous media. Int. J. of Computational Science **2(3)**, 315–335 (2008)
2. Abreu, E., Douglas Jr., J., Furtado, F., Pereira, F.: Operator splitting for three-phase flow in heterogeneous porous media. Communications in Computational Physics **6(1)**, 72–84 (2009)
3. Anderson, J., Lorenz, C., Travesset, A.: General purpose molecular dynamics simulations fully implemented on Graphics Processing Units. Journal of Computational Physics **227(10)**, 5342–5359 (2008)
4. Barrachina, S., Castillo, M., .Igual, F., Mayo, R., Quintana-Orti, E.: Solving dense linear systems on graphics processors. Tech. Rep. ICC 02-02-2008, Universidad Jaume I, Depto. de Ingenieria y Ciencia de Computadores (2008)
5. Bleiweiss, A.: GPU accelerated pathfinding. In: Proceedings of the 23rd ACM SIGGRAPH/Eurographics Symposium on Graphics Hardware, pp. 65–74. Aire-la-Ville, Switzerland (2008)
6. Brandvik, T., Pullan, G.: Acceleration of a 3D Euler solver using commodity graphics hardware. In: 46th AIAA Aerospace Sciences Meeting and Exhibit (2008)
7. Castillo, M., Chan, E., Igual, F., R. Mayo, E.Q.O., Quintana-Orti, G., van de Geijn, R., Zee, F.V.: Making programming synonymous with programming for linear algebra libraries. Tech. rep., University of Texas at Austin, Department of Computer Science (2008)
8. Chen, Z., Huan, G., Ma, Y.: Computational methods for multiphase flows in porous media. SIAM, Philadelphia, PA (2006)
9. Furtado, F., Pereira, F.: Crossover from nonlinearity controlled to heterogeneity controlled mixing in two-phase porous media fows. Comput. Geosciences **7(2)**, 115–135 (2003)
10. Furtado, F., Pereira, F., Douglas Jr., J.: On the numerical simulation of waterflooding of heterogeneous petroleum reservoirs. Comp. Geosciences **1**, 155–190 (1997)

11. Glimm, J., Lindquist, B., Pereira, F., Zhang, Q.: A theory of macrodispersion for the scale up problem. Transport in Porous Media **13**, 97–122 (1993)
12. Kirk, D.B., mei W. Hwu, W.: Programming Massively Parallel Processors: A Hands-on Approach. Morgan Kaufmann Publishers, Burlington, MA (2010)
13. Kurganov, A., Petrova, G.: A third-order semi-discrete genuinely multidimensional central scheme for hyperbolic conservation laws and related problems. Numeriche Mathematik **88(4)**, 683–729 (2001)
14. Kurganov, A., Tadmor, E.: New high-resolution central schemes for nonlinear conservation laws and convection-diffusion equations. J. of Computational Physics **160**, 241–282 (2000)
15. Liu, W., Schmidt, B., Voss, G., Muller-Wittig, W.: Molecular dynamics simulations on commodity GPUs with CUDA (2007)
16. Michalakes, J., Vachharajani, M.: GPU acceleration of numerical weather prediction. In: Proceedings of the IEEE International Symposium on Parallel and Distributed Processing. Washington, DC (2008)
17. Micikevicius, P.: 3D fnite difference computation on GPUs using CUDA. In: GPGPU-2: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units
18. NVIDIA: CUDA C Programming Best Practices Guide. NVIDIA Corp., Santa Clara, CA (2009)
19. NVIDIA: CUDA programming guide 2.3. NVIDIA Corp., Santa Clara, CA (2009)
20. Pereira, F., Rahunanthan, A.: A semi-discrete central scheme for the approximation of two-phase flows in three space dimensions. Submitted
21. Schatz, M., Trapnell, C., Delcher, A., Varshney, A.: High-throughput sequence alignment using graphics processing units. BMC Bioinformatics **8(474)** (2007)
22. Shu, C..W., Osher, S.: Efficient implementation of essentially nonoscillatory shock-capturing schemes II. Journal of Computational Phys. **83(1)**, 32–78 (1989)
23. Stone, J., Phillips, J., Freddolino, P., Hardy, D., Trabuco, L., Schulten, K.: Accelerating molecular modeling applications with graphics processors. J. Comp. Chem. **28**, 2618–2640 (2007)
24. Thibault, J.C., Senocak, I.: CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows. In: 47th AIAA Aerospace Sciences Meeting. Orlanda, FL (2009). Paper No:AIAA-2009-758
25. Tolke, J., Krafczyk, M.: TeraFLOP computing on a desktop PC with GPUs for 3D CFD. International Journal of Computational Fluid Dynamics **22(7)**, 443–456 (2008)
26. Ufimtsev, I., Martinez, T.: Quantum chemistry on graphical processing units. 1. strategies for two-electron integral evaluation. Journal of Chemical Theory and Computation **4(2)**, 222–231 (2008)