International Journal of Modern Physics C, Vol. 9, No. 8 (1998) 1–13
© World Scientific Publishing Company

ACCURACY AND COMPUTATIONAL EFFICIENCY IN 3D DISPERSION VIA LATTICE-BOLTZMANN: MODELS FOR DISPERSION IN ROUGH FRACTURES AND DOUBLE-DIFFUSIVE FINGERING*

HARLAN W. STOCKMAN and ROBERT J. GLASS
Sandia National Laboratories, Albuquerque
NM 87185-0750, USA

CLAY COOPER
Desert Research Institute
P. O. Box 60220, Reno, NV 89506-0220, USA

HARIHAR RAJARAM
Civil Engineering Department, University of Colorado
Campus Box 428, Boulder, CO 80309, USA

Accepted 5 October 1998

In the presence of buoyancy, multiple diffusion coefficients, and porous media, the dispersion of solutes can be remarkably complex. The lattice-Boltzmann (LB) method is ideal for modeling dispersion in flow through complex geometries; yet, LB models of solute fingers or slugs can suffer from peculiar numerical conditions (e.g., denormal generation) that degrade computational performance by factors of 6 or more. Simple code optimizations recover performance and yield simulation rates up to ~3 million site updates per second on inexpensive, single-CPU systems. Two examples illustrate limits of the methods: (1) Dispersion of solute in a thin duct is often approximated with dispersion between infinite parallel plates. However, Doshi, Daiya and Gill (DDG) showed that for a smooth-walled duct, this approximation is in error by a factor of ~8. But in the presence of wall roughness (found in all real fractures), the DDG phenomenon can be diminished. (2) Double-diffusive convection drives "salt-fingering", a process for mixing of fresh-cold and warm-salty waters in many coastal regions. Fingering experiments are typically performed in Hele-Shaw cells, and can be modeled with the 2D (pseudo-3D) LB method with velocity-proportional drag forces. However, the 2D models cannot capture Taylor–Aris dispersion from the cell walls. We compare 2D and true 3D fingering models against observations from laboratory experiments.

Keywords:

*This paper was presented at the 7th Int. Conf. on the Discrete Simulation of Fluids held at the University of Oxford, 14–18 July 1998.

1. Introduction

In many proposed repositories for nuclear or toxic waste, much of the porosity consists of thin, sub-planar fractures. The apertures, or gaps between the rock faces of the fracture, range from $\sim 10\ \mu$m to cm in width. Under natural pressure gradients in the earth's crust, dispersion through such fractures is characterized by Péclet numbers (Pe) ≤ 1. However, flow rates can be accelerated greatly by pumping, or by heating from radioactive waste, yielding Pe (defined relative to fracture aperture) of ~ 10 to ~ 100.[1,2]

There is great interest in measuring the sorption of radionuclides in flowing fractures, and careful laboratory tests have been performed.[3–5] For practical reasons, most lab experiments are run at higher Pe than field conditions, with relatively short flow paths. In rough-walled fractures, it is very difficult to distinguish true sorption from the enhanced dispersion that comes with stagnant zones, which suggests that one should use more regular geometries for lab tests, such as prismatic channels machined into solid rock samples.[3] Such regular geometries may lead to a peculiar dispersion effect not commonly recognized in the geologic and hydrologic literature. We will refer to this anomalous dispersion as the DDG effect, after Doshi, Daiya and Gill, who produced the first detailed description.

Subtle buoyancy effects may also enhance dispersion in vertical fractures. Double-diffusive fingering[7,8] can lead to rapid mixing, even in systems that appear to be gravitationally stable. Failure to understand such effects at the microscopic scale can result in gross errors in the extrapolation of lab tests to the kilometer-scale of transport in fractured rocks.

In this paper, we discuss the application of LB methods to the study of small scale dispersion in fractures, focusing on the DDG effect and double-diffusive fingering. Our intent is to show that with careful optimization and choice of boundary conditions, LB methods are sufficiently accurate and efficient to solve problems relevant to real-world hydrology and geochemistry.

2. Optimization and Performance

The LB method, and the BGK simplification, are well-described by Qian *et al.*[9] The evolution of the particle distribution function f_i satisfies

$$f_i(\boldsymbol{x} + \boldsymbol{e}_i, t+1) = (f_i(\boldsymbol{x}, t) - f_i^{\text{eq}}(\boldsymbol{x}, t))/\tau\,, \tag{1}$$

where $\boldsymbol{x}$ is a node position in the lattice, $\boldsymbol{e}_i$ is one of i allowed velocities (e.g., $i = 0, 1, \ldots, 18$ in D3Q19 model[9]), t is the time-step, f_i^{eq} is the equilibrium distribution, and τ is the BGK relaxation parameter, which determines the viscosity and diffusion coefficients. In typical LB computer programs, a translation or streaming function performs the left side Eq. (1), and a collision function performs the right side.

For the dispersion of dilute tracers, the standard BGK method can be simplified, and programs can be substantially optimized. A single lattice can be used for a carrier fluid, which determines a velocity field $\boldsymbol{u}$ obeying the Navier–Stokes equations;[10] typically we use the 19-vector D3Q19 lattice with the 2nd-order equilibrium distribution given by Martys and Chen[11]:

$$f_0^{\text{eq}} = \frac{\rho}{3}\left[1 - \frac{3}{2}u^2\right] \tag{2}$$

$$f_i^{\text{eq}} = t_i\rho\left[1 + 3\boldsymbol{e}_i\cdot\boldsymbol{u} + \frac{3}{2}(3\boldsymbol{e}_i\boldsymbol{e}_i:\boldsymbol{uu} - u^2)\right] \qquad \text{for i} \neq 0\,, \tag{3}$$

where $\rho = \sum_i f_i$, and $t_i = 1/18$ for the 6 vectors along the Cartesian axes, and $t_i = 1/36$ for the remaining vectors. The $\boldsymbol{u}$ determined from the carrier fluid is then used for each tracer. Flekkøy *et al.*[12] and Noble[13] noted the advection–dispersion equation is linear in $\boldsymbol{u}$, so the tracer equilibrium distributions need only be linear in $\boldsymbol{u}$ as well. For a tracer s, the equilibrium distribution is of the form:

$$f_{s,i}^{\text{eq}} = A + B(\boldsymbol{e}_{s,i}\cdot\boldsymbol{u})\,, \tag{4}$$

where A and B are fixed by the requirement that solute be conserved ($\rho_s = \sum f_{s,i}^{\text{eq}}$), and the requirement that the solute flux at equilibrium is due entirely to advection ($\sum f_{s,i}^{\text{eq}}\boldsymbol{e}_{s,i} = \rho_s\cdot\boldsymbol{u}$) Compared to Eq. (3), Eq. (4) requires far fewer floating point operations.

Further savings are realized by using lattices with fewer vectors for the tracers; in 2D, four Cartesian vectors are adequate, and in 3D, the six Cartesian vectors will suffice. Wolf-Gladrow[14] and others have suggested low-vector lattices for modeling diffusion, but Noble[13] appears to have first performed a Chapman–Enskog expansion for Cartesian lattices, recovering the advection–dispersion equation and estimating the error term. We compared both the 6-vector 3D lattice and the full 19-vector D3Q19 lattice for tracers dispersion in nontrivial geometries, for Pe $\sim$ 70 and Re $\sim$ 80, and found the dispersion coefficients from 6- and 19-vector methods agreed to within 0.2%. (We emphasize that low-vector lattices, and Eq. (4), are used *only* for the dilute tracers, not the carrier.)

In many dispersion problems, $\boldsymbol{u}$ is at steady state, and need not be recalculated at each time-step. Furthermore, periodic boundary conditions on $\boldsymbol{u}$ are often appropriate, even when tracer dispersion is not periodic, as in the SC dispersion problem described in Sec. 3. A strategy for such problems is to use a 19-vector carrier fluid in a single repeat unit, N_x by N_y by N_z lattice units on a side, then save the equilibrium $\boldsymbol{u}(x, y, z)$. The memory used for the 19-vector carrier fluid is then reclaimed, and reallocated for additional repeat units of the tracer lattices in the x direction. The velocity field at x, y, z is then $\boldsymbol{u}((x\text{MOD}\,N_x), y, z)$.

The efficiency of the overall algorithm depends a great deal on the storage of the 3D lattice in memory. For systems that contain a high proportion of solids, it is often practical to store only those sites that represent open fluid or solids bounding fluid; a 3D pointer table obtains the location of the vectors in a linear array. Such an approach often leads to high locality, and can speed up the overall algorithm, by reducing the number of loads to the cache. For simplicity, we focus here on data structures that allocate the same amount of memory to all sites, be they solid or fluid. We next consider the trade-offs inherent in data structures that optimize for the translation versus collision step.

Let indices s, i, z, y, and x represent, respectively, the chemical component, vector number (e.g. $i = 0$ through 18 in D3Q19), and the Cartesian directions. Two obvious alternative schemes (for coding in C) are:

```
float *****ff;  /*arrays ff[s][i][z][y][x] */,
```

and

```
typedef float site[NVELOC]; /* NVELOC = 19 or 6 for 3D */
site ****ff;   /*arrays ff[s][z][y][x][i] */
```

The latter holds all the vectors close in memory, and might seem much more efficient. However, the first method can make the translation step very fast, principally because much of the translation is accomplished by pointer swaps. The efficiency is most significant for 3D, 6-vector tracers; only two of the six vectors are actually moved in memory, and those moves (in the + and $-x$ directions) involve just one index and are easily optimized. The second method generates much cache traffic, and the translation step requires substantial effort to decompose and redistribute the vectors from each site.

Figure 1 gives simplified source code for a collision function involving just one 3D, 6-vector tracer and a predefined, static flow field `uu`. To lessen the burden of dereferencing five-dimensional arrays, the pointers are dereferenced manually in steps, at the beginning of the inner loops. To ease the alias problem that plagues the C computer language, the six vectors for each site are copied into local `f0,f1,...,f5` for most manipulations. Finally, by factoring the f^{eq} equations, the number of arithmetic operations is greatly reduced. With judicious use of register declarations, the most important pointers and local variables can be kept in registers throughout the calculation, even on an Intel $\times 86$ chip.

Table 1 shows performance achievable on single-processor CPUs for several applications, given in units of MUPs (millions of site updates per second). For comparison, rates of 0.14 to 2 MUPs have been reported[13,15] for 2- and 3D calculations on a 64-node partition of the CM-5, containing 256 vector processors. This comparison is not intended to denigrate parallel computers. We estimate the 6-vector dispersion algorithms (2nd test problem, Table 1) are nearing the RAM streaming limit, with speed controlled by the rate that site data can be fetched from and returned to memory. In such conditions, it makes more sense to split the workload among many processors, rather than seek a faster CPU.

The generation of "denormal" numbers can greatly degrade performance in LB dispersion calculations. Denormals are part of the IEEE 754 standard, and are used to provide gradual underflow, allowing one to represent numbers smaller than $\sim 10^{-37}$, the traditional lower limit of IEEE single precision.[16] Most microprocessors do not handle denormals in hardware; instead, they provide the option of either ignoring the IEEE standard, and rounding the denormal to 0, or they pass the value

```
/* f0=(1,0,0);f1=(0,1,0);f2=(-1,0,0);f3=(0,-1,0);f4=(0,0,1);f5=(0,0,-1) */
typedef float ff_t; typedef float u_t;
struct veloc_t { u_t x; u_t y; u_t z; u_t usqrd;} ***UU; […]
/*****************************************************/
/* Disperse ff[1] only as a 6-vector tracer; ff[0] is carrier fluid. */
void JustDisperse_1st_Tracer(ff_t ***** ff, double *const tau,
       struct veloc_t ***uu, char ***solid,
       int nz, int ny, int nx, int nx_rep)
{
int z,y,x,xx;
double ux, uy, uz, f0,f1,f2,f3,f4,f5; /* f's = 6-vector distribution */
double rho, omega, beta, mux,muy,muz, mul, halfOmega, sixthOmega;
ff_t *pf0,*pf1,*pf2,*pf3,*pf4,*pf5;
ff_t **pf0z,**pf1z,**pf2z,**pf3z,**pf4z,**pf5z;

struct veloc_t **uuzp, *uup;
int solidcode;
char **solidzp, *solidp;

omega = 1.0/tau[1]; beta = 1.0 - omega;
halfOmega=ONE_HALF*omega; sixthOmega=ONE_6TH*omega;

for(z=0; z<nz; z++){
     uuzp = uu[z]; solidzp = solid[z]; /* deref outer pointers */
     pf0z = ff[1][0][z]; pf1z = ff[1][1][z]; pf2z = ff[1][2][z];
     pf3z = ff[1][3][z]; pf4z = ff[1][4][z]; pf5z = ff[1][5][z];
     for(y=0; y<ny; y++){
          solidp = solidzp[y];
          uup = uuzp[y]; /* deref inner pointers */
          pf0 = pf0z[y];  pf1 = pf1z[y];  pf2 = pf2z[y];
          pf3 = pf3z[y];  pf4 = pf4z[y];  pf5 = pf5z[y];
          for(x=xx=0; x<nx; x++,xx++,++pf0, ++pf1, ++pf2,
             ++pf3, ++pf4, ++pf5){
               if(xx==nx_rep) xx = 0; /* xx = x MOD nx_rep */
               solidcode = solidp[x];
               if(solidcode==-2) continue; /* completely surrounded */
               ux = uup[xx].x;  uy = uup[xx].y;  uz = uup[xx].z;
               rho = f0 = *pf0;   rho += f1 = *pf1;  rho += f2 = *pf2;
               rho += f3 = *pf3;  rho += f4 = *pf4;  rho += f5 = *pf5;
               if(solidcode){ /*bounce-back */
                    *pf0 = f2;  *pf1 = f3;  *pf2 = f0;
                    *pf3 = f1;  *pf4 = f5;  *pf5 = f4;
                    continue;  /* on to next x */
                    }
               mul = rho*halfOmega; /* implicit feq calc */
               mux = ux*mul; muy = uy*mul; muz = mul*uz;
               rho *= sixthOmega;
               /* put back the new f[] values */
               *pf0 = f0*beta + rho + mux;
               *pf1 = f1*beta + rho + muy;
               *pf2 = f2*beta + rho - mux;
               *pf3 = f3*beta + rho - muy;
               *pf4 = f4*beta + rho + muz;
               *pf5 = f5*beta + rho - muz;
               }
          }
     }
}    /* END JustDisperse_1st_Tracer() */
```

Fig. 1. Simplified collision function for one 3D 6-vector tracer, $\partial \boldsymbol{u}/\partial t = 0$.

Table 1. Performance for LB algorithms (MUPs = Millions site Updates Per Second).

Processor	**MUPs**
Test problem: 19-vector flow R(no tracers), 50% solids	
200 MHz Pentium Pro	0.29
400 MHz Pentium II	0.70
500 MHz alpha 21164a	1.10
Test problem: 3D 6-vector tracer dispersion, steady flow field, 10% solids	
200 MHz Pentium Pro	1.55
400 MHz Pentium II	2.55
Test problem: 3D, 3-component double-diffusive fingering with buoyancy	
200 MHz Pentium Pro	0.15
400 MHz Pentium II	0.30

to a software routine for handling. Since the latter process can be excruciatingly slow, the compiler default is generally to round to 0. However the Intel $\times$ 86 processors, such as the Pentium Pro, not only handle denormals in hardware, but provide no option to round these values to 0. Manipulations of denormals in $\times$ 86 processors are slow, flushing the processor pipeline with each load. The problem can be delayed by using double precision, with a factor $\sim$1.6 loss in initial speed and factor 2 loss of memory efficiency, and no gain in true accuracy for dispersion coefficients.

Slug dispersion problems have a strong tendency to generate denormal numbers. As a slug of solute travels downstream, very low concentrations are produced on the leading and trailing edges of the slug. In short order, the system contains many concentrations less than 10^{-37}. Our experience with dispersion around cylinder and sphere arrays showed up to a factor 6 performance loss within $\sim 10^4$ steps. This weakness of denormal processing was recognized by William Kahan, the principal architect of the IEEE standard, and a simple solution is given in his Berkeley lecture series (available on the World Wide Web as ieee754.ps): all initial and boundary conditions that are formally at 0 concentration, are set instead to a small number, say 10^{-32}. This artifice eliminates the slowdown, and has no effect on the calculated dispersion coefficients.

3. Accuracy and Choice of Wall Conditions

In recent years, much was written on the choice of wall conditions for LB calculations. The basic problem is that after the streaming step, there are f_i pointing into the solids, but none pointing out from the solids; these missing particle distributions must be created in a consistent manner. For pure flow calculations, the simple bounce-back condition, which fills in the missing f_i by inverting particle distributions at the solid wall, can yield acceptable accuracy. To achieve such accuracy with bounce-back, the collision parameter τ is kept reasonably low, and the wall solid wall is taken to be 1/2 lattice unit out from the solid nodes.[17] The subterfuge of using

January 20, 1999 17:2 WSPC/141-IJMPC 0097

the 1/2 location is commonly referred to as "reinterpreted" bounce-back. However, when applied to tracer particles, the justification for re-interpreted bounce-back is less clear, since the tracers have some residence time "behind" the 1/2 wall.

An alternative wall condition for tracer dispersion was suggested by Noble.[13] In brief, one places the zero-flow position on the wall, by one of the many methods suggested in the literature.[17,18] Both carrier fluid and tracer particles are allowed to stream along the walls. For tracer particles, the missing part of the distribution is obtained by reflecting across a plane tangent to the wall; in effect, this is a slip condition. The tracer distributions then are sent through the relaxation step, in which they inherit the diffusion coefficients typical of the free fluid.

Figure 2(a) compares the accuracy of the traditional bounce-back, and the method suggested by Noble,[13] for 3D LB algorithms in the classic Taylor–Aris dispersion problem. For this test, a steady-state Poiseuille flow is allowed to develop between two parallel plates. We take the x-axis parallel to the flow direction, and the y-axis perpendicular to the plates. Then, a slug of solute is injected into the flow. Aris[19] showed that after some characteristic time $t_c \sim hy^2/D_m$ (where hy is the channel width and D_m is the molecular diffusion coefficient), the projection of the solute distribution onto the x-axis is nearly Gaussian, and the dispersion coefficient D^* follows the simple rule:

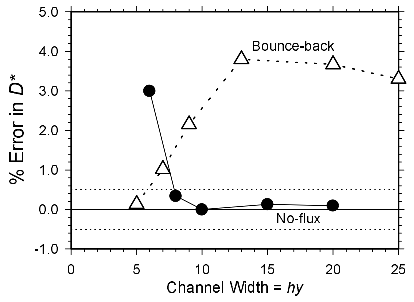
$$D^*/D_m = 1 + \mathrm{Pe}^2/210\,, \tag{5}$$

where the Péclet number $\mathrm{Pe} \equiv (hy \cdot U)/D_m$ (U is the cross-channel averaged speed). Equation (5) is taken as the predicted D^*/D_m. We can also "measure" D^* for our numerical experiment by the method of moments[19]:

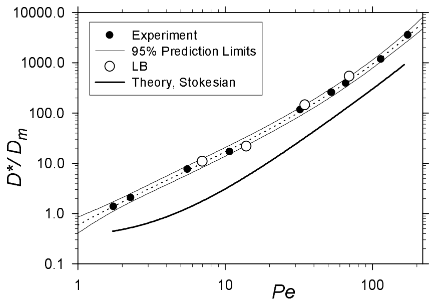
$$D^*/D_m = 1/2\, dm_2/dt/D_m\,, \tag{6}$$

where m_2 is the second moment, or variance of the solute distribution, projected onto the x-axis. The difference between Eq. (5) and Eq. (6) is our estimate of error in Fig. 2(a) (calculated for a fixed $\mathrm{Pe} = 20$). For all but the narrowest channels, the method suggested by Noble proves more accurate than the simple bounce-back. The bounce-back method reaches a peak error of about 3.8%, before slowly decreasing. The causes of this non-monotone behavior are not clear, but there are probably several competing effects.

Despite the modest accuracy, there are several good reasons to retain bounce-back for dispersion problems. The first is that the method leads to extremely simple and efficient code. Second, with bounce-back, there is a simple relationship between the number of solid nodes in the system and the volume of solids modeled; the benefits of this simple relationship become more obvious when one attempts to implement dissolution and precipitation, which can create very complicated patterns of dendritic growth. If growth occurs by a transition state rule,[20] the surface area and volume of each point in the automaton must be consistently known. Third, the use of bounce-back can be justified on a simpler basis: dispersion data in experiments are rarely measured to an accuracy of better than 10%, and theories for



(a)



(b)

Fig. 2. (a) Comparison of accuracy for bounce-back versus no-flux method (Ref. 13) for Taylor–Aris dispersion between infinite plates (3D algorithms). Fig. 2(b) Comparison of experimental results (Ref. 21), LB calculations (this study) and Stokesian theory (Ref. 22). Pe defined relative to particle diameter and averaged Darcy flow speed.

predicting dispersion in complex, 3D geometries rarely come within a factor of 2 of experimental measurements.

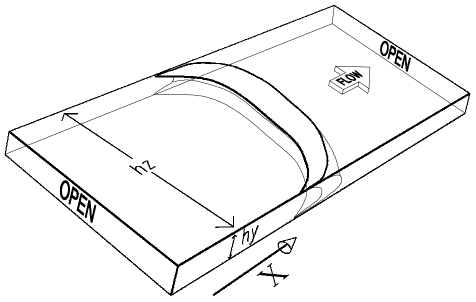
The latter point is illustrated by Fig. 2(b), which compares experimental data for dispersion in a simple cubic (SC) array of spheres (Gunn and Pryce[21]), with our LB results for the same geometry, and theory developed by Koch *et al.*[22] We might regard this as one of the simplest nontrivial 3D geometries, yet there is substantial disagreement between the lab results and Koch *et al.* analysis. To be fair, the latter authors recognized the short-comings of their approach (including the assumption of Stokesian flow, while modeling experiments with particle Re in excess of 100). However, several other studies have attempted to duplicate the Gunn and Pryce experiments, never obtaining better than a factor of two disagreement.[23–25] The variability of the experimental results in Fig. 2(b) is also notable. Most of these theories predict nearly straight-line behavior of D^*/D_m on a log–log plot, for sufficiently high Pe. We have plotted 95% "prediction lines" for the experimental data, based on a 3rd-order fit to the $\log(D^*/D_m)$ versus log(Pe) (3rd-order provided a narrower prediction band than orders 1, 2, 4 and 5). Though the LB results do not match the experimental data exactly, they do fall within the prediction bands.

4. Applications

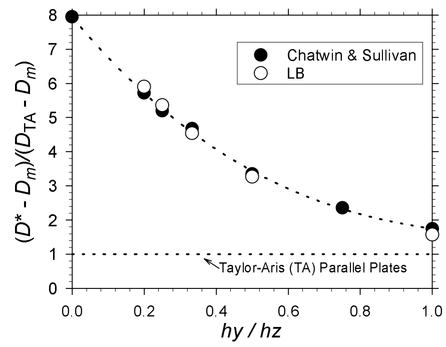
4.1. *Dispersion in fractures: the DDG effect*

Consider the 3D duct shown in Fig. 3(a), which extends infinitely along the x-axis, but is closed on the other four walls. It is commonly assumed that if $hy/hz << 1$, dispersion in the duct will follow the Taylor–Aris solution (Eq. 5). Surprisingly, as long as the duct has vertical sidewalls, dispersion never approaches the Taylor–Aris solution, regardless of the aspect ratio. As $hy/hz \to 0$, the solution approaches:

$$D^*/D_m = 1 + 7.95\,\mathrm{Pe}^2/210 \qquad (7)$$



(a)



(b)

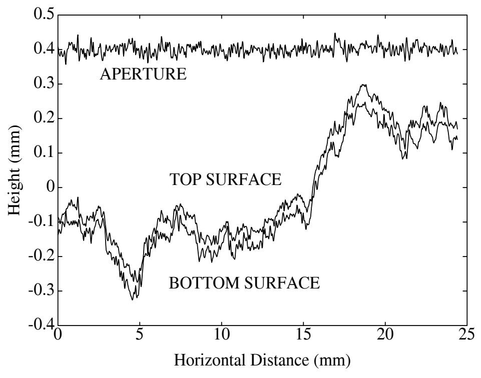
Fig. 3. (a) Geometry for duct dispersion. The DDG effect arises from sidewall drag on a slug of solute injected into the flow. Fig. 3(b) Comparison of LB calculations with Chatwin and Sullivan (Ref. 26); D_{TA} is the Taylor–Aris estimate of dispersion (Eq. 5).

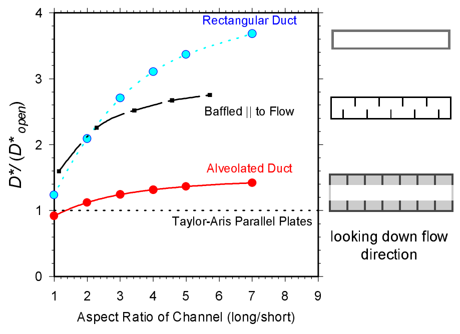
so for high Pe, the close-walled duct approaches a dispersion coefficient nearly eight times the Taylor–Aris value. Figure 3(a) gives a hint about the origin of this enhanced dispersion; the edges of the solute slug are swept back by the vertical duct walls, since the flow speed must vanish at the walls. Figure 3(b) compares our measurements of the DDG effect with the calculations of Chatwin and Sullivan;[26] the agreement is reasonably good, given that Chatwin and Sullivan used two methods that sometimes disagreed by 5%.

Curiously, the DDG effect is *not* observed in some experiments with rough-walled "fractures". For example, Tsuda *et al.*[27] performed experiments on dispersion in alveolated channels (models for the human lung), with stagnant pockets lining a central flow channel. They obtained reasonably good agreement between experiments and simple 2D solutions, that did not consider the finite extent of the system in the z direction. This result suggests that roughness may greatly diminish importance of the DDG effect.

By examining a cross-section of a real fracture (Fig. 4(a), provided by S. Brown of New England Research), we can propose two means by which roughness ameliorates the DDG effect. First, assume the flow is perpendicular to the page; the fracture aperture pinches and swells, containing numerous partial walls parallel to the direction of flow; thus we propose that there is always an effective duct wall near at hand for flow through a real fracture (like the walls parallel to the x–y planes in Fig. 3(a)). Second, the flow through such a rough fracture is likely channelized, with pockets of nearly stagnant fluid lining the main flow paths; perhaps the capacitance of these pockets so increases the D^* as to overwhelm the DDG effect.

Figure 4(b) shows an heuristic test of these hypotheses. The right side illustrates three simple geometries for comparison: the straight walled duct (at top); a duct with baffles parallel to the flow direction; and a duct lined by baffles both perpendicular and parallel to the flow, creating nearly stagnant fluid pockets. The





(a) (b)

Fig. 4. (a) Cross-section of real fracture between rock faces. Figure 4(b) Dispersion in closed-walled channel relative to open-walled channel, for 3 geometries on right; stagnant pockets in the alveolated channel (bottom) appear to minimize DDG effect.

Pe is kept reasonably constant for all three (Pe of 11 to 14), though there is some ambiguity in defining the effective "width" for the baffled geometries. At left are plotted the dispersion coefficients as functions of aspect ratio (normalized to a duct that is open on the left and right sides). The baffles parallel to the flow direction (middle case) have a modest effect, while the alveolated duct (bottom case) sees a much more rather dramatic reduction. These results suggest the stagnant pockets have such a marked effect on dispersion, that they probably overwhelm the DDG effect for many real fractures.

4.2. *Double-diffusive fingering: 3D versus 2D models*

When two solutions have solutes with very different D_m, they may admix rapidly by fingering, even when the initial layering is gravitationally stable. The leftmost side of Fig. 5 shows a small portion of a Hele–Shaw cell, which consists of two plexiglas plates, parallel to the page, separated by a 0.05 cm gap. Initially, the bottom of the cell contains a salt solution, and the top of the cell contains a less-dense sugar solution. Because the D_m of salt is ~ 3 times that of sugar, salt can diffuse upward into the sugar-rich region, creating a locally denser solution that is now gravitationally unstable, which leads to rapid fingering.[8] This phenomenon has been widely studied in the context of ocean mixing. For example, where the Mediterranean flows into the Atlantic, warm salty water overlies colder less saline water; heat diffuses 100 times as fast as salt, so as the Mediterranean heats up the underlying brine, eventually causing gravitational instability and fingering.[7]

For a Hele–Shaw cell, double-diffusive fingering is characterized by the solutal Rayleigh number for each component:

$$Ra = \beta \cdot \Delta C \cdot g \sin\theta \cdot (h^2/12) \cdot L/(D_m \cdot \nu)\,, \tag{8}$$

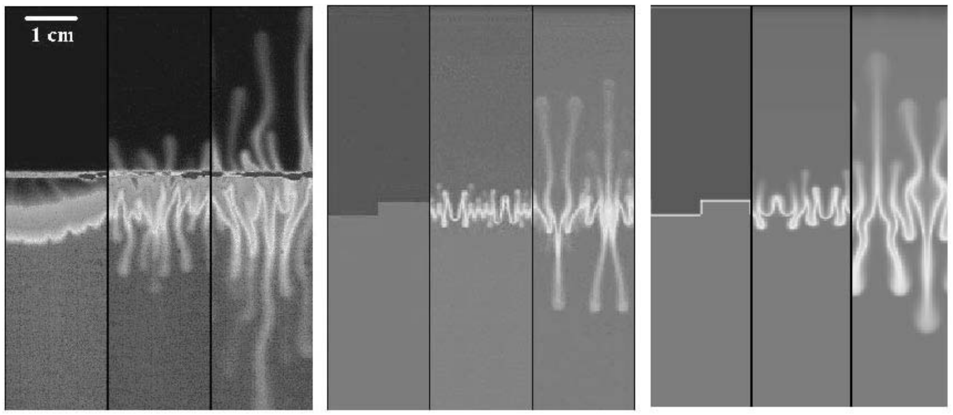


Fig. 5. Double-diffusive fingering in Hele–Shaw cell. Leftmost is experiment (less dense sugar solution initially in top of cell, and salt solution in bottom). Middle is 2D LB model; rightmost is 3D LB model.

where the β is a molar expansivity, h is the cell thickness, ≈ 0.05 cm; θ is the angle to horizontal; ΔC is concentration difference, top-to-bottom; and ν is kinematic viscosity. The L is a characteristic size of the initiating disturbance; it is customary to use the width of initial "smearing" after the divider pulled. However, the divider likely leaves a turbulent mixed zone, which is poorly characterized and constitutes a velocity initial condition as well as a concentration initial condition.

In the 2D LB model, the cell thickness is parameterized as h, and the drag force[12,28] is $8\nu\boldsymbol{u}/h^2$. The advantage of the 2D model is computational efficiency, and low memory use, so very large simulations can be run. The disadvantage is that it cannot directly capture the Taylor–Aris dispersion from the small separation (h) in the z direction. The fluid speed need not be great to cause substantial Taylor–Aris smearing; 3 to 10 cm/hour will double the effective dispersion coefficient. Many fingers reach this speed by the end of an experiment; and higher speeds can be reached in the initial stages, when the plexiglas strip is pulled and the oscillations in the interface quickly relax.

Figure 5 compares a lab experiment (leftmost) with the 2D and 3D LB models (middle and right, respectively). Both LB models use a simple step of height L (Eq. (8)) to initiate fingering. The 3D model uses 5 fluid cells in the z direction, and captures the Taylor–Aris dispersion; consequently, the fingers are wider and more "smeared", with less fine structure than is seen in the 2D model. Otherwise, the rates of finger growth, and the number of fingers per distance on the horizontal axis, are remarkably similar in the two LB models. The LB models also bear striking similarity to the fingers observed in lab experiments. However, for both LB models, the rate of finger growth was ~ 2 to 4 times slower than in the experiments; this discrepancy may reflect the difficulty in matching the conditions of the lab Hele–Shaw cell, after the strip separating the fluids is pulled.

5. Conclusions

With simple optimizations, the BGK LB method is fast enough, on a single-CPU computer, for modeling rough fractures containing millions of nodes. Simple bounce-back wall conditions are less accurate than other methods,[13] but yield acceptable errors ($< 4\%$), given the inherent uncertainty of dispersion experiments in complex geometries. Application of LB to the DDG problem suggests that in rough fractures, the capacitance of stagnant pockets will overwhelm the effects of duct sidewalls. LB provide useful models for double-diffusive fingering, probably limited most by the difficulty of matching initial conditions for concentration and velocity. 3D LB models are necessary to capture the effects of wall dispersion in Hele–Shaw cells, but the 3D fingering models provide simulations qualitatively similar to the 2D models in terms of finger growth rate and number.

Acknowledgments

This work was performed at Sandia National Laboratories supported by the U.S. Dept. of Energy under contract # DE-AC04-94AL85000, and was funded by the U.S. Department of Energy Basic Energy Sciences (BES) Program and the Sandia National Laboratories LDRD Program.

References

1. H. W. Stockman, *Water Resources Res.* **33**, 1823 (1997).
2. H. W. Stockman, C. Li, and J. L. Wilson, *Geophys. Res. Lett.* **24**, 1515 (1997).
3. C. Wels, L. Smith, and T. T. Vandergraaf, *Water Resources Res.* **32**, 1943 (1996).
4. T. T. Vandegraaf, D. J. Drew, D. Archambault, and K. V. Ticknor, *J. Contaminant Hydrology* **26**, 83 (1997).
5. Y. Fujikawa, F. Masami, D. J. Drew, and T. T. Vandegraaf, *J. Contaminant Hydrology* **14**, 207 (1993).
6. M. R. Doshi, P. M. Daiya, and W. N. Gill, *Chem. Engrg. Sci.* **33**, 795 (1978).
7. J. Y. Holyer, *J. Fluid Mech.* **147**, 169 (1984).
8. C. Cooper, R. J. Glass, and S. Tyler, *Water Resources Res.* **33**, 517 (1997).
9. Y. H. Qian, D. d'Humières, and P. Lallemand, *Europhys. Lett.* **17**, 479 (1992).
10. X. Shan, *Phys. Rev. E* **55**, 2780 (1997).
11. N. Martys and H. Chen, *Phys. Rev. E* **53**, 743 (1996).
12. E. G. Flekkøy, U. Oxaal, T. Feder, and T. Jøssang, *Phys. Rev. E* **52**, 4952 (1995).
13. D. R. Noble, Ph.D. Thesis, Univ. of Illinois, Urbana-Champaign, 1996.
14. D. Wolf-Gladrow, *J. Stat. Phys.* **79**, 1023 (1994).
15. D. Muders, Ph.D. thesis, Universität Bonn, 1995.
16. S. P. Morse, E. J. Isaacson, and D. J. Albert, *The 80386/387 Architecture* (John Wiley & Sons, New York, 1987).
17. S. J. Chen, D. Martínez, and R. Mei, *Phys. Fluids* **8**, 2527 (1996).
18. L. Ginzbourg and D. d'Humières, *J. Stat. Phys.* **84**, 927 (1996).
19. R. Aris, *Proc. Royal Soc.* 235A, 67 (1956).
20. A. C. Lasaga, in *Kinetics of Geochemical Processes, Rev. in Mineralogy 8*, ed. A.C. Lasaga *et al.* (Mineralogical Soc. America, Washington DC, 1981) p 135.
21. D. J. Gunn and C. Pryce, *Trans. Inst. Chem. Engrs.* **47**, T341 (1969).

22. D. L. Koch, R. G. Cox, H. Brenner, and J. F. Brady, *J. Fluid Mech.* **200**, 173 (1989).
23. C. K. Lee, C.-C. Sun, and C. C. Mei, *Int. J. Heat Mass Transfer* **39**, 661 (1996).
24. A. Eidsath, R. G. Carbonell, S. Whitaker, and L. R. Herrmann, *Chem. Engrg. Sci.* **38**, 1803 (1983).
25. J. Salles, J.-F. Thovert, R. Delaney, L. Prevors, J.-L. Auriault, and P. M. Adler, *Phys. Fluids* **5** 2348 (1993).
26. P. C. Chatwin and P. J. Sullivan, *J. Fluid Mech.* **120**, 347 (1982).
27. A. Tsuda, W. J. Federspiel, P. A. Grant, Jr., and J. J. Fredberg, *Chem. Engrg. Sci.* **46**, 1419 (1991).
28. R. Holme and D. H. Rothman, *J. Stat. Phys.* **68**, 409 (1992).