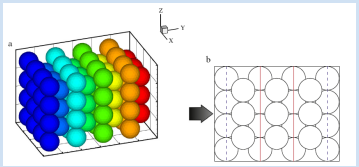
Optimal Parallelization of Colloid Tracking Through Porous Media

Charles Andersen
Lian Ping Wang, Queming Qiu
UD Peta-Apps Cloud Physics Team
Mechanical Engineering

Abstract

•Previously Professor Wang and his team have developed code for the tracking of colloids in porous media. They have developed both two dimensional and three dimensional models. These models track the movement of very small particles, called colloids, through artificially created "glass beads" transported by water. My project has been to take this code and optimize its implementation by finding bottlenecks in the code and think of ways to improve its performance. Parallelization of the code is absolutely vital to afford the computation power needed to solve problems such as this. The movement of each colloid must be tracked very carefully. It has been seen that because the flow moves slowly through the medium the colloid particles have a very large inertia and that the (comparatively) large forces acting on the particle are constantly changing in direction and magnitude which leads to the need for a very small time-step to accurately track the colloid. The two parallelization techniques that are to be explored in conjuncture with the colloid tracking code are Open MP (open multi-processing) and MPI (message passing interface).There are important practical implications to this problem regarding the modeling of contaminated groundwater and predicting rates of retention of chemicals carried in the water. This work was made possible by a grant from UD's Peta-Apps Cloud Physics team with their funding from the National Science Foundation. My work has also benefited from use of computer time from the National Center for Atmospheric Research and their supercomputer cluster Bluefire.

Fig. (a)*



* H. Gao, et al., Three-dimensional microscale flow simulation and colloid transport modeling in saturated soil prous media, Computers and Methematics with Applications (200), doi: 10.1016/j.camwa. 2009.08.057

Modeling

•The three dimensional porous flow is modeled in two parts. First, by calculating the flow field around modeled solid spheres, which have been densely packed, referred to as "glass beads" and then by injecting colloids which will travel along that flow field. The "glass beads" act as sensors to detect collision and deposition. A three dimensional picture of the setup is shown in figure (a). Colloids are injected at one boundary and flow until they reach the end of the domain where they are injected at the same location but at the initial boundary.

•High performance computing is the use of clustered computers to solve problems which require very large computation time by splitting the task among multiple processors, which is called parallelization. There are many methods of parallelization, however, the two I worked with this summer were OpenMP, or Open Multi-processing, and MPI, or Message Passing Interface. Because MPI can require reworking entire portions of the code I did not use MPI very much in my work this summer due to the relatively short time restriction.

•One of the reasons that high performance computing is necessary for the problem of tracking colloids is due to the many forces which are acting on the particles. These forces include Brownian forces, viscous drag forces, eletrokinetic forces, Lewis acid/base interaction forces, and the buoyancy force. The colloids tracking code also implemented using a dynamic time step. Instead of using one small time step in the code, whenever a colloid approaches a glass bead the time step must increase to be able to calculate position because it will now change much more rapidly than in the "bulk flow".

Colloid tracking code: Results and analysis

•I will examine only the results of the OpenMP implementation of the 3d colloids tracking code. The results shown here are for 1000 colloids injected into the system with 60 time steps between each injection. The results for this set up are listed in figure (d). The overall speedup of the code is marked below as "Total Time", and the speedup within the parallelized main section of the code is listed below as "Time in Loop". Figure (b) shows the results plotted as time vs. the number of threads used to execute the code on a log-log plot. A benchmark case from the simple code is also shown for a comparison of what a near perfect speedup would look like on a logarithm plot. Any code which scales well should have a slope similar to the simple code line. From this it can be visually confirmed that the "Time in Loop" has much better scalability. Figure (c) below shows the speed up for both timers graphically on a log-log plot. The maximum speed up for "Total Time" is calculated by dividing the time for 1 thread by the run time for 32 threads, approximately 966.75/267.62 = 3.61. The maximum speed up for the "Time in Loop" timer is approximately 684.82/59.48 = 11.51 times for 16 times the processors. It should be noted that at 32 processors the "Time in Loop" increases slightly which is caused by the cost of splitting up the task being greater then the benefit gained and could be fixed by injecting more particles. This may be limit of code at its current state and needs to be further investigated as how to improve for an even larger speedup.

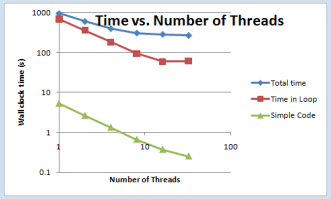


Fig. (b)

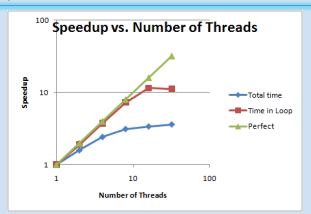


Fig. (c)

Threads	Total Time	Time in Loop	memory
1	966.75	684.82	221 MB
2	608.01	359.10	221 MB
4	399.13	183.01	221 MB
8	309.62	94.31	223 MB
16	285.63	59.48	223 MB
32	267.62	61.11	223 MB

Fig. (d)

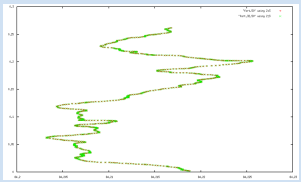


Fig. (e)

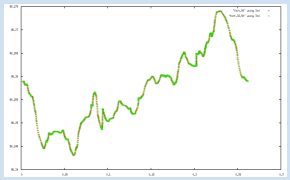


Fig. (f)

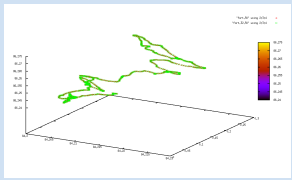


Fig. (g)

•The results of particle trajectory should be invariant under the number of processors used in calculation and to observe this is very important to ensure that the model is behaving correctly. Statements were inserted into the code to record the position of a single particle which was used with gnuplot to create the graphs shown above. In the example above the red marks show the particle trajectory when 8 processors were used and the green marks represent the particle trajectory when 32 processors were used. Note how the erratic particle follows the exact same path visually demonstrating the invariance. Figure (e) shows the cross section of the particle trajectory projected in the XY plane with the vertical axis as the Y direction and the horizontal axis as the X direction. Figure (f) shows the cross section of the particle trajectory projected in the YZ plane with the vertical axis as the Z direction and the horizontal axis as the Y direction. Figure (g) shows the path of the particle in three dimensional space.

Motivation

•There is a strong motivation for research into colloid transport and retention due to its importance in modeling the management of groundwater contamination where contaminate particles will be moving and may be deposited along their path. Colloid retention in practical applications depends heavily on individual features of an area, such as the geometry created by the local soil, so building a customizable model is of high importance. Even a highly simplified case of tracking colloid movement and retention in underground water through soil is very complicated to compute and models for doing so must be built on a carefully verified foundation of simpler cases.

Simple Code: Purpose and Analysis

•Simple test code is used to fully understand how the machine the work is being performed on as well as understanding the intricacies of OpenMP and MPI before making any changes to the colloid tracking code, such as implementing timers or testing thread assignment. The problem below is an example that was formulated for testing OpenMP on the Bluefire system. As shown on the graph to the left, this code was used to help benchmark the colloids transport code by showing what proper speedup for a problem should look like. Choosing a good problem to implement was very important. It needed to be a problem which requires a lot of computation <but not an impossible amount>. It should also be a problem for which the solution is known so that results can be quickly and easily verified. Calculating Euler's number e as a limit was a good choice, however it involved too little computation to see variation between trials. Re-writing the problem by binomial expansion gives the opposite problem of having too much to calculate due to the factorials involved. This problem can be overcome through some clever manipulation of the equation and becomes a good problem that takes neither too little nor too much time to compute. Figure (h) shows the results of these tests, and figure (i) shows these results in a log-log plot by itself.

The test problem is to compute e as

$$e = \lim_{n\to\infty}\left[1+\frac{1}{n}\right]^n = 1+\sum_{p=0}^{n-1}\frac{n!}{(n-p)!\,p!}\left(\frac{1}{n}\right)^{n-p}$$

$$= 1+\sum_{p=0}^{n-1}\left\{\prod_{k=0}^{k=n-p}\frac{(n-k)}{(n-p-k)n}\right\}$$

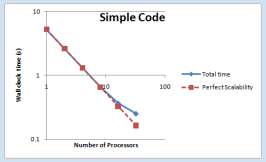


Fig. (i)

Threads	CPU/ thread For looping (s)	Total for code	memory
1	5.24	5.240	4 MB
2	2.62	2.620	4 MB
4	1.31	1.311	4 MB
8	0.655	0.655	4 MB
16	0.327 – 0.330	0.368	3 to 4 MB
32 nonexclusive	0.164-0.251	0.251	4 MB
32 exclusive	0.164-0.204	0.225	4 MB

Fig. (h)