MIT Libraries | DSpace@MIT

MIT Open Access Articles

A Multi-Core Numerical Framework for Characterizing Flow in Oil Reservoirs

The MIT Faculty has made this article openly available. ***Please share*** how this access benefits you. Your story matters.

Citation: Leonardi, Christopher R. et al. "A Multi-Core Numerical Framework for Characterizing Flow in Oil Reservoirs." in Papers of the 19th High Performance Computing Symposium (HPC 2011) Boston, Massachusetts, USA April 4–6, 2011.

As Published: http://hosting.cs.vt.edu/hpc2011/final-prog.html

Publisher: Society for Modeling & Simulation International

Persistent URL: http://hdl.handle.net/1721.1/67451

Version: Author's final manuscript: final author's manuscript post peer review, without publisher's formatting or copy editing

Terms of use: Creative Commons Attribution-Noncommercial-Share Alike 3.0

Massachusetts Institute of Technology

Presented at the SCS Spring Simulation Multi-Conference – SpringSim 2011, April 4-7, 2011 – Boston, USA
Awarded Best Paper in the 19th High Performance Computing Symposium and Best Overall Paper at SpringSim 2011

A Multi-Core Numerical Framework for Characterizing Flow in Oil Reservoirs

Christopher R. Leonardi, Civil and Environmental Engineering, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139 chrisleo@mit.edu
David W. Holmes, Department of Mechanical Engineering, James Cook University, Angus Smith Drive, Douglas, QLD 4811, Australia david.holmes1@jcu.edu.au
John R. Williams, Civil and Environmental Engineering and Engineering Systems, Massachusetts Institute of Technology, 77 Massachusetts Avenue, Cambridge, MA 02139 jrw@mit.edu
Peter G. Tilke, Department of Mathematics and Modeling, Schlumberger-Doll Research Center, 1 Hampshire Street, Cambridge, MA 02139 tilke@slb.com

Keywords: Parallel computation, multi-core, smoothed particle hydrodynamics, lattice Boltzmann method, enhanced oil recovery

Abstract

This paper presents a numerical framework that enables scalable, parallel execution of engineering simulations on multi-core, shared memory architectures. Distribution of the simulations is done by selective hash-tabling of the model domain which spatially decomposes it into a number of orthogonal computational tasks. These tasks, the size of which is critical to optimal cache blocking and consequently performance, are then distributed for execution to multiple threads using the previously presented task management algorithm, H-Dispatch. Two numerical methods, smoothed particle hydrodynamics (SPH) and the lattice Boltzmann method (LBM), are discussed in the present work, although the framework is general enough to be used with any explicit time integration scheme. The implementation of both SPH and the LBM within the parallel framework is outlined, and the performance of each is presented in terms of speed-up and efficiency. On the 24-core server used in this research, near linear scalability was achieved for both numerical methods with utilization efficiencies up to 95%. To close, the framework is employed to simulate fluid flow in a porous rock specimen, which is of broad geophysical significance, particularly in enhanced oil recovery.

1. INTRODUCTION

The extension of engineering computations from serial to parallel has had a profound effect on the scale and complexity of problems that can be modeled in continuum and discontinuum mechanics. Traditionally, such parallel computing has almost exclusively been undertaken with distributed memory parallel architectures, such as clusters of single-processor machines. A number of authors have reported on parallel particle methods (of which smoothed particle hydrodynamics, SPH, is an example) demonstrating scalability on such architectures, for example Walther and Sbalzarini [1] and Ferrari et al. [2].

The lattice Boltzmann method (LBM) has also been a popular candidate for distributed computing which is unsurprising due to the naturally parallel characteristics of its traditionally regular, orthogonal grid and local node operations. For example, Vidal et al. [3] presented results incorporating five billion LBM nodes with a speed-up efficiency of 75% on 128 processors. Götz et al. [4] simulated dense particle suspensions with the LBM and a rigid body physics engine on an SGI Altix system with 8192 cores (based on dual-core processors). At 7800 processors an efficiency of approximately 60% is achieved in a simulation featuring 15.6 billion LBM nodes and 4.6 million suspended particles. Bernaschi et al. [5] utilized a GPU implementation of a multi-component LBM and in 2D simulations of 4.2 million nodes achieved a speed-up factor of 13 over their benchmark CPU performance. Of particular relevance to this study is the work of Zeiser et al. [6] in which a parallel cache blocking strategy was used to optimally decompose space-time of their LBM simulations. In addition, this strategy was purported to be cache-oblivious so that the decomposed blocks were automatically matched to the cache size of the hardware used, minimizing the latency of memory access during the simulation.

Shared memory multi-core processors have emerged in the last five years as a relatively inexpensive "commercial-off-the-shelf" hardware option for technical computing. Their development has been motivated by the current clock-speed limitations that are hindering the advancement, at least in terms of pure performance, of single processors [7]. However, as a comparatively young technology, there exists little published work ([8] is one example) addressing the implementation of numerical codes on shared memory multi-core processors. With the expense and high demand for compute time on cluster systems, multi-core represents an attractive and accessible HPC alternative, but the known challenges of software development on such architectures (i.e. thread safety and memory bandwidth issues [9, 10, 11, 12]) must be addressed. Multi-core technologies are importantly beginning to infiltrate all levels of computing, including within each node of modern cross-machine

clusters. As such, the development of scalable programming strategies for multi-core will have widespread benefit.

A variety of approaches to programming on multi-core have been proposed to date. Commonly, concurrency tools from traditional cluster computing like MPI [13] and OpenMP [14] have been used to achieve fast take-up of the new technology. Unfortunately, fundamental differences in the cross-machine and multi-core architectures mean that such approaches are rarely optimal for multi-core and result in poor scalability for many applications. In response to this, Chrysanthakopoulos and co-workers [15, 16], based on earlier work by Stewart [17], have implemented multi-core concurrency libraries using Port based abstractions. These mimic the functionality of a message passing library like MPI, but use shared memory as the medium for data exchange, rather than exchanging serialized packets over TCP/IP. Such an approach provides flexibility in program structure, while still capitalizing on the speed advantages of shared memory. Perhaps as a reflection of the growing importance of multi-core software development, a number of other concurrency libraries have been developed such as Axum and Cilk++. In addition, the latest .NET Framework includes a Task Parallel Library (TPL) which provides methods and types, with varying degrees of abstraction, which can be used with minimal programmatic difficulty to distribute tasks on multiple threads.

In an earlier paper [18] we have shown that a programming model developed using such port-based techniques described in [15, 16] provides significant performance advantages over approaches like MPI and OpenMP. Importantly, it was found that the H-Dispatch distribution model facilitated adjustable cache-blocking which allowed performance to be tuned via the computational task size. In this paper, we apply the proposed programming model to the parallelization of both particle based methods and fixed-grid numerical methods on multi-core. The unique challenges in parallel implementation of both methods will be discussed and the performance improvements will be presented.

The layout of this paper is as follows. In Section 2 a brief description of the multi-core distribution model, H-Dispatch, is provided. Both the SPH and LBM numerical methods are outlined in Section 3 and the relevant aspects of their implementation in the multi-core framework, including thread safety and cache memory efficiency, are discussed. Section 4 presents performance test results from both the SPH and LBM simulators as run on a 24-core server and, finally, an application of the multi-core numerical framework to a porous media flow problem relevant to enhanced oil recovery is presented in Section 5.

2. MULTI-CORE DISTRIBUTION

One of the hindrances to scalable, cross-machine distribution of numerical methods is the communication of *ghost regions*. These regions correspond to neighboring sections of the problem domain (resident in memory on other cluster nodes) which are required on a cluster node for the processing of its own sub-domain. In the LBM this is typically a 'layer' of grid points that encapsulates the local sub-domain, but in SPH the layer of neighboring particles required is equal to the radius of the compact support zone. In 3D it can be shown that, depending on the sub-domain size, the communicated fraction of the problem domain can easily exceed 50%. In this situation Amdahl's Law [7], and the fact that traditional cross-machine parallelism using messaging packages is a serial process, dictates that this type of distributed memory approach will scale poorly.

If a problem is divided into spatial sub-domains for multi-core distribution, ghost regions are no longer necessary because adjacent data is readily available in shared memory. Further, the removal of relatively slow network communications required in cluster computing allows for an entirely new programming paradigm. Sub-domains can take on any simple shape or size and threaded programming means many small sub-domains can be processed on each core from an events queue rather than needing to approximate a single large, computationally balanced domain for each processor. Consequently, dynamic domain decomposition becomes unnecessary and a particle's position in a domain can be as simple as a spatial hashing, allowing advection to proceed with minimal management. Such characteristics mean that multi-core is perfectly suited to the parallel implementation of particle methods, however, shared memory challenges such as thread safety and bandwidth limitations must be addressed.

The decomposition of the spatial domain of a numerical method creates a number of computational tasks. Multi-core distribution of these tasks requires the use of a coordination tool to manage them onto processing cores in a load balanced way. While such tasks could easily be distributed using a traditional approach like scatter-gather, here the H-Dispatch programming model of [18] has been used because of the demonstrated advantages for performance and memory efficiency.

A schematic illustrating the functionality of the H-Dispatch programming model is shown in Figure 1. The figure shows three enduring threads (corresponding to three processing cores) that remain active through each time step of the analysis. A simple problem space with nine decomposed tasks is distributed across these threads by H-Dispatch. The novel feature of H-Dispatch is the way in which tasks are distributed to threads. Rather than a scatter or *push* of tasks from the manager to threads, here threads request values when free. H-Dispatch manages requests and distributes cells to the requesting threads accordingly. It is this *pull* mechanism that enables the use of a single thread per core as threads only request a value when free, thus, there is never more than one task at a time associated with a

given enduring thread (and its associated local variable memory). Additionally, when all tasks in the problem space have been dispatched and processed, H-Dispatch identifies step completion (i.e. synchronization) and the process can begin again.

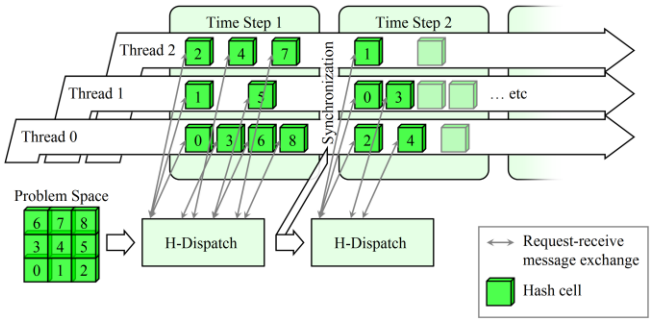

Figure 1. Schematic representation of the H-Dispatch programming model [18] used to distribute tasks to cores. An enduring processing thread is available for each core, which is three in this simplified representation, and H-Dispatch coordinates tasks to threads in a load balanced way over a number of time steps.

The key benefit of such an approach from the perspective of memory usage is in the ability to maintain a single set of local variables for each enduring thread. The numerical task associated with analysis on a sub-domain will inevitably require local calculation variables, often with significant memory requirements (particularly for the case of particle methods). Overwriting this memory with each allocated cell means the number of local variable sets will match core count, rather than total cell count. Considering that most problems will be run with core counts in the 10's or 100's, but cell counts in the 1,000's or 10,000's, this can significantly improve the memory efficiency of a code. Additionally, in managed codes like C#.NET and Java, because thread local variable memory remains active throughout the analysis, it is not repeatedly reclaimed by the garbage collector, a process that holds all other threads until completion and degrades performance markedly (see [18]).

3. NUMERICAL METHODS

The multi-core numerical framework featured in this paper has been designed in a general fashion so as to accommodate any explicit numerical method, such as SPH, LBM, the discrete element method (DEM), the finite element method (FEM) or finite difference (FD) techniques. It is worth noting that it could be adapted to accommodate implicit, iterative schemes with the correct data structures for thread safety but that is not the focus of this work. Instead, this study will focus on SPH and LBM, however the performance of the multi-core framework with an FD scheme has been previously reported [18].

3.1. Smoothed Particle Hydrodynamics

SPH is a mesh-free Lagrangian particle method which was first proposed for the study of astrophysical problems by Lucy [19] and Gingold and Monaghan [20], but is now widely applied to fluid mechanics problems [21]. A key advantage of particle methods such as SPH (see also dissipative particle dynamics (DPD) [22]) is in their ability to advect mass with each particle, thus removing the need to explicitly track phase interfaces for problems involving multiple fluid phases or free surface flows. However, the management of free particles brings with it the associated computational cost of performing spatial reasoning at every time step. This requires a search algorithm to determine which particles fall within the compact support (i.e. interaction) zone of a particle and then processing each interacting pair. Nevertheless, in many circumstances this expense can be justified by the versatility with which a variety of multi-physics phenomena can be included.

SPH theory has been detailed widely in the literature with various formulations having been proposed. The methodology of authors such as Tartakovsky and Meakin [23, 24] and Hu and Adams [25] has been shown to perform well for the case of multi-phase fluid flows. Their *particle number density* variant of the conventional SPH formulation removes erroneous artificial surface tension effects between phases and allows for phases of significantly differing densities. Such a method has been used for the performance testing in this work.

The discretized particle number density SPH equations for some field quantity, A_i, is given as,

$$A_i = \sum_j \frac{A_j}{n_j} W\left(\mathbf{r}_i - \mathbf{r}_j, h\right), \tag{1}$$

along with its gradient,

$$\nabla A_i = \sum_j \frac{A_j}{n_j} \nabla_i W\left(\mathbf{r}_i - \mathbf{r}_j, h\right), \tag{2}$$

where $n_i = \rho_i / m_i = \sum_j W\left(\mathbf{r}_i - \mathbf{r}_j, h\right)$ is the particle number density term, while W is the smoothing function (typically a Gaussian or some form of spline), h is the smoothing length and $\mathbf{r}_i$ and $\mathbf{r}_j$ are position vectors. These expressions are applied to the Navier-Stokes conservation equations to determine the SPH equations of motion.

Computing density directly from (1) gives,

$$\rho_i = m_i \sum_j W\left(\mathbf{r}_i - \mathbf{r}_j, h\right), \tag{3}$$

where this expression conserves mass exactly, much like the summation density approach of conventional SPH.

An appropriate term for particle velocity rate has been provided by Morris et al. [26], and used by Tartakovsky and Meakin [23], where,

$$\frac{d\mathbf{v}_i}{dt} = \frac{-1}{m_i}\sum_{j=1}^{N}\left(\frac{P_i}{\rho_i^2}+\frac{P_j}{\rho_j^2}\right)\frac{dW_{ij}}{d\mathbf{r}_i} + \mathbf{F}_i + \frac{1}{m_i}\sum_{j=1}^{N}\frac{\mu_i+\mu_j}{n_i n_j}\left(\mathbf{v}_i-\mathbf{v}_j\right)\frac{\mathbf{r}_i-\mathbf{r}_j}{\left|\mathbf{r}_i-\mathbf{r}_j\right|^2}\cdot\frac{dW_{ij}}{d\mathbf{r}_i} \quad (4)$$

in which P_i is the particle pressure, μ_i is the dynamic viscosity, $\mathbf{v}_i$ is the particle velocity and $\mathbf{F}_i$ is the body force applied on the i^{th} particle.

Surface tension is introduced into the method via the superimposition of pair-wise inter-particle forces following Tartakovsky and Meakin [24],

$$\mathbf{F}_{ij} = \begin{cases} s_{ij}\cos\left(\frac{3\pi}{2\kappa h}\left|\mathbf{r}_j-\mathbf{r}_i\right|\right)\frac{\mathbf{r}_j-\mathbf{r}_i}{\left|\mathbf{r}_j-\mathbf{r}_i\right|} & ,\left|\mathbf{r}_j-\mathbf{r}_i\right| \le \kappa h \\ 0 & ,\left|\mathbf{r}_j-\mathbf{r}_i\right| > \kappa h \end{cases}, \quad (5)$$

wherein s_{ij} is the strength of force between particles *i* and *j*, while κh is the interaction distance of a particle. By defining s_{ij} as being stronger between particles of the same phase, than between particles of a different phase, surface tension manifests naturally as a result of force imbalances at phase interfaces. Similarly, s_{ij} can be defined to control the wettability properties of a solid.

Solid boundaries in the simulator are defined using rows of virtual particles similar to that used by Morris et al [26], and no-slip boundary conditions are enforced for low Reynolds number flow simulations using an artificially imposed boundary velocity method developed in [27] and shown to produce high accuracy results.

3.1.1. Multi-Core Implementation of SPH

Because particle methods necessitate the recalculation of interacting particle pairs at regular intervals, algorithms that reduce the number of candidate interacting particles to check are critical to numerical efficiency. This is achieved by spatial hashing, which assigns particles to cells or 'bins' based on their Cartesian coordinates. With a cell side length greater than or equal to the interaction depth of a particle, all candidates for interaction with some target particle will be contained in the target cell, or one of the immediately neighboring cells. The storage of particle cells is handled using hash table abstractions such as the Dictionary<Tkey, Tvalue> class in C#.NET [28], and parallel distribution is performed by allocation of cell keys to processors from an events queue. In cases where data is required from particles in adjacent cells, it is addressed directly using the key of the relevant cell.

With the described *particle cell* decomposition of the domain care must be taken to avoid common shared memory problems like race conditions and thread contention. To circumvent the problems associated with using locks (coarse grained locking scales poorly, while fine grained locking is tedious to implement and can introduce deadlocking conditions [29]) the SPH data can be structured to remove the possibility of thread contention altogether. By storing the present and previous values of the SPH field variables, necessary gradient terms can be calculated as functions of values in previous memory, while updates are written to the current value memory. This reduces the number of synchronizations per time step from two (if the gradient terms are calculated before synchronizing, followed by the update of the field variables) to one, and a rolling memory algorithm switches the index of previous and current data with successive time steps.

An important advantage of the use of spatial hashing to create particle cells is the ease with which the cell size can be used to optimize cache blocking. By adjusting the cell size, the associated computational task can be re-sized to fit in cache close to the processor (e.g. L1 or L2 cache levels). It can be shown that cells fitting completely in cache demonstrate a significantly better performance (15 to 30%) than those that overflow cache causing an increase in cache misses, because cache misses require that data then be retrieved from RAM with a greater memory latency.

3.2. The Lattice Boltzmann Method

The lattice Boltzmann method (LBM) (see [30] for a review) has been established in the last 20 years as a powerful numerical method for the simulation of fluid flows. It has found application in a vast array of problems including magnetohydrodynamics, multiphase and multicomponent flows, flows in porous media, turbulent flows and particle suspensions.

The primary variables in the LBM are the particle distribution functions, $f_i(\mathbf{x},t)$, which exist at each of the lattice nodes that comprise the fluid domain. These functions relate the probable amount of fluid 'particles' moving with a discrete speed in a discrete direction at each lattice node at each time increment. The particle distribution functions are evolved at each time step via the two-stage, collide-stream process as defined in the lattice-Bhatnagar-Gross-Krook [31] equation (LBGK),

$$f_i(\mathbf{x}+\mathbf{c}_i\Delta t, t+\Delta t) = f_i(\mathbf{x},t) - \frac{\Delta t}{\tau}\left[f_i(\mathbf{x},t) - f_i^{eq}(\mathbf{x},t)\right] + A\mathbf{G}\cdot\mathbf{c}_i, \quad (6)$$

in which $\mathbf{x}$ defines the node coordinates, Δt is the explicit time step, $\mathbf{c}_i = \Delta\mathbf{x}/\Delta t$ defines the lattice velocities, τ is the relaxation time, $f_i^{eq}(\mathbf{x},t)$ are the nodal equilibrium functions, $\mathbf{G}$ is a body force (e.g. gravity) and A is a mass-conserving constant. The collision process, which is described by the first two terms in the RHS of (6), monotonically relaxes the particle distribution functions towards their respective equilibria. The redistributed

functions are then adjusted by the body force term, after which the streaming process propagates them to their nearest neighbor nodes.

Spatial discretization in the LBM is typically based on a periodic array of polyhedra, but this is not mandatory [32]. A choice of lattices is available in two and three dimensions with an increasing number of velocities and therefore symmetry. However, the benefits of increased symmetry can be offset by the associated computational cost, especially in 3D. In the present work the D3Q15 lattice is employed, whose velocity vectors are included in (7).

$$\mathbf{c}_i = c\left[\begin{Bmatrix}0\\0\\0\end{Bmatrix}\begin{Bmatrix}1\\0\\0\end{Bmatrix}\begin{Bmatrix}-1\\0\\0\end{Bmatrix}\begin{Bmatrix}0\\1\\0\end{Bmatrix}\begin{Bmatrix}0\\-1\\0\end{Bmatrix}\begin{Bmatrix}0\\0\\1\end{Bmatrix}\begin{Bmatrix}0\\0\\-1\end{Bmatrix}\right.$$
$$\left.\begin{Bmatrix}1\\1\\1\end{Bmatrix}\begin{Bmatrix}-1\\-1\\1\end{Bmatrix}\begin{Bmatrix}1\\1\\-1\end{Bmatrix}\begin{Bmatrix}-1\\1\\1\end{Bmatrix}\begin{Bmatrix}1\\-1\\-1\end{Bmatrix}\begin{Bmatrix}-1\\1\\-1\end{Bmatrix}\begin{Bmatrix}1\\-1\\1\end{Bmatrix}\begin{Bmatrix}-1\\1\\1\end{Bmatrix}\right] \quad (7)$$

The macroscopic fluid variables, density, $\rho = \sum_i f_i$, and momentum flux, $\rho = \sum_i f_i \mathbf{c}_i$, are calculated at each lattice node as velocity moments of the particle distribution functions. The definitions of the fluid pressure and viscosity are by-products of the Chapman-Enskog expansion (see [34] for details), which shows how the Navier-Stokes equations are recovered in the near-incompressible limit with isotropy, Galilean invariance and a velocity independent pressure. An isothermal equation of state, $p = c_s^2 \rho$, in which $c_s = c/\sqrt{3}$ is the lattice speed of sound, is used to calculate the pressure directly from the density, while the kinematic viscosity,

$$\nu = \frac{1}{3}\left(\tau - \frac{1}{2}\right)\frac{\Delta \mathbf{x}^2}{\Delta t}, \quad (8)$$

is evaluated from the relaxation and discretization parameters. The requirement of positive viscosity in (8) mandates that $\tau > 1/2$ and to ensure near-incompressibility of the flow the computational Mach number is limited, $Ma = u/c_s << 1$.

The most straightforward approach to handling wall boundary conditions is to employ the bounce-back technique. Although it has been shown to be generally first-order in accuracy [35], as opposed to the second order accuracy of the lattice Boltzmann equation at internal fluid nodes [30], its operations are local and the orientation of the boundary with respect to the grid is irrelevant. A number of alternative wall boundary techniques [36, 37] that offer generalized second-order convergence are available in the LBM, however these are at the expense of the locality and simplicity of the bounce-back condition.

In the present work, the immersed moving boundary (IMB) method of Noble and Torczynski [38] is employed to handle the hydrodynamic coupling of the fluid and structure. In this method the LBE is modified to include an additional collision term which is dependent on the proportion of the nodal cell that is covered by solid, thus improving the boundary representation and smoothing the hydrodynamic forces calculated at an obstacle's boundary nodes as it moves relative to the grid. Consequently, it overcomes the momentum discontinuity of bounce-back and link-bounce-back-based [39] techniques and provides adequate representation of non-conforming boundaries at lower grid resolutions. It also retains two critical advantages of the LBM, namely the locality of the collision operator and the simple linear streaming operator, and thus facilitate solutions involving large numbers of irregular-shaped, moving boundaries. Further details of the IMB method and the coupling of the LBM to the DEM, including an assessment of mixed boundary conditions in various flow geometries, can be found in Owen et al. [40].

3.2.1. Multi-Core Implementation of the LBM

Two characteristic aspects of the LBM often result in it being described as a naturally parallel numerical method. The first feature is the regular, orthogonal discretization of space, which is typical of Eulerian schemes, and can simplify domain decomposition. The second feature is the use of only local data to perform nodal operations, which consequently results in particle distribution functions at a node being updated using only the previous values. However, it should be noted that inclusion of additional features such as flux boundary conditions and non-Newtonian rheology, if not implemented carefully, can negate the locality of operations.

The obvious choice for decomposition of the LBM domain is to use cubic *nodal bundles*, as shown schematically in Figure 2. The bundles are analogous to the particle cells that were used in SPH, and similarly H-Dispatch is used to distribute bundle keys to processors. Data storage is handled using a Dictionary of bundles, which are in turn Dictionaries of nodes. This technique is used, as opposed to a master Dictionary of all nodes, to overcome problems that can occur with Collection limits (approximately 90 million on the 64-bit server used here).

By definition, the LBM nodal bundles can be used to perform cache blocking just as the particle cells were in SPH. With the correct bundle size, the associated computational task can be stored sequentially in processor cache and the latency associated with RAM access can be minimized. Similar techniques for the LBM have been reported in [41, 42] and extended to perform decomposition of space-time [6] (as opposed to just space) in a way that is independent of cache size (a recursive algorithm is used to determine the optimal block size).

To ensure thread safety, two copies of the LBM particle distribution functions at each node are stored. Nodal processing is undertaken using the current values, which are

then overwritten and propagated to the future data sets of appropriate neighbor nodes. Techniques such as SHIFT [43] have been presented which employ specialized data structures that remove the need for storing two copies of the particle distribution functions, however this is at the expense of the flexibility of the code. Note that the collide-push sequence implemented here can easily be reordered to a pull-collide sequence, with each having their own subtle conveniences and performance benefits depending on the data structure and hardware employed [44, 42].

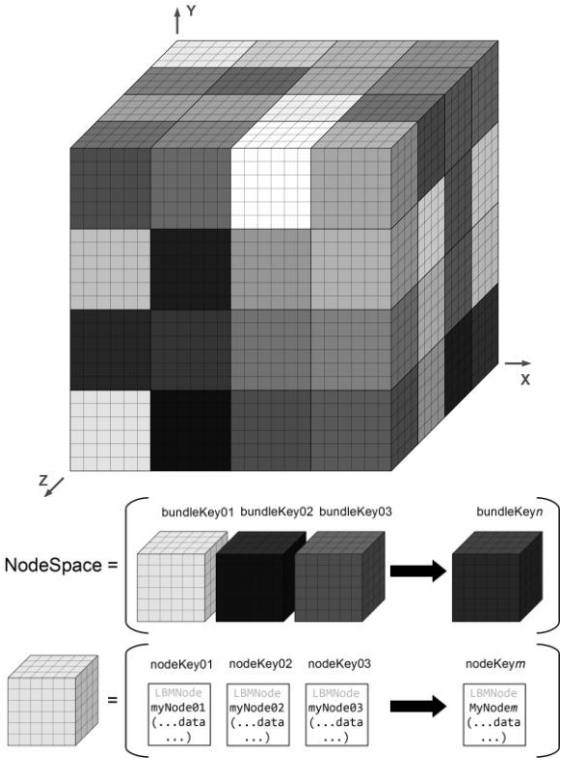

Figure 2. Schematic representation of the decomposition of the LBM domain into nodal bundles. Data storage is handled using a Dictionary of bundles, which are in turn Dictionaries of nodes.

4. PARALLEL PERFORMANCE OF METHODS

The parallel performance of SPH and the LBM in the multi-core numerical framework was tested on a 24-core Dell Server PER900 with Intel Xeon CPU, E7450 @ 2.40 GHz, running 64-bit Windows Server Enterprise 2007. Here, two metrics are used to define the scalability of the simulation framework, namely $SpeedUp = t_{1proc} / t_{nproc}$ and $Efficiency = SpeedUp / N$. Obviously, idealized maximum performance corresponds to a speed-up ratio equal to the number of cores at which point efficiency would be 100%.

Figure 3 graphs the increasing speed-up of the SPH solver with increasing cores. The test problem simulated flow through a porous geometry determined from microtomographic images of oil reservoir rock (see Section 5). Approximately 1.4 million particles were used in the simulation and the execution duration was defined as the time in seconds taken to complete a time step, averaged over 100 steps. For the double-search algorithm a speed-up of approximately 22 was achieved with 24 cores, which corresponds to an efficiency of approximately 92%. This, in conjunction with the fact that the processor scaling response is near linear, is an excellent result.

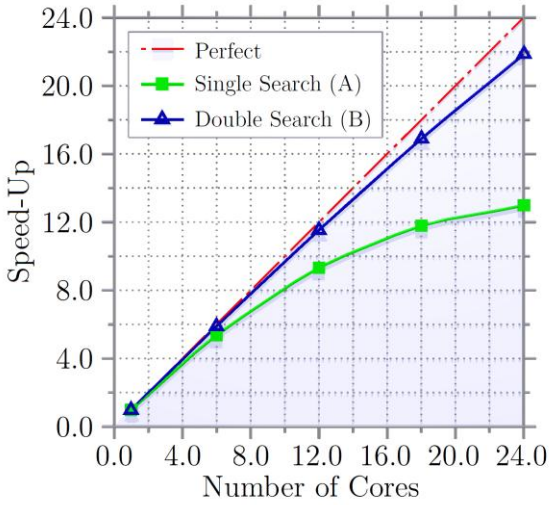

Figure 3. Multi-core, parallel performance of the SPH solver contrasting the counterintuitive scalability of the single-search and double-search algorithms.

The results in Figure 3 also provide an interesting insight into the comparative benefits of minimizing computation or minimizing memory commit when using multi-core hardware. The single-search result is attained with a version of the solver that performs the spatial reasoning once per time step and stores the results in memory for use twice per time step. Conversely, the double-search results are achieved when the code is modified to perform the spatial search twice per time step, as needed. Intuitively, the single-search approach requires a greater memory commit but the double-search approach requires more computation. However, it is counterintuitive to see that double-search significantly outperforms single-search, especially as the number of processors increases. This can be attributed to better cache blocking of the second approach and the smaller amount of data experiencing latency when loaded from RAM to cache. The fact that such performance gains only manifest when more than 10 cores are used, suggests that for less than 10 cores, RAM pipeline bandwidth is sufficient to handle a global interaction list.

As in the SPH testing, the LBM solver was assessed in terms of speed-up and efficiency. Figure 4 graphs speed-up against the number of cores for a 3D duct flow problem on

a 200^3 domain. Periodic boundaries were employed on the in-flow and out-flow surfaces and the bounce-back wall boundary condition was used on the remaining surfaces. A constant body force was used to drive the flow.

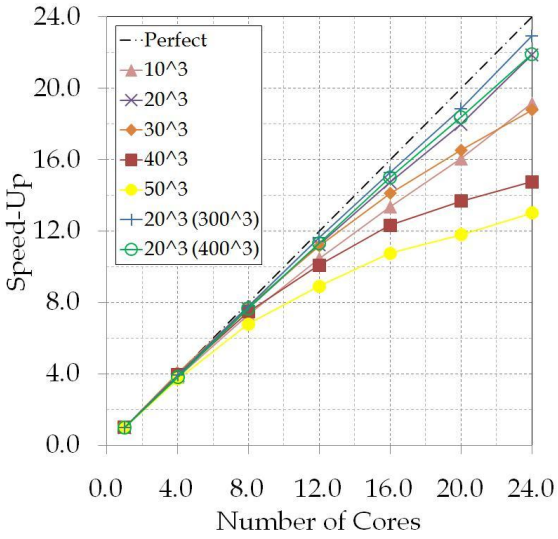

Figure 4. Multi-core, parallel performance of the LBM solver for varying bundle and domain sizes.

The side length, in nodes, of the bundles was varied between 10 and 50 and the difference in performance for each size can clearly be seen. Optimal performance is achieved at a side length of 20, where the speed-up and efficiency are approximately 22 and 92%, respectively, on 24 cores. This bundle size represents the best cache blocking scenario for the tested hardware. As the bundle size is increased, performance degrades due to the computational tasks becoming too large for storage in cache which results in 'slow' communication with RAM for data.

The bundle side length of 20 was then transferred to identical problems using a 300^3 and 400^3 domain, and the results of these tests are also included in Figure 4. As in the smaller problem, near linear scalability is achieved and at 24 cores the speed ups are approximately 23 and 22, and the efficiencies are approximately 95% and 92% for 300^3 and 400^3, respectively. This is an important result, as it suggests that the optimum bundle size for 3D LBM problems can be determined in an *a priori* fashion for a specific architecture.

5. APPLIED PERMEABILITY EXPERIMENT

The permeability of reservoir rocks to single and multiple fluid phases is of importance to many enhanced oil recovery procedures. Traditionally, these data are determined experimentally from cored samples of rock. To be able to perform these experiments numerically would present significant cost and time savings, and therefore it is the focus of the present study.

The multi-core framework, using the SPH solver, was applied to numerically determine the porosity-permeability relationship of a sample of Dolomite. The structural model geometry was generated from X-ray microtomographic images of the sample, which were taken from a 4.95*mm* diameter cylindrical core sample, 5.43*mm* in length and with an image resolution of 2.8μm. This produced a voxelated image set that is 1840×1840×1940 in size. Current hardware limitations prevent the full sample from being analyzed in one numerical experiment, therefore sub-blocks (see the insets in Figure 5) of voxel dimensions 200^3 were taken from the full image set to carry out flow testing and map the porosity-permeability relationship.

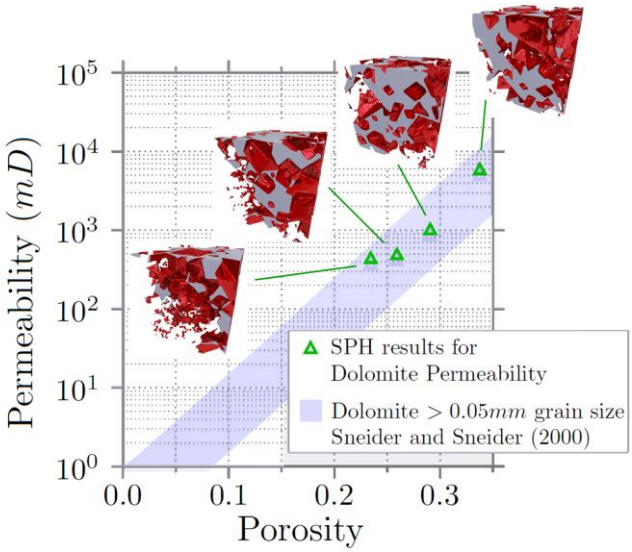

Figure 5. Results of the Dolomite permeability tests undertaken using SPH and the multi-core framework. Experimental data [45] are included for comparison.

The assemblage of SPH particles was initialized with a density of approximately one particle per voxel. Fluid particles (i.e. those located in the pore space) were assigned the properties of water ($\rho_0 = 10^3 kgm^{-3}$, $\nu = 10^{-6} m^2 s^{-1}$) and boundary particles located further than 6h from a boundary surface were deleted for efficiency. The four domain surfaces parallel to the direction of flow were designated no-flow boundaries and the in-flow and out-flow surfaces were specified as periodic. Due to the incompatibility of the two rock surfaces at these boundaries, periodicity could not be applied directly. Instead, the experimental arrangement was replicated by adding a narrow volume of fluid at the top and bottom of the domain. Finally, all simulations were driven from rest by a constant body force equivalent to an applied pressure differential across the sample.

The results of the permeability tests are graphed in Figure 5 and experimental data [45] relevant to the grain size of the sample are included for comparison. It can be

seen that the numerical results lie within the experimental band, suggesting that the presented numerical procedure is appropriate. As expected, each sub-block exhibits a different porosity and by analyzing a range of sub-blocks a porosity-permeability curve for the rock sample can be defined.

6. CONCLUDING REMARKS

In this paper a parallel, multi-core framework has been applied to the SPH and LBM numerical methods for the solution of fluid-structure interaction problems in enhanced oil recovery. Important aspects of their implementation, including spatial decomposition, data structuring and the management of thread safety have been briefly discussed.

Near linear speed-up over 24 cores was found in testing and peak efficiencies of 92% in SPH and 95% in LBM were attained at 24 cores. The importance of optimal cache blocking was demonstrated, in particular in the LBM results, by varying the distributed computational task size via the size of the nodal bundles. This minimized the cache misses during execution and the latency associated with accessing RAM. In addition, it was found that the optimal nodal bundle size in the 3D LBM could be transferred to larger problem domains and achieve similar performance, suggesting an *a priori* technique for determining the best computational task size for parallel distribution.

Finally, the multi-core framework with the SPH solver was applied in a numerical experiment to determine the porosity-permeability relationship of a sample of Dolomite (i.e. a candidate reservoir rock). Due to hardware limitations, a number of 200^3 sub-blocks of the complete microtomographic image of the rock sample were tested. Each sub-block was found to have a unique porosity and corresponding permeability, and when these were superimposed on relevant experimental data the correlation was excellent. This result provides strong support for the numerical experimentation technique presented.

Future work will extend testing of the multi-core framework to 64-core and 256-core server architectures. However, the next major numerical development lies in the extension of the fluid capabilities in SPH and the LBM to multiple fluid phases. This will allow the prediction of the relative permeability of rock samples which is essential to drainage and imbibition processes in enhanced oil recovery.

Acknowledgements
The authors are grateful to the Schlumberger-Doll Research Center for their support of this research.

References

[1] J. H. Walther and I. F. Sbalzarini. Large-scale parallel discrete element simulations of granular flow. *Engineering Computations*, 26(6):688-697, 2009.

[2] A. Ferrari, M. Dumbser, E. F. Toro, and A. Armanini. A new 3D parallel SPH scheme for free surface flows. *Computers & Fluids*, 38(6):1203-1217, 2009.

[3] D. Vidal, R. Roy, and F. Bertrand. A parallel workload balanced and memory efficient lattice-Boltzmann algorithm with single unit BGK relaxation time for laminar Newtonian flows. *Computers & Fluids*, 39(8):1411-1423, 2010.

[4] J. Götz, K. Iglberger, C. Feichtinger, S. Donath, and U. Rüde. Coupling multibody dynamics and computational fluid dynamics on 8192 processor cores. *Parallel Computing*, 36(2-3):142-151, 2010.

[5] M. Bernaschi, L. Rossi, R. Benzi, M. Sbragaglia, and S. Succi. Graphics processing unit implementation of lattice Boltzmann models for flowing soft systems. *Physical Review E*, 80(6):066707, 2009.

[6] T. Zeiser, G. Wellein, A. Nitsure, K. Iglberger, U. Rude, and G. Hager. Introducing a parallel cache oblivious blocking approach for the lattice Boltzmann method. *Progress in Computational Fluid Dynamics*, 8(1-4):179-188, 2008.

[7] M. Herlihy and N. Shavit. *The art of multiprocessor programming*. Morgan Kaufman, 2008.

[8] L. Valiant. A bridging model for multi-core computing. *Lecture Notes in Computer Science*, 5193:13-28, 2008.

[9] K. Poulsen. Software bug contributed to blackout. *Security Focus*, 2004.

[10] J. Dongarra, D. Gannon, G. Fox, and K. Kennedy. The impact of multicore on computational science software. *CTWatch Quarterly*, 3(1), 2007.

[11] C. E. Leiserson and I. B. Mirman. *How to Survive the Multicore Software Revolution (or at Least Survive the Hype)*. Cilk Arts, Cambridge, 2008.

[12] N. Singer. More chip cores can mean slower supercomputing, Sandia simulation shows. *Sandia National Laboratories News Release*, 2009. Available: http://www.sandia.gov/news/resources/releases/2009/multicore.html

[13] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming With the Message-Passing Interface*. MIT Press, Cambridge, 1999.

[14] M. Curtis-Maury, X. Ding, C. D. Antonopoulos, and D. S. Nikolopoulos. An evaluation of OpenMP on current and emerging multithreaded/multicore processors. In M. S. Mueller, B. M. Chapman, B. R. de Supinski, A. D. Malony, and M. Voss, editors, OpenMP *Shared Memory Parallel Programming*, *Lecture Notes in Computer Science*, 4315/2008: 133-144, Springer, Berlin/Heidelberg, 2008.

[15] G. Chrysanthakopoulos and S. Singh. An asynchronous messaging library for C#. In *Proceedings of the Workshop on Synchronization and Concurrency in Object-Oriented Languages*, 89-97, San Diego, 2005.

[16] X. Qiu, G. Fox, G. Chrysanthakopoulos, and H. F. Nielsen. *High performance multi-paradigm messaging runtime on multicore systems*. Technical report, Indiana

University, 2007. Available: http://grids.ucs.indiana.edu/ptliupages/publications/CCRApril16open.pdf.
[17] D. B. Stewart, R. A. Volpe, and P. K. Khosla. Design of dynamically reconfigurable real-time software using port-based objects. *IEEE Transactions on Software Engineering*, 23:759-776, 1997.
[18] D. W. Holmes, J. R. Williams, and P. G. Tilke. An events based algorithm for distributing concurrent tasks on multi-core architectures. *Computer Physics Communications*, 181(2):341-354, 2010.
[19] L. B. Lucy. A numerical approach to the testing of the fission hypothesis. *Astronomical Journal*, 82:1013-1024, 1977.
[20] R. A. Gingold and J. J. Monaghan. Smoothed particle hydrodynamics: Theory and application to non-spherical stars. *Monthly Notices of the Royal Astronomical Society*, 181:375-389, 1977.
[21] G. R. Liu and M. B. Liu. *Smoothed Particle Hydrodynamics: a meshfree particle method*. World Scientific, Singapore, 2007.
[22] M. Liu, P. Meakin, and H. Huang. Dissipative particle dynamics simulations of multiphase fluid flow in microchannels and microchannel networks. *Physics of Fluids*, 19:033302, 2007.
[23] A. M. Tartakovsky and P. Meakin. A smoothed particle hydrodynamics model for miscible flow in three-dimensional fractures and the two-dimensional Rayleigh-Taylor instability. *Journal of Computational Physics*, 207:610-624, 2005.
[24] A. M. Tartakovsky and P. Meakin. Pore scale modeling of immiscible and miscible fluid flows using smoothed particle hydrodynamics. *Advances in Water Resources*, 29:1464-1478, 2006.
[25] X. Y. Hu and N. A. Adams. A multi-phase SPH method for macroscopic and mesoscopic flows. *Journal of Computational Physics*, 213:844-861, 2006.
[26] J. P. Morris, P. J. Fox, and Y. Zhu. Modeling low Reynolds number incompressible flows using SPH. *Journal of Computational Physics*, 136:214-226, 1997.
[27] D. W. Holmes, J. R. Williams, and P. G. Tilke. Smooth particle hydrodynamics simulations of low Reynolds number flows through porous media. *International Journal for Numerical and Analytical Methods in Geomechanics*, n/a. doi: 10.1002/nag.898, 2010.
[28] J. Liberty and D. Xie. *Programming C# 3.0: 5th Edition*. O'Reilly Media, Sebastopol, 2007.
[29] A. R. Adl-Tabatabai, C. Kozyrakis, and B. Saha. Unlocking concurrency. *Queue*, 4(10):24-33, 2007.
[30] S. Chen and G. D. Doolen. Lattice Boltzmann method for fluid flows. *Annual Review of Fluid Mechanics*, 30:329-364, 1998.
[31] H. Chen, S. Chen, and W. H. Matthaeus. Recovery of the Navier-Stokes equations using a lattice-gas Boltzmann method. *Physical Review A*, 45(8):R5339-R5342, 1992.
[32] X. He and G. D. Doolen. Lattice Boltzmann method on a curvilinear coordinate system: Vortex shedding behind a circular cylinder. *Physical Review E*, 56(1):434440, 1997.
[33] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the Navier-Stokes equation. *Physical Review Letters*, 56(14):1505-1508, 1986.
[34] S. Hou, Q. Zou, S. Chen, G. Doolen, and A. C. Cogley. Simulation of cavity flow by the lattice Boltzmann method. *Journal of Computational Physics*, 118(2):329-347, 1995.
[35] R. Cornubert, D. d'Humières, and D. Levermore. A Knudsen layer theory for lattice gases. *Physica D*, 47(1-2):241-259, 1991.
[36] T. Inamuro, M. Yoshino, and F. Ogino. A non-slip boundary condition for lattice Boltzmann simulations. *Physics of Fluids*, 7(12):2928-2930, 1995.
[37] D. R. Noble, S. Chen, J. G. Georgiadis, and R. O. Buckius. A consistent hydrodynamic boundary condition for the lattice Boltzmann method. *Physics of Fluids*, 7(1):203-209, 1995.
[38] D. R. Noble and J. R. Torczynski. A lattice-Boltzmann method for partially saturated computational cells. *International Journal of Modern Physics C*, 9(8):1189-1201, 1998.
[39] A. J. C. Ladd. Numerical simulations of particulate suspensions via a discretized Boltzmann equation. Part 1. Theoretical foundation. *Journal of Fluid Mechanics*, 271:285-309, 1994.
[40] D. R. J. Owen, C. R. Leonardi, and Y. T. Feng. An efficient framework for fluid-structure interaction using the lattice Boltzmann method and immersed moving boundaries. *International Journal for Numerical Methods in Engineering*, n/a. doi: 10.1002/nme.2985, 2010.
[41] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, and T. Zeiser. Performance evaluation of parallel large-scale Lattice Boltzmann applications on three supercomputing architectures. In SC '04: *Proceedings of the 2004 ACM/IEEE conference on Supercomputing*, 21-33, Washington, 2004.
[42] G. Wellein, T. Zeiser, G. Hager, and S. Donath. On the single processor performance of simple lattice Boltzmann kernels. *Computers & Fluids*, 35(8-9):910-919, 2006.
[43] J. Ma, K. Wu, Z. Jiang, and G. D. Couples. SHIFT: An implementation for lattice Boltzmann simulation in low-porosity porous media. *Physical Review E*, 81(5):056702, 2010.
[44] T. Pohl, M. Kowarschik, J. Wilke, K. Iglberger, and U. Rüde. Optimization and profiling of the cache performance of parallel Lattice Boltzmann codes. *Parallel Processing Letters*, 13(4):549-560, 2003.
[45] R. M. Sneider and J. S. Sneider. New oil in old places. *Search and Discovery*, 10007, 2000. Available: http://www.searchanddiscovery.com:16080/documents/sneider/