Journal of Computational Physics 270 (2014) 577–586

Contents lists available at ScienceDirect

Journal of Computational Physics

www.elsevier.com/locate/jcp

Parallel in time approximation of the lattice Boltzmann method for laminar flows

Amanda Randles [*], Efthimios Kaxiras

Department of Physics and School of Engineering and Applied Sciences, Harvard University, Cambridge, MA 02138, USA

ARTICLE INFO

Article history:
Received 1 April 2013
Received in revised form 1 April 2014
Accepted 3 April 2014
Available online 13 April 2014

Keywords:
Lattice Boltzmann
Fluid dynamics
Parareal algorithm
Parallel in time methods
High performance computing

ABSTRACT

Fluid dynamics simulations using grid-based methods, such as the lattice Boltzmann equation, can benefit from parallel-in-space computation. However, for a fixed-size simulation of this type, the efficiency of larger processor counts will saturate when the number of grid points per core becomes too small. To overcome this fundamental strong scaling limit in space-parallel approaches, we present a novel time-parallel version of the lattice Boltzmann method using the parareal algorithm. This method is based on a predictor–corrector scheme combined with mesh refinement to enable the simulation of larger number of time steps. We present results of up to a 32× increase in speed for a model system consisting of a cylinder with conditions for laminar flow. The parallel gain obtainable is predicted with strong accuracy, providing a quantitative understanding of the potential impact of this method.

1. Introduction

A large amount of work has been devoted to decomposing computational fluid dynamics methods spatially in an effort to enable large-scale simulations using high performance parallel computing. While such methods have been successful (cf. [1–4]), the speedup gained from spatial decomposition will inevitably reach a plateau for a fixed problem size: increasing the spatial discretization will lead to saturation of parallel speedup when the number of lattice points per core becomes too small. Increasingly, large simulations need to be completed on a short time scale in order to make a significant impact on scientific outcomes, making the wall-clock time a crucial component. In such cases, if there are sufficient computational resources, the introduction of temporal parallelization can help overcome the strong scaling limit of spatial parallelization and significantly extend the overall simulated time in a fixed amount of wall-clock time, or reduce the wall-clock time for a simulation of fixed duration. In this paper, we present an efficient method for time parallelization of the lattice Boltzmann method (LBM), one of the most efficient and versatile approaches for simulations of fluid dynamics (cf. [5–7]).

We will focus on the parareal algorithm first introduced by Lions et al. [8], which consists of a coarse iterator calculated serially to initialize data at discrete time steps that is used as the input for a fine grid iterator that runs in parallel. The coarse solver is based on a larger time step and typically a coarser space discretization. Iterative refinement enables the compute-intensive fine iterator to be handled by the temporal parallelization. This method can be viewed as a predictor–corrector scheme in which the coarse solver is used to predict initial values for the full time range and these initial guesses are iteratively refined through application of the fine solver, with the refinement (correction step) completed in parallel.

[*] Corresponding author. Tel.: +1 617 858 1520.
E-mail address: amanda.randles@post.harvard.edu (A. Randles).

http://dx.doi.org/10.1016/j.jcp.2014.04.006
0021-9991/© 2014 Elsevier Inc. All rights reserved.

The algorithm consists of a series of these iterations completing when the results have converged within a certain tolerance and is advantageous only if convergence happens faster than it would have taken for the fine iterator to simply run serially. The parareal algorithm was first applied to plasma fluid dynamics by Fischer et al. in their work on Navier–Stokes for FEM/SEM discretization [9], and has since been shown to be effective for other models of the turbulent flow of plasma [10–12]. Alternative time-parallel methods such as PITA [13] and PFASST [14] have also been applied to study fluid flow and fluid-structure applications. The parareal algorithm has been shown to be effective in a variety of fluid dynamics applications including the development of turbulent flow [9–12]. Here we introduce a method to enable parallel-in-time simulation of the lattice Boltzmann method (LBM) for modeling fluid dynamics, using a multigrid approach to define the coarse and fine iterators. This method therefore reduces the real time duration of simulations and lengthens the bound on time trajectories that can be modeled. The main contributions of this work are: (a) the first application of Parareal to the LBM; (b) a method for using a coarser spatial mesh in the Parareal framework; (c) a quantitative verification of parallel performance gains achievable; and (d) demonstration of a $32\times$ speedup.

The paper is organized as follows: in Section 2 we give the general formulation of the parareal algorithm and in Section 3 we briefly introduce the LBM. The definition of the coarse and fine iterators and necessary coupling mechanism is presented in Section 4, where we also discuss techniques for conserving macroscopic quantities like viscosity across the refinement levels, and the expected performance model. In Section 5, we present the results of LBM with temporal parallelization, showing that we can achieve up to a $32\times$ increase in speed in a model system consisting of a cylinder with conditions for laminar flow. We also provide a quantitative understanding of the potential impact of the method by demonstrating that the parallel speedup can be predicted with strong accuracy. Finally, in Section 6 we provide conclusions and perspectives for future extensions.

2. Parareal algorithm

Lions, Maday, and Turinici (LMT) [8] first proposed the parareal algorithm as a method to decompose an ODE solver in the time domain. The method enables more effective utilization of high performance parallel systems by numerically solving a time dependent system in a shorter overall wall-clock time [15]. Temporal decomposition is achieved through a predictor–corrector scheme consisting of a prediction step, denoted by $\mathcal{G}$, that provides a coarse approximation and a correction step, denoted by $\mathcal{F}$, that contributes a more accurate but more computationally expensive solution. This method alternates between the serial update of initial conditions with $\mathcal{G}$ followed by the parallel application of $\mathcal{F}$ until the corrected values converge within a predetermined tolerance, ε. By computing $\mathcal{F}$ in parallel, a shorter overall runtime is obtained [16]. Inherent to the method is the need for $\mathcal{G}$ to be less computationally expensive than $\mathcal{F}$. The disparity between these two iterators determines the potential speedup of the time parallelization.

In the parareal algorithm, the time interval to be modeled by the simulation is divided into N separate intervals of equal size, $[t_{n-1}, t_n]$, $n = 1 \ldots N$, with n denoting the nth time step. For simulations that are split entirely in the temporal domain, N is set to the number of processors in the system. A different processor is responsible for each of these discrete time intervals. For each interval, a succession of approximations denoted U_{n+1}^K are calculated where K is the parareal iteration number. For each parareal iteration, K, the following steps are taken [10,12]:

0. ($K = 0$) The coarse prediction step $\mathcal{G}$ is first applied serially on all processors to calculate U_n^0 for the full simulation time, $0, \ldots, t_N$.
1. ($K > 0$) Using the initial value U_n^0, each processor can calculate $\mathcal{F}$ in parallel on its respective time interval, t_{n-1} to t_n. The result is then propagated to the next processor in line.
2. The serial correction step is calculated through:
$$U_{n+1}^{K+1} = \mathcal{G}\big(t_{n+1}, t_n, U_n^{K+1}\big) + \mathcal{F}\big(t_{n+1}, t_n, U_n^K\big) - \mathcal{G}\big(t_{n+1}, t_n, U_n^K\big) \tag{1}$$
Note that the second and third terms on the right-hand side of this expression have been obtained in previous iterations and steps.
3. Convergence is checked through the condition $|U_n^K - U_n^{K-1}| < \varepsilon$ where ε is the predetermined tolerance value. If the difference between the solutions for two successive parareal K iterations is smaller than ε for all time intervals, the parareal cycle completes.

In the work described in this paper, we use a pipelined implementation of the parareal algorithm in which the fine approximation commences as soon as the coarse approximation is available for the time interval to be calculated by each processor, rather than waiting for all processors to complete the $\mathcal{G}$ calculation [17,18]. This way, every processor computes each step of the parareal algorithm as soon as possible, removing the need for processors to remain idle while waiting for all of the other processors to finish calculating $\mathcal{G}$ [17]. Fig. 1 illustrates the method in a diagram for $K = 3$. The blue arrows indicate the wall-clock time (vertical component) for each step of the coarse iterator. In the example shown here, each processor handles the duration of one coarse time step (horizontal component of blue arrow). The red arrows show the wall-clock time (vertical component) required for the fine iterator, as described in Step 1. As will be discussed in more detail later, the fine iterator requires more time steps and a larger number of grid points at each time step, which implies a

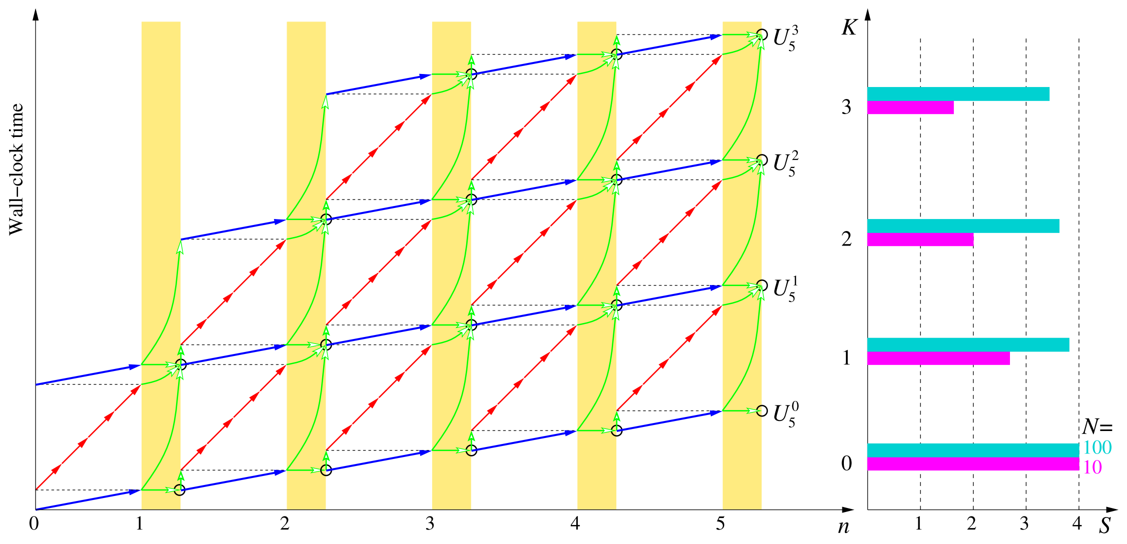

Fig. 1. Computational cost of the pipelined parareal method for $K = 3$: each processor is handling the time duration covered by one coarse time step, as shown along the horizontal axis. The cost of $\mathcal{G}$ in terms of wall-clock seconds per step is shown by the blue arrows and the cost of $\mathcal{F}$ is shown by the red arrows. The green arrows indicate the propagation of the data used in the correction step and the shaded regions correspond to communication between processors. At each black circle, converge tests are conducted. The corresponding speedup for each K value is shown on the right. The magenta bars indicate speedup given 10 processors and the aqua bars show the speedup given 100 processors. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

higher computational cost for each fine step, as indicated by the magnitude and the slope of the red arrows relative to the blue arrows in Fig. 1. The green arrows indicate the propagation of data needed for Eq. (1) in the serial correction described in Step 2. Communication between processors occurs in the yellow shaded regions and is presumed to have negligible computational cost. The convergence tests are conducted at each black circle by comparing the result at that circle with that of the circle vertically below it, as described by Step 3.

In order to gain insight into the performance that can be expected from the use of this technique, we calculate an upper bound on our strong scaling capabilities. To this end, we assume that the hardware system is homogeneous in that each processor is identical and that the communication time between the processors is negligible. The time for one processor to compute one time interval by the fine iterator is denoted by τ_F and similarly, the time to compute one time interval of by the coarse iterator is denoted by τ_C. The number of steps required by $\mathcal{G}$ and $\mathcal{F}$ during the course of one iteration are defined by Q_G and Q_F, respectively, and the total computational cost as γ_G and γ_F. Through use of the pipeline method shown in Fig. 1, each processor proceeds as soon as the initial conditions are available, and so the cost of each K iteration is given by $Q_F\tau_F + Q_G\tau_G = \gamma_F + \gamma_G$. We follow the procedure outlined by Minion [17] to calculate the parallel speedup, S, for pipelined parareal implementations, which is given by

$$S = \frac{N\gamma_F}{N\gamma_G + K(\gamma_G + \gamma_F)} = \frac{1}{\alpha + (K/N)(\alpha + 1)} \tag{2}$$

with N the total number of processors and K the number of parareal iterations, and we have defined $\alpha = \gamma_G/\gamma_F$. This model was used to demonstrate the speedup at different K iterations for both 10 and 100 processor runs in Fig. 1. The potential speedup is strongly influenced by the size of both α and K. To maximize S, both need to be minimized; however, reducing α requires that τ_G be reduced. The only way to reduce the cost of the coarse iterator is to lower the resolution and thereby make $\mathcal{G}$ less computationally intense. This reduction often leads to a less accurate coarse approximation and wider disparity between the two grids. As this disparity increases, the number of K iterations required to converge will consequently increase. Thus, the speedup cannot be increased at will, but requires a careful balance between the ratio of time cost for the fine and coarse steps (α) and the number of iterations (K) required to achieve the desired convergence level.

3. The lattice Boltzmann method

The LBM is an alternative to conventional computational fluid dynamics techniques that has become popular in recent years due to its advantageous handling of flow phenomena in complex boundary conditions, its numerical stability, and the high level of scalability achieved on parallel systems (see, for example, [1–4]). It is based on a minimal form of the Boltzmann equation that recovers hydrodynamic behavior in the limit of small Knudsen numbers. The LBM uses a discretized velocity space that reflects the lattice spacing of a regular Cartesian grid spanning the volume of a 3-dimensional system; the fluid motion is determined by simultaneously solving for the particle distribution function at each grid point [5]. This function, denoted by $f_i(\vec{x}, t)$, describes the probability of finding a fluid particle with discrete velocity $\vec{c}_i$, at lattice point $\vec{x}$, at time step t, and evolves with time Δt through:

$$f_i(\vec{x} + \vec{c}_i \Delta t, t + \Delta t) = f_i(\vec{x}, t) - \omega \Delta t \big[f_i(\vec{x}, t) - f_i^{eq}(\vec{x}, t) \big] \tag{3}$$

where $f_i^{eq}(\vec{x}, t)$ is the equilibrium distribution and ω is the dimensionless relaxation parameter (related to the frequency of particle collisions). The particles move only along discretized velocity paths defined by the lattice. In this work we adopt the popular 19-speed cubic D3Q19 lattice connecting each lattice point to its first and second neighbors [5]. The algorithm consists of two main components: streaming and colliding. The streaming step involves propagating the fluid particles along the appropriate velocity trajectories. The collision step is calculated through a relaxation towards local equilibrium, as shown in the right-hand side of Eq. (3). Here, we use the Bhatnagar–Gross–Krook (BGK) collision operator with a single relaxation time scale [19]. The local equilibrium in terms of the density ρ, average speed $\vec{u}$, and the speed of sound in the lattice c_s, is given by:

$$f_i^{eq} = w_i \rho \left[1 + \frac{\vec{c}_i \cdot \vec{u}}{c_s^2} + \frac{1}{2} \left(\frac{(\vec{c}_i \cdot \vec{u})^2}{(c_s^2)^2} - \frac{u^2}{c_s^2} \right) \right] \tag{4}$$

where w_i are weights determined by the lattice structure and normalized to unity; this expression is the result of a second-order expansion in the fluid velocity of a local Maxwellian with speed $\vec{u}$. A full bounce back method is used at the wall to ensure a no-slip boundary condition. In this case, particles moving from a fluid node to a wall node will reflect back on the discrete velocity being followed [20].

4. The parareal algorithm for the lattice Boltzmann method

We next define our approach to enabling parallel-in-time simulation of the lattice Boltzmann method. To this end, the coarse and fine iterators are defined through use of grids at different resolutions. As discussed in Section 3, the LBM relies on discrete particles moving and interacting on a regular uniform lattice. Different resolutions of the simulation are achieved through the use of different size grid discretization levels, which define the coarse and fine grid iterators. The use of regular grids in which the local connectivity remains unchanged but the lattice spacing itself is refined is not new to the LBM. Typically, grids of different spacing have been used to produce multiscale models with finer resolution at specific regions in the fluid domain, such as at the walls or boundaries. Research has also shown the success of locally embedded grids for multigrid schemes modeling high Reynolds number flows around objects (cf. [21–23]). Our definition of the two different grid levels to enable temporal parallelism builds on previous work using mesh refinement [24]. Other factors that need to be taken into account include the time stepping mechanism and continuity of the kinematic velocity across the grids. These are each discussed in detail in the following.

Filippova and Hänel [22] developed a node-based approach for mesh refinement that relies on a hierarchical refinement of coarse and fine grids for the LBM. This work was extended to enable the use of fewer time steps on refined grids without any degradation in accuracy in space or time [25]. In the present approach, we leverage such a two-level hierarchical grid refinement strategy in which a coarse grid covers the entire spatial domain and a finer grid is superimposed. Grid refinement is implemented by dividing the spatial discretization by a refinement factor m. Fig. 2 shows a refinement in which the coarse grid is half the resolution of the fine grid, with $m = 2$. The red dashed lines indicate the placement of the fine grid while the blue lines show the coarse grid. Fig. 2 also depicts the lattice points encompassed by the simulation to emphasize the overlapping node-based method. The fine iterator uses all grid points (colored blue and red) and the coarse iterator only simulates the flow at the blue lattice points.

In contrast to conventional multigrid methods, we cover the entire spatial domain with the grids for the two different iterators and the fluid motion on each is modeled separately. Furthermore, the key challenge for multigrid and adaptive mesh refinement implementations is the handling of advection from one grid resolution to the other. However, due to the fact that the grids span the full spatial system, fluid particles are not streaming from one grid spacing to another between time steps for the introduced time-parallel technique. Conversely, the hurdle we had to overcome lies in the propagation of initial conditions from one grid resolution to the other and subsequently the coupling between the grids.

There are two key operations needed to address this challenge and coordinate between grid hierarchies: coarsening and interpolating. In order to transfer information from the fine grid to the coarse grid, a coarsening process must take place in which the distribution function at each velocity for the fine grid is averaged to pass the data to the coarse grid. This operates on conserved values and introduces no further truncation error. In order to move in the opposite direction, an interpolation method is required to transfer information from the coarse grid to the fine grid, that is, to fill in the data for the new lattice sites required by the fine iterator.

Within the two grid schemes, the time steps required for a simulation with each grid differ in size, resulting in a different number of time steps to be modeled by each iterator. In addition, each iterator relies on different relaxation parameters. Specifically, the modeling of fluid motion on the coarse grid requires the use of large time steps while modeling of the fluid motion on a fine grid relies on many smaller time steps. The size of a time step, dt, is related directly to the square of the grid resolution, dx in the LBM: $dt = dx^2(\nu/\nu_o)$ where ν is the viscosity of the fluid in lattice Boltzmann units and ν_o is the viscosity in terms of physical units (m^2/s). There is therefore a connection between the size of the time step and the grid

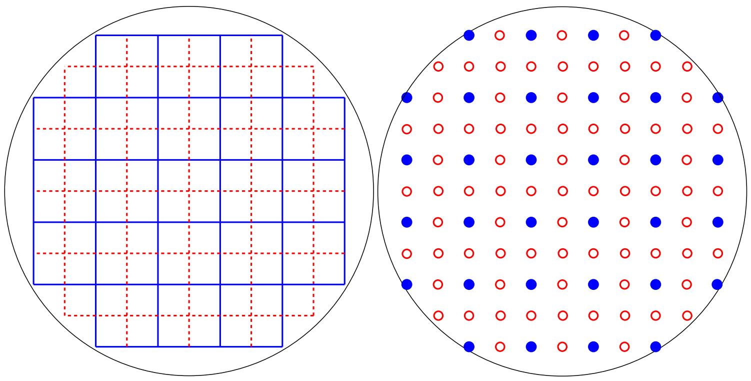

Fig. 2. Left panel: a two level grid where the fine grid is represented with red dashed lines and the coarse grid with solid lines. The fine grid in this example has twice the resolution of the coarse grid, with $m = 2$. Right panel: the lattice points highlighted to demonstrate the overlapping method used. The fine iterator applies to each grid point, both red and blue, whereas the coarse iterator applies only to the blue ones. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

resolution which cannot be circumvented for LBM simulations. The number of time steps calculated by the coarse and fine operators is determined by the size of each respective time step.

We developed a local second-order refinement solution for coupling between the grids that relies on different relaxation parameters and lattice spacing to transition between the grids as described in [22]. The kinematic viscosity in the LBM is defined by $\nu = c_s^2(\frac{1}{\omega} - \frac{dt}{2})$ where dx is the lattice spacing and ω is in dimensionless lattice Boltzmann units. For multigrid methods, the relaxation parameter ω in Eq. (3) must be rescaled to keep the viscosity constant across both the coarse and fine grids [22]. Here, we use the modification introduced by Dupuis and Chopard [26] that removes the potential singularity from the initial formulation and redefines ω as:

$$\omega_f = \frac{\Delta x_c}{\Delta x_f}\left(\omega_c - \frac{1}{2}\right) + \frac{1}{2},$$

where Δx_c and Δx_f are the spatial discretization size for the coarse and fine grids, respectively, and ω_c and ω_f are the corresponding relaxation parameters. This ensures that the simulation remains stable and introduces an upper bound on N by requiring ω_c to remain close to 2 and ω_f to be greater than 1 [22]. The use of this modified definition of ω for each grid imposes a finite limit on the disparity between the two iterator's resolutions.

The parareal algorithm applied to the LBM involves the following steps in which each processor, n, computes the simulation for time domain $[t_{n-1}, t_n]$:

0. ($K = 0$) Initialize the coarse grid with a serial LBM simulation for the time domain $[t_{n-1}, t_n]$ for each processor; interpolate to initialize the fine iterator.
1. ($K = K + 1$) Each processor separately applies $\mathcal{F}$ starting with the initial values provided by the previous iteration for t_{n-1} to determine the distribution function at t_n for its respective interval of time. This is completed in parallel and the result is shared with the next processor. As a serial process, $\mathcal{G}$ is applied.
2. The correction to $\mathcal{F}$ is calculated via Eq. (1); the result is coarsened to update the initial conditions for $\mathcal{G}$ and propagated to the next processor.
3. Convergence is checked: if all intervals of time have converged, exit the cycle; else, return to Step 1.

The key components to this scheme are the steps required to link the two different grid resolution levels in the *interpolate* and *coarsen* steps. We note a few important points: errors are introduced near the walls at boundaries as the initial conditions are taken from the coarse grid. When a boundary is interpolated, fluid cells are introduced that did not exist in the coarse grid and have no fluid cell from which to draw initial data. In this case, the average of all surrounding fluid cells is used to initialize the distribution function at this point. With each K-cycle, this approximation is updated from previous results on the fine grid, reducing the error at the boundaries over time. While the primary source of error is the difference between the coarse and fine grid resolutions sizes, the lack of data feeding the linear interpolation at the boundaries impact the rate of convergence of the coarsening scheme. We define convergence by requiring the average relative error of the solution in the Kth iteration of f_i to differ from the solution of the $(K + 1)$th iteration of f_i by an amount less than a prescribed tolerance ε [12].

5. Results

The limiting factor in previous computational models has been the total extent of real time that can be simulated on current hardware. To gain a better understanding of the role parallelization-in-time plays on scalability and runtime, we

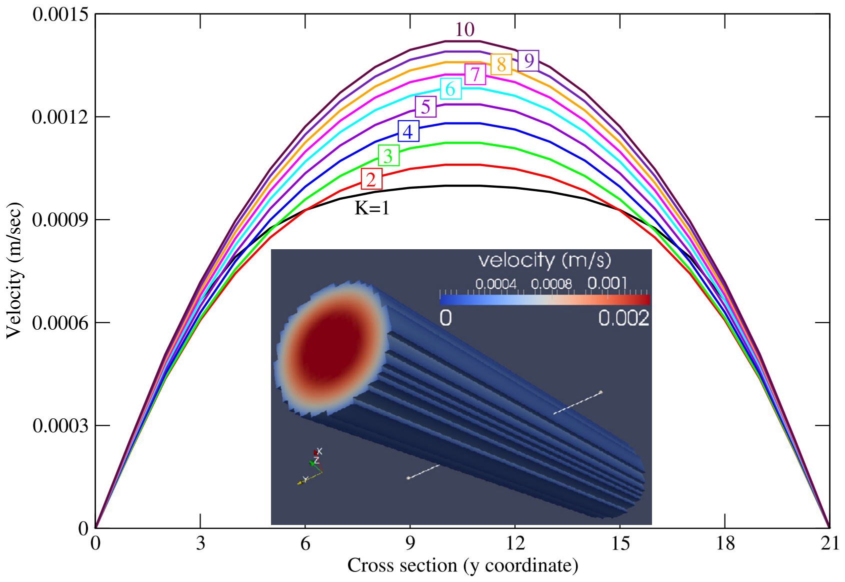

Fig. 3. The magnitude of the velocity across the y-axis for a system broken into $N = 10$ temporal domains simulated on ten processors. The lines are labeled and color-coded according to the K value, running from $K = 1$ to $K = 10$ (converged). The inset shows the 3-dimensional velocity profile of the tube with the white line indicating the cross-section along which the velocities are plotted. (For interpretation of the references to color in this figure, the reader is referred to the web version of this article.)

undertook a series of tests to study both the implications on the accuracy of the simulation as well as on the parallel efficiency.

5.1. Small-scale accuracy study

To study the accuracy of the results at different K iteration levels, we modeled fluid flowing through a straight cylindrical tube of 1 cm in length and 1 mm in diameter. The flow was subject to a constant velocity at the inlet of the tube and fixed pressure gradient at the outlet. We set up the simulation with a 100 μm resolution for the coarse iterator and a 50 μm resolution for the fine iterator. For the fine iterator, this corresponds to a cylinder with a height of 100 lattice points and a diameter of 10 lattice points. A bounding box of dimensions $10 \times 10 \times 100$ lattice points is used to encapsulate the fine grid. This corresponds to 1200 time steps for the coarse iterator and 4800 time steps for the fine iterator, for the simulation of 0.5 s of flow.

In Fig. 3 we show the velocity profile on a cross-section of the system at $x = 5$ and $z = 50$. The units in this case define the lattice point coordinates of the cross section. The inset of Fig. 3 shows the tube with the white line indicating the cross section used for the velocity plot. The velocity of the fluid is calculated from $(1/\rho)\sum_i f_i(\vec{x},t)\vec{c}_i$. The deep purple line shows the result of the simulation at $K = 10$ which is the correct result of the fine iterator as this simulation used $N = 10$ temporal domains simulated on ten processors. The black line shows the velocity estimate at $K = 1$ and highlights the disparity between the initial estimate and the correct result.

Full convergence with machine accuracy to the $\mathcal{F}$ solution requires $K = N$ iterations when using N processors. The results presented and discussed in this section are intended to show that convergence within a set tolerance can be achieved with fewer K iterations than the number of processors ($K < N$). In the initial study, ten 2.66 GHz Intel Xeon X5650 cores were used. To assess the impact of the parareal algorithm, the value of the velocity at the center of the cylindrical tube, corresponding to lattice point $(5, 5, 50)$, at each K iteration was compared. This point was selected as it represents the magnitude of the velocity at the center of the tube, which is the peak of the parabolic cross-section profile. The error was assessed through comparison with the result of one serial run of the fine iterator, $\mathcal{F}$. The results are shown in Fig. 4(a).

When simulating the fluid flow using simply the coarse iterator, the results have an error of nearly 30%, and with six iterations of the parareal predictor–corrector method, this error is reduced by two-thirds to less than 10%. The impact of each iteration diminishes as K increases and the results converge toward the $\mathcal{F}$ solution. This is demonstrated in Fig. 4(a) as the difference in the error reduces as K increases. For example, two iterations reduces the error by 5% compared to only one iteration whereas nine iterations only reduces the error by 2% as compared to eight iterations. Fig. 4(b) shows the impact of each K-level on the overall wall-clock time using a temporal break up of only 10 domains ($N = 10$). The results show a linear increase in overall runtime as more K iterations are included in the simulation, indicating that the ability to restrict to few K iterations while remaining in an acceptable convergence tolerance can lead to a large improvement in time-to-solution. For example, assuming that an error of 10% is acceptable, which corresponds to $K = 6$, gives a time to solution of 140 s, about a 30% gain compared to the time to calculate the fine iterator serially which was 197 s.

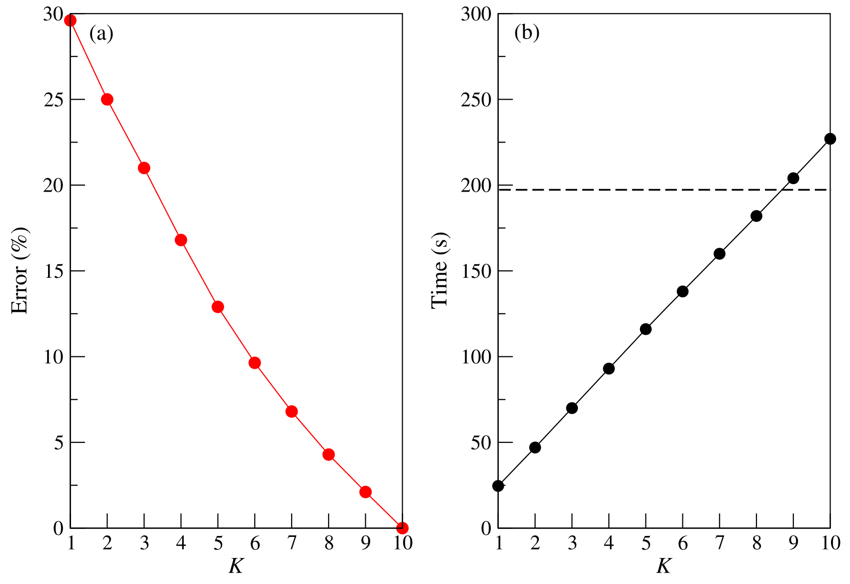

Fig. 4. Accuracy test for a system broken into $N = 10$ temporal domains simulated on ten processors. (a) The red points show the percent error of the y-component of the velocity at point $(5, 5, 50)$. (b) The wall-clock time for each K level in a ten processor run. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

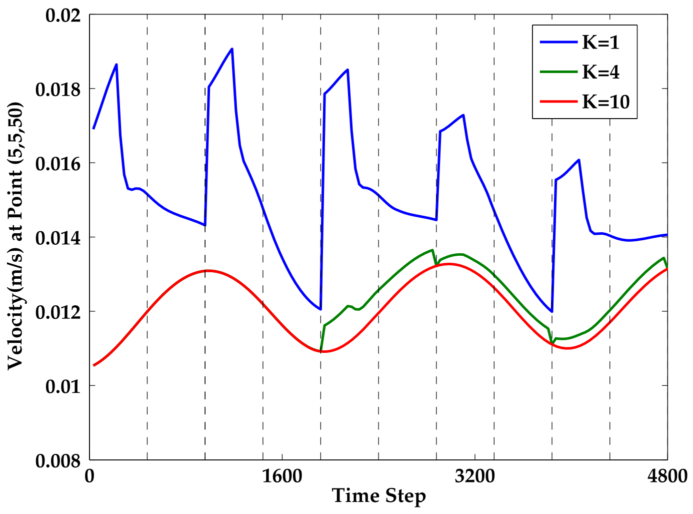

Fig. 5. Test to recover time dependent phenomena for a system broken into $N = 10$ temporal domains simulated on ten processors. The blue line shows the magnitude of the velocity over time at point $(5, 5, 50)$ after the first K iteration. The green and red lines represent $K = 4$ and $K = 10$ respectively. The vertical dashed lines indicate the break point between regions of time handled by each processor. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

5.2. Time-dependent phenomena study

In order to determine the success of recovering time dependent phenomena in the simulation, pulsatile flow through the cylinder was modeled. A varying inflow velocity was introduced via the Zou–He boundary conditions. In this instance, the inlet nodes are provides a set velocity and a constant pressure gradient is asserted at the outlet nodes [27]. The inbound velocity was set to oscillate on a sine wave between 0.011 m/s and 0.014 m/s. Fig. 5 shows the magnitude of the fluid velocity at point $(5, 5, 50)$ for a range of K levels. The regions of time handled by each processor are indicated by the vertical dashed lines.

5.3. Large-scale performance study

To assess the performance of our method on large systems, we supplemented our studies with a larger test domain of $200 \times 800 \times 200$ using an IBM Blue Gene/P system with 2048 processors. This is a distributed memory system that allowed

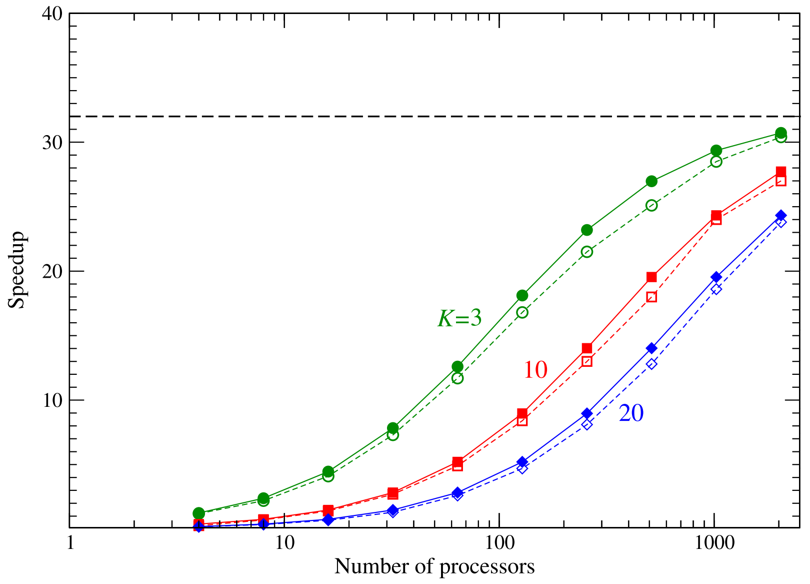

Fig. 6. Performance tests: The solid lines indicate the theoretical speedup and the dashed lines depict the simulation results on the IBM Blue Gene/P system with 2048 processors, demonstrating the strong correlation between the theoretically expected performance from Eq. (2) and experimental results. All processors are used in the temporal direction.

us to assess the impact of the communication on performance. The time-parallelism was implemented using non-blocking MPI communication protocols to maximize the overlap of the computation of each K-level and the communication between iterations. In LBM schemes that are spatially decomposed, communication is required with all nearest neighbors after every time step. With the algorithm proposed here, many time steps are completed between communication calls. At the end of each K-level, each core only communicates with the two cores handling the temporal regions before and after the region handled locally. This reduces the number of messages and the regular pattern allows optimal topology mapping to reduce potential latency issues.

Such a large-scale system allowed us to determine how close our method comes to meeting the theoretical performance prescribed by Eq. (2). Furthermore, these studies identified the peak potential speedup we could achieve for given K iteration levels. The goal of this work was to shorten the overall time-to-solution, so we focus here on the strong scaling capabilities in which a fixed system size is used as we increase the number of processors.

Fig. 6 shows the correlation between the theoretical performance model previously discussed for speedup, Eq. (2), and the experimental results of the simulation. The decomposition is entirely in the temporal domain. The data shows that for given number of K-iterations, we can accurately estimate the potential speedup S for varying processor counts. The calculations are based on the coarse and fine iterators defined previously ($\Delta x_f = 50$ μm and $\Delta x_c = 100$ μm resolution). These resolutions determine the value of the ratio $\alpha = \gamma_G/\gamma_F$ as they dictate the computational cost of both $\mathcal{G}$ and $\mathcal{F}$. They were chosen to ensure the stability of the simulation. A change to either grid size would inherently impact α and the associated speedup S to be gained.

Since the theoretical model of Eq. (2) reproduces the data very well, we also investigated the limit on the potential speedup to be gained through the use of our method. Fig. 6 shows the theoretical speedup for a range of K values up to 20,000 processors. For every K iteration level, the potential speedup reaches a plateau at an upper bound value.

As Eq. (2) demonstrates, the potential speedup is limited by $\frac{1}{\alpha}$. With the iterators prescribed in this study, the coarse iterator is defined with a grid resolution of one half the resolution of the grid used by the fine iterator. This variance corresponds to the fine iterator having four times as many time steps as the coarse iterator. Taking account the factor of 2 difference in all spatial dimensions combined with the factor of 4 difference in the number of time steps, the limit to the cost ratio, α, of the coarse to fine iterator is 32. Fig. 6 shows both the experimental and theoretical data confirming a plateau at a 32× speedup. The specific K for each simulation will depend on the chosen convergence tolerance, ε, which can vary in each problem. In all cases, the potential speedup reaches a limit, and beyond this point, the parallel efficiency will begin to drop dramatically.

6. Conclusions and discussion

In this work, we introduced a time-parallel method to speedup fluid simulations based on the lattice Boltzmann method, thus enabling the modeling of more time steps in a shorter wall-clock time. Previously, a limiting factor to parallel efficiency of simulations of a fixed problem size was the saturation of spatial parallelization. While LBM is well suited to large-scale spatial parallelism, parallel efficiency starts to decline as the number of mesh points per processor becomes too low. Typically, in these simulations there is a fixed resolution requirement in which further discretization of the space will not yield better accuracy. In such cases, when the limits of spatial parallelism have been reached, the method introduced here to parallelize the LBM in time overcomes this intrinsic strong scaling limit.

In order to use a time parallel LBM code on large scales efficiently, carefully designed coarse and fine iterators are needed. We showed that this goal can be met by means of the multigrid approach. By combining mesh refinement methods with the parareal algorithm, we were able to study longer time intervals of a fluid simulation in a shorter wall-clock time, within which, through iterative refinement, the compute-intensive fine iterator is modeled with temporal parallelization. Thus, the novel combination of mesh refinement and the parareal algorithm presented in this work provide a method to considerably extend the intrinsic strong scaling limit of classical LBM codes and therefore minimize the runtime of fixed-size fluid simulations.

Finally, we comment on some issues related to future extensions and further development of the method. The implementation described here requires that processors that have achieved convergence with the fine iterator remain idle until the solution for the entire time domain has converged within the desired tolerance. The parallel efficiency of this method can be improved through reuse of these processors. One such option would be to re-allocate their use for further spatial discretization of remaining time intervals. Another subtle point is that if a grid point is defined as a boundary node on the coarse mesh, it will translate to a wall plus fluid nodes in the fine mesh. The value of the fluid quantities near the wall will be inaccurate as they are initialized to zero due to the averaging from the coarseness. The additional fluid nodes encompassed by the fine grid have contributions from coarse wall nodes in their initialization. The contribution from the wall nodes lowers the real distribution value seen by the fluid at the wall. Through each K-iteration, the value will be refined and propagated to the mesh. After the first iteration they will be based on values from the first fine full iteration and propagated through the system. This may take more time but only impacts the few lattice points directly next to the wall in the fine grid. In an extremely porous material, this could be an issue but otherwise (as in the case of the model system presented here) it does not have serious adverse effects. For porous materials, we suggest that an interpolation function should take nearby fluid data to initialize the conditions instead of using the zero contribution from the wall.

Acknowledgements

This work was supported by the Department of Energy's Computational Science Graduate Fellowship, grant number DOE CSGF DE-FG02-97ER25308. We wish to thank M. Minion for comments and D. Singer and F. Mintzer for their assistance and support regarding access to the Blue Gene/P system at the IBM T.J. Watson Research Center.

References

[1] A. Peters, S. Melchionna, E. Kaxiras, J. Lätt, J. Sircar, M. Bernaschi, M. Bison, S. Succi, Multiscale simulation of cardiovascular flows on the IBM Blue Gene/P: full heart-circulation system at red-blood cell resolution, in: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '10, IEEE Computer Society, 2010.
[2] J. Carter, M. Soe, L. Oliker, Y. Tsuda, G. Vahala, L. Vahala, A. Macnab, Magnetohydrodynamic turbulence simulations on the Earth simulator using the lattice Boltzmann method, in: Proceedings of the 2005 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '05, IEEE Computer Society, 2005.
[3] S. Williams, L. Oliker, J. Carter, J. Shalf, Extracting ultra-scale lattice Boltzmann performance via hierarchical and distributed auto-tuning, in: Proceedings of the 2011 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, IEEE Computer Society, 2011, pp. 1–10.
[4] T. Pohl, F. Deserno, N. Thurey, U. Rude, P. Lammers, G. Wellein, T. Zeiser, Performance evaluation of parallel large-scale lattice Boltzmann applications on three supercomputing architectures, in: Proceedings of the 2004 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis, SC '04, IEEE Computer Society, 2004.
[5] S. Succi, The Lattice Boltzmann Equation for Fluid Dynamics and Beyond, Oxford University Press, 2001.
[6] S. Melchionna, M. Bernaschi, S. Succi, E. Kaxiras, F. Rybicki, D. Mitsouras, A. Coskun, C. Feldman, Hydrokinetic approach to large-scale cardiovascular blood flow, Comput. Phys. Commun. 181 (3) (2010) 462–472.
[7] M. Fyta, E. Kaxiras, S. Melchionna, S. Succi, Multiscale simulation of nanobiological flows, Comput. Sci. Eng. 10 (4) (2008) 10–19.
[8] J. Lions, Y. Maday, G. Turinici, A parareal in time discretization of PDE's, C. R. Math. Acad. Sci. Paris, Sér. I 332 (2001) 661–668.
[9] P. Fischer, F. Hecht, Y. Maday, A parareal in time semi-implicit approximation of the Navier–Stokes equations, in: Proceedings of Fifteen International Conference on Domain Decomposition Methods, Springer-Verlag, 2004, pp. 433–440.
[10] J.M. Reynolds-Barredo, D.E. Newman, R. Sanchez, D. Samaddar, L.A. Berry, W.R. Elwasif, Mechanisms for the convergence of time-parallelized, parareal turbulent plasma simulations, J. Comput. Phys. 231 (23) (2012) 7851–7867.
[11] J. Reynolds-Barredo, D. Newman, J. Reynolds-Barredo, R. Sanchez, L. Berry, Modelling parareal convergence in 2D drift wave plasma turbulence, in: International Conference on High Performance Computing and Simulation (HPCS), IEEE, 2012, pp. 726–727.
[12] D. Samaddar, D. Newman, R. Sánchez, Parallelization in time of numerical simulations of fully-developed plasma turbulence using the parareal algorithm, J. Comput. Phys. 229 (18) (2010) 6558–6573.
[13] C. Farhat, M. Chandesris, Time-decomposed parallel time-integrators: theory and feasibility studies for fluid, structure, and fluid–structure applications, Int. J. Numer. Methods Eng. 58 (9) (2003) 1397–1434.
[14] M. Emmett, M.L. Minion, Toward an efficient parallel in time method for partial differential equations, Commun. Appl. Math. Comput. Sci. 7 (1) (2012) 105–132.
[15] L. Berry, W. Elwasif, J. Reynolds-Barredo, D. Samaddar, R. Sanchez, D. Newman, Event-based parareal: a data-flow based implementation of parareal, J. Comput. Phys. 231 (17) (2012) 5945–5954, http://dx.doi.org/10.1016/j.jcp.2012.05.016.
[16] G. Bal, Y. Maday, A parareal time discretization for non-linear PDE's with application to the pricing of an American put, in: Lecture Notes in Computational Science and Engineering, vol. 23, 2002, pp. 189–202.
[17] M. Minion, A hybrid parareal spectral deferred corrections method, Commun. Appl. Math. Comput. Sci. 5 (2010) 265–301.

[18] N. Kok Fu, N. Mohd Ali, Improving pipelined time stepping algorithm for distributed memory multicomputers, Sains Malays. 39 (6) (2010) 1041–1048.
[19] P. Bhatnagar, E. Gross, M. Krook, A model for collision processes in gases, Phys. Rev. Lett. 94 (1954) 511.
[20] R. Zhang, X. Shan, H. Chen, Efficient kinetic method for fluid simulation beyond the Navier–Stokes equation, Phys. Rev. E 74 (2006) 1–7.
[21] S. Succi, O. Filippova, G. Smith, E. Kaxiras, Applying the lattice Boltzmann equation to multiscale fluid problems, Comput. Sci. Eng. 3 (6) (2001) 26–37.
[22] O. Filippova, D. Hänel, Grid refinement for lattice-BGK models, J. Comput. Phys. 147 (1) (1998) 219–228.
[23] O. Filippova, S. Succi, F. Mazzocco, C. Arrighetti, G. Bella, D. Hänel, Multiscale lattice Boltzmann schemes with turbulence modeling, J. Comput. Phys. 170 (2) (2001) 812–829.
[24] M. Berger, J. Oliger, Adaptive mesh refinement for hyperbolic partial differential equations, J. Comput. Phys. 53 (3) (1984) 484–512.
[25] O. Filippova, D. Hänel, Acceleration of lattice-BGK schemes with grid refinement, J. Comput. Phys. 165 (2) (2000) 407–427.
[26] A. Dupuis, B. Chopard, Theory and applications of an alternative lattice Boltzmann grid refinement algorithm, Phys. Rev. E 67 (6) (2003) 066707.
[27] Q. Zou, X. He, On pressure and velocity boundary conditions for the lattice Boltzmann BGK model, Phys. Fluids 9 (6) (1997) 1591–1598.