UPC Architecture for High-Performance Computational Hydrodynamics

Tung T. Vu, Alvin Wei Ze Chew and Adrian Wing-Keung Law

1 Introduction

Computational hydrodynamics (CHD) simulation has become a popular tool to accelerate the evaluation and optimization of engineering applications. Examples include flow simulation within the tight spacers of membrane modules for mitigating fouling tendency and optimizing flow configuration [1–3]. Besides membrane applications, the combination of CHD with other computational tools such as discrete element method (DEM) in porous media-related applications has also been introduced [4, 5]. Typically, a large-scale CHD simulation run would require significant data storage and proper management of the computer architecture. For example, a 100 million two-dimensional (2D) mesh involving three equations (continuity and momentum equations) would result in an approximate 600 million cell information (100 million * 2 * 3) to be managed during each iteration.

The two common computer architectures for large-scale CHD applications are (a) message passing interface (MPI) and (b) Open Message Interface (OpenMP). The communication among the processors in MPI can either be point-to-point or collective [6]. For the former, the data exchange is between two sets of tasks, whereas the latter involves the communication among all CPUs for a given task. Both types of communication can involve blocking or non-blocking methodologies. The blocking method puts the program execution on hold until the message buffer slots within the computer memory are ready, which might incur a significant idle

T. T. Vu · A. W. -K. Law (✉)
Interdisciplinary Graduate School, Environmental Process Modelling Centre (EPMC), Nanyang Environment and Water Research Institute, Nanyang Technological University, Singapore, Singapore
e-mail: cwklaw@ntu.edu.sg

A. W. Z. Chew · A. W. -K. Law
School of Civil and Environmental Engineering, Nanyang Technological University, Singapore, Singapore

P. Gourbesville et al. (eds.), *Advances in Hydroinformatics*, Springer Water,
https://doi.org/10.1007/978-981-10-7218-5_3

time for large number of CPUs. The non-blocking method proceeds on with the program execution and does not wait for the completion of the communication buffer. Thus, the idle time is eliminated but data loss may be incurred. MPI has been posited to be unsuitable for CHD architectures having large number of CPUs and high levels of memory hierarchy [6, 7]. Comparatively, OpenMP utilizes a shared memory architecture and does not require the message passing in MPI, which thus makes it straightforward for application. However, its scalability is restricted especially for industrial-scale flow problems via parallelization of the flow solver [6].

By coupling the PGAS-UPC architecture with the two-step explicit numerical scheme from the Lax–Wendroff family of predictors and correctors, a UPC-CHD model was developed and evaluated on three incompressible, viscous flow cases having moderate flow velocities under laminar conditions, namely (a) Blasius boundary layer, (b) Poiseuille's flow, and (c) Couette's flow. Validation of the implemented numerical scheme was achieved by comparing the three cases with their respective analytical solutions for the given hydrodynamic conditions, which showed good overall agreement. Lastly, we shall show that UPC-CHD performed more efficiently than MPI and OpenMP at their base designs in an SGI UV-2000 server with a maximum of 100 cores in this study.

This paper is structured as follows. In Sect. 2, we describe the employed numerical scheme for resolving the following unsteady-state 2D incompressible CHD flow cases: (CHD case A) Blasius boundary layer (BL) flow, (CHD case B) Poiseuille's plate flow and (CHD case C) Couette's flow. This is followed by the description of the developed PGAS-UPC architecture in Sect. 3. The computational performance of the developed architecture is then compared with that of the shared memory SGI system and distributed memory HPC server in Sect. 4. Lastly, Sect. 5 describes the salient pointers as derived from this work.

2 Numerical Discretization

The governing equations for the viscous, unsteady, and incompressible flow in full conservative form [8–10], with the absence of external body forces, can be expressed in the compact form of Eq. 1.

$$\frac{\partial Q}{\partial t} + \frac{\partial F}{\partial x} + \frac{\partial G}{\partial y} = \frac{\partial G_{Vx}}{\partial x} + \frac{\partial G_{Vy}}{\partial y} \tag{1}$$

where Q is the conservative temporal term, F and G are the convective flux vectors in the x and y directions, respectively, and G_{Vx} and G_{Vy} are the viscous flux vectors in the x and y directions, respectively.

The exact representations of the Q, F, G, G_{Vx} and G_{Vy} [8–10] are as follows:

$$Q = \begin{bmatrix} \rho \\ \rho u \\ \rho v \\ \rho E_t \end{bmatrix}, F = \begin{bmatrix} \rho u \\ p + \rho u^2 \\ \rho uv \\ \rho(E_t u) + pu \end{bmatrix}, F = \begin{bmatrix} \rho v \\ \rho uv \\ p + \rho v^2 \\ \rho(E_t v) + pv \end{bmatrix}$$
$$G_{Vx} = \mu \begin{bmatrix} 0 \\ u_x \\ v_x \\ 2uu_x + vu_y + vv_x \end{bmatrix}, G_{Vy} = \mu \begin{bmatrix} 0 \\ u_y \\ v_y \\ 2vv_y + uu_y + uv_x \end{bmatrix} \quad (2)$$

where ρ is the density of water (kg/m^3), u is the horizontal velocity (m/s), v is the vertical velocity (m/s), p is the pressure (kg/m^2 s), E_t is the energy term (kg m^2/s), μ is the dynamic viscosity of water (kg/m s), u_x is the x-derivative of the u velocity, u_y is the y-derivative of the u velocity, v_x is the x-derivative of u, and v_y is the y-derivative of v.

Considering a representative control volume of a single node in Fig. 1, Eq. 2 is discretized over the control volume as shown in Eq. 3 [9, 10]. All others nodes within the numerical domain undergo the same discretization procedure.

$$\frac{\partial \overline{Q}}{\partial t} + \frac{F_E - F_W}{\Delta x} + \frac{G_N - G_S}{\Delta y} = \frac{G_{Vx,E} - G_{Vx,W}}{\Delta x} + \frac{G_{Vy,N} - G_{Vy,S}}{\Delta y} \quad (3)$$

The convective fluxes (F and G) in Eq. 4 are computed by the Roe scheme coupled with the third-order biased approximations. The viscous terms (G_{Vx} and G_{Vy}) in Eq. 4 are resolved using the second-order central differencing scheme [9, 10].

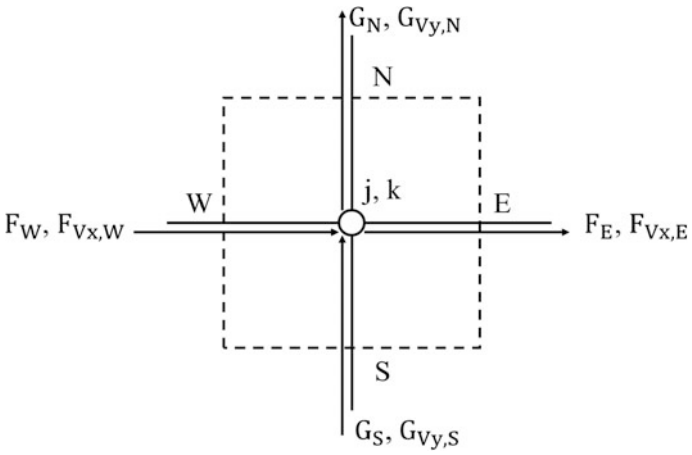

Fig. 1 Representative control volume of a single node for discretizing the viscous, unsteady, and incompressible Navier–Stokes (NS) equations

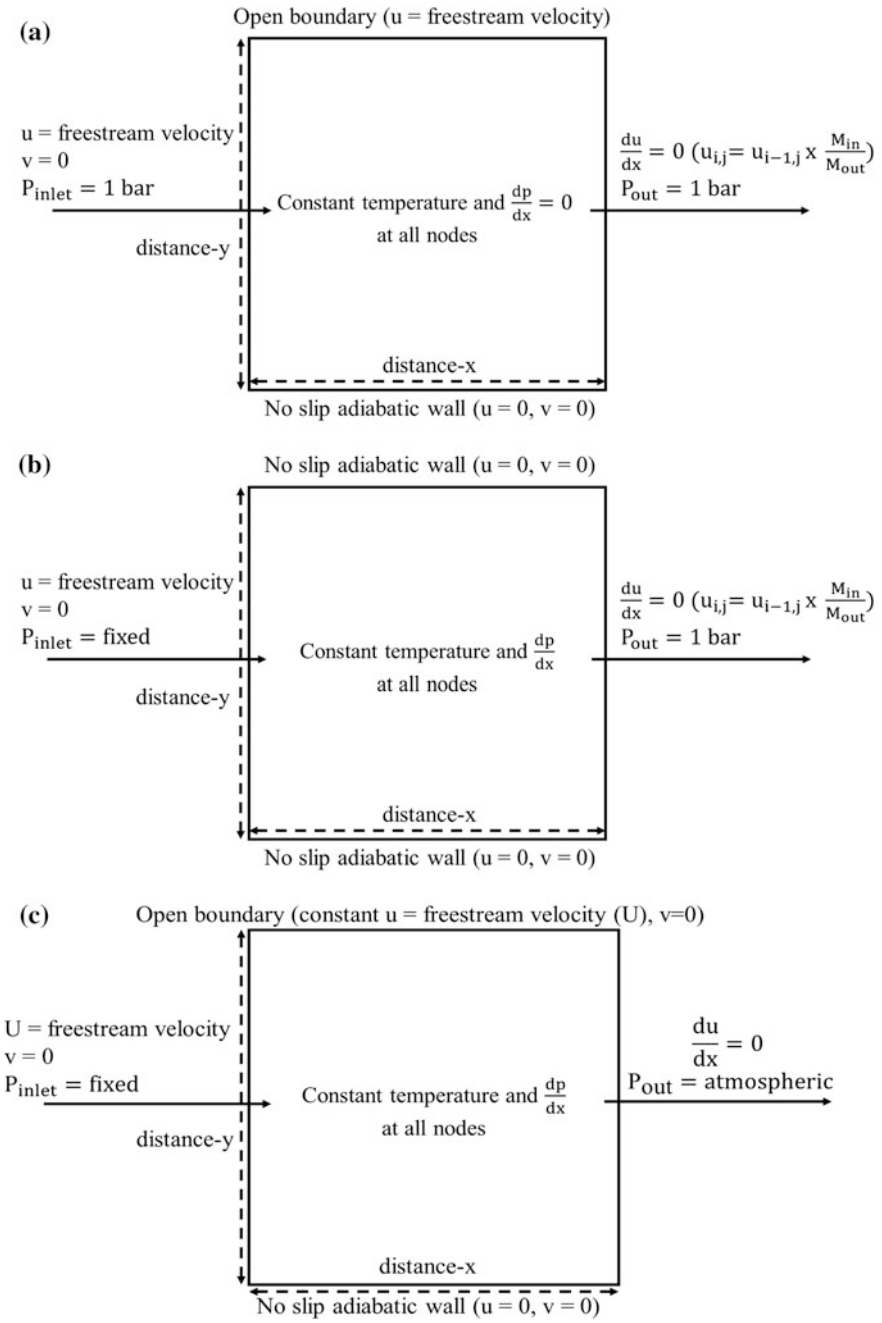

Fig. 2 **a** Numerical domain for Blasius boundary layer (BL) flow (Case A), **b** numerical domain for Poiseuille's plate flow (Case B), **c** numerical domain for Couette's flow (Case C)

For further details, the reader is referred to references [8–10]. The implemented numerical scheme was examined for the three incompressible CHD flow cases (CHD case A–C) by adopting the respective boundary and initial conditions in Fig. 2a–c. Finally, we note that the temperature of all numerical domains was kept at 293.15 K.

3 UPC Implementation of CHD

The UPC-CHD model was first developed by the selected numerical scheme as described in Sect. 2 followed by adopting the following parallelisation procedures: (i) time-consuming functions and different forms of data dependences are first identified, (ii) appropriate algorithms are then adopted for data divisions and storage as based on the data dependences and model workflow, and (iii) lastly the unique work-sharing function of PGAS-UPC has been introduced to parallelize the workflow internally (Fig. 3).

The computational structure of UPC-CHD is summarized as follows. The flux predictor at the $n + 1/2$ time level is first computed and the flux correction at the $n +2$ time level is then computed by repeating the predictor computations with the fluxes from the half-time step as the input data. Both the flux predictor and flux corrector are within the nested loop for which the complexity of the algorithm is identified as $O(N^2)$ where N is the number of nodes in a singular direction. The implemented parallel algorithm in UPC-CHD aims to minimize the total run time of the predictor and corrector fluxes within each cell as both consume the most significant portion of the total computational time. Within the developed model, the original nested loop is first divided into multi-sub-loops to prevent data confliction issue. After every new nested loop, a breakpoint is inserted using a UPC function called *upc_barrier* to synchronize all threads before going the next function.

The computations within the nested loops are distributed using a work-sharing function termed as *upc_forall*. In UPC-CHD, the total number of threads employing the UPC identifier, THREADS whereby each thread can be identified by using another identifier, MYTHREAD. All threads with MYTHREAD from 0 to THREADS-1 run through the identical code (i.e., the nested loop) except for the fluxes computation at the first and last row of each sub-domain respectively. Each thread computes the fluxes on the different sub-domains. We note that this proposed approach is termed as the single-program multiple-data method (SPMD).

To demonstrate the viability of the PGAS concept, we shall now examine the accuracy and performance of the developed UPC-CHD model for the three incompressible CHD flow test cases.

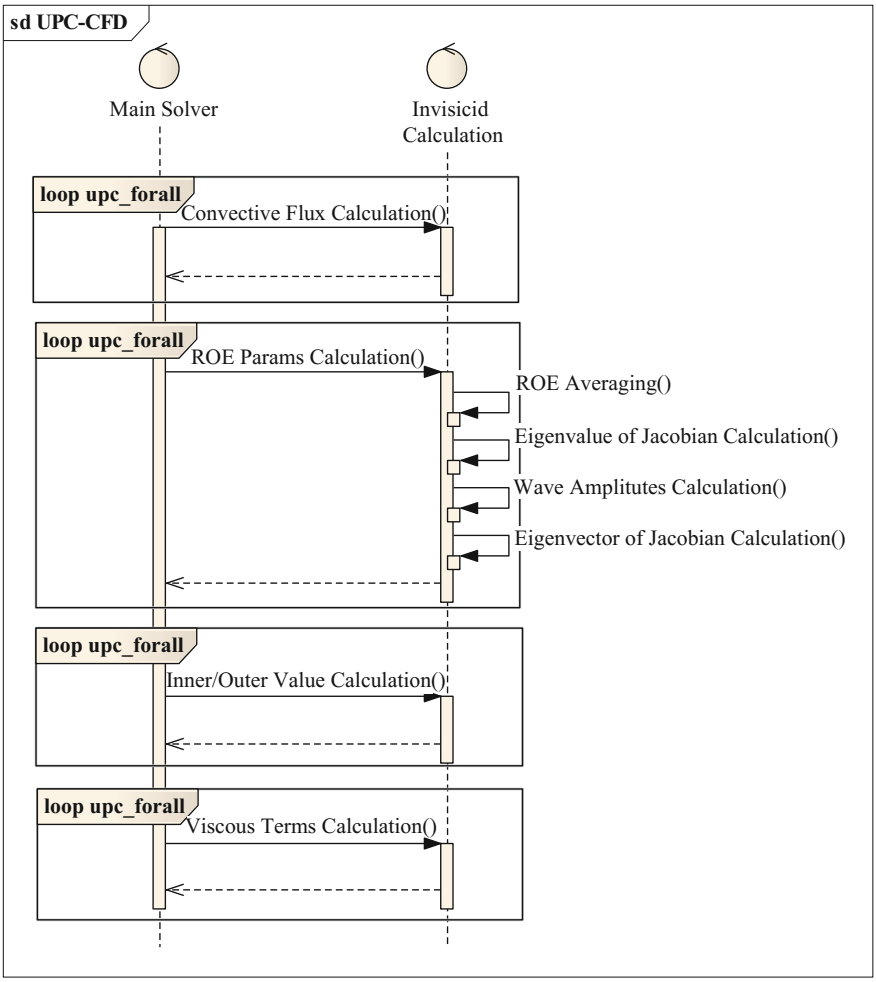

Fig. 3 Parallel computational structure for sub-loops of UPC-CHD

4 Model Verification and Performance Evaluation

The physical dimensions of the deployed numerical domains and the initial flow conditions for Cases A–C are summarized in Table 1. The numerical predictions derived for Cases A–C were compared with the respective analytical solution: (Case A) with the analytical solution of White's [11], (Case B) with Eq. 4 [12], and (Case C) with Eq. 5 [12]. It can be observed from Fig. 4a–c that there is a very good agreement between the obtained numerical values and the respective analytical values for all CHD test cases.

Table 1 Physical dimensions and initial conditions of deployed numerical domains for validating implemented numerical cases in UPC-CHD (Cases A to C)

Parameter	Case A	Case B	Case C
Distance-x, x (m)	0.3	0.5	0.5
Distance-y, h (m)	0.02	0.00016	0.00016
No. of nodes	65 × 65	300 × 45	300 × 45
Freestream velocity, U (m/s)	5	10	10
Re	1,500,000	1600	1600
$\frac{dp}{dx}\left(\frac{Pa}{m}\right)$	0	4.70×10^6	4.70×10^6
delta-t (s)	10^{-6}	10^{-6}	10^{-6}
Total runtime (s)	0.01	0.01	0.01
Temperature (K)	293.15	293.15	293.15
Kinematic viscosity (m^2/s)	10^{-6}	10^{-6}	10^{-6}

$$\frac{u_x}{U} = \frac{\left(y^2 - h^2\right)}{2 * U * \mu} \left(\frac{\partial p}{\partial x}\right) \tag{4}$$

$$\frac{u_x}{U} = \frac{y}{h} - \frac{h^2}{2 * U * \mu} \left(\frac{\partial p}{\partial x}\right) \left(1 - \frac{y}{h}\right) \left(\frac{y}{h}\right) \tag{5}$$

where u_x is the horizontal velocity value obtained (m/s), U is the freestream velocity (m/s), y is the respective y-distance (m), h is the total vertical height of the domain (m), μ is the dynamic viscosity of the fluid in the domain (kg/m s), x is the total horizontal distance of the domain (m), and p is the pressure (kg/m s^2).

With reference to Eq. 6, the parallelism performance of UPC was then compared with that of OpenMP and MPI at their basic designs in an SGI UV-2000 server for Cases A and B having ultra-large numerical domains. The physical dimensions of the deployed numerical domains with its initial conditions for performance evaluation are summarized in Table 2.

$$S(N) = \frac{T(N)}{T_1(N)} \tag{6}$$

where $T(N)$ is the run time of the parallel algorithm, and $T_1(N)$ is the run time of the model which employs a singular core. The configuration of the SG UV-2000 server is summarized in Table 3.

With reference to Figs. 5 and 6, the speedup achieved by UPC, OpenMP, and MPI were near-similar when the number of cores was less than 8. This is because both UPC and OpenMP exploit the advantages of the data locality and are able to read and write directly to the internal memory section without experiencing any delay. The amount of time for querying and retrieving with messages in the MPI approach is trivial when the number of cores used is small. However, beyond 16 cores and up to 100 cores maximum in this study, UPC and MPI outperform

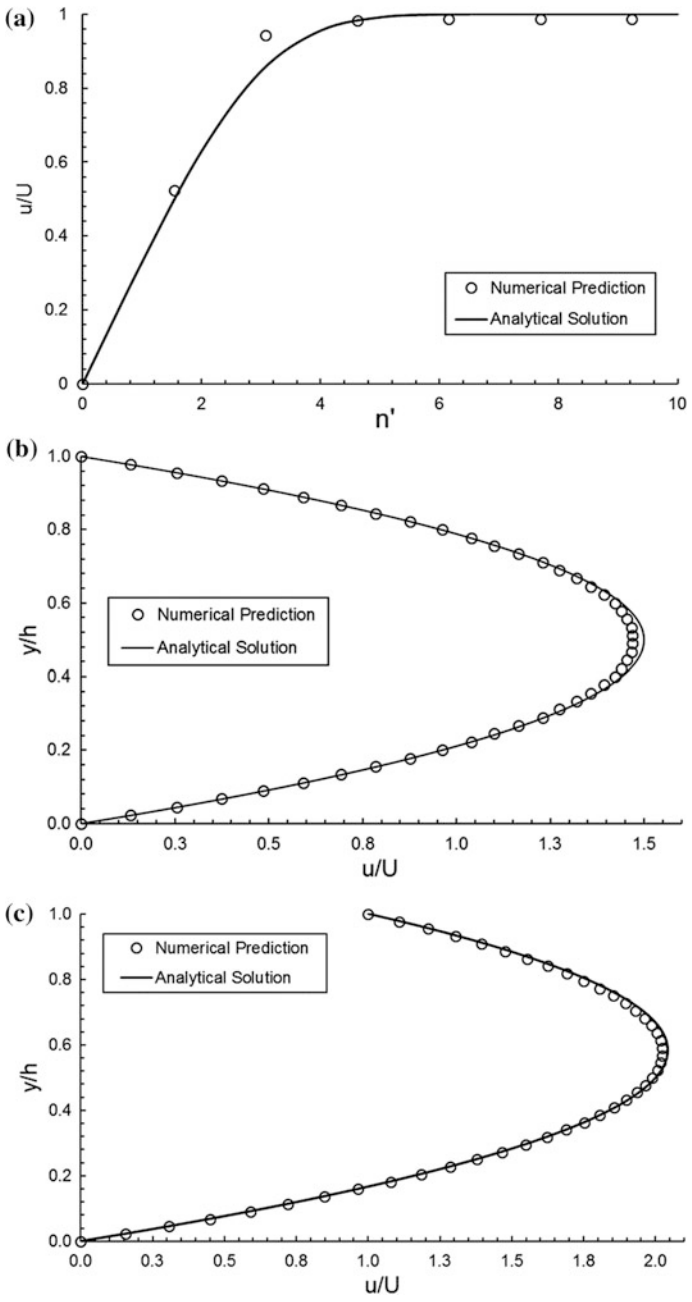

Fig. 4 **a** Comparison between numerical predictions and analytical solutions for Blasius boundary layer flow at location $x = 0.2$ m (Case A) **b** comparison between numerical predictions and analytical solutions for Poiseuille's flow at location $x = 0.5$ m (Case B) **c** comparison between numerical predictions and analytical solutions for Couette's flow at location $x = 0.5$ m (Case C)

Table 2 Physical dimensions and initial conditions of deployed numerical domains for evaluating computational parallelism efficiency of UPC, OpenMP, and MPI in UPC-CHD (Cases A to C)

Parameter	Case A	Case B	Case C
Distance-x, x (m)	0.3	0.5	0.5
Distance-y, h (m)	0.02	0.00016	0.00016
Number of nodes	5000 × 5000	10,000 × 10,000	10,000 × 10,000
Freestream velocity, U (m/s)	5	10	10
Re	1500,000	1600	1600
$\frac{dp}{dx}\left(\frac{Pa}{m}\right)$	0	4.70×10^6	4.70×10^6
delta-t (s)	10^{-6}	10^{-6}	10^{-6}
Total runtime (s)	0.01	0.01	0.01
Temperature (K)	293.15	293.15	293.15
Kinematic viscosity (m^2/s)	10^{-6}	10^{-6}	10^{-6}

Table 3 SGI UV-2000 cluster used for model testing

Cluster	Node CPUs	CPU speeds (GHz)	Cores per node	Node RAM (TB)	Available nodes	Communication switch
SGI UV-2000	Intel Xeon E5-4657LV	2.4	8	2	10 (up to 100 cores)	InfiniBand Shared-memory

OpenMP significantly. In fact, UPC remains effective in terms of the computational speedup with no sign of performance decline after reaching the maximum. In other words, further acceleration can still be achieved if additional computer cores are available.

Beyond 8 cores in Figs. 5 and 6, the performance of OpenMP deteriorats with the deployment of 32, 64, and 100 cores as compared to 8 cores which could be attributed to over-accessing to the shared memory section. The SGI UV-2000 server allows applications to access all available memory in a unified manner as a virtual shared memory block; the memory is still physically located in different nodes which are connected to each other using the network cable. Therefore, when extending the parallel computation to multiple nodes, the access to the shared memory section by the OpenMP approach will be subjected to the communication delay.

With MPI, the speedup is evident but becomes less significant beyond 64 computer cores in Figs. 5 and 6 which reiterates the limitation of MPI. In the MPI architecture, the amount of message passing in the system will augment exponentially with an increasing number of processing cores. Consequently, the total message processing time surpasses the actual computational time on each CPU and obviates any further improvement in the speedup performance. For the studied

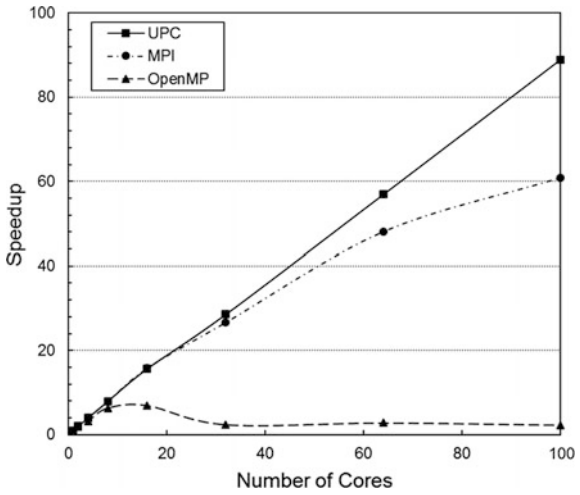

Fig. 5 Comparison of speedup among OpenMP, MPI, and UPC for Blasius boundary layer flow (Case A) at varying number of computer cores

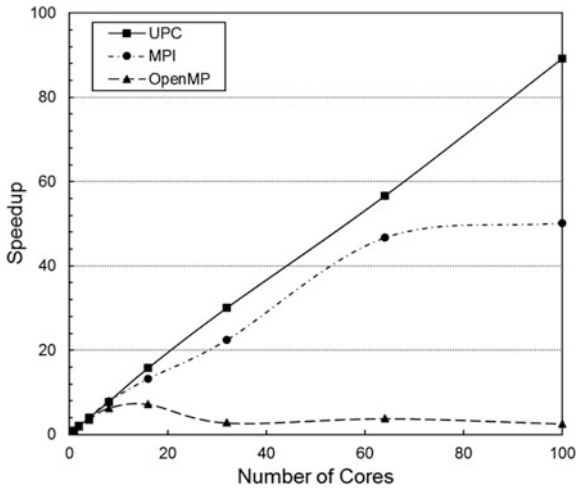

Fig. 6 Comparison of speedup among OpenMP, MPI, and UPC for Poiseuille's flow (Case B) at varying number of computer cores

CHD cases, thread T_i needs to send multi-synchronized messages every time step, which include the following: (1) data of velocity in x- and y- directions to thread T_{i-1} and T_{i+1}, (2) data of computed convective fluxes in x- and y- directions, and (3) updated data to the main thread. At 100 cores, over 900 synchronized messages are needed to be processed in the system for each time step despite having only 100 rows of data to be calculated for each thread. Thus, it is possible that the total message processing time outweighs the actual computational time on each core which restricts the continual speedup with an increasing number of cores.

Finally, we investigated the impact of affinity on the speedup performance in CHD case C. The computational data of the domain is first stored in block to gain the memory locality properties, while the global memory accessing activities are

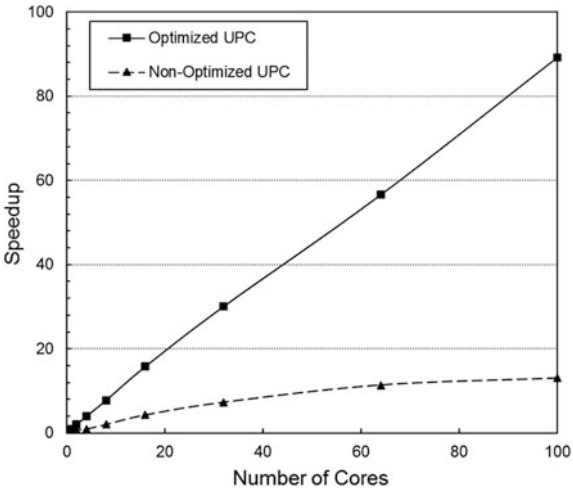

Fig. 7 Comparison of speedup between optimized UPC and non-optimized UPC for Couette's flow (Case C) at varying number of computer cores

overlapped with remote control technique using the split-phase barrier to conceal the synchronization cost. We then evaluated the performance of UPC for Case C under two scenarios: (a) UPC-A, i.e., UPC with optimizations, and (b) UPC-NA, i.e., UPC without optimizations, by employing the defaults setting of the GPAS compilers. With reference to Fig. 7, the performance of UPC-A is superior than that of UPC-NA due to the proper distribution of the array of data in the contiguous blocks with the former.

5 Conclusion

An alternative parallelized computational hydrodynamics (CHD) model with the UPC-PGAS architecture has been developed in this work. First, the accuracy of the proposed model was verified on three incompressible CHD flow cases by comparing with the respective analytical solutions. After which, the model performance was evaluated by comparing the total computational run time with that of both MPI and OpenMP on an SGI UV-2000 server with 100 CPUs. It has been demonstrated that UPC performs more efficiently than MPI and OpenMP with a near linear speedup till 100 CPUs. The performance evaluation underlines UPC's capability in expediting the total run time by exploiting the data locality during parallelism. Finally, we recommend the adoption of the affinity optimization to maximize the parallelism performance of the developed UPC-PGAS architecture.

Acknowledgements This research study is funded by the internal core funding from the Nanyang Environment and Water Research Institute (NEWRI), Nanyang Technological University (NTU), Singapore. The first author is grateful to NTU's Interdisciplinary Graduate School (IGS) for the 4-year Ph.D. scholarship for his study. The second author is grateful to NTU for the 4-year Nanyang President Graduate Scholarship (NPGS) for his Ph.D. study.

References

1. Li, Y. -L., Lin, P. -J., & Tung, K. -L. (2011). CFD analysis of fluid flow through a spacer-filled disk-type membrane module. *Desalination, 283,* 140–147.
2. Sousa, P., Soares, A., Monteiro, E., & Rouboa, A. (2014). A CFD study of the hydrodynamics in a desalination membrane filled with spacers. *Desalination, 349,* 22–30.
3. Bucs, S. S., Radu, A. I., Lavric, V., Vrouwenvelder, J. S., & Picioreanu, C. (2014). Effect of different commercial feed spacers on biofouling of reverse osmosis membrane systems: A numerical study. *Desalination, 343,* 26–37.
4. Sobieski, W., & Zhang, Q. (2017). Multi-scale modeling of flow resistance in granular porous media. *Mathematics and Computers in Simulation, 132,* 159–171.
5. Jajcevic, D., Siegmann, E., Radeke, C., & Khinast, J. G. (2013). Large-scale CFD–DEM simulations of fluidized granular systems. *Chemical Engineering Science, 98,* 298–310.
6. Jamshed, S. (2015). The way the HPC works in CFD. In *Using HPC for computational fluid dynamics* (pp. 41–79). Oxford: Academic Press.
7. Gourdain, N., Gicquel, L., Montagnac, M., Vermorel, O., Gazaix, M., & Staffelbach, G. (2009). High performance parallel computing of flows in complex geometries: I. methods. *Computational Science & Discovery, 2,* 015003.
8. Toro, E. F. (2009). The Riemann Solver of Roe. In *Riemann solvers and numerical methods for fluid dynamics: A practical introduction* (pp. 345–376). Berlin, Heidelberg: Springer.
9. Kermani, M., & Plett, E. (2001). Roe scheme in generalized coordinates. I—formulations. In *39th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics.
10. Kermani, M., & Plett, E. (2001). Roe scheme in generalized coordinates. II—application to inviscid and viscous flows. In *39th Aerospace Sciences Meeting and Exhibit*. American Institute of Aeronautics and Astronautics.
11. White, F. M. (1991). Ch. 7. *Viscous fluid flow* (2nd ed., pp. 457–528). New York: McGraw-Hill.
12. Munson, B. R., Young, D. F., & Okiishi, T. H. (2006). Ch. 6. *Fundamentals of fluid mechanics* (6th ed., pp. 263–331). Hoboken, NJ: Wiley.