High Performance Computational Hydrodynamic Simulations: UPC Parallel Architecture as a Future Alternative

*Alvin Wei Ze CHEW[1], Tung Thanh VU[2], **Adrian Wing-Keung LAW[1, 2]

1 School of Civil and Environmental Engineering, Nanyang Technological University, N1-01c-98, 50 Nanyang Avenue, Singapore 639798

2 Environmental Process Modelling Centre (EPMC), Nanyang Environment and Water Research Institute (NEWRI), 1 Cleantech Loop, CleanTech One, #06-08, Singapore 637141

*Corresponding author's email: wzchew1@e.ntu.edu.sg

**Corresponding author's email: cwklaw@ntu.edu.sg

Abstract. Developments in high performance computing (HPC) has today transformed the manner of how computational hydrodynamic (CHD) simulations are performed. Till now, the message passing interface (MPI) remains the common parallelism architecture and has been adopted widely in CHD simulations. However, its bottleneck problem remains for some large-scale simulation cases due to delays during message passing whereby the total communication time may exceed the total simulation runtime with an increasing number of computer processers. In this study, we utilise an alternative parallelism architecture, known as PGAS-UPC, to develop our own UPC-CHD model with a 2-step explicit scheme from the Lax-Wendroff family of predictors-correctors. The model is evaluated on three incompressible, adiabatic viscous 2D flow cases having moderate flow velocities. Model validation is achieved by the reasonably good agreement between the predicted and respective analytical values. We then compare the computational performance between UPC-CHD and that of MPI in its base design in a SGI UV-2000 server till 100 processers maximum in this study. The former achieves a near 1:1 speedup which demonstrates its efficiency potential for very large-scale CHD simulations, while the later experiences slowdown at some point. Extension of UPC-CHD remains our main objective which can be achieved by the following additions: (a) inclusions of other numerical schemes to accommodate for other types of fluid simulations, and (b) coupling UPC-CHD with Amazon Web Service (AWS) to further exploit its parallelism efficiency as a viable alternative.

Keywords: parallel computing; viscous incompressible laminar flow; MPI; UPC; computational hydrodynamic (CHD) simulations

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

1 Introduction

Computational hydrodynamic (CHD) simulations has become a useful tool for engineers and scientists to accelerate their quantitative understandings of physical, chemical and even biological processes. For example, CHD has been coupled with the multiscale perturbation analysis to numerically resolve for various properties at the varying scales which can be difficult to compute analytically (Dalwadi et al., 2015, 2016; Chang et al., 2017). Other CHD works include flow simulations in tight membrane spacers to better understand the physics of membrane fouling and flow short-circuiting (Jajcevic et al., 2013; Bucs. et al., 2014; Sousa et al., 2014). Typically, a large-scale CHD simulation run with high performance computing (HPC) inclusion requires proper management of the parallelization algorithm to achieve optimization. For example, a 100 million two-dimensional (2D) mesh involving three important equations (continuity and momentum equations only) result in an approximate 600 million cell information to be managed during each iterative step. Data sharing among computer processers, i.e. threads, is unavoidable in CHD mesh-bounded numerical domains which underlines the difficulty to achieve optimization.

At present, the message passing interface (MPI) architecture remains the most popular parallelism option. Examples include the utilisation of MPI with a million cores by Balaji et al. (2009) to thoroughly examine its scalability, and many others. At the same time, difficulties have also been reported when designing MPI applications for considerable number of threads with escalated levels of memory hierarchy (Gourdain et al., 2009; Jamshed, 2015). Thus, the question remains if MPI can effectively accelerate CHD simulations with the availability of computer threads. The answer is complex as it depends on a multitude of factors which include: (a) type of numerical scheme implemented, (b) size of computational domain, (c) type of flow problem analysed, and most importantly (d) the domain decomposition algorithm which determines the distribution of the threads within the numerical domain, i.e. how many sub-domains are assigned to each thread after decomposition. We underline that pointer (d) is most significant in affecting the speedup of CHD simulation runs.

In this study, we adopt an alternative parallelism architecture for CHD simulations by coupling the Partitioned Global Address Space (PGAS) computing concept with Berkeley's Unified Parallel C (UPC) compiler (Chen et al., 2003) as the programming language. Two key advantages are expected with PGAS: (a) locality in the shared memory structure which facilitates the ease of use, and (b) data layout control of MPI's which enables performance scalability. We note that the PGAS architecture has been utilised before (Johnson, 2006 and Simmendinger. et al., 2011). However, to the best of our knowledge, coupling CHD simulations with the PGAS architecture has been limited by far. It is also worth noting that no porting of the various computer algorithms/architectures is carried out in this study. Rather, the above-mentioned advantages are already inherent in the adopted parallelism architecture of PGAS-UPC. The key objective is to evaluate the viability of harnessing PGAS-UPC as an alternative to accelerate large-

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

scale CHD simulations, at least by achieving the same parallelism performance as that of MPI's in its base design.

We first develop UPC-CHD by coupling the PGAS-UPC architecture with the 2-step explicit numerical scheme from the Lax-Wendroff family of predictors and correctors. UPC-CHD is then examined on three incompressible, viscous and adiabatic two-dimensional (2D) flow cases having moderate velocities under laminar conditions. Validation of UPC-CHD is achieved by the good agreement between the respective analytical and predicted values. We then demonstrate how UPC-CHD provides early indication of its parallelism efficiency with 100 computer threads maximum in this study as our first initiation. Finally, this paper is structured as follows. In Section 2, we describe the numerical scheme implemented in UPC CHD. This is followed by describing UPC-CHD development with the adopted PGAS computing concept in Section 3. The parallelism performance of UPC-CHD is then examined in Section 4. Finally, Section 5 describes the salient pointers as obtained from this study.

2 Governing equations

To fully describe the unsteady 2D behaviour of an incompressible viscous fluid under laminar adiabatic flow conditions, we adhere to (1) which compactly conserves the mass continuity, momentum flow and energy equations (Anderson, 2009).

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

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

The exact forms of Q, F, G, G_{Vx} and G_{Vy} are described in (2) as shown.

$$Q=\begin{bmatrix}0\\u\\v\\E_t\end{bmatrix}, F=\begin{bmatrix}u\\\frac{p}{\rho}+u^2\\uv\\(E_t u)+\frac{p}{\rho}u\end{bmatrix}, G=\begin{bmatrix}v\\uv\\\frac{p}{\rho}+v^2\\(E_t v)+\frac{p}{\rho}v\end{bmatrix},$$

$$G_{Vx}=\upsilon\begin{bmatrix}0\\u_x\\v_x\\\frac{u\tau_{xx}+v\tau_{xy}}{\mu}\end{bmatrix}, G_{Vy}=\upsilon\begin{bmatrix}0\\u_y\\v_y\\\frac{u\tau_{yx}+v\tau_{yy}}{\mu}\end{bmatrix} \qquad (2)$$

where u is the horizontal velocity $[LT^{-1}]$, v the vertical velocity $[LT^{-1}]$, $\frac{p}{\rho}$ the pressure divided by the fluid density $[L^2T^{-2}]$, E_t the total energy per unit mass $[L^2T^{-2}]$, υ the

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

kinematic viscosity $[L^2T^{-1}]$, μ the dynamic viscosity $[ML^{-1}T^{-1}]$, τ_{xx} and τ_{yy} the normal stresses $[ML^{-1}T^{-2}]$, and τ_{xy} and τ_{yx} the shear stresses $[ML^{-1}T^{-2}]$.

3 Numerical discretization

To resolve (1) over a 2D numerical domain of regular grids, we adhere to the following numerical schemes (Kermani and Plett, 2001) for the respective terms in their discretized forms: (a) two-step explicit approximation from the Lax-Wendroff family of predictors-correctors for Q, (b) Roe linear approximation with the 3rd-order upwind biased algorithm for F and G, and (c) 2nd-order central differencing for G_{Vx} and G_{Vy}.

The implemented numerical schemes are then examined for the following CHD cases, namely (a) Blasius boundary layer, (b) Poiseuille's flow, and (c) Couette's flow. In UPC-CHD, there are four boundaries of concern for a simplified 2D numerical domain as illustrated in Fig. 1. The exact conditions (BCs) adopted for these boundaries in each CHD case are then described in Table 1.

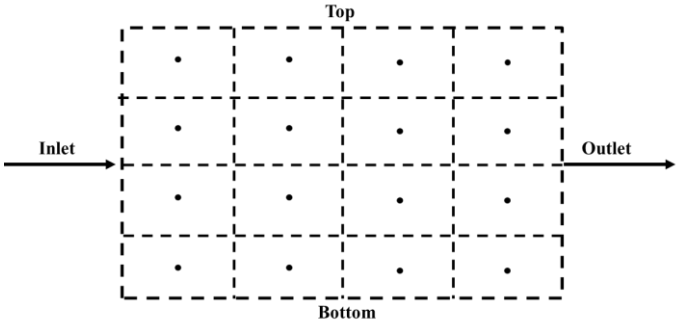

Fig. 1. Boundaries of concern for a simplified 2D numerical domain adopted in UPC-CHD

Table 1. Exact boundary conditions (BCs) implemented in UPC-CHD for respective CHD cases with reference to Fig. 1.

BCs	Blasius boundary layer (CHD case A)	Poiseuille's flow (CHD case B)	Couette's flow (CHD case C)
Inlet	*u = freestream velocity* *v = 0* $\frac{p}{\rho} \approx 98.1 m^2 s^{-2}$	*u = freestream velocity* *v = 0* $\frac{p}{\rho}$ = *fixed value*	*u = freestream velocity* *v = 0* $\frac{p}{\rho}$ = *fixed value*
Outlet	$\frac{du}{dx} = 0$ $\frac{p}{\rho} \approx 98.1 m^2 s^{-2}$	$\frac{du}{dx} = 0$ $\frac{p}{\rho} \approx 98.1 m^2 s^{-2}$	$\frac{du}{dx} = 0$ $\frac{p}{\rho} \approx 98.1 m^2 s^{-2}$
Top	*u = freestream velocity*	*u = 0* *v = 0*	*u = freestream velocity* *v = 0*
Bottom	*u = 0; v = 0*	*u = 0; v = 0*	*u = 0; v = 0*

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

4 UPC-CHD model development

Within UPC-CHD, we introduce the following procedures to achieve our parallelization objective. Firstly, we identify the time-consuming computational functions and data dependences involved. We then implement appropriate algorithms for data divisions and storage as based on the required data dependences and model workflow. There are three components to fully describe UPC-CHD, namely (a) implemented computational structure, (b) domain decomposition and data storage algorithms, and (c) unique work-sharing function.

(a) Computational structure

For each node within the numerical domain (Fig. 1.), the flux predictor at the $(n + 1/2)$ time level is first computed followed by the flux correction at the $(n + 1)$ time level. Both the flux predictor and corrector are defined within a nested loop and the complexity of the nested loop algorithm is defined as $O(N^2)$, where N is the number of nodes in a singular direction. The original nested loop is divided into multi sub-loops to prevent data conflicts. After every new nested loop, a synchronization point is inserted using an UPC function, termed as *upc_barrier*, to synchronize all threads before proceeding to the next function. We note that the fluxes predictions and correctors at the respective time steps contribue to the most time-consuming functions in the algorithm which will be addressed in the following sub-section.

(b) Domain decomposition and data storage algorithms

The 2D numerical domain of N by N size (Fig. 1.) is first divided into a distinct number of computational rows. A defined group of rows then constitutes to a sub-domain having affinity to a computer thread. With T number of threads, we consider the following protocol for domain decomposition: (i) if *N* is divisible by *T*, then there will be $\frac{N}{T}$ sub-domains and each contains the same number of rows, or (ii) if *N* is not divisible by *T*, then the first $int(\frac{N}{T})$ sub-domains contain the same number of rows, the last sub-domain contains the remaining rows. For example, if $N = 8$ and $T = 3$ then the first 2 threads handle the first two sub-domains with three rows each whereas the last thread contains the third sub-domain with two rows. At this moment, UPC-CHD is only considering an ideal square-shaped numerical domain as our first approach. The above-discussed details is most critical to manage the time-consuming functions in the algorithm in terms of the data distribution in the sub-domains which directly affect the total communication time. The following questions remain to be further investigated: (i) ideal number of sub-domains to be deployed with respect to the number of computer threads available, and (ii) the ideal number of cores to be used.

Finally, the respective threads assigned to the first and last sub-domain are termed as $thread_{start}$ and $thread_{end}$, and we note that the domain decomposition is first performed on thread 0. The last thread is termed as $(T - 1)$. The required fluxes

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

$(Q, F, G, G_{Vx}, G_{Vy})$ for each computational node within the domain are computed via a row-by-row method by utilising the respective data values from the two upper and two lower rows. To minimize the communication time involved, each sub-domain is directly affiliated with thread T_i by using the blocked-cyclic technique as illustrated in Fig. 2. We should note that the only exceptions for T_i to access data outside of its assigned sub-domain are restricted to the latter's respective first and last rows.

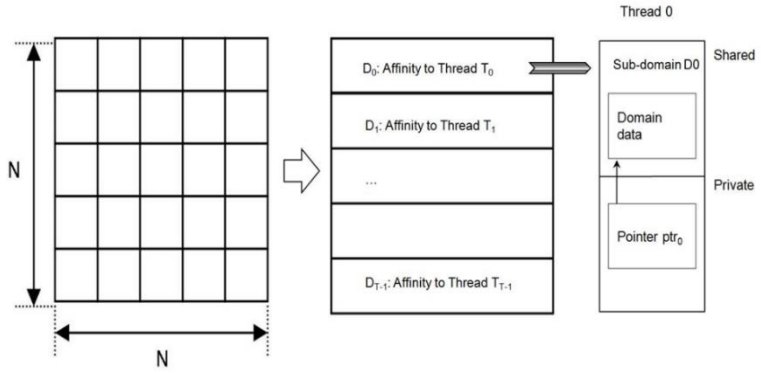

Fig. 2. Simplified illustration of domain decomposition and data storage methodology introduced for a 2D numerical domain of *N* by *N* size; D_i represents sub-domain *i* having affinity to a unique CPU thread

(c) UPC's work-sharing function

In Berkeley's UPC compiler, the computations within the nested loops, as discussed previously in sub-section (a), are distributed using a work-sharing function, termed as *upc_forall*. In UPC, the total number of threads is determined with a UPC identifier, *THREADS*. Each thread is identified by using another identifier, *MYTHREAD*. With *upc_forall*, all threads with *MYTHREAD* from *0* to *THREADS-1* undergo the same computational steps for the fluxes involved. Each unique thread is designed to compute the fluxes on the different sub-domain which is identified by using different $thread_{start}$ and $thread_{end}$.

5 Results and discussions

5.1. Numerical validations

To validate the selected numerical schemes in UPC-CHD, the numerical predictions derived for the three CHD cases are compared with the respective analytical solution: (case A) with analytical solution of White's (White, 1991), (case B) with (3) in the following (Munson et al., 2006), and (case C) with (4) in the following (Munson et al., 2006).

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

$$\frac{u}{U} = \frac{(y^2 - h^2)}{2Uv}\left(\frac{\partial\left(\frac{p}{\rho}\right)}{\partial x}\right) \quad (3)$$

$$\frac{u}{U} = \frac{y}{h} - \frac{h^2}{2Uv}\left(\frac{\partial\left(\frac{p}{\rho}\right)}{\partial x}\right)\left(1 - \frac{y}{h}\right)\left(\frac{y}{h}\right) \quad (4)$$

where U is the freestream velocity $[LT^{-1}]$, h is the total vertical height of the domain $[L]$.

The physical dimensions of the deployed numerical domains and the initial flow conditions adopted for the respective CHD cases are summarized in Table 2. Fig. 3 indicates reasonably good agreement between the predicted and analytical values which validates the implemented numerical scheme in UPC-CHD. We note that the maximum error quantification in Table 2, particularly for CHD case A, can generally be attributed to the following reasons: (i) use of regular meshes for sensitive flow regions whereby further refinements are required, and (ii) possible inaccuracies in the imposed boundary conditions at the top and outlet boundaries for Case A.

Table 2: Dimensions and initial conditions of deployed numerical domains to validate the numerical values obtained for CHD cases A to C

Parameter	Case A	Case B	Case C
$x\ (L)$	0.3	0.5	0.5
$h\ (L)$	0.02	0.00016	0.00016
$N\ x\ N$ (number of nodes)	65 x 65	300 x 45	300 x 45
$U\ (m/s)$	0.05	10	10
Re	15000	1600	1600
$\frac{\partial\left(\frac{p}{\rho}\right)}{\partial x}\ (ms^{-2})$	0	4700	4700
$\Delta t\ (s)$	10^{-6}	10^{-6}	10^{-6}
Total runtime (s)	0.01	0.01	0.01
Temperature (K)	293.15	293.15	293.15
$v\ (m^2s^{-1})$	10^{-6}	10^{-6}	10^{-6}
Maximum error percentage (%)	~25	~1.7	~0.5

5.2. Parallelism performance

We adhere to the speedup (S) parameter in (5) to compare the parallelism performance of UPC-CHD with that of MPI at their basic designs in a SGI UV-2000 server for all CHD cases of very large numerical domains. Technical specifications of the server used are described in Table 3. All other conditions from Table 2 are unchanged with the exception

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

for the N by N parameter for the respective cases: (A) 5000 x 5000, (B) 10000 x 10000, (C) 10000 x 10000.

$$S(c) = \frac{T_1}{T(c)} \quad (5)$$

where $T(c)$ is the run time of the parallel algorithm, T_1 is the run time of the model which employs a singular thread and c is the number of computer threads. We first note that the parallelism evaluation between UPC-CHD and MPI is confined till 100 computer threads maximum in this study due to resources limitations. The key objective is to evaluate the adopted parallelism architecture of UPC-CHD as a viable alternative to that of MPI's in its base design.

Table 3. Technical specifications SGI-UV 2000

Cluster	Node CPUs	CPU speeds (GHz)	Cores per node	Node RAM (TB)	Available cores	Communication switch
SGI UV-2000	Intel Xeon E5-4657LV & E5-2670	2.4	16	2	up to 100 cores	InfiniBand Shared-memory

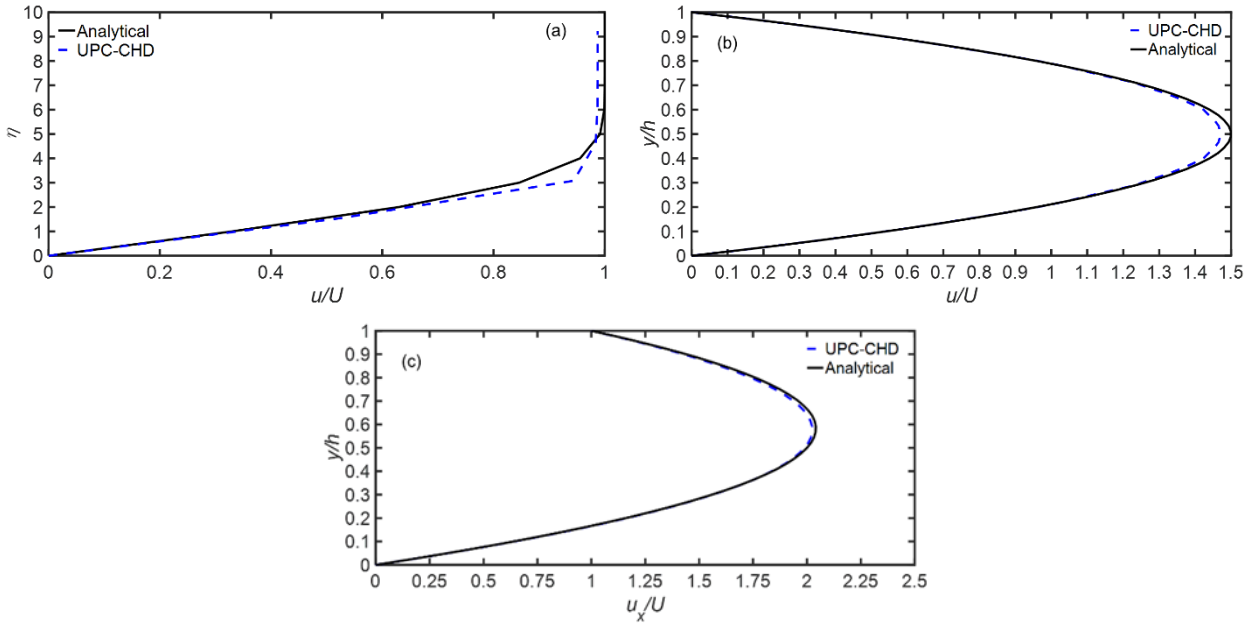

Fig. 3. Comparison between analytical and predicted values for respective CHD cases in UPC-CHD: (a) Blasius boundary layer, (b) Poiseuille's flow, and (c) Couette's flow.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

Generally, both UPC-CHD and MPI achieve a near identical S values up to $c = 16$ in Fig. 4. However, beyond 16 and up to $c = 100$ maximum in this study, UPC's speedup is most significant by having a close ratio of 1:1 and reaches nearly 90 times speedup for CHD cases A and B (Fig. 4.).

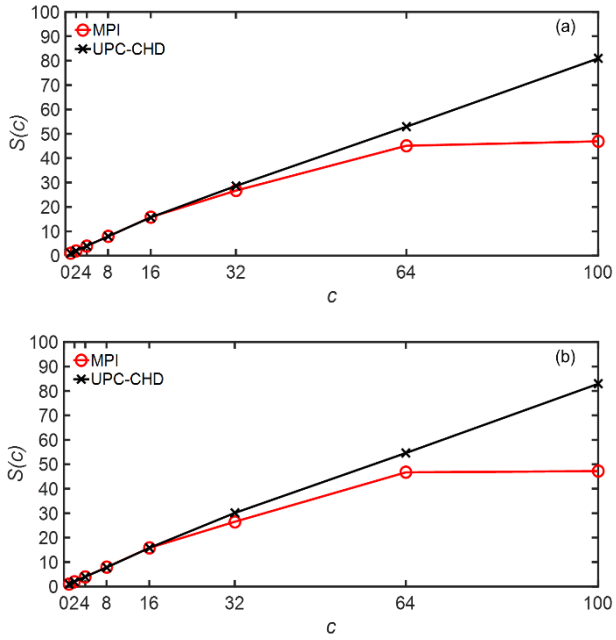

Fig. 4. Comparison of parallelism performance between MPI and UPC-CHD: (a) Blasius boundary layer – CHD Case A, and (b) Poiseuille's flow – CHD Case B.

MPI achieves relatively significant S values for Cases A and B in Fig. 4 for c beyond 16. The difference in the $S(c)$ between Cases A and B is likely ascribed to the difference in the number of computational nodes deployed (25 million nodes for Case A and 100 million nodes for Case B) which affect the number of messages being transmitted within each assigned sub-domain. For instance, when running with $c = 32$, MPI achieves S values of 26.7 and 22.5 for Case A and B respectively (Fig. 4.). With MPI, thread T_i transmits multiple messages to the neighboring threads at every time-step which include: (a) velocity and convective fluxes data values in x- and y- directions corresponding to thread T_{i-1} and T_{i+1}, and (b) updated data to the main thread. At the maximum $c = 100$, over 900 messages must be processed in the system at each iterative step, despite having only 100 rows of data to be computed for each thread. The respective differences in the number of messages to be transmitted in Cases A and B result in different processing time for each message. As c increases beyond 32 with the MPI architecture, the idling time among the threads takes effect whereby each thread must wait for the other threads to complete their respective computations at each time step which thus explains the gradual stagnation in the parallelism performance after 64 cores and thereafter. Consequently, the total message processing time outweighs the total computational time in

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

each thread, which restricts the continual speedup with an increasing number of threads with MPI. On the contrary, Fig. 4 indicates that UPC-CHD better manages the total message processing time beyond $c = 64$ due to its inherent shared memory component which enables the threads to access the data within the shared memory via a global address.

The advantage of embedded locality consciousness in UPC-CHD is further investigated in Case C by further examining the impact of thread affinity on its parallelism performance. The computational data of each sub-domain are first stored in-block to gain memory locality properties, while the global memory accessing activities are overlapped with remote control technique using the split-phase barrier to conceal the synchronization cost. To further illustrate this advantage, we evaluate the performance for Case C under two scenarios: (a) UPC-A, i.e. UPC with optimizations, and (b) UPC-NA, i.e. UPC without optimizations and employs the defaults setting of the GPAS compilers.

As expected, UPC-NA's speedup performance is vividly inferior to that of UPC-A's as shown in Fig. 5. For instance, at $c = 16$, the respective S values attained are 15.8 and 4.3 for UPC-A and UPC-NA, respectively.

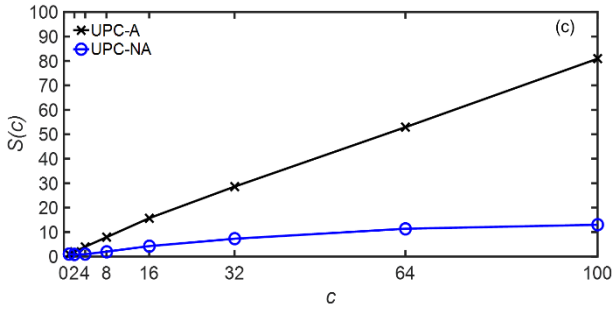

Fig. 5. Comparison of parallelism performance between UPC-A and UPC-NA for CHD case C (Couette's flow)

The difference in the attained speedup between UPC-A and UPC-NA can be further described from Fig. 6 by using a 3 x 3 numerical domain example having an affinity block of 3. In UPC-NA, thread 0 contains the fluxes data of element a, d and g in its local memory section whereby d and g belongs to the other threads, whereas in UPC-A, thread 0 contains the fluxes data of element a, b and c in its local memory (Fig. 6) whereby all three elements belong to the common thread. The observed inferior performance of UPC-NA (Fig. 5) is caused by the need for thread 0 to function with non-affinity data, i.e. element b in thread 1 and element c in thread 2, which results in longer computational run time for UPC-NA. With $c = 1$, there is only a singular thread which computes the fluxes in the entire numerical domain, and functions only with data having affinity with. With an increasing number of threads, the latency issue arises which results in less than ideal computational performance. For instance, at $c = 2$ and 4 respectively, 50% and 25% of the total runtime are attributed to the need to access non-affinity data by the respective threads. While the addition of c would reduce the amount of global

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

accessing activity in UPC-NA, optimization is still unlikely as observed in Fig. 5. This observed behavior for S needs to be further investigated.

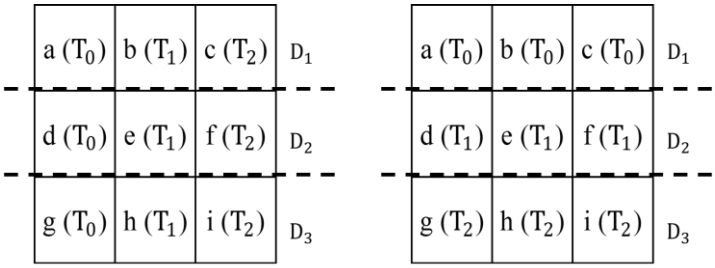

Fig. 6. Schematic representation of UPC-NA (left) and UPC-A (right) concepts

In summary, we recommend distributing the array of data in contiguous blocks as demonstrated in UPC-A, which enables each thread to attend to a system of elements as dependent on the number of threads available. We hypothesize that the proposed UPC-PGAS architecture without any affinity optimization is likely unsuitable in accelerating the speedup performance when compared with to that of MPI's.

6 Conclusion

An alternative parallelism architecture, which couples the PGAS computing concept with Berkeley's UPC compiler, is developed for large-scale computational hydrodynamic (CHD) simulations. The parallelism model developed is termed as UPC-CHD. As our first initiation, UPC-CHD is examined on three incompressible, adiabatic and viscous 2D flow cases having moderate flow velocities. Varying numerical schemes are adopted to resolve the discretized forms for the respective temporal, viscous and convective fluxes within each computational node in UPC-CHD. The selected schemes are then verified by the reasonably good agreement obtained between the predicted and analytical values for all CHD flow cases. Subsequently, the parallelism performance is compared between UPC-CHD and MPI in its base design till 100 computer threads maximum in this study as our first approach. The obtained speedup results provide an early indication of the parallelism capability of UPC-CHD for large-scale numerical domains which can be further exploited by performing the following.

- Introduction of other numerical schemes to examine a wider range of CHD flow cases which include turbulent incompressible flow, heat transfer, porous media flow etc.
- Coupling of UPC-CHD with AWS cloud computing to exploit a greater number of cores to further evaluate the parallelism capability of the former for large-scale simulation domains.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

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 for the 4-year Nanyang President Graduate Scholarship (NPGS) for his PhD study.

References

1. Anderson, J. D. (2009), "Governing Equations of Fluid Dynamics," In J. F. Wendt (Ed.), *Computational Fluid Dynamics*. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 15 - 51.
2. Balaji, P., Buntinas, D., Goodell, D., Gropp, W., Kumar, S., Lusk, E., Lusk, Thakur R., Träff, J. L. (2009), "MPI on a Million Cores.,"In M. Ropo, J. Westerholm & J. Dongarra (Eds.), Recent Advances in Parallel Virtual Machine and Message Passing Interface: 16th European PVM/MPI Users' Group Meeting, Espoo, Finland, September 7-10, 2009. Proceedings (pp. 20-30). Berlin, Heidelberg: Springer Berlin Heidelberg.
3. Bucs, S. S., Radu, A. I., Lavric, V., Vrouwenvelder, J. S. , and Picioreanu, C. (2014), "Effect of different commercial feed spacers on biofouling of reverse osmosis membrane systems: A numerical study," *Desalination*, vol. 343, pp. 26-37.
4. Chang, C.-W., Liu, P. L. F., Mei, C. C., & Maza, M. (2017), "Modeling transient long waves propagating through a heterogeneous coastal forest of arbitrary shape," *Coastal Engineering*, vol. 122(Supplement C), pp. 124-140.
5. Chen, WY, Bonachea, D., Duell, J., Husbands, P., Iancu, C., & Yelick, K. (2003)," A Performance Analysis of the Berkeley UPC Compiler," *Lawrence Berkeley National Laboratory*. Retrieved from https://escholarship.org/uc/item/91v1j2jw
6. Dalwadi, M.P., Griffiths, I.M., and Bruna, M., (2015), "Understanding how porosity gradients can make a better filter using homogenization theory," *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Science*, vol. 471 (2182).
7. Dalwadi, M., Bruna, M., and Griffiths, I., (2016), "A multiscale method to calculate filter blockage," *Journal of Fluid Mechanics*, vol. 809, pp. 264-289.
8. 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*, vol. 2 (1), p. 015003.
9. Jajcevic, D,, Siegmann, E., Radeke, C,, and Khinast, J. G. (2013), "Large-scale CFD–DEM simulations of fluidized granular systems," *Chemical Engineering Science*, vol. 98, pp. 298-310.
10. Jamshed, S. (2015), "Chapter 3 - The Way the HPC Works in CFD," in Using HPC for Computational Fluid Dynamics, ed Oxford: Academic Press, pp. 41-79.
11. Johnson, A.A. (2006), "Using Unified Parallel C to Enable New Types of CFD Applications on the Cray X1/E", Cray User Group Conference.
12. Kermani, M. and Plett, E. (2001), "Roe scheme in generalized coordinates. I - Formulations," in 39th Aerospace Sciences Meeting and Exhibit, ed: American Institute of Aeronautics and Astronautics.
13. Kermani, M. and Plett, E. (2001), "Roe scheme in generalized coordinates. II - Application to inviscid and viscous flows," in 39th Aerospace Sciences Meeting and Exhibit, ed: American Institute of Aeronautics and Astronautics.
14. Munson, B. R., Young, D. F., & Okiishi, T. H. (2006). Fundamentals of fluid mechanics. 6th Ed., Hoboken, NJ: J. Wiley & Sons, ch. 6. pp. 263-331

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34

15. Simmendinger, C., Jägersküpper, J., Machado, R., Lojewski, C. (2011), "A PGAS-based Implementation for the Unstructured CFD Solver TAU", Partitioned Global Address Space Programming Models, Galveston Island.
16. Sousa, P., Soares, A., Monteiro, E., and Rouboa, A. (2014), "A CFD study of the hydrodynamics in a desalination membrane filled with spacers," *Desalination*, vol. 349, pp. 22-30.
17. White, F. M. (1991), Viscous Fluid Flow, 2nd Ed.., McGraw-Hill, Chapter. 7, pp. 457-528.

ICCS Camera Ready Version 2018
To cite this paper please use the final published version:
DOI: 10.1007/978-3-319-93698-7_34