MPI/Open-MP Hybridization of Higher Order WENO Scheme for the Incompressible Navier-Stokes Equations

Mukundhan Selvam and Klaus A. Hoffmann
Department of Aerospace Engineering
Wichita State University, Wichita, KS

With the contemporary emerging transition for the past two decades in high performance computing (HPC) towards clusters of nodes, it is possible to obtain an efficiently scalable CFD code by hybridization of Message Passing Interface (MPI) and Open Multi-Processing (Open-MP). The parallelization of an incompressible higher order DNS solver using Navier-Stokes equations utilizing WENO (weighted Essentially Non-Oscillatory) scheme is presented in this paper. Initially, a time dependent two dimensional diffusion equation was parallelized by MPI in an effort to explore the concepts. Subsequently, the procedure is extended with several advanced MPI routines for the parallelization of higher order finite-difference schemes. The parallelized code is subsequently examined on a benchmark problem for incompressible flows – lid driven cavity. The results of the developed code were compared and validated with the serial model. The overall objective is to achieve the best performance of the incompressible Navier-Stokes solver by carefully implementing, profiling and optimizing both MPI communications and Open-MP parallelization. The results demonstrate that, incorporating Open-MP parallelization technique contributes a slight advantage in performance relative to pure MPI implementation. MPI processes are limited by the number of communicators and grid size of the simulation. However, fine grain parallelization is possible by incorporating Open-MP threads in certain segments of the code.

Nomenclature

u_i	=	Non-dimensional velocity (i=1,2 ; u,v)
P	=	Non-dimensional mean pressure
Re	=	Reynolds number
Δt	=	time step
u_i^n	=	velocity at time t or at numerical step n
u_i^{n+1}	=	velocity at next time step n+1
$\left(u_i\right)^*$	=	Intermediate velocity.

I. Introduction

Fueled by the impressive advances in the hardware architectures on one side, and the continuous progress in the numerical methods on other side, parallelizing CFD codes have become much essential and attained a stage that has been considered as cost effective alternative in most industries and their areas of interest. In this paper, we focus on finite difference discretizations of the two-dimensional (2D) Navier-stokes equations which are solved using the exact projection method.

The numerical method implemented here is the Weighted Essentially Non-Oscillatory (WENO) scheme for the incompressible Navier-Stokes (NS) equations. Although, the Navier-Stokes equations has been known for more than a century, many of the important fluid features elude our thorough comprehension which brings to a point that the fluid behavior is still unpredictable under many conditions of practical interest. Nowadays, it is widely

acknowledged that parallel high performance super-computers are one of the best options to meet the needs. The current WENO scheme is designed to execute at high resolutions to model the turbulence without artificial addition produced by a separate turbulence model. Such CFD models are known as Direct Numerical Simulation (DNS). The finite difference scheme developed here uses an exact projection to solve the incompressible Navier-stokes equations. For the spatial operator of the convective terms, fifth order WENO is implemented and sixth order compact finite difference scheme is used to solve the diffusion term in the NS equations. For the time integration, a fractional step scheme in conjunction with the third order Runge-Kutta scheme along with total variation diminishing (TVD) scheme is applied [1]. The parallel strategy implemented is by dividing the computational work or domain between several processors which performs the computations concurrently. We use the concept of domain decomposition method in which the computational domain is sub-divided into multiple sub-domains and each sub-domain is assigned to one processor. We also rely on BLAS [2] and LAPACK [3] library routines that perform basic vector and matrix operations, which further speeds up the computation.

Cappello, Richard, and Etiemble [4,5] were among the first to present the hybrid programming model of using MPI [6] in conjunction with a threading model such as OpenMP. Cappello and Etiemble [7] demonstrated that, it is possible to increase efficiency by using a mixture of shared memory and message passing models and established a comparison between them.

II. Computational Scheme

We consider the incompressible Navier stokes equations which are described by the conservation of mass and conservation of momentum and mathematically expressed as,

$$\frac{\partial u_i}{\partial t} + \frac{\partial (u_i u_j)}{\partial x_j} = -\frac{\partial P}{\partial x_i} + \frac{1}{\mathrm{Re}} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \qquad (1.1)$$

$$\frac{\partial u_i}{\partial x_i} = 0 \qquad (1.2)$$

To resolve the incompressible flow, it is necessary to solve the equations (1.1) and (1.2) for velocity and pressure with appropriate boundary and initial conditions. In case of two dimensional incompressible flows, the NS equations must be solved for two coordinate velocities, say u and v and the pressure field, P. Equation (1.1) represents momentum along the two coordinate directions and can be used to solve for two components of velocity. However, there is no explicit equation for pressure. If a correct pressure field is known, then equation (1.1) will provide a velocity field, which will be divergence free; in other words will satisfy the equation (1.2) [1]. Wu and Jiang [8] presented a higher order finite difference WENO scheme for solving the incompressible fluid flow equations which applies a fractional time-step method to enforce the incompressibility conditions and established the WENO scheme was numerically stable. Here, we use an exact projection/fractional step scheme to solve the NS equations.

$$\left(\frac{\partial u_i}{\partial t}\right)^* = \frac{\partial u_i u_j}{\partial x_j} + \frac{1}{\mathrm{Re}} \frac{\partial^2 u_i}{\partial x_j \partial x_j} \qquad (1.3)$$

$$\frac{\partial^2 P}{\partial x_i \partial x_i} = \frac{1}{\Delta t} \frac{\partial (u_i)^*}{\partial x_i} \qquad (1.4)$$

$$\left(u_i\right)^{n+1} = \left(u_i\right)^* - \frac{\partial P}{\partial x_i} \Delta t \qquad (1.5)$$

Altogether, the governing equations (1.3) through (1.5) are solved numerically for an incompressible flow. In equation (1.3), Runge-Kutta integration is applied to obtain $(u_i)^*$. Next, the poisson solver is used to solve the equation (1.4) to obtain pressure, P. Eventually, with the intermediate velocity and the obtained pressure, equation (1.5) is solved to calculate velocity at the next time step $\left(\overset{+}{u}_i\right)^{n\ 1}$. This is the main loop of a single Runge-Kutta step, wherein this loop is repeated three times for a single time step. The above equations are expressed in a two-dimensional general coordinate system and in a useful form for WENO schemes numerically. The significant advantage of the projection method is that the computations of the velocity and pressure fields are decoupled. And the feature implemented from Chorin's projection method [9] is that the solver is validated with the discrete continuity equation at the end of each time step.

III. Numerical Example

The developed numerical code is investigated on a well-known benchmark problem for incompressible flows : lid driven cavity flow. This problem has been extensively investigated because it has certain desirable flow features such as boundary layer on the wall, flow separation from one wall and reattachment on the perpendicular wall, attachment and flow separation on the same wall, multiple flow separations and vortices. A detailed work was performed by Khurshid and Hoffmann [1] with the application of the WENO scheme. In the current paper, the discretization is performed on a uniform, Cartesian grid, with a grid size of 320 x 320, comprising above than one hundred thousand grid cells on the entire computation domain. The initial and boundary conditions are shown in the Figure 1. A time step of dt = $1x10^{-3}$ and Reynolds number of Re=3200 is used for the simulations.

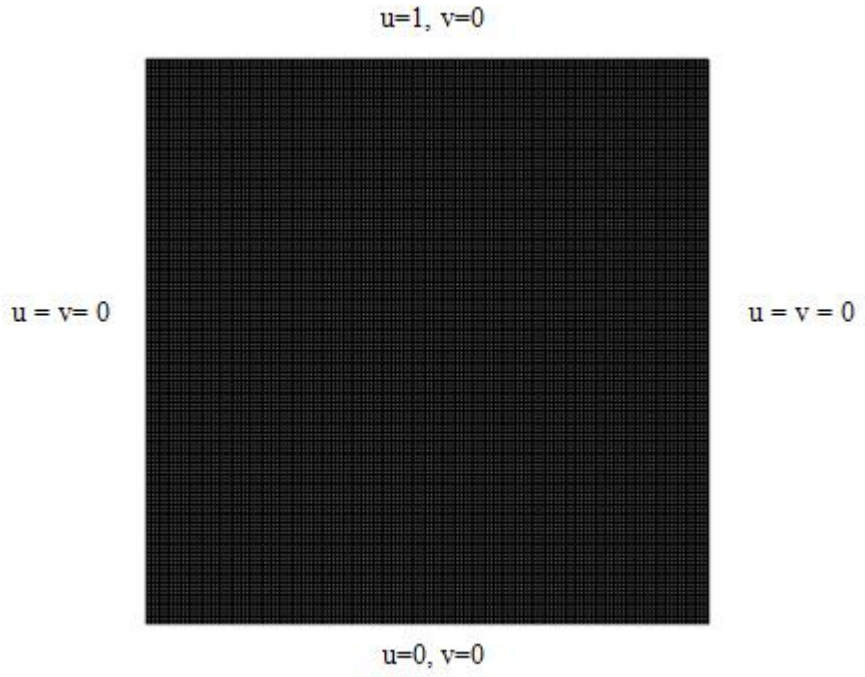

Figure 1 : Initial, boundary conditions and mesh for driven cavity flow.

A. Domain Discretization

Out of the several partitioning in parallelization, domain decomposition parallelism is utilized. In this method, every processor is assigned a copy of their domain, which results in a uniform domain decomposition in each direction. Hence, the physical square domain is logically divided up into smaller sub-domains.

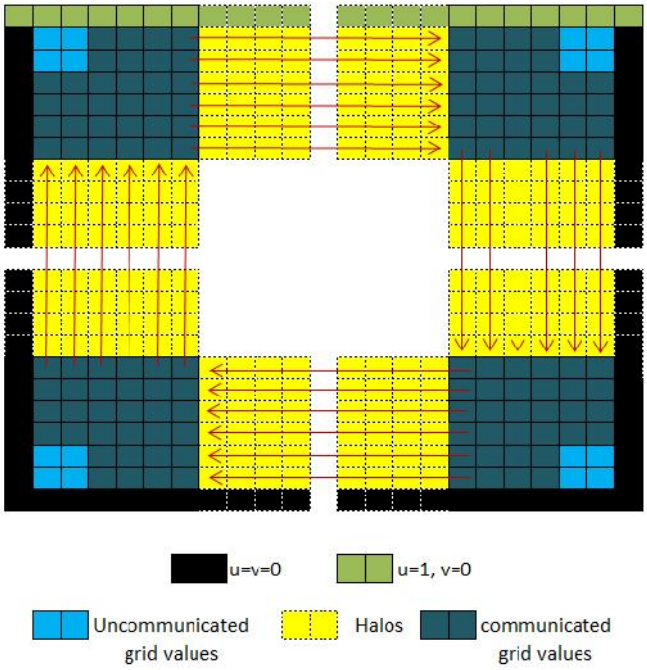

Figure 2. An example of the Cartesian process topology for the Lid driven cavity.

Figure 2 illustrates an example of the break-down of the global domain into 4 sub-domains. This type of MPI virtual topology is known as Cartesian process topology.

B. Parallel Approach

From MPI's point of view, it ignores the fact that cores inside a single node work on shared memory. Also, it implicitly assumes utilizing the message passing is the correct paradigm on all levels of parallelism in the application. Once the domain is equally divided into sub-domains, we assign each subdomain to one MPI process in such a way that each process has the same amount of computational workload. The decomposition is dependent on the number of grid size (Nx, Ny) and the number of processors (np). Thus, one needs to modify the boundary conditions that have been defined, so the computations on each sub-domain can be performed in parallel. The boundaries signify the real physical boundaries of the problem and the interfaces between the sub-domains. It is essential to develop the program with a virtually mapped topology in relevance with the number of MPI processes, and the data are communicated as necessary.

Subsequently, the communication phase of the paralle model is programmed. Here, blocking point-to-point MPI routines are employed, namely MPI_SENDRECV. The processors need to communicate the information for every overlapping portion in the sub-domains. Hence, it is required to communicate the data from the interior points to halo cells. In our example, velocities in both the coordinate axis (u ,v) and along with mean pressure (P) are communicated for each column/row of halo cells. Thus, each processor has to call MPI communication routines MPI_SENDRECV twelve times in order to send/receive necessary non-contiguous data at the interface to/from its neighbours; suppose each process has four neighbours. Every halo corresponds to a real grid point on another MPI process and the message passing stages update this information as needed between calculation steps during many iterations. Call to multiple send and receive routines from the processes, brings the problem of deadlock. To overcome this issue, we have used MPI_SENDRECV which has both the send and receive in a single call.

As known, MPI communications are the core component and crucial phase of MPI parallelization. It is mandatory to optimize the communication routines to achieve the best performance. Moreover, MPI has an efficient approach in handling non-contiguous data and they are known as derived datatypes which are built from basic datatypes. We will use MPI_TYPE_VECTOR which creates a datatype consisting of uniformly spaced blocks of data in a sub-matrix as shown in the Figure 3. The sub-matrix is subsequently communicated with the MPI_SENDRECV to send and receive the data which is relatively much simple. By this approach, call to MPI_SENDRECV is reduced to four times for one call of the message passing stage. The message passing stage is called once at the end of each Runge-Kutta step.

```
      call MPI_TYPE_VECTOR(4,arraynum,arraynum,
     &                     MPI_DOUBLE_PRECISION,columntype,ierr)
      call MPI_TYPE_COMMIT(columntype,ierr)
      if(ytcoord.NE.(ynumnodes-1)) then
```

Figure 3. Example of a Derived datatype, utilized to communicate Non-contiguous data

On the other hand, the implementation of OpenMP is quite straightforward. Before the OpenMP implementation, each part of the code is thoroughly analyzed and profiled with the help of VampirTrace view as shown in the Figure 4.

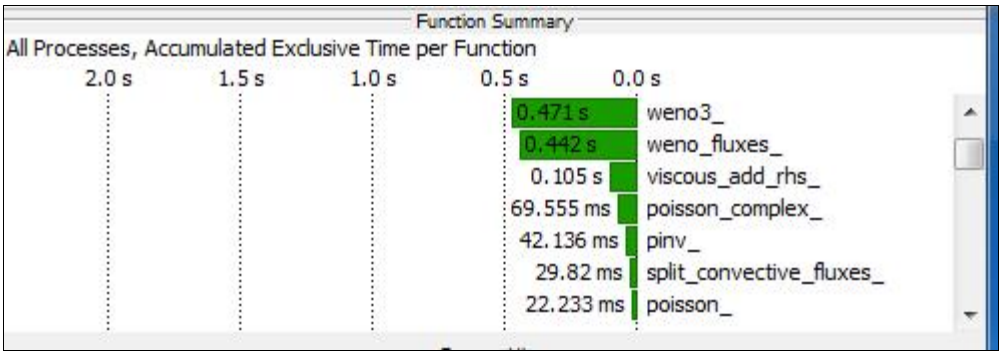

Figure 4. Profiling with VampirTrace.

This necessitated removing OpenMP functionality from areas of the code that were not heavily used, and therefore, removing the overheads for those sections and re-implementing at crucial functionalities to achieve the best performance. OpenMP pragmas are specifically applied in computation subroutine of the WENO fluxes and derivatives to reduce its execution time. Thus, the most time consuming routine of the code is parallelized with OpenMP (loop level parallelism). All that is necessary is to share the variables required by parallelization within the DO loops, and it is managed by the OpenMP directives. However, there were only few parts of the code that had been parallelized with OpenMP.

IV. Performance and Results

We have evaluated the performance of hybrid parallelization on the cluster called Athena, managed by High Performance Computing Center at Wichita State University. Athena is comprised of 22 compute nodes of Dell PowerEdge R815, each with four 8-code AMD processors at 2.33 GHz.

320x320	Wall-clock time (s)				
Number of MPI Processes	Pure MPI	MPI + OMP(2)	MPI + OMP(4)	Speed-up	Efficiency
1	300.879	286.53	292.631	-	-
2	131.510	125.07	121.461	2.4771655	1.23858
4	51.683	49.589	49.730	6.0674544	**1.5168636**
8	38.431	**37.031**	39.487	8.1250573	1.0156321
16	41.896	42.690	43.130	7.1819982	0.4488748
32	45.030	47.811	45.388	6.6817455	0.2088045

Table.1: Parallelization results comparison

Table.1, demonstrates the speed-up and efficiency of different combinations of the parallel techniques utilized. Collectively, the idea of parallelization is to i) reduce the execution time to achieve the finest speed-up performance and ii) design a system to share the loadwork equally to produce higher efficiency.

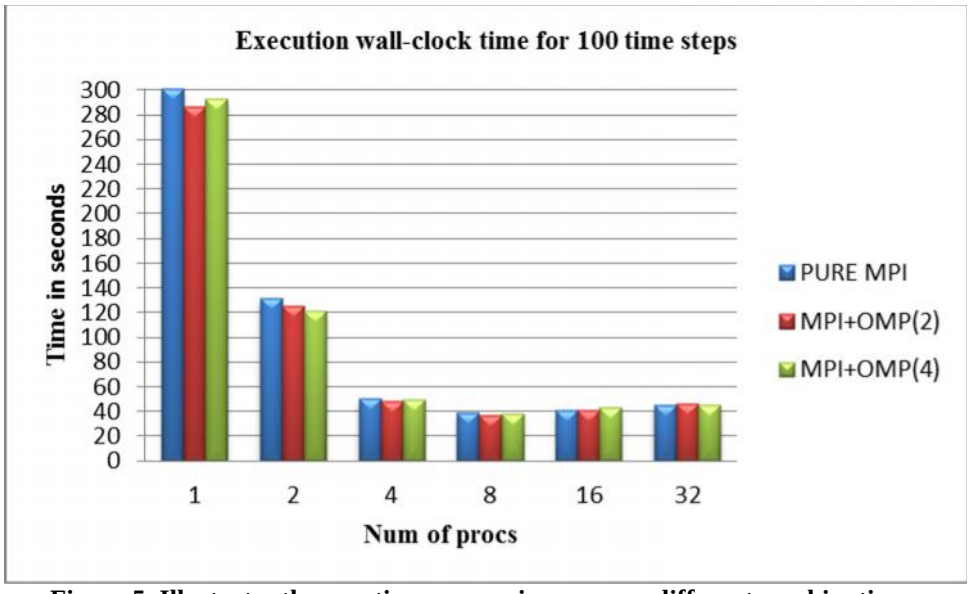

Figure 5. Illustrates the run-time comparison among different combinations.

We have also compared different combinations of MPI tasks and OpenMP threads in the parallel model. From Figure 5, it can be seen that the best possible configuration is with 8 MPI tasks and 2 OpenMP threads for a problem of 320 x 320. When the number of MPI processes and OpenMP thread increases, the communication overhead builds up resulting in a inefficient parallel performance, as seen in the combination of 32 MPI tasks with 4 OpenMP threads. As the grid on each process becomes smaller, leading the communication overhead too large compared to the amount of computations performed on that process. From the performance graph, one can summarise that the hybrid model shows strong scaling for the current size of the problem. Figure 6 compares the results of serial model against those produced with the parallel model. As can be seen, the results show excellent correspondence, aiding us in the implementation of the parallelization.

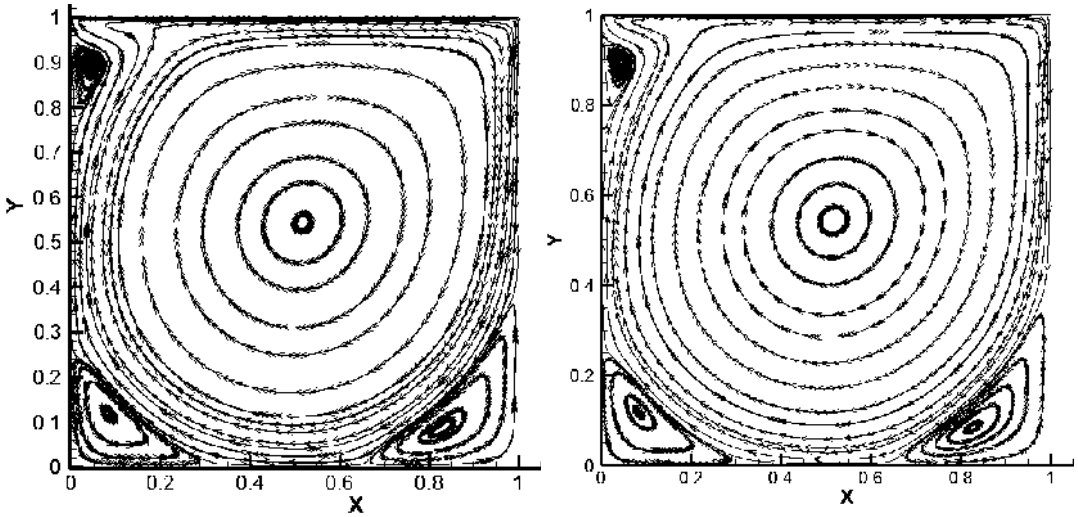

Figure 6. Streamtraces for lid driven cavity at Re =3200 between the serial & hybrid parallel model.

V. Conclusion

We have investigated an hybrid parallel model (MPI + OpenMP) implementation focusing on the performance of serial communication modes (MPI_THREAD_SERIALIZE). It is shown that OpenMP hybridization provides an increase in the performance of the solver at certain combinations with MPI implementation. The significant ingredient increasing the performance of the code on modern HPC is to consistently reduce the memory allocation of a simulation for each processes in a problem. Based on the limited applications, it is evident that the hybrid parallel technique is much vital and beneficial.

References

1. Khurshid, H., and Hoffmann K.A., "Implementation of a WENO scheme to the incompressible Navier-Stokes Equations in Generalized Coordinate System", AIAA-2010-4993, 40th Fluid Dynamics Conference and Exhibit, Chicago, IL, June 28-July 1, 2010.
2. Basic Linear Algebra Subprograms Technical Forum Standard 16 (1).
3. E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, et al., LAPACK User's guide, SIAM, Philadelphia, 1999, third edition.
4. Cappello, F., Richard, O., and Etiemble, D., "Performance of the NAS Benchmarks on a Cluster of SMP PCs Using a Parallelization of the MPI Programs with OpenMP," PaCT, 1999, pp. 339–350.
5. Cappello, F., Richard, O., and Etiemble, D., "Investigating the Performance of Two Programming Models for Clusters of SMP PCs," HPCA, 2000, pp. 349–359.
6. M. P. I. Forum, MPI : A Message-Passing Interface Standard, Int. J. Supercomputer Applications and High Performance Computing, 1994.
7. Cappello, F. and Etiemble, D., "MPI versus MPI+OpenMP on IBM SP for the NAS benchmarks," Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM), IEEE Computer Society, Washington, D.C., United States, 2000.

8. Wu, C.C., and Jiang, G.S.,"A high-order WENO finite difference scheme for the equations of idea Magnetohydrodynamics," Journal of Computational Physics, Vol. 150, 1999, pp. 561-94.
9. Chorin, A.J., "Numerical solutions to Navier-Stokes equations," Mathematics of Computation, Vol. 22, 1968, pp. 745-762.