Performance analysis of large scale parallel CFD computing based on Code_Saturne

Zhi Shang
Department of Computational Science and Engineering, Science and Technology Facilities Council, Daresbury Laboratory, Warrington WA4 4AD, United Kingdom

ARTICLE INFO

Article history:
Received 21 December 2011
Received in revised form 18 September 2012
Accepted 22 September 2012
Available online 28 September 2012

Keywords:
CFD
Petascale
Exascale
Optimization
Code_Saturne
HPC

ABSTRACT

In order to run computational fluid dynamics (CFD) codes on large scales, parallel computing has to be employed. For instance, on Petascale computing, general parallel computing without any optimization is not enough, especially for complex industrial issues that employ a large number of mesh cells to capture the details of the geometry. How to distribute these mesh cells among the multi-processors for Terascale and Petascale systems to obtain a good performance on parallel computing is really a challenge. Some mesh partitioning software packages, such as Metis, ParMetis, PT-Scotch and Zoltan, were chosen as the candidates ported into Code_Saturne to test if they can lead Code_Saturne towards Petascale and Exascale parallel CFD computing. Through the studies, it was found that mesh partitioning optimization software packages based on the graph mesh partitioning method can help the CFD code obtain good mesh distributions for high performance computing (HPC).

1. Introduction

Nowadays CFD plays an important role in industrial design, environmental estimations and academic research [1–3]. In order to deal with complex geometries and fluid flows, a large number of meshes have to be employed. Therefore parallel computing has to be introduced to cope with the large number of mesh cells [4,5]. However the general parallel computing is unsuitable for the future Petascale computing, which will run on hundreds of thousands of processors [6]. Therefore the scalability of both numerical methods and parallel optimization has to be investigated. In this paper, the new version of Code_Saturne 2.0.0-beta2, which has been selected as one of the core application benchmark codes in PRACE representing CFD applications [6], is chosen as the CFD tool suite for the studies.

Code_Saturne is an multi-purpose open source CFD software package [7]. The code was originally developed by EDF [7] and designed for industrial applications and scientific research activities in several fields related to energy production. They include nuclear power thermal-hydraulics, gas and coal combustion, turbo-machinery, heating, ventilation and air conditioning.

Prior to the studies of large scale parallel computing using Code_Saturne, a validation case of flow around a DARPA submarine was tested. Fig. 1 shows the geometry of the submarine [8,9]. It includes the nose, the tail and the appendages located at the tail. Every detail of the submarine model mimics a real submarine.

The flow parameter for the simulation is same as the experiments of DARPA 2 [8,9]. The flow is with a 9 m/s flow speed attacking the submarine’s nose with a zero angle. The corresponding Reynolds number reaches \(3.89 \times 10^7\) based on the length of the submarine.

The RANS model (\(k-\varepsilon\) model) is employed to model the turbulence. The standard wall function is therefore chosen as the near wall treatment. The whole domain was meshed by tetrahedral type mesh cells and a stretched prism mesh with growth rate 1.2 near wall was adopted; this kept the near wall \(+y^+\) value as 30 on average and within the range of 25 to 70.

The simulations were carried by solving the Navier–Stokes equations for water flow around the solid body.

\[
\partial \rho / \partial t + \nabla \cdot (\rho U) = 0
\]

\[
\partial (\rho U) / \partial t + \nabla \cdot (\rho U U) = -\nabla p + \rho g + \nabla \cdot \left[ (\mu + \mu_t) \left( \nabla U + \nabla U^T \right) \right] + G - \rho \varepsilon
\]

\[
\partial (\rho k) / \partial t + \nabla \cdot (\rho U k) = \nabla \cdot \left[ \left( \mu + \frac{\mu_t}{\sigma_k} \right) \nabla k \right] + G - \rho \varepsilon
\]

\[
\partial (\rho \varepsilon) / \partial t + \nabla \cdot (\rho U \varepsilon) = \nabla \cdot \left[ \left( \mu + \frac{\mu_t}{\sigma_\varepsilon} \right) \nabla \varepsilon \right] + \frac{\varepsilon}{k} (C_1 G - C_2 \rho \varepsilon)
\]

\[
\mu_t = C_\mu \frac{k^2}{\varepsilon}
\]

Copyright © 2012 Published by Elsevier B.V. All rights reserved.
where \( \rho \) is the density, \( \mathbf{U} \) are the velocity vectors, \( p \) is the pressure, \( g \) is the gravitational acceleration vector, \( \mu \) is the viscosity, \( \mu_t \) is the production of turbulence kinetic energy, \( \sigma_k, \sigma_\varepsilon, C_1, C_2 \) are constants for the standard k-\( \varepsilon \) turbulence model, shown in Table 1.

During the simulations, the flow regime was set up the same as the experiments of DARPA 2 [8,9]. The flow was considered as a steady state flow, i.e. all the time derivatives of the variables are zero, i.e. \( \partial \sigma_t / \partial t = 0 \). The inlet was set up with a uniform velocity of 9 m/s entering the domain. The outlet was treated as a fully developed boundary because the location was set up far away from the submarine body. The sides of the domain were treated as symmetrical boundaries to weaken the influence on the flow fields from outside. The convergence criteria were set up as the errors of all variables to be less than \( 10^{-3} \). In this paper, the algebraic multi-grid method, which was parallelized by Code-Saturne, was employed as the solver.

Fig. 2 shows the validation results simulated by Code_Saturne compare with experimental and several CFD software packages on the pressure coefficients (\( C_p \)) at different cross sections along the DARPA submarine’s body.

The curve shows a quite good agreement with experimental results. Code_Saturne obtains simulation results equivalent to the well-known commercial CFD software packages Fluent [10] and STAR-CD [11]. Based on this submarine model, a case that had 121,989,150 (121 M) cells for testing the large scale parallel computing was built up.

Several open source mesh partitioning software packages, such as Metis [12], ParMetis [13], PT-Scotch [14] and Zoltan [15], were ported into Code_Saturne 2.0.0-beta2 as the mesh partitioning tools to optimize the parallel computing. The aim is to test if these mesh partitioning software packages can help Code_Saturne to improve the capacity of high-performance computing (HPC) in dealing with massive meshes for the future Petascale computing. The tests were performed on HECToR Cray XT4, which is a high-end computing resource in the UK [16]. HECToR used AMD quad cores in each node with a clock speed of 2.3 GHz. The total number of nodes was 5664. Therefore the total number of cores reached 22,656. The maximum available number of cores was 8192 to every single user. The network of a 3D torus topology was employed to perform the inter-connection [17].

### 2. Mesh partitioning software packages porting into Code_Saturne

Several factors will affect HPC of a CFD code on large-scale parallel computing. Besides the numerical algorithms, for instance Gauss–Seidel iteration, conjugate gradient and algebraic multi-grid methods, the quality of the mesh partitioning has a strong effect on HPC of CFD codes [4,5]. Generally, the numerical algorithms, which have already been developed for decades, are approaching complete sophistication. It is difficult to enhance the parallel performance much by improving the numerical algorithms. Therefore a good mesh partitioning (domain decomposition) strategy will be the crucial issue for the high-performance parallel computing, especially for cases having a large number of mesh cells.

Two mesh partitioning strategies are generally used in CFD codes. They are graph and geometry mesh partitioning methods. The graph mesh partitioning method converts the original meshes into a graph with edges connected through vertexes. The optimization is to minimize the edge-cut. After the mesh partitioning, the original mesh can be distributed into a number of sub-domains that equals to the number of processors [12]. The geometry mesh partitioning method uses the physical coordinates of the original mesh as the target to partition the points to be physically close to each other in space [15], which is very useful for a situation where the geometric locality is important.

The candidate software packages, which were chosen to be ported into Code_Saturne for performing the mesh partitioning, can be separated into sequential software packages and parallel software packages. According to the types of mesh partitioning functions and methods, different mesh partitioning software packages were adopted to connect with Code_Saturne.

The METIS_PartGraphKWay function was employed in Metis by Code_Saturne. The library of Metis 5.0pre2 was used by Code_Saturne through the preprocessor because Metis is a serial code. ParMetis with the corresponding library of ParMetis 3.1.1 was employed by Code_Saturne through the kernel for connecting to the MPI parallel library, and the ParMETIS_V3_PartKway function was employed. PT-Scotch 5.1 with the corresponding library was employed by Code_Saturne through the kernel like ParMetis because PT-Scotch is a parallel code. The specific SCOTCH_dgraphPart function was employed for the mesh partitioning. Since Zoltan can provide a robust geometry mesh partitioning for the domain decomposition, it was ported into Code_Saturne with its geometry mesh partitioning. The Hilbert space-filling curve (HSFC) in Zoltan was employed for all the geometry partitioning in this paper. The same as ParMetis and PT-Scotch, the corresponding library of Zoltan 3.1 was employed by Code_Saturne through the kernel. The Zoltan_LB_Partition function was employed by Code_Saturne to perform the mesh partitioning in parallel.
3. Mesh partitioning quality

Normally CFD simulations are greatly affected by the quality of the mesh generation [8]. A high-quality mesh generation can make CFD simulations not only converge quickly, but also accurately. However, a poor meshing may lead CFD simulations to be divergent without any result. As for the influence of mesh generation, the quality of mesh partitioning will affect the performance of HPC. The mesh partitioning quality can be measured by some criteria, such as load imbalance, distributions of halo cells and neighbors.

The load imbalance is defined as the processors’ number multiplying the maximum number of cells among processors and then the multiplied result divided by 121,989,150 (121 M) cells in this paper. The distributions of halo cells and neighbors can be measured by their average number and maximum number among the processors.

As is well known, in parallel CFD computing, data communication among processors should be performed. Usually the halo cells are used to perform the data exchange. The processing of mesh partitioning therefore should contain the generation of the halo cells.

Fig. 3 shows the working mechanism of halo cells. Actually the halo cells are the inner boundaries between two different processors. When the initial mesh cells were distributed into the processors, the corresponding halo cells along the boundary of each processor are generated for the data exchange. Needless to say, good mesh distributions including halo cells will help the parallel computing efficiently and fast.

As mentioned above, the halo cells are employed by parallel CFD computing to perform data communications between two neighboring processors. In a practical parallel CFD computing, these data communications will take time. Therefore the distribution of halo cells and neighbors is important for HPC to perform parallel CFD computing [4,5].

Due to the large memory requirement, all the partitioning of Metis 5.0pre2 was carried out on an SGI machine in Daresbury Laboratory [18], which has 96 GB memory at one processor. The peak value of memory used by Metis 5.0pre2 on the mesh partitioning of the 121 M case was around 30 GB for all the mesh partitioning in this paper.

Parallel performance can get rid of the memory limits at one processor for large scale mesh partitioning, because the original large mesh cells were decomposed and distributed into processors as small parts. Every processor only uses one part of the original mesh cells locally. Accordingly a single processor only needs a small memory requirement to store a small fraction of the mesh cells. In contrast, the serial program, which is performed only by a single processor, has to employ large size memory to store all the large mesh cells. Hence, it often hits memory limits. ParMetis 3.1.1, PT-Scotch 5.1 and Zoltan 3.1 (HSFC), which are parallel software packages, can be run on clusters. A large memory is not required at each processor.

Fig. 4 shows the results for average number of halo cells under different processors (sub-domains). The processors were separated into two groups: a coarse group of grids, including a number of processors from 32 to 128; and a fine group of grids, including a number of processors from 512 to 8192—due to HECToR Cray XT4 only providing 8192 processors to a user.

It can be seen that in the coarse group of grids, Zoltan (HSFC) has the highest average number of halo cells. Metis shows an rapidly diminishing curve with an increasing number of processors. At 128 processors, all the halo cells of Metis, ParMetis, PT-Scotch and Zoltan (HSFC) approach similar values. In the fine group of grids, ParMetis has the lowest average number of halo cells compared with the others. PT-Scotch has the highest average number of halo cells. Zoltan (HSFC) has the lowest average number of halo cells in the case with 8192 processors.

Fig. 5 shows the results of the average number of neighbors under different processors. It can be seen that in the coarse group of grids, Zoltan (HSFC) has the highest average number of neighbors, almost twice than others. The number of neighbors of Metis, ParMetis and PT-Scotch increases slightly with an increasing number of processors. In the fine group of grids, PT-Scotch has an average number of neighbors greater than Metis and ParMetis. Zoltan (HSFC) has a filling curve for the number of neighbors against the number of processors. When the number of processors is between 512 and 2048, ParMetis has the lowest average number of neighbors compared with the others. When the number of processors is greater than 2048, PT-Scotch has a markedly rising curve. When the number of processors reaches 8192, PT-Scotch has
an average number of neighbors almost twice that of Metis and ParMetis.

The information of halo cells and neighbors affects the data exchange during parallel computing. It affects the time primarily through the data communications rather than the iterations themselves. The time for iterations is mainly affected by the number of cells allocated to the processors. Normally, they can be measured by the load imbalance. From the definition of the load imbalance, it can be seen that a smaller load imbalance will lead to uniform mesh distributions among the processors. It will reduce the idle time of processors waiting for synchronization and improve HPC. The ideal minimum load imbalance should be 1.0. However, in parallel CFD computing, the load imbalance should include the statistics of the halo cells because the iterations use the halo cells as the boundaries of a single processor, which will participate in the computing. Therefore the number of cells at each processor should be the sum of the standard cells and the halo cells, due to the halo cells being treated the same as standard cells during the numerical iterations.

Fig. 6 shows the results of the load imbalance under different processors. It can be seen that in the coarse group of grids, PT-Scotch has a greater load imbalance. The others have similar load imbalances, especially when the number of processors is close to 128. In the fine group of grids, Zoltan (HSFC) has an almost rising curve. When the number of processors is greater than 2048, Zoltan (HSFC) has a load imbalance larger than the others. Metis, ParMetis and PT-Scotch have similar load imbalances, but Metis is lower than PaMetis and PT-Scotch.

Fig. 7 shows the average number of cells, including the statistics of halo cells, against the number of processors. It can be seen that all mesh partitioning schemes have similar average numbers of cells. Actually, the practical iteration CPU time is determined by number of cells at the processor. If the number of cells is large, the CPU will spend a lot of time for the iterations by the processor. Hence, the average CPU time of the processors relies on the average number of cells of the processors. A large number of processors, which has a lower average number of cells at each processor, should spend less CPU time on the iteration computing than the case with a small number of processors. Therefore, in general, Exascale parallel CFD computing will be faster than Petascale, and Petascale will be faster than Terascale. This consequence is evident, especially on HPC for a large number of mesh cells [19].

4. Parallel performance on HECToR

From the comparisons of the quality of the mesh partitioning from different software packages in Section 3, it can be seen that Metis, ParMetis and PT-Scotch have relatively higher mesh partitioning quality. They should have good HPC on parallel CFD computing. The amount of CPU time spent can be used to measure the HPC of parallel computing. For future Petascale computing, it will require tens of thousands of processors. It means that only a large number of processors is significant for future Petascale computing. Therefore the parallel performance of the tests focuses only on the fine group of grids, which includes a greater number of processors than 128.

Fig. 8 shows the CPU time, which is the average CPU time per iteration under different numbers of processors. It can be seen that with an increasing number of processors, the CPU time decreases overall. Metis has the most fastest falling curve. The CPU time of Metis is chosen as the reference. PT-Scotch has a similar CPU time with Metis always. When the number of processors is less than 1024, ParMetis has the lowest CPU time. When the number of processors is greater than 1024, Zoltan has the most CPU time, on average higher than Metis by about 200%. However, when the number of processors is 2048 and 4096 respectively, the CPU time spent by ParMetis is higher than Metis by about 100% and PT-Scotch by about 50%.

From the analysis of the mesh partitioning quality in Section 3, ParMetis has the lowest average number of halo cells and neighbors at numbers of processors of 2048 and 4196. One could make a deduction that ParMetis should have a shorter CPU time. What is the reason for ParMetis having the opposite performance to this deduction?

Actually the CPU time, shown in Fig. 8, includes the time for both numerical iteration and data communication. The time for numerical iteration is used to solve the variables of the fluid flow fields, such as pressure, velocity and turbulence variables (kinetic energy and dissipation rate of turbulence energy, etc.). The time for data communication is mainly to carry out data exchange between neighboring processors. The communication information will be transferred to the halo cells from the neighbors. During the communication, although the numerical iteration is suspended, the CPU is still working in performing data receiving and sending.

The data communication CPU time depends not only on the average number of halo cells and neighbors, but also the maximum number of neighbors. If one local processor has too many neighbors, it means this processor will perform data communications with the neighbors quite often. Actually this process is bi-directional. It means the process will require data communications with neighbors frequently and neighbors will ask for data communications with this processor frequently. Therefore the whole parallel computing system will be busy coping with data exchanges rather than iterations. In some extreme situations, this
Fig. 8. CPU time under different numbers of processors.

Fig. 9. Maximum number of neighbors under different numbers of processors.

Processor will delay the whole parallel processing due to it being too time-consuming.

From Fig. 9, it can be seen that ParMetis has a greater maximum number of neighbors than Metis and PT-Scotch with the number of processors being 2048 and 4096. This could be the reason why ParMetis requires more CPU time than Metis and PT-Scotch with the number of processors being 2048 and 4096.

At 8192 processors, ParMetis has a faster CPU time than PT-Scotch and is close to Metis. This is because PT-Scotch has an average number of neighbors about twice as great as ParMetis and Metis, shown in Fig. 5. Therefore PT-Scotch will spend more time on data communications than ParMetis and Metis on average.

From the studies, it can be seen that Metis and ParMetis have an outstanding capacity to perform large-scale HPC in CFD, especially with a large number of processors, for example 8192 processors. Fig. 10 shows the speed-up curves of different mesh partitioning software packages ported into Code_Saturn. From the comparisons of the speed-up curves, one can confirm the consequences of Metis and ParMetis on HPC.

To extend the investigation, especially for the limit capacity of mesh partitioning of ParMetis through Code_Saturn on the memory utilization, a more complex whole airplane case was built up. The test is carried out on a personal computer (PC) with 8 GB memory. Through the tests, it was found the top limit of capacity of mesh partitioning of ParMetis through Code_Saturn is about 50 million cells for the whole airplane case.

Fig. 11 shows the results of mesh partitioning into 4 subdomains. From Fig. 11, it can be seen that parallel mesh partitioning by ParMetis through Code_Saturn can realize the simulation of a complex geometry with 50 million cells even on a PC.

5. Conclusions

From the studies, it can be seen that several factors, such as load imbalance, halo cells and neighbors, will affect large-scale (Petascale) parallel CFD computing. The average number of cells per processor mainly affects the CPU iteration time of parallel computing. Load imbalance can affect the CPU iteration locally. The data communication time is affected by the average number of halo cells and neighbors per processor. Fewer halo cells and neighbors can reduce the burden of data communications among processors and accelerate parallel computing.

From the comparisons of HPC, it can be seen that Metis has the best high parallel performance synthetically. However, it has to use large memory to perform the mesh partitioning for large-scale parallel CFD applications due to Metis being a sequential code that has to be carried out by a single processor.

Parallel mesh partitioning software packages can remove the memory limit. However, the quality of mesh partitioning is slightly lower than Metis. Within the parallel version, it can be seen that ParMetis and PT-Scotch have similar high parallel performance. Zoltan (HSFC) has the lowest HPC among the parallel mesh partitioning software packages through Code_Saturn.

If future CFD computing wants to deal with large-scale computing, i.e. Petascale and Exascale high-performance parallel computing, it should at least pay attention to improving the mesh partitioning criteria of load imbalance and the distributions of halo
cells and neighbors on the CFD code itself and the choice of the
mesh partitioning methods. At the moment, if not considering
the memory limit, Metis can be used for large-scale parallel
CFD applications. Among the parallel mesh partitioning software
packages, ParMetis and PT-Scotch are recommended for future
Petascale and Exascale parallel CFD computing.

Acknowledgments

The author would like to thank Daresbury Laboratory of Science
and Technology Facilities Council (STFC) for providing the facilities
to support the research. He is grateful for funding support from
HECTOR CSE, The Numerical Algorithms Group Ltd. (NAG). He is
grateful for the help of Mr. Charles Moulinec (STFC Daresbury
Laboratory).

References

521.
337.
Department, Science and Technology Facilities Council, UK, Frontiers, 2010,
p. 59.
[9] C.C. Whitfield, Master Thesis of the Faculty of the Virginia Polytechnic Institute
and State University, USA, 1999.
p. 1.
p. 1.
scitech.ac.uk/about_us/Frontiers2010/index.shtml.