Mathematical Modelling and Analysis 2005. Pages 179–184
Proceedings of the 10th International Conference MMA2005&CMAM2, Trakai
 ISBN 9986-05-924-0

PREDICTING PARALLEL EFFICIENCY IN COMPUTATIONAL FLUID DYNAMICS

J. ROKICKI, M. KRAUSE AND M. WICHULSKI

Institute of Aeronautics and Applied Mechanics,
Warsaw University of Technology

Nowowiejska 24, 00-665 Warsaw, Poland

E-mail: [jack,mkrause,wichulski]@meil.pw.edu.pl

Abstract. The paper discusses the performance of parallelization in Computational Fluid Dynamics. A theoretical model is presented for parallelization based on domain decomposition. Such model allows for quantitative prediction of speed-up and efficiency. Finally, certain unresolved problems related to grid environment are reported.

Key words: CFD, parallelization, prediction of the efficiency

1. Introduction

In Computational Fluid Dynamics, flow simulations consist of a long sequence of consecutive iterations which aim to solve a very large system of nonlinear algebraic equations. The number of equations is equal to $5N$, where N denotes the number of mesh cells. These equations stem from discretized Euler/Navier-Stokes partial differential equations, governing the fluid motion.

Typical computational meshes contain millions of cells, but even such detailed spatial resolution is still not sufficient to effectively predict certain phenomena. Reduction of computational time is possible (at the present level of algorithm development), only through application of parallel processing.

The present paper deals with parallelization based on domain decomposition [2]. This means, that prior to the actual calculations, the computational mesh is divided into parts each of which is served by a different processor. The exchange of boundary information (on fictitious interfaces) occurs at every nonlinear iteration step of the main solution algorithm. This exchange is limited to the information available in the immediate neighborhood of the fictitious boundary.

The parallel efficiency of this procedure depends on:

- proper load balancing between processors,
- limiting the volume of communication,
- speed/bandwidth ratio.

In the present paper these issues are discussed and theoretical model of such parallelization is presented. The model allows for quantitative predictions of speed-up and efficiency (as a function both of the problem size and number of processors used). Load balancing issues are discussed for both structured and unstructured meshes.

The results and conclusions are formulated for the particular problem and particular CFD method (described in [1]). It is evident however that they remain valid for many problems of mathematical physics, described by nonlinear partial differential equations.

2. Parallel Algorithm

The parallel algorithm consist of the following steps:

1. Each node initializes calculations reading its own grid and restart files (the latter only in the restart mode) as well as configuration file.
2. Communication is initialized by preparing and exchanging information about the size of the future data packets (for each pair of neighbouring nodes).
3. Each node performs separately one (or more) iteration(s) of the local nonlinear solver (on its own part of the mesh).
4. The interfacial boundary information is exchanged between all neighbouring nodes. The boundary conditions at each node are updated.
5. The convergence criterion is checked at each node, subsequently reduced (logical AND) and the result is scattered to all nodes. If FALSE is returned, the control goes back to step 3.
6. Communication link is terminated and each node stores the corresponding restart and solution output files.

In the above, the neighbourhood is understood in the sense of mesh partition topology. The nodes are considered as neighbours if their meshes have a common interface (or overlap as it was the case in [2]).

The main computational effort corresponds to the step 3 of the algorithm. This effort can be assumed proportional to the number of mesh cells, perhaps with the exception of the vicinity of the physical boundary, where the computational cost can be higher (this however will be disregarded in the further analysis below).

The main communication effort is located in step 4 and is proportional to the length of the data packet (the latency is not significant in this case). The communication presented in step 5 can also be neglected as its volume is negligible in comparison with the one in step 4 (at least for a moderate number of processors used in computations).

The workload of each processor is proportional to the number of cells presented in its local mesh. Thus with N denoting the total number of mesh cells and assuming ideal load balancing we can estimate the computation time on the L-processor system by:

$$\tau_{\mathrm{COMP}}[L] \;=\; \frac{A\,N}{L}, \tag{2.1}$$

where A stands for the proportionality factor (which can depend both on the algorithm as well as on the processor speed).

3. Communication Models

Two possible communication models are considered here.

The first model is based on the *master-worker* concept. In this approach one of the nodes, *master* is responsible for gathering, re-calculating and scattering data that is exchanged between nodes (*workers*) (see Fig. 1).

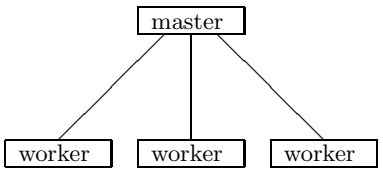

Figure 1. *Master-worker* communication.

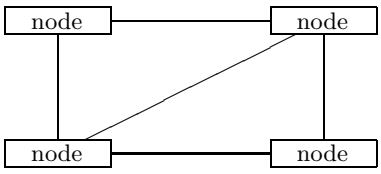

Figure 2. *Neighbour to neighbour* communication.

In contrast *neighbour to neighbour* approach assumes that communication exists between these nodes which have common interface, without the need for a centralized control (see Fig. 2).

Here simultaneous exchange of data between various nodes allows for significant reduction of communication overhead. Both approaches can be rooted in the solution algorithm. Yet the success of implementation heavily depends on the communication hardware.

In particular in clusters, the computing nodes are in fact connected via a single switching device. Therefore efficiency of *neighbour to neighbour* communication will be limited by the switch ability to simultaneously transmit data between separate pairs of nodes. This hardware arrangement typical for clusters is present also in some shared memory architectures where fast switching devices connect computing nodes with separate memory banks (e.g., Compaq ES40).

4. Performance Analysis

The performance of algorithm described in Section 2. can be evaluated by considering average time $\tau[L]$ necessary to perform single iteration (steps 3,

4, 5) (L denotes number of processors). In the present analysis, it is tacitly assumed that the overall computational effort and total number of iterations do not depend on the number of processors.

4.1. Communication time

Communication time depends on the range of factors and in particular on: (i) numerical algorithm, (ii) dimension of the computational problem (2D or 3D), (iii) quality of partition into subdomains, (iv) communication model *master-worker* or *neighbour to neighbour*, (v) hardware properties.

All of these issues cannot be reasonably included into the present analysis. Instead we aim at obtaining some optimal estimation, e.g., when partition into subdomains minimizes and evenly distributes the communication volume.

We assume that the communication volume assigned to each processor is proportional to the number of interfacial (boundary) cells in each local mesh [1]. In particular in 2D space, the checker-board partition of a rectangular domain results in a number of interfacial cells (per computing node) proportional to $\sqrt{N/L}$. In contrast partition into stripes increases this result to $\sqrt{N}$.

In 3D space the partition of the cubic domain into smaller cubes gives the number of interfacial cells (per computing node) proportional to $(N/L)^{2/3}$. Again this partition can be regarded as optimal.

The communication time can thus be estimated as:

$$\tau_{\mathrm{COMM}}[L] = B_{d,\mu} \left(\frac{N}{L}\right)^{\frac{d-1}{d}} L^{\mu} + \tau_{\mathrm{LAT}}$$

where in the above: (i) τ_{LAT} stands for latency, (ii) $d = 2, 3$ denotes the dimension of the problem, (iii) μ is equal 0 in case of the successful *neighbour to neighbour* model and is equal 1 for the *master-worker* model, (iv) $B_{d,\mu}$ is a proportionality constant depending both on the space dimension as well as on the communication model.

It is worth stressing that for *master-worker* approach the communication time depends on the total communication volume (hence $\mu = 1$) whereas for the *neighbour to neighbour* model only local communication has to be taken into account (hence $\mu = 0$). As a result the communication time increases with the number of processors for the first approach and decreases for the latter approach.

4.2. Parallel efficiency

Taking into account all considered subcases parallel efficiency can be estimated as:

$$\eta[L] = \frac{1}{L} \frac{\tau_{\mathrm{COMP}}[1]}{\tau_{\mathrm{COMP}}[L] + \tau_{\mathrm{COMM}}[L]} = \frac{1}{\beta[L] + \frac{L}{AN}\left[BN^{1-1/d}L^{\mu-1/d} + \tau_{\mathrm{LAT}}\right]}, \tag{4.1}$$

where for clarity reasons subscripts of the proportionality constant B are dropped.

For large problems when latency is negligible and by assuming ideal load balancing one obtains a simple formula for efficiency:

$$\eta[L] = \frac{1}{1 + L^{1+\mu-1/d} N^{-1/d} B/A}. \tag{4.2}$$

Formula (4.2) allows us in turn to obtain the isoefficiency function of the parallel algorithm

$$N[\eta, L] = \left(\frac{B}{A}\right)^d L^{d(1+\mu)-1} \left(\frac{\eta}{\eta - 1}\right)^d. \tag{4.3}$$

In particular, for the *master-worker* model keeping fixed efficiency is possible by increasing problem size N proportionally to L^3 in 2D and to L^5 in 3D. For the *neighbour to neighbour* model the same effect is obtained if problem size grows as L^1 in 2D and to L^2 in 3D. This is why the latter seems so attractive from the point of view of the parallel program design.

The current formulas (4.2) and (4.3) form a generalization of the one presented and verified in [2] for $d = 2$, $\mu = 1$ and for $L = 2, \ldots, 21$.

5. Parallelization in Grid Environment

Traditional parallelization is characterized by the requirement to distribute computations over separate processors belonging to a single system. In such system the RAM memory is often divided and accessible only locally.

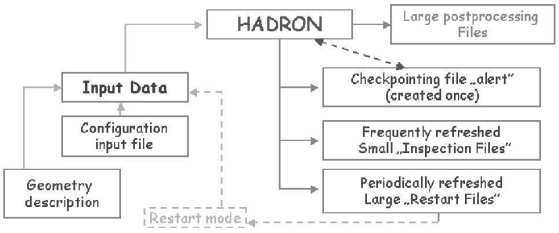

Figure 3. File system used by the HADRON code for checkpointing and to facilitate restart mechanism.

In contrast in grid environment not only RAM but also a disk memory is distributed over two or more local systems. This itself is not a major drawback since Broker is designed to replicate and gather all data for the user.

When computations, however, take a very long time to complete, the application is usually equipped with some restart mechanism (see Fig. 3 which presents the file system used by the CFD HADRON). This means (as described in Section 2) that each processor stores to its own file, in regular

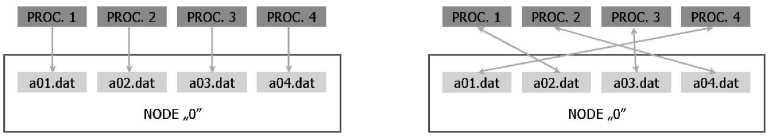

Figure 4. Regular (left) and restart (right) modes on system with common disk memory.

intervals, a huge amount of data – the file name is often parameterized by the process number (see Fig. 4). When the computations are terminated (either by accident, or because of encountering error or by the action of the user) these files are gathered by the Broker and transferred to a single location.

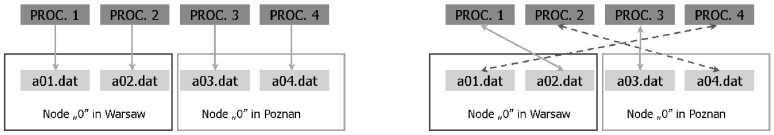

Figure 5. Regular (left) and restart (right) modes on system with distributed disk memory (grid) – dashed lines describe incorrectly positioned files.

When the user decides to restart the computations the Broker should perform the inverse operation. In order to operate correctly each new process should have its own restart file available locally (the name of the file should match the process number). This number however is neither available in advance to the Broker, nor can be guessed by the application itself. As a consequence the application will not work correctly in the restart mode (see Fig. 5).

It is of course possible to add a mechanism to the application, which recognizes existing files and re-numbers computing nodes. This however is not very practical and should be solved in a different way.

Still other issues that remain open are related to the control of the running application (e.g., inspection of output files as well as checkpointing).

References

[1] D. Drikakis, J. Majewski, J. Rokicki and J. Żółtak. Investigation of blending-function-based overlapping-grid technique for compressible flows. *Computer Methods in Applied Mechanics and Engineering*, **190**(39), 5173 – 5195, 2001.

[2] J. Rokicki, J. Żółtak, D. Drikakis and J. Majewski. Parallel performance of overlapping mesh techniques for compressible flows. *Future Generation Computer Systems*, **1**(19), 3 – 15, 2001.