Using Distributed Computing in Computational Fluid Dynamics

Zoran Constantinescu[a] , Jens Holmen[b] and Pavel Petrovic[c]

[a]Department of Computer and Information Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway, zoran@idi.ntnu.no

[b]SINTEF Applied Mathematics, N-7465 Trondheim, Norway, Jens.Holmen@math.sintef.no

[c]Department of Computer and Information Sciences, Norwegian University of Science and Technology, N-7491 Trondheim, Norway, petrovic@idi.ntnu.no

Keywords: distributed computing, parallel solver, incompressible flow

1. Introduction

Advanced computation and simulation in science and engineering constitute an important area in rapid growth. The background for this is the continuing development of efficient hardware, and the need for more realistic simulations of mathematical models. Often, large scale simulations are performed with complex geometries using sophisticated numerical techniques. Due to the high amount of computational resources needed, these simulations are usually done using parallel computer architecturesr: on supercomputers or dedicated commodity clusters. However, the initial high costs of infrastructure and later maintenance and upgrade costs are limiting the large scale availability of these environments. An alternative, cost effective approach is to use the computational resources of the existing desktop systems (very often powerful PCs) in a distributed computing environment.

This paper presents some first experiences with running a typical Computational Fluid Dynamics problem in a distributed computing environment. The QADPZ software system for distributed computing is described, as is the numerical method for solving the incompressible Navier–Stokes equations, which is used in a test case for the system. Further, a description of the implementation of both the simulation software and QADPZ system are given. An evaluation of the performance of the system is presented, by comparing it with running the same simulation on a typical dedicated cluster environment.

2. Distributed Computing

In many cases the computing power and memory resources of a single desktop computer is not enough for running large scale numerical simulations of realistic problems in science and engineering. Both large running times of the simulations and insufficient disk/memory resources on a desktop are requiring a parallel approach.

Distributed computing harnesses the idle processing cycles of available workstations in a local network and makes them available for participating in computationally intensive problems that would otherwise require a supercomputer or a dedicated cluster of computers. The idea is that most of the computers in offices and labs are not used at their full computing capacity most of the time. Such an

environment consists of a pool of computational nodes, usually PCs with their own CPU, memory, and storage, and one or more central servers responsible for managing these resources. A given application would be split into smaller tasks, which can be independent from each other or not, and executed on the individual nodes.

QADPZ [3] is an open-source system [2] for distributed computing on non-dedicated desktop computers. This is an ongoing research project in our group, which is investigating the possibilities of using existing computing resources in a local network (e.g. computers from labs and offices) for computationally intensive tasks. QADPZ is an multithreaded, object-oriented system, with support for multiple users and different operating systems. Supported platforms are Linux, Windows, and most of the UNIX versions. The design of the system was made in such way to minimize the effort needed to install and use the software. Minimal knowledge about the system's internals is required in order to be able to use it.

QADPZ consists mainly of three types of computers: slaves, masters, and clients. A slave represents one of the computing desktop PCs, running a small software, which is responsible for reporting its current status to the central master computer and also for starting the actual computational task sent by the master. Only slaves which are not used interactively by other users can execute tasks. One parallel application can consist of multiple tasks running on different slave computers (possibly using different operating systems), and which communicate to each other using message passing. The client is a computer from where a user interacts with the system, describing the applications to be executed in the system.

3. Numerical Solution Method

To solve the incompressible Navier–Stokes equations we use a version the well–known projection method, in which equations for the velocity components and pressure are solved sequentially at each time step. Solving the discretized, fully coupled equations can be very expensive due to the nested iterations, and this decoupling procedure is found to be computationally efficient for transient problems, particularly for higher Reynolds numbers [11]. In general, the separation of pressure and velocity can be performed on both the continuous equations and the discretized equations [5,9,10]. While the latter is attractive due to the straight forward interpretation of boundary conditions, these methods give a significantly more complicated pressure equation and we therefore prefer a splitting at the differential level.

The operator of the momentum equation is split in two parts, resulting in the following semi–discretized equations when the implicit Crank–Nicolson method combined with a second order Adams–Bashforth method is used to advance the solution in time:

$$\frac{\tilde{\mathbf{u}} - \mathbf{u}^n}{\delta t} = -\frac{3}{2}\left(\mathbf{u}^n \cdot \nabla\right)\mathbf{u}^n + \frac{1}{2}\left(\mathbf{u}^{n-1} \cdot \nabla\right)\mathbf{u}^{n-1} + \frac{1}{2}\nabla \cdot \left(\nu\nabla\mathbf{u}^{n+1} + \nu\nabla\mathbf{u}^n\right) + \mathbf{f}^{\mathbf{n+1}}, \quad (1)$$

$$\frac{\mathbf{u}^{n+1} - \tilde{\mathbf{u}}}{\delta t} = -\nabla p^{n+1}. \quad (2)$$

$\tilde{\mathbf{u}}$ is an intermediate quantity in the computation without any particular physical meaning. Taking the divergence of (2) and applying the continuity constraint gives a Poisson equation for the pressure;

$$\delta t \Delta p^{n+1} = \nabla\tilde{\mathbf{u}}. \quad (3)$$

The equations are solved sequentially as (1), (3), and (2).

The Galerkin finite element method is used to discretize in space. The pressure is fixed in one node to ensure a unique solution, and has homogeneous Neumann conditions on all boundaries.

Although this is an approximation to the theoretical boundary conditions [7], we have found it to be sufficiently accurate for our purpose. $\mathbf{u}$ and $\tilde{\mathbf{u}}$ are given the same boundary conditions.

The pressure Poisson equation is preconditioned with a mixed additive/multiplicative domain decomposition method: one iteration of overlapping Schwartz combined with a coarse grid correction. The first velocity step (1) has no preconditioner, while the updating step (2) is preconditioned with the lumped mass matrix. Conjugated gradients and BiCGStab are used as linear solvers.

4. Implementation

Implementation of the flow solver was done is C++ using the object oriented numerical library Diffpack. A parallel version of Diffpack [6] was used, which is based on a standard Message Passing Interface (MPI) for communication. We used Linux as a development platform.

The QADPZ system was developed in C++ simultaneously in Linux and Windows environments. This made the integration of different platforms much easier and helped us to find the flaws in the source code faster. Communication in QADPZ is based on UDP, an unreliable communication protocol, in which packets are not guaranteed to arrive and if they do, they may arrive out of order. The advantage of UDP over TCP/IP is that UDP is fast, reducing the connection setup and teardown overhead, and is connectionless, making the scalability of the system better. We implemented a higher level reliable protocol, which is message based, allowing us to overcome the disadvantages of UDP. This is based on message confirmation. Each message contains a sequence number, and each time it is sent, it is followed by an acknowledgment from the receiver. Each sent or received message is accounted, together with the corresponding acknowledge, and in case of not receiving an acknowledgment, the message is resent a few more times. An acknowledge and a normal message can be combined into one message to reduce the network traffic. This layer permits both synchronous and asynchronous message sending.

An MPI library with a subset of the most use MPI calls was implemented on top of QADPZ's communication protocol. This QADPZ-MPI library allows us to use QADPZ as a straightforward replacement communication system for most of the applications which are based on MPI.

5. Evaluation and Results

A series of tests were performed on a dedicated cluster of 40 PCs, with Athlon XP 1.46GHz CPU, 1 GByte memory, and 100 MBps network interconnection between nodes, running a Linux distribution. First, we used the original implementation of the solver software, which is using MPICH as a parallel communication protocol, to run the cluster version of the simulation. Second, we recompiled the solver using the QADPZ-MPI library to create the distributed computing version of the simulation. The solver was run using exactly the same computers from the cluster (i.e. identical hardware setup). A third test case was using a pool of 8 computers with similar hardware specifications. These computers were ordinary desktop PCs from our labs, connected to our LAN, together with other computers. Simulations were done in two different times of the day: during the night, when network traffic in the LAN is minimal, and during working hours, when LAN traffic is much higher.

The results are from simulating some time steps of an oscillating flow around a fixed cylinder in three dimensions. The grid has 81600 nodes and is made of 8-node isoparametric elements, while the coarse mesh (used for pressure preconditiong) has approximately 2000 nodes.

6. Conclusions

The use of already existing hardware resources, like for example desktop PCs, in a distributed computing environment has a tremendous potential of providing a computing platform on which large scale Computational Fluid Dynamics simulations can be executed in a time comparable with supercomputers and dedicated clusters. This approach is also a very affordable alternative, as investment costs and later maintenance are minimal compared to other systems.

REFERENCES

1. Diffpack homepage. http://www.diffpack.com.
2. QADPZ homepage. http://qadpz.sourceforge.net.
3. Zoran Constantinescu, Pavel Petrovic, and Atle Pedersen. Q2ADPZ * An Open system for distributed computing. In *NordU 2002, The 4th EurOpen/USENIX Conference, Helsinki*, 2002.
4. Vijay K. Garg. *Elements of Distributed Computing*. Wiley-Interscience, 2002.
5. P.M. Gresho and R.L. Sani. *Incompressible flow and the finite element method: advection–diffusion and isothermal laminar flow*. Wiley, Chichester, 1998.
6. H. P. Langtangen. *Computational partial differential equations, numerical methods and Diffpack programming*, volume 2 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin/Heidelberg, 1999.
7. S.A. Orszag, M. Israeli, and M.O. Deville. Boundary conditions for incompressible flows. *J. Scient. Comp.*, 1(1):75–111, 1986.
8. Peter S. Pacheco. *Parallel Programming with MPI*. Morgan Kaufmann Publishers, Inc., 1997.
9. J.B. Perot. An analysis of the fractional step method. *J. Comput. Phys.*, 108:51–58, 1993.
10. A. Quarteroni, F. Saleri, and A. Veneziani. Factorization methods for the numerical approximation of Navier–Stokes equations. *Comput. Methods Appl. Mech. Engrg.*, 188:505–526, 2000.
11. S. Turek. *Efficient solvers for incompressible flow problems: an algorithmic and computational approach*, volume 6 of *Lecture Notes in Computational Science and Engineering*. Springer, Berlin, 1999.