ORNL/M-3986

MPI Implementation of CFD Program PHOENICS

Srđan Šimunović and Thomas Zacharia [1]
Material Process Modeling Group, Metals and Ceramics Division
Oak Ridge National Laboratory
P.O. Box 2008, Oak Ridge, TN 37831–6140

October 26, 1994

[1] The research was sponsored by the Division of Material Sciences, U.S. Department of Energy, under contract DE-AC05-84OR21400 with Martin Marietta Energy Systems, Inc.

[illegible]

Abstract

This document describes the work on Message Passing Interface (MPI) standard implementation of the parallel version of the solver EARTH for the Computational Fluid Dynamics (CFD) program PHOENICS. The Intel Paragon NX and MPI versions of the program have been developed and tested on massively parallel (MP) supercomputers Intel Paragon XP/S 5, XP/S 35, and Kendall Square Research (KSR) supercomputers at Oak Ridge National Laboratory (ORNL). The preliminary testing results of the developed program have shown scalable performance for reasonably sized computational domains.

1 Introduction

PHOENICS is a suite of computational analysis programs that are used for simulation of fluid flow, heat transfer, and dynamical reaction processes [1]. The program consists of three main modules, pre–processor (SATELLITE), solver (EARTH), and post–processor (PHOTON). The computational requirements of the solution module are the most demanding part of an analysis process and it has been redesigned by the CHAM development group for parallel implementation [2] on the PARSYTEC transputer [3] based architecture. PARSYTEC computers are built from interconnected INMOS T800 series transputers. New generation of the PARSYTEC computers, GigaCube GC–2, will incorporate T9000 transputer and C104 router chip. The transputer–router system was not available at the time of code development causing the resulting software to be more dependent on the underlying hardware design. The details of the PARSYTEC architecture may be found in, for example, Reference 4.

Message Passing Interface (MPI) standard has been recently established [5] in order to provide a stable environment for parallel program and operating system development. The variety of hardware designs and communication protocols make porting of parallel computer programs to different computer architectures a considerable effort that does not significantly increase value of the software. Writing a parallel program using standard conventions for communication alleviates the programing task and moves the design of communication to the domain of the operating system designer.

Implementation of MPI version of PHOENICS makes the computational tool portable to wide range of parallel machines and enables the use of high performance computing for large scale computational simulations. MPI libraries are available on several parallel architectures making the program usable across different architectures as well as on heterogeneous computer networks.

2 PARSYTEC Implementation of PHOENICS Module EARTH

As a principal method for exploiting concurrent processing, the domain decomposition technique has been employed for PHOENICS module EARTH. The program uses one dimensional domain decomposition in z direction, decomposing the computational domain into slabs of approximately equal size. Each slab is assigned to a processor that has its identification number (id) equal to the slab ordinal number with respect to slab position along z axis, the first slab having the ordinal number 0. Implemeted soulution algorithm requires a processor to communicate with the processors that are assigned to the adjacent slabs, and with all the processors in the active topology in order to perform global operations. Such a computational domain assignment and communication pattern makes a line processor topology suitable for efficient processing. A ring topology could increase performance of global operations but because the PARSYTEC transputer machine allows communication between neighboring transputers only, the ring topology was not implemented.

Characteristic of the domain decomposition approach is that for a given problem size there is a limit of the number of processors that can be effectively used. After the computational effort, which is proportional to sub–domain size, becomes comparable to communication, the program execution time can not be further reduced by the increasing number of processors. Therefore, the communication phase of a program must be efficiently designed in order to delay as much as possible the occurence of the utilization horizon. The PARSYTEC operating system called PARIX provides limited communication capabilities that consists of primitives for synchronous message passing and organization of virtual transputer topologies. The cut–through routing message mode, although envisioned in current PARSYTEC design, was not yet available because of the lack of a routing processor [4]. This fact forces all the communication except with adjacent transputers to be done in software, effectively implementing store–and–forward message routing which was shown to be inferior to the cut–through approach. The new PARSYTEC computers that are planned to employ transputers–routers design should allow for general message passing and more complex transputer topologies.

The central part of EARTH solver is a linear equation solver. The solution algorithm implemented in the serial version was not suitable for parallelization and was replaced by conjugate residual accelerator [6] combined with Jacobi iteration method. The communication required by the solver includes communication with adjacent processors in a line topology, global summation of inner products and determination of maximum and minimum value across all processors. The nearest neighbor communication is not considerably influenced by a line topology of processors but the global operations are inevitably slower because of the store–and–forward routing.

3 NX Implementation

As an initial approach, a translator library was developed that transformed PARIX communication function calls to Intel NX calls without intervention in the program. Essential message identifiers in PARIX were mapped to their NX counterparts. The implications of such an implementation were analyzed and program consistency and performance were evaluated.

The initial version of NX implementation was obtained by translating communication primitives to NX calls without unique message type specification. In this document, only the communication function essential for program implementation are provided. The more complete library translating PARIX to NX functions are available from authors. The procedure for building line processor topology, `makepipe`, in PARIX has been translated as:

```
integer function makepipe(reqid,size,xmin,xmax,ymin,ymax,zmin,zmax,id,link)
integer reqid,size,xmin,xmax,ymin,ymax,zmin,zmax,id,link(2)
external mynode
integer mynode
id=mynode()
link(1)=id-1
link(2)=id+1
```

```
if(id.eq.(size-1))then
     link(2)=-1
endif
makepipe=reqid
return
end
```

Function makepipe creates the communication structure with size processors. The processor partition will be mapped as described by xmin, xmax, ymin, ymax, zmin, and zmax. In order to distinguish links while establishing several topologies simultaneously, reqid is used. The PARIX functions send and recv were translated to corresponding NX call as:

```
integer function send(topid,loglinkid,data,size)
integer topid,loglinkid,data(*),size
call csend(topid,data,size,loglinkid,0)
send = size
return
end

integer function recv(topid,loglinkid,data,size)
integer topid,loglinkid,data(*),size
call crecv(topid,data,size)
recv = size
return
end
```

Variable topid is an identifier of the active topology and it is same for all nodes; loglinkid denotes a link to communication channel over which message is to be delivered; data and size are message buffer and its size in bytes, respectively. In NX implementation, topid was initialized to 0, and loglinkid was defined in send and recv as an identification number of a processor which receives or sends the message, respectively. This version of the program was causing the Intel Paragon to hang at random number of iterations, which was especially frequent for runs on larger number of processors (for the example problem provided in this document, > 16). The cause of this problem lays in the fact that PARIX implemetation of the program does not provide for definition of message types. Such communication design is feasible in transputer based computers where messages are pipelined into specific channels, longlinkid, and the order of communication is maintained. Channels are unidirectional and message passing is unbuffered so that the sending process must wait until the receiver is ready and vice versa. However, for the Intel Paragon the message type is the identificator of a message and it has to be unique. If the message type is neglected, the resulting communication may be incoherent. To illustrate such behavior assume that a processor receives two messages from its neighboring processors that have the same message type, in our case 0. The NX function crecv can not distinguish the origins of the received messages and although enough messages are in the receiving buffer, the final destination of data in a program will depend on the message arrival time and as such will be highly unreliable.

The initial experience with the program indicated that more stringent management of message passing may solve the problem of message origin–destination ambiguity. Two versions of the message passing control were implemented. It the first version NX call **crecvx** was used to specify the message source with no unique message type. The second approach used **crecv** calls and message types were set to node ids. Both versions of the program gave consistent results for test runs. The communication primitives for these two versions are:

```
1.      integer function send(topid,loglinkid,data,size)
        integer topid,loglinkid,data(*),size
        call csend(topid,data,size,loglinkid,0)
        send = size
        return
        end

        integer function recv(topid,loglinkid,data,size)
        integer topid,loglinkid,data(*),size,info(8)
        call crecvx(topid,data,size,loglinkid,0,info)
        recv = size
        return
        end

2.      integer function send(topid,loglinkid,data,size)
        integer topid,loglinkid,data(*),size
        common/parix/iam
        call csend(iam,data,size,loglinkid,0)
        send = size
        return
        end

        integer function recv(topid,loglinkid,data,size)
        integer topid,loglinkid,data(*),size
        call crecv(loglinkid,data,size)
        recv = size
        return
        end
```

where **iam** denotes processors' identification number in the communication group which has to be previously initialized. The authors preference was on the second version because it required less complex NX call and as such should be faster. Although resolving problems with communication incoherence in EARTH program, such approaches may not be valid on some message passing oriented parallel machines. A situation can be envisioned when even the use of source specific addressing would result in communication incoherence because the processors may be physically distant and messages do not have to arrive in the order that they were sent. However, this may happen only on the architectures that allow for dynamic message routing which was not the case with the current target machines.

4 I/O Operations

During the process of NX implementation of program EARTH certain I/O deficiencies of the PARIX version were manifested. The parallel implementation of the program allocates first processor in a line topology for I/O operations. The I/O processor broadcasts the input data to all processors in active topology. The broadcast function that were used for the input data distribution synchronized the program, thereby constantly controlling the amount of data present in the communication network. The output phase, however, could not be consistently used as originally designed since the size and dynamics of communication exceeded the message buffer resources of the XP/S. The message buffer size overflow was the result of the adopted output strategy where each processor formats its data for the final output and sends it in character form to the adjacent lower numbered processor. In addition, a processor also routes the messages received from higher numbered processors to its lower neighbor. Although elegant, this approach increases the amount of data that needs to be transferred. A floating point number that needs four bytes of memory space is now converted to a string of characters each of a size of one byte. If the number of significant digits is large enough, the data size is multiplied without adding any information. Additionally, the pipeline mode in which data is sent to I/O processor is causing heavy traffic in the lower numbered processors close to I/O processor. The synchronizing character of the send operation in transputer based machines is keeping this traffic constant. The flow of output messages using PARIX system is controllable since the machine is synchronized by both send and receive calls, and a programmer can be sure that the flow of messages is consistent. The **send** call in PARIX is synchronous, the transputer waits until the message is delivered to its destination whereas in NX **csend** waits only until the message is delivered to the network. This may result in too many messages sent to a certain processor whose buffer can not handle such a traffic.

Because of the above deficiencies, the output phase of the program has been modified. Each processor directly sends its data to the I/O processor. When using Intel mesh topology messages may arrive at different times and the data in the output pipeline may become unordered. In order to maintain the consistency of data flow, the processors send data in the increasing order of their processor ids. The output schedule is maintained by synchronous message passing of control messages between processors. When a processor completes sending its output data to I/O processor it sends a control message to the processor with the immediately higher numbered id, triggering its output phase.

5 MPI Implementation

The overall objective of this project was to produce a parallel implemetation of a CFD program that may work on as general parallel machine as possible. Having this goal in mind, the direct translator approach with its potential shortcomings was abandoned in favor of unique message typesetting. An MPI implementation was developed on the basis and experiences with communication patterns investigated in NX implementations. The communication driven by specification of the message source used in the final NX version has been replaced by the message type based control. The source and destination were determined for every communication and unique message types were assigned to send–receive pairs. Such an implementation allows for a more flexible communication design and asynchronous message passing which will be more advantageous for analysis of large computational grids using large number of processors. Global operations that were developed in EARTH module for PARIX implementation were replaced with MPI global function calls. The developed program was tested and have produced the same results as NX version.

Basic synchronous communication using MPI_SEND and MPI_RECV were used to replace `csend` and `recv` NX calls. There were no requirements for dynamic group formations so only the basic group consisting of all the processors in the active processor topology was used. Asynchronous message passing using MPI_ISEND and MPI_IRECV has also been implemented in conjugate residual solver to alleviate the most intensive communication operations. The global operation calls were implemented using MPI_ALLREDUCE since the information from global operations had to be available to every processor. Function MPI_BCAST was used for one–to–all communication in the input phase of the program, and the processors were synchronized using MPI_BARRIER function. The details of the MPI library implementation are encapsulated in C functions that are interfaced with FORTRAN source with function calls only. MPI library for NX environment is available from authors.

6 Example

A simple CFD model, simulating flow of air around a group of buildings was used to assess the validity and the performance of the MPI implementation of the program. The computational grid of the test problem consists of 15, 10 and 192 cells in x, y and z directions, respectively. The computations were performed on massively parallel computers at Center for Computational Sciences at ORNL. The Intel Paragon [4] XP/S 5, XP/S 35, and Kendall Square Research 1 [4] timings for one iteration of the program solution phase are given in Figure 1. Only the results for the MPI version of the program are presented since the NX version gives essentially the same results. Because of the line processor topology, predominantly nearest neighbor communication and the small amount of data used in global operations, the communication deficiencies of the NX version were not as apparent as they would be for a multi–directional domain decomposition approach when global communications become much more complex. The input files were decomposed for 4, 8, 12, 16, 32, 64, and 96 processors. Speedup information using 4 processors as the baseline timing was shown in Figure-2. It can be seen that the increase in speedup is negligible after 32 processors. The small computational domain makes the communication a predominant operation and the further increase in number of processors may even result in an increased computation time. For example, in the given CFD problem using 96 processors, the computational domain is reduced to 2 z intervals and data from 4 intervals have to be interchanged with the adjacent processors.

7 Conclusions

The Intel Paragon NX and the general MPI implementation of the parallel solver for CFD program PHOENICS has been successfully completed. The MPI implementation is developed for general message passing environment. The communication part of the program was rewritten in order to produce a consistent communication. The I/O phase has been modified in order to avoid network congestion and data incoherence. The timing results on ORNL supercomputers illustrate the potential of the developed program for the analysis of large scale CFD problems.

8 Acknowledgements

This research has been supported by the Division of Material Sciences, U.S. Department of Energy, under contract DE–AC05–84OR21400 with Martin Marietta Energy Systems, Inc.

References

[1] D. B. Spalding. Mathematical modeling of fluid–mechanics, heat–transfer and chemical–reaction processes. Technical Report CFDU Report HTS/809/1, Imperial College, 1980.

[2] N. D. Baltas and D. B. Spalding. MIMD PHOENICS: Porting a computational fluid dynamics application to a distributed memory MIMD computer. In *International Conference on Massively Parallel Processing*, TU Delft, Netherlands, 1994.

[3] M. D. May, P. W. Thompson, and P. H. Welch. *Networks, Routers and Transputers: Function, Performance and Applications.* INMOS Limited, 1993.

[4] K. E. Gates and W. P. Petersen. A techincal description of some parallel computers. Technical report, Swiss Federal Institute of Technology (ETH), 1993.

[5] Message Passing Interface Forum. MPI: A message–passing–interface standard. Technical report, University of Tennessee, 1994.

[6] P. J. O'Rourke and A. A. Amsden. Implementation of a conjugate residual iteration in the KIVA computer program. Technical Report LA-I0849-Ms, Los Alamos National Laboratory, 1986.

List of Figures

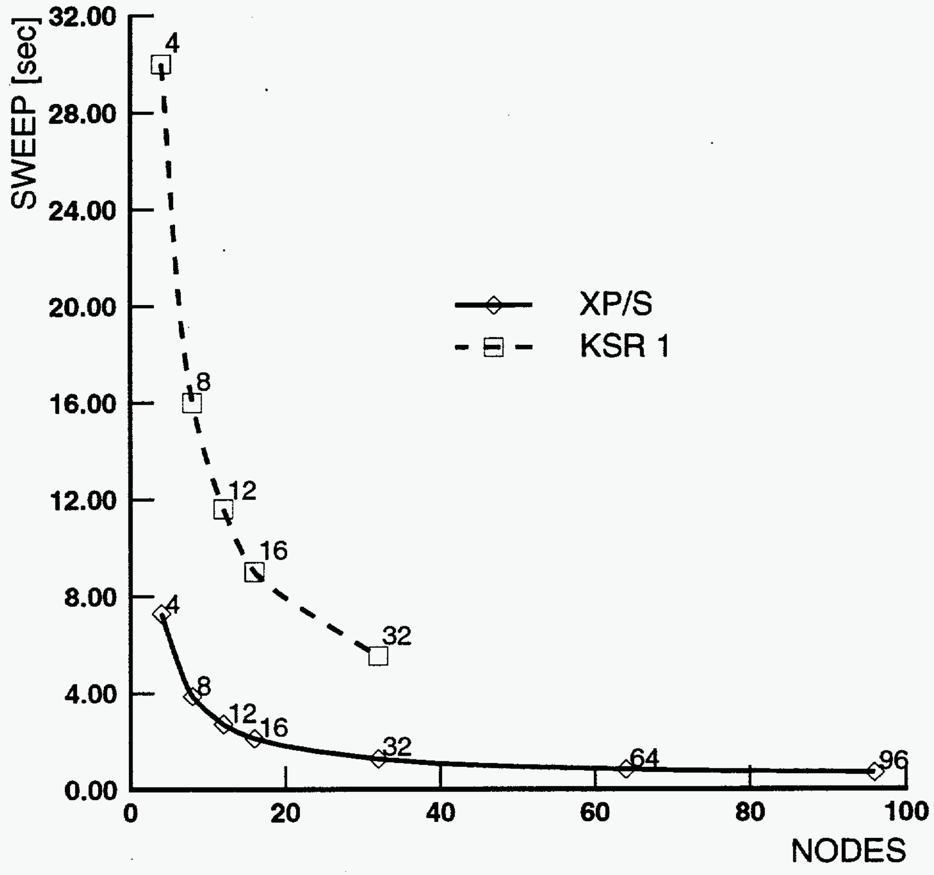

Figure 1: Timing Results for EARTH Solver

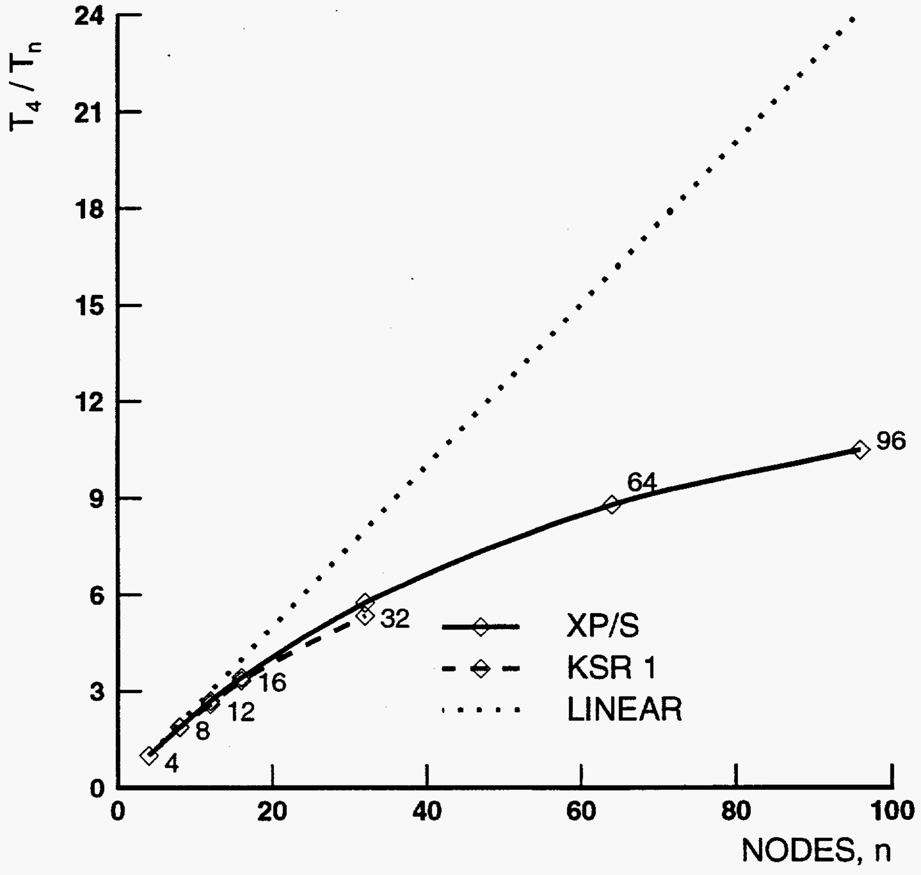

Figure 2: Scalling of EARTH Solver