'/1 · ~1 /

044226

NASA Contractor Report 201722

Asynchronous Communication of TLNS3DMB Boundary Exchange

Dana P. Hammond
Computer Sciences Corporation, Hampton, Virginia

Contract NAS1-20431

June 1997

National Aeronautics and
Space Administration
Langley Research Center
Hampton, Virginia 23681-0001

Abstract

This paper describes the recognition of implicit serialization due to coarse-grain, synchronous communication and demonstrates the conversion to asynchronous communication for the exchange of boundary condition information in the Thin-Layer Navier Stokes 3-Dimensional Multi Block (TLNS3DMB) code. The implementation details of using asynchronous communication is provided including buffer allocation, message identification, and barrier control. The IBM SP2 was used for the tests presented.

Introduction

A coarse-grained parallel version of Thin-Layer Navier Stokes 3-Dimensional Multi Block (TLNS3DMB) was developed by Dr. Veer Vatsa and Bruce Wedan [1]. The goal was to develop a parallel and scalable version with minimal code changes to the sequential code. Since the code is multi-block, it was structured to readily incorporate coarse-grain parallelism, wherein one or more blocks of the global grid are assigned to each processor of a workstation cluster or parallel computer. More than one block may reside on a processor, but coarse granularity implies that a block cannot be split among multiple processors.

The developed code was written to run in either a serial or distributed environment. The user chooses between two underlying message passing libraries, either Message Passing Interface (MPI) or Parallel Virtual Machine (PVM). The primary considerations of the parallelization task were minimal code changes to the original sequential version, and the capability to generate sequential, distributed, and parallel versions from one code using simple compiler directives.

The parallel implementation of TLNS3DMB maintained almost all of its original features, yet yielded significant performance speedups when used in a high-performance environment. Linear speedups are approachable if the biggest blocks are partitioned to allow good load (i.e., data) balancing. The distributed version displays a near-linear speedup with the number of processors on the IBM SP2, Intel Paragon, and a heterogeneous cluster of workstations (SGI, SUN, etc.).

Load Balancing and Scalability

Program performance depends largely on the effective mapping of blocks to the processors. The more load balanced the workload among the processors, the smaller the idle time for nodes with smaller blocks. This results in a reduction of the overall time and memory requirements. To assist the TLNS3DMB user in achieving the most effective use of the resources, a program called <u>Mapper</u> has been developed and is available in the TLNS3DMB executables directory (i.e., tlns3d-dist/RS6K).

The <u>Mapper</u> provides information on the grid block sizes and the effectiveness of using a varying number of processors for the user's problem. The Mapper requires the standard TLNS3D-MB input file as input, and reads the associated grid file for the problem. The following example illustrates the use of Mapper and the results for the 8block case:

```
cd tlns3d-dist/8block
 ../RS6K/mapper < m6w8b.inp

 8-block m6 wing
 formatted grid from m6w8b.gr.fmt

Grid Block Sizes
 block    imax    jmax    kmax      total
 -----    ----    ----    ----      -----
     1     137      25      17      58225
     2     137      25      17      58225
     3      57      25      17      24225
     4     57       25       17       24225
     5    137       25       17       58225
     6    137       25       17       58225
     7     57       25       17       24225
     8     57       25       17       24225

 nodes    maxpts    minpts    avgpts    %avgdev    megawords    exetime
 -----    ------    ------    ------    -------    ---------    -------
     1    329800    329800    329800       .000       18.469      1.000
     2    164900    164900    164900       .000        9.234       .500
     3    116450    106675    109933      3.952        6.521       .353
     4     82450     82450     82450       .000        4.617       .250
     5     82450     58225     65960     14.072        4.617       .250
     6     58225     48450     54966      7.904        3.261       .177
     7     58225     24225     47114     27.762        3.261       .177
     8     58225     24225     41225     41.237        3.261       .177
```

This case has eight blocks, four with 58225 points and four with 24225 points. If only one processor is used, then over 18 megawords would be required, and the execution time would be 1.00 (no reduction as a result of parallelization). If two processors are used, the memory requirements per node would be half and the execution time half (.500). When using 4 nodes, each node would process a 58225 and 24225 block (as shown by maxpts), and therefore have an execution time of .250. For 5 nodes, the maximum number of points by at least one block remains 82450 (58225+24225). For 6 nodes, the largest work load (maxpts) is 58225, and therefore the execution time is .177. As can be seen, there is no advantage of using more than six nodes for this test, as the memory requirements and execution time are unaffected. For an 8 even block case, the estimated execution time (exetime) decreases near linearly as expected.

```
 8-block m6 wing
  formatted grid from m6w8b.gr.fmt

  Grid Block Sizes
  block    imax    jmax    kmax      total
  -----    ----    ----    ----      -----
      1      97      25      17      41225
      2      97      25      17      41225
      3      97      25      17      41225
      4      97      25      17      41225
      5      97      25      17      41225
      6      97      25      17      41225
      7      97      25      17      41225
      8      97      25      17      41225

 nodes    maxpts    minpts    avgpts    %avgdev    megawords    exetime
 -----    ------    ------    ------    -------    ---------    -------
     1    329800    329800    329800       .000       18.469      1.000
     2    164900    164900    164900       .000        9.234       .500
     3    123675     82450    109933     16.667        6.926       .375
     4     82450     82450     82450       .000        4.617       .250
     5     82450     41225     65960     30.000        4.617       .250
     6     82450     41225     54966     33.333        4.617       .250
     7     82450     41225     47114     21.428        4.617       .250
     8     41225     41225     41225       .000        2.309       .125
```

The Mapper's estimates closely match actual performance as demonstrated with the 8 block case and 8 even block case. Note, slight deviations in performance for nodes 4 through 7 of the 8 even block case, are due system background noise (i.e., the job mix affecting the High Performance Switch on the IBM SP2).

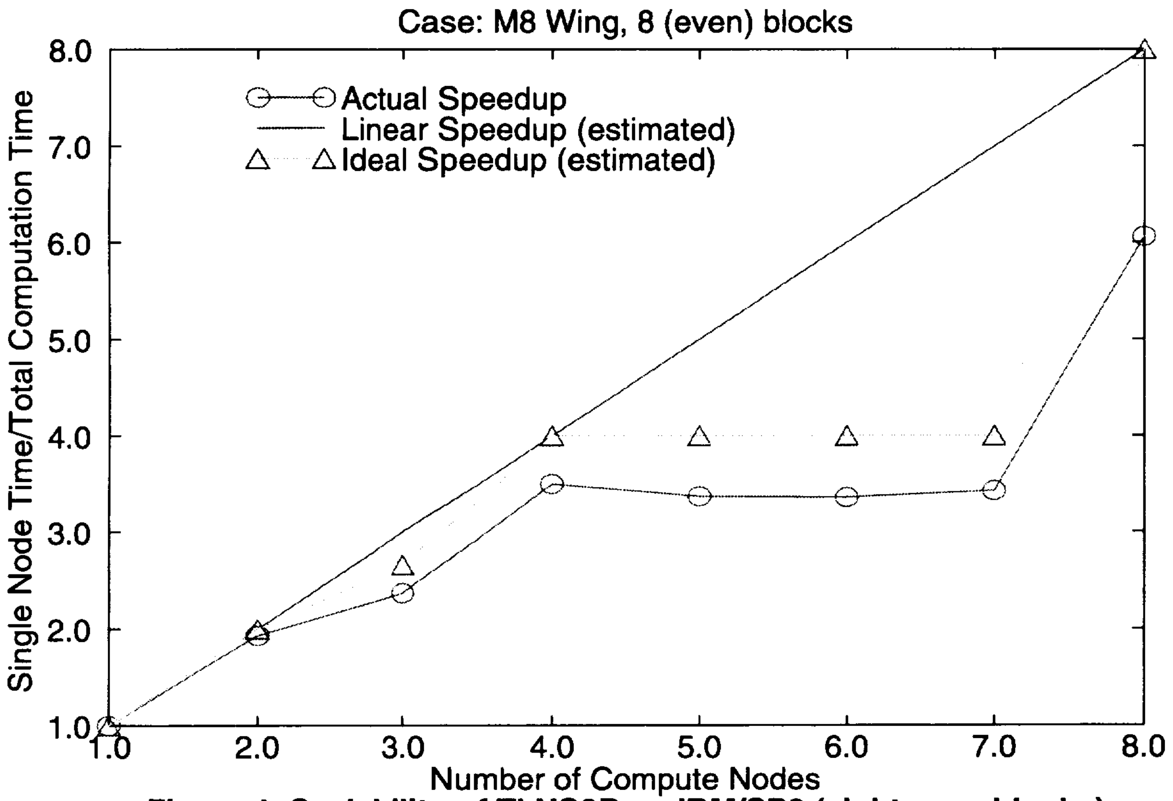

Figure 1. Scalability of TLNS3D on IBM/SP2 (eight even blocks)

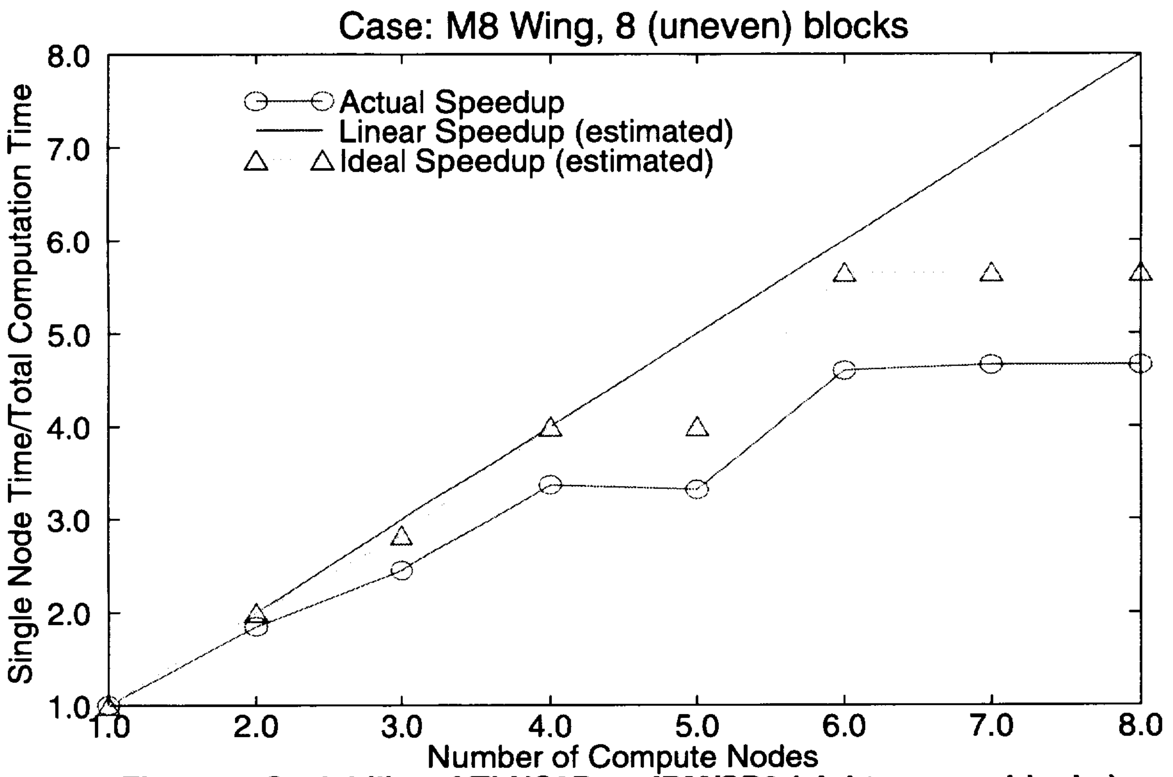

Figure 2. Scalability of TLNS3D on IBM/SP2 (eight uneven blocks)

As larger problems were tried, the performance degraded. A 32 block patched grid test case, this time with uneven sized blocks, was also used for performance tests. The Mapper's output, as shown completely in Appendix A, accounts for the estimated execution time based on the block sizes. The results of the execution of this 32 block case for 50/50 cycles, along with the Mapper's execution estimate follows.

	Sync	Ideal (Mapper)	Actual (sync/2525)
1	2525.00	1.000	1.000
2	1431.00	0.500	0.567
4	840.00	0.250	0.333
8	557.00	0.125	0.221
16	410.00	0.062	0.162
32	356.00	0.036	0.141

Synchronous Communication

An analysis of the program's scalability performance indicated the communication bottleneck for the exchange of boundary conditions. The communications concerning the boundary condition within TLNS3DMB are contained within two routines: bcflow and bcturb. The communication was performed with synchronous communication (i.e., send/recv pairs). The processor issuing the send blocks until the associated processor completes the receive. The following is an outline of communication sequence of the routine bcflow.

```
  /* ... declarations */
  ...
  /* begin outer loop on the blocks for interface blocks */
  do 1000 ibloc = 1,nbloc
  ns       =  nseg(ibloc)

    /* begin outer loop on the segments */
    do 100 iseg = 1,ns
       < ... code to compute cell attributes based on block and segment >

         /* get ghost cell variables from source block */
         if (src.eq.me) then /* (nodes(nblocs).eq.myrank) */
            < get ghost cell values at block interfaces and interior cuts >
            if (dst.ne.me) call MPI_Send /* (nodes(ibloc).ne.myrank) */
         endif

         /* update ghost cell variables on target block */
         if (dst.eq.me) then /* (nodes(ibloc).eq.myrank) */
            /*receive ghost cell variables from node if not already local */

             if (src.ne.me) call MPI_Recv /* (nodes(nblocs).ne.myrank) */
             < ... code to compute pressure at ghost cells >
         endif

    100 continue
 1000 continue
```

The following graph depicts the communication exchange for the eight (uneven block) case on eight nodes using synchronous SENDs/RECVs. The communication graph was created with the Visualization Tool (VT) [3]. The Parallel Programming Environment's Visualization Tool (VT) is designed to show graphically the performance and characteristics of an application program and also to act as an on-line monitor.

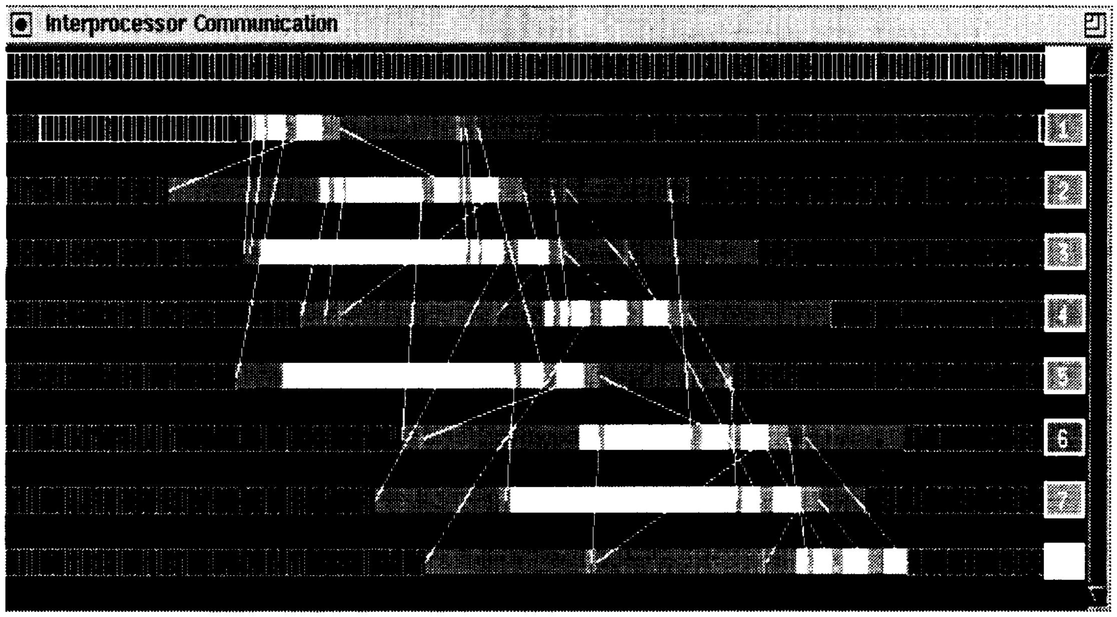

Figure 3. Synchronous communication exchange for eight uneven block

Although not obvious, the communication pattern establishes an implicit sequentialization. The sequentialization is caused by sends blocking until the matching receives are posted and the pattern of exchange for this program. See Chapter 4 of *Using MPI* [2] for a complete description of sequentialization. In TLNS3D, because the time to perform the calculations and the determination of whether to send/receive a message is small, the overhead should be minor for the entire process. However, as shown by the above communication pattern, the time does accumulate.

Asynchronous Communication

One method to overcome the delay associated with implicit sequentialization is asynchronous communication. Generally the send and receive operations in asynchronous communication place no constraints on each other in terms of completion, and thereby enable overlap of computation. In a typical implementation, the nonblocking receives (RECVs) are posted by the receiving nodes prior to the associated sends from the sending nodes. Then, the sends are issued, followed by a barrier (e.g., MPI_WAITALL) to ensure all send/recv pairs have completed. This method allows the intermediate calculations to be computed in a natural progression not waiting for the receive.

For bcflow, asynchronous communications could have been performed over the segment (inner loop) or over the block (outer loop). Optimizing over the block loop, provides nonblocking communications over a larger number of cells, and offers a greater potential. The outline of the asynchronous code over the entire block for bcflow is shown below.

Post receives
```
      do 999 ibloc = 1,nbloc
        do 99 iseg = 1,ns
            < ... code to compute cell attributes based on block and segment >
            if (dst.eq.me) then
               if (src.ne.me) then
                  call MPI_IRecv
                endif
            endif
         99 continue
      999 continue
```

Send data
```
      do 1000 ibloc = 1,nbloc
        do 100 iseg = 1,ns
           < ... code to compute cell attributes based on block and segment >
            if (src.eq.me) then
               if (dst.ne.me) then
                 < get ghost cell values at block interfaces and interior cuts>
                  call MPI_Send
                endif
            endif
         100 continue
      1000 continue
```

Wait All
```
      call mpi_waitall
```

Compute calculations
```
      do 1001 ibloc = 1,nbloc
        do 101 iseg = 1,ns
            < ... code to compute cell attributes based on block and segment >
            if (dst.eq.me) then
               if (src.ne.me) then
                  < ... code to compute pressure at ghost cells >
                endif
            endif
         101 continue
      1001 continue
```

Asynchronous Conditions

In order to use asynchronous communication, the order of the calculations being sent must be order independent (i.e., loop independent). In the case of bcflow, this means converting from a loop containing a SEND/RECV pair, to a separate loop containing calculations and a SEND. The underlying condition is that the values being sent for subsequent loop iterations are independent of the information received. This is necessary if multiple SENDs can be posted without intermediate RECVs.

```
Loop                    Loop
  C0                      irecv
  send                  end loop
  recv                  Loop
  C1                      C0
end loop                  send
C2                      end loop
                        wait all
                        Loop
                          C1
                        end loop
                        C2

Flow 1                  Flow 2
```

If the received information in Flow 1 (above) affects the calculation C0 of the next loop iteration, then there is a loop dependence, and asynchronous communication cannot be performed as shown in Flow 2.

Implementation Issues

Asynchronous communication requires the handling of several factors: the allocation and management of message buffers, the identification of messages (message tags), and the use of a barrier.

In synchronous communication, a blocking send waits until the associated receive completes. This approach ensures that the contents of the message buffer remains intact until the operation is complete. For asynchronous communication, the user must ensure that the operation is completed before the message buffer is reused. Since the nonblocking receives are posted first, a buffer for each message is needed.

The number and size of the asynchronous messages, and therefore buffers, is problem dependent. For TLNS3DMB, an associated program called the Sizer was written to create a parameter file defining the workspace needed for a given problem. See URL

http://hpccp-www.larc.nasa.gov:80/~dana/T6_load.html

for more detail. The Sizer program has been augmented to accommodate the asynchronous buffer sizes, and in turn represent them as constants (parameters) in the generated parameter file.

Because TLNS3DMB is designed to handle multiple blocks per nodes, it is often the case that a process will send multiple messages to another processor over the block in which messages are being changed. In this event, it is necessary for the receiving process to distinguish between messages. The concept of a "message tag" (message identification) is often used. A tag is an arbitrary nonnegative integer to used to restrict receipt of the message (sometimes also called "message type").

Additionally, the use of a barrier is required to ensure all processes have completed exchanging messages. This requires knowing the number of messages being sent by each individual processor. The following is an outline of the asynchronous approach including the message tags and barrier call.

Post receives
```
      isegtag=0 /* counter for every cell to make unique message tag */
      isegnum=0 /* count number of message received */
      do 999 ibloc = 1,nbloc
        do 99 iseg = 1,ns
```

```
          < code for cell attributes (i.e., imap array) indexed by iseg,ibloc
            example: nblocs = imap(7,iseg,ibloc) >
          isegtag=isegtag+1 /* increment tag tp from unique message id */
           < ... code to compute cell attributes based on block and segment >
           if (dst.eq.me) then
              if (src.ne.me) then
                isegnum=isegnum+1 /* increment offset into temporary buffer */
                 call MPI_IRecv(recv_buffer(isegnum*maxbufsize),
                   ...,TAG_FLOW+isegtag,...,ireq(isegnum),ierr)
               endif
           endif
        99 continue
     999 continue
```

Send data

```
     isegtag=0 /* counter for every cell to make unique message tag */
     do 1000 ibloc = 1,nbloc
       do 100 iseg = 1,ns
          isegtag=isegtag+1 /* increment tag to form unique message id */
           < ... code for cell attributes; i.e., imap(n,iseg,ibloc) >
           < ... code to compute cell attributes based on block and segment >
           if (src.eq.me) then
              if (dst.ne.me) then
                < get ghost cell values at block interfaces and interior cuts>
                 call MPI_Send(buffer,...,TAG_FLOW+isegtag,...)
               endif
           endif
        100 continue
     1000 continue
```

Wait All

```
     /* barrier to wait until all messages to be received are processed */
     call mpi_waitall(isegnum,ireq,status_array,ierr)
```

Compute calculations

```
     /* after mpi_waitall, all message have been exchanged */
     isegnum=0 /* count messages received for offset into received buffer */
     do 1001 ibloc = 1,nbloc
       do 101 iseg = 1,ns
           < ... code for cell attributes; i.e., imap(n,iseg,ibloc) >
           < ... code to compute cell attributes based on block and segment >
           if (dst.eq.me) then
              if (src.ne.me) then
                 isegnum=isegnum+1 /* increment offset */
                 do i=1,msgsize
                  /* copy values from temp. buffer to array based on offset */
                    wk2d(i)=buffer(isegnum*maxbufsize+idana)
                 enddo
                 < ... code to compute pressure at ghost cells >
               endif
           endif
        101 continue
     1001 continue
```

The following is the communication exchange for the eight (uneven block) case on eight nodes using asynchronous SENDs/RECVs. The pattern shows that the SENDs are sent when encountered thereby preventing any sequentialization. Each node blocks with a barrier, until all messages are received.

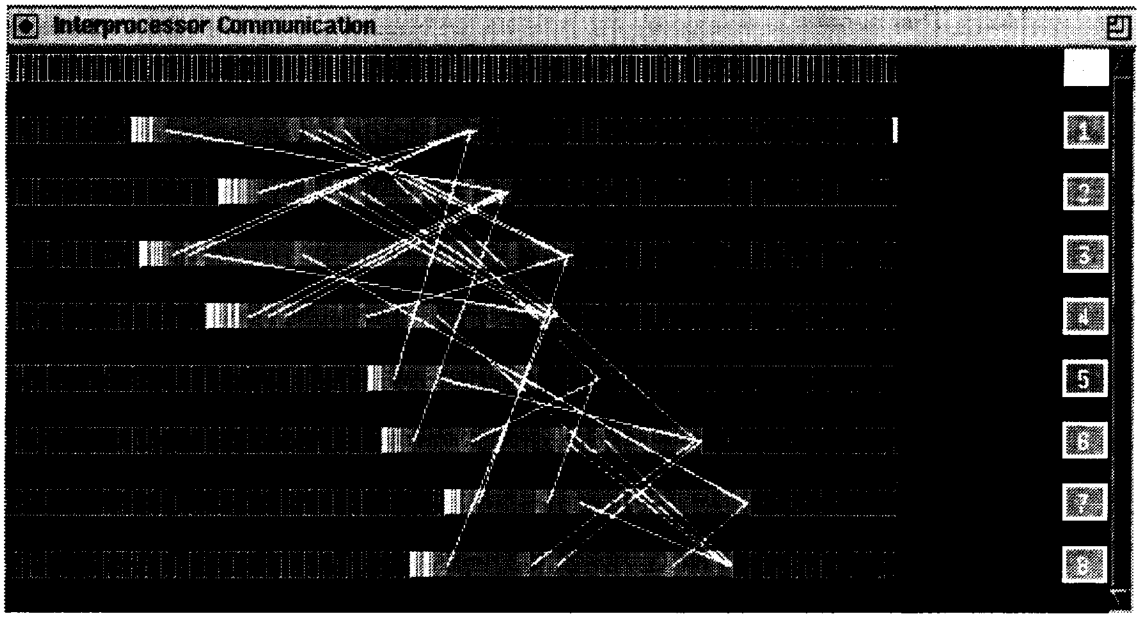

Figure 4. Asynchronous communication exchange for eight uneven block

For larger size problems, asynchronous communication clearly outperforms synchronous communication.

32-block Asynchronous Results

	Sync	Async	Ideal
1	2525.00	-	1.00
2	1431.00	1383.00	0.500
4	840.00	741.00	0.250
8	557.00	412.00	0.125
16	410.00	246.00	0.062
32	356.00	179.00	0.036

Summary

In summary, the boundary exchange for TLNS3DMB was presented using both synchronous and asynchronous communications. An explanation of the necessary conditions (data independence) and the method (buffers and tags) to convert from isochronous to asynchronous was described. As expected with larger problems and larger number of nodes, the asynchronous communication performed better.

Bibliography

[1] *Parallelization of a Multiblock Flow Code: An Engineering Implementation.* Veer N. Vatsa and Bruce W. Wedan

[2] *Using MPI. Portable Parallel Programming with the Message-Passing Interface.* William Gropp, Ewing Lusk, Anthony Skjellum. The MIT Press, 1994.

[3] *IBM AIX Parallel Environment Operation and Use Release 2.1 (SH26-7230-01).* Second Edition, December 1994. Chapter 5 Visualizing Program and System Performance.

Appendix A. Complete Mapper Output for a 32-block Case

The following is the complete Mapper output for a 32 (uneven) block case. The first table represents the grid block sizes. Based on the block sizes, the Mapper distributes the blocks achieving the best possible workload balance. The column maxpts represents the largest number of points held by any single block for the given workload. That number is divided by the "maxpts for 1-node" to determine the projected execution time (i.e., the exetime column). The "exetime" value in the second table estimates the ideal execution time based on the workload distributed to the associated number of processors.

It should be noted, that adding an additional node does not always re-distribute the workload to reduce the maxpts. Therefore, adding additional nodes does not always decrease the projected execution time.

```
0:32-block m6 wing
  0:formatted grid from m6w32b.gr.fmt
  0:
  0:Grid Block Sizes
  0:block   imax   jmax   kmax     total
  0:-----   ----   ----   ----     -----
  0:    1     49     13     17     10829
  0:    2     49     13     17     10829
  0:    3     49     13     17     10829
  0:    4     49     13     17     10829
  0:    5     49     15     17     12495
  0:    6     49     15     17     12495
  0:    7     49     15     17     12495
  0:    8     49     15     17     12495
  0:    9     49     11     17      9163
  0:   10     49     11     17      9163
  0:   11     49     11     17      9163
  0:   12     49     11     17      9163
  0:   13     49     13     17     10829
  0:   14     49     13     17     10829
  0:   15     49     13     17     10829
  0:   16     49     13     17     10829
  0:   17     49     13     17     10829
  0:   18     49     13     17     10829
  0:   19     49     13     17     10829
  0:   20     49     13     17     10829
  0:   21     49     11     17      9163
  0:   22     49     11     17      9163
  0:   23     49     11     17      9163
  0:   24     49     11     17      9163
  0:   25     49     15     17     12495
  0:   26     49     15     17     12495
  0:   27     49     15     17     12495
  0:   28     49     15     17     12495
  0:   29     49     13     17     10829
  0:   30     49     13     17     10829
  0:   31     49     13     17     10829
  0:   32     49     13     17     10829
  0:
  0:nodes   maxpts   minpts   avgpts   %avgdev   megawords   exetime
  0:-----   ------   ------   ------   -------   ---------   -------
  0:    1   346528   346528   346528      .000      19.406     1.000
  0:    2   173264   173264   173264      .000       9.703      .500
```

```
 0:    3   119119   109956   115509     3.205      6.671      .344
 0:    4    86632    86632    86632      .000      4.851      .250
 0:    5    74137    64974    69305     5.577      4.152      .214
 0:    6    63308    54145    57754     6.410      3.545      .183
 0:    7    54145    44982    49504     7.830      3.032      .156
 0:    8    43316    43316    43316      .000      2.426      .125
 0:    9    43316    34153    38503    10.043      2.426      .125
 0:   10    41650    32487    34652     8.076      2.332      .120
 0:   11    32487    23324    31502     5.115      1.819      .094
 0:   12    32487    23324    28877    12.821      1.819      .094
 0:   13    30821    21658    26656    14.423      1.726      .089
 0:   14    30821    21658    24752    14.011      1.726      .089
 0:   15    30821    21658    23101     9.165      1.726      .089
 0:   16    21658    21658    21658      .000      1.213      .062
 0:   17    21658    12495    20384     9.559      1.213      .062
 0:   18    21658    12495    19251    15.600      1.213      .062
 0:   19    21658    12495    18238    19.889      1.213      .062
 0:   20    21658    12495    17326    22.308      1.213      .062
 0:   21    21658    10829    16501    25.046      1.213      .062
 0:   22    21658    10829    15751    26.398      1.213      .062
 0:   23    21658    10829    15066    26.546      1.213      .062
 0:   24    19992    10829    14438    25.639      1.120      .058
 0:   25    19992    10829    13861    23.807      1.120      .058
 0:   26    19992    10829    13328    21.154      1.120      .058
 0:   27    19992    10829    12834    17.769      1.120      .058
 0:   28    18326    10829    12376    14.286      1.026      .053
 0:   29    18326     9163    11949    13.561      1.026      .053
 0:   30    18326     9163    11550    12.177      1.026      .053
 0:   31    18326     9163    11178    10.204      1.026      .053
 0:   32    12495     9163    10829     7.692       .700      .036
>
```

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 07704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** June 1997	**3. REPORT TYPE AND DATES COVERED** Contractor Report

4. TITLE AND SUBTITLE Asynchronous Communication of TLNS3DMB Boundary Exchange	**5. FUNDING NUMBERS** C NAS1-20431
6. AUTHOR(S) Dana P. Hammond	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Computer Sciences Corporation, Systems Sciences Division, Hampton, VA 23666-1379	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) National Aeronautics and Space Administration, Langley Research Center, Hampton, VA 23681-0001	**10. SPONSORING/MONITORING AGENCY REPORT NUMBER** NASA CR-201722

11. SUPPLEMENTARY NOTES
Langley Technical Monitor: Geoffrey M. Tennille

12a. DISTRIBUTION/AVAILABILITY STATEMENT Unclassified–Unlimited, Subject Category 61, Availability: NASA CASI (301) 621-0390	**12b. DISTRIBUTION CODE**

13. ABSTRACT ***(Maximum 200 words)***

This paper describes the recognition of implicit serialization due to coarse-grain, synchronous communication and demonstrates the conversion to asynchronous communication for the exchange of boundary condition information in the Thin-Layer Navier Stokes 3-Dimensional Multi Block (TLNS3DMB) code. The implementation details of using asynchronous communication is provided including buffer allocation, message identification, and barrier control. The IBM SP2 was used for the tests presented

14. SUBJECT TERMS Computer Program, CFD, Parallelization	**15. NUMBER OF PAGES** 12
	16. PRICE CODE A03

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT**