WP 5 Project Deliverable D5.3

Parallelisation efficiency guide

Project Number	IST-2000-29266
Project Title	Virtual Real Time Fire Emergency Simulator
Deliverable Type	Deliverable
Deliverable Class	Internal

Deliverable Number	D5.3
Title of Deliverable	Parallelisation efficiency guide
Nature of the Deliverable	Report
Contributing WPs	WP 5
Contractual Date of Delivery	October 31st, 2003
Actual Date of Delivery	November 17th, 2003
URL	www.virtualfires.org
Authors	Christian Redl (CD), Wilhelm Brandstätter (CD)
Contact Details	Institute for Structural Analysis / SiTu Research Univ. Prof. Dipl.-Ing. Dr. techn. Gernot Beer Lessingstrasse 25/II 8010 Graz / Austria Tel.: +43 316 8736180 Fax: +43 316 8736185 Email: gernot.beer@ifb.tu-graz.ac.at

Abstract	This report describes the parallelisation strategy for the Lattice Boltzmann solver ICE used for Virtual Fires. Several tests were done to ensure the coincidence of the parallel and serial results. Performance data were obtained on a DEC Alpha ES 40 and on a Linux cluster. These data confirm the potential of the Lattice Boltzmann method for solving flow problems in real time.
Keywords	Parallelisation, MPI, LAM, Lattice Boltzmann, Real time

CONTENTS

1. INTRODUCTION

Computer programs are usually sequential meaning that their execution is done by only one processor. The computational time varies extremely from a few seconds up to several weeks. Especially in the second case speeding up the computation is of particular interest. One can rely on the steady increasing performance of processor technology or try to parallelise computer codes. The idea of parallelisation is to spread the work load to several processors and therefore speed up computation.

The ultimate objective of the Virtual Fires system is to perform real time simulations of tunnel fires in a concurrent VR environment. Since in general CFD calculations are very CPU demanding, it is not possible to perform this on today's single processor systems. Therefore, the parallelisation of the Lattice Boltzmann code ICE is a must to achieve the aforementioned objective. In what follows, only the work done with respect to parallelisation is presented. A detailed description of the flow solver ICE and the underlying theory can be found in the User Guide Version 1.0 [1].

There mainly exist two classes of parallel programming paradigms:

The shared memory paradigm consists of sharing data through a common memory by using compilation directives. This paradigm allows using parallel machines without major changes to the sequential code, but becomes inefficient for larger numbers of involved processors due to memory bandwidth limitations or in inhomogenous environments.

The distributed memory paradigm consists in distributing the data to the processors to share the work load. Processors requiring information located in another processor have to communicate through messages. As the messages are sent over a network connecting the processors the amount of communication should be minimal. The distributed memory model requires much more programming effort but leads to more efficient codes and can be even used for cluster solutions.

More in depth information about parallel programming can be found e. g. in ref. [2] and [3].

2. SELECTED PROGRAMMING MODEL

In view of the available hardware and the requirements to the program it was decided to use a distributed memory parallelisation.

Distributed memory systems are characterised by a high scalability and the large physical memory available normally. The performance is influenced by the balance between CPU speed and network speed and depends heavily on the programmer.

In the MPMD (multiple program – multiple data) programming model each processor executes its own program. The communication between the processors is performed by sending and receiving messages. A subroutine library which contains functions for sending and receiving messages. The data distribution and communication must be explicitly defined by the programmer.

Although it is probably the most difficult approach to parallel programming it is selected due to the following reasons:

- It promises the highest performance on distributed memory systems with a large number of processors.
- It is the most portable approach.

- The programmer has all freedom to optimise communication.

There are a few message passing libraries available, but for Virtual Fires the Message Passing Interface (MPI), the de facto standard, is used for portability. A full description of the MPI standard is given in [4] and [5]. Up-to-date information can be found on the website of the MPI forum [6].

3. DOMAIN DECOMPOSITION

There exist a number of possibilities for decomposing a computational domain. The most popular are geometry based methods (e.g. Cartesian coordinate bisection, coordinate bisection, recursive coordinate bisection) and graph based methods (e. g. recursive graph bisection, Greedy algorithm, recursive spectral bisection, Kernighan-Lin method).

For the strictly regular computational meshes used within the Virtual Fires project two simple domain decomposition methods promise the best performance due to a minimum amount of communication between the processors.

The simplest one is a one-dimensional decomposition (ODD) which works along the direction of longest extend of the computational domain. This method works for any number of processors, but will lead to bad surface/volume ratios for higher number of processors as can be inferred from fig. 1. The term surface/volume ratio characterises the number of computational mesh cells located on an inter-processor communication boundary to the number of computational mesh cells allocated to an individual processor.

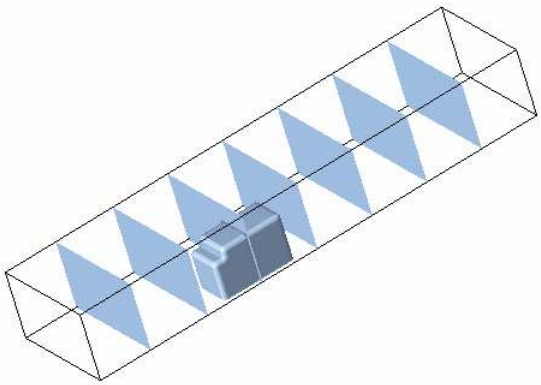

Figure 1: One dimensional domain decomposition

Alternatively for the recursive co-ordinate bisection method (RCB) the number of sub domains must be a power of 2, but in most cases it delivers better topologies of the sub domains (fig. 2). The Send/Receive lists for the inter-processor communication are almost of equal size in contrast to the ODD method. The major disadvantage is that each processor has to communicate with a larger number of neighbours as in the ODD method (2 processor neighbours as a maximum).

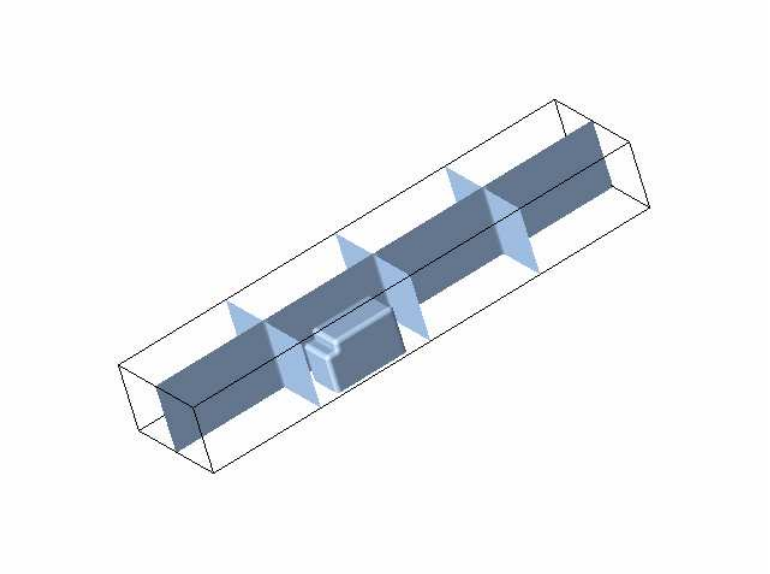

Figure 2: Recursive co-ordinate bisection method

Both decomposition methods have been implemented. It was observed at the performance tests below, that for the tested configurations the ODD method always shows higher performance. This is primary because of the larger number of processors involved in the communication process. As the ODD also offers more flexibility (arbitrary number of processors possible) it is the default domain decomposition used in MP**ICE** from now on.

4. PARALLELISATION STRATEGY

The parallelisation is done in a way that only the particle distribution functions are communicated between the involved processors. The hydrodynamic variables (velocity, pressure, temperature and smoke concentration) are collected only on demand by the root processor and subsequently written to the file system (which is the bottleneck of the hardware-system). Fig. 3 shows a scheme of the algorithm (the 1D/3D coupling is not yet implemented in the parallel version).

Each computational cell contains the information from the particle distribution functions and the hydrodynamic variables. At the beginning of the simulation a global ID (valid throughout the entire computational domain) is assigned to each cell. After the domain decomposition a local ID is assigned to the cells within each processors domain. The indexing scheme of the particle distribution functions is completely locally.

Afterwards the indices of the distribution functions, which are to be shifted to a neighbouring processor and the corresponding target ID are determined. Fig. 4 highlights the particle distributions, which are involved in the communication process (Explanation: Red coloured distributions are sent to processor red, blue ones are sent to processor blue, …). Both IDs are stored in SEND and RECEIVE lists. These lists are first sorted by ascending processor ID and second sorted by ascending distribution ID within the processor.

Within the calculation procedure the actual particle distributions are copied to a one dimensional array and this array is sent to the corresponding neighbouring processor. The neighbouring processor receives the message and the content of the array is assigned to the corresponding particle distribution function.

As the Lattice Boltzmann-Method works completely local except the translation step, where the particle distribution functions are shifted to the neighbour cells, only the communication described above is necessary for the calculation. No information about hydrodynamic variables has to be exchanged.

The hydrodynamic variables are collected by the root processor only, if a result data file is required by the data manager.

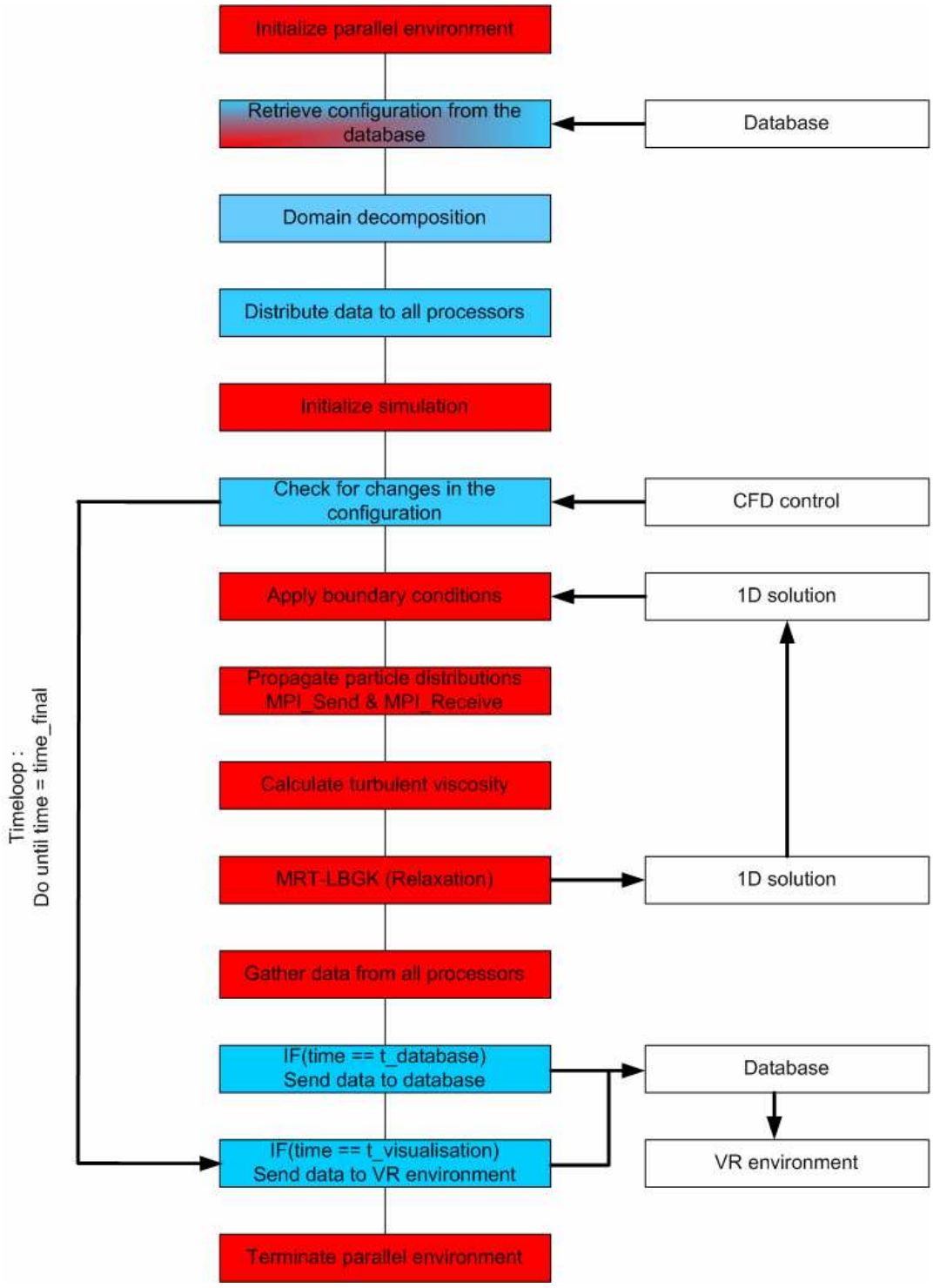

Figure 3: Parallel algorithm

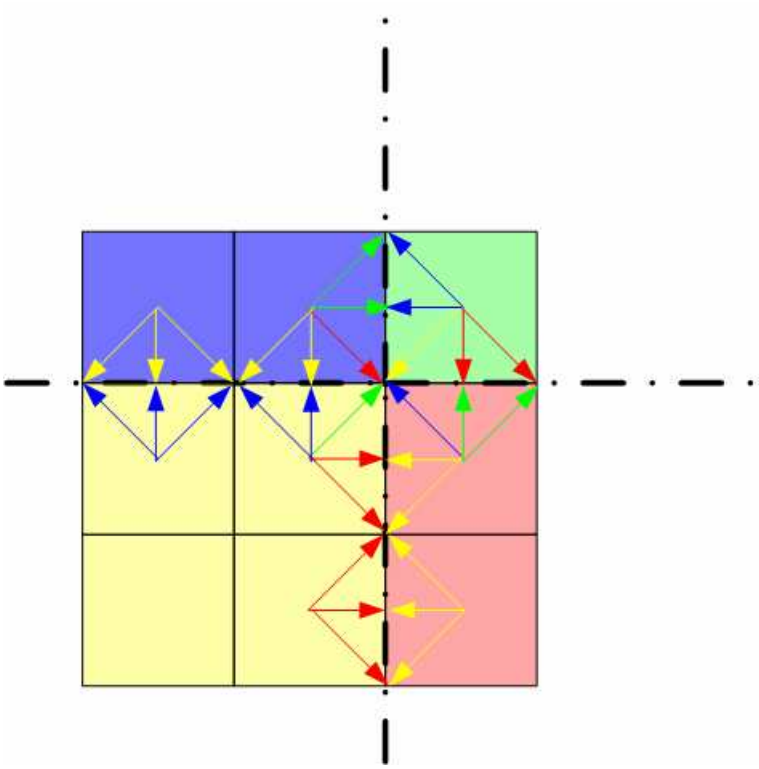

Figure 4: Particle distribution functions involved in communication

A pseudo code of the program is given in Appendix A.

5. AVAILABLE HARDWARE

Hardware at PDC/KTH

The initial plan was the use the Power Parallel SP “Strindberg”, a distributed memory system from IBM consisting of 170 processor node with a theoretical peak performance (TPP) of 198 GFLOPs. In the mean time a number of nodes have been decommissioned.

Eventually the HP Itanium 2 cluster “Lucidor” will replace “Strindberg”. “Lucidor” consists of 90 nodes each equipped with two 900 MHz Itanium 2 “Mc Kinley” processors resulting in a TPP of 648 GFLOPs. The processors are interconnected via Myrinet.

Hardware at CD

The Compaq DEC Alpha ES 40 is equipped with 4 833 MHz processors.

The Linux based cluster “Waltons” is available at CD consisting of 4 Intel Pentium IV 2.4 GHz processors and 2 Intel Pentium IV 2.8 GHz processors connected via GBit Ethernet

Additionally performance tests were carried out on the available workstations (6 Intel Xeon 2.8 GHz processors) using Mbit Ethernet.

6. USED SOFTWARE

On the Compaq Dec Alpha ES 40 Compaq MPI 1.96 is available. It supports MPICH 1.1.1 and was released in May 2001. In view of the consolidation of Compaq with Hewlett-Packard a further development of Compaq MPI is not to be expected. The native Fortran90 compiler f90 was used.

For running parallel processes at the cluster at CD the LAM/MPI 7.0.2 [7] environment was used. LAM/MPI provides a complete implementation of the MPI 1.2 standard and supports large portions of the MPI 2 standard. An outstanding feature of LAM/MPI is the very easy configuration procedure even on in-homogenous environments. The Intel FORTRAN Compiler 7.1 for Linux was used for compilation.

7. VALIDATION

The most important requirement in parallel computing (apart from performance) is that the parallel version of the code produces exactly the same results compared to the single processor version. For explicit methods, which require just the variables from the previous time level, this can be accomplished much easier than for implicit methods. Nevertheless the results of the parallel simulation have to be validated.

A few different tests were done to verify the coincidence of the serial and parallel results:

1. The three dimensional domain was initialised with a zero concentration field everywhere except the center where the concentration was set equal to one. The velocity field was set equal zero. The diffusion of the concentration was observed for 100 time steps. The result is shown in fig. 5.

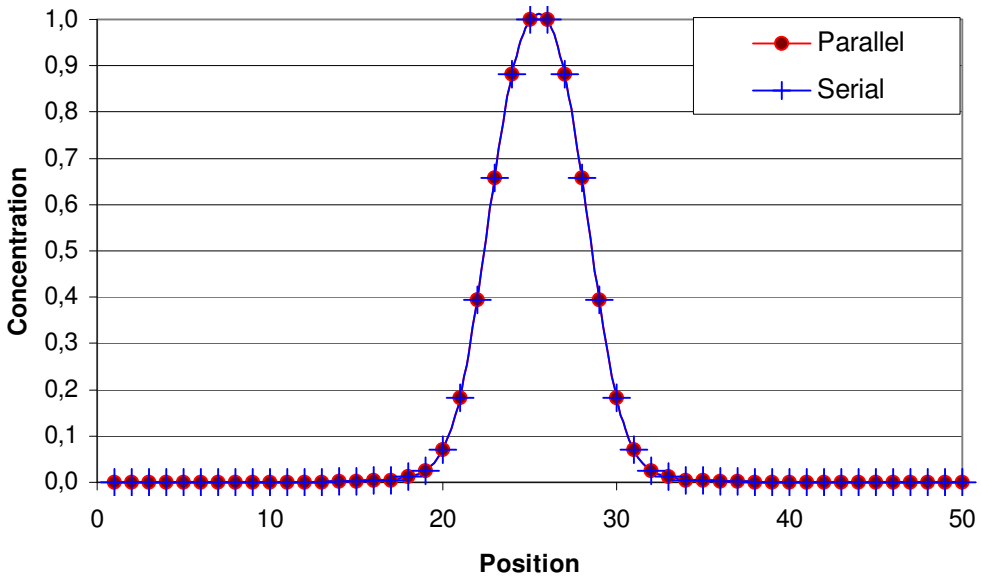

Figure 5: Diffusion - Comparison between serial and parallel results

2. A concentration cone was transported by the fluid through the domain. The figure below shows instantaneous velocities and the 0.5 isosurface of the concentration for an arbitrary point in time.

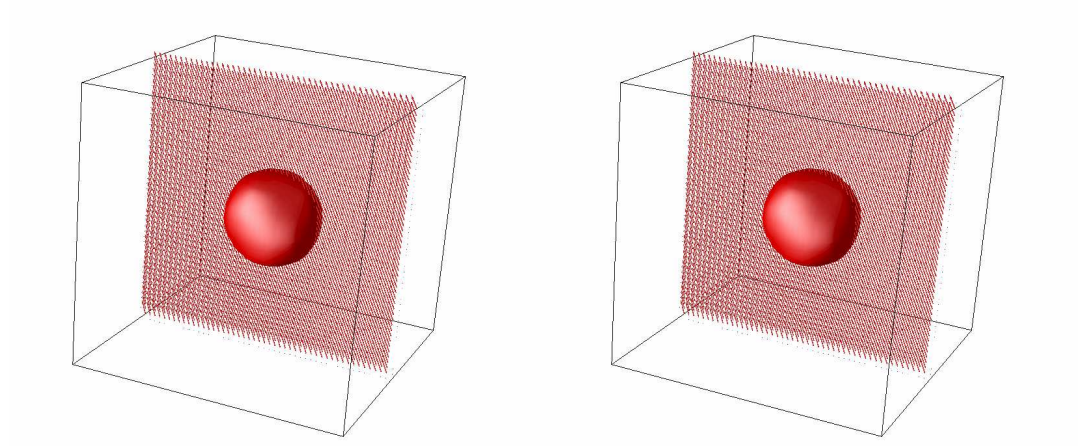

Figure 6: Convection-Diffusion – Left: Parallel, right: serial

3. Smoke development resulting from a train fire in a part of the subway station provided by the Fire Department Dortmund was simulated. The results of the serial and parallel version are depicted below. Figs. 7a – 7d show the temporal evolution of the 400 [K] temperature isosurface (red) and three different isosurfaces of the smoke concentration (grey). The left image corresponds to the parallel simulation, the right one to the serial calculation.

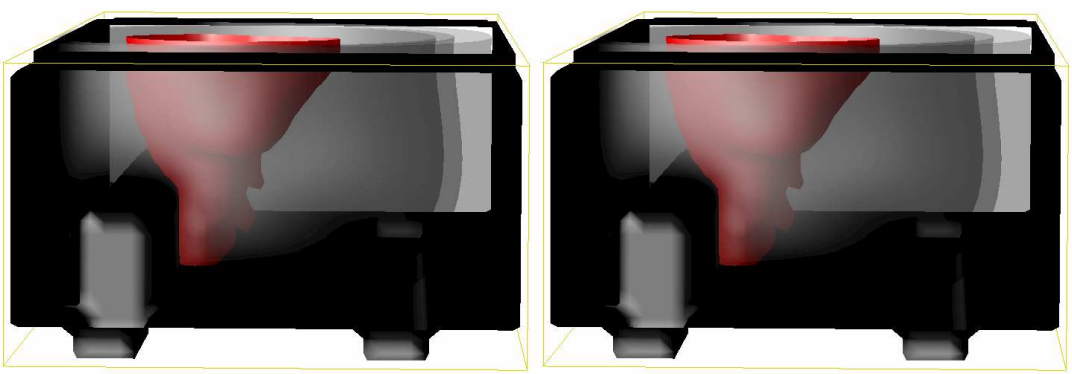

Figure 7a: Subway station at time t = 1000

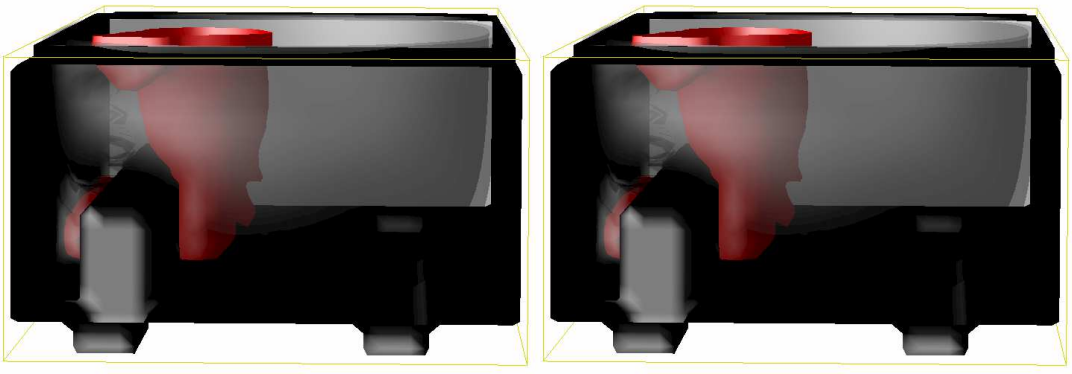

Figure 7b: Subway station at time t = 3000 (left: parallel, right: serial)

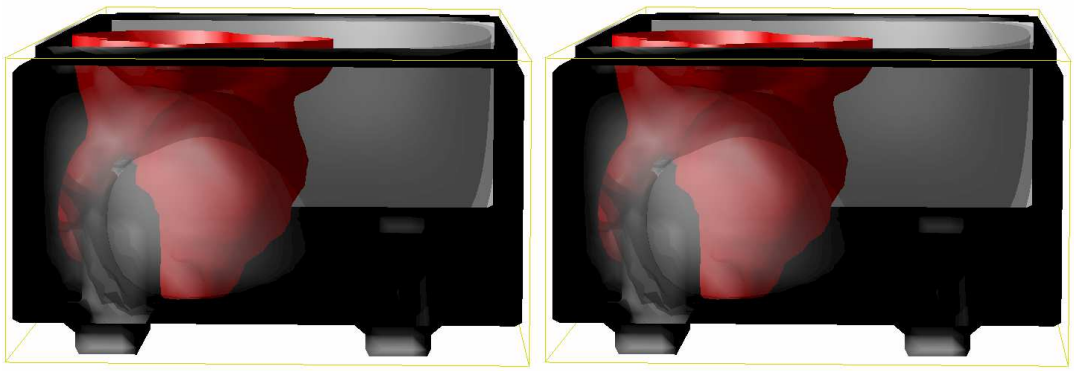

Figure 7c. Subway station at time t= 5000 (left: parallel, right: serial)

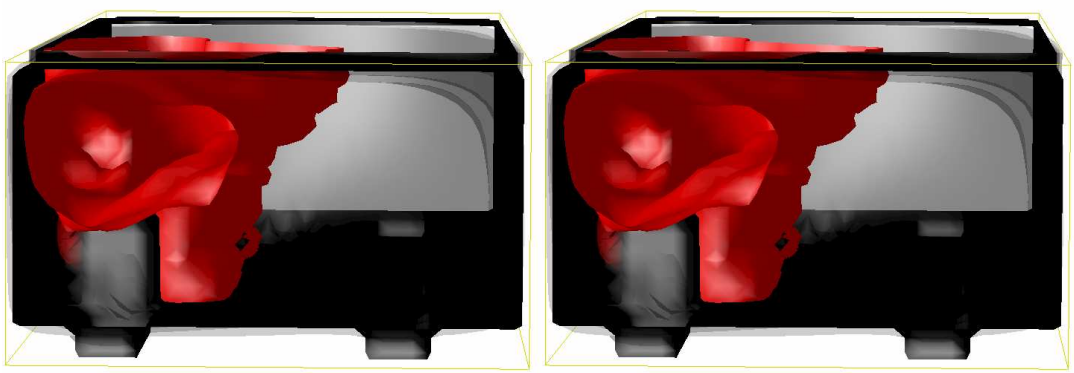

Figure 7d. Subway station at time t = 8000 (left: parallel, right: serial)

Even if the results seem to be equivalent on a qualitative basis, it is necessary to validate it also quantitatively. Therefore the velocity, pressure and concentration fields for different time steps were compared between the serial and the parallel calculations. The differences in the fields were in the order of machine accuracy.

8. PERFORMANCE TESTS

Performance tests on the following four computational environments were carried out:

- Compaq DEC Alpha ES 40 equipped with 4 processors (cross bar switch)
- Linux cluster "Waltons" with 6 processors interconnected with GigaBit Ethernet
- A Linux cluster using 6 processors of "Waltons" and 2 to 6 processors of a "pseudo cluster" called "Hobeln" consisting of different workstations connected by 100 MBit Ethernet

"Lucidor" is still in an open test mode and therefore the performance tests were done only on CDs hardware. The performance on "Lucidor" is expected to be similar to the results obtained on the Linux cluster "Waltons".

It is important to note that only the performance of the calculation part of MP**ICE** was measured. The I/O was reduced to a minimum.

Three typical configurations have been considered:

Testcase	#Cells (X)	#Cells (Y)	#Cells (Z)	#Cells	Calculated quantities
A	100	30	20	60.000	Flow & Temperature
B	100	50	50	250.000	Flow & Temperature
C	200	100	100	2.000.000	Flow & Temperature

Only the flow solution together with the temperature field was calculated. No additional scalar field (smoke concentration, …) was considered. Configuration A is the typical domain size for the Virtual Fires test cases.

In what follows the serial version of ICE was always taken as the reference solution (number of processors equal 1). The reference solution of "Waltons" was obtained on one of the fast processors with 2.8 GHz clockrate. For all test cases five simulations over 100 time steps were performed and the results were averaged.

Compaq DEC Alpha ES 40

The compiler settings used for the DEC compiler f90 were: `-arch ev6 -tune ev6 -O4`

The parallel performance data for the DEC Alpha ES 40 are listed in tables 1 & 2. Fig. 8 shows the speed up for the three test cases.

	Number of processors		
Testcase	1	2	4
A	5,20 [s]	1,60 [s]	0,80 [s]
B	33,40 [s]	17,96 [s]	7,61 [s]
C	348,00 [s]	185,00 [s]	94,57 [s]

Table 1: DEC Alpha ES 40 - Computation time

	Number of processors		
Testcase	1	2	4
A	1,00	3,25	6,50
B	1,00	1,86	4,38
C	1,00	1,88	3,68

Table 2: DEC Alpha ES 40 - Speed up

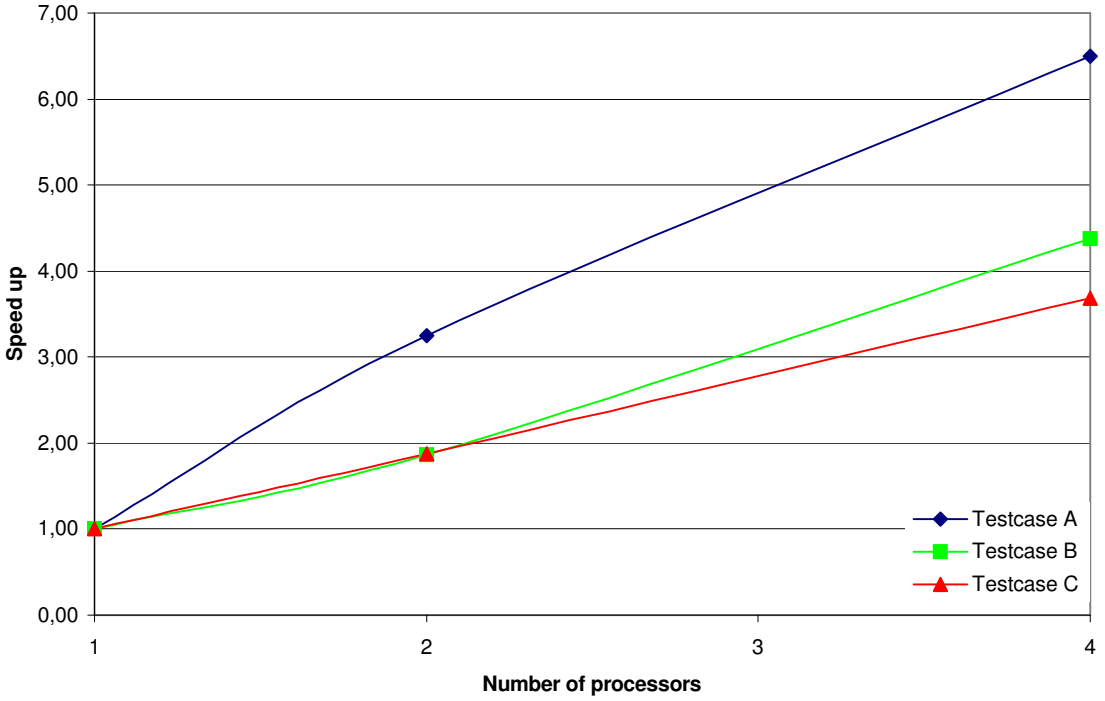

Figure 8: Performance on the DEC Alpha ES 40

It was observed that the fourth processor always delivers only about 50 to 70 percent of its performance; the remaining performance is consumed by system processes. This may explain the relatively low speed up of 3.68 for test case C using 4 processors.

The super linear speed up for test case A is probably due to very strong caching effects. One must keep in mind that the sub domains for each processor only contain 15 000 cells. To verify the result a comparison with a serial simulation containing 15 000 cells was done. This results in a computational time of 0.76 [s] for the serial version compared to 0.80 [s] for the parallel version running 15.000 cells on each processor.

Waltons & Hobeln

The standard compiler setting regarding optimisation were used for the Intel compiler ifc: `-O2`

The parallel performance data are listed in tables 3 and 4. The results for 1 to 6 processors are solely obtained on the Linux cluster "Waltons". For the results marked in blue (8 to 12 processors) additional workstations (called "Hobeln") were used. The results are depicted in fig 9.

	Number of processors						
Testcase	1	2	4	6	8	10	12
A	3.94 [s]	1.93 [s]	1.09 [s]	0.66 [s]	0.57 [s]	0.45 [s]	0.54 [s]
B	17.94 [s]	8.75 [s]	4.45 [s]	3.09 [s]	2.78 [s]	2.20 [s]	2.04 [s]
C	138.67 [s]	74.38 [s]	41.15 [s]	28.26 [s]	23.76 [s]	17.20 [s]	12.72 [s]

Table 3: Linux cluster - Computation time

	Number of processors						
Testcase	1	2	4	6	8	10	12
A	1.00	2.04	3.61	5.97	6.91	8.75	7.29
B	1.00	2.05	4.03	5.81	6.45	8.15	8.79
C	1.00	1.86	3.36	4.91	5.84	8.06	10.90

Table 4: Linux cluster - Speed up

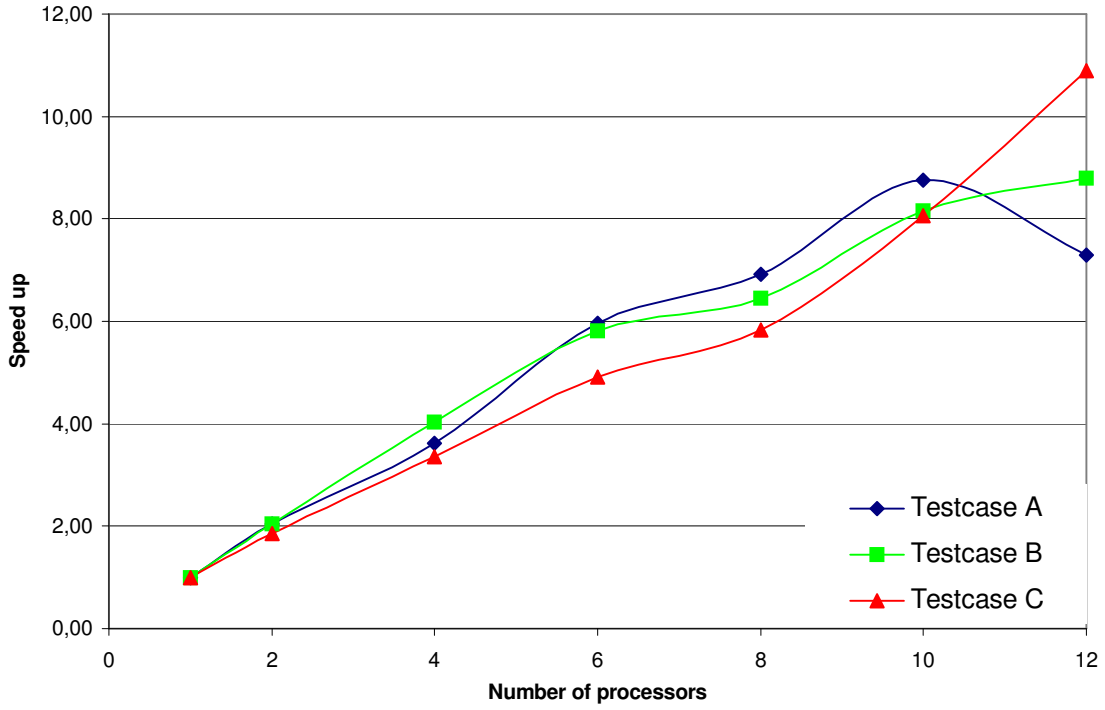

Figure 9: Performance on the Linux cluster

Judging the results above it is important to note that the reference solution is obtained on a 2.8 GHz processor and in the results for 4 and 6 processors there are also 2.4 GHz processors involved. It is also very interesting that despite a relatively slow connection (100 MBit Ethernet) to the workstations a speed up of about 8 can be obtained for all cases using 10 processors. For test case C almost a speed up of 11 is obtained on 12 processors. For test case A the speed up breaks down using all processors. This is not astonishing as each sub domain only contains 5000 cells and therefore the communication is of greater weight than the calculation. This is in accordance to Amdahl's law if the inter-processor communication is taken as limiting factor.

Comparison

Figs. 10 - 12 show a comparison of the calculation times between the DEC Alpha ES 40 and the Linux cluster. As expected the single processor performance is much higher on the 2.8 GHz Intel processor than on the 833 MHz Alpha processors, which uses a 3 year old technology.

The speed up obtained for test case A with 60.000 cells is better on the DEC Alpha ES 40, which is not astonishing and probably due to caching effects and the very fast cross bar switch connecting the processors.

For the two other test cases the Linux cluster is outperforming the DEC Alpha ES 40, but the differences in speed up become smaller for 4 processors which can again be attributed to the fast cross bar switch.

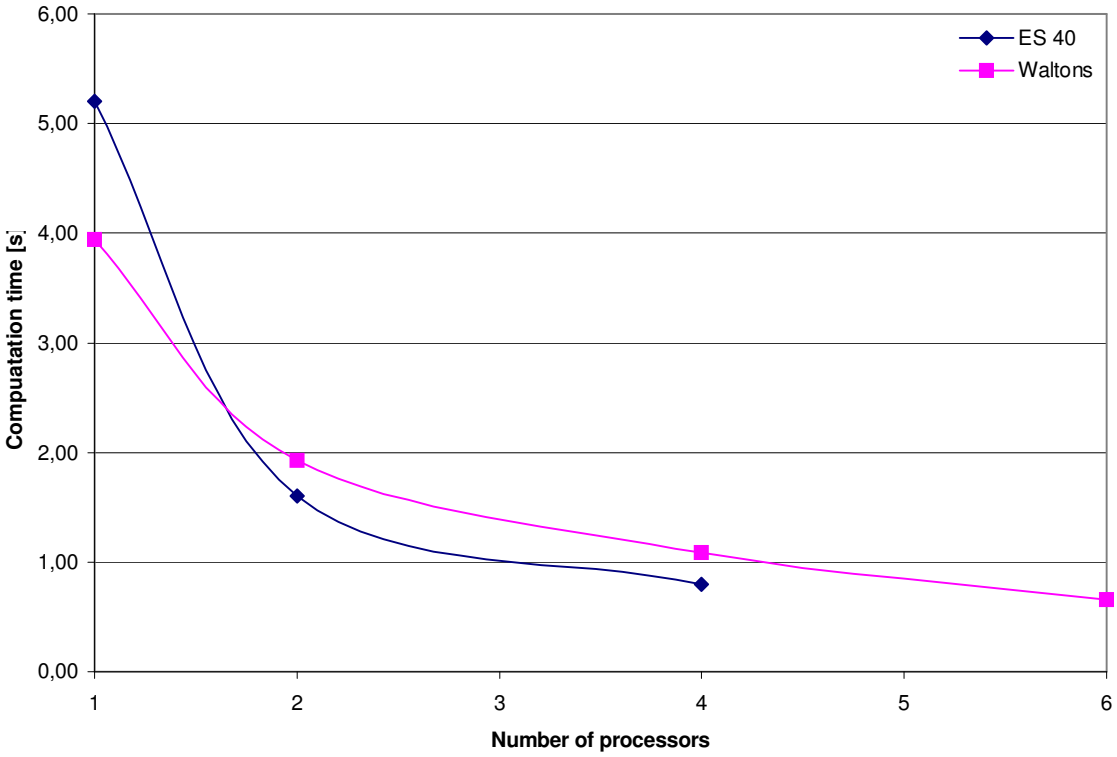

Figure 10: Test case A - Comparison DEC Alpha / Linux cluster

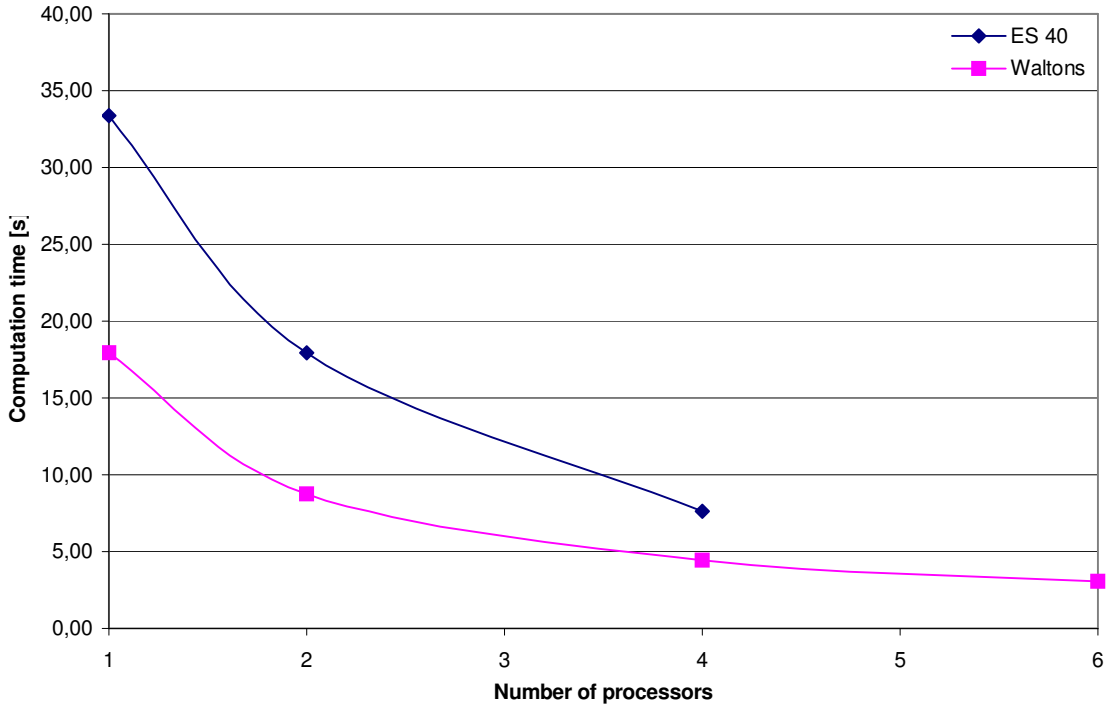

Figure 11: Test case B - Comparison DEC Alpha / Linux cluster

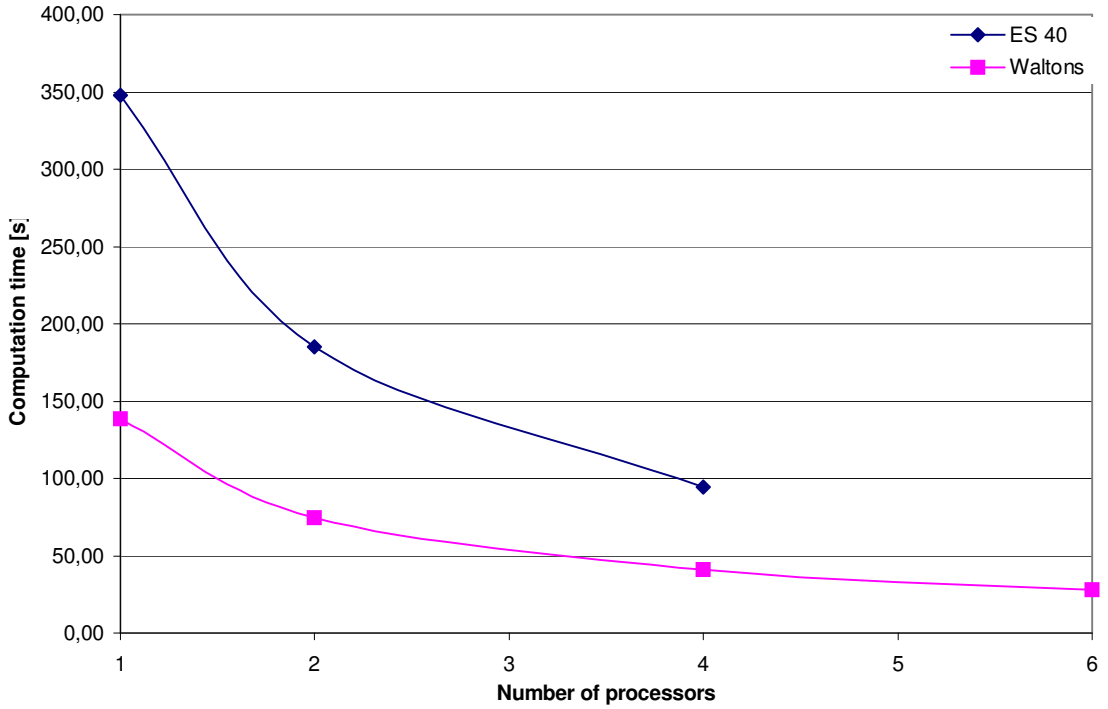

Figure 12: Test case C - Comparison DEC Alpha / Linux cluster

9. CONCLUSION AND OUTLOOK

The parallel version MP**ICE** shows excellent performance even on environments using relatively slow inter processor connection. The results on the Linux cluster are very promising even in view of the very low costs for a cluster solution compared to a dedicated parallel computer.

A cluster consisting of several Linux based workstations seems to be very attractive for the end users, who are not willing to spend a lot of budget for a super computer.

One topic not touched in this report is the parallelization of I/O. The output to the file system is very slow and the clear bottle neck of the system. The only way to overcome this problem will probably be the direct communication with the VR system (e.g. via sockets). In this stage of the project this seems to be impossible due to limitations in the VR software.

For the remaining project time the optimization of the code is still ongoing. There is some work to be done on the distribution of the case data at the start up of the simulation. Even the communication with the data manager (DMC) must be extended and tested extensively.

The authors are confident that a real time simulation (which requires the calculation of about 200 time steps per second) including temperature and smoke concentration can be obtained on the Linux-cluster “Lucidor” at PDC/KTH for a problem size of 100.000 cells provided that the currently existing communication bottleneck between the VR-system and MP**ICE** can be removed.

BIBLIOGRAPHY

[1] Redl, C., Brandstätter, W., *ICE – User Guid Version 1.0*, VirtualFires Project Deliverable 5.4, 2002.

[2] Foster, I., *Designing and Building Parallel Programs*, Addison-Wesley (1995), 379 pp.

[3] Pacheco, P. S., *Parallel Programming with MPI*, Morgan Kaufmann (1997), 418 pp.

[4] Snir, M., Otto, S., Huss-Ledermann, S., Walker, D., Dongarra, J., *MPI: The complete Reference*, MIT Press (1996), 336 pp.

[5] The MPI Forum, *MPI-2: Extensions to the Message-Passing Interface*, University of Tennessee (1997), 362 pp.

[6] http://www.mpi-forum.org

[7] http://www.lam-mpi.org

[8] http://www.pdc.kth.se

APPENDIX A – PSEUDO CODE OF "MPICE"

```
PROGRAM MPIce

!---Include the MPI library

INCLUDE "mpif.h"

!---Initialise MPI
  CALL MPI_INIT(IERR)
!---Communicate the number of processors
  CALL MPI_COMM_SIZE(MPI_COMM_WORLD, NP)
!---Every processor finds its ID
  CALL MPI_COMM_RANK(MPI_COMM_WORLD, MYID)

!---Processor 0 performs the data input
data_input : IF(MYID == 0) THEN

!---Read geometry data and perform the domain decomposition
          OPEN <ICE_Basename>.GEO
          READ geometry
          DO domain_decomposition
          DISTRIBUTE subdomain extensions to all processors

!---Set up the SEND and RECEIVE lists
          Loop over all boundary cells of each sub domain
          Determine SEND ID and corresponding RECEIVE ID
               for each boundary distribution
          Sort SEND and RECEIVE IDs and store it
               in the SEND and RECEIVE lists
          Communicate SEND and RECEIVE lists

!---Read case file, boundary definitions and initial data
          READ <ICE_Basename>.CAS, .INI or .CGNS
          DISTRIBUTE case and data information

ELSEIF(MYID /= 0)
          RECEIVE subdomain information, SEND and RECEIVE
               lists, case and data information

ENDIF data_input
```

```
!---All processors perform the calculation
solve_equations : DO time = 1, number_of_timesteps

!---Check if boundary conditions are to be updated
     IF(DMC_Update_BC) Update boundary conditions

     Impose boundary conditions at boundary points
     Update the solution at all grid points in the
          interior of the sub domains
     Copy DISTRIBUTION -> SEND list
     Communicate (MPI_SEND/MPI_RECEIVE) distribution
          functions via SEND/RECEIVE lists
          Copy RECEIVE list -> DISTRIBUTION

!---Check if results are required by the DMC
          dump_results : IF(DMC_Require_data) THEN
!---All data are gathered by Processor 0
               CALL MPI_GATHER(data)
!---Processor 0 send the data to
               IF(MYID == 0) CALL SENT_DATA_TO_DATABASE
          end dump_results : ENDIF(DMC_Require_data)

END DO solve equations

!---Finalize the parallel execution
CALL MPI_FINALIZE(IERR)

END PROGRAM MPIce
```