Porting Legacy Engineering Applications onto Distributed NT Systems.

N.K. Allsopp, T.P. Cooper, P. Ftakas
Parallel Applications Centre, 2 Venture Road, Chilworth, Southampton SO16 7NP
P.C. Macey
SER Systems Ltd, 39 Nottingham Rd., Stapleford, Nottingham NG9 8AD

Abstract

In this paper we present our experiences developing two distributed computing applications on NT. In both examples a legacy application is ported from Unix to NT and is then further developed to be a distributed application within the NT environment. We shall present two different approaches to managing the remote execution of tasks. One is a port of a serial vibroacoustic analysis code called PAFEC VibroAcoustic and the other is the parallelisation of the non-linear analysis modules of the LUSAS FE analysis package. We shall show in these two projects that it is technically possible to carry out scientific computing on a distributed NT resource

1. Introduction

In this paper we present our experiences developing two distributed computing applications on NT. In both examples a legacy application is ported from Unix to NT and is then further developed to be a distributed application within the NT environment. We shall present two different approaches to managing the remote execution of tasks. For both applications we shall present performance metrics and discuss the benefits of running distributed applications on NT.

The first application is a port of a serial acoustic analysis code, PAFEC VibroAcoustic, from Unix to run in parallel on a cluster of NT workstations. The port was funded by the European Union in the project PACAN-D. The University of Southampton Parallel Applications Centre (PAC) and SER Systems Ltd (code owners) carried out the parallelisation of the code before being assessed by an industrial end-user, Celestion International. Celestion are a small manufacturing company who design and build loudspeakers for home entertainment. As a small concern wishing to minimise costs, they were obviously attracted to NT. They became involved in the project to determine the conditions under which their cluster of desktop machines could also be used for running numerical simulations. In particular they wanted to test whether the cluster could be fully dual use, or whether they would still need to invest in extra computing resource to serve their simulations. The parallelisation of the code was implemented using MPI, enabling the same source to be used for NT as for Unix applications. Celestion tested the code on their cluster of single processor NT machines. The machines were dual use, in that they were used for other tasks during the day and were available for execution of large tasks over night.

The second application is the parallelisation of the non-linear analysis modules of the LUSAS FE analysis package from FEA Ltd. The code was well suited to a domain decomposition approach, and a major part of the effort in the project was the porting to NT of an intelligent resource manager (Intrepid), initially developed by PAC for heterogeneous clusters of Unix workstations. The issues that had to be addressed in performing this task were wide ranging. The Intrepid code was over 70000 lines of C and C++ and utilised a number of Unix tools to compile, not all of which were available on NT. In addition the functionality on which a resource manager relies, such as the methods of controlling remote execution, of monitoring tasks and of copying data sets all had to be completely redesigned. The work was funded by the EU as part of the project PARACOMP and evaluated by Messier-Dowty on a cluster of NT workstations. Again these machines were dual use (although unlike Celestion there was some spare capacity). One other feature of this cluster was its heterogeneity; machines varied between 166Mhz and 400Mhz clock speed, with a similar variation in memory and disk performance. One issue that had to be addressed is that of controlling access to machines. A problem that takes just over 4 days to solve in serial may only take 1 day to run on

4 processors. However if it prevents engineers from working on those machines for that time, there is no increase in productivity. The issue for Messier-Dowty was to have a code that would deliver a result in the same elapsed time, but would utilise the parallel speed-up to enable it to run only overnight.

2. Background to Projects

2.1 The PACAN-D Project

The aim of the PACAN-D project was to deploy a parallel version of the PAFEC VibroAcoustic finite element analysis package in a loudspeaker business. The work was based upon an existing parallel PAFEC VibroAcoustic code that was developed in a previous collaboration between SER Systems Ltd and the PAC. This code was developed in 1993 before the appearance of effective standard message passing interfaces. The code was also specific to the Intel iPSC/860 and Paragon platforms.

The objectives of the port can therefore be summarised as to port the old parallel PAFEC VibroAcoustic code from the Intel iPSC/860 to modern parallel systems and standards. As the code is under constant development it was decided to consolidate the parallel code with the latest version. As the code was originally developed on a UNIX based operating system and to reduce the number of platform dependant version of the code it was decided to select a message passing protocol, which could be easily used on different platforms. Therefore it was decided to use MPI for the message passing.

The PAFEC VibroAcoustic system is mainly written in FORTRAN, but some of the low level machine dependent parts are written in C. There are several hundred thousand lines of FORTRAN code. Original sections of the code were written in FORTRAN IV. More recently FORTRAN 77 and FORTRAN 90 have been used. The system is still being actively developed. There are many different executables in the system, some for running different stages of the analysis, and some for converting dictionary files from ASCII to binary form. It is possible to run jobs including user supplied FORTRAN routines, either as an efficient way of specifying data or to use a modified version of a standard system routine. Under these conditions the system runs a shellscript to perform a compile and link for the appropriate executable, while running the analysis. The system is currently developed in a Unix environment on HP workstations.

2.2 The PARACOMP Project

The PARACOMP project was an ESPRIT-TTN project whose main aim was "to demonstrate the deployment of a parallel code to perform composites analysis on a network of NT workstations, and to disseminate the benefits". Three companies were involved in the project:

- FEA Ltd. supplied the finite element analysis code for the project. They implemented a parallel solver, which was based on their legacy FORTRAN finite element analysis solver called Lusas.

- Parallel Applications Centre supplied the resource management system and the integration software for the project. An intelligent resource manager called Intrepid and developed by the PAC was used in the project. Intrepid was originally written for the UNIX operating system, but it has been ported to Windows NT for this project.

- Messier-Dowty tested the software for the analysis of composite materials.

The Intrepid parallel scheduler is a tool that allows the scheduling, control and execution of a number of tasks (programs) on a heterogeneous network of workstations. It allows the user to control both the resources used by the scheduler to run programs on, as well as the programs running themselves.

The general structure of the Intrepid parallel scheduler (“set in stone” by the Unix version of the software) is shown in figure 1.

As can be seen from the figure, the Intrepid parallel scheduler consists of two main components:

pacS: This is the main scheduler executable and the program that contains the global network and schedule information for the system. This module takes all the scheduling decisions in Intrepid. There is only one instance of pacS running in a functioning Intrepid system.

pacD: This is the daemon that provides support to the Intrepid parallel scheduler. There is one of these daemons in each "node" in the network. An Intrepid daemon's main function is to pass back information to pacS about the state of the "node" which it is controlling, and to launch and monitor applications. Secondary functionality includes all the necessary actions to start an application (e.g. creation of temporary di-

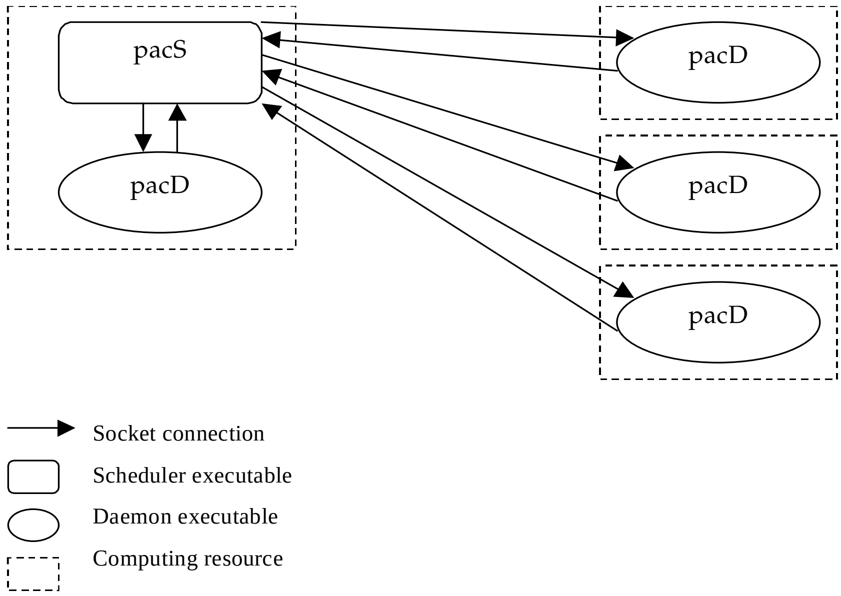

Figure 1: General Structure of the Intrepid system.

rectories, if required, transfer of all input files before and output files after the execution, etc.).

Communications between pacS and the various pacD daemons are implemented via TCP/IP sockets. This allows Intrepid to control heterogeneous networks. The user of Intrepid submits a “schedule” to the system. A schedule consists of a number of tasks (processes) to be executed and the dependencies between them. The Intrepid system then assigns tasks to specific processors in the network, so that the overall time of execution for the whole schedule is minimal. The system also performs any data transfers that might be required to allow a task to run on a specific machine. Any number of users can submit any number of schedules to Intrepid simultaneously and the system is able to deal with the added complexity.

3. Techniques Used

3.1 The PACAN-D Port

When porting onto the PC with the Digital Visual FORTRAN compiler it was found to be very time consuming to set up the visual environment. Indeed it was not clear how best this should be done for a system containing thousands of routines and with multiple executables. Attempts at setting up the environment were hampered by a bug in IE3, which has since been fixed, which prevented large numbers of files being copied at once. Consequently everything was done using command shells. This did not help at a later stage with debugging the test jobs which failed, as less information was available on error conditions. There are limits on the command line length, but the shells generated in Unix ports depended on long lines to link in all the appropriate libraries. Thus it was necessary to use resource files rather than wild cards and this necessitated a fundamental change to the structure of one of the dictionary files. Problems were encountered with handling sequential files, which seem to be compiler dependent. Furthermore there were problems with opening files, which did not work in dynamic link libraries. The machine dependent FORTRAN parts were handled by a rewrite of routines from a previous PC port, which used a DOS extender, and worked using DOS interrupts. The Digital Visual FORTRAN compiler was relatively strict, compared with those used in many Unix ports, and detected some occurrences of inconsistent numbers of parameters in routine calls. As always code errors found were fixed in the development level, making the code more robust and portable for the future.

The PAFEC VibroAcoustic system consists of a suite of programs called *phases*. There are 10 phases in total, phase 1 to 6 performs the pre-processing of the element data of the model to be analysed. Phase 7 carried out the solution of the equations, the numerically intensive part of the whole system. Phases 8 to 10 handle the plotting and visualisation of the calculated data. Information passes between the phases via files held on disk. It was apparent that this phase structure of the PAFEC VibroAcoustic code need not be parallelised in its entirety. The phases are essentially standalone programs each of which may be parallelised independently of the others. Profiling the code showed that phase 7 carried out most of the computationally intensive operations. The most striking result of the profiling exercise was that a majority of the execution time was spent within a very small number of routines i.e. 50 routines out of the 18000 routines of the whole PAFEC VibroAcoustic code. It was therefore decided to concentrate on these numerically intensive routines which occurred at four points within phase 7.

The underlying ethos to the parallelism of phase 7 of the PAFEC VibroAcoustic is that the master processor proceeds through the code in the same way as the serial version. At the point where the master processor is about to enter a numerically intensive subroutine a message is sent to the other slave processors to indicate which routine is being entered. On entering the numerically intensive routine, all the processors perform an equal amount of the required calculation. When the routine is finished the master processor continues to progress on through the serial code whilst the slave processors wait for the next numerically intensive section to be reached by the master processor.

For a fully coupled vibroacoustic solution, using an acoustic BE mesh coupled to a mesh of structural FE the set of equations to be solved can be written as:

$$\begin{bmatrix} & [S] & [T] \\ [G] & [E] & [H] \end{bmatrix} \begin{Bmatrix} \{U\} \\ \{p\} \end{Bmatrix} = \begin{Bmatrix} \{F\} \\ \{p_I\} \end{Bmatrix}$$

Where:

[S] Contains the structural stiffness matrices which are large and sparse,

[H], [G] are small dense matrices derived from the BE formulation.

[T], [E] are coupling matrices.

Where {u} is a vector of displacements on the structural mesh, {p} is a vector of pressures on the BE. [S], [C] and [M] are the structural stiffness, damping and mass matrices and are large and sparse. [H] and [G] are small dense matrices derived from the BE formulation. [T] and [E] are coupling matrices. Sometimes the structural representation is simplified using a smaller modal model of the structure, but this does not permit variation of properties with frequency, which occurs for the surround and cone on a loudspeaker. The current work was based on a full solution of the above equation, using the four stages below.

- Stage 1 - FE merging/reduction. The dynamic stiffness matrix and coupling matrix [T] are formed by merging contributions from individual finite elements. The equations are simultaneously solved. Degrees of freedom are eliminated as early as possible. The matrices are shared between processors and the elimination is done in parallel.

- Stage 2 - forming the BE matrices. For each collocation point on the BE surface it is necessary to integrate over the surface to form a row in the BE matrices. Parallelization is achieved by sharing these collocation points between the processors. Distributed BE matrices are formed.

- Stage 3 - reducing the BE matrices. The matrix $-\omega^2\rho[G][E]^t$ is formed and reduced using resolution with the structural elimination equations from stage 1. As above the matrices are distributed between processors.

- Stage 4 - Gaussian elimination of final equations. The resulting compact dense set of equations is solved using a parallelized form of Gaussian elimination on the distributed matrix.

3.2 The PARACOMP Port

3.2.1 Porting Intrepid to Windows NT

The Intrepid system was originally implemented on UNIX systems. The dependencies of the source code on libraries and development tools were kept minimal (even in the UNIX world there are a large number of different flavours of UNIX and Intrepid was designed to be portable). Intrepid relied on the following external dependencies:

- C++ compiler. In UNIX, native C++ compilers on the corresponding platforms were used to compile Intrepid. Although the code is not ANSI C, or POSIX compliant, the transfer from UNIX to Visual C++ did not present large difficulties, as the main bulk of the code is written as a console application and does not rely on any GUI functions.

The UNIX GUI to Intrepid (which existed in Tcl/Tk and X/Motif incarnations) has been lost with the transit to Windows NT.

- Socket library. The windows socket library presented minimal problems during the porting process. Problems were mostly solved with the use of C pre-processor macros to distinguish between dissimilar UNIX and Windows32 API socket function calls and constants. The reason that it was decided to use sockets for communications between the various components of Intrepid was historical. Sockets are the de-facto standard for fast low-level communications in all UNIX systems. The use of sockets though, had another consequence: the Intrepid resource manager can be used to control a heterogeneous network of workstations, running different operating systems and not having access to a common, uniform file system. The system is able to distinguish between a number of different operating system types and treat workstations running those operating systems accordingly. It also implements file transfers for input and output files (through sockets, or using the Windows32 drive mapping mechanism and file copying functions to perform the transfer).

- Environment variables. Although environment variables exist in the case of Windows NT as well as UNIX, a cleaner solution in Windows NT would be to have application information held in the registry and not in environment variables. This was partially implemented during the PARACOMP project. Although all the environment variables are available through the registry, they have been introduced under the HKEY_LOCAL_MACHINE key. Since Intrepid is a multi-user system, this has security implications for the system, since all users will see the same values for the environment variables. In UNIX, Intrepid relies on various shell scripts being executed at login for the user, in order to correctly implement security.

- rshd. At start-up, the Intrepid system has to start (daemon) processes on every workstation that will be controlled by the system. In the case of UNIX machines, as long as the workstation is reachable and the username is accepted, Intrepid can execute a remote shell command on the remote machine and start up the (daemon) processes. Unfortunately, Windows NT does not come with a remote shell daemon (rshd) as standard, and most system administrators are not willing to give such powers to their users. The solution we have come up with was to implement these (daemon) processes as Windows NT services. Instead of Intrepid explicitly spawning these processes at start-up, these processes always exist on the background, waiting to connect to the resource management system. In reality they are blocked listening on a socket, so impact on system resources is minimal while Intrepid is not running.

- lex/yacc. Part of the Intrepid system was a parser that was written using lex and yacc. The only versions of lex and yacc that we could have access to on Windows NT would be the GNU implementations. However, the UNIX code was written with the BSD versions of lex and yacc, which has some slight differences. The solution we used was to execute the BSD lex and yacc tool implementations on a UNIX workstation to create the C code for the parser. We then performed some minor changes to the included files and definitions in the source code (a program was written to do this automatically) and finally included the resulting code in the VC++ project for Intrepid.

- Data transfers. The Intrepid version of UNIX used two different methods to perform data transfers: direct copying using the standard UNIX copy command cp, or a pair of helper programs that are used to implement the data transfer using sockets. The former method cannot be used in NT since it presumes the existence of NFS for it to work. The latter method is generic enough to work in the context of a Windows NT environment. However, after the porting of helper programs from UNIX to NT, we found that the system was too slow. We implemented an alternative data transferring mechanism that uses drive mappings to effect the file copy. In Windows NT, one computer can have access to the file store of another computer through the drive mapping mechanism. There is a notion of network file names called by Microsoft UNC (Universal Naming Convention). However, these UNC file names cannot be used directly with the Win32 API functions that handle files[5]. The solution Intrepid has come up with, is to map the UNC directory to a free drive letter and then use a normal file path (containing the new drive letter, of course) to perform any operations on the file. Once the file is copied to a local execution directory the drive letter can be unmapped, so that it can be used again on some other file transfer. The limitation of this method is that the system administrator must set up the system in such a way that the products that are defined in the task graph file point to the intended files. This means that if a product is defined to consist of a file at location "\\machine\dir1\file.dat", then the directory **dir1** must be shared on the workstation named **machine**.

3.2.2 Interfacing Intrepid with LUSAS

Lusas consists of a number of components. The two main ones are the solver itself (Lusas solver) which is the legacy Fortran code for the finite element analysis, and the GUI front end (Lusas modeller) which is a Windows32 application and is used to design the model and present results.

The usual mode of operation for Lusas is that a user will design a model using the modeller, use the solver on the created model and finally display the results on the modeller window. To start the solver on a model all a user has to do is click a button.

We wanted to keep the same style of operations in PARACOMP. However, there are a number of steps involved in generating the parallel solution that have to be hidden from the user. When a model is ready to submit to the parallel solver, the first thing that PARACOMP must do is split the model into the appropriate number of sub-domains in order to perform the finite element analysis in parallel. Appropriate data files have to be generated for each of the sub-domains of the model. An input file to Intrepid has to be generated (the task graph file) that contains information about the dependencies between the composite parts of the finite element analysis. For each of the sub-domains a different process running the parallel solver will have to be spawned on the target machine. Before the process is started the necessary data files that represent the domain have to be copied over to a local directory on the target machine. When the finite element analysis run is finished the result files have to be copied back to the machine from which the user submitted the job. All this is handled by the Intrepid system and a small Visual Basic script that performs the domain decomposition and essentially "glues" Lusas and Intrepid together [5]. All the user input required is the number of sub-domains that the model must be split into.

There is also a need for another GUI program, which would configure Intrepid for the specific network that Intrepid is used on. Because of time limitations and because PARACOMP only addressed networks of Windows NT workstations, the software that performs this function does not have the full functionality of the UNIX GUI for setting up Intrepid. It assumes that the network only contains NT machines and the options that the user can set at configuration time are comparatively limited.

4. Results

4.1 PACAN-D

The following results were obtained using the test case supplied by Celestion International. The size of the test case represented the maximum size of problem that could be simulated on their existing single machine. This particular test case is representative of a typical loudspeaker system under test at Celestion. The system represented a half model, its parameters are 141 structural elements, 1917 structural degree of freedom and 914 acoustics degrees of freedom (interior and exterior).

This test case was run on a cluster of 166 MHz Pentium Pro machines linked together with standard 10Mbit Ethernet. To reduce traffic conflicts over the Ethernet the cluster were linked via a switch that effectively isolated the cluster from the rest of the network. The same test case was then run using the same code but compiled with a different version of MPI[1,4] on a shared memory SGI machine consisting of eight 75MHz processors although only 4 processors were used.

Stage	1 Proc.	2 Proc.s	3 Proc.s	4 Proc.s
1	73	74	73	75
2	165	86	58	41
3	975	484	315	227
4	132	95	85	104
Total	1345	739	531	447

Table 1: Results on 75MHz SGI Machine using MPICH.

Stage	1 Proc.	2 Proc.s	3 Proc.s	4 Proc.s
1	49	56	66	70
2	81	42	31	18
3	508	286	173	134
4	76	84	100	96
Total	714	468	369	318

Table 2: Results on 166MHz Pentium Pro using WMPI 1.01.

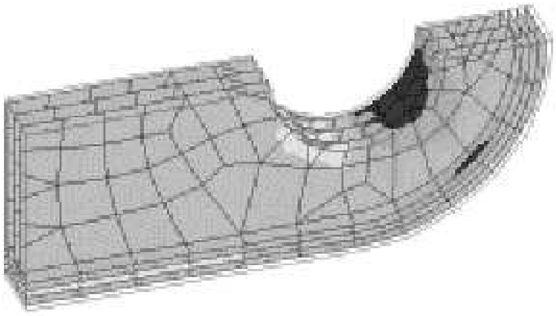

Figure 2: Landing gear component from Messier-Dowty.

4.2 PARACOMP

The final stage of the PARACOMP project involved the use and evaluation of the final software by an end user, which was Messier-Dowty. The software was installed and run on a model defined by Messier-Dowty. The model is one half of a lug which is the main load bearing component of a landing gear. The load is transmitted in plane through the landing gear and up through the lug into the wing. The aim of the investigation was to identify the onset of delamination in the lug, and thence provide input into the type of physical testing to be carried out.

The experiment consisted of the parallel finite element analysis of a design decomposed into two sub-domains. The relative size of the two sub-domains is important since the target machines had different processing power. One (which we will name machine A) was a 400MHz Pentium II machine and the other (machine B) was a 166MHz Pentium. The time taken to execute the full sequential solution on machine A was 46 minutes and on machine B 110 minutes. If we assume that the execution time for a sub-domain on a processor is a linear function of the size of the sub-domain, then we can calculate that the optimal run for the parallel case is 33 minutes. This can be seen as the minimum value of the execution time curve in figure 3. When the software was used to perform a parallel run on the same model, the actual execution time was 29 minutes. We think that such a discrepancy exists because the code is I/O bound. By distributing the I/O between the two machines, we achieve a super-linear speed-up.

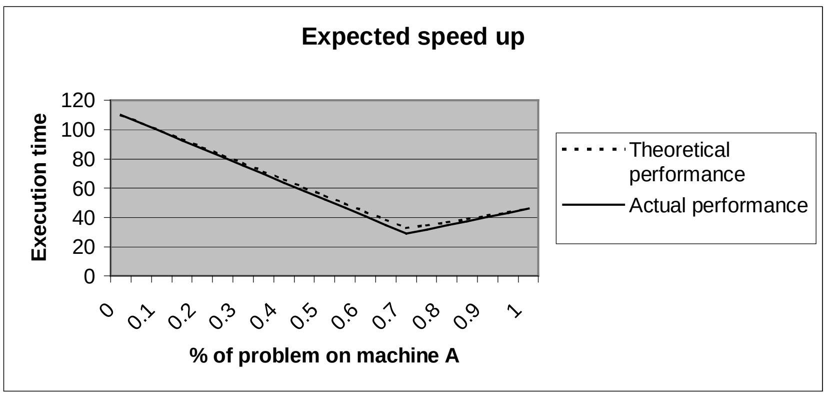

Figure 3: Results on a heterogeneous cluster of a 400MHz Pentium II and a 166MHz Pentium Pro.

5. Conclusion

The MPI protocol has been previously proven to provide portability of source code between UNIX platforms. In the PACAN-D project we have shown that this portability extends to NT, and that performance is sufficient to yield satisfactory speed-ups.

One of the main concerns about NT is that absence of remote shell procedures make controlling the remote execution of jobs difficult. However the mapping of a Unix daemon to an NT service has been demonstrated in the port of Intrepid to NT. In addition it is worth noting that the MPI daemons are also implemented as NT services, and it is our conclusion that the NT service functionality provides a robust way of managing remote execution in the NT environment.

Three methods of data transfer between machines have been tested: using WMPI calls, using sockets and using drive mappings to move files. The WMPI layer is fast enough to support serious numerical computation. A comparison between the use of sockets and drive mappings was made as part of the Intrepid port, and drive mappings were found to be faster. It is not clear whether this is fundamental to the design of sockets on NT, or because of the implementation of the transfer within Intrepid.

With the ever-increasing processor speed of NT platform coupled with the decreasing unit cost, the proposition of porting legacy commercial codes to NT is increasingly attractive. We have shown in these two projects that it is technically possible to carry out scientific computing on a distributed NT resource. The next stage of the work will be to assess whether this computation can be carried out on truly dual use hardware. If this proves to be possible, then the potential cost benefits for smaller organisations that wish to invest in numerical simulation become enormous.

6. Acknowledgements

This work has been funded as part of ESPRIT projects, no. 24871 PACAN-D and no. 24474, PARACOMP, both part of the HPCN TTN Network supported by the EC. The authors would like to thank MPI Software Technology Inc. and Genias Gmbh for their support of this research

.

7. References

[1] Message Passing Interface Forum, MPI: A Message-Passing Interface Standard, May 5, 1994, University of Tennessee, Knoxville, Report No. CS-94-230

[2] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, A high-performance, portable implementation of the MPI message passing interface standard.

[3] W. Gropp and B. Smith, Chameleon parallel programming tools users manual. Technical Report ANL-93/23, Argonne National Laboratory, March 1993.

[4] J.M.M. Marinho, Instituto supererior de Engenharia de Coimbra, Portugal.

[5] Sing Li, Professional Visual C++ ActiveX/COM Control Programming, Wrox Press Ltd, 1997.

[6] Microsoft Corporation, Moving UNIX Applications to Windows NT, version 0.9 (part of Visual C++ online documentation), 1994.

[7] Meecham, K; Floros, N; Surridge, M. Industrial Stochastic Simulations on a European Meta-Computer, Proceedings 4th International EuroPar Conference, EuroPar 98 Parallel Processing.

[8] Cooper, T. Case studies of 4 industrial meta-applications, Proceedings HPCN Europe 99.