Monograph:
Chipperfield, A.J., Fleming, P.J. and Browne, A.R. (1993) Design and use of the MATLAB Parallel Processing Gateway. Research Report. ACSE Research Report 486 . Department of Automatic Control and Systems Engineering

Design and use of the MATLAB Parallel Processing Gateway

A. J. Chipperfield, P. J. Fleming and A. R. Browne

Department of Automatic Control and Systems Engineering
University of Sheffield
PO Box 600
Mappin Street
Sheffield S1 4DU

e-mail: A.Chipperfield@shef.ac.uk
P.Fleming@shef.ac.uk
A.Browne@shef.ac.uk

September 28th 1993

ACSE Research Report No. 486

Abstract: This report describes a collection of software utilities that together form a "parallel processing gateway" for the computer aided control system design software package, MATLAB. These utilities allow the control engineer to configure and boot processes on the parallel computer and access concurrent and parallel routines transparently from the MATLAB command line. Here, the requirements of such a gateway, its design, implementation and use, and features that enable the control engineer to readily exploit a parallel computer are described and discussed. In addition, the performance of the gateway is assessed in terms of the communications achievable between MATLAB and the parallel computer.

Design and use of the MATLAB Parallel Processing Gateway

A. J. Chipperfield, P. J. Fleming and A. R. Browne

Department of Automatic Control and Systems Engineering
University of Sheffield
PO Box 600
Mappin Street
Sheffield S1 4DU

1. Introduction

Parallel computers appear to offer an attractive solution to the computational problems associated with parametric optimization techniques and Computer Aided Control System Design (CACSD) in general. However, despite the falling cost of network accessible parallel computers, there has been a relatively low uptake in their use in such applications. There are three principal reasons for this. With the wide range of hardware architectures and programming languages, programs written for one class of machine are rarely portable to another. The cost of developing a parallel program is greater than that of its sequential equivalent due to the extra complexities inherent in the design and implementation process. Finally, commercial CACSD packages seldom support the features required to securely and quickly communicate with the parallel computer.

This report describes a collection of software utilities that together form a "parallel processing gateway" for the CACSD software package, MATLAB [1]. These utilities allow the parallel CACSD user to configure and boot processes on the parallel computer and access concurrent and parallel routines transparently from within the MATLAB environment. Acting as a buffer between MATLAB and the Transputer operating system, GENESYS [2], these utilities mask the user from the extra difficulties encountered in employing parallel programs in CACSD. Here we describe the requirements of such a gateway, its design and implementation and use and a number of other features that enable the control engineer to readily exploit a parallel computer in CACSD.

A demonstration of the MATLAB parallel processing gateway appears in ACSE Technical Report No. 487 [3], accompanied by an interactive demonstration script file.

2. The Parallel CACSD Environment

The parallel CACSD environment, shown in Fig. 1, is based around the MATLAB software package with a number of utilities to support parallel processing. Rapidly becoming a *de-facto* standard for the control engineer, MATLAB is a high performance interactive software tool for scientific computation and engineering analysis. The standard interface allows the user to solve mathematical problems in a direct manner without having to generate specialised software procedures. New commands can readily be constructed from the supplied functions and

incorporated into the users own system through the use of m-files. The m-file is a sequence of ordinary MATLAB statements, possibly including references to other m-files or self-reference, that take arguments from the command line and return results to the workspace. Through the use of m-files, collections of related routines can gathered together as toolboxes. For example, toolboxes exist for optimization, control system design, neural network simulation and signal processing.

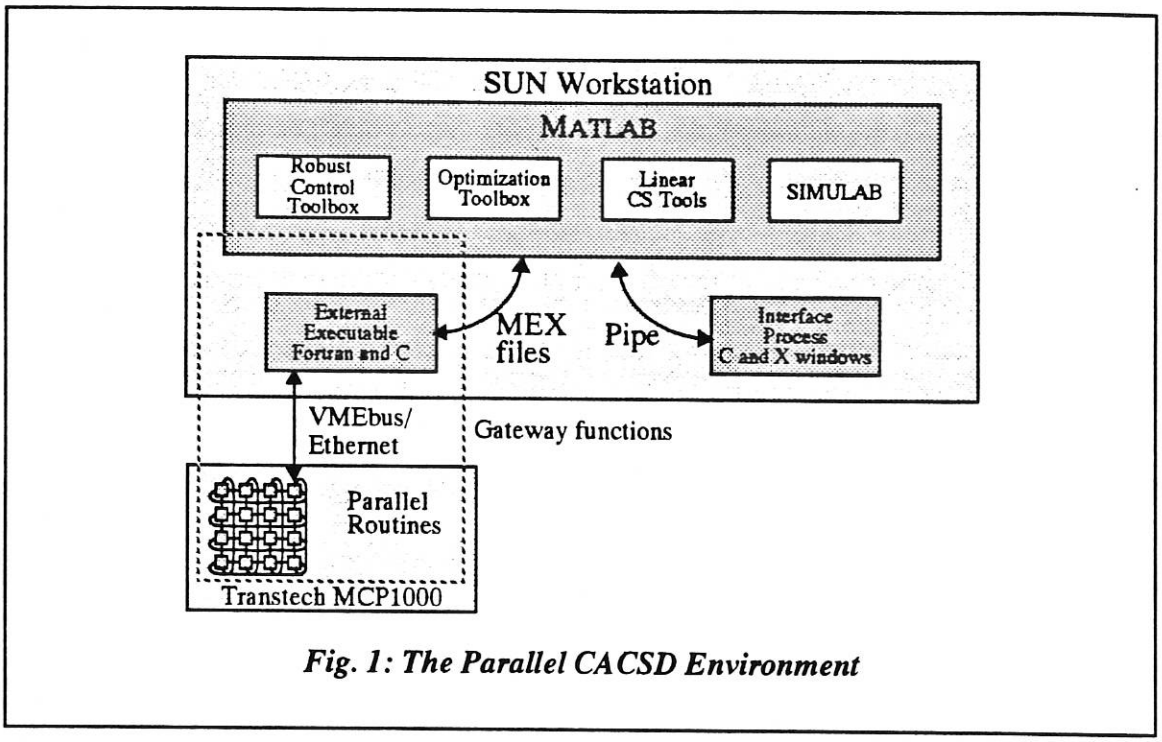

Fig. 1: The Parallel CACSD Environment

MATLAB also offers a facility essential for fast and secure communications with a parallel processing platform, the mex-file. Mex-files allow the user to dynamically link their own C and Fortran code into the MATLAB address space. It is through the use of mex-files that we have constructed the gateway between MATLAB and the parallel processing platform. The mex-file interface is described in more detail in section 3.

The parallel processing gateway is a collection of functions that allow the user to configure and boot processes on the parallel platform and access these routines transparently from within MATLAB. Routines running on the parallel platform and called by the gateway appear as if they are built-in functions in the same manner as m- and mex-files. Thus the gateway may be used to incorporate concurrent and parallel routines directly into MATLAB for use in a number of ways. In this study we have concentrated on applying parallel processing methods into the optimization process. Specifically we have addressed the parallelism inherent in multiobjective optimization problems arising from the evaluation of design goals at each iteration of the optimization algorithm. The gateway described here is, however, more general in nature and can be applied to a wide range of parallel processing tasks within CACSD.

In addition, to ease the problems of constructing code for simulating control systems and evaluating design goals, we have produced a library of core routines for CACSD problems.

Drawn from the basic MATLAB commands, LINPACK [4] and EISPACK [5], and the Control System Toolbox, this library can be used in the construction of programs to run as mex-files or through the gateway on the transputer platform. A number of the higher level routines, such as time and frequency responses, have been parallelised to further exploit the parallel processing platform.

3. The MATLAB Mex-file Interface and Utility Routines

The MATLAB mex-file interface allows users to dynamically link pre-compiled C and Fortran code into the address space at run-time. This is achieved by the provision of a library of interface routines and a driver for the compiler/linker that allows the dynamic location of the executable code called cmex and fmex for C and Fortran respectively. All parameters to and from mex-files are passed through the function entry point using the standard MATLAB Matrix data-structure. The entry point and matrix data structure are defined as follows:

```
user_fcn( nlhs, plhs, nrhs, prhs )
int nlhs ; /* Number of lhs arguments */
int nrhs ; /* Number or rhs arguments */
Matrix *plhs[] ; /* Array of nlhs pointers to lhs arguments */
Matrix *prhs[] ; /* Array of nrhs pointers to rhs arguments */

typedef struct mat {
    char *name;      /* ptr to matrix name */
    int type;        /* variable type (1=text) */
    unsigned int m;  /* row dimension */
    unsigned int n;  /* column dimension */
    double *pr;      /* pointer to real part of matrix */
    double *pi;      /* pointer to imag part of matrix */
} Matrix;
```

The mex-file entry point is **user_fcn**. The integer variables **nlhs** and **nrhs** contain the number of inputs and outputs to the mex-file respectively. The parameters **plhs** and **prhs** are pointers to arrays of pointers of length nlhs and nrhs containing the input and output parameters of the **Matrix** data type. Non-complex matrices have the pointer to the imaginary part, pi, set to NULL. Text variables have the type variable set to 1 and are recovered by converting the array of doubles to type char entry by entry

Table 1 lists the C programming functions available to the mex programmer. Of particular interest in the design of the parallel processing gateway are the functions matlab_fcn and matlab_trap. The function matlab_fcn allows mex-files to call matlab commands, m-files and other mex-files. The function matlab_trap allows the standard error handler to be overridden whilst using

matlab_fcn, allowing the mex-file to handle the error situation rather than returning to MATLAB.

Table 1: MATLAB Mex-file Routines

Function name	Description
create_matrix	Create a matrix on the MATLAB heap
free_matrix	Free matrix and return memory to MATLAB heap
mex_calloc	Arbitrary memory allocation
mex_free	Free memory allocated with mex_calloc
mex_error	Issue error message and return control to MATLAB
mex_printf	Print a string to the display
matlab_fcn	Call a MATLAB function
matlab_trap	Set error trap handling for matlab_fcn
get_global	Get pointer to global variable

4. Interfacing a Parallel Platform to MATLAB

MATLAB provides a facility to allow precompiled C and Fortran code to be dynamically linked into the MATLAB address space called mex-files. The Transputer Operating System, GENESYS, allows communications between a process running on the host computer and individual processes running on the parallel platform. In addition, GENESYS handles all the message buffering and routing between nodes. Thus a mex-file that is also a GENESYS process may be used to communicate data and instructions between MATLAB and the parallel platform. A routine running on the parallel platform is called from MATLAB using the standard form, i.e.

[lhs1, lhs2, ..., lhsm] = parallel_command(rhs1, rhs2, ..., rhsn) ;

Right hand side arguments are input parameters and results are returned as the values of the left hand side arguments. Fig.2 shows how an interface between MATLAB and the parallel platform may be achieved to evaluate a set of functions concurrently on individual compute nodes. The function evaluations are called using the MATLAB command line

Y = demo_func(X) ;

where X = [x1, x2, x3] and are the individual function parameters and Y = [y1, y2, y3] are the corresponding function values of f1, f2 and f3.

On invocation, the mex-file demo_func is passed the one dimensional matrix X containing the input parameters. The mex-file converts the matrix data-type used by MATLAB into the message format used by GENESYS and passes the input argument(s) to the host node on the parallel platform. The host node receives the GENESYS message and routes the rhs arguments to the nodes

that this command will use (T1, T2 and T3). The compute nodes perform the appropriate computation and return their results to the host node. The host node multi-reels the incoming messages - receives messages from all sending nodes concurrently - and bundles the results into a single message for transmission to the GENESYS/mex-file process on the host workstation. Upon receipt of this message, the mex-file converts the result data into the MATLAB matrix data structure and returns these results as the lhs arguments to the MATLAB workspace, in this case to the vector Y. In this way the mex-file used for this example, demo_func, acts as both a MATLAB function and a GENESYS process having access to the transputer network.

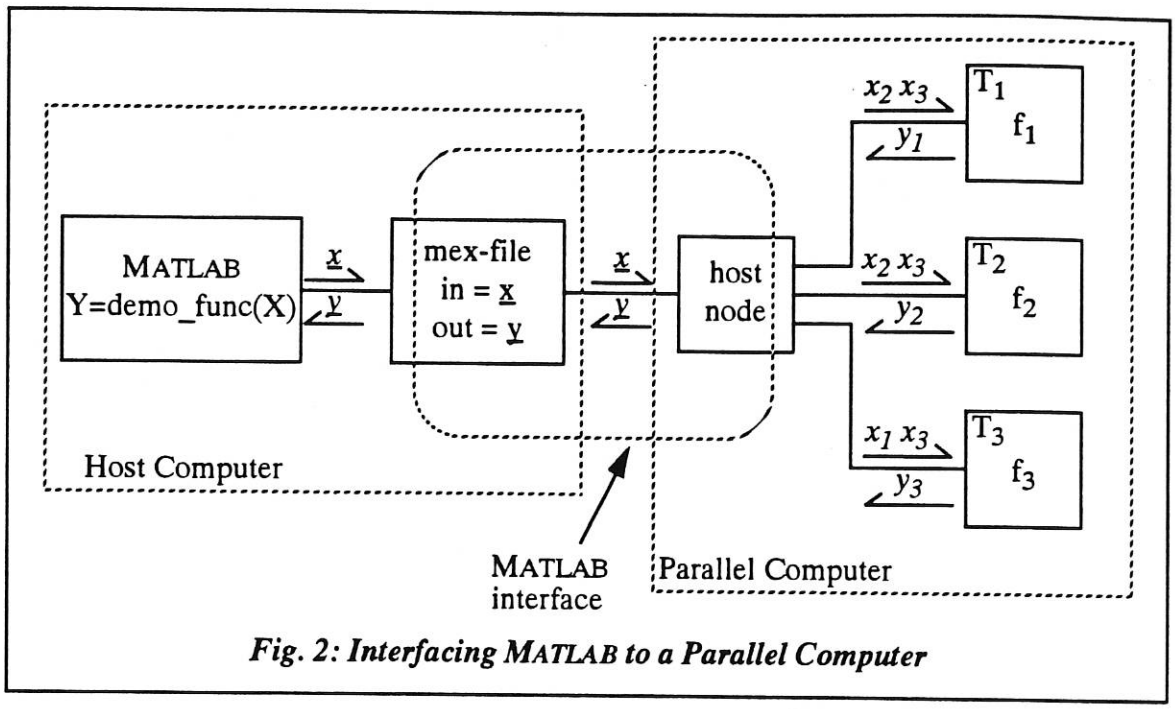

Fig. 2: Interfacing MATLAB to a Parallel Computer

Clearly, the overall execution time is going to be that of the largest process plus the communications overhead associated with passing incoming data from MATLAB to the compute nodes and returning the results into the users workspace.

In this example, the mex-file/GENESYS process and host transputer process have explicit information about the parameters that they are routing between the MATLAB environment and concurrent processes. Whilst this works well for individual parallel function calls, it does mean that a new MATLAB gateway routine has to be constructed for each parallel function and that particular care is made in ensuring that multiple processes on individual compute nodes receive and return the correct parameters. For example, if there are two processes, P_{21} and P_{22}, running on compute node T_2, then the host node must ensure that the parameters are sent to the correct process. This requires the calling MATLAB process to include information to determine which process running on the compute node should be used and introduces an extra level of complexity in using this form of gateway.

5. Requirements of the Parallel Processing Gateway

In order to transparently support parallel processing the gateway must fulfil a number of requirements. These are discussed in this section.

- Support the full C mex-file interface between MATLAB and remote processors. An initial requirement of the gateway was that any C mex-file could be recompiled and run on any Transputer in the network transparently to the user. Thus, the full C mex-file interface described in Section 3 has to be available to any processes running on the parallel platform under the control of the gateway. This allows the input and output parameters and other data to be passed to functions in the same manner as the standard C mex-file. A library of Transputer gateway interface routines are used in conjunction with the Transputer C compiler to mask these operations from the user where necessary (see section 3.5.1).

- Support MATLAB function calls from remote processors including reading the values of global variables. This is an extension of the first requirement and allows users of the gateway to make use of any of the MATLAB functions by passing a complete MATLAB function call back to the users workspace and receiving the results on the local node. This allows the user to make use of the MATLAB functions not implemented in the transputer library, whilst still exploiting the parallel platform. Similarly, global variables in the user workspace may have to be read or updated by functions running on transputer nodes. Care must therefore be taken to ensure that global variables are not subject to corruption when more than one remote process reads or writes to that variable. This can be achieved by designing co-operative parallel programs that propagate the values of global variables to all nodes requiring those variables through a single call to the read global variable function.

- Allow execution of any C mex-file on any transputer. By allowing any process to run on any processor, the user is free to define and change the network configuration and process placement without the need to recompile the functions being used. This means that the gateway has to handle all of the communications between the MATLAB and remote processes. A single function distributed across many nodes would therefore either have to route all inter-process communications through the gateway or use a single process as a "master" that may be placed on any node in the network, and use fixed process placement for the remaining processes. This detail is described further in Section 6.1.

- Allow the concurrent execution of many processes on the transputer platform. This requirement implies that the gateway must be capable of passing, possibly different, messages to a number of processing nodes at any call and returning the results in a predictable order to the workspace. That is, a number of different operations with varying numbers of input and out parameters should be able to be called in a single command line in such a way that all the operations in the command line are executed in parallel with one another.

- Allow concurrent execution of multiple processes on a single transputer. At any time more than one process should allowed to be active on any processor. The gateway has therefore to mask messages to processes in such a way that processes on a given node

receive the correct parameters and return results to the correct data-structure in the gateway. This provision is necessary so that a processes can remain loaded and blocked on a node, thus reducing the time overhead when making a call to it.

- Support communications between processes on the transputer platform. Under certain conditions, processes running on the parallel platform may need to communicate with each other through the gateway rather than through the GENESYS operating system. This may be the case when trying to determine the load balance of a number of coarse grained tasks. Here the gateway receives a message from a processes containing a number of parameters and the name of the destination process. The gateway then routes this message to the node and process corresponding to the desired destination.

- Provide process management for remote processes inside MATLAB environment. In order to allow the user a degree of control over processes running on the parallel computer it is necessary for the gateway to provide facilities for loading and killing remote processes. This is achieved via process tables. At any one time more than one process table may exist, allowing tables to be built that relate directly to routines that are commonly called together. When a process is loaded onto a compute node a new process table may be created or an entry may be added to an existing table. However, it is possible that a process may be killed and an entry to it still remain in a process table. If a non-existent process is called the gateway will return an error message.

6. Design of the Transputer Gateway

The transputer gateway is made up of a collection of routines for configuring and booting the parallel platform, loading and unloading processes onto processors and executing remote function calls. The design, implementation and use of these routines are discussed in detail in this section. Section 6.1 discusses the utility routine that support the use of the gateway. This includes the functions for configuring and booting the parallel platform, examining the state of processors and message buffers and resetting the platform. Section 6.2 reviews the gateway core routines that are used to link the parallel platform into the MATLAB environment.

6.1 Support Routines

The support routines are not part of the parallel processing gateway as such, but are a collection of tools that faccilitate the configuration and running of the parallel CACSD environment.

6.1.1 Mex-file transputer compiler: tcc_mex

The program tcc_mex is the transputer equivalent of the MATLAB utility command cmex. This program is essentially a preprocessor for the Transputer compiler that allows C mex-files to be compiled as transputer programs. For example,

tcc_mex foo.c bar.to -lcomplex

is a legal command that will produce transputer executable code with gateway communications protocols. The file foo.c is a C mex source files and bar.to is a transputer object code file. The

library 'complex' would be linked to the executable by the -lcomplex.

The transputer compiler is used to compile the original C mex source file and the resulting object code is then linked with the 'transputer gateway server' library. This library serves a number of purposes, it translates between messages sent over the GENESYS communications channels and the MATLAB Matrix data structure, shadows the mex function calls and provides local process management. The server gateway stays active until it receives a message to terminate. Thus a routine compiled using tcc_mex will only need to be loaded once and can be called as often as required. This is similar to the way MATLAB retains the mex-file entry point in its address space after a routine has first been invoked. This means that the overhead associated with loading a process is not carried on every call made to that process. When the routine makes a call to one of the mex utilities of table 1, the gateway library is used to generate a message which is sent back to the host machine to perform the desired operations. Results, if any, of these operations are returned to the process, again through the gateway.

Parallel routines may be accommodated by the gateway if all data transmission between MATLAB and the compute nodes is handled by a single mex-file process and all inter-process communications by the transputer operating system. Only the process under the control of the gateway will require compilation with tcc_mex, other processes should be compiled with the standard compiler.

6.1.2 Transputer configuration and booting: tboot

Before the gateway can be used, the parallel computer will have to be configured and the operating system loaded. The Transtech MCP1000 parallel platform supports four sites of up to 8 compute nodes each. Each site may be used by a different user, possibly from different machines, or an individual user may user more than one site. Therefore, up to 4 users may have simultaneous access to different sites on the parallel platform. Each site has hard-wired links between nodes in that site and a number of software configurable links that may be used to connect links across sites. Whilst this can be achieved using GENESYS utilities, it is desirable to be able to perform the system configuration from within the MATLAB environment. The function tboot is an m-file that allows the user access to the GENESYS booting utilities. For example, to boot a 2 by 4 array of transputers, starting at the first available site, the following command would be used

```
>> tboot( 2, 4 ) ;
```

To boot the a 1 by 8 array on site 2 the following command would be used

```
>> tboot( 2, 1, 8 ) ;
```

More complicated booting schema might require user-defined links to be configured and different operating system processes. These can be accommodated using a boot schema and configuration file as the arguments to tboot, e.g.

```
>> tboot( 'my_conf', 'my_schema') .
```

The file **my_conf** contains information about processor numbering and connections described in

NaIL (Node and Interconnect Language) [2]. To use different operating system options and processes the user must define a boot schema file, **my_schema**, specifying the GENESYS processes required. If only one file name is given as an argument to tboot, it is assumed to be the system configuration. A -v option is provided that reports the progress of the boot process. Warnings are displayed if tboot is unable to boot the required network or other errors are encountered.

6.1.3 Resetting the Gateway: reset_tran

The function reset_tran is used to reset the transputer platform to its default configuration. This involves killing all processes on all nodes, resetting software configurable links, disconnecting the operating system from the host machine and removing lock files from the /tmp directory. The function reset_tran is implemented as a call to a shell script containing the GENESYS commands necessary to reset the platform. Reset_tran takes no arguments and only reports warning messages in the event of an unsuccessful operation.

6.1.4 Examining processes and buffers: tstate, tbstate

The functions tstate and tbstate implement calls to the GENESYS commands state and bfstate respectively. Tstate is used to get information about the status of processes in the users configuration. Tbstate is used to report on the status of the message passing buffers. These commands can be used inside m-files or from the command line to assist in debugging parallel routines or to monitor process activity. The commands tstate and tbstate takes one input argument of a text string defining the numbers of the nodes that the user is interested in, for example

```
>> tstate( '0-5' )
```

or

```
>> bfstate( 'h' )
```

The first example reports on the status of processes on processors 0 through to 5 in the current booted system. The second example would show what messages were queued in the host machine's buffer.

6.2 Gateway Utilities

The parallel processing gateway is made up of four commands - **loadtran**, **exectran**, **unloadtran** and **killtran** - and a transputer gateway server. Fig 3. shows a context diagram for the whole gateway split in host (MATLAB) and remote (running on the parallel computer) processes. The transputer gateway server is composed of two parts, the host gateway and the Transputer gateway, and the utility commands operate by making calls on or through the gateway as a whole.

6.2.1 Loading processes: loadtran

The command loadtran is used to place processes on processors without actually making a call to that processes. Using loadtran, concurrent and parallel routines can be placed on processors in the

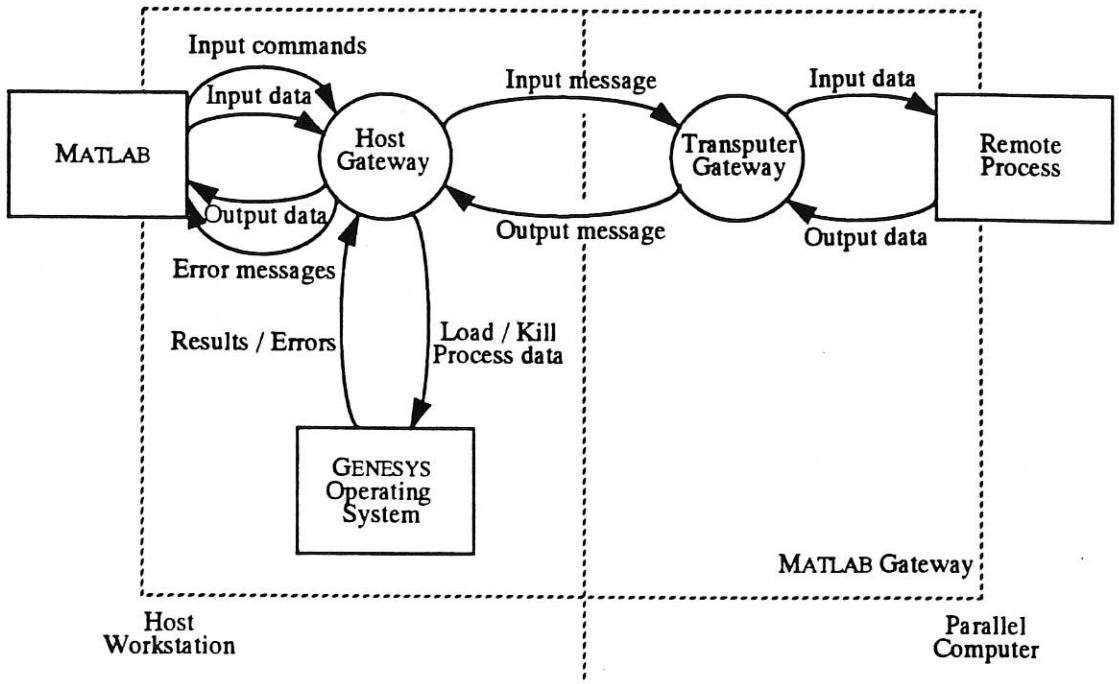

Fig. 3: Parallel Processing Gateway Context Diagram

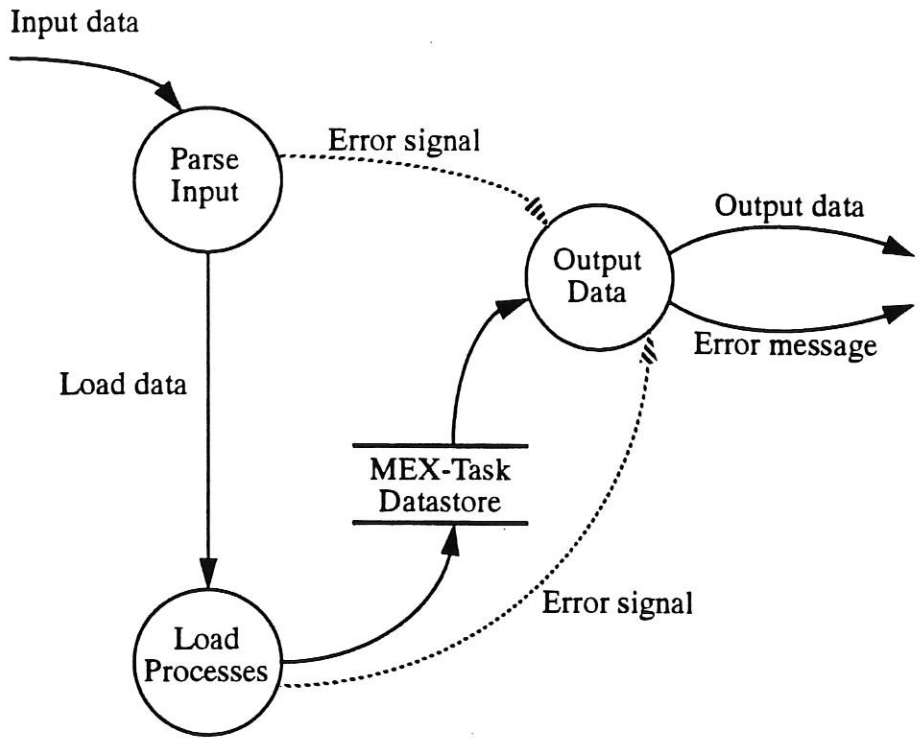

Fig. 4: Dataflow Diagram for Loadtran

network before they are used from MATLAB, and after the parallel platform has been configured and booted using the utility commands. Examples of the use of loadtran are:

```
[Proc_Map, Errors] = loadtran( 'Filename' ) ;
[Proc_Map, Errors] = loadtran( '-nNode Filename' ) ;
[Proc_Map, Errors] = loadtran( '-N Filename', Nodes ) ;
[Proc_Map, Errors] = loadtran( 'Filename1', 'Filename2', 'Filename3', ... ),
```

where **Filename** is the name of the an executable file, **Node** is the index number of a node in the transputer network, and **Nodes** is a column vector containing a list of transputer index numbers. The output parameters are **Proc_map** and **Errors**. Proc_map is a matrix having one row for each successfully loaded process and 3 columns containing the node number, index number and process identifier respectively. Errors is a 2 by 1 matrix containing the number of successfully loaded processes and the number of errors in loading processes used for simple error checking.

In the first example, a single function is loaded onto the first available compute node, usually node 0. The second example demonstrates how a file can be loaded to a particular node or nodes using the -n option. Here Node may be a single number, a list of nodes separated by commas (e.g. 1,2,5,6) or a range of nodes (e.g. 0-6 or 0-3,7-9). The third example demonstrates the -N option for passing the destination node numbers in column vector. Finally, the last example shows how more than one process may be loaded at a single invocation of loadtran.

Fig. 4 shows the data flow diagram for loadtran. On invocation, the input data from MATLAB is received by Parse_Input. The information contained in the input arguments is extracted by Parse_Input, sorted and passed on to Load_Processes in the event of no error(s) being detected. If an error(s) is detected this is flagged, and Output_Data reports back to MATLAB. Load_Processes takes the load data table from Parse_Input and attempts to load the specified processes onto processors. For every successful load, a new record is attached to MEX-Task Datastore containing the node number, process id and an index number. If Load_Processes detects an error, then it reports this and terminates. Output_data is used to report error message back to MATLAB and return information from the MEX-Task Datastore to the workspace for process management and calling.

6.2.2 Calling parallel routines: exectran

The command exectran is used to call one or more routines running on one or more compute nodes on the parallel computer. Exectran passes invocation arguments to routines already resident on nodes on the parallel platform, provides any communication required between these routines and MATLAB, and returns results to the MATLAB workspace. In the event of a routine not having been placed on a compute node, exectran will load the desired code from the file system and perform the appropriate function calls.

The command line for calling exectran can take the following forms:

```
1) [A, B, C, ...] = exectran( 'Filename', D, E, F, ... ) ;
2) [A, B, C, ...] = exectran( '-nNode Filename', D, E, F, ... ) ;
```

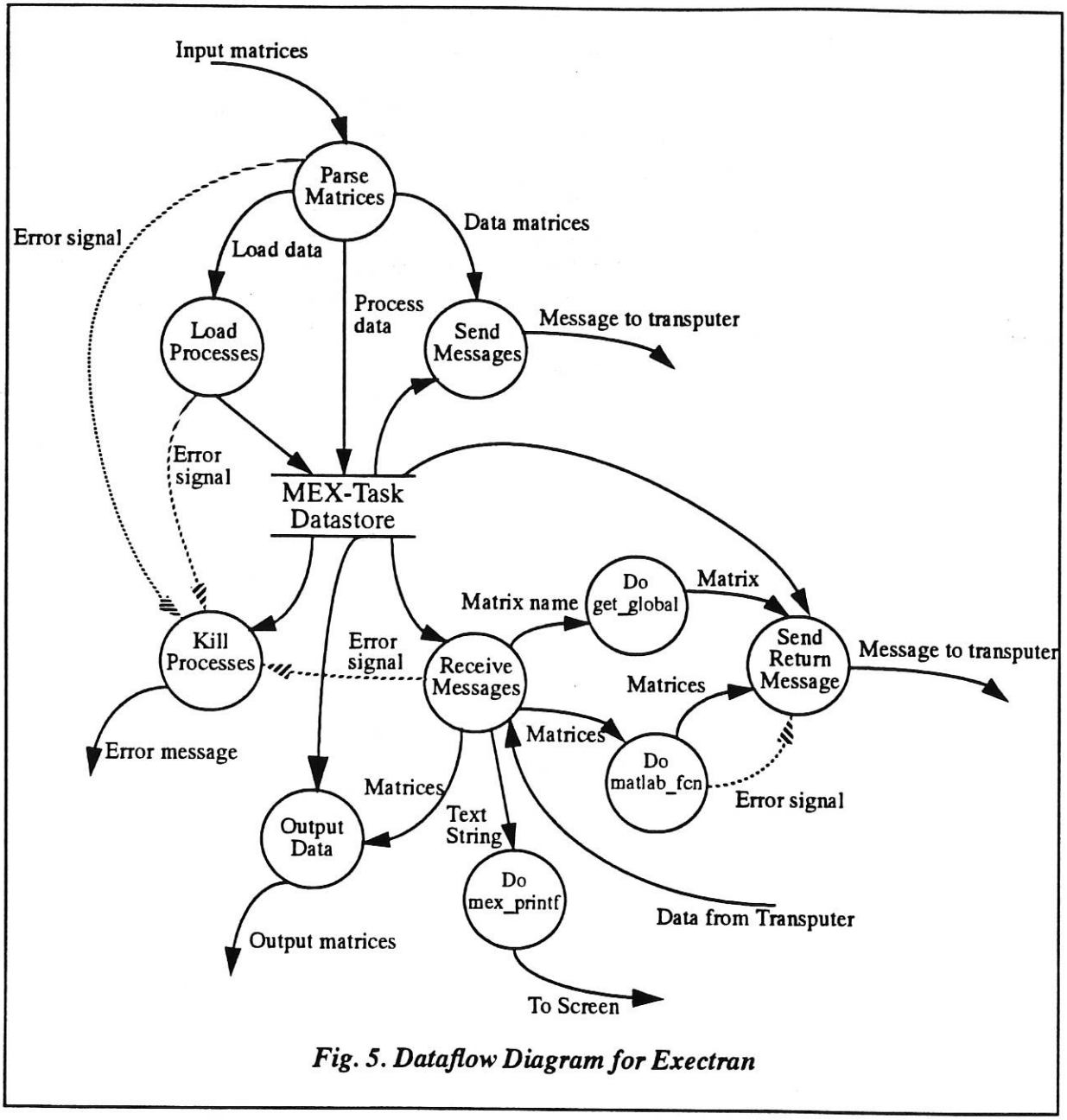

Fig. 5. Dataflow Diagram for Exectran

3) [A, B, C, ...] = exectran('-N Filename', Nodes, D, E, F, ...) ;
4) [A, B, C, ...] = exectran('-T', Proc_Map, D, E, F, ...) ;
5) [A, B, C, ...] = exectran('-r Result Filename', D, E, F, ...) ;
6) [A, B, C, ...] = exectran('-R Filename', Returns, D, E, F, ...) ;

where **Filename** is the name of the executable code, **Node** is the index number(s) of the required compute node(s) and **Nodes** is a column matrix containing the index of compute nodes to be used. Proc_Map is the process map generated by the command load_tran described in the previous section. The variable **Result** is the number of expected output matrices from a routine and **Results** is a column matrix containing the number of return matrices expected from each routine called.

Matrices D, E, F, ... are input parameters to the functions called by exectran. Matrices A, B, C, ... are the data structures in which results from the remote function calls will be placed.

Example (1) demonstrates the basic operation of exectran. The first rhs parameter is a text string that determines the mode of operation of the gateway, (in this case the name of an executable file), and is referred to as the command matrix. The executable code is loaded from the filesystem onto the first available compute node, and the subsequent parameters (D, E and F) passed as input arguments. On return, the results obtained from the process Filename are stored in A, B and C. If the gateway parser comes across another text string in a command line, this is assumed to be another command matrix. This allows calls to different routines with different input and output parameters to be made in the same function call. For example, to load and run two routines, P1 and P2, on different nodes with inputs A and B passed to P1 and C and D passed to P2 and each returning one result in X and Y respectively, the following command line would be used:

[X, Y] = exectran('P1', A, B, 'P2', C, D).

Text strings may be passed as input arguments to remote routines by prefixing them with a dollar symbol, "$". If an error occurs during exectran's run time, a warning message is displayed in the MATLAB window and all routines loaded by that exectran call are terminated. Error handling by remote routines is determined by that routine.

Examples (2) and (3) have a similar format as that employed in loadtran (Section 6.2.1). The '-n' and '-N' options being used to explicitly place routines on processors. Example (4) uses the '-T' option to indicate that the required routines are loaded and only input/output parameters need be handled by exectran. The '-T' option should not be used in conjunction with '-n' or '-N' options.

Examples (5) and (6) show the use of the '-r' and '-R' options for instructing the gateway of the required number of result matrices from each remote routine. When '-R' is used in conjunction with '-N', '-n' or '-T', the Results matrix should come after the parameters associated with these options, but before the actual input parameters to the remote routine.

The dataflow diagram for exectran is shown in Fig 5. The input data contained in the command line arguments is first parsed to determine what processes, if any, require loading onto compute nodes and what data matrices should be sent to which nodes. If any errors are detected whilst parsing the input data, an error signal is generated and a terminate message sent to any processes that have been loaded already in this call to exectran. After any unloaded processes have been loaded, and the appropriate data messages have be sent to compute nodes, exectran blocks awaiting messages from the called routines. These messages may be either results or requests for further actions, or calls to mex-file interface routines. Incoming messages containing results are stripped and the data contained in them written into a Matrix structure corresponding to the left hand side arguments in the exectran call. Messages requiring calls to MATLAB are serviced as and when they arrive in sequential order and the results of the MATLAB calls are returned to the calling process. On termination of exectran, processes that have been loaded are not killed but remain loaded for future calls. The utilities unloadtran and killtran may be used to remove processes from processors when they are no longer required.

6.2.3 Removing processes: unloadtran

Unwanted processes may be removed from processors by the **unloadtran** command. Examples of its usage are:

```
>> unloadtran( [2, 14, 57698] ) ;
>> unloadtran( Proc_Map ) ;
```

Both examples take one argument, an $n \times 3$ matrix containing in each row the processor number, process index number and process id. In the first example, the process with id 57698 and index 14 on node 2 is terminated and removed from memory. The index and process id are returned from the loading routine, loadtran, or obtained from tstate (see Section 6.1.4). The second example unloads all the processes contained in the process map **Proc_Map**.

Processes are removed from processors by sending the "terminate" message to the particular process' transputer gateway server. This causes a function to be invoked that clears all memory used and halts the programs after sending a status message back to unloadtran.

6.2.4 Killing processes: killtran

Under certain circumstances it may not be possible to terminate a process using unloadtran. This may occur when an error occurs in the main body of the routine and control can not be regained by the transputer gateway server. The routine **killtran** is called with the same parameters as unloadtran, i.e.,

```
>> killtran( [2, 14, 57698] ) ;
>> killtran( Proc_Map ) ;
```

and has the same effect as unloadtran except that it can not guarantee to reclaim all the memory that the killed process may have used. Killtran is implemented as a direct call to the GENESYS utility **doom**.

6.2.5 Transputer Gateway Server

The transputer gateway server is linked to the function code by tcc_mex at compile time (see Section 6.1.1). It is effectively a shell to provide the transputer run-time environment for C mex-files and library of routines for shadowing the MATLAB mex-file interface functions. A dataflow diagram for the gateway server is shown in Fig.6.

When a message is received from the host gateway, it will either contain data matrices for the remote function call or a message telling the process to terminate. In the case of the latter, the gateway server clears all memory allocated by the process, sends a message back to the host to report on the outcome and removes itself from memory. Incoming data messages are stripped and the corresponding Matrix data-structures constructed and used in a call to the function entry point "user_fcn" (see Section 3).

During execution, the process may make a call to "matlab_fcn", "get_global" or "mex_printf".

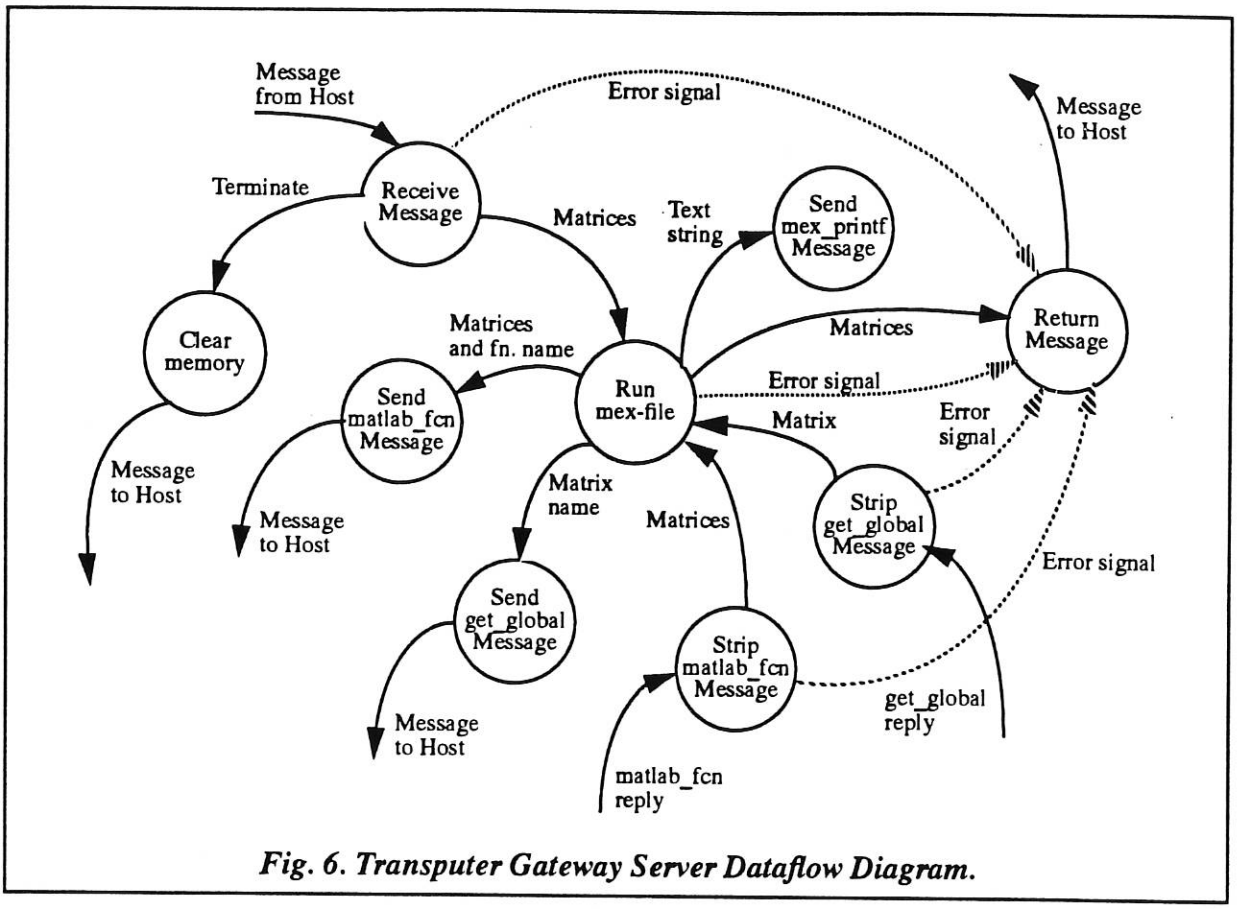

Fig. 6. Transputer Gateway Server Dataflow Diagram.

The parameters of these calls are extracted by the transputer gateway and placed in a message structure to be sent to the host gateway. The process will then block awaiting a response from MATLAB. When the return message is received, the reverse procedure is applied, stripping data out of the message and returning it to the process' workspace. On completion, user_fcn returns its results to the transputer gateway server which then passes these parameters back to the host gateway and eventually to the MATLAB workspace.

7. C Programming library

As the parallel platform can only be programmed in C, Fortran and Occam, the high level, descriptive approach of MATLAB is not available to the user. In order to ease the construction of code for routines running on the transputer nodes, a library of core routines, written in C, drawn mostly from the basic MATLAB functions, LINPACK [4] and EISPACK [5] and the Control Systems Toolbox [6], has been constructed. These routines are functionally equivalent to their MATLAB counterparts, where they exist, and by use of conditional compilations can be used in mex-files, directly on the transputer platform or as part of larger programs. Where C library counterparts of MATLAB commands do not exist, it is possible to call MATLAB commands from processes on the parallel platform using the standard mex-file commands.

Appendix A contains a list of the C programming library functions. These routines use the same data structure as MATLAB and as such operate on and return a variable size, possibly complex,

matrix.

8. Timing of Primitive Gateway Operations

In this section the performance of the primitive operations of the gateway, such as passing data messages to and from processes on transputer nodes and calling MATLAB functions from transputer processes, are considered. These timings are considered in relation to the "hard-wired" MATLAB interface described in Section 4. Because the parallel platform is accessed over the Ethernet, exact communications times cannot be obtained. Network traffic, machine loading and other factors will influence the time taken to communicate between the host machine and the parallel platform. Thus, timings presented in this and future Sections are typical times, averaged over a number of samples.

Here a host node is considered to be one in which the gateway communicates directly with the transputer and a remote node is one in which the communications to and from it are managed by a host node. As such, there may be more than one host node active at any one time and any processor may be a host node to more than one process where a process is a routine called from MATLAB and running on the parallel platform.

8.1 Attaching to the GENESYS OS

Before communications can be established between a MATLAB process and a process running on the parallel platform, MATLAB must attach itself to the GENESYS OS using *kinit* [2]. If a processes no longer requires to communicate with the parallel platform it can be detached from the OS using the *kdetach* routine. GENESYS imposes a limit of 32 times that a process can be attached and detached from the OS.

Table 3. shows the time taken for a process to attach and then detach from GENESYS. The timings are an average over 20 separate executions of a mex-file that only attaches and detaches from the OS. Once a process has been attached to GENESYS, subsequent attachment is considerably less than the original time. This is because the functions entry point is already inside the MATLAB address space.

Table 2: Time taken for a process to attach/detach from GENESYS OS

	Time (ms)
SUN IPX	75
SUN 4/110	140

8.2 Passing messages to the host node

The most basic gateway operation is to pass a message to a previously loaded process and receive a message from that process after execution of its instructions. The C code for the routine test1.c, shown in Fig. 7, is a C mex-file that copies an input matrix to the output matrix.

```
/* test1.c - copy real input matrix to a real output matrix */
#include "cmex.h"
user_fcn( nlhs, plhs, nrhs, prhs )
int        nlhs, nrhs;
Matrix     *plhs[], *prhs[];
{
        Matrix *lhs;
        double *pi, *po ;
        int    i, m, n ;
        /* get dimensions and pointer to real part of input matrix*/
        pi = prhs[0]->pr;
        m = prhs[0]->m ; n = prhs[0]->n ;
        /* assign memory in Matlab workspace */
        lhs = create_matrix( m, n, REAL );
        po = lhs->pr;
        m = m * n ;
        /* copy input matrix to output matrix */
        for( i = 0 ; i < m ; ++i )
                *po++ = *pi++ ;
        /* put output matrix into Matlab address space */
        plhs[0] = lhs;
}
```

Fig. 7: C code for test1.c

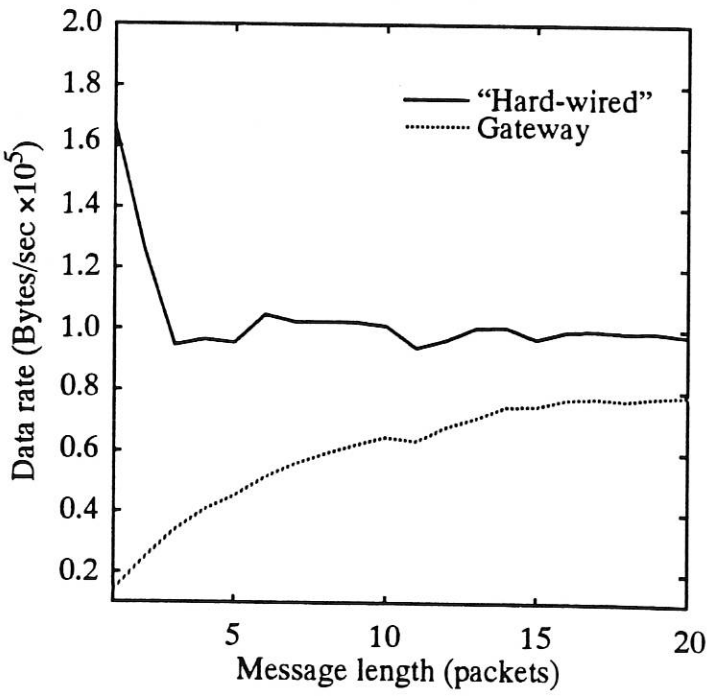

Fig. 8: Communications rates between MATLAB and a host node for varying message sizes.

To compile the code for use with the gateway, the following command is used

tcc_mex test1.c .

From the MATLAB command line, the following commands are used to load the executable code onto a compute node and then call the routine:

```
>> tboot(0,1,1) ;
>> [PM E] = loadtran('test1') ;
>> B = exectran('-T', PM, A ) ;
```

Fig. 8 shows the data communication rate for the matrix copying routine run using the gateway and in hard-wired form for varying message sizes.

In the GENESYS OS, the data packet size is fixed a 4096 bytes or 512 doubles. Using the hard-wired routine, the data transmission rate falls rapidly with an increasing number of packets levelling off at approx. 100 KB/sec. This is due to messages larger than the packet size being broken up into a number of messages, each with a maximum size of data packet. The performance of the data transfer between the SUN and the parallel platform at first sight appears very limited given that the bandwidth of the VMEbus is rated at 40MB/sec. However, other research [7] has shown that the limiting factor in data transmission rates between SUNs and VMEbus Transputer platforms is the clock cycle time of the SUN's processor. As the source code for the VMEbus link driver is not supplied with the GENESYS operating system, enhancements in the data transmission rate, as suggested in [7], are not feasible.

Examining the performance of the gateway, for small messages the data transmission rate is significantly lower than that offered by the hard-wired interface. This is due to the extra overhead of the gateway routines in parsing the input/output commands and routing messages to the correct process/processor. As the size of the message is increased this overhead becomes less significant compared with that of the GENESYS OS and the data transmission rate approaches that of the hard-wired interface. The slight fluctuation in the curves are due to background UNIX processes.

8.3 Passing messages to a remote node via the host node

Fig. 9 shows the data communications rates achieved in passing a message between a host node and adjacent (nearest neighbor) node using the GENESYS network message passing function **nsend** [2]. When messages of greater than one packet size are sent, the GENESYS communications routines automatically break the message down into a number of smaller ones, introducing a number of messages into the network

For the smallest message of 1 packet, 4096 bytes, the data rate achieved is approx. 1.4 Mbytes/sec, compared to the maximum theoretical link speed of a Transputer of 1.8 Mbytes/sec. Assuming that the Transputer links are operating at their maximum data rate and that the receiving processes is ready, this gives a setup overhead for the data send operation of 1.2 msec. When messages larger than one packet are sent, the increased time taken to setup the send

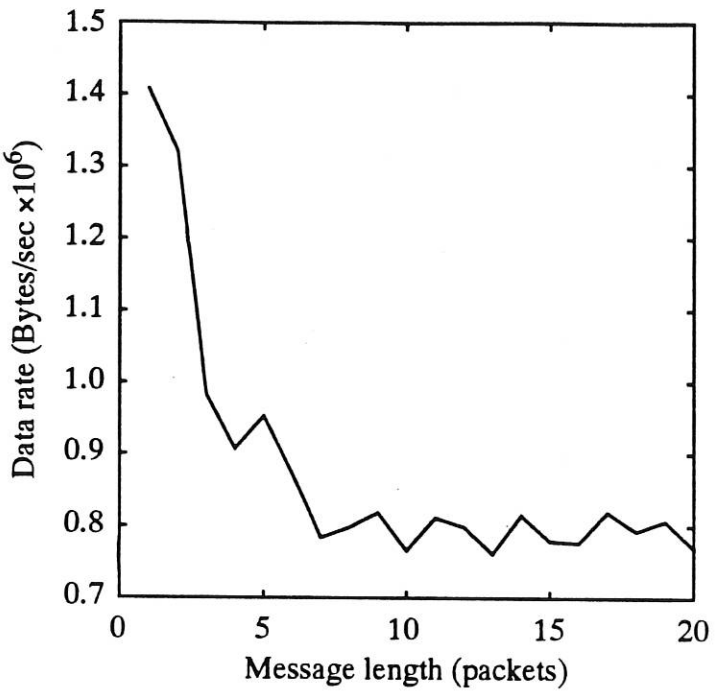

Fig. 9: Communications rates between a host node and an adjacent node for varying message sizes.

operation and the multiple receives and the recombination of the smaller data packets into the original message impacts on the data transmission rate significantly as shown in Fig 9.

8.3.1 Major Gateway Routines

This section gives timing information for the major gateway functions, loadtran, exectran and unloadtran. Timings have been taken on both the machine hosting the parallel platform, a SUN 4/110, and from a SUN SparcStation connected to the transputer platform by Ethernet. The mex-file used to obtain these timings only copies the pointer from an input matrix to the output matrix, thus the timings show the overhead associated with the gateway functions. In all cases the times are averaged over 10 calls at a time when network traffic is low minimizing variation in results. The input argument is a 10 x10 matrix of random real numbers. These results are summarised in Table 3.

From these results, it can be seen that the gateway imposes a significant overhead in accessing the parallel platform. However, these are worst case results, when the function is called for the first time. On subsequent calls, the overhead is significantly reduced as the function has been dynamically linked into the MATLAB address space.

Table 3: Execution times for major gateway functions

		Time (sec)		
		1 process	4 processes	8 processes
SUN IPX	loadtran	1.40	5.15	9.90
	exectran	0.55	0.70	0.80
	unloadtran	0.10	0.25	0.40
SUN 4/110	loadtran	1.65	6.20	12.20
	exectran	0.60	0.95	1.45
	unloadtran	0.25	0.60	1.00

9. Concluding Remarks

We have constructed a reliable gateway between MATLAB and a parallel processing platform. Also, a number of utility routines for easing the task of using a parallel platform from within MATLAB have been implemented. From the timing results, it appears that the gateway imposes a substantial overhead compared to using direct-wired routines. However, a number of factors influence the performance of the gateway. The relatively slow communications time between the host workstation and the transputer platform appears to be related to the host machine's internal clock cycle and bus usage. This may be overcome in newer interface technologies such as the SBus and SCSIbus link adapters employed on newer parallel computers. The effects of network loading also play an important role in the gateway's performance.

A demonstration of the parallel processing gateway and hard-wired interface routines is given in ACSE Research Report No. 487 [3] and an accompanying script-file, ppdemo.m.

10. Acknowledgments

The authors gratefully acknowledge the support of this research by the United Kingdom SERC grant on "Parallel Computing in CACSD" (GR/F94064) and the ESPRIT Parallel Computing Action (project No. PCA4093).

11. References

[1] C. Moler, S. Bangert and J. Little *"Pro-MATLAB User's Guide"*, The Mathworks, South Natick, MA 01760, 1990.

[2] Genesys C Reference Manual, Transtech Technology, High Wycombe, UK. 1990.

[3] A. J. Chipperfield, P. J. Fleming, *"Demonstration of the MATLAB Parallel Processing*

Gateway", Research Report No. 487, Department of Automatic Control and Systems Engineering, University of Sheffield, UK, 1993.

[4] J.J. Dongarra, C.B. Moler, J.R. Bunch and G.W. Stewart, *LINPACK Users' Guide*, Society of Industrial and Applied Mathematics, Philadelphia, 1979.

[5] B.T. Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikebe, V.C. Klema and C.B. Moler, *Matrix Eigensystem Routines - EISPACK Guide*, Lecture Notes in Computer Science, Volume 6, 2nd ed., Springer-Verlag, 1976.

[6] A. Grace, A.J. Laub, J.N. Little, C. Thompson, *Control System Toolbox User's Guide*, The Mathworks, South Natick, MA 01760, 1990.

[7] M. S. Atkins and Y. Chen, "Performance of SUN-Transputer Interlaces: some surprises", *Transputing '91*, Vol. 1, pp124-138, 1991.

Appendix A

C Programming Library

Table A.1: General Routines

function	description
Convert_in	convert a matrix from MATLAB to C
Convert_out	convert a matrix from C to MATLAB
Create	allocate storage for a new matrix
Destroy	de-allocate storage space
Display	display a matrix on screen
Add	addition
Sub	subtraction
Mult	multiplication
Inverse	matrix inversion
L(R)Div	left (right) division
Pow	power
Transpose	transpose and conjugate transpose
Identity	identity matrix
Constant	constant value matrix

Table A.2: Elementary Matrix Functions

function	description
Expm	matrix exponential
Logm	matrix logarithm
Sqrtm	matrix square root
<Funcm>	arbitrary matrix function
Poly	characteristic polynomial
Det	determinant
Trace	trace
Kron	Kronecker tensor product

Table A.3: Decomposition and Factorization

function	description
Balance	produce balanced form
Backsub	back substitution
Eigs	eigenvalues and eigenvectors
Hess	Hessenberg form
LUdecomp	LU decomposition
Null	null space
Orthog	orthogonalisation
QR	orthogonal triangular-decomposition
Schur	Schur decomposition
Svd	singular value decomposition

Table A.4: Control system functions

Function	Description
Ltitr	Linear time-invariant time response
Ltifr	Linear time-invariant frequency response
Lyap	Lyapunov equation solver
Lqr	Linear quadratic regulator design
Bode	Bode style frequency response
Step	Unit step response
Margin	gain and phase margins
Rlocus	Evans root-locus
Are	algebraic Riccati equation solver
Impulse	Unit impulse response
C2d (D2c)	Continuous (discrete) to discrete (continuous) -time conversion
Tf2ss (Ss2tf)	Transfer function (state-space) to state-space (transfer function) conversion