Akademie věd české republiky
ÚSTAV TEORIE INFORMACE A AUTOMATIZACE

Academy of Sciences of the Czech Republic
INSTITUTE OF INFORMATION THEORY AND AUTOMATION

RESEARCH REPORT

J.Kadlec

DIRECT SOFTWARE BRIDGE MATLAB–TRANSPUTER BOARDS

No. 1819 November 1994

ÚTIA AV ČR P. O. Box 18, 182 08 Prague, Czech Republic
Telex: 122018 atom c, Fax: (+42)(2)6641 4903
e-mail: utia@utia.cas.cz

This report constitutes an unrefereed manuscript which is intended to be submitted for publication. Any opinions and conclusions expressed in this report are those of the author(s) and do not necessarily represent the views of the Institute.

This work has been partially supported by the research grant No. 102/93/0897 of the Grant Agency of the Czech Republic: "Systolic Arrays of Processors for Identification of Systems with Variable Structure" and by the SERC Grant "Algorithm Engineering for Concurrent Control" (UK). J. Kadlec is with Control Engineering Research Centre, Department of Electrical and Electronic Engineering, The Queen's University of Belfast, Ashby Building, Stranmillis Road, Belfast BT9 5AH, Northern Ireland, Email: j.kadlec@ee.qub.ac.uk

DIRECT SOFTWARE BRIDGE MATLAB–TRANSPUTER BOARDS

J. KADLEC

ABSTRACT. Presented software toolbox “PARAbridge” connects the new PC MS Windows 3.1 based MATLAB 4 with 3L Parallel C on Inmos B004 compatible transputer boards. This combination enables the running of data-parallel transputer applications from within MATLAB for the verification and testing of systolic array algorithms. The MATLAB user can program the transputer application, which may use MATLAB as a programmable high level graphical interface. This can significantly reduce the development time of transputer based applications.
KEYWORDS. Systolic array, normalization, fixed point, probability.

1. Introduction

In the last decade, MATLAB (MathWorks Inc.) [1] has reached a dominant position in CAD of signal processing algorithm simulations. The current version 4, working under Windows, offers not only the outstanding numerical computing techniques but also a sophisticated interface to graphical presentation.

MATLAB is an open shell system, providing means for coding interfaces to other software in the form of (dynamically linked) MEX - functions in C language or FORTRAN. The design of MEX functions is straightforward and the resulting software enjoys broad range of system services including graphic support.

With growing complexity of applications, a need arises to implement a part of software on parallel (transputer based) computing systems [2]. Up to now, such implementations have lacked of adequate formalization, technical and computer tools support.

This paper describes the PARAbridge toolbox for MATLAB, which represents a significant step toward the solution of this problem.

The PARAbridge [3], [4] operates as a MATLAB-like (reduced) interpreter operating in the root transputer workspace. It offers the ability of running MATLAB MEX functions on transputer. The toolbox supports communication between MATLAB and the transputer matrix environments using support functions with syntax consistent with the MATLAB philosophy.

The transputer board can be used as a parallel numerical accelerator programmable from MATLAB. Alternatively the transputer system can be viewed as the master processor with MATLAB acting as a graphics or computation "engine" and server for

transputer data initialization and presentation of results.

2. Elementary Operations of PARAbridge

Using the toolbox [3], the user operates with two related matrix environments: The MATLAB shell and the PARAbridge server running on the root transputer. Both environments communicate using a set of service functions compiled as MATLAB MEX (.dll) functions. This section provides a list of supported elementary operations [4].

2.1 Matrix Operations Support

2.1.1 Copy PC Matrices to Transputer

This operation is solved by the functions `boot` and `boottp`. The first sends permanent (named) MATLAB matrices to the transputer, e.g.: `boot(A,B,C)` copies the matrices `A, B` and `C` to the transputer under the same name. The second service allows the copy of results of expressions to the transputer, e.g.: `boottp(A*B,'AB',i,'i')` sends the product to transputer matrix `AB` and the imaginary unit `i` to the transputer under the same name.

2.1.2 Copy Transputer Matrices to PC

The function `bootpc` carries out this operation, e.g.: `[A, B] = bootpc('TPA', 'TPB')` creates MATLAB matrices `A` and `B` as copies of the transputer matrices `TPA` and `TPB`.

2.1.3 Allocate Transputer Matrices

This task is solved by the function `alloctp`, e.g.: `alloctp('TPM', 10, 20)` creates the transputer 10-by-20 matrix `TMP`.

2.1.4 Display and Clear of Transputer Matrices

The functions `whotp`, `whostp` and `cleartp` are functionally similar to the `who`, `whos` and `clear` MATLAB functions.

2.1.5 Parallel Evaluation of String Commands

The transputer PARAbridge server not only maintains the matrix environment, but also allows interpretation of command lines. The function `evalpar` and `evalwait` are available for this purpose. The function `evalwait` suspends MATLAB processing while the string is executed on the transputer. The function `evalpar` disconnects the PC and transputer and lets MATLAB continue. Both processors continue in parallel up to a point of synchronization which is implemented by the service function `waittp`.

2.2 Example of Parallel Interpreter Use

Let us demonstrate the elementary PARAbridge functions using a simple example. The function `sum` evaluates the sum of matrix columns. The standard solution would be for MATLAB to call the MEX code by:

```
≫A = magic(3)
A =
8     1     6
3     5     7
4     9     2
≫eval('U=sum(A)');
U =
15    15    15
```

The same problem can be solved by the transputer MEX function by:

```
≫boot(A)
≫evalpar('U=sum(A)')
≫U=bootpc('U')
U =
15    15    15
```

The repetitive interpretation of commands is both inconvenient and time consuming. Therefore the PARAbridge toolbox introduces a concept of remote programming of the root transputer.

2.3 Remote Evaluation of Program on Transputers

Programs consist of sequentially numbered lines which are introduced and recorded on the transputer by the function `pgmline` (and listed by `pgmlist`). Let us for example introduce the following program:

```
≫pgmline('U=sum(A)');
≫pgmline('V=sum(U)')
≫pgmlist
Transputer program
1     U=sum(A)
2     V=sum(U)
```

The program is carried out by the functions `callpar` and `callwait` that differ in the time schedule as explained above in the context of string commands, e.g.:

```
≫callwait, V = bootpc('V')
V =
45
```

2.3.1 Remote Programming Adds Flexibility

Note, that the remote programming of the transputer shell adds flexibility and simplifies the testing of alternative versions of algorithms running in the parallel environment of the transputer MEX functions.

2.3.2 Reduction of PC-transputer Data Exchange

The example shows how the PARAbridge transputer server allocates and handles matrices in the local workspace of the root transputer (the matrix `U`). This can reduce the slow PC-Transputer data exchange and therefore considerably reduce the computation times, especially if loops of multiple calls are employed.

2.4 Call of Transputer MEX Functions

Up to this point the interpretation of commands with parameters as matrices located in the transputer workspace only has been considered. However, the most frequent task is to call a transputer MEX function with the input and output arguments defined as matrices in the MATLAB workspace.

This problem may be solved by naming the matrices, booting them to the transputer, evaluating the relevant string, booting back the outputs and clearing the transputer matrices. This sequence is automated by the service functions `callwait` and `callpar`. The use of `callwait` is simple, e.g.:

```
≫U = callwait('sum', magic(3))
U =
15    15    15
```

3. Transputer MEX Function Design

MATLAB 4 has established a new standard for accsses to matrices. Matrices are manipulated by a library of functions used by users programming in MEX code for MATLAB. The routines are grouped into three classes distinguished by prefixes: `mx` for access to matrices in the MATLAB workspace; `mex` for access to the interpreted `.m` or compiled `.dll` MATLAB functions; `mat` for access to Matlab data files in `.MAT` format.

The PARAbridge toolbox provides the equivalent set of C routines for the code running on the root transputer MEX (with the exception of MAT-file routines). Therefore, no source code changes are required when moving C source code of MEX functions (written originally for MATLAB) to transputer for most applications. These PARAbridge equivalents of the MATLAB 4 support functions provide access to the matrices and functions managed by the PARAbridge server in the memory workspace of the root transputer.

3.1 Access from Transputer to MATLAB

In addition, the PARAbridge toolbox [3], [4] provides a new class of functions for users programming transputer MEX functions in 3L Parallel C language [2]. These functions carry the prefix `par`.

The service functions enable C code on the root transputer to copy matrices from/to the MATLAB workspace. It is also possible to use the MATLAB shell for a remote evaluation of a string commands and to call interpreted `.m` or compiled `.dll` MEX functions. Here is the a brief list of the functions documented in [4]:

`parGetMatrix` gets a Matrix from MATLAB.

`parPutMatrix` puts a Matrix to MATLAB.

`parCreateFull` creates a full Matrix in MATLAB workspace.

`parMatrixFree` frees a Matrix from MATLAB workspace.

`parEvalString` lets MATLAB evaluate a string. The eventual inputs and output matrices are residing in the MATLAB workspace.

`parCallShell` calls a MATLAB function (.m or MEX). The input and output parameters are matrices residing in the root transputer workspace.

`parWait` waits for results from the MATLAB function to synchronize processing.

The use of the callbacks `parEvalString` and `parCallShell` is similar to the use of MATLAB as a computation "engine" from MATLAB MEX code via the `engEvalString` and `mexCallMATLAB` functions introduced in Matlab 4 [1].

The mechanism by which PARAbridge callbacks from the root transputer to MATLAB is described schematically in **Figure 1.** This callback mechanism enables to use Matlab as a high level graphical engine with the Windows type of dialogues, buttons, sliders and other widgets.

Notice, that callback functions can be coded as interpreted `.m` MATLAB files. This reduces development time in the design of the complex communication panels for the transputer based real time and parallel applications.

4 Parallel MEX Programming for MATLAB

The concept of the parallel transputer MEX functions based on the Master-Workers model is described in this section.

The computation of a Mandelbrot set benchmark [2] has been selected as a representative example of a data-parallel implementation of an algorithm operating on a MATLAB matrices.

4.1 Mandelbrot Set Benchmark Example

The benchmark presented in this paper computes a double precision matrix with a (50,50) dimension. The elements of the matrix are computed by a numeric iteration, dependent on the coordinates of the matrix element and additional parameters identical for all matrix elements. The computation of all elements is independent, enabling the processing to be distributed across a network of processors. See **Figure 1.**

The user is required to write sequential C code, specifying the computation for all element of the matrix or for blocks of elements. In this example it will be a block of (5,5) elements. Therefore the complete (50,50) matrix task is divided into 100 jobs running on worker transputers.

The automatic flood-fill configurer and loader provided by 3L Parallel C [2] automatically tests the connections and number of processors in the network and loads it with the worker code. The parallel MEX function running on the root transputer is responsible for collecting the parameters and returning the computed matrix to the MATLAB workspace. There is a predefined template of parallel processes - threads - started by the parallel MEX function running on the root transputer. See [2] for details. These processes distribute parameters - the (5,5) sections of the final matrix - to workers and collect the results.

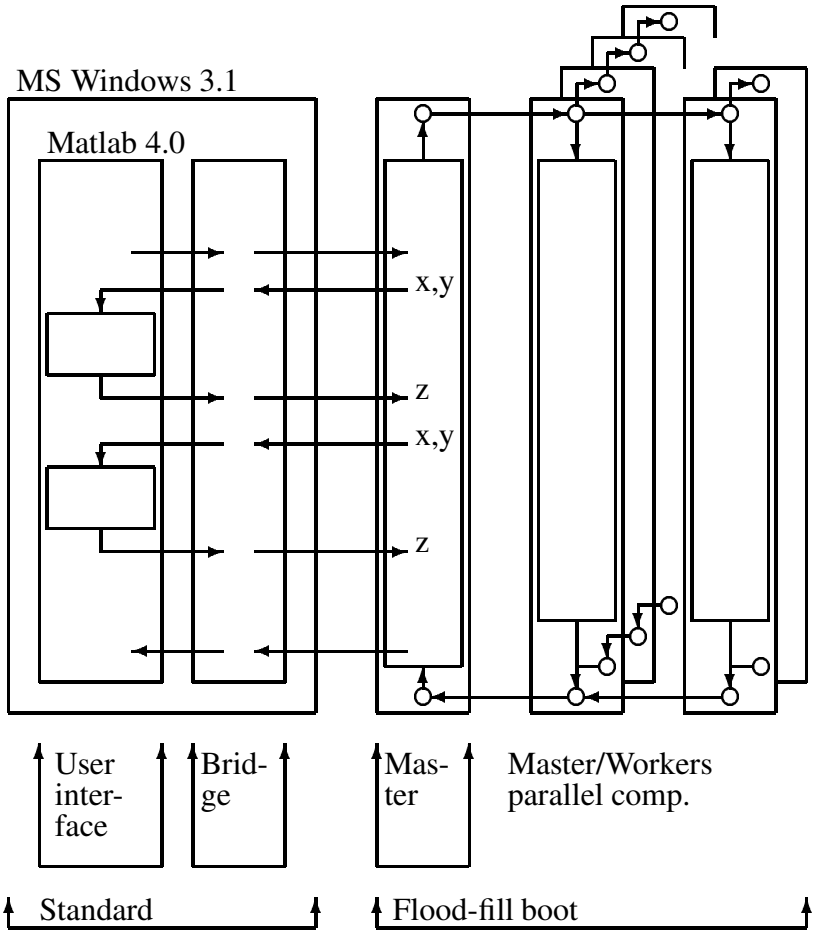

Figure 1. Basic structure of the parallel MEX application.

The designer of the parallel transputer MEX can concentrate on the body of the sequential code of the worker processors. Details of the high priority processes organising the communication on the network are supported by the network service routines of 3L Parallel C [2].

The parallel transputer MEX implementation of the Mandelbrot set benchmark is called from MATLAB by:

```
≫a=callwait('mandtp',p1,p2,..);
≫contour(a);
```

The PARAbridge service function `callwait` is used to call the transputer MEX function `mandtp`, with some parameters defining the requested dimension of the problem and the shape of the final result. The computation is supervised by the root transputer and performed on an arbitrary sized array of transputers. The result is returned into the MATLAB matrix `a`. The matrix is allocated in the MATLAB workspace by the

PARAbridge interface.

Figure 2. presents the resulting mandelbrot set plotted by MATLAB command `contour(a)`. It scans the elements of the matrix and displays a set of equidistant contours, presenting in 2D the content of the matrix.

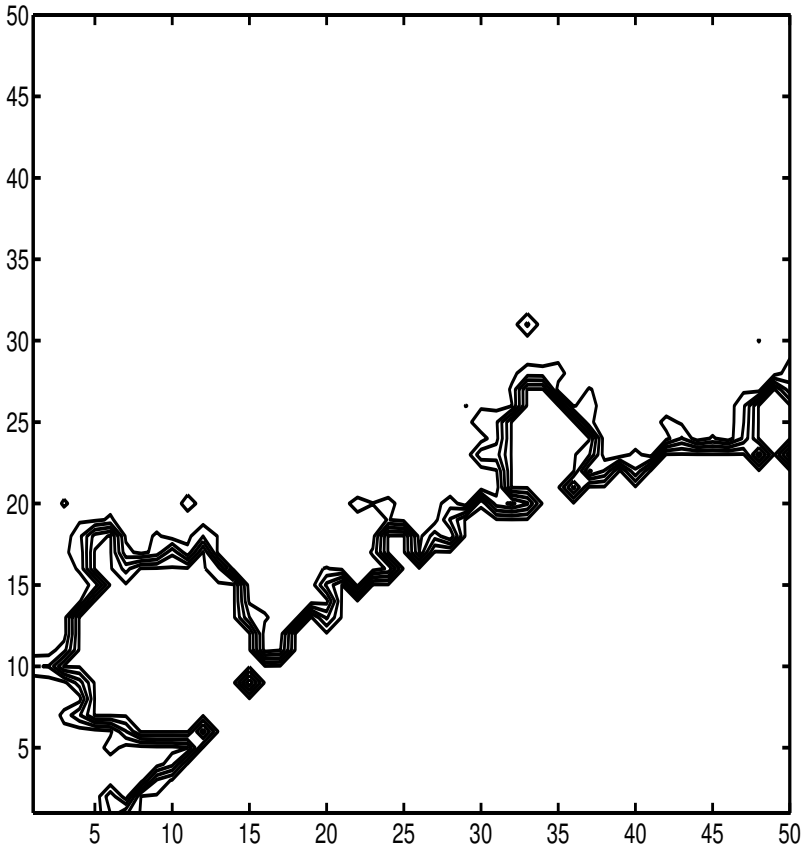

Figure 2. Mandelbrot set generated by the example.

4.2 Arbitrary Matrix Dimensions

Notice, that the concept of the parallel MEX implementation is principally independent

of the problem dimensions. This is in line with the syntax of the MATLAB functions. The problem dimensions are taken from the matrix arguments by default.

4.3 Flood-fill Configuration on Arbitrary Network

The real topology or dimension of the transputer array can be arbitrary or even unknown to the user. However, the code of the parallel transputer MEX remains identical because the binary code contains the automatic flood-fill booting loader, examining all worker transputers available at boot time. It automatically distributes the worker tasks across an arbitrary network and routes work packets from the master to the workers.

It is also possible to run the application on a single transputer. This will work automatically if the application is configured using flood configurer and the network will consist of the root transputer only. Alternatively, a static single-transputer configuration may be built by hand.

5. Speed -Up Comparison

The mandebrot set example has been programmed in a number of ways to give a representative picture of the computation times on different platforms. All implementations generate the matrix result identical to **Figure 2.**.

Table 1. presents the computation times. The MATLAB interpreted code (.m), is significantly slower in comparison with the MATLAB MEX implementations (.dll) compiled by the MS C/C++ 7.0 Optimizing Compiler. There are 3 versions of the .dll code for PC in **Table 1.**

1. complete information for debugging - this is the default setting distributed with MATLAB 4 for PC.

2. no debugging information and no code optimization.

3. maximal optimization of code for time. This involves internal redefinition of variables, changes to loops, tests for special operands like a multiplication by zero, to reduce the computation time.

These sequential implementations are compared with the parallel transputer MEX implementation on 1 transputer and on four transputers (1 master and 3 workers). The time measurement starts in MATLAB, before the call of the function `a=callwait('mandtp',p1,p2,..);` and stops after complete return of result into MATLAB. Therefore the measured time includes all of the hidden allocations and data transfers provided by the PARAbridge.

The computation times were measured on an IBM PC 386,87, 33MHz, 64K cache, 8M RAM, under DOS 5, Win 3.1. and MATLAB 4. The transputer platform was a B004 compatible TPM-PC board from Parsytec. The board has 4 T805 transputer modules, with 30MHz internal clock, and a standard (slow) 70ns dynamic 1 MByte RAM.

Finaly, **Table 1.** also includes the computation results measured for SUN SPARC IPX and HP 9000 720 workstations, running MATLAB 4.

The benchmark computes in all cases the mandelbrot set for a (50,50) matrix with all variables in double precision (8byte) floating point format. In all cases the job was divided into 100 sub-task of (5,5) sub matrices.

Method of coding	Time	Speed
PC Matlab 4 (.m)	721.6 s	0.18x
PC Matlab 4 (.dll) Default	129.5 s	1.0x
PC Matlab 4 (.dll) No-Debug	71.1 s	1.8x
PC Matlab 4 (.dll) Max. Opt.	9.94 s	13x
PC Matlab 4 PARAbridge: 1 T805	2.69 s	48x
PC Matlab 4 PARAbridge: 4 T805	1.1 s	117x
Workstations:		
SPARC IPX Matlab 4 (.m)	218.9 s	0.59x
SPARC IPX Matlab 4 (.mex)	1.48 s	87.5x
HP9000 720 Matlab 4 (.m)	119.4 s	1.08x
HP9000 720 Matlab 4 (.mex)	0.495 s	282x

Table 1. Comparison of computation times

6. Simulation of Systolic Arrays

The basic template of the data-parallel transputer MEX implementation presented above can be repeated on a broad range of signal processing algorithms operating on matrix-like arrays of cells.

The only requirement is that the problem should be formulated in a manner which enables the computation of cells in any order and independently. In the case of the parallel evaluation of systolic arrays this can be solved (for one systolic step) by the definition of separately stored matrices of cell inputs, internal states and outputs. A cell with the coordinates `[i,j]` will operate on the data stored in the matrices of inputs, internal states and outputs with the coordinates `[i,j]`. The index `[i,j]` can be used for selection of the function of the particular cell. These units (independent within one systolic step) can be computed in an arbitrary order by the farm of workers controlled by the master code.

The actual data dependencies and data flow of the simulated array can be realized back in MATLAB by assigning the matrices of cell outputs to matrices of the cell inputs for the next stage of parallel simulation.

The matrices used for the interface with the PARAbridge systolic array simulation contain the pipe-line time snap-shots of the systolic array in a form analogous to a real parallel implementation.

The main advantage is the independence on the simulated array dimension. The

flexibility of the MATLAB language can be used for connecting of the simulated array to data generated by the MATLAB toolboxes. Results of the simulation can be collected from the pipelined information propagated in the PARAbridge matrix interface to a separate set of standard MATLAB matrices. These results can be compared with the reference outputs from other sequential MATLAB toolboxes.

7. Conclusion

The paper describes a novel concept of the PARAbridge toolbox environment for parallel programming under MATLAB 4. The computation times measured for the Mandelbrot set benchmark demonstrate the current possibilities and limitations of the transputer hardware. The transputers bring considerable acceleration to the PC based MATLAB installation.

The algorithmic design oriented research simulations can utilise the presented toolbox for the verification of parallel algorithms for systolic arrays. Cells of a simulated array can be evaluated by a farm of worker processes. The farm is formed by an arbitrary network of transputers linked by the PARAbridge to MATLAB. This appears to provide a balanced trade-of between flexibility and the overheads related inevitably with the propagation of the work packets on the transputer network.

The PARAbridge concept of a parallel interpreter running on the root transputer can provide an alternative for the MATLAB user as well as those developing transputer applications.

Researchers running MATLAB simulations and those dealing with real time parallel processing can and should share their valuable results. Unfortunately these two worlds often find it difficult to bridge the ever widening gaps caused by rapid technology changes. The PARAbridge toolbox is an attempt to facilitate joint work. The project has been started at the Institute of the Information Theory and Automation of The Academy of Science of The Czech Republic, Prague. A PC version of the toolbox will be presented at the conference.

References

[1] **Matlab (1992):** Reference Guide, The MathWorks, Inc. Natick, Mass 01760.

[2] **Parallel C ver 2.2 (1991):** Reference Manual, 3L Edinburgh. UK.

[3] **Kadlec J., P. Nedoma (1994):** PARAbridge 1.0, MATLAB 4 interface to 3L Parallel C for transputers, User Guide. Institute of the Information Theory and Automation of The Academy of Science of The Czech Republic, Prague.

[4] **Kadlec J., P. Nedoma (1994):** PARAbridge 1.0, MATLAB 4 interface to 3L Parallel C for transputers, Reference Guide. Institute of the Information Theory and Automation of The Academy of Science of The Czech Republic, Prague.