An Investigation of Parallelization of Code Written in MATLAB

Authors

D. J. Fabozzi, II
The University of New Mexico
Maui High Performance Computing Center

Peter Young
Institution of Michigan - Environmental Research

Melody Bohn
Maui Community College

Blaise Barney
The University of New Mexico
Maui High Performance Computing Center

Paul De Lauretis
Maui Community College

Mathew Green
The University of Massachusetts
Disclaimer

The High Performance Computing, Education & Research Center (HPCERC) provides a focus for high performance computing and communication at the University of New Mexico (UNM). HPCERC is committed to innovative research in computational and computer science with emphasis on both algorithm development and application. As part of this commitment, HPCERC sponsors this technical report series. The technical reports are subject to internal review by the HPCERC. However, the material, as presented, does not necessarily reflect any position of the HPCERC. Further, neither UNM, nor the AHPCC, makes any warranty or assumes any legal liability or responsibility for the accuracy, completeness, or usefulness of any information contained in this report.

Frank L. Gilfeather, Executive Director, HPCERC
Brian T. Smith, Chief Scientist, HPCERC
John S. Sobolewski, Chief Technologist, HPCERC
Ernest D. Herrera, Associate Director, HPCERC
Susan Atlas, Associate Director, AHPCC Computer Systems Research
Bob Ballance, Associate Director, AHPCC Science & Engineering Research
An Investigation of Parallelization of Code Written in MATLAB

D.J. Fabozzi II, Blaise Barney
Maui High Performance Computing Center
fabozzi@mhpcc.edu, blaise@mhpcc.edu

Peter Young
Environmental Research Institute of Michigan
young@erim-int.com

Paul De Lauretis, Melody Bohn
Maui Community College
edoc2@aloha.net, kas@maui.net

Mathew Green
University of Massachusetts
mgreen@cs.umass.edu

ABSTRACT

The MathWorks’ MATLAB is a very popular tool among scientists and engineers due to its rich mathematical library set, flexibility, and ease of use. However, its interpretive operation is an obstacle to high performance on either embedded processors or Massively Parallel Processors. As an alternative to the manual translation of MATLAB syntax to a compilable language, there have been a number of utilities which perform language conversions, integrate parallel libraries into the MATLAB user code, or a combination of both. It was found that the degree of improvement achieved by each of these developments was dependent on many factors such as MATLAB code structure, communication interface, and execution environment. This report reviews the various approaches to improve MATLAB and further details the results of an on-site evaluation of three third-party utilities. In particular, the MathWorks MATLAB compiler [1], MultiMATLAB [2], and Real Time Express [3] utilities were evaluated at the Maui High Performance Computing Center to examine performance, applicability, availability, and development cost when utilized on a diverse set of test codes. Following the results of the on-site evaluation, recommendations are given for both user and utility developers to improve MATLAB computing in a high performance environment. This work was funded by the Defense Advanced Research Projects Agency/ Sensor Technology Office (DARPA/STO) under Air Force contract F30602-95-C-0117.
TABLE OF CONTENTS

LIST OF FIGURES ........................................................................................................... i

LIST OF TABLES .............................................................................................................. ii

1 INTRODUCTION ...................................................................................................... 1
  1.1 Survey Of Related Work ..................................................................................... 1
  1.2 Review Of Evaluated Parallel-MATLAB Utilities ........................................... 2

2 COMPILER APPROACHES .................................................................................... 3
  2.1 FALCON - A MATLAB to Fortran 90 Translator ............................................. 3
  2.2 MathTools’ MATCOM – A MATLAB to C/C++ Compiler ................................ 5
  2.3 MathWorks’ MATLAB Compiler ...................................................................... 6

3 INTERPRETIVE APPROACHES ........................................................................... 7
  3.1 Matpar: Parallel Extensions to MATLAB ......................................................... 7
  3.2 PT: Parallel Toolbox for MATLAB ................................................................... 8
  3.3 MultiMATLAB: MATLAB on Multiple Processors .......................................... 9
    3.3.1 Migrating MATLAB code to MultiMATLAB and Matrix Multiply .............. 11
    3.3.2 3D .............................................................................................................. 14
    3.3.3 Child Processor Collection Approach and Matrix Inversion ...................... 14
    3.3.4 Fast Fourier Transform (FFT) .................................................................. 16

4 FULL SUITE APPROACH – REAL TIME (RT) EXPRESS ......................... 17
  4.1 OVERVIEW ....................................................................................................... 17
  4.2 RTExpress – Usage and Dependencies ............................................................. 18
    4.2.1 Automatic Data Parallelism .................................................................... 18
    4.2.2 Function /Task Parallelism, Pipelined Parallelism .................................... 19
  4.3 Evaluation of RTExpress ................................................................................ 19

5 RECOMMENDATIONS ............................................................................................ 26

APPENDIX A - SOURCE MATLAB TEST CODE

A.1 AQ .................................................................................................................. 29
A.2 CQ .................................................................................................................. 2
A.3 FD .................................................................................................................. 2
A.4 DI .................................................................................................................. 3
A.5 GA .................................................................................................................. 3
A.6 EC .................................................................................................................. 4
A.7 RK .................................................................................................................. 4
LIST OF FIGURES

Figure 1. serialMul.m - serial matrix multiplication ......................................................... 12
Figure 2. MultiMATLAB version - Master Processor .......................................................... 12
Figure 3. Slave Processor Execution .................................................................................. 13
Figure 4. Matrix Multiply Performance with MultiMATLAB.............................................. 13
Figure 5. “3D” Performance with MultiMATLAB ............................................................... 14
Figure 6. Matrix Inverse Performance with MultiMATLAB using "Get" ......................... 15
Figure 7. Matrix Inverse Performance with MultiMATLAB using "Sum" ......................... 15
Figure 8. Column-wise 2D FFT Performance with MultiMATLAB .................................. 17
Figure 9. Function Flow of Various RTExpress Components ............................................. 16
Figure 10. Timings for test code tst_dirich.m ................................................................... 20
Figure 11. Timings for test code inv.m ............................................................................... 21
Figure 12. Timings for test code simple.m ......................................................................... 22
Figure 13. Timings for test code filt.m ............................................................................... 22
Figure 14. Timings for test code rawtofft1.m .................................................................. 23
Figure 15. Timings for test code rawtofft2.m .................................................................. 24
Figure 16. Vectorized and Non-Vectorized Versions of Matrix Multiply ......................... 25
LIST OF TABLES

Table 1. Survey of Leading Approaches ................................................................. 2
Table 2. "FALCON" Code Conversion Results ....................................................... 7
Table 3. Available Matpar Commands ................................................................. 8
Table 4. Available Parallel Toolbox Commands .................................................... 9
Table 5. MultiMATLAB Commands .................................................................. 11
Table 6. MultiMATLAB Test Codes ................................................................. 11
Table 7. RTExpress Test Codes ........................................................................ 20
Table 8. RTExpress Communication Variations .................................................... 20
Table 9. Timings for test code loops.m (non-vector) ............................................. 24
Table 10. Timings for test code loops.m (vectorized) ........................................... 25
Table 11. Timings for test code tst_icn.m and tst_finedif.m ................................ 25
1 INTRODUCTION

MATLAB is a very popular tool among scientists and engineers due to its rich mathematical library set, flexibility, and ease of use. However, its interpretive operation can be an obstacle to achieving high performance on Massively Parallel Processors (MPP). As a result, there have been a number of utilities to assist users in improving MATLAB code performance on either single or distributed processors through either conversion to compiled languages, substituting message passing within the MATLAB user code or a combination of both. The degree of improvement achieved by each of these utilities, however, depends on many factors such as MATLAB source code structure, improvement goal, labor, MATLAB licenses, and available computing resources. This report surveys the related MATLAB-based computing techniques and further details the results of the on-site evaluation of three leading solutions: the MathWorks’ MATLAB compiler, Cornell Theory Center’s MultiMATLAB, and Integrated Sensors Incorporated Real Time Express. Each solution was evaluated against a diverse suite of codes to demonstrate various performance and integration details.

1.1 Survey Of Related Work

The results of our survey of found that the field of work of improving MATLAB performance can be classified into three categories:

1. Compiler approaches—utilities which convert MATLAB code to compiled machine language.
2. Interpretive approaches—utilities which allow MATLAB algorithms to be distributed and executed within the MATLAB environment on independent processors.
3. “Full Suite” approaches—utilities which convert MATLAB code to compiled code such as C/C++ and subsequently insert parallel constructs into the generated language.

These are summarized in Table 1:
Table 1. Survey of Leading Approaches

<table>
<thead>
<tr>
<th>Category</th>
<th>Title</th>
<th>Developer info</th>
<th>Approach</th>
<th>Availability</th>
<th>Reference</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Falcon</td>
<td>L. DeRose, D. Padua, CSRD, University of Illinois</td>
<td>MATLAB to Fortran 90 Translater</td>
<td>Authors</td>
<td><a href="http://www.csrd.uiuc.edu/falcon/">www.csrd.uiuc.edu/falcon/</a></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>MatCom</td>
<td>MathTools, Ltd.</td>
<td>MATLAB to C/C++</td>
<td>COTS</td>
<td><a href="http://www.mathworks.com/">www.mathworks.com/</a></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Mcc</td>
<td>MathWorks, Inc.</td>
<td>MATLAB to C/C++</td>
<td>COTS</td>
<td><a href="http://www.mathworks.com/">www.mathworks.com/</a></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MatPar</td>
<td>Paul L. Springer, Jet Propulsion Laboratory</td>
<td>Interpretive HPC</td>
<td>Author</td>
<td><a href="http://www.hpc.jpl.nasa.gov/PS/MATPAR/index.html">www.hpc.jpl.nasa.gov/PS/MATPAR/index.html</a></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>MultiMATLAB</td>
<td>Anne E. Treethan, Cornell Theory Center</td>
<td>Interpretive HPC</td>
<td>Author</td>
<td><a href="http://www.tc.cornell.edu/~anne/projects/MM.html">www.tc.cornell.edu/~anne/projects/MM.html</a></td>
<td>Tested at MHPCC</td>
</tr>
<tr>
<td>3</td>
<td>RTExpress</td>
<td>Integrated Sensors Inc.</td>
<td>Translation to C, MPI</td>
<td>COTS</td>
<td><a href="http://www.rtexpress.com/">http://www.rtexpress.com/</a></td>
<td>Tested at MHPCC</td>
</tr>
</tbody>
</table>

1.2 Review Of Evaluated Parallel-MATLAB Utilities

A brief description of the MATLAB test codes will properly introduce the evaluation results. The test codes were selected based on size, complexity, and code structure. The first set, referred to as the “Falcon” codes[4], consisted of eleven independent algorithms of 100 lines or less containing data structures of 500 elements or less. The second set of MATLAB test codes[5] consisted of variations of FFT processing, nested loops, and matrix multiply operations. The third test code [6] consisted of a Space-Time Adaptive Processing (STAP) distribution consisting of 47 library routines containing on average 100 lines of code. Where applicable, each code suite was tested with each third-party utility.

The first utility examined was the MathWorks’ MATLAB compiler, which converts MATLAB code to C/C++. The converter/compiler demonstrated varying success depending on code style, use of data types (i.e. INTEGER, REAL, COMPLEX, LOGICAL), rank (i.e. SCALAR, VECTOR, MATRIX) and shape of arrays (i.e. size of each dimension). We found, for example, the MATLAB compiler data type inference provided speed-up ranging from 1 to 581 on the Falcon codes, depending on the use of complex numbers, “for” loops, and matrices. However, the MATLAB compiler provided only a negligible speedup on the STAP codes because of the strong use of complex numbers and MATLAB library routines. These as well as other factors including dependency on matrix element access and MATLAB callback routines are discussed in this report.

The second utility, Cornell Theory Center’s MultiMATLAB, allows users to integrate parallel MEX routines into serial MATLAB programs for multiprocessor execution.
Message Passing Interface (MPI)—based parallel routines are called from within the interpretive MATLAB environment to perform process control, message passing, data distribution, and arithmetic operations. Though MultiMATLAB allows for the quick development of parallel MATLAB software, the utility was found to improve algorithm performance only for data parallel models in which data transfer time is only a small fraction of the computation time. Our testing revealed that the matrix multiply, FFT, matrix inverse and eigenvalue test codes all achieved improvement, but the improvement exhibited strong dependence on the selection of block gathering method.

The most aggressive approach to integrating parallel constructs into MATLAB is achieved by Integrated Sensors’ RTExpress™ (Real-Time Express). RTExpress performs a conversion of MATLAB source to C followed by the automatic integration of message passing constructs and parallel libraries. Testing of RTExpress found measurable improvement for vector operations, FFT’s, and data intensive parallel operations but found that it required a certain amount of training and that it did not perform well on code containing “for” loops.

While no utility offers a universal solution to improving MATLAB performance, this report illustrates various factors which influence the performance of MATLAB code in a High Performance Computing (HPC) environment. This report concludes with recommendations for both third party utility developers and MATLAB users to improve the development of future High Performance Computing applications with MATLAB. Appendix A lists all of the relevant MATLAB source codes used in the testing.

2 COMPILER APPROACHES

The compiler approaches are those that translate the MATLAB syntax to a compilable language such as C, C++, or Fortran 90. We examined three projects, one from academia and two from commercial companies.

2.1 FALCON - A MATLAB to Fortran90 Translator

This project involved the use of a translator to convert MATLAB code to Fortran 90. This project built inference mechanisms to determine functions and variable characteristics from the MATLAB syntax. Both static and dynamic inference strategies were implemented.
against a simplified branching model for variable identification properties including type (i.e. integer, real, complex, or logical), rank (i.e. scalar, vector, or matrix), and shape (i.e. size of each dimension). Through the use of both forward and backward propagation strategies, the project achieved results which were not only comparable to hand written Fortran 90 equivalent code but superior to results achieved through the use of the MATLAB compiler.

Some of FALCON’s test parameters are explained. The FALCON project’s static inference mechanism uses a Static Single Assignment (SSA) representation of MATLAB programs in which each variable is assigned a value by at most one date type. The intrinsic types of variables are determined individually by the interpreter from the values of the operands. For non-SSA test code which contained variables whose types which cannot be determined immediately, such as those near looping assignments, types are temporarily assigned and resolved through forward and backward analysis of the MATLAB code.

Similarly, rank information is inferred by the examination of the operands as well during the same compiler pass. Rank determination is very important because if all data types are converted to the default matrix type, the computational labor to process and manage matrix data increases significantly in comparison to other inference classes. Lastly, though shape is found similarly to rank information, it was found to be best inferred through both static and dynamic examinations. Shape information is influential in code efficiency because element-wise access to matrices results in the computational overhead of dynamic allocation.

The FALCON project instruments dynamic type and shape inference through the use of shadow variables and conditional statements that are inserted into the code under test during execution. Also implemented is the use of a dimension propagation algorithm in which variable index accesses are counted so as to inform the resulting Fortran version how to implement memory requests. In other words, it was found that the overhead due to MATLAB memory requests with each increase in a matrix’s size could be minimized if the final size of the matrix was known early and allocated only once.

The FALCON project tested these algorithms against 10 MATLAB codes which perform varying computational actions, the results of which are summarized as follows. The conversion to a compiled language will yield little performance improvement if the MATLAB code uses mostly complex matrices, spends most of its time performing library calls and does not perform incremental array indexing. Codes converted to Fortran 90 which
utilize scalar variables will be dramatically improved through the use of type inference. The use of shape inference against these same codes, however, will only improve them proportional to the number of matrices used. Compiled codes that will experience the greatest speedup over their MATLAB counterparts are those which utilize non-complex data types and little use of library routines.

When compared to the Mathwork’s MATLAB compiler, the FALCON translator demonstrated superior performance due to the preallocation of variables and the handling of small dimension matrix operations.

The disadvantages of the FALCON compiler include support and testing. As this work is not a commercial product, its support and availability are questionable. Also, it is yet to be determined how the compiler performs on MATLAB subroutines over 60 lines as well as on a suite of operational test code.

### 2.2 MathTools’ MATCOM – A MATLAB to C/C++ Compiler

MATCOM is a MATLAB to C/C++ compiler, much like the MathWork’s compiler. As with the MathWork’s compiler, MATCOM can create MEX files as well as standalone applications. Regarding the C++ objects produced, MATCOM uses templates “T” and declares the base object “M” as an instance of the class “T”. MATCOM does provide the global inference of a float data type and does allow global specification of types with variable names. As of this writing MATCOM contains many features the MathWork’s compiler does not including graphical user interface (GUI) functions (Windows only), sparse matrix support, multidimensional imaging support, enhanced error reporting of error location, compilation of S-funcs for Simulink, m-files for RTWLoad/save of V5 mat-files format, and recursive function search. This product was briefly evaluated at the MHPCC but was found to have substandard error reporting. Another feature we found useful in MATCOM was the recursive subroutine search which MATLAB did not perform at the time of this writing. For the large STAP distributions the MATCOM compiler recursively searched all routines that were called by the original function, where the MathWork’s compiler needed to be independently invoked for each function call.
2.3 MathWorks’ MATLAB Compiler

Mathworks, Inc. provides a facility to convert MATLAB code to C language code in either a standalone or as a MATLAB executable callable from MATLAB. The MathWork’s compiler contains similar inference mechanisms as MATCOM for imputing type information or matrix handling. The MathWork’s compiler V1.2 offered two command line switches (–r and –i) which perform imputation of data types. The “–i” compiler option generates code that fixes array size, eliminates boundary checking and type checking. The “–r” switch on the other hand tells the compiler to convert all data types to real rather than complex. The MATLAB compiler also has the ability to control the type imputations by specifying particular MATLAB variables individually through pragmas, which impute on a file basis. The pragmas also include a %#ivdep switch which tells the compiler to ignore vector dependencies. The MathWork’s compiler also contains three special functions—reallog, realpow, and realsqrt which are real-only versions of log, , and sqrt functions.

The MathWork’s compiler contains a library of routines which can be manually substituted for MATLAB language routines to improve performance over callbacks to the MATLAB interpreter. However, these libraries were not investigated for this report.

Operation of the MathWork’s compiler on small sets of code is straightforward but the conversion of a full suite of code of requires attention to each called subroutine and more operator intervention. Utilities were developed to facilitate the use of MATLAB “load,” “eval,” “save,” and “input” commands which are not handled by the MathWork’s compiler.

Table 2 lists the results of the conversion of the FALCON test suite with the MathWork’s compiler. The performance of the test code after translation through the MATLAB compiler’s in general agreed with both our expectations and with the FALCON report results.
Table 2. "FALCON" Code Conversion Results

* - ri switches are not applicable for all test codes

In general all test codes yielded speedup through compilation and syntax assumptions made earlier were consistent. The conversion to a compiled language yielded little performance improvement if the MATLAB code used complex matrices, library calls and array indexing. Degrees of improvement are proportional to these primary dependencies.

3 INTERPRETIVE APPROACHES

3.1 Matpar: Parallel Extensions to MATLAB

Of the interpretive approaches which insert the parallel constructs directly in MATLAB syntax, Matpar[7] incorporates PVM message passing utilities for data movement. These routines, which provide access to Scalapack, PBLAS, BLAS, and BLACS libraries, are listed in Table 2.
### Table 3. Available Matpar Commands

<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>p_config()</td>
<td>Specify parallel computer and quality of nodes</td>
</tr>
<tr>
<td>p_add()</td>
<td>Matrix add</td>
</tr>
<tr>
<td>p_subtract()</td>
<td>Matrix subtract</td>
</tr>
<tr>
<td>p_bode()</td>
<td>Generate frequency response of a matrix</td>
</tr>
<tr>
<td>p_delete()</td>
<td>Delete a persistent matrix</td>
</tr>
<tr>
<td>p_eye()</td>
<td>Generate an identity matrix on the server</td>
</tr>
<tr>
<td>p_freqresp()</td>
<td>Generate frequency response of a matrix</td>
</tr>
<tr>
<td>p_inv()</td>
<td>Matrix inverse</td>
</tr>
<tr>
<td>p.lu()</td>
<td>LU factorization of a matrix</td>
</tr>
<tr>
<td>p_mult()</td>
<td>Matrix multiply</td>
</tr>
<tr>
<td>p_multrans()</td>
<td>Matrix transpose multiply</td>
</tr>
<tr>
<td>pPersist()</td>
<td>Keep matrix on remote server</td>
</tr>
<tr>
<td>p_pinv()</td>
<td>Pseudoinverse</td>
</tr>
<tr>
<td>p_qr()</td>
<td>QR factorization</td>
</tr>
<tr>
<td>p_smult()</td>
<td>Scalar matrix multiply</td>
</tr>
<tr>
<td>p_solv()</td>
<td>Matrix division</td>
</tr>
<tr>
<td>p_svd()</td>
<td>Single value decomposition</td>
</tr>
<tr>
<td>p_trace()</td>
<td>Compute trace</td>
</tr>
</tbody>
</table>

Matpar was developed and tested on a variety of architectures including HPExamplar, SunUltra SPARC, Intel Paragon, and Cray T3-D.

Though Matpar was not tested on site, the documentation indicates that the implementation does not require multiple MATLAB licenses for each distributed process. Matpar’s limitations, however, are in its limited number of libraries implemented and limited distribution.

### 3.2 PT: Parallel Toolbox for MATLAB

Parallel Toolbox (PT)[8] allows distributed MATLAB programs to execute in a Single Program Multiple Data (SPMD) configuration. PT also uses PVM to implement a master-slave paradigm for concurrent program execution. Though similar in using PVM, PT contrasts with Matpar in that it provides both data management and process control. Table 4 lists some of the available PT commands.
<table>
<thead>
<tr>
<th>COMMAND</th>
<th>ACTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>pt_addEngines()</td>
<td>add one or more compute engines</td>
</tr>
<tr>
<td>pt_barrier()</td>
<td>synchronize members of a group</td>
</tr>
<tr>
<td>pt_broadcast()</td>
<td>broadcast a matrix to a group</td>
</tr>
<tr>
<td>pt_cleanup()</td>
<td>cleanup after job failure</td>
</tr>
<tr>
<td>pt_delete()</td>
<td>Delete a compute engine from a group</td>
</tr>
<tr>
<td>pt_erReset()</td>
<td>Resets error log file</td>
</tr>
<tr>
<td>pt_exit()</td>
<td>exit a worker from the group</td>
</tr>
<tr>
<td>pt_getinst()</td>
<td>determine the instance # from a group</td>
</tr>
<tr>
<td>pt_getwid()</td>
<td>determine worker id</td>
</tr>
<tr>
<td>pt_getsize()</td>
<td>get the # of members of a group</td>
</tr>
<tr>
<td>pt_hosts()</td>
<td>find all hosts where engines are present</td>
</tr>
<tr>
<td>pt_joingroup()</td>
<td>Enroll a worker in a group</td>
</tr>
<tr>
<td>pt_kill()</td>
<td>kill PT tasks in PVM</td>
</tr>
<tr>
<td>pt_lvgroup()</td>
<td>remove worker from a group</td>
</tr>
<tr>
<td>pt_mywid()</td>
<td>get worker id</td>
</tr>
<tr>
<td>pt_send()</td>
<td>send a matrix to a worker</td>
</tr>
<tr>
<td>pt_recv()</td>
<td>receive a matrix from another worker</td>
</tr>
<tr>
<td>pt_shutdown()</td>
<td>shutdown engines</td>
</tr>
</tbody>
</table>

Table 4. Available Parallel Toolbox Commands

Though PT was not evaluated on-site, the documentation revealed that this implementation requires MATLAB licenses for each worker node. Lastly, like Matpar, PT lacks significant distribution.

3.3 MultiMATLAB: MATLAB on Multiple Processors

MultiMATLAB is similar to PT in that it provides data distribution library routines which are interleaved directly into the original MATLAB source code. MultiMATLAB was developed on an IBM SP system utilizing the P4 implementation of MPICH for the underlying communication. Table 5 lists the available MultiMATLAB commands.
Most commands can be run only on the master process (process 0).

Commands marked by * can be run on the master process or on the remote processes (1:Nproc-1).

### Starting and Stopping MultiMATLAB.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Start</td>
<td>Initialize remote processes and begin MultiMATLAB session</td>
</tr>
<tr>
<td>Interrupt</td>
<td>Interrupt MultiMATLAB processes during computation</td>
</tr>
<tr>
<td>Abort</td>
<td>Abort MultiMATLAB session remaining in originally interactive session</td>
</tr>
<tr>
<td>Quit</td>
<td>Terminate remote processes and end MultiMATLAB session</td>
</tr>
</tbody>
</table>

### Process Arrangement and Identification.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*ID</td>
<td>Task ID of a process</td>
</tr>
<tr>
<td>*Nproc</td>
<td>Total number of MultiMATLAB processes active</td>
</tr>
<tr>
<td>Grid</td>
<td>Arrange the processes in a grid</td>
</tr>
<tr>
<td>*Gridsize</td>
<td>Dimensions of the grid of processes</td>
</tr>
<tr>
<td>*Coord</td>
<td>Coordinates of a process in the grid</td>
</tr>
</tbody>
</table>

### Running Commands on Multiple Processes

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Eval</td>
<td>Evaluate a command on one or more processes</td>
</tr>
</tbody>
</table>

### Communication

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>*Send</td>
<td>Send data from one process to another</td>
</tr>
<tr>
<td>*Recv</td>
<td>Receive data sent from another process</td>
</tr>
<tr>
<td>*Probe</td>
<td>Determine if communication has been completed</td>
</tr>
<tr>
<td>*Barrier</td>
<td>Synchronize processes</td>
</tr>
<tr>
<td>Put</td>
<td>Put data from the master process onto remote processes</td>
</tr>
<tr>
<td>Get</td>
<td>Put data from remote process onto the master process</td>
</tr>
<tr>
<td>Bcast</td>
<td>Transmit data to all processes using a tree structure</td>
</tr>
</tbody>
</table>

### Distribution

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distribute</td>
<td>Distribute a matrix according to the values of Coord</td>
</tr>
<tr>
<td>Collect</td>
<td>Collect a matrix according to the mask created by Distribute</td>
</tr>
<tr>
<td>Shift</td>
<td>Shift data between processes</td>
</tr>
</tbody>
</table>

### Arithmetic

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>Find the pointwise maximum of matrices on several processes</td>
</tr>
<tr>
<td>Min</td>
<td>Find the pointwise minimum of matrices on several processes</td>
</tr>
<tr>
<td>Sum</td>
<td>Find the pointwise sum of matrices on several processes</td>
</tr>
<tr>
<td>Prod</td>
<td>Find the pointwise product of matrices on several processes</td>
</tr>
</tbody>
</table>
MultiMATLAB was evaluated and found to be useful for quick development of embarrassingly parallel (replication of the same algorithm on blocks of data) and SPMD parallel routines. The following sections discuss our findings resulting from the migration of test routines in Table 6 to the MultiMATLAB environment.

### Table 5. MultiMATLAB Commands

<table>
<thead>
<tr>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Window</td>
<td>Arrange figures in a grid according to function parameters</td>
</tr>
<tr>
<td>Reset</td>
<td>Reset default window position to MATLAB default</td>
</tr>
<tr>
<td>Refresh</td>
<td>Repaint all current figures</td>
</tr>
</tbody>
</table>

### Table 6. MultiMATLAB Test Codes

<table>
<thead>
<tr>
<th>Test Code</th>
<th>Description</th>
<th>Problem Size</th>
<th># Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>Generation of 3D-surface</td>
<td>(X \times Y \times Z = [51:51 \times 31:31 \times 21:21])</td>
<td>30</td>
</tr>
<tr>
<td>MAT_INV4</td>
<td>matrix inverse</td>
<td>(X = Y = [100:800])</td>
<td>43</td>
</tr>
<tr>
<td>FFT</td>
<td>MATLAB column-wise FFT</td>
<td>(X = Y = [100:800])</td>
<td>51</td>
</tr>
<tr>
<td>Simple</td>
<td>MATLAB FFT2 algorithm</td>
<td>(X = Y = [100:2500])</td>
<td>40</td>
</tr>
<tr>
<td>MM</td>
<td>Matrix Multiply</td>
<td>(X = Y = [50:800])</td>
<td>25</td>
</tr>
</tbody>
</table>

### 3.3.1 Migrating MATLAB Code to MultiMATLAB and Matrix Multiply

To introduce our findings, the process of migrating source code to MultiMATLAB is discussed, beginning with our serial matrix multiply example in Figure 1.

```matlab
% serialMul.m
% This MultiMATLAB M-file serial matrix multiplication
%```
function z = serialMul(n);
    A = randn(n);
    B = randn(n);
    tic;
    for i = 1 : n
        for j = 1 : n
            C(i, j) = A(i, :) * B(:, j);
        end
    end
    z = toc;

Figure 1. serialMul.m - serial matrix multiplication

Figure 2 illustrates how a master process broadcasts the matrices to the slave processors and calls a new routine, remMul.m in Figure 3, which performs the partitioned matrix multiply. As shown, this partitioning is performed on each processor according to limits set by values of “L” (Lower) and “U” (Upper) which are determined by the child process environment variable “Nproc.” Upon completion, the submatrix computation on all slave processors in blocks “C” are returned to the master processor.

% matMul.m
% This MultiMATLAB M-file performs matrix multiplication of two ( n x n ) square matrices % w/ random entries for scalability timing tests.
function z = matMul(n)
    A = randn(n);
    B = randn(n);
    C = zeros(n);
    tic;
    Bcast('A');
    Bcast('B');
    Bcast('C');
    Eval('remMul');
    D = Sum('C');

Figure 2. MultiMATLAB version - Master Processor
L = ID * n / Nproc + 1 ;
U = (ID+1)* n / Nproc ;

for i = 1 : n
    for j = L : U
        C( i, j ) = A( i, : ) * B( :, j ) ;
    end
end

Figure 3. Slave Processor Execution

The matrix multiply example was easily implemented and found to yield measurable improvement for matrix sizes exceeding n=100, as indicated in Figure 4. The top trace represents the serial performance, the middle trace represents the performance with MultiMATLAB, and the lowest trace represents the communication overhead from MultiMATLAB. Other test codes, as discussed in the following sections, yielded varying results.
3.3.2 3D

The “3D” test code generates eigenvectors of 3x3 matrices. Because the computation load exceeds the communication latency, MultiMATLAB improves the execution as the number of eigenvector calculations performed grows to $10^6$.

![Figure 5. “3D” Performance with MultiMATLAB](image)

3.3.3 Child Processor Collection Approach and Matrix Inversion

As MATLAB provides a library module for matrix inverse, we developed an implementation of matrix inverse computation [9] based on the formula:

$$A^{-1} = \frac{1}{\text{det}(A)} \times \text{adj}(A)$$

where

\text{det}(A) is the matrix determinant of A and \text{adj}(A) is the adjoint of A.

However, because the matrix summation here utilized an ordered “for” loop, performance degraded with increased processors. As our first implementation utilized the
MultiMATLAB “Get” command, which orders data collection from child processors, our example experiences degraded performance, as indicated in Figure 6.

Figure 6. Matrix Inverse Performance with MultiMATLAB using “Get”

However, when our algorithm utilized the unordered “Sum” to collect matrix blocks from child processors, we achieved measured speedup with increased processors, as indicated in Figure 7.
3.3.4 Fast Fourier Transform (FFT)

To validate our claim regarding the influence of child collection, we examined two FFT algorithms, a column-wise and two-dimensional matrix Fast Fourier Transforms. Though our column-wise algorithm, “Simple FFT,” does have speedup at 2 processors, these experienced similar degradation because of the results gathering approach, as indicated in Figures 8 and 9.

![Figure 7. Matrix Inverse Performance with MultiMATLAB Using "Sum"](image)

![Figure 8. “Simple FFT” Performance with MultiMATLAB](image)
In summary, we found MultiMATLAB to be effective for scenarios in which the algorithm is a data parallel problem, and the developer has many MATLAB licences and little development time. However, though MultiMATLAB is simple to install and use, it is not yet commercially supported.

4 FULL SUITE APPROACH – Real Time (RT) Express

4.1 OVERVIEW

RTExpress™ performs automatic compilation to C and parallelization of software written in MATLAB. Parallelization of the application can be done automatically by RTExpress, explicitly directed by the developer, or a combination of both.

RTExpress’ graphical user interface (GUI) Target Balancing Tool directs the interaction with the underlying software which control:

- Parallel communications
- Data acquisition and output (I/O)
- Real-time user displays and controls
- Real-time performance monitoring
- Post-mortem analysis
RTExpress utilizes and works in conjunction with the MATLAB compiler, mathematical subroutine libraries, parallel function libraries, BLAS and BLACS libraries, the native C compiler and MPI. Figure 10 depicts the function flow of the various RTExpress components.

![Figure 10. Function Flow of RTExpress Components](image)

### 4.2 RTExpress – Usage and Dependencies

RTExpress can be used to translate a serial MATLAB application into a parallel application by several different methods which vary greatly in the amount of developer knowledge and effort required. They can also vary greatly in the efficiency of the resulting parallel application. The primary parallel paradigms RTExpress implements are described in this section.

#### 4.2.1 Automatic Data Parallelism

In accordance with the data parallel model, the developer simply specifies the number of instances of the executable to be run and allows RTExpress to distribute the work across the node pool and produce the translated code with all of the necessary parallel constructs for data parallelism. RTExpress automatically distributes the matrix data and operations according to the number of instances without any developer interaction.
4.2.2 Function /Task Parallelism, Pipelined Parallelism

This involves decomposing a single serial application into multiple modules for concurrent execution. RTE recognizes lines of code which can be executed concurrently and partitions code into discreet groups. Each "group" is then viewed as a separate executable (task) Inter-task communications for shared data is handled automatically by RTExpress by means of its groupimport and groupexport routines. RTExpress allows the simultaneous incorporation of both data and functional parallelism.

4.3 Evaluation of RTExpress

RTExpress was evaluated on site at the MHPCC in the areas of translation effectiveness to parallel models, speedup, scalability, portability and robustness. These along with other characteristics are detailed in this section.

Over a dozen codes were tested with RTExpress of which nine achieved significant results. Though the remaining codes did not operate under RTE, they did provide valuable insights into the product’s behavior and limitations. A brief description of each of these codes appears in the Table 7.

<table>
<thead>
<tr>
<th>Test Code</th>
<th>Description</th>
<th>#Lines*</th>
</tr>
</thead>
<tbody>
<tr>
<td>tst_icn.m</td>
<td>Incomplete Cholesky factorization of matrix. Uses a double nested loop to compute the incomplete Cholesky factorization of a matrix. Main feature is use of a multi-typed built-in (sqrt). Problems size: 400x400</td>
<td>33</td>
</tr>
<tr>
<td>tst_dirich.m</td>
<td>Dirichlet solution to Laplace’s equation. An iterative method for the solution to Laplace’s equation. An elementary-operation intensive program which requires element-wise access of grid elements. Problem size: 41x41</td>
<td>39</td>
</tr>
<tr>
<td>tst_finedif.m</td>
<td>Finite difference solution to the wave equation. A numeric approximation method for the solution of hyperbolic differential equations. This is an elementary-operation intensive program that performs indexed updates to a two dimensional grid. Problem size: 451x451</td>
<td>28</td>
</tr>
<tr>
<td>inv.m</td>
<td>Inverse matrix operation performed on a 2000 x 2000 matrix. The MATLAB inv() function is used to perform the operation.</td>
<td>8</td>
</tr>
<tr>
<td>loops.m</td>
<td>Four simple loops, each of which is double nested, where i=2000 and j=2000. A simple assignment is made for each element in a 2000x2000 matrix.</td>
<td>51</td>
</tr>
<tr>
<td>simple.m</td>
<td>2D FFT operations. Using the MATLAB fft() and ifft() functions on a 1024x1024 matrix.</td>
<td>24</td>
</tr>
<tr>
<td>filt.m</td>
<td>Image filtering operations. An input file provides multiple frames of an initial image. Each frame is comprised of a 278x392 matrix of reals. An edge detection operation is performed on each frame. 10 frames are processed.</td>
<td>74</td>
</tr>
</tbody>
</table>
All timings were performed in "batch" mode to ensure dedicated CPU usage. The IBM SP nodes used were P2SC, 160 MHz "thin" nodes with 512 MB of memory (2 x 256 MB memory cards). Data cache was 64 KB with 128 byte cache lines. Results shown are the average of a minimum of five independent runs per code. The MATLAB "tic" and "toc" calls were used to obtain the timings.

The legends for graphs should be interpreted from Table 8:

| IP | RTExpress produced C language object module run with Internet Protocol communications. |
| US | RTExpress produced C language object module run with User Space communications. |
| MAT-I | Execution within MATLAB as interpreted source. Always run serially on one processor. |
| MAT-C | Execution within MATLAB as a MATLAB compiled C language object module. Always run serially on one processor. |

This code exhibits a case where translation by RTExpress dramatically decreased the performance. Figure 11 shows that performance continued to decrease as more processors...
were added. The sizeable differences between IP and US protocols is suggestive of a significant amount of data transfer taking place over the network between the multiple instances of the parallel program. Note that the MATLAB compiled executable demonstrated an execution time of less than one second when it was run serially on a single processor.

The `tst_dirich.m` code is an iterative method and contains multiple loops, one of which is double nested inside of a while loop. According to the vendor, RTExpress performs very poorly on loops, particularly nested loops, and suggested that codes should be written as much as possible using vector syntax instead.

![Timings for test code inv.m](image)

**Figure 12. Timings for test code inv.m**

The test code Inv.m consists of a call to the MATLAB `inv()` function and demonstrates considerable improvement with RTExpress, as indicated in Figure12. The matrix size was 2000 x 2000 real elements, approximately 32 MB in size, making it a reasonably large data set. Note that on one processor, the MATLAB compiler only marginally improved the interpreted execution. Also note the overhead imposed by RTExpress for the execution one processor.

Especially worth noting is the 400+ % speedup produced when the RTExpress executable is run on 2 processors versus 1. Adding additional processors continues to improve performance, though the results suggest that any gains achieved beyond 6 processors does not significantly improve performance. The relatively small differences between IP and US timings, considering that US is generally three times faster than IP, indicates that there is little interprocess communications.
The effort to parallelize this code was minimal, relying upon RTExpress to automatically analyze and implement the data decomposition.

**Figure 13. Timings for test code simple.m**

The “simple.m” test code, as shown in Figure 13, illustrates RTExpress performance for an "embarrassingly parallel" code. RTExpress implements the FFT operations using data distribution of the matrix by columns to the number of available processes. As shown, very good speedup is achieved with 2 processors and weakened improvement after 6 processors.

Again, the effort to parallelize this code was minimal, relying upon RTExpress to automatically analyze and implement the data decomposition.

**Figure 14. Timings for test code filt.m**
The `filt.m` evaluation code utilized the mixed-mode parallel paradigm of RTExpress. The original serial m-file was partitioned into four distinct groups using the RTExpress Target Balancing Tool editor. The first group continually reads a 5.5 MB input file consisting of multiple frames of image data stored as individual matrices. The other groups work in parallel to perform an image processing operation on the matrix, in this case, edge detection. The edge detection algorithm code was easily decomposed into three functional tasks. As each frame is read by the first group, the data is passed to each of the other compute groups, which then operate independently. This functional/task parallelism is enhanced by adding additional processors to the compute tasks, allowing them to perform data parallelism within their group.

Figure 14 demonstrates improvement obtained by adding additional processors to each of the edge detection groups. For example, when NPROCS is 4, then the "read" group has one processor, as does each of the three compute groups. When NPROCS is 7, the "read" task still has one processor, but each of the compute groups now has two processors. Because I/O is an inherently serial operation in RTExpress, there is no benefit to using more than one processor with the "read" group. During this evaluation, compute groups were always kept equal in the number of processors assigned to them. Additional testing could certainly be done to determine execution behavior when the processors within a group vary.

![Timings for test code rawtofft1.m](image)

**Figure 15.** Timings for test code rawtofft1.m
The two evaluation codes in Figures 15 and 16 performed the Fast Fourier Transforms utilizing opposing algorithms. The first code demonstrated weaker parallelization due to a heavy utilization of looping control.

The “loops.m” test code further demonstrate the RTE’s dependency on MATLAB loop control. In the serial version of the code, 4 loops are executed sequentially. Each code fragment consists of a doubly nested loop over a 2000x2000 matrix which performs a simple assignment to each element in the matrix.

As shown in Table 9, RTExpress parallel execution took 4 times a long as the RTExpress serial execution (4 processors x 601 seconds). Upon discovery the developers of RTExpress advised rewriting the algorithm utilizing vector operation. Following this advice, as shown in Figure 17, yielded considerable speedup as shown in Table 10.

### Timings for Test Code loops.m (non-vector)

<table>
<thead>
<tr>
<th>Method</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MATLAB interpreted</td>
<td>477.8</td>
</tr>
<tr>
<td>MATLAB compiled</td>
<td>16.2</td>
</tr>
<tr>
<td>RTExpress. Parallel.</td>
<td>596.5</td>
</tr>
<tr>
<td></td>
<td>US communications</td>
</tr>
<tr>
<td></td>
<td>IP communications</td>
</tr>
</tbody>
</table>

Table 9. Timings for test code loops.m (non-vector)
The “loops.m” example here provides insight into the overhead imposed by RTExpress by comparison with the times for the compiled (10.8s) and RTExpress operation (37.2s) on one processor.

The test codes listed in Table 11 further illustrate performance degradation with resulting from nested loops.
5 RECOMMENDATIONS

As mentioned, the degree of improvement achieved by each of these utilities is dependent on many factors such as MATLAB code structure, communication interface, and execution environment. In addition, there are other factors users need to consider such as number of MATLAB licenses available, amount of development time available, along with availability and support, which factor into using one of the mentioned parallel-MATLAB utilities. As such, we provide recommendations for both users and developers for each of the classes of approaches.

Compiler Approaches, utility developers

- **MathTools, MathWorks**: The Falcon report demonstrated two primary areas which would make both MATCOM and MathWorks compilers better products: dynamic inference and inference phase comparisons. Utilizing both of these techniques will improve these commercial products.

- **MathWorks**: MATLAB’s latest compiler version V2.0.1 does not allow any manual inference customization, this being solely handled by the MATLAB compiler. It was shown here and in the Falcon report that inference control can yield measurable performance differences. In short, allow users the control of inference by returning the use of the –r and –i switches and pragmas.

Compiler Approaches, users:

The following is a list of recommendations that developers should keep in mind when using the compilers approaches:

- MATLAB programs which spend a significant time performing library calls will not be greatly improved. Use of the MATLAB “tic” and “toc” commands can quickly determine the location of your heavy computations.

- MATLAB routines with significant matrix accesses will benefit greatly from intelligent shape inference and matrix preallocation. Therefore, if you don’t require variable matrix size, preallocate your matrices.

- MATLAB “for” loops should be vectorized as much as possible.
• Small MATLAB routines should be written inline (same file or subroutine) as much as possible.
• Identify your target compiled language data types at development time. Simply put comments around your data structures indicating your estimate of what the data types would be in a compiled language.

Interpretive Approaches, utility developers:
• As the Parallel Toolbox (PT) product requires a MATLAB license for each processor, and it is not distributed by a commercial entity it is not recommended as a general purpose solution.
• Matpar and MultiMATLAB complement each other in that Matpar requires only one license and utilizes high performance libraries. Further, both incorporate the defacto High Performance Computing communication mechanism, message passing. It is our recommendation to the developers that that the technologies be encapsulated and distributed together.

Interpretive Approaches, users:
• Use of the interpretive approaches is with risk in that none of the approaches discussed are commercially maintained and distributed.
• Users need to closely manage the computation/communication ratio in that all of the solutions investigated are built on slow communication interfaces.
• The MultiMATLAB and Matpar utilities are valuable in that they require less time to implement a solution and the user stays within the familiar MATLAB environment for all stages of development.
• Do not use the MultiMATLAB “Get” call to collect data from worker nodes. The time spent in sequencial waits will overwhelm any savings from a Massively Parallel distribution.

Full Suite approach, utility developers:
• RTExpress utilizes MATLAB’s mcc compiler which has limitations listed above. RTE may benefit from by allowing the use of either MathWork’s or MathTool’s compilers.

• RTExpress hides the control of mcc v1.2 compiler switch settings (-r-I). When mcc returns manual editing of variable inference, RTExpress should pass this capability up to the developer.

Full Suite approach, users:

• RTExpress does require some training, but the potential for MATLAB improvement can be great depending on your code size and time available.

• Vectorize loops whenever possible.
APPENDIX A - SOURCE MATLAB TEST CODE

This section lists all of the source MATLAB codes used for the on-site testing.

```matlab
function [SRmat,quad,err] =
tst_adapt(a,b,sz_guess,tol)
% Sample call
% [SRmat,quad,err] = adapt('f',a,b,tol)
% Inputs
% f name of the function
% a left endpoint of [a,b]
% b right endpoint of [a,b]
% tol convergence tolerance
% Return
% SRmat matrix of adaptive Simpson quadrature values
% quad adaptive Simpson quadrature
% err error estimate
%
% NUMERICAL METHODS: MATLAB Programs, (c) John H.
% Mathews 1995
% To accompany the text:
% NUMERICAL METHODS for Mathematics, Science and
% Engineering, 2nd Ed, 1992
% Prentice Hall, Englewood Cliffs, New Jersey, 07632,
% U.S.A.
% Prentice Hall, Inc.; USA, Canada, Mexico ISBN 0-13-
% 624990-6
% 625047-5
% This free software is compliments of the author.
% E-mail address:      in"mathews@fullerton.edu"
%
% Algorithm 7.5 (Adaptive Quadrature Using Simpson's
% Rule).
% Section 7.4, Adaptive Quadrature, Page 389
%
SRmat = zeros(sz_guess,6);
iterating = 0;
done = 1;
% SRvec = zeros(6); not necessary.

% SRvec = my_srule('f',a,b,tol);

h = (b - a)/2;
c = (a + b)/2;
Fa = 13.*(a - a.*2).*exp(-3.*a./2); %f(a);
Fc = 13.*(c - c.*2).*exp(-3.*c./2); %f(c);
Fb = 13.*(b - b.*2).*exp(-3.*b./2); %f(b);
S = h*(Fa + 4*Fc + Fb)/3;
S2 = S;
tol1 = tol;
err = tol;
SRvec = [a b S S2 err tol1];

for j =: 1:1:
    p = j;
    SR0vec = SRmat(p,:);
    err = SR0vec(5);
tol = SR0vec(6);
    if (tol <= err),
        state = done
    SR1vec = SR0vec;
    SR2vec = SR0vec;
    a = SR0vec(1);
b = SR0vec(2);
c = (a + b)/2;
err = SR0vec(5);
tol = SR0vec(6);
tol2 = tol/2;
    SR1vec = my_srule('f',a,c,tol2);
a0 = a;
b0 = c;
tol0 = tol2;
h = (b0 - a0)/2;
c0 = (a0 + b0)/2;
Fa0 = 13.*(a0 - a0.*2).*exp(-3.*a0./2); %f(a0);
Fc0 = 13.*(c0 - c0.*2).*exp(-3.*c0./2); %f(c0);
Fb0 = 13.*(b0 - b0.*2).*exp(-3.*b0./2); %f(b0);
S0 = h*(Fa0 + 4*Fc0 + Fb0)/3;
S20 = S0;
tol10 = tol0;
err1 = tol0;
SR1vec = [a0 b0 S0 S20 err1 tol10];

% SR2vec = my_srule('f',c,b,tol2);
    a0 = c;
b0 = b;
tol0 = tol2;
h = (b0 - a0)/2;
c0 = (a0 + b0)/2;
Fa0 = 13.*(a0 - a0.*2).*exp(-3.*a0./2); %f(a0);
Fc0 = 13.*(c0 - c0.*2).*exp(-3.*c0./2); %f(c0);
Fb0 = 13.*(b0 - b0.*2).*exp(-3.*b0./2); %f(b0);
S0 = h*(Fa0 + 4*Fc0 + Fb0)/3;
S20 = S0;
tol10 = tol0;
err1 = tol0;
SR2vec = [a0 b0 S0 S20 err1 tol10];
    err = abs(SR0vec(3)-SR1vec(3)-SR2vec(3))/10;
    if (err < tol),
        SRmat(p,:) = SR0vec;
        SRmat(p,4) = SR1vec(3) + SR2vec(3);
        SRmat(p,5) = err;
    else
        SRmat(p,1:m1,:) = SRmat(p:m1,:);
m = m1;
        SRmat(p,:) = SR1vec;
        SRmat(p,:1:m1) = SR2vec;
        state = iterating;
    end
end
end
quad = sum(SRmat(:,4));
err = sum(abs(SRmat(:,5)));
SRmat = SRmat(1:m,1:6);
```
% The M-file was created to supplement "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods," by Richard Barrett, Michael Berry, Tony Chan, James Demmel, June Donato, Jack Dongarra, Victor Eijkhout, Roldan Pozo, Charles Romine, and Henk van der Vorst (SIAM, 1994).
% You are free to modify any of the files and create new functions, provided
% that you acknowledge the source in any publication and do not sell the modified file.

function [x, flag, Error, iter] = tst_cgopt(A, b, max_it, tol, x)  
% -- Iterative template routine --
% Univ. of Tennessee and Oak Ridge National Laboratory
% October 1, 1993
% Details of this algorithm are described in "Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods," Barrett, Berry, Chan, Demmel, Donato, Dongarra, Eijkhout, Pozo, Romine, and van der Vorst, SIAM Publications, 1993. (ftp netlib2.cs.utk.edu; cd linalg; get templates.ps).
% [x, error, iter, flag] = cg(A, x, b, M, max_it, tol)
% cg.m solves the symmetric positive definite linear system Ax=b
% using the Conjugate Gradient method with preconditioning.
% Input A REAL symmetric positive definite matrix
% x REAL initial guess vector
% b REAL right hand side vector
% M REAL preconditioner matrix (LDR - removed from original M-file)
% max_it INTEGER maximum number of iterations
% tol REAL error tolerance
% Output x REAL solution vector
% error REAL error norm
% iter INTEGER number of iterations performed
% flag INTEGER: 0 = solution found to tolerance
% 1 = no convergence given max_it
% Modified by Luiz A. De Rose (derose@cs.uiuc.edu)

A.3 FD

function U = tst_finedif(a,b,c,n,m)  
%-----------------------------------------------------
%FINEDIF   Finite difference solution to the wave equation.
% Sample call
%   U = finedif('f','g',a,b,c,n,m)
% Inputs
% f name of a boundary function
% g name of a boundary function
% a is the width of interval [0 a]: 0<=x<=a
% b is the width of interval [0 b]: 0<=t<=b
% c is the constant in the wave equation
% n is the number of grid points over [0 a]
% m is the number of grid points over [0 b]
% Return
% U solution: matrix
% % NUMERICAL METHODS: MATLAB Programs, (c) John H. Mathews 1995
% To accompany the text:
% NUMERICAL METHODS for Mathematics, Science and Engineering, 2nd Ed, 1992
% Prentice Hall, Englewood Cliffs, New Jersey, 07632, U.S.A.
% Prentice Hall, Inc.; USA, Canada, Mexico ISBN 0-13-624990-6
% This free software is compliments of the author.
% E-mail address: in%mathews@fullerton.edu
% Algorithm 10.1 (Finite-Difference Solution for the Wave Equation).
% Section 10.1, Hyperbolic Equations, Page 507

%-----------------------------------------------------
rm1 = n - 1;

% Algorithm 10.4 (Dirichlet Method for Laplace's Equation).
% Section 10.3, Elliptic Equations, Page 531

%-----------------------------------------------------

A.4 Di

function U = tst_dirich(f1,f2,f3,f4,a,b,h,tol,max1)
%-----------------------------------------------------
%DIRICH   Dirichlet solution to Laplace's equation.
% Sample call
%   U = dirich('f1','f2','f3','f4',a,b,h,tol,max1)
% Inputs
%   f1     name of a boundary function
%   f2     name of a boundary function
%   f3     name of a boundary function
%   f4     name of a boundary function
%   a      width of interval [0 a]: 0<=x<=a
%   b      width of interval [0 b]: 0<=y<=b
%   h      step size
%   tol    convergence tolerance
%   max1   maximum number of iterations
% Return
%   U      solution: matrix
%
% NUMERICAL METHODS: MATLAB Programs, (c) John H.
% Mathews 1995
% To accompany the text:
% "Numerical Methods for Physics Using MATLAB"
% (Prentice Hall).
% 2-dimensions using Galerkin method (Neumann boundary cond.)
% eps0 = 8.8542e-12;  % Permittivity (C^2/(N m^2))
% L = 1;  % System size
% M=2;  % Number of charges (M=2 is dipole)
% Initialize position and charge of line charges
% d = 0.1*L;  % Dipole separation
% for pz=1:M % Dipole separation
%   xq(1) = L/2;
%   yq(1) = L/2+d/2;
%   q(1) = 1;
%   xq(2) = L/2;
%   yq(2) = L/2-d/2;
%   q(2) = -q(1);
%   a = zeros(80);
%   for i=1:N % Sun over charges
%     temp = cos((0:N-1)*pi*xq(1)/L);
%     temp = cos((0:N-1)*pi*yq(1)/L);
%     for k=1:M
%         delt = ones(N,1);  % delt(i) = 1 + delta(i,1)
%         delt = 2;  % Sun over charges
%         a(i, j) = a(i, j) + q(k)*tempx(i)*tempy(j)...
%               + (i-1)*delt(i)*delt(j) +
%               eps)*delt(i)*delt(j) );
%   end
% end

#include <assert.h>

#define PI 3.14159265358979323846

int main()
{
    double a, b, h, k, r, r2, r22, s1, s2, U[n][m];
    for (i=2; i<n, m;i++)
        U(i,j) = 2*r2 U(i-1,j) + U(i+1,j)
    for (j=3; j<m), i=2; i<n)
        U(i,j) = U(i,j) + relx;
        if (err<=abs(relx))
            err=abs(relx);
            return; 
    end
    end

A.5 Ga

function [theta, phi, free] = tst_galrkn(N, rho, Nplot)
% The routine supplement the book,
% "Numerical Methods for Physics Using MATLAB"
% (Prentice Hall).
% 2-dimensions using Galerkin method (Neumann boundary cond.)
eps0 = 8.8542e-12;  % Permittivity (C^2/(N m^2))
L = 1;  % System size
M=2;  % Number of charges (M=2 is dipole)
Initialize position and charge of line charges
d = 0.1*L;  % Dipole separation
for k=1:M % Sun over charges
    tempx = cos((0:N-1)*pi*xq(1)/L);
    tempy = cos((0:N-1)*pi*yq(1)/L);
    for i=1:N % Sun over charges
        a(i, j) = a(i, j) + q(k)*tempx(i)*tempy(j)...
               + (i-1)*delt(i)*delt(j) +
               eps)*delt(i)*delt(j) );
    end
    end
end
a = 4/(eps0*pi^2) * a; % Throw in the factor out in front
phi = zeros(Nplot,1);
theta = pi * (0:Nplot-1)/(Nplot-1);
for k=1:Nplot
x = L/2 + rho*sin(theta(k)); % Coordinates at which to
y = L/2 + rho*cos(theta(k)); % evaluate potential
for i=1:N
% tempx=cos((i-1)*pi*x/L);
xtemp = cos((i-1)*pi*x/L);
for j=1:N
% phi(k) = phi(k) + a(i,j)*tempx*cos((j-1)*pi*y/L);
phi(k) = phi(k) + a(i,j)*xtemp*cos((j-1)*pi*y/L);
end
end
% Plot potential and compare with free dipole
r_rc = [rho*sin(theta(k)) rho*cos(theta(k))];
Tmn = r_rc - [0 d/2];
Tpl = r_rc + [0 d/2];
free(k) = -q(1)/(2*pi*eps0)*(log(norm(Tmn)) -
%               log(norm(r_rc + [0  d/2]))) - ...
%   log(norm(r_rc + [0  d/2])));

A.6 Ec
function [thplot, rplot, kinetic, potential, tplot, totalE] = ...
tst_orbec(r0, v0, tau, nstep);
% copyrighted, 1993, by Alejandro Garcia
% The routine supplement the book, "Numerical Methods for Physics Using MATLAB" (Prentice Hall).
% orbe - Program to compute the orbit of a comet using the Euler method.
clear;  help orbe;  % Clear memory and print header
r = [r0 0];
v = [0 v0];
GM = 4*pi^2;      % Grav. const. * Mass of Sun
mass = 1.;        % Mass of projectile
%%%%% MAIN LOOP %%%%%%
time = 0;
for istep=1:nstep
rplot(istep) = norm(r);       % Record orbit for
polar plot
thplot(istep) = atan2(r(2),r(1));
% Plot orbit for polar plot
kinetic(istep) = .5*mass*norm(v)^2;  % Record
energies
potential(istep) = -GM*mass/norm(r); % Calculate new position and velocity
accel = -GM*r/norm(r)^3; % Gravity
v = v + tau*accel;
% Euler-Cromer step
time = time + tau;
end
totalE = kinetic + potential;

A.7 RK
function xout = rk4_orb(x,t,tau,param)
% copyrighted, 1993, by Alejandro Garcia
% The routine supplement the book, "Numerical Methods for Physics Using MATLAB" (Prentice Hall).
% Runge-Kutta integrator (4th order)
% Input arguments -
%   x = current value of dependent variable
%   t = independent variable (usually time)
%   tau = step size (usually timestep)
%   derivsRK = right hand side of the ODE, derivsRK is the
% the name of the function which returns dx/dt
% Calling format derivsRK(x,t,param).
% Output arguments -
%   xout = new value of x after a step of size tau
% half_tau = 0.5*tau;
% F1 = feval(derivsRK,x,t,param);
% t_half = t + half_tau;
% xtemp = x + half_tau*F1;
% F2 = feval(derivsRK,xtemp,t_half,param);
% xtemp = x + half_tau*F2;
% F3 = feval(derivsRK,xtemp,t_half,param);
% F4 = gravrk(xtemp,t_half,param);
% xout = x + tau/6.*(F1 + F4 + 2.*(F2+F3));

A.8 IC
% Incomplete Cholesky factorization of matrix A, with the same sparsity pattern as A. ICCG(0).
%  1 December 1993
%  R. Bramley
%  Department of Computer Science
%  Indiana University
% function [L,Error] = tst_icn(A);
n = size(A,1);
L = A;
Error = 0;
for j = 1:n
s = 0;
for i = 1:j-1;
s = s + L(i,j)*L(i,j);
end
r = sqrt(L(j,j) - s);
if (r <= 0),
Error = j;
L(j,j) = 1;
else
L(j,j) = r;
end
end
for i = 1:n
for j = 1:n;
end
end
if (L(i,j) ~= 0),
    s = 0;
    for k = 1:j-1;
        s = s + L(i,k)*L(j,k);
    end;
    L(i,j) = t*(L(i,j) - s);
    % if (L(i,j) ~= 0)
end;  % for j
L = tril(L);

% Generation of a three dimensional surface
% Mei-Qin Chen
% The Citadel

function [c,b,dd,ind] = tst_3D(amin, amax, bmin, bmax,
cmin, cmax, h)
a=amin:h:amax;
b=bmin:h:bmax;
c=cmin:h:cmax;
na=length(a);
nb=length(b);
cn=length(c);
dd=zeros(nb,nc);
ind=0;
for kk=1:na,
    for ii=1:nb;
        for jj=1:nc;
            amat=[a(1,kk) b(1,ii) c(1,jj);1 0 0;0 1
0];
            ev=eig(amat);
            znorm=real(ev).^2+imag(ev).^2;
            if max(znorm)<1,
                ind=ind+1;
                dd(ii,jj)=a(1,kk);
            end;
        end;
    end;
end;
% surf(c,b,dd)

% generate data
% a = 1000 .* rand(1024,1024);
tic;
% compute 2-D FFT in separate steps
% b = fft(a);
c = fft(b.');
d = c.;
f = ifft(d);
g = ifft(f.);
h = g.;
% compute performance
% ttime = toc;
fflops = 1165664
mflops = fflops ./ ttime
% check result
%
err = max(max(abs(a - h)))

A11FILT

functionfilt1
raw = figure(2);
colormap(gray);
edg = figure(3);
colormap(gray);

gdata = fopen('dummygray.bin', 'r');

hk = [-1 -1 -1; 0 0 0; 1 1 1];
vk = [-1 0 1; -1 0 1; -1 0 1];

theshe = 100.0;
while (1)
    [imgin,count] = fread(gdata, [278,392],'uchar');
    if (count == 0)
        fclose(gdata);
        break;
    else
        figure(raw);
        image(imgin);
        img = imgin - mean(mean(imgin));
        hf = abs(conv2(img, hk, 'same'));
        vf = abs(conv2(img, vk, 'same'));
        cf = max(threshe, (hf + vf)) - theshe;
        figure(edg);
        image(cf);
    end
end

A12RawToFFT1

function [dcr] = rawToFFT1(x, numP, numR, numC)
%
% range weight data
%
range_offset = 116;
inv_denom  = 1 / (32768*range_offset*range_offset);
for range = 1:numR,
    x((range-1)*numP+1:range*numP,:)=... 
        x((range-1)*numP+1:range*numP,:)*(range_offset+range-1) * inv_denom;
end;
%
% compute doppler weighting matrix
%
w = zeros(numP-1, numC);
wend = hanning(numP-1);
for c=1:numC,
    w(:,c) = wndw;
end;

for ir = 1:numR,
    xx(1:(numP-1),:) = w .* x((ir-1)*numP+1:ir*numP-1,:);
    xx(numP,:)=zeros(1,numC);
    dcr(1:numC*numP,ir) = reshape(fft(xx),numP*numC,1);
    xx(1:(numP-1),:) = w .* x((ir-1)*numP+2:ir*numP,:);
    dcr(numC*numP+1 : 2*numC*numP ,ir) = reshape(fft(xx),numP*numC,1);
end;

A.13 RawToFFT2

function [dcr] = rawToFFT2(x, numP, numR, numC)
    numRnumC = numR*numC;
    numPnumC = numP*numC;
    numPnumR = numP*numR;
    % range weight data
    range_offset = 116;
    inv_denom = 1 ./ (32768*range_offset*range_offset);
    l = range_offset+floor(linspace(0,(numPnumR-1)./numP,
        numPnumR));
    l = l .* l * inv_denom;
    l = l.';
    rwght = l * ones(1,numC);
    x = rwght .* x;
    % compute doppler weighting matrix
    wndw = hanning(numP-1);
    wndwpad = [wndw ; 0];
    dwght = wndwpad * ones(1, numPnumC);
    % doppler filter
    xx = zeros(numP,numC);
    for c=0:numC-1
        dcr(c*numP+1:(c+1)*numP,:) = xx(:,
            c*numR+1:(c+1)*numR);
    end;
end;
ACKNOWLEDGEMENTS

The authors wish to thank the personnel at DARPA for sponsoring this effort and the personnel at the Maui High Performance Computing Center the allowing them to complete this report. The authors wish to thank Luiz DeRose from University of Illinois, Fred Pearson from Lincoln Laboratory for their donations of test code. Further, the support from Air Force Research Laboratory, Science Applications International Corporation, and Albuquerque High Performance Research Center is also appreciated.
REFERENCES


