

Parallel MATLAB: Doing It Right

RON CHOY AND ALAN EDELMAN

Invited Paper

MATLAB is one of the most widely used mathematical computing environments in technical computing. It is an interactive environment that provides high-performance computational routines and an easy-to-use, C-like scripting language. It started out as an interactive interface to EISPACK and LINPACK and has remained a serial program. In 1995, C. Moler of Mathworks argued that there was no market at the time for a parallel MATLAB. But times have changed and we are seeing increasing interest in developing a parallel MATLAB, from both academic and commercial sectors. In a recent survey, 27 parallel MATLAB projects have been identified.

*In this paper, we expand upon that survey and discuss the approaches the projects have taken to parallelize MATLAB. Also, we describe innovative features in some of the parallel MATLAB projects. Then we will conclude with an idea of a “right” parallel MATLAB. Finally we will give an example of what we think is a “right” parallel MATLAB: MATLAB * P.*

Keywords—MATLAB, MATLAB* P, parallel, Star-P.

I. MATLAB

MATLAB [29] is one of the most widely used tools in scientific and technical computing. It started in the 1970s as an interactive interface to EISPACK [41] and LINPACK [19], a set of eigenvalue and linear system solution routines. It has since grown to a feature-rich product utilizing modern numerical libraries such as ATLAS [46] and FFTW [23] and with toolboxes in a number of application areas, for example, financial mathematics, neural networks, and control theory. It has a built-in interpreted language that is similar to Fortran 90, and the flexible matrix indexing makes it very suitable for programming matrix problems. Also, it provides hooks to Java classes and dynamically linked libraries, making integration with compiled code easy.

MATLAB gained popularity because of its user-friendliness. It has seen widespread use in classrooms as a teaching tool. Its strong graphical capabilities makes it a good data

analysis tool. Also, researchers have been known to build very complex systems using MATLAB scripts and toolboxes.

II. WHY THERE SHOULD BE A PARALLEL MATLAB

Because of its roots in serial numerical libraries, MATLAB has always been a serial program. In 1995, C. Moler of Mathworks wrote a paper [37] stating Mathworks’ intention not to develop a parallel MATLAB at that time. His arguments could be summarized as follows.

- 1) *Memory model:* Distributed memory was the dominant model for parallel computers, and for linear algebra applications, scatter/gather of the matrix took too long to make parallel computation worthwhile.
- 2) *Granularity:* For typical use, MATLAB spends most of its time in the parser, interpreter and graphics routines, where any parallelism is difficult to find. Also, to handle embarrassingly parallel applications, which only requires a collection of results at the end, MATLAB would require fundamental changes in its architecture.
- 3) *Business situation:* There were not enough customers with parallel computers to support the development.

It has been nine years since the article was written, and we have seen tremendous changes in the computing world. These changes have invalidated the arguments that there should not be a parallel MATLAB.

- 1) *Memory model:* As modern scientific and engineering problems grow in complexity, the computation time and memory requirements skyrocket. The increase in processor speed and the amount of memory that can fit in a single machine could not catch up with the pace of computation requirements. Very often, current scientific problems simply do not fit into the memory of a single machine, making parallel computation a necessity. This has always been true throughout the history of computing—problems just do not fit into memory. But we have hit the point where a lot of interesting and practical problems from different areas do not fit into memory of a single machine. This has made parallel computing a necessity.

Manuscript received November 17, 2003; revised October 15, 2004. This work was supported in part by the Singapore-MIT Alliance.

The authors are with the Computer Science AI Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139 USA (e-mail: cly@mit.edu; edelman@math.mit.edu).

Digital Object Identifier 10.1109/JPROC.2004.840490

Table 1
CiteSeer and Google Search for Three Popular Math Packages,
Measured on April 7, 2004

	CiteSeer	Google
MATLAB	2961	1860000
Maple	1678	Common word
Mathematica	1710	1340000

- 2) *Granularity*: Over the past eight years simple parallel MATLAB projects, some consisting of only 2 m-files, have shown that multiple MATLAB instances running on a parallel computer could be used to solve embarrassingly parallel problems, without any change to MATLAB itself. Also, increase in problem sizes and processor speed have reduced the portion of time spent in noncomputation related routines.
- 3) *Business situation*: The past few years have seen the proliferation of Beowulf clusters. Beowulf clusters are parallel computers made from commodity off-the-shelf (COTS) hardware. They often consist of workstations connected together with Ethernet or other common, nonproprietary interconnect. Researchers prefer Beowulf clusters over traditional supercomputers because Beowulfs are quite easy to set up and maintain and are cheap enough so that a researcher can get his or her “personal supercomputer.” Also, in the financial industry, large “compute farms” which are basically large clusters are widely used for various pricing and value at risk (VaR) simulations. However, often researchers in science wanting to use a parallel computer to solve problems are not experts in parallel programming. The dominant way of parallel programming, Message Passing Interface (MPI) [1], is too low level and too error prone. MATLAB is well known for its user-friendliness. There is a huge potential market for a MATLAB that could be used to program parallel computers.

III. MATLAB, MAPLE, AND MATHEMATICA

Besides MATLAB, there are two other very popular technical computing environments. First there is Maple, developed by Maplesoft. Then there is Mathematica, developed by Wolfram Research.

We are interested in looking at parallel MATLAB mainly for the following reasons.

- 1) *MATLAB is popular*: Compared to Maple and Mathematica, we feel a parallel MATLAB would reach a wider audience. MATLAB has seen extensive use in classrooms at the Massachusetts Institute of Technology (MIT), Cambridge, and worldwide. To get a rough idea of the popularity of the three software packages, we did a search on CiteSeer [10] for the number of citations and Google for the number of page hits, as shown in Table 1.
- 2) *MATLAB is user friendly*: From our experience of using the three packages, we feel that MATLAB has the best user interface. Its C/Fortran 90-like scripting language is the most intuitive among the three for

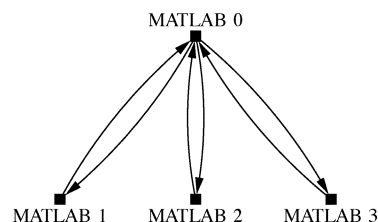


Fig. 1. Embarrassingly Parallel.

numerical parallel computing applications. This is in some sense a matter of taste, but the proliferation of MATLAB in classrooms seem to at least partially substantiate this claim.

IV. PARALLEL MATLAB SURVEY

The popularity of MATLAB and the fact that it could only utilize one processor sparked a lot of interest in creating a parallel MATLAB. We have done a survey [16] and found through extensive Web searching 27 parallel MATLAB projects. These projects vary in their scope: some are one-man projects that provide basic embarrassingly parallel capabilities to MATLAB; some are university or government lab research projects; while some are commercial projects that enables the use of MATLAB in product development. Also, their approaches to making MATLAB parallel are different: some compile MATLAB scripts into parallel native code; some provide a parallel back end to MATLAB, using MATLAB as a graphical front end; and some others coordinate multiple MATLAB processes to work in parallel. These projects also vary widely in their status: some are now defunct and exist only in the Google Web cache, while some are entering their second or third revision.

For each approach we try to include descriptions for some representative projects.

A. Embarrassingly Parallel

See Fig. 1. Software that makes use of this approach: Multi [33], Paralize [2], PLab [32], and Parmatlab [5]

This approach makes use of multiple MATLAB processes running on different machines or a single machine with multiple processors. However no coordination between the MATLAB processes is provided. Instead, a parent process passes off data to the child processes. Then all processes work on its local data and return the result to the parent process.

Under this model, the type of functions that can be parallelized is limited. For example, a for-loop with any data dependency across iterations would be impossible to parallelize under this model. However, this model has the advantage of simplicity. In the software we found that utilize this approach, usually no change in existing code is needed to parallelize the code, if the code is parallelizable using this simple approach. From our experience, this approach is sufficient for a lot of real world applications. For example, Seti@Home belongs to this category.

1) *PLab*: PLab is one of the most sophisticated embarrassingly parallel MATLABs, consisting of 15 m-files and 26

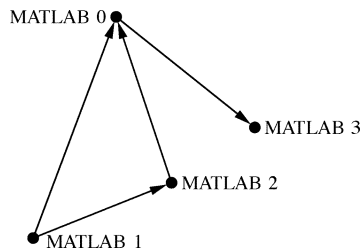


Fig. 2. Message Passing.

C/C++ files. It runs task in parallel on a “MATLAB Parallel Machine,” created either locally through *fork()* or remotely through *rsh*, *ssh*, *rexec*, or a custom Beowulf Job Manager. In local mode MATLAB workspace variables are automatically transferred to the slave processes through the *fork()*. In the remote case, they have to be transferred over sockets. PLab also supports integration with MOSIX [8] and can migrate slave processes to nodes with smaller load.

PLab supports two modes of parallelization. *pfor* emulates a parallel for-loop. For example

```
t = -5 : 0.01 : 5; aa = pfor(t, 'r = sin(t)', 'r');
```

is equivalent to the for-loop

```
ii = 1;
for t = -5 : 0.01 : 5,
    a = sin(t);
    aa(ii) = a;
    ii = ii + 1;
end
```

run in parallel. Two scheduling methods are built into PLab for *pfor*. In the *fixed* method, the server hands out fixed number of loop iterations in chunks to each slave. In the *dynamic* method, the number of iterations passed to each slave at a time varies dynamically trying to make each chunk take 1 second of computing time.

The second mode, *peval*, is the typical embarrassingly parallel *eval* function.

B. Message Passing

See Fig. 2. Software that makes use of this approach: MultiMATLAB [35], CMTM [47], DPToolbox [39], MPITB/PVMTB [22], PMI [34], PTToolbox [26], MatlabMPI [30], pMatlab [31], and Matlab Parallelization Toolkit [25]

This approach provides message passing routines between MATLAB processes. This enables users to write their own parallel programs in MATLAB in a fashion similar to writing parallel programs with a compiled language using MPI. In fact, for some of the projects [22], [30], the routines provided are wrappers for MPI routines. This approach has the advantage of flexibility: in theory, users are able to build any parallel system in MATLAB that they can build in compiled

languages with MPI. This approach is a superset of the embarrassingly parallel approach.

1) *MultiMATLAB*: MultiMATLAB is perhaps one of the most well known parallel MATLABs. It was developed by A. Trefethen, V. Menon, C. Chang, G. Czajkowski, C. Myers, and L. Trefethen at Cornell University. In MultiMATLAB, a MATLAB user can start MATLAB processes on other machines and then pass commands and data between these various processes. The most important commands are:

- *Init*—initialize the MultiMATLAB environment;
- *Nproc*—returns the number of processes in the environment; equivalent of MPI Comm size();
- *ID*—returns the rank of the process; equivalent of MPI Comm rank();
- *Send/Recv*—sends and receives data between processes;
- *Eval*—evaluates a string in every process in the environment.

This is essentially the MPI “Basic Six” without MPI *Finalize()*, and with *Eval* added. The *Send* and *Recv* commands in MultiMATLAB are blocking, so one is not able to take advantage of pipelining.

MultiMATLAB is built upon the P4 device of MPICH [24], an implementation of MPI developed at Argonne National Laboratory and Mississippi State University. Because of that, it runs over a heterogeneous network of workstations.

C. Back-End Support

Software that makes use of this approach: NetSolve [13], DLab [38], Matpar [42], PLAPACK [45], PARAMAT [43], and MATLAB * P [27]

This approach uses MATLAB as a front end for a parallel computation engine. Computation is done on the engine, usually making use of high-performance numerical computation libraries like ScaLAPACK [9]. For some projects e.g. [13], the data reside in MATLAB and are passed to the engine and back. And for some projects, e.g., [27], the data reside on the server and are passed back to MATLAB only upon request. The latter approach has the advantage that there is less data traffic. This is important for performance when data sets are large.

The advantage of this approach is that it only requires one MATLAB session (therefore, only one license), and it usually does not require the end user to have knowledge of parallel programming.

1) *NetSolve*: NetSolve is actually much more than a parallel MATLAB. It is a project at University of Tennessee, Knoxville, “that aims to bring together disparate computational resources connected by computer networks” (from their Web page). An interface to the NetSolve system exists for MATLAB. The user can submit jobs to the system, and then NetSolve will search for computational resources on a network and choose the best one for the task. It supports fault-tolerance and load-balancing. In some sense, it is a very user-friendly batch system.

Jobs can be submitted as

```
a = randn(1000); [x y] = netsolve('eig', a);
```

This creates a 1000×1000 matrix *locally*, makes the call to `netsolve`, and blocks. When the appropriate computational resource is found, the matrix is transferred over the network and the server performs `eig` on the matrix. The result is then passed back to the client, in this case, MATLAB. There exists a nonblocking version of the `netsolve` call, *netsolve nb*.

Netsolve comes with an interface to

- dense linear algebra—ScaLAPACK [9];
- sparse iterative solver—PetSc [6], Aztec [44];
- sparse direct solver—SuperLU [17], MA28 [21].

D. MATLAB Compilers

Software that makes use of this approach: Otter [40], RT-Express [28], ParAL [36], FALCON [18], CONLAB [20], MATCH [7], and Menhir [14]

These projects compile MATLAB scripts into an executable, sometimes translating the scripts into a compiled language as an intermediate step. Some projects, e.g., [40] and [20], link the compiled code with parallel numerical libraries, while some software, e.g., [18], generates code that is already parallel.

This approach has the advantage that the compiled code runs without the overhead incurred by MATLAB. Also an executable is usually easier to manage if it is to be run non-interactively regularly. In this approach, MATLAB is used as a development platform instead of a computing environment. This allows the produced parallel program to run on platforms which does not support MATLAB (e.g. SGI).

E. Shared Memory

Software that make use of this approach: MATmarks [3]

MATmarks, the only project in this category, allows multiple running instances of MATLAB to share the value of variables. It is based on Treadmarks [4], a distributed shared memory system developed at Rice University.

A new project has been started at MIT to build shared variables support into MATLAB * P, using GASNet [11].

F. Parallel MATLABs

The Parallel MATLABs are defined in Tables 2–6.

V. PARALLEL MATLAB FEATURES

In evaluating the 27 parallel MATLAB projects, we found that some contain features that are innovative in the parallel MATLAB world.

A. File-System-Based Communication

Some parallel MATLAB projects utilize a file-system-based communication system. Machines in a Beowulf cluster often share a common file system, e.g., Network File System (NFS), and it can be exploited for communication. One of the earliest parallel MATLAB projects to make use of this is Paralize [2] (1998). Sends and receives are handled through writing to and reading from files, and synchronization is done through checking for the existence of certain

Table 2
Message Passing

Matlab Parallelization Toolkit [25]	Linköpings Universitet, Sweden
MultiMATLAB [35]	Cornell University
CMTM [47]	Cornell University
DP-Toolbox [39]	Universität Rostock, Germany
MPITB/PVMTB [22]	University of Granada, Spain
PMI [34]	Lucent Technologies
PT Toolbox [26]	Wake Forest University
MatlabMPI [30]	MIT Lincoln Lab
pMatlab [31]	MIT Lincoln Lab

Table 3
Embarrassingly Parallel

MULTI Toolbox [33]	Purdue University
Paralize [2]	Chalmers University of Technology, Sweden
PLab [32]	Technical University of Denmark
Parmatlab [5]	Northeastern University

Table 4
Back-End Support

Netsolve [13]	University of Tennessee, Knoxville
DLab [38]	UIUC
Matpar [42]	Jet Propulsion Lab
PLAPACK [45]	University of Texas at Austin
Paramat [43]	Alpha Data Parallel Systems
MATLAB*P [27]	MIT

Table 5
MATLAB Compiler

Otter [40]	Oregon State University
RTExpress [28]	Integrated Sensors Inc.
ParAL [36]	University of Sydney, Australia
FALCON [18]	UIUC
CONLAB [20]	University of Umea, Sweden
MATCH [7]	Northwestern University
Menhir [14]	IRISA, France

Table 6
Shared Memory

MATmarks [3]	UIUC
--------------	------

lock files. MatlabMPI [30] implements basic MPI functions in MATLAB using cross-mounted directories. Bandwidth comparable to C-based MPI is reported, although the latency is inferior.

B. M-File Implementation

Simple parallel MATLAB approaches like the embarrassingly parallel approach or MPI could be implemented with only MATLAB m-files. One of the earliest parallel MATLAB to be implemented this way is Paralize [2] (1998), a parallel MATLAB using the embarrassingly parallel approach and implementing dynamic load balancing in only 120 lines of MATLAB code. Pure m-file implementation has the advantage of being portable to any platform on which MATLAB runs. Also, it makes installation simple and user-friendly—no compilation is needed. pMATLAB

[31] is trying to take this approach further by implementing global structures like distributed matrices with only m-files.

C. Parallelism Through Polymorphism

Parallel MATLABs often introduce new commands into MATLAB—for example, a *pfor* function for parallel for-loops or a set of MPI-like functions for message passing operations. The problem with this approach is that it does not reduce the complexity of writing a parallel program. MPI is not the level general users want to or should want to program at.

Since the number of functions in MATLAB is finite and most, if not all, of them depends on a set of built-in functions, it is sufficient to parallelize those built-in functions in MATLAB through overloading. Users can call the parallelized built-in functions directly or call functions which depend on those parallel built-in functions. Toolboxes can also be parallelized in this way, by parallelizing the underlying built-in functions. This is introduced in MATLAB * P [27] seamlessly in the concept of *parallelism through polymorphism*. MATLAB * P is a parallel MATLAB using the back-end support approach, attaching a parallel back-end server to MATLAB. This will be explained further in a later section.

D. Lazy Evaluation

Back-end support parallel MATLABs utilize a server to perform the computation. While the server is busy computing, the front-end MATLAB often idles, wasting precious cycles. To remedy this problem, DLab [38] introduces a concept called *lazy evaluation*. When a command is sent to the server, the MATLAB program does not block and wait for the result. Instead, the MATLAB program only blocks when it tries to make a server call again. This way, additional computations could be done in the time between the server call and the next, which is wasteful in other back-end support parallel MATLABs.

VI. WHAT DOES THE USER WANT IN A PARALLEL MATLAB?

A. MATLAB Experience

We could reflect on the experience of MATLAB to understand what a user would want in a parallel MATLAB.

MATLAB started out as an interactive interface to EISPACK and LINPACK, a set of eigenvalue and linear system solution routines. EISPACK and LINPACK were written in Fortran, so normally a user would have to write a program in Fortran to use the routines. That way the user will have to take care of the memory allocation, indexing, and studying the (quirky) EISPACK/LINPACK syntaxes. Furthermore, there is no way to visualize the result without writing your own code.

But MATLAB took care of all of this. User calls create matrices and operate on them using simple, intuitive calls. Visualization could also be done in the same framework. Furthermore, the scripting language in MATLAB contains features from Fortran 90, making it easier to use than Fortran and more suitable for matrix computation than C.

All this convenience came at a cost of performance. The graphical user interface, parser, interpreter, and the lack of type inference take away processor cycles which could be used for computation. Also, the interpreted scripts could not match the performance of compiled C or Fortran code. Yet still MATLAB is a huge success. We can see that what the users really want is ease of use, not peak performance. Users prefer a system that is easy to use and has good performance over a system with peak performance but is hard to use and clumsy. The gain in performance in the latter system is easily offset by the additional time needed to program it.

B. Modes of Parallel Computation

Over the years we have seen many parallel programs from various application areas: e.g., signal processing, graphics, artificial intelligence, computational mathematics, and climate modeling. From the code we have seen, we divide parallel computation in general into four categories.

- 1) *A lot of small problems that require no communication.* Also known as embarrassingly parallel problems. The problem size is small enough so that it will fit into the memory of one machine, but there are a lot of them, so parallel computation is needed. No communication is required between the parallel threads of computation, except for an initial scatter—to distribute the computation, and a final gather—to collect the results. An example of this type of computation would be Monte Carlo simulations, currently heavily used in the finance industry for risk management and product pricing. From anecdotal evidence this seems to be the largest class of problems, partly due to the fact that no easy-to-use tools exists for the two other classes.
- 2) *A lot of small problems that require some communication.* Just like the first case, the problem size is small enough to fit into the memory of one machine. However, in this category it is necessary to communicate between the parallel threads during the computation.
- 3) *Large problems.* This class of problems has a problem size that would not fit into the memory of a single machine. Example: the high-performance Linpack (HPL) benchmark
- 4) *A mixture of the three.* In this class of problems, the data is sometimes processed individually in embarrassingly parallel mode, while sometimes it is treated as a global data structure. We have seen an example of this in climate modeling.

A good parallel MATLAB should address at least one of these areas well.

C. “Right” Parallel MATLAB

When building the “right” parallel MATLAB targeting the widest possible audience (and, thus, would be most commercially viable), we should take the above arguments into account. First, the interface in the parallel MATLAB should be easy to use, should not differ much from ordinary MATLAB, and should not require learning on the users’ part. Second,

it should allow the four modes of parallel computation described above.

VII. MATLAB * P

We present MATLAB*P [15] as an example of what could be a “right” parallel MATLAB. MATLAB * P is a parallel MATLAB using the back-end support approach, aimed at widespread circulation among a general audience. In order to achieve this, we took ideas from the approaches used by other software found in the survey. For example, the *embarrassingly parallel* approach allow simple, yet useful, division of work into multiple MATLAB sessions. The *message passing* approach, which is a superset of the *embarrassingly parallel* approach, allows finer control between the MATLAB sessions.

The main idea in MATLAB*P is that data exist on the parallel server as distributed matrices. Any operations on a distributed matrix (which exists in the MATLAB front end only as a handle) will be relayed to the server transparently. The server calls the appropriate routine from a parallel numerical library (e.g. ScaLAPACK, FFTW, . . .) and the results stay on the server until explicitly requested.

The “transparency” comes from the use of polymorphism in MATLAB. This will be explained in the next section.

MATLAB * P has progressed a lot since the original version by Husbands [27]. New developments include improved sparse matrix support, support for complex numbers, garbage collection for distributed objects, the “Multi-MATLAB” mode, and visualization routines.

VIII. FEATURES OF MATLAB * P

A. Parallelism Through Polymorphism—*p

The key to the parallelism lies in the *p* variable. It is an object of *dlayout* class in MATLAB, defined by the MATLAB * P system. By overloading MATLAB functions for the class *dlayout*, we were able to create a parallel MATLAB that has exactly the same interface as MATLAB.

Through the use of the *p* variable, matrices that are distributed on the server could be created. For example

```
X = randn(8192 * p, 8192);
```

The above creates a row-distributed, 8192×8192 normally distributed random matrix on the server. $8192 * p$ multiplies the MATLAB double *8192* with the *dlayout* instance *p*. Multiplication for *dlayout* instances is overloaded to embed the 8192, which is the size information, within the *dlayout* that will be created. *randn* with arguments that are of class *dlayout* is overloaded to make calls to the server to create the distributed matrix.

X is a handle to the distributed matrix, identified by MATLAB as a *ddense* class object. By overloading *randn* and many other built-in functions in MATLAB, we are able to tie in the parallel support transparent to the user. This is called *parallelism through polymorphism*. Note that the

syntax is exactly the same as ordinary MATLAB except for the additional multiplication with the *p* variable

```
e = eig(X);
```

The command computes the eigenvalues of X by calling the appropriate ScaLAPACK routines and stores the result in a matrix *e*, which resides on the server. The result is not returned to the client unless explicitly requested, to reduce data traffic. Again the syntax is the same as ordinary MATLAB

```
E = pp2matlab(e);
```

This command returns the result to MATLAB. This is one of the few commands in MATLAB * P not found in ordinary MATLAB (*matlab2pp* is another).

The use of the *p* variable along with overloaded MATLAB routines enable existing MATLAB scripts to be reused. For example

```
function H = hilb(n)
J = 1 : n;
J = J(ones(n, 1), :);
I = J';
E = ones(n, n);
H = E./(I + J - 1);
```

The above is the built-in MATLAB routine to construct a Hilbert matrix (obtained through *type hilb*). Because the operators in the routine (colon, ones, subsasgn, transpose, rdivide, +, -) are overloaded to work with *p*, typing

```
H = hilb(16384 * p)
```

would create a 16384-by-16384 Hilbert matrix on the server. By exploiting MATLAB’s object-oriented features in this way, many existing scripts would run in parallel under MATLAB * P without any modification.

B. MultiMATLAB/MultiOctave Mode

One of the goals of the project is to make the software to be useful to as wide an audience as possible. In order to achieve this, we found that it would be fruitful to combine other parallel MATLAB approaches into MATLAB * P, to provide a unified parallel MATLAB framework.

In conjunction with P. Husbands, we developed a prototype implementation of a MultiMATLAB [35]-like, distributed MATLAB package in MATLAB * P, which we call the PEngine. With this package and associated m-files, we can run multiple MATLAB processes on the back end and evaluate MATLAB functions in parallel on dense matrices.

The system works by starting up MATLAB engine instances on each node through calls to the MATLAB engine interface. From that point on, MATLAB commands can be relayed to the MATLAB engine. The command to do this is *mm*.

Examples of the usage of *mm* follow.

```

>> % Example 1
>> a = 1 : 100 * p;
>> b = mm('chi2rnd', a);

```

The first example creates a distributed matrix of length 100, then fills it with random values from the chi-square distribution through calls to the function `chi2rnd` from the MATLAB statistics toolbox.

```

>> % Example 2
>> a = rand(100 100 * p);
>> b = rand(100 100 * p);
>> c = mm('plus', a, b);

```

This example creates two column distributed matrices of size 100×100 , adds them, and puts the result in another matrix. This is the slow way of doing the equivalent of

```

>> a = rand(100 100 * p);
>> b = rand(100 100 * p);
>> c = a + b;

```

The “MultiOctave” mode works exactly the same as “MultiMATLAB” mode, only using Octave, a freely available MATLAB-like scientific computing software, for the computation. The command to use “MultiOctave” mode is `mo`.

```

>> % Example 3
>> a = randn(4, 4 * p);
>> b = mm('sin', a)
>> c = mo('sin', a)
>> norm(b - c)
ans =
    6.7820e - 07

```

The above interesting example shows that Octave and MATLAB use a different algorithm for the sine function. Octave serves the purpose of an embarrassingly parallel back end very well, because although its graphical capabilities are not as good as MATLAB, its numerical parts are up to par with MATLAB. As a back-end engine, we are mostly concerned with numerical performance.

```

>> % Example 4
>> a = (0 : (np - 1) * p) / np;
>> b = a + (1 / np);
>> mypi = mm('quad', '4./(1 + x.^2)', a, b);
>> disp ('Pi from quad in mm mode')
>> pi_from_quad = sum(mypi)
Pi from quad in mm mode
pi_from_quad =
    3.1416

```

The above example illustrates how `np`, the variable that returns the number of processes running on the back-end server, can be used in a script to write adaptive code. When the above example is run on four processes, `a` is `0:0.25:0.75`, and `b` is `0.25:0.25:1`. In the “MultiMATLAB” call, each slave MATLAB will compute the adaptive Simpson quadrature of

$$\frac{4}{1+x^2}$$

in the intervals $(0,0.25)$, $(0.25,0.50)$, $(0.50,0.75)$, $(0.75,1.0)$, respectively. The result from each slave MATLAB is summed to form π .

C. Distributed File I/O

When the user wants to handle data that is too large to fit in the client MATLAB’s memory, the mechanism of `matlab2pp/pp2matlab` can no longer be used, as those calls involve moving data to client MATLAB. In this situation, we provide two routines, `dload` and `dsave`, which stands for distributed load/save. Currently the only format supported is the MATLAB MAT file format. Users can choose to load/save a single file, which will then be distributed and become the distributed matrix, or can choose to load/save a collection of files, which represent the local “pieces” of a distributed matrix.

D. Visualization Package

This visualization package was written by Bruening, Holloway, and Sulejmanpasic, under supervision of R. Choy, as a term project for the class 6.338/18.337—Applied Parallel Computing at MIT. It has since then been merged into the main MATLAB * P source.

This package adds `spy`, `surf`, and `mesh` routines to MATLAB * P. This enable visualization of very large matrices. The rendering is done in parallel, using the Mesa OpenGL library.

Figs. 3, 4, and 5 shows the routines in action.

All three routines allow zooming into a portion of the matrix. Also, `ppsurf` and `ppmesh` allow changing of the camera angle, just as supported in MATLAB.

IX. BENCHMARKS

A. Test Platform

- Beowulf cluster with 15 nodes.
- Dual-processor nodes, each with two Intel Xeon 2.4 GHz with Hyperthreading.
- All tests are run with one process per processor. Also, the minimum number of nodes are used. So if a test is using four processors, two nodes are used. Thus, in this case we are using two shared-memory pairs of processors.
- Two gigabytes of RAM per node. No swapping occurred during the tests.
- Fast Ethernet (100 Mb/s) interconnect. Intel Etherfast 410T switch.

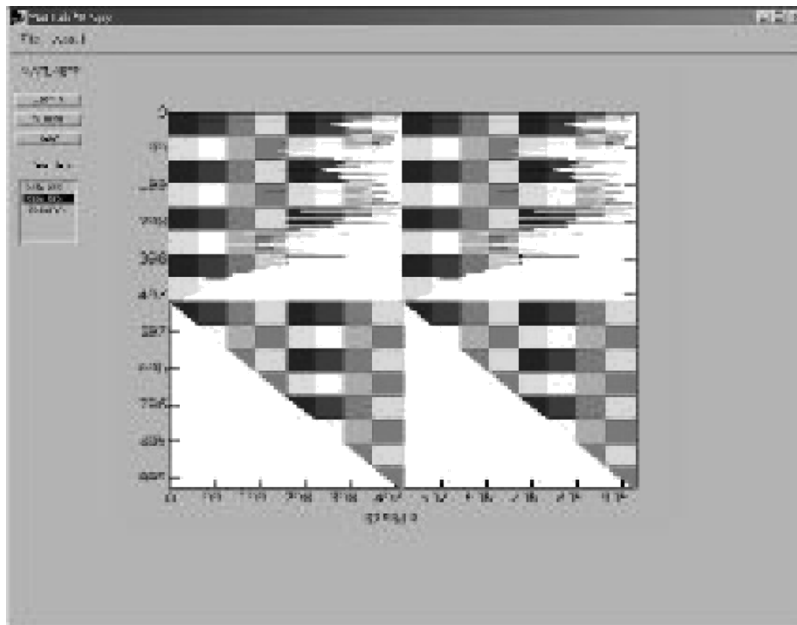


Fig. 3. ppspy on a distributed 1024×1024 matrix on eight nodes.

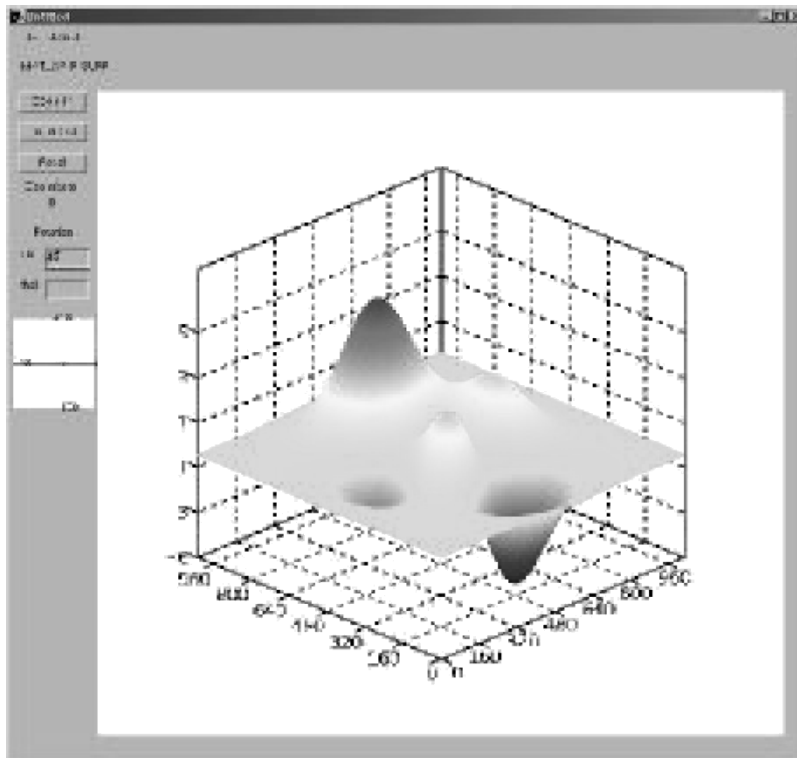


Fig. 4. ppsurf on the distributed 1024×1024 "peaks" matrix.

- Linux kernel version: 2.4.20-9 smp.
- MATLAB 6.1.0 R12.1.

B. Linear System Solve

We compare the time required by MATLAB and MATLAB * P to solve a linear system of equations with one right-hand side. The one processor case in the graph is serial MATLAB, while the cases with more than one processor are results from MATLAB * P.

For $n = 4000$ and $n = 8000$, as we go from serial MATLAB to two processors in MATLAB * P, we notice superlinear speedup. The memory available to MATLAB and MATLAB * P in this case are the same, as the two processors come from a single node in the cluster. Thus, we can eliminate swapping as the cause of this behavior, as the amount of memory available in the serial case and the two processors case is the same. Instead, this is most likely caused by more cache memory being available. Each Xeon processor

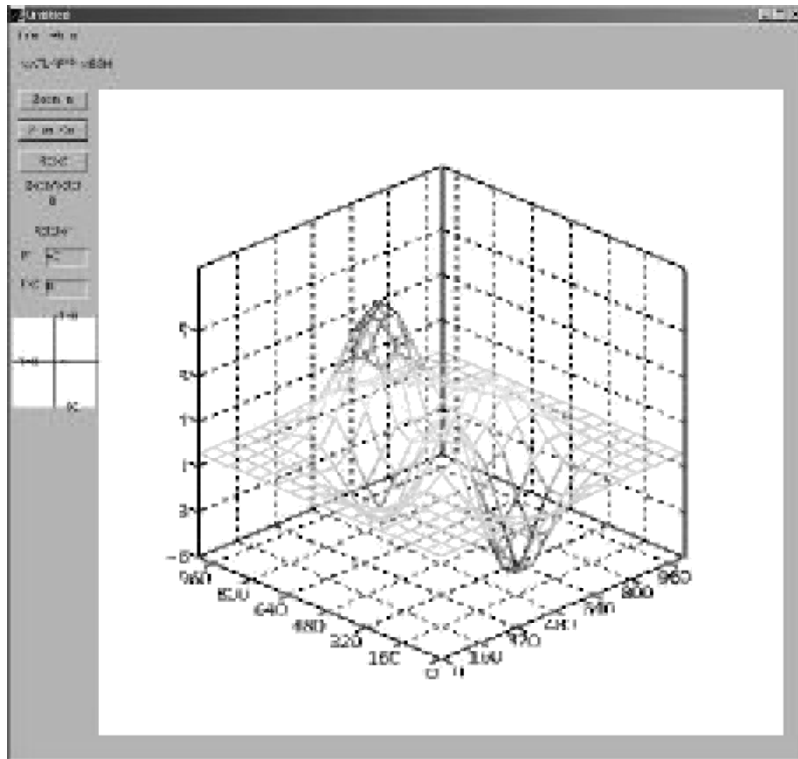


Fig. 5. ppmesh on the distributed 1024×1024 “peaks” matrix.

has 512 KB of L2 cache, so there is a total of 1 MB of cache memory available to the MATLAB * P computation.

Then for each problem size, as we increase the number of processors used, we observe that the amount of time needed either increases or decreases by very little. This is due to granularity—as we break the problem into smaller and smaller pieces, any performance gain is overwhelmed by the increase in communication cost.

As MATLAB * P uses the PDGESV routine from ScaLAPACK, it would be interesting to compare the overhead involved.

As we can see, MATLAB*P is always slower than the bare ScaLAPACK PDGESV call. The biggest overhead is caused by the fact that when the user calls

$$C = A \setminus B;$$

in MATLAB, he would expect A and B to be unchanged. This is not true for the ScaLAPACK PDGESV routine, which uses A to return the LU factors of the input A and B to return the solution. To preserve the MATLAB call property, MATLAB*P has to create copies of the inputs to preserve them. The time needed for this is listed in Table 7.

This table can explain the poor performance of MATLAB*P versus ScaLAPACK for $n = 2000$. It appears the input copying time depends on a function of number processor plus a function of the input size. As $n = 2000$ is very small, the first function dominates. Also, the time needed by PDGESV is very small. Thus, the ratio of MATLAB * P solution time, which is PDGESV time added by the copying cost, to the PDGESV time is large.

Table 7
Breakdown of Time Taken by MATLAB * P Solve

	Copying	PDGESV
np=2, n=2000	0.5628	1.6344
np=2, n=4000	0.7863	11.0234
np=2, n=8000	1.9294	69.1374
np=4, n=2000	0.8060	2.6315
np=4, n=4000	0.8780	13.7856
np=4, n=8000	1.5254	73.5604
np=8, n=2000	1.0059	3.3152
np=8, n=4000	1.0188	14.3020
np=8, n=8000	1.3469	70.9407

Figs. 6 and 7 show how the MATLAB * P linear system solve scale with the number of processors and compare the performance with ScaLAPACK.

C. RT-STAP

The Real-Time Space-Time Adaptive Processing (RT-STAP) benchmark [12] is a benchmark for real-time signal processing systems developed by MITRE Corporation. In the hard version of the benchmark that we are testing, the input to the MATLAB code is a data cube of 22 (channels) $\times 64 \times 480$ doubles. The code performs the following three steps:

- 1) conversion of the input data to baseband;
- 2) Doppler processing;
- 3) weight computation and application to find the range-Doppler matrix.

Upon running the code in serial MATLAB, we notice that step 1, the conversion of the input data to baseband, is the most time-consuming step. This is surprising, as the weight computation should be the step that has the highest

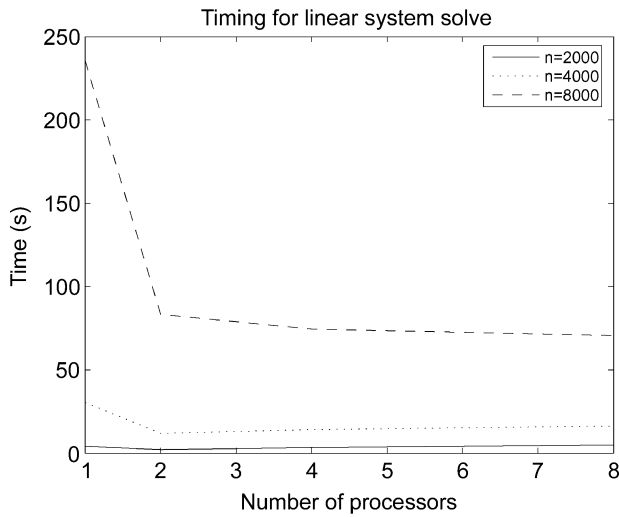


Fig. 6. Timing for linear system solve.

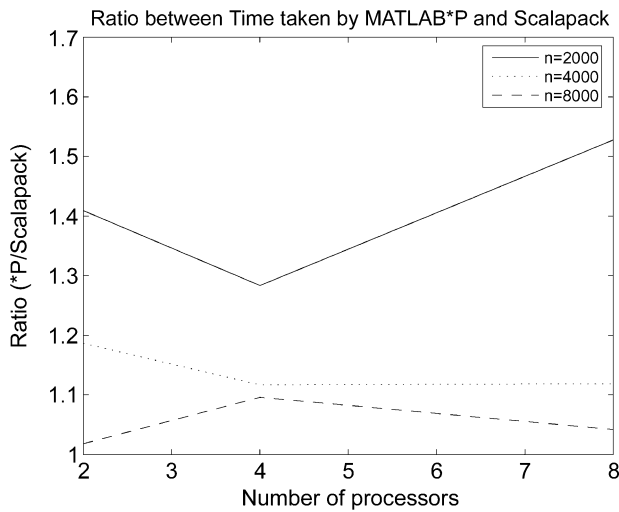


Fig. 7. Ratio of time taken by MATLAB * P and ScaLAPACK.

flop count. It turns out that this is caused by problematic MATLAB coding in the benchmark MATLAB code. But as the point of this is to time MATLAB * P on a typical application, we chose to proceed without modifying the code. The conversion step in the original MATLAB code is the following loop:

```

for channum = 1:NCHAN
    xx = CPI1_INITIAL(:,channum);
    CPI1(:,channum) = baseband_convert(...
xx, NRNG, NPRItotal, MM, w_demod,
w_LPT, ...
NTAPS_LPF, LPFdecFlag);
end

```

It loops over the input channels and process them in an embarrassingly parallel fashion. It is a natural candidate for mm mode, and we converted the loop to run in MATLAB * P by changing the loop into a function call and putting in an mm mode call

```
P_CPI1_INITIAL = mat-
```

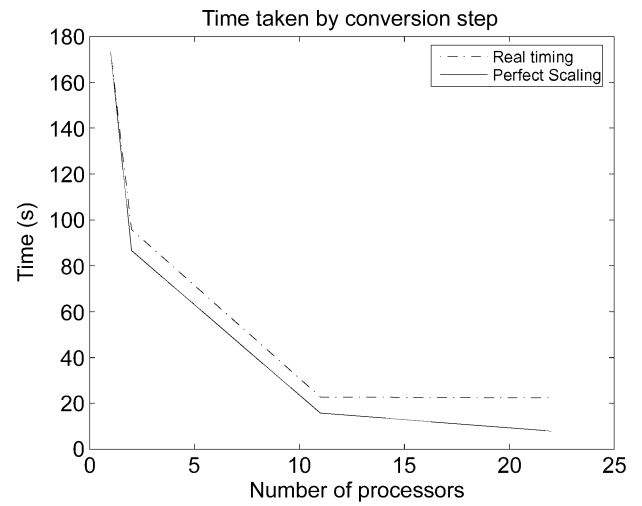


Fig. 8. Time taken by RT STAP conversion step.

```

lab2pp (CPI1_INITIAL, 2);
CPI1 = mm('convert_loop',...
P_CPI1_INITIAL, NRNG, NPRItotal, MM,
...
w_demod, w_LPF, NTAPS_LPF,
LPFdecFlag);
CPI1 = CPI1(:,:);

```

Note that the calls before and after the mm call are used to transfer data to the server and back. Time used by those calls are included in our timings. We compared the timing results of serial MATLAB and MATLAB * P on 2, 11, and 22 processors.

The solid line shows the timings that would be obtain if the code scales perfectly, i.e., when the number of processors is doubled, the amount of time needed is halved. The real timings follow the solid line quite closely except for the 22 processors case. Going from 11 processors to 22 processors provide no additional benefits. This is easy to understand in terms of granularity. As the input data cube has only 22 channels, in the 22 processors case each processor has only one channel of work, versus two channels in the 11 processors case. So there is very little to gain from using additional processors, and any benefit is overcome by the additional time needed for communication.

Fig. 8 shows the scalability of the RT-STAP benchmark under MATLAB * P.

X. CONCLUSION

MATLAB * P shows how a parallel MATLAB system can address different needs of parallel computing. Back-end calls to ScaLAPACK handles large problems in parallel. The MultiMATLAB/MultiOctave mode takes care of problems that are embarrassingly parallel in nature and collects the result in a distributed matrix so that global operations can be performed on the result. Last but not least, a parallel MATLAB system has to be user-friendly in order to be useful. MATLAB * P achieves this through transparent parallelization with *parallelism through polymorphism*.

The system has been used to solve dense linear system of size 100 000 × 100 000 and two-dimensional FFT of size

64 000 × 64 000 with success. It has been used for biomedical imaging and climate modeling applications, as well as a teaching tool in MIT.

REFERENCES

- [1] *MPI: The Complete Reference*, 2nd ed. Cambridge, MA: MIT Press, 1998.
- [2] T. Abrahamsson. (1998) Paralyze. [Online]. Available: <ftp://ftp.mathworks.com/pub/contrib/v5/tools/paralyze/>
- [3] G. Almasi, C. Cascaval, and D. A. Padua, "Matmarks: a shared memory environment for MATLAB programming," presented at the IEEE Int. Symp. High Performance Distributed Computing Symp., Redondo Beach, CA, 1999.
- [4] C. Amza, A. L. Cox, S. Dworkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel, "Treadmarks: Shared memory computing on networks of workstations," *IEEE Computer*, vol. 29, no. 2, pp. 18–28, Feb. 1996.
- [5] L. Andrade. (2001) Parmatlab. [Online]. Available: <ftp://ftp.mathworks.com/pub/contrib/v5/tools/parmatlab/>
- [6] S. Balay, W. D. Gropp, L. C. McInnes, and B. F. Smith, "PETSc 2.0 users manual," Argonne Nat. Lab., ANL-95/11, Revision 2.0.29, 2000.
- [7] P. Banerjee and U. N. Shenoy *et al.*, "A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems," in *Proc. IEEE Symp. Field-Programmable Custom Computing Machines*, 2000, pp. 39–48.
- [8] A. Barak and O. La'adan, "The MOSIX multicomputer operating system for high performance cluster computing," *J. Future Generat. Comput. Syst.*, vol. 13, no. 4–5, pp. 361–372, Mar. 1998.
- [9] L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley, *ScaLAPACK Users' Guide*. Philadelphia, PA: SIAM, 1997.
- [10] K. Bollacker, S. Lawrence, and C. Lee Giles, "A system for automatic personalized tracking of scientific literature on the Web," in *Proc. Digital Libraries 99—4th ACM Conf. Digital Libraries*, pp. 105–113.
- [11] D. Bonachea, "GASNet Specification, v1.1," Univ. California, Berkeley, Tech. Rep. UCB/CSD-02-1207, 2002.
- [12] K. C. Cain, J. A. Torres, and R. T. Williams, "RT STAP: Real Time Space-Time Adaptive Processing benchmark," MITRE Corp., Tech. Rep. MTR 96B0000021, Feb. 1997.
- [13] H. Casanova and J. Dongarra, "NetSolve: a network-enabled server for solving computational science problems," *Int. J. Supercomput. Appl. High Perform. Comput.*, no. 3, pp. 212–223, Fall 1997.
- [14] S. Chauveau and F. Bodin, "Menhir: an environment for high performance MATLAB," in *Languages, Compilers, and Run-Time Systems for Scalable Computers*. New York: Springer-Verlag, 1998, pp. 27–40.
- [15] R. Choy, "MATLAB* p 2.0: Interactive supercomputing made practical," M.Sc. thesis, Massachusetts Inst. Technol., Cambridge, 2002.
- [16] R. Choy. (2001) Parallel MATLAB survey. [Online]. Available: <http://theory.lcs.mit.edu/cly/survey.html>
- [17] J. W. Demmel, J. Gilbert, and X. S. Li, "SuperLU users' guide," Comput. Sci. Div., Univ. California, Berkeley, Tech. Rep. CSD-97-944, 1997.
- [18] L. DeRose, K. Gallivan, E. Gallopoulos, B. Marsolf, and D. Padua, "Falcon: A MATLAB interactive restructuring compiler," in *Lecture Notes in Computer Science, Languages and Compilers for Parallel Computing*. Heidelberg, Germany: Springer-Verlag, 1995, pp. 269–288.
- [19] J. J. Dongarra, J. R. Bunch, C. B. Moler, and G. W. Stewart, *LINPACK User's Guide*. Philadelphia, PA: SIAM, 1979.
- [20] P. Drakenberg, P. Jacobson, and B. Kågström, "A CONLAB compiler for a distributed memory multicomputer," in *Proc. 6th SIAM Conf. Parallel Processing for Scientific Computation*, vol. 2, 1993, pp. 814–821.
- [21] I. Duff, "MA28—A set of Fortran subroutines for sparse unsymmetric linear equations," HMSO, London, U.K., Rep. AERE R8730, 1977.
- [22] J. Fernández, A. Canas, A. F. Díaz, J. González, J. Ortega, and A. Prieto, "Performance of message-passing MATLAB toolboxes," presented at the High Performance Computing for Computational Science—VECPAR 2002: 5th Int. Conf., Porto, Portugal.
- [23] M. Frigo and S. Johnson, "FFTf: an adaptive software architecture for the FFT," in *Proc. Int. Conf. Acoustics, Speech and Signal Processing*, vol. 3, 1998, p. 1381.
- [24] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "High-performance, portable implementation of the MPI message passing interface standard," *Parallel Comput.*, vol. 22, no. 6, pp. 789–828, 1996.
- [25] E. Heiberg. (2003) MATLAB parallelization toolkit. [Online]. Available: <http://hem.passagen.se/cinar/heiberg/documentation.html>
- [26] J. Hollingsworth, K. Liu, and P. Pauca. (1996) Parallel toolbox for MATLAB Pt v. 1.00: Manual and reference pages. [Online]. Available: <http://www.mthsc.wfu.edu/pt/pt.html>
- [27] P. Husbands and C. Isbell, "MITMatlab: A tool for interactive supercomputing," presented at the 9th SIAM Conf. Parallel Processing for Scientific Computing, San Antonio, TX, 1999.
- [28] Integrated Sensors Inc.. RTExpress. [Online]. Available: <http://www.rtxpress.com/>
- [29] Mathworks Inc. (2001) MATLAB 6 user's guide. [Online]. Available: <http://www.mathworks.com>
- [30] J. Kepner, "Parallel programming with MatlabMPI," presented at the High Performance Embedded Computing (HPEC 2001) Workshop, Lexington, MA, 2001.
- [31] J. Kepner and N. Travinin, "Parallel MATLAB: The next generation," presented at the 7th High Performance Embedded Computing Workshop (HPEC 2003), Lexington, MA.
- [32] U. Kjems. (2000) PLab home page. [Online]. Available: <http://bond.imm.dtu.dk/plab/>
- [33] T. Krauss. COMMSIM and MULTI Toolbox home page. [Online]. Available: http://www.lapsi.eletrou.ufrgs.br/Disciplinas/ENG_ELETRICA/CAD-ENG/Matlab/CommSim/COMMSIM
- [34] D. D. Lee. (1999) PMI Toolbox home page. [Online]. Available: <ftp://ftp.mathworks.com/pub/contrib/v5/tools/PMI>
- [35] V. Menon and A. E. Trefethen, "MultiMATLAB: Integrating MATLAB with high performance parallel computing," presented at the Supercomputing '97, San Jose, CA.
- [36] I. Z. Milosavljevic and M. A. Jabri, "Automatic array alignment in parallel MATLAB scripts," presented at the 13th Int. Parallel Processing Symp. and 10th Symp. Parallel and Distributed Processing, San Juan, Puerto Rico, 1999.
- [37] C. Moler. (1995) Why there isn't a parallel MATLAB. [Online]. Available: <http://www.mathworks.com/company/newsletter/pdf/spr95cleve.pdf>
- [38] B. R. Norris, "An environment for interactive parallel numerical computing," Univ. Illinois, Urbana, IL, Tech. Rep. UIUCDCS-R-99-2123, 1999.
- [39] S. Pawletta, T. Pawletta, W. Drewelow, P. Duenow, and M. Suesse, "A MATLAB toolbox for distributed and parallel processing," presented at the Matlab Conf. '95, Cambridge, MA.
- [40] M. Quinn, A. Malishevsky, and N. Seclam, "Otter: Bridging the gap between MATLAB and ScaLAPACK," in *Proc. 7th IEEE Int. Symp. High Performance Distributed Computing*, 1998, pp. 114–123.
- [41] B. T. Smith, J. M. Boyle, J. J. Dongarra, B. S. Garbow, Y. Ilebe, V. C. Kelma, and C. B. Moler, *Matrix Eigensystem Routines—EISPACK Guide*, 2nd ed. New York: Springer-Verlag, 1976.
- [42] P. L. Springer, "Matpar: parallel extensions for MATLAB," in *Proc. Int. Conf. Parallel and Distributed Processing Techniques and Applications*, vol. 3, 1998, pp. 1191–1195.
- [43] Alpha Data Parallel Systems. (1999) Paramat. [Online]. Available: <http://www.alpha-data.com>
- [44] S. A. Hutchinson, J. N. Shadid, and R. S. Tuminaro, "Aztec User's Guide: Version 1.1," Sandia Nat. Labs., Albuquerque, NM, Tech. Rep. SAND95-1559, 1999.
- [45] R. A. van de Geijn, *Using PLAPACK: Parallel Linear Algebra Package*. Cambridge, MA: MIT Press, 1997.
- [46] R. Clint Whaley, A. Petitet, and J. J. Dongarra, "Automated empirical optimization of software and the ATLAS project," *Parallel Comput.*, vol. 27, no. 1–2, pp. 3–35, 2001.
- [47] J. Zollweg. (2001) Cornell Multitask Toolbox for MATLAB home page. [Online]. Available: <http://www.tc.cornell.edu/Services/Software/CMTM/>

Ron Choy, photograph and biography not available at the time of publication.

Alan Edelman, photograph and biography not available at the time of publication.