# Solving Multiple Classes of Problems in Parallel with MATLAB*P

Ron Choy, Alan Edelman
CSAIL,
Massachusetts Institute of Technology,
Cambridge, MA 02139
Email: cly@mit.edu, edelman@math.mit.edu

***Abstract*— MATLAB [7] 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. Mathworks, the company that develops MATLAB, currently does not provide a version of MATLAB that can utilize parallel computing [9]. This has led to academic and commercial efforts outside Mathworks to build a parallel MATLAB, using a variety of approaches.**

**MATLAB*P is a parallel MATLAB that focus on enhancing productivity by providing an easy to use parallel computing tool. Using syntaxes identical to regular MATLAB, it can be used to solve large scale algebraic problems as well as multiple small problems in parallel. This paper describes how the innovative combination of '*p mode' and 'MultiMATLAB/MultiOctave mode' in MATLAB*P can be used to solve a large range of real world problems.**

## I. Background

MATLAB [7] is one of the most widely used tools in scientific and technical computing. It started in 1970s as an interactive interface to EISPACK [10] and LINPACK [4], a set of eigenvalue and linear system solution routines. It has since grown to a feature rich product utilizing modern numerical libraries like ATLAS [12] and FFTW [5], 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 C and HPF, and the flexible matrix indexing makes it very suitable for programming matrix problems. It is a very useful tool to a wide audience. For example, it comes in very handy as a matrix calculator for simple problems, because of its easy-to-use command syntax. Its strong graphical capabilities makes it a good data analysis tool. Also researchers have been known to build very complex systems using MATLAB scripts. Compared to similar software like Maple and Mathematica, MATLAB is known to have a larger number of users. For example, at MIT, MATLAB is the dominant computing software used by the student body.

Because of its roots in serial numerical libraries, MATLAB has always been a serial program. However, as modern engineering and simulation problems become more and more complex, space and computing time requirements for solutions skyrocketed, and a serial MATLAB is no longer able to handle them. MATLAB*P [6] tries to solve the problem by providing a parallel backend to MATLAB.

## II. Targeted Problem Classes

A tool would be of no use if it does not solve real problems. When developing MATLAB*P we tried to generalize large scientific computing problems into categories, and provide tools that would help solve the problems. We focus on the following classes of problems:

1) Large problems - These are problems that has either large memory requirement or large computation time requirement, or both. They are very common in engineering calculations that involve models with large number of variables, e.g. structural engineering or fluid dynamics.
2) Large number of smaller problems - This is a common class of problems that is ubiquitous in scientific computing. Basically there is a large number of independent data sets to be processed using the same program. Some statistics are usually collected at the end. For example, pair-wise protein alignment and computer graphics rendering both belong to this category.

Traditionally the first class of problem is solved by a ready-made parallel numerical library, e.g. ScaLAPACK [1] or PARPACK [8]. The problem with this approach is that the user will have to program in a traditional programming language like C++ or FORTRAN, and will have to learn and handle the data distribution format used by the particular library. These are all time consuming and error prone.

There are no well defined approach to solve the second class of problems. Users have been known to use C++ with MPI to launch processes and collect statistics at the end on a cluster, or use elaborately written shell scripts with file I/O for the same purpose. In this paper we will describe how MATLAB*P can allow rapid development for the solution of these two kinds of problems.

## III. MATLAB*P 2.0 Overview

MATLAB*P 2.0 is a parallel MATLAB using the backend 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 a recent survey [3]. For example, the *embarrassingly parallel* approach allow simplistic, yet useful, division of work into multiple MATLAB sessions. The *message passing* approach, which is a superset

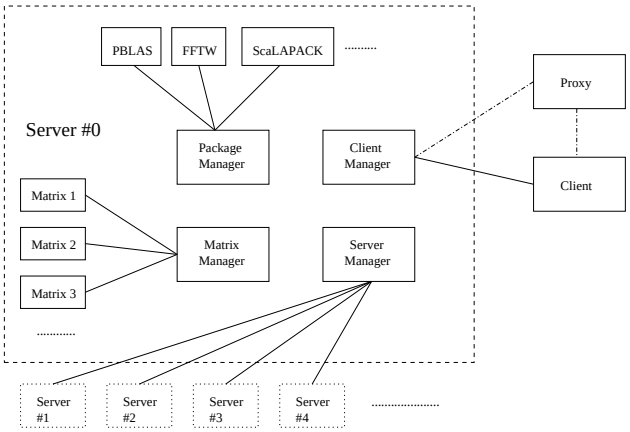


Fig. 1. Structure of MATLAB*P 2.0

of the embarrassingly parallel approach, allows finer control between the MATLAB sessions.

In order to make MATLAB*P useful for a wider range of audience, these other approaches are incorporated into the system as well.

### A. Structure of MATLAB*P 2.0 system

MATLAB*P 2.0 is a complete rewrite of MATLAB*P in C++. The code is organized in a way to ensure easy extension of the software.

The server itself is divided in four self-contained parts:

1) Client Connection Manager
   Client Connection Manager is responsible for communications with the client. It provides functions for reading commands and arguments from the client and sending the results back to the client. It is only used in the head server process.
2) Server Connection Manager
   Server Connection Manager takes care of communication between server processes. It mainly controls broadcasting of commands and arguments from head process to the slave processes, and collection of results and error codes. Also it provides rank and size information to the processes.
3) Package Manager
   Package Manager is responsible for maintaining a list of available packages and functions provided by them. When initiated by the server process, Package Manager will also perform the actual call to the functions.
4) Matrix Manager
   Matrix Manager contains all the functions needed to create, delete and change the matrices on the server processes. It maintains a mapping from client-side matrix identifiers to actual matrices on the server. It is also responsible for performing garbage collection.

This organization offers great advantages. First of all, debugging is made easier because bugs are localized and thus are much easier to track down. Also, this compartmentized approach allows easier extension of the server. For example, the basic Server Connection Manager makes use of MPI (Message Passing Interface) as the means of communication between server processes. However, one could write a Server Connection Manager that uses PVM (Parallel Virtual Machine) instead. As long as the new version implements all the public functions in the class, no change is needed in any other part of the code.

Similar extensions can be made to Client Connection Manager as well. The basic Client Connection Manager uses TCP socket. An interesting replacement would be to make a Client Connection Manager that act as an interface to a language like C++ or Java.

## IV. Features of MATLAB*P 2.0

### A. Parallelism through Polymorphism - *p mode

The key to the system lies in the *p variable. It is an object of *layout* class in 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 x 8192 normally distributed random matrix on the server. 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*

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

The command computes the eigenvalues of X by calling the appropriate ScaLAPACK routines, and store 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.

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

This command returns the result to MATLAB.

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. 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 Parry Husbands, we developed a prototype implementation of a MultiMATLAB[11]-like, distributed MATLAB package in MATLAB*P, which we call the PPEngine. With this package and associated m-files, we can run multiple MATLAB processes on the backend 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.

Examples of the usage of the PPEngine system:

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

The first example creates a distributed matrix of length 100, then fill it with random values from the chi-square distribution through calls to the function chi2rnd from 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 100x100, 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.

```
>> % 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 uses a different algorithm for the sine function.

```
>> % Example 4
>> a = (0:(np-1)*p)/np;
>> b = a + (1/np);
>> [mypi, fcnt] = mm('quadl','4./(1+x.^2)',a,b);

>> disp('Pi calculated from quadl in mm mode')
>> pi_from_quadl=sum(mypi)

>> disp('Number of function evalua-
tion on each processor')
>> fcnt(:)

Pi calculated from quadl in mm mode

pi_from_quadl =

    3.1416

Number of function evalua-
tion on each processor

ans =

    18
    18
    18
    18
```

The above example illustrates how *np*, the variable that returns the number of processes running on the backend server, can be used in a script to write adaptive code. When the above example is run on 4 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 Lobatto 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 *pi*.

### *C. Visualization Package*

This visualization package was written by Bruning, Holloway and Sulejmanpasic, under supervision of Ron 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.

Figure 2, 3, 4 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

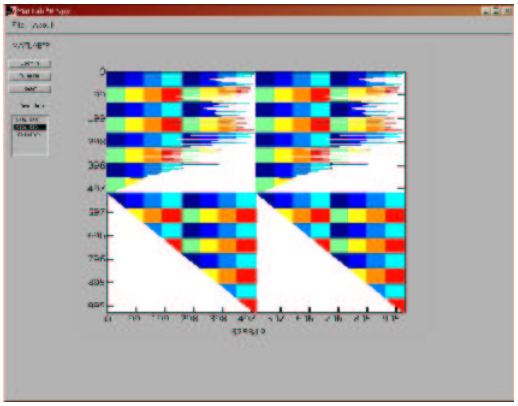

Fig. 2. ppspy on 1024x1024 matrix on eight nodes

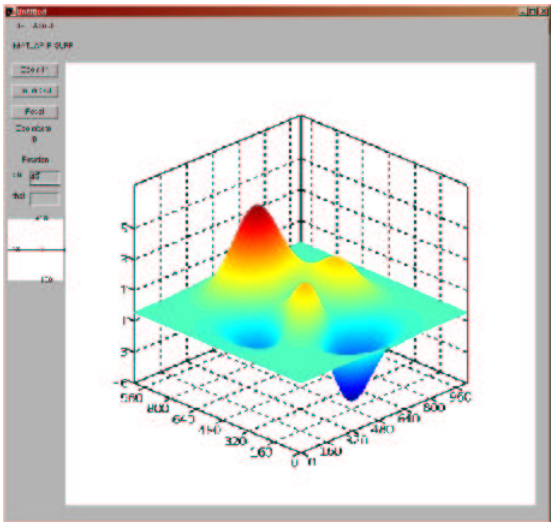

Fig. 3. ppsurf on the 1024x1024 'peaks' matrix

angle, just as supported in MATLAB.

## V. Pagerank with MATLAB*P

Google [2] is a large-scale hypertext search engine which makes heavy use of the structure present in hypertext. It assigns Pagerank, an objective measure of its importance as indicated by citation, to each webpage. The biggest challenge in computing Pagerank is to compute the eigenvalue of a huge sparse matrix.

MIT undergraduate David Cheng has developed a 'toy' version of the Pagerank computation in MATLAB*P. The code, which consists of only 53 lines of MATLAB (with comments), is listed in the appendix. The interesting thing about the code is that it would run perfectly fine in regular MATLAB. This demonstrates the power and flexibility of MATLAB*P in parallelizing MATLAB code.

The code uses the power method to compute the largest eigenvalues of a sparse matrix. Currently the sparse routines

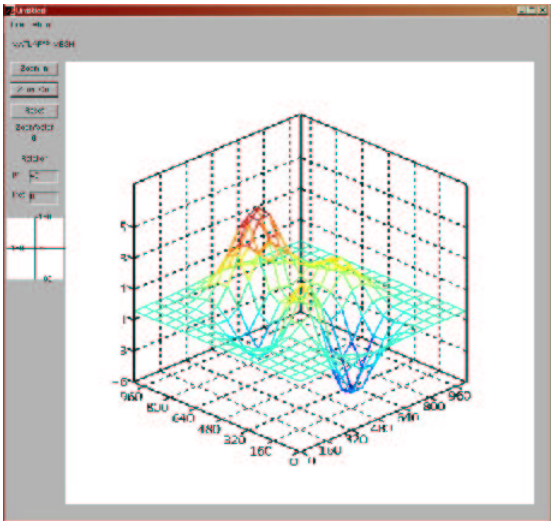

Fig. 4. ppmesh on the 1024x1024 'peaks' matrix

in MATLAB*P is not finalized, so the code is suffering from performance problems. But as the dominant operation in the power method is the sparse matrix times dense vector multiplication, which parallelizes very well, we expect that the performamce of the code would improve dramatically in the near future.

## VI. Signal Processing in MATLAB*P with MultiMATLAB mode

This once again demonstrates the power of MATLAB*P in parallelizing existing MATLAB code. We were given a 3000 lines signal processing code that perform the following operation on input radar signal data:

1) Normalize and pulse compress
2) Calculate Doppler window coefficients and calculate FFT
3) Doppler process

The first step took around 97

```
for i = 1:num_of_channels
    output(:,i) = normal-
ize_compress(input(:,i));
end
```

We were able to parallelize this for-loop with 6 lines of changes to the original serial code. Basically the for-loop is replaced with a 'MultiMATLAB' call to a script which contains the original loop. The input data is transferred to the server, and the output data is transferred back.

The following is the timing results on 10.6MB on input radar signal data. MATLAB*P was run on a 8 processor linux cluster with AMD XP 1800+ processors and with fast ethernet interconnect:

| MATLAB | MATLAB*P |
|---|---|
| 149.87s | 30.07s |

The performance of the MATLAB*P version can be improved even further by loading the input data in parallel on the MATLAB*P server.

## References


[1] 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*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 1997.

[2] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7):107–117, 1998.

[3] R. Choy. Parallel matlab survey. *http://theory.lcs.mit.edu /∼cly/survey.html*, 2001.

[4] J.J. Dongarra, J.R.Bunch, C.B.Moler, and G.W.Stewart. *LINPACK User's Guide*. SIAM, Philadelphia, 1979.

[5] M. Frigo and S. Johnson. Fftw: An adaptive software architecture for the fft. *Proc. Intl. Conf. of Acoustics, Speech and Signal Processing, 1998, v. 3, p. 1381*, 1998.

[6] P. Husbands and C. Isbell. Matlab*p: A tool for interactive supercomputing. *The Ninth SIAM Conference on Parallel Processing for Scientific Computing*, 1999.

[7] Mathworks Inc. *MATLAB 6 User's Guide*. 2001.

[8] Kristi Maschhoff and Danny Sorensen. A portable implementation of arpack for distributed memory parallel archituectures. *Preliminary proceedings, Copper Mountain Conference on Iterative Methods*, 1996.

[9] C. Moler. Why there isn't a parallel matlab. *http://www.mathworks.com/company/newsletter/pdf/spr95cleve.pdf*, 1995.

[10] 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*. Springer-Verlag, 2nd edition, 1976.

[11] A.E. Trefethen, V.S. Menon, C.C. Chang, G.J. Czajkowski, C. Myers, and L.N. Trefethen. Multimatlab: Matlab on multiple processors. Technical report, 1996.

[12] R. Clint Whaley and Jack J. Dongarra. Automatically tuned linear algebra software. Technical Report UT-CS-97-366, 1997.


## Appendix

```
function [x,cnt] = pagerankpow(G,U,damp)
%   [X,CNT]=PAGERANKPOW(G) re-
turns the pagerank of each page in the link matrix
%   G.  An optional return param-
eter CNT provides the total num-
ber of iterations
%   made (the algorithm is moti-
vated by the power method to find the principal
%   eigenvector).
%   [...]=PAGERANKPOW(G,U) dis-
plays the top few pagerank val-
ues and their
%   corresponding urls taken from U.
%   [...]=PAGERANKPOW(G,U,damp) uses the damp-
ing factor damp (default .85) used
%   in the pagerank computa-
tion.  Also displays the top few ranked pages.

% Link structure

[n,n] = size(G);
n = double(n);

% use histogram to find nonze-
ros in each column of G
%c = histc(find(G), 1:n:(n*n));
nz = find(G);
c = histc(nz(:), 1:n:(n*n));
cz = find(c == 0);
c(cz) = 1;  % so we don't di-
vide by zero later

% Power method

if nargin < 3, damp = .85; end

delta = (1-damp)/n;
x = ones(n,1)/n;  % start with uni-
form dist.
z = zeros(n,1);
cnt = 0;
while max(abs(x-z)) > .000001
   z = x;

   x = G * (z ./ c);
   x = damp*x + (damp * (sum(z(cz)) / n) + delta);

   cnt = cnt+1;
end

if nargin > 1,
  % Print URLs in page rank order.

  r = sum(G, 2);
  [ignore,p] = sort(-x);
  disp('     page-rank  in  out  url')
  k = 1;
  while (k <= 10)
     j = p(k);
     disp(sprintf(' %3.0f %8.4f %4.0f %4.0f  %s',
j,x(j),r(j),c(j),U{j}))
     k = k+1;
  end
end
```