MultiMATLAB: Integrating MATLAB with High-Performance Parallel Computing

Vijay Menon Anne E. Trefethen
Cornell University

This talk is about MultiMATLAB, an integration of the MATLAB environment with high-performance parallel computing.

This is joint work with Anne Trefethen at Cornell University.

Motivation

Two concerns in developing scientific & engineering applications:

- Time to write
- Time to run

Inversely related

In this talk, we consider with the development of scientific and engineering applications.

In particular, a developer of such an application trys to minimize two things:

- the time it takes to write, and
- the time it takes to run

Of course, we can't really do both. Instead, we must choose something between an easy-to-write, but not-very-fast prototype on one hand ...

and tedious-to-write, but extremely fast production code on the other.

One Extreme

MATLAB

- Numerical problem solving environment
- Standard in many scientific & engineering communities
- 300,000 users in 50 countries
- MATLAB is easy to use!
  - Interactive
  - High level language
  - Numerical programs and Toolboxes
  - Graphics
  - Extensible

For the types of applications we're interested in, one extreme is best characterized by MATLAB.

MATLAB is a numerical problem solving environment that has become the de facto standard in many areas of science and engineering.

In 1996, it boasted roughly 300,000 users in 50 countries, and this is a figure that's continually growing. This includes users in every U.S. national research lab and every federal reserve bank.

MATLAB is popular precisely because it's easy to use. It's an interactive environment with a number of convenient features. It has a high-level language designed for matrix computations. It comes with an extensive library of numerical routines that are commonly used in science and engineering. It's an extensible environment in which multiple users can easily shared code.

Caveat

- Performance can be limited
  - Interpreted, but ...
    - Common operations are already provided
    - Users can link in C or Fortran code
  - Sequential

The caveat to MATLAB is that performance can be limited. There are two primary reasons for this:

First, MATLAB is interpreted. But, this is not as serious as it may seem. As I said before, many common routines are already provided, and these could, presumably, already be optimized. Also, users can link in their own C or Fortran codes and call them from MATLAB. So, the computationally intensive program of a MATLAB program are not neccessarily interpreted.

One the other hand, a more limiting problem is that MATLAB is sequential. It is specifically designed for uniprocessors.

Another Extreme

Distributed Memory Multiprocessors

- Speed!
  - Fastest computers available today
  - Only means of sufficient performance for many applications
- Hard to use
  - Software is limited
  - Difficult to program

This leads us directly to our other extreme. Distributed memory multiprocessors are the fastest computers today. And often, these machines are the only means of providing sufficient performance for many large scientific and engineering applications.

The caveat here is that these machines are really hard to use. Compared to uniprocessors, software is limited and they are extremely difficult to efficiently program.

Problem

Rapid development is often incompatible with rapid execution

Goal

A common environment that provides:

- Ease of use as in MATLAB
- Performance of distributed memory multiprocessors

The key problem in this picture is that the best tools for prototyping are often incompatible with the best tools for performance.

For example: a prototype written in MATLAB cannot even use already existing parallel routines.

As a result: we cannot incremently tranform a MATLAB prototype into efficiently parallel code. Instead, we have to start from scratch.

We would like to alleviate this problem by providing a common environment that's as easy to use as MATLAB but provides access to parallel performance.

MultiMATLAB

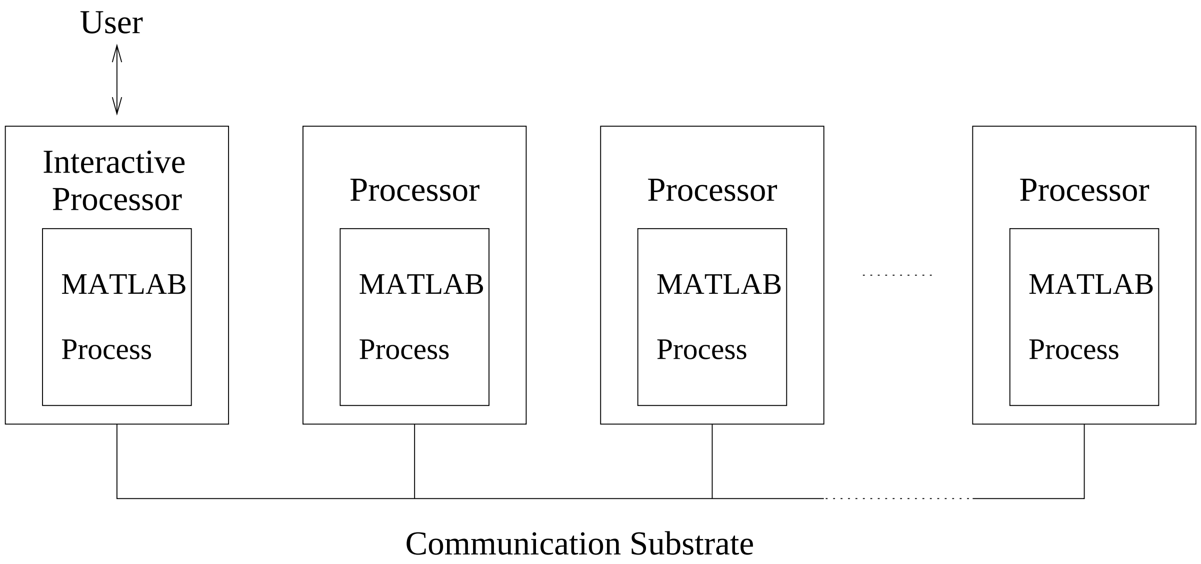
Extension of MATLAB environment to distributed memory multiprocessors

- Tools to interactively develop parallel programs in MATLAB
- Ability to extend environment with efficient parallel routines using:
  - MEX interface
  - Underlying communication layer
- Common, portable, parallel computing platform for users and third-party vendors

We propose MultiMATLAB as a solution. MultiMATLAB is an extension of the MATLAB environment to distributed memory multiprocessors. It allows users to interactively perform parallel computations and develop parallel programs within MATLAB. It provides users with the ability to link in their efficient parallel routines using MATLAB's MEX interface and the underlying communication substrate. And, it's a common, portable, parallel computing environment in which users and third-party venders can share code.

Architecture and Implementation

Run unmodified MATLAB on each processor



Communication in MPI

MATLAB + MPI $\Rightarrow$ Portability

MultiMATLAB is designed as follows. A separate MATLAB process is started up on each processor of the parallel machine. A user interacts directly with one process - which call the interactive process. Data and commands are sent from the interactive process to the other remote MATLABs, and all communication is done in MPI.

As you can see MultiMATLAB is highly portable. MATLAB will run practically anywhere, and MPI has become the communication standard for parallel machines. We have successfully run MultiMATLAB on a network of Unix workstations running MPICH, a popular publicly available implementation of MPI, and on an IBM SP2 running MPI-F, IBM's optimized version of MPI for the SP2.

Using MultiMATLAB

Basic Commands:

```
>> [NProcs ID]
ans = 4        0

>> Eval ID
ans = 1
ans = 3
ans = 0
ans = 2
```

Now, let me show you how to use MultiMATLAB. Again, we are working directly with the interactive MATLAB process. The command NProcs tells us the number of processors MultiMATLAB is running on - in this case 4. ID gives a unique ID between 0 and 1 less than NProcs. The ID of the interactive process is always 0.

The Eval command allows us to execute MATLAB code on one or more processors. For example, Eval ID, executes the command ID on all processors. The output, as you can see, is asynchronous and the results come back in some random order.

Parallel Evaluation

```
>> Eval(1,'radius = max(abs(eig(randn(400))))')
>> ....
radius =
   20.8024

>> Eval max(abs(eig(randn(400))))
ans = 20.9729
ans = 20.8508
ans = 21.0364
ans = 21.0312
```

We don't have to execute commands on all processors. In this example, we compute the spectral radius of a random 400 by 400 matrix on processor 1. While this computation is being performed, we're free to do anything else in the interactive MATLAB. When processor 1 is done, we'll see the result outputted.

Of course, we could have run this computation on all processors. In this case, each processor computes the spectral radius of a different random matrix, and, in fact, we see that the answer should be around 20.

Parallel Programming in MATLAB

Two programming paradigms:

- Master/Slave
- SPMD/Message Passing

Now, upto this point our examples haven't required any communication, and this is something we need for parallel programs.

In MultiMATLAB, we currently provide two different ways of writing parallel programs in MATLAB. The first is a simple master-slave paradigm. The second is a more general SPMD-message passing paradigm.

Master/Slave

One-sided communication from interactive process

```
>> A = ....
>> Put(1,'A');
>> Eval(1,'w = eig(A);')
....

>> Get(1,'w');
```

In the Master-Slave paradigm, we provide one-sided communication between the interactive MATLAB process and a remote process. The key commands here are Put and Get. In this example, we've computed a matrix A on the interactive process and we'd like to compute its eigenvalues. We can perform this computation remotely by using Put to send A to processor 1 and then using Eval to calculate the eigenvalues. Again, we're immediately free to do anything else on the interactive process. Later, we can retrieve the eigenvalues using Get. Get will only return once the correct value of w is available.

SPMD/Message Passing

MPI-like Message Passing primitives

- Point to point: Send, Recv
- Collective communication: Barrier, Reduce, Gather

Note:

- Communication on matrices, matrix sections, other MATLAB 5.0 data structures
- Performs memory management, data marshalling

We provide a more general mechanism for parallel programming with MPI-like message passing commands in MATLAB.

These include send and receive commands for point to point communication, and various commands for collective communication.

MultiMATLAB simplifies message passing by hiding tedious details from the programmer. For example, communication is performed on matrices, matrix sections, of other MATLAB 5.0 data structures - as opposed to send and receive buffers. Also, MultiMATLAB performs any necessary memory management and data marshalling.

SPMD Example : mysum.m

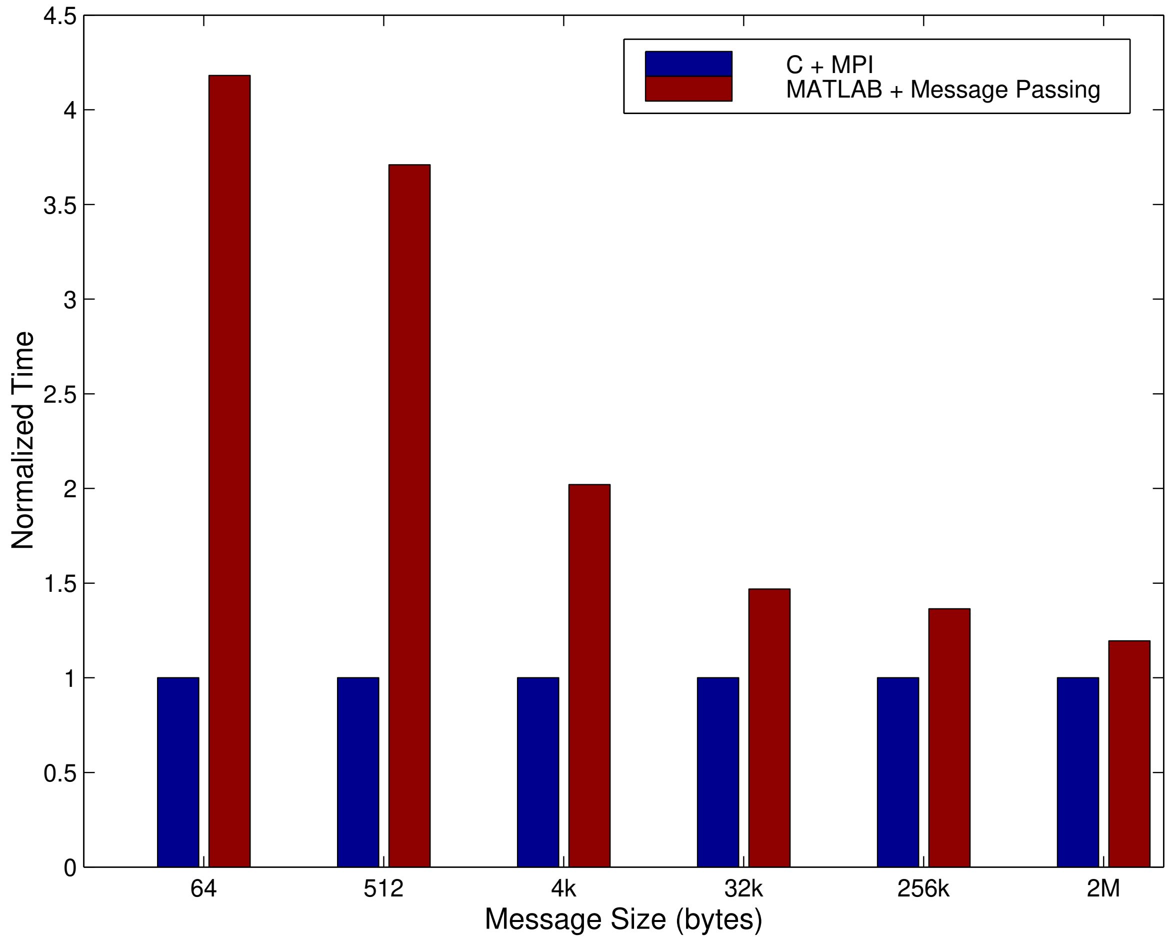
```
function y = mysum(x)
        y = x;
        if(2*ID+2 < NProcs)
                y += Recv(2*ID+2);
        end
        if(2*ID+1 < NProcs)
                y += Recv(2*ID+1);
        end
        if(ID > 0)
                Send(floor((ID-1)/2),y)
        end
end

>> Eval('k = mysum([ID NProcs]);');
>> k
k = 6 16
```

The following is an example of a MATLAB message passing program written in a SPMD style. The following is an m-file - a MATLAB function - to perform a global sum of a value over all processors. The function is run on all processors and a common tree style algorithm is used to do the sum. The processors ID implicitly define the tree. Each processor takes its own value and sums it with those sent by any children it may have. And then, if it isn't the root of the tree, it sends its sum on to its parent. At completion, processor 0, the interactive MATLAB, returns the global sum.

To run this function, we use Eval. In this case, we are computing a sum over the local values of ID and NProcs. With 4 processors, the resulting vector on the interactive process is [6 16].

Round Trip Latency (MPI-F on SP2)



Here are performance numbers. We measure the round trip latency of data on the IBM SP2 using MPI-F. We compare the MultiMATLAB Send and Recv commands in MATLAB with the corresponding MPI calls used in C. The red bars denote MATLAB latency times and the blue denote C with MPI. Our numbers are normalized with respect to the latter, which should give us a rough baseline for the best performance we can expect on the SP2. For small blocks of data, the overhead in MATLAB is quite high. For a 64 byte message, which requires only a single packet of the SP2, the overhead is more than a factor of four. With larger blocks of data, the overhead is better amortized. For a 2MB block - which corresponds to a 512 by 512 real matrix - the overhead is about 30

So, MATLAB message passing isn't really appropriate for fine-grain communication. But, if communication is infrequent or can be done in bulk, it can be effective.

Better Performance via MEX

MATLAB MEX-files

- Compiled C or Fortran functions
- MATLAB gateway to functions
- Directly callable from MATLAB

MultiMATLAB MEX-files

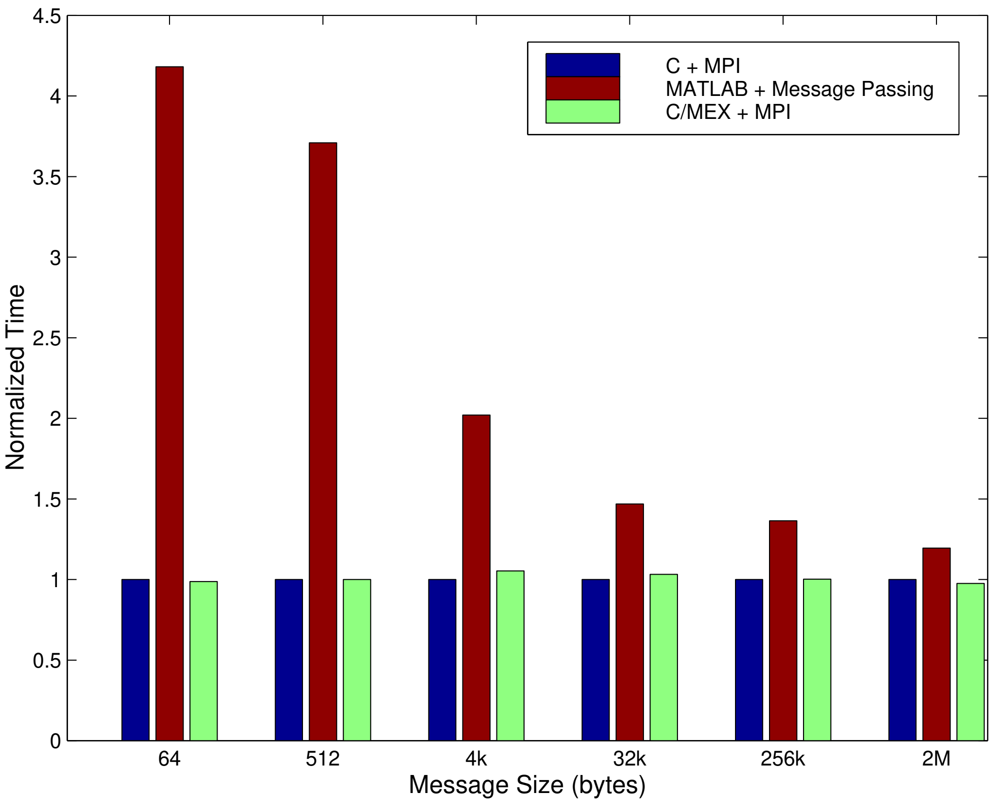
- Compiled C or Fortran with MPI calls
- Eval on 1 or more processors

Extensibility!

We can get better performance in MATLAB programs by using MATLAB's MEX interface. MEX-files are compiled C or Fortran functions that can be dynamically loaded and directly called from MATLAB. These functions provide a gateway to MATLAB that permits data to be communicated. In MultiMATLAB, users may compile C or Fortran functions with MPI calls to produce MEX-files. These MEX-files may then be run in parallel on 1 or more processors using Eval.

This is what provides extensibility in MultiMATLAB. Commands described earlier, for example Send and Recv, are simply MEX-files written in C that in turn call MPI_Send and MPI_Recv. So, users can easily add there own parallel commands.

Round Trip Latency (MPI-F on SP2)



Computation/communication intensive kernels in parallel C/MEX!

We compare the round-trip latency using MPI calls in MEX file with our earlier numbers. The new green bar is our MEX-file, and the blue bar is again our C program. We can see that the performance in the MEX-file is indistinguishable from that of the C program. The code is essentially the same in both cases - the MEX file launched from MATLAB using Eval, and the C file is a standalone program.

What these numbers suggest is that we can replace computation and communication intensive kernels of MATLAB programs with parallel C/MEX code and expect much better performance!

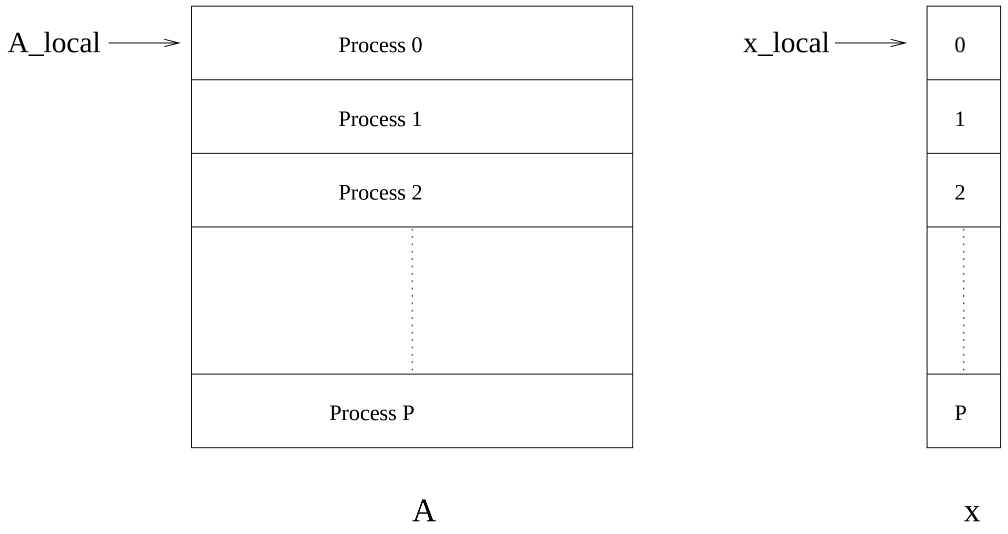
Benchmark: Conjugate Gradients

Iterative method to solve A x = b

Sequential Code

```
for k = 1 : k_max
  ...
  r = .. A * r ..
  ...
end
```

Parallel Data Distribution



We use a simple benchmark to highlight this. Conjugate Gradients is an iterative method used to solve the linear system A x = b, where A is a symmetric, positive-definite matrix and x and b are vectors. The primary computation occurring in this method is a matrix vector multiplication occuring in each iteration. In this case, the new value of a vector r is computed from the product of A with the old value of r.

A simple way to parallelize this method is to distribute all data in a row-wise manner. Each processor locally stores a contigious block of the rows of A and a contigious block of each vector. When performing the matrix-vector product, each processor only computes it local portion of r. Note that each processor only needs its local piece of A, but must obtain the entire global r each iteration.

Parallel MATLAB Code

Message Passing

```
for k = 1 : k_max
  ...
  r = Gather(r_local);
  r_local = .. A_local * r ..
  ...
end
```

Parallel MEX Call

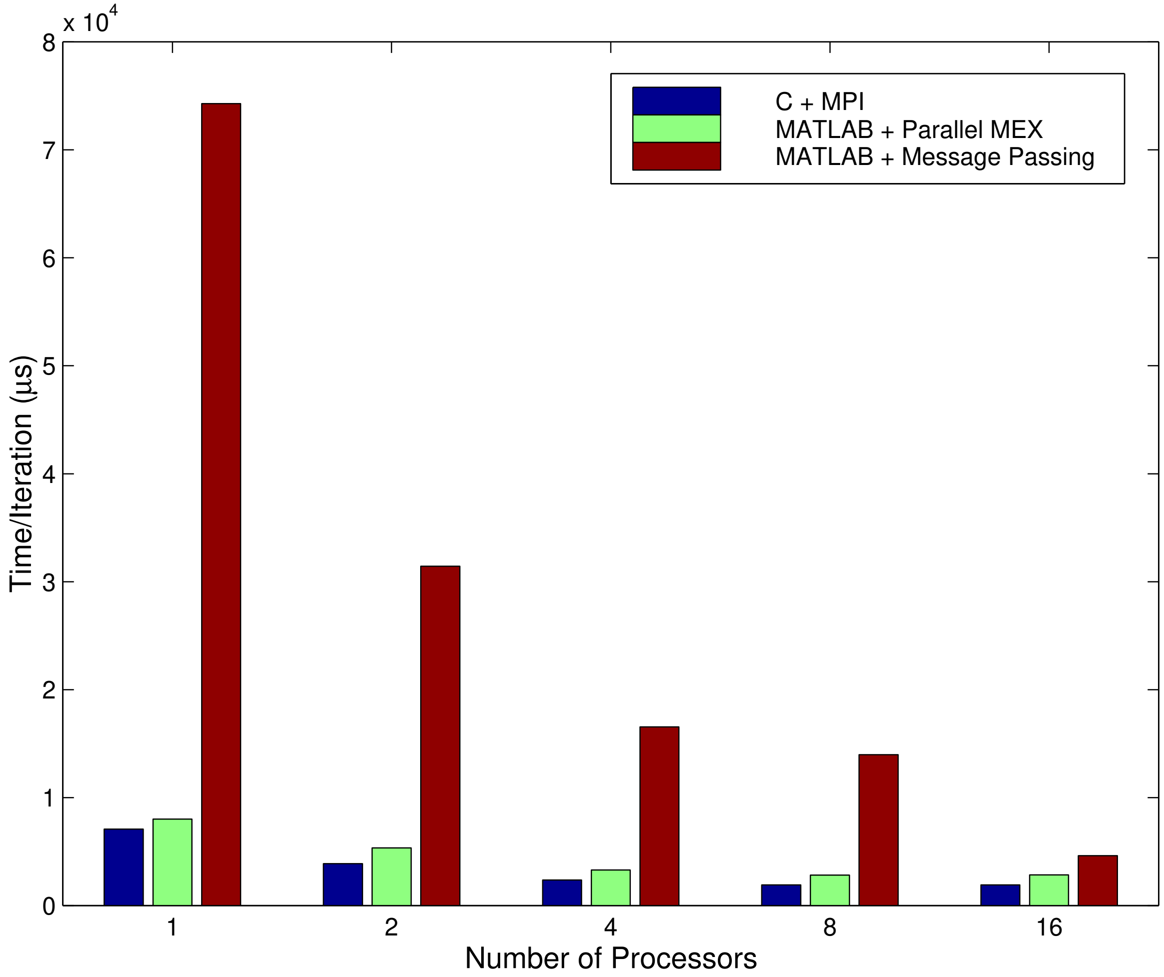
```
for k = 1 : k_max
  ...
  r_local =
   .. Pmvm(A_local, r_local) ..
  ...
end
```

Here are two MATLAB versions of our parallel algorithm written in SPMD style. In the version the left, the MultiMATLAB command Gather, which maps to MPI's All_gather call, constructs the global r from the local portion and proceeds with the local matrix vector multiplication.

In the version on the right, we perform the global matrix vector multiplication in a parallel MEX-file called Pmvm. This routine is written in C with MPI. The algorithm is the same as before, but the communication and computation for the multiplication is coalesced into a single MEX-file.

We compare both of these versions with an implementation of the same algorithm written entirely in C with MPI...

Performance : 512x512 Matrix on SP2



Here is the average time per iteration on a 512 by 512 matrix. The blue bar on the left is the version written entirely in C. The middle, green bar is the MATLAB/MEX version - that is the MATLAB program with the MEX-call doing the global multiplication. And the third, red bar is the MATLAB code using the Gather message passing command.

The first thing that jumps out is that this last version is substantially worse than the other two versions - even on one processor! What happening here is that MATLAB's matrix vector multiplication behaves very poorly for large matrices. In the MATLAB message passing code, the local multiplication is performed by MATLAB. In the other two cases, the multiplication is done in C - identical C code in both cases. Note that on 16 processors, the MATLAB message passing code performs much better - in this case, the local portion of the matrix A finally fits into the cache - and MATLAB multiplication routine is much more competitive. Still, even in this case, the MATLAB/MEX version is much better.

Although much of the code in this version is written in MATLAB, the bulk of both the computation and communication occurs in C. Because of this, the resulting program is much faster. In fact, across the board, this version is pretty competitive with the pure C code.

Ongoing Work

- New Paradigms
  - Distributed matrix class: ScaLAPACK, PLAPACK
- Compiler integration
  - MCC (The MathWorks)
  - FALCON (University of Illinois)
- Shared memory multiprocessors

MultiMATLAB is still an ongoing project. Here are some of the areas we've been looking at:

We're interested in supporting different parallel programming paradigms. In particular, we'd like to using MATLAB 5.0's object oriented features to build a distributed matrix class. MATLAB operations would then be overloaded to use parallel implementations. Ideally, we'd be able to use a library such as Scalapack or Plapack to perform these operations.

We're also interested in integrating a compiler with MultiMATLAB. There are a number MATLAB compilers out there. One of these is MCC, a MATLAB to C compiler which comes with MATLAB. Another is FALCON, a research compiler developed at Illinois that translates MATLAB to Fortran 90. We're interested in translating MATLAB with MultiMATLAB commands into more efficient C or Fortran code with MPI calls. As a bigger step, we're also interested compiling sequentially MATLAB with HPF-style annotations to parallel MEX-files.

Also, MultiMATLAB is designed for distributed memory machines. The system may need to be reengineered to better take advantage of shared memory machines and, at this point, we probably should have shared memory primitives in MATLAB.

Previous and related work

- MultiMATLAB 1.0:
  http://www.tc.cornell.edu/~anne/projects/MM.html
- Parallel Toolbox for MATLAB (Wake Forest)
- DP-Toolbox (University of Rostock)

Not efficiently extensible

Here's some similar work. First of all, we have a previous version of MultiMATLAB that has already been distributed and should be available at the listed web site. The older version is a little bit different from the one I described in this talk.

There are also two similar systems, one from Wake Forest and the other from Rostock in Germany, that based on PVM.

One of differences between what I presented and these other systems - including our older system - is that these other systems are not as efficiently extensible in the manner I described. They do not permit MEX-files with arbitrary calls to PVM or MPI. These systems all provide some manner of message passing in MATLAB, and all user code must then be built on top of that.

Users of MultiMATLAB

- Researchers at
  - Cornell Theory Center
    - Buckling Analysis
    - Speckle Reduction
    - Financial Scenario Optimization
  - Kodak - Simulation of scattering in films
- Students in
  - Undergraduate/Masters class in parallel scientific programming

Here are some of the current users of MultiMATLAB. Several researchers at the Cornell Theory center are using it for a variety of projects. A researcher from Kodak has used it to simulate scattering in photo-sensitive films. Also, MultiMATLAB has been used for educational purposes in undergraduate and Master's level course at Cornell.

Conclusion

MultiMATLAB extends the MATLAB environment to provide a viable platform for high-performance parallel computing

More information at:

http://www.tc.cornell.edu/˜anne/projects/MM.html

In conclusion, we present MultiMATLAB as a viable platform for high-performance computing. Questions?