SURVEY OF PARALLEL MATLAB TECHNIQUES AND APPLICATIONS TO SIGNAL AND IMAGE PROCESSING

Ashok Krishnamurthy, John Nehrbass, Juan Carlos Chaves and Siddharth Samsi

Ohio Supercomputer Center, Columbus, OH
{ashok, nehrbass, chaves, samsi}@osc.edu

ABSTRACT

We present a survey of modern parallel MATLAB techniques. We concentrate on the most promising and well supported techniques with an emphasis in SIP applications. Some of these methods require writing explicit code to perform inter-processor communication while others hide the complexities of communication and computation by using higher level programming interfaces. We cover each approach with special emphasis given to performance and productivity issues.

Index Terms— Distributed algorithms, Distributed computing, Array signal processing, Image processing

1. INTRODUCTION

There is an increasing recognition that High-Level Languages, and in particular scripting languages such as MATLAB, provide enormous productivity gains in developing technical and scientific code [1]. MATLAB is widely used in academia, the government and industry (almost 1 million users by some estimates) and has emerged as an important tool used by many Signal and Image Processing (SIP) scientists and engineers. The attraction of MATLAB is that it combines an easy-to-use scientific programming language with a common environment for prototyping, coding, and visualization. A wide variety of add-on toolboxes addressing a number of specialized application areas, along with a very large and active user community [2], make MATLAB the software platform of choice for many SIP applications.

Many widely used SIP algorithms are also computationally intensive and can benefit from parallelization and execution on High Performance Computers (HPCs). In a typical scenario, a scientist or engineer develops a prototype MATLAB algorithm to be run on a limited computational resource, such as a single processor PC or workstation. MATLAB is chosen over other traditional languages because 1) it is substantially easier to collaborate with non computer science experts, allowing non-experts to make changes and experiment; 2) it is interpretive and interactive; 3) it has excellent debugging capabilities; 4) it has integrated help with built-in example code; and 5) and the time to implementation is typically several orders of magnitude quicker.

The next step is to modify the simulation parameters to solve more realistic problems. This typically increases memory requirements beyond what can be accessed on the single processor PC or workstation, or it results in long-running simulations that decreases productivity or may render the simulation impractical. The traditional approach to handle these large problems was to translate the MATLAB code into C / C++ or FORTRAN, parallelize the resulting code using MPI or OpenMP, and then execute it on a HPC platform. Needlessly to say this is an expensive, error-prone and time-consuming activity. Moreover, it is very difficult to propagate changes from the MATLAB code to the corresponding C code, especially since a single line of MATLAB code may correspond to many lines of C code.

Fortunately, with the HPC emphasis rapidly shifting to high productivity metrics, in which productivity and value are more important than raw performance [3] this scenario is evolving. Time-to-solution is becoming one of the major metrics of value to technical users. Time-to-solution includes: time to cast the physical problem into suitable algorithms; time to write and debug the computer code that expresses those algorithms; time to optimize the code; time to compute the desired results; time to analyze and visualize those results; and time to refine the analysis into improved understanding of the original problem that enables scientific or engineering advances.

In this paper, we survey several modern parallel MATLAB techniques that promise to make HPC easier and to decrease time to solution by promoting ease of use, code reusability, transparent access to highly optimized libraries (BLAS, LAPACK, FFTW, etc), portable performance and isolation from the inherent complexities of HPC low level programming (C, FORTRAN, MPI, OpenMP). This makes these parallel techniques a very attractive option to address the complex computational and analysis challenges of the SIP and other communities.

We concentrate on the most promising and well supported techniques with an emphasis in SIP application examples, namely: MatlabMPI [4], bcMPI [5], pMatlab [6], Star-P [7] and the MATLAB Distributed Computing Toolbox (DCT) [8]. MatlabMPI and bcMPI require writing code explicitly to perform inter-processor communication at the HPC platform. In contrast, pMatlab, StarP and the DCT provide high level constructs that hide the complexities of inter-processor communication from the end user.

2. MatlabMPI

MatlabMPI, developed at MIT Lincoln Labs is an implementation of a subset of the MPI standard [9]. The popular MPI look and feel has been implemented using just the MATLAB language, resulting in a compact and portable implementation, which can be run anywhere MATLAB is available. Another interesting characteristic of MatlabMPI is that implements MPI communication on top of standard MATLAB file I/O. This requires a shared file system accessible by all processors. MATLAB software requires a license per computational node. Thus if one runs MatlabMPI on a HPC shared memory machine, only one MATLAB license is required to run MatlabMPI independently of the number of processors. However, when MatlabMPI is run on a distributed system, such as a Linux cluster, a license for each multi CPU node is required. A way to avoid this limitation is to use intelligent compiler configuration tools for MatlabMPI developed at OSC. These tools turn the MatlabMPI scripts into stand alone executables that require no MATLAB licenses to run in parallel.

2.1. MatlabMPI Architecture

MatlabMPI is a set of MATLAB scripts using approximately 300 lines of MATLAB code. MatlabMPI implements the basic six functions that are the core of the MPI point-to-point communications standard, namely: *MPI_Init, MPI_Comm_size(communicator), MPI_Send (destination, tag, comm, M_variable), MPI_Recv (source, tag, communicator), MPI_Finalize, MPI_Comm_rank (communicator).*

The basic communicator MPI_COMM_WORLD is a parameter for most MPI calls and is defined by MatlabMPI at the initialization phase of every MatlabMPI program. It contains all the information about the parallel environment where the program is operating. Processes within the communicator are assigned numbers (ranks) 0 to n-1. MPI_COMM_WORLD is implemented as a MATLAB structure, which contains process rank, the number of processes, the group members, and the default file I/O directory. An important attribute of this technology is that the messages can be any valid MATLAB variable. This presents a great advantage over regular MPI as MATLAB variables could be very complex objects (e.g. multidimensional matrices). These messages are transferred from one processor to another in a synchronous transfer, that is, the call does not return until the message is sent or received.

2.2. MatlabMPI for SIP Applications

MatlabMPI has been successfully applied to a variety of SIP applications. For example, we successfully parallelized an application for the formation of wide-bandwidth and wide-beamwidth SAR imagery using MatlabMPI. Also, we implemented in MatlabMPI two SIP related algorithms: a Support Vector Machine and a Content-Based Image Compression algorithm. Another application successfully parallelized was an image matching code.

2.3. Sample MatlabMPI code

As previously mentioned MatlabMPI requires writing MPI code explicitly to perform inter-processor communication. The following is a code snippet taken from a typical SIP application parallelized using MatlabMPI:

```
% send to cpu=0 and sum f_back -- partial sum
if(my_cpu>0)
   tag=my_cpu;
   MPI_Send(0,tag,comm,f_back)
else
   for i=1:(ncpu-1)
      tag=i;
      f_part=MPI_Recv(i,tag,comm); %blocking
      f_back=f_back+f_part;    end
% Remove carrier in range domain
   kxc=(kxmax+kxmin)/2;
   f_back=f_back.*exp(-cj*x(:)*kxc*ones(1,ny));
```

3. bcMPI

bcMPI developed by the Blue Collar Computing software initiative at OSC, implements MPI extensions for MATLAB and Octave [10], a open source alternative to MATLAB. bcMPI is extensible (the core library makes it easy to add additional MPI functions and interpreter data types), portable (no dependencies on any machine, operating system, or specific MPI library implementations) and most importantly scalable (uses efficient algorithms and takes advantage of native MPI library and communications hardware). Therefore, bcMPI can be used not only for embarrassingly parallel applications but also for more generic SIP problems where communication may be significant.

bcMPI consists of a core library (libbcmpi) that interfaces to the MPI library, a toolbox for MATLAB (mexmpi), and a toolbox for Octave (octmpi). Similarly to MatlabMPI, bcMPI implements a subset of the MPI API. In contrast to C or FORTRAN the MATLAB language bindings are simpler than the standard bindings for these languages. For example, data types are detected at run time,

received data is returned by value, and data communication functions accept variable number of parameters. Where possible, compatibility with MatlabMPI has been maintained. However, it is important to notice that unlike MatlabMPI, bcmpi uses blocking sends.

3.1. bcMPI for SIP Applications

Even though bcMPI was officially released recently a complex SIP application has already been ported to bcMPI. The 2006 Benchmark Challenge Executable Specification of DARPA's High Productivity Computing Systems (HPCS) Scalable Synthetic Compact Applications (SCCA) #3 on Sensor Processing and Knowledge Formation and File I/O has been ported to bcMPI at OSC [5].

4. pMatlab

pMatlab also developed by MIT Lincoln Labs is a parallel programming toolbox that overlays global array semantics on top of the MATLAB language. It uses by default MatlabMPI as its underlying communication library. However, in principle it can use much more efficient technologies for communication such as OSC's bcMPI. A user creates distributed matrices (denoted as dmats) in pMatlab by using the existing MATLAB matrix constructors, but by passing additional arguments (denoted as maps) that specify how to lay out the matrices in a grid of MATLAB processes. pMatlab provides many overloaded operators and regular communication primitives for distributed matrices many of them specially targeted toward SIP applications.

4.1. Sample pMatlab code

The following is pMatlab code snippet that performs a prototypical SIP function: a parallel FFT. The full example and many more are provided in the pMatlab official distribution. Additional information about pMatlab is presented in the ICASSP 2007 paper "pMatlab: Parallel Matlab Library for Signal Processing Applications".

```
N = 2^10; % NxN Matrix size.
PARALLEL = 1; % Turn parallelism on or off.
pMatlab_Init;  % Initialize pMatlab.
Ncpus = pMATLAB.comm_size;
my_rank = pMATLAB.my_rank;
mapX = 1;  mapY = 1;  % Create Maps.
if (PARALLEL)
% Break up channels.
mapX = map([1 Ncpus], {}, 0:Ncpus-1 );
mapY = map([1 Ncpus], {}, 0:Ncpus-1 ); end
% Allocate data structures.
X = rand(N,mapX);
Y = zeros(N,mapY);
Y(:,:) = fft(X);  % Do fft.  Changes Y from real to complex.
disp('SUCCESS');
pMatlab_Finalize;  % Finalize the pMATLAB program
```

5. Star-P

Star-P by Interactive Supercomputing Corporation is a commercially available high level solution for enabling parallelism in MATLAB programs. Star-P is a sophisticated client-server parallel computation tool that consists of two main components: an interface to an existing high-level desktop environment such as MATLAB (the client) and a computational kernel that runs on a HPC platform (the server). A user creates distributed matrices from within MATLAB running at the PC environment simply by adding a *p to one or more of its dimensions. MATLAB existing mathematical operators on regular (non-distributed) matrices are then overloaded to operate on distributed matrices on the parallel machine. A powerful and useful characteristic is that these overloaded operators implicitly propagate parallelism. Therefore, operations on distributed matrices lead to other distributed matrices.

The computational kernel in Star-P is written with C++ and MPI, and the computations are performed either with existing high-performance parallel numerical libraries such as ScaLAPACK [11] or via custom implementations. This allows obtaining performance for many commonly used SIP procedures comparable to the optimized programs written in low-level languages (e.g. C and MPI) but with significantly more productivity (using fewer lines of code).

5.1. Sample Star-P Code

The following Star-P code written on the PC MATLAB client computes the inverse of a relatively large matrix (e.g. a covariance matrix) automatically distributed by Star-P on a HPC parallel platform acting as the computational server:

```
>> x=rand(2000*p,2000*p);
>> xinv=inv(x);
>> xo=x*xinv;
>> max(max(xo))
ans = 1.0000
>> ppwhos
Your variables are;
ans 1x1 8 double array
x 2000px2000p 32000000 double array
xinv 2000px2000p 32000000 double array
xo 2000px2000p 32000000 double array
```

Notice that to parallelize this familiar MATLAB code for the parallel processors, just the *p construct was added to tell MATLAB to invoke Star-P to distribute the 2000 elements of rand to the parallel processors memory space. As indicated in the ppwhos trace (the Star-P analog to MATLAB whos) the x variable becomes a dense matrix residing in the distributed memory (ddense). It is importance to notice that the subsequent inverse and multiplication operations inherit distribution from their variables and respective results and become ddense. In contrast, when the result of an operation results in a scalar variable, this variable is placed on the front-end as

illustrated by the variable ans (it does not make sense to distribute a scalar value). Therefore the *p mechanism offers straightforward parallelization. Additional information about the Star-P system is presented in the ICASSP 2007 paper "StarP: An interactive supercomputing environment".

6. MATLAB DCT

Another commercially available solution is the Distributed Computing Toolbox (DCT) by MathWorks, Inc. The DCT lets MATLAB users distribute and control a set of computational tasks across several computational workers running under the Distributed Computing Engine (DCE). In the latest version, the workers are simply MPI processes, and the toolbox provides a limited MPI consisting of commands to perform non-blocking sends, blocking receives, broadcasts, and global reductions. The toolbox also contains additional job-control functions. Most importantly, the DCT supports high level constructs such as parallel for loops and distributed arrays (global array semantics), more than 150 overloaded math functions, and full ScaLAPACK support. Another advantage of the DCT is that it lets users utilize existing MATLAB and Simulink toolboxes and blocksets, provided the data is available locally.

6.1. Sample MATLAB DCT Code

The following MATLAB DCT code snippet implements a simple sine wave function computation in parallel:

```
%Create a 1-by-100 distributed array of 0s.
%With 4 labs, each lab has a 1-by-25 segment
P>> D = zeros(1,100,distributor)
%Populate the array with a sine wave, each lab does 1/4th.
P>> parfor i=1:100
D(i) = sin(i*2*pi/100); end;
%Gather the array so all contained in lab workspaces.
P>> P = gather(D);
%Transfer from the workspace of lab 1 to the client
%Plotting the array from the client.
%The | character means execute only in the client.
P>> |pmode lab2client P 1
P>> |plot(P)
```

Additional information about MATLAB DCT is presented in the ICASSP 2007 paper "MATLAB Distributed Computing Toolbox and applications in signal processing".

7. CONCLUSIONS

In this paper we have surveyed five contemporary approaches that provide parallel capabilities to MATLAB. All of them can and have been applied with different degrees of portability, performance and productivity to prototypical SIP applications. MatlabMPI and bcMPI are MPI based solutions that allow greater control of performance considerations but required writing explicit communication code. bcMPI take advantage of native MPI libraries and communications hardware. On the other hand MatlabMPI uses the file system for communication. In contrast to the MPI based solutions, pMatlab, Star-P and DCT provide high level interfaces that facilitated porting and development of parallel applications but that may sacrifice performance in pursuit of productivity. Several tradeoffs are possible but it is clear that powerful alternatives are available for the SIP practitioner for parallelization of MATLAB code.

ACKNOWLEDGMENTS

This publication was made possible through support provided by DoD HPCMP PET activities through Mississippi State University under contract. The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University.

8. REFERENCES

[1] A. Edelman, P. Husbands, S. Leibman, "Interactive Supercomputing's Star-P Platform: Parallel MATLAB and MPI Homework Classroom Study on High Level Language Productivity," *HPEC*, 2006
[2] MATLAB Central: http://www.mathworks.com/matlabcentral/
[3] HPCS: http://www.highproductivity.org/
[4] MatlabMPI: http://www.ll.mit.edu/MatlabMPI/
[5] bcMPI: http://www.osc.edu/hpc/software/apps/bcmpi.shtml
[6] pMatlab: http://www.ll.mit.edu/pMatlab/
[7] Star-P: http://www.interactivesupercomputing.com/
[8] DCT: http://www.mathworks.com/products/distribtb/
[9] MPI standard: http://www.mpi-forum.org/
[10] GNU Octave: http://www.gnu.org/software/octave/
[11] ScaLAPACK: http://www.netlib.org/scalapack/