# An Efficient Distributed Shared Memory Toolbox for MATLAB*

Rajkiran Panuganti[1] Muthu Manikandan Baskaran[1] Ashok Krishnamurthy[2]
Jarek Nieplocha[3] Atanas Rountev[1] P. Sadayappan[1]

[1]The Ohio State University
Columbus, OH 43210, USA
{panugant,baskaran,rountev,saday}@cse.ohio-state.edu

[2]Ohio Supercomputer Center
Columbus, OH 43212, USA
ashok@osc.edu

[3]Pacific Northwest National Laboratory
Richland, WA 99352, USA
jarek.nieplocha@pnl.gov

## Abstract

*MATLAB, the most popular high-level language for scientific computing, has significant shortcomings when used for large-scale computationally intensive applications that require very high performance and/or significant amounts of memory. Many efforts such as ParaM, pMATLAB, Star-P, and Mathworks' Distributed Computing Toolbox (DCT) are currently underway to enable the convenient development of such applications directly in MATLAB. In this paper we describe GAMMA, an efficient distributed shared memory programming model for MATLAB built using the Global Arrays library suite. GAMMA defines a set of high-level abstractions for developing parallel MATLAB programs, and provides convenient support to build high-performance parallel libraries for high-level MATLAB models such as ParaM, Star-P and DCT. We discuss the inherent challenges for an efficient implementation of GAMMA, due to the conceptual discrepancies between the programming models being integrated. Experimental results on a Pentium cluster demonstrate the effectiveness of GAMMA.*

## 1 Introduction

The power of computers has made dramatic strides over the last two decades. However, programming of complex parallel applications still remains a daunting task. Today this is recognized as one of the most significant challenges in the effective use of high-performance computing, as highlighted by the DARPA High Productivity Computing Systems (HPCS) program [9].

Applications developed using C and Fortran can achieve high performance, but their creation remains a difficult task. Parallel programming is even harder, as the programmer has to explicitly manage additional concerns (e.g., different threads of execution) and to orchestrate the interaction between concurrent components.

There is an increasing recognition that high-level languages, and in particular scripting languages such as MATLAB and Python, can provide enormous productivity advantages. An easy to use programming model with array-based semantics, powerful built-in visualization facilities, and an integrated development environment make MATLAB a highly productive environment for the scientific programmer. Another key aspect for the success of MATLAB has been the availability of a variety of domain-specific "toolboxes" (library components) for various fields. However, MATLAB currently does not meet the computational demands of many compute-intensive scientific applications. Table 1 illustrates this problem by comparing the execution times of the sequential NAS benchmarks implemented using MATLAB and using traditional languages (C/Fortran).

In the world of parallel programming, alternatives to message passing have been proposed to improve programmer productivity. One example is global shared-address space (GAS) models, which are generally easier to program than message-passing models. A number of efforts have targeted the development of scalable shared-memory models for parallel computing [26]. One of the notable successes with GAS models is the Global Arrays (GA) suite [17] developed at Pacific Northwest National Laboratory.

The fundamental goal of the ParaM [19] project at the Ohio Supercomputer Center (OSC) is the development of an environment that will enable MATLAB users to develop large-scale, high-performance applica-

*1-4244-0910-1/07/$20.00 2007 IEEE

| Application | Class A | | Class B | |
|---|---|---|---|---|
| | C/Fortran | MATLAB | C/Fortran | MATLAB |
| FT | 13.66 | 88.23 | 192.77 | Out-Of-Memory |
| CG | 3.67 | 10.29 | 652.07 | 1268 |
| IS | 2.35 | 49.48 | 10.3 | 199.12 |
| EP | 53.66 | 371.64 | 206.7 | Out-Of-Memory |

**Table 1. Execution time (in seconds) of sequential NAS benchmarks.**

tions. In this paper we describe a component of the ParaM effort, GAMMA, a global-shared-address space parallel programming system for MATLAB. A discussion of various efforts to address the performance limitations of sequential MATLAB is presented in Section 2. An overview of our system and the features of the programming model are described in Section 3, together with the challenges arising from the discrepancies in the programming models being integrated. Section 4 presents results on various benchmarks to demonstrate that GAMMA is an effective tool for building high-performance parallel libraries, with high programmer productivity. Finally, Section 5 concludes our the discussion and discusses further enhancements that are currently being pursued.

## 2 Related Work

The popularity of MATLAB has motivated various research projects to address its performance limitations. These projects vary widely in their approaches and functionalities. Broadly, these efforts can be classified into the following categories [7].

### 2.1 Compilation Approach

One of the key performance overheads of MATLAB is due to the interpreted environment. Many projects such as Otter [20], RTExpress [22], FALCON [21], CONLAB [10], MATCH [3], Menhir [6], and Telescoping Languages [5] use a compilation-based approach to address the performance limitations of sequential MATLAB by eliminating the overhead of interpretation. MaJIC [1] provides an interactive MATLAB-like frontend and compiles/optimizes code employing just-in-time compilation. However, compilation approaches require efficient implementations, in the target language, of the numerous and ever increasing MATLAB functions. Furthermore, whenever newer constructs are added in MATLAB, the compilation approaches require fundamental changes in their implementation to support the newer features. We, in our future work on ParaM, intend to follow a compilation based approach building upon GAMMA to address the performance overheads of interpreted environment. The compilation approach utilizes the parallel versions of the library functions built in GAMMA.

### 2.2 Embarrassingly Parallel Approach

Research projects such as PLab [14] and Parmatlab [16] provide support for embarrassingly parallel applications in MATLAB. Each process works only on its local data and sends the result to the parent process. However, this approach severely limits the type of applications that can be parallelized.

### 2.3 Message Passing Support

Projects such as MultiMATLAB [25], MPITB [2], MatlabMPI [13], and Mathworks' DCT 2.0 [23] add message passing primitives to MATLAB. With this approach, users have maximum flexibility to build their parallel applications using a basic set of communication primitives. MatlabMPI uses a file-based communication mechanism, while MPITB builds upon LAM/MPI communication library. However, development using message passing model requires significant developmental effort for efficiently implementing the parallel libraries and applications and defeats the productivity advantages of using MATLAB.

### 2.4 Global Address Space Models

Projects such as DLAB [18], Star-P [8], pMATLAB [24] and Mathworks' DCT 3.0 [23] provide a global address space programming model. Star-P's approach uses a client-server model where a MATLAB session acts as a front end (client) and the computation is done using a backend parallel computation engine (server). All of them provide a special MATLAB class of distributed arrays and some overloaded (parallelized) functions for these distributed arrays. Fraguela et al [11] provide a new class called Hierarchically Tiled Arrays (HTAs) in MATLAB. The objects of this class are divided into tiles which are distributed over a mesh of processors. HTAs make parallel programming easy in MATLAB by providing overloaded operators of indexed assignment and computation operators.

One of the primary reasons for MATLAB's success are the numerous and ever increasing domain-specific

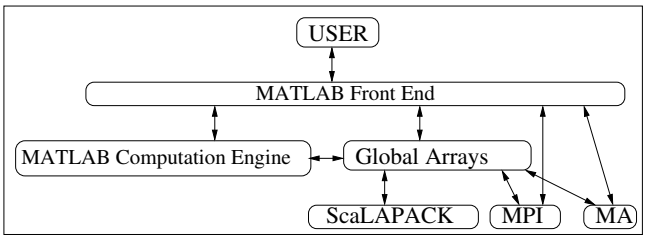


**Figure 1. Architecture of GAMMA**

library functions. Providing parallel implementations for all of these functions using message passing is very labor-intensive. pMATLAB uses MatlabMPI for the implementation of the overloaded functions. DCT also uses message passing infrastructure to develop parallel libraries for distributed arrays. While a Star-P user is restricted by the library routines that are supported, pMATLAB and Mathworks' DCT provide a mixed programming model wherein a user can also program using message passing semantics wherever library routines are not available. To address this issue, Star-P provides a software development kit (SDK) which requires library writers to program in one of the traditional languages (C++) using MPI.

All of these approaches impose a significant development burden on the library writers, as they need to overload (parallelize) thousands of existing library functions written in MATLAB. As a result, the real-world applicability of these approaches remains severely limited. The GAMMA system, described in the next section, could be used to develop high-performance distributed parallel implementations of MATLAB library functions. Such implementations could be utilized (using the functions we have provided to transform from each of the above projects' data structures to data strctures provided by GAMMA and vice versa) by any of projects listed above, in order to improve library writers' productivity when developing scalable parallel libraries.

# 3 Programming Model and Toolbox Details

The GAMMA system has been built as a MATLAB toolbox with parallel constructs using the Global Arrays (GA) [17] and MVAPICH [15] libraries. The software architecture of GAMMA is shown in Figure 1. The GA library uses Aggregate Remote Memory Copy Interface (ARMCI) [12] in its communication substrate.

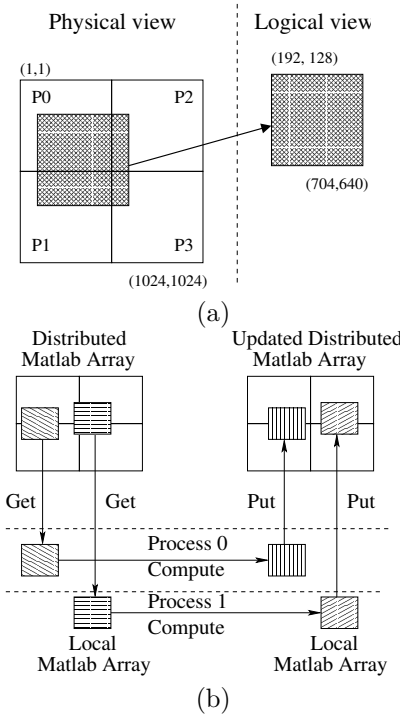


**Figure 2. (a) Global shared view of a physically distributed MATLAB array and (b) The Get-Compute-Put computation model**

## 3.1 Features of the GAMMA programming model

The features of the GAMMA programming model can be summarized as follows:

- **Global shared view of a physically distributed Array**: GAMMA provides a new distributed array datatype, for parallel programming in MATLAB, called "Global Arrays" that presents each user with a global shared view of the MATLAB arrays that are physically distributed across various processes. Figure 2(a) illustrates this model for an array that is distributed across processes $P_0, \ldots, P_3$; the required data that might span across multiple processes can be accessed by referencing it as a single logical block.
- **Get-Compute-Put computation model**: The model inherently supports a Get-Compute-Put computation style, as illustrated in Figure 2(b). The data for the computation is fetched from the distributed array independently using a *GA_Get* routine. A logical block, $A[192 : 704, 128 : 640]$, where A is a handle to an array of size $1024 \times 1024$, can be obtained by any process by a call to *GA_Get* as *block* =

*GA_Get*($A$, [192, 128], [704, 640]). ([192, 128] represents the lower indices of the logical block and [704, 640] represents the higher indices of the block as shown in Figure 2(a)). The computed data is then stored into the global array, again independently, using a *GA_Put* routine. The computation model enables a GAMMA user to make full utilization of the extensive set of library functions provided by sequential MATLAB.

- **Pass-By-Reference Semantics**: The model provides a pass-by-reference semantics for distributed arrays with the belief that it might reduce redundant memory copies of large distributed arrays. By providing pass-by-reference feature, GAMMA compromises on being fully compatible with MATLAB's value semantics based programming model. However, since GAMMA is primarily targeted for library builders, we believe that pass-by-reference semantics gives better flexibility in the model to exploit performance optimizations wherever possible.
- **Management of data locality**: The model provides support to control the data distribution and also to access the locality information and therefore gives explicit control to the user to exploit data locality. This encourages library writers to develop locality-aware code.
- **Synchronization**: Support for one-sided and asynchronous access to the global data requires user to handle synchronization mechanisms to ensure consistency. The user is provided with various explicit synchronization primitives to ensure the consistency of the distributed data.
- **Data parallelism**: The model provides support for data parallel operations using collective operations that operate on the distributed data e.g., common matrix operations such as *transpose*, *sum*, *scale*, etc. These routines provide efficient implementations of numerous MATLAB operators.
- **Data replication**: GAMMA also provides support to replicate near-neighbor data, i.e., data residing in the boundary of the remote neighbor process.
- **Distributions**: The toolbox supports both regular and irregular block distributions of distributed arrays.
- **Support for MPI**: GAMMA also provides support for message passing semantics to utilize any existing programs written in MATLAB using message passing semantics.
- **Processor groups**: The toolbox provides a facility to divide the parallel computing domain into subsets of processors that can act independently of other subsets. This functionality allows improved load balance.

```
    [ rank   nproc ] = GA_Begin();
% define column distribution
    dims = [ N   N ]; distr = [ N   N/nproc ];
    A = GA_Create(dims, distr);
    [ loA   hiA ] = GA_Distribution(A, rank);
    GA_Fill(A, 1);
% perform fft on each column of the initial array
    tmp = GA_Get(A, loA, hiA);
    tmp = fft(tmp);
    GA_Put(A, loA, hiA, tmp);
    GA_Sync();
% GA_Transpose requires the resultant array to be different
thatn source
    ATmp = GA_Create(dims, distr);
    GA_Transpose(A, ATmp);
    GA_Sync();
% perform fft on each column of the transposed array
    [ loATmp   hiATmp ] = GA_Distribution(ATmp, rank);
    tmp = GA_Get(ATmp, loATmp, hiATmp);
    tmp = fft(tmp);
    GA_Put(ATmp, loATmp, hiATmp, tmp);
    GA_Sync();
    GA_Transpose(ATmp, A);
    GA_Sync();
    GA_End();
```

**Figure 3. Parallel 2D Fast Fourier Transform in GAMMA**

### 3.2 Challenges in the implementation of the GAMMA system

The implementation of the toolbox presents several challenges.

First, MATLAB is an *untyped language*. Hence, the toolbox dynamically tracks the type of the data on which the operations are being performed in order to make the appropriate calls to underlying layers. Further, in MATLAB, a user is not exposed to any explicit memory management routines. Therefore, the memory in the user space (i.e., MATLAB space) is managed automatically by the toolbox.

Secondly, in MATLAB, a variable name is not bound to a particular memory location as in languages like C and Fortran. Due to this property, upon providing access to the local portion of the global array, the data that is being written to the array subsequently need not be written in the same location. This might happen due to the copy-on-write optimization present in MATLAB which might move the data to a different location when a memory location is being shared by two

or more variables. This makes in-place write impossible if both Global Arrays and MATLAB use different memory managers.

Thirdly, MATLAB is based on value based semantics, while we provide reference based semantics to our datatype, "Global Arrays". This is because GAMMA is intended for building parallel programming libraries and providing value based semantics for the "distributed" arrays might result in copies of large arrays. GAMMA provides Get and Put functions to transfer data between the two different operating spaces and, all the regular MATLAB functions can be used to operate on the data that is brought into the MATLAB's space. This however, might lead to scalability issues in cases where frequent updates of the distributed arrays are required to be made (the case when different processes interact frequently). To address this issue, we provide a special class of objects, called, "local". An object of type "local" provides a handle to the local part of the distributed global array. We provide in-place operations (for e.g., inplacePlus) for MATLAB operators (for e.g., plus), which accepts the two arrays on which the operation is to be performed and the array where the result is to be stored. These operations involve no copy and provide reference based semantics which a programmer can utilize to achieve superior scalability. Programs written in MATLAB are built on the basic set of operators and few in-built functions. Since, all the operators have been overloaded for objects of the type "local", existing programs written in MATLAB language require no change to execute in the system with the "local" objects.

GAMMA also handles transfer of data between the MATLAB workspace and the GA workspace. In addition, during the data transfer from the GA workspace to the MATLAB workspace, the toolbox dynamically creates a data block in the MATLAB workspace inferring the type, size, and dimensions of the block from the Get request. Furthermore, the toolbox handles the data layout incompatibility issues between MATLAB and GA and preserves the MATLAB semantics for the user. The toolbox also supports out-of-order (arbitrary) array indexing and thereby preserves an important feature of MATLAB. For example, consider a vector $A[1:100]$. A user can index the vector in an arbitrary fashion as $A([54\ 87\ 15])$.

### 3.3 Illustration of the GAMMA model

Figure 3 shows the code for parallel 2D Fast Fourier Transform (FFT) using GAMMA. The code is a straightforward implementation of a standard parallel 2D FFT algorithm. The call to *GA_Begin* initializes the underlying layers (MPI, ARMCI, and GA) and returns the rank of the process and the total number of processes. The use of *distr* in the call to *GA_Create* defines the data distribution: each block is of size $N \times (N/nproc)$, and process $P_i$ is assigned the block with logical indices for the upper left corner $(1, 1 + i \times N/nproc)$ and lower right corner $(N, (i+1) \times N/nproc)$. For better illustration, the example assumes that the global array $A$ is initialized with values of 1, using *GA_Fill*. Each process gets the block of data to operate on locally through a one-sided *GA_Get* routine (the values of *loA* and *hiA* are 2-element vectors). Every process then computes their local result using the sequential built-in *fft* function in MATLAB, and puts back the computed data into the distributed array using a one-sided *GA_Put* call. The example also demonstrates how the programming model allows programmer to utilize the functions provided by sequential MATLAB.

## 4 Experimental Results

In this section, we present a detailed assessment of GAMMA along the dimensions of programmability, and performance scalability. Using GAMMA, we implemented the NAS parallel benchmarks: FT (Fourier Transform), CG (Conjugate Gradient), IS (Integer Sort), and EP (Embarrassingly Parallel). These implementations were evaluated against the standard MPI-based Fortran (or C) implementations. All the above benchmarks, namely, NAS EP, FT, CG, and IS incurred no redundant copy from the Global Array space to the MATLAB space. To illustrate the benefit of the in-place array access, we have implemented the Jacobi iterative solver in which accessing the local portions of the global array using reference based semantics can provide significant performance gains by avoiding redundant data copy from the global address space to the MATLAB space. The results are compared with the naive version wherein data is required to be explicitly transferred from Global Arrays address space to MATLAB and vice-versa.

### 4.1 Experimental Setup

The experiments were conducted on the Ohio Supercomputer Center's Intel Pentium 4 cluster constructed from commodity PC components running the Linux operating system. The hardware and software configuration of the cluster is as follows: two 2.4 GHz Intel P4 Xeon processors on each node; 4GB RAM on each node; InfiniBand interconnection network; Red Hat Linux with kernel 2.6.6; MATLAB Version 7.0.1.24704 (R14) Service Pack 3. All experiments were conducted such that no two processes were on the same node in the cluster, ensuring that the parallel processing environment is fully distributed.

| Application | Serial C/Fortran | C/MPI or Fortran/MPI | MATLAB | GAMMA |
|---|---|---|---|---|
| FT | 665 | 1205 | 189 | 209 |
| CG | 506 | 1036 | 59 | 98 |
| IS | 422 | 665 | 128 | 197 |
| EP | 130 | 177 | 35 | 39 |

**Table 2. Lines of source code for the NAS benchmarks.**

| | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| FT(GAMMA) | 0.97 | 1.85 | 3.49 | 7.13 | 16.90 | 41.82 |
| FT(C/F+MPI) | 0.77 | 1.37 | 2.78 | 5.15 | 8.53 | 32.56 |
| CG(GAMMA) | 0.96 | 1.80 | 3.47 | 5.44 | 9.01 | 10.54 |
| CG(C/F+MPI) | 0.90 | 1.62 | 3.22 | 5.48 | 8.54 | 10.19 |
| IS(GAMMA) | 0.99 | 1.92 | 3.54 | 6.21 | 10.97 | 17.19 |
| IS(C/F+MPI) | 0.96 | 1.69 | 3.09 | 5.46 | 9.04 | 19.58 |
| EP(GAMMA) | 0.98 | 2.04 | 4.65 | 9.43 | 19.17 | 40.27 |
| EP(C/F+MPI) | 0.92 | 1.91 | 4.08 | 8.14 | 16.26 | 33.12 |

**Table 3. Speedup: NAS, Class A**

### 4.2 Programmability

GAMMA retains the programmability features of MATLAB which makes it an attractive system for achieving both high productivity and high performance for computationally intensive applications. Even though there does not exist an ideal metric for evaluating programmability of a parallel language/toolbox, the number of source lines of code (SLOC) is typically being used as a metric to measure programmability (ease of use) [4]. Table 2 compares the SLOC required to implement the NAS parallel benchmarks using GAMMA with those required to implement the sequential versions in MATLAB and the parallel and sequential versions of the benchmarks in Fortran (or C). The measurements in Table 2 clearly show that the GAMMA-based implementations require only a modest increase in the code size, compared to sequential MATLAB. Furthermore, compared to the standard MPI-based Fortran (or C) implementations of the NAS benchmarks, the number of SLOC is reduced significantly, and sometimes even dramatically (e.g., by a factor of 10 for the CG benchmark). The results clearly indicate that programming in GAMMA could potentially achieve substantial productivity benefits, compared to a message-passing-based programming model. These benefits can be attributed to GAMMA's feature-rich programming model, array-based semantics and the utilization of MATLAB's extensive set of sequential libraries.

### 4.3 Performance Analysis

GAMMA not only provides good programmability, but also achieves scalable performance. This section presents the execution time and speedup results of the NAS benchmarks FT, CG, IS and EP, written using GAMMA. We also present these results for the standard MPI-based Fortran (or C) NAS implementation (version 3.2). The sequential MATLAB versions of the benchmarks are executed with MATLAB 7.0.1, with the just-in-time compilation feature enabled. To demonstrate the benefit of in-place array access, we present the speedup results of Jacobi iterative solver implemented using the naive version of GAMMA (involving copy) and the optimized GAMMA version providing reference-based semantics.

- **NAS FT**: NAS FT solves a 3D partial differential equation using FFTs. This benchmark captures the essence of many spectral codes. From Table 3 and Table 5 that present the speedup of the GAMMA-based implementation and that of the standard hand-coded Fortran/MPI (or C/MPI) implementation for Class A and Class B, and Table 1 that presents the execution times of sequential MATLAB implementation and sequential Fortran/C implementation for Class A and Class B, it can be inferred that the execution times of the GAMMA implementation become comparable to that of the Fortran implementation with increasing number of processors. Our measurements also show that the speedups achieved by GAMMA implementation are slightly better than the speedups achieved by standard Fortran implementation. This is because the communication efficiency of GAMMA is comparable to that achieved using MPI in traditional languages. The computation efficiency of MATLAB also increases superlinearly as the problem size handled by each processor decreases (with the increase in number of processors) in NAS FT.
- **NAS CG**: The NAS Conjugate Gradient (CG) benchmark is a scientific kernel that uses an inverse power method to find the largest eigenvalue of a symmetric definite random sparse matrix. As in the case of NAS FT, from Table 3, Table 5 and Table 1, it can be observed that the execution times of the GAMMA implementation gradually become comparable to that of the Fortran implementation. This is possible because of the utilization of the "processor group" feature available in the GAMMA system. It can also be observed that

| | 1 | 2 | 4 | 8 | 16 |
|---|---|---|---|---|---|
| Naive (N=512) | 0.882 | 2.565 | 4.661 | 8.940 | 10.640 |
| No Copy (N=512) | 0.965 | 2.843 | 5.023 | 9.634 | 14.212 |
| Naive (N=1024) | 0.847 | 2.880 | 4.768 | 9.304 | 11.395 |
| No Copy (N=1024) | 0.869 | 3.213 | 5.708 | 10.07 | 14.502 |

**Table 4. Speedups using reference based semantics to access local portions of the global arrays for Jacobi iterative solver.**

| | 1 | 2 | 4 | 8 | 16 | 32 |
|---|---|---|---|---|---|---|
| FT(GAMMA) | - | - | - | 1 | 2.29 | 5.49 |
| FT(C/F+MPI) | - | - | - | 1 | 1.98 | 3.96 |
| CG(GAMMA) | 0.99 | 2.01 | 8.97 | 19.87 | 39.85 | 68.50 |
| CG(C/F+MPI) | 0.99 | 4.35 | 8.35 | 22.53 | 37.45 | 68.06 |
| IS(GAMMA) | 0.99 | 1.94 | 3.52 | 6.28 | 10.96 | 17.38 |
| IS(C/F+MPI) | 0.94 | 1.71 | 3.32 | 5.95 | 9.72 | 16.35 |
| EP(GAMMA) | - | - | 1 | 1.90 | 4.02 | 9.30 |
| EP(C/F+MPI) | - | - | 1 | 1.99 | 3.99 | 8.02 |

**Table 5. Speedup: NAS, Class B**

the speedups achieved by GAMMA implementations are either comparable or slightly better than the speedups achieved by Fortran implementations due to the reasons described earlier.

- **NAS IS**: The NAS IS benchmark is a parallel integer sort kernel. It performs a sorting operation that is important in particle method codes; the benchmark tests both integer computation speed and communication performance. As in the cases of NAS FT and CG, the execution times of the GAMMA implementation eventually get closer to that of the C implementation. Superior speedups can be observed even in this case.
- **NAS EP**: The NAS EP benchmark is an embarrassingly parallel kernel. It provides an estimate of the upper achievable limits for floating point performance - that is, performance without significant interprocess communication. As with the other benchmarks, it can also be observed here that the performance gap between the two implementations decrease with increasing number of processors.
- **Jacobi iterative solver**: The Jacobi iterative solver uses the Jacobi method to solve a linear system of equations arising in the solution of a discretized partial differential equation. All data were distributed using a two-dimensional block distribution. Table 4 demonstrates the benefit of providing reference-based semantics for accessing the local portions of the global array. The sequential execution times are 16.218 sec and 72.209 sec for the problem sizes with N=512 and N=1024 respectively. This benchmark also utilizes the in-place semantics provided by GAMMA to achieve superior speedups.

### 4.4 Overcoming MATLAB's Memory Limitations

The use of MATLAB for large-scale computationally intensive applications is also limited because of memory constraints. For example, the sequential version of MATLAB runs out of memory and is unable to execute the NAS benchmarks FT and EP for the Class B problem size. One of the direct advantages of parallel computing is the ability to run problems of larger scale due to the availability of memory from multiple nodes. By enabling parallel computing, GAMMA allows MATLAB users to run large-scale problems with data distributed across multiple processors. This is illustrated with the results for FT and EP for Class B problem size. With GAMMA, NAS EP Class B can be run on four or more processors and NAS FT Class B can be run on eight or more processors. The speedup is calculated with respect to the execution time using the minimum number of processors that could successfully run the benchmark in GAMMA. Table 5 shows the speedup for GAMMA and Fortran implementations of NAS EP and FT Class B, with the execution time on four processors as the reference for EP and that on eight processors as the reference for FT.

These experimental results clearly indicate that (1) the GAMMA implementations of the benchmarks achieve good scalability, (2) the performance gap between the Fortran (or C) implementation of the NAS benchmarks and the corresponding GAMMA implementation decreases as more processors are used, (3) the GAMMA system provides significant productivity benefits for library writers to build their custom parallel libraries using distributed arrays, and (4) MATLAB programmers can run computations of larger problem sizes that are impossible to be run using the standard sequential MATLAB.

## 5 Conclusions and Future Work

This paper has described GAMMA, a parallel MATLAB environment providing a global shared view of arrays that are physically distributed on clusters, and a get-compute-put model of parallel computation. Several examples were provided, illustrating the ease of programming coupled with high efficiency. Use of the get-compute-put model also facilitates effective reuse of existing sequential MATLAB libraries as part of a parallel application. An additional benefit of using the GAMMA system is the ability to run larger problems than sequential MATLAB.

The GAMMA toolbox is currently being used by

the staff and users at the Ohio Supercomputer Center and will soon be made available as a public release. GAMMA is part of a larger effort to develop a high-productivity environment called ParaM [19] - a parallel MATLAB project that aims at combining compilation technology along with parallelization, to enable very high performance. Efforts are currently underway to develop a number of parallel MATLAB applications and numerical libraries using GAMMA.

**Acknowledgements**

This work was supported in part by funding to the Ohio Supercomputer Center Springfield Project through the Department of Energy ASC program

## References

[1] G. Almasi and D. Padua. Majic: compiling matlab for speed and responsiveness. In *PLDI '02: Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation*, pages 294–303, New York, NY, USA, 2002. ACM Press.

[2] J. F. Baldomero. MPI/PVM toolbox for Matlab. http://atc.ugr.es/javier-bin/pvmtb_eng.

[3] P. Banerjee, N. Shenoy, A. Choudhary, S. Hauck, C. Bachmann, M. Haldar, P. Joisha, A. Jones, A. Kanhare, A. Nayak, S. Periyacheri, M. Walkden, and D. Zaretsky. A MATLAB compiler for distributed, heterogeneous, reconfigurable computing systems. In *Symposium on Field-Programmable Custom Computing Machines*, pages 39–49, 2000.

[4] L. Briand, T. Langley, and I. Wieczorek. A replicated assessment and comparison of common software cost modeling techniques. In *International Conference on Software Engineering*, pages 377–386, 2000.

[5] A. Chauhan. *Telescoping MATLAB for DSP Applications*. PhD thesis, Rice University.

[6] S. Chauveau and F. Bodin. Menhir: An environment for high performance Matlab. In *International Workshop on Languages, Compilers, and Run-Time Systems for Scalable Computers*, LNCS 1511, pages 27–40, 1998.

[7] R. Choy and A. Edelman. Parallel MATLAB: Doing it right. *Proceedings of the IEEE*, 93(2):331–341, Feb. 2005.

[8] R. Choy, A. Edelman, J. R. Gilbert, V. Shah, and D. Cheng. Star-P: High productivity parallel computing. In *Workshop on High Performance Embedded Computing*, 2004.

[9] DARPA. High productivity computing systems (HPCS) program. http://www.highproductivity.org.

[10] P. Drakenberg, P. Jacobsen, and B. Kåström. A CONLAB compiler for a distributed memory multicomputer. In *SIAM Conference on Parallel Processing for Scientific Computing*, pages 814–821, 1993.

[11] B. B. Fraguela, J. Guo, G. Bikshandi, M. J. Garzaran, G. Almasi, J. Moreira, and D. Padua. The hierarchically tiled arrays programming approach. In *LCR '04: Proceedings of the 7th workshop on Workshop on languages, compilers, and run-time support for scalable systems*, pages 1–12, New York, NY, USA, 2004. ACM Press.

[12] J. Nieplocha and B. Carpenter. ARMCI: A portable remote memory copy library for distributed array libraries and compiler run-time systems. In *IPPS/SPDP Workshops*, LNCS 1586, pages 533–546, 1999.

[13] J. Kepner and S. Ahalt. MatlabMPI. *Journal of Parallel and Distributed Computing*, 64(8):997–1005, Aug. 2004.

[14] U. Kjems. PLab: reference page, 2000. http://bond.imm.dtu.dk/plab/.

[15] J. Liu, J. Wu, S. Kini, D. Buntinas, W. Yu, B. Chandrasekaran, R. Noronha, P. Wyckoff, and D. Panda. MPI over InfiniBand: Early experiences. Technical Report OSU-CISRC-10/02-TR25, Ohio State University, Oct. 2002.

[16] Lucio Andrade. Parmatlab, 2001.

[17] J. Nieplocha, R. J. Harrison, and R. J. Littlefield. Global arrays: A nonuniform memory access programming model for high-performance computers. *The Journal of Supercomputing*, 10(2):169–189, June 1996.

[18] B. R. Norris. An environment for interactive parallel numerical computing. Technical Report 2123, Urbana, Illinois, 1999.

[19] Ohio Supercomputer Center. ParaM: Compilation of MATLAB for parallel execution. http://www.osc.edu/springfield/research/matlab.shtml.

[20] M. Quinn, A. Malishevsky, N. Seelam, and Y. Zhao. Preliminary results from a parallel MATLAB compiler. In *International Parallel Processing Symposium*, pages 81–87, 1998.

[21] L. D. Rose, K. Gallivan, E. Gallopoulos, B. A. Marsolf, and D. A. Padua. FALCON: A MATLAB interactive restructuring compiler. In *Languages and Compilers for Parallel Computing*, pages 269–288, 1995.

[22] RTExpress. Integrated Sensors Inc. http://www.rtexpress.com.

[23] The Mathworks. Distributed Computing Toolbox and MATLAB Distributed Computing Engine 2.0.1. http://www.mathworks.com/products/distribtb/.

[24] N. Travinin, R. Bond, J. Kepner, and H. Kim. pMatlab: High Productivity, High Performance Scientific Computing. In *2005 SIAM Conference on Computational Science and Engineering*, 2005.

[25] A. E. Trefethen, V. S. Menon, C.-C. Chang, G. Czajkowski, C. Myers, and L. N. Trefethen. MultiMATLAB: MATLAB on multiple processors. Technical Report TR96-239, Cornell Theory Center, Cornell University, 1996.

[26] UC Berkeley/LBNL. Berkeley UPC - Unified Parallel C. http://upc.nersc.gov.