MAT*marks*: A Shared Memory Environment for MATLAB Programming

George Almasi, Calin Cascaval, and David A. Padua
Department of Computer Science
University of Illinois at Urbana-Champaign
{galmasi,cascaval,padua}@cs.uiuc.edu

Abstract

*MAT*marks *is an extension of the MATLAB tool that enables shared memory programming on a network of workstations by adding a small set of commands. In this paper, we present a high-level overview of the MAT*marks *system, the commands we added to MATLAB, and the performance gains we achieved as a result.*

1. MAT*marks* Overview

In this paper[1], we describe MAT*marks*, a shared memory extension to the popular MATLAB [5] matrix manipulation language and environment. MAT*marks* provides a means by which the user can run several MATLAB interpreters in one session executing parallel codes following the SPMD (Single Program Multiple Data) programming model [2].

The MATLAB package itself is not modified in any way. The MAT*marks*system defines a small number of new primitives to manage parallelism and works behind the scenes to maintain the illusion that some MATLAB variables are shared (i.e. participating MATLAB processes see the same value).

SPMD behavior is achieved by the MAT*marks* GUI which, in effect, is a front-end that bootstraps the subordinated MATLAB processes and distributes user input appropriately to each process.

MAT*marks* is based on the TreadMarks [1] virtual distributed shared memory system. TreadMarks provides a shared memory view for processes running on a network of workstations. MAT*marks*extends this behavior to MATLAB. Unlike TreadMarks, which requires shared memory regions to be declared at compile time, MAT*marks* allows sharing of MATLAB variables to occur dynamically.

[1]This work was supported by the National Science Foundation under grant ACI-9870687.

TreadMarks (and hence MAT*marks*) uses data replication to implement the illusion of shared memory. Unlike most hardware-based shared memory systems, TreadMarks is not sequentially consistent [4]. Instead, it implements a *lazy release consistency model* [3] in order to minimize the number of messages exchanged through the network. In practice, this means that good care must be taken not to allow data races in the programs because unexpected behavior will certainly result. Data races can be avoided using the synchronization primitives provided by MAT*marks*.

2. MAT*marks* Primitives

To keep the system simple, we tried to keep the number of extension primitives as small as possible. They fall into the following categories:

- Initialization and termination. Initialization starts several separate MATLAB interpreters and connects the underlying TreadMarks layers of each process.

- Process synchronization and process identification. These are basically TreadMarks [1] commands mirrored in MAT*marks* that enable SPMD programming.

- Declaration of shared variables. The `MMK_Share` command, when executed by an interpreter, initializes a variable to be shared; if any interpreter in the MAT*marks* process group changes the value of a shared variable, the changed value will be propagated, at the appropriate synchronization points, to all interpreters that have declared it shared.

The following example exposes the behavior of `MMK_Share`:

* interp 1 *	* interp 2 *
>MMK_Share ('a', 45);	>MMK_Share ('a', 45);
>a = 1001;	>
>MMK_Barrier;	>MMK_Barrier;
>a	>a
a = 1001	a = 1001

Here, two interpreters share the value of the variable `a`. When the variable is first initialized, the variable in each interpreter contains the value given in the initialization statement (the second argument of `MMK_Share`). The value of `a` is synchronized at the first lock or barrier, conforming to the lazy release consistency model implemented by TreadMarks.

3. Experimental Results

All five of our benchmarks are parallel versions of simple MATLAB programs. Each of them is 100 lines or less in length. We executed the benchmarks on an array of UltraSparc machines.

Our first two benchmarks, MM and MM2, are respectively the "normal" (FORTRAN-style, with loops) and vectorized (MATLAB, or FORTRAN-90 style) versions of matrix multiplication. We used the MM and MM2 benchmarks to gauge the effect of system load on the performance of MAT*marks*. Synchronization between processes is minimal.

Our next two benchmarks, JACOBI and JACOBIOPT, implement the Jacobi relaxation method to solve a two-dimensional elliptic PDE. JACOBI is a fairly naive implementation, and synchronizes the main work array in every iteration. JACOBIOPT attempts to reduce synchronization traffic by exchanging only the boundary regions between processors. Thus, the number of bytes per message decreases, but the number of messages remains the same.

The last benchmark, SIEVE, is a parallelized implementation of Eratosthenes' sieve for computing primes. This benchmark is designed to showcase the ease of executing irregular loops in parallel with MAT*marks*.

All benchmarks show approximately linear speedups; the matrix multiplication algorithms and SIEVE show close to ideal speedups, whereas both JACOBI algorithms show 50% efficiency.

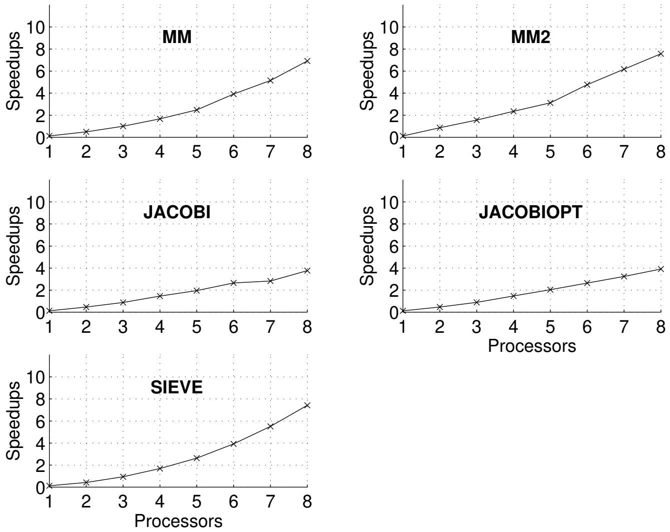

Figure 1. Parallel speedup curves for the five benchmarks on up to 8 processors

4. Conclusions and Future Directions

MAT*marks* is simple, portable, and reliable. It offers faster execution on a network of workstations while preserving the advantage of an interactive environment. The changes to the MATLAB environment are minimal.

Performance results show that linear speedup can be achieved on a moderate number of workstations. While transforming a serial program into a shared memory parallel one is much easier than writing a message-based parallel program, for good performance one needs to be more careful coding a parallel algorithm. More research is needed to pin down the factors that limit parallel speedups.

The full MAT*marks* paper is available at http://polaris.cs.uiuc.edu/matmarks.

References

[1] C. Amza, A. Cox, S. Dwarkadas, P. Keleher, H. Lu, R. Rajamony, W. Yu, and W. Zwaenepoel. Treadmarks: Shared Memory Computing on Networks of Workstations. *IEEE Computer*, 29(2):18–28, February 1996.

[2] F. Darema, D. A. George, V. A. Norton, and G. F. Pfister. A single-program-multiple-data computational model for epex/fortran. *Parallel Computing*, 7(1):11–24, April 1988.

[3] P. Keleher. *Lazy Release Consistency for Distributed Shared Memory*. PhD thesis, Department of Computer Science, Rice University, December 1994.

[4] L. Lamport. How to make a multiprocessor computer that correctly executes multiprocess programs. *IEEE Transactions on Computers*, C-28(9):241–248, September 1979.

[5] Mathworks Inc. homepage. `www.mathworks.com`.