Parallel Performance of Pure MATLAB "M-files" versus "C-code" as applied to formation of Wide-Bandwidth and Wide-Beamwidth SAR Imagery

Dr. John Nehrbass[1] nehrbass@ee.eng.ohio-state.edu **937-904-5139,**
Dr. Mehrdad Soumekh[2] msoum@eng.buffalo.edu **716 645-3115 x 2138,**
Dr. Stan Ahalt[1] sca@ee.eng.ohio-state.edu **614-292-0068,**
Dr. Ashok Krishnamurthy[1] akk@ee.eng.ohio-state.edu **614-292-5604,**
Dr. Juan Carlos Chaves[1] jchaves@arl.army.mil **301-394-0408**
[1] **Department of Electrical Engineering, The Ohio State University, Columbus, Ohio 43210**
[2] **Deparment of Electrical Engineering, State University of New York at Buffalo**,
332 Bonner Hall, **Amherst, NY 14260**

Applicable Topic Areas:

- Case Study Examples of High Performance Embedded Computing
- Performance Modeling and Simulation for Benchmarking Embedded Systems
- Software Architecture, Reusability, Scalability, and Standards

Abstract

Once an algorithm is stable and leaves the research and development stage, it can be optimized for particular hardware architectures and run in production mode. Most code written never gets to this advanced stage, however the need for speed may be critical for realistic simulations. Likewise, one may be impressed with the long run times of simulations, which run for weeks or longer while using hundreds of processors in parallel to provide the needed information. However, it is often overlooked that the resources required to place such problems into formats suitable for such lengthy simulations many span several years. Grid quality, a labor-intensive step, supercedes numerical convergence criteria for accurate RCS predictions. Similarly, it is rare that an author is able to publish after only one run on the first bug free version of their code.

To the extent that tools can allow rapid prototyping, increased value is achieved. For scientists in the area of Signal and Image processing, MATLAB is often the tool of choice. MATLAB allows extremely rapid prototyping and debugging of complicated studies with a minimal of computer science expertise. Frequently an idea is implemented in MATLAB code and tested on small data sets. These small data sets provide outputs in sufficient time such that performance studies can be conducted. However, as the studies progress and the same codes are used on much larger real data sets, run times may grow to unrealistic lengths. This may force the research scientist to use another implementation language, such as C and MPI, and implement the code on parallel architectures. This code conversion is undesirable, time consuming, and error prone.

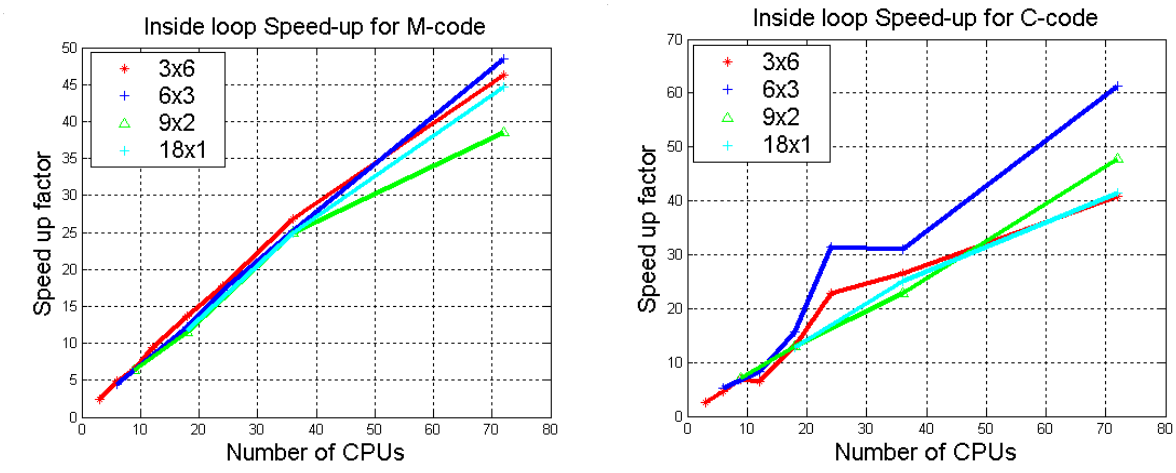
An alternative and convenient way to accomplish run time reduction is to use the MatlabMPI suite written by Jeremy Kepner of MIT. This suite of pure MATLAB code allows one to simulate many of the MPI functions within MATLAB. It accomplishes inter-processor communication via disk I/O and a common disk drive. MATLAB software requires a license per computational node. Thus if one runs MatlabMPI on a shared memory machine, only one MATLAB license need be used for any given implementation. When run a distributed system, such as the IBM SP2/3 or a set of Linux clusters, then a license for each grouping of machines is required. For large jobs, (i.e. 256 processors or more) the additional costs of licenses may not be justified. Realizing the combined value of MATLAB, MatlabMPI, the High Performance Computational Modernization Program (HPCMP), and distributed architectures, an alternative is presented. MatlabMPI works by creating scripts that contain instructions for starting individual MATLAB processes, one for each processor desired on a given node. Executable code can be substituted within the scripts wherever a MATLAB process was identified. The MATLAB compiler toolbox may be used to create MATLAB executable code (MEX) fused within a C wrapper. The C wrapper simply allows one to easily define the rank of a launched process and pass this information to the MEX code.

These altered scripts no longer require any MATALB licenses at run time. Thus distributed systems can now affordably be added to the suit of resources that MatlabMPI can be used on. This presentation will illustrate the comparison of timing between running pure parallel MATLAB code with the automatically created C compiled version of the same code. A sample speed up comparison is illustrated below.

As rapid prototyping is the main focus of this effort, the illustration is implemented on a fairly complicated image formation code that generates large SAR images. Initial development was conducted on smaller sets of simulated data and thus relatively small images were produced. However, the real applications needed to generate images with dimensions of 4 by 7 kilometers at resolution of 3 meters, which generates images of dimensions 1333 pixels by 2333 pixels. The phase history processed consisted of over 18,000 slow time samples each containing thousands of frequencies or fast time complex data samples. When processing real data on 2Ghz Pentium IV Pc machines, run times spanned from over 1 hour to approximately 2 hours. The variations in run times were caused by variations in optimizing parameters and availability of RAM. After a very modest effort of porting this code to HPC resources and using parallel processors to run the same parameter sets, run times were reduced to less than 2 minutes. It is believed that with a finer grain parallization, run times will be reduced to less than 15 seconds and thus achieve real time processing claims. These results, images, timings, and details will be presented at the HPEC conference.

The formation of general wide-bandwidth and wide-beamwidth SAR imagery follows an approach that is based on the SAR ***wavefront*** theory. The SAR wavefront theory was introduced in the past decade for approximation-free image formation. Meanwhile, the fundamental SAR signal properties that are established via this theory have been proven to be useful not only for image formation but also for addressing pre- and post-processing problems that are encountered in practical SAR systems. These include RFI suppression, suppression of azimuthal spatial aliasing in wide-beamwidth SAR systems, motion compensation and auto-calibration, geo-registration and calibration of multipass data for change detection. The specific issue that is examined in this study is the large integration angle of these SAR systems; in this case, the user faces processing relatively large FFTs for Doppler processing to avoid azimuthal spatial aliasing. A process that we refer to as ***subaperture-based digital spotlighting*** is used to circumvent the above-mentioned problem; this also makes the implementation ideal in a multi-processor environment.

In order to facilitate a more efficient I/O the phase history data is broken up in to many files (subapertures), each containing only hundreds of subsets of the total slow time data sets. Processing through each of the files is done in an outer most loop. Auto focusing through one file is independent of processing through another, and thus this technique is an excellent candidate for parallel implementation. If 1,2,3,6,9, or 18 processors are used, the outer loop can be perfectly load balanced. Likewise, an inner loop independently performs an FFT over sub-apertures. This loop is also optimal for parallization. For this data set 72 sub-apertures are possible and thus the code was scaled from 1 to 72 processors.



This publication was made possible through support provided by DoD HPCMP PET activities through Mississippi State University under the terms of Agreement No. #GS04T01BFC0060. The opinions expressed herein are those of the author(s) and do not necessarily reflect the views of the DoD or Mississippi State University.