# HPEC Challenge SAR Benchmark pMatlab Implementation and Performance

Julia S. Mullen, Theresa Meuse, Jeremy Kepner
{jsm,tmeuse,kepner}@ll.mit.edu
MIT Lincoln Laboratory, 244 Wood Street, Lexington, MA 02420

## Introduction

Synthetic Aperture Radar (SAR) is an important application for the HPEC and HPCS communities. The HPEC Challenge SAR Benchmark has been designed to test the performance of parallel computing and parallel storage systems using a representative set of computational and I/O intensive tasks associated with signal processing. By design, the benchmark is scalable and appropriate for deployment on systems ranging from desktop to petascale parallel architectures.

This HPEC Benchmark is also part of the High Productivity Computing Systems (HPCS) Scalable Synthetic Compact Application (SSCA) Suite. While this suite is designed to stress test large multiprocessor systems, the HPCS Program recognizes and works to address the inherent high cost associated with parallel code development. As part of this effort, we describe a method for parallelizing the benchmark using pMatlab. We show that this is a highly productive approach, providing good speed-up for minimal code changes.

## SAR System Benchmarks

The SSCA #3 represents a generalized sensor processing chain that consists of a front-end sensor processing stage and a back-end knowledge formation stage, both of which include significant data I/O components. This two stage processing is representative of a broad range of military and commercial image processing applications where data is acquired and processed in one order and later retrieved and processed in a different order. The Benchmark kernels were described and presented at HPEC 2005 [1]. Prior to and in addition to the two stages mentioned above, the Scalable Data Generator creates and stores the 'raw' SAR data.

The diagram in Figure 1 details the individual stages. In Stage 1, the raw data is read, an image is formed, templates are inserted and an image is written to disk for a user specified number of images. Stage 2 consists of a similar loop within which a specified number of image sequences are randomly chosen to be read, each through its own full grid depth (number of images in the sequence). For each chosen image sequence, the difference between each of its sequential pairs of images is computed, to discern all newly changed regions. These newly changed regions (correspond to newly inserted templates) are then identified (detected) and stored in labeled sub images.

The benchmark is validated by comparing the location, identity, and rotation of the letters detected against the set that was known to be inserted at that particular sensor processing stage.

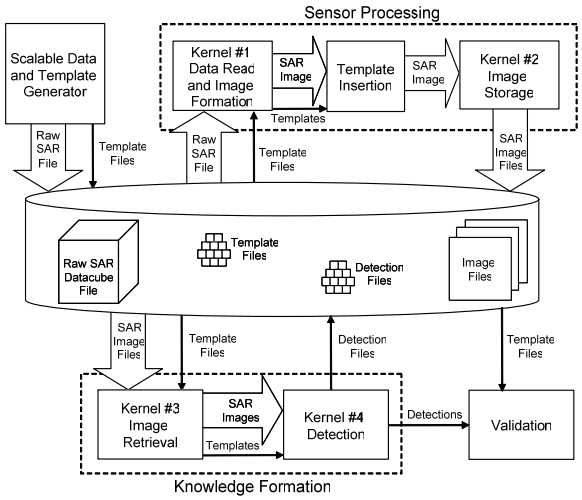


**Figure 1: Block diagram of HPCS SSCA#3 Benchmark**

## Approach to Parallelization

A quick profile of the serial execution indicates that the bulk of the run time is spent generating the 'raw' SAR data using the Scalable Data Generator (SDG) module, and forming the SAR image (Kernel 1). The main loop of both Stage 1 and Stage 2 appeared to be straightforward to parallelize, while the SDG required slightly more care to correctly collect and sum the contributions of different reflectors.

The serial code is written in MATLAB® and the parallel version used the pMatlab toolkit.[2] The choice of code environment reflects the growing use of MATLAB as a rapid, highly productive development environment for both proof of concept and production. Similarly, pMatlab provides a rapid serial-to-parallel porting of vetted code. Specifically, the use of parallel global array semantics (PGAS) within pMatlab better reflects the engineer's mathematical and physical models of the simulation as compared with traditional local-global mapping and MPI. The global array semantics and distributed maps take care of the local-to-global mapping and communication, allowing the user to rapidly parallelize a serial code with

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendations are those of the author and are not necessarily endorsed by the United States Government

minimal code changes. The function overloading capability of MATLAB allows the user to call the MATLAB functions of their serial code with the distributed maps

Initially the benchmark was parallelized along strictly parameter sweep, or embarrassingly parallel lines, focusing on three main modules; the SDG, the image formation (Kernel 1) and target detection (Kernel 4). The parallelization of Kernels 1 and 4 used a simple loop distribution strategy such that each processor formed a block of images, and all computation for a given image remained within the processor. (The I/O is thus performed by each processor as it reads in data, processes it and writes the result to disk.) The SDG required the distribution, gathering, and summing of the data calculation. In total, this approach to parallelization resulted in the creation of three distributed arrays, one for each module of the computation stages, and one for the SDG. The number of new lines of pMatlab code was less than 30 in a code base of over 1400 lines of MATLAB code.

## Parallel Results

Preliminary tests used the LLGrid cluster consisting of Dell PowerEdge dual-processor Xeons connected by Gigabit Ethernet.[3] Initial tests of the SDG parallelization used up to 16 processors and a scale factor of 10 corresponding to an image size of roughly 2.5K x 3.8 K pixels. Speedup results indicate that linear speed-up is achieved.

The results presented here are from slightly larger runs using up to 64 processors and a scale of 12 which generates an image of approximately 3K by 4.5K pixels. (This pixel size also corresponds to the size of the 2D FFT computed as part of the image formation). In Figure 2, we see that close to linear speedup was achieved for the image formation kernel. The target detection speed-up, while almost linear for 2 processors, does not perform as well as for image formation, likely due to the variation of the image depths of across image sequences, leading to load imbalance.

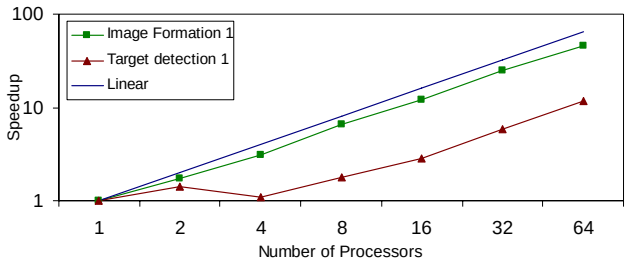


**Figure 2: Speedup for 2, 4, 8, 16, 32, and 64 processors**

The Overall Compute and I/O mode performance results are presented in Figure 3. As in Figure 2, the Overall Compute (image formation and target detection) performance shows close to linear speedup. The Overall I/O includes all I/O in Stage 1 and Stage 2. In Stage 1 the raw SAR data is read in (Kernel 1), and the image is written out to disk (Kernel 2). In Stage 2 all the z images (depth) are read in for a given x, y location (Kernel 3). After the targets have been detected the sub-images containing the targets are written to disk (Kernel 4). From one image sequence to another the depth varies and therefore the number of images to be analyzed can vary greatly from processor to processor. The resulting load imbalance indicates a trend away from linear speedup as the number of processors increases.

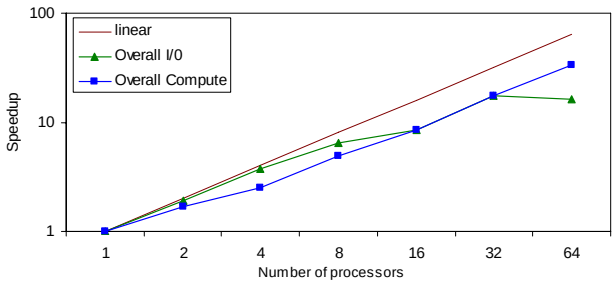


**Figure 3: Overall File I/O and Computation Performance for 2, 4, 8, 16, 32, and 64 processors**

## Summary

The HPEC Challenge SAR Benchmark has been parallelized using pMatlab. This benchmark highlights several key challenges in parallel processing; the creation and analysis of large images (and thus computation of large 2-D FFTs), load imbalances (as in the detection kernel mentioned above), and various file I/O issues relating to reading and writing of many small files as well as individual large files. The parallel code achieved linear speedup with the addition of less than 30 lines of new code (approximately 2% of the original serial code).

## References

[1] Ryan Haney, Theresa Meuse, Jeremy Kepner and James Lebak, "The HPEC Challenge Benchmark Suite", *High Performance Embedded Computing (HPEC) Workshop 2005*, September 2005.

[2] N. Travinin, R. Bond, J. Kepner, H. Kim, R. Haney, "pMatlab: High Productivity, High Performance Scientific Computing", *SIAM CSE 2005*, February 12-15, 2005, Orlando, FL

[3] A.I. Reuther, T. Currie, J. Kepner, H.G. Kim, A. McCabe, M.P. Moore, N. Travinin, "On-Demand Grid Computing Using gridMatlab and pMatlab," In *Proceedings of the HPCMO Users Group Conference*, Williamsburg, VA, 8 June 2004.