Octave and Python: High-Level Scripting Languages Productivity and Performance Evaluation

Juan Carlos Chaves, John Nehrbass, Brian Guilfoos, Judy Gardiner, Stanley Ahalt, Ashok Krishnamurthy, Jose Unpingco, Alan Chalker, Andy Warnock, and Siddharth Samsi
Ohio Supercomputer Center, Columbus, OH
{jchaves, nehrbass, guilfoos, judithg, ahalt, ashok, unpingo, alanc, awarnok, samsi}@osc.edu

Abstract

Octave and Python are open source alternatives to MATLAB, which is widely used by the High Performance Computing Modernization Program (HPCMP) community. These languages are two well known examples of high-level scripting languages that promise to increase productivity without compromising performance on HPC systems. In this paper, we report our work and experience with these two non-traditional programming languages at the HPCMP Centers. We used a representative sample of SIP codes for the study, with special emphasis given to the understanding of issues such as portability, degree of complexity, productivity and suitability of Octave and Python to address Signal/Image Processing (SIP) problems on the HPCMP HPC platforms. We implemented a relatively simple two-dimensional (2-D) FFT and a more complex image enhancement algorithm in Octave and Python and benchmarked these SIP codes on several HPCMP platforms, paying special attention to usability, productivity and performance aspects. Moreover, we performed a thorough benchmark containing important low level SIP core functions and algorithms and compared the outcome with the corresponding results for MATLAB. We found that the capabilities of these languages are comparable to MATLAB and they are powerful enough to efficiently implement complex SIP algorithms. Productivity and performance results for each language vary depending on the specific task and the availability of high level functions in each system to address such tasks. Therefore, the choice of the best language to use in a particular instance will strongly depend upon the specifics of the SIP application that needs to be addressed. We concluded that Octave and Python look like promising tools that may provide an alternative to MATLAB without compromising performance and productivity. Their syntax and functionality are similar enough to MATLAB to present a very shallow learning curve for experienced MATLAB users.

1. Introduction

The new emphasis of high end computing systems is rapidly evolving towards productivity and value rather than traditional HPC standards such as raw theoretical peak computing performance. Total end-user computing life-cycle costs and mission responsiveness are becoming increasingly critical to operational scenarios of modern Department of Defense (DoD) and homeland defense systems. To address these urgent but complex needs, researchers' idea-to-solution or time-to-solution is becoming more important than raw computing capacity. Ultimately, the goal is to decrease the time-to-solution, which means decreasing both the execution time and development time of an application on a particular system.

There is an increasing recognition that high-level languages, and in particular, scripting languages such as MATLAB, Octave, and Python may provide enormous productivity gains in developing technical and scientific code. With the HPC emphasis rapidly shifting to high productivity metrics, where productivity and value are more important than raw performance; modern high-level languages promise to make HPCs easier and more productive to use. As clearly demonstrated by the immense success of products such as MATLAB, time to solution is becoming one of the major metrics of value to technical users, which 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. High-level scripting languages promise to decrease time to solution in HPC systems by promoting ease of use, code reusability, transparent access to highly

HPCMP Users Group Conference (HPCMP-UGC'06)
0-7695-2797-3/06 $20.00 © 2006 IEEE

optimized libraries, portable performance and isolation from the inherent complexities of HPC low level programming. In addition, MATLAB, Octave and Python enjoy a very large and active open source user community that constantly contributes algorithms and improvements to the base products. Of course, in the case of MATLAB there is also commercial support for the parent company, The MathWorks, and several third party companies that produce a wide variety of toolboxes (collections of specialized application code). This makes these languages a very attractive option to address the complex computational and analysis challenges of the SIP, IMT, CEA, CCM, and other communities.

Until recently, the technical community mostly used high-level scripting languages for serial code development in high end PCs and workstations. This limited its use to performing prototyping studies and low scale studies. If a user needed to perform realistic simulations or process very large datasets the execution time could be weeks or even months. If the dataset sizes were too large to load into the desktop memory or the results were required in hours instead of days, the only viable option was to translate the code into C or FORTRAN and parallelize the resulting code by hand using low level programming models like MPI or OpenMP, and then execute on a batch oriented HPC system. Needless to say, this approach is very expensive, error-prone, and time-consuming. Moreover, this approach tends to shift the focus from the computational science problem to a very complex parallel programming task with the undesired consequence that the time to solution dramatically increases. Each of these steps may take several months, therefore scientists and engineers are limited to how much iteration to the algorithms and models they may make. Notice that this all happens before they ever get to actual utilization of their models, solving the problems they have set out to solve. More than 75 percent of the time to solution is spent programming the models for use on HPC platforms, rather than developing and refining them up front, or using them in production mode to make decisions and discoveries. Fortunately, as demonstrated by the success of products such as MATLAB and its parallel extensions, high-level scripting languages are slowly starting to evolve into valuable HPC languages that may enable a very productive computing environment in which the user becomes empowered as the borders between the desktop and the HPC environment blur and time to solution decreases dramatically.

We looked at rapid prototyping languages with respect to portability, suitability to number crunching, and the size of the user community. Based on these criteria we decided to investigate two languages: Octave and Python. We endeavored to evaluate the usability, portability, performance, and scalability aspects of Octave and Python on HPCMP resources versus the MATLAB standard with special emphasis in usability and productivity aspects of these two packages.

2. Methodology

2.1. 2D FFT

To begin testing the feasibility of Octave and Python for HPCMP platforms, a simple SIP algorithm was implemented in each language. The algorithm is the two-dimensional fast Fourier transform (2D FFT). For this relatively simple task, Octave and Python appeared equally easy to use. Similarly as with MATLAB, both languages have the advantage of command-line interpreters for testing code. Also, like MATLAB, Octave and Python have access to optimized 2D FFT algorithms that are ready-to-use and much faster than manually coded implementations.

2.2. Pattern Matching Algorithm

To further test the feasibility of Octave and Python for HPCMP platforms, a more complex SIP algorithm was then implemented in each language. The algorithm is a pattern matching algorithm in which a template image is located within a field image. The particular algorithm we used is based on the paper Real-Time Pattern Matching Using Projection Kernels by Yacov Hel-Or and Hagit Hel-Or (IEEE Transactions on Pattern Analysis and Machine Intelligence, 27:9, September 2005). The algorithm uses an efficient scheme to project both the template image and windows, or areas, of the field image onto two-dimensional Walsh-Hadamard (WH) kernels. A lower bound between the Euclidean distances of the template and windows of the field may be calculated from these projections. Field windows with low distances to the template are possible matches. Only the first few projections are needed for good performance. The first projection may be omitted to obtain a pattern matching algorithm that is invariant with respect to illumination, though this can sometimes lead to poorer results in general. The time complexity of computation may be reduced by two orders of magnitude compared to traditional approaches, though it uses more memory. One limitation of this algorithm is that the template must be square with side lengths that are a power of two.

Our algorithm searches for the window in the field with the lowest distance from the template using three to four WH kernel projections. It can search across different scalings and clockwise rotations of the template that are specified by the user. Actually, the algorithm scales and rotates the field for better accuracy and because of the restrictions on the template size, but conceptually this can

be thought of as scaling and rotating the template. The algorithm assumes that there is at most one instance of the template in the field, and that this instance lies entirely within the image. If it finds an instance of the template in the field, it creates an image file containing the grayscale version of the field with the located template pattern outlined in red. If the field window with the lowest distance results in a match that lies only partially in the field, the algorithm reports that no match was found and does not create an output image.

Overall, Python seemed to be the best language for this application. Python has a command-line interpreter that can be used to test small bits of code, and this speeds up development. The Python language also has very good, built-in support for list types that make complex structures easy to manage. It is simple to access, add, and subtract items from list and sequence types, and it is easy to iterate over a list. The support for classes makes code more manageable and makes code reuse easier. For this particular application, the Python Imaging Library is a bug-free and easy way to access and manipulate images. Installation of the Python Imaging Library did pose some problems, but they were resolved.

Octave, like Python, does have a command-line interpreter. Unfortunately, it does not support classes, thus making the code less organized and harder to reuse. Moreover, for this algorithm, it required several external applications like OctaveForge and ImageMagick, making the already difficult installation of Octave even more difficult. Octave is also the slowest to execute for this algorithm. One upside is that Octave code is very similar to MATLAB, so MATLAB code that does not use classes or other unsupported functions can be transferred to Octave quite readily. However, the difficulties in installation as well as the long execution times made Octave a difficult choice for this application.

2.3. Benchmarks

For this study, three sets of benchmarks were run for Octave, Python and MATLAB on a variety of HPCMP Linux clusters across the country: Powell and JVN at ARL MSRC, HHPC at AFRL/IF, and Seafarer at SSC-SD. The first set is for the 2D FFT, the second is for the pattern matching algorithm, and the third is a set of general benchmarks that were originally available for Octave and MATLAB and we ported to Python.

2.3.1. 2D FFT Benchmarks

Table 1 shows the average runtimes for the 2D FFT for each language on various HPCMP platforms. The data show that Octave, Python, and MATLAB are fairly close in performance, with Octave being slightly faster on some machines and Python on others. The reason for this is that Python, Octave, and MATLAB have FFT functions either built-in or as part of a library. These FFT functions are actually using interfaces to FORTRAN for Octave, C code for Python, and probably optimized C code for MATLAB. As it is easy to appreciate, this is a clear instance where Octave or Python are excellent alternatives to MATLAB. For example on the Seafarer cluster MATLAB is not available. However, users of this platform still may take advantage of the availability of powerful and easy to use FFT algorithms thanks to the availability of Octave and Python on this machine.

Table 1. Average times over three trials each for the 2D FFT. The 2D FFT was performed three times for each language on random square matrices of image data (values 0–255) with sizes 512×512, 1024×1024, and 2048×2048.

		Octave				MATLAB				Python			
		Powell	JVN	HHPC	Seafarer	Powell	JVN	HHPC	Seafarer	Powell	JVN	HHPC	Seafarer
2D FFT	512	0.129	0.078	0.111	0.139	0.131	0.091	0.160	N/A	0.116	0.076	0.15	0.103
	1024	0.515	0.314	0.55	0.561	0.574	0.461	0.682	N/A	0.469	0.315	0.6142	0.450
	2048	2.112	1.353	2.059	2.253	2.298	1.665	2.416	N/A	1.977	1.306	2.716	1.730
Total		**2.755**	**1.744**	**2.72**	**2.953**	**3.003**	**2.227**	**3.258**	**N/A**	**2.562**	**1.697**	**3.478**	**2.283**
Mean		**0.918**	**0.581**	**0.907**	**0.984**	**1.001**	**0.742**	**1.086**	**N/A**	**0.854**	**0.566**	**1.1593**	**0.761**

2.3.2. Pattern Matching Algorithm Benchmarks

Table 2 shows run times for the pattern matching algorithm. Each time shown in the table is the average taken over three trials. The tests are as follows:

- SIP Application 1 – searches for the template in the field at a rotation of -11º and a scale of 1.1 with no illumination invariance.
- SIP Application 2 – searches for the template in the field at rotations in increments of 1º between -5º and 5º and at scales in increments of .1

between 1 and 1.5 with no illumination invariance.

- SIP Application 3 – searches for the template in the field with no rotation and no scaling with illumination invariance.
- SIP Application 4 – searches for the template in the field at a rotation of 15º at a scale of 1 with no illumination invariance.

Table 2. Average run times over three trials each for the pattern matching algorithm. Mean* is the trimmed geometric mean.

		Octave				MATLAB				Python			
		Powell	JVN	HHPC	Seafarer	Powell	JVN	HHPC	Seafarer	Powell	JVN	HHPC	Seafarer
SIP Application	1	47.8	18.23	N/A	26.76	5.451	2.960	N/A	N/A	22.06	9.605	25.26	18.365
	2	760	328.3	N/A	527.97	100.6	61.71	N/A	N/A	537.1	264.7	589.63	447.765
	3	17.14	7.748	N/A	11.49	1.741	1.109	N/A	N/A	10.02	4.073	8.446	7.111
	4	29.94	13.66	N/A	20.76	3.730	2.308	N/A	N/A	18.1	7.474	20.221	14.984
Total		**854.9**	**368**	**N/A**	**586.98**	**111.51**	**68.08**	**N/A**	**N/A**	**587.3**	**285.9**	**634.55**	**488.225**
Mean		**207.3**	**91.99**	**N/A**	**146.74**	**27.88**	**17.02**	**N/A**	**N/A**	**146.8**	**71.47**	**160.89**	**122.056**
Mean*		**37.83**	**15.78**	**N/A**	**23.57**	**3.030**	**2.295**	**N/A**	**N/A**	**19.98**	**8.473**	**22.601**	**16.588**

Times marked as N/A are unavailable due to installation problems or software unavailability on the specific platform being tested. The data show that MATLAB is much faster than Python and Octave for this application, and Python is substantially faster than Octave. Due to the complexity of the code, it is difficult to determine the exact reason for this. Some possible explanations are that there are substantial speed differences in the many image processing functions available for each language, that memory management is done more efficiently in some languages than in others (this is a relatively memory intensive algorithm), or that due to differences in some of the available image processing functions, extra coding was required in some of the languages. However, we want to emphasize that even for complex problems like the Pattern Matching algorithm Octave and Python are useful alternatives to MATLAB. For example, despite the complete lack of MATLAB and the Image Processing Toolbox on Seafarer, this platform has been enabled for tackling complex SIP problems due to the recent availability of the Octave and Python open source solutions.

2.3.3. General Benchmarks

A series of benchmarks for MATLAB, Octave, and other languages may be found online at http://www.sciviews.org/benchmark/. These benchmarks are more general in nature, though they do focus on matrix operations that are extremely important for SIP and other CTA applications. In order to do matrix operations in Python, the NumPy package was used. Table 3 shows the results for Octave, MATLAB, and Python.

The tests are organized into three categories: matrix calculation, matrix function, and programming. The individual tests are as follows:

- I.1 – Creation, transposition, and deformation of a 1500×1500 matrix.
- I.2 – Creation of an 800×800 normally distributed random matrix and taking the 30th power of all its elements.
- I.3 – Sorting of 2,000,000 random values.
- I.4 – 700×700 cross-product matrix (b = a′ * a).
- I.5 – Linear regression over a 600×600 matrix (b = a\b′).
- II.1 – Fast Fourier transform over 800,000 values.
- II.2 – Eigenvalues of a 320×320 random matrix.
- II.3 – Determinant of a 650×650 random matrix.
- II.4 – Cholesky decomposition of a 900×900 matrix.
- II.5 – Inverse of a 400×400 random matrix.
- III.1 – 750,000 Fibonacci numbers calculation.
- III.2 – Creation of a 2250×2250 Hilbert Matrix.
- III.3 – Grand common divisors of 70,000 pairs (recursively).
- III.4 – Creation of a 220×220 Toeplitz matrix.
- III.5 – Escoufier's method on a 37×37 random matrix.

Table 3. More general benchmarking results. Each entry is an average over three trials. All times are in seconds. Mean* and Overall Mean* are trimmed (two extremes eliminated) geometric means.

		Octave				MATLAB				Python			
		Powell	JVN	HHPC	Seafarer	Powell	JVN	HHPC	Seafarer	Powell	JVN	HHPC	Seafarer
Matrix Calculation	I.1	0.82	0.39	1.85	1.23	0.38	0.20	0.40	N/A	N/A	0.53	1.36	0.99
	I.2	0.09	0.20	0.15	0.07	0.53	0.25	0.58	N/A	N/A	0.20	0.16	0.09
	I.3	0.87	0.58	6.86	0.69	0.66	0.43	0.70	N/A	N/A	0.93	4.01	2.59
	I.4	3.07	3.70	1.76	2.29	0.25	0.22	0.34	N/A	N/A	3.96	4.41	3.53
	I.5	0.67	0.82	0.66	0.54	0.11	0.11	0.13	N/A	N/A	1.57	3.18	2.17
	Mean*	**0.78**	**057**	**1.29**	**0.77**	**0.37**	**0.22**	**0.43**	**N/A**	**N/A**	**0.91**	**2.59**	**1.77**
Matrix Functions	II.1	1.14	0.53	1.14	1.05	0.31	0.18	0.34	N/A	N/A	0.04	0.07	0.05
	II.2	0.86	0.97	2.46	0.67	0.80	0.49	0.88	N/A	N/A	0.74	1.34	0.98
	II.3	0.84	1.03	0.83	0.66	0.10	0.07	0.11	N/A	N/A	0.58	1.27	0.89
	II.4	0.46	1.20	0.39	0.37	0.11	0.09	0.12	N/A	N/A	0.98	2.23	1.58
	II.5	0.53	0.73	0.36	0.42	0.08	0.67	0.08	N/A	N/A	0.43	0.95	0.66
	Mean*	**0.73**	**0.90**	**0.72**	**0.57**	**0.15**	**0.20**	**0.17**	**N/A**	**N/A**	**0.57**	**1.17**	**0.83**
Programming	III.1	0.49	0.54	0.60	0.40	1.11	0.36	1.23	N/A	N/A	0.52	0.53	0.38
	III.2	0.68	0.50	0.57	0.69	0.49	0.32	0.51	N/A	N/A	0.51	0.70	0.60
	III.3	0.37	0.26	0.57	0.26	0.31	1.31	0.38	N/A	N/A	0.01	0.02	0.02
	III.4	2.21	1.16	1.49	1.45	0.00	0.00	0.00	N/A	N/A	0.04	0.12	0.10
	III.5	2.58	2.06	1.40	2.14	0.75	0.40	0.83	N/A	N/A	1.68	3.47	1.80
	Mean*	**0.90**	**0.68**	**0.78**	**0.74**	**0.48**	**0.36**	**0.54**	**N/A**	**N/A**	**0.21**	**0.35**	**0.28**
Total		**15.68**	**14.67**	**21.09**	**12.96**	**5.99**	**5.11**	**6.65**	**N/A**	**N/A**	**12.71**	**23.81**	**16.45**
Overall Mean*		**0.80**	**0.70**	**0.90**	**0.69**	**0.30**	**0.25**	**0.34**	**N/A**	**N/A**	**0.48**	**1.02**	**0.74**

We attempted to write the Python test so that it is coded like it is in Octave or MATLAB. Deviations in test times are then due to either inherent differences in the language, the coding of the algorithm, or the system on that it is run. It should be noted that Octave has an advantage over Python for some of the tests in this benchmark since it has an extensive number of built-in and optimized matrix functions, though the overall trimmed geometric means for Octave and Python are on the same order of magnitude. Also, it is clear that MATLAB is substantially faster than Octave and Python for these set of tests. As mentioned before, the N/A column for MATLAB at Seafarer is due to the unavailability of MATLAB in that platform. Also, for some reason that we could not trace, the set for Python was not able to complete on the Powell cluster. Therefore, we did not include those results. Again, even though performance-wise MATLAB appears superior to Octave and Python for this test, Octave and Python are acceptable alternatives to MATLAB and their importance may not be underestimated, especially on a platform such as Seafarer where MATLAB is not available.

3. Results

For this project we have successfully installed and used Octave and Python to code two SIP algorithms: a simple 2D FFT and a more complex pattern matching algorithm. These algorithms along with another more general benchmark that was originally available for Octave and MATLAB were used to benchmark the languages on various HPCMP systems and compare with the de facto standard for high level scripting languages, MATLAB.

The 2D FFT was relatively simple to code in all three languages. In fact for Octave and MATLAB a single source code may be used. The more complex pattern matching algorithm was more difficult to code and revealed some of the pros and cons of each language. Python and Octave have built-in command line interpreters like MATLAB. Python has support for classes as MATLAB does, while Octave does not. Python also has a great deal of functions available, though some do not come with the standard installation. Octave

also has a lot of functions available, however many are packaged with outside programs such as Octave-Forge and ImageMagick.

The 2D FFT benchmarks showed Python and Octave to be about equally fast as MATLAB. The pattern matching benchmark showed MATLAB to be the fastest, followed by Python, followed by Octave. The algorithm is quite complex, however, so the exact reason for the difference in runtimes between Octave and Python is difficult to determine. In addition, the more general benchmarks, which contain many important low level functions used in SIP problems, showed Octave and Python to be about equally fast. MATLAB's superior performance for the pattern matching benchmark and the more generic benchmarks most likely is due to its just-in-time (JIT) accelerator technology.

Octave and Python have their strengths and weaknesses. Both are powerful enough to implement a complex algorithm in a very efficient manner. We concluded that productivity and performance results for each language vary depending on the specific task and the availability of high level functions in each system to address such tasks. Therefore, the choice of the best language to use in a particular instance will strongly depend upon the specifics of the SIP problems to be solved. However, Octave and Python look like promising tools that may provide an alternative to MATLAB without compromising productivity and with acceptable performance. Most importantly, their syntax and functionality are similar enough to MATLAB to present a very shallow learning curve for experienced MATLAB users and are the sole choices for MATLAB- like programming in several HPCMP platforms not supported by The MathWorks, Inc.

Acknowledgements

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.