Lessons Learned from the NASA-UVA Summer School and Internship Program

Katherine Holcomb
Research Computing
University of Virginia
Charlottesville, Virginia
kah3f@virginia.edu

Jacalyn Huband
Research Computing
University of Virginia
Charlottesville, Virginia
jmh5ad@virginia.edu

Tsengdar Lee
High-End Computing Program
NASA
Washington, D.C.
tsengdar.lee@nasa.gov

ABSTRACT

From 2013 to 2018 the University of Virginia operated a summer school and internship program in partnership with NASA. The goal was to improve the software skills of students in environmental and earth sciences and to introduce them to high-performance computing. In this paper, we describe the program and discuss its evolution in response to student needs and changes in the high-performance computing landscape. The future direction for the summer school and plans for the materials developed are also discussed.

Keywords

computer science education, scientific computing, curriculum development, mentoring.

1. INTRODUCTION

The University of Virginia-NASA Summer School and Internship Program was motivated by perceived gaps in basic software-engineering and high-performance computing skills in students in science programs, particularly environmental and earth sciences. As science moves in an increasingly computational direction, the preparation of students lags further behind the demands of their future research careers. While most engineering undergraduates at the majority of higher-education institutions are required to take some form of introductory programming course, usually taught in a Computer Science department, the same is not true of science students in most cases. If they do take programming courses, they are often taught by science faculty untrained in modern software principles. A further omission is high-performance computing and parallel computing; not only is this rarely taught even to computer-science majors, but as an advanced topic it requires proficiency in basic programming before students are able to assimilate it. Our program was intended to explore ways to address these issues.

The program took place in two phases. In the first phase, which we named the Intensive Summer School for Computing in Environmental Sciences (ISSCENS), we accepted 20 students, with 10 of these selected for subsequent 8-week internships at NASA centers. In the second phase, Advanced Computing for Earth Sciences (ACES), we accepted a minimum of 20 students, all of whom were placed in 8-week internships. ISSCENS ran during the summers of 2013, 2014, and 2015. ACES took place in 2016, 2017 and 2018. Through all sessions of the Summer School we continuously modified the curriculum and emphasis while retaining the basic structure established for ISSCENS.

DOI: https://doi.org/10.22369/issn.2153-4136/11/1/1

For both programs, applicants were required to be enrolled in or to have just completed an academic degree program at a United States institution of higher education. For undergraduate applicants, upper-division students were preferred, but this was not a requirement. During the ISSCENS phase, we accepted international students for the 10 slots that did not lead to NASA internships. For the other ISSCENS students, and for all ACES students, United States citizenship was required since this is necessary for regular NASA internships. Students were housed in a dormitory on the campus of the University of Virginia for the Summer School and were provided breakfasts and lunches on weekdays.

No prior programming experience was required, but many attendees had some minimal exposure, often through simple scripting in a language like MATLAB™, while some, particularly the ACES participants, had significant computing backgrounds. The attendees came from 57 different US academic institutions. In both programs, locally-residing students at the University of Virginia, most of whom were graduate students in the Department of Environmental Sciences, were invited to attend the full program, along with the out-of-town students. Two to five UVA students attended each year. The participants were almost equally divided between male (a total of 79) and female (total 73). We did not formally track ethnicity but the overwhelming majority would be described as white.

2. CONTENT

The program content was based on two courses taught through the Department of Computer Science at the University of Virginia, *Computing as a Research Tool* and *Introduction to Parallel Computing*. *Computing as a Research Tool* was aimed at graduate students who needed to apply computing to their research; basic programming was taught in the student's choice of language from the selection offered, along with Unix command-line skills and using a resource manager. *Introduction to Parallel Computing* teaches high-throughput computing, threaded computing, and MPI programming in a compiled language only (usually plain C). *Computing as a Research Too*l was taught by staff of the Research Computing support group, while *Introduction to Parallel Computing* is taught by a Computer Science faculty member.

In recognition of the fact that formal coursework can never reach all researchers who could potentially benefit, in 2008 the Department of Computer Science developed a week-long, accelerated "High Performance Computing Bootcamp" offered jointly with Virginia

Tech. Taking place in early summer, it was open to all faculty, staff, and students from institutions of higher education in the Commonwealth of Virginia. For a few years this "Bootcamp" alternated between the University of Virginia and Virginia Tech venues, but in 2011 Virginia Tech developed a local version and UVA continued on its own. Staff from Research Computing then began to play an increasingly important role in teaching the Bootcamp. When the ISSCENS program was developed, we condensed our *Computing as a Research Tool* course into the introductory 1.5 weeks and utilized the "Bootcamp" as the final week. In 2016 Research Computing took full responsibility for the Bootcamp and replaced some of the more theoretical material with an introduction to data analytics for the high-performance platform.

The goal of ISSCENS and ACES was to reach students whose universities do not offer coursework such as this at all, or who cannot devote a semester to for-credit courses in computer-science topics. The format was modeled after an accelerated traditional course, with aspects of "flipped" classroom as much as possible. Each class day consisted of a morning series of lectures, with short hands-on exercises at regular intervals to the extent feasible. Afternoons consisted of lab sessions with a set of "homework" problems. A major adaptation over the length of the program was to differentiate beginner, intermediate, and advanced programming projects for each set of topics. This was particularly important for beginning students and significantly increased their satisfaction with the program. Evidence indicates that student comfort with their level of understanding of an aspect of programming correlates best with overall learning [1][2]; so, making sure students master at least the basics of the beginning topics is critical for their success. Motivation is another key factor in student learning [3] but our attendees generally were highly motivated to enhance their career skills. In fact, one student criticism of our program was that the assignments were not based on "real" research problems.

3. STRATEGY

We settled on teaching Python as the common language. Python is easy for most students to learn yet has sufficient power to serve as an introduction to modern programming and basic software engineering. Its popularity has grown substantially at high-ranked computer science programs in the United States, passing Java recently as the most popular language taught [4]. It is widespread in online introductory courses from EdX, Udacity, Coursera, MIT Open Courseware, etc. It has also exploded in popularity in many sciences. In particular, it is displacing the commercial software IDL for areas of interest to NASA, specifically Earth Sciences and Astronomy/Astrophysics, particularly for data analysis. The ready availability and ease-of-use of packages such as astropy (astronomy) [5], Basemap [6], xarray [7], and many others for Earth sciences, make it well suited for students in those fields. It is also free and open source so that students are able to install it on their personally-owned laptops without licensing expenses or concerns. Finally, while the critical parts of data-mining and machine-learning systems such as Theano [8] or Tensorflow [9] are generally written in a compiled language such as C++, most users interact through these packages' Python bindings. On modern computing systems, the general slowness of Python (and most other interpreted languages) is usually not a significant issue.

Once Python was chosen, it was necessary to select a version and then a distribution. We chose the older Python 2.7 simply because some packages had not been ported to Python 3.3 and up at the time, on at least one platform (e.g. Basemap was not ported to 3.N on Windows). At all points at which differences are significant, we taught both forms. By the time Python 2.7 reaches end-of-life in 2020 [10] we expect all packages to have been ported or replaced; for instance, Cartopy [11] should have implemented all features of Basemap. In 2018 we allowed students to choose but remained with a base of Python 2.7, with an increase in discussion of maintaining code for both versions [12].

Since our emphasis was on software development with an eye toward more complex code projects, we focused on the Spyder IDE [13] rather than the popular Jupyter notebook. Jupyter is oriented toward data exploration and distribution of a "narrative" of code and analysis, whereas Spyder is a lightweight but traditional IDE with many features helpful for code development, including variable and object viewers and direct access to the built-in debugger and profiler. Spyder also marks syntax errors dynamically and can look up and show names and documentation for functions in modules and packages on the fly. We demonstrated use of Jupyter during exercises with the statistical package Pandas, but mostly used Spyder. Students generally find Spyder a comfortable working environment and often prefer it to Jupyter.

For the Python language support, we quickly settled on the Anaconda distribution from Continuum Analytics [14]. This distribution is comprehensive, cross-platform, and usually very easy to install, even for novices. Over the past six years Continuum has improved the usability of their application and Anaconda now provides a graphical interface for managing packages, a significant improvement over the manual procedure using their Conda package manager from a command line [15]. Conda is still functional for standalone applications, with more powerful features such as conda environments and "pinning" package versions in an environment. There is also the pip installer for packages they do not support directly [16], but for beginners a graphical package manager is very helpful.

Modern Fortran, taught in an accelerated fashion, was used as the class compiled language for the first four sessions. Students who needed to learn it would be motivated to continue, using resources we provided as well as their own references, while students who would not need it (or would need it only occasionally) gain some exposure without being forced to spend too much time with it. Fortran is widely used in the Environmental/Earth Sciences community. There are also many legacy codes written in Fortran that students are likely to encounter if they remain in their fields. However, it is rarely taught except to advanced students majoring in atmospheric sciences or meteorology, and even then it often is not taught particularly well. Modern Fortran (the 2003 standard and up), includes many features of newer programming languages, including arrays as a container data structure, modules, subprogram interfaces, and more sophisticated data structures, including classes. We particularly emphasized array operations, which in our testing using recent compilers have proven to be remarkably fast. Obsolete constructs were described so students would recognize them, but we did not use them in examples or exercises.

For the last two years we taught Fortran and C++ side by side, allowing students to choose the one most useful to their research goals.

For the final week of each session, in which high-performance and parallel computing were introduced, we supported C/C++, Fortran, Python, and sometimes R.

4. CURRICULUM

The program started the Wednesday following Memorial Day and ran for two and a half weeks. We began with three days of instruction in a common scripting language (i.e., Python). The next Monday and Tuesday were devoted to object-oriented programming concepts and an introduction to software engineering. The next day was generally focused on advanced visualization, with instruction in compiled-language programming for the last two days of the week. The last week was variable over the life of the project; but in the three ACES sessions it consisted of Unix, bash scripting, and use of a queueing system on Monday, either optimization and high-throughput computing or data analytics on Tuesday, MPI usually on Wednesday and Thursday, and programming using a multicore paradigm usually on Friday.

The curriculum evolved significantly from the first session in 2013. For the summers of 2013-2015 students applied directly to the ISSCENS program, and we then recruited NASA mentors for 10 of them. As a result, most of the applicants to ISSCENS had little to no programming experience and we spent more time on basic skills. We also included a session on scripting ESRI's ArcGIS Desktop with Python. We spent three days on a mixture of Fortran and Unix skills. The final week, the "High-Performance Computing Bootcamp," focused on high-throughput computing, using a resource manager, code optimization with an emphasis on compiled languages, OpenMP, and MPI.

For the second phase (ACES), students applied directly to NASA centers through the One-Stop Shopping Initiative. The mentors were recruited by NASA education officers, and the students were selected by the mentors. This changed the typical background of the students, since mentors usually wanted students who were more prepared to work on computational projects. Consequently, we reorganized the ISSCENS curriculum. We continued to teach three days of introductory Python and one day of object-oriented Python, but we increased the material for the "software engineering" day, and upped the sophistication of the "advanced visualization" day. Fortran was consolidated into two days, with Unix and bash moved to the first day of the "HPC Bootcamp." Coinciding with the first ACES session in 2016, Research Computing took over full responsibility for the "Bootcamp" from the Computer Science Department and drastically reduced the amount of theory in the instructional materials, substituting more practical and hands-on content in its place. The Unix/bash/SLURM session enabled students to use larger systems at NASA immediately upon arrival at their internship, if appropriate to their projects. Most ACES students, even those with computational experience, arrived never having used anything other than personal laptops or institutional desktops, so this was also essential preparation for the rest of the week. In 2017, in recognition of the growing importance of "big data" analytics and machine learning, we devoted the second day of the "Bootcamp" to this topic, including some exercises in applying Tensorflow to remote-sensing images. We discussed how cloud resources, such as Amazon Web Services, could be used for implementations of Spark Machine Learning.

In 2017 we also combined serial optimization with OpenMP/multiprocessing (the latter for Python users). Serial optimization is especially important for Python programmers; so we increased the content in that area beyond the original Computer Science material. We reduced the MPI introduction to a minimum and provided information about accelerators such as the Intel Knight's Landing and NVIDIA GPGPUs, although we had no KNL nodes and only two NVIDIA-equipped nodes at the time, which made direct experience difficult. In any case, the students' general lack of computer-science background and, often, C/C++ programming skills, as well as the short time available to devote to accelerators, prohibited any attempt to teach CUDA programming for GPUs. Therefore, our focus was on introducing OpenACC [17] instead

For 2018 we "packaged" different portions of the curriculum into workshops that we offered to the larger University of Virginia community. The only session we did not open to University attendees at that time was the "software engineering" session. It is important to note that we taught little modern software engineering. Our emphasis is on readable code, careful consideration of code data and algorithmic structures, team development, and testing and debugging skills.

The HPC "Bootcamp" for 2018 was broken into three logical portions, each of which was accessible to all UVA students and faculty as well as to ACES students. By that time we also had accelerator hardware available for hands-on access, and so had the opportunity to restructure appropriate sections to take more advantage of these devices, including KNL. We also acquired more GPUs on our HPC system so that we demonstrated Tensorflow in its most usual environment (in 2017 we used a core-only version). The "big data on HPC" day was also a workshop available to UVA affiliates. The final segment was a three-day block; the first day covered serial optimization, OpenMP, and OpenACC. The two following days introduced MPI, including mpi4py [19]; the introduction to the Intel Knight's Landing MIC occurred on the final day, along with PGAS concepts such as co-array Fortran [20][21] and UPC++ [22]

5. OUTCOMES

Our effectiveness was variable and depended very much on the background of individual students. The biggest challenge we faced throughout the program was that the participants' experience, interests, and aptitudes ranged from complete beginners who had never programmed in any language to students finishing a master's degree in computer science. The intention of the program was to train students in HPC, but students who have not mastered basic serial programming have very little ability to absorb parallel computing, particularly in an intensive setting with a short timeframe.

Assessments administered to the students showed that the majority felt that the pace was reasonable; it was fast, but they felt they could go back later to pick up more details. In aggregate, approximately 10% thought the pace was too fast, and 5% thought it too slow.

In the 2017 session we forced students to work in small teams on a programming project for the "software engineering" session. Somewhat to our surprise, most of the students had never worked collaboratively on programming, where one programmer must code to an API that other programmers established. Many of the students felt that this was one of the most valuable learning experiences of the entire program.

The most important assessment is their performance in their later research and internships. For the first three summers only 10 students went to NASA internships sponsored by us, but many went to internships with other organizations. Others returned to their research. For the final summers all but one non-UVA attendee went to NASA internships. Student assessments were quite positive of their internships with one exception. Assessments were anonymous but several alumni contacted us personally to state that their internships would have been much more difficult without the training program.

Example quotes:
"I was able to make more progress than my mentor had expected."
"I know that I would have been woefully unprepared for this summer if not for the lessons and guidance provided to us through ACES."

Several NASA mentors also requested students who had participated in our program in previous years. In at least one case, a mentor also sent an intern from the previous summer to ACES before the student's second summer internship.

In a few cases, our program was career-altering. During the first ISSCENS session, an attendee who had never programmed before discovered she had aptitude in that area, and reoriented her area of interest in her field of hydrology toward computational modeling.

Another outcome of this project was a professionally-produced video series on introductory Python, aimed specifically at scientists and engineers and therefore emphasizing NumPy, SciPy, Pandas, and other packages and techniques used heavily by scientific applications. Seven students in an advanced chemistry course at the University of Virginia were recruited as a focus group to use this series to learn to program using these videos. Even those who had no programming experience were able to contribute to a final group programming project for the course.

6. FUTURE OF THE PROGRAM

Once we opened the "packaged" sessions to University of Virginia affiliates, we found that they were extremely popular, so we have continued the program as the Research Computing Summer Education Series (SES). For 2019, with no ACES students attending, we condensed the Python sessions to 3 days, largely by eliminating some topics that are primarily of interest to environmental-science students as well as a few more advanced topics. Response was so enthusiastic that we had to find a larger auditorium. The C++/Fortran short course also proved popular. Inspired by our success with Python, we added short courses in R and MATLAB™. We opened the "software design" short course and it attracted considerable interest as well. We augmented the last week's HPC training with more data analytics, including machine learning, and added short courses in image processing and bioinformatics, the latter emphasizing HPC applications.

A major difference between the Summer School students and the UVA-only attendees was the level of motivation and distraction. Summer School attendees expected to devote their entire day to the material, were motivated to learn in order to do well at their internships, and maintained focus. Summer Education Series attendees were not as reliable at returning for lab sessions. Lab attendance was very good for Python and R, less so for compiled languages and advanced topics, and highly variable for HPC-specific topics. SES students also tended to be low skill. We will adjust our material to reflect that, by incorporating some of our "beginner" level material as exercises within the lecture/hands-on sessions.

We currently have all Summer Education Series course materials online but not in a polished or readily accessible form. Our plan for the next year is to convert the lectures and hands-on materials into a combination of Markdown and, where appropriate, Jupyter notebooks. These will be posted to a public site which will also host our Python video series; this site currently is https://learning.arcs.virginia.edu but this will be consolidated in the near future with other education sites managed by UVA Research Computing. This will make self-guided learning possible. It will also allow us to disseminate the coursework to any institution that would be interested in replicating our series. Only the "Introduction to HPC" session is particularly site-specific, and even that could be easily modified for other sites and resource managers.

7. SUMMARY AND LESSIONS LEARNED

For six sessions over five years we operated a successful summer program to train students, primarily in Earth and environmental sciences, in programming skills, scientific visualization, software design, and high-performance computing. The greatest challenge was to provide a good program for students with widely varying backgrounds, skills, and expectations. We accommodated the diversity by increasing hands-on exercises interspersed with lectures and by expanding our bank of programming exercises to span a range of abilities. We also encouraged students to work more in small groups rather than individually.

The most important conclusion we have drawn is that it is possible to provide students a "crash course" in programming that enables even beginners to handle research-level scientific programming tasks, and for more advanced students to produce near professional-quality software. From our experience, a one- to three-day targeted course seems to well serve the needs of researchers and research students. They rarely have the time or opportunity to take for-credit courses in programming, yet self-teaching or the very short (less than a day) workshops frequently offered often do not adequately prepare them for real-world programming. Many of our students have also expressed dissatisfaction with well-known online courses and found they achieved more when assistance was available, even if they were responsible for most of the material on their own. Over the course of the programs we also moved to include more team-based programming projects, more basic to intermediate projects so that beginners can progress more smoothly without too large a jump in difficulty, and greater integration of hands-on exercises and projects within the "lectures" rather than having dedicated lecture sessions each day. These changes considerably enhanced student satisfaction and assimilation. One clear pattern emerged, however, and that is the importance of the programming exercises, ideally including the more complex "homework" problems. For those, the availability of expert assistance is extremely helpful, and we would advise other groups wishing to undertake a similar experiment to prioritize face-to-face assistance.

8. ACKNOWLEDGMENTS

We thank the education officers and intern coordinators at the NASA centers that have hosted our students, especially Blanche Meeson and Mablelene Burrell of Goddard Space Flight Center in Greenbelt, Maryland, who worked to recruit mentors for our students and to make sure they had an enriching internship experience. We are grateful to the many NASA scientists who mentored our students. We also thank Professors Andrew Grimshaw and Aaron Bloomfield for developing and delivering the original material for the "HPC Bootcamp." This work was supported by NASA grants NNX16AB18G and NNX12AP99G.

9. REFERENCES

[1] Bergin, Susan, and Reilly, Ronan 2005. Programming: factors that influence success. *ACM SIGCSE Bull.* 37, 1, 411-415. DOI=http://dx.doi.org/10.1145/1047124.1047480.

[2] Wilson, Brenda 2002. A study of factors promoting success in computer science including gender differences. *Computer Science Education* 12,1-2 , 141-164.
DOI=http://dx.doi.org/10.1076/csed.12.1.141.8211

[3] Liu, Ou Lydia, Bridgeman, Brent, and Adler, Rachel M. 2002. Measuring learning outcomes in higher education: motivation matters. *Educational Researcher* 41, 9, 352-362.
DOI=https://doi.org/10.3102/0013189X12459679

[4] Guo, Philip 2014. Python is now the most popular introductory teaching language at top US universities. [Online] Available: https://cacm.acm.org/blogs/blog-cacm/176450-python-is-now-the-most-popular-introductory-teaching-language-at-top-u-s-universities/fulltext. [Accessed 26 July 2019]

[5] The Astropy Project. 2016. [Online] Available: http://www.astropy.org/ [Accessed 26 July 2019]

[6] Whitaker, Jeffrey. 2011. Basemap. [Online] Available: https://matplotlib.org/basemap [Accessed 26 July 2019]

[7] Hoyer, Stephan, and Hamman, Joe 2017. "xarray: N-D labeled arrays and datasets in Python." Software Sustainability Institute: *Journal of Open Research Software* 5, 1, 10, 2017. DOI: http://doi.org/10.5334/jors.148

[8] Theano Development Team. 2017. [Online] Available: http://deeplearning.net/software/theano/ [Accessed 26 July 2019]

[9] Abadi, Martin et al. 2015. *Tensorflow: large-scale machine learning on heterogeneous distributed systems*. Preliminary White Paper. Google Research. [Online] Available:

https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf [Accessed 26 July 2019]

[10] Python Software Foundation 2008. PEP 373 – Python 2.7 release schedule. [Online] Available: https://www.python.org/dev/peps/pep-0373/ [Accessed 26 July 2019]

[11] Met Office (UK). 2017. Cartopy. [Online] Available https://scitools.org.uk/cartopy/docs/latest [Accessed 26 July 2019]

[12] Python Software Foundation. 2017. [Online] Available https://docs.python.org/3/howto/pyporting.html [Accessed 26 July 2019]

[13] Pierre Raybaut. 2017. Spyder: The Scientific Python Development Environment [Online] Available: https://spyder-ide.org [Accessed 26 July 2019]

[14] Anaconda, Inc. , 2019. The Anaconda Distribution [Online] Available: https://docs.anaconda.com/anaconda [Accessed 16-Aug-2017]

[15] Anaconda, Inc. 2017. “Conda.” [Online] Available: https://docs.conda.io/projects/conda/en/latest/ [Accessed 26 July 2019]

[16] Anaconda, Inc. 2017. “Managing Packages.” [Online] Available:. https://conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html [Accessed 26 July 2019]

[17] OpenACC-standard.org. 2015. “OpenACC programming and best practices guide.” [Online] Available: https://www.openacc.org/sites/default/files/inlinefiles/OpenACC_Programming_Guide_0.pdf [Accessed 26 July 2019]

[18] Lu, Xiaoyi, Shankar, Dipti, Gugnami, Shashank, and Panda, Dhabaleswar K 2016. High-performance design of Apache Spark with RDMA and its benefits on various workloads. *IEEE International Conference on Big Data (Big Data)*, 253-262. DOI=10.1109/BigData.2016.7840611.

[19] Dalcin, Lisandro. 2019. “MPI for Python.” [Online] Available: http://mpi4py.readthedocs.io/en/stable/ [Accessed 26 July 2019]

[20] Reid, John, and Numrich, Robert W. 2007. Co-arrays in the Next Fortran Standard. *Scientific Programming*, 15, 1, 9-26.

DOI=10.1155/2007/954503

[21] Fanfarillo, Alessandro, Burnus, Tobias, Cardellini, Valeria, Filippone, Salvatore, Nagle, Dan, and Rouson, Damian 2014. OpenCoarrays: Open-source Transport Layers Supporting Coarray Fortran Compilers. In *Proceedings of the 8th International Conference on Partitioned Global Address Space Programming Models* (PGAS '14). ACM, New York, NY, USA, Article 4 , 11 pages.

DOI=http://dx.doi.org/10.1145/2676870.2676876

[22] Zhen, Yili, Kamil, Amir, Driscoll, Michael B., Shan, Hongzhang, and Yelick, Katherine 2014. UPC++: A PGAS extension for C++, *IEEE 28th International Parallel and Distributed Processing Symposium*, pp. 1105-1114.