Implications of the Dataflow Paradigm in the Exascale Era of Scientific Computation

Dylan Fouché
Department of Computer Science
University of Cape Town
fchdyl001@myuct.ac.za

ABSTRACT

The move towards the exascale era of computation has opened up new possibilities in the scientific community. Problems previously labelled as effectively non-computable can now be approximated astonishingly quickly. However, there are challenges associated with this, such as the new design paradigms needed to leverage the processing power of these new massively parallel and heterogeneous distributed systems.

With the use case of high-throughput astronomical computation in mind, we explore the validity of the dataflow model for use in these exascale scientific computations through a review of related literature. The use of interpreted languages for these problems is addressed, as well as the new systemic requirements of exascale computers. A number of successful implementations of this nature are presented, and a set of conclusions are made on its use for a wider range of problems.

CCS CONCEPTS

• **General and reference** → General literature; Performance; • **Computer systems organisation** → *Pipeline computing*; • **Software and its engineering** → **Multiprocessing / multiprogramming / multitasking**; **Data flow architectures**; **Data flow languages**.

KEYWORDS

dataflow, exascale, Python, visualisation, astronomy

1 INTRODUCTION

The Cube Analysis and Rendering Tool for Astronomy (CARTA) [2] is designed to visualise data from the Atacama Large Millimetre Array, the National Radio Astronomy Observatory, and the Square Kilometre Array. As the image size of these modern telescopes has been increasing rapidly over the past years, this analysis and visualisation has become a very computationally expensive task.

With parallelism and scalability as their core values, the current CARTA back-end [8] is written almost entirely in C++ [6]. This is widely regarded as a highly performant language. But the approach to the design of computational models is shifting as we move into the exascale era, and there is evidence that suggests merit to an interpreted approach to solve these problems.

We propose a novel implementation of the CARTA back-end system, using the Python language [24]. Back-end components will be prototyped using the Dask library [10, 11] enabling advanced parallelism and distributed computation. Other libraries such as AstroPy [3, 19] and H5Py [9] will help to abstract away from domain-specific implementation details.

The performance of these prototype components will be compared with the current object-oriented implementation and a set of recommendations will be made. The implications of this are wider than one implementation, however, as we aim to assess the validity of the dataflow model using interpreted language for various high-throughput scientific computations in the exascale era.

2 INTERPRETED LANGUAGES FOR SCIENTIFIC COMPUTATION

In recent years, interpreted languages such as Python have found widespread popularity in the scientific community. The reason is not immediately clear, however, as interpreted languages are commonly slower than compiled languages at run time. Zhang et al [27] demonstrate that, ignoring compile time, Python is significantly slower than C at some typical problem involving repeated floating point multiplication. However, it was also demonstrated that Python's extensibility to compiled libraries (Cython [4], for instance) can improve its performance significantly.

Regardless of the performance implications, Python is still widely used for intensive scientific computations. Oliphant [17] suggests that one reason may be its highly expressive syntax: "comparable to executable pseudo-code". This is attractive to academics who do not have extensive programming experience. Furthermore, the interpreter is platform-independent, meaning that the target architecture need not be considered when developing software. This may be particularly useful in the case of distributed computation.

Pérez et al [18] described Python as "an ecosystem for scientific computing", noting that the flexibility and extensibility of the language may be more important to some than pure performance, especially for once-off computations and simulations. There are a huge number of libraries that are useful for scientific work that can be installed with one call to python's built-in installer, and there is no need to worry about linking binaries at compile time. They also point out that the scientific computing landscape is moving away from just performing floating point arithmetic and towards the use of more complex data structures, many of which Python would have native support for.

Another use case was described by Oliphant [17], where Python is used as a steering language for scientific code written in some other (often compiled) language. Mulder et al [16] introduced the idea of computational steering with a review of the relevant environments and architectures, but this has since become one of the most common uses for Python: as a high-level scripting language and automation tool. But even in entirely Python-based implementations, slight performance overheads are often trumped by ease of development and deployment.

Python may still be optimal for time-constrained algorithms with the help of some libraries designed exclusively to bring parallelism and scalability to the language.

3 THE EXASCALE ERA

An exascale computing system is one that is capable of at least one exaflops. That is, at least 10^{18} floating point operations each second. With specific mention of the Square Kilometer Array, Bobák et al [5] make the observation that modern telescope arrays can produce several petabytes of data each month. With data storage infrastructure struggling to keep up with this massive throughput, it is often necessary to perform some type of pre-processing or feature extraction on the data in real time as it is generated. This lessens the load on the storage infrastructure, but requires compute power at a truly astronomical scale. Even visualising and analysing archived data at this scale is a challenge ill-suited to traditional architectures and algorithmic design.

Shalf et al [21] recognise that the transition to exascale computation looks rather different than the previous transition to petascale computation. Previously, it was an increase in processor clock speeds that allotted faster computing, but now these clock rates cannot increase much further due to the dynamic power wall. Now it is mainly an increase in the number of processing cores that is driving the increase in processing power. Heterogeneity and concurrency are becoming increasingly integral in our modern computing infrastructure, and software design needs to adapt to take advantage of this new technology.

However, this new paradigm is not without its concerns. The issue of resilience is raised by Cappello et al [7], where they argue that the greater number of processors and threads implicitly creates a greater number of potential points of failure. Thus, robustness needs to be a primary concern of exascale architecture.

Entirely new architectures are being developed for exascale computation, such as the "codelet" approach suggested by Lauderdale et al [14]. The idea here is to define the atomic unit of computation as a codelet, which may be implemented with one or more threads. Each codelet behaves like a process that is incapable of blocking and is entirely responsible for its own error handling. They propose that this creates software that is more robust and more easily scheduled in heterogeneous computing systems.

Other architectures have been identified for use in exascale systems, such as the implicitly parallel pipe and filter pattern, sometimes referred to as the dataflow model.

4 THE DATAFLOW MODEL OF COMPUTATION

In the seminal work, Culler [12] suggests thinking of the dataflow model as a directed acyclic graph. Each node is some function, and values (or tokens) travel along the arcs that connect the nodes. This is a fundamentally different way of representing computation, as there is in essence no program counter, the order of execution of instructions is implicitly non-deterministic.

The abstract dataflow model assumes that each arc has unbounded storage, or that infinitely many tokens can be on an a given arc at any point in time. It goes without saying that this is practically impossible, and the way in which this is implemented gives rise to two different dataflow types: the static dataflow model and the dynamic dataflow model. In the static model, the maximum number of tokens per arc is constrained to one, allowing for a much simpler deterministic implementation. The dynamic model labels each token and pools them in a large common memory space, leaving the capacity of each arc essentially unbounded, but also leaving us unable to enforce any particular ordering of tokens on each arc. And as always, non-determinism in concurrent programs means we have to watch out for fatal issues such as bad inter-leavings and deadlock situations.

Verdosica et al [25] suggest that the static dataflow model is a valid approach for exascale computation, on the condition that the computing system is homogeneous. Others propose dynamic dataflow systems to work across heterogeneous compute topologies, such as the extreme-scale many-task system (ExM) proposed by Armstrong et al [1] to abstract away from painful implementation details for massively concurrent distributed algorithms.

Silva et al [22] demonstrated the efficiency of this approach for analysing large raw-data files over distributed systems, which is a typical use case in the astronomical domain.

5 VISUALISATION AND THE ASTRONOMICAL DOMAIN

As mentioned previously, analysing and visualising large raw data files can be very computationally expensive. Resultantly, many have turned to massive parallelism to achieve this in real time or sufficiently close to real time.

Mao et al [15], for instance, have developed a graph-based dataflow model to schedule jobs over a highly distributed computing network in order to process the exascale throughput from the Square Kilometre Array. The authors built upon previous work such as the open-source Celery [23] scheduler developed for the MeerKAT telescope. Furthermore, a number of optimisation methods are proposed for this system including a meta-heuristic approach such as genetic algorithm optimisation, as well as other critical-path aware hierarchical scheduling algorithms.

Wang et al [26] also use a dynamic dataflow model, in this case composing dynamic web services for highly parallel astronomy data processing. They aim to reach one of the primary goals of service-oriented architecture (SOA): allowing end-user service development without manual programming. It is shown that the PMGrid system developed allows for asynchronous parallelism, while increasing computational efficiency and helping to avoid human error.

Dataflow systems for astronomy data processing have been proposed by many others at different implementation levels. Sane et al [20] developed a low-level dataflow system for astronomical digital signal processing targeted at field programmable gate arrays. The VISTA system developed by Irwin et al [13] uses a higher-level pipeline processing system to analyse large volumes of raw infrared (IR) data.

Zhang et al [28] implemented another dataflow system on the massively scalable Amazon Web Servives EC2 cloud compute infrastructure. The authors use a computation model based on directed acyclic graphs, similar to that of Mao et al [15]. Given this, the authors report a 3.7X speedup over a naïve C implementation with multi-threading.

6 CONCLUSIONS

Our aim was to explore the validity of the dataflow paradigm for exascale scientific computation. While there is a much literature describing successful implementations of this nature, there is a lack of modern research describing the efficacy of this approach to exascale scientific computation as a whole.

Many have admitted the performance restrictions of interpreted languages such as Python, and justified their use in the scientific community with ease of use and massive extensibility. Its use may not be so widespread had a vast set of libraries for scientific computation not been developed for the language. The shifting landscape of the exascale era was described, as well as the new challenges it poses for algorithmic design. New highly parallel approaches are required to leverage this increase in computational power, and the dataflow paradigm is one such model that can take advantage of this.

The dataflow model, described originally as a hardware architecture and later adapted as a parallel programming pattern not dissimilar to the pipe and filter model, is said to be highly effective across heterogeneous distributed systems under certain implementation constraints. This model has already been adapted widely in the astronomical domain, from low-level signal processing implementations to massively scalable cloud compute systems. Still, the benefits of using this approach versus the traditional object-oriented design are rarely discussed.

Nevertheless, this appears to be a valid approach to address the new requirements of exascale systems.

ACKNOWLEDGMENTS

This work is based on the research supported wholly or in part by the National Research Foundation of South Africa (grant number MND190716456261).

REFERENCES

[1] Timothy G Armstrong, Justin M Wozniak, Michael Wilde, Ketan Maheshwari, Daniel S Katz, Matei Ripeanu, Ewing L Lusk, and Ian T Foster. 2012. ExM: High level dataflow programming for extreme-scale systems. In *4th USENIX Workshop on Hot Topics in Parallelism (HotPar), poster, Berkeley, CA*.

[2] NRAO ASIAA, IDIA. 2020. *Cube Analysis and Rendering Tool for Astronomy*. https://cartavis.github.io

[3] Astropy Collaboration, T. P. Robitaille, E. J. Tollerud, P. Greenfield, M. Droettboom, E. Bray, T. Aldcroft, M. Davis, A. Ginsburg, A. M. Price-Whelan, W. E. Kerzendorf, A. Conley, N. Crighton, K. Barbary, D. Muna, H. Ferguson, F. Grollier, M. M. Parikh, P. H. Nair, H. M. Unther, C. Deil, J. Woillez, S. Conseil, R. Kramer, J. E. H. Turner, L. Singer, R. Fox, B. A. Weaver, V. Zabalza, Z. I. Edwards, K. Azalee Bostroem, D. J. Burke, A. R. Casey, S. M. Crawford, N. Dencheva, J. Ely, T. Jenness, K. Labrie, P. L. Lim, F. Pierfederici, A. Pontzen, A. Ptak, B. Refsdal, M. Servillat, and O. Streicher. 2013. Astropy: A community Python package for astronomy. 558, Article A33 (Oct. 2013), A33 pages. https://doi.org/10.1051/0004-6361/201322068 arXiv:astro-ph.IM/1307.6212

[4] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D.S. Seljebotn, and K. Smith. 2011. Cython: The Best of Both Worlds. *Computing in Science Engineering* 13, 2 (march 2011), 31 –39. https://doi.org/10.1109/MCSE.2010.118

[5] M. Bobák, L. Hluchy, A. Belloum, R. Cushing, J. Meizner, P. Nowakowski, V. Tran, O. Habala, J. Maassen, B. Somosköi, M. Graziani, M. Heikkurinen, M. Höb, and J. Schmidt. 2019. Reference Exascale Architecture. In *2019 15th International Conference on eScience (eScience)*. 479–487.

[6] British Standards Institute. 2003. *The C++ Standard: incorporating Technical Corrigendum 1: BS ISO* (second ed.). Wiley, New York, NY, USA. xxxiv + 782 pages.

[7] Franck Cappello, Al Geist, Bill Gropp, Laxmikant Kale, Bill Kramer, and Marc Snir. 2009. Toward exascale resilience. *The International Journal of High Performance Computing Applications* 23, 4 (2009), 374–388.

[8] CARTAvis community. 2020. *CARTA Backend*. https://github.com/CARTAvis/carta-backend/

[9] Andrew Collette & contributors. 2020. *HDF5 for Python*. https://www.h5py.org

[10] Dask core developers. 2019. *Dask: Natively scales Python*. https://dask.org

[11] James Crist. 2016. Dask & Numba: Simple libraries for optimizing scientific python code. In *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, 2342–2343.

[12] David E Culler. 1986. Dataflow architectures. *Annual review of computer science* 1, 1 (1986), 225–253.

[13] Mike J Irwin, Jim Lewis, Simon Hodgkin, Peter Bunclark, Dafydd Evans, Richard McMahon, James P Emerson, Malcolm Stewart, and Steven Beard. 2004. VISTA data flow system: pipeline processing for WFCAM and VISTA. In *Optimizing scientific return for astronomy through information technologies*, Vol. 5493. International Society for Optics and Photonics, 411–422.

[14] Christopher Lauderdale and Rishi Khan. 2012. Towards a Codelet-Based Runtime for Exascale Computing: Position Paper. In *Proceedings of the 2nd International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era* (London, United Kingdom) *(EXADAPT '12)*. Association for Computing Machinery, New York, NY, USA, 21–26. https://doi.org/10.1145/2185475.2185478

[15] Yishu Mao, Yongxin Zhu, Tian Huang, Han Song, and Qixuan Xue. 2016. DAG Constrained Scheduling Prototype for an Astronomy Exa-Scale HPC Application. In *2016 IEEE 18th International Conference on High Performance Computing and Communications; IEEE 14th International Conference on Smart City; IEEE 2nd International Conference on Data Science and Systems (HPCC/SmartCity/DSS)*. IEEE, 631–638.

[16] Jurriaan D Mulder, Jarke J Van Wijk, and Robert Van Liere. 1999. A survey of computational steering environments. *Future generation computer systems* 15, 1 (1999), 119–129.

[17] Travis E Oliphant. 2007. Python for scientific computing. *Computing in Science & Engineering* 9, 3 (2007), 10–20.

[18] Fernando Perez, Brian E Granger, and John D Hunter. 2010. Python: an ecosystem for scientific computing. *Computing in Science & Engineering* 13, 2 (2010), 13–21.

[19] A. M. Price-Whelan, B. M. Sipőcz, H. M. Günther, P. L. Lim, S. M. Crawford, S. Conseil, D. L. Shupe, M. W. Craig, N. Dencheva, A. Ginsburg, J. T. VanderPlas, L. D. Bradley, D. Pérez-Suárez, M. de Val-Borro, (Primary Paper Contributors, T. L. Aldcroft, K. L. Cruz, T. P. Robitaille, E. J. Tollerud, (Astropy Coordination Committee, C. Ardelean, T. Babej, Y. P. Bach, M. Bachetti, A. V. Bakanov, S. P. Bamford, G. Barentsen, P. Barmby, A. Baumbach, K. L. Berry, F. Biscani, M. Boquien, K. A. Bostroem, L. G. Bouma, G. B. Brammer, E. M. Bray, H. Breytenbach, H. Buddelmeijer, D. J. Burke, G. Calderone, J. L. Cano Rodríguez, M. Cara, J. V. M. Cardoso, S. Cheedella, Y. Copin, L. Corrales, D. Crichton, D. D'Avella, C. Deil, É. Depagne, J. P. Dietrich, A. Donath, M. Droettboom, N. Earl, T. Erben, S. Fabbro, L. A. Ferreira, T. Finethy, R. T. Fox, L. H. Garrison, S. L. J. Gibbons, D. A. Goldstein, R. Gommers, J. P. Greco, P. Greenfield, A. M. Groener, F. Grollier, A. Hagen, P. Hirst, D. Homeier, A. J. Horton, G. Hosseinzadeh, L. Hu, J. S. Hunkeler, Ž. Ivezić, A. Jain, T. Jenness, G. Kanarek, S. Kendrew, N. S. Kern, W. E. Kerzendorf, A. Khvalko, J. King, D. Kirkby, A. M. Kulkarni, Astropy Contributors, et al. 2018. The Astropy Project: Building an Open-science Project and Status of the v2.0 Core Package. 156, Article 123 (Sept. 2018), 123 pages. https://doi.org/10.3847/1538-3881/aabc4f

[20] Nimish Sane, John Ford, Andrew I Harris, and Shuvra S Bhattacharyya. 2012. Prototyping scalable digital signal processing systems for radio astronomy using dataflow models. *Radio Science* 47, 03 (2012), 1–14.

[21] John Shalf, Sudip Dosanjh, and John Morrison. 2011. Exascale Computing Technology Challenges. In *High Performance Computing for Computational Science – VECPAR 2010*, José M. Laginha M. Palma, Michel Daydé, Osni Marques, and João Correia Lopes (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–25.

[22] Vitor Silva, Daniel de Oliveira, and Marta Mattoso. 2014. Exploratory analysis of raw data files through dataflows. In *2014 International Symposium on Computer Architecture and High Performance Computing Workshop*. IEEE, 114–119.

[23] Ask Solem et al. 2013. Celery: Distributed Task Queue. *URL http://docs. celeryproject. org/en/latest/index. html* (2013).

[24] G. van Rossum. 1995. *Python tutorial*. Technical Report CS-R9526. Centrum voor Wiskunde en Informatica (CWI), Amsterdam.

[25] Lorenzo Verdoscia and Roberto Vaccaro. 2013. Position paper: Validity of the static dataflow approach for exascale computing challenges. In *2013 Data-Flow Execution Models for Extreme Scale Computing*. IEEE, 38–41.

[26] Man Wang, Zhihui Du, Yinong Chen, Shihui Zhu, and Weihua Zhu. 2007. Dynamic dataflow driven service composition mechanism for astronomy data processing. In *IEEE International Conference on e-Business Engineering (ICEBE'07)*. IEEE, 596–599.

[27] H. Zhang and J. Nie. 2016. Program performance test based on different computing environment. In *2016 IEEE International Conference of Online Analysis and Computing Science (ICOACS)*. 174–177.

[28] Zhao Zhang, Kyle Barbary, Frank Austin Nothaft, Evan Sparks, Oliver Zahn, Michael J Franklin, David A Patterson, and Saul Perlmutter. 2015. Scientific computing meets big data technology: An astronomy use case. In *2015 IEEE International Conference on Big Data (Big Data)*. IEEE, 918–927.