# A Software Ecosystem for Scientific Streams– Developing and Maintaining Community Software

David A. Bader
*New Jersey Institute of Technology*
*Newark, New Jersey 07102*
*Email: bader@njit.edu*

## 1. Challenges of Scientific Streaming Software

Data exploration [1] has become a new science paradigm. Furthermore, more and more emerging applications, such as from astrophysics, social networks, cybersecurity, and bioinformatics, generate data that often arrive in the form of streams. To explore such in-motion data, there are two challenges: (1) scalable software supporting real-time and low latency data processing since some unique scientific values will lose their utility after a short time window. (2) reliable software to draw correct and accurate scientific conclusions.

For example, in multi-messenger astronomy [2], low latency data processing may enable us to collect different messages (gravitational waves, magnetic and electronic signals) from the same events and this will help us to learn the science behind the events. For many countries, the COVID-19 infection curves reveal a remarkable linear growth over extended time periods. This observation is practically impossible to understand with traditional epidemiological models [3]. How can we know if the conclusion from the software is correct or not [4]?

The huge volume of streaming data is a grand challenge for existing software to achieve sufficient performance to control the latency. At the same time, testing based software reliability evaluation [5] often cannot meet the requirements of correctness and accuracy for streaming software. So novel scientific methods are needed in the development and use of software for streaming data.

## 2. Key Efforts

Any single aspect cannot attack the proposed software challenge problems. So a *community ecosystem* with multiple disciplines, multiple areas, and multiple sciences is the first and most important effort to develop and maintain community software packages. We need to bring together a community of interest, from application experts to developers and form a novel scientific software development paradigm. The advantage of this novel development paradigm is that it can directly connect all parts of a software system together and build a complete and integrated design, development, optimization, evaluation and application space. This complete software space will be the foundation of developing and maintaining high performance and high quality software.

The second effort is a novel software architecture to support highly flexible mapping of different subspaces and highly automatic intra-inter subspace optimization. So the software may adapt to and take advantage of very different applications, data, systems and hardware resources well.

The third effort is the domain-specific customized solution to significantly improve the entire performance of given scientific software. A practical streaming software will often focus on some specific scientific problems. These constraints often can provide opportunities to simplify or even remove some operations and procedures in a standard scientific workflow. At the same time, some trade-offs can be implemented to further improve the performance of scientific streaming software.

Among all the three efforts, building a *community ecosystem* is the basis and the key for the success of a community scientific streaming software. Fig. 1 shows the three efforts and their relationship.

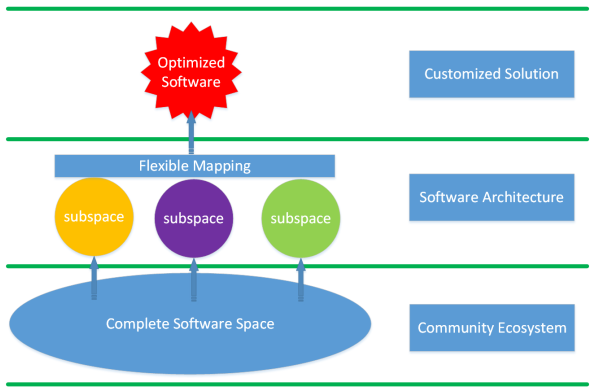

Figure 1: Key efforts for a software ecosystem

The largest non-monetary impediment is accepting such new community software development and use paradigm. At first, the paradigm may make the community software development and use more difficult. However, this paradigm will pave the way to produce highly optimized and much easier to use community software finally. At the same time, such community software will be very flexible to adapt to new requirements. Such additional effort currently has not been well funded by the Federal government. To maximize the impact, community and paradigm based funding are needed to support the novel community software.

## 3. Software Scaling Metric

Big data [6], [7], [8] has reshaped almost every aspect of our society and life. It also connects almost everything. Based on data, all related parties can be integrated together to form the community to achieve scalable software development. Here we define a novel software scaling metric that should meet the following requirement.

$$T(s_1 \times w, s_2 \times r) = \frac{\alpha(s_1)}{\beta(s_2)} \times T(w, r) \tag{1}$$

where $w$,$r$ are the unit of workload and resource. $s_1 \geq 1$, $s_2 \geq 1$ are two positive integers and they mean how many workloads and resources will be used. $\alpha$ and $\beta$ are two functions that meet $0 < \alpha(x) \leq x$ and $\beta(x) \geq x$. $T(w, r)$ is the execution time of a give software with workload $w$ and resource $r$.

When we let $s_1$ be a constant and $\beta(x) = x$, Eq.(1) stands for strong scaling. If we let $s_1/s_2$ be a constant and $\alpha(x) = x, \beta(x) = x$, Eq.(1) stands for weak scaling.

Both strong scaling and weak scaling model the software system as a linear system. However, the novel paradigm will model the community software as a nonlinear system. This means that $\alpha(x) > x$ or $\beta(x) < x$ or even $\alpha(x) > x \wedge \beta(x) < x$. Thus the linear increase in workload will not cause a linear increase in execution time. The linear increase in hardware resources will cause a much faster execution time better than a linear relationship. In this way, we can model the nonlinear scaling of highly optimized and customized community software.

If the resources are heterogeneous, we may extend the model as Eq.(2).

$$\begin{aligned} T((w_1, r_1), (w_2, r_2)) &= max(T(w_1', r_1), T(w_2', r_2)) \\ &\leq max(T(w_1, r_1), T(w_2, r_2)) \end{aligned} \tag{2}$$

where $w_1$ is the workload on resource $r_1$ and $w_2$ is the workload on another kind of resource $r_2$; $w_1' + w_2' = w_1 + w_2$. Eq.(2) means that when we integrate the resources and their workloads, we can achieve better performance than executing workloads on their resources independently.

Both Eq.(1) and Eq.(2) show what the nonlinear scaling feature of optimized community software should achieve. They will be important metrics to measure the scalability of the community software.

The basic methodology is that a community ecosystem will provide novel performance optimization methods in multiple effective optimization spaces. The optimization methods will cross the existing boundaries between applications, algorithms, systems, and hardware to make the software match the given problem and the corresponding solution well. Knowledge from different domains, models, and algorithms with different features and heterogeneous architecture hardware may work together to improve software scalability.

## 4. Correct-by-Construction Software Development

To guarantee the correctness of software, Dijkstra [9] promoted a program derivation method, which can be summarized in the engagement to "develop proof and program hand in hand". Testing is still performed on it, but only to validate the correct-by-construction process rather than to find bugs in the program. Such methods have been employed in areas such as distributed systems [10], fault-tolerant [11] and ultimately makes it increasingly difficult to introduce errors in the software development process. The central tenet of those methods was to capture, formulate and valuate specification constraints expressed in terms of pre- and post-conditions and invariants, to associate them to subprograms, and to prove them locally on them [12].

In fact, the accuracy of community software can also be guaranteed using a similar method.

For machine learning software, there is another method but aims to the same objective. Physics-informed neural networks (PINN) [13], where physical conservation laws and prior physical knowledge are encoded into the neural network, are innovative. These models combine input data with the underlining physics such as described by partial differential equations (PDEs) to constrain the evolution. A PINN is able to unify the formulation and generalize the procedure of solving physical problems governed by PDEs regardless of the structure of the equations. Examples include the Navier–Stokes and the KdV equations [14], stochastic PDEs [15], [16], and fractional PDEs [17].

While PINN models used in the literature add the constraints arising from physics at the output of the neural network, domain knowledge can also be incorporated as inputs. For the task of predicting execution times of programs, existing work [18] showed that the accuracy of neural networks significantly improves when the computational complexity is provided as an additional input. Such additional domain-specific inputs help steer the neural networks in the right direction, especially when the inference is to be performed on data outside the range of training data.

We can improve PINN models by training with a time sequence of input data instead of individual snapshots, to utilize not only the spatial but also the temporal information for a more robust output.

For scientific software, its correctness and accuracy should meet the requirement of scientific conclusions. We can take advantage of the proposed correct-by-construction and PINN methods to achieve the goal. Both correctness and accuracy should be taken into account from the outset in all aspects of software development and use. However, the community ecosystem will be the key to solve these challenges. The role of the community ecosystem will cover the software development and use. The knowledge proposed from the community ecosystem will directly be used in the software execution to provide guarantee in performance, correctness and accuracy.

## 5. Necessity

The major reason for the proposed method is that more and more applications are generating huge amounts of streaming data the corresponding problem solving is highly dependent on the scientific software to handle the data. Considering the complexity of the emerging applications, we need a novel software development paradigm and method to attack the problem. The impact of success is a complete novel community software development and use methodology. The performance and quality of community software will be improved significantly.

## References

[1] T. Hey, S. Tansley, and K. M. Tolle, “Jim Gray on eScience: a transformed scientific method.” 2009.

[2] M. Branchesi, “Multi-messenger astronomy: gravitational waves, neutrinos, photons, and cosmic rays,” in *Journal of Physics: Conference Series*, vol. 718, no. 2. IOP Publishing, 2016, p. 022004.

[3] S. Thurner, P. Klimek, and R. Hanel, “A network-based explanation of why most COVID-19 infection curves are linear,” *Proceedings of the National Academy of Sciences*, vol. 117, no. 37, pp. 22 684–22 689, 2020.

[4] P. F. Dubois, “Maintaining correctness in scientific programs,” *Computing in Science & Engineering*, vol. 7, no. 3, pp. 80–85, 2005.

[5] J. Brown and M. Lipow, “Testing for software reliability,” in *Proceedings of the international conference on Reliable software*, 1975, pp. 518–527.

[6] V. Marx, “The big challenges of big data,” *Nature*, vol. 498, no. 7453, pp. 255–260, 2013.

[7] F. Suchanek and G. Weikum, “Knowledge harvesting in the big-data era,” in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 2013, pp. 933–938.

[8] S. Sagiroglu and D. Sinanc, “Big data: A review,” in *2013 international conference on collaboration technologies and systems (CTS)*. IEEE, 2013, pp. 42–47.

[9] E. Dijkstra, “Concern for correctness as a guiding principle for program composition. ewd288, july. iittp,” 1970.

[10] S. Benyagoub, M. Ouederni, and Y. Aït-Ameur, “Incremental correct-by-construction of distributed systems,” *Journal of Computer Languages*, vol. 57, p. 100942, 2020.

[11] L. Yang, “Correct-by-construction fault-tolerant control,” Ph.D. dissertation, 2020.

[12] L. Baracchi, S. Mazzini, S. Puri, and T. Vardanega, “Lessons learned in a journey toward correct-by-construction model-based development,” in *Ada-Europe International Conference on Reliable Software Technologies*. Springer, 2016, pp. 113–128.

[13] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis, “PPINN: Parareal physics-informed neural network for time-dependent pdes,” *Computer Methods in Applied Mechanics and Engineering*, vol. 370, p. 113250, 2020.

[14] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.

[15] Y. Yang and P. Perdikaris, “Adversarial uncertainty quantification in physics-informed neural networks,” *Journal of Computational Physics*, vol. 394, pp. 136–152, 2019.

[16] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis, “Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems,” *Journal of Computational Physics*, vol. 397, p. 108850, 2019.

[17] G. Pang, L. Lu, and G. E. Karniadakis, “fpinns: Fractional physics-informed neural networks,” *SIAM Journal on Scientific Computing*, vol. 41, no. 4, pp. A2603–A2626, 2019.

[18] A. Srivastava, N. Zhang, R. Kannan, and V. K. Prasanna, “Towards high performance, portability, and productivity: Lightweight augmented neural networks for performance prediction,” *arXiv preprint arXiv:2003.07497*, 2020.