Toward Computational Experiment Management via Multi-language Applications

Justin M. Wozniak,*‡ Timothy G. Armstrong,† Daniel S. Katz,‡ Michael Wilde,*‡ Ian T. Foster*†‡
* Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA
† Dept. of Computer Science, University of Chicago, Chicago, IL, USA
‡ Computation Institute, University of Chicago and Argonne National Laboratory, Chicago, IL, USA

I. Introduction

Within the computer sciences, different subdisciplines have different requirements for software quality and assurance. Additionally, there are different expectations for the ease of development, programming time investment, and time to create and deploy new features. Embedded/real-time computing, web programming, and scientific computing have starkly different software development practices and expectations. There exist applications critical to society with respect to assurance and timeliness of development such as climate modeling, energy production and distribution, and materials design. It is desirable to **move scientific computing towards the high quality and reliability** found in digital control systems while increasing productivity with the high-level tools found in less critical software development areas, such as the web. These methodologies must be pervasive across scaling efforts, debugging/logging tools, and data management cycles.

II. Case study: Crystal structure analysis

DISCUS [1] is a Fortran-based program for computing diffuse scattering of a simulated crystal structure. DISCUS allows a user to run simulated experiments on crystals and produce outputs analogous to those of real experiments, for example the images that would be produced from an X-ray scattering experiment. A recent effort used DISCUS to fit input parameters (crystal configurations) to experimental data. The output of a simulated DISCUS experiment is compared against results of a real experiment, and an evolutionary algorithm is used [2] to iteratively improve the fit, as shown in Figure 1.

Two levels of parallelism have been identified in this compute-intensive process. First, a DISCUS run can be parallelized at the thread level via OpenMP. Second, multiple DISCUS runs can be called concurrently. Initial efforts by the DISCUS team stalled when attempting to fit complex DISCUS parameter data into an ad hoc master-worker parameter passing scheme. This situation motivated us to apply Swift [3], since it includes a load balancer in a scalable master-worker scheme with multiple masters, along with flexible interlanguage data handling. The DISCUS team built Python bindings for the required features. These Python functions are easily callable from Swift, which can be run with an embedded Python interpreter. The evolutionary algorithm was easily implemented as a Swift script, linked into a composite MPI program, and packaged for very large scale machines.

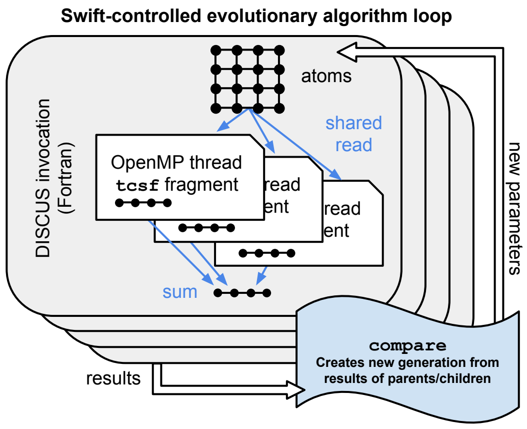
Swift-controlled evolutionary algorithm loop
atoms
shared read
DISCUS invocation (Fortran)
OpenMP thread
tcsf fragment
read
ent
read
ent
sum
new parameters
compare
Creates new generation from results of parents/children
results

Figure 1. Multi-level parallelism in DISCUS for crystal structure modeling.

III. Interlanguage solutions and complexities

Our proposed approach, as motivated by the DISCUS example, involves wrapping existing codes with bindings for higher level languages, then building composite applications that stitch together the functionality into a complete experiment. Concurrency is achieved in multiple ways: lower-level threading and messaging libraries are used for fine-grained parallelism, while high-level ensemble control structures are used for statistical methods such as Monte Carlo, parameter searches, and uncertainty quantification.

This approach is based on previously developed techniques [4], [5], [6], [7] but differs in important ways. First, we **emphasize the construction of concurrent programming models**. Second, we **de-emphasize the use of interlanguage data formats**, but rely on simpler parameters, including scalar numbers, byte arrays, and in some cases, string representations of small but complex data structures- an approach highly compatible with modern tools.

This approach offers multiple benefits, including **rapid development of concurrent functionalry** with good scaling, **ease of integration** of components in different programming languages, and **high testability**. For example, we aim to scale DISCUS to approximately 100,000 cores, which is expected to be achievable without code changes (some software integration is currently being performed). Components including the original Fortran codebase, new Python bindings for it, and the numerical library Numpy have been integrated with relative ease through the use of `f2py`. This integration required no changes to the DISCUS codebase, allowing unchanged software quality as the highly parallel application is developed.

However, multiple challenges must be addressed. First, the use of multiple languages may add to the learning curve for science-based developers; the **use of these technologies must be strictly limited to required features**. Second, some applications may require complex data passing conventions beyond Fortran-style array pointer passing; the **use of an advanced interface description language** (or an existing one) may be appropriate. Third, while we use high-level languages only at the coarsest level and these perform well on extreme-scale systems such as the IBM Blue Gene/Q and Cray Blue Waters, users will need documentation of **performance characteristics and other best practices**.

An increase in the use of high-level tools, however, creates another problem: the **prevention and detection of defects** in the high-level program. Traditional debuggers, designed for operating on highly popular, line-oriented languages (Fortran, C, C++) operate at too low a level to detect defects with the use of the high-level tool. Tracing system calls or MPI calls is also useless to the high-level tool user, because these calls are generated internally by the tool implementation. Thus, the tool builders must integrate **effective, scalable, high-level logging and debugging functionality**.

IV. Interlanguage tools for data management

The overabundance of disk-resident data is well publicized (Big Data). It is properly understood as a software productivity issue, but **existing industry-based solutions are not well-suited to scientific data formats**. Storage formats and filesystem architectures differ greatly between location-oblivious parallel file systems and location-aware analysis clusters [8], as shown in Figure 2. Programming model support must be developed to resolve these differences, including in situ steps.

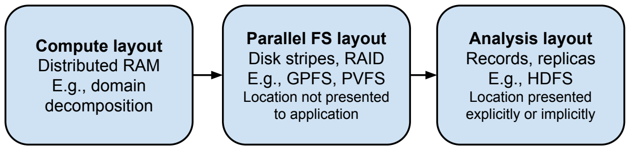

Figure 2. Transition of data locality scheme over workflow stages.

While MapReduce [9] offers many features for record-oriented processing applications, its ease of use for scientific developers is unproven. Consider the complexity of attaching a Fortran routine to the Java-based Hadoop framework. Work must be done to ease the process of **plugging existing scientific software into location-aware analysis frameworks**.

While object storage systems [10] may be able to bridge the gap between parallel file system storage and analysis layouts, performance portability for the analysis codes must be considered. Additionally, as systems present deeper, exposed storage hierarchies [11], these **new storage levels must be brought into the application workflow**.

V. Recent successes

Swift has had multiple recent successes related to this research direction, including: 1) scalable, efficient support for very large applications [3], including applications composed of MPI libraries [12] and high-level tools [13]; 2) prototype support for high-level logging and debugging [14]; and 3) support for data-location-aware task dispatch for data analysis.

VI. Future directions

Future work topics that complement those given in the SWP4XS solicitation are:

1) Improve support for experiment management frameworks in order to accelerate development of the research campaign, not just the individual program invocation.
2) Identify and repurpose applicable programming existing software tools developed in non-science domains.
3) Support rich logging and debugging for high-level tools.
4) Integrate programming languages (not just libraries) that filter and reduce data sets at various levels before data migration to tape or elsewhere off of the supercomputer.
5) Include programming framework support for transition across filesystems and storage formats.

Acknowledgments: This work was supported in part by the U.S. Dept. of Energy, Office of Science, Office of Advanced Scientific Computing Research, under Contract DE-AC02-06CH11357.

References

[1] T. Proffen and R. Neder, “DISCUS: A program for diffuse scattering and defect-structure simulation,” *Journal of Applied Crystallography*, vol. 30, no. 2, pp. 171–175, 1997.
[2] K. Price, R. Storn, and J. Lampinen, *Differential evolution*. Springer, 2005.
[3] J. M. Wozniak, T. G. Armstrong, M. Wilde, D. S. Katz, E. Lusk, and I. T. Foster, “Swift/T: Scalable data flow programming for many-task applications,” in *Proc. CCGrid*, 2013.
[4] D. Beazley, “Automated scientific software scripting with SWIG,” *Future Generation Computer Systems*, vol. 19, no. 5, pp. 599–609, 2003.
[5] J. K. Ousterhout, “Scripting: higher level programming for the 21st century,” *Computer*, vol. 31, no. 3, pp. 23 –30, Mar. 1998.
[6] V. Sarkar and J. Hennessy, “Partitioning parallel programs for macro-dataflow,” in *Proc. LFP*, 1986.
[7] “BABEL: High-performance language interoperability,” http://computation.llnl.gov/casc/components.
[8] W. Tantisiriroj, S. Patil, G. Gibson, S. W. Son, S. Lang, and R. Ross, “On the duality of data-intensive file system design: Reconciling HDFS and PVFS,” in *Proc. SC*, 2011.
[9] J. Dean and S. Ghemawat, “MapReduce: Simplified data processing on large clusters,” in *Proc. OSDI*, 2004.
[10] C. Karakoyunlu, D. Kimpe, P. Carns, K. Harms, R. Ross, and L. Ward, “Towards a unified object storage foundation for scalable storage systems,” in *Proc. IASDS at Cluster*, 2013.
[11] H. M. Monti, A. R. Butt, and S. S. Vazhkudai, “/Scratch as a cache: Rethinking HPC center scratch storage,” in *Proc. ICS*, 2009.
[12] J. M. Wozniak, T. Peterka, T. G. Armstrong, J. Dinan, E. Lusk, M. Wilde, and I. Foster, “Dataflow coordination of data-parallel tasks via MPI 3.0,” in *Proc. EuroMPI*, 2013.
[13] J. M. Wozniak, T. G. Armstrong, S. J. Krieder, K. Maheshwari, M. Wilde, and I. T. Foster, “Mega Python: Scalable interlanguage scripting for scientific computing,” 2013, under review for PyHPC, preprint ANL/MCS-P5020-0913.
[14] J. M. Wozniak, A. Chan, T. G. Armstrong, M. Wilde, E. Lusk, and I. T. Foster, “A model for tracing and debugging large-scale task-parallel programs with MPE,” in *Proc. LASH-C at PPoPP*, 2013.