A Task-Based Abstraction Layer for User Productivity and Performance Portability in Post-Moore's Era Supercomputing

Steve Petruzza, Attila Gyulassy, Valerio Pascucci, and Peer-Timo Bremer

Abstract—The proliferation of heterogeneous computing architectures in current and future supercomputing systems dramatically increases the complexity of software development and exacerbates the divergence of software stacks. Currently, task-based runtimes attempt to alleviate these impediments, however their effective use requires expertise and deep integration that does not facilitate reuse and portability. We propose to introduce a task-based abstraction layer that separates the definition of the algorithm from the runtime-specific implementation, while maintaining performance portability.

1 INTRODUCTION

Recent and future supercomputing systems are shifting towards more heterogeneous computing architectures to address post-Moore's era supercomputing (PMES) and to increase their power efficiency. At the same time, software infrastructures are becoming more heterogeneous, and a number of libraries and frameworks have been proposed to simplify accessing, aggregating and staging simulation data as well as facilitating the implementation of algorithms and integrate them into simulations [4], [6], [9], [13]. Despite these efforts, implementing scalable simulations and analysis algorithms remains challenging, requiring both domain-specific expertise, and in-depth knowledge of the chosen libraries or runtimes. While many runtimes promise genericity, instead, achieving performance comes from tightly coupling application to the underlying software stack, frequently even optimizing to specific machines. Certain simulations can currently afford this tight coupling as they are designed with performance foremost in mind with specific machines and software stack targets (e.g. NAMD [12], ChaNGa [3], Legion S3D [2]), however utility and analytics codes that are designed to be run across simulations cannot be re-implemented for every use case. As data movement becomes the driving bottleneck for exascale computation, these diverse algorithms must be deployed in-situ and their performance portability will determine the success or failure of scientific workflows.

To leverage the increasing complexity of modern machines, task-based runtime systems are being adopted to manage resources, scheduling and execution of tasks on heterogenous architectures. While simulation codes choose to integrate with these software stacks in order to enable high performance and scalability, in-situ analysis algorithms are restricted to the host application software stack making the portability challenging and expensive. Therefore, developers of analysis packages are faced with the choice of restricting their efforts to a particular runtime or even individual applications or maintaining (and optimizing) an ever growing variation of implementations. The existing task based runtimes provide the ability to decompose every application in computational components of arbitrary granularity, enabling the definition and implementation of tasks for different hardware architectures. However, comparing the performance of the same algorithm on different runtimes becomes a very expensive task and introduce the big challenge of understanding how the same algorithm should be designed, implemented and executed by different runtimes taking advantage of their distinct data and execution models.

- *S. Petruzza, A. Gyulassy and V. Pascucci are with the SCI Institute - University of Utah.*
  E-mail: spetruzza@sci.utah.edu
- *Peer-Timo Bremer is with Lawrence Livermore National Laboraty*

PMES Workshop, Dallas, 11 Nov 2018. `http://j.mp/pmes18`

2 TASK ABSTRACTION LAYER

We propose a new programming paradigm for PMES systems using a task graph to encode the data dependencies between data processing stages in algorithms. In our model, a task simply represents the data transformation between inputs and outputs, and is the natural level at which programmers express data relationships when designing algorithms. A task graph expresses not only dependencies between internal stages of an algorithm, but also dependencies introduced when data is decomposed for distributed computation. For instance, in figure 1, an in-situ distributed volume rendering algorithm is represented in terms of tasks for local rendering and compositing, with the directed edges of the graph encoding data dependencies. A task-based model provides an inherent separation of concerns in which the algorithm developer is not exposed to any communication, synchronization or other runtime-related concepts, and is also insulated from the architectural properties of a target machine. Additionally, the design naturally allows over-decomposition, which is not only useful for runtimes that provide load balancing but also simplifies debugging at scale.

The task-based abstraction implemented as an Embedded Domain-Specific Language (EDSL) together with a thin software layer is sufficient to automatically adapt a task-based model to the specific runtime systems that are used in a scientific workflow. In contrast to existing approaches, where significant investment is required to deploy an algorithm on a specific runtime system, a task-based model and EDSL allows developers to maintain a single implementation of an algorithm that nevertheless provides a native interface and efficient implementation for a number of different software stacks.

The EDSL and software layer needs to have a frontend with a stable and easy to program interface to express various

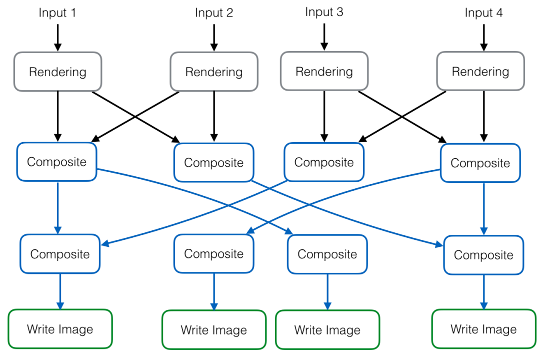

Fig. 1: The task graph for binary-swap distributed rendering has tasks for local rendering, compositing, and writing to storage. The arrows indicate data dependence between stages of the algorithm.

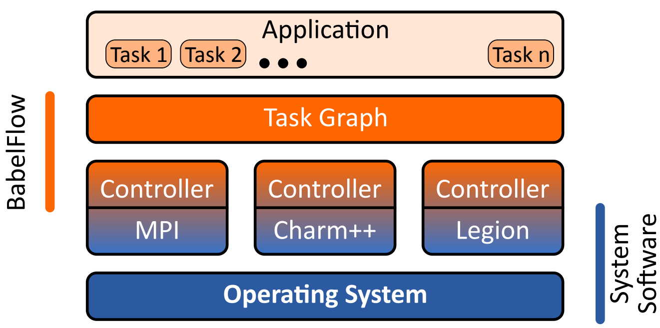

Fig. 2: BabelFlow architecture: The application implements a set of tasks and expresses its dataflow using an EDSL in form of a task graph. A BabelFlow controller, implemented natively in one of several runtimes, then executes the task graph.

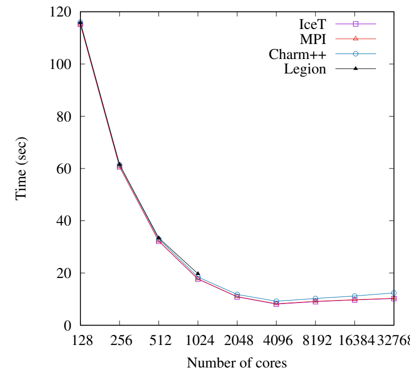

Fig. 3: As reported in [10], we demonstrated strong scaling using Babelflow of VTK's distributed volume rendering implemented with a binary-swap task graph, using several runtimes as backends. The overall performance was equivalent to a hand-tuned MPI implementation (i.e. using IceT [8]).

algorithms, while the backend needs to efficiently map the implementation to the various software stacks. Simultaneously, this layer should be easy to integrate into other (likely large and complex) software systems both in terms of development and build complexity as well as with respect to exchanging data and execution control.

3 Preliminary Results: Babelflow

Much of the complexity of todays runtimes, i.e. MPI, Charm++, Legion, etc. is that they allow user to define an almost arbitrarily complex algorithm or dataflow at runtime. However, in many use cases, like visualization, analysis, or file I/O the necessary steps are known a priory and are static. Relying on this restriction can allow much simpler descriptions of such algorithms. Providing backends that translate a high level description, for example, in form of a Domain Specific Language (DSL), to an arbitrary software stack will provide a highly productive environment for the cross-cutting development while delivering a native interface to any consumers of the software in their chosen system environment.

An example of effort toward this direction is the BabelFlow Embedded DSL [10] [11], which defines a task based abstraction layer to design the algorithm and implements backends for different runtimes (i.e., currently MPI, Legion [1] and Charm++ [5]). The EDSL provides simple syntax for specifying tasks and data dependencies, independent of the underlying runtime. We demonstrated (figure 3) that the Babelflow controller was able to translate the task representation to MPI, Charm++ and Legion runtimes, and execute a volume rendering and image compositing workflow with the same scaling behavior as natively implemented code (i.e., using the IceT library [8]). Additional experiments demonstrated performance portability for distributed merge tree computation and 3D image registration.

Although currently limited to static task graphs, the Babelflow controller could be extended to dynamic graphs as well as data-dependent control flow dynamically changes task dependencies. In the future this framework could also target higher level frameworks such as ADIOS or Glean. Similarly, the system can exploit new datamodels such as Conduit [7] to transparently access simulation data and further uncouple the implementation of an algorithm from the specific application that uses it.

4 Conclusion

The unstoppable divergence of simulation and software stacks will continue in the Post-Moore's era of supercomputing. More than ever, we now need to focus on how to quickly and transparently port algorithms to different software stacks in order to easily integrate with different applications and maximize the utilization of the heterogeneous resources available on the machines. Separating the definition of the algorithm from the specific runtime implementation allows the user to focus more on the algorithm specification and less on acquiring new skills to implement their application on yet another runtime. We implemented and EDSL that translates a task based algorithm definition to different task-based runtimes implementation and demonstrated that software portability can be easily achieved as well as overall performance portability. We believe that this direction can increase user productivity and also allow small contributions, in terms of analysis and visualization algorithms, to find widespread adoption over multiple communities using different software stacks.

Acknowledgments

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. This work is also supported in part by NSF: CGV: Award:1314896, NSF:IIP Award: 1602127 NSF:ACI:award 1649923, DOE/SciDAC DESC0007446, CCMSC DE-NA0002375, and PIPER: ER26142

DE-SC0010498 and by the Department of Energy under the guidance of Dr. Lucy Nowell and Richard Carson. This research used the resources of the Supercomputing Laboratory at KAUST, Saudi Arabia.

REFERENCES

[1] M. Bauer, S. Treichler *et al.*, "Legion: Expressing locality and independence with logical regions," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '12. Los Alamitos, CA, USA: IEEE Computer Society Press, 2012, pp. 66:1–66:11. [Online]. Available: http://dl.acm.org/citation.cfm?id=2388996.2389086

[2] M. E. Bauer, "Legion: Programming distributed heterogeneous architectures with logical regions," Ph.D. dissertation, Stanford University, 2014.

[3] P. Jetley, F. Gioachin *et al.*, "Massively parallel cosmological simulations with changa," in *Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on*, April 2008, pp. 1–12.

[4] T. Jin, F. Zhang *et al.*, "Using cross-layer adaptations for dynamic data management in large scale coupled scientific workflows," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, ser. SC '13. New York, NY, USA: ACM, 2013, pp. 74:1–74:12. [Online]. Available: http://doi.acm.org/10.1145/2503210.2503301

[5] L. V. Kale and S. Krishnan, "Charm++: A portable concurrent object oriented system based on c++," *SIGPLAN Not.*, vol. 28, no. 10, pp. 91–108, Oct. 1993. [Online]. Available: http://doi.acm.org/10.1145/167962.165874

[6] Q. Liu, J. Logan *et al.*, "Hello adios: the challenges and lessons of developing leadership class i/o frameworks," *Concurrency and Computation: Practice and Experience*, vol. 26, no. 7, pp. 1453–1473, 2014. [Online]. Available: http://dx.doi.org/10.1002/cpe.3125

[7] LLNL. (2014) Conduit. [Online]. Available: https://software.llnl.gov/conduit/

[8] K. Moreland, "Icet users' guide and reference," *Sandia National Lab, Tech. Rep*, 2011.

[9] T. Peterka, R. Ross *et al.*, "Scalable parallel building blocks for custom data analysis," in *Proceedings of Large Data Analysis and Visualization Symposium LDAV'11*, Providence, RI, 2011.

[10] S. Petruzza, S. Treichler *et al.*, "Babelflow: An embedded domain specific language for parallel analysis and visualization," in *2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2018, pp. 463–473.

[11] S. Petruzza, A. Venkat *et al.*, "Isavs: Interactive scalable analysis and visualization system," in *Proceedings of SIGGRAPH Asia 2017 Symposium on Visualization*, ser. SA '17. New York, NY, USA: ACM, 2017, pp. 18:1–18:8. [Online]. Available: http://doi.acm.org/10.1145/3139295.3139299

[12] J. C. Phillips, R. Braun *et al.*, "Scalable molecular dynamics with namd," *Journal of Computational Chemistry*, vol. 26, no. 16, pp. 1781–1802, 2005. [Online]. Available: http://dx.doi.org/10.1002/jcc.20289

[13] V. Vishwanath, M. Hereld *et al.*, "Topology-aware data movement and staging for i/o acceleration on blue gene/p supercomputing systems," in *Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis*, ser. SC '11. New York, NY, USA: ACM, 2011, pp. 19:1–19:11. [Online]. Available: http://doi.acm.org/10.1145/2063384.2063409