

High Performance Computing/Domain Specific Language Symbiosis for Exascale Computing

Erik Saule^{1†}, Kamer Kaya[†] and Ümit V. Çatalyürek^{†‡}

[†] Dept. Biomedical Informatics, The Ohio State University

[‡] Dept. Electrical and Computer Engineering, The Ohio State University

1. INTRODUCTION

The increasing need in computation resources pushes hardware manufacturers to build ever larger machines. The maximum aggregated performance we reached was Teraflops in 1995, Petaflops in 2005, and the Exaflops is expected to be reached in the next couple of years. With this great performance comes great complexity. Single or multi-processor architectures transformed into a myriad of specialized processors making the interconnect between them an important concern. Exploiting such machines efficiently will be challenging because they will be heterogeneous, massively parallel, hierarchical, and distributed memory.

At first, exascale machines will certainly be used only by application scientists. The technical knowledge of these users is sufficient to use large computing platforms, but it is nowhere near the expertise required to exploit Exascale platforms efficiently. Large scientific institutions typically hire High Performance Computing (HPC) experts to get help on the development and optimization of scientific applications. However, making HPC platforms and applications ever more complex induces ever longer development cycles.

In a typical scientific application, implementing the application logic (equation solving, simulations) according to the scientific model is fused with its execution on the high performance computing platform. In this positional paper, we propose to decouple the two components explicitly. By exposing a Domain Specific Language (DSL) to application developers and leaving its implementation to HPC experts, we claim that we can easily leverage Exascale Computing for a large range of applications. We exemplify our point on sparse linear algebra computations but it definitely has a broader impact.

2. DOMAIN SPECIFIC LANGUAGES ON TOP OF EFFICIENT RUNTIME SYSTEM

Scientific applications should be implemented in a high-level programming language such as a Domain Specific Language. The DSL lets **developers focus only on the semantics of the applications** rather than on their implementation and execution: it should not matter to the developer how operations are implemented and how and where the data is stored. For example in massive sparse linear algebra computations, it should be inconsequential to the developer whether the matrices are stored in Compressed Row Storage format or in Coordinate Based format, or using hybrid storing techniques. The idea of using DSL languages to conceal implementation details while achieving high performance has been applied in databases through SQL. In linear algebra, multiple high-level languages exist under the form of the mathematical software interpreters.

Since the developer sees only the application through the lens of high-level primitives, a front-end that binds the DSL to the middleware is proposed as an important opportunity to setup an **execution plan** that will optimize the performance more. For instance, as long as the data does not need to be displayed to the user or used in an explicit test, synchronizations can be minimized by pipelining task executions. Also by querying the middleware for **execution statistics**, the front-end can apply **Just-in-Time optimization**: it can change the partitioning of the data at will or choose the algorithm and storage format to maximize the application performance. For instance, by noticing the amount of cache misses that occur during a sparse-matrix vector multiplication and knowing that the same operation is likely to be performed multiple times, the front-end can decide to repartition and reorder some parts of the matrix to increase memory locality. In linear algebra, many different optimization techniques (data representation, ordering, partitioning) and performance predictions (auto tuning, communication modeling) are known [2, 3, 4, 8, 9].

To support the efficient execution of the application expressed in the DSL on Exascale platforms, we propose to use a graph-based middleware where each vertex in the graph represents a unique computation which is tagged with its data dependencies. The edges of the graph represent dependencies between the computations. Such a representation of an application has many useful properties. Since the computation is abstracted based on the input and output of the task, it is possible to change its underlying implementation. This allows us to **easily tune the implementations of the tasks** with different algorithms and **support**

¹Contact Author: esaule@bmi.osu.edu

heterogeneous processors, such as general purpose CPUs, or specialized accelerators such as GPUs, FPGAs, and the upcoming Intel MIC [6]. Also, if the data can be transported over the network, one can **dynamically re-balance the load** of the application. Large tasks can potentially be decomposed into multiple tasks so as to **increase the parallelism** [1]. Depending on the shape of the application graph, the middleware can switch between multiple scheduling techniques: if there are many unstructured dependencies, then classical DAG scheduling should be applied. If there are almost no dependencies, the bag-of-task model will sustain almost the peak performance. If the amount of data is small then work-stealing techniques will achieve asymptotically optimal performance. If the computations are strongly structured and iterative then pipelined scheduling techniques will maximize the system throughput. Moreover, having tasks modeled as a graph allows us to add **fault tolerance** properties to the system which are important while running on Exascale machines. The model natively supports passive and active replication of task and data [5]. And, the application state can be checkpointed by saving the input data of a vertex cut of the graph [7].

Finally, the tasks and their communication should be implemented under the dataflow programming abstraction which natively allows us to trigger computations when all the data is available. It minimizes idle times by making the system asynchronous and **leverages communication and computation overlap** [6].

The DAGuE/DPLASMA [1] and DOoC+LAF [10] are distributed memory middleware which can be considered as prototypes. The former focuses on dense linear algebra, pins tasks to computing nodes and executes tasks on a node using a workstealing algorithm. The latter is originally designed for sparse linear algebra and supports arbitrary data representations. Its scheduling is based on data partitioning and it supports out-of-core functionalities which can be used for checkpointing. Both support accelerators and provide communication and computation overlapping. But their front-end is very tied to their execution engine, do not perform just-in-time optimizations, and have limited node-level scheduling.

3. CONCLUSION

In summary, we postulate that a combination of techniques can be deployed to achieve high performance on a significant class of problem for Exascale Computing. Scientists should write their applications using a Domain Specific Language. The DSL front-end gathers statistics and chooses the appropriate data representation and algorithms. The execution of the algorithm is represented as a DAG of tasks which allows us to balance the load across the machine and to specialize kernels for a target architecture or the specificity of the data at hand. The tasks are executed by a dataflow middleware which provides efficient network resource utilization.

Such an approach will improve the programmability of an Exascale machine while using the system efficiently. All the bricks already exist and prototypes have already been designed. It also should be extendable so as to span across multiple application domains. Will this approach be the be-all end-all of Exascale computing? Certainly not. But such a silver bullet has not been discovered despite a large community of researchers tried to build it. Moreover, our approach will definitely work for many classes of systems and applications. It will be useful and can be build in a reasonable time. Let's do it!

DIMENSIONAL ASSESSMENT

Challenges addressed: Provide an easy-to-program system for extreme scale, failure-prone, heterogeneous, hierarchical memory Exascale machines for a class of large important scientific applications.

Maturity: Several techniques have been proposed to solve each problem mentioned above independently. Nothing seems to indicate that these existing techniques are incompatible.

Uniqueness: The proper optimizations for extreme-scale systems are significantly different than optimizations for classical large-scale systems. While components might be individually developed under other research programs, the modelization of the platform and underlying optimization can only be performed on real-size extreme-scale machines.

Novelty: The integrated aspect of the proposed solution is what differentiate it from all other approaches. Existing approaches seem to be either efficient or easy to use, not both simultaneously.

Applicability: The back-end can certainly be reused in other areas. A novel and dedicated DSL front-end would need to be rewritten.

Effort: A partial prototype has already been developed. An expert compiler team can develop a front-end in practical time. Then various optimizations and hardware-specific modules should be implemented.

REFERENCES

- [1] George Bosilca, Aurelien Bouteiller, Anthony Danalis, Thomas Hérault, Pierre Lemarinier, and Jack Dongarra. DAGuE: A generic distributed DAG engine for high performance computing. *Parallel Computing*, 38(1-2):37–51, 2012.
- [2] Umit V. Catalyurek, Cevdet Aykanat, and Bora Ucar. On two-dimensional sparse matrix partitioning: Models, methods, and a recipe. *SIAM Journal on Scientific Computing*, 32(2):656–683, 2010.
- [3] Pietro Cicotti, Xiaoye S. Li, and Scott B. Baden. Performance modeling tools for parallel sparse linear algebra computations. *Parallel Computing*, 2009.
- [4] Jack Dongarra and Victor Eijkhout. Self-adapting numerical software for next generation applications. *International Journal of High Performance Computing and Applications*, 17(2):125–131, 2003.
- [5] Alain Girault, Erik Saule, and Denis Trystram. Reliability versus performance for critical applications. *Journal of Parallel and Distributed Computing*, 69(3):326–336, March 2009.
- [6] Timothy D. R. Hartley, Erik Saule, and Umit V. Catalyurek. Improving performance of adaptive component-based dataflow middleware. *Parallel Computing*, 38(6-7):289–309, 2012.
- [7] Samir Jafar, Axel W. Krings, and Thierry Gautier. Flexible rollback recovery in dynamic heterogeneous grid computing. *IEEE Transactions on Dependable and Secure Computing*, 6(1), 2009.
- [8] Rich Vuduc, James Demmel, Katherine A. Yelick, Shoaib Kamil, Rajesh Nishtala, and Benjamin C. Lee. Performance optimizations and bounds for sparse matrix-vector multiply. *Super Computing*, page 35, 2002.
- [9] Samuel Williams, Leonid Oliker, Richard W. Vuduc, John Shalf, and Katherine A. Yelick and James Demmel. Optimization of sparse matrix-vector multiplication on emerging multicore platforms. *Super Computing*, page 38, 2007.
- [10] Zheng Zhou, Erik Saule, Hasan Metin Aktulga, Chao Yang, Esmond G. Ng, Pieter Maris, James P. Vary, and Umit V. Catalyurek. An out-of-core eigensolver on ssd-equipped clusters. In *Proc. of IEEE Cluster*, Sep 2012. (to appear).