

COMBINATORIAL SCIENTIFIC COMPUTING FOR EXASCALE SYSTEMS AND APPLICATIONS

KAREN D. DEVINE, SIVASANKARAN RAJAMANICKAM AND ERIK G. BOMAN
SANDIA NATIONAL LABORATORIES, ALBUQUERQUE, NM

INTRODUCTION: EVOLUTION IN ARCHITECTURES AND ALGORITHMS

Combinatorial scientific computing has played a pivotal role in helping applications achieve high performance in the distributed memory systems that have dominated parallel computing in the past few decades. This success has been accomplished both by expanding the frontiers of combinatorial scientific computing (through various new partitioning, ordering, coloring, and matching methods) and by representing various problems of interest (e.g., reduced communication volume, social network analysis, community detection) in combinatorial terms in order to solve them efficiently. While many challenges were faced in this work, distributed memory systems were somewhat simple compared to the systems that will be needed to achieve exascale: their processors were largely homogeneous; their communication model was straightforward; and their programming paradigm (message-passing) was well-defined and standardized (via MPI).

Exascale systems, by necessity, will be more complicated, but combinatorial scientific computing has many new roles to play. Combinations of multicore CPUs and accelerators add new complexity to traditional partitioning strategies, which now must be more aware of the underlying architecture than ever before. Memory hierarchies complicate communication models such that any data movement must be considered a form of communication. Data ordering plays a greater role in maintaining data locality in multithreaded applications. Matching and coloring can be used to more optimally map tasks to processors and exploit underlying parallelism. Indeed, combinatorial scientific computing will extend beyond the application level, incorporating architectural information into models and application information into run-time systems. A few examples of this evolution of algorithms, ranging from redefinition of traditional partitioning problems to closer integration of algorithms with run-time and operating systems, are described below.

FLEXIBLE PARTITIONING FOR RUN-TIME PERFORMANCE OPTIMIZATION

With the massive push towards on-node parallelism and accelerators (GPUs, MICs), there are many new challenges for the next decade. Current partitioning methods, for example, are designed for the SIMD programming model, where they partition the problem and assign entities to an owner. While this owner-computes model has worked very well in the past, modern architectures, programming models (multithreading), and run-time systems require a redefinition of traditional partitioning. In multithreaded environments, for example, run-time systems have options to use various scheduling strategies for better performance. The run-time systems, however, are disconnected from the load balancing and the “communication cost” information that can be provided by applications. The run-time scheduling could be improved by better guidance from partitioners, as partitioners have both the load balance and communication cost information.

However, in multithreaded environments, prescribing one owner to an entity is too limiting. A better approach is to assign some entities to a single owner and some entities to a range of owners, any of whom could perform computations on the entities; the range of owners can be thought of as a “thread team.” This approach may have advantages over simple over-partitioning of a problem into a large number of tasks with arbitrary ownership (e.g., [6]), where data locality between tasks is not

considered. With thread teams, the run-time system can schedule entities to cores in ways that balance the load while still maintaining data locality.

The integration of this guidance at the level of run-time systems requires new research into partitioning algorithms and extension of run-time systems. While flexible assignment of tasks has been studied before [7], that work was in the context of assigning unique owners, not assigning a group of owners. The run-time systems have to improve as well in order to take better guidance from applications than a simple work-chunk size.

ARCHITECTURE-AWARE COMBINATORIAL ALGORITHMS

To achieve high performance in exascale systems, algorithms must be more aware of the computers' processors, accelerators, memory hierarchies, and network topology, and must integrate that information with application work loads and communication patterns. Incorporation of architecture information has begun — for example, hierarchical partitioning strategies have become more common for multicore systems [8, 9] — but much work remains. Partitioning for combinations of multicore processors and accelerators, controlling costs of data transfer from CPU to accelerators, scheduling computations to maximize data reuse, and partitioning of data into fixed-size parts to fit into accelerator memory are all areas requiring further research and development.

Moreover, many aspects of parallel computing currently ignore information that could be exploited for better parallel performance. For example, scheduling systems (e.g., SLURM [5]) allocate and assign resources, but do not account for dependencies between application tasks in assigning processes to cores. Intelligent mapping (matching) strategies that account for both the application communication patterns and architecture topology are needed to more effectively assign tasks to node allocations and the specific node-architectures within them.

New algorithms that exploit real-time application and system performance information also are needed to effectively utilize resources in exascale systems. Such algorithms could, for example, dynamically allow cores to be undersubscribed if using more cores would cause memory contention and hurt overall execution time. Partitioning algorithms could adapt to the greater variability in exascale systems; for example, they could reassign work to balance loads when cores are under-clocked for energy savings or heat reduction. In such cases, lightweight repartitioning within subsets of cores may be more appropriate than current global repartitioning strategies.

Many pieces of the exascale solution exist: for example, `hwloc` [3] provides static node-level architecture information; `OVIS/LDMS` [2] aggregates real-time system-level metrics; `LibTopoMap` [4] maps tasks to network topologies; `Zoltan` [1] provides a toolkit of combinatorial algorithms. But their effective integration is an important area of research. When new combinatorial algorithms are developed that blend these technologies to incorporate real-time, architecture-specific data with application data, the greatest benefit to exascale computing will be realized.

ACKNOWLEDGEMENT

Sandia is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

REFERENCES

- [1] E. BOMAN, Ü. ÇATALYÜREK, C. CHEVALIER, AND K. DEVINE, *The Zoltan and Isorropia parallel toolkits for combinatorial scientific computing: Partitioning, ordering and coloring*, Scientific Programming, 20 (2012), pp. 129–150.
- [2] J. M. BRANDT, F. CHEN, A. C. GENTILE, C. LEANGSUKSUN, J. R. MAYO, P. P. PÉBAY, D. ROE, N. TAERAT, D. C. THOMPSON, AND M. H. WONG, *Framework for enabling system understanding*, in Euro-Par Workshops (2), 2011, pp. 231–240.
- [3] F. BROQUEDIS, J. CLET ORTEGA, S. MOREAUD, N. FURMENTO, B. GOGLIN, G. MERCIER, S. THIBAUT, AND R. NAMYST, *hwloc: a Generic Framework for Managing Hardware Affinities in HPC Applications*, in PDP 2010 -

The 18th Euromicro International Conference on Parallel, Distributed and Network-Based Computing, IEEE, ed., Pisa Italie, 02 2010.

- [4] T. HOEFLER AND M. SNIR, *Generic Topology Mapping Strategies for Large-scale Parallel Architectures*, in Proceedings of the 2011 ACM International Conference on Supercomputing (ICS'11), ACM, Jun. 2011, pp. 75–85.
- [5] M. A. JETTE, A. B. YOO, AND M. GRONDONA, *SLURM: Simple Linux Utility for Resource Management*, in In Lecture Notes in Computer Science: Proceedings of Job Scheduling Strategies for Parallel Processing (JSSPP) 2003, Springer-Verlag, 2002, pp. 44–60.
- [6] L. KALE, A. ARYA, A. BHATELE, A. GUPTA, N. JAIN, P. JETLEY, J. LIFFLANDER, P. MILLER, Y. SUN, R. VENKATARAMAN, L. WESOLOWSKI, AND G. ZHENG, *Charm++ for productivity and performance: A submission to the 2011 HPC class II challenge*, Tech. Rep. 11-49, Parallel Programming Laboratory, November 2011.
- [7] A. PINAR AND B. HENDRICKSON, *Improving load balance with flexibly assignable tasks*, IEEE Transactions on Parallel and Distributed Systems, 16 (2005), pp. 956–965.
- [8] L. A. RIESEN, E. G. BOMAN, K. D. DEVINE, AND S. RAJAMANICKAM, *Tuning applications for multicore architectures with hierarchical partitioning and local ordering*. Presentation at SIAM Parallel Processing, 2012.
- [9] M. ZHOU, O. SAHNI, K. D. DEVINE, M. S. SHEPHARD, AND K. E. JANSEN, *Controlling unstructured mesh partitions for massively parallel simulations*, SIAM J. Sci. Comput., 32 (2010), pp. 3201–3227.