Load Balancing Scientific Applications

Olga Pearce

Department of Computer Science and Engineering, Texas A&M University
Lawrence Livermore National Laboratory
olga@cse.tamu.edu, olga@llnl.gov

1. Introduction

Optimizing high-performance physical simulations to run on ever-growing supercomputing systems is challenging. One of the challenges is that the dynamic behavior of large modern parallel simulation codes can lead to imbalances in computational load among processors. Load imbalance is particularly expensive at scale, because hundreds of thousands of idle processors may wait on a single overloaded processor. Future machines will support even more parallelism and efficiently redistributing and balancing load will be critical for good performance.

In this thesis, I address how to evaluate load imbalance and make its correction affordable. I developed a model for comparison of load balance algorithms in the context of a specific application imbalance scenario, enabling the selection of a balancing algorithm that will minimize overall runtime (Section 2). I provide an accurate and fast method to balance the load in simulations with highly non-uniform density, using adaptive sampling to construct an easy to partition hypergraph of a manageable size (Section 3). I devised a framework for decoupling the load balancer from the application, enabling asynchronous balancing and use of balancing algorithms otherwise not scalable to the full scale of the application, i.e., graph partitioners (Section 4).

2. Quantifying the Effectiveness of Load Balance Algorithms

Current load balance mechanisms are often application-specific and make implicit assumptions about the load [3, 10]. Some strategies place the burden of providing accurate load information, including the decision on when to balance, on the application [4, 9, 12]. Existing application-independent mechanisms simply measure the application load without any knowledge of the migratable tasks in the application [5, 11], which limits them to identifying the imbalance without correcting it.

Based on the attribution of load to the migratable tasks in the application, we developed a methodology to evaluate the accuracy of load information, and a cost model to decide when to balance the application and which load balance algorithm to use. While *global* load balance algorithms may correct the imbalance in a single step at a cost of high overhead, *diffusive* algorithms correct the imbalance gradually but pay a penalty for taking many steps to converge. Diffusive algorithms may be sufficient for localized imbalances, but drastic and expensive imbalances across a large system may require a drastic correction. Our cost model enables runtime evaluation of the imbalance in the application in terms of the effectiveness of the available load balancing algorithms. Our model correctly selects the algorithm that achieves the lowest runtime in up to 96% of the cases, and can achieve a 19% gain over selecting a single balancing algorithm for all cases.

Our work on quantifying the effectiveness of load balance algorithms was published at the *International Conference on Supercomputing* (ICS) 2012 [8].

3. Efficient Load Balance Algorithm for N-Body Simulations with Highly Non-Uniform Density

N-body methods simulate the evolution of systems of particles (or bodies). They are critical for scientific research in fields as diverse as molecular dynamics, astrophysics, and material science [6, 10, 13]. Most load balancing techniques for N-body methods use particle count to approximate computational work [3, 10]. The assumption that particles are work is inaccurate, especially for systems with high density variation, because work in an N-body simulation is proportional to the particle *density*, not the particle count.

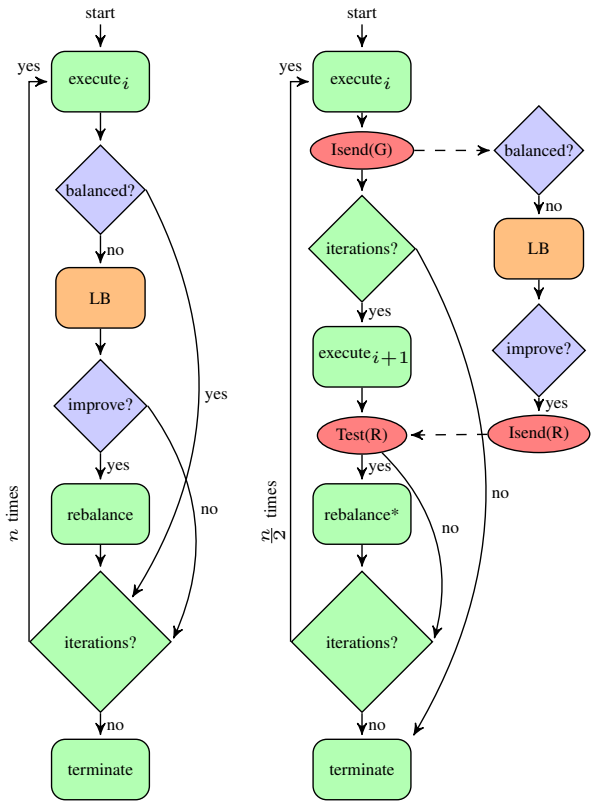
We have demonstrated that existing techniques do not perform well at scale when particle density is highly non-uniform, and developed a load balance technique that efficiently migrates interactions instead of particles. We use adaptive sampling to create migratable tasks (interaction subsets) that are uniform in size and therefore easy to partition. The migratable tasks are used as a basis for the hypergraph that is partitioned; our aggressive sampling makes the partitioning affordable by reducing the size of the graph by several orders of magnitude. We implement and evaluate our approach on a Barnes-Hut algorithm [1] and a large-scale dislocation dynamics application, ParaDiS [3]. Our method achieves up to 26% improvement in overall performance of Barnes-Hut and 18% in ParaDiS.

Our methodology for explicitly load balancing N-body simulations with highly non-uniform density was published at the *International Conference on Supercomputing* (ICS) 2014 [7].

4. Lazy Load Balancing: Offloading Load Balance Computation to Non-Application Resources

Dynamic load balancing of parallel applications becomes more critical at scale, while also being expensive. We propose a lazy approach to load balancing, delaying the load reassignment until the load balance directions are computed on resources available in the system. During the delay, the simulation has advanced and its state has changed. We demonstrate how delayed directions can be applied to the current state of the simulation, based on the key observation that the simulation changes slowly and has significant temporal locality. We describe how the delay impacts the correctness and performance of the application. We implemented a framework that allows us to decouple the resources used by the load balancing algorithm from the resources used by the application, and use it to offload the load balancer to dedicated resources, overlapping the load balance computation with the application execution.

This work was partially performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344 (LLNL-MI-657788).



(a) Balancing Inline (b) Balancing Asynchronously

Figure 1. Inline vs. Asynchronous Load Balancing. Application components are shown in green, load balancing decisions are shown in blue, external libraries are shown in orange, asynchronous communication between application and load balancing tool is shown in red.

The following are the main components of balancing an application, as shown in Figure 1(a):

1. *when*: evaluate the imbalance, decide whether to correct load imbalance at this point in execution;
2. *how*: use a load balancing method to compute a balanced task assignment;
3. *rebalance* the application if needed.

As demonstrated in Figure 1(a), all these steps are typically performed inline with the application. The application execution is paused while the load balancing decisions are made. Previously, it was common to run such auxiliary tasks as load balancing sequentially. While the load balance algorithm can be parallelized, it may scale differently than the application. At scale, this results in a long wait for thousands of application processes. We have developed a framework that enables us to dedicate resources to the load balancing decisions. The application can continue running while the load balancing algorithm evaluates the imbalance and decides how it should be corrected, as shown in Figure 1(b). The delayed task assignment needs to produce a correct state of the application which has *drifted* while the load balancer was making a decision; we developed a scheme to allow this delayed assignment and discuss its impact on load balance accuracy.

We explore the trade-offs of running the load balancer using application's resources and in a separate partition. While Charm++ [2] runs the load balancer asynchronously to the application, its parallel object model is not suitable for the tightly coupled parallel applications we address in this dissertation. We present an MPMD solution that looks like SPMD to the application. We show performance benefits of balancing load lazily on the Barnes-Hut benchmark and ParaDiS dislocation dynamics simulation.

5. Summary

Research challenges of our work include describing the computation of a diverse set of applications in a uniform manner, allowing flexibility in instrumentation and efficiency in evaluating the expense of the computation, evaluating load imbalance asynchronously to the main computation, regulating the frequency of load balance correction, and reusing load balancing algorithms while providing a way to add new ones. This work makes the following contributions:

1. A model enabling the comparison of and selection among balancing algorithms for a specific state of a simulation;
2. A framework for decoupling and offloading the load balance computation, enabling lazy load balancing;
3. An accurate and fast method to evaluate and balance the load in N-Body simulations with highly non-uniform density, based on adaptive sampling.

References

[1] J. Barnes and P. Hut. A Hierarchical O(N log N) Force-Calculation Algorithm. *Nature*, 324:446–449, Dec. 1986.

[2] A. Bhatelé, L. V. Kalé, and S. Kumar. Dynamic Topology Aware Load Balancing Algorithms for MD Applications. In *SC'09*, November 2009.

[3] V. Bulatov, W. Cai, J. Fier, M. Hiratani, G. Hommes, T. Pierce, M. Tang, M. Rhee, K. Yates, and T. Arsenlis. Scalable Line Dynamics in ParaDiS. In *SC'04*, November 2004.

[4] K. Devine, E. Boman, R. Heaphy, B. Hendrickson, J. Teresco, J. Faik, J. Flaherty, and L. Gervasio. New Challanges in Dynamic Load Balancing. *Applied Numerical Mathematics*, 52(2-3):133–152, 2005.

[5] K. A. Huck and J. Labarta. Detailed Load Balance Analysis of Large Scale Parallel Applications. In *International Conference on Parallel Processing*, September 2010.

[6] N. Komatsu, T. Kiwata, and S. Kimura. Thermodynamic Properties of an Evaporation Process in Self-Fravitating N-Body systems. *Physics Review E*, 82, Aug 2010.

[7] O. Pearce, T. Gamblin, B. R. de Supinski, T. Arsenlis, and N. M. Amato. Load Balancing N-Body Simulations with Highly Non-Uniform Density. In *International Conference on Supercomputing (ICS)*, June 2014.

[8] O. Pearce, T. Gamblin, B. R. de Supinski, M. Schulz, and N. M. Amato. Quantifying the Effectiveness of Load Balance Algorithms. In *International Conference on Supercomputing (ICS)*, June 2012.

[9] K. Schloegel, G. Karypis, and V. Kumar. A Unified Algorithm for Load-Balancing Adaptive Scientific Simulations. In *SC'00*, November 2000.

[10] F. Streitz, J. Glosli, M. Patel, B. Chan, R. Yates, B. de Supinski, J. Sexton, and J. Gunnels. Simulating Solidification in Metals at High Pressure: The Drive to Petascale Computing. *Journal of Physics: Conference Series*, 46:254–267, 2006.

[11] N. R. Tallent, J. M. Mellor-Crummey, M. Franco, R. Landrum, and L. Adhianto. Scalable Fine-Grained Call Path Tracing. In *International Conference on Supercomputing (ICS)*, June 2011.

[12] C. Walshaw and M. Cross. Parallel Optimization Algorithms for Multilevel Mesh Partitioning. *Parallel Computing*, 26(12):1635–1660, 2000.

[13] M. S. Warren and J. K. Salmon. Astrophysical N-Body Simulations Using Hierarchical Tree Data Structures. In *SC'92*, November 1992.