**Dynamic Load Balancing for Parallel Particle Tracing**



In flow visualization and analysis, particle tracing is a fundamental technique for visualizing and analyzing flow fields. By tracing particles in the data domain, users can conduct many applications, for example, generating streamlines or pathlines, tracking particles from source to destination regions for source-destination queries, or computing FTLE fields to characterize the boundary of flow fields. In particle tracing, we always need to handle large data and the computational costs are expensive, so scalable and parallel solutions are needed.

The most common parallel particle tracing is data-parallel particle tracing. In the initialization stage, the flow data are statically partitioned into blocks and then distributed to processes. During the run time, particles are traced in each block and are exchanged between blocks iteratively, until every particle no longer moves or is out of domain. However, it is likely that the block workload is imbalanced. For example, some blocks may contain vortices. These vortices will trap particles and make these particles to be traced more steps. In this case, the workload of each process is imbalanced. Although there are some static load-balancing methods, such as static workload estimation and feature-based data partitioning, they always have a time-consuming preprocessing stage before running applications.

We proposed two solutions to address these problems. The first one is a dynamic load-balancing method based on data repartitioning for parallel particle tracing, as shown in Figure(a). In this method, we use general data partitioning and assignment in the initialization. During run time, we periodically perform data repartitioning to balance the workload of each process based on the estimation of the block workload.

The run-time parallel particle tracing of this method is managed into multiple rounds. In each round, each process independently traces particles inside its blocks. A particle can travel through a specified number of blocks at most in a round. We define this number as the tracing depth, represented by N. When all particles have travelled through N blocks starting from the originating block, or they have gone out the subdomain of the corresponding process or have finished before that time, this round is complete. Dynamic data repartitioning is then performed if there are still unfinished particles after one round of tracing.

Before repartitioning the data, we need first to estimate the workload of each block in the next round and take it as the basis of repartitioning. Here we present a dynamic workload estimation method, which predicts the workload of unfinished particles in each block in the next round. Our estimation method assumes the workload is directly related to the historical tracing and the number of unfinished particles. Based on the historical traces in the block, we can first calculate the average workload of a particle travelled in this block. The workload of this block is then estimated by the product of the average particle workload and the number of unfinished particles that will be traced in this block in the next round.

When the tracing depth is 1, we can directly estimate the workload of each block using above method. But when the tracing depth is larger than 1, particles can travel another a few blocks other than the originating block. In order to estimate the complete workload of particles in a block, we need to know all the blocks that these particles may access along the tracing depth. Therefore, we built an access dependency graph on the fly according to the historical tracing. The access dependency graph records the transition probability from one block to its neighboring block. It can be applied in data prefetching. In this method, based on this graph, we can predict which blocks these particles may access and how many of them will travel in these blocks. Then, we estimate the workload in each of these blocks and sum them as the total workload of the particles in the originating block for the next round.

After workload estimation of each block, we use recursive coordinate bisection (RCB) algorithm to repartition the data. The RCB algorithm takes the coordinates of data blocks as input, and the coordinates are weighted by the estimated workload of the blocks. The repartitioning algorithm will recursively divide the block coordinates into two balanced parts while also maximizing the overlap between the old and new partitions. After repartitioning, each process will then be reassigned a new data partition with near equal estimated workloads. The data blocks together with inside particles will be exchanged according to the new assignment in the communication stage. Note that we predict the workload of each block by also taking those blocks along the tracing depth into account, so those blocks will also be duplicated in the corresponding process after repartitioning.

We studied this data-repartitioning method with different data sets on Vesta, a Blue Gene/Q supercomputer at Argonne National Laboratory. Compared with other load-balancing algorithms, our method does not need any preprocessing on the raw data and does not require any dedicated process for work scheduling, while it has the capability to balance uneven workload efficiently. The performance study shows improved load balance and high efficiency of this method on tracing particles in both steady and unsteady flow.

However, the above method still requires data block movement during run time, which makes it difficult to scale up. In fact, the root cause of load imbalance is that the particle distribution always changes over time. It is very likely that the distribution becomes very uneven gradually. This observation prompted us to design a load-balancing algorithm from the perspective of particle distribution. Therefore, we proposed the second method, a dynamically load-balanced algorithm using k-d (short for k-dimensional) trees, which uses k-d tree decomposition to periodically evenly redistribute particles across processes for load balancing.

The k-d trees have been successfully used to balance workloads in N-body simulations, Delaunay tessellations, and sort-first parallel rendering by evenly (re)distribute particles, data points, or pixels across parallel processes. In order to realize the k-d tree decomposition in parallel particle tracing, there are two straightforward solutions. One could need full data duplication because processes may have particles that are located anywhere in the data domain after redistribution. However, loading data with size larger than the memory is impossible. The other solution is through dynamic data exchange. But it will induce prohibitively high communication cost. The goal of our method is to use static data repartitioning that enables dynamic load balancing with k-d trees. Therefore, these solutions are not feasible.

Our final solution is a novel redesign of k-d tree decomposition, namely, the constrained k-d tree, to redistribute particles in the data-parallel particle tracing, as shown in Figure(b). In the initialization stage, we first partition the data into non-overlapping, axis-aligned, and equal-sized blocks. For each block, we then expand it in all dimensions by adding a ghost layer, so that the block will overlap with its neighboring blocks. The thickness of the expanded block is bounded by the available memory of the process. In the computation stage, we alternately execute particle redistribution (realized by constrained k-d tree decomposition) and independent particle tracing. During the k-d tree decomposition, the cutting planes are limited in the overlapped regions of the blocks. On the basis of this, our method can balance the particles as much as possible.

This constrained k-d tree method has also been evaluated with various flow visualization and analysis tasks on Vesta. With up to 8K parallel processes, we demonstrated that compared with the baseline data-parallel particle tracing method, our constrained k-d tree approach significantly improves the performance in both load balancing and scalability. Compared with other load-balancing algorithms for parallel particle tracing, our proposed method does not require any preanalysis, does not use any heuristics based on flow features, does not make any assumptions about initial particle distribution, does not move any data blocks during the run, and does not need any master process for work redistribution.

[1] Jiang Zhang, Hanqi Guo, Xiaoru Yuan, and Tom Peterka

Dynamic Data Repartitioning for Load-Balanced Parallel Particle Tracing
In Proceedings of IEEE Pacific Visualization Symposium (PacificVis '18), pages 86-95, Kobe, Japan, April 10-13, 2018.

[2] Jiang Zhang, Hanqi Guo, Fan Hong, Xiaoru Yuan, and Tom Peterka

Dynamic Load Balancing Based on Constrained K-D Tree Decomposition for Parallel Particle Tracing
IEEE Transactions on Visualization and Computer Graphics (VIS '17), 24(1):954-963, 2018.