Accelerated Signed Distance Queries for Performance Portable Multi-Material Simulations

Jordan Backes[†], Evan DeSantola[*], *Advisors:* George Zagaris[‡], Kenneth Weiss[‡], Matt Larsen[‡], Cyrus Harrison[‡]
[†] (Co-author) Electrical and Computer Engineering, University of Missouri
[*] (Co-author) Computer Science Department, Carnegie Mellon University
[‡] (Advisor) Lawrence Livermore National Laboratory

***Abstract*—Signed distance is commonly employed to numerically represent material interfaces with complex boundaries in multi-material numerical simulations. However, the performance of computing the signed distance field is hindered by the complexity and size of the input. Recent trends in High-Performance Computing architecture consist of multi-core CPUs and accelerators that collectively expose tens to thousands of cores to the application. Harnessing this massive parallelism for computing the signed distance field presents significant challenges. Chief among them is the design and implementation of a performance portable solution that can work across architectures. Addressing these challenges to accelerate signed distance queries is the primary contribution of this work. Specifically, in this work we employ the RAJA programming model, which provides a loop-level abstraction that decouples the loop-body from the parallel execution and insulates application developers from non-portable compiler and platform-specific directives. Implementation and performance results are discussed in more detail.**

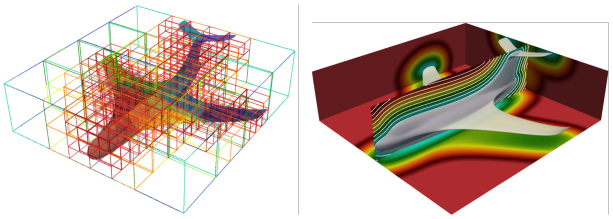
I. Introduction

Recent trends in high-performance computing architecture consist of a combination of multi-core CPUs and accelerators that collectively expose tens to thousands of cores to the application. This started with Roadrunner (IBM-Cell), the first petascale supercomputer, and has continued with Titan (NVIDIA-GPU) and more recently with the upcoming Trinity (Intel-Xeon Phi) and Sierra (NVIDIA-GPU) supercomputers. Currently over 12% of the Top 500 supercomputers incorporate accelerators[1] which are an important component in the roadmaps for supercomputers leading to the exascale era.

The focus of this work is on computing signed distances to material boundaries in the context of multi-material physics simulations. Signed distances are a common implicit shape representation in scientific computing and they been successfully employed to reconstruct surfaces in the Volume of Fluids (VOF) method and in Immersed/Embedded Boundary techniques, among others. While the spatial complexity and input size of the material boundaries impact the performance of signed distance computation, much of the underlying computation is amenable to parallelization.

Harnessing the massive parallelism of these accelerators for computing signed distances can yield significant performance boosts but presents significant challenges. Chief among them, developing a portable solution across several emerging architectures is non-trivial due to the evolving HPC landscape, where hardware architectures and programming models are changing concurrently.

[1]According to the 47^{th} edition of the Top500 list of the world's supercomputers released June, 2016. Available from http://www.top500.org.



(a) BVH subdivision (b) Signed distance field on 32^3 grid

Fig. 1. Signed distance of generic jet configuration (∼375K surface elements).

In this work, we employ the RAJA performance portability programming model[2], developed at Lawrence Livermore National Laboratory, to parallelize the computation of the signed distance field. The RAJA programming model provides a loop-level abstraction that decouples loop bodies from their execution strategy and insulates application developers from non-portable compiler and platform-specific directives. The next section presents a summary of our implementation, parallelization strategy and preliminary results.

II. Accelerated Signed Distance

Given a material Ω_i defined by an oriented material interface boundary $\partial\Omega_i$ the signed distance ϕ_i from an arbitrary point $\mathbf{x}$ to $\partial\Omega_i$ is defined in terms of the minimum distance d:

$$\phi_i(\mathbf{x}) = \begin{cases} -d, & \text{if } \mathbf{x} \in \Omega_i \text{ (inside)} \\ 0, & \text{if } \mathbf{x} \in \partial\Omega_i \text{ (on boundary)} \\ +d, & \text{if } \mathbf{x} \notin \Omega_i \text{ (outside)} \end{cases} \quad (1)$$

Geometric approaches for querying the signed distance function $\phi_i(\mathbf{x})$ commonly employ *spatial acceleration* data-structures to minimize the search space of the algorithm, i.e., the number of surface elements to check for each point. We employ a Bounding Volume Hierarchy (BVH) to partition the surface elements of the material (see Figure 1).

A. Parallel Implementation using RAJA

The BVH traversal and signed distance computation for each point are independent and can be executed in parallel.

[2]RAJA URL: https://github.com/LLNL/RAJA

```
typedef RAJA::omp_parallel_for_exec
    DefaultOMPPolicy;

RAJA::forall<DefaultOMPPolicy>
    (0,nnodes,[=](int index){
   quest::Point<double,3> pt;
   umesh->getMeshNode(index, pt.data());
   phi_store[index] = sd->computeDistance(pt);
   double dist = phi_store[index];
   phi_union[index] = std::min(dist,
       phi_wing[index]);
} );
```

Listing 1. Query loop using default OpenMP thread assignment strategy.

```
typedef RAJA::IndexSet::ExecPolicy<
      RAJA::omp_parallel_for_segit,
      RAJA::simd_exec> IndexSetPolicy;
RAJA::forall<IndexSetPolicy> >
    (idxSet,[=](int index){
   quest::Point<double,3> pt;
   umesh->getMeshNode(index, pt.data());
   phi_store[index] = sd->computeDistance(pt);
   double dist = phi_store[index];
   phi_union[index] = std::min(dist,
       phi_wing[index]);
});
```

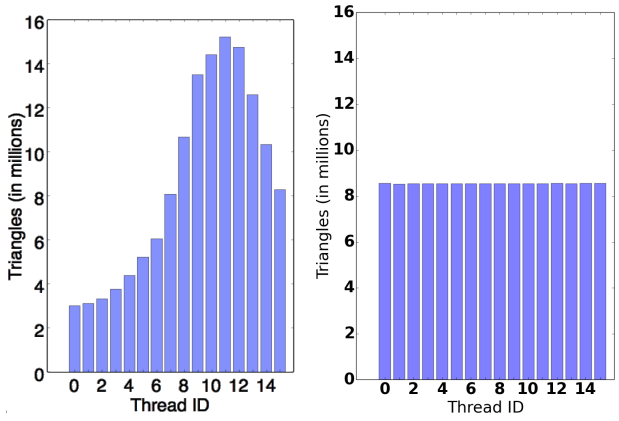
Listing 2. Query loop using `IndexSet` thread assignment strategy.

This observation led us to the straightforward parallel implementation of the inner loop illustrated in Listing 1.

We conducted a performance evaluation of this parallelization strategy using a RAJA OpenMP execution policy on a single compute node consisting of an Intel Xeon CPU @2.6 GHz equipped with 16 cores and $256GB$ of memory.

Although this simple parallelization strategy improved performance over serial execution, it also lead to significant load imbalance depending on the orientation of the surface mesh with respect to the query point. In particular, query points in some regions needed to consider a much larger set of candidate surface elements (triangles) than points in other regions. Figure 2(a) shows the total number of triangles that each of the 16 threads checked in a given cycle using the default OpenMP thread assignment strategy.

To address load imbalance, we used the RAJA programming model's `IndexSet` abstraction to partition the iteration space into contiguous chunks, called `Segments`. We used a different RAJA execution policy to assign these `Segments` to threads in a round-robin fashion. Figure 2(b) shows how this strategy, illustrated in Listing 2, improved the load balance between threads. Strong and weak scaling results for our initial and load-balanced implementations are summarized in Figure 3. After load-balancing, we achieved a $12X$ speedup over the serial version. The final poster will also include performance results from GPUs.



(a) Before load balancing (b) After load balancing

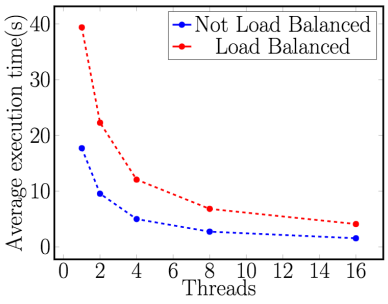
Fig. 2. Number of point-triangle intersection tests per thread on a 16 thread run before (a) and after (b) load-balancing with RAJA `IndexSets`

III. Conclusion & Future Work

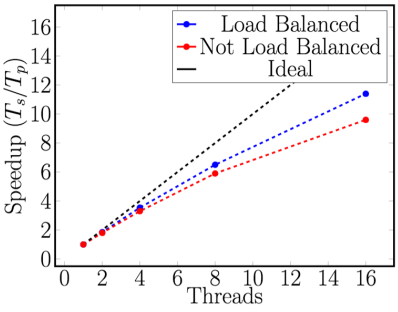
The work presented herein is geared towards accelerating signed distance queries to numerically represent material interface boundaries in multi-material simulations. Our approach employs the RAJA programming model to achieve portable performance across different architectures. Our preliminary performance evaluation indicated that there was a significant load imbalance among threads. By leveraging the RAJA IndexSet abstraction, we developed a strategy to address the load imbalances and improve the overall performance of our algorithm. Future work is focused on porting and evaluating the performance of our implementation on GPUs.

Acknowledgement

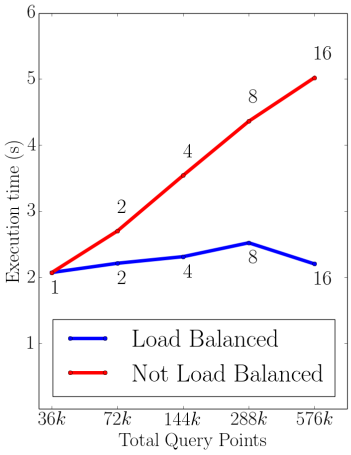
The authors would like to acknowledge support by the Institute for Scientific Computing Research (ISCR) summer scholar program for making this work possible. In addition, the authors would like to thank Rich Hornung and Rob Neely, from Lawrence Livermore National Lab, for their keen interest in this work and Will Killian from University of Delaware for useful discussions and help with RAJA. This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344. LLNL-POST-698437



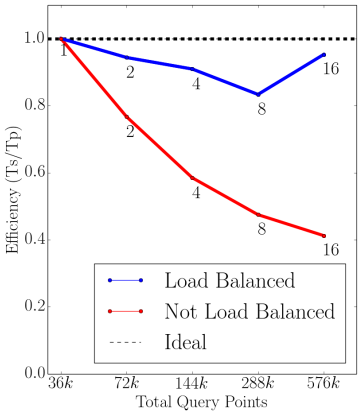
(a) Strong scaling timing



(b) Strong scaling speedup



(c) Weak scaling timing



(d) Weak scaling efficiency

Fig. 3. Preliminary performance results: Strong and weak scaling.