Introduction

Chaparral is a library designed to solve radiation heat transfer problems across large, three-dimensional meshes. A major portion of this calculation involves comparing each pair of faces in the mesh to find the View Factor (VF) matrix.

The calculation of each element in the VF matrix can be easily distributed because each view factor depends only on the two faces it corresponds to (except for possible obstructions). However, in the current implementation, the data for the full mesh must be loaded into each process. We explore an alternative scheme using Charm++ for distributing the mesh data that will eliminate the need for replicating the mesh.

Parallel Calculation of the Radiation View Factor Matrix Using Charm++

William Rosenberger
Mentor: Neil Carlson

View Factors with Charm – Chare Types

How can the VF matrix calculation take advantage of over-decomposition?

There are two entities involved in Chaparral’s calculation of the view factor matrix: facets and view factor scalars. The design for both of our approaches is based around these entities.

Three types of chares:

• Facet chare – one-dimensional array, each element corresponding to a facet in the mesh.
• VF chare – two-dimensional array, each element corresponding to the view factor between two of the facet chares.
• Aggregator chare – merges each row of VF chares into a single array.

In this implementation, each chare corresponds to a single face or a single VF. An improved but more complex implementation would instead have each chare responsible for a patch of facets and VFs.

View Factors with Charm – Naive Approach

• Facet chares: Read in a facet and send the facet data to every VF chare that needs it.
• VF chares: When the data for both facet chares are received, calculate the view factor for that pair.
• On startup: Each facet chare sends 2n messages, each message containing the data for its face.

These messages are queued in Charm++’s RTS until a processor is available to calculate the VF for the corresponding faces.

• Memory usage: There are n facets, each of which send 2n messages. This means that the memory usage excluding the VF matrix itself is:

\[ O(n^2) \]

Conclusions

In the current MPI implementation of Chaparral, the mesh file needs to be replicated to each processor. We have created a framework using Charm++ that allows distributed access to a single instance of the mesh from any process in the program, allowing for better scaling on many-core architectures.

The approaches outlined here assume each chare instance holds only one face in the mesh. While this has allowed us to experiment with the Charm++ system and improve our understanding of how the chares interact with each other, the overhead of each chare instance is overwhelming the useful work each chare does. One approach that would be useful to examine would be to have each chare responsible for a patch of faces, rather than just one. By using this patched approach, the overall overhead for the program could be decreased.

References and Acknowledgements

• Charm++, http://charm.cs.illinois.edu/research/charm

Thanks to:
• Neil Carlson (CCS-2) for his guidance with Chaparral and view factors as well as for his design suggestions.
• Jozsef Bakosi (CCS-2) for his invaluable help with learning and using Charm++.
• The ASC Integrated Codes Program for funding.
• The LANL Information Science & Technology Institute (ISTI) Parallel Computing Summer Research Internship (PCSRI).

This work was carried out under the auspices of the National Nuclear Security Administration of the US Department of Energy at Los Alamos National Laboratory supported by Contract No. DE-AC52-06NA25396.