**p4est: A Parallel Software Toolbox for Efficient Mesh Refinement and Partitioning**

By Cristen Burstedde

Any proven principles from traditional serial programs remain current and valuable in the development of scientific software. Three long-time favorites are (1) Do one thing and do it well, (2) Keep it simple, stupid, and (3) Use the source. Luke. In this article, I will review how these principles apply to the development of the p4est software library for adaptive mesh refinement (AMR). p4est adapts and partitions meshes in parallel and is used as a mesh provider for various scientific applications, as well as for general numerical mathematics libraries such as deal.ii and PETSc.

Forest-of-linear-octrees AMR

In 2007, during my time as a postdoctoral researcher at the University of Texas at Austin’s Center for Computational Geosciences and Optimization, we realized that AMR was the tool for global convection simulations due to the vast discrepancy of geological scales. We had learned a lot from Tanka Ts—author of the octdata code that implemented a pointer-based, distributed Cartesian octree and scaled well to several thousand Message Passing Interface (MPI) processes. Yet we had no solution for spherical domains. We did consider several options, including fictitious or embedded domains and the use of multiple octrees.

One day, I approached my fellow postdoc Lucas Wilcox (now at Naval Postgraduate School) with a proposal to reimplement deal.ii and build it along the algorithmic conventions. Lucas immediately suggested the connectivity of tree roots, which them­selves constitute a conforming hexahedral mesh. In two dimensions and for each face, we record which neighboring tree connects at which face and whether the connection is flipped. For each tree corner, we separately record which other trees and respective corners connect—there are any number of these. In three dimensions, we have four possible rotations at a face and an arbitrary number of neighbor edges across any edge, possibly flipped [4]. This concept allows for near arbitrary domain topologies, including periodicity (see Figure 1).

The quadrant object encodes any two- or three-dimensional tree node. Its length is a (negative) power of two in relation to the root, and the coordinates of its lower left corner are integers aligned at multiples of its length. Obtaining a parent quadrant, a given child, a sibling, or a face, edge, or corner neighbor amounts to bitwise operations on this coordinate tuple (see Figure 2). Because each tuple has an equivalent interpretation as an index in a space-filling curve, an array of quadrants sortable and searchable by the C library functions qsort and bsearch.

Figure 2. A quadrant stores the x,y,z child of a quadrant.

We benefit from the use of linear arrays of leaves that are directly suitable as send and receive buffers with regard to MPI. Since we continue the space-filling curve through all trees in order, using MPI to replicate the lower left corner and tree number of the first quadrant on each rank, the allgather routine can sufficiently encode the entire partition’s shape. One can use top-down traversals to search for arbitrary sets of local and remote points or geometric objects [2]. p4est algorithms determine message pairs and sizes ahead of time, allowing us to post asynchronous point-to-point messages with known envelopes and buffer allocations. Repartitioning the mesh works in this manner: the algorithm executes consistently in under one second (see Figure 3).

Application Interfacing

The boundary between an application and p4est is fairly sharp; the application indicates where to refine and to which p4est builds the updated mesh in parallel—with the communication out of sight on the inside.

The application may query the mesh on several levels, trading off generality and ease of interfacing. The p4est ghost layer algorithm, which collects the set of all remote neighbors adjacent to any local leaf, permits an application to define any type of discretization; we used this approach to create the p4est mesh backend for the finite element library deal.ii [1].

For some common cases, we added the globally consistent numbering of degrees of freedom as interface functions and inter­nally queried the ghost layer. For example, the original mantle convection project calls the piecewise linear variant (see Figure 4). To reduce the impact of log(N/P) time searches, Tobin Isaac (now at Georgia Institute of Technology) implemented an amortized top-down implementation that forms an application over every quadrant interface across faces, edges, and corners [5]. This approach supports all types of element-local discretizations and becomes necessary for integrating p4est with the PETSc software.

Considerations for Adaptors

Given that p4est offers flexibility and scalability, what must a user invest? The primary answer is that p4est works with non-conforming, hanging-node meshes. Many discretizations can accommodate this with the addition of element-local interpolation and projection operators. Users can decide whether these additions compromise accuracy and stability. The adaptive routine in Figure 3 is p4est’s take-over of element ordering, which determines the partition’s geometric shape. The third solution point to our encoding scheme of neighbor trees and elements, into which the application must adopt or translate. The associated authoritative documentation is still a big comment block in the p4est connectivity header file.

Our collection of examples in the source tree is now quite broad. In practice, users might study them and devise a thin wrapping layer around p4est based on their preferred conventions (C++ interface templated on the space dimension is one such example).

Acknowledgments:

I would like to thank Lucas Wilcox for his smart and deep contributions. I am grateful to Wolfgang Bangerth (Colorado State University) and Matt Knepley (University at Buffalo) for their support. Last but not least, I owe huge thanks to Omar Ghattas (UT Austin) for allowing us to work on p4est and release it as free software.

Finally, I would like to thank all current and future contributors and users and invite them to the p4est Hausdorff School,1 which will provide ample opportunity for technical discussion and hands-on experience. The school will be held July 20-24, 2020 in Bonn, Germany.

References


1 http://www.hcm.uni-bonn.de/events/events/hausdorff-school/hausdorff-school-2020.html#event=20200420


Carsten Burstedde is a professor of scientific computing at the Institute for Numerical Simulation at the University of Bonn, Germany.