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

By Carsten Burstedde

Many proven principles from traditional systems programming 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 arguably (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 is necessary for global mantle convection simulations due to the vast discrepancy of geological scales. We had learned a lot from Tiankai Tu—author of the octor code that implemented a pointer-based, distributed Cartesian octree and scaled well to several thousand Message Passing Interface (MPI) processes—yet had no solution for spherical domains. We did consider several options, including fictitious or embedded domains and the use of multiple octrees.

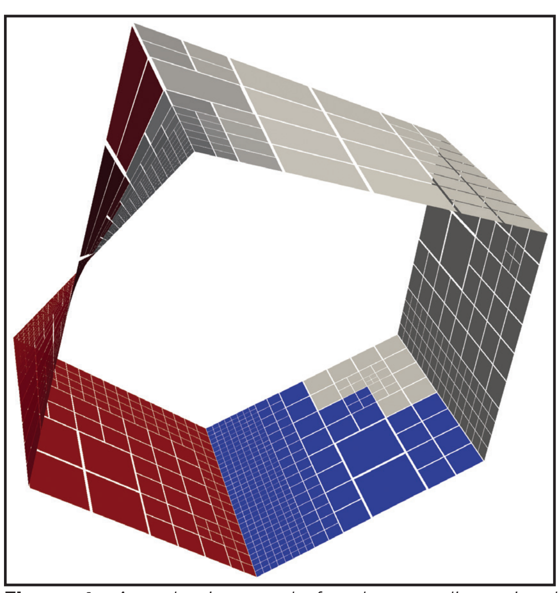

Figure 1. *An adaptive mesh for the two-dimensional Moebius strip embedded in three-dimensional space. We have executed the 2:1 balance algorithm, which limits the size difference between neighbor leaf quadrants. The color encodes a ± 1 leaf partition on three MPI ranks. Figure courtesy of [4].*

One day, I approached my fellow postdoc Lucas Wilcox (now at Naval Postgraduate School) with a proposal to reimplement octor and really understand its method of operation. Lucas immediately suggested extending the new code to a forest of octrees and building it along the algorithmic concepts of Hari Sundar and Rahul Sampath, who were working with George Biros at the University of Pennsylvania at the time, using flat arrays rather than pointers and storing only the leaves of the octree. After about a month of pair-programming, we were able to refine a two-dimensional forest manifold and write VTK files. Spring and summer resulted in 2:1 balance capabilities, a ghost layer algorithm, three-dimensional support, and node numbering for piecewise d-linear finite elements — scaling to 62e3 cores of the Texas Advanced Computing Center's Ranger supercomputer.

Simplicity, Correctness, and Performance

It was clear to us that we wanted a library that "just" does the meshing. For the forest, the first requirement was an encoding of the connectivity of tree roots, which themselves constitute a conforming hexahedral mesh. In two dimensions and for each tree 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 can be any number of these. In three dimensions, we have four possible rotations at a face and an arbitrary number of tree neighbors across any edge, possibly flipped [4]. This concept allows for near arbitrary domain topologies, including periodicity (see Figure 1).

SOFTWARE AND PROGRAMMING

```
void
p4est_quadrant_child (const p4est_quadrant_t * q, p4est_quadrant_t * r,
                      int child_id)
{
  const p4est_qcoord_t shift = P4EST_QUADRANT_LEN (q->level + 1);

  P4EST_ASSERT (p4est_quadrant_is_extended (q));
  P4EST_ASSERT (q->level < P4EST_QMAXLEVEL);
  P4EST_ASSERT (child_id >= 0 && child_id < P4EST_CHILDREN);

  r->x = child_id & 0x01 ? (q->x | shift) : q->x;
  r->y = child_id & 0x02 ? (q->y | shift) : q->y;
#ifdef P4_TO_P8
  r->z = child_id & 0x04 ? (q->z | shift) : q->z;
#endif
  r->level = q->level + 1;
  P4EST_ASSERT (p4est_quadrant_is_parent (q, r));
}
```

Figure 2. *`p4est` quadrant child computes the i-th child of a quadrant.*

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 is sortable and searchable by the `C` library functions `qsort` and `bsearch`.

Figure 2 documents several time-tested `p4est` features, such as dimension-independence. We compile both two- and three-dimensional code from the same source based on a preprocessor definition. Another feature is favoring clarity over optimization. A third and most underrated attribute is assertion; many functions have about as many assertions as lines of actual code, which reliably catches mistakes during development and certainly helps keep the number of bugs we find to below one per year on average.

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, and 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 when to repartition, and `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 leaves 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].

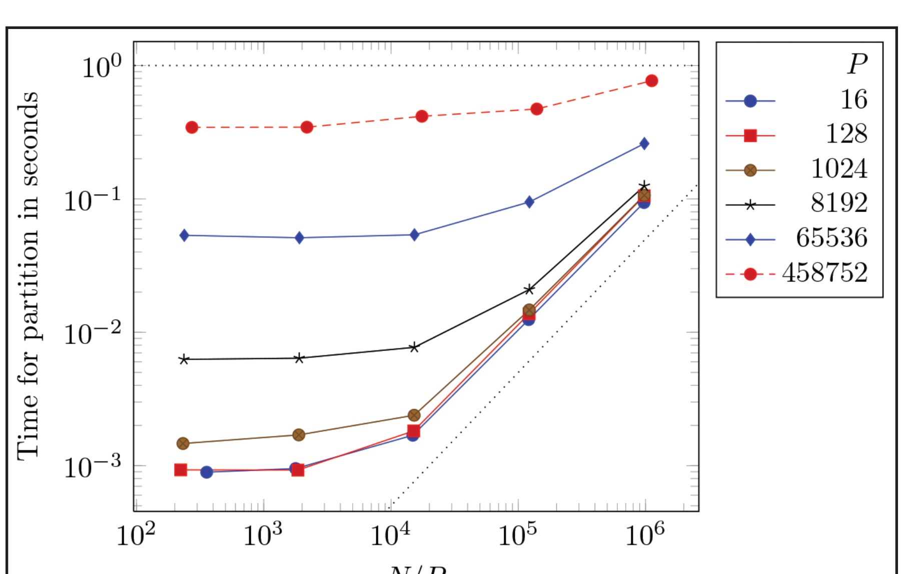

Figure 3. *Scalability of mesh repartitioning on "Juqueen." Its wall-clock time is between one second and one millisecond, revealing two regimes: one is linear in the number of local leaf quadrants N/P, and the other depends on total process count P through partition encoding. The maximum number of leaves is over .5e12. Figure courtesy of [3].*

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 iteration that informs an application about every quadrant interface across faces, edges, and corners [5]. This approach supports all types of element-local discretizations and became the basis for integrating `p4est` with the `PETSc` software.

Considerations for Adopters

Given that `p4est` offers flexibility and scalability, in 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 second attribute is `p4est`'s takeover of element ordering, which determines the partition's geometric shape. The third solution points 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 practise, users might study them and devise a thin wrapping layer around `p4est` based on their preferred conventions (a `C++` interface templated on the space dimension is one such example).

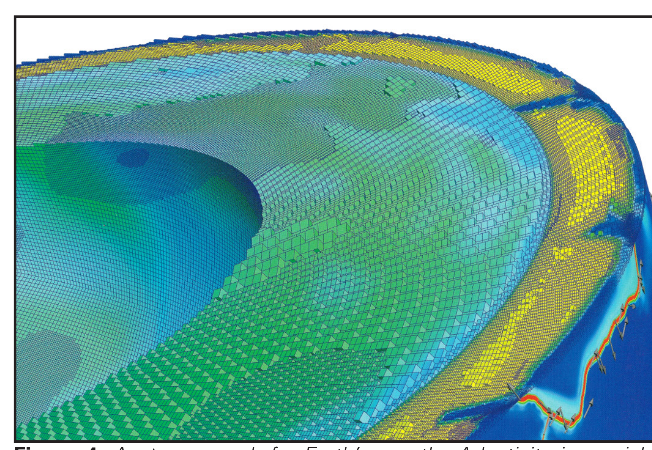

Figure 4. *A `p4est` mesh for Earth's mantle. Adaptivity is crucial to resolve tectonic plate boundaries at one-kilometer resolution; this keeps the elements coarser elsewhere for a total leaf count of only a few 100 million. Figure courtesy of [6].*

Acknowledgments: I would first like to acknowledge Lucas Wilcox without whom `p4est` would not be what it is. Secondly, I wish to thank Tobin Isaac 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 the freedom 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] Bangerth, W., Burstedde, C., Heister, T., & Kronbichler, M. (2011). Algorithms and data structures for massively parallel generic adaptive finite element codes. *ACM Trans. Math. Soft., 38*, 14:1-14:28.

[1] http://www.hcm.uni-bonn.de/events/eventpages/hausdorff-school/hausdorff-school-2020/p4est2020/

[2] Burstedde, C. (2018). Parallel tree algorithms for AMR and non-standard data access. Preprint, *arXiv:1803.08432*.

[3] Burstedde, C., & Holke, J. (2016). p4est: Scalable algorithms for parallel adaptive mesh refinement. In D. Brömmel, W. Frings, & B.J.N. Wylie (Eds.), *JUQUEEN Extreme Scaling Workshop* (pp. 49-54). Jülich, Germany: Jülich Supercomputing Centre.

[4] Burstedde, C., Wilcox, L.C., & Ghattas, O. (2011). p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM J. Sci. Comp., 33*, 1103-1133.

[5] Isaac, T., Burstedde, C., Wilcox, L.C., & Ghattas, O. (2015). Recursive algorithms for distributed forests of octrees. *SIAM J. Sci. Comp., 37*, C497-C531.

[6] Stadler, G., Gurnis, M., Burstedde, C., Wilcox, L.C., Alisic, L., & Ghattas, O. (2010). The dynamics of plate tectonics and mantle flow: From local to global scales. *Science, 329*, 1033-1038.

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