Brief p4est interface schematics

Carsten Burstedde

May 22, 2013

Abstract

We describe the general procedure of using p4est from application codes. p4est is a software library that stores and modifies a forest-of-octrees refinement structure using distributed (MPI) parallelism. It expects the description of the domain as a coarse mesh of conforming hexahedra. Non-conforming adaptive mesh refinement (AMR), coarsening, and other operations that modify the forest are implemented as p4est API functions. To inform the application about the refinement structure, several methods are provided that encode this information.

1 Starting point

We generally separate the adaptive mesh refinement (AMR) topology from any numerical information. The former is stored and modified internally by the p4est software library, while an application is free to define the way it uses this information and arranges numerical and other data. This document is inteded to describe the interface by which p4est transfers mesh information to the application.

The general, modular AMR pipeline is described in [3], which is not specific to p4est but can in principle be applied to any AMR provider. The p4est algorithms and main interface routines are destribed in [4]. An example usage of p4est as scalable mesh backend for the general-purpose finite element software deal.II is described in [1]. A reference implementation of p4est in C can be freely downloaded [2] and used and extended under the GNU General Public License. This software is best installed standalone into a dedicated directory, where an application code can then find the header and library files to compile and link against, respectively.

In this document, we document the three distinct tasks to

A create a coarse mesh (Figure 1),

B modify the refinement and partition structure internal to p4est,

C and to transfer the mesh information to an application.

Unless indicated otherwise, all operations described below are understood as MPI collectives, that is, they are called on all processors simultaneously. Currently, part A needs to be performed redundantly on all processors, which is acceptable for up to 10^5–10^6 coarse cells (octree roots). In parts B and C, runtime and memory are roughly proportional to the number of elements (octree leaves) on a given processor, independent of the number of octrees.

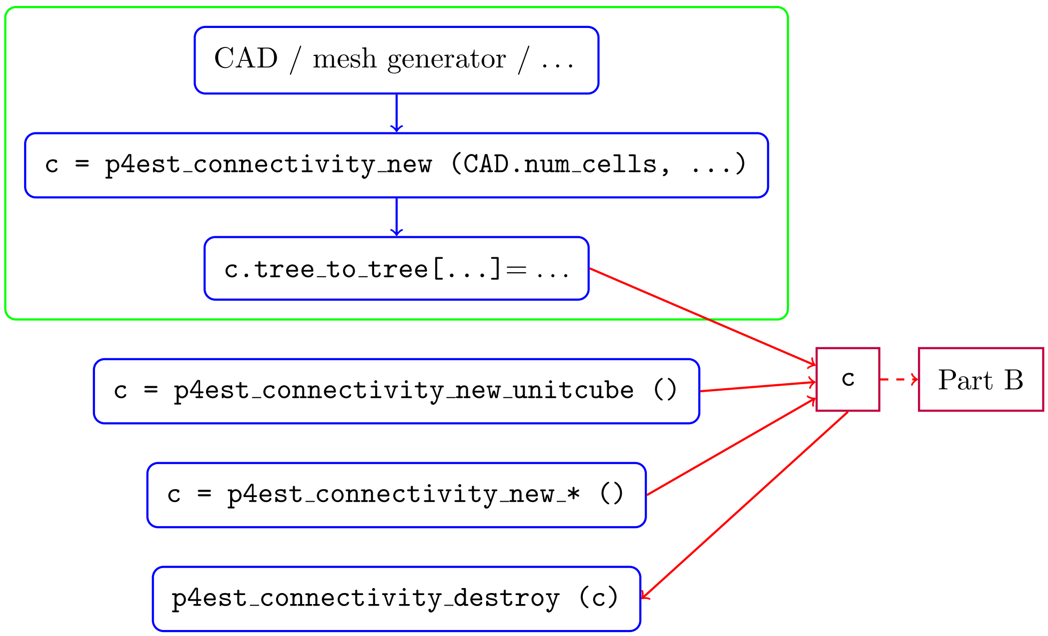

Figure 1: Part A, creating the coarse mesh connectivity. The connectivity `c` is a `C struct` that contains numbers and orientations of neighboring coarse cells. It can be created by translating CAD or mesh data file formats or by using one of several `p4est` convenience functions. The data format is documented in the big comment blocks in `p4est_connectivity.h` (2D) and `p8est_connectivity.h` (3D).

References

[1] W. Bangerth, C. Burstedde, T. Heister, and M. Kronbichler, *Algorithms and data structures for massively parallel generic adaptive finite element codes*, ACM Transactions on Mathematical Software, 38 (2011), pp. 14:1–14:28.

[2] C. Burstedde, *p4est: Parallel AMR on forests of octrees*, 2010. http://www.p4est.org/.

[3] C. Burstedde, O. Ghattas, G. Stadler, T. Tu, and L. C. Wilcox, *Towards adaptive mesh PDE simulations on petascale computers*, in Proceedings of Teragrid '08, 2008.

[4] C. Burstedde, L. C. Wilcox, and O. Ghattas, *p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees*, SIAM Journal on Scientific Computing, 33 (2011), pp. 1103–1133.