Parallel Orthogonal Recursive Bisection

Team Metropolis:
Jamshid 'James' Farzidayeri, JJ Lay, and Graham West

COMS 7900, Capstone

Abstract

In our first project, we implemented a parallel sorting algorithm which utilized a local gradient-type optimization search to equalize the amount of data across different compute nodes in order to achieve maximum efficiency. In this project, we applied this algorithm to the problem of parallel orthogonal recursive bisection, i.e., the construction of k-d trees. In order to do this, we had to heavily modify the sorting algorithm in several ways, including 1) turning it into a callable function, 2) letting the rank 0 head node perform work while still managing the tasks, 3) incorporating the use of different MPI communicators, and 4) altering the adaptive binning technique for better convergence.

In this paper, we will discuss how our k-d tree algorithm works, how we solved the various issues plaguing parallel sort (mentioned above), and how we tested and validated our work. We conclude with a discussion of the major difficulties in completing this project and how these difficulties could be minimized in the future.

Contents

1 Introduction

How we went about the project...

new GitHub repo

EXTREME CODING FTW: catch errors quickly, no merge conflicts, everyone writes/knows all the code

We all wrote prototypes in several languages before attempting the full C++ parallel version. We chose to follow the format Graham used in his MATLAB implementation.

2 Implementation

Here we discuss our implementation of the code

2.1 Altered parallel sorting

2.1.1 General changes

2.1.2 `parallelSort`

2.1.3 `adaptBins`

Although our original adaptive binning scheme performed well, we wanted something which could do even better since k-d trees require many parallelSort calls. As a review, here is our original method:

$$\begin{aligned} \Delta C &= 2.0(\mathrm{binC}_{i+1}^{0,m} - \mathrm{binC}_{i}^{0,m})/(\mathrm{binC}_{i+1}^{0,m} + \mathrm{binC}_{i}^{0,m}) \\ \Delta B &= \mathrm{binE}_{i+1}^{m} - \mathrm{binE}_{i}^{m} \\ \mathrm{binE}_{i}^{m+1} &= \mathrm{binE}_{i}^{m} + \alpha \Delta C \Delta B \end{aligned} \tag{1}$$

where $\alpha < 0.5$. This method will occasionally devolve into oscillatory behavior and not converge to the correct value. To combat this in the k-d tree project, we added a scale factor S which decreases over time:

$$\begin{aligned} S &= 1 - (1 - 0.1)(1 - \exp(-0.03m) \\ \mathrm{binE}_{i}^{m+1} &= \mathrm{binE}_{i}^{m} + \alpha S \Delta C \Delta B \end{aligned} \tag{2}$$

Since this method is local, it converges slowly at bin edges far away from high-density clusters. As such, we attempted to replace this method with a global method which uses linear interpolation to estimate where the bins would be evenly distributed.

We used the same binary search-based binning technique and stopping criterion from the previous project.

2.2 K-d tree

Used a Tree struct...

2.2.1 `buildTree`

2.2.2 `searchTree`

3 Testing and Validation

4 Results

5 Conclusion

6 Bibliography