Parallel Orthogonal Recursive Bisection

Team Metropolis:
James Farzi, JJ Lay, Graham West

April 10, 2019

Outline

Introduction

Implementation

Building the tree

To build the tree, we use several functions which perform different aspects/sections of the task

Functions:

- `buildTree`
- `buildTree_serial`
- `buildTree_parallel`
- `getSortDim`

Building the tree

`buildTree` checks the number of compute nodes in the current communicator and determines whether to call the parallel or serial versions of the code

Algorithm 1: `buildTree`$(\cdots)$

1: $q =$ Size of current communicator
2: **if** $q > 1$ **then**
3: `buildTree_parallel`$(\cdots)$
4: **else**
5: `buildTree_serial`$(\cdots)$
6: **end if**

Building the tree

`buildTree_serial` performs ORB using a single compute node

Algorithm 2: `buildTree_serial`$(data, tree, \cdots)$

1: **if** $tree.n > 1$ **then**
2: Calculate x, y, z mins, maxs, ranges, and partition center
3: Sort $data$ over $sortDim = \text{argmax}(x, y, z \text{ ranges})$
4: Split $data$: $dataL, dataR$
5: **if** $|dataL| > 0$ **then**
6: Create $tree.L$
7: `buildTree_serial`($dataL$, $tree.L, \cdots$)
8: **end if**
9: **if** $|dataR| > 0$ **then**
10: Create $tree.R$
11: `buildTree_serial`($dataR$, $tree.R, \cdots$)
12: **end if**
13: **else**
14: Store $data$ (a single point)
15: **end if**

Building the tree

`buildTree_parallel` performs ORB using a multiple compute nodes

Algorithm 3: `buildTree_parallel`($data$, $tree$, $comm$, $\cdots$)

1: Call `getSortDim`($\cdots$): calculates x, y, z mins, maxs, ranges, partition center, and returns $sortDim$
2: Sort $data$ over $sortDim$ using `parallelSort`($data$, $sortDim$, $comm$, $\cdots$)
3: **if** $myRank < numNodes/2$ **then**
4: Create $tree.L$, $commL$
5: `buildTree_parallel`($data$, $tree.L$, $comm$, $\cdots$)
6: **else**
7: Create $tree.R$, $commR$
8: `buildTree_parallel`($data$, $tree.R$, $comm$, $\cdots$)
9: **end if**

It is assumed that $tree.n > 1$ will never occur in `build/tree_parallel` since we usually deal with large amounts of data

Building the tree

`getSortDim` finds the longest axis and stores several key tree fields

Algorithm 4: `getSortDim`($data$, $tree$, $comm$, $\cdots$)

1: Each process gets it local x, y, z min and max
2: Rank 0 receives these, determines the global x, y, z min and max, determines the sortDim, and Bcast's all of these values back to the other nodes
3: The global mins/maxs, partition center, and partition radius are stored in $tree$
4: return $sortDim$

serial 501
review
conversion to function rank 0 multiple communicators
old new alternate

Validation

Validation

MATLAB Demos:

- 2D
- 3D

Results

Conclusions