

# An optimization-inspired approach to parallel sorting

Team Metropolis:  
James Farzi, JJ Lay, and Graham West  
COMS 7900, Capstone

## Abstract

We present a novel method influenced by ideas in the field optimization to efficiently sort large amounts of data in parallel on a cluster of computing nodes. We describe in depth the different challenges which beset such a method, including distributing/importing files, locally sorting the data on each node, uniformly binning the data, and exchanging data between the nodes. We also present the results of several different timing tests applied to the method. These tests demonstrate how the method scales when the number of files and/or the number of nodes is increased. Finally, we summarize the the greatest challenges in implementing the method, as well as the components of the method which were the most successful. We conclude with a discussion on ways in which the method could be improved.

## Contents

|          |   |          |
|----------|---|----------|
| <b>1</b> | <b>The Method</b>                                   | <b>2</b> |
| 1.1      | File I/O . . . . .                                  | 2        |
| 1.1.1    | Distributing files . . . . .                        | 2        |
| 1.1.2    | Importing files . . . . .                           | 2        |
| 1.2      | Sorting . . . . .                                   | 2        |
| 1.2.1    | Linked list merge sort . . . . .                    | 2        |
| 1.2.2    | Bubble sort . . . . .                               | 2        |
| 1.3      | Binning . . . . .                                   | 2        |
| 1.3.1    | Data binning w/ binary search . . . . .             | 2        |
| 1.3.2    | Stopping criterion: the uniformity metric . . . . . | 3        |
| 1.3.3    | Adapting bin edges . . . . .                        | 3        |
| 1.4      | Exchanging data . . . . .                           | 3        |
| 1.4.1    | Data swap . . . . .                                 | 3        |
| 1.4.2    | Cleanup . . . . .                                   | 3        |
| <b>2</b> | <b>Testing</b>                                      | <b>3</b> |
| 2.1      | File I/O . . . . .                                  | 3        |
| 2.2      | Sorting . . . . .                                   | 3        |
| 2.2.1    | Linked list merge sort . . . . .                    | 3        |
| 2.2.2    | Bubble sort . . . . .                               | 3        |
| 2.3      | Binning . . . . .                                   | 3        |
| 2.3.1    | Prototyping in MATLAB . . . . .                     | 3        |
| 2.3.2    | C++ runs . . . . .                                  | 3        |
| 2.4      | Exchanging data . . . . .                           | 3        |
| <b>3</b> | <b>Conclusions</b>                                  | <b>3</b> |
| 3.1      | Challenges and successes . . . . .                  | 3        |
| 3.2      | Future work . . . . .                               | 3        |

# 1 The Method

Introduction:

We used C++ with C MPI calls  
 Used GitHub  
 Workflow description

Conventions (NOTE: be consistent with the indices):

$N$ : number of nodes  
 $W$ : number of workers ( $N - 1$ )  
 $L$ : number of lines to read per file  
 $L_w$ : number of lines on the  $w$ th worker  
 $M$ : max number of allowed time steps

Indices:

- a.  $m = 0, \dots, M$  is the time step of the bin adaptation scheme (likely less than  $M$ )
- b.  $n = 0, \dots, W$  spans the nodes
- c.  $w = 1, \dots, W$  spans the workers
- d.  $i = 0, \dots, W$  spans the bin edges/indices
- e.  $j = 1, \dots, W$  spans the bin counts (this will occasionally subscript binI/E as well)
- f.  $\ell_w = 0, \dots, L_w - 1$  spans the lines on the  $w$ th worker
- g.  $k = 0, \dots, 3$  is the data column being sorted

$\text{data}_{4\ell+k}^w$ : data point on  $k$ th line and  $\ell$ th column (0 indexed) on the  $w$ th worker (1 indexed)  
 $\text{binE}_j$ : bin edges (0 indexed)  
 $\text{binI}_j^w$ : bin indices on worker  $w$  (0 indexed)  
 $\text{binC}_j^w$ : bin counts on worker  $w$  (1 indexed w.r.t.  $w$ )  
 $\text{binC}_j^0$ : bin counts on head node (sum of worker binC's)

## 1.1 File I/O

### 1.1.1 Distributing files

### 1.1.2 Importing files

## 1.2 Sorting

### 1.2.1 Linked list merge sort

### 1.2.2 Bubble sort

## 1.3 Binning

We now discuss the optimization-inspired portion of the method. We use an iterative scheme to adjust the bin edges through time. As an initial condition/approximation, we set the bin edges to be equally-spaced between the global min and max of the data.

### 1.3.1 Data binning w/ binary search

Since each node could theoretically have millions of data points, repeatedly looping through the data to find the bin in which each point lies would be inefficient. As such, we use the fact that the data is sorted to expedite the process significantly by using binary search. Now, each worker will have the same bin edges  $\text{binE}_i$  (where  $i = 0, \dots, W$ ), but the the location of the edges with respect to the data indices on each worker will likely be quite different if the data distribution across workers varies. With this in mind, for each worker, we search for the data indices at which each bin edge lies, giving us the new variables  $\text{binI}_i^w$  (note the superscript  $w$ , indicating that each worker has its own unique set of bin indices). To do this, we search for the index  $\ell$  such that

$$\text{data}_{4\ell+k}^w < \text{binE}_i^m < \text{data}_{4(\ell+1)+k}^w \quad (1)$$

giving  $\text{binI}_i^w = \ell + 1$ . We also set  $\text{binI}_0^w = 0$  and  $\text{binI}_W^w = L_w$ . We then calculate  $\text{binC}_j^w$ :

$$\text{binC}_j^w = \text{binI}_j^w - \text{binI}_{j-1}^w, \quad j = 1, \dots, W \quad (2)$$

Lastly, we calculate the total bin counts on the master node:

$$\text{binC}_j^0 = \sum_{w=1}^W \text{binC}_j^w, \quad j = 1, \dots, W \quad (3)$$

### 1.3.2 Stopping criterion: the uniformity metric

$$U^n = \max\left(\frac{c_{\max} - c_{\text{avg}}}{c_{\text{avg}}}, \frac{c_{\text{avg}} - c_{\min}}{c_{\text{avg}}}\right) \quad (4)$$

### 1.3.3 Adapting bin edges

If the data is not uniform in the first step, we move on to adapt the bins.

## 1.4 Exchanging data

### 1.4.1 Data swap

### 1.4.2 Cleanup

## 2 Testing

### 2.1 File I/O

### 2.2 Sorting

#### 2.2.1 Linked list merge sort

#### 2.2.2 Bubble sort

### 2.3 Binning

#### 2.3.1 Prototyping in MATLAB

#### 2.3.2 C++ runs

### 2.4 Exchanging data

## 3 Conclusions

We will now conclude with two discussions on 1) the most difficult and most successful aspects of our method and 2) ways of improving the both the method's performance/efficiency and our workflow as a group.

### 3.1 Challenges and successes

### 3.2 Future work