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

1 The Method

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

Conventions (NOTE: be consistent with the indices):
HEAD node (not master node)
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 wth worker
M: max number of allowed time steps
Indices:

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

$\text{data}^w_{4\ell+k}$: data point on kth line and ℓth column (0 indexed) on the wth worker (1 indexed)
binE^m_j: bin edges (0 indexed) at time step m
$\text{binI}^{w,m}_j$: bin indices on worker w (0 indexed)
$\text{binC}^{w,m}_j$: bin counts on worker w (1 indexed w.r.t. w) at time step m
$\text{binC}^{0,m}_j$: bin counts on head node (sum of worker binC's) at time step m

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. It has optimization properties since we need to find the bin edges such that we maximum the uniformity of the distribution. However, it is a constrained form of optimization since the bin edges must always maintain the relation

$$\text{binE}^m_i < \text{binE}^m_{i+1} \tag{1}$$

As we will see, the way we construct our adaptation formulae ensures that this constraint is always met.

1.3.1 Data binning w/ binary search

Since we use an iterative scheme to adjust the bin edges through time, we use as an initial condition/approximation equally-spaced bin edges between the global min and max of the data. Now, since each worker 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 binE_i^m (where $i = 0, \cdots, W$), 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 $\text{binI}_i^{w,m}$ (note the superscript w, indicating that each worker has its own unique set of bin indices). To do this, we search for the index ℓ such that

$$\text{data}_{4\ell+k}^w < \text{binE}_i^m < \text{data}_{4(\ell+1)+k}^w \tag{2}$$

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

$$\text{binC}_j^{w,m} = \text{binI}_j^{w,m} - \text{binI}_{j-1}^{w,m}, \quad j = 1, \cdots, W \tag{3}$$

Lastly, all workers send their bin counts to the head node and we calculate the total bin counts:

$$\text{binC}_j^{0,m} = \sum_{w=1}^{W} \text{binC}_j^{w,m}, \quad j = 1, \cdots, W \tag{4}$$

1.3.2 Stopping criterion: the uniformity metric

Once the head node has calculated the total bin counts it then determines how uniformly the data distributed across the bins:

$$U^m = \max\left(\frac{\text{binC}_{\text{max}}^{0,m} - \text{binC}_{\text{avg}}^{0,m}}{\text{binC}_{\text{avg}}^{0,m}}, \frac{\text{binC}_{\text{avg}}^{0,m} - \text{binC}_{\text{min}}^{0,m}}{\text{binC}_{\text{avg}}^{0,m}} \right) \tag{5}$$

If U is below a set threshold (usually ≈ 0.1), then the the data distribution is deemed to be uniform in the sense that each worker will have within $\approx 10\%$ of the average data per worker; thus sorting on each worker will take roughly equal time.

1.3.3 Adapting bin edges

If the data is not uniform in the first step, we move on to adapt the interior bin edges:

$$\begin{aligned} \Delta C &= 2.0(\text{binC}_i^{0,m} - \text{binC}_{i-1}^{0,m})/(\text{binC}_i^{0,m} + \text{binC}_{i-1}^{0,m}) \\ \Delta B &= \text{binE}_{i+1}^m - \text{binE}_i^m \\ \text{binE}_i^{m+1} &= \text{binE}_i^m + \alpha \Delta C \Delta B \end{aligned} \tag{6}$$

where $0 < \alpha < 0.5$. Now, each of these terms is designed the allow the bins to adapt the maximum amount possible without the bin edges becoming out of order.

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