Supplementary Materials for Accelerating Large-scale Bundle Adjustment for LiDAR Mapping via Parallel Computing

I. Proof of Parallel Time Complexity

In this section, we first introduce the parallel time complexity for the four standard primitives in the main paper, followed by proof of the parallel time complexity of our methods.

A. Preliminaries

Following the Brent's theorem [1], we provide the total sequential work $W(n)$ and the depth of parallel computation $D(n)$ as below:

1) `sort_by_key`: The sequential work and the depth of parallel computation are $O(n\log(n))$ and $O(\log^2(n))$, respectively [2].
2) `reduce` & `reduce_by_key`: The sequential work and the depth of parallel computation are $O(n)$ and $\log(n)$, respectively [3].
3) `exclusive_scan`: The sequential work and the depth of parallel computation are $O(n)$ and $\log(n)$, respectively [4].
4) `scatter`: The sequential work and the depth of parallel computation are $O(n)$ and $\log(n)$, respectively [3].

where n is the number of data to be processed in parallel.

B. Pre-processing

The proof of Theorem 1 is established using the following three lemmas, which provide the parallel time complexity for multi-level resolution, voxel selection, and index mapper.

Lemma 1. *The total sequential work and the depth of parallel computation for multi-level voxelization is $O(N_{\mathtt{P}}\log(N_{\mathtt{P}}))$ and $O(\log^2(N_{\mathtt{P}}))$, where $N_{\mathtt{P}}$ is the number of LiDAR points.*

Proof. Suppose the preprocessing algorithm takes $N_{\mathtt{P}}$ LiDAR points as input. At the first level of voxelization, we compute the point index and point cluster representation for each point, yielding a total sequential work of $O(N_{\mathtt{P}})$. Since the sequential time complexity of the voxelization process and the transformation into point cluster representations is $O(1)$ per point, the depth of parallel computation is $O(1)$. Subsequently, `sort_by_key` and `reduce_by_key` are utilized to sort and sum the $N_{\mathtt{P}}$ point clusters, resulting in a total sequential work of $O(N_{\mathtt{P}}\log(N_{\mathtt{P}})) + O(N_{\mathtt{P}})$ and a parallel depth of $O(\log^2(N_{\mathtt{P}})) + O(\log(N_{\mathtt{P}}))$, respectively.

For the k-th level, assuming the number of point clusters after `reduce_by_key` is $N_{\mathtt{C}}^k$, we similarly derive a total sequential work of $O(N_{\mathtt{C}}^k\log(N_{\mathtt{C}}^k)) + O(N_{\mathtt{C}}^k)$ and a parallel depth of $O(\log^2(N_{\mathtt{C}}^k)) + O(\log(N_{\mathtt{C}}^k))$.

Therefore, the total sequential work and the depth of parallel computation for multi-level voxelization are derived as

$$\begin{aligned} W_{\mathtt{multi}} &= O(N_{\mathtt{P}}\log(N_{\mathtt{P}})) + O(N_{\mathtt{P}}) + \sum_{k=1}^{\mathtt{L}-1}\left(O(N_{\mathtt{C}}^k\log(N_{\mathtt{C}}^k)) + O(N_{\mathtt{C}}^k)\right) \\ D_{\mathtt{multi}} &= O(\log^2(N_{\mathtt{P}})) + O(\log(N_{\mathtt{P}})) + \sum_{k=1}^{\mathtt{L}-1}\left(O(\log^2(N_{\mathtt{C}}^k)) + O(\log(N_{\mathtt{C}}^k))\right) \end{aligned} \tag{1}$$

Since the `reduce_by_key` primitive aggregates point clusters sharing the same voxel index and pose index, the number of point clusters does not increase:

$$N_{\mathtt{P}} \geqslant N_{\mathtt{C}}^1 \geqslant \cdots \geqslant N_{\mathtt{C}}^{k-1} \tag{2}$$

Thus, the total sequential work and the parallel depth for multi-level voxelization can be derived from Equation 1 as:

$$\begin{aligned} W_{\mathtt{multi}} &= O(N_{\mathtt{P}}\log(N_{\mathtt{P}})) \\ D_{\mathtt{multi}} &= O(\log^2(N_{\mathtt{P}})) \end{aligned} \tag{3}$$

□

Lemma 2. *The total sequential work and the depth of parallel computation for voxel selection is* $O(N'_{\mathtt{C}}))$ *and* $O(\log(N'_{\mathtt{C}}))$*, where* $N'_{\mathtt{C}}$ *is the total number of point clusters after multi-level voxelization.*

Proof. Given $N'_{\mathtt{C}}$ point clusters from the multi-level voxelization process, voxel selection initially transforms the point clusters into the world coordinate and employs `reduce_by_key` to aggregate clusters sharing the same voxel index, resulting in a total sequential work of $O(N'_{\mathtt{C}})$ and a parallel depth of $O(\log(N'_{\mathtt{C}}))$, respectively.

Assuming that there are $N_{\mathtt{V}}$ clusters after aggregation, the parallel selection kernel requires a total sequential work of $O(N_{\mathtt{V}})$ and a parallel depth of $O(1)$.

As $N'_{\mathtt{C}} \geqslant N_{\mathtt{V}}$, the sequential work and the depth of parallel computation for voxel selection can be derived as:

$$\begin{aligned} W_{\mathtt{select}} &= O(N'_{\mathtt{C}}) + O(N_{\mathtt{V}}) \\ &= O(N'_{\mathtt{C}}) \\ D_{\mathtt{select}} &= O(\log(N'_{\mathtt{C}})) + O(1) \\ &= O(\log(N'_{\mathtt{C}})) \end{aligned} \tag{4}$$

□

Lemma 3. *The total sequential work and the depth of parallel computation for index mapper is* $O(N_{\mathtt{C}} \log(N_{\mathtt{C}}))$ *and* $O(\log^2(N_{\mathtt{C}}))$*, where* $N_{\mathtt{C}}$ *is the total number of point clusters after selection.*

Proof. Given $N_{\mathtt{C}}$ selected clusters, the index mapper obtains $\mathcal{M}_{\mathtt{v}}$ by `reduce_by_key`, `exclusive_scan` and a parallel kernel, leading to a total sequential work of $O(N_{\mathtt{C}})$ and a parallel depth of $O(\log(N_{\mathtt{C}}))$. For the preparation of the mapper $\mathcal{M}_{\mathtt{s}}$, sorting and scattering are utilized, yielding a sequential work of $O(N_{\mathtt{C}} \log(N_{\mathtt{C}}))$ and a parallel depth of $O(\log^2(N_{\mathtt{C}}))$. Therefore, the total sequential work and the depth of parallel computation for the index mapper are

$$\begin{aligned} W_{\mathtt{mapper}} &= O(N_{\mathtt{C}}) + O(N_{\mathtt{C}} \log(N_{\mathtt{C}})) \\ &= O(N_{\mathtt{C}} \log(N_{\mathtt{C}})) \\ D_{\mathtt{mapper}} &= O(\log(N_{\mathtt{C}})) + O(\log^2(N_{\mathtt{C}})) \\ &= O(\log^2(N_{\mathtt{C}})) \end{aligned} \tag{5}$$

□

Finally, based on the aforementioned lemmas, we present the proof of Theorem 1 in the main paper as follows:

Proof. The total sequential work and parallel depth for preprocessing are determined by summing the corresponding values for multi-level voxelization, voxel selection, and index mapper, as follows:

$$\begin{aligned} W_{\mathtt{pre}} &= W_{\mathtt{multi}} + W_{\mathtt{select}} + W_{\mathtt{mapper}} \\ &= O(N_{\mathtt{P}} \log(N_{\mathtt{P}})) + O(N'_{\mathtt{C}}) + O(N_{\mathtt{C}} \log(N_{\mathtt{C}})) \\ D_{\mathtt{pre}} &= D_{\mathtt{multi}} + D_{\mathtt{select}} + W_{\mathtt{mapper}} \\ &= O(\log^2(N_{\mathtt{P}})) + O(\log(N'_{\mathtt{C}})) + O(\log^2(N_{\mathtt{C}})) \end{aligned} \tag{6}$$

Since the number of clusters $N_{\mathtt{C}}$ after selection is less than or equal to the number of clusters $N'_{\mathtt{C}}$ from multi-level voxelization, which is in turn less than or equal to the number of LiDAR points $N_{\mathtt{P}}$, we can simplify the aforementioned equation as follows:

$$\begin{aligned} W_{\mathtt{pre}} &= O(N_{\mathtt{P}} \log(N_{\mathtt{P}})) \\ D_{\mathtt{pre}} &= O(\log^2(N_{\mathtt{P}})) \end{aligned} \tag{7}$$

Finally, the parallel time complexity for preprocessing is derived from Brent's theorem [1] as

$$\begin{aligned} T_p^{\mathtt{pre}} &= \frac{W_{\mathtt{pre}}}{N_{\mathtt{Th}}} + D_{\mathtt{pre}} \\ &= \frac{O(N_{\mathtt{P}} \log(N_{\mathtt{P}}))}{N_{\mathtt{Th}}} + O(\log^2(N_{\mathtt{P}})) \end{aligned} \tag{8}$$

□

C. Residuals, Jacobian, and Hessian matrices

The parallel time complexity for calculating the residuals, Jacobian matrices, and Hessian matrices, as stated in Theorem 2, is derived as follows:

Proof. Assuming there are $N_{\mathtt{C}}$ point clusters for optimization, the total sequential work and parallel depth for residual calculation are determined by combining the `reduce`, `reduce_by_key`, and a parallel kernel, as follows:

$$\begin{aligned} W_{\mathtt{res}} &= O(N_{\mathtt{C}}) \\ D_{\mathtt{res}} &= O(\log(N_{\mathtt{C}})) \end{aligned} \tag{9}$$

Similarly, the total sequential work and parallel depth for computing Jacobian and Hessian matrices are determined by combining `reduce_by_key` and a parallel kernel, as follows

$$\begin{aligned} W_{\mathtt{res}} &= O(N_{\mathtt{C}}) + O(N_{\mathtt{C}}) \\ &= O(N_{\mathtt{C}}) \\ D_{\mathtt{res}} &= O(\log(N_{\mathtt{C}})) + O(1) \\ &= O(\log(N_{\mathtt{C}})) \end{aligned} \tag{10}$$

Thus, we can summarize the parallel time complexity for computing the residuals, Jacobian matrices, and Hessian matrices as follows:

$$T_p^{\mathtt{compute}} = \frac{O(N_{\mathtt{C}})}{N_{\mathtt{Th}}} + O(\log(N_{\mathtt{C}})) \tag{11}$$

□

D. Solver

The parallel time complexity for the solver, as stated in Theorem 3, is straightforward to derive as follows:

Proof. Assuming there are $N_{\mathtt{T}}$ poses, the total sequential work and parallel depth are $O(N_{\mathtt{T}})$ and $O(1)$, respectively. Therefore, the parallel time complexity, derived using Brent's Theorem [1], is

$$\begin{aligned} T_p^{\mathtt{compute}} &= \frac{O(N_{\mathtt{T}})}{N_{\mathtt{Th}}} + O(1) \\ &= \frac{O(N_{\mathtt{T}})}{N_{\mathtt{Th}}} \end{aligned} \tag{12}$$

□

II. Experiment Dataset

Table I presents detailed information on the LiDAR types, number of poses, and number of LiDAR points for each sequence.

References

[1] R. P. Brent, "The parallel evaluation of general arithmetic expressions," *Journal of the ACM (JACM)*, vol. 21, no. 2, pp. 201–206, 1974.

[2] N. Satish, M. Harris, and M. Garland, "Designing efficient sorting algorithms for manycore gpus," in *2009 IEEE International Symposium on Parallel & Distributed Processing*. IEEE, 2009, pp. 1–10.

[3] J. JáJá, *Parallel algorithms*, 1992.

[4] G. E. Blelloch, "Scans as primitive parallel operations," *IEEE Transactions on computers*, vol. 38, no. 11, pp. 1526–1538, 1989.

TABLE I
PUBLIC DATASETS

Dataset	Sequence	LiDAR Type	Pose Number	LiDAR Points
HeLiPR	Roundabout_Avia01	Avia	27,294	454,775,771
	Roundabout_Avia02		20,845	350,848,446
	Roundabout_Avia03		25,147	428,177,040
	Bridge_Avia01		21,462	334,479,348
	Bridge_Avia02		25,615	401,864,934
	Bridge_Avia03		20,082	315,660,448
MaRS-LVIG	AMtown01	Avia	12,395	290,775,260
	AMtown02		6,299	147,228,735
	AMtown03		4,934	114,432,555
	AMvalley01		10,716	228,023,462
	AMvalley02		6,460	129,705,547
	AMvalley03		4,646	91,148,217
	HKairport01		6,959	161,397,667
	HKairport02		3,552	82,337,395
	HKairport03		3,168	72,677,450
	HKairport_GNSS01		6,988	161,391,862
	HKairport_GNSS02		3,680	85,040,253
	HKairport_GNSS03		2,974	67,970,032
	HKisland01		6,328	93,716,608
	HKisland02		3,458	50,556,318
	HKisland03		2,543	35,583,442
	HKisland_GNSS01		6,553	102,006,953
	HKisland_GNSS02		3,515	53,374,366
	HKisland_GNSS03		3,035	47,204,410
MulRan	DCC01	Ouster	5,539	219,481,971
	DCC02		7,557	295,072,112
	DCC03		7,474	298,449,798
	KAIST01		8,222	297,611,447
	KAIST02		8,938	329,501,757
	KAIST03		8,625	324,506,449
	Riverside01		5,533	171,254,089
	Riverside02		8,154	256,381,439
	Riverside03		10,473	319,675,593