Parallel Point Cloud Registration

Yujie Wei, Hanzhou Lu | CMU-15418 2016 Fall

SUMMARY

In this project, we plan to parallelize the Iterative Closest Point (ICP) registration algorithm for point cloud processing system using the NVIDIA CUDA library.

ALGORITHM

Point Cloud Registration (PCR) plays an important role in computer vision since a well-aligned point cloud model is the bedrock for many subsequent applications such as Simultaneous Localization and Mapping (SLAM) in the robotics and autonomous cars domain or Automatic Building Information Modeling in the architectural industry. Nowadays, point clouds are usually usually gathered by multiple cameras or laser scanners with their own coordinate systems. The objective of Point Cloud Registration (PCR) is to search a transformation that could align a reading point cloud with a reference point cloud in a consistent coordinate system. The process of finding the transformation and the closest point involves lots of matrix operations that are usually independent with each other. Given the intermediate level of dependency and the huge size of the problem, exploiting the parallelism can be a good alternative to speed up the algorithm.

The objective of the algorithm can be formally expressed as follows:

$$\mathcal{T}_A^B = \min_{\mathcal{T}}(Error(\mathcal{T}(\mathcal{P}_A), \mathcal{Q}_B))$$

where T is the transformation, P is the reading point cloud captured in the coordinate System A, and Q is the reference point cloud captured in the coordinate system B. The error is defined as the distance between each closest point pairs in different point clouds.

The generic algorithm (Besl, 1992) is shown below:

Algorithm 1 Summary of ICP algorithm.

```
Require: ${}^{\mathbb{A}}\mathcal{P}$                                  ▷ reading
Require: ${}^{\mathbb{B}}\mathcal{Q}$                                  ▷ reference
Require: $\mathcal{T}_{init}$                                  ▷ initial transformation
  ${}^{\mathbb{A}}\mathcal{P}' \leftarrow \text{datafilter}({}^{\mathbb{A}}\mathcal{P})$                 ▷ data filters
  ${}^{\mathbb{B}}\mathcal{Q}' \leftarrow \text{datafilter}({}^{\mathbb{B}}\mathcal{Q})$                 ▷ data filters
  ${}_{i-1}{}^{i}\mathcal{T} \leftarrow \mathcal{T}_{init}$
  repeat
    ${}^{i}\mathcal{P}' \leftarrow {}_{i-1}{}^{i}\mathcal{T}({}^{i-1}\mathcal{P}')$                 ▷ move reading
    $\mathcal{M}_i \leftarrow \text{match}({}^{i}\mathcal{P}', \mathcal{Q}')$                 ▷ associate points
    $\mathcal{W}_i \leftarrow \text{outlier}(\mathcal{M}_i)$                 ▷ filter outliers
    ${}^{i+1}_{i}\mathcal{T} \leftarrow \arg\min_{\mathcal{T}} \left(\text{error}\left(\mathcal{T}\left({}^{i}\mathcal{P}'\right), \mathcal{Q}'\right)\right)$
  until convergence
Ensure: ${}^{\mathbb{B}}_{\mathbb{A}}\hat{\mathcal{T}} = \left(\bigcirc_{i}\, {}_{i-1}{}^{i}\mathcal{T}\right) \circ \mathcal{T}_{init}$
```

The program will first apply filters to the point clouds to remove outliers, then start searching from the initial transformation. In each iteration, the algorithm establishes the correspondence, creates point pairs from the reading and reference, conducts transformation over the reading point cloud, and find the transformation that minimizes the match error. The process will continue until the result converges or reaches the maximum number of iteration.

A more recent implementation of ICP is shown below:

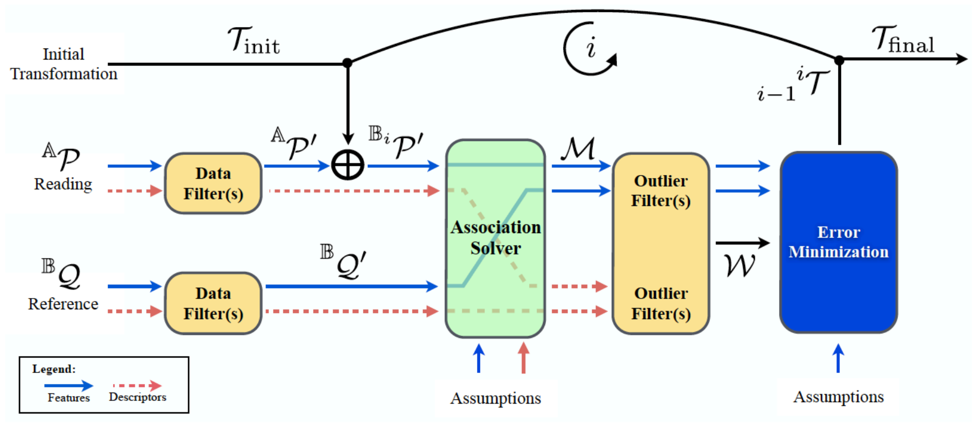
Initial Transformation
$\mathcal{T}_{\text{init}}$
i
$\mathcal{T}_{\text{final}}$
${}^{i}_{i-1}\mathcal{T}$
${}^{\mathbb{A}}\mathcal{P}$
Reading
Data Filter(s)
${}^{\mathbb{A}}\mathcal{P}'$
${}^{\mathbb{B}_i}\mathcal{P}'$
$\mathcal{M}$
Association Solver
Outlier Filter(s)
Error Minimization
$\mathcal{W}$
${}^{\mathbb{B}}\mathcal{Q}$
Reference
Data Filter(s)
${}^{\mathbb{B}}\mathcal{Q}'$
Outlier Filter(s)
Legend:
Features
Descriptors
Assumptions
Assumptions

Figure 1 Generic Scheme for Registration Algorithms (Pomerleau, 2015)

THE CHALLENGE

The challenges of the project come from the problem size and the dependency of the transformation series. A typical point cloud usually contains millions of points that may cause the cache performance to be quite poor. Also notice that the next transformation has to be calculated based on the previous result that limits the level of parallelism. Nowadays, to make the algorithm more robust when dealing with occlusions and outliers, some refined versions of ICP adopt different distance metrics and objective functions that brings new challenges. Our goal is to develop a robust parallel implementation of ICP that could be extended in the future to handle unexpected changes.

Besides, to accelerate the key computation process, the data structure should provide support for fast search and inference. According to S. Rusinkiewicz and M. Levoy (2001), k-d tree is a good candidate for ICP. Therefore, the project will also involve parallelizing the algorithm using the k-d tree data structure.

RESOURCE

The Point Cloud Library (n.d.) provides a CPU implementation of ICP based on their self-developed point cloud format which can be a good sequential reference of our project. However, since our project may run on different point cloud format, we'll implement our own sequential version and parallelize it in CPU and CUDA.

GOALS AND DELIVERABLES

SEQUENTIAL SOLUTION

In this phase, we are going to read papers and algorithm about implementation of image registration. And we will get it work as a sequential version on CPU.

PARALLEL SOLUTION ON CPU

In this phase, we will do a high level optimization on our sequential version including but not limited to parallelization over different blocks in a matrix, cache optimization and independence exploitation.

PARALLEL SOLUTION ON NVIDIA GPU

In this phase, we will do specific optimization such as vectorized operations, shared memory and other available resource in CUDA to make our code run faster on NVIDIA GPUs.

3D MODEL VISUALIZTION

We will provide a visualization of the aligned point cloud model to make the outcome understandable.

(OPTIONAL)COMBINE REGISTERATION WITH SEGMENTATION

After we accomplish goals above, we will try to do points segmentation which is partition points into different parts and label them. This part can also be speeded up by parallel programming.

PLATFORM CHOICE

We choose CUDA as our parallel library because shared memory space in NVIDIA GPUs can be used as a programmable cache. It allows us to cache specific data which is quite helpful while doing matrix calculation. For specific problem like matrix multiplication, the support for programmable cache and vector intrinsic will allow us more space to perform delicate tuning and get better performance.

SCHEDULE

	Start from	End by
Implement sequential solution	10/31/2016	11/07/2016
Implement parallel solution on CPU	11/08/2016	11/15/2015
Implement parallel solution on GPU	11/17/2015	12/02/2015
Data visualization and Report	12/03/2015	12/10/2015

REFERENCE

Besl, Paul J.; N.D. McKay (1992). "A Method for Registration of 3-D Shapes". IEEE Trans. on Pattern Analysis and Machine Intelligence. Los Alamitos, CA, USA: IEEE Computer Society. 14 (2): 239–256. doi:10.1109/34.121791

Pomerleau, François; Colas, Francis; Siegwart, Roland (2015). "A Review of Point Cloud Registration Algorithms for Mobile Robotics". Foundations and Trends in Robotics. 4 (1): 1–104. doi:10.1561/2300000035

S. Rusinkiewicz and M. Levoy (2001), "Efficient variants of the ICP algorithm," 3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on, Quebec City, Que., pp. 145-152. doi: 10.1109/IM.2001.924423

PCL - Point Cloud Library (PCL). (n.d.). Retrieved October 31, 2016, from http://www.pointclouds.org/