Near Real-time Educational Analysis of Makerspaces

Edwin Chng, Callie Sung, Tyler Yoo, Stephanie Zhang

Application and levels of parallelism

- Our application aims to support instructors in an open ended learning environment such as a Makerspace. The main goal of our project is to do near-real time data processing of the kinect data collected during class for social network and tool usage analysis.

- **Type of application:** Data-Intensive Application (Same task applied to large volumes of data)

- **Levels of parallelism exploited**: Coarse-grained (Task Level) + Fine-grained (Loop Level)

- **Types of parallelism:** Pipeline Parallelism

- **Parallel execution model:** Single Program-Multiple Data (SPMD)

Processing and analysis steps:

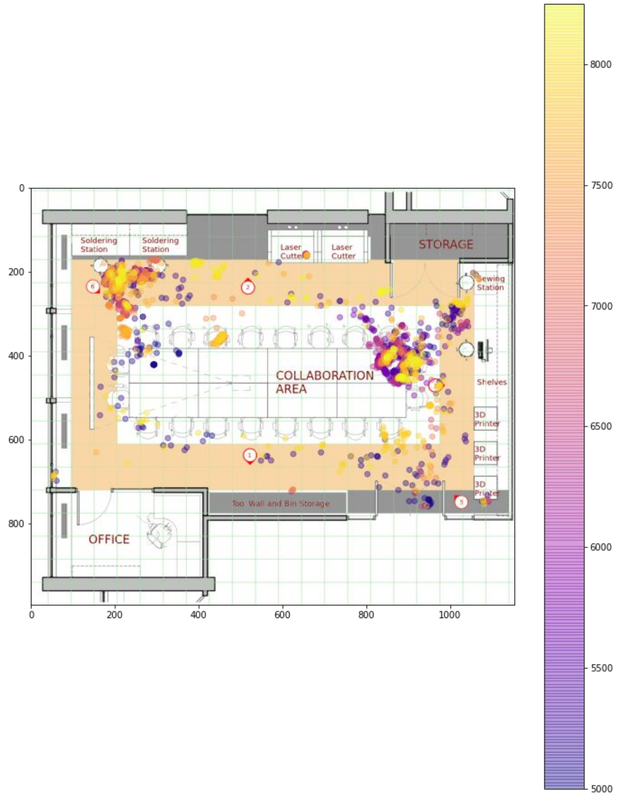
1. **Homography:** transform data from 4 different kinect cameras into one single frame.

2. **Calculate Proximity:** Calculate pairwise euclidean distance

3. **Create Social Network**: Create a symmetric, weighted social network from student proximities

4. **Calculate Metrics:** Calculate metrics of interest. E.g. influencers, tool most used, most active TF.

5. **Stretch goal: Show Outputs - Network visualization with metrics:** Create visualizations and output statistics for a given widow of time (e.g. every hour) based on the social network.



Programming Model, Infrastructure, and Code-profile

Aim of Application	Programming Model	Infrastructure	Analysis/Profiling of Code
Near Real-time data processing for tool usage and social network	Spark - Stream Data Processing (Windowed Computations)	Spark Hadoop Cluster on AWS	We have two working serial codes as of now out of the 5 steps. We profiled these two scripts. Homography: 20% Flattening the lists of coordinates 70% OpenCV homography module Interaction Analysis: 91% Accessing/editing the Pandas dataframe
Modelling and analyzing tool usage and social network within the space	Spark - Batch Data Processing (Transformations + Actions)		
	Python multiprocessing - Data-parallel computing on multiple ALUs (Loop constructs)		

Main overheads and mitigation

Main overheads

- For data preprocessing (homography and proximity calculations), the data in each partition can be processed independent of each other, and further parallelization is possible within loops, meaning task management would be the main overhead.
- For the analysis section of the program (social network and metrics), overhead lies mainly in communication between nodes, as we need summations across the entire data.
- Lastly, between steps (<-> within step), execution is sequential for the same data; the next calculations depend on the output of the previous code.

Mitigation

- Task management ➞ Find the right number of worker nodes. Refrain from creating too large clusters, as the actual data we use for the project is relatively small, but enough so that we achieve low enough latency for near-real time computation.
- Communication ➞ Minimize interprocess communication in python multiprocessing

Numerical complexity, theoretical speed-up and scalability of the algorithm

- In homography - proximity calculation - social network generation - metrics calculation, our two most computationally expensive tasks are homography and proximity calculation. The two steps after proximity calculation is a re-arrangement or summation of the results of the previous steps, which is approx. O(N).

Homography	**Proximity calculation**
- In code profiling, we see cv2.findHomography() takes 70% of the time. The rest are mainly sequential transformations of the matrices before and after homography calculations. - This function 1) solves for the perspective transformation H in X1 = H * X2, where X1 is source points, and X2 is destination point. Then it 2) transforms all the data points based on the estimated H. - H needs to be found only once per camera (we have 4); thus, asymptotically, numerical complexity is approx. $O(n^2)$, matrix multiplications for every data point. - Theoretical speedup: 1 / ((1-0.7)+0.7/p), under Amdahl's Law, where p = number of processors. **p/(0.3p + 0.7).**	- In order to calculate the proximity of two people in a certain time period, the dataset of size n is first partitioned into m subsets for each minute. - Each individual's location is then compared against every other person's location during that minute - Here, the numerical complexity would depend on the number of minutes (m) and the number of rows (n): $O(n^2/m)$ - Theoretical speedup: p (number of processors)

- Given this, scalability of our algorithm is only limited by the management and synchronization costs between a very large number of processors.