SIGMOD Programming Contest 2014

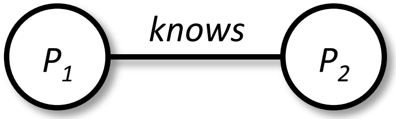
VIDA Team: Fernando Chirigati, Kien T. Pham, and Tuan-Anh Hoang-Vu

Supervised by Huy T. Vo

NYU | POLYTECHNIC SCHOOL OF ENGINEERING

Problem

- Given a *synthetic social network*, execute a set of queries *as quickly as possible*
  - Data: LDBC Social Network Benchmark
    - Main dataset: friendship relationship (*Persons Graph*)



    - Other datasets: comments, interest tags, forums, post likes, …
  - Queries: 4 types of query

- Different social network sizes are tested – from 1K to 1M persons

Solution Overview

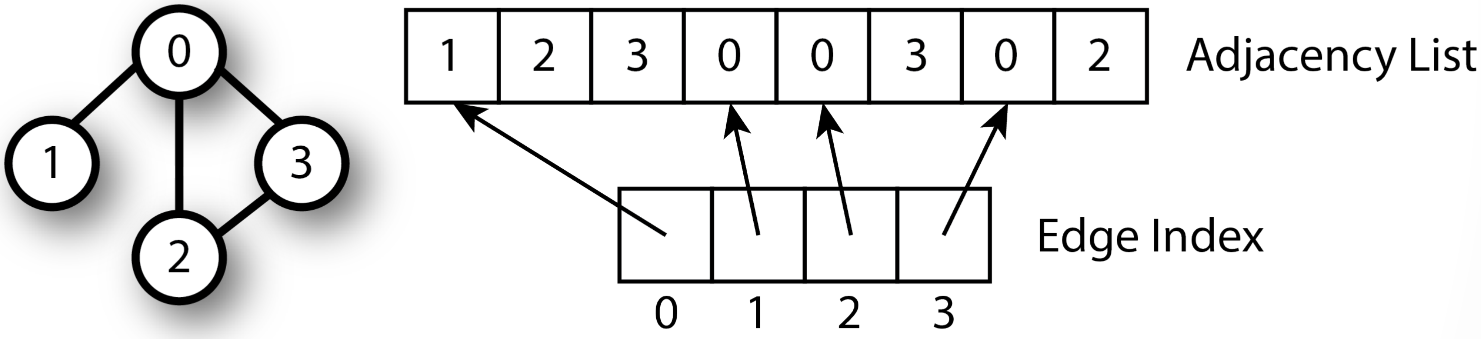
- Implementation in C++ (Standard Library and Boost)
- General optimizations
  - An efficient ***graph encoding*** to minimize dynamic allocation
  - A technique to execute multiple BFS concurrently in a single thread: ***MS-BFS*** (Multiple-Source BFS)
  - ***Multithreading strategy*** to efficiently use the available resources
- Query type-specific optimizations
  - ***Incremental reduction*** of the graph [Query Type 1]
  - ***Precomputation*** of solutions prior to query execution [Query Type 2]
  - ***Early termination*** of queries [Query Types 3 and 4]

Solution Overview

- Implementation in C++ (Standard Library and Boost)
- General optimizations
  - An efficient ***graph encoding*** to minimize dynamic allocation
  - A technique to execute multiple BFS concurrently in a single thread: ***MS-BFS*** (Multiple-Source BFS)
  - ***Multithreading strategy*** to efficiently use the available resources
- Query type-specific optimizations
  - ***Incremental reduction*** of the graph [Query Type 1]
  - ***Precomputation*** of solutions prior to query execution [Query Type 2]
  - ***Early termination*** of queries [Query Types 3 and 4]

Graph Encoding

- Use of *adjacency list*
- Implementation avoids dynamic allocations



Adjacency List

1	2	3	0	0	3	0	2

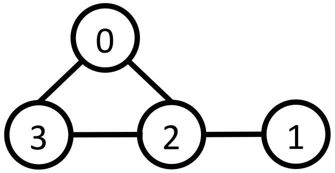
Edge Index

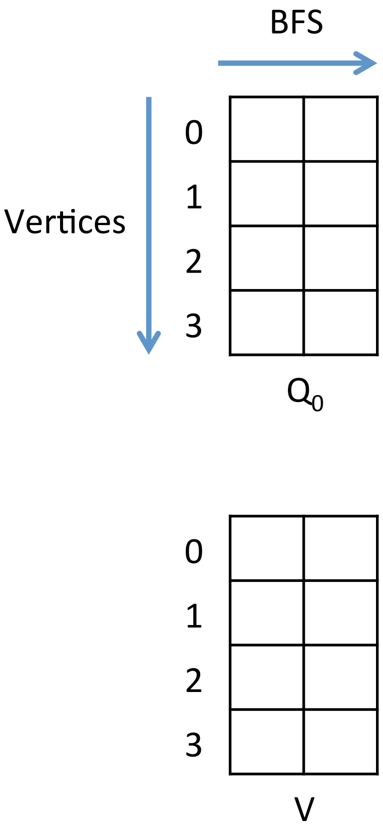
0	1	2	3

MS-BFS

- Stands for ***Multiple-Source BFS***
- General idea
  - MS-BFS can perform 64 BFS concurrently
  - There is no need for locking or multiple threads
  - MS-BFS updates queue and visited vertices using bit masks and efficient bit operations
  - Vertices can be *shared* and explored only once for multiple concurrent BFS

MS-BFS





BFS

Vertices

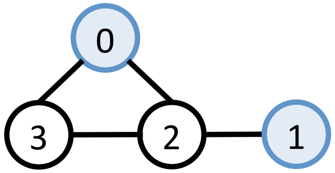
0
1
2
3

Q_0

0
1
2
3

V

MS-BFS



Hop = 0

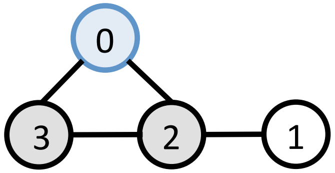
	0	***1***
0	X	
1		X
2		
3		

Q_0

0	X	
1		X
2		
3		

V

MS-BFS



Hop = 0

	0	*1*
0	X	
1		X
2		
3		

Q_0

0	X	
1		X
2		
3		

V

Hop = 1

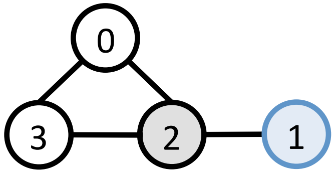
	0	*1*
0		
1		
2	X	
3	X	

Q_1

0	X	
1		X
2	X	
3	X	

V

MS-BFS



Hop = 0

	0	***1***
0	X	
1		X
2		
3		

Q_0

0	X	
1		X
2		
3		

V

Hop = 1

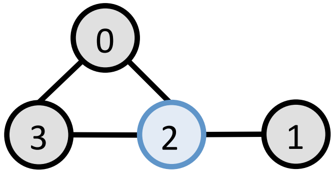
	0	***1***
0		
1		
2	X	X
3	X	

Q_1

0	X	
1		X
2	X	X
3	X	

V

MS-BFS



Hop = 0

	0	*1*
0	X	
1		X
2		
3		

Q_0

0	X	
1		X
2		
3		

V

Hop = 1

	0	*1*
0		
1		
2	X	X
3	X	

Q_1

0	X	
1		X
2	X	X
3	X	

V

Hop = 2

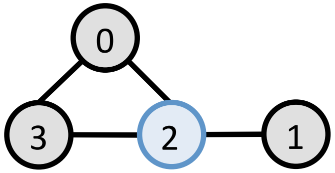
	0	*1*
0		X
1	X	
2		
3		X

Q_2

0	X	X
1	X	X
2	X	X
3	X	X

V

MS-BFS



Hop = 0

	0	*1*
0	X	
1		X
2		
3		

Q_0

0	X	
1		X
2		
3		

V

Hop = 1

	0	*1*
0		
1		
2	X	X
3	X	

Q_1

0	X	
1		X
2	X	X
3	X	

V

Hop = 2

	0	*1*
0		X
1	X	
2		
3		X

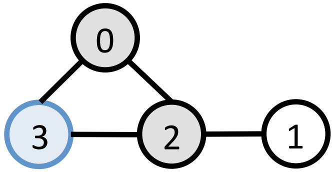
Q_2

0	X	X
1	X	X
2	X	X
3	X	X

V

Vertex 2 is being explored only once!

MS-BFS



Hop = 0

	0	*1*
0	X	
1		X
2		
3		

Q_0

	0	1
0	X	
1		X
2		
3		

V

Hop = 1

	0	*1*
0		
1		
2	X	X
3	X	

Q_1

	0	1
0	X	
1		X
2	X	X
3	X	

V

Hop = 2

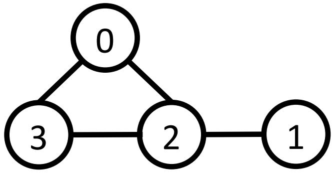
	0	*1*
0		X
1	X	
2		
3		X

Q_2

	0	1
0	X	X
1	X	X
2	X	X
3	X	X

V

MS-BFS



Hop = 0

	0	***1***
0	X	
1		X
2		
3		

Q_0

0	X	
1		X
2		
3		

V

Hop = 1

	0	***1***
0		
1		
2	X	X
3	X	

Q_1

0	X	
1		X
2	X	X
3	X	

V

Hop = 2

	0	***1***
0		X
1	X	
2		
3		X

Q_2

0	X	X
1	X	X
2	X	X
3	X	X

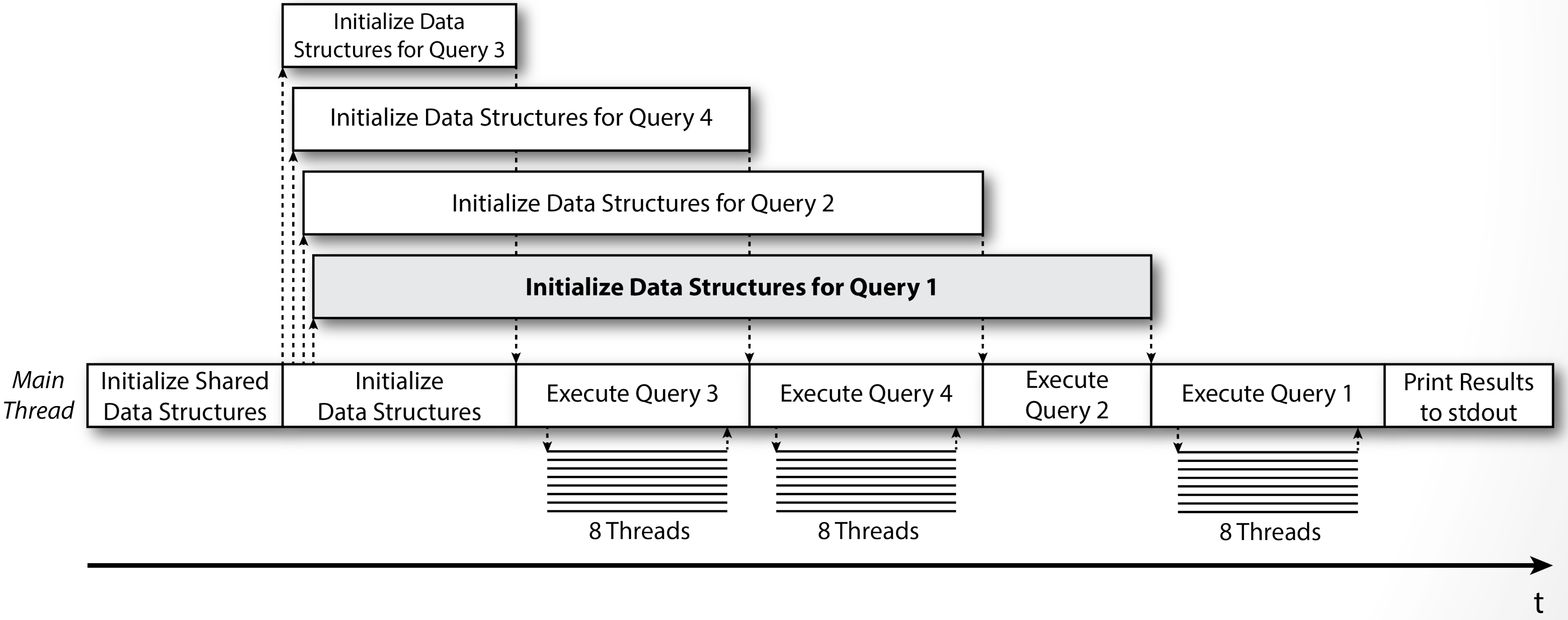
V

Bit Operations:

$Q_h[v]$ |= $Q_{h-1}[u]$ & ~$V[v]$
$V[u]$ |= $Q_h[v]$

Multithreading Strategy

- I/O for Query Type 1 is a bottleneck
- Strategy provides an efficient use of resources


Initialize Data Structures for Query 3
Initialize Data Structures for Query 4
Initialize Data Structures for Query 2
Initialize Data Structures for Query 1
Main Thread
Initialize Shared Data Structures
Initialize Data Structures
Execute Query 3
Execute Query 4
Execute Query 2
Execute Query 1
Print Results to stdout
8 Threads
8 Threads
8 Threads
t

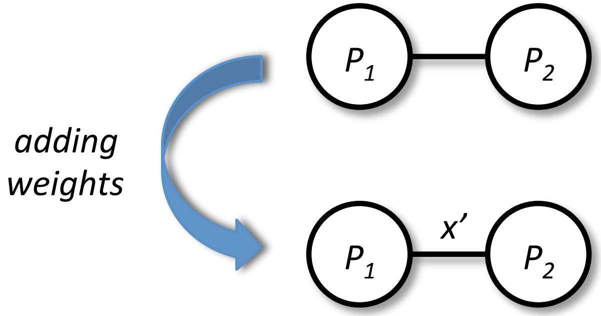
Solution Overview

- Implementation in C++ (Standard Library and Boost)
- General optimizations
  - An efficient ***graph encoding*** to minimize dynamic allocation
  - A technique to execute multiple BFS concurrently in a single thread: ***MS-BFS*** (Multiple-Source BFS)
  - ***Multithreading strategy*** to efficiently use the available resources
- Query type-specific optimizations
  - ***Incremental reduction*** of the graph [Query Type 1]
  - ***Precomputation*** of solutions prior to query execution [Query Type 2]
  - ***Early termination*** of queries [Query Types 3 and 4]

Query Type 1

query1(P_1 ,P_2 ,x) – *Find the shortest path between persons P_1 and P_2 in Persons Graph where all persons have made more than x comments to each other*

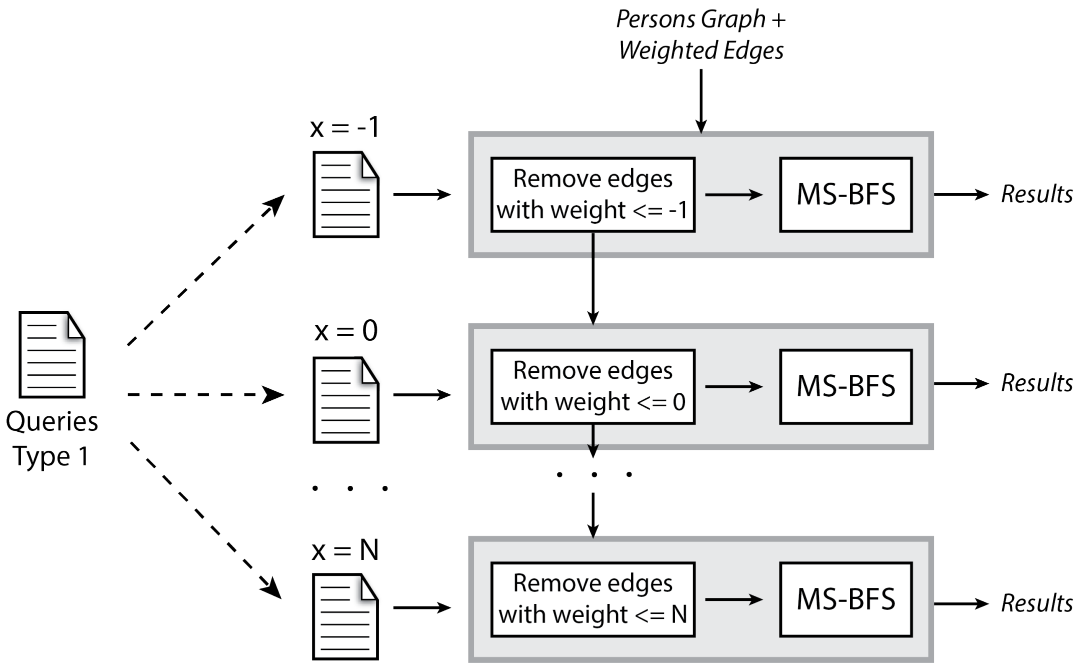
- Add number of comments in *Persons Graph*



Query Type 1

query1(P_1 ,P_2 ,x) – *Find the shortest path between persons P_1 and P_2 in Persons Graph where all persons have made more than x comments to each other*

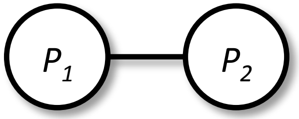
- Queries are grouped by *x* and graph is *incrementally reduced*



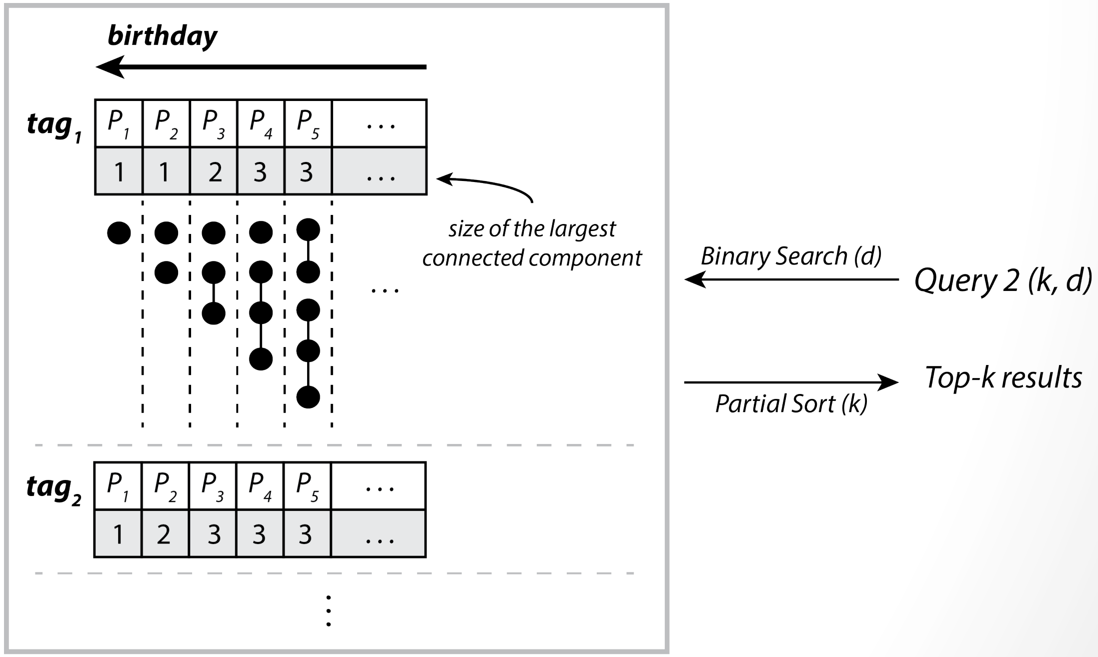
Query Type 2

query2(k, d) – *Find top k interest tags with largest communities of people that know each other and who were born on date d or later*

- *Precomputation*: size of connected components for each interest tag ordered by birthdate
- Use *binary search* to get the size of the largest component given birthdate *d*



+
interest tag information



tag₁	P_1	P_2	P_3	P_4	P_5	...
	1	1	2	3	3	...

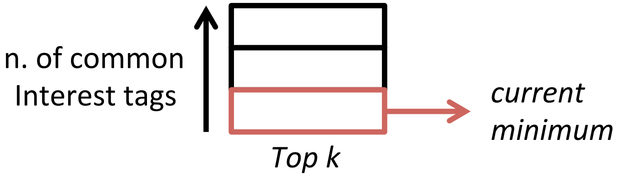
tag₂	P_1	P_2	P_3	P_4	P_5	...
	1	2	3	3	3	...

Pre-Computation

Query Type 3

query3(k, h, p) – *Find top k pairs of persons with respect to number of common interest tags; maximum number of hops between persons in Persons Graph is h; pair must be located in p, or study or work in organizations located in p*

- Co-located persons are sorted by number of interest tags
- BFS is executed in *Persons Graph* for each of these persons
- *Early termination*
  - Stop query execution when number of tags of upcoming person is less than the current minimum of top *k*



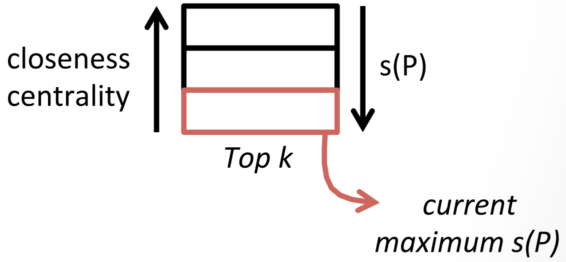
Query Type 4

query4(k ,t) – *Find top k persons with highest closeness centrality values in Persons Graph where all persons are members of forums with interest tag t*

- Closeness centrality:

$$cc(P) = \frac{(r(P)-1)\times(r(P)-1)}{(n-1)\times(s(P))}$$

- Persons who are not members in these forums are removed from *Persons Graph*
- Persons are sorted by degree
  - BFS is executed for each person
  - *Early termination*
    - Stop BFS when current accumulated s(P) is greater than the current maximum of top k

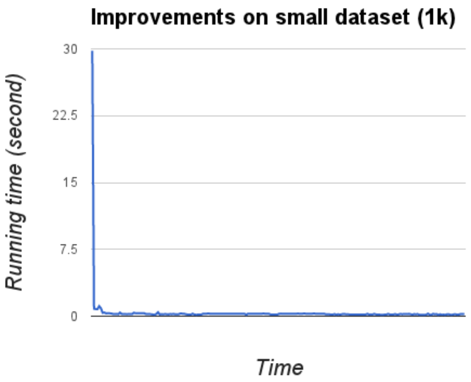


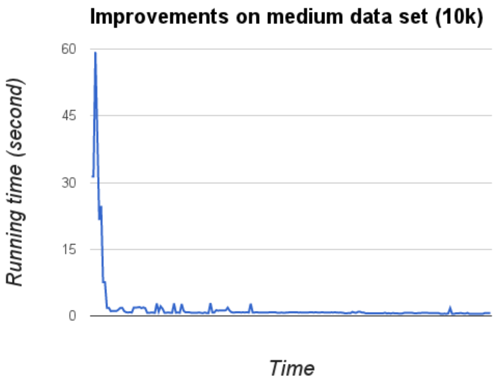
Further Optimizations

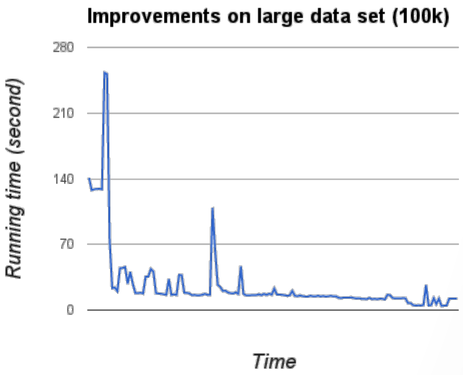
- We sacrifice memory to boost performance
- Some data structures are shared across different query types
  - E.g.: *Persons Graph*, *Persons* and *Tag* information, …
- Persons ids are normalized
- Vectors and arrays are used instead of maps
- Size of data structures are estimated based on file size
- Repeated queries are executed only once
- Memory mapped files (from Boost) are used to improve I/O performance

Statistics

- 2,556 lines of code
- 255 submissions / 45 days (01/03 – 04/14): around 5 submissions / day
- 39 failed submissions; 196 passed submissions


Improvements on small dataset (1k)


Improvements on medium data set (10k)


Improvements on large data set (100k)

Thank You!

Questions?

VIDA Team: Fernando Chirigati, Kien T. Pham, and Tuan-Anh Hoang-Vu

Supervised by Huy T. Vo

NYU | POLYTECHNIC SCHOOL OF ENGINEERING