The Problem: We want the fastest graph processing!
- High-performance graph processing is very interesting for data science
- High-performance computing is increasingly GPU/accelerator based
- Mapping irregular (graph) algorithms to GPU is hard
- Performance of irregular algorithms is data-dependent

### Thesi Goals
- Quantify performance impact of data dependence
- Model how performance relates to structural properties of the input graph
- Predict best parallelisation strategy for a given graph and algorithm
- Create an automated pipeline to repeat this work for new algorithms and parallelisation strategies

### Structural Variation
We have graphs from social networks, road networks, biology. They are different in structure and properties.

### Performance Variation
The performance of different parallelisation strategies varies by an order of magnitude or more across graphs.

### Dynamic Algorithms
For dynamic algorithms, where the relevant data changes over time, such as BFS, this effect is even stronger.

### Variation Within a Single Run
For dynamic computations like BFS, we even see these performance differences between implementation across different steps computed on the same graph.

### Thesis Results

#### BFS Prediction Results

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Optimal</th>
<th>1–2x</th>
<th>&gt;5x</th>
<th>Average</th>
<th>Worst</th>
</tr>
</thead>
<tbody>
<tr>
<td>Predicted</td>
<td>56%</td>
<td>41%</td>
<td>1%</td>
<td>1.40x</td>
<td>236x</td>
</tr>
<tr>
<td>Oracle</td>
<td>23%</td>
<td>55%</td>
<td>2%</td>
<td>1.65x</td>
<td>9x</td>
</tr>
<tr>
<td>Edge list</td>
<td>10%</td>
<td>61%</td>
<td>7%</td>
<td>2.22x</td>
<td>38x</td>
</tr>
<tr>
<td>Vertex Pull</td>
<td>0%</td>
<td>15%</td>
<td>27%</td>
<td>38.62x</td>
<td>2,671x</td>
</tr>
<tr>
<td>Vertex Push</td>
<td>9%</td>
<td>15%</td>
<td>53%</td>
<td>39.66x</td>
<td>1,048x</td>
</tr>
<tr>
<td>Push Warp</td>
<td>0%</td>
<td>0%</td>
<td>3%</td>
<td>18.69x</td>
<td>97x</td>
</tr>
</tbody>
</table>

Results across all KONECT graphs.

### Prediction Feasibility
For simple algorithms we can use this model as an oracle to select the best performing implementation for a specific graph. For algorithms whose behaviour changes at runtime, like BFS, we can do better. We can keep multiple representations in memory and switch between implementations at runtime for a classic time-space trade-off.

### In Summary
We show that using models trained on previously observed graph processing results lets us predict the best performing implementation of an algorithm for a given input graph.

We provide a framework for training such models and are investigating how much data is required to train an accurate and portable model for graph algorithms.

### References

