Good afternoon everybody. I am Reet and its my pleasure to present the poster titled ‘Evaluation of Vertex Reordering for graph applications’ that is based on a paper that was published in the IEEE International Symposium for Workload Characterization back in October 2020.

Let us start by looking at what Vertex Reordering is.

Given a graph g with vertex set v and edge set e, then a vertex reordering pi of v becomes a 1 to 1 mapping of the vertices to any linear order of 1 to n. The objective is to preserve the neighbourhood properties of a graph.

In the candidate ordering figure that we have on the right, from the point of view of vertex 1 which is shown in red, we can see the other vertices being coloured in a way where the more far apart they are, the redness starts fading and becomes more and more blue. For example, vertex 2 and 4 are orange because they are close to 1 while 6 and 7 are dark blue since they are farthest from 1. What reordering has done is that the vertex labels now reflect that closeness which was not there in the input ordering.

Next We take a look at the different reordering schemes that are in practice and classify them based on their methodology and purpose, whether they use degree as a heuristic, whether they use a sliding window based approach, or if they are based on partitioning tools, or if their objective is to reduce the fill of adjacency matrices.

Part of our objective is to compare a representative subset of these schemes based on their ability to preserve locality.

How do we compare? With the help of ‘Gap’ measures, time to formalize what that means. The idea is simple: we define the linear arrangement gap between any two vertices i and j as the absolute difference in the ranks in the ordering pi. So the average gap profile of a given ordering for an input graph becomes the average linear arrangement gap over all the edges of that graph.

A good reordering is one which minimizes this Average linear gap for an input graph.

We take a look at the distribution of gaps in input graphs and how that distribution changes for different ordering schemes. For this we used violin plots and here are the results….for three of the input graphs which produce quite a varied profile across the orderings. The observations here are that the distributions are multimodal in nature and the long tails are characteristics of log normal distribution, which show the skewedness of gap distribution.

Visually speaking, a good reordering scheme would be one which has a bottom-heavy distribution.

Part of our objective was also to compare a representative subset of these schemes based on their ability to optimize for the gap measures. And here is an important result:

The way to interpret this plot is that the y-axis shows the fraction of the 25 input data set while the x-axis shows the factor by which One scheme fares with respect to the best.

The takeaway here is that partition base schemes shown in red like rabbit order, METIS, etc. do much better than others. as for as, the average gap profile is concerned. RCM (shown in black) is not far behind.

Moving on to real-world application study, Prior works on ordering have predominantly focused on a standard suite of applications like PageRank, single source shortest paths, betweenness centrality etc. The prototypical end applications that we take a look at are:

community detection and influence maximization. Specifically, community detection as far as this poster is concerned.

Our choice of end application is motivated by the fact that these represent more advanced and complex graph operations that feature in several large scale scientific pipelines.

Furthermore, they also encapsulate two different type of algorithms. Community detection represents, classic, multi-level iterative graph algorithms, while influence maximization implementations entails running numerous stochastic BFSs over the graph to collect samples.

For this study we use 9 large real world graphs across 4 representative schemes and gauge their impact on the performance of a parallel community detection application.

We observe that grappolo ordering outperforms others in general as far as phase and iteration times are concerned.

As far as parallel efficiency is concerned, Grappolo ordering usually has the highest. While also having the lowest work per edge which is indicative of better load balancing.

While taking a look at memory latency and boundedness (which is not shown here) we go in with the expectation that the lower memory latency should correspond to boundedness at lower cache levels and lower iteration time should correlate to lower memory latency but that's not the case. Grappolo ordering makes the application more DRAM bound.

We believe that graph traversal cost may not be the dominant factor in an algorithm's execution time. The algorithms use of auxiliary data structures, here, it's a C++ map that contains the community structure, can result in additional memory accesses that negates the effect of ordering.

As far as influence maximization is concerned, not much separation was observed between different schemes because of the numerous parallel probabilistic BFSs that characterizes the core routine of the application.

To summarize, we see that partition-based schemes do better optimizing for average gap profile.

Reordering has more benefits when Community detection is the end application than when it is Influence Maximization.

It also shows us how the end application will use the memory hierarchy. And if we use the input graph enough times to amortize the cost of reordering, then it definitely adds value as a pre-processing step.

Thank you.