Good morning everybody present here. I am Reet Barik and it is my pleasure to present our paper titled “vertex reordering for real world graphs and applications: an empirical evaluation”.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Let's dive into it by prefacing this talk with what Vertex Reordering is.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

given a graph g with vertex set v and edge set e

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

we identify vertices by indexing them from 1 to n

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

then a vertex ordering pi of v becomes a 1 to 1 mapping of the vertices to any linear order of 1 to n

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We have an input ordering. Why do we need to reorder the vertices? What is the objective? Well, the goal is preserve the neighborhood properties of a graph. In other words, we want vertices which are closer to each other in a graph to have labels that reflect that closeness.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 colored 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 7 is blue since it is the farthest from 1. What reordering has done is that the vertex labels now reflect that closeness which was not there in the input ordering.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

How do we formalize this neighborhood properties? We define ‘Gap’ measures.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Here’s how we define the graph bandwidth: For any two pair of vertices i and j in the graph such that they are adjacent to each other, the maximum separation between their ranks in the vertex array is the graph bandwidth.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This is just an example to show how ordering can affect the metrics I just described. the graph on the left is the original one with the linear arrangement gap written over the edges. The reordered graph is on the right where you can see that the gap values have changed as a result. The Avg gap has reduced from 3.3 to 1.7 and the bandwidth from 5 to 3.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We take a look at the different reordering schemes that are in practice and classify them based on their methods methodology and purpose.

The Gap based methods directly try to optimize for the gap measures using algorithms like simulated annealing which doesn’t scale.

The degree based ones takes into account the degree of the vertices as a feature to come up with an ordering.

Gorder, is a window based approach which slides a window over the vertex array and optimize for a score that takes into account the neighborhood and sibling relationships among the vertex pairs in that window.

Partition based ones are actually graph partitioning or community detection tools which have been repurposed to generate an ordering.

While the fill reducing ones have the objective of generating an ordering that tries to concentrate the non zeros in the adjacency matrix of a graph near the diagonal.

Part of our objective is to compare a representative subset of these schemes based on their ability to optimize for the gap measures.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

For that we use 25 real worldl graphs as input across 11 schemes and their quality is judged by the gap measures.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

And here are the results.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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.

For example, g order shown as the second solid blue line does five times worse than the best which is metis shown in dashed red with triangle markers on 50 percent of the datasets

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The next result shows RCM (reverse cuthill McKee) outperforming others when the objective is to optimize for Graph Bandwidth.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We also took a look at the distribution of gaps in input graphs and what the distribution profile looks like 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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

community detection and influence maximization.

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 5 representative schemes and guage their impact on the performance of community detection tool Grappolo and the Influence Maximization tool Ripples.

In the results, we take a look at memory latency, which is measured as the average latency of loads to memory in cycles. We also take a look at memory hierarchy boundedness which shows us at which memory hierarchy level was the machine stalled while waiting on data.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

First, lets take a look at community detection

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We present the result as a heat map where redder is better.

The first table shows the time taken to complete the first phase of Grappolo. We consider only the first phase because subsequent phases analyzes a derivative compressed graph, which has little to do with the input ordering. The iteration heat map shows time per iteration and the next one shows the iteration count in the phase.

The fourth one shows the final modularity scores, the work percentage shows time the CPU spent in useful non-synchronization work. It is essentially a measure of parallel efficiency so higher values indicate lesser load imbalance. The final one shows average work per graph edge.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We observe that grappolo ordering outperforms others in general as far as phase and iteration times are concerned. For example, it outperforms Degree sort by a factor of around 2 to 4.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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 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 cover 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.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

We now turn our attention to influence maximization more specifically the state of the art, which is called ripples,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

A way to judge performance would be to look at the throughput of the algorithm sampling procedure, which is the hot routine of the implementation and the corresponding execution time. We will get to that.

First we take a look at the memory boundedness results for the largest graph for which is for which this analysis was possible, which is the skitter graph.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

The expectation was that the runtime profile would be more cache bound as a result of reordering which would reflect in the throughput and execution time.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

but that's not the case.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

for example degrees sort and Grappolo are more l1 bound as we can see here.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

but when we look at the throughput and execution time for those two orderings,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

they are on the opposite ends of the spectrum

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

we hypothesize that this is due to parallel threads competing for memory bandwidth and cache space.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Despite what looks like a modest role that reordering plays in the influence maximization applications we posit that one will see the benefit of reordering if the underlying implementation is made locality aware.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

RCM is best if we want to reduce the graph bandwidth.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

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

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Finally, does reordering matter?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Yes.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

It shows us how the end application will use the memory hierarchy.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Is reordering worth it?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Welllll…..

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

If we use the input graph enough times to amortize the cost of reordering.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

This brings us to the end of our presentation,

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

thank you.