We believe Reviewer C misunderstood the contributions of our paper. Their overall assessment states that we present a parallel algorithm for computing the tree decomposition (TD) for a graph; several related criticisms cite papers for this process. This work does parallelize that process, however we feel the main contribution is a parallel dynamic programming (DP) algorithm for solving MWIS after obtaining a TD, since this step dominates the time & memory complexity for large problems. Although we agree that we should have cited several prior works directly instead of referring to our other paper, the examples given (Lagergren & Bodlaender et al) differ from our work in two important ways: (1) both earlier works addressed only the problem of forming a TD in parallel; not that of subsequently performing the DP; (2) both made assumptions, such as the availability of O(n) processors, which are unrealistic for handling graphs with millions of nodes, even with a supercomputer.

Additionally, we disagree with their assessment that DP is “embarrassingly parallel” & limited by the depth of rooted tree. Our algorithms allow partial solutions to be propagated up the tree as soon as the children of a node start completing; it is not a layer-based bulk-synchronous algorithm.  Different TD nodes can have vastly different bag sizes, thus a great deal of variation in the amount of work required. While the tables for leaf nodes can be processed in an embarrassingly parallel fashion, this does not hold at all as one moves up the tree.

Finally, we would like to rebut the statement that “memory should not be an issue in real life.” As width increases, the expected # of entries in the DP table grows like O(w^{log w}) so memory is crucial to running time!  Empirically, some very small graphs (256/512 nodes from <http://neilsloane.com/doc/graphs.html>) have impossibly large memory requirements for this type of DP algorithm.  For 1dc.256, a good heuristic finds a width 138 TD.  The largest bag requires ~7 trillion table entries (estimation process outlined in serial work).  For 1dc.512 (width 313), the table requires ~5 septillion entries & 10^14 bytes of memory. Thus, to say that memory is not an issue in real life demonstrates a misunderstanding of the impact of width on TD-based DP.

Several reviewers noted our pseudocode was hard to follow; we were attempting to provide as much information as possible within the page limits, but are happy to incorporate the reviewers’ suggestions on improving the presentation of the algorithms. In a similar vein, we appreciate the suggestion of Reviewer B of including a small example graph & TD; we prepared such an example, but it was cut due to page restrictions. Once we reduce the amount of space allocated to pseudocode, we can include this, as well as additional information on how the elimination ordering relates to the TD that is formed.

Direct comparison to branch & bound algorithms (BB) for MWIS are difficult, as the size of the graphs (in terms of number of nodes & edges) solved in parallel are out of reach for most of the standard BB codes we are aware of. In our serial work, we found certain types of graphs where the TD-based DP was up to 5x faster than the leading mixed IP solver, Gurobi. Moreover, our code used a single thread/core while Gurobi used 4 cores. We didn’t feel such a comparison was warranted in the parallel case as we are unaware of any commercial MIP solvers that are able to leverage distributed processing. If the reviewer has a specific code in mind that offers parallel computation on graphs of this magnitude, we would be thrilled to run some comparisons & include them in a revised version. We would also like to clarify the hybrid nature of the algorithms; some confusion may have arisen due to the use of two separate paradigms for the TD construction versus DP steps of process. We use straight MPI + pthreads for TD. In the DP, MADNESS is used for distributing the computation at the bag-level across the machine & allows for the asynchronous updates of the tables as children complete. The OpenMP code is used for generating the independent sets within a single bag (in a single MADNESS task) and avoids interaction with futures.