The following states what was done and the success/issues that was encountered with the use of parallelism in the quest for performance increase of the MCR\_LLM solver (i.e. PLM).

# What is parallelism?

Briefly put, parallelism is the performance of several tasks in parallel, that is to say in a non-linear sequence.

# Variant

The following methods were used:

C\_plm parallelism: Here all individual for loops are done in a parallel manner as there is no communication between loops.

Non-embedded cython parallelism: Here again all for loops are done in parallel but instead of calling upon the PLM class method, they call upon the non-embedded cython module.

# Issues

In both variants no gain was observed. In fact both variants had slower computation times in comparison to the original file.

# Success

# Discussion

The biggest difficulty encountered with parallelism is the Global interpreter lock (GIL) that is present in windows. The major issue with the GIL is that when one parallelises a code, the multiple threads cannot be split on the multiple cores of a computer. Therefore, the major speed increase is not accessible. Therefore, you can have multiple threads on one core but there is no major increase in speed by doing so.

# Future ideas

More research would have to be done but I believe a C++ submodule could be used and would not be influenced by the GIL.