OpenParEM2D
Practical Tips

- mesh order
  - Generally avoid a `mesh.order` value of 1. It just takes too many iterations to converge and the results do not appear to be as accurate as with higher orders since the fields are not smooth.
  - Achievable accuracy for mesh orders of 2 and above seem about the same because they all result in smooth fields.
  - The larger the mesh order, the more quickly the results converge. Larger mesh orders can result in much quicker overall run times.
  - Larger mesh orders can lead to convergence that is more consistent from one iteration to the next because the mesh elements can better accommodate the needed field changes in one area when the mesh is refined in another.
  - The first solution at a given frequency slows quickly with increasing mesh order. After the first solution, the speed penalty is relatively small since the current solution is used as the initial guess for the next solution as long as `solution.use.initial.guess` is set to true.
  - A good strategy is to set `mesh.order` to the highest number possible that still results in a relatively quick initial solution. Experience to date has shown that mesh orders of 3 or 4 are effective.
  - ParaView only supports mesh orders up to 6. So while OpenParEM2D can use mesh orders above 6, ParaView cannot be used to view those solutions.
    - Visualization with orders above 6 may be possible with the VisIt visualizer (https://wci.llnl.gov/simulation/computer-codes/visit). Support for VisIt would be relatively easy to add to OpenParEM2D.

- `solution.tolerance`
  - Select a value of 1e-12 or 1e-13. Smaller values may cause the eigenvalue solver or the H-field solver to fail to converge. The smallest allowed value is 1e-15. If OpenParEM2D is having problems converging, try relaxing this keyword/value pair. However, tolerances as large as 1e-8 to 1e-6 can start to generate partially accurate or inaccurate results.
  - Values of 1e-14 or even 1e-15 can be tried if absolute maximum accuracy is desired. It may work, but if convergence issues occur, then the tolerance will need to be relaxed.
- `mesh.uniform_refinement.count`
  - When an initial mesh is very coarse [such a 4 triangles for a rectangular waveguide], OpenParEM2D can sometimes have trouble getting started on finding solutions to higher-order modes. To overcome this issue while avoiding the need to re-mesh the problem, the mesh density can be increased by sub-dividing every mesh element. Setting `mesh.uniform_refinement.count` to 1 causes this mesh refinement. Setting the keyword to 2 causes two uniform refinements, and so on.
- `solutions.mode.buffer`
  - If OpenParEM2D reports that not all of the requested solutions were found, simply increase `solutions.mode.buffer` until all the requested modes are found.
  - Setting `solutions.mode.buffer` higher than necessary simply slows the run time as the eigenvalue solver iteratively solves for unneeded solutions.
  - If only one mode is being solved, `solutions.mode.buffer` can usually be set to zero.
  - For coupled transmission lines, `solutions.mode.buffer` can also be typically set to zero.

- shift-and-invert method in solving the eigenvalue problem
  - The shift-and-invert method is common in solving eigenvalue problems to speed up simulations by creating more separation between eigenvalues.
  - Shift-and-invert is controlled by two keyword/value pairs in the setup file.
    - `solution.use.shift.invert` - Set to “true” to use shift-and-invert.
    - `solution.shift.factor` - When `solution.shift.invert` is set to true, in some cases the search space can be too small and solutions can be missed. This is observable when the eigenvalue solver keeps finding various small values for solutions. In this case, setting `solution.shift.factor` to a higher value, such as 10 (1 is the default) can enable to solution with larger eigenvalues. The larger solution space may run a little slower.
  - If OpenParEM2D seems to be struggling to find the expected solutions, try varying `solution.shift.factor`. Alternatively, set `solution.use.shift.invert` to “false” to use a slower but more reliable method.

- adaptive meshing
  - OpenParEM2D supports adaptive mesh refinement, where mesh elements with high errors are subdivided at each iteration to improve solution accuracy.
  - The MFEM 4.3 library does not support de-refinement for the mesh type used, so as iterative refinement progresses, if an area becomes over-meshed, it cannot be simplified even though the over-meshing can be detected.
  - To minimize over-meshing, the strategy in OpenParEM2D is to refine slowly to minimize mesh refinement in areas that ultimately lead to low errors. This strategy can lead to large numbers of iterations.
  - Mesh refinement is controlled using two keyword/value pairs in the setup file.
    - `mesh.refinement.cutoff` - Mesh elements with error greater than or equal to `mesh.refinement.cutoff` times the maximum mesh element error are marked for refinement. A lower value increases the number of elements refined per iteration and can accelerate convergence at the risk of over-meshing some areas.
    - `mesh.refinement.fraction` - The total number of elements to be refined is capped by `mesh.refinement.fraction` times the total number of mesh elements. A higher number increases the number of elements refined per iteration and can accelerate convergence at the risk of over-meshing some areas.
  - The default values for `mesh.refinement.cutoff` and `mesh.refinement.fraction` are optimized to provide balanced performance across the variety of cases in the regression suite. Customizing these values for a specific problem type can lead to substantial reductions in run times.
  - *Do not set the adaptive meshing variables such that there are almost no mesh elements refined at each iteration. With too few mesh elements added, the computed result cannot change much leading to convergence to a low-accuracy answer.*

- running OpenParEM2D
  - Use builder to create the needed input files when it supports the needed transmission line or waveguide type. Besides being vastly easier and quicker, experience to date has shown that it results in higher quality meshes for transmission lines due to the added mesh meshpoints and applied scaling.
  - Repeated runs
    - Repeated runs on 1 processor produce the exact same results every time.
    - Repeated runs on 2 or more processors can produce very slightly different results with each run.
      - Each run is equally as accurate as the others.
      - The root cause of the variability is the iterative eigenvalue solution combined with variations in how the calculation is spread across the processors and the orders in which the processors complete their tasks.
    - So if you see very slight changes in the results from run to run when running with 2 or more processors, there is nothing to be concerned about.
  - CPU loading
    - Run top while a longer job is running. For mpirun -np N, there should be N instances of OpenParEM2D running, with each peaking at 100% CPU utilization. If one or more processes are running at over 100%, then re-visit the installation documentation and check the required environment variable settings.

- parallel processing gains
  - Desktop microprocessors are not optimized for high-performance computing (HPC).
  - It is expected that the internal bandwidth of such a microprocessor saturates at 4-5 cores. On one consumer grade desktop, the experience is:
    - Running with 2 cores is a little slower than running with 1 core due to MPI overhead.
    - Running with 3 cores is about 2X faster than running with 1 core.
    - Running with 4 cores results in a small increase in performance over 3 cores.
    - Running with 5 cores may or may not result in performance improvement.
    - Running with 6 cores starts to show a slow down.
    - Running with even more cores then shows increasingly slower performance due to MPI overhead overwhelming the microprocessor's internal bandwidth.
  - Testing on HPC-oriented microprocessors is needed to find good hosts and to help find opportunities for MPI optimization in OpenParEM2D.
- skin depth
  - OpenParEM2D does not calculate the fields penetrating into conductors. For the conductor loss calculations to be accurate, the conductors must be thick compared to the skin depth.
  - It is up to the user to ensure that this condition is met.

- memory
  - 2D simulators are not generally big users of memory, but high-accuracy setups can drive high levels of memory usage.
  - OpenParEM2D is not programmed to gracefully exit if the computer runs out of memory. If you are executing a long, high-accuracy run, and the simulation fails without a good error message, then you have probably run out of memory.
  - Run top while running the failing job and watch the free amounts for both the memory and the swap space. If they are at or close to zero as the job fails, then your computer ran out of memory.
  - If the computer is running out of memory on the simulation, the options are to
    - Close other programs to free up memory.
    - Scale back the simulation to use less memory.
    - Increase the swap space on the computer.
      - This will likely cause a large increase to the run time because swapping to disk is slow.
    - Add RAM to the computer.
- Ubuntu tip
  - #tracker-store - uses 100% of a cpu while OpenParEM2D is running
  - To kill it, execute the following
    - systemctl --user mask tracker-store.service tracker-miner-fs.service tracker-miner-rss.service tracker-extract.service tracker-miner-apps.service tracker-writeback.service
    - tracker reset --hard
  - Note that you will lose file indexing for other applications.