```python
    """

    A minimum working example of a NEURON gap junction over MPI

    Author: Tom Close
    Date: 8/1/2013
    Email: tclose@oist.jp
    """

    import os
    import argparse
    import numpy as np
    # This is a hack I use on our cluster, to get MPI initialised=True. There is probably something
    # wrong with our setup but I can't be bothered trying to work out what it is at this point. All
    # suggestions welcome :)
    try:
        from mpi4py import MPI #@UnresolvedImport @UnusedImport
    except:
        print "mpi4py was not found, MPI will remain disabled if MPI initialized==False on startup"
    from neuron import h, load_mechanisms
    # Not sure this is necessary, or whether I can just use h.finitialize instead of h.stdinit
    h.load_file('stdrun.hoc')

    # The GID used for the gap junction connection. NB: this number is completely independent from the
    # GID's used for NEURON sections.
    GID_FOR_VAR = 0

    # Arguments to the script
    parser = argparse.ArgumentParser(description=__doc__)
    parser.add_argument('--plot', action='store_true', help="Plot the data instead of saving it")
    parser.add_argument('--output_dir', type=str, default=os.getcwd(),
                        help="The directory to save the output files into")
    parser.add_argument('--gap_mechanism_dir', type=str, default=os.getcwd(),
                        help="The directory to load the gap mechanism from")
    args = parser.parse_args()

    # Load gap mechanism from another directory if required
    if args.gap_mechanism_dir is not os.getcwd():
        load_mechanisms(args.gap_mechanism_dir)
    # Get the parallel context and related parameters
    pc = h.ParallelContext()
    num_processes = int(pc.nhost())
    mpi_rank = int(pc.id())
    print "On process {} of {}".format(mpi_rank+1, num_processes)

    print "Creating test network..."
    # The pre-synaptic cell is created on the first node and the post-synaptic cell on the last node
    # (NB: which will obviously be the same if there is only one node)
    if mpi_rank == 0:
        print "Creating pre-synaptic cell on process {}".format(mpi_rank)
        # Create the pre-synaptic cell
        pre_cell = h.Section()
        pre_cell.insert('pas')
        # Connect the voltage of the pre-synaptic cell to the gap junction on the post-synaptic cell
        pc.source_var(pre_cell(0.5)._ref_v, GID_FOR_VAR)
        # Stimulate the first cell to make it obvious whether gap junction is working
        stim = h.IClamp(pre_cell(0.5))
        stim.delay = 50
        stim.amp = 10
        stim.dur = 100
        # Record Voltage of pre-synaptic cell
        pre_v = h.Vector()
        pre_v.record(pre_cell(0.5)._ref_v)
    if mpi_rank == (num_processes - 1):
        print "Creating post-synaptic cell on process {}".format(mpi_rank)
        # Create the post-synaptic cell
        post_cell = h.Section()
        post_cell.insert('pas')
        # Insert gap junction
        gap_junction = h.gap(0.5, sec=post_cell)
        gap_junction.g = 1.0
        # Connect gap junction to pre-synaptic cell
        pc.target_var(gap_junction._ref_vgap, GID_FOR_VAR)
        # Record Voltage of post-synaptic cell
        post_v = h.Vector()
        post_v.record(post_cell(0.5)._ref_v)
    # Finalise construction of parallel context
    pc.setup_transfer()
    # Record time
    rec_t = h.Vector()
    rec_t.record(h._ref_t)
    print "Finished network construction on process {}".format(mpi_rank)

    # Run simulation
    print "Setting maxstep on process {}".format(mpi_rank)
    pc.set_maxstep(10)
    print "Finitialise on process {}".format(mpi_rank)
    #h.finitialize(-60)
    h.stdinit()
    print "Solving on process {}".format(mpi_rank)
    pc.psolve(100)
    print "Running worker on process {}".format(mpi_rank)
    pc.runworker()
    print "Completing parallel context on process {}".format(mpi_rank)
    pc.done()
    print "Finished run on process {}".format(mpi_rank)

    # Convert recorded data into Numpy arrays
    t_array = np.array(rec_t)
    if mpi_rank == 0:
        pre_v_array = np.array(pre_v)
    if mpi_rank == (num_processes - 1):
        post_v_array = np.array(post_v)

    # Either plot the recorded values
    if args.plot and num_processes == 1:
        print "Plotting..."
        import matplotlib.pyplot as plt
        if mpi_rank == 0:
            pre_fig = plt.figure()
            plt.plot(t_array, pre_v_array)
            plt.title("Pre-synaptic cell voltage")
            plt.xlabel("Time (ms)")
            plt.ylabel("Voltage (mV)")
        if mpi_rank == (num_processes - 1):
            pre_fig = plt.figure()
            plt.plot(t_array, post_v_array)
            plt.title("Post-synaptic cell voltage")
            plt.xlabel("Time (ms)")
            plt.ylabel("Voltage (mV)")
        plt.show()
    else:
        # Save data
        print "Saving data..."
        if mpi_rank == 0:
            np.savetxt(os.path.join(args.output_dir, "pre_v.dat"),
                       np.transpose(np.vstack((t_array, pre_v_array))))
        if mpi_rank == (num_processes - 1):
            np.savetxt(os.path.join(args.output_dir, "post_v.dat"),
                       np.transpose(np.vstack((t_array, post_v_array))))
    print "Done."
```