```
import sys
import numpy as np
#import matplotlib.pyplot as plt
from mpi4py import MPI
import json
with open("./meta.json","r") as f:
    meta = json.load(f)
    sys.path.append(meta["nrnpy-path"])
import neuron
sys.path.append("./modules")
import generateNetworkMod
import ioMod
import argparse
from glob import glob
import time
import datetime
from collections import OrderedDict

neuron.h.load_file("nrngui.hoc")
p = argparse.ArgumentParser(description='Multineuron simulator for Neuron with python',
                            add_help=True)
p.add_argument("--nostore", action="store_true")
p.add_argument('-f', '--file', help="execute simulation with target directories parameter
               default='')
p.add_argument('-s', '--setting', help="execute simulation with target setting files.", de
#p.add_argument('-h', '--home', help="simulation home path", default='')
args = p.parse_args()

# variable
noDisplay = True #for remote
Setting = (args.setting != "")
File = (args.file != "")
#Home = (args.home != "")

#if Home:

paths = {}
# default v_init and tstop and downsample rate
sim_params = [-65,1000,1]

# load external files
# parsing json simulation setting file
with open(args.setting) as f:
    df = json.load(f)
    paths['dynamics_def_path'] = df['dynamics_def_path']
    paths['connection_def_path'] = df['connection_def_path']
    paths['stim_setting_path'] = df["stim_setting_path"]
    paths['record_setting_path'] = df["record_setting_path"]
    if "v_init" in df:
        sim_params[0] = df["v_init"]
    sim_params[1] = df["tstop"]
    if "downsample" in df:
        sim_params[2] = df["downsample"]
    paths['setting_file_path'] = args.setting

print("nostore = " + str(args.nostore) + "\n")

# read external file
neuron_num, dynamics_list, neuron_connection, stim_settings, rec_list = ioMod.readExternal

v_init = sim_params[0]
tstop = sim_params[1]

# parallel context
simManager = generateNetworkMod.SimulationManager(N=neuron_num, dynamics_list=dynamics_lis
host_info = [simManager.pc.nhost(),simManager.pc.id()]


# recoding setting
print("set records")
```

```
# make unified time
strtime = ""
if host_info[1] == 0:
    strtime = datetime.datetime.now().isoformat().replace(":","_")
sref = neuron.h.ref(strtime)
simManager.pc.broadcast(sref, 0)
simManager.pc.barrier()
strtime = sref[0]

rec_vector_list = []
rec_nc_list = []
rec_t = neuron.h.Vector()
rec_t.record(neuron.h._ref_t)
for rec in rec_list:
    if "target_cellname" in rec:
        id = simManager.nametoid[rec["target_cellname"]]
    elif "target_cellid" in rec:
        id = rec["target_cellid"]
    else:
        print("each elements of rec file must contain key named target_cellname or target_
        exit()

    if "spike_record" not in rec or rec["spike_record"] is False:
        if id in simManager.generated_cellid_list:
            rec_vector = neuron.h.Vector()
            if "value" not in rec:
                value = "v"
            else:
                value = rec["value"]
            rec_var = getattr(simManager.cells[simManager.generated_cellid_list.index(id)]
            rec_vector.record(rec_var)
            rec_vector_list.append([rec,rec_vector])
    else:
        if id in simManager.generated_cellid_list:
            rec_vector = neuron.h.Vector()
            neuron.h('objref nil')
            src = simManager.cells[simManager.generated_cellid_list.index(id)].cell[rec["s
            nc = neuron.h.NetCon(getattr(src(rec["section"]["point"]),"_ref_v"), neuron.h.
            if "opt" in rec:
                for opt in rec["opt"].items():
                    setattr(nc,opt[0],opt[1])
            nc.record(rec_vector)
            rec_nc_list.append(nc)
            rec_vector_list.append([rec,rec_vector])


print("setting finish")
simManager.pc.barrier()

# simulation
print("before setup")
simManager.pc.set_maxstep(10)
simManager.pc.setup_transfer()
neuron.h.finitialize(sim_params[0])
print("before finitialize")
neuron.h.stdinit()
print("before RUN")
simManager.pc.barrier()
# gather the results
print("before psolve")
simManager.pc.psolve(tstop)
print("Finish psolve")

# gather the results
# https://www.neuron.yale.edu/neuron/static/py_doc/modelspec/programmatic/network/parcon.h
r_v_list = [[r_v[0],r_v[1].as_numpy()] for r_v in rec_vector_list]
# convert results
t = rec_t.as_numpy()

# downsampling
log_v_list = []
log_t = t[0:t.size:sim_params[2]]


if sim_params[2] != 1:
```

```
    for v_list in r_v_list:
        log_v_list.append([v_list[0],v_list[1][0:v_list[1].size:sim_params[2]]])
else:
    log_v_list = r_v_list
# pickle all parameters, settings, and results
if args.nostore is False:
    ioMod.pickleData(paths=paths, conditions=[v_init, tstop], results={'t': log_t, 'r_v_li


simManager.pc.barrier()
print("before runworker")
simManager.pc.runworker()
simManager.pc.done()
print("done")
```