**Support for more than one circuit?**

<https://sourceforge.net/p/electricdss/discussion/861977/thread/0138a1c9/>

[Paul](https://sourceforge.net/u/pm-rmi/profile/)- *2013-11-25*

Hi Roger and Wes,

Armed with the knowledge you both shared, I have started down another path to launching multiple instances of OpenDSS from Matlab. In an attempt to sidestep Python, I am trying to use a Windows batch file. My new approach is do call a batch file from Matlab that then initiates OpenDSS via the DLL. However, I think, to do so from the command-line, it seems that I need the entry-point function of OpenDSSengine.dll. According to MS, the command line for RunDLL (and I assume RunDLL32) is:

RUNDLL.EXE "dllname","entrypoint" "optional arguments"

Thus, the command so far is "RunDLL32.exe OpenDSSengine.dll,<entry-point function="">"

If this information can be shared, that would be great!

Thanks,

-Paul

**OpenDSS and parallel computing**

<https://sourceforge.net/p/electricdss/discussion/861976/thread/0f9191cc/>

[Roger Dugan](https://sourceforge.net/u/rdugan/profile/)- *2012-09-02*

Presently, OpenDSS is a uniprocessor app. I think you can run more than one instance of it, but we have learned that doesn't help as much as you would hope. Depends on the machine.

I don't know how you are implementing in Matlab, but sometimes OpenDSS is not the bottleneck. Depends on how much work you're depending on Matlab to do and whether or not you've compiled the Matlab code.  Strange as it may seem, some monte carlo runs using the COM interface actually run faster driven from Excel VBA, which is partially, if not mostly, compiled.  On a circuit of a few dozen buses, we were able to run 50,000 solutions in less than a minute. Again, it depends on what you're doing.

One of our engineers used Matlab to generate EV loadshapes by monte carlo methods and then simply wrote a script to have OpenDSS load in the annual loadshapes (took about 30 s) and execute them. On large circuits an annual simulation might take 10-20 min. We have exploited extra computers we had lying around for running these tasks. So this is parallel computing in essence. If you are doing something similar that requires annual simulations of (like a sequential monte carlo) see if you can do something similar. If you can force different OpenDSS processes into different CPUs, you might be able to achieve some gain. The problem may be how the operating system manages I/O from various process. Monitor objects are in memory, but Energymeters write to the harddisk may get collisions from different processes contending for the disk. I don't know.  You would think maybe a server could handle this problem ?  Try it and let us know how it works

**Parallelization of OpenDSS on Linux**

<https://sourceforge.net/p/electricdss/discussion/861976/thread/5703a79b3b/>

DSS SHARP DISCUSSION

**Parallel simulation in Python**

<https://sourceforge.net/p/electricdss/discussion/861976/thread/88e9706a7e/>

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2019-10-18*

Hello,

There must be something wrong with your script, check this example:

<https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/Examples/Parallel_Processing/Python/Example_1/Example_1.py>

Best regards

Davis

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2019-10-18*

Hello,

I've just added another python example for parallel processing. Download the example from here:

<https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/Examples/Parallel_Processing/Python/Example_2/>

It should be enough to cover your needs. Let me know how it works.

Best regards

Davis

**Parfor with Opendss**

<https://sourceforge.net/p/electricdss/discussion/861977/thread/9269e3c232/>

[A\_Y](https://sourceforge.net/u/shnarf/profile/)- *2022-01-25*

Hello,

I would like to perform parallel simulation of for loop (with very large amount of iterations), where each iteration of the for loop executes many snap shot solutions.

As I see it there are two options:

1. allocate different parts of the for loop to different CPUs similarly to what is done in: <https://sourceforge.net/p/electricdss/discussion/861977/thread/df51968038/?limit=25#2bff/cb35>
2. Doing parfor (parallel for loop). If I'm using parfor:  
   a. If I'm cloning CPUs as it is done in the opendss parallel scheme (as appears in the end of the message)  
   are the different snap shot solutions in each iteration of the parfor will be executed on different CPUs? (I'm not sure how to allocate different CPU with parfor. Will it automatically happen that different solve will operate on different CPUs once I have cloned them?)  
   b. If using single CPU, is it a problem to operate parallel loop which each iteration solves snap shots ?

I'm not sure which approach is better, and I guess it depends on the answers to my questions regarding the use of parfor with dss.

Thank you for your help!  
%%%%%%%%%%%%  
DSSText.Command = 'ClearAll' # Clears all instances of OpenDSS-PM  
DSSText.Command = 'Set Parallel=No' # Deactivates parallel processing  
DSSParallel = DSSCircuit.Parallel # Habdler for Parallel processing functions  
CPUs = DSSParallel.NumCPUs-1 # Gets how many CPUs this PC has  
DSSText.Command = 'Clone '+str(CPUs-2)  
DSSText.Command = 'Set Parallel=Yes'  
%%%%%%%%%%%

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2022-01-25*

Hello,

I'm afraid "parallel for" is not what you think (https://docs.microsoft.com/en-us/dotnet/api/system.threading.tasks.parallel.for?view=net-6.0). "Parallel for" is a modern programming structure that is in almost all programming languages nowadays and is intended to perform a large amount of iterations using threads, however, several conditions need to be met:

1. The variable(s) to iterate need to be part of the system base types.
2. The order in which the iterations are performed is not important and can be sorted after the process (if needed).

What this means is that the process invoking the "parallel for" will multi-thread a separated code segment for handling variables within the local scope. Each thread has "all CPU" affinity, which means that you don't know where each iteration is taking place, the OS takes car of that based on CPU workload.  
Why it doesn't work with OpenDSS? because OpenDSS is a process itself and creates threads internally. This means that for parallelizing multiple instances of OpenDSS, which is the worse way to go, you'll have to create a process for OpenDSS separately, increasing the number of threads used as a multiple of 2.  
Instead, using the parallel suite in OpenDSS the number of threads can increase linearly and the CPU control handled using OpenDSS commands. For more infor, please check this out: <https://docs.microsoft.com/en-us/windows/win32/procthread/processes-and-threads.>  
After that brief introduction to parallel computing (very brief), check the following example [https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/Examples/Parallel\_Processing/MATLAB/Example\_4/.](https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Version8/Distrib/Examples/Parallel_Processing/MATLAB/Example_4/) In this example there are several actors solved in time mode every 15 min, all of them are solved concurrently and the regulator on each is set at a different value. This is very close (for not saying the same) to what your are looking for.

Best regards

Davis

**Multiple instances of OpenDSS**

<https://sourceforge.net/p/electricdss/discussion/beginners/thread/1a55d342/>

[Grzegorz Kowalski](https://sourceforge.net/u/quebec313/profile/)- *2016-06-09*

Hi  
I'm trying to use multiple instances of OpenDSS, with direct DLL and under C++ environment. I made test app which loads OpenDSSDirect.dll, and uses DSSI and DSSPut\_Command functions to initialize OpenDSS and compile Run\_ckt5.dss file. Each instance of that application has own copy of OpenDSS and KLUSolve DLLS, and also own copy of ckt5 example simulation, so there's no way any instances messes around with another instance's files.

It works, BUT, when running multiple instances each instance is taking significantly more time to complete simulation (1 instance - ~48sec, 2 instances - ~68 sec, 4 instances - ~80 sec). PC has 8 cores, in task manager i can see that load is distributed across cores, and none has load above 50% (of course some are idling) and total load is below 25% when running 4 instances. CPU is definitely not the problem.

Each instance is separate process (not thread) and uses different copy of ckt5 simulation, so DLL's shouldn't use same data or synchronize in any way, but it looks like that's exactly what's happening here. Is there any mechanism implemented in OpenDSS that can cause that (like file mapping for example)? I've been looking through the source code but found nothing (nethertheless, I'm not delphi developer).

Thanks in advance for any help or hint

//edit: I probably should say what I'm trying to do. I want to run multiple different simulations simultaneously - for a multi-user environment. Each simulation solving different problem on different topology, not different time periods of same problem. I don't expect any improvement in performance of simulation, just want to run two or three simulations at the same time. I found few similiar threads on this forum, and after reading them I'm pretty sure that method described above should work.

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2016-06-09*

Hello,

The problem is clear: It looks like you are running both instances on the same thread, that's why if you create new instances the computing time gets increased. There are methods included on the Windows API to create new threads and handle each instance as an independent process. For example, in the case of delphi this is done with the class TThread.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms682516%28v=vs.85%29.aspx>

It is not enough to create new instances, if your programming language allows to execute everything on a single thread, the results will not be the expected ones, each instance needs its own thread to get real parallelism, otherwise, it is not going to work.

Best regards

Davis

[Grzegorz Kowalski](https://sourceforge.net/u/quebec313/profile/)- *2016-06-09*

I am running them on different threads! Even on different processes!  
Application that I made loads .DLL, starts OpenDSS and runs "compile ..." command. That's all it does (there's also some debug output, like time elapsed etc). I've launched from 1 to 8 instances of that **application**, each instance from separate directory, so there's separate .exe of that application, separate .dlls, separate DSS scripts. Each application starts one instance of OpenDSS. Multiple applications means multiple processes which means multiple threads. There's no way that multiple applications are running on the same thread. Sorry if that wasn't clear.

Inside of a directory:   
And i have 8 such directories:

Also, in one of applications that I made during last few days I even made separate thread for loading DLL, sending OpenDSS command etc. Again, i started from 1 to 8 instances of that application, and still there's big decrease in performance of each OpenDSS instance (so there were 8 applications, each with own process, and each process had two threads - one for starting one instance of OpenDSS, second one for UI (printf/scanf etc, right now it's just a console app)).

So we're talking about decrease in performance when several instances of application that is using OpenDSS are running simultaneously.

I thought that maybe all instances of application run on same core, but I clearly see that load is distributed between CPU cores (so there is some parallelization), and time required to finish simulation is almost exactly the same in every application instance. If I would run, for example, 3 applications at the same time and two of them would need 80 seconds to complete simulation, but third one only 60 seconds, that would probably mean that two of them were running on one core and third on another. But this is not the case - results are almost the same (to be exact: 80, 81, 81, 82 seconds for 4 instances) among app instances, and are almost the same during every run.

Would it help if I provide C++ code of my application?

Thank you for such quick response!

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2016-06-09*

Hello,

I don't think that is the right way to do it, because when the same application is opened several times (instances), not necessarily means that it will be assigned to a separate CPU. You have to allocate the thread using another mechanism, actually, we have done it before and as a result we have improved the performance as mentioned in:

<https://sourceforge.net/p/electricdss/discussion/861976/thread/4a9e4f69/>

If you don't allocate threads on different processors, no matter how many times you try to clone you application, all these copies will be executed using the same processor affecting negatively the processing time:

<http://programmers.stackexchange.com/questions/181157/how-to-program-thread-allocation-on-multicore-processors>

Check the examples on Julia, the code is open and that will give you an idea of what I'm mentioning here. The best solution is not to multiply things on disk and then try to load them, believe me, windows can easily recognize the same app replicated and to avoid crashing it will put them on a queue. I tried what you propose several years ago (like 5) so, my advice is, look for a thread allocation mechanism using your programming language and clone your instances within your program, copying the DLL on disk that many times is not the answer.

Best regards

Davis

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2016-06-09*

Hello,

Normally you experience this kind of issue when handling multiple threads due to the affinity of the thread, many people say that the processor affinity is handled automatically by the programming language, however, this is not completely true. Check the following info which can be useful, and if you are working with C++ this should help.

<https://msdn.microsoft.com/en-us/library/windows/desktop/ms686247%28v=vs.85%29.aspx>

Best regards

Davis

[Grzegorz Kowalski](https://sourceforge.net/u/quebec313/profile/)- *2016-06-10*

Davis,  
Thank you for that link, setting affinity correctly makes threads run on specific core (I definitely need to read about multithreading on windows). It looks like previously windows was assinging one thread to two cores (when one instance of app was running, there were two cores with about 50% load each, now there's 1 with 100% load - dunno if it's normal or is it because of intel's HyperThreading, but it doesn't matter).

Unfortunately running OpenDSS instances from separate threads is not enough in C++ on Windows. LoadLibrary function that is used to load DLL loads it only once. Subsequent calls to LoadLibrary gives same handle. Sending command to OpenDSS from two parallel threads in one process raises exception in OpenDSSDirect.dll. I had to make multiple processes (but, this time, each using the same .DLLs).

It works, but it still didn't solve issue with performance. Execution time is almost the same as previously.  
I guess that from the beginning overall performance is increased (one instance running ckt5 is ~50sec, two simulation one after another would be 100sec, two running in parallel is 70sec), but it takes more time to finish each simulation, which, in this case, is bad (user shouldn't have to wait longer for response for his small simulation only because another user is running really big simulation).

I understand that from your perspective everything is OK ( no explicitly used mutexes etc. to synchronise anything between multiple threads in OpenDSS code), so I have to read a bit more about how Windows handles DLLs and processes/threads memory, and try once more.

Again, thank you for your help!

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2016-06-14*

Hi man,

There is something about the solving stage, OpenDSS works with a second DLL called KLUSolve.dll, this is the one used to solve the system and it is thread safe, however, OpenDSS does not handles it using several instances. Probably that's where the bottle neck is, right now I'm adding some thread-safe structures to the memory handler in OpenDSS and modifying the KLUsolver calling sentence to create several instances considering the upper level caller, but this is going to take a while (is in the queue of my tasks).  
Let's try some other experiments like for example the one propsed here to see how your system behaves:

<https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Distrib/Doc/COM%20Speed%20Comparison.pdf>

Thanks for the energy you're putting on this.

Best regards

Davis

[Grzegorz Kowalski](https://sourceforge.net/u/quebec313/profile/)- *2016-06-29*

Right, I get it. COM still confuses me a bit. Thanks for explanation.  
So, right now any parallelization has to be done with multiple processes, both with DirectDLL and COM (at least that's the case for c++), and in the future it should be possible to create multiple instances in one process.

Well, I'm looking forward to test this feature when it will be ready ;)

**"Wait" command slowing down the script considerably!**

<https://sourceforge.net/p/electricdss/discussion/861976/thread/ee12653b02/>

[Mattia Secchi](https://sourceforge.net/u/smitchell91/profile/)- *2019-12-23*

Hello everyone, can anybody clarify if and when the "Wait" command is required in the parallel machine?

DSSText.Command='Set Parallel=Yes'*;*

DSSText.Command='set activeActor=1'*; % best practice (see paper)*

DSSObj.AllowForms=false*; % avoid showing the core progress window*

DSSText.Command='SolveAll'*;*

% % freeze the execution of the frontal panel actor **until** all the other

% % actors are available to start new processes

DSSText.Command='Wait'*;*

for example in the script above, the matlab profiler warns me that a considerable amount of time is spent in the "wait" line, see attached picture.

I was expecting the "SolveAll" command to be the most time consuming, but it seems like there is something else causing considerable delay.

Any clue?

Thanks!

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2019-12-23*

Hello,

I'm affraid your interpretation is wrong. When working in OpenDSS you have a minimum of 2 threads running: the caller interglface (exe, COM, DLL) and the actor's thread (the guy who does the simulation job). Depending in the numbers of actors you will increase the number of threads used. That being said, the interface also utilices CPU resources and is independent from the simualtion actors, so, if the parallel suite is active and the SolveAll command executed, the interface's thread (which is basically another actor) sends a message to all the other actors and then keeps its own execution.  
The wait command can be used to sync all the active actors with the interface. It forces the interface actor to wait until the other sctors are done before keep going with other taks. Depending on how and where your actors are allocated, the CPU utilization report will change.  
The interface thread is normallly assigned to CPU 0, however, its affinity is set for all processors so it will be the OS who will determine where to put it.  
The wait command is very useful for forcing sequential tasks when working in parallel, however, it can be done in many diffrent ways.  
Everything I just wrote is explained in the user's manual.

Best regards

Davis

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2021-01-22*

Hello,

There still a lot to learn:

1) each actor has its own thread and the interface to the actors (the one you use to communicate them from the external program and gives you control over COM/DLL) has a separate thread. So, for example, lets assume you have created 5 actors, that means that your application is actually using 6 threads: 5 actors + the interface. So, when creating the actors we always recommend to leave at least 2 threads free for the OS and the calling application, otherwise, the PC will freeze and many processes may collapse. It is very easy to do parallel processing in OpenDSS, but you need to know what you are doing.  
Normally, problems related to convergence in a parallel environment are related to the way the models are initialized on each actor, the best way to initialize them is sequentially and then go for full parallel. Another source of errors is when modifying the Y Bus matrix, that requires to be done sequentially, so we recommend to do a snap solution on the actor after a modification has been done (no parallel, just do the snap on the actor that changed), for later keep going with the parallel simulation. This doesn't happen always, but is a good way to control those problems.

2) Yes, why not? It all depends on what you are looking for. As mentioned above, we always recommend to leave at least 2 threads free for the OS and the calling interface (your program). In standard PC architectures a processor core can handle up to 2 threads concurrently, however, it comes with a cost, lets suppose you have a processor with nominal operating frequency of 3.10 GHz and max (turbo) 5 GHz, which is what hyper-threading is intended for. So, with one thread per core, the thread will execute at 3.1GHz, but when using 2 threads in the same core you'll use the turbo freq 5GHz.  
Nevertheless, it's 5GHz for 2 threads, which ideally means that each thread will execute at 2.5 GHz, going slower. Additionally, your processor is contained in a very limited space, having to deal with the heat generated while operating. The computer knows that, and if the temperature goes up drastically, it will reduce the speed to keep the core at good temperature, affecting the simulation performance. That's why the super computers and servers are so expensive and complex, they require a very fascinating cooling infrastructure to keep them operating at nominal values.  
<http://blog.logicalincrements.com/2019/10/cpu-what-are-threads/>

Best regards

Davis

Will a faster CPU make OpenDSS power flows faster?

<https://sourceforge.net/p/electricdss/discussion/861976/thread/94da38c6/>

[Andreas Procopiou](https://sourceforge.net/u/aprocopiou/profile/)- *2014-08-29*

Hello everyone,

I would like to ask if anyone knows if the OpenDSS power flow will be faster on a higher speed CPU.  
Currently I am using an Intel i5-3470 4-core @ 3.2GHz. I have a network with a very large number of nodes and it takes around 35 seconds for a single power flow simulation.

When doing multiple power flows I have to simulate thousands of times the same network and this results to days of simulations. Therefore saving time on simulations it is very important.

In line with the above, and considering that the OpenDSS is not a multicore software I would like to know if a higher speed processor (assuming the latest intel i7-4790K 4-core @ 4.0 - overclocked @4.6GHz) will make a signifigant difference to the simulation time. A signifigant difference for me will be at least 5-10 seconds.

I am looking forward to you comments.

Regards,  
Andreas

[Roger Dugan](https://sourceforge.net/u/rdugan/profile/)- *2014-08-29*

For each instance of OpenDSS, the circuit solution has to go through the sparse matrix solver. That takes a certain amount of time. There is also some setup through the Delphi code for each time step where the current injections have to be collected from each PCElement class device. So even if you completely eliminate the sparse solver time, there is always something left that is proportional to the problem size.

I think there are a couple of good choices for exploiting the "low-hanging fruit" in the OpenDSS techniques without going to some completely different solution technique:

1. Divide the circuit into a bunch of smaller circuits and assign a separate processor to each instance of OpenDSS. This is what Davis is doing with some success. Of course, you have to write some code for solving the problem of interconnections between the smaller circuits and have a platform that can manage parallel processes such as LabView, or some other custom-written system.
2. Divide the problem in time (or in the case of Monte Carlo, in specific instances) and assign a processor to each time segment. If the problem has several independent segments of time, such as a day or a week, you could assign each processor to solve for one day or one week, etc. Of course, this depends on the ability of the operating system to keep instances truly independent. Standard Windows does not seem to be particularly good at this when there is I/O occurring for each time step, but a large server might work. Several on this forum have promised to look into it, but have not got back with their results.

Parallel simulation

<https://sourceforge.net/p/electricdss/discussion/861977/thread/d2d73ed6/>

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2014-02-03*

Hi Santiago,

Well that's an expected behavior, remeber that it does not matter if you are multiplying the instances because you are working with the same DLL. To solve your problem there are two general solutions, the first one is to clone the OpenDSS instance (not to create new ones based in the same DLL) or create instances for copies of the same DLL (you only have to register one because the properties and method are the same in all the copies).

But to talk about real paralelism you have to solve more problems, because when you are working in a "regular" OS like Windows or a multitask OS where antivirus and other resident apps are running, you cannot take control of the cores, just the threads, and if you are working with threads in the same core basically you are working in a serial way.  
You can take control of the cores using embedded and Real-Time OS like we do in the DSSim-RT software, but right now we are distributing paralell simulations using multiple threads but in different cores, this development is going to be released in the version 1.2.0 of DSSim-PC (at the end of february).

I hope these ideas help you to solve your problem.

Regards

Davis

[santiago](https://sourceforge.net/u/krimen_sp/profile/)- *2014-02-03*

Well,  
I have tried cloning the class and it does not work, in fact I get the same result (refferences to the same instance)

I have asked in stack overflow about cloning COM, and the answers is that it is close to impossible (http://stackoverflow.com/questions/21524656/raise-several-instances-of-a-com-application?noredirect=1#comment32502325\_21524656)

So, I would like to know if someone has effectively invoked multiple instances of OpenDSS remotelly, and a brief explanation on how to achieve it. Meanwhile I'll keep on triying.

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2014-02-03*

Hi Santiago,

As I said before, we made it, but to do it we have to create multiple copies of the DLL so we can create an instance for each DLL by separately.

Right now we are capable to run even 10 instances of OpenDSSEngine in 10 different cores (distributed), of course over an embedded OS.

Good luck with that.

Regards

Davis

[santiago](https://sourceforge.net/u/krimen_sp/profile/)- *2014-02-03*

Dear Davis, thank you for your replies.

Let me see if I get this straight; you are saying that I go to C:\Program files\OpenDSS\x64, and make 10 copies of "OpenDSSengine.dll" and refference them separatelly into my App?

Or shall I install OpenDSS 10 times in different locations?

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2014-02-03*

Hi Santiago,

Well, in our case we are woking with a modified version of OpenDSSEngine, so we create multiple copies of the same DLL but changing little things like the name (adding a char and others). Those DLLs can be in the same PC or distributed in different machines like in the case of more than 4 cores.  
Another tip:"Have you notice that the OpenDSS executable does not need to register the DLL to run?"

I hope it helps

Regards

Davis

[Roger Dugan](https://sourceforge.net/u/rdugan/profile/)- *2014-02-06*

An interesting discussion thread! Maybe I can learn something from what you are doing.

Yes, OpenDSS.EXE is a stand-alone compiled version of the simulator and does not require the COM server. We put a main program and a few forms on top of the simulator code and compiled and linked it all together into one exe that you do not need to register.

We have run some tests where we have fired off different threads of the EXE version using Python to do it. Seemed to work, but we didn't get much performance improvement.

As for creating multiple copies of the COM server, I think you may need to acquire a Delphi XE2, or newer, compiler and create a number of different versions of OpenDSSEngine.DLL with different GUIDs at least for the DSS interface. The DSS interface is the only one registered and that's how Windows finds the server. I don't know if you need to create new GUIDs for the other items in the Type Library that have GUIDs. The other interfaces are all Internal and are created when the server is created. Maybe someone out there knows the answer.

I will be interested to know if anyone out there has done this.

We are hoping to run a parallel test some time this year, but it will be multiple processes of the EXE, each solving a different time period -- such as one week at 1-s intervals.

I may have said this before, but most of the performance complaints I have received from Matlab users have been because of looping in Matlab m files, not OpenDSS. Unless you have a REALLY BIG circuit model, I wouldn't spend much time trying to parallelize your monte carlo solution until you identify where the bottleneck is. We've done some monte carlo studies with ~50-bus 3-phase systems driven from Excel VBA that did 50,000 cases in 15-20 s on a standard-issue laptop. Of course, it depends on how many variables you are varying and what kind of solution you're attempting for each case. But just a thought.

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2018-09-19*

Hi,

You can do that with version 8, check the following links:

<https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Parallel_Version/Distrib/Doc/User_Instructions_for_Parallel_Processing.pdf>

<https://sourceforge.net/p/electricdss/code/HEAD/tree/trunk/Parallel_Version/Distrib/Examples/Parallel_Processing/>

Best regards

Davis

[Ying Xu](https://sourceforge.net/u/dahei/profile/)- *2018-10-05*

Hi Davis,  
I found the maxim number of actor is CPUcount. I mean the number of cpus is not the limit of threads, right?

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2018-10-05*

Hi,

In a computer system you can create as many threads as you want, however, in standard desktop architectures you can run only 2 at the same time per core, other architectures allow only 1 thread per core, it all depends on the computer type and use. It is then when concepts like “hyper threading” become important to guarantee the performance of your app. In OpenDSS, we are aware that you want to run your circuits simultaneously, for that reason we are limiting the number of actors to the number of threads (CPUs) in your PC, if you have 8 CPUs in your PC it means that you probably have only 4 cores (2 concurrent threads per core). Nevertheless, 2 threads running in the same core means that you are splitting the processing capability of that core for processing both threads, even with “hyper threading”, you will see the difference between handling 1 or 2 threads per core.

Best regards

Davis

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2018-10-16*

Hi Xu,

Well, your script has several mistakes considering that is running in parallel (you cannot keep thinking sequentially). When working in parallel (set parallel=yes) if you start a simulation task you need to make sure that is finished before sending a new simulation task, otherwise all the data will collapse, please take that into consideration the next time (remember: everything is happening at the same time). I've changed your script as follows:

ClearAll

set parallel=no

! Edit the path name to indicate the correct location of the Master file.

set cpu = 3

Compile (C:\Users\MGMS\Desktop\v7tov8\pm\_8500\8500-Node\_1\master.dss)

NewActor

set cpu = 2

Compile (C:\Users\MGMS\Desktop\v7tov8\pm\_8500\8500-Node\_2\master.dss)

set parallel=yes

!!!! Sets the properties for all the actors

set ActiveActor = \*

Set mode=dynamics number=200 h=0.01

!!!!!! run a few steps for everybody!!!!!!!!!!!!!!!!!!!!

SolveAll

!!!!! Waits until is done

Wait

!!!!! set disturbance in actor 1 !!!!!

set ActiveActor = 1

Redirect (C:\Users\MGMS\Desktop\v7tov8\pm\_8500\8500-Node\_1\dist\_setting.dss)

set ActiveActor = 2

!!!!! set disturbance in actor 2 !!!!!

Redirect (C:\Users\MGMS\Desktop\v7tov8\pm\_8500\8500-Node\_2\dist\_setting.dss)

!!!!! sets the number for all the active actors and solves !!!!

Set ActiveActor = \*

set number=6000

SolveAll

wait

set parallel=No

set ActiveActor = 2

!!!!!!!! plot !!!!!!!!!

Redirect (C:\Users\MGMS\Desktop\v7tov8\pm\_8500\8500-Node\_2\plot\_curves.dss)

set ActiveActor = 1

!!!!!!!! plot !!!!!!!!!

Redirect (C:\Users\MGMS\Desktop\v7tov8\pm\_8500\8500-Node\_1\plot\_curves.dss)

!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

You have to be careful when working in parallel, otherwise, just disable the feature to work sequentially.

Best regards

Davis

Question about parallel machine computing and general speedup!

<https://sourceforge.net/p/electricdss/discussion/861976/thread/f0c33090df/>

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2019-11-01*

Hello,

That's partially true, I mean, yes, you can load them at the beginning of the program to have them in memory for when you need them, but you can also create them on the fly, just define the new loadshape before you link it to the object is going to use it. For example:

DSSText.Command='New LoadShape.ABC ...';

DSSText.Command='PV.XYZ.daily=ABC';

There are several ways to do the same thing, just try to use the one that works the best for you.

Best regards

Davis

[Mattia Secchi](https://sourceforge.net/u/smitchell91/profile/)- *2019-11-01*

A minimal working example in Matlab:

[~,DSSText,DSSCircuit,~]=DSSStartup*;*

DSSText.Command='ClearAll'*; % Clears all instances of OpenDSS-PM*

DSSText.Command='Set Parallel=No'*; % Parallel Suite off*

% create shapefiles

**for** m=1:size(loads,2)

DSSText.Command=['New LoadShape.Shape',num2str(m),' npts=52560 mult=[',num2str(loadsP(:,m)'),'] UseActual=No'];

**end**

disp ('Compiling master on core #1')

DSSText.Command=['Compile "',source,'\input\_data\906BUS\Master1.dss"']*;*

DSSText.Command='set CPU=0'*;*

**for** n=2:cores

DSSText.Command='NewActor'*; % ACTOR NUMBER=CPU NUMBER+1*

DSSText.Command=['Compile "',source,'\input\_data\906BUS\Master',num2str(n),'.dss"']*;*

DSSText.Command=['set CPU=',num2str(n-1)]*;*

**end**

**end**

Now, what happens is that the compilation fails at actor 2 because the loadshapes are created inside actor 1, thus being inaccessible to the other actors.

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2019-11-11*

Hello,

Yes, that's one of the things you need to consider when working with parallel processing, and is that the hardware resources are limited. The actors created in OpenDSS run with the highest priority in the core, which means that no other thread can be executed in the dedicated thread of the processor. for that reason we alwayes recommend to leave 1 or 2 threads (CPUs) free for handling the graphics, OS tasks and others.  
The error you are experiencing is because probably you are asking for the simulation progress in MATLAB continuosly, which will kill the progress form (is a separate applciation called by OpenDSS) because the remaining thread is focused on MATLAB. The best you can do is while asking for the progress in MATLAB (in a lopp I guess) add a delay, 100-200 ms will be enough. As a result, the processor will have space for executing the progress form thread and both apps will look like running together.  
Because is only to show the progress, it's not critical to add delays.

Best regards

Davis

[Mattia Secchi](https://sourceforge.net/u/smitchell91/profile/)- *2019-11-11*

The script is running on 15 virtual cores out of 16, so that should be ok.

I am actually not asking for the progress to be visualised, what I do is, after defining the cores to use for each simulation:

DSSText.Command='Set Parallel=Yes'*;*

DSSText.Command='set activeActor=1'*; % best practice (see paper)*

DSSText.Command='SolveAll'*;*

DSSText.Command='Wait'*;*

so I am not really sure why the window is popping up.

I thought you needed a

DSSText.Command='Get ActorProgress'

but it seems to be showing it anyways.

[Davis Montenegro](https://sourceforge.net/u/davismont/profile/)- *2019-11-11*

Hello,

The solution is simple, use the allowforms to allow/prevent the form to show up. Like this;

DSSobj.AllowForms=**false**;

To bring it back, turn it ON again. Get actorprogress is to know the actor progress per actor in percentage.

Best regards

Davis