Table of Contents

Job Manager for local execution of ATK scripts

Version: 2015.2

Downloads & Links

- PDF version
- silicon.py
- cnt.py
- mpi_check.py

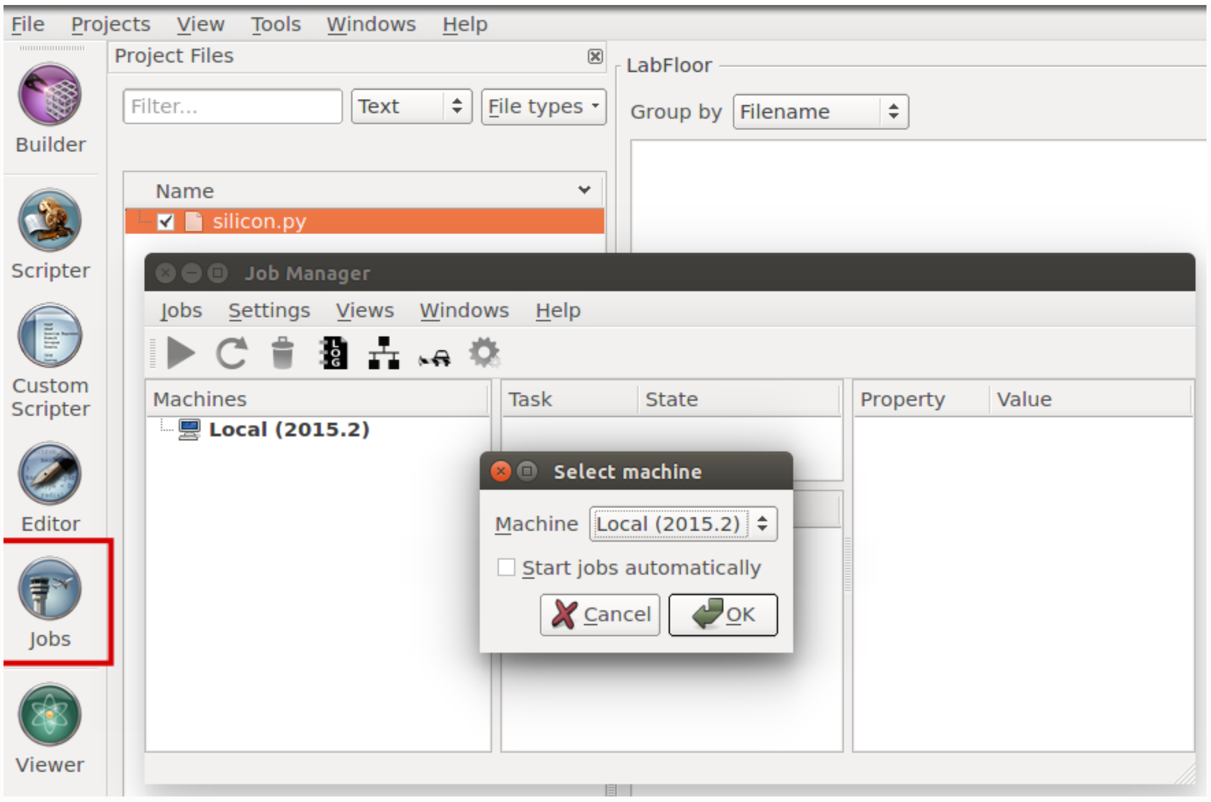
In this tutorial you will learn how to use the **Job Manager** for local execution of ATK scripts. Specifically, you will learn about:

1. queuing, running and managing ATK jobs;
2. local execution in serial;
3. local execution in parallel using threading;
4. local execution in parallel using MPI;
5. managing local machines.

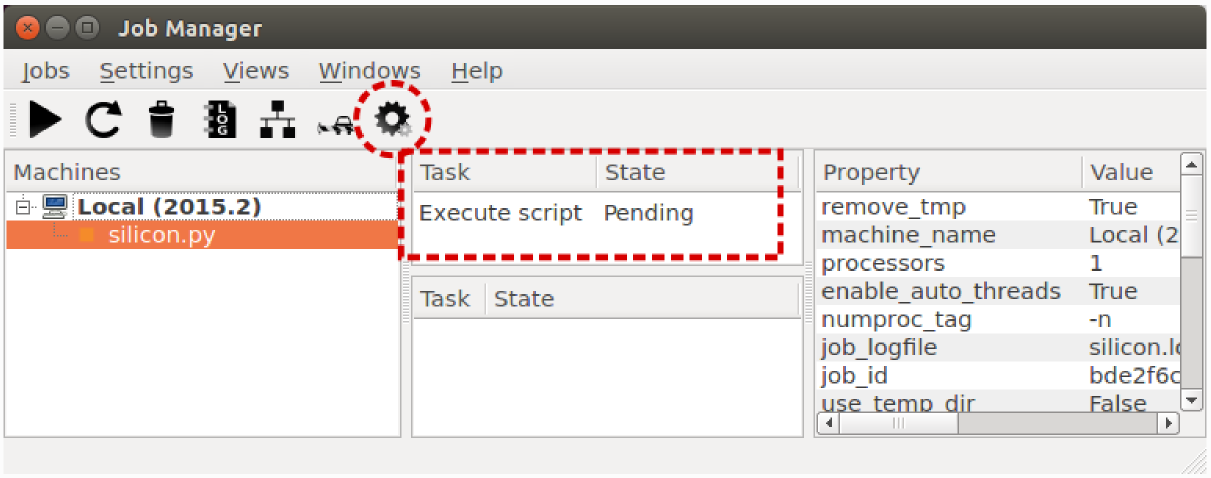
Serial execution

Create a new empty project and download the example script silicon.py, which runs an ATK-DFT calculation with very many k-points (31x31x31).

Drop it on the **Job Manager** and select a local machine for the job execution.



The job is now in the task state “pending”. Click the **Job Settings** ⚙ icon to edit the job settings.



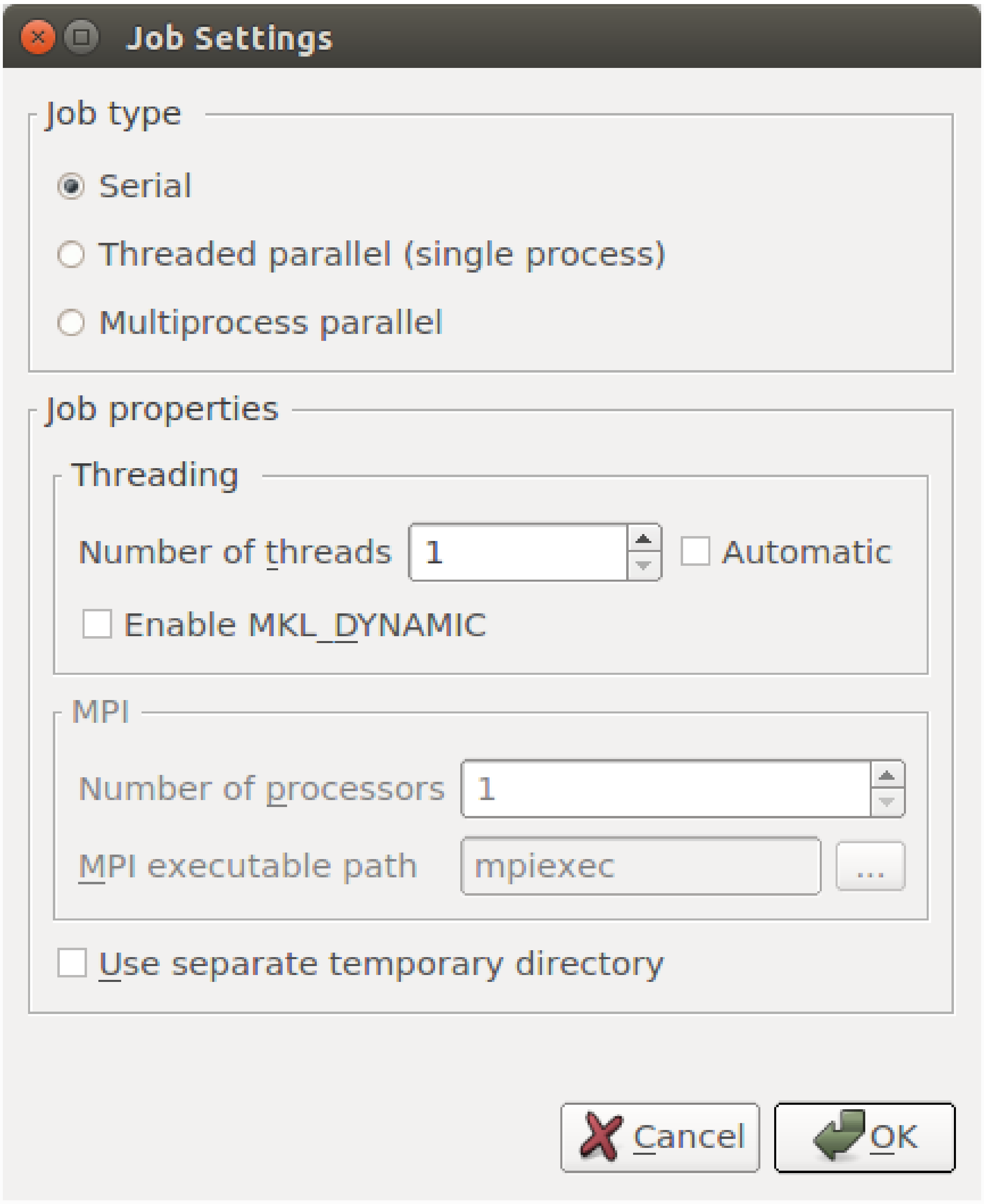
Task	State
Execute script	Pending

Property	Value
remove_tmp	True
machine_name	Local (2
processors	1
enable_auto_threads	True
numproc_tag	-n
job_logfile	silicon.l
job_id	bde2f6c
use_temp_dir	False

The **Job Settings** widget has three basic panels:

- Job type;
- Job properties;
- MPI settings;

Select a **Serial** job type as shown in the below figure, and note that threading is turned off (number of threads is 1), and MPI parallelization is not available.


Job Settings

Job type

- (●) Serial
- ( ) Threaded parallel (single process)
- ( ) Multiprocess parallel

Job properties

Threading

Number of threads: 1 [ ] Automatic

[ ] Enable MKL_DYNAMIC

MPI

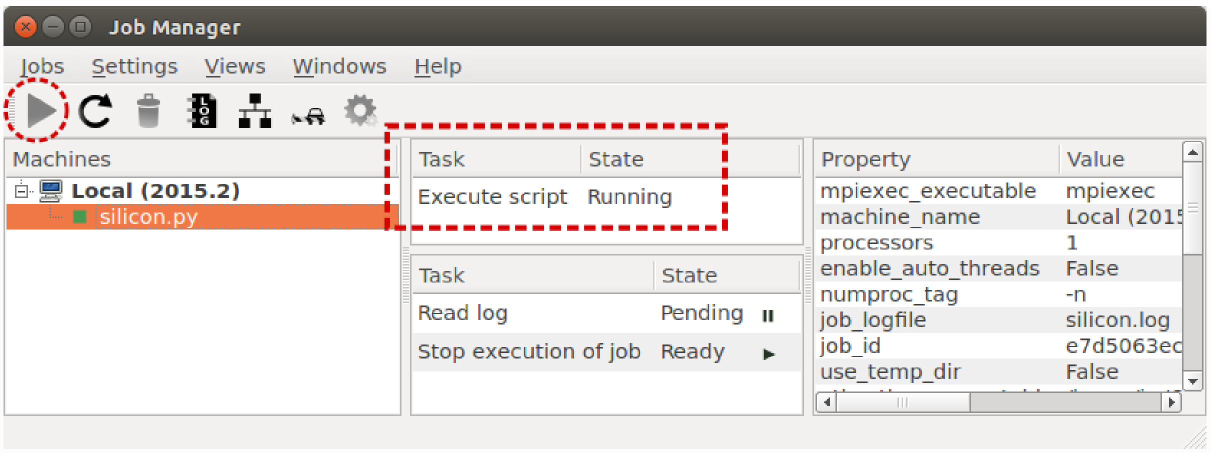
Number of processors: 1

MPI executable path: mpiexec

[ ] Use separate temporary directory

Click *OK* to approve the job settings (completely serial – no threading and no MPI).

Back in the **Job Manager**, click the **Run** ▶ icon to start the job. The task state changes from "Pending" to "Running".



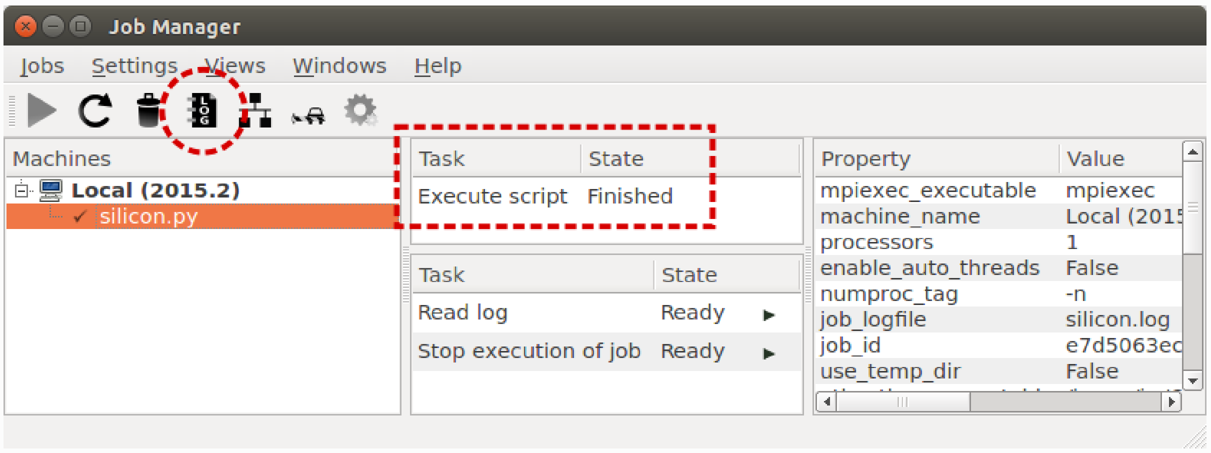
Machines
- Local (2015.2)
  - silicon.py

Task	State
Execute script	Running

Task	State
Read log	Pending
Stop execution of job	Ready

Property	Value
mpiexec_executable	mpiexec
machine_name	Local (2015
processors	1
enable_auto_threads	False
numproc_tag	-n
job_logfile	silicon.log
job_id	e7d5063ec
use_temp_dir	False

The job finishes after ca. 1 minute (2.5 GHz CPU). Note that the task state changes to “Finished”. You can inspect the job log file by clicking the **LOG** icon:

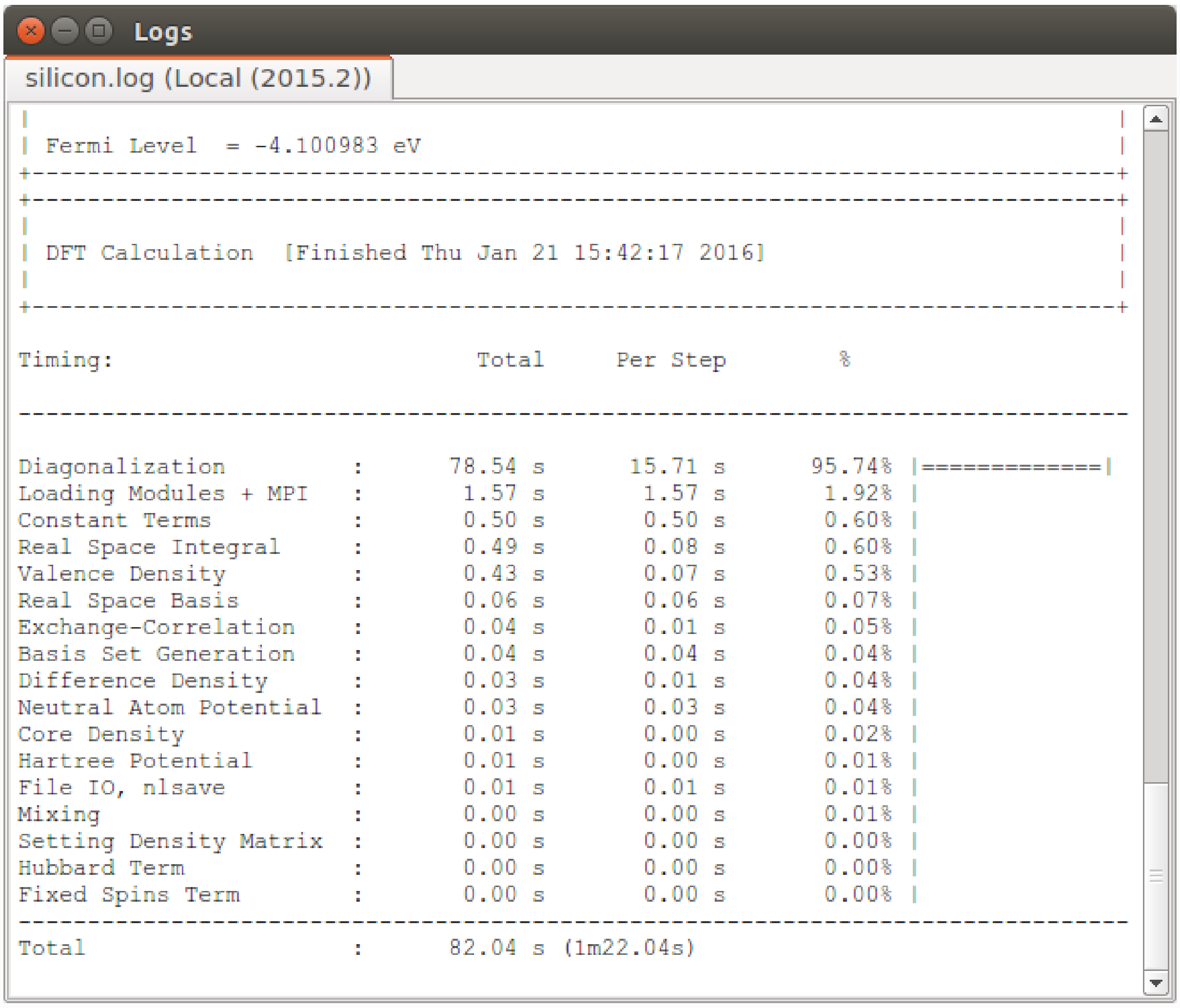


Machines
- Local (2015.2)
  - ✓ silicon.py

Task	State
Execute script	Finished

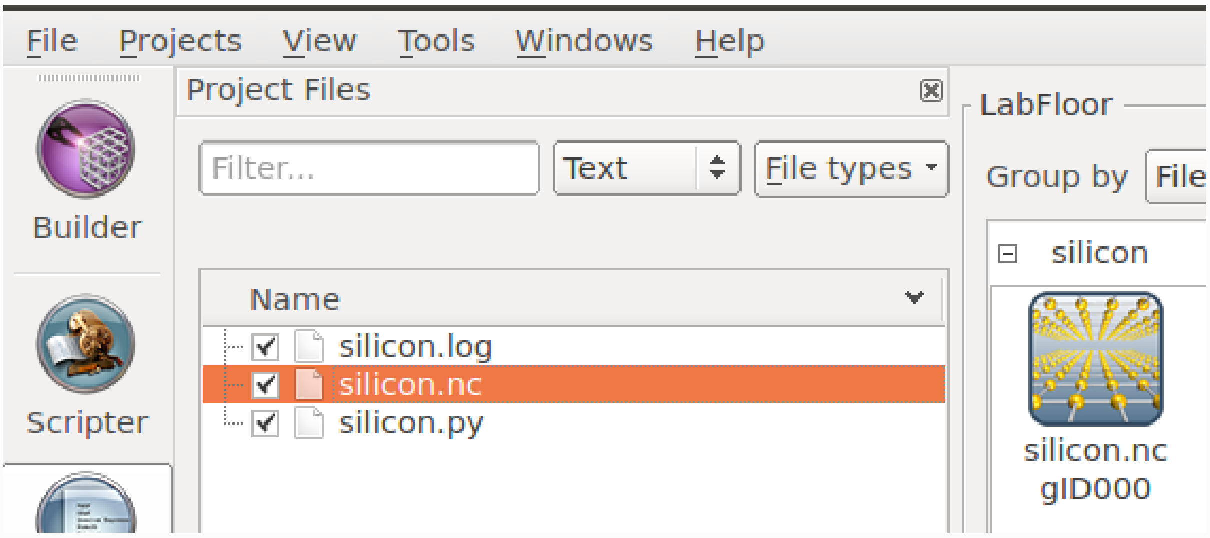
Task	State
Read log	Ready
Stop execution of job	Ready

Property	Value
mpiexec_executable	mpiexec
machine_name	Local (2015
processors	1
enable_auto_threads	False
numproc_tag	-n
job_logfile	silicon.log
job_id	e7d5063ec
use_temp_dir	False



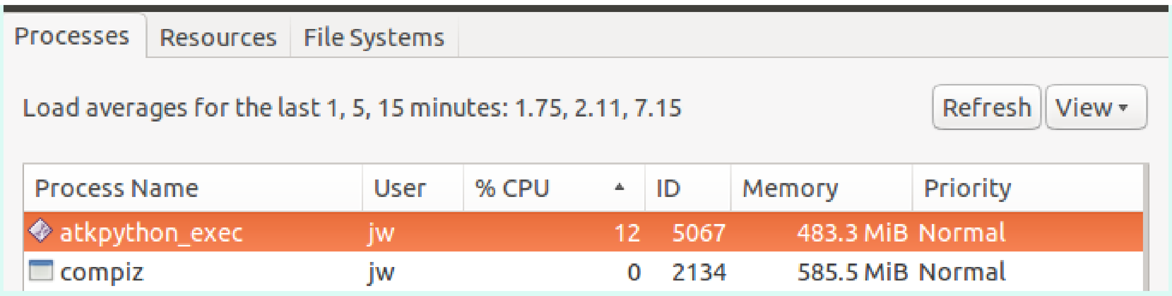
```
|                                                                              |
| Fermi Level  = -4.100983 eV                                                  |
+------------------------------------------------------------------------------+
+------------------------------------------------------------------------------+
|                                                                              |
| DFT Calculation  [Finished Thu Jan 21 15:42:17 2016]                         |
|                                                                              |
+------------------------------------------------------------------------------+

Timing:                          Total     Per Step        %

--------------------------------------------------------------------------------

Diagonalization           :      78.54 s      15.71 s      95.74% |=============|
Loading Modules + MPI     :       1.57 s       1.57 s       1.92% |
Constant Terms            :       0.50 s       0.50 s       0.60% |
Real Space Integral       :       0.49 s       0.08 s       0.60% |
Valence Density           :       0.43 s       0.07 s       0.53% |
Real Space Basis          :       0.06 s       0.06 s       0.07% |
Exchange-Correlation      :       0.04 s       0.01 s       0.05% |
Basis Set Generation      :       0.04 s       0.04 s       0.04% |
Difference Density        :       0.03 s       0.01 s       0.04% |
Neutral Atom Potential    :       0.03 s       0.03 s       0.04% |
Core Density              :       0.01 s       0.00 s       0.02% |
Hartree Potential         :       0.01 s       0.00 s       0.01% |
File IO, nlsave           :       0.01 s       0.01 s       0.01% |
Mixing                    :       0.00 s       0.00 s       0.01% |
Setting Density Matrix    :       0.00 s       0.00 s       0.00% |
Hubbard Term              :       0.00 s       0.00 s       0.00% |
Fixed Spins Term          :       0.00 s       0.00 s       0.00% |
--------------------------------------------------------------------------------
Total                     :      82.04 s (1m22.04s)
```

The job output of course appears on the VNL **LabFloor** after job execution.



Tip

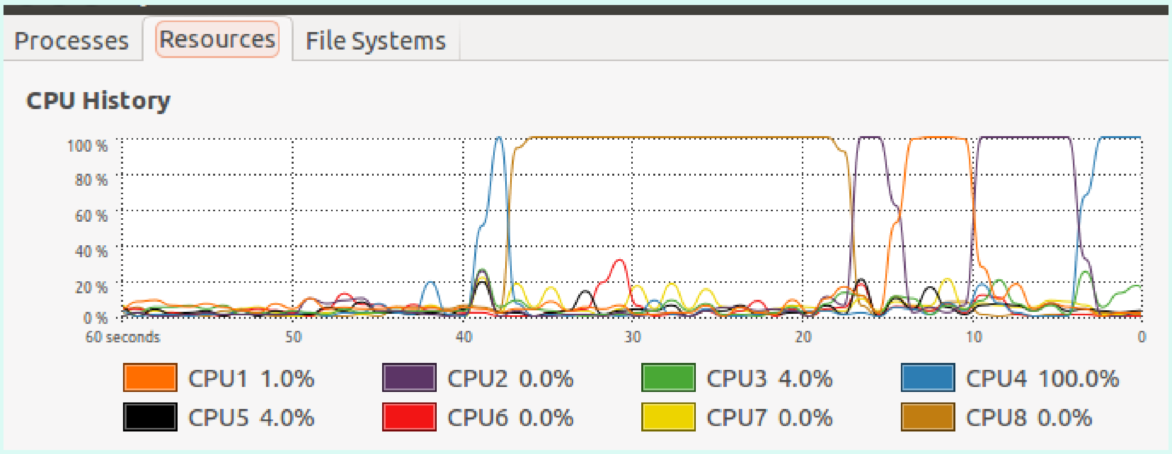
If you check the system load during local execution in serial, you should see that the serial job launches only a single computing task on a single CPU core.



Load averages for the last 1, 5, 15 minutes: 1.75, 2.11, 7.15

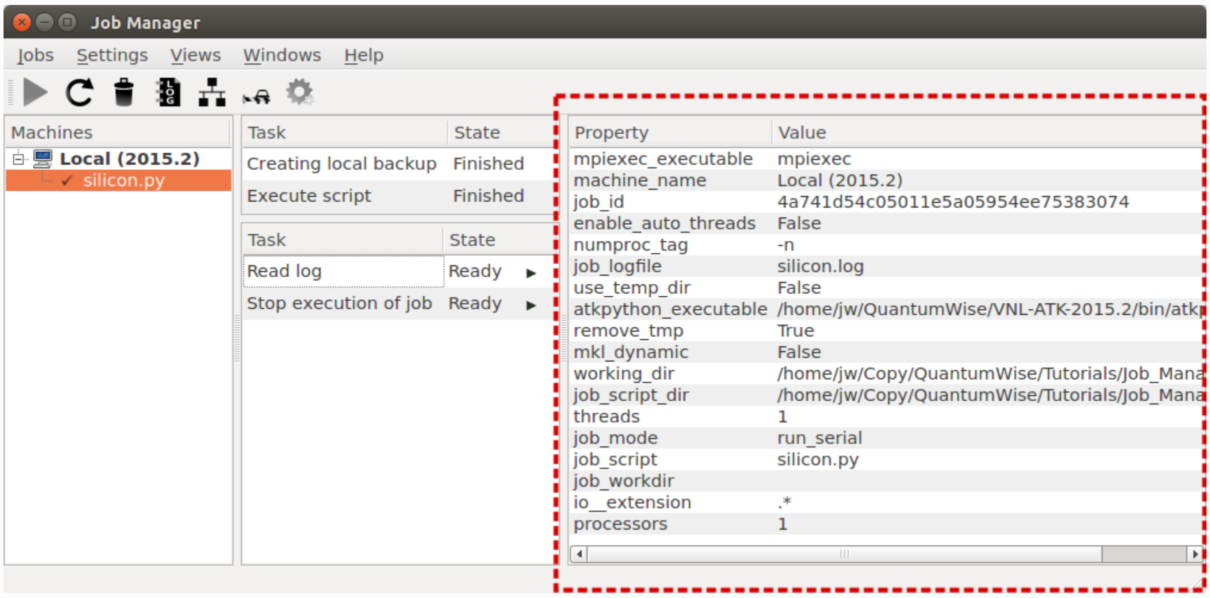
Process Name	User	% CPU	ID	Memory	Priority
atkpython_exec	jw	12	5067	483.3 MiB	Normal
compiz	jw	0	2134	585.5 MiB	Normal

Only one core is used at a time, but the hardware process manager may move the task between cores from time to time.



Back in the **Job Manager**, the Property–Value list shows all details of the settings used for job execution, including

- path to the ATK executable;
- name of the Python script and the log file.
- the job mode is serial, so the number of processors used is 1, and the number of thrads is also just 1.

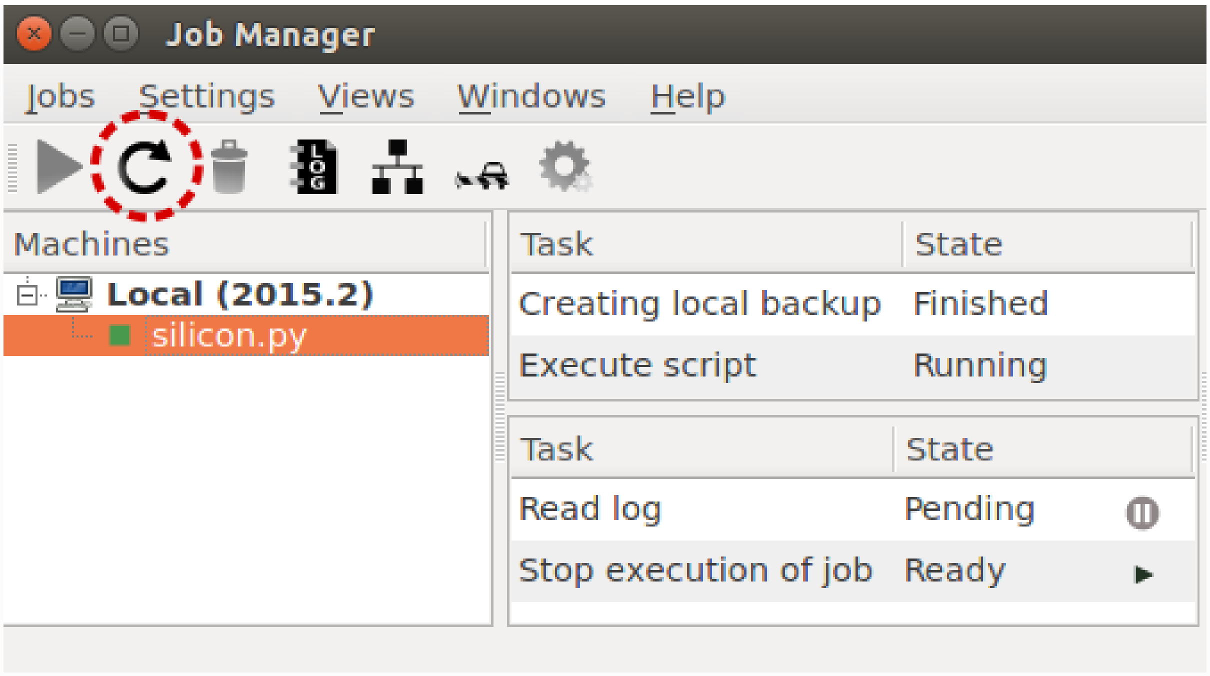


Task	State
Creating local backup	Finished
Execute script	Finished

Task	State
Read log	Ready ▸
Stop execution of job	Ready ▸

Property	Value
mpiexec_executable	mpiexec
machine_name	Local (2015.2)
job_id	4a741d54c05011e5a05954ee75383074
enable_auto_threads	False
numproc_tag	-n
job_logfile	silicon.log
use_temp_dir	False
atkpython_executable	/home/jw/QuantumWise/VNL-ATK-2015.2/bin/atk
remove_tmp	True
mkl_dynamic	False
working_dir	/home/jw/Copy/QuantumWise/Tutorials/Job_Mana
job_script_dir	/home/jw/Copy/QuantumWise/Tutorials/Job_Mana
threads	1
job_mode	run_serial
job_script	silicon.py
job_workdir	
io__extension	.*
processors	1

You can use the **Resubmit** ↻ icon to resubmit a script. Note that any changes that have been made to the script will be picked up by the new job.



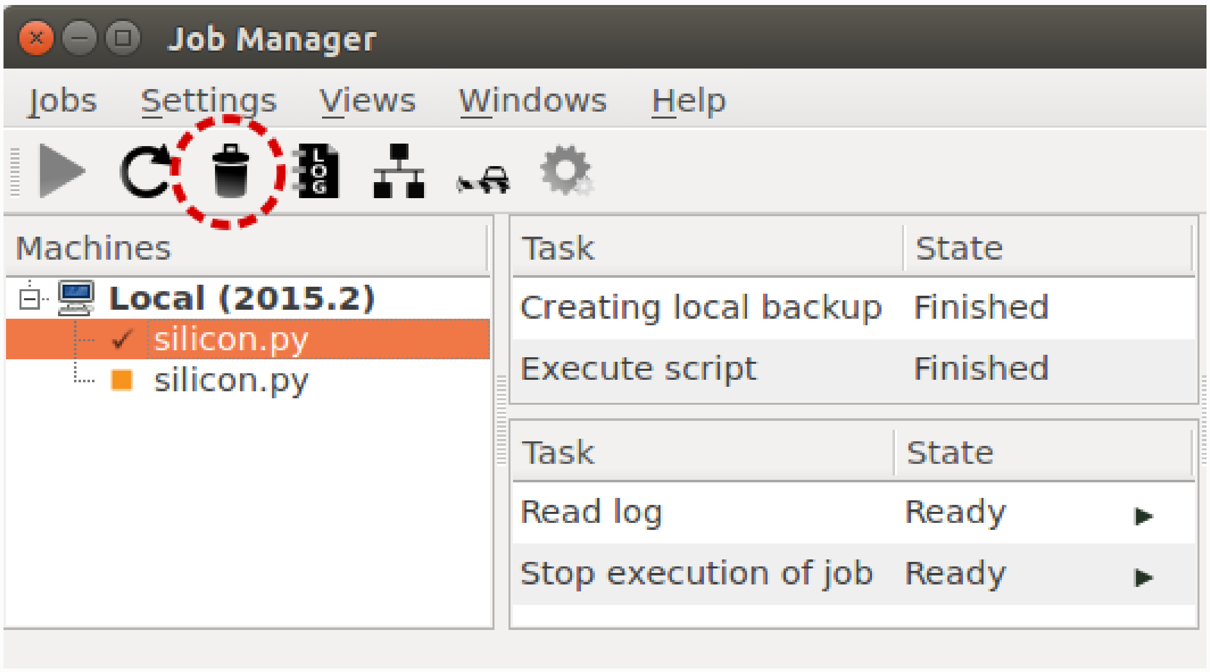
Task	State
Creating local backup	Finished
Execute script	Running

Task	State
Read log	Pending
Stop execution of job	Ready

Note

Remember that the default job type is “Threaded”. You can change this to “Serial” or “Multiprocess parallel” before starting the job.

Use the **Trash** icon to remove jobs from the job queue.



Task	State
Creating local backup	Finished
Execute script	Finished

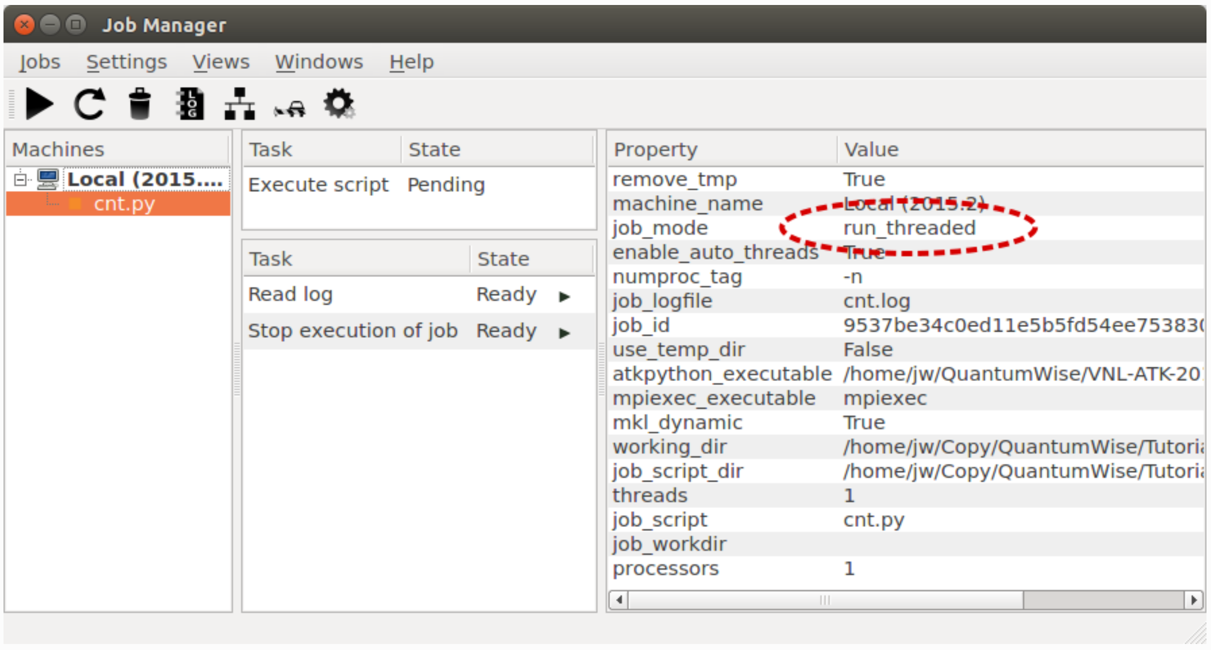
Task	State
Read log	Ready
Stop execution of job	Ready

Threading

Threading is one way to parallelize a computational job. ATK uses Intel MKL for openMP threading. Note that we do in general recommend MPI parallelization over threading for parallelizing DFT calculations. However, threading is often more efficient for parallelizing ATK-Classical calculations.

Download the script cnt.py, which uses ATK-Classical to calculate the dynamical matrix of a multiwall carbon nanotube.

Execute it using the **Job Manager**, and choose job type “Threaded parallel (single process)”. It should be pretty fast.



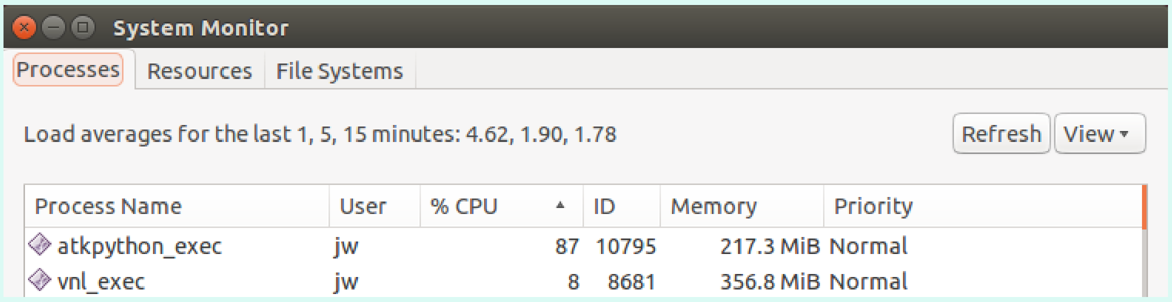
Task	State
Execute script	Pending

Task	State
Read log	Ready
Stop execution of job	Ready

Property	Value
remove_tmp	True
machine_name	Local (2015.2)
job_mode	run_threaded
enable_auto_threads	True
numproc_tag	-n
job_logfile	cnt.log
job_id	9537be34c0ed11e5b5fd54ee75383
use_temp_dir	False
atkpython_executable	/home/jw/QuantumWise/VNL-ATK-20
mpiexec_executable	mpiexec
mkl_dynamic	True
working_dir	/home/jw/Copy/QuantumWise/Tutori
job_script_dir	/home/jw/Copy/QuantumWise/Tutori
threads	1
job_script	cnt.py
job_workdir	
processors	1

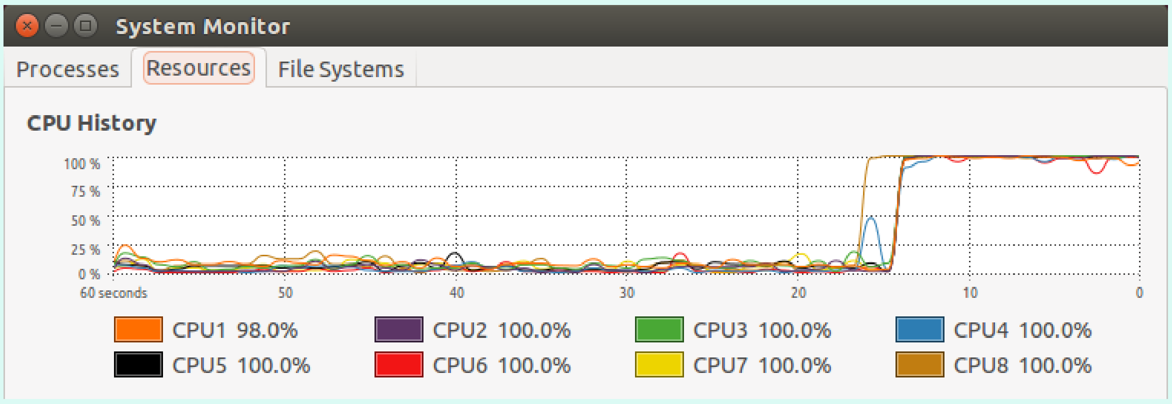
Tip

If you check the system load during execution of the calculation, you should see that only a single **atkpython** process is started, even though several cores appear to be busy. This is because the work load of the one process is split into a number of threads that may be distributed on more cores.



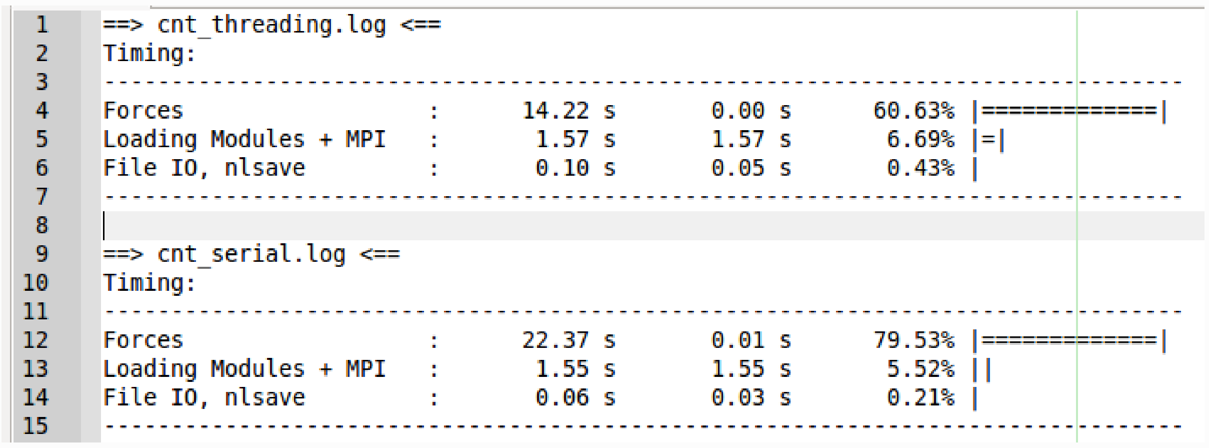
Load averages for the last 1, 5, 15 minutes: 4.62, 1.90, 1.78

Process Name	User	% CPU	ID	Memory	Priority
atkpython_exec	jw	87	10795	217.3 MiB	Normal
vnl_exec	jw	8	8681	356.8 MiB	Normal



If you also run the calculation in serial, you will see that the wall-clock time used for evaluating ATK-

Classical forces may decrease significantly when threading is switched on. In the example shown below, the time spent on force calculations is roughly halved.



```
==> cnt_threading.log <==
Timing:
--------------------------------------------------------------------------------
Forces                     :      14.22 s       0.00 s      60.63% |=============|
Loading Modules + MPI      :       1.57 s       1.57 s       6.69% |=|
File IO, nlsave            :       0.10 s       0.05 s       0.43% |
--------------------------------------------------------------------------------

==> cnt_serial.log <==
Timing:
--------------------------------------------------------------------------------
Forces                     :      22.37 s       0.01 s      79.53% |=============|
Loading Modules + MPI      :       1.55 s       1.55 s       5.52% ||
File IO, nlsave            :       0.06 s       0.03 s       0.21% |
--------------------------------------------------------------------------------
```

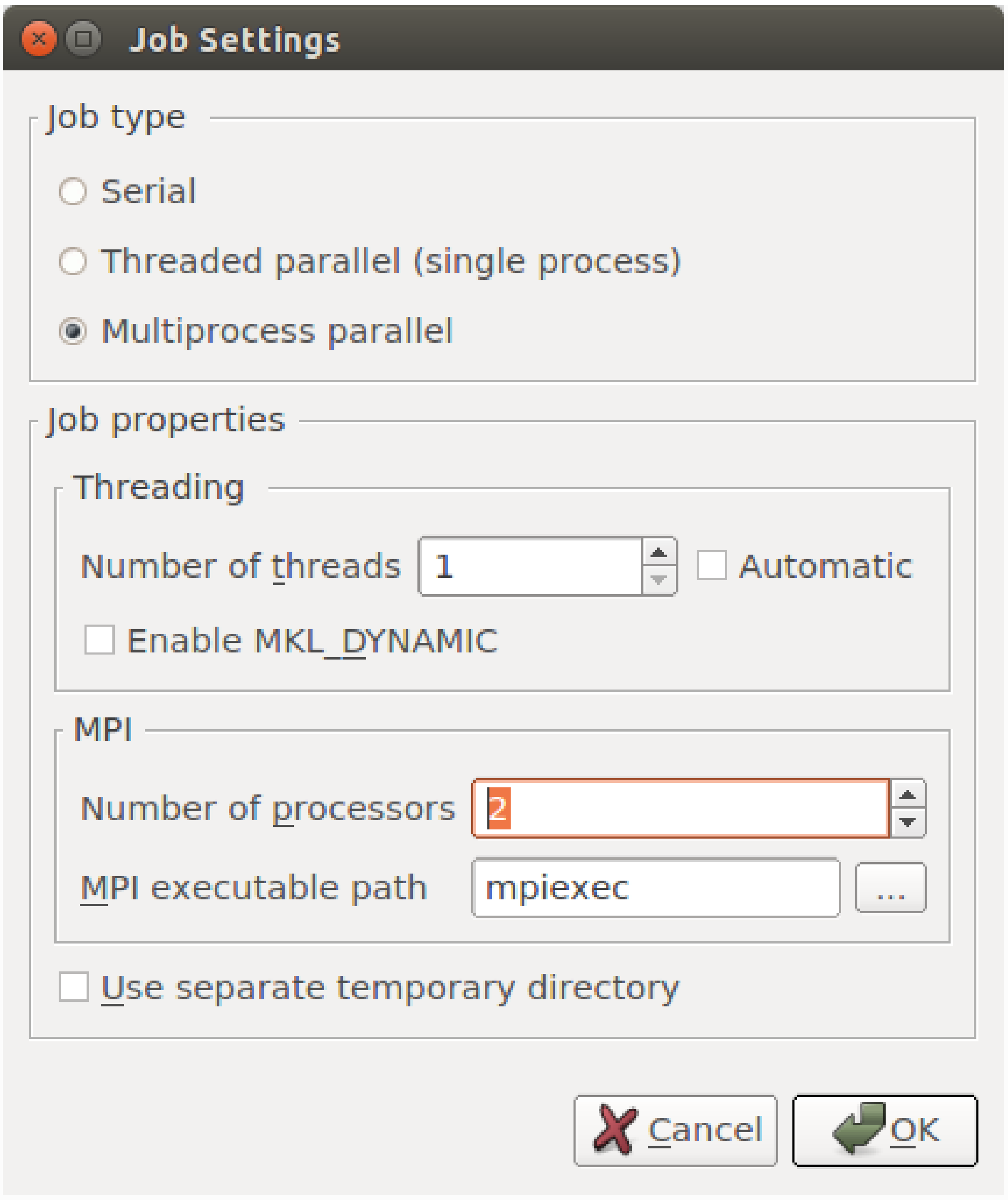
MPI parallelization

This section requires you have MPI installed on your local machine. If not, please check out the tutorial MPI setup for running ATK in parallel.

Before proceeding, you should test that MPI works properly. Download the test script mpi_check.py, run it in parallel on a few cores, and check that the expected number of MPI processes report back:

```
$ mpiexec -n 4 atkpython mpi_check.py
+----------------------------------------+
|                                        |
| Atomistix ToolKit 2015.2               |
|                                        |
+----------------------------------------+
MPI process 1 of 4 reporting.
MPI process 2 of 4 reporting.
MPI process 3 of 4 reporting.
MPI process 4 of 4 reporting.
```

Then use the **Job Manager** to execute the script `silicon.py` in MPI parallel: In **Job Settings** choose *Multiprocess parallel* and e.g. 2 MPI processes.



Job type

- Serial
- Threaded parallel (single process)
- [✓ Multiprocess parallel]

Job properties

Threading

Number of threads: 1 — Automatic ☐

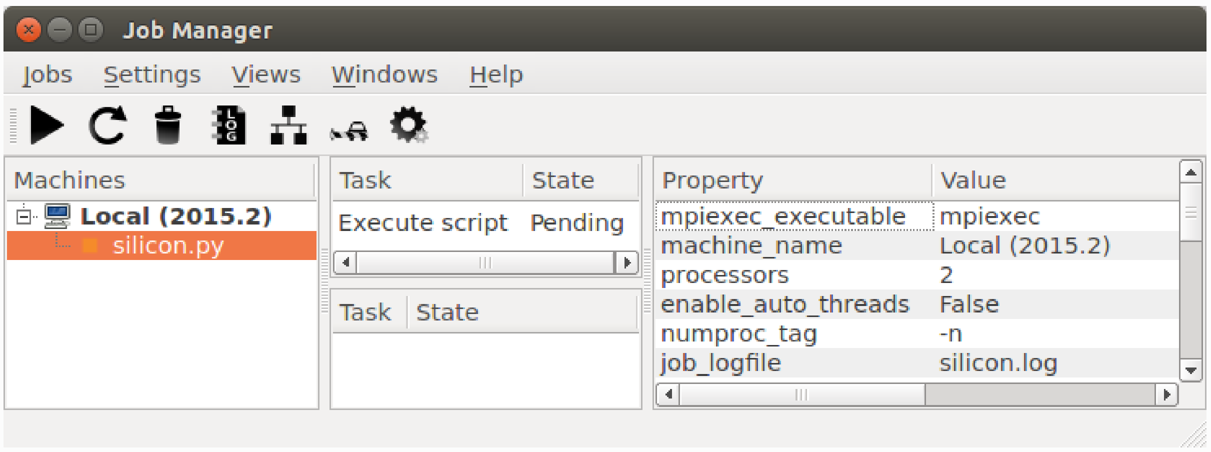
☐ Enable MKL_DYNAMIC

MPI

Number of processors: 2

MPI executable path: mpiexec

☐ Use separate temporary directory



Machines
Local (2015.2)
silicon.py

Task	State
Execute script	Pending

Task	State

Property	Value
mpiexec_executable	mpiexec
machine_name	Local (2015.2)
processors	2
enable_auto_threads	False
numproc_tag	-n
job_logfile	silicon.log

Fig. 22 The Property–Value list shows the name of the MPI executable and that 2 processors are used for MPI.

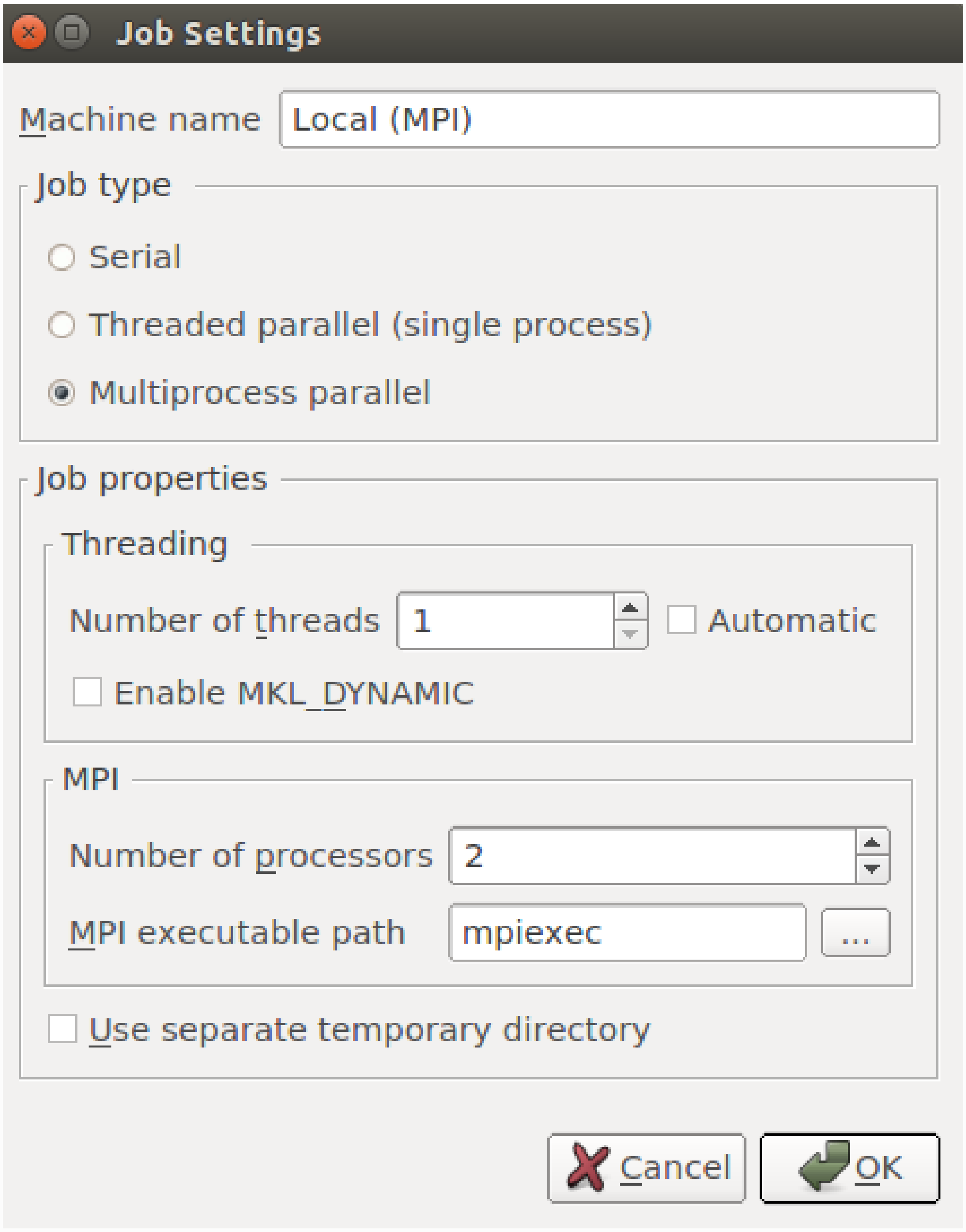
Machine Manager

It may sometimes be convenient to have a predefined local machine that is set up with MPI parallelization as default mode. You can easily add such a machine yourself.

In the Job Manager main window, click to open the **Machine Manager**, and click New ‣ Local.

Then edit the default job settings of the new machine in the window that pops up:

- Name the machine, e.g. “Local (MPI)”.
- Select *Multiprocess Parallel* as job type.
- Make sure threading is turned off (*Number of threads* = 1)
- Choose the default number of processors, e.g. 2.
- Click *OK* to add the new machine to the Machine Manager.


Job Settings

Machine name: Local (MPI)

Job type
- [ ] Serial
- [ ] Threaded parallel (single process)
- [x] Multiprocess parallel

Job properties

Threading
- Number of threads: 1
- [ ] Automatic
- [ ] Enable MKL_DYNAMIC

MPI
- Number of processors: 2
- MPI executable path: mpiexec ...

- [ ] Use separate temporary directory

Cancel OK

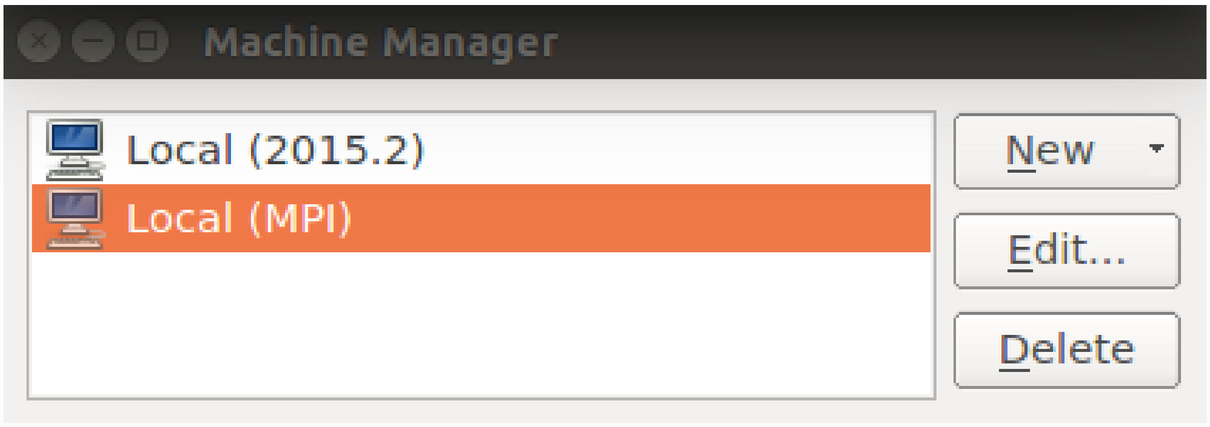


Fig. 23 You can add as many custom machines to the Machine Manager as you like.

Previous Next

© Copyright 2017, QuantumWise.