



# Atomistix ToolKit

## Guide to running in parallel

---

Version 12.2.0

---

# Atomistix ToolKit: Guide to running in parallel

Version 12.2.0

Copyright © 2008–2012 QuantumWise A/S

Atomistix ToolKit Copyright Notice

All rights reserved.

This publication may be freely redistributed in its full, unmodified form. No part of this publication may be incorporated or used in other publications without prior written permission from the publisher.

## TABLE OF CONTENTS

---

1. Introduction .....	1
2. Quick guide .....	2
Prerequisites .....	2
How to run .....	2
Test the parallel setup .....	3
How many MPI processes to use? .....	3
Additional parallelization via OpenMP threading (for multi-core machines) .....	3
3. Detailed parallel guide .....	5
System requirements .....	5
License considerations .....	6
Launching a parallel job using MPICH2 .....	6
Controlling threading .....	9
Resource allocation via queue systems .....	10
4. Parallelization strategies .....	11
Definitions .....	11
Which parts of ATK are parallelized? .....	12
How to choose a parallelization strategy .....	14
Summary .....	15
5. Appendix .....	16
Simple bash script for launching mpiexec .....	16
Advanced bash script for launching mpiexec .....	17
Simple PBS script .....	18
Longer PBS script .....	19
mpdboot .....	20
MPI safety in I/O operations .....	20

# CHAPTER 1. INTRODUCTION

---

Accurate atomic-scale electronic structure and transport calculations of realistic systems are often very time-consuming. One important way to reduce the calculation time is to employ **parallel** computing. The speed-up can easily be more than ten times on a cluster, but the efficiency improvement strongly depends on a number of factors, which are important to understand.

This document describes how you can take advantage of the parallel power of ATK for various scenarios. It describes which parts of the code that have been parallelized, and discuss how this influences the parallel performance for various types of systems and simulations.

Running in parallel does not necessarily imply that you must have a large number of computers or a super-cluster at your disposal. Even a handful, or just one quad-core PC can provide a performance boost for many systems, if the calculations are set up properly.

## CHAPTER 2. QUICK GUIDE

---

If you are already familiar with running parallel computations, and are just looking for a quick reference, here is a quick guide to running ATK in parallel. Otherwise, and for more details, please refer to [the longer guide](#), which also contains more information on e.g. which parts of ATK that are parallelized, etc.

### PREREQUISITES

---

To run ATK in parallel over MPI, you need

- MPICH2, or a compatible library, installed and properly configured. See the [detailed requirements](#) for important details.
- A parallel license for ATK – specifically your license file should contain "slave" features.

#### **Tip**

Even if you only have a serial license, you can still get a speed-up by parallelizing the calculations using [OpenMP threading](#).

#### **Note**

Both the Windows and Linux versions of ATK are MPI parallelized as well as OpenMP threaded.

### HOW TO RUN

---

To launch a parallel job, use a command of the form:

```
mpiexec -n N /opt/QuantumWise/atk-12.2.0/atkpython/bin/atkpython /home/user/script.py > /home/user/
```

Notes:

- **N** is here the number of MPI processes (see the [recommendations below](#) regarding how many processes to use vs. the number of computers and cores available).
- Edit the paths to reflect your local conditions; note that you must use full, absolute paths to both the ATK binary `atkpython` and the script.
- If you have a machine file specifying the nodes you would like to use, add the option `-machinefile mymachinefile`.

- 
- See [separate important notes on PBS resource allocation](#) if you are submitting your parallel jobs via a queuing system.

Some useful scripts for running ATK in parallel, both directly and via PBS, can be found in the [Appendix](#).

## TEST THE PARALLEL SETUP

---

To verify that the parallel environment is set up properly, run a small test script

```
if processIsMaster():
    print "Master node"
else:
    print "Slave node"
```

on, say, 4 nodes, using

```
mpixexec -n 4 /opt/QuantumWise/atk-12.2.0/atkpython/bin/atkpython /home/user/test_mpi.py > /home/user
```

This should write one line with "Master node" and three lines with "Slave node" (not necessarily in that order) to the log file.

## HOW MANY MPI PROCESSES TO USE?

---

It is very difficult to provide a simple answer, that applies to all situations, to this question. Our general recommendation is however:

- For regular calculations, which are not critical in memory usage, assign **one MPI process per socket** (essentially a socket is a CPU, although this term is outdated).
- For large calculations, which would use more than half of the RAM of each node in serial, assign **one MPI process per node** (that is, per physical machine).
- For very small calculations, in particular with many k-points, assign **one MPI process per core**.

Each MPI process uses the same amount of memory. Therefore, if you run e.g. 8 MPI processes on the same computer, you will consume 8 times the RAM compared to a serial calculation. This limits the number of atoms or size of basis set that can be used in parallel calculation, compared to a serial calculation, if more than one MPI process is assigned to a node.

For more details, see the dedicated section on [parallelization strategies](#).

## ADDITIONAL PARALLELIZATION VIA OPENMP THREADING (FOR MULTI-CORE MACHINES)

---

ATK is also able to use OpenMP threading to speed up the calculations on multi-core architectures. MPI and OpenMP can be used separately, or [in conjunction](#) for optimal performance.

MKL will detect the presence of a multi-core processor and enable OpenMP automatically. It is recommended to let the software figure out itself how many threads to use optimally.

---

More details about controlling the threading environment can be found in a [separate section](#).

## CHAPTER 3. DETAILED PARALLEL GUIDE

---

This part of the guide discusses the details of **how** to run ATK in parallel. In order to run the parallel calculations as efficiently as possible, it is also very important to understand properly the two different parallelization techniques used in ATK, and how they might influence the performance for different types of systems. These considerations have been collected in a separate chapter on [parallelization strategies](#).

### SYSTEM REQUIREMENTS

---

This section covers basic system requirements for parallel setups and related technical details. Generally, these considerations are a task for a system administrator rather than the end-user.

### HARDWARE CONSIDERATIONS

---

In order to perform a parallel computation, more than one computational unit is obviously required. There are however a wide variety of conceivable hardware configurations, including shared-memory multi-processor (SMP) supercomputers, regular multi-core workstations or even laptops, and clusters, which can be anything from an Beowulf-type office setup to a super-cluster with thousands of nodes. All of these architectures will provide a varying degree of performance improvement for ATK via parallelization.

The maximum advantage of running ATK in parallel is obtained by using several multi-core nodes, and use MPI to parallelize over the nodes and threading on each node (or possibly sockets; see the [section on parallelization strategies](#)).

### LIBRARY REQUIREMENTS

---

ATK 12.2.0 is **statically linked** against [MPICH2](#) (version 1.2.1p1). To run in parallel over MPI requires this library (or a newer version of it) or a compatible library such as [Intel MPI](#) or [MVAPICH](#) for Infiniband to be installed on your network.

#### **Tip**

MPICH2 can also be installed on a single machine for MPI parallelization. This typically requires the machine to have quite a large amount of memory, unless the calculation tasks are small.

### PROCESS MANAGERS

ATK supports the following MPICH2 process managers:

- 
- mpd (default)
  - gforker
  - hydra

Note that `smpd` is *not* supported.

## LICENSE CONSIDERATIONS

---

To parallelize an ATK calculation over the cores of a dual- or quad-core computer using OpenMP requires no additional licenses. Using MPI, on the other hand, does.

Each calculation requires one master license to launch the job (this will be used on the master node), and then one additional slave license for each slave node that is used. So, to run in MPI on 4 nodes, you need 1 master license and 3 slaves. The slave licenses are counted per MPI process, not physical nodes, so even if the 3 slave processes run on the same node, you still need 3 slave licenses.

Although the most common case is to use a floating license server in particular for parallel operation (the nodes check out licenses from the license server), it is not a strict requirement. In particular, you *can* run ATK over MPI using a **demo license** for testing purposes.

For questions about licenses, parallel or otherwise, please contact [QuantumWise](#) or your local distributor.

## LAUNCHING A PARALLEL JOB USING MPICH2

---

Below is given a brief description of the standard procedure for launching parallel MPI calculations. Note that some details may differ on your network, so check with your system administrator. In particular, it is not unlikely that you need to use a [queuing system](#) (such as PBS, Portable Batch System) to submit your jobs.

In the following it is assumed that MPICH2 has been properly installed on your system. You can test this by giving the following commands on the master node:

```
mpiexec -h
```

This should display the `mpiexec` help.

Moreover,

```
mpdtrace
```

should display information about the available computer nodes on the network to parallelize over.

### Important

If any of these steps fails, please contact your system administrator for assistance.

---

 **Note**

Your local setup might also require activation of the nodes that you want to use in the calculation using `mpdboot`. Please refer to the [MPICH2 User Guide](#) for more information.

Next, verify that Atomistix ToolKit is properly installed. When running in parallel, having the ATK binary directory in your PATH does not really help, and denote the directory in which the `atkpython` binary is located by `$ATK_BIN_DIR`; in fact it is a good idea to define the corresponding environment variable.

Test the ATK installation by giving the command

```
$ATK_BIN_DIR/atkpython --version
```

This should display the internal Python version number of ATK (currently 2.6.1).

Next, launch a small parallel script to make sure that ATK is properly configured to run in parallel on your system. Use the following script as input file:

```
if processIsMaster():
    print "Master node"
else:
    print "Slave node"
```

Save the script in your home directory, called `/home/user` in the following.

The most important parameter to `mpiexec` is the option “-n” which is used to specify the number of nodes to use for the calculation. In this test, you will use two nodes

```
mpiexec -n 2 $ATK_BIN_DIR/atkpython /home/user/test_mpi.py > /home/user/test_mpi.log
```

Please observe, that it is **always** necessary to give the full path to the ATK binary **and** the input file when using `mpiexec`; it is not enough to put the ATK binary directory in your PATH. This is where the usefulness of the additional environment variable mentioned above comes in!

If all works out, there will be two lines

```
Master node
Slave node
```

written on the terminal indicating that the calculation was indeed run in parallel (the order of the lines may vary; this is not an error).

If all looks fine, you can start to launch your own parallel calculations! If the test fails, please contact your system administrator, and/or consult the [ATK Installation Guide](#) if you suspect the problem is related to the ATK installation.

---

## HOW MANY NODES TO RUN ON?

 **Note**

It is, in general, not a good idea to utilize the maximum number of processors available for MPI parallelization!

---

This is because each MPI process uses the same amount of memory. Therefore, if the calculation uses 4 Gb of RAM in serial, assigning four MPI processes to a node with two dual-core sockets may require up to 16 Gb of RAM on that node!

In addition, if too many processes run on a single node, they will fight for access to the memory pipeline, the cache, etc. This may in fact cause the calculation to run slower than in serial!

A useful rule of thumb is, instead:

**Tip**

Assign one MPI process per [socket](#), or per node (meaning, physical machine) if the calculation uses a lot of memory.

It is recommended to study the chapter on [parallelization strategies](#) in order to maximize the performance of your calculations.

## USEFUL MPICH2 TIPS

---

Below is listed some further useful MPICH2 options that may be of your interest. Please refer to the MPICH2 manual for a complete description of these and other options. The program `mpiexec` is actually a script, that supports a long list of arguments, and may also have been modified locally on your platform to fit particular network needs and requirements.

- If you want to run `mpiexec` in the background, you can append an ampersand to the command. In this case, to avoid a lot of weird messages in the log file, one should also point stdin to `/dev/null`:

```
mpiexec -n 2 $ATK_BIN_DIR/atkpython script.py > script.log < /dev/null &
```

- To run a job locally, on the current node, using 2 processes:

```
mpiexec -localonly -n 2 $ATK_BIN_DIR/atkpython script.py > script.log
```

- To run ATK on 4 nodes:

```
mpiexec -n 4 $ATK_BIN_DIR/atkpython script.py > script.log
```

- Some cluster setups require the flag `-npnode` to be used with `mpiexec`, in order to ensure that the MPI processes are distributed equally among the nodes. For example, if you want to run a calculation on 4 nodes, with one MPI process per node, you should use

```
mpiexec -npnode 1 -n 4 $ATK_BIN_DIR/atkpython script.py > script.log
```

- If your cluster uses a queue system, the machine allocation is usually set up automatically. If for some reason you want to run on a specific set of machines, you can construct a machine file. To run 2 jobs on the specified machines:

```
mpiexec -n 2 -machinefile mymachinefile $ATK_BIN_DIR/atkpython script.py > script.log
```

In order to run on the two machines `slave1` and `slave2`, the file `mymachinefile` would look like:

```
slave1
slave2
```

The command:

```
mpich2version
```

gives information on which version of MPICH2 is installed.

### Tip

Most likely your MPICH2 configuration is set up to use SSH to communicate between the nodes. In this case it is necessary to provide your network password each time a node is added to the calculation. This will quickly become tiresome, at least if you use more than a handful of nodes. A convenient work-around is to use an **ssh-agent** with a password-less RSA/DSA public key. Contact your system administrator for more information.

## CONTROLLING THREADING

In ATK, the **Intel Math Kernel Library (MKL)** library is used for many [time-consuming parts of the calculation](#). MKL parallelizes via threading on multi-core CPUs, and this can provide a substantial performance benefit in many situations.

The threading environment is controlled via two environment variables, **MKL\_DYNAMIC** and **MKL\_NUM\_THREADS**.

### Note

Earlier versions of ATK required **MKL\_DYNAMIC** to be set to **FALSE** for optimal threading on Linux. As of ATK 11.2, threading works properly *without* setting any MKL environment variables. Thus this section is mainly provided for reference.

#### • **MKL\_DYNAMIC**

If this variable is unset (default) or set to TRUE, MKL will always try to pick what it considers the best number of threads, up to the maximum specified by **MKL\_NUM\_THREADS**.

If this variable is set to FALSE, MKL will normally try *not* to deviate from the number of threads the user requested via **MKL\_NUM\_THREADS**. This does not ensure that MKL *will* use the number of threads requested, if e.g. there are insufficient system resources, or the library considers a different number of threads more suitable for a particular task.

#### • **MKL\_NUM\_THREADS**

If this variable is not set, MKL will use a value for its internal maximum number of threads deduced from the hardware.

If the variable is set to a positive number, MKL will use the value as the maximum number of threads to use according to the value of **MKL\_DYNAMIC**.

More details can be found in the [MKL documentation](#).

---

## RESOURCE ALLOCATION VIA QUEUE SYSTEMS

---

Quite commonly, medium or large clusters operate a queue system, to distribute the calculations evenly among both compute nodes and users. This section provides some general tips, using the popular PBS queue manager as example. It does not cover all PBS options, but just focus on the point of resource allocation, since it is important to carefully balance the requested resources against the number of MPI processes launched by `mpiexec`.

### Important

Your cluster may operate differently. Always check with the system administrator for detailed usage instructions!

For instance, if you want to run 4 MPI processes (i.e. `mpiexec -n 4 ...`) on a cluster where each node has 2 dual-core sockets, use

```
#PBS -l nodes=2:ppn=4
```

to run one MPI process per socket.

If the job is large, a better alternative to avoid running out of memory is to use 4 physical nodes:

```
#PBS -l nodes=4:ppn=4
```

Note that both cases reserve all the cores on each node! That way, you prevent the queue manager from allocating other jobs to "your" nodes (which would limit the memory available to you calculation), and second you can now use the additional capacity for OpenMP threading, for an additional parallel performance improvement.

Complete sample PBS scripts can be found in the [Appendix](#).

# CHAPTER 4. PARALLELIZATION STRATEGIES

---

This chapter discusses how ATK is parallelized, and how this influences the strategies that provide an optimal performance benefit from running the calculations in parallel.

## DEFINITIONS

---

Before going further, there is some important terminology to be defined, to avoid misconceptions.

## CORES, SOCKETS, AND NODES

---

Before the advent of multi-core processors, the term CPU was used to identify and count individual processor units in a computer system. This term can still be used to describe a physical object that sits in your machine, but when discussing resource allocation for parallel computing the words "CPU" and "processor" are ambiguous, and are therefore best avoided.

Instead, there is a hierarchy of three levels: cores, sockets, and nodes.

1. **Node**, also called "host", "computer", "machine". The definition of a node is that a certain amount of memory (RAM) is physically allocated on each node. Nodes are typically separated by network connections, and each node has a unique network address (host name/IP number).

Each node contains one or more sockets; the node is then labeled e.g. a "two-socket" or "four-socket" node.

2. **Socket** refers to collection of cores with a direct pipe to memory. Note that this does not necessarily refer to a physical socket chip, but rather to the memory architecture of the machine, which will depend on the chip vendor specifications. Usually, however, the sockets resemble the old definition of a CPU (or single-core processor).

Each socket contains one or more cores; the socket is then labeled e.g. a "dual-core" or "quad-core" socket. Each socket has its own L2 cache chip, which typically is shared among its cores.

3. **Core** refers to a single processing unit capable of performing computations. A core is the smallest unit of resource allocation. Each core has its own L1 cache chip.

## PARALLELIZATION TECHNOLOGIES

---

ATK is parallelized on two levels:

- 
- MPI (Message Passing Interface) is typically used for parallelization over several nodes in a cluster, although one can assign MPI processes to individual sockets and even cores.
  - OpenMP threading, for multi-core architectures.

 **Note**

We refer to a **serial calculation** as one *not running under MPI*.

MPI and OpenMP use fundamentally different approaches to parallelization. By using a combination of both, you can often achieve a substantial speed-up of the calculations. However, it is important to know how they work, in order to employ them correctly.

## MPI

MPI (Message Passing Interface) is primarily intended for parallelization over nodes. The software distributes the computational load over the nodes by assigning each node with a particular task, and then collects the results in the end. The calculation is controlled by a master node (the one that distributes the tasks and collects the results); the others are called slave nodes.

It is in the nature of MPI that each process holds a complete copy of the entire calculation, but only performs its own dedicated sub-task in the parts of the software that are parallelized. In the non-parallelized parts, each node does the same job, which seems redundant but is necessary to eliminate the need to communicate all the common data between the nodes over the network.

For this reason, each MPI process holds a copy of several memory objects, and if two MPI processes run on the same node, more memory will be allocated on that node, compared to if the job was run in serial.

MPI parallelization typically works most efficiently when independent and equally sized (in time) tasks can be distributed to the different nodes. [Below](#) is described which parts of ATK have been designed to take advantage of this strategy.

## OPENMP

OpenMP, on the other hand, does not incur any memory overhead, since the processes (or rather threads) that run in parallel operate on a common data set. This parallelization takes place locally, on each node, and can thus very favorably be used also when running in serial on a multi-core architecture, as well as providing an additional boost to MPI-parallelized calculations.

The only part of the calculations in ATK that currently uses OpenMP threading is the **Intel Math Kernel Library (MKL)**. This library does however handle many of the most time-consuming operations, so letting the code thread can often give a noticeable performance improvement.

## WHICH PARTS OF ATK ARE PARALLELIZED?

---

A combination of MPI parallelization and OpenMP threading is employed in ATK to obtain optimal parallel performance. This section describes in which parts of the code these respective technologies are used.

## MPI

---

MPI is used for the calculation of matrix elements, and the evaluation of sums.

---

Each matrix element in the Hamiltonian or overlap matrices is independent of the other elements, so N MPI nodes can evaluate N matrix elements in the same time as a single one on a serial computer.

ATK is parallelized in terms of sums over

- **k-point sampling**, both for the **self-consistent calculation** and in the calculation of physical quantities like the **transmission spectrum**, **band structures**, and so on;
- energy points on the contour integration, both on the real axis (only present for finite-bias calculations) and semi-circular parts;
- energy sampling in the **transmission spectrum**.

In all these cases there is no requirement for cross-communication between the nodes. Also, the data packages sent between the master and slave nodes when collecting the results are small, so there are generally no bandwidth bottle-necks or any complicated issues related to the way the parallel network is put together when it comes to running ATK in parallel. You will therefore not see any particular performance difference between Ethernet and Infiniband clusters.

### MPI MEMORY PARALLELIZATION

It is possible to split up the data over different MPI nodes to reduce the memory usage on each node; this is called MPI memory parallelization.

Currently, the sums over k-points is MPI memory parallelized in ATK, i.e. each MPI process only holds a copy of the data for the k-points it is working on.

For **BulkConfigurations** with many k-points, this will reduce the peak memory usage of the MPI nodes compared to running ATK in serial. For **DeviceConfigurations** it will only reduce the peak memory usage for the electrode calculations, since a serial calculation does not need to hold copies of the different k-points for the device part of the calculation.

Additional MPI memory parallelization is planned for future versions of ATK.

### OPENMP THREADING

---

As mentioned above, OpenMP threading in ATK is limited to Intel MKL. This library is however responsible for some of the most time-consuming tasks in the code.

- **Matrix operations** such as diagonalization, multiplication, etc; notably, however, matrix inversion for device systems is handled differently in ATK, using specially designed algorithms which instead benefit from MPI parallelization.
- The **FFT solver** for the electrostatic potential is provided by MKL, and therefore threads. Solving the Poisson equation represents a large portion of many calculations, and can be an important way to speed things up when working on e.g. graphene systems, where there are not so many atoms (so, matrices are small) and few k-points, but a lot of vacuum and hence large grids to solve for.



#### Note

The multigrid solvers are not parallelized.

- Some **real-space integrals** are formulated as matrix problems, and can thus take advantage of the MKL threading.

---

## HOW TO CHOOSE A PARALLELIZATION STRATEGY

---

Using a proper parallelization strategy is crucial in order to achieve the optimal performance, such that the calculation take the shortest time possible. There are however several constraints to be considered: how many computer nodes are available, what type are they, and – not least – how many licenses are available.

It is very difficult to give a universal recipe for optimal performance, since it depends not only on the computer system, but also on the atomic structures and the numerical parameters. This section provides you with information, that will help you to select the best strategy for your system. In the end, you should run a few tests to see what works best in each particular case.

## BALANCING MPI AND THREADING

---

The information above [above](#), suggest how you should use MPI and OpenMP in conjunction: the terms in the k-point (and energy) sums are distributed over the nodes using MPI, while tasks required for each k-point (and energy) are parallelized using OpenMP on each node.

One of the difficult questions is **how many MPI processes to assign per node**. Obviously, you should never allocate more MPI processes per node than the number of cores on it. But with modern systems typically offering 4 or 8 cores per nodes, it does seem attractive to run 4 or 8 MPI processes per node. At least for ATK, this is however not always a good idea.

- As described [above](#), each MPI process allocates nearly a full copy of the entire calculation. Therefore, if the calculation would use 1.2 Gb of RAM in serial, assigning four MPI processes to a node with two dual-core sockets may require 4.8 Gb of RAM on that node.
- Even if the calculation fits in memory, the cores share certain system components, like the Level 2 cache. Running independent MPI processes on all the cores at once will force them to compete for memory access in a way that actually can cause the calculation to run slower than in serial! This is less of an issue for sockets, and thus one can often safely allocate as many MPI processes per node as there are sockets, provided the calculation fits in memory.
- If all the cores participate in the MPI parallelization, there is no additional room for OpenMP threading. Thus you will miss out on this level of parallelization, which often provides a very good performance improvement. To what extent this is compensated by having more MPI nodes depends on which parts of the calculation that are most time-consuming.

The above points can be summarized in a few simple recommendations:

- For regular calculations, which are not critical in memory usage, assign **one MPI process per socket**, and let MKL thread over the cores.
- For large calculations, which use more than half of the RAM of each node in serial, instead assign **one MPI process per node** (that is, per physical machine).
- For very small calculations, in particular with many k-points, assign **one MPI process per core**. This reduces the threading benefit, which however probably is smaller than the parallelization over k-points, due to the small matrix sizes.

There are however cases when following a different strategy will offer better performance. For instance, for a particular system, the benefit of MPI might be considerably larger than that of OpenMP, and so you could be better off skipping threading altogether and using more MPI processes per node, provided that the nodes has enough memory. This and other examples are considered in the next section.

---

## PARALLELIZATION STRATEGIES FOR DIFFERENT SYSTEMS

---

How much speed-up you can expect from parallelization depends on the system size, the parameters used in the calculation, and the strategy used.

Quite naturally, the more time-consuming the calculation is, the stronger the motivation is to speed it up by e.g. parallelization. Fortunately, larger systems tend to benefit more from parallelization as well. There are however various reasons why a calculation takes long time, and this influences the parallelization strategy.

The efforts for improving the performance through parallelization in ATK have been focused on the parts that typically are most time-consuming for large-scale transport calculations.

- Molecules parallelize poorly in MPI, since the only part which ATK can speed up is the evaluation of the matrix elements. On the other hand, in large molecules you need to diagonalize large matrices, and so there is a definite benefit of OpenMP threading.
- Small bulk systems tend to use more k-points. Here MPI will provide more of a benefit than OpenMP, since the matrices are small, and the electrostatic grids too. Conversely, large bulk systems often use fewer k-points and have large matrices, so the larger the system is, the more important it becomes to use threading. Since at the same time the calculation needs more memory, it is natural to use fewer MPI processes per node.
- Device systems always benefit from MPI parallelization because of the parallelization over the contour integral, and finite-bias calculations even more than zero-bias ones. In the same way as for bulk systems, the more k-points the larger the MPI benefit.
- The evaluation of analysis quantities like transmission spectrum and band structures scale almost linearly in MPI, even up to hundreds of nodes. The benefit of OpenMP is also good here, but most likely not linear, so it might be worth considering running more MPI processes per node, provided memory is not an issue.

## SUMMARY

---

Using the parallel version of ATK should display a significant performance improvement for bulk systems and device systems, due to the parallelization of the calculation of matrix elements, **k**-point sampling, and evaluating energy sums.

Running ATK in parallel is not more complicated than running it in serial, provided the parallel infrastructure is set up properly, which includes having the proper licenses installed.

ATK can employ both MPI and OpenMP parallelization (combined or separately) to take advantage both of multi-node clusters and multi-core computers for speeding up the calculations.

Understanding the various parallelization levels is crucial to obtain the best performance.

## CHAPTER 5. APPENDIX

---

This appendix provides, without too many comments or explanations, some sample scripts that can be useful when running ATK in parallel. You will have to modify at least the path and file names of the scripts to reflect your local conditions. Check with your system administrator for further details!

### SIMPLE BASH SCRIPT FOR LAUNCHING MPIEXEC

---

The script below sets up the environment and launches `mpiexec` directly, in `bash`, i.e. without using a queuing system. It defines some useful variables to make the commands shorter.

```
#!/bin/bash

# Where you installed ATK
ATK_BIN_DIR=/opt/QuantumWise/atk-11.2.0/atkpython/bin/

# Script to run
SCRIPT_DIR=/home/user
SCRIPT=myscript.py
LOGFILE=myscript.out

# How many MPI nodes to run on
n_proc=5

# Go!
mpiexec -n $n_proc $ATK_BIN_DIR/atkpython $SCRIPT_DIR/$SCRIPT > $SCRIPT_DIR/$LOGFILE
```

Save this as a text file, for instance `runatk.sh` and give it executable permissions using the command

```
chmod u+x runatk.sh
```

After editing the name and location of the ATK script, etc, run the bash script as

```
./runatk.sh &
```

#### Note

If you are using `bash` on Windows (e.g. via `Cygwin`), take care to specify `ATK_BIN_DIR` in Linux style (`/cygdrive/c/Program\ Files \ (x86\)/QuantumWise...`), but the path

---

to the script (**SCRIPT\_DIR**) in Windows Python style, like "C:/Users/me/My Documents/scripts" (the quotation marks are important if the path contains spaces!).

## ADVANCED BASH SCRIPT FOR LAUNCHING MPIEXEC

---

Here is a slightly more advanced version of the script above.

```
#!/bin/bash

if [ $# -ne 2 ]; then
    echo "Usage: runatk.sh script.py N"
    echo "where N is the number of MPI processes"
    exit
fi

# Where you installed ATK
ATK_BIN_DIR=/opt/QuantumWise/atk-11.2.0/atkpython/bin/

# Name of the script to run
SCRIPT=$1

# Script is assumed to be in the current directory
SCRIPT_DIR=$( pwd )

# How many MPI nodes to run on
n_proc=$2

# LOGFILE is set to "myscript_1234.out" if SCRIPT=myscript.py
# Here 1234 is the process ID of the run (unique, hopefully)
LOGFILE=$( echo $SCRIPT | sed 's/.py//g' )_$$$.out

mpiexec -n $n_proc $ATK_BIN_DIR/atkpython $SCRIPT_DIR/$SCRIPT > $SCRIPT_DIR/$LOGFILE
```

Save this as a text file, for instance `runatk.sh` and give it executable permissions using the command

```
chmod u+x runatk.sh
```

The main difference compared to the [simple bash script](#) is that there is less need to edit the bash script itself to run different calculations (even on different number of nodes), since you would use the command

```
./runatk.sh myscript.py 5
```

to run `myscript.py` on 5 nodes. The calculation script must reside in the directory where `runatk.sh` is run from.

Thus, you only need to edit the bash script to change the location of ATK itself (after an upgrade, for instance).

### **Tip**

If the calculation does not run as expected, check the logfile for error messages. A typical example is if `mpd` is not running, you will find instructions on how to start it in the logfile.

---

 **Note**

The "\$\$" in the log file name will append the process number, which should be unique for each run. That way you avoid overwriting the old log file by mistake if you re-run the script.

## SIMPLE PBS SCRIPT

---

Here is a simple PBS script that assumes your cluster is set up such that no special further configuration or queue settings are needed to run ATK. The script is quite minimal and doesn't print much information. The [longer script](#) further below is often more useful.

```
#!/bin/bash

# Give your job a name, to identify it in the queue
#PBS -N atkjob
# Allocate 2 nodes, and 4 processors (cores) on each
#PBS -l nodes=2:ppn=4
# Export all environment variables to the PBS queue
#PBS -V

# Define the job
ATK_BIN_DIR=/opt/QuantumWise/atk-11.2.0/atkpython/bin/
SCRIPT_DIR=/home/user/myscripts
SCRIPT=myscript.py


# How many MPI processes to use
NCPUs=4

PBS_0_WORKDIR=$SCRIPT_DIR
cd $PBS_0_WORKDIR

mpiexec -n $NCPUs $ATK_BIN_DIR/atkpython $SCRIPT_DIR/$SCRIPT
```

Save this script as `atk_pbs.sh` and submit it to the queue using

```
qsub atk_pbs.sh
```

 **Tip**

The command `cd $PBS_0_WORKDIR` means that ATK is run in the directory where the script resides. That way, if you write any files (like the NetCDF file!) you do not actually need to specify it with an absolute path, just the name.

 **Note**

In most cases, you should not pipe the output of the calculation to a specified file, when using a queue. Instead, `stdout` and `stderr` are automatically captured and returned in files with names like `job.o1234` and `job.e1234` where "job" is the name of the job specified by the "PBS -N" option and 1234 is the job sequence number.

 **Important**

For details on resource allocation, i.e. how to select the number of PBS nodes vs. MPI nodes, see the [general PBS notes](#).

---

## LONGER PBS SCRIPT

---

Here is a more complete PBS script, with more options. Again, note that your cluster may operate differently and you will have to change several settings to reflect your local conditions.

For definiteness, let us assume your cluster consists of several machines with 4 sockets per node, and each socket has a dual-core chip. The first example runs on 7 nodes; it allocates one MPI process per node, as would be suitable for a larger calculation.

### Note

We have also added the `mpiexec` option `-npernode 1` which is needed on some clusters for the case of one MPI process per node.

```
#!/bin/bash

# Give your job a name, to identify it in the queue (and the output files)
#PBS -N atkjob
# Choose queue
#PBS -q our_queue
# Resource allocation
#PBS -l nodes=7:ppn=8

# Send me an email when the jobs starts (b), finishes (e), or is aborted (a)
#PBS -m abe
#PBS -M myemail@mydomain.com

# Export all environment variables
#PBS -V

# Join STDERR to STDOUT
#PBS -j oe

# Define the job
ATK_BIN_DIR=/opt/QuantumWise/atk-11.2.0/atkpython/bin/
SCRIPT_DIR=/home/user/myscripts
SCRIPT=myscript.py

# Go to working directory
PBS_O_WORKDIR=$SCRIPT_DIR
cd $PBS_O_WORKDIR

# Assign one MPI process per node
NCPU=$( sort $PBS_NODEFILE | uniq | wc -l )

echo -----
echo 'This job is allocated on '${NCPU}' cpu(s)'
echo 'Job is running on node(s): '
cat $PBS_NODEFILE
echo -----
echo PBS: qsub is running on $PBS_O_HOST
echo PBS: originating queue is $PBS_O_QUEUE
echo PBS: executing queue is $PBS_QUEUE
echo PBS: execution mode is $PBS_ENVIRONMENT
echo PBS: current home directory is $PBS_O_HOME
echo PBS: working directory is $PBS_O_WORKDIR
echo PBS: job name is $PBS_JOBNAME
echo PBS: job identifier is $PBS_JOBID
echo PBS: PATH = $PBS_O_PATH
echo PBS: node file is $PBS_NODEFILE
echo -----

mpiexec -npernode 1 -n $NCPU $ATK_BIN_DIR/atkpython $SCRIPT_DIR/$SCRIPT
```

If, instead, you want to use all the cores as MPI nodes:

---

```
# Assign one MPI process per core
NCPU=$( sort $PBS_NODEFILE | wc -l )
...
mpiexec -n $NCPU $ATK_BIN_DIR/atkpython $SCRIPT_DIR/$SCRIPT
```

For the case of running one MPI process per socket, it's probably best to enter the numbers manually:

```
#PBS -l nodes=7:ppn=8
...
# Assign one MPI process per socket (4 dual-core sockets per node)
NCPUS=28
```



## Note

It is assumed that the cluster is configured such that the PBS scheduler, rather than `mpiexec`, handles the machine allocation, and therefore there is no need for any `-machinefile` specification.

## MPDBOOT

---

If your cluster setup requires you to start `mpd` manually, on all the nodes, you can use the following lines of code for inspiration.

```
#!/bin/bash

# ... the usual PBS stuff ...

cd $PBS_0_WORKDIR

# Create MPD node file from the PBS node file
sort $PBS_NODEFILE | uniq -c | gawk '{ printf("%s:%s\n", $2, $1); }' > mpd.nodes

# Start the MPD-daemons
NMPDNODES=$( wc -l < mpd.nodes )
export MPD_CON_EXT=${PBS_JOBID}
mpdboot -f mpd.nodes -n $NMPDNODES --remcons

# Test that the nodes started up successfully
sleep 5
mpdtrace -l
mpdringtest 100

mpiexec ...

mpdallexit
```

## MPI SAFETY IN I/O OPERATIONS

---

All internal functions in ATK are "MPI safe" meaning that they only perform I/O operations on the master node.

If you need to print something to the terminal/log file in scripts that are being run in parallel, use the function `nlprint` instead of the usual `print` statement:

---

```
nlprint("Only printed on the master node (1+1=%g)" % (1+1))
```

If other I/O operations, like writing to files, are required, you must protect them from being executed on the slave nodes by using the function `processIsMaster()`:

```
if processIsMaster():  
    print "Only printed on the master node"
```

### **Warning**

Make sure not to put any native ATK statements inside such a block, or they may hang the script, waiting for slave nodes to respond.

Note that *reading* from files should typically be done on all nodes, otherwise only the master will have the information.