Installation Manual

Amsterdam Modeling Suite 2020

www.scm.com

Oct 30, 2020

CONTENTS

CHAPTER

ONE

WINDOWS QUICKSTART GUIDE

This quickstart guide is for installing AMS on a Windows desktop/laptop machine. Please also read the *generic Installation manual* (page 11) if you encounter problems.

Start with downloading AMS2020 for Windows from the main download page (http://www.scm.com/support/downloads/) (click the orange *Download* button below the blue Windows logo), and save it in your Downloads folder. Open ams2020.101.pc64_windows.intelmpi.exe after the download is complete, for example by opening your Downloads folder in the Windows explorer and double-clicking it. Newer versions of AMS will have a different number in the file name.

Follow the on-screen instructions to install AMS2020 to your computer. The InstallShield Wizard asks where to install the program (C:AMS2020.101 by default), where to save your AMS files (C:AMS_DATA by default), and where to store temporary data (C:SCMTMP by default). All these paths can be changed, but should **NOT** contain any spaces!

During installation a text console window will open to extract a few more files needed for using AMS, do NOT close this window! If you do, the installation has to be started all over again.

Once the installation is complete, **double-click the "AMSjobs" shortcut** on the desktop to start AMS2020.

AMS2020 also includes a **bash/python** scripting environment, which can be started by starting *ams_command_line.bat* from the AMS2020 installation folder. This will open a windows shell with the correct environment set up for running AMS/ADF. The scripting environment also provides a *bash* shell (simply type **bash** and hit enter) and a python stack.

CHAPTER

TWO

LINUX QUICKSTART GUIDE

This quickstart guide is for installing AMS on a Linux desktop/laptop machine. For cluster installations please read the *generic Installation manual* (page 11)

NOTE: The following steps should be taken under a normal user account. Do not use the root account or superuser!

Start with downloading AMS2020 for Linux from the main download page (http://www.scm.com/support/downloads/) (click the orange *Download* button below the penguin), and save it in your Downloads folder. You do not need to open the file. If you have an AMD Zen processor (Ryzen/EPYC), then download the "Linux Intel MPI, optimized for AMD-Zen" binary instead.

Now **open a terminal** (Ctrl+Alt+T usually works, otherwise browse your application menus), and run the commands below to **extract the download into your homefolder**. Make sure to replace the ams2020.101.pc64_linux.intelmpi part to match the name of the file you downloaded (in case of a newer version, or if you downloaded a snapshot or development version). Try to use copy and paste (right-click with your mouse in the terminal screen to paste) to avoid mistyping the commands.

```
cd $HOME
tar -xf $HOME/Downloads/ams2020.101.pc64_linux.intelmpi.bin.tgz
```

Run the following command to **source the amsbashrc.sh file**, do not forget the dot and space at the beginning of the line! Also make sure to replace 2020.101 with the version you downloaded.

```
. $HOME/ams2020.101/amsbashrc.sh
```

Start up the GUI with this command:

```
amsjobs &
```

If this does not open a gui, then use your mouse to select all the text in the terminal window, copy it (right-click and select copy), and send us an email at support@scm.com with the text. DO NOT PROCEED WITH THE NEXT STEP!

If the AMS GUI started without problems, go back to ther terminal window and run the following command to **create a desktop icon** for AMS:

```
$AMSBIN/create_linux_icon.sh
```

Finally we can set up our terminal to automatically source the amsbashrc.sh file when starting. **Add the source command to the .bashrc file** with the following command in the terminal window:

```
echo '. $HOME/ams2020.101/amsbashrc.sh' >> $HOME/.bashrc
```

You can also open the .bashrc file in a text editor, and manually paste the part between the quotes on a new line at the end of the file.

CHAPTER

THREE

MACOS QUICKSTART GUIDE

This quickstart guide is for installing AMS on an Apple MacOS machine. Please also read the *generic Installation manual* (page 11) if you encounter problems.

Start with downloading AMS2020 for MacOS from the main download page (http://www.scm.com/support/downloads/) (click the orange *Download* button below the grey Apple logo), and save it on your computer. Open ams2020.101.macintel64.openmpi.dmg after the download is complete. Newer versions of AMS will have a different number in the file name.

Drag the AMS application (called AMS20XXX) from the disk image to your disk, for example by dropping it on the icon of the Applications folder in the disk image.

Important: the folder in which you store the AMS application should not contain spaces in its name (or in the names of any of its parent folders)!

Installing in the standard /Applications folder should work fine.

The AMS package needs a valid license file to run.

Double click the AMS20XXX application.

On MacOS Catalina when you open AMS20XXX for the first time from your Applications, you'll see a notice "AMS20XXX can't be opened because Apple cannot check it for malicious software". To Fix this

- In the Finder on your Mac, locate the app you want to open.
- Control-click the app icon, then choose Open from the shortcut menu.
- Click Open.

The app is then saved as an exception, and you can open it in the future by double-clicking it or open it from your Applications on the Dock.

If you have no valid license a window will appear that allows you to request and install a license file. If a license file is available for you it will be installed automatically.

Otherwise you will receive a mail when a license file is available. Double click the AMS20XXX application again and request a license again to install it. Note that you will never get more than one license for a particular machine, so no need to worry about requesting a license twice.

CHAPTER

FOUR

INTRODUCTION

This document describes the installation of the Amsterdam Modeling Suite (AMS) on the supported platforms. For optimal performance, some system specific tuning may be needed. Therefore, it is strongly recommended to also read the additional information and known issues .

The Amsterdam Modeling Suite consists of the following main classes of programs:

- Computational engines: ADF, BAND, COSMO-RS, DFTB, UFF, ReaxFF and MOPAC. Each of the engines has its own (text-based) input and output and can be used from scripts and the command line. New in the AMS2019 release is the AMS driver program, which houses the BAND, DFTB, UFF and ReaxFF engines. Within this manual we try to use the word *AMS* for all the computational engines, and *ADF* for the molecular DFT program.
- Utilities and property programs. These are used primarily for pre- and post-processing data of the computational engines.
- Graphical user interface (GUI), which is used to prepare input for computational engines and to visually present their results.

AMS is bundled as a complete package and all components are installed at once. Your license file determines which of the functionality will be available after installation.

The AMS package is written with a Unix-like environment in mind, but Unix/commandline knowledge is not needed install or use AMS from the GUI. Linux laptop and desktop users can follow the instructions in the *Linux Quickstart Guide* (page 3) to install AMS. Windows users can follow the on-screen instructions after starting the installer, or take a look at the *Windows Quickstart Guide* (page 1). MacOS/OSX users can simply drag&drop the AMS application to install it, see the *MacOS Quickstart Guide* (page 5) for more details.

If you plan to do advanced scripting or run AMS from the command line, then you will need to know how to write shell scripts and have some knowledge of the environment variables. If you are the one who is going to install the package on a shared Unix-like system, such as Linux cluster environment, then you need to know how to modify shell resource files such as .bashrc.

4.1 Requirements

4.1.1 Summary

AMS2020 can be used on anything from a simple laptop to big Linux cluster environments, and is tested on a wide variety of hardware.

Recommended minimum hardware specification:

- Intel i5 or better 64bit CPU
- 8GB RAM

- 250GB SSD
- OpenGL 3.2 graphics

Absolute minimum hardware specification:

- 64bit CPU
- 2GB RAM
- 6GB storage for installation
- OpenGL 1.4 graphics

Supported Operating Systems:

- Windows 7/8/10
- OSX 10.13 or newer
- Linux with GLIBC v2.11 or higher: CentOS/RHEL 6 or 7, Debian 6 or newer, SUSE 11.3 or newer, Ubuntu 10.04 or newer, etc. AMS is regularly tested on CentOS 7, Ubuntu 18.04/20.04 and Arch Linux.

Specific hardware and software requirements for the AMS package depend on the platform. The list of supported platforms, including information on the operating system, parallel environment (MPI), and compilers used is available in the Download section (http://www.scm.com/support/downloads/) of our web site.

4.1.2 Detailed Hardware requirements

CPU

AMS2020 runs on any x86_64 CPU, but performs best on modern Intel CPUs with AVX or AVX-512 instruction sets and AMD Zen CPUs (Ryzen, Threadripper, EPYC). Especially the AMD Zen2 processors give very good performance (Ryzen 3000, EPYC 7**2).

Memory

In a parallel calculation, the total amount of memory used by the job is a sum of that used by each process. Starting from ADF2010, some large chunks of data are placed in the shared memory so the sum rule does not strictly hold. In principle, it is more correct to count memory per process but since AMS is an MPI program it runs most efficiently when the number of processes corresponds to the number of physical processors cores. Therefore, all memory amounts below are per processor core.

The amount of RAM per core needed to run AMS depends greatly on the kind of calculation you perform. For small calculations, 256 MB will be sufficient, but if there is a lot of memory available AMS may run significantly faster. A large amount memory also reduces the load on the disks, which may speed up your calculation depending on the I/O sub-system and the operating system.

The memory requirement increases with the system size. For example, for a molecule containing 100 atoms with a DZ basis set it may be sufficient to have 256 MB but for a molecule with 1000 atoms up to 8 gigabytes may be required. Also, if you are going to perform TDDFT, relativistic spin-orbit or analytical frequency calculations then the amount of memory should be larger. As an indication, an analytical vibrational frequency calculation of a organometallic complex containing 105 atoms with a TZP basis set uses up to 1GB of RAM per process but it can be done with less, even though not as efficiently.

Disk

For installation of the package on Linux/Unix you need from about 5GB (without sources) to 8GB (with sources and compiled objects). The run-time (scratch) disk space requirements greatly depend on what type of calculation you perform. For the ADF program, it may range from a few megabytes for a small molecule up to a hundred gigabytes for a large molecule with a large basis set, the amount scaling quadratically with the system size. Using a Solid State Drive (SSD) helps performance, especially for bigger calculations.

Network

First of all, a network card must be present in the computer as its hardware MAC address is used as the computer's ID for the licensing.

In order to enable MPI on a standalone Windows computer, one may need to create a dummy network connection by adding a network "card" called Microsoft Loopback Adapter. This interface will be assigned an IP address from a private range.

Multi-host parallel performance.

As far as performance concerned, a switched Gigabit Ethernet network is typically sufficient for good parallel performance on up to four nodes if the nodes do not have too many CPU cores per node. If you are going to use more nodes, or nodes with high core count (>16 cores per node), you may need faster communication hardware, such as Infiniband, to get good performance. Please note that multi-host execution is not supported on Windows.

Graphics

Starting with ADF2017, the GUI requires OpenGL 3.2 to run. For Linux and Windows users there is an OpenGL software mode available for older hardware, please read more about it in the Remote GUI documentation.

4.1.3 Software requirements

Operating System

The package runs on Windows and on the following Unix variants: Linux, Mac OS X.

On the Apple systems the Mac OS X 10.13 and newer is supported.

On linux both the compute engines, python scripting environment and GUI require a GLIBC version of 2.11 or higher. AMS is compiled on CentOS 6, the code gets tested daily on CentOS 6, Ubuntu 18.04, Ubuntu 20.04 and Arch.

The Windows version of AMS is supported on the desktop editions of Windows (7, 8, 8.1 and 10). The Windows Server is **not** supported.

Additional libraries

Certain version of AMS will require different libraries to be installed on the system depending on the MPI library used.

Graphics

In order to run the the graphical user interface (GUI) the computer needs to have an OpenGL-capable graphics subsystem (hardware, drivers and libraries). Besides that, on Linux the following (or equivalent) packages must be installed:

```
fontconfig
freetype
libdrm
libICE
libSM
libstdc++
libX11
libXau
libXdmcp
libXext
libXft
libXrender
libXScrnSaver (Ubuntu users may need to install libXss1)
libXt
libXxf86vm
```

(continues on next page)

(continued from previous page)

```
mesa-libGL
mesa-libGLU
```

The GUI will not able to start without shared libraries provided by these packages.

NOTE: If you receive an error about libXss (libXss.so.1: cannot open shared object file: No such file or directory), you need to install libXScrnSaver (redhat/centos: yum install libXScrnSaver) or libxss1 (ubuntu/debian: sudo apt install libxss1).

Compiler

If you have a license for the source code, you can compile the source yourself, with or without your own modifications.

The source consists mainly of Fortran95/2003 code, with some small parts written in C. Some of the Fortran2003 features are also used so a compiler supporting it is required. You must use object-compatible Fortran and C compilers to those we are using on the same platform, since some parts of the code are available only as object modules. For all x86 platforms it is currently Intel Fortran 18 or newer. It is very unlikely that other compilers, or even a different major version of the same compiler, will work with these object modules. We cannot support compilers different from what we are using ourselves.

To check which compiler to use, check the detailed machine information on the Download section of our web site.

4.2 Important changes since AMS2018

No major technical changes have been made in AMS2019.

4.3 Important changes since ADF2017

Some of the technical changes made since ADF2017:

- No more support for 32-bit Windows: almost all PCs nowadays run on 64-bit processors and 64-bit Operating Systems. If you need a 32-bit version, we advise you to use ADF2017 instead.
- Automated OpenGL fallback mode for GUI on older graphics: available on 64-bit Windows and Linux.
- No more support for CentOS 5 on Linux: The Intel ifort 18 compiler unfortunately introduces an unavoidable GLIBC 2.11 requirement, which means AMS2018/AMS2019 no longer works on older Linux systems.
- Linux binaries for AMD Zen processors: AMS2018/AMS2019 is available with OpenBLAS instead of MKL, optimized for the AMD Zen architecture. These binaries should be used on AMD Ryzen / Threadripper and Eypc CPUs for better performance.
- Updated Python stack: Our python stack is now based on the Enthought Python v3.6 stack, most included modules have also been updated.
- AVX-512 support: AMS2018/AMS2019 is compiled with AVX-512 optimizations for the latest Intel Xeon Scalable Processors (Skylake SP).

CHAPTER

FIVE

INSTALLATION

Typically installation of the AMS package is simple and straightforward. If you have problems installing it, contact us for assistance at support@scm.com.

To install the AMS package you have to go through the following steps:

Below we discuss each step separately.

5.1 Decide which version to install

Choose the released version or a snapshot. Normally, you will install the released version from the main download page (http://www.scm.com/support/downloads/). The bug-fixed binaries (http://www.scm.com/support/downloads/bug-fixes/) contains the most recent bug fixes. You may want to install a snapshot version if you know that some bugs have been fixed recently. The development snapshots page (http://www.scm.com/support/downloads/development-snapshots/) contains the latest developments. You will only want to download it if you need to use the latest functionality not available in AMS2020.

Choose a platform. AMS is available for a number of platforms. A platform is a combination of the processor architecture, operating system, MPI implementation, and any other specialties such as CUDA or "MKL for AMD-Zen". Currently, the following platforms are officially supported and available from our website:

- Linux: x86-64 (64-bit): IntelMPI, OpenMPI, IntelMPI+CUDA (beta version only), IntelMPI optimized for AMD-Zen (Optimized for Ryzen/Threadripper/EPYC processors)
- Mac OS X Mavericks and later (10.13+): x86-64 (64-bit) with OpenMPI
- Windows: x86-64 (64-bit) with IntelMPI

32 vs. 64 bits. AMS2020 is only supported on 64-bit Operating Systems (and hardware). On 32-bit Windows ADF2017 can be used.

Choose MPI (on Linux). We advise to use the IntelMPI version if possible. It has been tested on both desktop and cluster machines and should work out-of-the-box on most cluster batch systems. The IntelMPI runtime environment is distributed with AMS. If you prefer to use the OpenMPI version instead, keep in mind that if you wish to run

multi-node calculations you need to use a local copy of OpenMPI 2.1 with support for your batch system build in. The OpenMPI runtime environment is distributed with AMS but it is limited to single-node (or desktop) usage.

Cray XC. Cray users can use the IntelMPI version of AMS because it is binary-compatible with Cray MPI shared libraries. Read the MPICH ABI compatibility section in the Cray documentation for more information, and talk to your local system administrator. If the machine has a "cray-mpich-abi" module available, you can try using that in combination with the SCM_USE_LOCAL_IMPI=1 environment setting.

GPU acceleration: **note: the CUDA version of AMS2020 is considered beta software** ADF can use NVidia GPUs for accelerating SCF cycles and frequency calculations. The GPU should have good double precision performance, be of the Pascal architecture or newer, and the operating system needs to be Linux with CUDA 11.0 installed. Examples of good cards for this are: Tesla P100, Quaddro GP100, Tesla V100, and Titan V. You can find the double precision performance on the wikipedia list of nvidia GPUs (https://en.wikipedia.org/wiki/List_of_Nvidia_graphics_processing_units#GeForce_10_series)

Optimized for AMD-Zen: AMS2018 introduced a new supported platform, optimized for AMD Zen (Ryzen/Threadripper/EPYC) processors. Intially this used the OpenBLAS library, but now this platform also uses Intel MKL. The MKL library is forced to run in AVX-2 mode, combined with AVX-2 optimized binaries. This version will NOT work on older AMD processors. Starting with AMS2020 we also have this version available for Windows.

5.2 Download and install the software

All systems: Download an installation package from one of the download pages mentioned above.

Unix and Linux: The installation package is a compressed tar archive. Untar it in the directory of your choice. Desktop users can follow the Linux Quickstart Guide to install AMS2020. Please note that in a cluster environment AMS must be installed on a shared file system accessible from all nodes of the cluster. The archive contains one directory, adf2020.xxx, which, when installed, will be referred to as *$AMSHOME*. The installation package has the IntelMPI runtime and Intel MKL libraries included, you do not need to install these separately!

Mac OS X: The installation package is a disk image (.dmg) file. To install AMS on Mac OS X, double click on the downloaded disk image and drag *AMS2020.xxx* somewhere on your hard disk, such as the *Applications* directory. Be sure to **read the enclosed ReadMe** file for further important details!

Windows: The downloaded file is an executable Microsoft Installer package containing an installation wizard that will guide you through the installation process. Open the file after downloading, and follow the instructions. **note: Do not close the console window that pops up during the installation!** . Please note that you need Administrator privileges to install AMS. After or during the installer you might be asked by the Windows Firewall to grant network access to some executables, this permission is optional. For third-party firewall applications you will need to grant access from scmd.exe to the localhost.

5.3 Set up the environment

Windows users can skip to the next section since for them the environment is set up by the installation wizard.

Mac users may follow the UNIX instructions if they (also) wish to run from the command line. However, read the ReadMe file inside the dmg file for some important extra details! Alternatively, Mac users can start by double clicking the AMS2020.xxx application. The AMS2020.xxx application will automatically set the environment, and start AMSjobs for you. Next, all tasks (GUI or computational engines) started from this AMSjobs will inherit the proper environment.

For users of any **Unix-like** system the following step is mandatory.

For a **single-user** installation, the end-user is likely also the person installing AMS. If the user has a bash shell, it should be sufficient to source the $AMSHOME/amsbashrc.sh:

```
. $HOME/ams2020.xxx/amsbashrc.sh
```

Alternatively, it is also possible to **edit** the $AMSHOME/Utils/amsrc.sh (for sh/bash/zsh users) or $AMSHOME/Utils/amsrc.csh (for csh/tcsh users) file to set a number of important environment variables and source the file:

```
. $HOME/ams2020.xxx/Utils/amsrc.sh
```

or (for tcsh)

```
source $HOME/ams2020.xxx/Utils/amsrc.csh
```

Note: If the amsrc.sh or amsrc.csh file is missing from your Utils folder, you can download then here: `Download amsrc.sh` , `Download amsrc.csh`

To set up the environment automatically when starting a new terminal, add the source command to the $HOME/.bashrc file. It is also possible to create a launcher icon for the GUI by running the $AMSBIN/create_linux_icon.sh script AFTER the environment has been set:

```
$AMSBIN/create_linux_icon.sh
```

For a **multi-user** installation, you can either copy both adfrc.* files to the /etc/profile.d directory (after editing them) or, if your system supports modules, create a module file for AMS2020. The following environment variables must be defined in the module file:

- AMSHOME: AMS installation directory
- AMSBIN: should be equal to $AMSHOME/bin
- AMSRESOURCES: should be equal to $AMSHOME/atomicdata
- SCMLICENSE: complete path of the license file, typically $AMSHOME/license.txt
- SCM_TMPDIR: path to the user's scratch directory, for example, /scratch/$USER. This directory must exist prior to the first execution of any program from the AMS package by that user. Thus it may be a good idea to add to the user's profile a command that creates the directory in case it does not exist. You can also choose to use the same directory for all users. See SCM_TMPDIR Environment variable for more details.

A complete list of environment variables is provided in Appendix A.

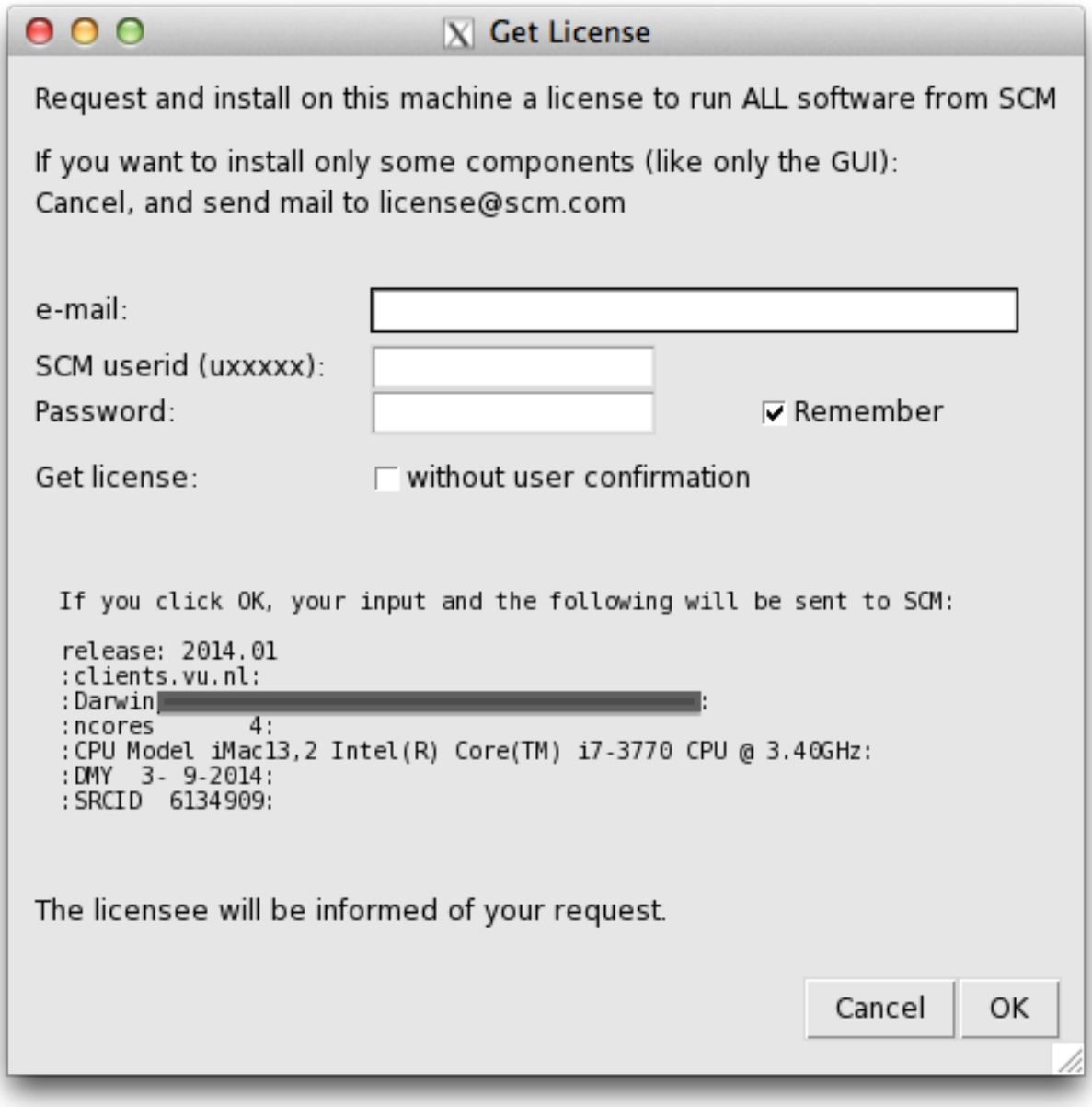
If you are planning on using the GUI on a remote machine (for example via ssh with X-forwarding), make sure to also take a look at Using the GUI on a remote machine.

If you plan on running jobs directly from the command line, please do **not** mpirun the programs directly. The MPI startup is done automatically, for more information see the Additional Information on running MPI jobs.

5.4 Set up the license

Which functionality can be used on each computer is determined by the license file pointed to by the SCMLICENSE environment variable.

When you start any GUI program from the AMS package (like ADF, BAND, AMSjobs or AMSinput) it will try to install a license file for you if necessary. To do this, some information is needed:



`e-mail:` Your e-mail address, this may be used to contact you and send you a license file.

`SCM userid (uxxxxx):` The same as the download login you should have received.

`Password:` The download password belonging to the SCM userid

`Remember:` If checked, the SCM userid and password will be saved on your computer, so you do not have to enter them again. They are saved in a readable format, so only do this on a computer and account you trust.

`Get license without user confirmation:` If checked, a license request will automatically be issued when necessary, without user intervention. The intended use is for running on clusters, or for use in classroom/workshop situations. It will work only after arranging this with SCM first.

Click OK to request and install a license to run on your current machine. The information will be sent to the SCM license server.

The same autolicense procedure can also be started from a shell environment:

```
$AMSBIN/autolicense nogui -u username -p password -m mailaddress
```

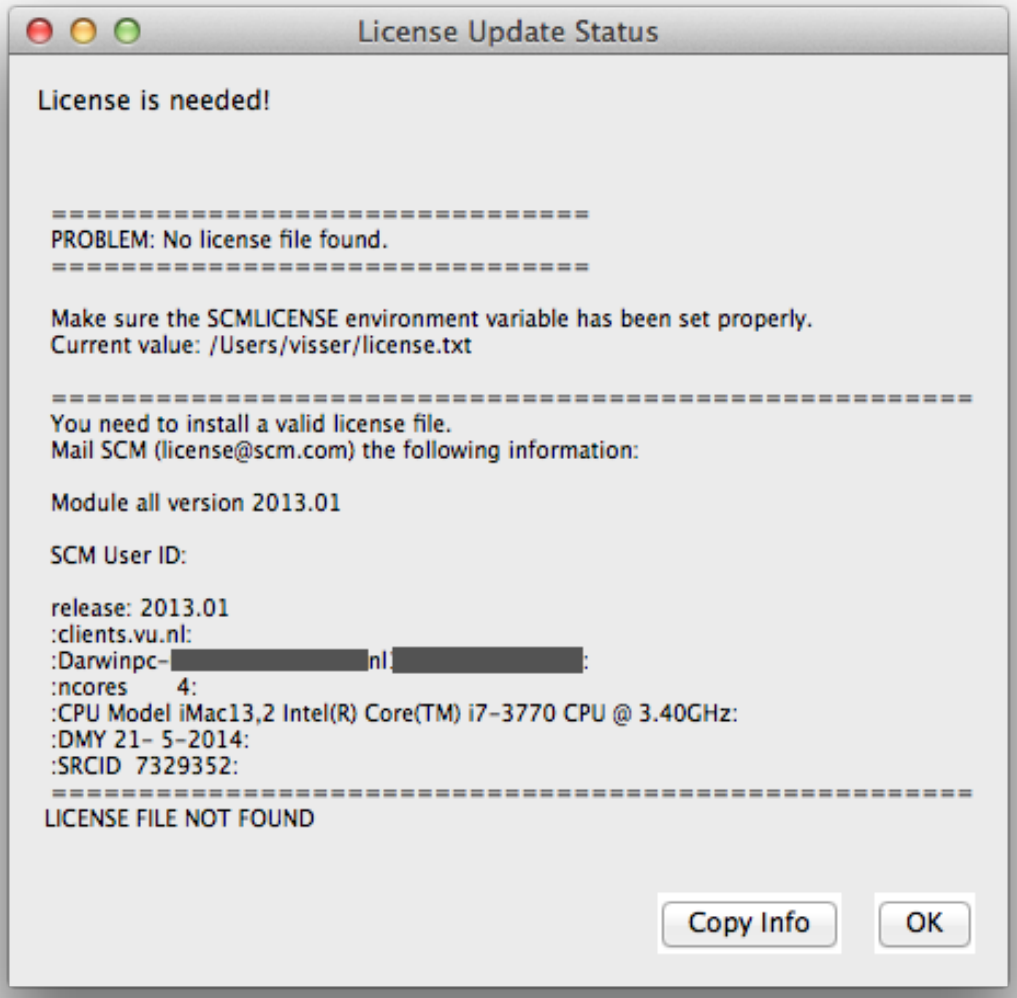
If a license is available, or can be generated by the license server (for **trial** users) it will be installed automatically. Obviously you will need to have an internet connection with access to the outside world for this to work.

For **Non-trial** users, license requests are handled manually. After some time (between hours and days) you will be notified by mail that a license file has been prepared for you.

Run the software again (on the same machine with the same SCM userid), and in many cases the license will automatically be installed for you. If not, follow the "Manual license installation" instructions (further down on this page).

Click Cancel when you do not want to request a license to run on the current machine, if your machine has no internet connection, or if you wish to request and install a license file manually.

A window will appear (either a regular Text editor on your system, or the (License Update Status) appears telling you a license is needed:



It will show the info that needs to be mailed to SCM to get a license. You can copy that information to your clipboard by clicking the Copy Info button, or the usual copy tool in your editor.

Next, mail this information to license@scm.com if you indeed wish to request a license for this machine.

After some time (hours-days as the license request is processed manually) you should receive a mail from SCM containing the license file.

You have receive a new license file via email. **Do not make any changes to it**, as that will break the license!

Mac users can normally just drop the license file on the AMS2020 application icon.

Other users should save the license file such that $SCMLICENSE points to it. Note the value of SCMLICENSE was shown in the License Update Status dialogue.

The default location of the license file (for Windows and the default SCMLICENSE value from a adfrc.* file) is set to $AMSHOME/license.txt, which means you should save the file as license.txt in the AMS installation directory. For macs the default location is "$HOME/Library/Application Support/SCM/license.txt".

Unix-like users can generate the license information by running the following command at the shell prompt in a terminal window:

```
$AMSBIN/dirac info
```

Output of this command from all computers on which AMS will be running, including all nodes of a cluster if applicable, must be sent to license@scm.com.

After receiving this information SCM will prepare a license file matching your license conditions and e-mail it to you with instructions on how to install it.

After receiving your license file you will need to save it so that $SCMLICENSE points to it.

In a multi-user environment, make sure that permissions on the license file allow read access for everyone who is going to run AMS.

5.4.1 Floating license

Note: floating licenses are only intended for Linux clusters.

If you have a floating license, you will need to follow these instructions below to set it up. The instructions are simple, but it is important you follow them exactly.

If you do not have a floating license you may skip this section.

Create a floating license directory

Make a new directory, for example FloatADF, to keep track of the running ADF processes.

This FloatADF directory must be:

- in a fixed location (the directory name should not change!)
- shared between / mounted on all nodes where you want to run ADF
- writable by all users that want to run ADF
- FloatADF must **not** be a subdirectory of $AMSHOME

For example:

```
cd /usr/share
mkdir FloatADF
chmod 1777 FloatADF
```

If you also have a floating license for other modules, you need to set up different directories and repeat this procedure for all these modules (for example FloatBAND, FloatReaxFF, FloatDFTB, . . .)

In the example, we have given all users read, write and execute permissions to this directory. If you wish to restrict this for security reasons, you may do so as long as all ADF users will have read, write and search permission for this directory. You may create an ADF user group for this purpose.

Important: the directory should **not** be a sub-directory of $AMSHOME as the directory name will change if you update to a new version! Also, do **not** put the license.txt file in the FloatADF directory.

The FloatADF directory may not be moved, deleted or renamed for the duration of your license because these actions will invalidate it!

E-mail us the license information Send the result of the following commands (using again the name FloatADF and the location /usr/share as an example) to license@scm.com:

```
cd /usr/share
ls -lid $PWD/FloatADF
```

Please note that output of the ls command must include the full path of the FloatADF directory.

Together with the output of the ls command above, you also need to send us output of the *$AMSBIN/dirac info* command from each computer on which ADF will possibly run, as the license file is still host-locked.

In the case of very large clusters, it is often sufficient if you send us the output of the *$AMSBIN/dirac info* command from the head node and 2 or 3 compute nodes. As most compute node names have the same beginning, we can then add them with a wild card (for example Linuxchem*). It is important when you use this facility, to let us know that the info you are sending are not all compute nodes. Otherwise the license will only run on these few compute nodes.

5.5 Set up the scratch space

Most programs from the ADF package use disk for temporary data. This data often takes a significant amount of space and is used frequently. To avoid run-time errors and performance loss you may want to make sure that the file system used for temporary files is both big and fast. The SCM_TMPDIR environment variable is used to tell the programs where to put their temporary data. Please note that SCM_TMPDIR should always be set. If it is not set then each process will create its own directory in the current working directory where is was started.

Please see Appendix A on additional information about the SCM_TMPDIR variable.

Using multiple disks

Sometimes, if you have multiple non-RAID disks in your system, you may want to spread scratch files across different physical disks for better performance. It is possible to request that every AMS MPI-rank creates its files in a different directory by adding "%d" in $SCM_TMPDIR. If a "%d" string is encountered in the value of SCM_TMPDIR variable it will be replaced by the MPI rank number of the process at run-time. This means, however, that you may have to create up to 128 or more (depending on the maximum number of processes in one parallel calculation) symbolic links on each node where AMS is supposed to run. You should also create a directory matching the SCM_TMPDIR's value literally so that any process that does not interpret '%d' could also run.

Example: suppose there are two scratch file systems, /scratch1 and /scratch2 located on two different physical disks of an 8-core workstation. We want the odd rank numbers to use /scratch2 and the even numbers to use /scratch1. One way to achieve this is to create an empty /scratch directory and create nine symlinks in it as follows:

```
ln -s /scratch1 /scratch/%d
ln -s /scratch1 /scratch/0
ln -s /scratch2 /scratch/1
ln -s /scratch1 /scratch/2
ln -s /scratch2 /scratch/3
ln -s /scratch1 /scratch/4
ln -s /scratch2 /scratch/5
ln -s /scratch1 /scratch/6
ln -s /scratch2 /scratch/7
```

After that set SCM_TMPDIR to "/scratch/%d" and the ranks 0, 2, 4, 6 will use /scratch1 while ranks 1, 3, 5, and 7 will use /scratch2. When running AMS on a cluster it is better to combine multiple disks in a RAID 0 (striping) configuration as you probably do not want to create hundreds of symbolic links on each node.

5.6 Test your installation

This is a very important step and it should never be skipped.

5.6.1 Check the license file

Windows users test their license by double-clicking the makelicinfo.bat file in the AMS installation folder.

Unix users: first check that the license file has been installed correctly by running (at the shell prompt):

```
$AMSBIN/dirac check
```

This should produce the output similar to the following:

```
Checked: /home/testadf/ams2020.xxx/license.txt

 License termination date (mm/dd/yyyy):  1/ 8/2021

 According to it, you are allowed to run:
           ADF version 2020.990
          BAND version 2020.990
          DFTB version 2020.990
        DCDFTB version 2022.000
        QNDFTB version 2020.990
        REAXFF version 2020.990
        ADFGUI version 2020.990
       BANDGUI version 2020.990
       DFTBGUI version 2020.990
     REAXFFGUI version 2020.990
      MOPACGUI version 2020.990
         QEGUI version 2020.990
           CRS version 2020.990
           GUI version 2020.990
         MOPAC version 2020.990
           NBO version 2020.990
          NBO6 version    6.000
           AMS version 2020.990
         Utils version 2020.990

 Number of procs you are allowed to use for:

 ADF    :    1024 procs
 BAND   :    1024 procs
 DFTB   :    1024 procs
 ReaxFF :    1024 procs

 ======================================================
 SCM User ID: u999999
 release: 2020.101
 :example.com:
 :Linuxmaster.example.com00:11:22:33:44:55:
 :ncores      12:
 :CPU Model Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz:
 :DMY  1- 7-2016:
 :SRCID  3217288:
 ======================================================
LICENSE INFO READY
```

If the license check is successful (i.e. you get the "LICENCE INFO READY" message) you can move on. Otherwise check that the license file has been installed according to instructions above.

5.6.2 Run some basic tests

Verify that you can now run examples provided with AMS and that they give correct results. We recommend that you consult the Examples document for notes on such comparisons: non-negligible differences do not necessarily indicate an error. If you have a license for the GUI you can also run a few GUI tutorials as a test.

Note: the example *.run files are complete Bourne shell scripts that should be executed as such, they are **not** input files to be fed into any program. The easiest way to run them is using AMSjobs.

5.6.3 Test the GUI

If the GUI is included in your license, check that you can start any of the GUI modules.

UNIX users:

Enter the following command:

```
$AMSBIN/amsjobs
```

An AMSjobs window should appear. If your GUI crashes, especially when starting AMSinput, try setting the SCM_OPENGL_SOFTWARE=1 environment variable before launching the GUI:

```
export SCM_OPENGL_SOFTWARE=1
$AMSBIN/amsjobs
```

Mac users:

Double click the AMS2020.xxx application to start it. An AMSjobs window should appear.

Windows users:

Double click the AMSjobs icon to start it. An AMSjobs window should appear. If the window does not appear or appears after a long delay then check the AMS-GUI requirements and check the firewall rules that should allow local communication.

All users should now be able to start AMSinput via the SCM menu on the top left of the menu bar.

5.6.4 Test parallel execution

It is very important to make sure that computer resources are utilized with the maximum efficiency. Therefore, you should check that each AMS job uses all processors/cores allocated to it and that the network is not overloaded with the disk I/O traffic.

Typically, when you submit a parallel job to the batch system you have a possibility to specify how many processors per node (ppn) to use. If the batch system you are using allows this, then make sure that you request ppn equal to the number of physical cores on each node. Note that AMD processors from the Familiy 15h based on the so-called "Bulldozer" cores have one floating-point unit (module) per two physical cores. On these Bulldozer architectures and its successors (Piledriver, Steamroller) AMS will probably perform best when you only run on half the physical cores. AMS tries to automatically detect the number of floating-point units (half the physical cores), but the user is advised to check this and otherwise set ppn accordingly. This is no longer the case for the AMD Zen (Ryzen/Threadripper/Epyc) architecture, but for these processors it is advisable to use the "IntelMPI optimized for AMD-Zen" version of AMS2020. This version forces Intel Math Kernel Library (MKL) to run in AVX-2 mode for better performance, as the auto-detection mechanism in MKL switches to the slowest code-path for non-Intel processors.

It is also possible that your batch system does not allow you to specify ppn but instead it always assumes that there only one processor per node. In this case, you will need to edit the $AMSBIN/start file and add some commands for processing the hostfile.

In order to check that everything is working as intended, create a reasonably small AMS job and start it, preferrably on more than one node. After the job has finished, open the job's .out file and find the table that looks like the following:

```
Parallel Execution: Process Information
================================================================================
Rank    Node Name                                NodeID    MyNodeRank   NodeMaster
   0    compute-0-0                                 0           0           0
   1    compute-0-0                                 0           1          -1
   2    compute-0-0                                 0           2          -1
   3    compute-0-0                                 0           3          -1
   4    compute-0-1                                 1           0           1
   5    compute-0-1                                 1           1          -1
   6    compute-0-1                                 1           2          -1
   7    compute-0-1                                 1           3          -1
================================================================================
```

Check the names in the "Node Name" column and verify that the number of tasks per node is correct. If it is, then move on to the next section.

5.6.5 Test parallel performance

If the process allocation to nodes looks as expected then scroll down a bit to the following lines:

```
Communication costs MPI_COMM_WORLD:      1.026 usec per message,   0.0320 usec per 8-
↪byte item
Communication costs for intra-node:      1.023 usec per message,   0.0318 usec per 8-
↪byte item
```

Make sure that you get reasonable numbers (less than 100 usec per message) both for intra-node and MPI_COMM_WORLD communication costs. Otherwise contact your system administrator. If only the MPI_COMM_WORLD value is large but the intra-node one looks reasonable, this may mean that the network link between machines is very slow and you may want to run single-node jobs only.

If all seems to be OK then move to the end of the file to the table called "Timing Statistics". Before the table, there is a line of text beginning with "Total Used" that might look as follows:

```
Total Used :   CPU=     8064.87      System=     1144.31     Elapsed=     9212.99
```

This shows how much time (in seconds) was spent in the AMS code (CPU) and in the kernel (System), and how long did it take for the calculation to complete (Elapsed). Ideally, the system time should be small compared to the CPU time and the latter should be close to the Elapsed time. The system time will not always be a small number, however, the sum of the System and CPU times should always give a number very close to the Elapsed time. If this is not the case then it means that AMS has to spend a lot of time waiting for something. This can be, for example, disk I/O or network communication.

If you notice that the system time portion is enormously large or that there is a large difference between the CPU+System and the Elapsed time, then repeat the job with the "PRINT TimingDetail" keyword in the input and contact the SCM support.

5.7 Configure AMSjobs queues and 3rd party software (optional)

If you would like to use the GUI as a front-end to submit your jobs to queues, you should configure those queues in AMSjobs. Third party software MOPAC and ffmpeg (export movies) may also be installed separately.

5.7.1 AMSjobs queues

A video shows how to set up remote queues on Windows (https://www.scm.com/wp-content/uploads/Videos/RemoteQueuesWithAMSjobs.mp4), other platforms are even easier.

AMSjobs can submit and monitor jobs for you on different queues, both locally and remotely. You can use the GUI on your desktop or laptop for creating, submitting and analyzing your jobs, while running the jobs on remote compute clusters. In that case you should establish and ssh-agent connection, to enable remote job managing by the GUI. If you just run local jobs without a queuing system, skip this section.

To set up a new batch queue, select **Queue** → **New...** → and choose an example queuing system to starting with (LSF, PBS, or SGE). Modify the input files such that they correspond to your queue configuration:

- change the **Name** of the queue
- set the **Remote host** to the hostname of your cluster
- set **Remote user** to your username on that cluster
- change the **Run command** in accordance with your typical queue set up. The **$options** corresponds to the input box that can be modified in AMSjobs before submitting, e.g. the time or number of nodes
- you can set what $options defaults to in the **Default Options** field
- change the **Kill**, **Job status**, **System status commands**, if necessary
- you may define a **Prolog** or **Epilog** command

5.7.2 Managing remote jobs

To manage remote jobs, you need automatic authentication with an ssh-agent via an ssh key pair. This is most easily set up on MacOS or Linux, but is a bit more involved for Windows.

ssh connection on MacOS and Linux

If you don't have an ssh key pair already, you can generate one in a terminal: ssh-keygen -t rsa Put your pass-word protected public key on the compute cluster(s) in ~/.ssh/authorized_keys.

Next, you should establish a connection via an ssh-agent. On MacOS an ssh-agent runs by default, and with keychain you just have to type your password once when you make the first connection.

On various Linux installations ssh-agent is also running by default. If an ssh-agent daemon is not yet running in the terminal where you will run AMSjobs, start the daemon, e.g. by adding eval $(ssh-agent bash) to your .bashrc. You need to add your private key to the ssh-agent daemon: ssh-add.

AMSjobs will use SSH to connect to the remote systems, making a new connection for everything it does. This means that there will be many connections to the remove machine(s). If you are using OpenSSH (both on the local and the remote machine), you can instead use one ssh connection and keep it alive, reusing it for many things. This makes AMSjobs faster, and may avoid complaining system administrators of the remote machines.

To do this, set the environment variable SCM_SSH_MULTIPLEXING to yes (for example via the AMSprefs application).

ssh connection on Windows

PuTTY tools, automatically installed with ADF, can be used to set up an ssh-agent connection. Follow these steps, using the PuTTY programs located in the bin\Putty directory in your AMS installation directory.

- Run puttygen.exe, set the key length to at least 2048 bits and press the Generate button. This will generate an ssh key pair. Type a key passphrase and save the private key to a file. Add the corresponding public key to the ~/.ssh/authorized_keys file on the remote host.
- Run pageant.exe and add your private ssh key (right-click on the Pageant icon in the task bar) with the correct passphrase.
- Open AMSjobs menu Help|Command-line and run "ssh user@host uptime" using correct username and hostname. Type **yes** if prompted to store the key in cache. If you get prompted for a password or get authentication errors then something is not configured correctly. The remote job submission will not work until all problems are resolved.

Now you can close the prompt and start using AMSjobs to manage remote jobs. You may test your queue configuration: **Jobs** → **Generate Test Job** and assign it to your new queue before running the test job.

Centrally defined queues

A system administrator may also define a queue centrally, which may then be picked up by any user automatically via Dynamic queues. Consult the GUI manual for information on how to set these up in AMSjobs.

ffmpeg

The AMSmovie GUI module can make MPEG videos of the system as you see it in the AMSmovie window. This can be an MD trajectory or vibrational modes, or the course of geometry optimization. For this to work, you need to install the free ffmpeg program from the FFmpeg website (http://ffmpeg.mplayerhq.hu/) and make sure it can be found in the path. The simplest way to do this is to copy the ffmpeg executable to $AMSBIN.

CHAPTER

SIX

ADDITIONAL INFORMATION AND KNOWN ISSUES

6.1 Windows Subsystem for Linux (WSL) and Docker

AMS does **NOT** run in parallel under WSL1, WSL2, or Docker on Windows, because the IntelMPI library does not support these configurations.

6.2 Shared memory and batch systems (SLURM)

When discussing memory usage by a parallel job one should distinguish the private and shared memory. The private memory is, well, private to each process and to get the total job's memory usage one should add up the sizes of private memory segments from all processes. The shared memory is used by several processes on a shared-memory node, so for the job's total one should add each shared memory segment only once. However most batch systems ignore the fact that shared memory segments are shared and instead of adding it to the total on the per-node basis they add it per-process. This leads to the batch system greatly overestimating the memory consumption for jobs that use a lot of shared memory, for example, ADF calculations with hybrid functionals. This may lead to such jobs being killed and/or the users being over-charged or forced to use the more expensive hugemem queues. This would be totally unnecessary if the job accounting was properly configured. Luckily, SLURM has the JobAcctGatherParams=UsePss (https://slurm.schedmd.com/slurm.conf.html) option (off by default) that is supposed to take care of the private vs shared memory difference and ensure the correct accounting of ADF jobs. You can check the status of JobAcctGatherParams on your SLURM system with the following command:

```
scontrol show config | grep JobAcctGatherParams
```

6.3 GPFS file system

Starting with AMS2019, the KF sub-system (used for handling binary files such as ADF's TAPE* files) has been rewritten to use memory-mapped files. The mmap() system call implementation is file-system dependent and, unfortunately, it is not equally efficient in different file systems. The memory-mapped files implementation in GPFS is extremely inefficient. Therefore the users should avoid using a GPFS for scratch files.

6.4 Running MPI jobs

Most programs in the Amsterdam Modeling Suite (AMS) are MPI-enabled, and they are set up to run in parallel automatically, both from the GUI and the command line. For example, to run a parallel AMS calculation from the command line, simply do:

```
$AMSBIN/ams < myinputfile
```

A common mistake by people who have worked with MPI programs before is to "mpirun" the programs manually, which results in a parallel mpirun call and usually a lot of errors. All MPI programs in the AMS suite are executed via *$AMSBIN/start* to set up the environment and call mpirun for you. If you need to modify the *mpirun* command, please edit the *$AMSBIN/start* script.

6.4.1 Technical explanation

In the example above, AMS is started in parallel, which involves a couple of files:

- $AMSBIN/ams: a symbolic link to the $AMSBIN/start script
- $AMSBIN/start: this script sources $AMSBIN/setenv.sh to set up most of the environment, determines the name of the used symlink, and then starts the related binary (ams.exe in this case) via mpirun. On Linux clusters it also attempts to detect the scheduler to launch the MPI program with the allocated resources.
- $AMSBIN/setenv.sh: when sourced, this script sets up the correct environment for locating libraries and runtimes such as MKL and IntelMPI
- myinputfile: A valid input file. Some examples can be found in $AMSHOME/examples, and of course the input is also documented per program in the manual.

6.5 More on running MPI jobs

MPI (Message Passing Interface) is a standard describing how to pass messages between programs running on the same or different machines.

MPI is a formal standard and it is actively supported by all major vendors. Some vendors have highly-optimized MPI libraries available on their systems. There are also a couple of open-source implementations of the MPI standard, such as MPICH and OpenMPI. There are also numerous commercial MPI implementations that support a wide range of systems and interconnects, for example, HP-MPI (formerly known as Platform-MPI) and IntelMPI.

Support for a particular MPI implementation in ADF can be considered at three levels: the source code, the configure script, and pre-compiled binaries. At each level different MPI implementations may be supported.

The ADF source code is not implementation-specific and thus theoretically it supports any MPI library. Many popular MPI implementations are supported at the level of the configure script, but depending on your local setup you may need to make some modifications in the buildinfo file after running configure. For example on 64-bit Linux IntelMPI and OpenMPI should work directly, but using other MPI flavours will most likely require manual changes to the *buildinfo* file correct the include and linker paths to the MPI libraries of your system. The configure script will also try to generate an appropriate $AMSBIN/start script, but this might also need modification when using different MPI libraries. In general it is best to use the same MPI version used by SCM for the precompiled binaries.

When choosing an MPI implementation for pre-compiled binaries, SCM considers many factors including (but not limited to) the re-distribution policy, performance, and built-in support for modern interconnects. IntelMPI is currently the standard MPI implementation supported by SCM because it has the most favorable combination of these factors at this moment. For platforms where IntelMPI is supported its runtime is distributed with ADF (Windows, Linux). OpenMPI builds are also available for linux, but should only be used in case of problems with IntelMPI. A different MPI implementation will be standard on a platform where IntelMPI is not available. It may or may not be distributed with ADF. For example, SGI MPT is standard on SGI machines and OpenMPI is standard on Mac OS X platforms, but only the latter is distributed together with ADF.

When pre-compiled binaries do not work on your computer(s) due to incompatibility of the standard MPI library with your soft- and/or hardware, the SCM staff will be glad to assist you in compiling ADF with the MPI implementation supported on your machine(s).

If you are going to use an MPI version of the ADF package, and it is not IntelMPI or OpenMPI, you will need to determine if the corresponding MPI run-time environment is already installed on your machine. If not, you will need to install it separately from ADF. As it has been already mentioned, IntelMPI and OpenMPI are bundled with the corresponding version of ADF so you don't need to worry about installing them separately.

Running with MPI on more than one node

When running on more than one machine (for example on a cluster **without** a batch system) you need to specify a list of hosts on which mpirun needs to spawn processes. In principle, this is implementation-specific and may be not required if the MPI is tightly integrated with your operating and/or batch system. For example for MPICH1 you can do this by preparing a file containing hostnames of the nodes (one per line) you will use in your parallel job. Then you set the SCM_MACHINEFILE environment variable pointing to the file.

When you submit a parallel job to a batch system the job scheduler usually provides a list of nodes allocated to the job. The $AMSBIN/start shell script has some logic to extract this information from the batch system and pass it to the MPI's launcher command (typically mpirun). In some cases, depending on your cluster configuration, this logic may fail. If this happens, you should examine the $AMSBIN/start file and edit the relevant portion of it. For example, you may need to add commands that process the batch system-provided nodelist or change mpirun's command-line options or even replace the mpirun command altogether.

6.6 IntelMPI and core-binding

IntelMPI by default uses core binding for the spawned processes (also known as process pinning). This can be disabled by setting the **I_MPI_PIN** environment variable to "off".

6.7 IntelMPI and SLURM

To get IntelMPI work under SLURM one needs to edit the $AMSBIN/start script and change the value of the I_MPI_PMI_LIBRARY environment variable to point to a correct libpmi library from SLURM. If your SLURM system is configured to use PMI2, then it could also be sufficient to comment out the I_MPI_PMI_LIBRARY line in the $AMSBIN/start script.

Depending on your version of SLURM, it might also be necessary to replace "mpirun -bootstrap slurm" with "srun" in the $AMSBIN/start file.

6.8 IntelMPI and SGE

To get IntelMPI working with Sun Grid Engine, one has to define a parallel environment. How this can be done is described on the intel website (https://software.intel.com/en-us/articles/integrating-intel-mpi-sge). It is important for modern versions of IntelMPI (as used in AMS2020) and newer to make sure to set "job_is_first_task FALSE" in the parallel environment, otherwise jobs will fail to start.

6.9 IntelMPI and ABI compatibility

IntelMPI v5.0 or newer is ABI (Application Binary Interface) compatible with Cray MPT v7.0.0 or newer and MPICH v3.1 and newer. This means that binaries compiled with one of these libraries can use the other ones during run-time

without problems. Our IntelMPI binaries should work out-of-the-box on Cray machines using the ABI compatibility, and can also be used in combination with MPICH 3.2.

To run ADF with MPICH instead of IntelMPI, simply **export SCM_USE_LOCAL_IMPI=true**, and make sure the MPICH mpirun command is available in your PATH variable. Core binding (process pinning) is disabled by default for MPICH, to enable this add "-bind-to core" to the mpirun commands in the $AMSBIN/start file.

6.10 Multi-node issues

A common reason for multi-node jobs failing where single-node jobs work, is a missing scratch folder on the secondary nodes. Especially SLURM systems are known to sometimes only create the TMPDIR folder on the first node of the job. To solve this, make sure that the SCM_TMPDIR exists before starting the AMS/ADF calculation. For example, under SLURM you could try "srun –ntasks-per-node=1 mkdir -p $SCM_TMPDIR" from your submit script, or even incorporate it into the $AMSBIN/start script.

6.11 OpenMPI on Linux

The OpenMPI 2.1.2 binaries supplied with AMS2020 should work on desktop, laptop and workstation machines out of the box (single-node usage). On cluster environments it might be necessary to compile an OpenMPI 2.1 library with support for the cluster queueing system and/or the infiniband solution. Make sure to **export SCM_USE_LOCAL_OMPI=true** before starting programs to enable your local OpenMPI version instead of the one shipped with ADF. Core binding (process pinning) is enabled by default for OpenMPI, to disable this add "–bind-to none" to the mpirun commands in the $AMSBIN/start file.

6.12 Corrupted License File

You may find that, after having installed the license file, the program still does not run and prints a message "LICENSE CORRUPT". There are a few possible causes. To explain how this error may come about, and how you overcome it, a few words on license files.

Each license file consists of pairs of lines. The first of each pair is text that states in a human-readable format a couple of typical aspects: A 'feature' that you are allowed to use (for instance 'ADF'), the expiration date, a (maximum) release (version) number of the software and so on. The second line contains the same information in encrypted format: a long string of characters that appear to make little sense. The program reads the license file and checks, with its internal encrypting formulas, that the two lines match. If not, it stops and prints a "LICENSE CORRUPT" message.

So, there are two common reasons why this may happen:

You can use the **fixlic** utility to try to fix this automatically. Please be aware that the **fixlic** utility will try to fix the file pointed to by the $SCMLICENSE environment variable and replace it with the fixed copy. Thus, you need to make a backup of your license file first and you need to have write permissions for it.

```
cp $SCMLICENSE $SCMLICENSE.backup
$AMSBIN/fixlic
```

6.13 Windows: running jobs from the command line

In order to run ADF or any other program from the package without the GUI, navigate to the ADF installation directory and double click the **adf_command_file.bat** file. It will start a Windows command interpreter and set up the environment specific for that installation of ADF. Once it has started, cd to your jobs directory by entering the following commands at the prompt:

```
C:
cd \ADF_DATA
```

Then, run your job as follows (assuming the job is called h2o):

```
sh h2o.job
```

You can also prepare a job from a .ams file and run it using only two commands:

```
sh amsprep -t h2o.ams -j h2o > h2o.job
sh h2o.job
```

Please note that you do need to use *sh* in the commands above because both h2o.job and amsprep are shell scripts and, thus, they must be interpreted by a shell.

If you are comfortable with a UNIX shell environment, you can also start a bash shell and enjoy a basic msys2 LINUX environment:

```
bash
```

CHAPTER

SEVEN

USING THE GUI ON A REMOTE MACHINE

Running the ADF GUI (amsinput) on a remote machine (X forwarding over SSH) can be tricky sometimes. This page will try to explain why and might contain some hints into how to get a remote GUI working. If your remote GUI worked fine with ADF2016 but stopped working with ADF2017, you can probably skip most of this page and read the last section about *ADF2017 and VTK7* (page 36).

7.1 X11 over SSH

When connecting to a remote system (let's call it RemoteBox) from your local machine (LocalBox) over SSH, there is the option to enable X forwarding:

```
ssh -X -Y user@RemoteBox
```

This allows you to run X11 GUI programs on RemoteBox while the window shows on LocalBox. You can try some simple X11 programs now:

```
xterm
xclock
```

If you got a terminal and a clock popping up in separate windows, you have working X11 forwarding over SSH.

If you got any errors and no window, most likely something fundamental is broken. Check if the RemoteBox allows X11 Forwarding (/etc/ssh/sshd_config should contain "X11Forwarding yes"), and try both the -X and -Y flag on the ssh command. Another thing to check is "xhost", which might also get in the way.

sidenote on ssh -X and -Y: There are two ways of enabling X forwarding with SSH, -X and -Y. The default -X flag enables X11 forwarding, but with extended security restrictions to protect LocalBox from malicious behavior of people on the RemoteBox. The -Y flag enables trusted X11 forwarding, which does not enable the additional security extensions. The security extensions are there for good reason, but they can break so many things that some distros (Debian for example) disable them by default, even if you use -X and not -Y. Which mode you decide to use is up to you of course, but if something doesn't work always try using -X -Y (or -XY) before asking for help.

Attention: If you did not succeed in getting an xterm or xclock window to show, then you MUST solve that problem before trying to use 3D graphics. The rest of this text assumes you have a working X11 setup.

7.2 X11 with OpenGL over SSH (3D graphics)

X11 over SSH can also support OpenGL 3D graphics, using the GLX extensions. To test this run:

```
glxgears
```

It should pop up a window with 3 gears, which may or may not be spinning. To get more info about the 3D driver stack, run:

```
glxinfo |  grep -E " version| string| rendering|display"
```

If the above two commands produce errors (For example: `Error:  couldn't find RGB GLX visual or fbconfig` or `X Error of failed request:`) something fundamental is broken with the 3D setup on RemoteBox or LocalBox. You should check the 3D driver stack on **both** machines, and pay special attention to the libGL.so and libGLX*.so libraries. Make sure you can run `glxgears` and `glxinfo` on LocalBox before investigating the remote machine.

7.2.1 Intel Graphics (mesa)

If LocalBox has intel graphics, you might run into problems if RemoteBox uses proprietary hardware & drivers. First check which libGL.so is used on RemoteBox by logging in via SSH and running:

```
ldd `which glxinfo`
```

The output will contain a line similar to:

```
libGL.so.1 => /usr/lib/x86_64-linux-gnu/libGL.so.1 (0x00007f76cdcaf000)
```

/usr/lib/x86_64-linux-gnu/libGL.so.1 is most likely a symlink, so use `ls -l` to find its true destination:

```
ls -l /usr/lib/x86_64-linux-gnu/libGL.so.1
```

if this points to a proprietary graphics library (for example: `lrwxrwxrwx 1 root root 30 aug 22 13:24 /usr/lib/x86_64-linux-gnu/libGL.so.1 -> /usr/lib/nvidia-361/libGL.so.1`), you can try pre-loading the mesa version of the library on the remote machine. Try to locate the mesa libGL.so:

```
find /usr -iname "*libGL.so*" -exec ls -l {} \;
```

If we are lucky we spot the mesa libGL.so in the output. It may look something like this:

```
/usr/lib/x86_64-linux-gnu/mesa/libGL.so
```

If you found the mesa libGL.so, try to put it in LD_PRELOAD:

```
export LD_PRELOAD=/usr/lib/x86_64-linux-gnu/mesa/libGL.so
glxinfo
```

If the system was already using the mesa libGL.so, you can try to use indirect rendering by setting the following environment variable:

```
export LIBGL_ALWAYS_INDIRECT=1
```

Other useful environment variables can be:

```
export LIBGL_DEBUG=verbose
export LIBGL_ALWAYS_SOFTWARE=1
```

7.2.2 NVidia Graphics

There are probably two libGL implementations on LocalBox if it uses the NVidia proprietary drivers (or 4 if you also have 32bit libraries installed): the opensource "mesa" library (`libGL.so.1.2.0`, most likely in a "mesa" subdirectory) and the proprietary NVidia library that comes with the NVidia drivers. Run the following command to find the libGL libraries on your system:

```
find /usr -iname "*libGL.so*" -exec ls -l {} \;
```

Make sure that the `libGL.so` and `libGL.so.1` symlinks in the generic folders (/usr/lib/, /usr/lib64, /usr/lib/x86_64_linux_gnu) eventually point towards the proprietary library. You will need to have root permissions to change the symlinks.

Warning: Never run any commands as root (or with sudo) to change your system setup if you do not understand them. It is not hard to break a Linux installation when making mistakes as root, so make backups before you change something and double-check what you type when using sudo or the root account!

Another important library when running OpenGL over SSH with NVidia hardware is `libGLX.so`. Locate it on your system with the following command:

```
find /usr -iname "*libGLX*.so*" -exec ls -l {} \;
```

Make sure that there are symlinks to the proprietary library in a generic location (/usr/lib on ubuntu).

You can check if the correct `libGL.so` is being used by checking the dynamic library dependencies of glxinfo:

```
ldd `which glxinfo`
```

The reported `libGL.so` dependency is most likely a symlink, so use `ls -l` on it to find out where it points to.

libGL.so examples

A correct setup on CentOS 6 with NVidia drivers for example should look something like this:

```
# first 3 lines are the NVidia lib
# second set of 3 lines are the mesa lib
# last two lines are the generic symlinks that point towards the NVidia lib
-rwxr-xr-x 1 root root 1220472 apr  6 02:51 /usr/lib64/nvidia/libGL.so.352.93
lrwxrwxrwx 1 root root 15 aug 19 13:33 /usr/lib64/nvidia/libGL.so.1 -> libGL.so.352.93
lrwxrwxrwx 1 root root 15 aug 19 13:33 /usr/lib64/nvidia/libGL.so -> libGL.so.352.93
-rwxr-xr-x 1 root root 561640 mei 11 06:38 /usr/lib64/mesa/libGL.so.1.2.0
lrwxrwxrwx 1 root root 14 jun  6 13:40 /usr/lib64/mesa/libGL.so.1 -> libGL.so.1.2.0
lrwxrwxrwx 1 root root 14 jun  6 13:40 /usr/lib64/mesa/libGL.so -> libGL.so.1.2.0
lrwxrwxrwx 1 root root 15 aug 19 13:48 /usr/lib64/libGL.so -> nvidia/libGL.so
lrwxrwxrwx 1 root root 17 aug 19 13:48 /usr/lib64/libGL.so.1 -> nvidia/libGL.so.1
```

Another example of an 64bit ubuntu installation with NVidia drivers and 32bit libraries available:

```
-rw-r--r-- 1 root root 439972 jul 18 05:47 /usr/lib32/nvidia-361/libGL.so.1.0.0 #␣
↪32bit nvidia lib
lrwxrwxrwx 1 root root 10 aug  3 06:14 /usr/lib32/nvidia-361/libGL.so -> libGL.so.1
lrwxrwxrwx 1 root root 14 aug  3 06:14 /usr/lib32/nvidia-361/libGL.so.1 -> libGL.so.1.
↪0.0
lrwxrwxrwx 1 root root 10 jun 13  2013 /usr/lib32/libGL.so -> libGL.so.1 #generic␣
↪32bit symlink, points to the next line
```

(continues on next page)

(continued from previous page)

```
lrwxrwxrwx 1 root root 21 aug 22 13:23 /usr/lib32/libGL.so.1 -> nvidia-361/libGL.so.1
↪# generic 32bit symlink, points to nvidia
-rw-r--r-- 1 root root 448200 jul 22 09:53 /usr/lib/i386-linux-gnu/mesa/libGL.so.1.2.
↪0 # 32bit mesa lib
lrwxrwxrwx 1 root root 14 jul 22 09:53 /usr/lib/i386-linux-gnu/mesa/libGL.so.1 ->␣
↪libGL.so.1.2.0
-rw-r--r-- 1 root root 579760 jul 18 05:50 /usr/lib/nvidia-361/libGL.so.1.0.0 # 64bit␣
↪nvidia lib
lrwxrwxrwx 1 root root 10 aug  3 06:14 /usr/lib/nvidia-361/libGL.so -> libGL.so.1
lrwxrwxrwx 1 root root 14 aug  3 06:14 /usr/lib/nvidia-361/libGL.so.1 -> libGL.so.1.0.
↪0
-rw-r--r-- 1 root root 459392 jul 22 09:52 /usr/lib/x86_64-linux-gnu/mesa/libGL.so.1.
↪2.0 # 64 bit mesa lib
lrwxrwxrwx 1 root root 14 jul 22 09:52 /usr/lib/x86_64-linux-gnu/mesa/libGL.so ->␣
↪libGL.so.1.2.0
lrwxrwxrwx 1 root root 14 jul 22 09:52 /usr/lib/x86_64-linux-gnu/mesa/libGL.so.1 ->␣
↪libGL.so.1.2.0
lrwxrwxrwx 1 root root 10 aug 22 13:25 /usr/lib/x86_64-linux-gnu/libGL.so -> libGL.so.
↪1 # generic 64bit symlink, points to next line
lrwxrwxrwx 1 root root 30 aug 22 13:24 /usr/lib/x86_64-linux-gnu/libGL.so.1 -> /usr/
↪lib/nvidia-361/libGL.so.1 # generic 64bit symlink, points to nvidia
```

7.2.3 AMD Graphics

AMD GPUs are reasonably well supported by the latest versions of mesa (ubuntu 16.04), and installing the proprietary Catalyst driver is not always a good idea. If you have an older distro (CentOS 6) you can install the fglrx Catalyst drivers. As a rule of thumb run the following command first:

```
glxinfo |  grep -E " version| string| rendering|display"
```

If this reports an OpenGL core profile version string of 4.x, do not install Catalyst. If the OpenGL core profile version string says 3.0, then check on google if fglrx is safe to install for your distribution.

libGL.so examples

An example of a CentOS 6 setup with AMD drivers should look a bit like this:

```
# first 4 lines are the 32bit AMD lib and symlinks
# next 4 lines are the 64bit AMD lib and symlinks
# last line is the renamed 64bit mesa library (renamed by the AMD driver installation)
-rwxr-xr-x. 1 root root 612220 Dec 18  2015 /usr/lib/fglrx/fglrx-libGL.so.1.2
lrwxrwxrwx. 1 root root 19 Aug 30 08:53 /usr/lib/libGL.so -> /usr/lib/libGL.so.1
lrwxrwxrwx. 1 root root 21 Aug 30 08:53 /usr/lib/libGL.so.1 -> /usr/lib/libGL.so.1.2
lrwxrwxrwx. 1 root root 33 Aug 30 08:53 /usr/lib/libGL.so.1.2 -> /usr/lib/fglrx/fglrx-
↪libGL.so.1.2
-rwxr-xr-x. 1 root root 921928 Dec 18  2015 /usr/lib64/fglrx/fglrx-libGL.so.1.2
lrwxrwxrwx. 1 root root 21 Aug 30 08:53 /usr/lib64/libGL.so -> /usr/lib64/libGL.so.1
lrwxrwxrwx. 1 root root 23 Aug 30 08:53 /usr/lib64/libGL.so.1 -> /usr/lib64/libGL.so.
↪1.2
lrwxrwxrwx. 1 root root 35 Aug 30 08:53 /usr/lib64/libGL.so.1.2 -> /usr/lib64/fglrx/
↪fglrx-libGL.so.1.2
-rwxr-xr-x. 1 root root 561640 May 11 06:38 /usr/lib64/FGL.renamed.libGL.so.1.2.0
```

7.3 OpenGL direct or indirect rendering

OpenGL with X11 can run in two different modes: direct rendering and indirect rendering. The difference between them is that indirect rendering uses the GLX protocol to relay the OpenGL commands from the program to the hardware, which limits OpenGL capabilities to OpenGL 1.4.

When using OpenGL over X11 with SSH, quite often direct rendering is not available and you have to use indirect rendering. Unfortunately indirect rendering is not always secure, so it got disabled by default on many recent Linux Distros. If you are using the mesa `libGL.so`, you can run the following commands to test if indirect rendering is working on LocalBox:

```
glxinfo
export LIBGL_ALWAYS_INDIRECT=1
glxinfo
```

If glxinfo crashes with something like:

```
name of display: :0
X Error of failed request:  GLXBadContext
  Major opcode of failed request:  154 (GLX)
  Minor opcode of failed request:  6 (X_GLXIsDirect)
  Serial number of failed request:  34
  Current serial number in output stream:  33
```

after setting LIBGL_ALWAYS_INDIRECT=1, then you might need to enable indirect rendering on LocalBox.

7.3.1 enabling indirect rendering on Xorg 1.17 and newer

X 1.17 disables indirect rendering by default. Enabling it can be a bit tricky, because the xorg.conf flag is only available in 1.19 and newer, so you need to use the **+iglx** command line flag when starting the X server.

> **Warning:** Be careful when making changes as root! Running these commands is at your own risk, and you should not execute them if you do not understand what they do.

CentOS 6

The X server call is hardcoded in gdm, so we need to create a wrapper script around Xorg. Log in as root on a console (Ctrl+Alt+F1) or via SSH and do:

```
cd /usr/bin/
mv Xorg Xorg.bin
echo -e '#!/bin/sh\nexec /usr/bin/Xorg.bin +iglx "$@"' > Xorg
chmod +x Xorg
init 3 # this stops the X server
init 5 # this starts the X server again
```

Ubuntu 16.04

```
sudo nano /usr/share/lightdm/lightdm.conf.d/50-xserver-command.conf
```

change the last line from `xserver-command=X -core` to `xserver-command=X -core +iglx` restart the machine, or restart the X server with:

```
sudo service lightdm restart
```

OSX / MacOS

Mac OS X users who update to XQuartz 2.7.9 will discover that they cannot use GLX applications remotely any more. This includes the ADF-GUI. To solve, at this point in time: stick to the older version of XQuartz (2.7.8), or install the 2.17.10 beta version: https://www.xquartz.org/releases/XQuartz-2.7.10_beta2.html. After installing they should run this command to enable GLX:

```
defaults write org.macosforge.xquartz.X11 enable_iglx -bool true
```

7.4 OpenGL2+ with X11 over SSH

If you need OpenGL2+ features, there are two options: use direct rendering (this usually only works if both LocalBox and RemoteBox use a recent mesa libGL.so), or use `VirtualGL`. The VirtualGL tool (http://www.virtualgl.org/) intercepts the OpenGL calls, does the 3D rendering on RemoteBox and then transports the resulting image to LocalBox. For more details on how this is achieved see their documentation (https://cdn.rawgit.com/VirtualGL/virtualgl/2.5/doc/index.html).

Install VirtualGL on LocalBox and RemoteBox, and configure (run `vglserver_config` as root, see the VirtualGL documentation (https://cdn.rawgit.com/VirtualGL/virtualgl/2.5/doc/index.html#hd006%20to%20setup%20RemoteBox)) RemoteBox to operate as a VirtualGL server. RemoteBox should of course also have proper 3D hardware and drivers (intel integrated graphics with recent mesa drivers, or a dedicated AMD/NVidia GPU), and be capable of indirect rendering (see above).

Once virtualGL is installed and set up on RemoteBox and installed on LocalBox, open a terminal window on LocalBox and connect to RemoteBox with:

```
vglconnect -s username@RemoteBox
```

This will start a VirtualGL client on LocalBox and set up two encrypted tunnels to RemoteBox. On RemoteBox you can now run OpenGL programs by starting them through `vglrun`:

```
vglrun glxinfo
vglrun glxgears
```

If you look at the output of the glxinfo command when running through vglrun, you will see that both the server and client glx vendor string changed into "VirtualGL", and the OpenGL renderer & version string should now say something about the hardware of RemoteBox.

7.5 ADF2017 and VTK7

ADF2017 uses VTK7 and newer OpenGL features to dramatically speed up the visualization of large systems. This comes with a higher requirement for the OpenGL version supported by the system: OpenGL 3.2. If your machine does not support this, you might get the following error message when starting the GUI:

```
Warning: In /home/builder/jenkins/workspace/trunk/label/centos6_impi_lxc/VTK/
↪Rendering/OpenGL2/vtkOpenGLRenderWindow.cxx, line 647
vtkXOpenGLRenderWindow (0x7b93c40): VTK is designed to work with OpenGL version 3.2␣
↪but it appears it has been given a context that does not support 3.2. VTK will run␣
↪in a compatibility mode designed to work with earlier versions of OpenGL but some␣
↪features may not work.
```

In such cases, a fallback mode is available that lowers the OpenGL requirement to version 1.4, but this fallback mode of course does not have the performance benefits of the newer OpenGL features. To enable the fallback mode, set the `SCM_OPENGL1_FALLBACK` environment variable to something non-zero:

```
export SCM_OPENGL1_FALLBACK=1
amsinput &
```

The OpenGL 3.2 requirement should not present any problems, unless your hardware or OS is really old. Known problematic cases are:

- CentOS 6 with intel graphics has OpenGL 2.1 (possible solutions: get an NVidia or AMD GPU with closed-source drivers, or use the fallback mode)
- OpenGL with X11 over SSH (possible solutions: use VirtualGL (see *OpenGL2+ with X11 over SSH* (page 36)) or the fallback mode)
- Remote Desktop on Windows: The 32bit Windows version of ADF2017.107+ has been made OpenGL 1.4 compatible to deal with older hardware and Remote Desktop problems. You can try to install the 32bit version of ADF2017 on your Windows machine if you have problems with starting the GUI on Windows.

7.6 AMS2019/AMS2018 and VTK7

AMS2019 uses the same VTK7 as AMS2018 and ADF2017, and thus also requires OpenGL 3.2 or newer. However, the OpenGL 1.4 fallback mode is now available on 64bit Windows and Linux, and should trigger automatically when it detects an OpenGL version on the system that is too old for the normal mode. The `SCM_OPENGL1_FALLBACK` environment variable can also still be used, for example when the detection is not working. Setting this on Windows can be done with the following steps: `Winkey+R`, `sysdm.cpl`, Advanced, Environment Variables... Here you can either add the `SCM_OPENGL1_FALLBACK` key with value 1 to the current user, or as admin for everyone on the system.

7.7 AMS2020 and VTK7

AMS2020 ships a software implementation of OpenGL on Linux and Windows. On Windows this is automatically enabled if the system has no support for OpenGL 3.2 or newer. The `SCM_OPENGL_SOFTWARE` environment variable can be set to use this on Linux, or force it on Windows. The `SCM_OPENGL1_FALLBACK` environment variable is no longer available.

7.8 Sources

The information on this page is a mashup of local knowledge, a lot of testing and reading on the web. The following pages contained some useful information:

- https://www.phoronix.com/scan.php?page=news_item&px=Xorg-IGLX-Potential-Bye-Bye
- https://wiki.archlinux.org/index.php/VirtualGL

- https://cdn.rawgit.com/VirtualGL/virtualgl/2.5/doc/index.html
- https://en.wikipedia.org/wiki/Direct_Rendering_Infrastructure
- https://unix.stackexchange.com/a/1441

CHAPTER

EIGHT

APPENDIX A. ENVIRONMENT VARIABLES

If you start the MacOSX ADF-GUI application the environment defined by your shell startup scripts is ignored. Instead the bundled ADF is used, and environment variables may be defined in a file $HOME/.scmenv .

The following environment varables must always be set for all ADF versions.

AMSHOME: full path of the ADF installation directory, for example, $HOME/ams2020.101 .

AMSBIN: full path ADF directory with binaries, typically set to $AMSHOME/bin .

AMSRESOURCES: full path of the directory with the ADF database (basis sets and so on), typically set to $AMSHOME/atomicdata

SCMLICENSE: full path-name of the license file, for example $AMSHOME/license.txt.

SCM_DOMAINCHECK: set to yes if you have a license based on your domain info (DNS). If it is defined but no proper DNS sever is available, delays will often occur.

SCM_TMPDIR: full path of the directory where all processes will create their temporary files. See also the Installation manual and the special section on SCM_TMPDIR below.

The following environment variable may be required at run-time by a parallel version.

NSCM: The number of processes to be used in a particular calculation. This variable should only be set per job. Do **not** add any NSCM definitions to your shell resource files. Please note that the NSCM value is typically ignored in job submitted through a batch system because then the number of processors is determined by the batch job's properties.

SCM_MACHINEFILE full path-name of the file containing a list nodes to use for a job; **Important:** this variable should **only** be set if multiple computers are used without any batch system and then it should be set on the per-job basis. The file pointed to by the variable must already exist and it must contain a list of nodes on which a particular job is about to be executed, possibly with their processor count.

SCM_USE_LOCAL_IMPI this applies only to IntelMPI distributions. Setting this environment variable to not be empty will disable the IntelMPI runtime environment shipped with the adf distribution, allowing the use of a local IntelMPI installation, or any other ABI-compatible local MPI installation (MPICH v3.1 or newer for example). The environment must be properly set up on the machine, meaning I_MPI_ROOT must be set, and mpirun should be in the PATH, and the libraries must be in LD_LIBRARY_PATH.

SCM_USE_LOCAL_OMPI this applies only to OpenMPI distributions. Setting this environment variable to not be empty will disable the OpenMPI runtime environment shipped with the adf distribution, allowing the use of a local OpenMPI 2.0 installation. The environment must be properly set up on the machine, meaning OPAL_PREFIX must be set, and mpirun should be in the PATH, and the libraries must be in LD_LIBRARY_PATH.

SCM_PYTHONDIR: This environment variable specifies where the AMS python distribution will search for a python virtual environment to use. If no suitable virtual environment exists in this directory, it will be created. To not use a virtual environment, set SCM_PYTHONDIR to an empty string.

SCM_PYTHONPATH: The contents of this environment variable gets prepended to the PYTHONPATH variable when using the AMS python distribution. You can add the paths to other python modules that you wish to use in combination with the shipped python distribution to this variable to make them available.

The following environment variables are relevant for the GUI modules. For a full list, see the GUI Environment Variables page.

SCM_ERROR_MAIL: e-mail address for error reports

SCM_GUIRC: location of the preferences file, by default $HOME/.scm_guirc

SCM_GUIPREFSDIR: location of the preferences folder, by default $HOME/.scm_gui/ (available since ADF2013.01b)

SCM_TPLDIR: location of the templates directory, by default no extra templates are loaded

SCM_STRUCTURES: location of the structures directory, by default no extra structures are loaded

SCM_RESULTDIR: location of the results directory, by default the current directory used

DISPLAY: specifies the X-window display to use and is required for all X11 programs on Linux/Unix and Mac OS X. On MaC OS X you should typically not set it as it will be set automatically. Setting it will break this.

SCM_QUEUES: path to the dynamic queues directory, by default AMSjobs will search the remote $HOME/.scmgui file

SCM_OPENGL_SOFTWARE: Linux and Windows (available since AMS2020). If set to be non-empty, the GUI will use a software OpenGL implementation. See Using the GUI on a remote machine for more information.

SCM_OPENGL1_FALLBACK: No longer used since AMS2019.305 (applies from ADF2017 until AMS2019.304), see SCM_OPENGL_SOFTWARE and Using the GUI on a remote machine for more information.

The AMS driver environment variables can be found in the AMS Documentation

The following environment variables are relevant for source distributions only, and only at the configure time.

MPIDIR and MATHDIR: see Compiling ADF from Sources

The following environment variables may be set to modify other aspects of ADF execution. All of them are optional and some are used for debugging only.

SCM_GPUENABLED Envrionment flag to turn GPU acceleration on or off. Only works for the CUDA-enabled binaries. Setting this variable to TRUE turns GPU acceleration on, setting it to FALSE turns it off. If the input contains the keyblock GPUENABLED, a FALSE in the environment variable will be ignored.

SCM_MAXCOMMLENGTH When performing collective MPI communications ADF breaks them into pieces of at most SCM_MAXCOMMLENGTH elements, each element being 4 or 8 bytes long. So in practice the largest collective MPI operation should not exceed SCM_MAXCOMMLENGTH*8 bytes in size. By default, SCM_MAXCOMMLENGTH is set to 268435455 on Linux and 4194304 on Windows and macOS. When the SCM_MAXCOMMLENGTH environment variable is set its value is used instead.

SCM_VECTORLENGTH Almost all programs within the ADF package use numerical integration, and this is normally the most time-consuming part of the code. Numerical integration involves summing together results for each 'integration point'. The best performance is achieved when handling a number of points at the same time. The number of integration points handled together is called the block length or the vector length. If the block length is too small, you will have a significant overhead and the programs may become very slow. If the block length is too large, lots of data will not fit in the processor cache and again the program will not run at the maximum speed. The optimal block length is somewhere in between, ranging from 32 to 4096 depending on your hardware. Sometimes it pays off to set the block length explicitly NOT to a power of 2 to avoid memory bank conflicts. Again, try it yourself with your typical calculation on your production machine to find out the optimal value for your situation. On most machines, the default 128 is a good value.

SCM_SHAR_EXCEPTIONS: setting this variable to "*" disables the use of shared arrays.

SCM_SHAR_LIMIT: sets the limit on the total size of shared arrays in megabytes. The default is a bit less than half of the node's total RAM. If a new shared array would cause the total amount of shared memory to go over the limit, then instead of placing the array into shared memory it is created as a shared file in scratch directory of the node-master. It then gets memory-mapped into the address space of all processes on the node. The effect will be the same as when the array is placed into shared memory except that there will be a delay due to disk I/O when the array is destroyed.

SCM_DEBUG: setting this to a non-empty string will cause each MPI rank to print values of relevant environment variables and some messages about copying files to/from SCM_TMPDIR.

SCM_NOMEMCHECK: setting this to a non-empty string disables checks on memory allocation failures. The usefulness of this variable is questionable.

SCM_NODOMAINCHECK: setting this to a non-empty string disables DNS requests when verifying the license. Use this variable if you experience long delays at the start of each calculation.

SCM_TRACETIMER: setting this to a non-empty string will produce additional output on entry/exit to/from internal timers.

SCM_DEBUG_ALL: setting this to yes is equivalent to specifying DEBUG $ALL in the input

8.1 More on the SCM_TMPDIR variable

Below we will explain in more detail how does the SCM_TMPDIR environment work. Every parallel job consists of one master and one or more slave tasks. Master and slaves behave a bit differently with respect to their scratch directories.

Slave processes

Slave processes will always create a directory for their scratch files in $SCM_TMPDIR and *chdir* to it to avoid any risk that shared files are updated by more that one process at the same time. For efficiency reasons, that directory should reside on a local disk unless you are using very, very fast shared file system for large files. You need write access to that directory, and the file system should have enough free space. Please note that the SCM_TMPDIR environment variable will be passed from the master to slaves. After the job is finished, slave processes will delete their scratch directories. This can disabled by setting the SCM_DEBUG environment variable to any text, for example, to "yes". In this case the scratch directory and all its contents will be left intact. This directory will also be left behind when a job has crashed or has been killed. Each slave writes its text output to a file called KidOutput located in its scratch directory. In case of an error this file will likely contain some sensible error message. If an error occurs and a slave process exits in a controllable way then in order to avoid losing the file ADF will copy the file to the directory, from which the job was started, as KidOutput__#, where # is the process' rank.

Master process or serial runs

The master process (which is the only process in a serial run) will also create its temporary files in its own sub-directory of $SCM_TMPDIR. There are some exceptions. Some files, such as logfile and TAPE13, will be created in the directory where ADF was started because they are not performance-critical but are convenient to have in the current directory for different reasons. For example, logfile is nice to have in the current directory in order to follow the calculation progress and the TAPE13 is an emergency restart file that can be used if ADF crashes or is killed. At the end of a calculation, the master will copy all result files from its scratch directory to the directory where it was started.

Using multiple scratch disks

It is possible to use multiple physical scratch disks in a parallel calculation. See the Installation manual for more information about this.

CHAPTER

NINE

APPENDIX B. DIRECTORY STRUCTURE OF THE AMS PACKAGE

Below is the list of major parts of the AMS package.

9.1 The bin directory

When the package is installed, the executable files are placed in $AMSHOME/bin. The binary executable file is usually called 'ams.exe', 'reaxff.exe', and so on. On Windows it is ams.3.exe, reaxff.3.exe, etc. There will also be files called just 'ams' or 'reaxff'. The latter are shell scripts that execute the corresponding binaries.

You should use the script files, rather than the executables directly. The script files provide a correct environment, and if needed prepare for running in parallel and then launch the binary using the correct mpirun command. See also the sample run scripts and the Examples document.

The $AMSBIN/setenv.sh and $AMSBIN/start scripts take care of setting up the environment and starting the binaries. If necessary, it parses the information provided by a batch system and sets up a machinefile (or an appfile) and runs tasks in parallel. Edit the file if you need to customize the way MPI programs are started.

9.2 The atomicdata directory

The directory atomicdata/ contains a large amount of data needed by programs from the AMS package at run time. For example, it contains basis sets for all atoms. Generally you should **not** modify any of the files in this directory.

The basis sets are described in detail in the ADF manual and BAND manual

9.3 The examples directory

The directory examples/ contains sample script and output files. The files with extension .run are shell scripts that also contain embedded input. Thus, these files should be executed, and not given as input to AMS.

The example calculations are documented in the ADF Examples documentation, and BAND Examples documentation.

9.4 The src directory

The source files are only visible if you have a copy of the source code distribution. The source code files found in the program and library directories and subdirectories thereof have extensions .f, .f90, .c or .cu and contain FORTAN or C/CUDA code. Other source files are include files (files with extension .fh or .h). When compiling, the object files are generated in the folder $AMSHOME/build, and archived into libraries.

Compilation is done by $AMSBIN/foray, and the configure options are located in $AMSHOME/Install/-machinetype-/Forayflags.

The Install directory contains a configure script, some data files which provide generic input for configure (start, starttcl, and some more), a portable preprocessor cpp (based on Mouse cpp) and machine-specific files that are unpacked into the bin folder (precompiled packages), or the build folder (precompiled libraries). The machine-specific folders start with x86 or x86_64.

9.5 The Doc directory

All the user documentation for AMS is present in html format in $AMSBIN/Doc. Documentation is also available on the SCM website (http://www.scm.com/support)

9.6 The scripting directory

This directory contains some useful scripts that are part of the AMS package.

CHAPTER

TEN

APPENDIX C. DEBUGGING MPI PROBLEMS

The Amsterdam Modelling Suite (AMS) is designed to work out of the box on as many platforms as possible, but it could be that you are experiencing problems with running our programs in parallel. This mostly happens on linux cluster environments, but the information below could also be useful for Windows or MacOS users. However, anything that mentions cluster environment or batch system is linux specific!

10.1 Technical introduction into AMS

Most of our compute programs use Message Passing Interface (MPI) communication for parallel operation. To make AMS as easy to use as possible we ship the MPI runtime libraries within the AMS package, and take care of launching of the MPI programs in parallel in our start script. Because of this setup it is not needed to "mpirun" our programs yourself, so **never do something like ''mpirun $AMSBIN/ams''** !

10.1.1 Execution order

When starting one of our programs, for example *$AMSBIN/ams*, you actually call our start script via a symbolic link. This start script then:

- checks which program it should launch, based on the name it was called from or the value of the -x flag
- detects how many cores should be used: -n or the NSCM environment variable are read, when both are empty we detect the cpu core count (see below for cluster environments!)
- sources the *$AMSBIN/setenv.sh* script to set up environment variables
- Write the given input to a temporary file
- detect if and which batch system it is running on
- do the correct parallel MPI launch of the program

`Old pre-AMS2020 info:` When running *$AMSBIN/adf*, this sequence is preceded by a program that first sets up and runs the "atomic create runs" in serial.

10.1.2 The $AMSBIN/start script

As described above, the start script takes care of selecting the number of cores to use for the calculation, sourcing the setenv.sh script, and launching the selected program in parallel. You might need to edit it if your AMS program fails to start, or does not run on the correct number of cores.

The start script tries to detect if it is running on a cluster environment with a batch system such as PBS/Torque, SLURM, SGE or LSB. If it detects one of the environment variables associated with these batch systems, it will leave figuring out the MPI job configuration (cores and nodes) to the MPI library. The MPI library should get the number

of cores to run on, and which compute nodes to use (in case of a multi-node calculation) from the batch system. the **NSCM** environment variable and the **-n** flag are ignored in this case!

If no batch system is detected, and both **NSCM** and **-n** are not used to explicitly set the number of cores to use for running the AMS program, then the start script attempts to detect the number of cores on the machine, and uses all available cores. We only count real CPU cores (no hyperthreading or modules with shared FPUs such as AMD Bulldozer), because the AMS programs do not benefit from hyperthreading.

Finally the start script launches the AMS program using the correct mpi commands for the situation.

10.1.3 The $AMSBIN/setenv.sh script

The *$AMSBIN/setenv.sh* script takes care of setting up environment variables needed for running the AMS programs. This includes among others:

- the LD_LIBRARY_PATH variable with locations to the required libraries (and DYLD_LIBRARY_PATH)
- the PYTHONPATH variable

The setenv script also responds to a couple of environment variables, such as **SCM_USE_LOCAL_IMPI** and **SCM_USE_LOCAL_OMPI**. See Appendix A on Environment Variables for details.

10.1.4 MPI runtimes

The Amsterdam Modelling Suite ships with the required MPI runtime inside the package. For MacOS this is OpenMPI, for Windows this is IntelMPI, and for Linux both IntelMPI and OpenMPI flavors exist. When working on Linux we advise to use the IntelMPI library, as this provides the best out-of-the-box experience. The OpenMPI runtime has no batch system integration build into it, so it will most likely not work under a batch system. The runtimes can be found in *$AMSBIN/intelmpi* or *$AMSBIN/openmpi*, and are loaded into the environment by the setenv.sh script. See the Additional information section for more details.

10.2 Debugging MPI issues

For non-batch system (desktop/laptop usage) related MPI issues, please contact support@scm.com as these should almost never happen.

For cluster usage we always advise to use the IntelMPI version. AMS ships with the IntelMPI runtime inside the package, you do not need to install this separately. IntelMPI can be used on Cray systems as well via the ABI compatibiliy, see the Additional information section for more details. If you for whatever reason need to use OpenMPI instead, make sure to compile and use your own OpenMPI version that matches the version used to build AMS (currently 2.1.2), and set SCM_USE_LOCAL_OMPI=true in the environment. Make sure to to read the Additional information and known issues document before you dive into debugging MPI!

The generic scheme for debugging IntelMPI / batch system issues consists of the following steps:

1. Start by setting up the IntelMPI runtime libaries in the environment: load it from a module, or load your AMS environment and execute ". $AMSBIN/setenv.sh" without the quotes.
2. get the intel mpirun to work. Environment variables that are useful for this are I_MPI_DEBUG=100 (or higher), I_MPI_OFI_PROVIDER_DUMP=1 (for information about the interconnects detected by IntelMPI), the I_MPI_FABRICS (for selecting a specifc fabric, options depend on the IntelMPI runtime version) and I_MPI_HYDRA_TOPOLIB. See the IntelMPI developer reference manual (https://software.intel.com/en-us/download/intel-mpi-library-for-linux-os-developer-reference-2018-update-3) for details on these variables.

3. get a small mpi test program that's compiled with IntelMPI and a Fortran compiler. To make things a bit easier you can download an example with both binary and source code here (https://downloads.scm.com/distr/MPItest.tgz).
4. try to mpirun the test program (mpirun ./mpitest) on a single node via the batch system. This usually respects the batch system reservation. In case it doesn't you will need to dive into the manuals to see what kind of environment variables you need to set, or arguments you need to give to get it to integrate with the batch system. IntelMPI developer references and guides (https://software.intel.com/en-us/articles/intel-mpi-library-documentation-overview) help out with this, but make sure to get the manual matching your IntelMPI version! The manual of your batch system might also hold some clues.
5. once single node jobs work, then it is time to see what needs to be done for multi-node jobs. Usually this can be set up via integration with the batch system, but if that fails you could always write a bit of code that generates a "machine file", and tell IntelMPI to use this file with the -machinefile flag (see this (https://software.intel.com/en-us/mpi-developer-guide-linux-controlling-process-placement) for how a machine file should look)

It might be that the IntelMPI runtime shipped with AMS (currently 2018 update 3) needs to be updated for a better integration with your batch system. In such a case you can download and install a newer IntelMPI runtime package (those are free to use), and repeat the previous steps. If this works better than the 2018 runtime shipped with AMS, then you can set the "export SCM_USE_LOCAL_IMPI=true" environment variable. This tells AMS not to load the distributed IntelMPI into the environment, but instead use the one already available. (You do not need to recompile AMS for this!)

Finally: when requesting support for MPI issues via support@scm.com, make sure to include as much information as possible. For example: which batch system are you using, what version is it, do you have a special interconnect such as Infiniband, and of course always send us all the input and output files produced by the failed job, including the sdtout and stderr of the batch system. From our experience the fastest way to resolve such issues is if somebody from SCM can work directly on the machine. Therefore you might want to consider setting up a form of temporary remote access for an SCM employee to help you out.

CHAPTER

ELEVEN

COMPILING AMS FROM SOURCES

THIS INFO IS ONLY RELEVANT FOR PEOPLE WITH A SOURCE CODE LICENSE! If you are unsure what this means, you can most likely skip reading this page!

Compiling AMS from sources by end users is not supported on Windows. The following instructions apply to Linux/Unix and Mac OS X only. Compiling AMS2019 from sources is supported for ifort version 18.0.2 with MKL and IntelMPI on linux.

11.1 Unpacking the distribution

Installing AMS with recompilation in mind is somewhat different from the binary-only installation. The downloaded source and binary tarballs must be unpacked in the following order (using IntelMPI on x86_64-linux in this example):

```
# First sources
tar xzf ams2020.101.src.tgz
# Then binaries
tar xzf ams2020.101.pc64_linux.intelmpi.bin.tgz
```

The result will be a ams2020.101 directory containing both binaries (for your platform) and sources.

Note that for Mac OS X, the downloading of the binaries is different. Follow the instructions for downloading and installation of the binaries. Copy the binaries from the downladed disk image to the directory ams2020.101 that was created from the source code. Depending oin where you have put the binaries it could be something like:

```
cp -r /Applications/AMS2020.101.app/Contents/MacOS/AMS.app/Contents/Resources/amshome/
↪* ams2020.101
```

11.2 Setting up environment

In addition to the standard environment variables discussed in the Installation manual, you may need to set additional ones depending on your platform:

- I_MPI_ROOT: this variable must be set if compiling with IntelMPI on linux (the default)
- MPIDIR: may be needed in the case of compiling AMS with non-default MPI, for example OpenMPI on linux.
- MATHDIR: this should be set to the the MKL root folder. If MKLROOT is defined and MATHDIR is not, then MKLROOT will be used.

11.3 Running Install/configure

After unpacking everything and setting up your environment properly, you need to run the *configure* script. This script is located in the $AMSHOME/Install directory, and it must be executed from the $AMSHOME directory. The script replaces some files in the bin directory with versions specifically targeted for your system. Further, *configure* creates the buildinfo file that you will need to compile AMS.

To see what options the configure script supports, use configure -h:

Example:

```
cd $AMSHOME
Install/configure -h
```

Configure can set up multiple build targets with different configuration options using the -b flag. The options regarding your build target should be wrapped in quotes following a -b flag, starting with the build name. The -b flag can be used multiple times to create different build targets. For example, to create a target with all current release build options:

```
cd $AMSHOME
Install/configure -b "release -p intelmpi -meadshared -dynamicmkl -plumed"
```

If a different MPI version is needed (for example OpenMPI) as well as a defaults target, simply run:

```
cd $AMSHOME
Install/configure -b "mydefaulttarget" -b "myompitarget -p openmpi"
```

11.4 Compiling AMS

Next, you need to compile the sources by executing foray located in $AMSBIN. Foray supports parallel compilation to make things faster, use -j N to tell foray you want it to use N processes (for example: -j 4 on a quadcore machine):

```
cd $AMSHOME
bin/foray -j 4
```

After a while, depending on the speed of your computer, everything should be ready to use, just as if you had installed the precompiled executables but now with your modifications included. Use *bin/foray -h* to get a list of all foray options.