

# Installation of 3<sup>rd</sup>-Party Software

This is a hands-on manual how to install the necessary software used for different model reduction approaches.

For my method, you need at least

- LAPACK
- UMFPACK
- CPPAD and
- OpenMP.

If, in addition, optimization is involved, please install

- IPOPT and
- MUMPS.

If you decide to run your applications on distributed systems, you need to install as well

- DUNE and
- MPI.

In case you are running a Linux machine, some of the aforementioned tools should already be available for your distribution. If not, you may follow the installation instructions below.

## Getting and using LAPACK

Suppose you're using  / -Linux<sup>1</sup>.

- ❶ Get the ATLAS packages:

```
# aptitude install libatlas3gf-base libatlas-base-dev
libatlas3gf-sse2 libatlas-sse2-dev libatlas-doc libatlas-test
libatlas-headers
```

- ❷ Get the BLAS packages:

```
# aptitude install libblas-dev libblas-doc libblas-test libblas3gf
```

- ❸ Get the LAPACK packages:

---

<sup>1</sup>cf. reference section if you're running a system which does not contain LAPACK and so forth.

```
# aptitude install liblapack-dev liblapack-doc liblapack-pic
liblapack3gf scalapack1-mpich2 scalapack-mpich-dev
```

⊛ Download the `-dev` files as well (they often contain new libraries or stuff needed for properly running linear algebra calculations!

ATLAS, BLAS and LAPACK are now installed in `/usr/lib` and `/usr/include`. You can change to the `/usr/lib` - directory and type `find liblapack.so.*` to check how the shared libraries are named (normally they should be called *liblapack.so.3gf*, *liblapack.so.3gf.0* and the like).

## A toy example

Assume you want to solve the following tridiagonal linear system

$$\begin{bmatrix} 2 & -1 & 0 & 0 & 0 \\ -1 & 2 & -1 & 0 & 0 \\ 0 & -2 & 3 & -1 & 0 \\ 0 & 0 & -1 & 3 & -2 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 2 \\ 1 \end{bmatrix}. \quad (1)$$

Clearly, the solution is given by

$$x^* = [x_0, x_1, x_2, x_3, x_4]^T = \begin{bmatrix} 6.5 \\ 12.0 \\ 15.5 \\ 19.5 \\ 20.5 \end{bmatrix}.$$

The linear algebraic system (1) may now be implemented in C++ with the help of LAPACK in a very simple manner



- Do NOT forget to specify an extern linkage convention via `extern "C">{...}`<sup>3</sup>.
- Do NOT forget the underscore '\_' after the FORTRAN function name inside the `extern "C"` environment (i.e. `dgtsv_(...)` and not only `dgtsv(...)`!)

**Note:** If you want to retain a once calculated solution of system (1) without being destroyed by the next call of *tridiagsyssolver*, simply add `static` in front of the return value of *tridiagsyssolver* (here it would read `static long tridiagsyssolver(...)`).

<sup>2</sup>This and the next one are quite optional and provide certain MPICH extensions.

<sup>3</sup>Otherwise an error occurs, like "error: ...dgtsv\_ was not declared in this scope" or "...(.text+0x9a): undefined reference to 'dgtsv\_(long const\*, long const\*, double\*, double\*, double\*, double\*, long const\*, long\*)' collect2: ld returned 1 exit status".

## How do I compile and link the program?

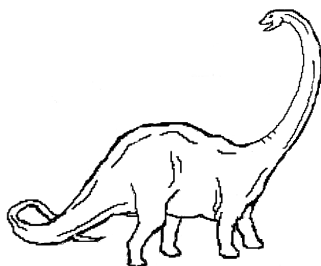
That's rather easy. Let's say the above program is called "simplelapack.cc". Just type

```
c++ -o simplelapack simplelapack.cc -llapack -lblas -lg2c -lm
```

Sometimes, especially in a Makefile one has to add `-L/usr/lib/liblapack.so.3gf` .

Any hidden mistakes? Run `valgrind --leak-check=full ./simplelapack`

As a matter of fact, the above programme is completely independent of the ATLAS libraries. If the optimized ATLAS versions are present it will use them, if they are absent the reference implementation will be used instead.



## Installation of Ipopt

- ❶ Download the Ipopt sources<sup>4</sup> from

<http://www.coin-or.org/download/source/Ipopt/>

- ❷ Create a new directory in your home directory, say

```
user@linux:~$ mkdir SOFTWARE
```

- ❸ Move the stuff to the newly created directory:

```
user@linux:~$ mv Ipopt-version.tgz SOFTWARE
```

where from here and now on *-version* denotes the version you have decided to download !

- ❹ Change to the newly created directory, extract the files from the downloaded file, change to the extracted directory *Ipopt-version*, then create directory *build* and change to this dir:

---

<sup>4</sup>Ipopt needs the extern linear solver MA27 and/or MA57. You have to register at <http://www.cse.clrc.ac.uk/nag/hsl/> in order to get them ! Once you've got 'em, copy the \*.f files into `/SOFTWARE/Ipopt-version/ThirdParty/HSL` and proceed as described above...

```

user@linux:~$ cd SOFTWARE
user@linux:~/SOFTWARE$ gunzip Ipoprt-version.tgz
user@linux:~/SOFTWARE$ tar xvf Ipoprt-version.tar
user@linux:~/SOFTWARE$ cd Ipoprt-version
user@linux:~/SOFTWARE/Ipoprt-version$ mkdir build; cd build

```

- ⑤ **Alternative download:** Get current version via svn:

```

svn co
https://projects.coin-or.org/svn/Ipoprt/stable/version _CoinIpoprt

```

- ⑥ For getting further external software which can be used in conjunction with IPOPT, change to *ThirdParty* and run in every subdirectory the `get.*` script.
- ⑦ Creating *build* is optional. Nevertheless they advise to do so and to compile the stuff in that separate subdirectory (Make sure the linear solvers MA27 and MC19 have been copied to `user@linux:~/SOFTWARE/Ipoprt-version/ThirdParty/HSL`):  
`../configure; make; make test; make install` and the installation can be found in `user@linux:~/SOFTWARE/Ipoprt-version/build`.  
However, since we want to install IPOPT **globally**, e.g. in `/usr/local/`, we do not change to *build*, but stay in the present directory (see prompt line in order to avoid confusion!). Next configure the stuff into the desired installation path and compile it via *make*:

```

user@linux:~/SOFTWARE/Ipoprt-version$ ./configure --prefix /usr/local/
--with-lapack="-L/usr/lib -llapack"
user@linux:~/SOFTWARE/Ipoprt-version$ make

```

The `./configure` script should terminate with Main configuration of Ipoprt successful

Note that the `-with-lapack` flag (and often also the corresponding `-with-blas="-L/usr/lib -latlas"` flag has to be set if you use your linux libraries instead of the 3<sup>rd</sup>-party versions provided by IPOPT .

- ⑧ Test compilation via

```

user@linux:~/SOFTWARE/Ipoprt-version$ make test

```

The output to screen should look somehow like the following:

```

Testing AMPL Solver Executable...
no AMPL solver executable found, skipping test...
Testing C++ Example...
Test passed!
Testing C Example...
Test passed!

```

Testing Fortran Example...

Test passed!

AMPL does not interest me because it is non-free ☺

⑨ Installation: `make install`

⑩ Add the following to your `./bashrc`:

```
export
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/home/mfein/SOFTWARE/CoinIpopt/build/lib/
```

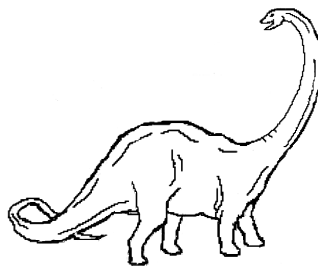
so that Ipopt's runtime libraries can be detected.

Now IPOPT has been installed globally and the source files and header files respectively can be found in `/usr/local/include/coin`. Binaries are in `/usr/local/lib`, e.g. `libipopt.so`, etc.

For later use you can always use the Ipopt makefile which is automatically delivered and lies in `~/SOFTWARE/Ipopt-version/Ipopt/examples/hs071_cpp`

**Uninstalling IPOPT :**

```
user@linux:~/SOFTWARE/Ipopt-version$ su <enter passwd>
user@linux:~/SOFTWARE/Ipopt-version$ make uninstall
```



## Installation of CppAD

① Download the sources from

<http://www.coin-or.org/CppAD/Doc/installunix.htm>

Here you can download the stuff under *Unix Tar Files*.

② Assume that you've already created a dir called *SOFTWARE* for 3rd Party Software. Furthermore, assume also you have created a dir called *CppAD* therein. Then simply copy the sources to there and extract them:

```
user@linux:~/SOFTWARE/CppAD$ tar -xvzf cppad-version.license.tgz
```

```
user@linux:~/SOFTWARE/CppAD$ cd cppad-version
```

(You can see if everything o.k. by checking `cppad-version/cppad/cppad.hpp`)

- ③ Also possible: If you don't want to install CPPAD simply copy the dir *cppad* to your desired location and, later on, include it in your makefile via *-I./* (assumed *cppad* is a subdirectory therein !).
- ④ **Global installation:** Otherwise enter the extracted directory and configure CPPAD in accordance with, e.g., IPOPT (under assumption that IPOPT 's headers are installed somewhere, for instance in */usr/local/include/coin*:

```
user@linux:~/SOFTWARE/CppAD$ cd cppad-version/; ./configure
--prefix=/usr/local --with-Documentation --with-Example --with-Speed
--with-Introduction5
```

Ⓢ Mind that only *./configure* is mandatory. All additional strings are merely optional and serve to include examples that can be tested after *make* has been invoked. Likewise the CPPAD - documentation will be installed in subdirs of the directory specified by the *prefix* string!

Ⓢ Mind to write */usr/local* anywhere (if you use additional strings with *./configure*) instead of */usr/local/* (i.e. omit forward slash character '/') !!)

- ⑤ Compile stuff

```
user@linux:~/SOFTWARE/CppAD/cppad-version$ make
```

---

<sup>5</sup>CPPAD says you can compile it also with IPOPT (provided IPOPT has been installed before), i.e. `IPOPT_DIR=IpoptDir` has to be chosen such that *IpoptDir/include/IpIpoptApplication.hpp* is a valid reference to the file *IpIpoptApplication.hpp*.

- ⑥ Optional: Test included examples, for instance, check the *get\_started.cpp* example where the derivative of a polynomial of degree 4 with all coefficients being equal to 1, i.e.  $f'(x) = 4x^3 + 3x^2 + 2x + 1$  is evaluated at  $x = 3$   
Check [http://www.coin-or.org/CppAD/Doc/get\\_started.cpp.htm](http://www.coin-or.org/CppAD/Doc/get_started.cpp.htm) for further details:

```
user@linux:~/SOFTWARE/CppAD/cppad-version$ cd introduction/get_started
$6 make; ./get_started
```

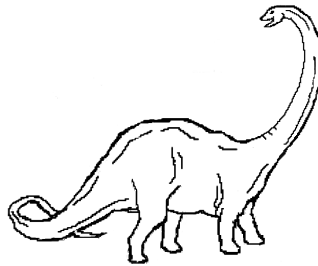
should return 'f'(3) computed by CppAD = 142'.

- ⑦ Install it, hunk

```
user@linux:~/SOFTWARE/CppAD/cppad-version$ su
(now enter the root password)
Then install it ('#' is my abbreviation for the root mode in the present directory)
# make install
```

Yet again, all the CPPAD source files are installed globally in */usr/local/include/cppad* and there should be no need to copy the directory to every project you're programming

☺



## Preparation of METIS

❗ Install it from the scratch! In particular, don't copy a patched version ;) Otherwise you might get errors like “undefined references to `_log2`”, etc.

- ❶ Get METIS

```
http://glaros.dtc.umn.edu/gkhome/metis/metis/download
```

- ❷ `tar xzvf metis-version.tar.gz`

- ❸ For metis-4.0 you'll need the following patch

```
http://www.math-linux.com/IMG/patch/metis-4.0.patch
```

- ❹ `cp metis-4.0.patch metis-4.0/`

- ❺ `cd metis-4.0/; patch -p1 < metis-4.0.patch`

- ❻ `make`

Upon successful completion you should have some executables, namely *graphchk*, *kmetis*, *mesh2dual*, *mesh2nodal*, *oemetis*, ....

## Installation of MUMPS

After having installed METIS, we proceed as follows:

- ❶ Get MUMPS from

```
http://graal.ens-lyon.fr/MUMPS/index.php?page=dwnld
```

- ❷ `tar xzvf MUMPS_version.tar.gz`

- ❸ `cd MUMPS_version`

- ❹ Edit `Makefile.inc` and change the following variables accordingly

- `LMETISDIR = $(HOME)/Sourcepath2Metis7/metis-4.0`
- `FC = gfortran`
- `FL = gfortran`
- Some compilers don't support the `-nofor_main` optimized option. Just uncomment all occurrences of it in `Makefile.inc`.

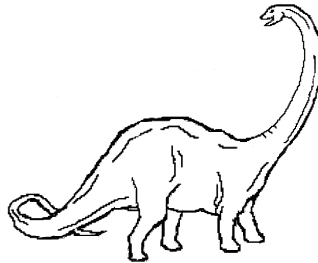
- ❺ `make clean; make`

---

<sup>7</sup> “Sourcepath2Metis” is a placeholder for your path to METIS



Upon successful completion, you should find the following files in *MUMPS\_version*: *examples, lib, libseq, src*.



## Installing UMFPACK

🛑 UMFPACK requires BLAS and LAPACK ;)

❶ Get the following current packages needed in conjunction with UMFPACK:

- <http://www.cise.ufl.edu/research/sparse/umfpack/current/>
- [http://www.cise.ufl.edu/research/sparse/SuiteSparse\\_config/current/](http://www.cise.ufl.edu/research/sparse/SuiteSparse_config/current/)
- <http://www.cise.ufl.edu/research/sparse/amd/current/>

❷ Copy them into the *same* directory

❸ Unpack the above packages via

```
tar xzvf UMFPACK.tar.gz
```

etc.

❹ `cd SuiteSparse_config`

- Edit *SuiteSparse\_config.mk*:
- comment line 184, i.e. `#UMFPACK_CONFIG`
- UNcomment line 187, i.e. write `UMFPACK_CONFIG = -DNCHOLMOD` to use the whole thing without CHOLMOD

# Installation of DUNE

Note that DUNE depends on several packages which all have to be obtained first. Therefore I subdivide this section into several others explaining briefly the crucial steps of their installation.

## ParaView – a fancy package for visualizing data

⊛ If you are using DEBIAN/GNU LINUX then you can simply ignore steps ❶ - ❺!!! Instead switch to root on command line (*su enter root passwd...*) and type  
**# aptitude install paraview**  
That's it :-).

However, if you're the poor guy who is running SUSE, you must stick to the following steps: Install the sources of the package PARAVIEW from

<http://www.paraview.org/paraview/resources/software.html>

The sources are at the end of the *Latest release (x.y.z)* list, closely followed by the Data sources (also beneficial, I suppose).

- ❶ A good installation guide (UNIX) is available from

[http://www.paraview.org/Wiki/ParaView:Build\\_And\\_Install#On\\_Unix-like\\_systems](http://www.paraview.org/Wiki/ParaView:Build_And_Install#On_Unix-like_systems)

- ❷ Prerequisites:

- (a) cmake (version  $\geq 2.4.5$ ):

```
cd $HOME
wget http://www.cmake.org/files/v2.4/cmake-2.4.5.tar.gz
tar xvfz cmake-2.4.5.tar.gz
mkdir cmake-2.4.5-bin
cd cmake-2.4.5-bin
../cmake-2.4.5/bootstrap --prefix=$HOME/software
make; su
make install
```

O.k. now cmake is on your system...

- (b) Trolltech's Qt qmake (version 4.3.5 recommended ⊛Stick to this version like the eye to the appearance of a beautiful woman (like with pretty women, other versions aren't supported yet!!))
- Download, e.g. *qt-x11-opensource-src-4.3.5.tar.gz* from  
<ftp://ftp.trolltech.no/qt/source/>
  - Install it by following the manual which is available on  
<http://doc.trolltech.com/4.3/install-x11.html>

- iii. Suppose again we have a directory *SOFTWARE/Qt* to which the sources have been downloaded. Extract them via

```
cd SOFTWARE/Qt
gunzip qt-x11-opensource-src-4.3.5.tar.gz
tar xvf qt-x11-opensource-src-4.3.5.tar
```

Now the package directory *qt-x11-opensource-src-4.3.5* is revealed. Change to it and build it

```
cd qt-x11-opensource-src-4.3.5
./configure
(automatically configures in /usr/local/Trolltech/Qt-4-3.5.
If you like change it via the prefix option but I use the default dir
(like for the other software)).
You are asked to accept the license, type yes
(for reconfiguring, type make confclean; configure
make
su
(enter root password)
make install
```

- iv.  Don't forget to set the environment variable for qmake in the file *\$HOME/.profile* !!!:

```
PATH=/usr/local/Trolltech/Qt-4.3.5/bin:$PATH
export PATH
```

That's it – QT is installed and ready to use ☺

(c) **Installing MPI such as MPICH**

This is an important issue, I fancy, not only for PARAVIEW since every scientific software relies on it sooner or later.

- i. Download "UNIX (all flavors)" sources *mpich.tar.gz* from

<http://www.mcs.anl.gov/research/projects/mpi/mpich1/download.html>

- ii. Choosing the correct devices and release, e.g. for

**Workstations networks, Beowulf Clusters and Individual Workstations:** *ch\_p4* (most general device, supports SMP nodes, MPMD programs, heterogeneous collections of systems, etc. This should suffice for most applications and is installed by default if you don't specify any device flags (cf. below) !) and *ch\_p4mpd* (only for homogeneous clusters)

**Symmetric Multiprocessors (SMPs):** *ch\_shmem*

Fortunately, both configurations and installations are the same ! (except for *ch\_p4mpd* where a special configure flag should be used). The installation guides can be found as PDF or PS files at the very bottom of the page.

**Hint:** `uname -a` gives you some general information about your system which may be useful.

- iii. **Note:** Depending on how you want to calculate later on, it is useful to install remote shell *rsh* as well as *rsh-server* in advance and enable it via

the following commands executed in root mode:

```
# chkconfig xinetd on
# chkconfig rsh on
# chkconfig rexec on
# chkconfig rlogin on
# service xinetd restart
```

Now you should be able to run rsh via, e.g. `rsh localhost` which, when executed, asks for your system password (like ssh would do).

iv. Run the set of commands

```
cd SOFTWARE; mkdir MPICH; cd MPICH
(assume you have copied the MPICH sources to this dir)
tar zxovf mpich.tar.gz
cd mpich-1.2.7p1
```

v. For security reasons it seems advisable to configure MPICH with ssh. To avoid tedious password inputs, all logins to slaves should be made password-free:

- Generate one ssh key via `ssh-keygen -r rsa`
- (Optional if no `.ssh` dir exists on host you want to log in without password later on: `ssh targethost mkdir -p .ssh` <enter targethost's passwd>)
- `cat .ssh/id_rsa.pub | ssh targethost 'cat >> .ssh/authorized_keys'`
- <enter targethost's passwd one last time...>

where *targethost* denotes any host you want to compute on (i.e. one ssh key suffices for all hosts)! In other words, simply execute the very last command for every user a on machine A (a@A) to user b on machine B (b@B).

vi. Configure it via

```
./configure -rsh=ssh8 --prefix=/usr/local/mpich-1.2.7 2>&1 |
tee c.log
```

from the dir of the MPICH sources, e.g. `SOFTWARE/MPICH/mpich-1.2.7`. You need not to create dir `/usr/local/mpich-1.2.7`. It will be created automatically during installation.

vii. `make 2>&1 | tee make.log`

viii. `su; (enter password)`

ix. `make install` (to uninstall, run the script `/usr/local/mpi-1.2.7/sbin/mpiuninstall`)

x.  Don't forget to set the environment paths in the file `.bashrc` and/or `.profile` as follows:

```
export LD_LIBRARY_PATH="$LD_LIBRARY_PATH:/usr/local/mpich-version/lib"
export MANPATH="$MANPATH:/usr/local/mpich-version/man"
export PATH="$PATH:/usr/local/mpich-version/bin"
```

---

<sup>8</sup>You can also specify special MPICH devices via `--with-device=ch_shmem`. The `-rsh=ssh` seems deprecated. By default it was substituted automatically by some flag `RSHCOMMAND` ...

Now MPICH is installed in `/usr/local/mpich-1.2.7`.

You can run test examples (cf. PDF guide) which I recommend !!! Change to the directory where the extracted MPICH sources are, e.g.

`$HOME/SOFTWARE/MPICH/mpich-1.2.7/`

Then again change to some subdirs therein, namely

```
$ cd examples/basic/  
$ make clean; make cpi; ../../bin/mpirun -np 4 cpi
```

Now you are asked to input your system password (not root password) at least once (in my case thrice ! ☺). The output, you should get, looks approximately like the following:

```
Process 0 of 4 on foo  
pi is approximately 3.1415926544231239, Error is 0.0000000008333307  
wall clock time = 0.000979  
Process 1 of 4 on foo  
Process 3 of 4 on foo  
Process 2 of 4 on foo
```

where *foo* denotes the name of your account.


You can also check `/usr/local/mpich-1.2.7/share`. There should be a default *machine* file with name *machines.LINUX* containing 5 times the name of localhost (*foo*) (By the way, such a machine file is used later on, e.g. in your home directory, to define the hosts you want to compute on ....

I also recommend a **thorough testing**..

Execute the following commands in the directory of the MPICH sources (in my case `/SOFTWARE/MPICH/mpich-1.2.7/`:

```
cd examples/test/  
make testing  
firefox summary.xml &
```

The last line gives you a clue about passed MPI tests and possibly failed ones (you should type `make clean` afterwards to free space).

 There were some problems with the `xset` stuff in my `.bashrc` file. I couldn't start the test suite unless I commented `xset b off` which, annoyingly, turned my system clock on again...

You should also run `/usr/local/mpich/sbin/tstmachines -v LINUX` which should return upon succession

```
Trying true on foo ...Trying user program on foo ...
```

also executable without the `-v LINUX` option which should return nothing upon succession.

**MPI usage:** you can copy and alter the Makefile in */usr/local/mpich-1.2.7/examples* for your future tasks.

(d) Luckily, `python` came along with the system so nothing else has to be installed for now.

- ③ ALL PREREQUISITES are assumed to be installed now. Next create your own build directory, say *buildMARC*:

```
cd SOFTWARE; mkdir ParaView; cd ParaView
tar xvf paraview-version.tar; unzip ParaViewData-version.zip
mkdir buildMARC; cd buildMARC
ccmake $HOME/SOFTWARE/ParaView/Para-version
```

O.k. a dialog appears and you have to press 'c' (means configure) iteratively untill 'g' (means generate appears). Type 'enter' and set button from OFF to ON etc.:

```
BUILD_SHARED_LIBS ON
PARAVIEW_DATA_ROOT $HOME/SOFTWARE/ParaView/ParaViewData3.4
PARAVIEW_ENABLE_PYTHON ON
(and for enabling MPICH )
PARAVIEW_USE_MPI ON
```

- ☹ For MPI, press 't' (means toggle) and manually set flags if it is not found automatically:
- 

```
MPI_INCLUDE_PATH /usr/local/mpich-1.2.7/include
MPI_COMPILER /usr/local/mpich-1.2.7/bin/mpicxx
PARAVIEW_DATA_ROOT $HOME/SOFTWARE/ParaView/ParaViewData3.4
QT_QMAKE_EXECUTABLE /usr/local/Trolltech/Qt-4.3.5/bin/qmake
PARAVIEW_ENABLE_PYTHON ON
PYTHON_INCLUDE_PATH /usr/include/python2.5
PYTHON_LIBRARY /usr/lib/python2.5/config/libpython2.5.a
```

- ④ Suppose option 'g' has been created, press 'g' and if, hopefully, no ERROR/WARNING messages are thrown ☺, type

```
make -j #(processors)
```

where *#processors* denotes the number of available processors.

- ⑤ Now it takes a long, long, long time but if compilation was successful enter

```
su
<root password>
make install
```

PARAVIEW is installed now and can be invoked on commandline via *paraview*<sup>9</sup>.

## Installing libtools

⊛ Again, if you are running DEBIAN GNU/LINUX as operating system a simple root command will install everything for you:

```
# aptitude install libtool libtool-doc
...that's it!
```

Otherwise, we have to do a little bit more:

- ❶ Download from

<http://ftp.gnu.org/gnu/libtool/>

- ❷ Prepare for compilation

```
cd SOFTWARE; mkdir Libtools; cd Libtools
copy libtools-version.tar.gz to this directory.
gunzip libtools-version.tar.gz; tar xvf libtools-version.tar
cd libtools-version
./configure --prefix=/usr
```

- ❸ Compile LIBTOOL

```
make
```

- ❹ Optional: Check test suite if it exists...

```
make test
```

- ❺ Installation

```
su
<enter password>
make install
```

## Installing ALUGrid

---

<sup>9</sup>To visualize data, type e.g. `paraview --data=convreacdiff_yasp_Q1_2d.vtu`

- ❶ Prerequisites for *parallel* installation: METIS,<sup>10</sup> PARMETIS<sup>11</sup> and PARTY LIB<sup>12</sup>
- ❷ Download sources from <http://www.mathematik.uni-freiburg.de/IAM/Research/alugrid/>
- ❸ extract them in *\$HOME/DUNE* via `tar xzf ALUGrid-1.x.tar.gz`
- ❹ `cd ALUGrid-version`
- ❺ 🔴 I really recommend to install ALUGRID via DUNE, thus omitting the next two steps and, instead, jumping to the next section 'An alternative way of installing ALUGrid via "dunecontrol"'  
🔵 However, since ALUGRID is not part of the software updated by DUNE , one should stick to \*this\* installation instruction.
- ❻ Build ALUGRID (parallel compilation)

```
./configure --prefix=/usr/local/alugrid CXX="mpiCC" \  
--with-metis=$HOME/DUNE/metis-4.0
```

- ❼ `make; su <enter password>; make install`

Now ALUGRID is installed in */usr/local/alugrid*.

## An alternative way of installing ALUGrid via "dunecontrol"

A much better way of installing ALUGRID is considered hereafter – I recommend to keep at this one only !! :

- ❶ Assume that the METIS and ALUGRID tarballs are in the directory containing the DUNE tarballs, e.g. *\$HOME/DUNE*

```
cd $HOME/DUNE; tar xzf ALUGrid-1.x.tar.gz
```

- ❷ `cd ALUGrid-version`

---

<sup>10</sup><http://glaros.dtc.umn.edu/gkhome/metis/metis/overview/>

Easy "Installation": 0.) Create a directory *DUNE* 1.) Extract stuff: `gunzip metis-4.0.tar.gz; tar -xvf metis-4.0.tar` into that dir 2.) Change to extracted directory *metis-4.0*, edit *Makefile.in* and adjust your compiler, etc. (CC = gcc will do it). 3.) Then run `make` and 11 files will be installed in the directory *~/DUNE/metis-4.0/Lib/* or any directory where you want to use METIS. That's it!

<sup>11</sup>Download from <http://glaros.dtc.umn.edu/gkhome/metis/parmetis/download>. Move it into the *DUNE* dir. Unpack it via `gunzip ParMetis-3.1.1.tar.gz ; tar -xvf ParMetis-3.1.1.tar`. Compilation: Go to */DUNE/ParMetis-{version}*. Type `make`. Check stuff: `cd Graphs; mpirun -np 4 ptest rotor.graph`. That's it!

<sup>12</sup><http://wwwcs.upb.de/fachbereich/AG/monien/RESEARCH/PART/party.html>. Unfortunately, you have to sign up by filling out a document, sign it and send it via non-digital mail to the author in order to receive an electronic version via email.

Luckily PARTY Lib isn't needed anymore by any parallel ALUGrid versions & you can disregard the compiler warning w.r.t. PARTY LIB ☺!



- ③ Next run DUNE's configure script with some options given just below<sup>13</sup>: (*version* should be replaced by your DUNE version):

```
./configure CC=gcc CXX=g++ F77=gfortran --prefix=/usr/local/alugrid /
--with-metis=$HOME/DUNE/metis-4.0 /
CPPFLAGS="'../dune-common-version/bin/mpi-config --cflags --disable-cxx /
--mpicc=mpicc'" LDFLAGS="'../dune-common-version/bin/mpi-config --libs/
--disable-cxx --mpicc=/usr/local/mpich-1.2.7/bin/mpicc'"
CXXFLAGS="-O3 -DNDEBUG" CFLAGS="-O3 -DNDEBUG"
```

If configuration is successful you should read something like this while configuration is proceeding:

ALUGrid is ready to compile for parallel computations!

The following components where found:

```
-----
METIS.....: yes (Version 4.0)
PARTY.....: no
MPI.....: yes
-----
```

See ./configure --help and config.log for reasons  
why a component wasn't found

- ④ make clean; make
- ⑤ su <enter root passwd> make install

... and ALUGRID should be installed in */usr/local/alugrid*.

## Installing ALBERTA<sup>14</sup>

- ① Get latest version from <http://www.alberta-fem.de/download.html> or a newer version form:  
<http://www.mathematik.uni-freiburg.de/IAM/homepages/claus/download/alberta/>
- ② Unpack it: `tar xzf alberta-version.tar.gz; cd alberta-version`

---

<sup>13</sup>Download all the DUNE files, save them in `~/DUNE/` and extract them via `tar xzf dune-*`  
NOTE: They needn't be installed, it just works that simple!

<sup>14</sup>Optional: Not necessarily required!

### 3 OPENGL :

Depending on the operating system in use, the configure programme might not find the OpenGL library *libGL* via `-lGL` though it had been installed previously. If you are to encounter such difficulties, do the following:

- Check if *libGL* is really installed: `$ ls -l /usr/lib | grep GL`
- Check also file `/usr/lib/libGL.so.1` and `ldd /usr/lib/libGL.so.1`
- This can give you a hint to create a soft link such that the programme can find the library:

```
# ln -s /usr/lib/libGL.so.1 /usr/lib/libGL.so
```

and it should work then (Check via `ls -l /usr/lib/libGL.so*` to see that link has really been created)! This works pretty well. Yet it has a drawback, namely, that with every reboot you have to switch to root and launch the link command again. One way out of this nuisance:

CUSTOMIZE COMMAND: This has been working for me:

(a) `cd /etc/init.d`

(b) `su <enter root passwd>`

(c) Create a file, let's say, *marcopengl*:

```
#!/bin/sh
```

```
### BEGIN INIT INFO
```

```
# Provides:          marcopengl
```

```
# Required-Start:    $local_fs $remote_fs $network $named $time
```

```
# Required-Stop:     $local_fs $remote_fs $network $named $time
```

```
# Default-Start:     2 3 4 5
```

```
# Default-Stop:
```

```
# Short-Description: Set softlink to openGL
```

```
# Description:       By defining a softlink you should be able
```

```
#                   to properly use -lGL
```

```
### END INIT INFO
```

```
echo -n "Creating LibGL softlink for proper usage of -lGL..."
```

```
ln -s /usr/lib/libGL.so.1 /usr/lib/libGL.so
```

```
echo "Done :-)"
```

```
exit 0
```

(d) Save it. Execute `chmod 755 /etc/init.d/marcopengl`

(e) Add links for the file to be invoked at boot time:

```
update-rc.d marcopengl defaults15
```

(f) If it's still not working: I had the problem with the file `/etc/init.d/nvidia-glx`. Do the following at the end of the file:

UNCOMMENT the line `rm -f /usr/lib/libGL.so || true` and save it. This

---

<sup>15</sup>The script can be removed from the startup sequence via `update-rc.d -f marcopengl remove`

should guarantee that the *libGL.so* won't be removed with the start of NVIDIA.

- ④ Apparently, the newer versions (second download link) can be installed via  
\$ `./configure --prefix=/usr/local/alberta`  
Then type `make; su... make install`  
Tools like GTOOLS, OPENDX, GRAPE, SILO seem to be optional and only for visual representation of solutions; I think PARAVIEW should suffice...
- ⑤ This and the following step may be used if you intend to install an older ALBERTA version ( $\leq 2.0.1$ ): Configure the stuff via

```
./configure --prefix=/directory/to/install/alberta/to  
            \--with-blas-name=blas-3 --disable-shared \  
            CC=gcc CXX=g++ F77=gfortran
```

where *directory/to/install* may be named, for instance, */usr/local/alberta* and *-with-blas-name* stands for the name of your BLAS (sometimes *blas*, etc.).

ⓈMandatory: *-disable-shared* !!!

- ⑥ Compile stuff: `make`
- ⑦ General installation (version independent, of course): `su <enter root passwd>`  
`make install` – That's it. ALBERTA is now in */usr/local/alberta*.

## Finally install DUNE packages

- ① Installation is well documented in the DUNE grid interface HOWTO from

<http://www.dune-project.org/doc/grid-howto/>

or on the homepage itself

<http://www.dune-project.org/doc/installation-notes.html>

- ② Download the stable core modules from

<http://www.dune-project.org/download.html>

. These should be at least the following:

- **dune-common** (*dune-common-version.tar.gz*)
- **dune-grid** (*dune-grid-version.tar.gz*)
- **dune-istl** (*dune-istl-version.tar.gz*)

A word of caution: If one copies the tutorial about developing dune grids "dune-grid-**dev**-howto" into a subdirectory then `./dune-common-version/bin/dunecontrol` returns error messages. Therefore don't copy it into any subdirectories of a dir you want to run and install DUNE !

In addition, *dune-fem*, *dune-localfunctions*,... may lie in `~/DUNE/`.

- ③ Copy those .tar.gz files into a common directory, e.g. `$HOME/DUNE`.
- ④ `cd ~/DUNE` and extract the sources therein via `tar xzf *-version.tar.gz` where *version* stands for the DUNE version you want to install.
- ⑤ You can create a file *parallel-debug.opts* in `$HOME/DUNE`<sup>16</sup> and write some options into it, e.g. the MPICH location has to be set here such that it can be found by `dunecontrol`!, e.g. if you have two processors at choice

```
CONFIGURE_FLAGS="--with-alugrid=/usr/local/alugrid --enable-parallel"
MAKE_FLAGS="-j 2"
```

A more detailed *parallel-debug.opts* file follows:

```
#DUNE option file for configuration
#Configuration flags are written to '.opts' file 'parallel-debug.opts'
# Marc added -O3 -finline-functions -funroll-loops -DNDEBUG
CONFIGURE_FLAGS="--prefix=/home/mfein/ARRAKIS --disable-documentation \
--disable-mpiruntest --enable-parallel --with-alugrid=/usr/local/alugrid \
--with-alberta=/usr/local/alberta CXX=g++ CC=gcc \
CXXFLAGS=\"-O0 -g -Wall -O3 -finline-functions -funroll-loops -DNDEBUG\"
CFLAGS=\"-O0 -g -Wall\" MAKE_FLAGS="all"
```

An alternative one including the on-the-fly visualisation tool GRAPE is the following one:

```
#DUNE option file for configuration
#Configuration flags are written to '.opts' file
CONFIGURE_FLAGS="--prefix=/home/mfein/ARRAKIS \
--x-includes=/usr/include/X11R6 \
--x-libraries=/usr/lib \
--disable-documentation --disable-mpiruntest \
--enable-parallel --with-alugrid=/usr/local/alugrid \
--with-alberta=/usr/local/alberta \
--with-grape=$HOME/DUNE/grape CXX=g++ \
CC=gcc CXXFLAGS=\"-Wall -O3 -finline-functions \
-funroll-loops -DNDEBUG\" CFLAGS=\"-g -Wall\"
MAKE_FLAGS="all"
```

---

<sup>16</sup>For a more detailed description of how to fill the *parallel-debug.opts* please visit the above sites. It is also possible to set options for `svn`. See also how to maintain DUNE modules cf. <http://www.dune-project.org/dunemodule.html>

Choose one of them, save it and run

```
./dune-common-version/bin/dunecontrol --opts=parallel-debug.opts all
```

which builds the stuff for all DUNE core modules in this directory.

A word of caution: Never ever set the path to where the *dune-\*.tar.gz* files lie (I chose *\$HOME/ARRAKIS* for installation whereas the *\*tar.gz* files are in *\$HOME/DUNE*)!!

Running the configuration should at least find:

Found the following Dune-components:

-----

```
dune-common.....: yes (/home/mfein/DUNE/dune-common-1.2.1) version 1.2.1
ALBERTA.....: yes (Version 3.0)
ALUGrid.....: yes
AmiraMesh.....: no
Grape.....: no
HDF5.....: no
MPI.....: yes (MPICH)
OpenGL.....: yes
UG.....: no
```

-----

NOTE: you should have installed 'automake' and 'autoconfig' via `# aptitude install automake autoconfig`

It seems also better to install DUNE in some dir in your *\$HOME* dir

- ⑥ Finally run (unless you have already specified it in *.opts*) in superuser mode (*su*):

```
./dune-common-version/bin/dunecontrol make install17
```

## Create your own DUNE project

To create your own DUNE project, make sure you're again in the directory with the DUNE sources (where you extracted the *dune-\*.tar.gz* files – in my case *\$HOME/DUNE*) and run

```
$ ./dune-common-version/bin/duneproject
```

---

<sup>17</sup>Likewise, `# ./dune-common-version/bin/dunecontrol make uninstall` removes DUNE !

Now you are asked to name your project, let's say you've chosen the name *Woodpecker* for it.

Next you are to specify it's version, the email address of the maintainer, DUNE modules to use later on, etc. (Check if it worked via `ls -l Woodpecker/` and see which files are there. You should recognise, among other things, mainly

```
Woodpecker.cc
configure.ac
Makefile.am
dune.module
README
```

*Woodpecker.cc* is a default file being meant to output some information whether you use sequential or parallel computations.

**The way of nonchalance:** Well, crucial files seem to me the red dyed ones, namely *configure.ac* as well as *Makefile.am* as they may be changed according to your intensions. In *configure.ac*: Replace the source directory specifier with your sources, e.g.

```
AC_CONFIG_SRCDIR([exercise1.cc])18
:
AC_CONFIG_FILES([ ...exercise1.pc ])
```

In *Makefile.am*: Replace all occurances of the default name *Woodpecker*, *Woodpecker.\** with your sources, e.g.:

```
:
noinst_PROGRAMS = exercise1 exercise2

exercise1_SOURCES = exercise1.cc

exercise1_CXXFLAGS = ...
exercise1_LDADD = ...
:
exercise2_SOURCES = exercise2.cc

exercise2_CXXFLAGS = ...
exercise2_LDADD = ...
:
```

### Run your project

After that simply type

```
$ ./dune-common-version/bin/dunecontrol --opts=parallel-debug.opts
--module=Woodpecker all
```

or if you do not want to configure every other modules again:

```
$ ./dune-common-version/bin/dunecontrol --opts=parallel-debug.opts
--only=Woodpecker all
```

⚠️Never write a slash (‘/’) at the end of the module name (since it won’t be recognized as directory *Woodpecker* any more)!

So now you have a couple of files more in *\$HOME/DUNE/Woodpecker*, namely *Makefile.in*, *Makefile*, some links and so on.

Of special interest are (at least in my example case here) the two executable files *exercise1* and *exercise2*. You can also have some *\*dgf* file which specifies your domain, e.g. *unitsquare.dgf* :

```
DGF
Interval
0 0 % first corner
1 1 % second corner
100 100 % write no of cells in x and in y direction here
BOUNDARYDOMAIN
default 1 % default boundary id
#BOUNDARYDOMAIN
#unitsquare.dgf
```

If you type now in the directory *\$HOME/DUNE/Woodpecker*

```
$ ./exercise1
```

and/or

```
$ ./exercise2
```

you maybe see some output and/or some *.vtu* files or that like which can now be visualised via

```
$ paraview --data=name.vtu
```

If there are plenty of *.vtu* files you can watch their film by clicking on PARAVIEW’s *File* button, then *open*, choose now the *\*vtu* and press *Apply* and you can see a cool movie...

**Note:** If you simply run

```
$ ./Woodpecker/Woodpecker
```

from *\$HOME/DUNE* without having specified your sources the default executable file *Woodpecker* will be executed and the output should look somehow like

```
Hello world! This is Woodpecker.
I am rank 0 of 1 processes!,
```

assumed you have set options to enable parallelism...

By the way:

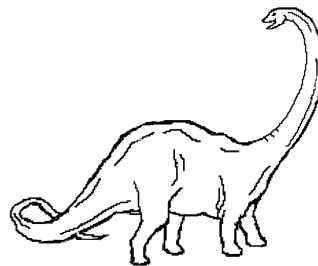
- ❶ Make sure that you haven't installed several MPI versions at the same time (e.g. MPICH and LAM/MPI). This might cause problems!
- ❷ When running mpi on let's say 4 processors via `mpirun -np 4 ./myexecutable` and you encounter an error message like

```
ssh: connect to host localhost port 22: Connection refused
p0_11581: p4_error: Child process exited while making connection to
remote process on localhost: 0
p0_11581: (6.006387) net_send: could not write to fd=4, errno = 32
```

you should check if you have installed the openssh-server and if so if it is running: Therefore, check the current system processes: `ps -e | grep ssh`. Execution of this command could return something similar to `2609 ? 00:00:00 sshd` in case of success. Try also the command `ssh -V` to see if it's there. Otherwise install it via `# aptitude install openssh-server`. After rebooting the openssh-server should work!

If run DUNE now in parallel via `mpirun -np #ofprocessors ./yourstuff` you will see that every step is executed and output to screen exactly `#ofprocessors` times (compare the third column):

HAVE A LOT OF FUN, HUNK ☺



### Problems with svn

**Problem:** svn gives *out-of-date* error (perhaps due to moving files and dirs without using `svn mv ...`).

**Solution:** Denote by *io* the damaged folder.

- Make a security copy of the folder to a lokal (non-svn) directory, e.g.: `mv io/ ~`
- Change to parent directory and delete it: `cd ..; svn delete io/`
- Revert all local changes: `svn revert io/`
- `svn delete io/`
- `svn commit`



- `mkdir io`
- `svn add io/`
- `svn commit io/`
- Copy back from security folder: `cp -r ~/io/* io`
- `cd io/; svn add *`
- `svn commit -m "resolved out-of-date error"`

...here you go! By the way, to remove locks, type `svn cleanup` in the dir under consideration.

