

# Green User Guide

## Contents

- [Description](#)
- [General information](#)
- [Quick getting started example](#)
- [Accessing the system](#)
- [Security](#)
- [Storage](#)
- [Environment](#)
- [Submitting jobs](#)
- [Compiling](#)
- [Math libraries](#)

## Description

Green is a cluster used by two research groups from IFM (The Department of Physics and Measurement Technology, Biology and Chemistry) at Linköpings Universitet. The groups are lead by Professor Igor Abrikosov (<http://www.ifm.liu.se/~abrikos>) and Professor Sven Stafström (<http://www.ifm.liu.se/~svens>).



## Hardware

74 Compute Nodes	<a href="#">Dell PowerEdge 750</a> . Pentium 4, 3.4 GHz, 1MB cache, 800 MHz FSB. 2GB ECC DDR 400MHz. 80 GB SATA (7200 RPM). 2 x On board Gigabit NIC.
1 Front End	<a href="#">Dell PowerEdge 700</a> . Pentium 4, 3.4 GHz, 1MB cache, 800 MHz FSB. 2GB ECC DDR 400MHz. CERC SATA RAID Controller. 4 x 250 GB SATA (7200 RPM).
1 Ethernet Switch	1 <a href="#">HP ProCurve Switch 4160GL</a> , 80 ports, 1000Base-T.

## Software

Operating System:	<a href="#">Centos 3.3</a> (A Red Hat Enterprise Linux 3 rebuild)
Resource Manager:	<a href="#">Torque</a>

Scheduler:	<a href="#">Maui</a>
Compilers:	<a href="#">Intel C++ Compiler 8.1 for Linux</a>
	<a href="#">Intel Fortran Compiler 8.1 for Linux</a>
	<a href="#">GCC 3.2.3</a>
Math library:	<a href="#">Intel Math Kernel Library 7.2</a>
	<a href="#">Intel Math Kernel Library 6.0</a>
	<a href="#">SLATEC Common Mathematical Library 4.1</a>
MPI:	<a href="#">Lam 7.1.1</a>
	<a href="#">Mpich 1.2.6</a>
Applications:	<a href="#">Vasp</a>
	<a href="#">Dalton 1.2.1</a>
	<a href="#">Gaussian 03</a>
	<a href="#">Molden 4.0</a>
	<a href="#">Matlab 7.0.1</a>

For detailed information about Vasp, go to:

<http://www.nsc.liu.se/software/physics/vasp/>

For detailed information about Dalton, Gaussian and Molden, go to:

<http://www.nsc.liu.se/software/chemistry/>

## General information

- Status information of Green can be found at <http://status.nsc.liu.se/#green> and <http://www.nsc.liu.se/cgi-bin/greenstatus>.
- Avoid running resource hungry applications directly on the front end, instead allocate resources through the [batch queue system](#) (for testing purposes, interactive jobs are suitable). This will prevent users from slowing down the overall system performance of Green.
- Forward all mail sent to your Green user to your real e-mail address. You will then receive notifications and status of your job to your real e-mail address. Do this by making a `.forward` file in your home directory which only contains your real e-mail address. Example:  

```
echo myemail@ifm.liu.se > ~/.forward
```
- Use the local disks (`/disk/local/`) for temporary storage instead of using the NFS mounted area `/home`. `/home` consists of two RAID 0 disks, it is globally mounted over NFS to every compute node and will perform poorly compared to the local disks attached to the compute nodes. Notice that `/disk/local` are not backed up and should be considered as temporary scratch areas.
- File transfer is available using `scp` and `sftp`.
- Backup of `/home` is made every night.
- If you use bash, do not remove the default entries in your `~/ .bashrc` (i.e. “`./etc/bashrc`”). If you do, you won't get the necessary environment when running your MPI applications.

- For questions, contact [support@nsc.liu.se](mailto:support@nsc.liu.se).

## Quick getting started example

All the files in this example are available [here](#).

1. Get a user account on Green. Contact Ulf Ekström at IFM ([ulfek@ifm.liu.se](mailto:ulfek@ifm.liu.se)).
2. Log in to Green using ssh:  

```
ssh username@green.nsc.liu.se
```
3. Once logged in, enable e-mail forwarding to your real e-mail address (has only to be done once):  

```
echo youremail@ifm.liu.se > ~/.forward
```
4. Compile an MPI application using the preferred Intel compiler (see [Compiling](#) for more details):  

```
icc hello_world.c -o hello_world.icc.mpich -Nmpich
```
5. Run the application as a batch job (see [Submitting jobs](#) for more details):
  1. Create a PBS-script. This file contains information about how many nodes you wish to use, how long you expect the job to run, how to start the application, etc. Here is sample PBS script: [pbsexample](#).
  2. Submit the job.  

```
qsub pbsexample
```
  4. When the job is finished, an e-mail will be sent to the e-mail address specified in `~/.forward`.

For more detailed information, read [Compiling](#) and [Submitting jobs](#).

## Accessing the system

Get a user account on Green. Contact Ulf Ekström at IFM ([ulfek@ifm.liu.se](mailto:ulfek@ifm.liu.se)).

Login to Green is available using *ssh*:

```
ssh username@green.nsc.liu.se
```

File transfer is available using *scp* and *sftp*:

Example: copying the file named example to Green:

```
scp ./example user@green.nsc.liu.se:~/documents
```

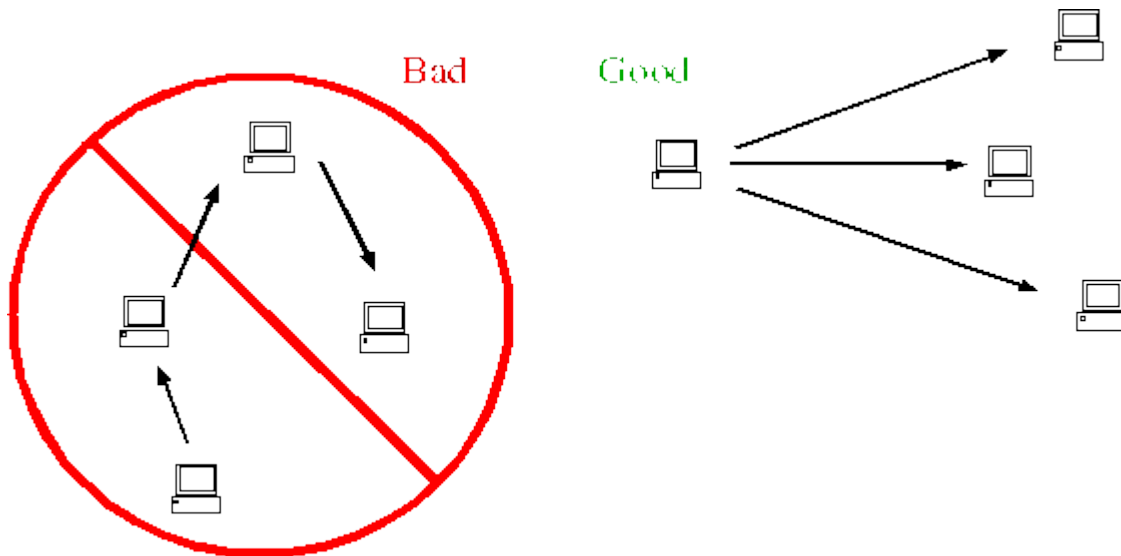
Example: connecting to Green using *sftp*:

```
sftp user@green.nsc.liu.se
Connecting to green.nsc.liu.se...
sftp>
```

## Security

When a system is compromised and passwords stolen, the thing that causes the most grief is when the stolen password can be used for more than one system. A user that

has accounts on many different computers and gets his/her shared password stolen will allow the intruders to easily cross administrative domains and further compromise other systems.



**Not using a trivial password** (your name, account, dogs name, etc.) is VERY important. However, using really hard passwords it not even close to as important as **not sharing them between systems**.

To login to a system and then continue from that system to a third (as illustrated above) **is bad** and should be avoided.

When logging into a system it only takes half a second to **read the “last login” information**. If you can't verify the information, contact [support@nsc.liu.se](mailto:support@nsc.liu.se) as soon as possible.

What users should do

- Use different passwords for different systems (VERY IMPORTANT)
- Do not use weak passwords
- Avoid chains of ssh sessions (see figure above)
- Check “Last login: DATE from MACHINE”

## SSH public-key authentication

There is an alternative to traditional passwords. This method of authentication is known as key-pair or public-key authentication. While a password is simple to understand (the secret is in your head until you give it to the ssh server which grants or denies access), a key-pair is somewhat **more complicated**.

Our recommendation is to use whichever method you feel comfortable with. If you invest some time to learn about key-pairs you will receive several benefits, including better security and easier work flow.

A key-pair is as the name suggests a pair of cryptographic keys. One of the keys is called the private key (this one should be kept secure and protected with a pass phrase) and a public key (this one can be passed around freely as the name suggests).

After you have created the pair, you have to copy the public key to all systems that you want to ssh to. The private key is kept as secure as possible and protected with a

**good** pass phrase. On your laptop/workstation you use a key-agent to hold the private key while you work.

- Can be much more secure than regular password authentication
- Can be less secure if used incorrectly (understand before use)
- Allows multiple logins without reentering password/pass phrase
- Allows safer use of ssh chains

How to use SSH public-key authentication instead of regular password authentication is described in chapter 4 in [SSH tips, tricks & protocol tutorial](#) by Damien Miller.

Short description of the necessary steps involved using SSH public-key authentication (read Damien Miller's guide above for more details):

- Generate a key-pair, choose a good pass phrase and make sure private key is secure (once).
- Put your public key into `~/.ssh/authorized_keys` on desired systems.
- Load your private key into your key-agent (ssh-add with [OpenSSH](#)).
- Run `ssh` all you want without reentering your pass phrase without the risk of anyone stealing your password.

## Storage

There are two types of storage available on Green:

<code>/home</code>	350 GB.	Backed up.	Globally mounted NFS area shared between all the users.
<code>/disk/local</code>	80 GB.	Not backed up.	Local disks attached to the compute nodes.

Use the local disks (`/disk/local/`) for *temporary* storage instead of using the NFS mounted area `/home`. `/home` consists of two RAID 0 disks, it is globally mounted over NFS to every compute node and will perform poorly compared to the local disks attached to the compute nodes.

**Important:** Notice that the files on `/disk/local` are deleted when the job is finished (see [Automatic cleanup and file stage in/out](#)) so no critical data should be placed at `/disk/local`. It should be considered as a temporary scratch area. Currently there is no way to rescue data on `/disk/local` if the job is killed (for example [a risk job that is preempted](#)) or crashes.

## Environment

We use something called *cmod* (or module) to handle the environment when there exists several versions of the same software installed. This application sets up the correct paths to the binaries, man-pages, libraries, etc. for the currently selected module.

The correct environment is set up by using *module*. Here is a list of the most useful arguments to module:

`module`

lists the available arguments

<code>module list</code>	lists currently loaded modules
<code>module avail</code>	lists the available modules for use
<code>module load example</code>	loads the environment specified in the module named example
<code>module unload example</code>	unloads the environment specified in the module named example

A default environment is automatically declared when you log in. The default modules are:

Intel 8.0

mkl 7.2

For example, we have Intel Compiler for Linux version 8.0 and version 8.1 installed, you could switch from 8.0 to 8.1 by the command:

```
[perl@green perl]$ module list
Currently loaded modules:
 1) intel
 2) mkl
 3) dotmodules
 4) default
[perl@green perl]$ module avail

In directory /etc/cmod/modulefiles:

+default          -intel/default    -pbs/default
+dotmodules       -mkl/7.2 (def)    -root
-intel/8.0 (def)  -mkl/default
-intel/8.1        -pbs/5.2.0 (def)
[perl@green perl]$ module unload intel/8.0
[perl@green perl]$ module list
Currently loaded modules:
 1) mkl
 2) dotmodules
 3) default
[perl@green perl]$ module load intel/8.1
[perl@green perl]$ module list
Currently loaded modules:
 1) intel/8.1
 2) mkl
 3) dotmodules
 4) default
```

Tip: The environment is specified in the files located under /etc/cmod/modulefiles.

## Submitting jobs

For a quick introduction how to compile and run jobs see [Quick getting started example](#)

There are two ways to submit jobs to the batch queue system, either as an **interactive** job or as a **batch job**. Interactive jobs are most useful for debugging as you get interactive access to the input and the output of the job when it is running. But the normal way to run the applications is by submitting them as batch jobs.

*qsub* is the name of the command for submitting jobs, either if it is a batch job or an

interactive job.

**Important:** *qsub* takes a submit script as a parameter and not standard in or any binary application. The PBS-script must have end its lines with a newline ( $\backslash n$ ) as is the default on Unix platforms. On Windows, line-endings are terminated with a combination of a carriage return ( $\backslash r$ ) and a newline( $\backslash n$ ) - this will **not work** with *qsub*.

Example of useful arguments to *qsub* (more important arguments are marked bold), read the man page for additional arguments and details:

-I	Run the job interactively.
-j oe	Join standard out (o) and standard error (e) to the same file. As default the standard out and standard error is saved in two different files.
<b>-l nodes=n</b>	<b>The number of nodes to run the job on, where n is an integer in the interval [1,74]. If using more than your guaranteed amount specify -q riskjobb also.</b>
<b>-l walltime=hh:mm:ss</b>	<b>The expected maximum execution time for the job.</b>
-m abe	Send mail to the local user@green.nsc.liu.se when the job begins (b), exits (e) and aborts (a). If none specified mail is only sent when the job is aborted.
-M email1@ifm.liu.se,email2@other.mail.se	List of e-mail addresses to send mail to. If not specified mail is sent to the local user@green.nsc.liu.se
-N myjobname	Name of the job, consisting of up to 15 printable, non white space characters with the first character alphabetic.
<b>-q riskjobb</b>	<b>Necessary if you want to run jobs on more nodes than your guaranteed amount.</b>

## Scheduling policy

- Every user has a guaranteed amount of nodes available on Green. If submitting jobs that uses no more nodes than your guaranteed amount then the job will be executed immediately.
- This means that on average, each user has a guaranteed amount of  $74/n$  nodes, where  $n$  is the total number of users on Green. There is no over allocation of nodes for each user, in contrast to most other clusters on NSC.
- This gives us a very high availability, but at the cost of low total throughput of jobs.
- In order to get a high throughput of jobs, and also to make it possible to use more nodes than your guaranteed amount, you may specify your job to run with the risk of being preempted at any time by an ordinary job.

There are 2 queues available on Green, one default queue for normal jobs not using more than your allotted amount of nodes and one queue called *riskjobb* for larger jobs. If you submit jobs with *qsub* without specifying to use the queue *riskjobb* the default queue is used. If you submit normal jobs using more than you amount of nodes, then it will be blocked. To be able to run such jobs you have to specify it as a risk job by adding the argument *-q riskjobb* to *qsub* (either in the PBS script or as a command-line argument).

**workq (default):** For normal jobs. Has higher priority than riskjobb. Only for jobs using less than or equal to your amount of nodes. May preempt any job running as riskjobb if that job is currently using nodes that the normal job wants and there are not enough nodes left. *Is used if not otherwise specified.*

**riskjobb:** For risk jobs. Has lower priority than normal jobs. For jobs using more than your specified amount of nodes. May be preempted at any time by a normal job (i.e. killed) and will then be requeued again and restarted whenever it gets resources. Specified with *-q riskjobb* to qsub.

Note that preempting of a risk job implies that it is killed. Since files residing on /disc/local will be deleted (read [Automatic cleanup and file stage in/out](#)), critical data should be saved on /home.

## Submitting batch jobs

Two sample PBS-scripts are available for download: [pbssample.simple](#) and [pbssample.advanced](#).

1. Create a PBS-script. This is a shell-script with additional declarations for the arguments to qsub, the arguments to qsub are declared as #PBS pbsargument, e.g. -N myjobname is specified as #PBS -N myjobname.

Example of a PBS-script named `pbssample.simple` using 24 nodes and is submitted to riskjobb. The wall clock time is 10 minutes and an e-mail will be sent when the job exits normally or exits with an error:

```
#!/bin/sh

# Request 24 nodes for the job and request 10 minutes of wall-clock time.
#PBS -l nodes=24,walltime=00:10:00

# Will run as a risk job.
#PBS -q riskjobb

# Request regular output (stdout) and error output (stderr) to the same
# file.
#PBS -j oe

# Send mail when the job exits normally (e) or aborts with an error (a).
#PBS -m ae

# Goto the directory from which you submitted the job.
cd $PBS_0_WORKDIR

# Start the job with mpprun on the nodes that the batch queue system have
# allocated for your job.
mpprun hello_world.icc.mpich
```

2. Submit the job, by specifying the PBS-script as the only argument to *qsub*:

```
[perl@green example]$ qsub pbssample.simple
5723.green
```

3. Check the status of the job with *qstat*. by specifying *qstat -n* you get information about allocated nodes. For even more details add *-f* as an argument. You should also try using *showq*.

```

[perl@green example]# qstat
Job id          Name                User                Time Use S Queue
-----
5723.green      pbssample.simpl     perl                00:00:00 R riskjobb

[perl@green example]# qstat -n

green:

Req'd          Elap                               Req'd
Job ID         Username Queue      Jobname      SessID NDS TSK Memory Time  S
Time
-----
5724.green     perl      riskjobb pbssample.   --    24  --    --   01:00 R
00:00
    n72/0+n71/0+n70/0+n69/0+n68/0+n67/0+n66/0+n65/0+n64/0+n63/0+n62/0+n61/0
    +n60/0+n59/0+n58/0+n57/0+n56/0+n55/0+n54/0+n53/0+n52/0+n51/0+n50/0+n49/0

```

4. Since we specified `-m ae`, an e-mail will be sent to the local user at Green when/if the job exits and aborts. The standard out and standard error is saved in the directory from where the job was submitted as `pbssample.simpl.o5723`

## Submitting interactive jobs

Submitting an interactive job is done at the command line by adding the argument `-I` to `qsub`.

Example: Interactive version of the PBS-script in [Submitting batch jobs](#)

```

[perl@green perl]$ qsub -I -lnodes=24,walltime=00:10:00 -q riskjobb
qsub: waiting for job 5972.green to start
qsub: job 5972.green ready

[perl@n11 perl]$ cd $PBS_O_WORKDIR
[perl@n11 perl]$ mpprun hello_world.icc.mpich

```

The output are displayed directly to the shell prompt.

Alternatively, you may run `mpprun` directly if you want to run a MPI application. `mpprun` will make an implicit call to `qsub` with a default wall time of 1 hour and automatically execute the application.

Example: Interactive job by calling `mpprun` directly.

```

[perl@green example]$ mpprun -np 24 hello_world.icc.lam
Using PBS to spawn an interactive job
spawn /usr/pbs/bin/qsub -I -lwalltime=1:00:00,nodes=24
qsub: waiting for job 5977.green to start
qsub: job 5977.green ready

[perl@n11 perl]$ exec /usr/local/bin/mpprun -Norder=default hello_world.icc.lam

```

## Automatic cleanup and file stage in/out

Automatic cleanup is performed when a job is finished. This includes killing all the user processes and removing everything from `/disk/local` on the compute nodes that participated in the batch job.

Note. Currently only file stage in is supported on Green. You may implement stage out in your code explicitly but that will not work if the application crashes or is being killed.

You may copy files locally to the nodes that are used in your job by using a PBS feature called stagein. This is specified with the argument `-W stagein=fileonexecheap@green:fileongreen[ , ... ]` where `[ , ... ]` means that you may specify a list of files to stage in. `fileonexecheap` is the path to the file on the compute nodes (`n1 - n74`), `fileongreen` is the path to the file that is to be copied from the frontend.

Example: copy `/home/myuser/example/src` from the frontend (Green) to `/disk/local/target` on the execution hosts:

```
#PBS -W stagein=/disk/local/target@green:/home/myuser/example/src
```

Please make sure that it works before using file stage in in your production jobs. See 'man qsub' for more details.

## Frequently used commands

Read the man-pages for more information about each listed command below. Commonly used commands are marked bold.

### Frequently used PBS user commands:

<b>qsub</b>	<b>Submits a job to the PBS queuing system.</b>
<b>qstat</b>	<b>Show status of PBS batch jobs.</b>
<b>qdel</b>	<b>Delete a PBS job from the queue.</b>

### Less frequently used PBS user commands:

qalter	Modifies the attributes of a job.
qhold	Requests that the PBS server place a hold on a job.
qrerun	Reruns a PBS batch job.
qrls	Release hold on PBS batch job.
qsig	Requests that a signal be send to the session leader of a batch job.

### Maui commands:

Maui is the scheduler. It takes care of how the jobs are prioritized, and where these jobs are run. Maui supports advance reservations, QOS levels, backfill, and allocation management.

<b>showq</b>	<b>List all jobs visible to the scheduler.</b>
showbf	Show resources available for immediate access.
showres	Show current reservations.
showstart	Makes a qualified guess about when a job will start.
checkjob	Display numerous scheduling details for a job.

## Compiling

For a quick introduction how to compile and run jobs see [Quick getting started example](#).

Intel Compiler for Linux 8.1 and GCC 3.2.3 is installed. The following compilers are

available:

	Intel	Gnu Compiler
C	icc	gcc
C++	icc	g++
Fortran	ifort	g77

We recommend using the Intel compilers.

## Compiling MPI applications

NSC Cluster Environment (NCE) is NSC's modification of GCC and Intel compilers which makes it easier to compile MPI-applications. Rather than having to specify a quite long list of libraries to be linked which also differs between different MPI implementations, you only have to compile with these additional arguments:

-Nlam	Compile with LAM support.
-Nmpich	Compile with MPICH support.

The options should be used both at compile-time (to specify the path to the include files) and at link-time (to specify the correct libraries).

Example: compiling the MPI-program, `mpiprogram.c` with `icc` for Lam (the same goes for `gcc`, `g++`, `g77` and `ifort`):

```
icc mpiprogram.c -o mpiprogram -Nlam
```

Example: compiling the MPI-program, `mpiprogram.f` with `ifort` for MPICH (the same goes for `gcc`, `g++`, `g77` and `icc`):

```
ifort mpiprogram.f -o mpiprogram -Nmpich
```

## Intel compiler, useful compiler options

Below are some useful compiler options, please do "man ifort" or "man icc" for more. For additional information, see:

- <http://www.intel.com/software/products/compiler/flin/> for Fortran.
- <http://www.intel.com/software/products/compiler/clin/> for C/C++.

### a) Optimisation

There are three different optimization levels in Intel's compilers:

-O0	Disable optimizations.
-O1, -O2	Enable optimizations (DEFAULT).
-O3	Enable -O2 plus more aggressive optimizations that may not improve performance for all programs.

A recommended flag for general code is `-O2` and for best performance "`-O3 -xP`" which will enable software vectorisation. As always however, aggressive optimisation runs a higher risk of encountering compiler limitations.

### b) Debugging

-g	Generate symbolic debug information.
----	--------------------------------------

-traceback	Generate extra information in the object file to allow the display of source file traceback information at runtime when a severe error occurs.
-fpe<n>	Specifies floating-point exception handling at run-time.
-mp	Maintains floating-point precision (while disabling some optimizations).

### c) Profiling

-p	Compile and link for function profiling with UNIX gprof tool.
----	---

### d) Options that only apply to Fortran programs

-assume byterecl	Specifies (for unformatted data files) that the units for the OPEN statement RECL specifier (record length) value are in bytes, not longwords (four-byte units). For formatted files, the RECL unit is always in bytes.
-r8	Set default size of REAL to 8 bytes.
-i8	Set default size of integer variables to 8 bytes.
-zero	Implicitly initialize all data to zero.
-save	Save variables (static allocation) except local variables within a recursive routine; opposite of -auto.
-CB	Performs run-time checks on whether array subscript and substring references are within declared bounds.

### e) Large File Support (LFS)

To read/write files larger than 2GB you need to specify some flags at compilation:

Fortran: no additional flags needed.

CC/C++: LFS is obtained by specifying the flags below when compiling and linking:

-D_FILE_OFFSET_BITS=64 -D_LARGEFILE_SOURCE
--

### f) Miscellaneous options

Little endian to Big endian conversion in Fortran is done through the F\_UFMTENDIAN environment variable. When set, the following operations are done:

- The WRITE operation converts little endian format to big endian format.
- The READ operation converts big endian format to little endian format.

F_UFMTENDIAN = big	Convert all files.
F_UFMTENDIAN = "big;little:8"	All files except those connected to unit 8 are converted.

### g) NCE options (locally supplied by NSC)

-Nlam	Compile with LAM support.
-Nmpich	Compile with MPICH support.

# Math libraries

## Intel Math Kernel Library

MKL Version 7.2 (default) and 6.0 is installed on Green.

For complementary information look at [Intel® Math Kernel Library 7.2 for Linux\\* Technical User Notes](#).

The Math Kernel Library includes the following groups of routines:

- Basic Linear Algebra Subprograms (BLAS):
  - vector operations
  - matrix-vector operations
  - matrix-matrix operations
- Sparse BLAS (basic vector operations on sparse vectors)
- Fast Fourier transform routines (with Fortran and C interfaces)
- LAPACK routines for solving systems of linear equations
- LAPACK routines for solving least-squares problems, eigenvalue and singular value problems, and Sylvester's equations
- Vector Mathematical Library (VML) functions for computing core mathematical functions on vector arguments (with Fortran and C interfaces).

Full documentation can be found at <http://www.intel.com/software/products/mkl/>

### Directory structure

mkl is located in `$MKL_ROOT`, defined at login. Semantically, MKL consists of two parts: LAPACK and processor specific kernels. The LAPACK library contains LAPACK routines and drivers that were optimized as without regard to processor so that it can be used effectively on processors from Pentium to Pentium 4. Processor specific kernels contain BLAS, FFTs, CBLAS, VML that were optimized for the specific processor. Threading software is supplied as a separate dynamic link library - `libguide.so`, when linking dynamically to MKL.

### Linking with MKL

To use LAPACK and BLAS software you must link two libraries: LAPACK and one of the processor specific kernels (i.e. `libmkl_p4` on Green). Please use `-L$MKL_ROOT` instead of hardcoding the path (e.g. do not use `-L/usr/local/intel/l_mkl_p_7.2.008/mkl72/lib/32`). This will ensure that correct libraries are used when switching [modules](#) between different mkl versions.

Example (LAPACK library, Pentium 4 processor kernel):

```
ld myprog.o -L$MKL_ROOT -lmkl_lapack -lmkl_p4
```

Example (Dynamic linking. DLL dispatcher will load the appropriate dll for the processor dynamic kernel):

```
ld myprog.o -L$MKL_ROOT -lmkl -lguide -lpthread
```

Example (Dynamic linking using ifort):

```
ifort -L$MKL_ROOT -lmkl_lapack -lmkl_p4 example.o -o example
```

## **SLATEC**

SLATEC Common Mathematical Library, Version 4.1, July 1993  
(<http://www.netlib.org/slatec/>) is installed on green located under /usr/local/lib/libslatec.a and /usr/local/lib/libslatec.so.  
Note that most of the corresponding routines in MKL are probably faster.

Example (Dynamic linking using ifort):

```
ifort -lslatec example.o -o example.executable
```

Example (Static linking using ifort):

```
ifort /usr/local/lib/libslatec.a example.o -o example
```