Compiling WIEN2k on INTEL based systems with SUSE LINUX.

Gerhard H. Fecher
Johannes Gutenberg - Universität, Mainz
Institut für anorganische Chemie und analytische Chemie
55099 Mainz, Germany
e-mail: fecher@uni-mainz.de

September 23, 2007

Abstract

A short description is given about *how to compile* WIEN2k. The present description focuses on INTEL based systems and SUSE LINUX.

The tips given below may principally work also on other processors and LINUX distributions, however, they were only tested with INTEL and SUSE configurations.

Be sure to read and understand the manuals.

1 Introduction

For those who do not have a fully licensed but an unsupported (free) non-commercial version of the INTEL Compiler and INTEL MKL, it may not be possible to download the older versions. Newcomers starting from scratch with WIEN2k and Fortran thus may not be able to make use of the options given in the *siteconfig* script for the older Versions.
The following description does not replace the WIEN2k Users Guide [1]. Visit also the Frequently Asked Questions page [1], in particular [2]. If not done already, please, read it carefully before continuing.
In case of troubles, check first the posts in the Users Forum to see if your problem was not already solved a long time ago. Note that LINUX systems are rather heterogeneous, so you may have to try your own way for a successful installation, I hope the following will help.

Good Luck !

1.1 Disclaimer

This work is still in progress, so there will most probably be typos and errors or according to Murphy's law: whatever bad things may happen will happen.
The following description was tested and should run without guarantee on German SUSE 10.1 and 10.2 distributions using the INTEL Fortran 10.0 compiler and MKL 9.1. The tests were performed on INTEL based systems (Notebook with Pentium M 760, 1 GByte RAM, Desktop with Pentium IV 2.8 GHz, 2 GByte RAM, Desktop with D830 3.0 GHz, 1 GByte RAM, dual-CPU Server with two Xeon 3.8 GHz,

[1] http://www.wien2k.at/reg_user/faq

4 GByte RAM, and a Dual-CPU Server with two Dual-Core Xeon 5160, 8 GByte RAM). The latest tested WIEN Version is WIEN2k_07.3 (for earlier compilations see in Appendix, section 7):

Processor		WIEN2k	SUSE	kernel	gcc	IFORT	MKL	Date
Pentium	P IV	07.3	10.1	2.6.16.27-0.9-smp	4.1.0	10.0.026	9.0.018*	31. 08. 2007
Pentium	M 760	07.3	10.1	2.6.16.21-0.13	4.1.0	10.0.026	9.1.023	31. 08. 2007
2 Xeon	5160	07.3	10.1			10.0.026	9.1.023	31. 08. 2007
2 Xeon	3.8 GHz	07.3	10.0	2.6.13-15.8-smp	4.0.2	10.0.026	9.1.023	31. 08. 2007

Note 1: * The MKL 9.1. (and all small sub versions) gave problems during installation at the old PIV system, therefore an 9.0 version was used.
Note 2: Use *cat /proc/cpuinfo* and *cat /proc/version* to find more about youre system.

Even it is rather impossible to respect all possible combinations of 32 and em64t systems, a try is made to give as much as possible information, notes, and remarks on differences between the various systems, versions, and combinations from old to new. If you find a problem and solve it, please let me know, but for sure I am not able to include all possible processors, LINUX distributions, or compilers.

This short introduction **cannot**, however, prevent that one has to read the manuals [1, 3, 4, 5, 6, 7].

2 Short

The short installation instruction is given on the code download page:
http://www.wien2k.at/reg_user/wien2k_download/.
Follow the instructions there and in the scripts. Move the WIEN2k-tar.gz file(s) into a (new) directory which will become your $WIENROOT directory, uncompress the package using (**0x** is the downloaded version !):

```
tar -xvf WIEN2k_0x.tar
gunzip *.gz
chmod +x ./expand_lapw
./expand_lapw
```

This will expand all files and copies various shell-scripts. In $WIENROOT/SRC you find the postscript version of the users guide. Proceed with reading the chapter *Installation*. Supposed a proper Fortran system is installed, you can then configure and compile WIEN2k using:

```
./siteconfig_lapw
```

After successful installation, every user should run:

```
./userconfig_lapw
```

3 Single processor system

Two steps are necessary for running WIEN2k successfully:

- Installing the Fortran compiler and Math Kernel Libraries.
- Compiling the source code.

The following suggestions are supposed to work on every single INTEL processor system starting from Pentium III. They do not make benefit from the particular improvements of dual-core or quad-core CPUs (D8xx, D9xx, Txxxx or Exxxx series) (see next section). They were, however, also successfully tested on dual-CPU Xeon and D830 systems.
Principally the suggestions should also work with AMD processors, at least with some options changed, but this was not tested ! (On AMD systems, you may also wish to check for optimised math-libraries for the BLAS and LAPACK routines.)

3.1 Intel Fortran 10.0 and MKL 9.1

First one needs to install the compiler and the math kernel library (MKL). Both can be downloaded from *www.intel.com*. Note that there are usually different downloadable versions available that are for evaluation or for free non-commercial use. The following was only tested with the free non-commercial versions.
One needs *root* access during the installation, so use either the root account or - easier - open a root shell. Follow the instructions how to install the compiler and the library. During installation one may change the path for the program and library. Below, the standard path is used.
/opt/intel/fc/10.0.026 or
/opt/intel/fce/10.0.026 for the fortran compiler and
/opt/intel/mkl/9.1.023 for the MKL library.
In the following it is assumed that these are the directories where to find the Intel development tools. The suggestions given below have to be changed if a different path is used for one or the other libraries.
Note: Be sure to download and install the correct version of the Fortran compiler depending whether you install on a 32-bit system, on one with 64-bit extensions (em64t), or on an Itanum system. The em64t version is found in the directory */opt/intel/fce* ! Do not mix up either the 32 or em64t systems (Pentium, Xeon), and do not muddle up those with the 64 (Itanum) systems !
After successful installation, one may tell the linker where to find the libraries. To do so, one has to edit the file */etc/ld.so.config* to include the following lines.

```
/opt/intel/fc/10.0.026/lib
/opt/intel/mkl/9.1.023/lib/32
```

for 32-bit systems (Pentium III, old Pentium IV, or Pentium M).
For *em64t* systems with 64-bit extensions (new Pentium or Xeon processors) it should be:

```
/opt/intel/fce/10.0.026/lib
/opt/intel/mkl/9.1.023/lib/em64t
```

After editing and saving the files, run *ldconfig* in */etc* to take the new settings into affect.

Note 1: Use */mkl/9.1.023/lib/em64t* but not */mkl/9.1.023/lib/64*, the latter contains the libraries for Itanum systems.
Note 2: You may also check if the path to other shared libraries is included in */etc/ld.so.config* that may be needed by the linker. However, this may depend on the LINUX version and installation.
The environment variables needed for compiling and linking may be set by means of the scripts given in the directories:
/opt/intel/fc/10.0.026/bin (or *.../fce*) for the compiler (e.g. *ifortvars.sh*) or
/opt/intel/mkl/9.1.023/tools/environment for the MKL (e.g.: *mklvars32.sh* or *mklvarsem64t.sh*).
Check carefully if you need the *.sh* or the *.csh* versions of the scripts, depending on the used shell and LINUX installation.

To make the installation more comfortable one should make some additions to the file */etc/profile.local* or to the hidden file */home/user/.bashrc* in the *user*s home directory. This can be done using the suggestions given in the INTEL script files mentioned above. The *profile.local* or *.bashrc* file should then include the following lines:

```
#
PATH="/opt/intel/fc/10.0.026/bin:${PATH}"
PATH="/opt/intel/idb/10.0.026/bin:${PATH}"
export PATH
#
MANPATH="/opt/intel/idb/10.0.026/man:${MANPATH}"
MANPATH="/opt/intel/fc/10.0.026/man:${MANPATH}"
export MANPATH
#
LD_LIBRARY_PATH="/opt/intel/mkl/9.1.023/lib/32:$LD_LIBRARY_PATH"
LD_LIBRARY_PATH="/opt/intel/fc/10.0.026/lib:$LD_LIBRARY_PATH"
export LD_LIBRARY_PATH
#
INCLUDE="/opt/intel/mkl/9.1.023/include:$INCLUDE"
export INCLUDE
#
INTEL_LICENSE_FILE="/opt/intel/licenses"
export INTEL_LICENSE_FILE
#
```

If there is no *profile.local* in */etc* one has to create it (at least on SUSE systems). Alternatively, one may include the above environment variables in the (hidden) *.bashrc* files of **all** users that are supposed to work with the compiler and libraries. This allow easier changes if one is working with multiple versions of the INTEL tools.
Note: The 32-bit MKL was used here (otherwise replace */32* by */em64t*). It is also assumed that optionally the INTEL debugger is installed in */opt/intel/idb/10.0.026*. On em64t system you may use the */fce* and */idbe* directories for the compiler and debugger path.

Finally, open a user shell from your regular account and check if the environment variables are set correctly for example by means of *echo $LD_LIBRARY_PATH*, and so on.

Option: In cases you are working more often with the INTEL compiler - and not just to install WIEN - it may be helpful to introduce some new environment variables in the *profile.local* or *.bashrc* files that point to the directories were the libraries are located. This saves some typing work and makes it easier to transport the configuration from one computer to another. The additions in *profile.local* or *.bashrc* may be:

```
EXPORT IFLPATH=/opt/intel/fc/10.0.026/lib
EXPORT MKLPATH=/opt/intel/mkl/9.1.023/lib/32
```

for the 32-bit libraries or accordingly with the correct path to the em64t libraries. Those who work with different processors may also want to set some flag for the processor specific compiler switch like *XPROC=-xX*. These switches may not work correctly with the WIEN2k makefiles in their present form. If experienced enough, one may write easily a short script that makes use of these environment variables in order to build the *OPTIONS* file (see below).
If everything is ok [2], the first step is finished and one can go on to set-up WIEN2k.

[2]For beginners, I recommend to test first with a smaller Fortran program whether everything works well, before starting with WIEN2k

3.2 Compiling Wien2k using *siteconfig*

The compilation of WIEN2k is managed by *siteconfig*. The script *siteconfig_lapw* changes the *makefiles* for all programs, runs *make* and copies the executables into the WIEN2k directory. The *makefiles* contain all necessary information how the programs have to be compiled and linked. The script may not be suited for the latest version of the compiler and math libraries because updates or new processors appear rather frequently. However, it can easily be changed for particular needs.
The following are the most important options, concerning the compilation of WIEN2k, that need to be set correctly:

- Compiler to be used,
- Compiler options,
- Linker options,
- Options for the Math Kernel Libraries.

The performance of WIEN2k will depend strongly on the correct settings, thus one should carefully prepare these options.
Note: Running the script for the first time, you may see warnings during the compilation of the programs that *clean* does not find files to be removed. This is clear, at the first compilation there are no object files that can be deleted. However, all other errors should be taken serious.

3.2.1 Compiler options (FOPT):

These options tell the compiler *ifort* how to process the fortran files i.e.: **.f*.
In principal, the only compiler option being essentially needed is *-FR* (same as *-free*). It tells the Fortran compiler ifort that the files are in free-form format. However, there are more useful compiler switches, in particular those who are responsible how the program is optimised and those controlling the numerical accuracy of the compilation. Recommended options are:

```
-FR -w -mp1 -prec_div -pad -ip
```

These settings should be save.
Note 1: In cases of doubt check the INTEL Fortran manuals for particular switches and their use, there are plenty lots of them.
Note 2: For a particular processor one may add the switch -x*Processor*. The possible switches are *-xB, -xK, -xN, -xP, -xT,* or *-xW* depending on the target processor (see also section 6.1 and the compiler manuals). These switches should be used together with the switch *-O3* for a higher optimisation level (*-O2* is used as default and needs not to be given). The program compiled with a *-xX* switch may not run on computer with a processor being not supported by that switch. It may also be necessary to link *libsvml* if these compiler switches are used.
Note 3: The switches *-mp1, -prec_div, -pad* are used to maintain floating point precession in the calculations. The switch *-w* suppresses that warnings are printed during compilation.
Note 4: The switch *-ip* performs single-file inter procedural optimisation, that is optimisation between subroutines located in the same *.f* file.
Note 5: Presently, it seems that only the subroutine *SRC_lapw1/hamilt.f* makes use of the macro INTEL_VML, therefore the switch *-DINTEL_VML* is not given in the options, at the present stage.
Note 6: Some switches being suggested in the original (or mostly in some older versions) *siteconfig_lapw* script are not used here for different reasons: *-pc80* is the default and thus not needed. *-Vaxlib* is from very old versions of the INTEL compiler and not longer in use.
The compiler options with optimisation for a Pentium M processor may look like:

```
-FR -w -mp1 -prec_div -pad -ip -DINTEL_VML -O3 -xB
```

Note 1: The use of these options may need some additional libraries (see below).
Note 2: Replace *-xB* by *-xP* if the target processor is for example a PIV with em64t extensions or a Xeon.

3.2.2 Linker options (LDFLAGS):

These options are passed from the compiler *ifort* to the LINUX linker *ld* in order to make the executables from the object files (**.o*). They also tell the linker which non-standard libraries should be used.
For dynamic linking of the libraries use:

```
-L/opt/intel/mkl/9.1.023/lib/32 -lguide -lpthread
```

Note: These settings should be save. Depending on the compiler switches, in particular for optimisation, one may need to give more libraries. (See also the following notes.)
The switch *-no-ipo* may be necessary on some systems. It prevents some Linker errors (see Section 3.2.3).
For static linking of the INTEL provided libraries one may use:

```
-L/opt/intel/fc/10.0.026/lib -i-static -lguide_stats -lsvml -lpthread
```

Note 1: *-L/PATH* tells the linker *ld* where to search for the library files. On some systems, *ld* searches in *-L/PATH* first for shared libraries *libxyz.so* and then for static libraries *libxyz.a*. If it finds the shared library then it will perform dynamic linking even so a static library is present (see also note below).
Note 2: One probably needs to give *-L/opt/intel/fc/10.0.026/lib* because there one finds the *libguide_stats*, and it will not be found if only giving the path to MKL, like used in the dynamic example.
Note 3: The switch *-i-static* tells ifort that only the INTEL libraries should be linked statically.
Note 4: The switch *-lsvml* is used together with processor specific optimisation like *-O3 -xB* or *-O3 -xP*.
Note 5: *libguide* is found in both the ifort and the MKL library path. It seems that at least the static versions are identical at present. This may change with other versions of the compiler and MKL.
Note 6: *libpthread* is not an INTEL provided library and should be found in the standard linker path.
Note 7: The switch *-static-libcxa* being suggested in older *siteconfig_lapw* scripts is not used here, as it seems that *libcxa* is not used in all parts of the program. This switch is used to link the *C++* compatibility libraries statically. The alternative switch is *-dynamic-libcxa* (see also INTEL manuals). **WARNING:** The switch *-static-libcxa* does not work with gcc on a 64-bit SUSE 10.0 with multiprocessor kernel !
Note 8: For static linking of a particular library one may also give the static library file with its full path explicitly like for example *MKLPATH/libguide.a* instead of *-lguide*. On the other hand, the type of linking can be changed by using the switches *-Bdynamic* for dynamic linking and *-Bstatic* for static linking. These switches are always acting on the library given after the switches. That means: *-L/opt/intel/mkl/9.1.023/lib -Bstatic -lguide -Bdynamic -lpthread* links *libguide* statically and *libpthread* dynamically.
Note 9: The behaviour of *ld* depends also on the settings in */etc/ld.so.config* as regards shared libraries. (*ldconfig -p* reports which shared libraries are known in the linker cache.)
Note 10: Be sure whether your compiler and libraries are in the *../fc* or the *../fce* directories depending on the version of the compiler 32 or em64t, respectively. Otherwise you will receive some messages on incompatible libraries and missing routines.

3.2.3 Linker problem with ifort 9.x and SUSE 10.1:

There is a problem with this combination as well as with other distributions, the compiler may report the following error:

```
IPO Link error: file not found "("
```

In case that this appears, you may like to add one of the switches *-no-ipo* or *-O0* to the linker flag (not to the compiler flags !) such that it looks like, for example:

```
-no-ipo -L/opt/intel/fc/10.0.026/lib -lguide -lpthread
```

It seems that some parameter is incorrectly passed to the linker, or interpreted wrong by it. Usually the interprocedural optimization are not used for the WIEN2k compilation. The error message does *not longer* appear in ifort 10.0 (SUSE 10.x).
In case you like to do experiments with the *-ipo* flag during compilation, you may need to create in each directory the two empty files: *(* and *AS_NEEDED*, e.g. by using:

```
echo NULL > \(
echo NULL > AS_NEEDED
```

It helps, but I do not like to know why !

3.2.4 BLAS-LAPACK options (R_LIBS):

These are special options for the linker concerning the MKL or other blas-lapack libraries. These libraries are not needed in all of the WIEN2k programs but only in some particular ones.
In case of a 32-bit system, the switches:

```
-L/opt/intel/mkl/9.1.023/lib/32 -lmkl_lapack -lmkl_ia32 -lguide -lpthread
```

should be save.

Note 1: The switch *-no-ipo* may be necessary on some systems. It prevents some Linker errors (see Section 3.2.3.
Note 2: For Pentium processors with 64-Bit extensions one may use *-L/opt/intel/mkl/9.1.023/lib/em64t* and *-lmkl_em64t*.
Note 3: For some of the Suse 9.3 (or newer) installations, there was no need to give the *libpthread* library explicitly, thus one may drop the *-lpthread* switch, it is probably somewhere in the linker path. On 64-bit SUSE 10.0 systems, the 64-bit version of the library will be chosen automatically in the case of dynamic linking.
Note 4: The reason to give some libraries twice (in *LDFLAGS* and *R_LIBS*) is the behaviour of the LINUX linker *ld* that needs to have the libraries given in a particular series if using the *-lxyz* form. In that case it searches *-L/PATH* only once. For example *libmkl_lapack* needs *libmkl_ia32* (or *libmkl_em64t*) and *libguide*, so *libmkl_ia32* and *libguide* have to be given after *libmkl_lapack* - like in the example given above - otherwise the linker complains about missing routines. On the other hand, the *libmkl_lapack* is only needed in few makefiles, but *libguide* or *libpthread* may be needed in other subroutines without *libmkl_lapack*. Therefore some libraries are needed in *LDFLAGS* and as *R_LIBS* comes in the make files after *LDFLAGS*, one may need to give some libraries another time.
At the command line, this type of searching behaviour of *ld* may be overcome by using *-(libraries -)* causing that the *-L/PATH* is searched multiply. *libraries* may be explicit files or *-l* options. This does not work, however, across different flags like used for example in the makefiles of WIEN (*LDFLAGS*, *R_LIBS*). It should be noted that ifort passes command line options to *ld* making use of the switch *-Bl*.
The above given options link the MKL libraries, at least some parts, statically even without use of the *-static* switch! The reason is that the static and dynamic versions of the MKL libraries have different names.
For dynamic linking of the MKL one needs to link the shared libraries *libzyx.so* instead of *libxyz.a*, these have slightly different names, such that the *R_LIBS* options may look like:

```
-L/opt/intel/mkl/9.1.023/lib/32 -lmkl_lapack64 -lmkl -lmkl_p4 -lvml
```

Note: If any errors occur during the link process complaining about missing subroutines, one can easily check which of the static library archives (*libxyz.a*) contains the missing subroutine. This is usually very helpful even so one may have to check a lot of files if the missing routine is in one of the general or GNU libraries. If there is a shared library (*libxyz.so*) with the same name then it should usually contain the same routines like the archive, and thus may be added to the list of libraries to be linked dynamically.
Moreover, finding the correct library may prevent to write patches just because a subroutine is moved - for whatever reason - from one to another library. This was for example the case some time ago with *libpthread* were the missing subroutine (*pthread_atfork*) was moved to *libpthread_nonshared*, at least for SUSE systems (see in */usr/lib*). Similar, a complain about a missing *pthread_atfork* may appear if you use the em64t version of the compiler but give the 32-bit libraries, however, for different reason.

3.2.5 Making Wien2k

Follow the installation instructions given on the WIEN2k code download page [3].

a) Running site_config for the first time
If installing WIEN2k the first time, one has to set correctly all options in *site_config* as suggested above. Among others, *site_config* will create a file *OPTIONS* where all the information about the compiler is stored. This file is very helpful for later changes, see next step.

b) Changing options and recompiling
Before using the script siteconfig (siteconfig_lapw) repeatedly, one may change the file *OPTIONS* in the WIEN2k directory. This file contains the compiler and library specific definitions. For a non-parallel version like discussed here, it should contain for example the lines (**replace *-xX* by the correct processor specific option !**):

```
current:FOPT:-FR -w -mp1 -prec_div -pad -ip -DINTEL_VML -O3 -xX
current:LDFLAGS:-L/opt/intel/fc/10.0.026/lib -i-static -lguide_stats -lsvml -lpthread
current:R_LIBS:-L/opt/intel/mkl/9.1.023/lib/32 -lmkl_lapack -lmkl_ia32 -lguide -lpthread
```

Note 1: This may link different libraries in different parts of WIEN2k, depending whether the accompanied makefile uses *R_LIBS* or not !
Note 2: Use the correct path on em64t systems !

3.2.6 Static versus dynamic linking

Actually, full static linking using *-static* may not work for "large" cases (see also Sections 5, 5.1, and 6.3 on errors). On the other hand, the gain of performance by static linking is in many cases not that tremendous. The drawback of (full) dynamic linking is, that the executables can not be transferred easily from one to another computer. If setting up on more than two computers, it becomes boring that one needs always to install the development tools such that the dynamic libraries are found correctly. A way to overcome this at least partially is to link *all* INTEL provided libraries statically and only the LINUX libraries dynamically. If the LINUX installations on the different computers are identical (check carefully if using different processors) then the executables should work on those computers, too. In that case one needs only to transfer the executables and scripts (do not forget the directories for the templates, manual, etc.) of WIEN2k from one to another computer instead of going always through a complete install procedure. The Appendix gives some ideas which linker switches may be used and how to find out which shared libraries are needed for different models.

[3] http://www.wien2k.at/reg_user

4 Dual-core or dual processor systems.

The set-up for the *siteconfig_lapw* script are principally the same as described above for a single processor system. The only differences may be the libraries needed to be linked and the optional compiler switch *xX = xP or xT* for processor specific optimisation (see also: Appendix). A *simple* case for a system with 2 dual-core XEON processors with *em64t* extensions may look like:

```
FOPT:    -FR -w -mp1 -prec_div -pad -ip -O3 -xT
LDFLAGS: -L/opt/intel/fce/10.0.026/lib -lguide -lpthread
R_LIBS:  -L/opt/intel/mkl/9.1.023/lib/em64t -lmkl_lapack -lmkl_em64t
```

For other cases follow the instructions given above for the single processor guide (see Appendix for more).

The number of processors installed in the system is reported by:

```
cat /proc/cpuinfo
```

The main difference in setting up *"dual"*-systems is to influence the behaviour of the MKL [4] being already prepared for parallel execution (see: [7]).
To force the library to serial mode, the environment variable *MKL_SERIAL* should be set to *YES*. *MKL_SERIAL* is not set by default. It works regardless of the *OMP_NUM_THREADS* value.
Furthermore, the parallel execution of the MKL routines is controlled by the environment variable *OMP_NUM_THREADS* [5], that is usually not set.
The number of threads should be set in the shell from which one starts WIEN2k (or for permanent use in *profile.local* [6]) to the desired value *N=1,2,..,N_{max}* where N_{max} is the maximum number of processors (cores) in the system [7] using:

```
export OMP_NUM_THREADS=N
```

Note 1: After changing the values for *OMP_NUM_THREADS* one may have to kill and restart *w2web* if it was already running.
Note 2: For *"dual"*-systems where more than one job - and not just a single WIEN calculation - is planned to be executed at once, it is suggested to start by setting *OMP_NUM_THREADS=1* and then run some benchmark tests with increasing numbers to find the best performance that fits to particular needs.

The consequence of the thread-setting is demonstrated in the following table giving the execution times for different number of jobs and threads (see Appendix for CPU and Wall times).

Table 1: Execution times [min:sec] for test_case on a dual Xeon machine.

	threads	1	2	4
1×	1 job	2:36	1:57	2:05
1×	2 jobs	3:11	3:25	3:22
2×	1 job	5:12	3:54	
1×	4 jobs	6:27	6:26	6:38
2×	2 jobs	6:22	6:50	
4×	1 job	10:24	7:48	

[4] The following was only tested with SUSE 10.1 (kernel).
[5] The default of OMP_NUM_THREADS is the number of processors installed while generating the executable [5]. If not set, the default for the MKL is one.
[6] Read carefully *all* files of the MKL and Fortran documentation before setting other values than 1.
[7] Note: The execution may be slowed down if less processors are available then specified !

The calculations in Tab.1 were performed using a version compiled with full processor specific optimisation and all INTEL provided libraries were linked statically. The serial and parallel jobs were started from a script file with the execution time taken from a date command at beginning and end of the script. No other background jobs were running.
From the results in Tab.1, one expects that *OMP_NUM_THREADS=2* is suited if just one single job is executed. This changes for two jobs in parallel where *OMP_NUM_THREADS=1* is obviously the better choice. In the particular example, the overall time for running two lapw1 calculations with 1 thread is 191 seconds if executing them in parallel but 233 seconds if running them consecutively with 2 threads.
Note: One should keep in mind, however, that these findings are for a very special example and set-up. The general behaviour, if different programs run at once on a different system, may lead to other conclusions. So check for your needs.

Another option for the compiler is to make use of the auto-paralleliser by means of the compiler switch *-parallel* (**not** the macro *-Dparallel* !). However, no particular enhancement for lapw1 was found on a dual-Xeon machine. This may be different for other parts of WIEN but was not yet tested. In other cases a slowing down of programs was observed if using *-parallel* in particular together with the mkl.

5 Runtime errors

This section is to prevent questions like *"Wien stops with an error! Why?"* or *"Wien does not run, please give advice."*. Such questions can not be answered ! Some of the tips given here are not just WIEN2k specific, but may be helpful also for other things programmed in Fortran.
If any error occurs, first check all input files and switches to run the WIEN2k scripts (see [1]), second check the installation (see this file). If everything is really well then try to figure out what else may have caused the error. The following may give some ideas about compiler and library related errors.
It may appear that one experiences some run time errors like *SIGSEGV message* [8]. In order to heal such errors, one has to know in detail where and why such errors occurs.
An oftenly reported error is: SIGSEGV (segmentation fault) [9]. A common answer is to increase the stack size, however, this may neither have the wanted effect (see sections 5.1 and 6.3) nor heal the error SIGSEGV (program stack overflow). Actually, I never experienced that the stack size setting was the cause for a SIGSEGV (segmentation fault) error but this may be different on other systems. On the other hand, it may appear that a segmentation fault is the result of a badly handled stack error. This example shows that one has to take care what the run-time error really tells.
If a SIGSEGV (segmentation fault) error appears, it is usually not given in what particular part of the program. Usually one receives an error like: *Error in lapw123* and something more with *show STDOUT*, so what to do ?. To find out where the error appears one may recompile the subroutine under question with the compiler option *-traceback*. One may either change the compile options in *siteconfig* and recompile either all programs or only the one under question. Experienced users will change the *makefile* of the affected subprogram by hand and start *make*. Traceback will hopefully report where the error appeared, but note that the program will probably become slower. If the optimisation level is not set to *-O0* then one may additionally use the switch *-fp* to allow for a trace back with optimisations switched on (see the compiler manuals for details).
In general, segmentation faults occur very often if an array element is accessed in the wrong way. Fortran arrays have as standard the first element a(1) and not a(0) if the dimension a(DIM) is used. To use a(0) one needs the dimension a(0:DIM). Trying to access a(0), a(-1), or a(i) with i>DIM might cause segmentation faults. Note that this may also be produced by a simple input and not only a program error.

[8] This is not a particular problem of WIEN2k, indeed. The WIEN2k programmers do a hard job to keep it free of bugs.
[9] On German systems you may receive instead: Speicherschutzverletzung. One may set LANG to en_US if preferring the English version of the error messages.

The switch *-CB* (same as *-check bounds*) performs run-time checks on whether array subscripts are within the declared dimensions and thus it may be very helpful to find such errors. In some cases, an undetected stack error may cause that an index of an array element is transferred in a wrong way to a subroutine and thus causing the segmentation fault. Similarly, some numerical over- or underflow can cause wrong indexing of arrays. This may sound complicated, but now you see, why it is not easy to trust the error messages, and finding errors correctly might be hard work.
Further, the option *-g* may be used that produces code for debugging. See the INTEL debugger manual for more information about working with the debugger.
Note 1: All debugging switches like *-traceback, -CB,*, *-g*, or others may slow down the program and will increase its size. Thus, it is a good idea to use them only for the effected subroutine and finally they should not be used for the version compiled for every days work.
Note 2: Errors seem to vanish sometimes if using a lower optimisation level. Indeed, this may be a compiler problem leading to some type of *"over-optimisation"* in particular if rearranging floating point operations. However, there is most probably also some *dirty* code [10]. In such cases one may check if there are for example any not initialised, undeclared variables, or other quick and dirty solutions by switching on the warnings during compile time. As an example: comment all *IMPLICIT* statements and use *IMPLICIT NONE* instead to find if there is a real variable named c but not declared as real.
Another appearance of segmentation faults seems to be related to some libraries not belonging to the INTEL compiler or MKL, so check which version of the libraries is linked to the compilation. For dynamically linked libraries, this can be checked by means of *ldd*. Go to the directory containing the program that caused the error, e.g.: for *lapw1* go to */SRC_lapw1*. Call *ldd lapw1*, this will show you which libraries are linked dynamically to *lapw1*. One may also use *ld -M lapw1* to find out which routines and symbols are used.
For more, see the *Run-Time Error Messages* section of [4].
In case of troubles, please check and report always which "small" x0.0.yyy version was used for the compilation. Most probably some libraries are changed between such small versions, and things downloaded tomorrow are different from those yesterday.

5.1 Limitations that cause errors.

The INTEL Fortran compiler and LINUX have obviously some limitations (see also section 6.3) that also cause SIGSEGV or similar nasty errors, if exceeding them.
1) The largest array size is $2^{31}-1$ that is 2 GByte even on some *em64t* systems [11]. Larger arrays will cause *segmentation faults* or similar errors. Unfortunately, these errors lead to serious crashes but their cause is actually not found by any of the debugging features provided by LINUX or the compiler. In some cases the crash may be that bad that one has to restart the computer (check for performance loss and other strange things).
2) Full static linking with *-static* causes crashes most probably if the necessary stack-size exceeds 2 GByte. This seems to be partially a 32-bit LINUX problem with *libpthread*, at least on many systems (see also section 6.3). Again, this limitation leads to serious crashes and its cause is not found by any of the debugging features.
It is interesting to note, that the Windows version of the INTEL Fortran compiler, coming with its own Linker, is able to handle those two problems correctly. (This just means that the linker has a better performance in detecting the stack size, but not automatically that the stack handling of Windows is better.)

[10] This is not a problem of the WIEN programmers, I experienced from my own programs that *dirty code* is *self-reproducing*!
[11] At present (whenever present is), one may have to use a "*real* 64-bit" processor like Itanum or non-INTEL systems to overcome that limitation.

6 Appendix

6.1 Processor specific switches *-xX*

The following processor specific optimisation switches (*-xProcessor*) are available (not Itanum) in ifort 10.0:

- -xB : Pentium M,
- -xK : Pentium III and Athlon XP,
- -xN : Pentium IV and compatible, with new optimisations,
- -xP : Pentium IV and Xeon, with SSE-3 instructions and em64t,
- -xT : Intel Dual or Quad Core Pentium and XEON,
- -xW : Pentium IV, Xeon, (em64t), Athlon 64, and Opteron.

Note 1: These are suggestions made in the INTEL manuals (Compiler Options [3] and Quick-Reference Guide), here only *-xT, -xB, -xW* were tested with INTEL processors. Not all switches are valid on *em64t* systems. Alternatively, one may use *-axProcessor* that always produces additionally generic iA32 code for compatibility.
Note 2: Alternatively, one may also use combinations of switches like *-axXY* or *-xXY*.

If needed, check the old compiler manuals (*ifort* 8.x, 9.x, or *ifc* 7.x) for the processor specific optimisation switches.

6.2 Dynamically linked libraries in *lapw1*

The following results are from SUSE 9.3 on a Pentium M and SUSE 10.0 on a dual Xeon. They may differ if using different systems. The examples may serve to guide how to find which dynamic libraries are used by the program. Some benchmark results are given for comparison.

a) The options for dynamic linking (not MKL) and without particular optimisations

```
-FR -w -mp1 -prec_div -pad -ip
-L/opt/intel/fc/lib -lguide -lpthread
-L/opt/intel/mkl/lib/32 -lmkl_lapack -lmkl_ia32
```

have the result that the following libraries are linked dynamically:

```
linux-gate.so.1 =>  (0xffffe000)
libguide.so => /opt/intel/fc/lib/libguide.so (0x40019000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x40060000)
libimf.so => /opt/intel/fc/lib/libimf.so (0x40072000)
libm.so.6 => /lib/tls/libm.so.6 (0x4024e000)
libc.so.6 => /lib/tls/libc.so.6 (0x40271000)
ibdl.so.2 => /lib/libdl.so.2 (0x4038a000)
/lib/ld-linux.so.2 (0x40000000)
```

The maximum CPU time was 457.24s for the *test_case* on a Pentium M 760.

The similar options for a processor with em64t extension

```
-FR -w -mp1 -prec_div -pad -ip
-L/opt/intel/fce/lib -lguide -lpthread
-L/opt/intel/mkl/lib/em64t -lmkl_lapack -lmkl_em64t
```

have the result that the following libraries are linked dynamically:

```
to be done
```

The maximum CPU time was 184.43s for the *test_case* on a dual Xeon machine with the number of threads set to $OMP\_NUM\_THREADS = 2$.

b) The options for full dynamic linking without particular optimisations (only default: *O2*) on a 32-bit system

```
FOPT:    -FR -w -mp1 -prec_div -pad -ip
LDFLAGS: -L/opt/intel/fc/lib -lguide -lpthread
R_LIBS:  -L/opt/intel/mkl/lib/32 -lmkl_lapack64 -lmkl -lmkl_p4 -lvml
```

have the result that the following libraries are linked dynamically:

```
linux-gate.so.1 =>  (0xffffe000)
libguide.so => /opt/intel/fc/lib/libguide.so (0x40019000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x40060000)
libmkl_lapack64.so => /opt/intel/mkl/lib/32/libmkl_lapack64.so (0x40072000)
libmkl.so => /opt/intel/mkl/lib/32/libmkl.so (0x40341000)
libmkl_p4.so => /opt/intel/mkl/lib/32/libmkl_p4.so (0x403a1000)
libvml.so => /opt/intel/mkl/lib/32/libvml.so (0x4082a000)
libimf.so => /opt/intel/fc/lib/libimf.so (0x40864000)
libm.so.6 => /lib/tls/libm.so.6 (0x40a40000)
libc.so.6 => /lib/tls/libc.so.6 (0x40a63000)
libdl.so.2 => /lib/libdl.so.2 (0x40b7c000)
/lib/ld-linux.so.2 (0x40000000)
```

The maximum CPU time was 457.04s for the *test_case* on a Pentium M 760.

c) The options for partially static linking with full optimisation for a Pentium M processor

```
-FR -w -mp1 -prec_div -pad -ip -DINTEL_VML -O3 -xB
-L/opt/intel/fc/lib -i-static -lguide -lguide_stats -lsvml -lpthread
-L/opt/intel/mkl/lib/32 -lmkl_lapack -lmkl_ia32
```

have the result that the following libraries are linked dynamically:

```
linux-gate.so.1 =>  (0xffffe000)
libguide.so => /opt/intel/fc/lib/libguide.so (0x40019000)
libguide_stats.so => /opt/intel/fc/lib/libguide_stats.so (0x40048000)
libsvml.so => /opt/intel/fc/lib/libsvml.so (0x40087000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x400f3000)
libm.so.6 => /lib/tls/libm.so.6 (0x40105000)
libc.so.6 => /lib/tls/libc.so.6 (0x40128000)
libdl.so.2 => /lib/libdl.so.2 (0x40242000)
/lib/ld-linux.so.2 (0x40000000)
```

The additional libraries are *libguide_stats* and *libsvml* whereas the compatibility library *libimf* is now linked statically.
The maximum CPU time was 445.6s for the test_case on a Pentium M 760.

d) The options for static linking of **all** INTEL provided libraries with full optimisation for a Pentium M processor

```
-FR -w -mp1 -prec_div -pad -ip -DINTEL_VML -O3 -xB
-L/opt/intel/fc/lib -i-static -Bstatic -lguide -lguide_stats -lsvml -Bdynamic -lpthread
-L/opt/intel/mkl/lib/32 -Bstatic -lmkl_lapack -lmkl_ia32 -lguide -Bdynamic -lpthread
```

have the result that the following libraries are linked dynamically:

```
linux-gate.so.1 =>  (0xffffe000)
libpthread.so.0 => /lib/tls/libpthread.so.0 (0x40031000)
libm.so.6 => /lib/tls/libm.so.6 (0x40043000)
libc.so.6 => /lib/tls/libc.so.6 (0x40066000)
/lib/ld-linux.so.2 (0x40000000)
```

The maximum CPU time was 445.35s for the test_case on a Pentium M 760.

The similar options for a processor with em64t extension

```
-FR -w -mp1 -prec_div -pad -ip -DINTEL_VML -O3 -xP
-L/opt/intel/fce/lib -i-static -Bstatic -lguide -lguide_stats -lsvml -Bdynamic -lpthread
-L/opt/intel/mkl/lib/em64t -Bstatic -lmkl_lapack -lmkl_em64t -lguide -Bdynamic -lpthread
```

have the result that the following libraries are linked dynamically:

```
to be done
```

The Maximum CPU times were 157s, 173s, or 281s for the *test_case* on a dual Xeon machine with *OMP_NUM_THREADS*= 1, 2, or 4, respectively. The accompanied maximum wall clock times were 156s, 117s, and 125s executing a single job from the command line of a ssh shell (This times may differ if the *test_case* is started from *w2web*). Starting two *test_case* jobs in parallel (using a shell script), the maximum CPU times were 192s, 321s, and 303s with wall clock times of 191s, 205s, and 202s for each job, again for *OMP_NUM_THREADS*= 1, 2, or 4.

e) Finally, the use of the linker switch *-static* should produce executables without any dynamically linked libraries (message from ldd: *not a dynamic executable*).
Warning: It seems, however, that statically linked executables for WIEN2k, and most probably also for other large programs, lead presently for certain configurations still to segmentation faults, for not well known reasons. It seems that small cases (few atoms) work, whereas "larger" cases cause segmentation faults (or other crashes) in lapw1(c) or lapw2(c). In particular the *test_case* does not work with *-static*. The problem seems to be related with the LINUX library *libpthread*, as the static linking of all INTEL provided libraries does not cause problems, at least with the *test_case* (see above). The failure is most probably due to a LINUX 2 GB limitation [6], see next section.

6.3 The Intel homepage tells:

The INTEL Fortran Compilers 8.0 or higher allocate more temporaries on the stack than previous INTEL Fortran compilers. Temporaries include automatic arrays and array sub-sections corresponding to actual arguments. If the program is not afforded adequate stack space at runtime relative to the total size of the temporaries, the program will terminate with a segmentation fault on LINUX. On LINUX, the stack space can be increased using (e.g. *ulimit -s unlimited*) for bash shell or (e.g. *limit stacksize unlimited*) for csh shell.
For INTEL Fortran Compilers 10.0: The heap-arrays compiler option directs the compiler to put the automatic arrays and arrays created for temporary computations on the heap instead of the stack.
Note: The size of *"unlimited"* varies by LINUX configuration, so you may need to specify a larger, specific number to *ulimit* (for example, 999999999). On LINUX also note that many 32bit LINUX distributions ship with a pthread static library (*libpthread.a*) that at runtime will fix the stacksize to 2093056 bytes regardless of the *ulimit* setting. To avoid this problem do not link with the *-static* option or the *-fast* option. Instead of *-fast*, use options: *-ipo -O3 -no-prec-div -xP*. This only affects the 32bit LINUX distributions and does not apply to the 64bit LINUX distributions.

7 Earlier tested configurations.

:
The above given description was tested during the last years and should run without guarantee on German SUSE 9.2, 9.2 (US), 9.3, 10.0, 10.1, and 10.2 distributions using the INTEL Fortran 8.0 to 10.0 compiler together with MKL 8.0 to 9.1. The tests were performed on several INTEL based systems (To name a few: Notebook with Pentium M 760, 1 GByte RAM, Desktop with Pentium IV 2.8 GHz, 2 GByte RAM, Desktop with D830 3.0 GHz, 1 GByte RAM, dual-CPU Server with two Xeon 3.8 GHz, 4 GByte RAM, and a Dual-CPU Server with two Dual-Core Xeon 5160, 8 GByte RAM). The tested WIEN Versions were WIEN2k_05, WIEN2k_06, and WIEN2k_07. Some of the more recent tests are given in the following table:

	Processor		WIEN2k	SUSE	IFORT	MKL	Date
1	2 Xeon	3.8 GHz	05.6	10.0	9.0.021	8.0.19	
2	Pentium	M 760	06.2	9.3	9.0.021	8.0.19	
3	Pentium	D 830	06.2	10.0	9.0.031	8.0.2.004	
4	Pentium	M 760	06.4	10.1	9.1.041	8.1.014	4. Jan. 2007
5	Pentium	D 830	07.1	10.1	9.1.041	8.1.014	27. Jan. 2007
6	Pentium	M 760	07.1	10.1	9.1.041	8.1.014	28. Jan. 2007
7	Pentium	M 760	07.2	10.1	10.0.023	9.1.018	11. Jun. 2007
8	2 Xeon	3.8 GHz	07.2	10.0	10.0.023	9.1.018	12. Jun. 2007
9	2 Xeon	5160	07.2	10.2	10.0.023	9.1.018	12. Jun. 2007
10	Pentium	P IV	07.3	10.1	10.0.026	9.0.018*	31. Aug. 2007
11	Pentium	M 760	07.3	10.1	10.0.026	9.1.023	31. Aug. 2007
12	2 Xeon	5160	07.3	10.1	10.0.026	9.1.023	31. Aug. 2007
13	2 Xeon	3.8 GHz	07.3	10.0	10.0.026	9.1.023	31. Aug. 2007

* The MKL 9.1.0xx (and all small .0xx sub versions) gave problems during installation at the old PIV system, therefore an 9.0 version was used.

8 Acknowledgement

I like to thank all those who gave helpful advices in the WIEN users forum and in particular L. D. Marks. Special thanks go to U. Stumm who tells me which character to type when I receive for the 111^{th} time not what I expect (I should know meanwhile that it is ldd and not ld, telling me which libraries are linked dynamically).
Finally, I admire the patience of Peter after so much junk mail.

References

[1] *WIEN2k-Usersguide* (pdf); http://www.wien2k.at/reg_user/textbooks/usersguide.pdf

[2] *Compiling WIEN2k under Linux using ifc (pgf90) and optimized libraries*; http://www.wien2k.at/reg_user/faq/OptimizingWIEN2k.htm

[3] Intel Fortran Compiler **Options**.

[4] Intel Fortran Compiler for Linux **Building Applications**.

[5] Intel Fortran Compiler **Optimizing Applications**.

[6] Intel Fortran Compiler 10.0 for Linux **Release Notes**.

[7] Intel Math Kernel Library 9.1 **Reference Manual**.