WIEN2k 软件的安装

WIEN2k 软件是在 Linux 系统下安装、运行的第一原理计算软件，其编译、安装主要参考 WIEN2k_Users-Guide[?] 的第 11 章 Installation and Dimensioning。

除了 Linux 操作系统，编译、安装环境建议如下，(以下括号中的版本号都是我当前使用的版本)：

- csh 或 tcsh
- intel 编译器 (ifort / icc) (版本 13.0.0) + MKL 库函数
- 2017 版需要 intel_2015及更高版本的编译器和库函数
- ∗VASP-5.4.4需要 intel_composer_xe_2013_sp1.3.174及更高版本的编译器和库函数
  否则编译文件 wave.f90 时会提示: “指针数组连续未声明”:
  wave.F(669): error #8378: Pointer array is not contiguous unless it is declared CONTIGUOUS. [CPTWFP]
  WUP%CPTWFP=> W%CPTWFP(:,:,:,1)
  ----------^

为了支持 mpi 版本的 WIEN2k，还需要如下支持：

- mpi 编译器 openmpi (版本 openmpi-1.8.4) 或 mpich (版本 mpich-3.1.4)
- fftw 库 (版本 fftw-3.3.4)

注意：以下安装假定系统和环境设置的默认编译器和编译环境为 intel 编译器环境和 mkl 库 (即~/.bashrc 中的环境设置为)：

```
################ INTEL_compiler ####################
. /home/soft/intel/Compiler/11.0/083/bin/intel64/ifortvars_intel64.sh
. /home/soft/intel/Compiler/11.0/083/bin/intel64/iccvars_intel64.sh
################ INTEL_mkl ####################
export LD_LIBRARY_PATH=/home/soft/intel/mkl/10.1.2.024/lib/em64t/:$LD_LIBRARY_PATH
```

或者如果直接安装了 INTEL 编译器完整版

```
################ INTEL_compiler_mkl ####################
source 你的 INTEL 安装目录/bin/compilervars.sh intel64
```

1 openmpi 的安装：

```
tar -xvzf openmpi-1.8.4.tar.gz
cd openmpi-1.8.4
./configure --prefix=你的 MPI 安装目录 CC=icc CXX=icpc F77=ifort FC=ifort --enable-static
make && make install
```

安装完毕后，新增~/.bashrc 中的环境变量设置：

```
############## OPENMP ###########
export PATH=你的 MPI 安装目录/bin:$PATH
```

注意：如果安装 mpich，操作完全类似

注意：查看有关 mpi 本身指向的编译器等信息的命令 mpiexec -info

MPI 编译器封装的编译器查看命令: mpif90 –version/-v 两个命令略有区别

2 fftw 的安装：

一般地，Intel 的 MKL 自带了 FFTW 的库函数，其中

- 头文件位于 $MKLROOT/include/fftw
- 静态 fortran 库函数位于 $MKLROOT/interfaces/fftw3xf/libfftw3xf_intel.a
- 静态 c 库函数位于 $MKLROOT/interfaces/fftw3xc/libfftw3xc_intel.a

注意: $MKLROOT/include/fftw 的文件包括

fftw3.f fftw3_mkl.f fftw3_mkl.h fftw3-mpi_mkl.h fftw.h fftw_threads.h rfftw_mpi.h fftw3.h fftw3_mkl_f77.h fftw3-mpi.h fftw_f77.i fftw_mpi.h rfftw.h rfftw_threads.h

因此如果源代码包括 fftw3.f03 fftw3.h fftw3-mpi.f03 fftw3-mpi.h 等这样的文件

则必须安装 **fftw** 库函数

tar -xvzf fftw-3.3.4.tar.gz

cd fftw-3.3.4

./configure --prefix=你的 FFTW 安装目录 CC=icc MPICC=mpicc F77=ifort FC=ifort MPILIBS=-I你的 MPI 安装目录/include --enable-mpi --enable-float --enable-threads --enable-sse --enable-sse2

make && make install

如果要生成 fftw/fftw_mpi 动态库，用如下的设置

./configure --prefix=你的 FFTW 安装目录 CC=icc CXX=icpc CFLAGS=-fPIC F77=ifort FC=ifort --enable-shared 仅出现动态库

./configure --prefix=你的 FFTW 安装目录 CC=icc MPICC=mpicc F77=ifort FC=ifort MPILIBS=-I你的 MPI 安装目录/include --enable-mpi --enable-shared

如果要生成包含fftw3f/fftw3f_mpi/_thread等动态库，用如下的设置

./configure --prefix=你的 FFTW 安装目录 CC=icc MPICC=mpicc F77=ifort FC=mpif90 MPILIBS=-I你的 MPI 安装目录/include --enable-mpi --enable-shared --enable-float --enable-threads --enable-sse --enable-sse2

如果用gcc/gfortran编译器编译

./configure --prefix=你的 FFTW 安装目录 CC=gcc MPICC=mpicc F77=gfortran FC=mpif90 MPILIBS=-I你的 MPI 安装目录/include --enable-mpi --enable-shared --enable-float --enable-threads --enable-sse --enable-sse2

3 库函数检查的有关命令

nm

-D 或–dynamic 显示动态符号。该任选项仅对于动态目标 (例如特定类型的共享库) 有意义

-g 或–extern-only 仅显示外部符号

-u 或–undefined-only 仅显示没有定义的符号 (那些外部符号)

ldd

ldd 命令用于判断某个可执行的 binary 文件含有什么动态函式库

-d –data-relocs 执行符号重部署，并报告缺少的目标对象 (只对 ELF 格式适用)

-r –function-relocs 对目标对象和函数执行重新部署，并报告缺少的目标对象和函数 (只对 ELF 格式适用)

readelf

readelf 命令查看共享库的依赖库 (NEED) 和搜索名 (SONAME)

-d 查看动态库的真实名字

4 WIEN2k 的安装

选定安装目录 (没有则新建一个目录)，将 WIN2k 安装文件包安放在该目录下。根据 WIEN2k_Users-Guide[?] 的第 11 章 Installation and Dimensioning 说明，WIEN2k 的安装如下：

tar -xvf WIEN2k-version.tar

gunzip *.gz

./expand_lapw

建议：新增~/.bashrc 的环境变量设置：

MKLPATH = 你的 INTEL_MKL 库目录

接下来的编译安装主要通过脚本 siteconfig_lapw 完成，其中涉及的最重要的编译参数设置如下：

FOPT = -FR -mp1 -w -prec_div -pc80 -pad -ip -DINTEL_VML -traceback -assume buffered_io -DFFTW3 -I你的 FFTW 安装目录/include

LDFLAGS = $(FOPT) -L$MKLPATH -pthread

R_LIBS = -lfftw3 -lmkl_lapack95_lp64 -lmkl_intel_lp64 -lmkl_intel_thread -lmkl_core -openmp -L你的 FFTW 安装目录/lib -lpthread

RP_LIBS = -lfftw3_mpi $MKLPATH/libmkl_scalapack_lp64.a -Wl,–start-group $MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_blacs_openmpi_lp64.a $MKLPATH/libmkl_core.a -Wl,–end-group $(R_LIBS)

特别地，⋆ 对 LAPW0 的 mpi 并行：

RP_LIBS =-lfftw3_mpi $MKLPATH/libmkl_scalapack_lp64.a -Wl,–start-group $MKLPATH/libmkl_sequential.a $MKLPATH/libmkl_core.a $MKLPATH/libmkl_blacs_openmpi_lp64.a -Wl,–end-group $(R_LIBS)

注意：如果是用 mpich，将 RP_LIBS 中的选项 $MKLPATH/libmkl_blacs_openmpi_lp64.a 替换为 $MKLPATH/libmkl_blacs_intelmpi_lp64.a

或者，将 RP_LIBS 中的 RP_LIBS 选项设置为

RP_LIBS = -lfftw3_mpi -lmkl_scalapack_lp64 -lmkl_sequential -lmkl_blacs_intelmpi_lp64 $MKLPATH/libmkl_blas95_lp64.a -L你的 MPICH 安装目录/lib -lmpich -lmpichf90 $(R_LIBS)

5 安装后环境设置

安装后的环境配置主要通过脚本 userconfig_lapw 完成，我自己选择的文本编辑器是 vim(默认的是 emacs)，其余的都选用默认值。

完成环境配置后，如有必要，还可以通过编辑~/.bashrc 修改有关参数：

注意：如选用 mpi 并行版本，建议对~/.bashrc 作下列修改

- 注释以下这行：
  #ulimit -s unlimited
- 修改 parallel_options
  setenv WIEN_MPIRUN “mpirun -machinefile _HOSTS_ -np _NP_ _EXEC_”

6 配置 web 界面

用 root 用户打开 apache 服务

service apache2 start
在普通用户下执行
w2web
将打开 WIEN2k 默认的 7890 端口作为 WIEN2k 的 web 界面

7 .machines 文件的编写

.machines 文件指定了并行计算所使用的计算资源，因此需要平衡计算资源的负载平衡。.machines 文件中每一项详细说明参见 WIEN2k_Users-Guide[?] 的第 5 章之 5.5 Running programs in parallel mode。根据.machines 文件不同决定进行 k-point 或 mpi 并行计算：

k-point 并行的.machines：
granularity:1
1:node31:1 # 格式：1: 指定节点名:1 (k-point 并行方式)
1:node31:1
1:node32:1
1:node32:1
lapw0:node31:2 node32:2 # 指定 lapw0 并行方式：lapw0: 指定节点名: 核数 n
extrafine:1

mpi 并行的.machines：
granularity:1
1:node31:2 # 格式：1: 指定节点名: 核数 n (mpi 并行)
1:node32:2
lapw0:node31:2 node32:2 # 指定 lapw0 并行方式：lapw0: 指定节点名: 核数 n
extrafine:1

8 采用作业调度提交作业

在 pbs 作业管理系统中，手动编辑.machines 实现 WIEN2k 的并行费时费力，这里提供一个脚本 (wien2k.pbs) 可以根据分配的计算资源自动生成.machines。

注意：该脚本使用时需要根据计算环境修改计算参数

cat wien2k.pbs

```
#############################################################
# #
# Script for submitting parallel wien2k jobs to Dawning cluster. #
# #
#############################################################
#############################################################
# Lines that begin with # PBS are PBS directives (not comments).
# True comments begin with "# " (i.e., # followed by a space).
#############################################################
#PBS -S /bin/bash
#PBS -N TiO2
#PBS -j oe
#PBS -l nodes=1:ppn=8
#PBS -V
```

```
#############################################################
#############################################################
# -S: shell the job will run under
# -o: name of the queue error filename
# -j: merges stdout and stderr to the same file
# -l: resources required by the job: number of nodes and processors per node
# -l: resources required by the job: maximun job time length
#############################################################
######### parallel mode is mpi/kpoint ############
PARALLEL=mpi // 表示采用 mpi 并行或 k 点并行
echo $PARALLEL
#############################################################
NP=‘cat $PBS_NODEFILE | wc -l‘
NODE_NUM=‘cat $PBS_NODEFILE|uniq |wc -l‘
NP_PER_NODE=‘expr $NP / $NODE_NUM‘
username=‘whoami‘
export WIENROOT=你的 WIEN2k 安装目录
export PATH=$PATH:$WIENROOT:.
WIEN2K_RUNDIR=/scratch/$username.$PBS_JOBID
export SCRATCH=$WIEN2K_RUNDIR
# creat scratch dir
if [ ! -a $WIEN2K_RUNDIR ]; then
echo ”Scratch directory $WIEN2K_RUNDIR created.”
mkdir -p $WIEN2K_RUNDIR
fi
cd $PBS_O_WORKDIR
######### Creating .machines ############
case $PARALLEL in
mpi)
echo ”granularity:1” > .machines
for i in ‘cat $PBS_NODEFILE | uniq ‘
do
echo ”1:”$i”:”$NP_PER_NODE >> .machines
done
printf ”lapw0:” >> .machines
######### lapw0 用 mpi 并行 ############
for i in ‘cat $PBS_NODEFILE | uniq‘
do
printf $i:$NP_PER_NODE” ” >> .machines
done
#####################################
######### lapw0 用 mpi 并行报错的算例用以下 mpi_error_lapw0 ############
# printf ‘cat $PBS_NODEFILE| uniq | head -1‘:1 >> .machines
############## end ##################
printf ”
n” >> .machines
echo ”extrafine:1” >> .machines
```

```
;;
kpoint)
echo ”granularity:1” > .machines
for i in ‘cat $PBS_NODEFILE‘
do
echo ”1:”$i”:” 1 >> .machines
done
printf ”lapw0:” >> .machines
##### lapw0 用 mpi 并行 #############
for i in ‘cat $PBS_NODEFILE | uniq‘
do
printf i :NP_PER_NODE” ” >> .machines
done
#####################################
#### lapw0 用 mpi 并行报错的算例用以下 mpi_error_lapw0 ########
# printf ‘cat $PBS_NODEFILE | uniq | head -1‘:1 >> .machines
############## end ##################
printf ”
n” >> .machines
echo ”extrafine:1” >> .machines
;;
esac
################## end creating ####################
######## Run the parallel executable ”WIEN2K” #########
instgen_lapw
init_lapw -b
clean -s
echo ”################### start time is ‘date‘ ##########################”
run_lapw -p
echo ”################### end time is ‘date‘ ##########################”
rm -rf $WIEN2K_RUNDIR
########################### END ##########################
```

该脚本可以实现算例的初始化，必须在存在 *.struct 的前提下进行。