Running WIEN2K on Ranger

with both coarse and fine parallelism

Hang Liu

Texas Advanced Computing Center

May 8, 2012

Outline

- Introduction
- Setting WIEN2K in User's Account
- Executing WIEN2K in Parallel

WIEN2K

- Software package for electronic structure calculations of solids using density functional theory (DFT)
- Based on the full-potential (linearized) augmented plane-wave (LAPW) + local orbitals (LO) methods
- Written in FORTRAN 90 and requires a UNIX operating system since the programs are linked together via C-shell scripts.
- Licensed software, not installed and supported system wide on TACC systems.

User's Major Requests

- How to setup WIEN2K on Ranger with optimal options ?
- How to execute WIEN2K in parallel ?

Setting WIEN2K in User's Account

- Well described in the Part III of user manual
- Run the siteconfig_lapw script and follow the guided setup process
  - intel compiler and mvapich
  - R_LIBS: blas, lapack, fftw
  - PR_LIBS: blas, lapack, scalapack, fftw, fftw_mpi
- Run the userconfig_lapw script to setup the proper envs
  - Set a path to WIEN2k programs
  - Set the stacksize to unlimited
  - Set aliases
  - Set environment variables ($WIENROOT, $SCRATCH)

Two Levels of Parallelism

- Coarse grained parallelism: distributes k-points over multiple task groups by utilizing c-shell scripts, shared file system and passwordless login. NOT managed by MPI or OpenMP
- Fine grained parallelism: multiple tasks in one group for one k-point calculation are managed by MPI and execute scalapack and fftw_mpi operations
- The TACC ibrun command for launching usual MPI applications does not directly support WIEN2K parallel execution
- The file named as .machines has to be presented in the current working directory.

The Structure of the .machines file

```
#======================================
#This is a valid .machines file
#
granularity:1
1:alpha
1:beta
3:gamma:2 delta
3:delta:1 epsilon:4
residue:delta:2
lapw0:gamma:2 delta:2 epsilon:4
======================================
```

The sample from manual for following system

- 5 computers: alpha, ... epsilon
- epsilon has 4, and delta and gamma 2 cpus
- gamma, delta and epsilon are 3 times faster than alpha and beta

The layout of the .machines file

- A multiple row structure, and three types of the rows
- Rows started with keyword like granularity and residue: for load balancing management
- Rows started with a number like 1 and 3: for host management used in lapw1 and lapw2
- Row started with lapw0: for host management used in lapw0

The columns per row for lapw1 and lapw2: weight:machine name1:number1 machine name2:number2 ...

- weight is an integer to mark the relative speed of computers
- machine name[1/2/...] specifies the computer names
- number[1/2/...] specifies the number of cpus used on the computers

The load balancing keyword

- granularity enhances load balancing on heterogeneous environments
- residue specify the machine that calculates the residual k-points. Alternatively, remaining k-points can be distributed one by one over all machines by setting extrafine:1

On Ranger:

- granularity and weight are set to 1 since cores on Ranger nodes are homogeneous
- residue can be ignored since extrafine:1 is more straightforward
- number[1/2/...] should be set as 1 since each core is identified as an individual machine(host)
- machine name[1/2/...] shall be organized into the required "matrix" form: the rows suggest the multiple concurrency for all k-points calculations; the columns per row suggest the multiple concurrency of scalapack and fftw_mpi operations for each k-point
- The only concern is: the machine names are not known before the SGE batch started.
  - A utility script is needed and used in job script to generate the .machines file in fly
  - This script should group the tasks according to users' inputs: the number of rows/columns in.machines file

Hosts in a SGE batch

- -pe TpNway NoNx16: job will be executed using the specified number of tasks (cores to use) per node ("wayness") and the number of nodes times 16 (total number of cores).
- $PE_HOSTFILE will be created according to NoN, e.g.-pe 4way 32

```
i115-303.ranger.tacc.utexas.edu
i182-102.ranger.tacc.utexas.edu
```

- Accounting the TpN, the host list will be saved into $hostfile_tacc

```
i115-303.ranger.tacc.utexas.edu
i115-303.ranger.tacc.utexas.edu
i115-303.ranger.tacc.utexas.edu
i115-303.ranger.tacc.utexas.edu
i182-102.ranger.tacc.utexas.edu
i182-102.ranger.tacc.utexas.edu
i182-102.ranger.tacc.utexas.edu
i182-102.ranger.tacc.utexas.edu
```

- This is what ibrun command does to prepare hostfile for usual MPI jobs

Constructing .machine from $hostfile_tacc, e.g. for lapw1 and lapw2 parts:

```
==========================================
set proclist=‘cat $hostfile_tacc‘
set nproc=‘cat hostfile_tacc | wc -l‘
set i=1
while ($i <= $nproc )
echo -n ’1:’ >>.machines
@ i1 = $i + $mpisize_per_k
@ i2 = $i1 - 1
echo $proclist[$i-$i2] ’:1’ >>.machines
set i=$i1
end
echo ’granularity:1’ >>.machines
echo ’extrafine:1’ >>.machines
==========================================
```

Or more efficiently, just one awk statement to replace the while loop

- Rows for lapw1 and lapw2:

```
awk -v div=mpisize_per_k '{_=int(NR/(div+1.0e-10))} {a[_]=((a[_])?a[_]FS:x)$1;
    l=(_>l)?_:l}END{for(i=0;i<=l;++i)print "1:"a[i]":1"}'
    $hostfile_tacc >>.machines
```

- Row lapw0

```
awk -v div=mpisize_lapw0 '{_=int(NR/(div+1.0e-10))} {a[_]=((a[_])?a[_]FS:x)$1;
    l=(_>l)?_:l}END{for(i=0;i<=0;++i)print "lapw0:"a[i]":1"}'
    $hostfile_tacc >>.machines
```

Packing up: wien2k_tasks utility script:

- Usage: wien2k_tasks mpisize_lapw0 mpisize_per_k
- mpisize_lapw0: number of machines used by lapw0_mpi, should be smaller than the total number of the hosts of the job
- mpisize_per_k: number of machines used by lapw1_mpi and lapw2_mpi for each k-point, should be a divisor of the total number of the hosts of the job

Sample usage in usual SGE script:

```
================================
!/bin/tcsh
#$ -V
#$ -cwd
#$ -N wien2k
#$ -e err.$JOB_ID
#$ -o out.$JOB_ID
#$ -pe 16way 64
#$ -q development
#$ -l h_rt=00:05:00
./wien2k_tasks 8 8
runsp_lapw -p -i 1
================================
```

E.g. when -pe 16way 64, the ./wien2k_tasks 8 8 will generate .machines file as

```
#=============================================================================
#
granularity:1
lapw0:i115-203 i115-203 i115-203 i115-203 i115-203 i115-203 i115-203 i115-203:1
1:i115-203 i115-203 i115-203 i115-203 i115-203 i115-203 i115-203 i115-203:1
1:i115-203 i115-203 i115-203 i115-203 i115-203 i115-203 i115-203 i115-203:1
1:i115-301 i115-301 i115-301 i115-301 i115-301 i115-301 i115-301 i115-301:1
1:i115-301 i115-301 i115-301 i115-301 i115-301 i115-301 i115-301 i115-301:1
1:i182-103 i182-103 i182-103 i182-103 i182-103 i182-103 i182-103 i182-103:1
1:i182-103 i182-103 i182-103 i182-103 i182-103 i182-103 i182-103 i182-103:1
1:i182-203 i182-203 i182-203 i182-203 i182-203 i182-203 i182-203 i182-203:1
1:i182-203 i182-203 i182-203 i182-203 i182-203 i182-203 i182-203 i182-203:1
extrafine:1
#=============================================================================
```

- Since mpisize_lapw0=8, there are 8 cores used for lapw0_mpi
- All 64 cores are used for lapw1_mpi and lapw2_mpi
- Since mpisize_per_k=8, there are $\frac{64}{8} = 8$ groups(lines) of cores, each k-point calculation will be carried out by one group of cores with parallelism boosted by scalapack and fftw_mpi
- Assuming there are 120 k-points, and there are 8 groups, so each one group will compute $\frac{120}{8} = 15$ k-points
- Results will be summed up when all k-point calculations are finished

Summary

- The wien2k_tasks mpisize_lapw0 mpisize_per_k works conveniently for users to manage the task geometry of the coarse and fine grained parallelism in WIEN2K
- The optimal values for mpisize_lapw0 mpisize_per_k need to be figured out by users according to the special calculations they do. The k-point parallelism should be considered at first because it is trivially in parallel and the most efficient
- One more thing still under investigation: process affinity when multiple MPI binaries executed on ONE node.