“天工”超算集群的系统操作及LINUX平台下软件的编译和安装

松山湖材料实验室·材料计算与数据库平台

何志海 2020/2/26

目录

目录

“天工”超算集群简介

“天工”超算集群于2019年5月底完成一期建设。目前，共有计算节点179个，CPU核心数6776个，总算力达到500 Tflops，可用存储空间超过1 PB，节点间采用100 Gbps EDR InfiniBand高速互联。

管理节点（2个）：

用于系统管理，普通用户无权登录。

登录节点（4个）：

用于用户登录、编译及通过作业调动系统提交作业等。

Intel Xeon CPU计算节点（168个）：

用于多数作业。单节点配置2颗Intel Xeon Gold 6140 CPU，共36核，192 GB内存。

Intel Xeon CPU 1 TB计算节点（2个）：

用于大内存作业。单节点配置4颗Intel Xeon Gold 6140 CPU，共72核，1024 GB内存。

Hygon CPU计算节点（8个）：

单节点配置2颗Hygon 7185 CPU，共64核，256 GB内存。

GPU计算节点（1个）：

适合GPU应用。配置4颗Intel Xeon Gold 6140 CPU，共72核，384 GB内存，8张NVIDIA Tesla V100 GPU卡，单卡32 GB显存。

系统登录

本超算的操作系统为x86_64架构的CentOS 7.4 Linux，用户需以SSH方式登录到登录节点后进行编译、提交作业等操作。具体登录方式见申请账号的邮件。

常见SSH登录软件：

Windows系统：Putty、Xshell、MobaXterm、WinSCP(文件传输)…
Linux/Mac系统：自带终端。
个人登录方式：Win10+内置Ubuntu子系统（具备常规Linux系统的基本功能，可正常安装运行各种软件，如Intel编译器、VASP）。

系统注意事项：

1. 请使用复杂密码。系统安全，人人有责。
2. 禁止在登录节点上直接运行程序，请通过作业调度系统提交作业。
3. 请保持个人磁盘空间小于500 GB（du -sh . 查看当前目录占用空间），定期数据备份及清理无用数据。

环境变量加载

本超算系统安装了多种编译环境及应用，为方便用户使用，配置有Environment Modules工具对其进行封装，用户可以使用module命令设置所需要的环境变量。

常用命令：

- module avail：显示可使用的模块。
- module list：显示已加载的模块。
- module load：加载模块。
- module unload：卸载已加载的模块。
- module purge：卸载所有已加载的模块。
- module show：显示环境变量信息。

MODULE举例

```
(base) [zhhe@login1 ~]# module avail

------------------------------------------------------------ /public/software/modules ------------------------------------------------------------
apps/TDAP/2.1.0/mvapich                 apps/vasp/5.4.4/intel_2017.4.239        mathlib/netcdf/intel/4.1.3
apps/abinit/8.10.2/intelmpi             apps/vasp/5.4.4/intelmpi                mpi/intelmpi/2017.4.239
apps/lammps/12Dec18/intelmpi-gpu        apps/vasp/5.4.4/openmpi                 mpi/intelmpi/2019u3
apps/lammps/12Dec18/intelmpi-normal     apps/vasp/5.4.4_vtst/intel_2017.4.239   mpi/mpich/gcc/3.3
apps/lammps/12Dec18/intelmpi-opt        compiler/cuda/10.0                      mpi/mpich/intel/3.3
apps/openmx/3.8.5/intel_2019u3          compiler/intel/intel-compiler-2017.5.239 mpi/mvapich2/gnu/2.3.1
apps/qe/6.2/intelmpi                    compiler/intel/intel-compiler-2019u3    mpi/mvapich2/gnu/2.3b
apps/qe/6.4.1/intelmpi                  mathlib/fftw/intelmpi/3.3.7_double      mpi/openmpi/gcc/3.1.4
apps/siesta/gnu/4.1-b4                  mathlib/fftw/intelmpi/3.3.7_float       mpi/openmpi/intel/3.1.4
apps/vasp/5.4.1/intelmpi                mathlib/hdf5/intel/1.8.12
apps/vasp/5.4.1/openmpi                 mathlib/lapack/intel/3.4.2
(base) [zhhe@login1 ~]#
```

```
(base) [zhhe@login1 ~]# which mpiifort
/usr/bin/which: no mpiifort in (/public/home/zhhe/soft/vaspkit_1
me/zhhe/perl5/bin:/opt/gridview/slurm/bin:/opt/gridview/slurm/sb
pdate3/compilers_and_libraries_2019/linux/bin/intel64:/opt/clusc
ic/home/zhhe/bin)
(base) [zhhe@login1 ~]# module load mpi/intelmpi/2017.4.239
(base) [zhhe@login1 ~]# which mpiifort
/public/software/mpi/intelmpi/2017.4.239/intel64/bin/mpiifort
(base) [zhhe@login1 ~]#
```

```
(base) [zhhe@login1 ~]# module list
Currently Loaded Modulefiles:
  1) mpi/intelmpi/2017.4.239
(base) [zhhe@login1 ~]# module purge
(base) [zhhe@login1 ~]# module list
No Modulefiles Currently Loaded.
(base) [zhhe@login1 ~]#
```

作业提交系统

Slurm(Simple Linux Utility for Resource Management)是开源的、具有容错性和高度可扩展性的Linux集群资源管理和作业调度系统。利用Slurm进行资源和作业管理，可避免相互干扰，提高运行效率。所有需运行的作业无论是程序调试还是业务计算均必须通过Slurm的命令提交。请不要在登录节点直接运行作业（编译除外），以免影响其他用户的正常使用。

常用命令：

- ***sbatch***：提交作业脚本使其运行。
- ***scancel***：取消排队或运行中的作业。
- ***scontrol***：显示或设定Slurm作业、队列、节点等状态。
- ***sinfo***：显示队列或节点状态。
- ***squeue***：显示队列中的作业状态。
- ***srun***：运行并行作业。

三种作业提交模式：

- **批处理作业（*sbatch*）**：作业被调度运行后，在所分配的首个节点上执行作业脚本，在作业脚本中也可以使用*srun*命令。
- **交互式作业（*srun*）**：在登录shell中执行时，首先向系统提交作业请求并等待资源分配，然后在所分配的节点上加载作业任务。
- **分配模式作业（*salloc*）**：相当于交互式模式和批处理模式的融合。

VASP作业提交举例： sbatch vasp.sh

```
(base) [zhhe@login2 tmp]$ ls
INCAR  KPOINTS  POSCAR  POTCAR  vasp.sh
(base) [zhhe@login2 tmp]$ sbatch vasp.sh
Submitted batch job 992457
(base) [zhhe@login2 tmp]$
```

作业号

```
#!/bin/bash
#SBATCH -p debug                #目标队列
#SBATCH -N 1                    #申请节点数
#SBATCH --ntasks-per-node=36    #每个节点的进程数

echo Begin Time is `date`
echo This job runs on the following nodes:
echo $SLURM_JOB_NODELIST
echo "------------------------------------------------------------"

module load apps/vasp/5.4.4/intel_2017.4.239    #加载vasp需要的环境变量
srun --mpi=pmi2 vasp_std                        #使用 srun 命令运行并行作业

echo End Time is `date`
```

SLURM命令举例

显示队列、节点信息： sinfo

```
(base) [zhhe@login2 tmp]$ sinfo
PARTITION AVAIL  TIMELIMIT  NODES  STATE NODELIST
gpu          up   infinite      1   idle gpu1
debug        up   infinite      1  down* comput135
debug        up   infinite    125    mix comput[1-11,13-17,19-23,25-28,30-33,36,38-39,41-42,4
155-158,161,164,166-168]
debug        up   infinite     42  alloc comput[12,18,24,29,34-35,37,40,43-44,46-47,49,51,54,
regular*     up   infinite      1  down* comput135
regular*     up   infinite    125    mix comput[1-11,13-17,19-23,25-28,30-33,36,38-39,41-42,4
155-158,161,164,166-168]
regular*     up   infinite     42  alloc comput[12,18,24,29,34-35,37,40,43-44,46-47,49,51,54,
premium      up   infinite      1  down* comput135
```

```
(base) [zhhe@login2 tmp]$ sinfo -o '%10P %5a %4I %12L %10A %5c %8t'
PARTITION  AVAIL PRIO DEFAULTTIME  NODES(A/I) CPUS  STATE
gpu        up    2    12:00:00    0/1        36    idle
debug      up    1    30:00       0/0        36    down*
debug      up    1    30:00       125/0      36    mix
debug      up    1    30:00       42/0       36    alloc
regular*   up    3    3-00:00:00  0/0        36    down*
regular*   up    3    3-00:00:00  125/0      36    mix
regular*   up    3    3-00:00:00  42/0       36    alloc
premium    up    1    3-00:00:00  0/0        36    down*
premium    up    1    3-00:00:00  125/0      36    mix
premium    up    1    3-00:00:00  42/0       36    alloc
hygon      up    2    3-00:00:00  8/0        64    alloc
bigmem     up    2    2-00:00:00  2/0        72    alloc
highthro   up    3    1-00:00:00  0/0        36    down*
highthro   up    3    1-00:00:00  125/0      36    mix
highthro   up    3    1-00:00:00  42/0       36    alloc
long       up    1    10-00:00:00 0/0        36    down*
long       up    1    10-00:00:00 125/0      36    mix
long       up    1    10-00:00:00 42/0       36    alloc
(base) [zhhe@login2 tmp]$
```

man sinfo

SLURM命令举例

显示队列中的作业信息： squeue

```
(base) [zhhe@login2 tmp]$ sbatch vasp.sh
Submitted batch job 992494
(base) [zhhe@login2 tmp]$ squeue -u zhhe
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
            992494     debug  vasp.sh     zhhe  R       0:08      1 comput62
```

```
(base) [zhhe@login2 tmp]$ squeue -u zhhe -o '%8i %9P %5D %4C %3t %10M %100Z'
JOBID    PARTITION NODES CPUS ST  TIME       WORK_DIR
992494   debug     1     36   R   0:16       /public/home/zhhe/tmp
(base) [zhhe@login2 tmp]$
```

取消排队或运行中的作业： scancel

```
(base) [zhhe@login2 tmp]$ scancel 992494
(base) [zhhe@login2 tmp]$ squeue -u zhhe
             JOBID PARTITION     NAME     USER ST       TIME  NODES NODELIST(REASON)
(base) [zhhe@login2 tmp]$
```

查看详细作业信息： scontrol show job

```
(base) [zhhe@login2 tmp]$ scontrol show job 992509
JobId=992509 JobName=vasp.sh
   UserId=zhhe(1102) GroupId=users(100) MCS_label=N/A
   Priority=333 Nice=0 Account=sslakelab QOS=normal
   JobState=RUNNING Reason=None Dependency=(null)
   Requeue=1 Restarts=0 BatchFlag=1 Reboot=0 ExitCode=0:0
   RunTime=00:00:37 TimeLimit=00:30:00 TimeMin=N/A
   SubmitTime=2020-02-24T09:49:27 EligibleTime=2020-02-24T09:49:27
   AccrueTime=2020-02-24T09:49:27
   StartTime=2020-02-24T09:50:37 EndTime=2020-02-24T10:20:38 Deadline=N/A
   PreemptTime=None SuspendTime=None SecsPreSuspend=0
   LastSchedEval=2020-02-24T09:50:37
```

目录

GNU C/C++ FORTRAN 编译器

系统自带的默认编译器，用户一般无需特殊设置即可使用。

- 编译Fortran 77和9x、200x源程序的命令：分别为g77和gfortran：
  - gfortran：属于GCC 4系列，可以编译Fortran 77及9x、200x源程序
  - g77：属于GCC 3.4系列，不可编译Fortran 9x、200x源程序
- 编译C、C++源程序的命令：分别为gcc和g++

GNU FORTRAN编译器用法

基本格式：

4.x.y版本编译器：gfortran [options] file1 [file2 ...]

3.x.y版本编译器：g77 [options] file1 [file2 ...]

注意：

[ ]表示是其内部的选项可选

文件名和选项区分大小写

输出文件后缀与类型的关系

编译器默认将输出按照文件类型与后缀相对应

文件名	解释	生成方式
filename.o	目标文件	编译时添加-c选项生成
filename.so	共享库文件	编译时指定为共享型，如添加-shared，并不含-c
filename.mod	模块文件	编译含有MODULE声明时的源文件生成
filename.s	汇编文件	编译时添加-S选项生成
a.out	默认生成的可执行文件	编译时没有指定-c时生成

重要编译选项

编译选项对运行速度、编译的兼容性等有影响。

一般选项

- -c：仅编译成目标文件（.o文件），并不进行链接。
- -o file：指定生成的文件名。
- -v：详细模式，在每个命令执行前显示其命令行。
- -fsyntax-only：仅仅检查代码的语法错误，并不进行其余操作。

调试选项

- -g：包含调试信息。
- -ggdb：包含利用gbd调试时所需要的信息。

重要编译选项

优化选项

●-O[level]：设置优化级别。优化级别level可以设置为0、1、2、3、s，默认为-O0。

链接选项

●-static：静态链接所需的库。

●-shared：生成共享对象文件而不是可执行文件，必须在编译每个目标文件时使用-fpic选项。

●-I<头文件目录>：指明头文件的搜索路径。

●-l<库文件>：指明需链接的库名，如库为libxyz.a，则可用-lxyz指定。

●-L<库目录>：指明库的搜索路径。

程序编译举例

- 将Fortran 90程序yourprog.f90编译为目标文件yourprog.o：

  gfortran -c yourprog.f90

- 将Fortran 90程序yourprog.f90编译为可执行文件yourprog：

  gfortran -o yourprog yourprog.f90

- 将使用lapack库的Fortran 90程序yourprog.f90编译为可执行文件yourprog：

  gfortran -o yourprog -L/opt/lib -llapack yourprog.f90

- 将Fortran 90程序yourprog.f90静态编译为O3优化的可执行文件yourprog：

  gfortran -O3 -static -o yourprog yourprog.f90

INTEL C/C++ FORTRAN编译器

主要针对用Intel平台的高性能编译器，可用于开发复杂且要进行大量计算的C/C++、Fortran程序。

编译命令：

C: icc　　C++: icpc　　Fortran: ifort

基本格式：

ifort [options] file1 [file2 ...]

注意：[ ]表示是其内部的选项可选，文件名和选项区分大小写。

Intel编译器用法与GNU编译器基本相同，Intel对编译器做了大量的优化，一般来说ifort要比gfortran、g77快许多。

重要编译选项

- -O*n*：设定优化级别。优化级别可以设置为0、1、2、3，默认O2，推荐使用。O3在O2基础上增加更激进的优化，但有些情况下速度可能更慢。
- -x*code*：采用何种指令集进行编译优化，如-xAVX。
- -xHost：生成编译主机处理器能支持的最高指令集。
- -fast：最大化优化整个程序的速度，相当于设置 -ipo、-O3、-no-prec-div、-static、-fp-model fast=2和-xHost。
- -ip：在单个文件中进行过程间优化。
- -ipo、-no-ipo：是否在多文件中进行过程间优化。
- -qopenmp：编译OpenMP程序。

MPI编译环境简介

- 各种MPI编译环境实际上为MPI标准的不同实现。
- 利用在普通编译器（比如Intel编译器）基础上添加必要的MPI参数以指定MPI库的路径等链接MPI库进行编译。
- 除具体MPI实现的参数之外，其调用的普通编译器的参数继续有效。
- 优化等不仅需参考此MPI编译环境也要参考调用的普通编译器。

主流MPI环境

- Intel MPI、Open MPI和MPICH 3：既支持InfiniBand，也支持以太网。
- MPICH和MPICH2：支持以太网，不支持InfiniBand网络。
- MVAPICH、MVAPICH2：基于MPICH和MPICH2，支持InfiniBand网络。
- 编译命令基本一致。
- 一些编译参数有些不同。

INTEL MPI库简介

一种多模消息传递接口(MPI)库，可以使开发者采用新技术改变或升级其处理器和互联网络而无需改编软件或操作环境成为可能。

主要包含以下内容：

- 运行时环境(RTO)：具有运行程序所需要的工具，包含多功能守护进程(MPD)、Hydra及支持的工具、共享库(.so)和文档。
- 开发套件(SDK)：含所有运行时环境组件和编译工具，含编译器命令，如mpiifort、mpiicc、头文件和模块、静态库(.a)、调试库、追踪库和测试代码。

主页：http://software.intel.com/en-us/intel-mpi-library/

Intel MPI与Open MPI、MPICH等MPI实现不同：

- mpiicc、mpiicpc和mpiifort命令：使用Intel编译器
- mpicc、mpif90和mpifc命令：默认使用GNU编译器

编译命令	调用的默认编译器命令	支持的语言
通用编译器		
mpicc	gcc, cc	C
mpicxx	g++	C/C++
mpifc	gfortran	Fortran77*/Fortran 95*
GNU Compilers Versions 3 and Higher		
mpigcc	gcc	C
mpigxx	g++	C/C++
mpif77	g77	Fortran 77
mpif90	gfortran	Fortran 95
Intel Compilers Versions 11.1 and Higher		
mpiicc	icc	C
mpiicpc	icpc	C++
mpiifort	ifort	Fortran77/Fortran 95

- -{cc,cxx,fc,f77,f90}=<compiler>：选择使用的编译器。如：

mpicc -cc=icc -c test.c　　mpiifort -fc=ifort -c test.f90

MPI编译举例

- 将C语言MPI并行程序prog_mpi.c编译为可执行文件prog_mpi：

  mpiicc -o prog_mpi prog_mpi.c

- 将C++语言MPI程序prog_mpi.cpp编译为可执行文件prog_mpi ：

  mpiicxx -o prog_mpi prog_mpi.cpp

- 将Fortran90语言MPI程序prog_mpi.f90编译为O3优化的可执行文件prog_mpi：

  mpiifort -O3 -o prog_mpi prog_mpi.f90

- MPI程序运行：

  mpirun -np 4 ./prog_mpi

INTEL MKL

Intel核心数学库(Math Kernel Library, MKL)，用户可以直接调用，以提高性能、加快开发。

MKL主目录，如：/opt/intel/compilers_and_libraries/linux/mkl

主要内容：

- 基本线性代数子系统库(BLAS)和线性代数库(LAPACK)：提供向量、向量-矩阵、矩阵-矩阵操作。
- ScaLPACK分布式线性代数库：含基础线性代数通信子程序(BLACS)和并行基础线性代数子程序(PBLAS)。
- 快速傅里叶变换方程(FFT)：支持1、2、3维，并有分布式版本。
- 向量数学库(VML)：提供针对向量优化的数学操作。
- 向量统计库(VSL)：提供高性能的向量化随机数生成算子，可用于一些几率分布、剪辑和相关例程的汇总统计功能。
- 数据拟合库：提供基于样条函数逼近、函数的导数和积分，及搜索。

举例

```
#!/bin/bash
mpiifort -xHost -O2 -o TB.x Main.F90 cBand.F90 cRep.F90 MD.F90 OPT.F90 \
-L/opt/intel/compilers_and_libraries/linux/mkl/lib/intel64/  \
-lmkl_blas95_lp64 \
-lmkl_lapack95_lp64 \
-lmkl_scalapack_lp64 -Wl,--start-group  \
-lmkl_cdft_core \
-lmkl_intel_lp64 \
-lmkl_sequential \
-lmkl_core \
-lmkl_blacs_intelmpi_lp64 -Wl,--end-group -lpthread -lm
echo "Done!"
```

-Wl,--start-group -Wl,--end-group之间的参数，是链接参数，编译时只传递给链接器，而语法分析等时不处理。

GNU MAKE

- make工具用以自动化编译过程，当编译一个包括成百上千个文件的大型项目时，逐个文件编译繁琐且低效，并且当更新某个文件之后，还需要重新再编译一次。make工具正是为解决此类问题而诞生的。make在20世纪70年代被发明，现在仍被作为大多数的编程项目的核心编译辅助工具在使用。它甚至可以构建Linux内核。
- make的思路很简单：当你改变了源文件并想重新构建程序或者其他的输出文件时，make检查时间戳看哪些改变了，并按要求重新构建这些文件，而不需要浪费时间重新构建其他文件。但是在这个基本的原则之上，make提供了丰富的选项集合，让你能够操作多个目录，为不同平台构建不同的版本，能够以其他的方式定制你的构建过程。

GNU make 仿照make的标准功能(透过clean-room工程)重新改写,并加入作者觉得值得加入的新功能,常和GNU编译系统一起被使用,是大多数GNU Linux安装的一部分。

A SIMPLE MAKEFILE

Makefile：

```
hello: hello.f90
	ifort hello.f90 -o hello
```

To build the program：

```
$ make
```

```
[zhhe@cluster-E hello]$ ls
Makefile  hello.F90
[zhhe@cluster-E hello]$ make
ifort hello.F90 -o hello
[zhhe@cluster-E hello]$ ls
Makefile  hello  hello.F90
[zhhe@cluster-E hello]$
```

EXAMPLE A

```
.SUFFIXES: .inc .f .f90 .F .F90
.PHONY:clean
SOURCE=Main.F90 Ham.F90 Band.F90 Berry.F90 Dipole.F90
EXEC=a.out

FC =mpiifort
FC_FLAGS =-xHost -O2 -static

LIBS_ELPA =/opt/elpa_2018/lib -lelpa_onenode
LIBS_MKL =/opt/intel/compilers_and_libraries/linux/mkl/lib/intel64/  \
-lmkl_blas95_lp64 \
-lmkl_lapack95_lp64 \
-lmkl_scalapack_lp64 -Wl,--start-group  \
-lmkl_cdft_core \
-lmkl_intel_lp64 \
-lmkl_sequential \
-lmkl_core \
-lmkl_blacs_intelmpi_lp64 -Wl,--end-group -lpthread -lm

LIBS =-L$(LIBS_ELPA) -L$(LIBS_MKL)

INCLUDE_ELPA =/opt/elpa_2018/include/elpa_onenode-2018.05.001/modules
INCLUDES =$(INCLUDE_ELPA)
```

定义变量

```
$(EXEC):$(addsuffix .o,$(basename $(SOURCE)))
        $(FC) -o $@ $^ -I $(INCLUDES) $(LIBS)
```

目标及生成规则

```
%.o:%.F90
        $(FC) -c $< -I $(INCLUDES) $(LIBS)
```

设置各种规则

```
clean:
        -rm -f *.g *.o *.L *.mod
```

特殊工作指令

EXAMPLE B —— VASP

vasp_5.4.1/5.4.4 的主makefile

```
VERSIONS = std gam ncl gpu gpu_ncl

.PHONY: all veryclean versions $(VERSIONS)

all: std gam ncl

versions: $(VERSIONS)

$(VERSIONS):
        mkdir build/$@ ; \
        cp src/makefile src/.objects makefile.include build/$@ ; \
        $(MAKE) -C build/$@ VERSION=$@ all
#       $(MAKE) -C build/$@ VERSION=$@ dependencies -j1 ; \
#       $(MAKE) -C build/$@ VERSION=$@ all


veryclean:
        rm -rf build/*
```

只编译std版本：make std　　编译所有版本：make all

EXAMPLE B —— VASP

src/makefile　设置源代码的编译规则，系统环境无关。一般无需修改。

```
.SUFFIXES:
.SUFFIXES: .mod .o .f90
.PRECIOUS: %.f90
# .SECONDEXPANSION:

FUFFIX=.F
SUFFIX=.f90

OFLAG_1=-O1
OFLAG_2=-O2
OFLAG_3=-O3
OFLAG_4=

OFLAG=$(OFLAG_2)
OFLAG_IN=$(OFLAG)

LIB=lib
LLIB=-Llib -ldmy

SRCDIR=../../src
BINDIR=../../bin

include .objects
include makefile.include

# FPP=$(CPP) $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)

FPP=$(filter-out -DwNGZhalf -DNGZhalf -DwNGXhalf -DNGXhalf,$(CPP))

DMPI=$(findstring DMPI,$(subst DMPI_,,$(CPP)))

LIB+=$(filter-out lib,$(LIBS))
LLIB+=$(filter-out -Llib -ldmy,$(LLIBS))

OBJS=$(SOURCE) $(OBJECTS)
```

```
%.o: %$(SUFFIX)
	$(FC) $(FREE) $(FFLAGS) $(OFLAG) $(INCS) -c $*$(SUFFIX)

# $(OBJS_01): $(F90SRC_01)
$(OBJS_01): %.o: %$(SUFFIX)
	$(FC) $(FREE) $(FFLAGS_1) $(OFLAG_1) $(INCS_1) -c $*$(SUFFIX)

# $(OBJS_02): $(F90SRC_02)
$(OBJS_02): %.o: %$(SUFFIX)
	$(FC) $(FREE) $(FFLAGS_2) $(OFLAG_2) $(INCS_2) -c $*$(SUFFIX)

# $(OBJS_03): $(F90SRC_03)
$(OBJS_03): %.o: %$(SUFFIX)
	$(FC) $(FREE) $(FFLAGS_3) $(OFLAG_3) $(INCS_3) -c $*$(SUFFIX)

# $(OBJS_IN): $(F90SRC_IN)
$(OBJS_IN): %.o: %$(SUFFIX)
	$(FC) $(FREE) $(FFLAGS_IN) $(OFLAG_IN) $(INCS_IN) -c $*$(SUFFIX)

cleanall: clean
	rm -rf $(LIB) *.F test.x

clean:
	rm -f *.o *.mod *.f90

#-----------------------------------------------------------------------
# special rules
#-----------------------------------------------------------------------

main.o : main$(SUFFIX)
	$(FC) $(FREE) $(FFLAGS) $(DEBUG) $(INCS) -c $*$(SUFFIX)

fft3dlib_f77.o: fft3dlib_f77$(SUFFIX)
	$(F77) $(FFLAGS_F77) -c $*$(SUFFIX)
```

EXAMPLE B —— VASP

makefile.include 编译器相关设置，系统环境相关。拷贝自arch/makefile.include.linux_intel。需要注意的地方如红色线框所示。

```
CPP_OPTIONS= -DHOST=\"Tiangong\" \
             -DMPI -DMPI_BLOCK=65536 \
             -Duse_collective \
             -DscaLAPACK \
             -DCACHE_SIZE=12000 \
             -Davoidalloc \
             -Duse_bse_te \
             -Dtbdyn \
             -Duse_shmem

CPP        = fpp -f_com=no -free -w0  $*$(FUFFIX) $*$(SUFFIX) $(CPP_OPTIONS)

FC         = mpiifort
FCL        = mpiifort -mkl=sequential -lstdc++

FREE       = -free -names lowercase

FFLAGS     = -assume byterecl -heap-arrays 64
OFLAG      = -xHost -O3 -static
OFLAG_IN   = $(OFLAG)
DEBUG      = -O0

MKL_PATH   = $(MKLROOT)/lib/intel64
BLAS       =
LAPACK     =
BLACS      = -lmkl_blacs_intelmpi_lp64
SCALAPACK  = $(MKL_PATH)/libmkl_scalapack_lp64.a $(BLACS)

OBJECTS    = fftmpiw.o fftmpi_map.o fft3dlib.o fftw3d.o \
 $(MKLROOT)/interfaces/fftw3xf/libfftw3xf_intel.a
```

需在对应目录下预先编译libfftw3xf_intel.a文件(make libintel64)

INTEL PARALLEL STUDIO XE 安装

Intel编译器、MPI、MKL需下载Intel Parallel Studio XE软件包。凭学校邮箱可申请教育版序列号，免费使用一年（一个序列号最多可在5台不同机器上使用，到期后可以继续申请序列号）。

下载地址：https://software.intel.com/en-us/parallel-studio-xe/choose-download

注意：不推荐使用2019版的parallel studio，此版本的MPI对vasp支持不好，可能会出现一些莫名的问题，如内存耗散。

安装方法：

解压后执行install.sh，按照提示安装（采用序列号在线激活需联网）。默认安装目录：/opt/intel。安装好之后在~/.bashrc中加入以下语句：（以2018update1版本为例）

```
source /opt/intel/compilers_and_libraries_2018.1.163/linux/bin/compilervars.sh intel64
source /opt/intel/compilers_and_libraries_2018.1.163/linux/mkl/bin/mklvars.sh intel64
source /opt/intel/impi/2018.1.163/bin64/mpivars.sh intel64
```

刷新环境变量或重新登录终端即可使用。

Thank you !