杭州电子科技大学超算平台常用作业脚本示例

Slurm作业脚本说明

超算平台部署了slurm作业调度系统，用于多用户批量任务的统一管理。

作业脚本格式介绍如下（红色内容是需要根据需要进行修改的）：

```
#!/bin/bash
#SBATCH -J jobname    (定义本次作业名)
#SBATCH -p normal     (指定本次作业运行的队列名，队列是一组由管理员设定计算节点资源集合)
#SBATCH -N 2          (指定本次作业使用几个计算节点，不能大于队列中计算节点总数)
#SBATCH --ntasks-per-node=64  (指定队列中各个计算节点使用的CPU核心数，不能大于单计算节点总核数)
#SBATCH --gres=gpu:2      (指定队列中各个计算节点使用的GPU卡数量，不能大于单计算节点GPU卡总数，不使用GPU删除此行)
#SBATCH --time=24:00:00   (指定本次作业运行的最大时长)
（节点数*单节点核数*最大运行时长）+（节点数*GPU卡数*费率*最大运行时间）不能大于可用机时余额。
cd $SLURM_SUBMIT_DIR     （不用修改，意思是该任务在脚本所在目录运行，一般将作业脚本与算例放在同一目录）
##################################################################################
source /public/software/profile.d/***.**          (加载计算软件运行所需要的所有环境变量)

srun hostname -s | sort -n >slurm.hosts            (根据资源申请，生成计算资源列表文件)
 OMP_NUM_THREADS= $SLURM_NPROCS          (定义openmp程序可以使用的线程数，纯mpi并行程序则删除该行)
mpirun -np $SLURM_NPROCS -machinefile slurm.hosts program_name   (计算程序运行的行)
（ $SLURM_NPROCS是本次运行总核心， slurm.hosts是生成的计算资源列表， program_name是你的计算程序名，通过变量和文件的引入，
确保程序运行时使用的计算资源与slurm分配的资源一致 ）
```

#号以上的部分为申请资源的部分，

#以下的部分为程序运行所必需要的步骤。

Slurm作业脚本使用说明

使用超算平台，一般先调试好计算程序后，将程序运行的环境变量、命令、参数和步骤插入slurm作业脚本中，替换“计算程序运行的行”、然后运行“sbatch 脚本文件名”，按该格式提交slurm作业脚本即可，注意提交后会有一个作业ID生成，这个ID对作业运行过程中的监控管理有用。

为了确保计算程序使用的计算资源与slurm分配的计算资源一致，所以在计算程序中，如果有涉及执行的计算节点名、核心数、GPU卡数、GPU卡id时，需要将这些内容使用slurm作业调度系统变量进行替代。

如：使用node1,node2每节点使用64核心，共个节点共计128个核心执行一个mpi程序，手动运行调试时的步骤是：

1、加载软件运行的编译、mpi、计算程序环境变量：

```
source   /public/software/profile.d/***.**
```

2、手动编写运行的machinefile文件slurm.hosts（该文件内容为node1和node2各写64行）。

3、手动执行mpirun启动测试。

```
mpirun -np 128  -machinefile slurm.hosts program_name
```

转为脚本运行时，可以将脚本#以下程序运行的部分内容改为，即完成的命令的集成：

```
source   /public/software/profile.d/***.**
srun hostname -s | sort -n >slurm.hosts
mpirun -np $SLURM_NPROCS -machinefile slurm.hosts program_name >run.log
```

>run.log的意思是将运行过程中的输入信息，存储到run.log文件中。

Slurm作业脚本使用说明（续）

作业脚本常用参数（可选）：

- --begin=<time>：设定作业开始运行时间，如--begin=“18:00:00”。
- --cpu-per-task：需要的CPU核数。
- --mem：内存限制数，以MB为单位
- --error=<filename>：设定存储出错信息的文件名。
- --exclude=<names>：设定不采用（即排除）运行的节点。
- --exclusive[=user|mcs]：设定排它性运行，不允许该节点有它人或某user用户或mcs的作业同时运行。
- -N<minnodes[-maxnodes]>：设定所需要的节点数。
- --nodelist=<names>：设定需要的特定节点名，格式类似node[1-10,11,13-28]。
- --output=<filename>：设定存储标准输出信息的文件名。

Slurm作业管理

sbatch:提交作业

sbatch <SLURM 脚本文件名> 提交作业，如sbatch jobfile.sh提交作业脚本名为jobfile.sh的任务。

scancel：取消作业

jobid<job_id_list>：设定作业号。
--name=<name>：设定作业名。
--partition=<name>：设定采用队列的作业。
--reservation=<name>：设定采用了预留测略的作业。
--nodelist=<name>：设定采用特定节点名的作业，格式类似node[1-10,11,13-28]。

squeue：查看作业信息

--format=<spec>：格式化输出。
--jobid<job_id_list>：设定作业号。
--name=<name>：设定作业名。
--partition=<name>：设定采用队列的作业。
--start：显示作业开始时间。
--state=<state_list>：显示特定状态的作业信息。（ST：状态。PD：排队中，PENDING。R：运行中，RUNNING。CA：已取消，CANCELLED。CF：配置中，CONFIGURING。CG：完成中，COMPLETING。CD：已完成，COMPLETED。CF：已失败，FAILED。TO：超时，TIMEOUT。NF：节点失效，NODE FAILURE。SE：特殊退出状态，SPECIAL EXIT STATE。）

Slurm作业管理

scontrol：查看作业、节点和队列等信息

--details：显示更详细信息。

--oneline：所有信息显示在同一行。

show ENTITY ID：显示特定入口信息，ENTITY可为：job、node、partition等，ID可为作业号、节点名、队列名等。

update SPECIFICATION：修改特定信息，用户一般只能修改作业的。

如scontrol show jobid=1查看作业ID为1的详情；

如scontrol update jobid=<jobID> timelimit=12:00:00更改作业运行时长限制（普通用户只可修改自己的作业，且只能缩短时间限制，无权延长时间限制，如需延长需联系管理员）。

如scontrol update jobid=<jobID> partition=gpu gres=gpu:1 更改作业运行的队列和所需GPU数量（仅限更改处于排队状态的作业）。

如scontrol update jobid=<jobID> ReqNodeList=<nodelist> 更改作业运行的计算节点（仅限提交脚本中使用-w参数指定过作业运行节点才可以使用该命令，更改处于排队状态的作业申请运行的节点）

Slurm作业阵列

很多时候我们需要批量运行一组作业，这些作业所需的资源、运行的参数非常相似，只是一些如输入文件名参数不相同。这个时候借助 Job Array 就可以很方便地批量提交这些作业。Job Array 中的 每一个作业在调度时视为独立的作业，仍然受到队列以及服务器的资源限制。

在 SLURM 脚本中使用 #SBATCH -a <range> 即可指定 Job Array 的数字范围，其中的 range 参数需要是连续或不连续的整数。

下面几种参数指定方式都是合法的：

#SBATCH -a 0-9 作业编号范围是 0 到 9，均含边界。

#SBATCH -a 0,2,4 作业编号是 0,2,4。

#SBATCH -a 0-9:3 同上，作业编号为 0,3,6,9，即0~9之间按间隔 3连续取数。

#SBATCH -a 0-9,20,40 混合指定也是可以的。

在脚本运行中，SLURM 使用环境变量来表示作业数组，具体为:

SLURM_ARRAY_JOB_ID 作业数组中第一个作业的 ID。

SLURM_ARRAY_TASK_ID 该作业在数组中的索引。

SLURM_ARRAY_TASK_COUNT 作业数组中作业总数。

SLURM_ARRAY_TASK_MAX 作业数组中最后一个作业的索引。

SLURM_ARRAY_TASK_MIN 作业数组中第一个作业的做引。

可用以上变量来区分不同组内的任务，以便于处理不同的输入参数。对于每个数组内的作业，它的默认输出文件的命名方式为 slurm-JOBID_TASKID.out。

对于作业管理来讲，可以批量管理JOBID整个阵列作业，也可以管理JOBID_TASKID阵列中的某个具体作业。管理命令和格式与常规作业任务相同，只是输入的作业号格式略有区别。

集群脚本申请资源的一些技巧

F：我的程序是单进程单线程的串行程序还是OpenMP单进程多线程程序的并行程序，怎么区分与编写作业脚本申请合适的资源？

Q：可以手动运行程序，然后通过如下查询命令（如查询python任务的进程和子进程信息情况），查询进程的线程数：

ps -ef |grep python |awk ' {print $1,$2,$3,$8} ' |sort -k 3

查询结果如右图，第一列为程序所用的用户名，第二列为进程的线程id,第三列为进程id，第四列为程序名，如果同一个进程id下有多个线程，则为OpenMP程序，如果只有1个线程，则为串行程序，在通过作业脚本申请资源时，所需要申请的CPU核心与线程数量保持一致。

```
u214205+ 61070 59357 python3 -u run.py --RUN=train
u214205+ 61090 59359 python3 -u run.py --RUN=train
u214205+ 61099 59359 python3 -u run.py --RUN=train
u214205+ 61108 59359 python3 -u run.py --RUN=train
u214205+ 61125 59359 python3 -u run.py --RUN=train
u214205+ 61133 59359 python3 -u run.py --RUN=train
u214205+ 61144 59359 python3 -u run.py --RUN=train
u214205+ 61146 59359 python3 -u run.py --RUN=train
u214205+ 61149 59359 python3 -u run.py --RUN=train
u214205+ 61005 59361 python3 -u run.py --RUN=train
u214205+ 61015 59361 python3 -u run.py --RUN=train
u214205+ 61023 59361 python3 -u run.py --RUN=train
u214205+ 61029 59361 python3 -u run.py --RUN=train
u214205+ 61033 59361 python3 -u run.py --RUN=train
u214205+ 61038 59361 python3 -u run.py --RUN=train
u214205+ 61047 59361 python3 -u run.py --RUN=train
u214205+ 61053 59361 python3 -u run.py --RUN=train
u214205+ 6094 6093 python -u test.py
u214205+ 12175 954 python test.py
```

进程号为5939的python3下面共有8个子进程，这样的任务，需要在作业脚本中申请1节点8个CPU核。GPU卡数量按需申请。

进程号为6093的python下面共有1个子进程，这样的任务，需要在作业脚本中申请1节点1个CPU核即可，GPU卡数量按需申请。

F：我提交了一个申请8块GPU卡作业任务，但是一直排队，为什么？

Q：单节点共计8块GPU卡，作业调度系统按作业先到先算、资源先符合先算的策略调度，如果其他用户申请的GPU卡数量少，如只申请1块或2块GPU卡，则节点如果有GPU资源释放出来，即满足其它用户申请的资源条件，作业就被优先调试执行，所以节点同时空出8张GPU卡的概率就很少，从而导致申请8张卡的作业一直处理排队状态，此时可以减少GPU卡数量，来确保作业能够优先调度，可执行control update jobid=<jobID> partition=gpu gres=gpu:1 调整作业申请的GPU卡数量。

F：我提交了作业直接被取消运行了是怎么回事？

Q：可以查看作业是否有输出日志文件，从日志中分析作业取消的原因，如果没有输出日志，应该是机时余额不够，请联系管理员对账户机时余额进行充值，或将作业脚本中申请的总机时控制在可用余额范围。余额可以web页面查看。

F：作业扣费机制是什么？

Q：集群按申请多少，就预扣多少，预扣后的余额可供下一次作业预约，作业退出后，使用多少扣除多少，未使用机时自动退回账户余额中，注意：扣费会按照申请的总核数*实际任务运行时间来进行最终扣费，只有实际任务运行时间是动态的，核数是按申请多少就是多少来计算的，不是按程序实际使用多少核来计算的，

运行vasp的示例

```
#!/bin/bash
#SBATCH -J vast_jobname
#SBATCH -p normal
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
# SBATCH --time=24:00:00

cd $SLURM_SUBMIT_DIR

source /public/software/profile.d/compiler_intel-compiler-2021.3.0.sh
source /public/software/profile.d/mpi_intelmpi-2021.3.0.sh
source /public/software/profile.d/apps_vasp-intelmpi-5.4.4.sh

srun hostname -s | sort -n >slurm.hosts
mpirun -np $SLURM_NPROCS -machinefile slurm.hosts vasp_std >>run.log
```

vasp_std（Standard版本，即标准版，常用版本）
vasp_gam（Gamma版本，用于单k点计算）
vasp_ncl（non-collinear版本，即非共线版）。当开启非共线计算时（LNONCOLLINEAR = T，计算SOC时往往会开启），需要使用对应的vasp_ncl版本提交计算。

运行vasp VTST版本的示例

```
#!/bin/bash
#SBATCH -J vast_jobname
#SBATCH -p normal
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
# SBATCH --time=24:00:00

cd $SLURM_SUBMIT_DIR

source /public/software/profile.d/apps_vasp-5.4.4-vtst-openmpi-4.0.3.sh

srun hostname -s | sort -n >slurm.hosts
mpirun -np $SLURM_NPROCS -machinefile slurm.hosts vasp_std >>run.log
```

vasp_std（Standard版本，即标准版，常用版本）
vasp_gam（Gamma版本，用于单k点计算）
vasp_ncl（non-collinear版本，即非共线版）。当开启非共线计算时（LNONCOLLINEAR = T，计算SOC时往往会开启），需要使用对应的vasp_ncl版本提交计算。

使用作业阵列批量运行Gaussian的示例

```
#!/bin/bash
#SBATCH -J g16
#SBATCH -p normal
#SBATCH -N 1                          #每个高斯任务仅申请1个节点
#SBATCH --ntasks-per-node=8           #每个高斯任务申请使用8个核，与每个高斯算例的模型文件中%nprocs=8定义的数字必须保持一致
#SBATCH --time=1:00:00                #每个高斯任务最长运行1小时

cd $SLURM_SUBMIT_DIR
JOB=quintet-Fe-heme-OMe.gjf
JOBNAME=`basename "$JOB".gif`
source /public/software/profile.d/apps_g16.sh
g16  <$JOB > "$JOBNAME.log"
```

```
[user00@tc6000 g16]$ cat /public/software/profile.d/apps_g16.sh
#!/bin/bash -x
g16root=/public/software/apps/g16
export GAUSS_SCRDIR=/tmp   定义g16运行时的临时文件存储路径
source $g16root/g16/bsd/g16.profile
[user00@tc6000 g16]$
```

如果计算过程的临时文件不大于计算节点/目录容量，则可将计算临时文件按上面定义，会提升高斯计算效率，如果大于，则需要将路径定义到自己家目录下。

```
[user00@tc6000 g16]$ cat quintet-Fe-heme-SMe-sp_def2tzvp.gjf
%chk=quintet-Fe-heme-SMe-sp_def2tzvp.chk
%mem=84GB   限制运行时最大使用内存，防止内存耗尽导致程序崩溃
%nprocshared=64   设定g16最大线程数，一般与运行程序所在计算节点总核数相同
#p uwb97xd def2tzvp g09def scf=yqc

Title Card Required

-1 5
 Fe                   0.11442200   -0.08990900    0.40586800
 N                   -1.46211000   -1.44460700   -0.22758900
 N                    1.24629800    1.47860200    0.21849500
```

在算例文件中定义内存约束和总线程约束，其中线程约束需要与作业脚本--ntasks-per-node值一致。

使用作业阵列批量运行Gaussian的示例

```
#!/bin/bash
#SBATCH -J g16
#SBATCH -p normal
#SBATCH -N 1                          #每个高斯任务仅申请1个节点
#SBATCH --ntasks-per-node=8           #每个高斯任务申请使用8个核，与每个高斯算例的模型文件中%nprocs=8定义的数字必须保持一致
#SBATCH --time=1:00:00                #每个高斯任务最长运行1小时
#SBATCH -a 0-4                        #批量作业阵列数组设置的行，总共批量运行4个高斯任务

cd $SLURM_SUBMIT_DIR
source /public/software/profile.d/apps_g16.sh
g16<"$SLURM_ARRAY_TASK_ID"_beta.gjf> "$SLURM_ARRAY_TASK_ID"-beta.log
#运行高斯程序，高斯读入算例模型文件名的数字部分使用“$SLURM_ARRAY_TASK_ID”变量代替
```

将该脚本文件与高斯模型文件放到同一个文件夹目录中，该示例中算例文件名实际分别为：1_beta.gjf、2_beta.gjf、3_beta.gjf、4_beta.gjf
因为是批量提交，文件名必须用连续的数字代替。然后sbatch提交该作业脚本即可。

注：如果模型文件名数字不连续，如1,5,7，则阵列数组设置的行可为：$SBATCH -a 1,5,7
如果文件名数字有规律，如0，3，6，9,中间间隔为3，则阵列数组设置的行可为：$SBATCH -a 0-9:3

阵列中的作业ID编号格式为：<作业ID>_<任务ID>，如50_0、50_1、50_2、50_3，在管理阵列作业时，管理命令格式与普通作业一致，但需注意的是如果只输入作业ID，则管理的是整个阵列中的作业，如果输入具体的阵列作业的ID编号，则只管理该作业任务。

运行gromacs的作业脚本

```
#!/bin/bash
#SBATCH -J gromacs_jobname
#SBATCH -p normal
#SBATCH -N 2
#SBATCH --ntasks-per-node=64
#SBATCH --time=24:00:00

cd $SLURM_SUBMIT_DIR

source /public/software/profile.d/compiler_intel-compiler-2021.3.0.sh
source /public/software/profile.d/mpi_intelmpi-2021.3.0.sh
source /public/software/profile.d/compiler_gcc-7.3.1.sh
source /public/software/profile.d/apps_gromacs-intelmpi-2020.6.sh
srun hostname -s | sort -n >slurm.hosts

mpirun -np $SLURM_NPROCS -machinefile slurm.hosts gmx_mpi grompp -f rf_verlet.mdp >>log.dat
mpirun -np $SLURM_NPROCS -machinefile slurm.hosts gmx_mpi mdrun >>log.dat
```

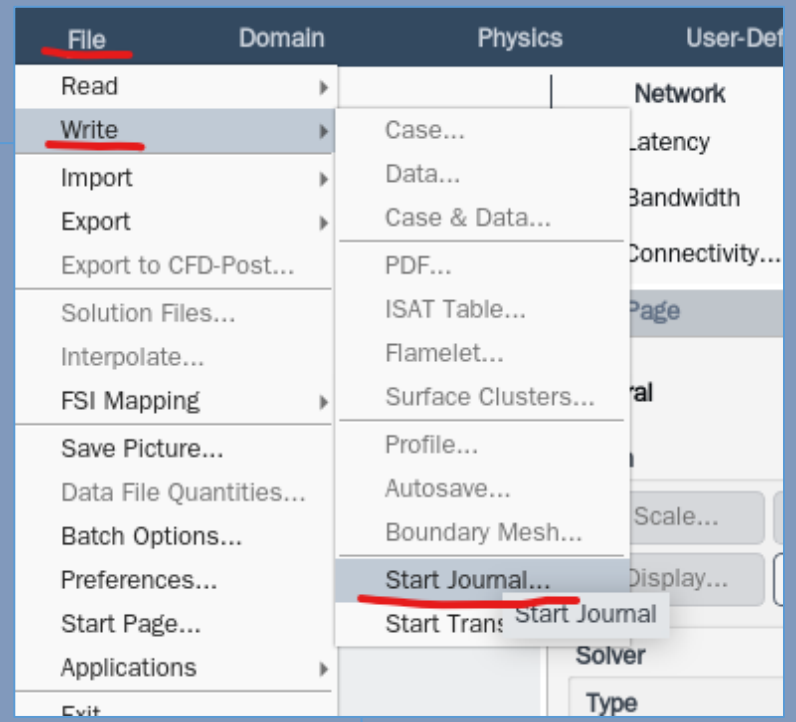
运行fuent的作业脚本

```
#!/bin/bash
#SBATCH -J fluent_test
#SBATCH -p normal
#SBATCH -N 10
#SBATCH --ntasks-per-node=24
#SBATCH --time=24:00:00

cd $SLURM_SUBMIT_DIR

source /public/software/profile.d/apps_ansys194.sh
source /public/software/profile.d/mpi_intelmpi-2021.3.0.sh

srun hostname -s | sort -n >slurm.hosts
fluent 2d -t$SLURM_NPROCS -pinfiniband -cnf=./slurm.hosts -ssh -mpi=intel -g -i EOFluentInput >run.log
```



fluent 2d -t$SLURM_NPROCS -pinfiniband -cnf=. /slurm.hosts -mpi=intel -nm -ssh -license=enterprise -g -i EOFluentInput >run.log
21版ansys如果许可兼容enterprise、 premium和pro版，则如上红色部分定义调用许可版本。

EOFluentInput是fluent joumal自动运行脚本，内容如右下角部分，该脚本可以通过如右上角fluent图形界面的start joumal菜单进行录制，录制时，正常进行图形界面操作即可，全过程部门都会保存到指定文件中，注意，最后必须以exit和、yes/no结尾，否则，脚本提交的fluent计算任务即使完成也不会退出。

```
file/read-case engine1.cas
parallel/partition/method/cartesian-axes 4
solve/initialize/initialize-flow
solve/iterate 10000
exit
yes
```

运行CFX的作业脚本

```
#!/bin/bash
#SBATCH -J cfx_test
#SBATCH -p normal
#SBATCH -N 10
#SBATCH --ntasks-per-node=24
#SBATCH --time=24:00:00

cd $SLURM_SUBMIT_DIR
srun hostname -s | sort -n >slurm.hosts
hostlist=""
for node in `cat slurm.hosts| uniq`
do
num=`cat slurm.hosts | grep $node | wc -l`
if [ -z $hostlist ]
then hostlist=$node*$num
else hostlist=$hostlist,$node*$num
fi
done
echo $hostlist
source /public/software/profile.d/apps_ansys194.sh
source /public/software/profile.d/mpi_intelmpi-2021.3.0.sh
cfx5solve -size 2 -def $CFX_DEF_FILE -ini $CFX_RES_FILE -part-mode rcb -par-dist $hostlist -start-method "HP MPI
Distributed Parallel"
```

运行comsol的作业脚本示例一

```
#!/bin/bash
#SBATCH -N=2          #申请使用2个节点
#SBATCH --ntasks-per-node=64       #每个节点申请使用64个核心
#SBATCH --job-name="COMSOL"
#SBATCH --mem= 8000      #限制作业运行的内存大小8000MB，根据实际修改，
#SBATCH --time=4:00:00
#SBATCH --partition=normal

cd $SLURM_SUBMIT_DIR
source /public/software/profile.d/apps_comsol56.sh

MODELTOCOMPUTE="mymodel.mph"
INPUTFILE="${HOME}/input/${MODELTOCOMPUTE}"
OUTPUTFILE="${HOME}/output/${MODELTOCOMPUTE}"
BATCHLOG="${HOME}/logs/${MODELTOCOMPUTE}.log"

comsol batch -mpibootstrap slurm -inputfile ${INPUTFILE} -outputfile ${OUTPUTFILE} \ -batchlog ${BATCHLOG} -alivetime 15 -
prefermph -recover -mpidebug 10
```

该脚本为由comsol识别slurm调度系统，并由slurm调用srun提交任务并行。

如果模型较大，计算过程的临时文件大小会超过系统临时目录大小，比如预估会超过300GB，则需要重新定义comsol临时文件存储路径，一般可定义到模型文件同一目录下的某个文件夹即可，定义的方式是在comsol启动参数中加入：
-prefsdir /临时文件自定义目录绝对路径 -tmpdir /临时文件自定义目录绝对路径 -configuration /首选项文件自定义目录绝对路径

具体可参考：http://cn.comsol.com/support/knowledgebase/1083

运行comsol的作业脚本示例二

```
#!/bin/bash
#SBATCH -N=2          #申请使用2个节点
#SBATCH --ntasks-per-node=64        #每个节点申请使用64个核心
#SBATCH --job-name="COMSOL"
#SBATCH --mem= 8000       #限制作业运行的内存大小8000MB，根据实际修改，
#SBATCH --time=4:00:00
#SBATCH --partition=normal

cd $SLURM_SUBMIT_DIR
source /public/software/profile.d/apps_comsol56.sh
srun hostname -s |uniq >slurm.hosts
NODEPROCS= ${SLURM_TASKS_PER_NODE:0:1}

MODELTOCOMPUTE="mymodel.mph"
INPUTFILE="${HOME}/input/${MODELTOCOMPUTE}"
OUTPUTFILE="${HOME}/output/${MODELTOCOMPUTE}"
BATCHLOG="${HOME}/logs/${MODELTOCOMPUTE}.log"

comsol batch -nn $SLURM_NPROCS -nnhost $NODEPROCS -np 1 -f slurm.hosts -inputfile ${INPUTFILE} -outputfile ${OUTPUTFILE} -
batchlog ${BATCHLOG}
```

该脚本为由comsol调用内置mpi启动任务并行，其中-nn定义并行进程总数量，-nnhost是每台物理服务器上启动多少个进程，-np是comsol进程又启动多少个线程。-f是指并行任务运行在哪些物理服务器上。

混合计算各值如何优化使用，可参考：

1：在集群上并行运行 COMSOL
https://cn.comsol.com/support/knowledgebase/1001

2：COMSOL 博客 混合计算：共享内存与分布式内存相结合的优势
https://cn.comsol.com/blogs/hybrid-computing-advantages-shared-distributed-memory-combined/

运行GPU的作业脚本示例

```
#!/bin/bash
#SBATCH -J gpu_test
#SBATCH -p gpu
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --gres=gpu:2
#SBATCH --time=24:00:00


cd $SLURM_SUBMIT_DIR

source ~/*/anaconda*.*

CUDA_VISIBLE_DEVICES=$SLURM_JOB_GPUS
python -u train.py>train.txt>&1 &
```

$SLURM_JOB_GPUS定义了使用GPU的物理ID，该ID为slurm作业调度系统自动分配，
如申请使用2块卡，slurm分配了使用节点的3，5号卡来使用，则：
CUDA_VISIBLE_DEVICES=$SLURM_JOB_GPUS
实际即为CUDA_VISIBLE_DEVICES=3,5即只有3，5号GPU卡对程序可见，在代码中，gpu[0]则是指3号GPU卡，GPU[1]则是指5号GPU卡。

为了保证anaconda环境的稳定，建议自行在家目录下安装所需的anaconda环境。脚本中source的变量为自己所安装的anaconda环境变量。

运行matlab串行计算例子

```
#!/bin/bash
#SBATCH -J matlab_job
#SBATCH -p normal
#SBATCH -N 1
#SBATCH --ntasks-per-node=1
#SBATCH --time=24:00:00


cd $SLURM_SUBMIT_DIR
source
/public/software/profile.d/apps_matlab2020a.sh

matlab -nodesktop -nosplash -nodisplay -r test
```

```
% for i = 1:2048
A = zeros(100,1);
parfor i = 1:100
A(i) = sin(i*2*pi/20);
end
save myResult.mat A
```

默认情况下matlab程序只会使用到单核心，代码文件名为test.m，在matlab运行的命令行中，不需要加入后缀名.m，代码内容如右图所示。

-nodesktop -nosplash -nodisplay 表示禁用 MATLAB 的图形界面和 欢迎界面，而 -r 选项的含义是执行代码文件。

运行matlab单节点并行计算例子

```
#!/bin/bash
#SBATCH -J matlab_job
#SBATCH -p normal
#SBATCH -N 1
#SBATCH --ntasks-per-node=64
#SBATCH--time=24:00:00


cd $SLURM_SUBMIT_DIR
source
/public/software/profile.d/apps_matlab2020a.sh

matlab -nodesktop -nosplash -nodisplay -r test
```

```
parpool('local’ ,64)
% parfor i = 1:2048
A = zeros(100,1);
parfor i = 1:100
A(i) = sin(i*2*pi/20);
end
save myResult.mat A
```

代码内容如右图所示，注意：parpool的并行的数量不要超过申请的核心的数量。