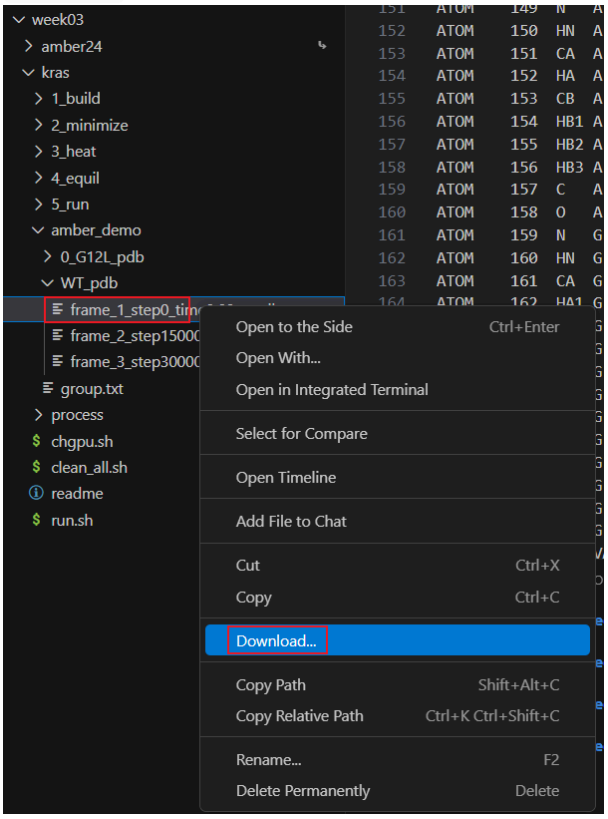
1. 文件修改核对表

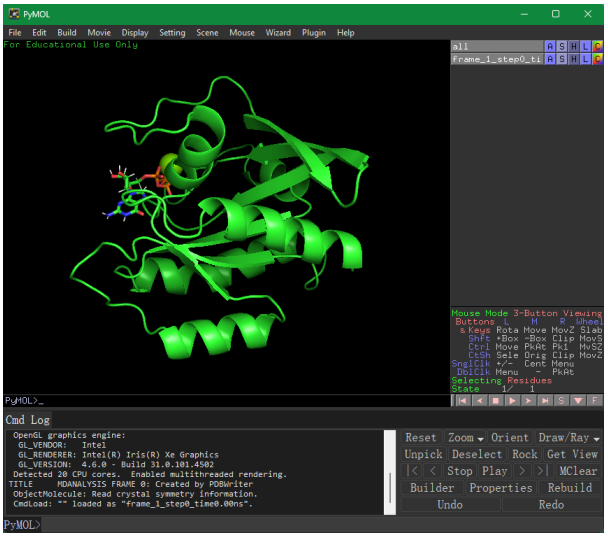
- [ ] 突变12号残基，2.2步骤2
- [ ] 体系电中性，2.2步骤4
- [ ] GPU编号，2.3~2.7
- [ ] 保存原子数，2.5~2.7步骤4
- [ ] 长时间模拟预约GPU，2.7步骤2

2. 生物背景

1. 同步文件
   将文件同步到个人目录
   ```
   rsync2 /home/data/public/week03/ ~/work/{学号}/week03
   ```
2. 下载初始的结构文件
   在VSCode文件管理器的文件`/home/md_ta/work/week03/kras/amber_demo/WT_pdb/frame_1_step0_time0.00ns.pdb`上点击右键
   

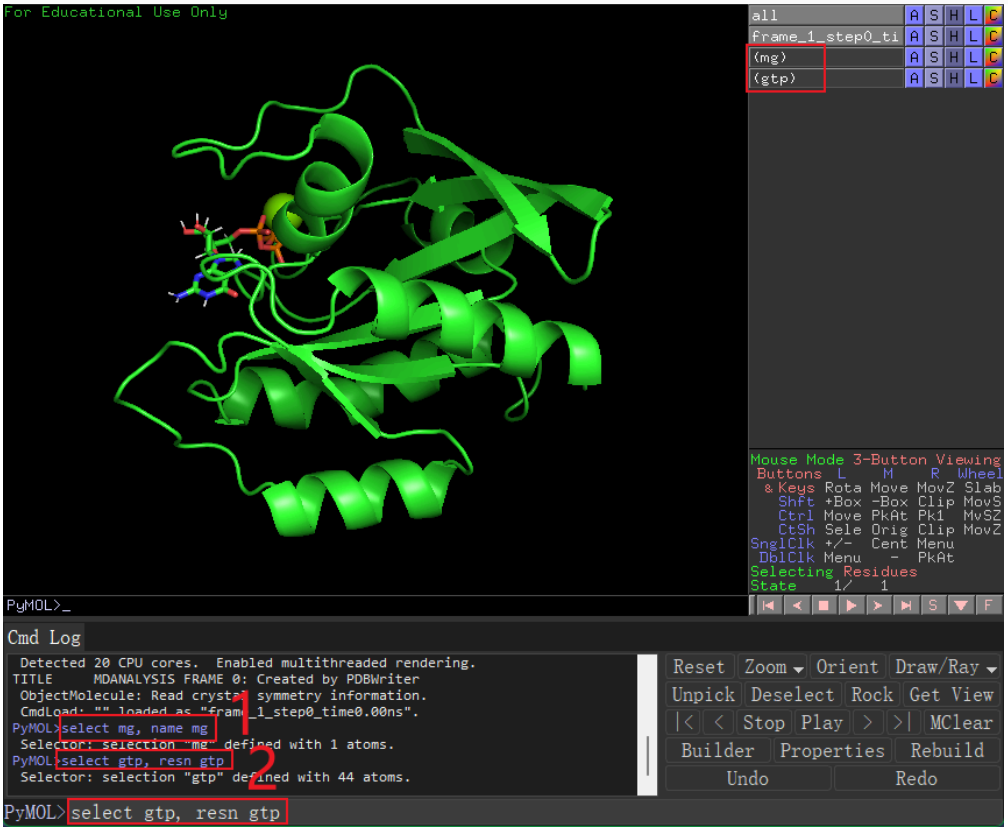
3. 在本地打开结构文件
   如果本地没有安装PyMOL，详见biocomput:guide

在本地双击PDB文件打开



4. 选中镁离子和GTP
   根据原子名 `mg` 选中镁离子，并创建一个名为 `mg` 的选择
   同理，根据残基名 `gtp` 选中GTP整个分子
   `select` 是PyMOL的内置指令，紧跟一个空格，后面是所有的参数，用逗号隔开
   第一个参数是选择的名字
   逗号之后是第二个参数，是基于PyMOL的选择语法，详见[Selection Algebra - PyMOL Wiki]

```
select mg, name mg
select gtp, resn gtp
```

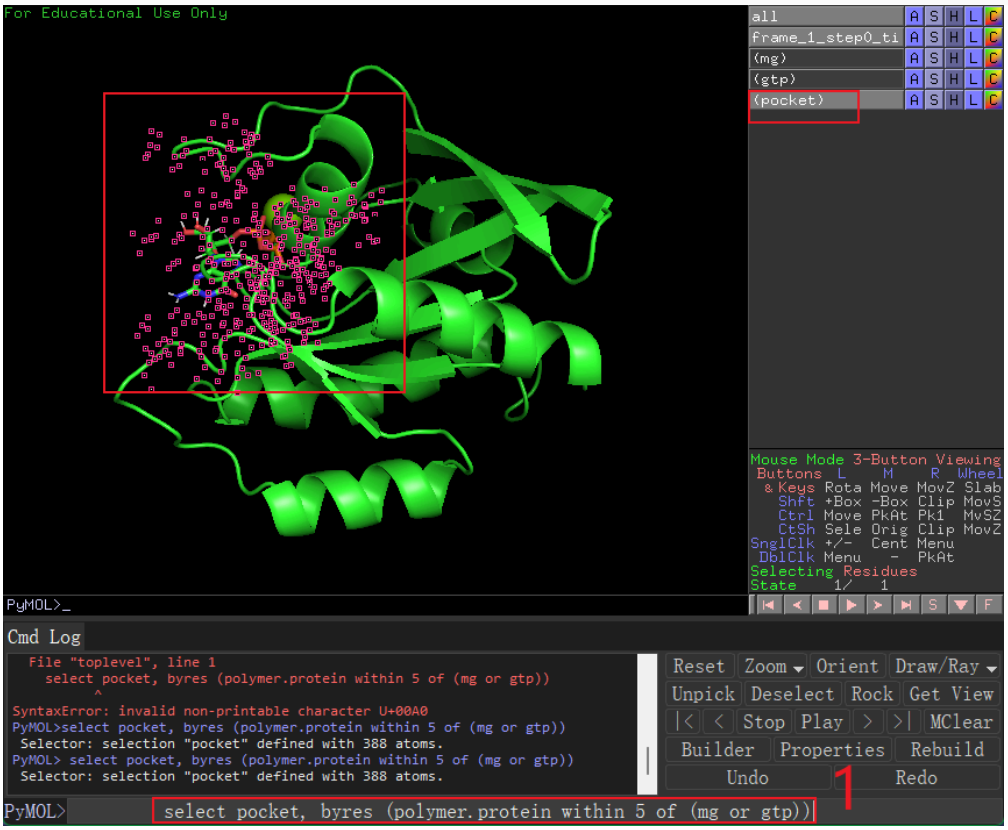


5. 选中结构口袋
   将镁离子和GTP分子周围5 A之内的残基选中
   `mg or gtp` 表示这两个选择的并集
   `polymer.protein` 表示体系中所有的蛋白质

`polymer.protein within 5 of ...`表示所有和镁离子和GTP距离在5 A之内的蛋白质体系的原子

`byres ...`表示后方紧跟着的选择所涉及的所有完整残基

```
select pocket, byres (polymer.protein within 5 of (mg or gtp))
```



6. 将口袋设计的残基表示为lines

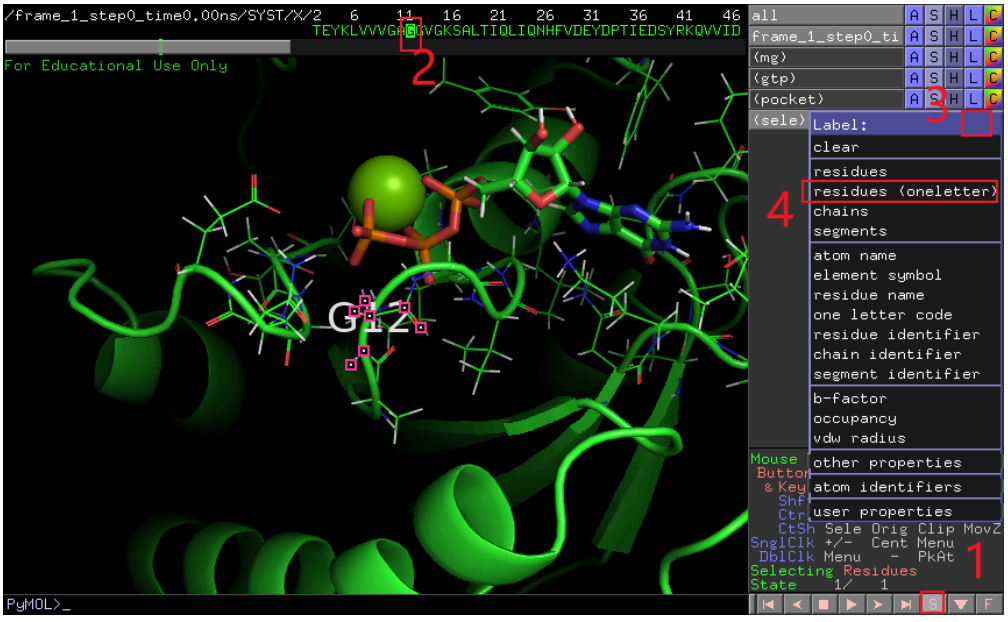
```
show lines, pocket
```

7. 观察12号残基周围

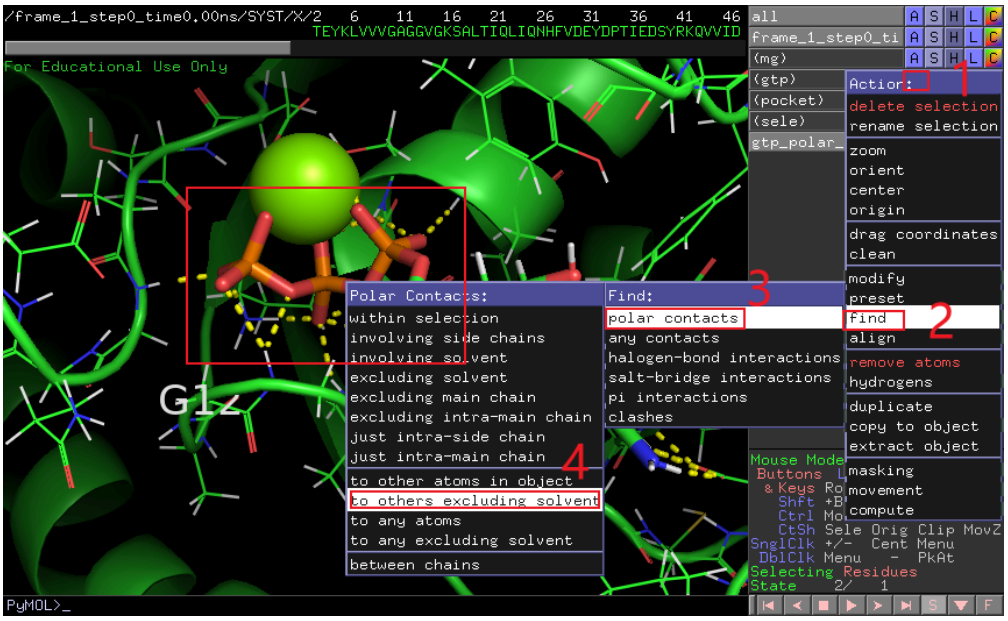
   点击右下角`S`按钮，以显示序列

   鼠标中键点击12号残基对应的`G`，可将该残基居中显示

   左键点击12号残基，点击`sele`的`L`按钮，并点击`residues (oneletter)`，以表注该残基



8. 显示GTP与口袋形成的氢键
   点击 `gtp` 的 `A` 按钮，然后依次点击下列各个按钮
   黄色的线即是可能存在的相互作用（氢键/盐桥）



3. 课后练习

Kras是一个GTP酶，我们提供了一个Kras与GTP结合的初始结构。我们希望模拟Kras的12号残基的多种突变，观察不同突变体与GTP结合的差异。

全班一共10个组，每个组分配到一个突变，每位组员独立的完成一次突变体的模拟，所以一共是十种突变，各三次（或两次）重复。

在进行最后一步长时间的模拟前，一定要对照 `1.文件修改核对表` 确定各项已经完成。手动操作的部分，有些繁琐，但是我们希望对大家的理解能有帮助。

3.1. 建立体系

1. 找到对应组别需要突变的残基，位置在 `week03/kras/amber_demo/group.txt`
2. 突变12号残基
   手动打开文件 `week03/kras/1_build/frame_1....pdb` ，将表示12号残基的部分，仅保留原子名为 `N` / `CA` / `C` 的三行，并将残基名从 `GLY` 改为本组突变的名字，残基名详见 `group.txt`
   修改前：

```
159   ATOM   157  C   ALA X  11      26.270  33.860  26.720  1.00  0.00      SYST
160   ATOM   158  O   ALA X  11      26.500  34.390  27.790  1.00  0.00      SYST
161   ATOM   159  N   GLY X  12      25.350  34.340  25.850  1.00  0.00      SYST
162   ATOM   160  HN  GLY X  12      25.200  33.880  24.980  1.00  0.00      SYST
163   ATOM   161  CA  GLY X  12      24.390  35.450  26.180  1.00  0.00      SYST
164   ATOM   162  HA1 GLY X  12      24.970  36.280  26.560  1.00  0.00      SYST
165   ATOM   163  HA2 GLY X  12      23.770  35.650  25.320  1.00  0.00      SYST
166   ATOM   164  C   GLY X  12      23.510  35.170  27.410  1.00  0.00      SYST
167   ATOM   165  O   GLY X  12      23.020  34.070  27.480  1.00  0.00      SYST
168   ATOM   166  N   GLY X  13      23.200  36.170  28.220  1.00  0.00      SYST
169   ATOM   167  HN  GLY X  13      23.580  37.080  28.070  1.00  0.00      SYST
```

修改后：

```
159    ATOM    157  C    ALA X  11      26.270  33.860  26.720  1.00  0.00       SYST
160    ATOM    158  O    ALA X  11      26.500  34.390  27.790  1.00  0.00       SYST
161    ATOM    159  N    LEU X  12      25.350  34.340  25.850  1.00  0.00       SYST
162    ATOM    161  CA   LEU X  12      24.390  35.450  26.180  1.00  0.00       SYST
163    ATOM    164  C    LEU X  12      23.510  35.170  27.410  1.00  0.00       SYST
164    ATOM    166  N    GLY X  13      23.200  36.170  28.220  1.00  0.00       SYST
165    ATOM    167  HN   GLY X  13      23.580  37.080  28.070  1.00  0.00       SYST
```

3. 通过脚本 `convert.py` 清洗PDB文件
   在文件管理器 `week03/kras/1_build` 处右键，点击第三行的 `Open in Integrated Terminal` 以打开一个位于此处的终端，在打开的终端中运行指令 `python convert.py`
   观察新生成的文件 `input.pdb`
4. 利用程序 `tleap` 生成拓扑结构文件 `wbox.prmtop` 和 `wbox.inpcrd` 文件
   执行指令 `tleap -s -f ./leap.in`，根据文件输出中的 `The unperturbed charge of the unit xxx is not zero` 来判断，是否需要修改 `leap.in` 文件中的 `addions wbox` 钾离子和钠离子的个数，反复尝试执行上述指令确保体系电中性
   观察新生成的文件 `wbox.prmtop` / `wbox.inpcrd` / `wbox.pdb`，以及日志文件 `leap.log`
5. 利用脚本 `use4fs.sh` 重新分配氢原子质量，便于增大步长
   执行指令 `./use4fs.sh`
   观察新生成的文件 `wbox.prmtop`，比较与旧文件 `wbox_tmp.prmtop` 的异同（搜索 `FLAG MASS`，这一部分的每个数字按顺序表示每个原子的相对原子质量，对比 `wbox.pdb` 观察哪些原子的质量改变了，改变了多少）
6. 利用程序 `cpptraj` 创建一个仅包含溶质部分的结构文件
   执行指令 `cpptraj -i strip.in`
   观察新生成的文件 `mol.prmtop` / `mol.pdb`

3.2. 体系能量最小化

1. 同3.1，打开一个在 `2_minimize` 处的终端
2. 修改脚本 `run.sh` 中指定的GPU编号
   打开文件，修改第3行中 `export CUDA_VISIBLE_DEVICES="1"` 的数字，从0-7任选一个
3. 运行脚本 `run.sh`
   执行指令 `./run.sh`

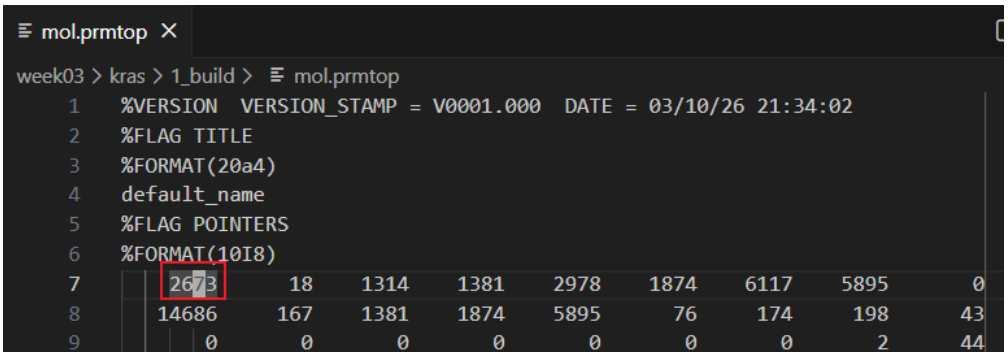
3.3. 对体系升温

1. 同3.1，打开一个在 `3_heat` 的终端
2. 修改脚本 `run.sh` 中的GPU编号
3. 运行脚本 `run.sh`

3.4. 预平衡体系

1. 同3.1，打开一个在 `4_equil` 的终端
2. 修改脚本 `run.sh` 中的GPU编号

3. 确定溶质部分总原子数
   打开文件 `1_build/mol.prmtop`，记住第7行第一列的数，为我们想要溶质部分的总原子数，随突变残基种类会发生改变。



4. 修改文件 `equil.in`
   打开文件，修改第12行 `ntwprt = XXXX` 的数字为溶质部分总原子数（在当前的问题中，我们并不感兴趣水的运动，为了节省硬盘空间，节省文件读写压力，在运行轨迹中仅保留溶质的部分）（回过头观察 `1_build/strip.in` 以及 `wbox.pdb` / `mol.pdb`，思考2.2的第六步发生了什么，in文件为什么要这么写）
5. 运行脚本 `run.sh`

3.5. 长时间模拟（测试）

同3.4

1. 同2.1，打开一个在 `5_run` 的终端
2. 修改脚本 `run.sh` 中的GPU编号
3. 修改文件 `run.in` 中变量 `ntwprt`
4. 运行脚本 `run.sh`

3.6. 长时间模拟（正式）

基本同3.5，区别是GPU编号的设定和模拟的起始/终止

1. 同2.1，打开一个在 `5_run` 的终端
2. 修改脚本 `run.sh` 的变量 `istart` 和 `iend`
   两个变量的数值，表示模拟的起始时间和终止时间（闭区间），初始为1/1，表示运行1 ns至1 ns共1 ns长的模拟，完成后可以改成2/1000，表示运行2 ns至1000 ns共999 ns长的模拟
3. 修改脚本 `run.sh` 中的GPU编号
   计算资源有限，无法满足每人一块卡，所以需要一个预约机制
   预约机制：参考在线预约文档，填写预约信息（起始时间/终止时间/预约人）
   取消预约：1 ns的模拟在单块卡的时长在67秒上下，参考第二步计算需要运行的模拟时长，填写预约终止时间，并在任务确定完成后取消预约
   纠错机制：在任意终端执行指令 `nvidia-smi` 或 `smi` 可以查看当前所有显卡的运行状态，如果发现有显卡在一小时前即被预约但没有任务运行，可以在群中联系该同学确认情况，如果一定时间没有反馈，可以覆盖此处预约（可能是该同学遇到了不容易解决的错误，也可能是该同学的任务已经运行结束，但还没有取消预约）

下图表示GPU0正在运行任务

```
(base) md_ta@oulab-cs10:kras$ nvidia-smi

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|    0   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    0   N/A  N/A    913236      C   pmemd.cuda                        572MiB |
|    1   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    2   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    3   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    4   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    5   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    6   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
|    7   N/A  N/A      2804      G   /usr/libexec/Xorg                   4MiB |
+-----------------------------------------------------------------------------+
```

4. 修改文件 `run.in` 中变量 `ntwprt`
5. 运行脚本 `run.sh`
   如果是第一次运行该脚本，建议将 `istart` / `iend` 设置较短的范围，可以在第一步打开的终端执行 `./run.sh` ，通过 `runXXX.out` 文件可以简单判断运行进程，该终端不能关闭
   如果是确定脚本可以正常运行，且第二步中GPU的预约正常，显卡上也没有任务，可以执行 `Run` (原理是执行了 `nohup ./run.sh > log.log 2>&1 < /dev/null &` ，将任务挂到了后台，且不会受到终端关闭的影响)（关闭VSCode前务必退出该终端）
6. 等待任务运行
   特殊情况下，任务可能中断，需要重新预约GPU并修改变量 `istart` 为生成的最后一个nc文件，通过在第一步打开的终端执行指令 `lt *.nc` （比如最后一个文件是 `run363.nc` ，任务中断了，说明363 ns出现了问题，需要重新跑，所以 `istart` 需要指定为363，而不是364）
   1000 ns的模拟，大概需要18个小时，运行速率可以在任意一个已完成的输出文件 `runXXX.out` 的末尾看到

另:

1. GPU预约的规则看上去有点复杂，但是初心是希望大家能有序地使用计算资源。自己要用了就预约，不用了就取消预约；预约之后，算一下大概要用的时间，填好预期终止时间，以便在显卡被占满的情况下，其他同学可以确定自己下次什么时候再来看；如果显卡被预约但空闲可以在沟通后覆盖上一次预约，因为长时间（暂定一个小时内）没有被使用，则大概率显卡是空闲的。
2. 多个任务挤占同一块卡的危害比想象中更大，如果单卡单任务需要一分钟，即一个任务需要一个GPU分钟，两块卡各运行一个任务也需要一分钟，即两个任务需要两个GPU分钟，单卡两任务可能需要三分钟，即两个任务需要三个GPU分钟，效率大大降低。
3. 理想情况下，一周内服务器可以完成70微秒的模拟，而我们只有28位同学，所以资源是够的，希望大家能友好相处，有序完成。