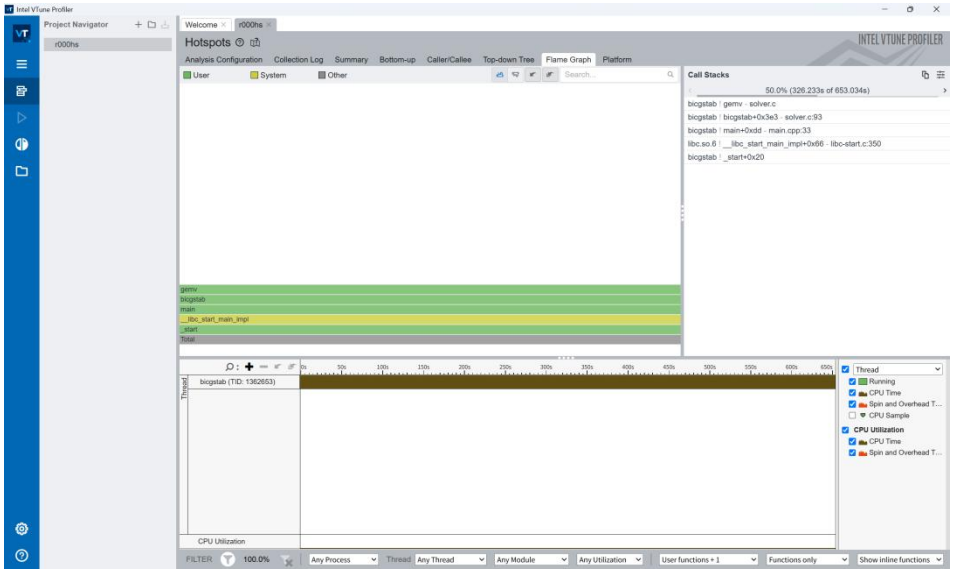
Lab4

3240104875
王耀
2025/8/6

Profile

耗时最长的函数时 gemv，不过这个 flame graph，只有 gemv 的使用是 99.99%，其他都是 100%，难道是因为 gemv 的调用在最内层吗？



openMP 线程级并行

在这里先提一嘴，-O3 的才能略微能算，1e6 次计算还是太大了，我决定改小一点，1e5 吧，如果之后能优化得很好再改回 1e6。

对不起，我理解错了，这只是个检验算法精确度的参数，不是真的执行 1e6 次，我刚刚开的任务 16495 次迭代就算完了。还好我没改。

原始代码-O3 算了 158.924 秒，我向其中加入巨量的 openmp 试试。在我无脑添加 omp 之后，却意外的发现，运行时间反而是 615 秒，非常慢。在略微修改之后，更是出现了残差越来越大的情况。发现是 omp 并行没有给加写入锁，导致点积计算错误。

但是加入 omp 之后仍然非常慢。这下直接 866 秒了。我的 profile 显示 omp 的调度库耗时非常多，所以我减少了 N 循环的 omp，仅仅对 N*N 的矩阵乘法进行了 omp。但是这样的结果还是一百九十多，仍然慢于一点都不改的速度。

我在想，会不会是我始终在用 2001 的输入的问题？我决定扩大输入规模，再测一下。从前几个迭代的报告来看，原始-O3 算了很长时间，几乎无法完成；全部加上 omp 并行的则更为耗时。

我以为是 run.sh 里面每进程核心数设置的有问题，结果更改来还是非常慢。给每个循环加上#pragma omp parallel for 产生了文档里说的反效果: (

对不起，助教，我没设置环境变量。设置了之后速度来到了 58s。

OpenMPI

以下测试默认基于 2001 的输入。

写完了 openmpi，运行测试之后发现逻辑是没问题的，即分割 N，利用 local_N 作为一

个进程处理的数据量，但是提速并不明显，在 18s，我觉得很可能是我的通信太过频繁了。我 profile 一下试试。

profile 显示，我的 cpu 时间大部分浪费在了 barrier。但是这个问题在我重新写了 sbatch 脚本给了环境变量之后改善了，时间也来到了 15s。这时的 profile 上显示，大部分时间在 gemv，其次就是 barrier 导致的空旋。这时再-O3，就可以实现 6.9 秒，这时的 profile 显示时间大部分在 barrier。

Function / Call Stack CPU Time Module Function (Full) Source File Start Address

__kmp_fork_barrier 105.122s libiomp5.so __kmp_fork_barrier(int, int) kmp_barrier.cpp 0x82740

不，不对，我写的 mpi 根本就没有正常运行，始终都是一个节点在自说自话，输出中始终说没有找到 pmi 服务，但是之前我始终忽略了这一点。我修改了一下，但是却在调用 MPI_Scatterv 时发生了死锁。

而且过了一会之后，./data/case_2001.bin 还读取不到了？？？

似乎是 m7 节点的问题。我换成 m6 就正常了。

经过我的 printf 调试，我发现 mpi 卡死的原因是 rank 非零节点在第二次迭代出现了 flag 值异常的问题，导致进程 1 去进行了 gather，进程 0 没有执行，进程 0 继续进行遇到下一个 mpi 函数，最终导致死锁。

我不知道为什么 flag 会突然变成异常值，但是每个 iter 重置一下 flag 也没有解决。我再次定位，确定是 MPI_Bcast 的问题，让 rank1 的 flag 突然变成了一个恒定的 978588160 异常值。但是为什么呢？

找到原因了，是循环末尾的 rho 的广播被我一不小心加了一个 if(!rank)，导致那个数据串到了 flag 上，所以每次 flag 都是稳定的 978588160.

改好了之后时间来到了 4s 多。但是出现了 6 个进程的情况下有时候出现-nan 数据的问题。经过我的仔细检查，发现是可能由于分的太小了和 double 的精度问题，导致 rho_old=0 的情况出现了，作为除数就出现了 nan。但是我感觉这种解释似乎有点牵强，精度能到 e-15 的怎么会出现这种问题？

写完之后根据最新的 profile，mpi 的通讯似乎并没有成为瓶颈，仍然是 omp 的 barrier 问题比较大。我把几个 omp parallel for 并成了一个并行域，获得了不太明显的提升。

SIMD

一波未平，一波又起。我正想看看如果不加 simd 选项会怎么样，结果直接算发散了。原来是我直接把 simd 替换成//simd 导致了把一个归约锁给注释掉了，然后导致的发散。

然后就出现了去掉所有的 simd 速度反而更快的情况。此时 fork barrier 占用 cpu 时间大概 45%，加 simd 会略微多 0.5%。

我决定进行手写 simd 优化。写完了之后 2001 可以达到 2.7s，但是 4001 反而比之前更慢，77s。这是为什么呢？难道是因为寄存器放不下这么多向量吗？

但是又测试了一遍，时间是 37？再测测。

再测一下到 33 了，看来是刚才机魂不悦所导致的。

Run.sh

经过反复尝试，开 3 个节点，每节点两进程 2001 慢一点，但是 4001 和 6001 比较快。

Omp 的线程数根据助教的建议，选择了 8。

我在思考绑核参数，我记得 lab1 里绑核参数是有提升的。我试了 ldoms，变慢了；none 快一点，2.4；core 几乎没有区别，但是在 4001 上就慢了 6 秒；threads 整体会快一些(2.31,21.35)；sockets 和 threads 速度类似（2.4,21.6）

OJ

我很迷惑，为什么本地运行的很好的程序到了 oj 上就已知 exit code 137？看起来是等了半小时还没算完所导致的，而且出现了两个报错，一个是==> Error: [Errno 13] Permission denied: '/.spack'，一个是 sudo: unable to resolve host soj-judgement: Name or service not known，而且 squeue 还提示 user env retrieval failed requeued held。但是编译却成功了，在提交作业的阶段无法恢复用户环境？

好恐怖，昨晚上 oj 给我 ctrl C 了，那个任务挂了一天。对不起，学长。

我把 run.sh 的命令改成了 echo “test”，仍然是这样挂着，那是否可能是我上面环境变量设置的问题呢？

但是这里已经挂了两个 oj 提交的任务了。

晚上再看一下，还是挂着两个 oj，这很难办，我不是很敢尝试，我担心堆了太多的 soj-user 申请的 slurm 导致 oj 任务额度超出限制。

所以我把自己运行的 sbatch 结果截图了，数据放在了 record.out 文件里。

（已修）

我又尝试了一下，我发现什么都不写，原始的 run.sh，只写一个 echo 是能跑的。那我逐渐增加环境变量试试。难道是我多加载一个 intel-oneapi-compilers 的问题？

```
#SBATCH --export=ALL,OMP_NUM_THREADS=8
#SBATCH --export=ALL,I_MPI_PMI_LIBRARY=/slurm/libpmi2.so.0.0.0
```

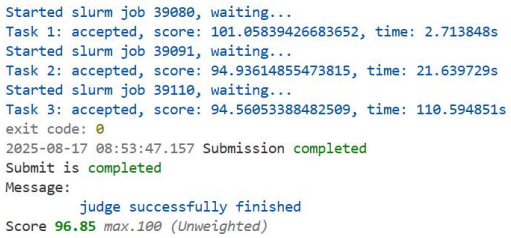
是这两个环境变量的问题，我认真看了看这个 ALL 的真正含义，我似乎被 gpt 诈骗了，这个 ALL 根本就不是为所有节点设置，是设置所有环境变量，此外设置一个 OMP_NUM_THREADS，可能 oj 的环境变量比较多，所以导致了环境变量设置有误，而我自己登陆节点的环境变量比较少，所以自己 sbatch 没有出现什么问题。

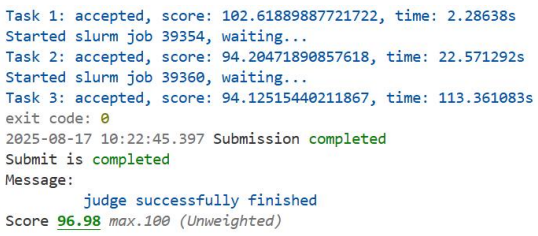
现在 oj 提交 slurm 不会卡死了，但是却又说 failed to run, failed to get the job output，我不知道为什么。群里有同学说是运行速度慢了，超出时间限制，但是我自己 sbatch 只有 2.7 秒，至少 2001 的分应该能拿到。

Oj 似乎并不能编译-O3，只能这么解释，改成-O2 就能跑了。

与 run.sh 自己 sbatch 的情况产生了差异，threads 并非最快，反而什么都不加是最快的，此外还有一个趋势：core 会随着计算规模的增大效果明显，none 则反之。

最终 oj 测试结果如下：





None 最后凭借着更高的 2001 略胜一筹，毕竟运算数据好像偏小。

Fortran

因为 oj 没测成，做个 bonus，希望能弥补一下。

感觉有 C 的基础上，fortran 还是很好学的，感觉还有点像脚本，就是注意 fortran 传递的参数都是地址形式就行了。

竟然要变量都在开头处声明，一下又感觉我在写嵌入式。

二维矩阵刚才没注意到与 C 不同的从 1 开始导致的存储差异，所以错了，修好了就可以了。跑了 57s，可见 fortran 的速度比纯 C 好像更快一点。

```
h3240104875@sct101:~/test/HPC101/src/lab4$ srun -p M6 ./build/bicgstab-fortran ./data/case_2001.bin
 Iteration        1000 , residul =    169.708869806438
 Iteration        2000 , residul =    71.8842353576092
 Iteration        3000 , residul =    68.7048714738074
 Iteration        4000 , residul =    49.3827940677804
 Iteration        5000 , residul =    6.57844731510050
 Iteration        6000 , residul =    7.09295282980920
 Iteration        7000 , residul =    6.21013201379561
 Iteration        8000 , residul =    10.1603569714221
 Iteration        9000 , residul =   0.432354163429559
 Iteration       10000 , residul =   8.329421219005731E-002
 Iteration       11000 , residul =   3.291270843586954E-003
 Iteration       12000 , residul =   2.139109232143721E-005
 Iteration       13000 , residul =   3.237086321009993E-005
 Iteration       14000 , residul =   6.278538521417569E-009
 Iteration       15000 , residul =   6.041261716484613E-011
 ====================== Summary ======================
 Elapsed time: 57.0187 s
 Status: Converged after 15714 iterations.
 Check: Relative residual = 2.572791e-15
 Result: Accepted, great job! :)
 =====================================================
```