第四次作业说明

2022 年 11 月 9 日

1 作业背景

MPI 和 OpenMP 都是并行编程的重要工具，被广泛应用在各种并行软件的开发中。编写并行程序可以有效的利用计算机资源，提高程序的运行效率。上手实操 MPI 和 OpenMP，可以让我们掌握设计并行程序的优化策略。

模板计算是程序中常见的循环运算模式，比如熟知的，高斯塞德尔法迭代法，雅克比迭代法，牛顿迭代法，热传导问题，有限差分问题等，都可以归结为模板计算问题，熟悉模板计算的优化可以让我们更容易的将并行优化的技术应用到实际当中。

2 作业描述

2.1 作业目标

掌握 OpenMP 与 MPI 并行编程方法。
初步体验并行程序的加速效果。
将并行优化的技术应用到实际问题中。
了解并行优化的角度以及方法。

2.2 作业要求

1. 独立完成代码实现与优化。

2. 提交文件夹命名格式为学号 + 作业编号 + 姓名，如第二次作业：2019000000_h2_name，其中包含文件夹 dgemm，stencil，和报告 report.pdf。

3. 注意 DDL，需要在 2022 年 11 月 23 号之前提交

2.3 作业任务

2.3.1 7 点模板计算

实现模板计算的多线程和多进程版本

```
1  Var Matrix a,b,c = grid ;
2  Var Double a7,b7,c7 ;
3  for t ← 1 to nt do
4      for z ← z_start to z_end do
5          for y ← y_start to y_end do
6              for x ← x_start to x_end do
7                  a7 = p01 * a_{t-1}(x,y,z)
                     + p02 * a_{t-1}(x+1,y,z) + p03 * a_{t-1}(x-1,y,z)
                     + p04 * a_{t-1}(x,y+1,z) + p05 * a_{t-1}(x,y-1,z)
                     + p06 * a_{t-1}(x,y,z+1) + p07 * a_{t-1}(x,y,z-1)
8                  b7 = p11 * b_{t-1}(x,y,z)
                     + p12 * b_{t-1}(x+1,y,z) + p13 * b_{t-1}(x-1,y,z)
                     + p14 * b_{t-1}(x,y+1,z) + p15 * b_{t-1}(x,y-1,z)
                     + p16 * b_{t-1}(x,y,z+1) + p17 * b_{t-1}(x,y,z-1)
9                  c7 = p21 * c_{t-1}(x,y,z)
                     + p22 * c_{t-1}(x+1,y,z) + p23 * c_{t-1}(x-1,y,z)
                     + p24 * c_{t-1}(x,y+1,z) + p25 * c_{t-1}(x,y-1,z)
                     + p26 * c_{t-1}(x,y,z+1) + p27 * c_{t-1}(x,y,z-1)

10                 a_t(x,y,z) = a7 + b7*c7/(b7+c7)
11                 b_t(x,y,z) = b7 + a7*c7/(a7+c7)
12                 c_t(x,y,z) = c7 + a7*b7/(a7+b7)
13             end
14         end
15     end
16 end
17 return a_t
```

Algorithm 1: Stencil

解压 homework4.zip 后，有 hello_world_omp, hello_world_mpi 和

stencil 三个文件夹。其中前两个为 OpenMP 和 MPI 版本的 Hello_World 程序，供同学们熟悉最基础的 OpenMP,MPI 的实现及 Makefile 文件的编写。

stencil 文件夹中，为模板计算的框架代码。其中，stencil-naïve 为模板计算的基础串行版本，供同学们熟悉。而 stencil-MPI 和 stencil-omp 为需要自行实现的部分。其中 create_dist_grid 函数一般不用修改，函数中的 halo_size，可以理解为了方便计算，人为在最外层加了一层网格点，以避免边界额外的通信与判断。若有同学使用的优化算法对边界有特殊要求，则可以自行修改 halo_size 的值。

stencil_7 为需要实现的 7 点模板计算。函数的前两个输入参数为两个指针，指向两个三维数组，第一个数组 grid 为输入数据，第二个数组 aux 用于存储中间结果。两个数组最外层额外添加的 halo 层均已初始化为 0。最后的结果写到 grid 或 aux 中，为了区分具体写入了哪个数组，应返回指向该数组的指针。第四个参数，为迭代的步数。

/gpfs/home/bxjs_01/stencil_data/stencil_data 目录下是数据文件，stencil_data_*为输入文件。

编译可以通过提供的 Makefile 来实现

```
make
```

benchmark.sh 为运行脚本，提供可执行文件以及节点数，即可运行，会显示出当前算法的性能。

```
sh benchmark.sh./benchmark – omp 1
```

```
sh benchmark.sh./benchmark – mpi 2
```

其中 test.sh 为正确性验证脚本，只有通过了 test.sh 才能被判断为有效的作业。注意在迭代过程中，运行 test.sh 必然会产生精度误差，只有当误差大于 10^{-12}，脚本输出“Significant numeric error.”才代表程序存在正确性问题。

```
sh test.sh./benchmark – omp 1
```

```
sh test.sh./benchmark – mpi 2
```

2.3.2 多线程矩阵乘法

实现矩阵乘法 $C = C + A * B$, 其中，A，B，C 是 $N * N$ 的双精度稠密矩阵

Input: A: 输入稠密矩阵
B: 输入稠密矩阵

Output: C: $A * B$ 的结果

```
1 receive matrix A B C;
2 for i ← 1 to N do
3 |   for j ← 1 to N do
4 |   |   for k ← 1 to N do
5 |   |   |   C(i,j) ⟵ C(i,j) + A(i,k) * B(k,j);
6 |   |   end
7 |   end
8 end
```

Algorithm 2: DGEMM

本次作业，利用**多线程**来优化第一次作业。

2.4 作业要求

请统一使用双精度浮点数据类型进行 DGEMM 和 Stencil 计算，运算结果由自带的 benchmark 给出。

完成 DGEMM 和 Stencil 的 OpenMP 和 Stencil 的 MPI 并行化和针对性的性能优化。

OpenMP 版本需要测试程序在 1、2、4、8、12、16、24、32 个线程下的并行性能（请留意测试服务器集群单机拥有两个处理器，每个处理器 16 核，NUMA 架构，测试和优化的过程中需要对此进行考虑）。

2.5 作业评分

2.6 多线程矩阵乘法 30%

1. 评测 dgemm 的性能结果，按照提交后的性能排序结果，以及代码质量进行打分 (10%)

2. **详细描述**在实现 dgemm 中采取的优化手段，代码对应的部分，以及对应的实验结果，来解释目前取得的性能结果 (10%)。

3. 给出一张完整的实验结果图，描述当前算法的性能，横坐标为矩阵规模，纵坐标为 $Gflop/s$(10%)。

2.7 模板计算 70%

4. 评测 stencil 的性能结果，按照提交后的性能排序结果，以及代码质量进行打分 (50%)

5. **详细描述**在实现 stencil 中采取的优化手段，代码对应的部分，以及对应的实验结果，来解释目前取得的性能结果 (10%)。

6. 给出一张完整的实验结果图，描述当前算法的性能，横坐标为矩阵规模，纵坐标为 $Gflop/s$(10%)。

2.8 作业提示

1.OpenMP 优化的过程中要考虑变量的属性，以及负载的分配方式

2.MPI 优化的过程中，要考虑通讯的开销

3. 编译器选项也能提升性能

4. 与助教和老师及时交流。

3 快速上手编程

为了方便大家快速上手编程，为大家准备了 2 个 Helloworld 的小 demo 对应编译的 Makefile，然后从网上或者文档的小 demo 入手，了解 MPI 和 OpenMP 的优化角度再来上手作业。

4 参考资料

编程的文档可以参考OpenMP 官方文档，以及StencilProbe。

如果想先了解模板计算可以先看 [2], 算法入门可以参考 [3]，目前关于模板计算在集群上的优化可以参考比较新的 [1], 模板计算经过了漫长的发展，大家也可以在网上很容易的找到很多相关知识。

参考文献

[1] Uday Bondhugula, Vinayaka Bandishti, and Irshad Pananilath. Diamond tiling: Tiling techniques to maximize parallelism for stencil computations. *IEEE Transactions on Parallel and Distributed Systems*, 28(5):1285–1298, 2016.

[2] Anthony Nguyen, Nadathur Satish, Jatin Chhugani, Changkyu Kim, and Pradeep Dubey. 3.5-d blocking optimization for stencil computations on modern cpus and gpus. In *SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–13. IEEE, 2010.

[3] Gabriel Rivera and Chau-Wen Tseng. Tiling optimizations for 3d scientific computations. In *SC'00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing*, pages 32–32. IEEE, 2000.