并行计算基础第二次作业说明

黄浩鹏

2025 年 3 月 13 日

1 作业背景

OpenMP 都是并行编程的重要标准之一，被广泛应用在并行软件的开发中。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。

模板计算是科学计算中常见的循环运算模式，比如熟知的热传导问题，有限差分问题的显式求解等，都可以归结为模板计算问题。模板计算问题有天然的并行性，计算相对规则，访存优化和指令级并行，以及寻找算法级的时间并行性是其主要的优化手段。

2 作业描述

2.1 作业目标

掌握 OpenMP 多线程的并行编程和优化方法。

2.2 作业要求

1. 独立完成代码实现与优化。

2. 可以使用 AI 工具（如 ChatGPT、DeepSeek、GitHub Copilot 等）辅助编程，但必须确保对代码的理解。提交内容应基于自己的理解与实现，禁止直接抄袭他人代码或让 AI 工具生成完整的作业代码。

3. 仅在登录节点上编译程序，并使用 srun 和 sbatch 提交到计算节点运行程序。请勿在登录节点上直接运行程序，否则可能影响系统响应，相关进程会被强制终止。请大家自觉遵守。

4. 注意作业截止时间，以网络学堂发布的为准。

5. 提交单个压缩包，命名格式如：2024000000_h2_name.zip（学号、作业编号、姓名），其中包含代码文件夹 code、编译运行脚本和报告 report.pdf。

2.3 作业任务：CPU 模板计算并行优化

2.3.1 任务描述

使用多线程完成模板计算在单 CPU 和单节点上的实现与优化。

在本次作业中需要实现的模板计算是 27 点模板计算，伪代码如下，其中 p01-p27 为相应元素的计算系数，在作业代码中已有定义，无需改动：

```
1  Var Matrix a = "grid";
2  for t ← 1 to nt do
3      for z ← z_start to z_end do
4          for y ← y_start to y_end do
5              for x ← x_start to x_end do
6                  a_t(x,y,z) = p01 * a_{t-1}(x,y,z)
7                  + p02*a_{t-1}(x-1,y,z) + p03 * a_{t-1}(x+1,y,z)
8                  + p04*a_{t-1}(x,y-1,z) + p05 * a_{t-1}(x,y+1,z)
9                  + p06*a_{t-1}(x,y,z-1) + p07 * a_{t-1}(x,y,z+1)
10                 + p08*a_{t-1}(x-1,y-1,z) + p09 * a_{t-1}(x+1,y-1,z)
11                 + p10*a_{t-1}(x-1,y+1,z) + p11 * a_{t-1}(x+1,y+1,z)
12                 + p12*a_{t-1}(x-1,y,z-1) + p13 * a_{t-1}(x+1,y,z-1)
13                 + p14*a_{t-1}(x-1,y,z+1) + p15 * a_{t-1}(x+1,y,z+1)
14                 + p16*a_{t-1}(x,y-1,z-1) + p17 * a_{t-1}(x,y+1,z-1)
15                 + p18*a_{t-1}(x,y-1,z+1) + p19 * a_{t-1}(x,y+1,z+1)
16                 + p20*a_{t-1}(x-1,y-1,z-1) + p21 * a_{t-1}(x+1,y-1,z-1)
17                 + p22*a_{t-1}(x-1,y+1,z-1) + p23 * a_{t-1}(x+1,y+1,z-1)
18                 + p24*a_{t-1}(x-1,y-1,z+1) + p25 * a_{t-1}(x+1,y-1,z+1)
19                 + p26*a_{t-1}(x-1,y+1,z+1) + p27 * a_{t-1}(x+1,y+1,z+1)
20             end
21         end
22     end
23 end
```

Algorithm 1: stencil27

2.3.2 正确性验证

采用 **double 双精度浮点数据类型**进行运算，运算结果通过作业基础代码中的正确性验证。

2.3.3 运行方法

使用 OpenMP 多线程进行优化，**并行度用满 (1) 单 CPU 和 (2) 单节点**。

样例代码可以从网络学堂上获取，其中附有 example.txt 日志可供参考。课程集群的登录和程序运行请参考网络学堂文件《课程集群使用手册》。code 文件夹中，stencil-naive 是基础的实现，供同学们参考，而 stencil-optimized 是需要自行实现的部分。

create_dist_grid 函数主要负责多进程程序中的进程网格划分，对于本次作业要求的纯 OpenMP 多线程程序，一般无需改动。若有同学使用的优化算法对边界有特殊要求，可自行修改。

stencil_27 函数为需要实现的模板计算。函数的前两个输入参数为两个指针，指向两个数组，其中第一个数组 grid 在输入时存储的是初值，即初始输入的数据；aux 是用于存储迭代过程中间值的另一个数组，在迭代过程中，grid 和 aux 这两个数组会作为滚动数组循环使用。两个数组外层均有额外添加 halo，且 halo 中的数据初始值为 0（未填充邻居进程的边界数据）。依据迭代步数，最终的计算结果会存在其中一个数组中，请返回对应数组的指针。第三个参数是模板计算的网格维度以及 halo 区的信息。第四个参数是迭代的步数。

编译可以通过提供的 Makefile 来实现（基础版本使用了 GCC 和 OpenMPI 编译器套件，对应软件包为 gcc@10.4.0 和 openmpi，若需 Intel 套件，请自行更改 Makefile 文件）。

```
make benchmark-optimized
```

benchmark.sh 是性能测试脚本，有三个命令行参数，分别是可执行文件、总进程数和每进程线程数。

test.sh 是正确性验证脚本，只有通过了 test.sh 才会被判断为有效的作业。test.sh 的命令行参数以及其含义和 benchmark.sh 相同。

2.4 作业评分

2.4.1 模板计算优化 100%

1. 评测单 CPU（24 核心）stencil 的性能结果，按照多个固定计算规模的 stencil_27 程序平均性能进行打分 (40%)。

2. 评测单节点（48 核心）stencil 的性能结果，按照多个固定计算规模的 stencil_27 程序平均性能进行打分 (20%)。

3. **详细描述**采取的优化手段，代码对应的部分，以及对应的实验方案 (例如测试次数，测试性能取值方式等) 与结果，可以采用性能工具或者模型来解释目前取得的性能结果 (30%)。

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

2.5 作业提示

1.OpenMP 优化的过程中要考虑变量的共享或私有属性, 以及 **NUMA 效应带来的影响**。

2. NUMA 内存模型常出现于现代高性能计算机中,不同的 CPU 对各个 DRAM/HBM meomory node 有不同的亲合度，使得跨 NUMA 访存有显著的性能下降。lscpu 命令可以查看 CPU 核心与哪个 NUMA memory node 进行了绑定。同学们可以根据线程所在的 NUMA node,使用numa_alloc_onnode 函数管理不同 NUMA 上的内存分配和使用。

3. 有任何问题欢迎与助教和老师交流。

3 参考资料

基础的多线程并行编程参考资料：OpenMP-examples。

stencil 计算可以参考stencilProbe，里面介绍了一些模板计算的基本优化手段。除此之外，还有一些文献介绍了这方面上的工作，[1], [2], [3], [4], [5], [6], [7], [8]。大家可以选择性的阅读，也可以在网上自行查找其他的相关工作。

参考文献

[1] David Wonnacott. Time skewing: A value-based approach to optimizing for memory locality. *Submitted for publication*, 1999.

[2] 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.

[3] R Fowler, G Jin, and J Mellor-Crummey. Increasing temporal locality with skewing and recursive blocking. *Proceedings of SC01: High-Performance Computing and Networking.(November 2001)*, 2001.

[4] Lakshminarayanan Renganarayana, Manjukumar Harthikote-Matha, Rinku Dewri, and Sanjay Rajopadhye. Towards optimal multi-level tiling for stencil computations. In *2007 IEEE International Parallel and Distributed Processing Symposium*, pages 1–10. IEEE, 2007.

[5] Werner Augustin, Vincent Heuveline, and Jan-Philipp Weiss. Optimized stencil computation using in-place calculation on modern multicore systems. In *European Conference on Parallel Processing*, pages 772–784. Springer, 2009.

[6] 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.

[7] Vinayaka Bandishti, Irshad Pananilath, and Uday Bondhugula. Tiling stencil computations to maximize parallelism. In *SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, pages 1–11. IEEE, 2012.

[8] 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.