第一次作业说明

姚杰男

2023 年 10 月 24 日

1 作业背景

矩阵乘法是 HPC 领域最为基础的计算核心之一，广泛应用于数值计算、机器学习等领域。本次作业的内容是单核矩阵乘法。可以选用的优化包括但不限于：内存对齐 Alignment，数据预取 Prefetch，循环变化 Loop refactorization，分块 Cache blocking，向量化 Vectorization(intrinsic 指令)，循环展开 Loop unrolling。熟悉这些方法的原理，评估其效果，可以让我们更好地掌握程序优化方法。

2 作业描述

2.1 作业目标

掌握主要的体系结构优化手段。

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，如第一次作业：2023000000_h1_name，其中包含文件夹 sgemm，和报告 report.pdf。

3. 推荐仅在登陆节点上编译程序，禁止在登录节点上运行程序，可采用 srun 或 sbatch 来提交任务，希望大家自觉遵守。

4. 注意 DDL，作业提交的截止时间参见网络学堂。

5. 不能将 BLAS 或 BLIS 等数学库的实现作为作业成果。

2.3 作业任务：单核矩阵乘法

2.3.1 任务描述

矩阵乘法 $C = C + A * B$, 其中，A，B，C 是 $N * N$ 的单精度稠密矩阵。完成单核矩阵乘法的串行优化。

Input: A: 输入稠密矩阵
B: 输入稠密矩阵

Output: C: A * B 的结果

```
receive matrix A B C;
for i ← 1 to N do
    for j ← 1 to N do
        for k ← 1 to N do
            C(i,j) ⟵ C(i,j) + A(i,k) * B(k,j);
        end
    end
end
```

Algorithm 1: GEMM

2.3.2 正确性检验

1. 采用单精度运算，运算结果通过以下的正确性验证，eps 为机器的单精度极小值，约为 10^{-6} 左右：

$$\|square\_gemm(n, A, B, 0) - A * B\| < 3 * n * eps * \|A\| * \|B\|$$

2. 矩阵乘法的复杂度为 $O(N^3)$，在计算性能指标的时候采用 $(2N^3)$ 计算，如果采用了一些非 $O(N^3)$ 算法而导致通过不了正确性测试，这种情况可以适当且合理地放宽精度的要求，但是需要在作业报告中指出。

3. 开展必要的性能分析，比如某些矩阵规模性能出现明显的降低，可以采用性能分析的工具进行性能分析。

2.3.3 运行方法

本次作业，需要在课程集群中运行，作业的基础版本代码可以从网络学堂上获取。解压之后，在 sgemm 子目录下包含 sgemm-naive.c，sgemm-

blas.c，sgemm-blocked.c 三个样例，naive 作为最简单的实现，blas 作为本次作业性能的参考上限，blocked 作为本次作业要改进和优化的基础代码。集群初次登录使用和所需软件（如 intel-oneapi-compilers 和 mkl）动态加载的设置步骤请参考`https://parallel-comp.thu.fail/intro/`.

编译 benchmark-blocked，可以通过提供的 Makefile 执行：

```
make benchmark-test
```

测试 benchmark-blocked 的性能，可以通过执行：

```
srun -n 1 ./benchmark-test
```

排他执行可以采用下面的命令来保证节点上只有自己的程序在运行，避免多个程序在相同节点运行带来的性能干扰。但由于会降低集群的同时并发作业数量，建议仅在测试最后性能结果时使用：

```
srun -n 1 --exclusive --job-name = benchmark-
  test ./benchmark-test
```

2.4 作业评分

2.4.1 sgemm 100%

1. 评测 sgemm 在不同输入下的性能结果，按照提交后的性能排序结果，以及代码质量进行打分 (60%)

2. **详细描述**在实现 sgemm 中采取的优化手段，代码对应的部分，以及对应的实验结果，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%)。

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%)。

2.4.2 额外加分 20%

实现其他的 gemm 算法，比如Strassen 算法，实现之后请同时提供不同规模下的计算时间与标准算法的对比，正确性测试上需要通过 benchmark.c 中提供的检验。

2.5 作业提示

1.input sensitivity 问题，算法性能的表现和矩阵规模的大小具有很强的相关性，分析 input sensitivity 并设计更为合理的优化策略将能更有效的提升性能。

2. 可以先利用编译选项来辅助自己完成一部分优化从而减少代码量（能提升 10% 到 20% 的性能），再进行代码级别的优化。

3. 提升局部性是第一次作业的核心，需要考虑矩阵应该如何分块才能有效地利用 cache(考虑 cache 的空间局部性)。向量化 (intrinsic 指令) 也是一个主要的优化手段。

4. 优化中会存在一些超参数，例如各个维度上的分块大小，可以搜索最优参数，以进一步提升性能。

5. 可以利用 perf，gptl 以及 intel 性能分析工具，帮助理解性能问题、性能优化的过程和分析原因。

6. 可以添加其他的.c 或者.h 文件辅助编程，默认使用 Makefile 生成的对应可执行文件 benchmark-test 作为最终结果，若有修改，请在报告中详细说明修改之后的代码结构和运行方法，有以免出现没有成绩的情况。

7. 请确保作业按时完成，晚交会对成绩有一定影响，但不交一定 0 分。

8. 有问题与助教和老师及时交流。

3 参考资料

BLAS 是一个高性能矩阵运算库,可以参考 GotoBLAS[1], 或者 Franchetti 的 [2]，其中介绍了 gemm(即本次作业) 的一些思考角度，Rothberg 在 [3] 中详细介绍了分块的策略。BLIS 提供了一个通过简单的 kernel 表达和优化复杂 BLAS 操作（BLAS2 和 BLAS3）的框架，能有效提高使用的效率，具体内容可参考 [4].

关于向量化推荐大家先看看上课件上的向量化指令的样例，更详细的可以参考Intel 编译器 Intrinsic 的官方文档。

Slurm 的使用只需掌握本文中的命令即可完成作业，当然如果想了解更多可以参考Slurm 的官方文档。

本学期实验的 CPU 硬件平台是基于 x86 架构的 Intel 处理器，关于 Intel 的一系列工具可以参考官方文档1、2、3等等。

参考文献

[1] Kazushige Goto and Robert A van de Geijn. Anatomy of high-performance matrix multiplication. *ACM Transactions on Mathematical Software (TOMS)*, 34(3):1–25, 2008.

[2] Srinivas Chellappa, Franz Franchetti, and Markus Püschel. How to write fast numerical code: A small introduction. In *International Summer School on Generative and Transformational Techniques in Software Engineering*, pages 196–259. Springer, 2007.

[3] Monica D Lam, Edward E Rothberg, and Michael E Wolf. The cache performance and optimizations of blocked algorithms. *ACM SIGOPS Operating Systems Review*, 25(Special Issue):63–74, 1991.

[4] Field G. Van Zee and Robert A. van de Geijn. Blis: A framework for rapidly instantiating blas functionality. *ACM Trans. Math. Softw.*, 41(3), jun 2015.