计算机科学
COMPUTER SCIENCE

基于多面体模型的矩阵乘法自动混合精度优化

何昊天, 周蓓, 郭绍忠, 张作言, 郝江伟, 许瑾晨

引用本文

何昊天, 周蓓, 郭绍忠, 张作言, 郝江伟, 许瑾晨. 基于多面体模型的矩阵乘法自动混合精度优化[J]. 计算机科学, 2024, 51(12): 110-119.

HE Haotian, ZHOU Bei, GUO Shaozhong, ZHANG Zuoyan, HAO Jiangwei, XU Jinchen. Optimisation of Automatic Matrix Multiplication Mixing Accuracy Based on Polyhedral Models [J]. Computer Science, 2024, 51(12): 110-119.

相似文章推荐（请使用火狐或 IE 浏览器查看文章）

Similar articles recommended (Please use Firefox or IE to view the article)

基于数据局部性的循环分块选择算法

Tile Selection Algorithm Based on Data Locality

计算机科学, 2024, 51(12): 100-109. https://doi.org/10.11896/jsjkx.231100060

高阶密码算子在FPGA的编译优化与实现

Compilation Optimization and Implementation of High-order Cryptographic Operators on FPGA

计算机科学, 2024, 51(11A): 231200184-11. https://doi.org/10.11896/jsjkx.231200184

面向矩阵乘计算的自动混合精度优化

Automatic Mixing Precision Optimization for Matrix Multiplication Calculation

计算机科学, 2024, 51(11A): 240300057-10. https://doi.org/10.11896/jsjkx.240300057

基于混合精度的分布式GMRES算法优化

Optimizing Distributed GMRES Algorithm with Mixed Precision

计算机科学, 2024, 51(9): 15-22. https://doi.org/10.11896/jsjkx.231000204

矩阵乘法的GPU并行计算时耗模型与最优配置方法

Time Cost Model and Optimal Configuration Method for GPU Parallel Computation of Matrix Multiplication

计算机科学, 2024, 51(6A): 230300200-8. https://doi.org/10.11896/jsjkx.230300200

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230800106

基于多面体模型的矩阵乘法自动混合精度优化

何昊天　周　蓓　郭绍忠　张作言　郝江伟　许瑾晨
信息工程大学网络空间安全学院　郑州 450001
(m18503880251@163.com)

摘　要　混合精度是计算机中的一种数值计算技术,通过将计算中的部分数据类型从高精度转换成低精度来提高计算效率。矩阵乘法在计算机科学和数学中有着重要而广泛的应用,在矩阵乘法中使用混合精度技术来加速计算过程是一项很有挑战性的工作。现有的混合精度优化存在一些问题,例如存储开销大,必须在特定的硬件单元上实现,限制了模型或算法的部署选项并降低了其可移植性。针对上述问题,提出并实现了基于多面体模型的混合精度代码自动生成工具 AGMMMPC。通过将低精度乘高精度加基础混合精度矩阵乘代码生成功能添加到“源-源”的 PPCG 编译器中,并使用精度调优算法(Precision Tuning,PT)找到基础混合精度计算中的高频误差点,将这些点用高精度计算,其余点用基础混合精度计算,有效减小基础混合精度计算中的误差,首次实现了源到源的面向矩阵乘计算的混合精度代码自动生成。实验表明,以高精度计算为基准,AGMMMPC 生成的高级混合精度代码在 X86 平台上的最大加速比为1.39,几何平均加速比为 1.14。

关键词:混合精度;矩阵乘法;多面体模型;调度变换;代码生成
中图分类号　TP314

Optimisation of Automatic Matrix Multiplication Mixing Accuracy Based on Polyhedral Models

HE Haotian,ZHOU Bei,GUO Shaozhong,ZHANG Zuoyan,HAO Jiangwei and XU Jinchen
School of Cyberspace Security,University of Information Engineering,Zhengzhou 450001,China

Abstract　Mixed precision is a numerical computation technique in computers that improves the efficiency of computation by converting some of the data types in the computation from high precision to low precision. Matrix multiplication has an important and wide application in computer science and mathematics,using mixed precision techniques in matrix multiplication to speed up the computational process is a challenging task. Existing mixed-precision optimisation suffers from several problems,such as high storage overhead,having to be implemented on specific hardware units,limiting the deployment options of models or algorithms and reducing their portability. In the face of the above problems,this paper proposes and implements an automatic mixed-precision code generation tool based on polyhedral models AGMMMPC. By adding the low-precision-by-high-precision-plus-basic-mixed-precision matrix multiplication code generation functionality to the source-source PPCG compiler,and using the precision tuning (PT) algorithm to find high-frequency errors in the basic mixed-precision computation. These points are processed by high precision calculation method,while the rest are processed by the basic mixed precision calculation method,which effectively reduces the error in the basic mixed precision calculation,and realises the automatic generation of source-to-source mixed-precision code for matrix multiplication for the first time. Experiments show that the maximum acceleration ratio of the advanced mixed-precision code generated by AGMMMPC is 1.39 and the geometric mean acceleration ratio is 1.14 on X86 platform with high-precision computation as the benchmark.

Keywords　Mixed precision,Matrix multiplication,Polyhedral model,Scheduling transformation,Code Generation

1　引言

提高基本计算效率可能会产生广泛的影响,因为它直接影响到大量计算任务的整体速度[1]。矩阵乘法是一种常见的基本计算任务,是许多计算任务的核心,例如矩阵求逆、计算行列式和求解线性系统。矩阵乘法被广泛应用于图像处理、机器学习、神经网络[2]和科学计算等领域,因此,如何更好地提升矩阵乘法的执行效率变得尤为重要。

混合精度技术在矩阵乘法优化方面取得了许多进展。目前,混合精度优化主要有两种混合方式,一种是计算精度与存储精度分离,即使用一种精度进行计算,另一种精度进行存储。例如,Huang 等[3]提出的混合精度方法在混合精度训练时,矩阵乘运算使用 FP16 进行计算,但最终结果会被累积并存储在 FP32 格式中,从而获得了 1.5～3 倍的加速效果。然而,这种混合方式需要将计算阶段得到的结果转换为高精度浮点数并存储起来,此过程需要占用额外的存储空间,会导致

到稿日期:2023-08-16　返修日期:2024-01-15
通信作者:许瑾晨(atao728208@126.com)

存储开销过大;并且由于存储精度不同,在计算和存储阶段之间进行数据转换可能会增加内存访问延迟,并降低计算效率。另一种混合方式是计算阶段精度划分,即同时在计算阶段使用不同精度。例如,NVIDIA 推出的硬件加速器 Tensor Cores 实现了混合精度计算,在矩阵乘法过程中,使用半精度浮点数(FP16)进行乘法操作,而使用单精度浮点数(FP32)进行加法操作。这种混合精度计算方式能够显著加快矩阵乘法的计算速度,相比传统的单精度矩阵乘法能提高 8 倍以上的计算速度,目前这种混合方式只在硬件上实现了。但是用硬件实现低精度乘高精度加混合精度计算,需要在特定硬件单元上实现。由于硬件实现更容易受到舍入误差和数值不稳定性引起的噪声干扰,没有这种特定硬件支持的系统则无法使用。

为解决现有问题,本文在软件层面上实现了低精度乘高精度加混合精度方法对矩阵乘计算进行优化。然而,在软件层面实现低精度乘高精度加混合精度优化复杂度较高。一方面需要分析数据结构并设计相应的算法,进行数据类型转换和存储管理,这可能会给开发者带来更大的工作量和技术挑战。因此,本文基于多面体模型,利用 PPCG[4] 对矩阵乘计算中的数据流进行分析,通过调度树转换进行数据类型的转换,将低精度乘高精度加混合精度矩阵乘代码生成功能添加到“源-源”的 PPCG 编译器中,从而自动生成性能更加高效的基础混合精度矩阵乘代码。

另一方面,与高精度矩阵乘计算相比,低精度乘高精度加混合精度矩阵乘计算势必会引入误差。为了平衡性能和精度,需要统计混合精度计算中的误差分布,并使用高精度计算来处理误差较大的点,以降低误差。然而,不同的输入会引入不同的误差,并且高于误差阈值的点也各不相同,难以确定混合精度计算中误差较大的点。如果每组输入都要找到误差较大的点,这将变得繁琐并且费时费力。因此,本文提出精度调优算法(PT),在基础混合精度矩阵乘计算中找到出现频率较高且误差较大的点。为了降低误差,采用高频且误差大的点用高精度计算,其余点使用混合精度计算的策略,并根据此策略自动生成高级混合精度代码。最后对生成的代码进行测试。实验结果表明,以高精度计算为基准,生成的高级混合精度代码在 X86 平台上的最大加速比为 1.39,几何平均加速比为 1.14。

本文的主要贡献如下:

1)结合多面体模型和混合精度技术,自动生成低精度乘高精度加基础混合精度矩阵乘代码。

2)提出精度调优算法(PT)对基础混合精度代码进行精度分析,结合性能收益设计并实现了一个面向矩阵乘计算的自动混合精度工具 AGMMMPC(Automatically Generated Matrix Multiplication Mixed Precision Code)。

3)选取不同规模下的矩阵乘法测试用例进行实验,结果表明,本文所实现的面向矩阵乘计算的自动混合精度工具能够自动生成混合精度优化后的代码,并且可以充分挖掘矩阵乘计算中潜在的精度冗余和性能。

本文第 2 章介绍了研究现状以及多面体模型和调度树的相关基础知识;第 3 章介绍自动混合精度工具 AGMMMPC 的框架及实现;第 4 章对生成的混合精度代码进行测试和分析;最后总结全文并展望未来。

2 相关工作

2.1 矩阵乘优化

近年来,矩阵乘优化成为了计算机科学和高性能计算领域的研究热点。矩阵乘法是一种常见的数值计算操作,被广泛应用于图像处理、机器学习、科学计算等领域。然而,由于矩阵乘法的计算复杂度较高,因此,如何提高其执行效率成为了学术界和工业界的关注焦点。为了提高矩阵乘法的执行速度,研究人员探索了多种方法和技术。2022 年,Fawzi 等[1]提出了一种深度强化学习方法,用于发现有效且可证明正确的算法,以解决任意矩阵乘法的问题;Dughmi 等利用随机化技术对矩阵乘法进行优化[5],提出了一种算法,该算法利用随机采样来近似矩阵乘法,在保证结果精度的前提下,显著降低了计算复杂度。还有一些方法使用基本优化技术,包括循环展开、数据重用、寄存器变量和数据对齐。这些技术旨在降低计算量和内存访问延迟,提高指令级并行性和局部性。此外,还有一些缓存友好的算法被提出,例如 Demmel[6]提出了基于分块技术对矩阵乘法进行优化,将大规模矩阵乘法分解成较小的分块矩阵乘法,以利用计算和存储资源的局部性,提高并行性能。编译器优化是另一个重要的方向,自动向量化[7]和循环重排等技术可以由编译器自动完成,其将代码转换为更高效的形式,从而提高执行效率。

2.2 混合精度

混合精度是一种在计算过程中同时使用不同精度数据类型的技术,主要应用于存在精度冗余的程序中,可以同时使用高精度计算和低精度计算。目前,混合精度技术的研究分为两类:一类是算子级混合精度,主要用于人工智能领域的深度学习训练;另一类是变量级混合精度,主要用于各种精度调整和分析工具。分析精度调整后的混合精度程序是否满足要求有动态和静态分析两种方法,使用静态分析的混合精度工具包括 AMPT-GA[8],FPTuner[9],Daisy[10],Rosa[11]、TAFFO[12]等,使用动态分析的混合精度工具包括 Precimonious[13], Blame Analysis[14], ADAPT[15], fpPrecisionTuning[16],GPUMixer[17]等。无论是算子级混合精度还是变量级混合精度,它们的目标都是在保持适当计算精度的同时,提高计算效率和降低资源消耗。混合精度技术的研究和应用对于深度学习训练、HPC 计算和其他相关领域的性能优化具有重要意义。通过合理选择和应用混合精度技术,可以取得更好的计算效果,并提高解决各种复杂问题的效率。

2.3 多面体模型

多面体模型是(PolyhedralModel)[18]将语句实例抽象成空间多面体,并通过这些多面体上的集合操作来分析和指导循环变换的编译优化模型。多面体模型通常使用迭代空间(语句实例集合)、访存关系(语句实例与数据访问的映射关系)、依赖关系(语句实例之间的依赖关系)和调度(语句执行顺序)表示程序及其语义。

多面体编译工具的一般编译流程如图 1 所示,其一般由抽象分析、调度变换和代码生成 3 个阶段组成,且线性整数

规划始终贯穿其中[19]。抽象分析过程用于提取程序的多面体模型,并进行依赖分析。多面体模型是对程序结构和计算依赖关系的抽象表示,根据依赖分析的结果,调度变换被应用于实现合法的循环变换组合。通过调度变换,程序的循环结构可以被重新排列和优化,以提高并行性和性能。代码生成阶段,将基于多面体模型的中间表示根据目标体系结构的编程模型生成相应的并行程序。这意味着生成的代码可以与目标体系结构的特点和要求相匹配,以充分发挥并行计算的优势。

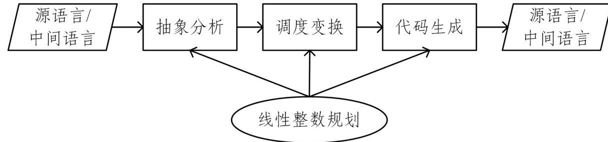


图 1　多面体模型一般编译流程

Fig. 1　General compilation flow of polyhedral models

以图 2 的矩阵乘计算为例,多面体模型将该计算抽象成三维空间上的多面体形式。其调度可以被表示为:

$$\{S_0(i,j)\rightarrow(0,i,j);S_1(i,j,k)\rightarrow(1,i,j,k)\} \tag{1}$$

其中,$S_0(i,j)$和$S_1(i,j,k)$表示语句的实例,分别对应图 2 中 S_0初始化语句和 S_1矩阵乘计算语句;$(0,i,j)$和$(1,i,j,k)$表示语句的执行顺序;“$\rightarrow$”表示从语句实例到语句执行顺序的映射关系。以 S_0语句为例,它有 i 和 j 两个维度,因此本文用三维数组(i,j,k)来表示所有 S_0语句实例的执行顺序。其中,$(i,j,0)$表示在 i 轴上执行完后,再按照 j 轴的顺序执行;而$(i,j,1)$则表示在 j 轴上执行完成后再执行。需要注意的是,数组中的 0 和 1 并不是实际的时间单位,而是用来区分执行顺序的常量。在数组的某一维度上,任何一个被标记为 1 的语句实例都必须在标记为 0 的语句实例之后执行。根据计算出的循环嵌套依赖关系,多面体模型能够计算出一个新的调度,其结果为:

$$\{S_0(i,j)\rightarrow(i,j,0);S_1(i,j,k)\rightarrow C(i,j,1,k)\} \tag{2}$$

```
#pragma scop
for(i=0;i<M;i++)
        for(j=0;j<N;j++)
S0          C[i][j]=0;
for(i=0;i<M;i++)
      for(j=0;j<N;j++)
            for(k=0;k<K;k++)
S1                C[i][j]=C[i][j]+A[i][k]*B[k][j];
#pragma endscop
```

图 2　矩阵乘计算 C 语言代码

Fig. 2　C language code for matrix multiplication calculation

从式(1)到式(2)的转换是应用多面体模型对矩阵乘法计算实施循环合并优化得到的。在优化后的代码中,循环合并的过程被表示为多面体模型,可以看作是一种更高级别的抽象。这个优化的过程使得程序的执行效率得到了提升,代码简化程度也得到了增强。图 3 展示了优化后的代码。

在程序自动并行化领域,多面体模型已经取得了许多引人注目的成果。此外,它还被广泛应用于 Pluto 编译工具[20]、PPCG 编译工具、GCC 中的 Graphite 框架[21] 和 LLVM 中的 Polly[22] 模块等开源工具和商业应用中。因此本文基于多面体模型开展面向矩阵乘计算的自动混合精度优化研究。

```
for(i=0;i<M;i++)
    for(j=0;j<N;j++){
        C[i][j]=0;
        for(k=0;k<K;k++)
           C[i][j]=C[i][j]+A[i][k]*B[k][j];
    }
```

图 3　循环合并后矩阵乘计算 C 语言代码

Fig. 3　C language code for matrix multiplication after loop merge

2.4　调度树

调度树(Schedule Tree)是一种用于表示程序循环嵌套结构的数据结构,主要用于在并行计算中对程序进行并行和优化。由于多面体模型用于表示程序中语句实例的执行顺序,因此调度在本质上可以被看作是树的形式。在多面体模型中,除了常见的集合和映射之外,还存在多种其他中间表示形式(Intermediate Representation, IR)用于表示调度过程,如 Kelly 等[23]提出的中间表示、“$2d+1$”形式的中间表示[24]、Presburger 算术表达式[25]以及调度树等。本文采用调度树作为多面体模型的中间表示形式。

调度树通常由多个结点组成,每个结点代表一个循环结构,包括循环体、循环范围、迭代次数等信息。一般而言,调度树中可以包含 domain, sequence, filter, band, mark 和 extension 等类型的结点。其中 domain 结点是调度树的根结点; sequence 结点的子结点必须按先后顺序执行; filter 结点一般是 sequence 等结点的子结点,表示当前子树中所有语句实例的集合; band 结点与循环嵌套相对应; mark 结点表示当前子树中的附加信息; extension 结点是一个可由程序员操作的扩展结点。有关调度树的结点信息以及由其生成 AST 的算法和实现原理可以参考文献[26]。

基于调度树,本文可以很方便地对矩阵乘循环程序进行各种优化。调度树的形状类似于树形结构,易于理解和修改,程序开发人员可以方便地进行修改和插入操作以实现特定的功能。此外,多面体编译器如 PPCG 可以根据调度树生成其对应的抽象语法树(AST),由此自动生成针对特定体系结构的并行代码,程序员只需确保调度树等信息符合规范即可。因此,本文采用调度树作为多面体模型中调度的中间表示形式,以实现针对矩阵乘计算的自动混合精度优化。

3　AGMMMPC 的框架及实现

本文面向矩阵乘计算设计实现了一个混合精度代码自动生成工具 AGMMMPC。流程图如图 4 所示,以高精度矩阵乘源程序作为输入,利用 PPCG 对输入的矩阵乘程序进行初始化和基础混合,生成低精度乘高精度加混合精度代码,采用精度调整算法(PT)循环多次随机初始化输入矩阵的值,用高精度矩阵乘计算的结果矩阵和混合精度矩阵乘计算的结果矩阵的差值得到多组误差矩阵(如图中灰色矩阵所示),设置误差阈值,其中,深灰色为误差矩阵中高于误差阈值的点。初始化和结果矩阵同等规模且元素值全为 0 的 record 矩阵,用来统计误差分布,根据误差分布记录高频误差点,将这些点的

行列号保存在 json 文件中。为了减小混合精度矩阵乘计算的误差，采用高级混合精度策略，即在混合精度矩阵乘计算时将 json 文件中的高频误差点用高精度计算，其余点用混合精度计算。此外，将高级混合精度策略自动生成代码输出。其中，利用精度调整算法找到高频误差点，确定高级混合精度策略并自动生成代码是本文工作的核心。

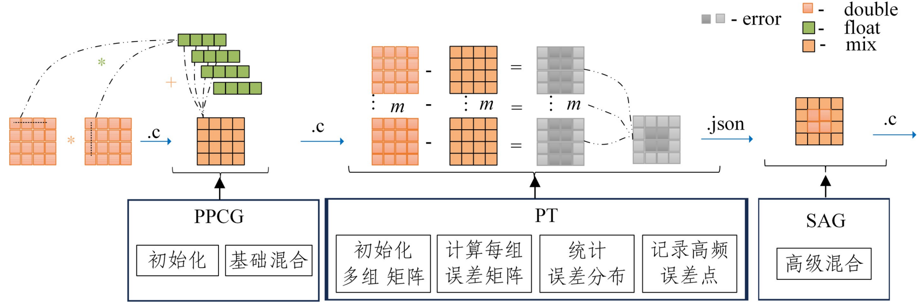


图 4　AGMMMPC 框架示意图

Fig. 4　Schematic diagram of AGMMMPC framework

本章以常见的矩阵乘计算为例，详细介绍了 AGMMMPC 的基本工作流程。AGMMMPC 主要分为初始化、基础混合、精度调优和高级混合 4 个模块。

3.1　初始化

初始化子模块包含抽象分析和调度变换两个阶段。在抽象分析阶段，预处理模块使用 PET[27] 库，从用户输入的 C 语言矩阵乘源代码中识别出指定的程序段，然后构建该程序段的多面体模型，其中包括迭代空间、访存关系以及调度，均以仿射约束的形式表示。

迭代空间指用来表示输入程序中语句实例集合的空间。在这个空间中，每个语句实例都可以通过迭代变量的值来唯一确定。示例矩阵乘计算的核心代码如图 3 所示，其迭代空间如式(3)所示，其中每一个迭代向量(i,j,k)都与语句 S_0 和 S_1 的一次执行实例一一对应。

$$\{S_0(i,j):0\leqslant i<M\wedge 0\leqslant j<N;S_1(i,j,k):\\ 0\leqslant i<M\wedge 0\leqslant j<N\wedge 0\leqslant k<K\} \tag{3}$$

访存关系是语句实例与其访问的数据元素之间的映射关系，包括了写访存关系与读访存关系。示例矩阵乘计算在满足式(1)中迭代空间约束的前提下，其写访存关系为：

$$\{S_0(i,j)\to C(i,j):(0\leqslant i<M\wedge 0\leqslant j<N);S_1(i,j,k)\to\\ C(i,j):(0\leqslant i<M\wedge 0\leqslant j<N)\} \tag{4}$$

读访存关系为：

$$\{S_1(i,j,k)\to A(i,k):(0\leqslant i<M\wedge 0\leqslant k<K);S_1(i,j,k)\to\\ B(k,j):(0\leqslant k<K\wedge 0\leqslant j<N);S_1(i,j,k)\to C(i,j):\\ (0\leqslant i<M\wedge 0\leqslant j<N)\} \tag{5}$$

调度被用来表示程序中语句实例的执行顺序，并将每个语句实例映射到一个整数元组上。这些整数元组按照字典序的方式进行排序，以确定程序中语句实例的执行顺序。例如，在矩阵乘法计算中，原始的调度如式(1)所示。

依赖关系是判断程序是否正确执行的关键和保证程序变换合法性的前提。无论是针对循环或其他重排序变换，都必须先满足依赖基本定理，才能保证程序按照预期正确执行。因此，在使用调度变换来发掘程序并行性之前，必须对程序的依赖关系进行仔细分析处理，以确保程序的正确性和可靠性。

以矩阵乘计算为例，根据计算出的访存关系，本文获得了其语句实例间的依赖关系：

$$\{S_0(i,j)\to S_1(i,j,0);S_1(i,j,k)\to S_1(i,j,k+1)\} \tag{6}$$

调度变换(Schedule Transformation)指对程序的调度进行变换，以达到提高程序并行性和减少执行时间的目的。调度变换是多面体模型中的核心。为了能让程序有更好的并行性和局部性，AGMMMPC 在依赖分析的基础上，采用 isl 库中改进的 isl 调度算法[28]来对循环嵌套进行变换。调度变换前后的调度示意图如图 5 所示。

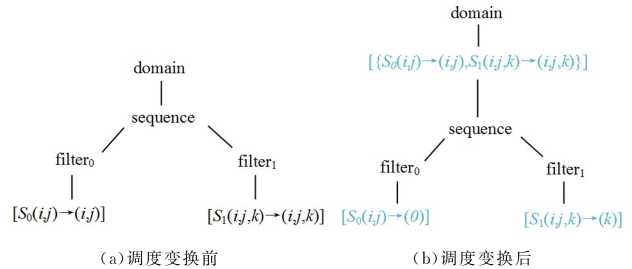


图 5　调度变换前后的调度树示意图

Fig. 5　Schematic of scheduling tree before and after scheduling transformation

3.2　基础混合

基础混合子模块主要对经过预处理子模块得到的调度树进行数据流分析、调度树转换、代码生成等操作。最终，基础混合模块会生成一个基础混合精度矩阵乘代码。

3.2.1　*数据流分析*

矩阵乘法是由乘法操作和加法操作组成。在计算矩阵乘法时，乘法操作的数量通常远远多于加法操作的数量。因此，在优化矩阵乘法性能时，可以通过减少乘法操作的性能开销来提升矩阵乘法的计算速度。众所周知，计算机中，低精度乘法的计算速度比高精度乘法的计算速度快，低精度乘法可以在同一时间内处理更多的数值，从而提高计算性能。加法操作涉及相对较小的数值计算，并且它们更容易受到累积误差的影响。使用高精度加法可以减小数值计算过程中的舍入误差。为了平衡性能和精度，对矩阵乘法的乘法操作使用低精度，而对加法操作使用高精度，可以在保持计算性能的同时获得较高的数值精度。

该模型首先通过对程序中的数组引用和语句实例进行分析；然后根据访存映射、数组引用等情况对它们进行分组。

分组策略是首先进行初始化分组，即将一个数组的引用以及与之相关的信息初始化为一个组。但是，如果两个组的访问关系都只是读取，那么将这两个组合并为一组；如果两个组的访问关系是写入或者写入和读取，则将这两个组合并为一组。然后对合并后的数组引用分组进行遍历、判断和合并，根据合并结果计算索引偏移量等信息，最后更新本地数组副本中的信息。利用数组引用分组信息计算本地数组被引用部分的大小，并将其与预处理阶段提取的上下文等信息相结合，以更新和简化本地数组中的大小和索引偏移量等信息。

以矩阵乘计算为例，输入矩阵 **A** 和 **B** 两个数组都只是被一个一次读取的引用组所访问，所以将 **A** 和 **B** 这两个组合并为一组，还需要根据数组引用组计算本地数组被引用部分的大小，其大小即为混合精度优化阶段需要申请的低精度数组大小，从而降低开销。

在处理完所有数组引用组后，对所有非空的本地数组进行标记，并为这些数组生成强制类型转换的流映射关系，将其存储在对应数组引用组中。同时，在低精度所在的子树中插入一个包含流映射关系的 extension 结点，从而使计算机能够快速访问并使用这些流映射关系。以上操作可以有效实现混合精度计算，提高算法的运行效率和精度。矩阵乘计算实例的强制类型转换的流映射关系如式(7)所示：

$$\begin{cases}[[]\rightarrow A(i,j)]\rightarrow low_A(ii,j)\\ [[]\rightarrow B(i,j)]\rightarrow low_B(i,j)\end{cases} \tag{7}$$

对于输入流，其作用是从数据源中读取矩阵 **A** 与 **B** 的数据，并将其存储在相应的 ***low_A*** 与 ***low_B*** 的位置上。而对于输出流，其含义则与输入流相反。经过数据流分析之后的调度树如图 6 所示。

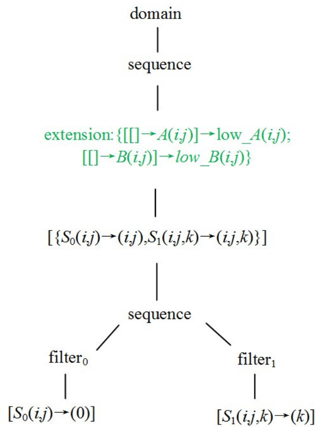


图 6 数据流分析之后的调度树

Fig. 6 Scheduling tree after data flow analysis

3.2.2 调度树转换

调度树转换主要是根据数据流分析的结果，通过对原始调度树中的结点进行各种操作，将其转换成符合混合精度计算的新的调度树的过程。其过程具体步骤如下：

1)利用本地数组中的信息计算低精度计算所需的输入和输出流，再结合流映射关系，从而计算得到强制类型转换对应的调度，并生成对应的 band 结点。

2)在 extension 结点处创建一个 sequence 结点作为子结点，然后将强制类型转换后得到的低精度 band 结点和原始的高精度 band 结点按照计算的先后顺序依次插入 sequence 结点下。由于调度转换过程中需要对调度树中的结点进行各种操作，设置大量 mark 结点来标记子树的附加信息，以方便用户理解和修改，最后将这些 mark 结点删除，保证调度树的简洁。

以矩阵乘计算为例，其低精度计算的输入流和输出流可以用式(8)表示，其转换之后的调度树如图 7 所示，

$$\begin{array}{l}\{read[[]\rightarrow A[o0,o1]]:0\leqslant o0\leqslant M-1 \text{ and } 0\leqslant o1\leqslant K-1\}\\ \{read[[]\rightarrow B[o0,o1]]:0\leqslant o0\leqslant K-1 \text{ and } 0\leqslant o1\leqslant N-1\}\\ \{write[[]\rightarrow C[o0,o1]]:0\leqslant o0\leqslant M-1 \text{ and } 0\leqslant o1\leqslant N-1\}\end{array} \tag{8}$$

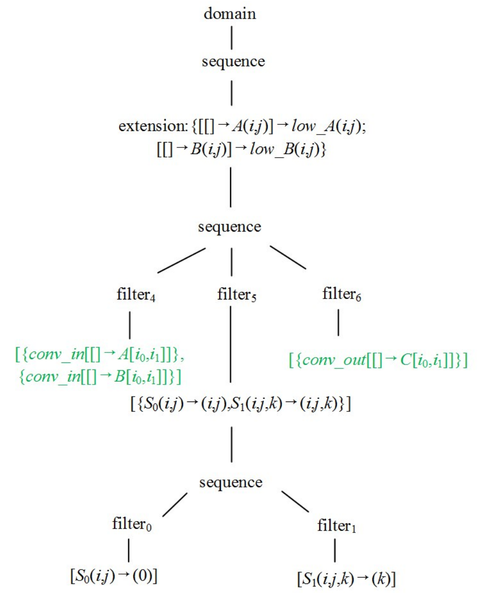


图 7 转换之后的调度树

Fig. 7 Scheduling tree after conversion

3.2.3 代码生成

代码生成子模块将经过混合精度优化后的调度树作为输入，利用多面体模型中的多面体扫描技术生成与调度树相对应的抽象语法树(AST)，最终参照 C 语言规范使用 AST 生成相应的代码。多面体扫描技术的任务是遍历调度树中迭代空间的每一个点，并按照调度树所表示的信息生成对应的 AST。

在该模块中，本文选择“调度树→AST→C 语言代码”的代码生成流程，其中由调度树生成 AST 的过程可利用 isl 库实现；从 AST 到最终代码的生成仅需要遍历抽象语法树(AST)，根据其结点信息向目标文件中输出相应的语句。最后，通过基础编译器进行编译链接，生成可执行文件，即可得到最终的目标程序。

本文在实现过程中对 PPCG 的代码生成模块进行了一定的扩展和改进，以支持混合精度优化等特殊情况下的强制类型转换等特殊代码的生成。由于 PPCG 的代码生成模块已经非常细致和完善，因此这些改进只是对其进行了一些补充，故此处不再赘述代码生成的细节。

生成的低精度乘高精度加混合精度矩阵乘计算的代码如图 8 所示，代码中首先根据计算出的本地数组大小声明了低精度的数组，然后将高精度的数组强制类型转换为低精度的

数组，最后进行了低精度乘高精度加混合精度矩阵乘计算。经过一系列的实验，结果表明，混合精度优化后的代码与原始代码相比具有更快的执行速度和更好的性能。

```
float low_A[1024][1024];
float low_B[1024][1024];
// low
// amp_kernel
{
for(int c0=0;c0 <= 1023;c0 += 1)
    for(int c1=0;c1 <= 1023;c1 += 1)
        low_A[c0][c1] =(float)A[c0][c1];
for(int c0=0;c0 <= 1023;c0 += 1)
    for(int c1=0;c1 <= 1023;c1 += 1)
        low_B[c0][c1] =(float)B[c0][c1];
for(int c0=0;c0 <= 1023;c0 += 1)
    for(int c1=0;c1 <= 1023;c1 += 1)
        for(int c2=0;c2 <= 1023;c2 += 1)
          C[c0][c1] =(C[c0][c1] +(low_A[c0][c2] * low_B[c2]
          [c1]));
}
```

图 8 矩阵乘混合精度计算代码核心段

Fig. 8 Code core segment of matrix multiplication mixed precision calculation

3.3 精度调优

精度调优模型由初始化多组矩阵、计算每组误差矩阵、统计误差分布、记录高频误差点等子模块组成，4 个子模块通过精度调优算法实现。本节主要是通过精度调优算法找到基础混合精度矩阵乘计算中误差出现频率较高的点，从而为后续的高级混合提供依据，优化矩阵乘法的计算结果，减小计算误差，并提高计算的准确性和可靠性。

与高精度矩阵乘计算相比，基础混合精度矩阵乘计算势必会引入误差，当误差超过用户所能接受的误差阈值时，需要降低误差，从而提高计算的准确性。因此本文通过精度调优(PT)算法找到混合精度计算中出现的高频误差点，其算法描述如算法 1 所示。

算法 1 精度调优算法

输入：(A，B，low_A，low_B，threshold)

输出：(record matrix)

```
1. for i←1 to num_trials do:
2. randomly initialize matrix A and matrix B with the required dimen-
   sions;
3.   C=zeros(M,N);
4.   for i←1 to M do
5.     for j←1 to N do
6.       for k←1 to K do
7.         C[i][j]=C[i][j] + A[i][k] * B[k][j];
8.   C_mix=zeros(M,N);
9.   for i←1 to M do
10.    for j←1 to N do
11.      for k←1 to K do
12.        C_mix[i][j]=C_mix[i][j] + low_A[i][k] * low_B[k]
           [j];
13.  initialize record matrix with the same size as C and C_mix;
14.  for i←1 to M do
15.    for j←1 to N do
16.      if abs(C[i][j]-C_mix[i][j]) > threshold then:
17.        record[i][j] += 1;
18.  record thresholds= num_trials * 0.01;
19.  result=[];
20.  for i←1 to M do
21.    for j←1 to N do
22.      if record[i][j] > record thresholds then:
23.        result.append((i,j,record[i][j]));
24. return record,result.
```

此算法主要分为 4 个子模块，1－12 行是初始化多组矩阵子模块，主要通过多次随机初始化输入矩阵 ***A*** 和 ***B***，针对每组不同的输入矩阵计算出每组高精度矩阵乘计算的结果矩阵 ***C*** 和基础混合精度矩阵乘运算的结果矩阵 ***C_mix***。13－17 行是计算每组误差矩阵子模块和统计误差分布子模块，主要通过计算每组 ***C*** 矩阵和 ***C_mix*** 矩阵的绝对误差得到多组误差矩阵。设置误差阈值，初始化一个和误差矩阵同等规模的 ***record*** 矩阵，用来记录超过误差阈值次数即超过误差阈值的频率，每次都将误差矩阵中高于误差阈值的点在 ***record*** 矩阵的对应位置上累加 1，经过多次训练后，最终得到的 ***record*** 矩阵即为误差分布。***record*** 矩阵的元素值表示在混合精度计算过程中，某个点的误差超过误差阈值的次数，因此，record 值越大，就意味着该点在混合精度计算中出现大误差的频率越高。18－24 行高频误差点子模块，主要是为了识别出频率较高的点，设定了一个 record 阈值，该阈值代表了超过这个次数的点具有较高的频率。通过设置这个阈值，可以确定哪些点经常出现较大的误差。

3.4 高级混合

高级混合是一种针对基础混合精度计算中引入的误差进行优化的方法。经过精度调优算法的分析，能够找到那些在计算中频繁引起较大误差的高频误差点。为了减小这些误差并提高计算的准确性，本文采取了高级混合精度计算策略。具体而言，本文选择使用高精度计算方法来处理这些高频误差点，以保证其计算结果的准确性，对于其他点，则继续使用基础混合精度计算。这样的策略能够有效地减小基础混合精度计算中引入的误差，进而提高整体计算的准确性。同时，为了方便应用高级混合精度计算策略，本文方法还能自动生成相应的高级混合精度代码，从而简化开发过程并提高代码的可维护性。高级混合方法的工作流程如图 9 所示。

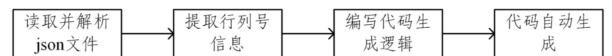


图 9 高级混合工作流程

Fig. 9 Advanced hybrid workflow

通过精度调优算法找到高频误差点，将这些点的行列号保存在 json 文件中。首先利用脚本读取 json 文件中的内容，将其解析为数据结构，以便后续处理。接着根据 ison 文件的结构，提取出高频误差点的行列号信息，这些信息将用于后续的代码生成过程。然后根据高频误差点的行列号信息，编写生成混合精度代码的逻辑。使用条件语句来判断某个点是否

为高频误差点，如果是，则生成高精度计算的代码，否则生成混合精度计算的代码。最后将代码生成逻辑封装在脚本中，并将数据提取与代码生成过程结合起来，实现自动化的代码生成。这样遍历所有数据点，并根据其行列号信息判断是否为高频误差点，然后生成相应的混合精度代码。这种自动化的方式可以显著提高开发效率，减少手动工作的时间并降低发生错误的风险。

4 实验分析

面向矩阵乘计算自动生成的混合精度代码都用 C 语言实现。将生成的高级混合精度代码与原始程序的高精度代码和低精度代码的运行结果进行对比，以准确地评估其性能和误差。

4.1 实验平台及测试用例

实验选取了 13 个规模在 128 * 128 * 128 到 1024 * 1024 * 1024之间的矩阵乘法程序作为测试用例。实验平台信息如表 1 所列。

表 1 测试平台信息

Table 1 Test platform information

type	X86 platform
CPU	Intel(R) Core(TM) i7-9700K CPU
Compiler	GCC 9.4.0
ICC	-O0
PPCG	0.08.1
PPCG	--target=c
OS	Ubuntu 20.04.4 LTS
TAFFO	TAFFO V0.3.0

4.2 性能测试

为了更准确地测试高级混合精度代码的性能，同时还测试了基础混合精度代码、高精度矩阵乘代码的性能。为了保证性能测试的稳定性，采用以下性能测试方法：获取 10 次程序运行时间，对 10 次程序执行时间求平均值。只要有一次执行时间超过平均值的 5%，就重新获取 10 次程序运行时间，计算新的平均值，重复判断，判断 5 次后，若结果仍不符合条件，则去掉这组数据中高于平均值 5% 的值，对其余执行时间求取平均值。

从表 2 的测试数据可以看出，record 阈值越大，混合精度矩阵乘计算时用高精度计算的点越少，性能越好。因此可以得出 record 阈值的大致范围，record 阈值的下界是式(9)求得的值，record 阈值依次加 5 生成混合精度方案，直到方案中需要用高精度计算的点是 0 为止。

$$\arg\max(count([floor(A(1,1)/10)\times 10,\cdots,floor(A(m,n)/10)\times 10])) \tag{9}$$

其中，$A(i,j)$ 表示矩阵中的元素，$floor(x)$ 表示向下取整函数，arg max 表示取最大值对应的参数。

确定循环初始化输入矩阵的次数时，一方面，若循环次数太多，则执行时间较长，性能开销较大，而且生成混合精度方案时，record 阈值较小的改变可能会生成不同的混合精度方案，因次需要多次设置 record 阈值才能找到合适的混合精度方案；另一方面，若循环次数太少，则生成的混合精度方案不稳定。因此本文选取循环初始化输入矩阵的次数为 100 次。

表 2 不同混合精度方案的性能表现

Table 2 Performance of different hybrid accuracy schemes

Matrix size	Record thresholds	Error thresholds	interval	Points to be calculated with high precision(number)	execution time
1024×1024×1024	45	0.0000005	[0,1]	2	5.168758
1024×1024×1024	40	0.0000005	[0,1]	256	5.447382
1024×1024×1024	35	0.0000005	[0,1]	7746	5.692934
1024×1024×1024	30	0.0000005	[0,1]	92951	7.397988
1024×1024×1024	25	0.0000005	[0,1]	433608	7.888669

对基础混合精度代码、高精度矩阵乘计算代码、高级混合精度代码进行性能测试，实验选取了 13 个规模在 128×128×128 到 1024×1024×1024 之间的矩阵乘法程序作为测试用例。首先测试了 13 个不同矩阵规模在相同的误差阈值下(误差阈值为 0.0000001)找到各自的最短执行耗时，并以 double 精度计算为基准计算基础混合精度计算的加速比和高级混合精度计算的加速比，从而量化高级混合精度的加速效果。加速比的计算式如式(10)所示：

$$speedup=\frac{T_{before}}{T_{after}} \tag{10}$$

其中，T_{before} 代表加速前的执行时间，T_{after} 代表加速后的执行时间。所有测试规模的性能表现如图 10 所示，其中 mix 是低精度乘高精度加基础混合精度代码，recompute 是高级混合精度代码。从图 10 中可以看到，基础混合精度计算的最大加速比为 1.49，几何平均加速比为 1.16；高级混合精度计算的最大加速比为 1.39，几何平均加速比为 1.14。

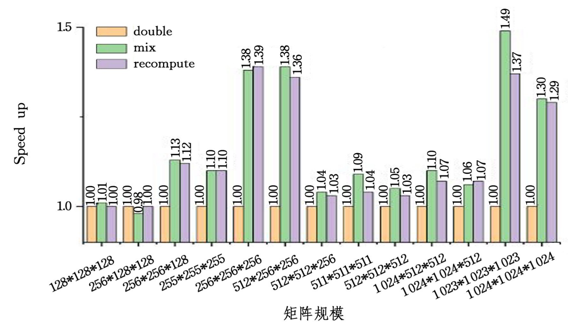


图 10 不同测试规模的加速比

Fig. 10 Acceleration ratios at different test sizes

为了更加充分地考虑测试的各种情况，验证高级混合精度程序的效果，又测试了 4 个矩阵规模在不同误差阈值下各自的最短执行耗时，并以 double 精度计算为基准计算基础混

合精度计算的加速比、高级混合精度计算的加速比,从而量化高级混合精度的加速效果。256 * 256 * 256 规模的矩阵性能表现如图 11 所示,基础混合精度计算的最大加速比1.40,几何平均加速比为 1.39;高级混合精度计算的最大加速比为 1.39,几何平均加速比为 1.38。

511 * 511 * 511 规模的矩阵性能表现如图 12 所示,基础混合精度计算的最大加速比为 1.09,几何平均加速比为 1.07;高级混合精度计算的最大加速比为 1.11,几何平均加速比是 1.06。

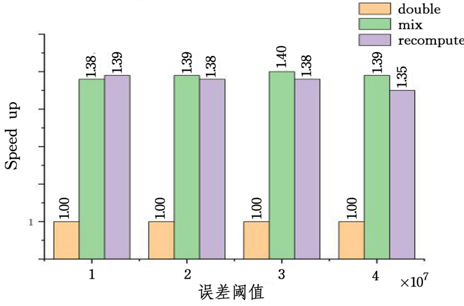


图 11 256 * 256 * 256 规模的矩阵性能表现

Fig. 11 Matrix performance at 256 * 256 * 256 scale

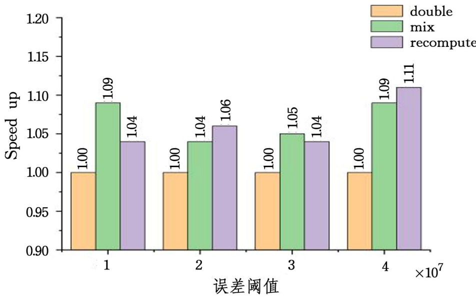


图 12 511 * 511 * 511 规模的矩阵性能表现

Fig. 12 Matrix performance at 511 * 511 * 511 scale

512 * 512 * 512 规模的矩阵性能表现如图 13 所示,基础混合精度计算的最大加速比为1.05,几何平均加速比为1.05;高级混合精度计算的最大加速比为 1.06,几何平均加速比为 1.04。

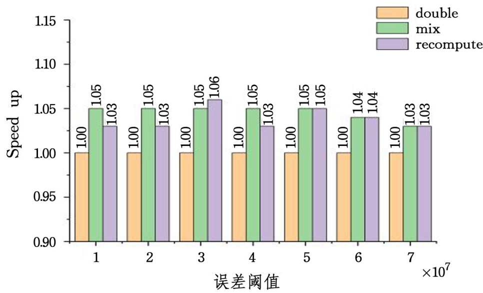


图 13 512 * 512 * 512 规模的矩阵性能表现

Fig. 13 Matrix performance at 512 * 512 * 512 scale

1 024 * 1 024 * 1 024 规模的矩阵性能表现如图 14 所示,基础混合精度计算的最大加速比为 1.33,几何平均加速比为 1.30;高级混合精度计算的最大加速比为 1.32,几何平均加速比为 1.29。

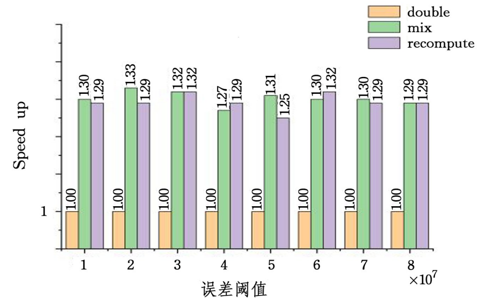


图 14 1 024 * 1 024 * 1 024 规模的矩阵性能表现

Fig. 14 Matrix performance at 1 024 * 1 024 * 1 024 scale

为了更准确地评估本文面向矩阵乘进行自动混合精度计算生成与优化的实际效果,本文将自动混合精度计算系统(下文简称 AGMMMPC)与国际主流的精度优化工具 TAFFO[12] 工具和国内主流的混合精度工具 AMPO-SC[29] 进行对比测试。AGMMMPC 和 AMPO-SC 都是源到源的工具,而 TAFFO[12] 是基于 LLVM 实现浮点数到定点数转换的自动混合精度工具,其生成的结果是二进制可执行文件。为了保持和 TAFFO 对比时环境配置的一致性,本文对其脚本进行了统一的修改,之后用该脚本来测试 AGMMMPC 和 AMPO-SC 生成的源代码,以保证各种环境配置的统一。

AMPO-SC 混合精度方法主要是对循环嵌套程序中的最外层循环进行拆分,一部分用高精度计算,其余部分用低精度计算,从而实现混合精度计算,用 double 占比控制混合精度计算。在测试时,为了准确得到 AMPO-SC 工具在每个矩阵规模下最大的加速比,本文测试每个矩阵规模的性能表现时选取的混合比例是 $\{rate \mid rate\%5=0, rate\in[5,95]\}$,测试各个混合比例下的加速比并选出最大的加速比作为这个矩阵规模的最大加速比。

以 double 计算为基准,在每个的矩阵规模下,对比 3 个工具的最大加速比。对比结果如图 15 所示。当矩阵规模为 512 * 512 * 256,511 * 511 * 511,512 * 512 * 512 时,AMPO-SC 加速比高于 AGMMMPC 的加速比;矩阵规模为 511 * 511 * 511 时,TAFFO 加速比高于 AGMMMPC 的加速比。AGMMMPC 在多个矩阵规模下的加速比高于 AMPO-SC 和 TAFFO 的加速比,同时当矩阵规模越大时,AGMMMPC 加速比明显高于 AMPO-SC 和 TAFFO。

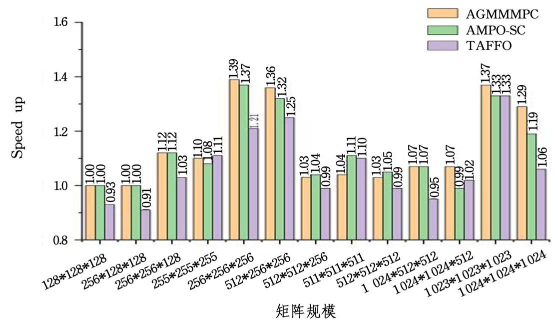


图 15 性能对比测试结果

Fig. 15 Performance comparison test results

总体而言,AGMMMPC 的加速比明显高于 AMPO-SC

和 TAFFO。主要是因为本文实现的 AGMMMPC 工具提出的混合精度方法是根据矩阵乘计算的规律:乘法操作的次数通常比加法操作多得多。在矩阵乘计算过程中实现混合精度,将乘法操作采用低精度实现,能够大幅加快计算速度。

4.3 精度测试

在对生成的高级混合精度代码进行误差测试时,本文采用平均误差来刻画不同结果的可靠程度,并以高精度矩阵乘计算为基准,以低精度计算的结果作为对比。选取了 13 个规模在 128 * 128 * 128 到 1 024 * 1 024 * 1 024 之间的矩阵乘法程序作为测试用例,在相同误差阈值下,测试各个矩阵规模的平均误差,每次测试得到的误差可能会因为计算过程中的舍入误差和数值不稳定性而有所不同。为了选择评估指标,可以通过记录每次测试的误差,并从中选择最大值。选择最大误差作为评估指标是一种保守的方法,确保在各种测试情况下都能满足预期的精度要求。平均误差的计算方法是用矩阵中的绝对误差总和除以矩阵规模。

图 16 中,数据类型 1.0 表示 double 和 float 的平均误差,数据类型 2.0 表示 double 和 mix 的平均误差,数据类型 3.0 表示 double 和 recompute 的平均误差。从图中可以看出,各个规模下,低精度计算的平均误差最大,recompute 计算的平均误差最小。recompute 可以减小低精度乘高精度加混合精度计算中引入的误差,从而在尽可能提升程序性能的前提下,提高程序的正确性。

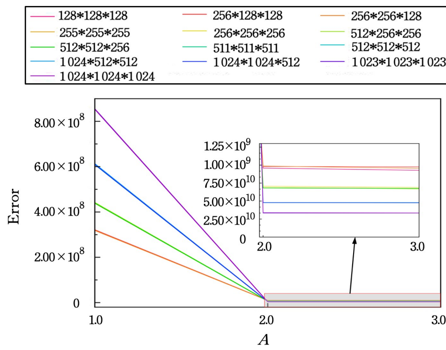


图 16 各个矩阵规模的平均误差

Fig. 16 Average error at different matrix sizes

选取每个矩阵规模下性能最好时的代码进行精度测试,以 double 计算为基准,测试 3 个工具在每个的矩阵规模下引入的平均误差,对比结果如表 3 所列。

表 3 误差对比测试结果

Table 3 Error comparison test results

Matrix size	AGMMMPC	AMPO-SC	TAFFO
128×128×128	9.201×10^{-10}	2.36×10^{-1}	2.41×10^{-4}
256×128×128	9.201×10^{-10}	2.36×10^{-1}	2.42×10^{-4}
256×256×128	9.581×10^{-10}	2.23×10^{-1}	2.43×10^{-4}
255×255×255	6.628×10^{-10}	2.25×10^{-1}	4.87×10^{-4}
256×256×256	6.600×10^{-10}	2.23×10^{-1}	4.87×10^{-4}
512×256×256	6.661×10^{-10}	2.24×10^{-1}	4.88×10^{-4}
512×512×256	6.770×10^{-10}	2.12×10^{-1}	4.88×10^{-4}
511×511×511	4.759×10^{-10}	2.37×10^{-1}	9.76×10^{-4}
512×512×512	4.767×10^{-10}	2.24×10^{-1}	9.76×10^{-4}
1 024×512×512	4.781×10^{-10}	2.37×10^{-1}	9.76×10^{-4}
1 024×1 024×512	4.755×10^{-10}	1.88×10^{-1}	9.76×10^{-4}
1 024×1 024×1 024	3.336×10^{-10}	2.37×10^{-1}	1.95×10^{-3}

AGMMMPC 的误差远小于 AMPO-SC 和 TAFFO 的误差,即 AGMMMPC 在减少高精度计算中的精度冗余时不会产生过大的误差。主要是因为本文实现的 AGMMMPC 工具提出的精度调优算法可以提前找到误差出现频率较高的点,并且通过高精度计算减小计算误差,并提高计算的准确性和可靠性。

结束语 为了生成高效的矩阵乘法代码,本文基于多面体模型,利用 PPCG 对矩阵乘计算的数据流进行分析,通过调度树转换进行数据类型转换,将低精度乘高精度加混合精度矩阵乘代码生成功能添加到"源-源"的 PPCG[6] 编译器中,从而自动生成性能更优的基础混合精度矩阵乘代码。通过精度调优算法(PT)统计基础混合精度计算中的误差分布,找到出现频率较高且误差较大的点。为了减小误差,对高频且误差大的点用高精度计算,其余点使用混合精度计算的策略,并根据此策略自动生成高级混合精度代码。最后对生成的代码进行测试。

本文虽然实现了混合精度矩阵乘代码自动生成,但是仍有很多有待优化的地方,因此,下一步的工作有:1)结合循环分块手段,进一步进行混合精度优化,提升矩阵乘法的性能;2)在生成混合精度代码之后,对其误差和性能的提升以及损耗进行建模,并通过计算该模型直接得到较好的混合精度方案,不需要用户指定 record 阈值参数,从而直接自动生成最优化的混合精度代码。

参 考 文 献

[1] FAWZI A, BALOG M, HUANG A. et al. Discovering faster matrix multiplication algorithms with reinforcement learning[J]. Nature, 2022, 610: 47-53.

[2] MICIKEVICIUS P, MARANG S, ALBEN J, et al. Mixed precision training for deep neural networks[J]. arXiv: 1710.03740, 2017.

[3] HUANG X, ZHANG J, CHEN T, et al. Mixed-Precision Training for NLP and Speech Recognition with OpenSeq2Seq[J]. arXiv: 2010.12895, 2018.

[4] VERDOOLAEGE S, CARLOS JUEGA J, COHEN A, et al. Polyhedral Parallel Code Generation for CUDA[J]. ACM Transactions on Architecture & Code Optimization, 2013, 9(4): 1-23.

[5] DUGHMI S, XU H F. Algorithmic Bayesian persuasion [C]// Proceedings of the forty-eighth annual ACM symposium on Theory of Computing (STOC'16). Association for Computing Machinery, New York, NY, USA, 2016: 412-425.

[6] DEMMEL J. Applied Numerical Linear Algebra[M]. Tsinghua University Press: Siam, 2011.

[7] WANG B Y, PANG J M, XU J L, et al. Matrix multiplication vector code generation based on polyhedral model[J]. Computer Science, 2022, 49(10): 44-51.

[8] KOTIPALLI P V, SINGH R, WOOD P, et al. AMPT-GA: Automatic mixed precision floating point tuning for GPU applications[C]// Proceedings of the ACM International Conference on Supercomputing (ICS'19). Phoenix Arizona: ACM, 2019: 160-170.

[9] SOLOVYEV A,JACOBSEN C,RAKAMARIĆZ,et al. Rigorous estimation of floating-point round-off errors with symbolic taylor expansions[M]. Cham: Springer International Publishing, 2015:532-550.

[10] DARULOVA E,HORN E,SHARMA S. Sound mixed-precision optimization with rewriting[C]//2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems(ICCPS). IEEE, 2018:208-219.

[11] DARULOVA E,KUNCAK V. Towards a compiler for reals [J]. ACM Transactions on Programming Languages and Systems(TOPLAS),2017,39(2):1-28.

[12] CHERUBIN S,CATTANEO D,CHIARI M,et al. TAFFO: Tuning assistant for floating to fixed point optimization[J]. IEEE Embedded Systems Letters,2019,12(1):5-8.

[13] RUBIO-GONZÁLEZ C,NGUYEN C,NGUYEN H D,et al. Precimonious: Tuning assistant for floating-point precision [C]//Proceedings of the International Conference on High Performance Computing,Networking,Storage and Analysis. IEEE, 2013:1-12.

[14] RUBIO-GONZÁLEZ C,NGUYEN C,MEHNE B,et al. Floating-point precision tuning using blame analysis[C]// 2016 IEEE/ACM 38th International Conference on Software Engineering(ICSE). IEEE,2016:1074-1085.

[15] MENON H,LAM M O,OSEI-KUFFUOR D,et al. ADAPT: Algorithmic differentiation applied to floatingpoint precision tuning[C]// International Conference for High Performance Computing,Networking,Storage and Analysis. IEEE,2018:614-626.

[16] HO N M,MANOGARAN E,WONG W F,et al. Efficient floating point precision tuning for approximate computing[C]// 2017 22nd Asia and South Pacific Design Automation Conference(ASP-DAC). IEEE,2017:63-68.

[17] LAGUNA I,WOOD P C,SINGH R,et al. Gpumixer:Performance-driven floating-point tuning for gpu scientific applications [C]// International Conference on High Performance Computing. Cham:Springer,2019:227-246.

[18] FEAUTRIER P,LENGAUER C. Polyhedron model[C]// Proceedings of the Encyclopedia of Parallel Computing. 2011:1581-1592.

[19] ZHAO J,LI Y Y,ZHAO R C. Compilation “black magic” based on polyhedral model[J]. Journal of Software,2018,29(8):2371-2396.

[20] BONDHUGULA U,HARTONO A,RAMANUJAM J,et al. A practical automatic polyhedral parallelizer and locality optimizer [C]// Proceedings of the 29th ACM SIGPLAN ConfErence on Programming Language Design and Implementation(PLDI). ACM Press,2008:101-113.

[21] TRIFUNOVIĆ K,COHEN A,EDELSOHN D,et al. Graphite two years a fter:First lessons learned from real-world polyhedral compilation[C]//Proceedings of the 2nd GCC Research Opportunities Workshop(GROW). 2010.

[22] GROSSER T,GROESSLINGER A,LENGAUER C. Polly-Performing polyhedral optimizations on a low-level intermediate representation[J]. Parallel Processing Letters, 2012, 22(4): 1250010.

[23] KELLY W,PUGH W. A unifying framework for iteration reordering transformations[C]//Proceedings of the IEEE 1st International Conference on Algorithms and Architectures for Parallel Processing(ICAPP'95). 1995.

[24] GIRBAL S,VASILACHE N,BASTOUL C,et al. Semi-Automatic Composition of Loop Transformations for Deep Parallelism and Memory Hierarchies[J]. International Journal of Parallel Programming,2006,34(3):261-317.

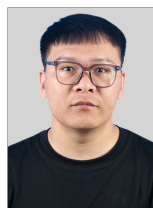
[25] VERDOOLAEGE S. Counting affine calculator and applications [C]// Proceedings of the 1st International Workshop on Polyhedral Compilation Techniques(IMPACT'11). Charmonix, 2011.

[26] GUO H,RUBIO-GONZÁLEZ C. Exploiting community structure for floating-point precision tuning[C]// Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis. Amsterdam Netherlands: ACM, 2018: 333-343.

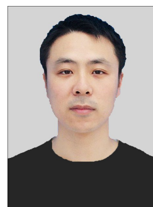
[27] VERDOOLAEGE S,GROSSER T. Polyhedral extraction tool [C]// Proceedings of the 2nd Int'l Workshop on Polyhedra l Compilation Techniques(IMPACT). 2012.

[28] VERDOOLAEGE S. ISL:An integer set library for the polyhedral model[C]// Proceedings of the ICMS 2010. Berlin,Heidelberg:Springer-Verlag,2010:299-302.

[29] SONG G H,GUO S Z,ZHAO J,et al. Automatic hybrid accuracy optimization for Stencil computing[J]. Journal of Software, 2023,34(12):5704-5723.



HE Haotian, born in 1997, postgraduate. His main research interest is high-performance computing.



XU Jinchen, born in 1987, Ph.D, associate professor. His main research interest is high-performance computing.

(责任编辑:何杨)