ParaGraph：用于高性能计算内核性能优化的加权图表示

Ali TehraniJamsaz[1]，Alok Mishra[2]，Akash Dutta[1],Abid M. Malik[3],Barbara Chapman[2],Ali Jannesari[1]艾奥瓦州立大学，艾姆斯，艾奥瓦州，美国{tehrani, adutta, jannesar}@iastate.edu[2]石溪大学，石溪，纽约州，美国{almishra, bchapman}@cs.stonybrook.edu[3]布鲁克海文国家实验室，厄普顿，纽约州，美国amalik@bnl.gov

arXiv:2304.03487v1 [cs.DC] 7 Apr 2023

摘要—**基于GPU 的高性能计算集群凭借其广泛的并行性和能效，正吸引越来越多的科学应用开发人员。为了实现多种多核/众核架构之间的可移植性，应用开发人员普遍选择使用基于指令的并行编程模型，例如OpenMP。然而，即使使用 OpenMP，开发人员仍必须从众多利用 GPU或 CPU 的策略中进行选择。近年来，机器学习（ML）方法推动了高性能计算应用优化的重大进展。为此，人们提出了多种方式来表示应用程序的特征，以供机器学习模型使用。然而，现有技术无法捕捉对于揭示并行性至关重要的特征。在本文中，我们为<стильid='3'>并行应用程序引入了一种新的基于图的程序表示，该表示扩展了抽象语法树，以表示控制流和数据流信息。这项工作的创新之处在于，增加了利用抽象语法树中的隐式顺序和父子关系的新边，并引入边权重来体现循环和条件信息。我们通过训练图神经网络（GNN），预测跨CPU和 GPU 的 OpenMP 代码区域的运行时间，以评估所提出的表示。我们利用循环折叠以及 CPU与 GPU 之间的数据传输进行各种变换，以构建数据集。模型预测的运行时间用于确定哪种变换能够提供最佳性能。结果表明，我们的方法确实有效，其运行时间预测的归一化均方根误差低至 4×10^{-3}，最高为 1×10^{-2}。**

索引词—**OpenMP、高性能计算、卸载、程序表示**

I. 引言

多年来，片上核心数量的增加显著提升了并行代码的性能。为充分利用不断增强的计算能力，人们很容易通过 Pthreads 或 OpenMP 构造来修改应用程序。过去十年中，通用图形处理器（GPGPU）日益普及。它们能够以较低的功耗处理大规模数据并行性。因此，如今大多数高性能计算平台都配备了 GPGPU。高性能计算平台将继续支持更多加速器，但即便是利用和配置单个加速器都困难重重，未来利用和配置多个异构加速器将变得愈发困难。另一方面，利用 GPU。

实际上，这带来了需要重新设计代码和应用程序的挑战。如果未被正确使用，图形处理器的全部计算能力可能无法得到利用。此外，主机（中央处理器）与设备（图形处理器）之间不恰当的数据传输可能会导致巨大的开销。对于开发人员来说，为具有多个设备的高度异构平台创建应用程序极具挑战性且负担沉重。近年来出现的工具和编程模型旨在自动化应用程序适应异构平台的过程。最流行的并行编程模型之一 OpenMP [1],旨在简化开发能够在不同架构上运行的并行程序的过程。尽管如此，对于大型且复杂的应用程序而言，优化代码以正确使用 OpenMP 指令仍然是一项繁琐的任务。

深度学习（DL）的最新进展使研究人员能够将深度学习应用于各种软件工程任务和挑战，包括代码注释生成、编译器优化以及异构平台适配。由于源代码无法直接输入深度学习模型，我们需要为应用程序提供合适的表示，以作为各种深度学习模型的输入。本文提出$ParaGraph$，一种基于图的程序表示，旨在揭示高性能计算应用的关键特征（例如循环中的迭代次数）。据我们所知，现有程序表示并非为揭示和表示并行高性能计算应用的特征而设计。这些程序表示通常面向串行代码，未纳入并行代码的特征。因此，基于这些表示构建的深度学习模型无法对并行代码固有的特征进行建模。

此前的工作，例如 COMPOFF [2]，，依靠特征工程来克服这些缺点。然而，特征工程需要专家知识。对于高性能计算这样快速发展的领域，始终依赖专家干预来进行此类特征工程并不现实。因此，我们需要一种能够自动提取此类特征的自适应方法。我们提出的方法旨在解决这一问题，并使用图神经网络（GNN）对 $ParaGraph$ 生成的代码图进行建模。实验结果表明，我们的模型能够预测高性能计算内核的运行时间，且具有.....

极低的误差率（以归一化均方根误差计，最多为 1×10^{-2}），证实了我们策略性方法的有效性。

在本文的其余部分中，我们首先在第二节讨论一些背景工作，包括程序表示、OpenMP GPU 卸载、编译器中的深度学习、OpenMP Advisor（我们用来为本研究生成各种内核的工具）以及其他相关工作。接着，我们在第三节定义 $ParaGraph$。第四节介绍本文开展的实验，第五节讨论实验结果。最后，我们在第六节总结本研究，并讨论未来计划。

II. 背景与其他相关工作

本节将讨论与程序表示、OpenMP 并行以及相关工具有关的研究和一些背景知识。

A. 程序表示

近年来，随着机器学习（ML），尤其是深度学习（DL）的突破，研究人员开始将数据驱动方法应用于各种软件工程任务和挑战，范围从代码注释生成 [3] 到编译器优化 [4]。然而，源代码不能直接输入深度学习模型，因为深度学习模型不接受字符串作为输入。因此，需要以模型能够使用的形式呈现应用程序。一些研究将程序表示为词元序列 [5], [6]。尽管这种表示方式在某些下游任务中效果良好，但将程序表示为词元序列会丢弃程序至关重要的句法和语义特征，而这些特征对于深度学习模型进行推理不可或缺。

近年来，为了更好地呈现程序的语义和语法信息，研究者提出了基于图的表示。Allamanis 等人 [7]提出了增强抽象语法树，并添加了新的边来呈现控制流信息。他们将增强抽象语法树应用于两项任务：变量误用检测和变量名建议。受他们工作的启发，我们也通过添加新的边来增强抽象语法树。不过，我们还加入了额外的边，以更好地表示 $loops$ 和 if 条件。

Cummins 等人在 [4] 提出了一种基于 LLVM 中间表示的低级图表示，用于解决编译器优化任务。这些表示对于下游任务（如简单的算法分类）是有效的。据我们所知，尚不存在一种专门用于表示并行 循环和条件语句特征的表示。

B. OpenMP GPU 卸载

当不同架构分别支持不同的原生语言，或针对同一种语言采用不同的优化技术时，程序可移植性就会成为一个严重问题。遗憾的是，编程语言和编译器仍远不足以独立解决可移植性问题。对于需要对同一数据进行多次设备内存访问的开发者，或必须确保程序在具有不同节点架构的系统上高效运行的开发者而言，程序可移植性应当是首要考虑因素，例如众核节点与 GPU 加速节点。

采用基于指令的编程范式，例如 OpenMP（C/C++和 Fortran 并行编程的事实标准），是确保跨多个架构实现可移植性的一种方法。OpenMP 致力于转向极其异构的架构 [8]，并从 4.0 规范开始支持 GPU 卸载。不幸的是，即使使用 OpenMP，优化大规模应用程序仍然是一项具有挑战性的任务。

由于GPU具有高度并行性，程序员应充分利用这一事实，以获得尽可能高的性能。单独使用“OpenMP parallelfor指令”pragma只能为 CPU 并行化代码；它不会将计算卸载到GPU。我们期望像GPU 这样的平台具备高水平的粗粒度并行性。GPU 能够利用的并行性规模受到这一设计的限制。图 1说明了 OpenMPteams 和 distribute 指令如何创建额外的并行级别。在target 区域开始时，只有一个 team 和一个成员线程处于活动状态。teams distribute指令将完整的循环迭代空间分配给所有可用的 team。此外，对于嵌套并行循环，我们可以在其中使用 parallel for 指令，进一步将嵌套循环的迭代分配给team内的线程。当只有一个级别的并行循环，或外层循环具有足够的并行性时，我们使用组合指令“teamsdistribute parallel for”，将一个循环的迭代空间分配给team 以及 team 内的线程。team 用于对线程进行分组，而 distribute 则使一个 team 组能够被调度以执行循环。team 类似于 NVIDIAGPU上的 CUDA threadblock，因为没有同步原语可以充当不同 team 的线程之间的屏障。而每个team 内的线程通常使用 parallel for 进行并行化。

目前正在开发多种框架，这些框架将自动协助应用开发人员处-理严重的异构性。这些框架需要分析性成本模型，帮助他们选择应用程序所需的优化类型。目前广泛采用手工调优的成本函-数；然而，计算优化成本需要更深入地了解底层硬件。尽管手动为单一架构构建成本模型很有效，但可能需要数月时间。由于成本函数至关重要且手动调优非常耗时，编译器工程师正在研究 Machine，而

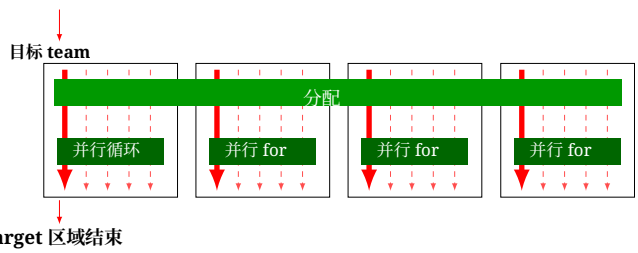


图 1：OpenMP target teams distribute parallel for。

将深度学习方法作为实现该过程自动化的手段。

C. 编译器中的机器学习

近年来，许多研究也致力于探讨如何在编译过程中运用基于学习的技术。早期在编译器中应用机器学习的工作，主要探索如何利用它来帮助优化顺序程序。然而，过去十年间，随着多核平台日益普及，以及近来异构系统的出现，机器学习在优化并行程序任务中的应用受到广泛关注 [9], [10]。Mirka 等人 [11] 设计了一种基于决策树的方法，用于预测 OpenMP 并行区域的调度策略。此外，Denoyelle 等人 [12] 使用机器学习技术，针对调度策略和线程数优化 OpenMP 程序。Alcaraz等人 [13] 使用神经网络和性能计数器来预测线程数。Dutta等人 [14] 使用基于 LLVM-IR 的图表示和性能计数器训练图神经网络模型，以预测 OpenMP 循环的线程数、块大小和调度策略。Malik等人 [15], 也使用了基于树和图的特征，并提出了一种独特的基于图的特征表示方法。研究人员还使用基于学习的技术构建分类器，以判断是否应卸载 OpenCL 代码 [16], 并选择处理器应运行的时钟频率 [17]。研究报告称准确率很高；然而，由于这些工作并未尝试生成修改后的代码，因此无法量化其收益。他们还探索了回归技术，通过构建曲线拟合模型来寻找处理器之间工作划分的最佳点 [18] ，或实现能耗与性能之间的权衡 [19]。此前将机器学习应用于编译器优化的研究成果令人鼓舞。然而，还需要探索新的特征工程实践，以帮助基于学习的方法更深入地了解代码及其计算需求。在 COMPOFF [2], 中，作者提供了一个概念验证，展示如何使用人工神经网络模型，借助 OpenMP 更好地决定是否将内核卸载到图形处理器。他们采用全连接前馈网络，也称为多层感知机（MLP）；这类网络本质上由多个线性回归层堆叠而成。

D. OpenMP Advisor

OpenMP Advisor [20]是一款首创的编译器工具，可通过机器学习实现将 OpenMP 代码卸载到图形处理器。Advisor 分为三个主要模块：内核分析、成本模型和代码转换。内核分析模块会为给定的应用程序推荐各种变体，而代码转换模块则为这些变体生成代码。在我们的工作中，我们使用 OpenMP Advisor 的代码转换模块生成各种内核，以训练我们的模型。通过使用基于机器学习的成本模型 COMPOFF，该工具可以预测内核的运行时间，从而识别最适合卸载的内核。但 COMPOFF 存在一些局限性。它需要确定一个内核中包含多少操作，而这本身就是一项具有挑战性的任务。Advisor 当前的功能仅限于图形处理器，但未来有望扩展，以支持其他支持 OpenMP 的硬件。扩展到其他设备的一种方法是开发一个同样能够在图形处理器之外运行的模型。将 COMPOFF 扩展到图形处理器以外是一项艰巨的任务，因此需要一种能够弥合这一差距的新成本模型。

E. 其他相关工作

一种名为 BLISS [21]的近期工具，无需插桩或领域专属知识，即可对并行应用进行自动调优。该自动调优器借助贝叶斯优化构建多种轻量级模型，其输出质量可与复杂、庞大的机器学习（ML）模型相媲美。在同一领域，其他自动调优器也经常使用贝叶斯优化，包括 BOHB [22], 、HiPerBOt [23], 、GPTune [24] 和 ytopt [25]。遗憾的是，由于评估函数开销高昂，要调优日益异构、复杂且评估成本高昂的大规模高性能计算系统仍然颇具挑战。尽管如此，上述大多数自动调优器都是在线的，这意味着即使在推理阶段，它们也会迭代执行应用程序来进行调优。因此，对于复杂、异构的大规模高性能计算系统，应用这些调优器的成本会非常高。相比之下，$ParaGraph$ 是一种离线方法，在推理阶段无需执行应用程序。COMPOFF [2] 提供了一种新颖、可移植的成本模型，可静态计算 OpenMP 在各种架构上的卸载成本。COMPOFF 也是一种离线方法。在这项工作中，我们将 $ParaGraph$ 与 COMPOFF 进行比较，因为两者都能静态预测内核在图形处理器上的运行时间。

III. 段落

在本节中，我们介绍 $ParaGraph$，一种能够捕捉与高性能计算内核相关特征的程序加权图表示。抽象语法树是 $ParaGraph$ 的基础。事实上，我们在抽象语法树中引入额外的边和属性，以构建 $ParaGraph$。一些先前的研究提出了多种扩展抽象语法树以呈现语义信息 [7], [26] 的方法。尽管这些基于抽象语法树的程序表示在训练用于变量误用检测等下游任务的深度学习模型时很有效，但对于如何更好地表示对应用程序性能影响较大的程序区域，例如 $loops$ 和 if 条件，关注仍然不足。在 $ParaGraph$ 中，我们尝试解决这些不足。如前所述，$ParaGraph$ 基于抽象语法树；不过，它包含新的边，用以呈现控制流或节点子节点顺序等特征。此外，$loops$ 由特殊边表示，这些边传达循环条件及其循环体迭代执行的顺序。另外，我们为边添加权重，以表示程序执行期间每个节点将被使用的次数。值得注意的是，所有这些扩展都是静态应用的；因此，

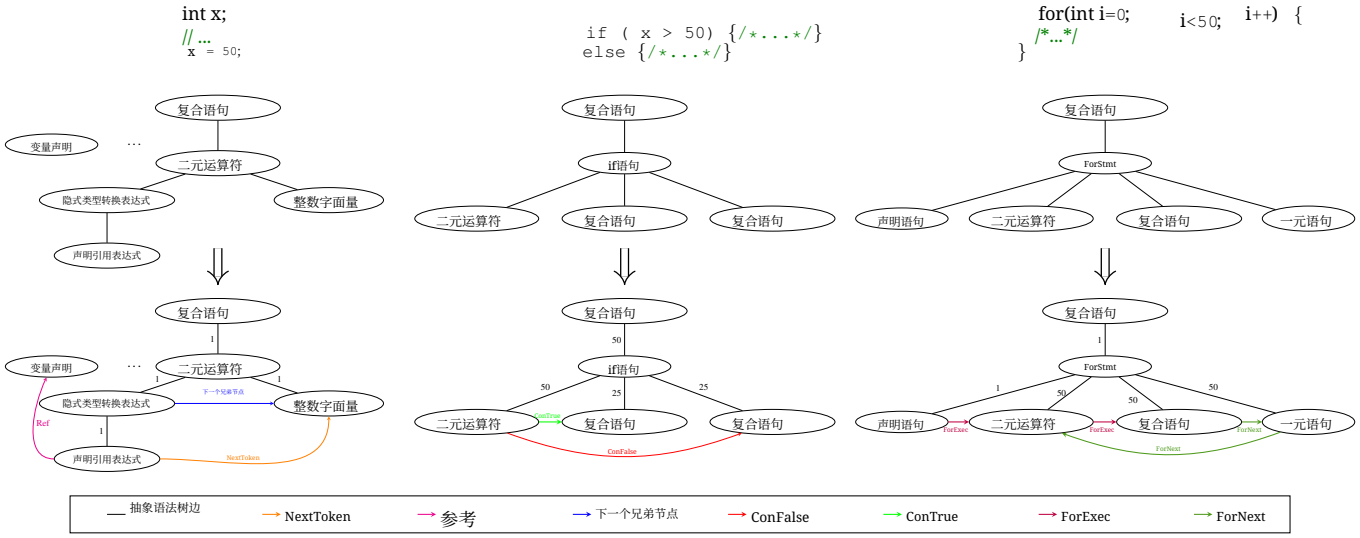


图2：对抽象语法树进行修改，以构建 ParaGraph 的增强抽象语法树

它们不会造成显著开销。 *ParaGraph* 可直接被图神经网络模型利用，我们的实验结果表明，在性能优化方面，这种表示非常有效，并且优于当前最先进的方法。

A. ParaGraph 构建

1) 抽象语法树： 如前所述， *ParaGraph*的基础是抽象语法树。通常，可以使用任何类型的编译器来生成抽象语法树。本研究使用广受欢迎的 C/C++ 前端 Clang 来解析和编译 C/C++程序。抽象语法树包含两种类型的节点：非终结和终结。非终结节点通常称为语法节点，而终结节点称为语法词元。抽象语法树中的节点之间具有父子关系。获取抽象语法树后，我们会添加额外的边和属性，以描述与控制流和数据流相关的信息。以下小节将进一步介绍这些扩展。

2) 增强抽象语法树： 抽象语法树通常提供程序的结构和语法信息。尽管这些信息在一定程度上有助于神经网络学习程序的特征，但已有研究表明，添加额外的属性和信息，尤其是从编译器的角度出发的信息，可以提升深度学习模型的学习能力。受现有研究启发 [7], [26],，我们引入以下额外边来扩展抽象语法树（如图2所示），以表示与程序控制流和数据流相关的信息。

– **NextToken：** 默认情况下，语法词元之间没有规定顺序。然而，从编译器的角度来看，语法词元是有顺序的。语法词元的顺序是从最左侧的词元到最右侧的词元。为了在图中呈现这一信息，我们引入了一种名为 NextToken 的新边类型。NextToken 将每个语法词元与其右侧的语法词元相连。换言之，它将每个语法词元 与下一个语法词元相连。这样一来，语法词元之间就有了顺序。图2（左侧的抽象语法树）中以橙色显示了这种边类型。

–**NextSib:** 如上所述，抽象语法树中的边表示父子关系。编译器假定子节点之间存在顺序。例如，除法这样的二元运算符有两个子节点。在抽象语法树中，左子节点始终是分子，右子节点始终是分母。因此，有必要展示子节点的顺序。NextSib呈现了这一信息。NextSib将每个语法节点连接到其右侧的兄弟节点（图2中的蓝色边）。

– **Ref:** 在Clang的抽象语法树中，被引用变量由DecRefExpr节点表示。这些节点是终端节点，没有子节点。在图构建过程中，我们引入Ref边（图2中以粉色显示），将DecRefExpr节点连接到相应变量的定义位置。这传达了声明变量在整个图中被使用的位置。

–**ForNext, ForExec:** 循环通常在 Clang的抽象语法树中表示为 ForStmt 节点。ForStmt 节点通常有四个子节点。第一个子节点与初始化循环计数器有关，而第二个子节点用于检查循环条件，决定是否应执行下一次迭代，或循环是否已经结束。第三个子节点是复合语句，用于表示循环体。最后一个子节点是用于修改循环计数器的修饰符，例如将其递增1。这四个子节点之间的关系揭示了循环的关键特征。编译器了解这些特征，但它们并未在抽象语法树中明确呈现。我们添加了ForExec 边，将第一个子节点（计数器初始化）连接到第二个子节点（循环条件），并按照图2右侧所示，将第二个子节点连接到第三个子节点（循环体）。事实上，ForExec 边展示了流向

是否执行循环的下一次迭代。我们将 ForStmt 节点的第三个子节点（循环体）通过一种名为 ForNext 的新类型边连接到第四个子节点（循环计数器修改器）。基本上，ForNext 表示与是否需要执行循环的下一次迭代相关的信息，而 ForExec 表示循环的下一次执行。第四个子节点（循环计数器修改器）之后是第二个子节点（循环条件），它会检查是否将进行下一次迭代，或者循环是否已结束。因此，如图2所示，第四个子节点通过一条 *ForNext* 边连接到第二个子节点。

– **ConTrue, ConFalse：** 抽象语法树中的条件语句由 if 语句节点表示。if语句节点通常有三个子节点。第一个子节点表示比较条件；第二个子节点是 if 条件的主体，第三个子节点是 else 部分的主体。为表示这些信息，我们通过一种称为 ConTrue 的新边类型将第一个子节点连接到第二个子节点，以表示当 if 语句为真时，流程转向第二个子节点。另一方面，第一个子节点通过 ConFalse 边连接到第三个子节点。该边表示，如果 if 语句的条件不满足，流程将转向第三个子节点。

– **Child：** Child 边是普通的 AST 边，表示 AST 中节点之间的父子关系。

3) 加权边： 在上一节中，我们解释了如何通过引入新类型的边来扩充抽象语法树，从而更好地表示程序。不过，我们的表示中仍然缺少一些信息。例如，遇到程序中的循环时，循环体可能会执行多次；而 if 语句的各个分支执行的次数也可能不同。为了在图结构中表示这些信息，我们通过为边添加权重来扩充边。边权重根据区域和边类型计算。权重只添加到 Child 边上，因为它们是抽象语法树边，而且编译器会使用抽象语法树节点和 Child 边，将抽象语法树转换为适用于机器执行的低级表示。图2展示了部分边权重，以作说明。我们按如下方式添加权重：

– **默认边权重：**默认情况下，我们为边添加权重1，以表示代码中的每条语句仅执行一次，并且一条语句执行后，控制流会转移到下一条语句。

–**循环：** 尽管通常的语句只执行一次，但循环体中的语句会根据迭代次数执行多次。为了在我们的图表示中体现迭代次数，我们首先观察循环中的迭代次数，然后将边权重乘以该数值。此外，如果循环调度是静态的，每个线程的工作负载也会被隐式纳入考虑。具体做法是将迭代次数除以线程数。例如，如果一个循环有 100 次迭代，并在四个线程之间进行静态调度，我们大致假设每个线程执行 25 次迭代；因此，循环体内的边将具有 25 的权重。

循环体内的边将具有 25 的权重。

– **条件语句：**一个条件语句的某个分支下的节点和边与另一个分支下的节点和边，其执行次数不会相同。因此，有必要对条件语句的边进行校正。为说明分支的执行情况，我们设定一个概率。假设每个条件语句分支的概率为$_1$ $_2$。因此，条件语句中的边权重除以2。实验结果表明，这种条件语句表示方式能够提供更好的结果，边权重的进一步优化将留待未来工作。

B. 对图神经网络的益处

$ParaGraph$ 可由现有的图神经网络（GNN）模型用于下游任务；图神经网络是一类能够在图上运行的神经网络。尤其是与性能优化相关的任务。 $ParaGraph$ 既可以作为异构图使用，也可以作为同构图使用。为简单起见，在本研究中，我们将 $ParaGraph$ 视为同构图。通常，图如公式1所示，其中 V 是一组节点， E 包含一个邻接矩阵。 E 中的元素为 1 或 0，用于表示两个节点之间是否存在边。

$$G = (V, E) \tag{1}$$

我们通过加入 NextToken、NextSib 以及前文解释过的其他类型的边，扩展了用于新表示的抽象语法树。我们将 $ParaGraph$ 正式定义如下：

$$ParaGraph = (V, E, T, W) \tag{2}$$

其中， V 和 E 已在公式 1 中定义， $T \in \mathbb{Z}^+$ 表示边的类型（例如 NextToken、NextSib 等）。 $W \in \mathbb{Z}^+$ 表示权重，对于 Child 以外的任何边类型，其权重均为零。

这些额外的边和属性都是静态添加到图中的。因此，一个显而易见的问题是：我们为什么不让模型自行推理这些额外属性，而要将它们呈现在图中？一方面，将这些属性添加到图中并不会带来太大开销；另一方面，我们不希望浪费模型的资源和学习能力去识别这些显而易见的事实。因此，通过增强抽象语法树并加入前文提到的属性，我们可以更有效地训练图神经网络模型。随后，我们的消融研究进一步表明，模型能够从 $ParaGraph$ 中呈现的属性和信息中受益。

我们改进了关系图注意力网络（RGAT）[27] ，并将 $ParaGraph$ 表示作为输入来训练模型，以预测应用程序在不同加速器上的运行时间。在 RGAT 中，会针对每种边类型分别计算注意力 logits。在结果部分，我们将看到，借助 $ParaGraph$，模型的预测误差非常小，性能优于当前最先进的方法。

图 3 展示了我们基于图神经网络的运行时间预测流程的整体工作流。第一步是准备

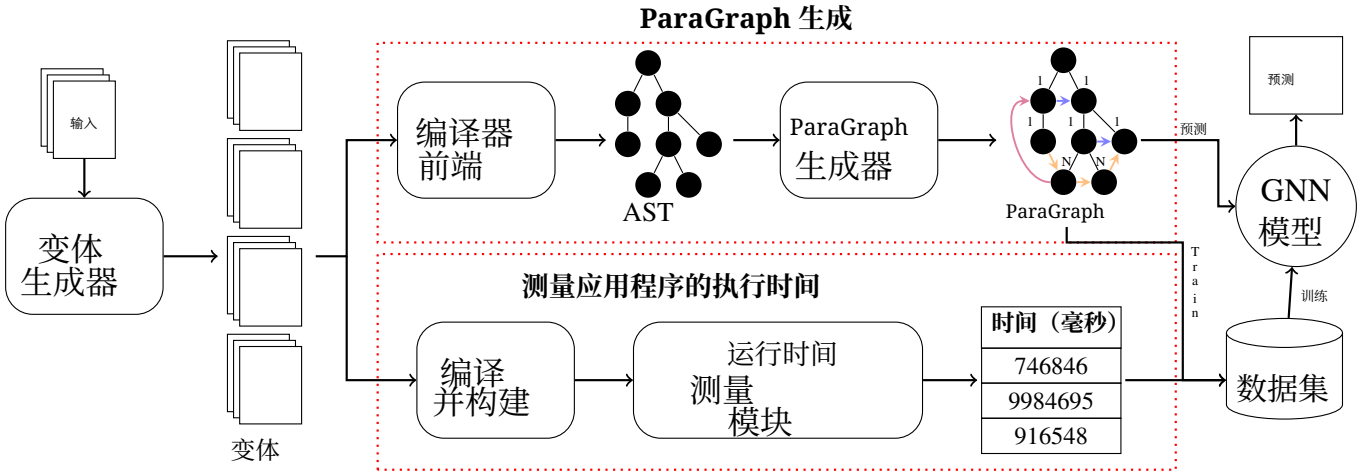


图 3：ParaGraph 的工作流程

本研究中针对每种加速器所使用的应用程序的不同变体。下一节将对此步骤作进一步详细说明。随后，使用 Clang 生成抽象语法树，并按照前文所述对其进行一系列增强，以构建 *ParaGraph*。为了训练能够预测应用程序运行时间的模型，我们需要创建一个数据集，其中包含应用程序列表及其变体，并附上相应的运行时间。因此，我们在指定的加速器上执行每个变体并测量运行时间。最后，变体的 *ParaGraph* 表示及其运行时间被用于训练图神经网络模型。除了 *ParaGraph*s 之外，我们的特征集还包括执行应用程序时使用的 team 数量和线程数。随后将这些特征输入基于图神经网络的模型，以预测运行时间。

IV. 实验与设置

我们使用两个集群和编译器来测试并评估我们的工具。实验在 ORNL 的 Summit 超级计算集群 [28]上进行，使用 LLVM/Clang（版本 13.0）和面向 GPU 的 nvptx 后端；同时也在 LLNL 的 Corona 集群 [29] 上进行，使用 LLVM/Clang（版本 15.0）和面向 GPU 的 rocm 后端。本研究中，每个集群节点仅使用一个 GPU。

与任何其他数据驱动方法一样，我们面临的主要障碍之一是缺乏可公开访问的数据集。构建可用于训练模型的数据集，是建立数据驱动成本模型的第一步。尽管有许多可公开获取的 OpenMP 基准测试套件，但针对 OpenMP GPU 卸载基准测试的工作却很少。可公开获取的 OpenMP GPU 卸载基准测试非常少。尽管有一些基准测试，例如 Rodinia 基准测试套件 [30]，，包含实现 OpenMP GPU 卸载的内核，但由于缺少不同的内核变体，我们无法使用它们。因此，第一步是创建一组利用 OpenMP GPU 卸载的基准测试。目标是涵盖广泛类别的基准测试，以覆盖一个统计模拟、线性代数、数据挖掘等领域。

A. 数据收集

我们的数据收集分为三部分：代码变体生成、图生成和运行时间收集。

应用程序	Num 内核	领域
相关系数 [31]	1	统计学
协方差	2	概率论
高斯-赛德尔方法	1	线性代数
K 近邻 [30]	1	数据挖掘
拉普拉斯方程 [32]	2	数值分析
矩阵乘法	1	线性代数
矩阵向量乘法	1	线性代数
矩阵转置	1	线性代数
粒子滤波器 [30]	7	医学成像

表 I：基准应用

1) 代码变体生成：第一步是创建不同的内核，以便收集数据。为了在计算机科学的不同领域中获得更好的预测，我们需要涵盖多个领域的应用程序内核。表 I 列出了我们为收集数据所确定的九个基准应用程序。然而，九个应用程序还不够。若统计这些应用程序中的内核数量，总共只有十七个内核。随着这一模型在未来不断发展，我们还会添加更多应用程序。

作为这项研究的概念验证，我们仅考虑以下六种变换：

- **中央处理器**：使用 OpenMP parallel for 指令的中央处理器并行内核。
- **CPU_循环折叠**：对于可折叠嵌套循环，我们使用 OpenMP parallel for collapse(2) 指令对其进行循环折叠。
- **图形处理器**：一个使用组合式 OpenMP targetteamsdistributeparallelfor 指令的图形处理器内核。所有数据均已被视为存在于图形处理器上。
- **GPU_循环折叠**：使用 OpenMP targetteams distribute parallel 指令的包含可折叠嵌套循环的 GPU 内核

对于循环折叠(2)指令。所有数据均已被视为存在于图形处理器上。

- **gpu mem**：与组合 GPU 卸载相同，但包含数据传输。
- **gpu collapse mem**：与组合 GPU 循环折叠相同，但包含数据传输。

我们使用 OpenMP Advisor 工具 [20] 生成这些内核变体。即使为每个内核创建了多个变体，我们也只有 66 个不同的内核。为了生成更多内核，我们改变所有这些内核的并行级别和所用数据。我们针对每个应用程序修改这些参数，为数据集生成至少 2000 至 3000 个不同的内核。将所有应用程序中的内核合并后，数据集中几乎有 26,000 条独特的信息。这一步与运行内核所使用的架构无关。

2）ParaGraph 生成：在流水线的这一步中，我们为上一节生成的内核创建 $ParaGraphs$。对于要训练的每个内核，数据集应包含两部分——图神经网络图和运行时间。训练的第一步是为每个内核生成 $ParaGraph$。由于 $ParaGraph$ 基于抽象语法树，因此会编译每个内核并生成抽象语法树。为了生成 $ParaGraph$，我们会遍历每棵抽象语法树，并按照第三节中的说明添加额外的边和权重。边类型传达编译器视角的信息，而边权重则揭示执行流的信息，更重要的是，揭示内核的热点，例如循环。通常，ForLoop 内的边具有更高的权重，以表示循环的迭代次数。

3) 运行时间采集：下一步，我们收集每个内核的实际运行时间。此过程完全取决于该内核所针对的硬件。该过程的复杂性取决于我们是否有机会访问高性能计算集群的计算节点。这一步骤中的失败很难通过交互方式处理，我们必须等待单个作业失败后，才能意识到错误。因此，在提交作业以在这些集群上运行时，我们必须格外谨慎。

使用 OpenMP Advisor，我们能够生成已经收集内核运行时间信息的代码。它通过在内核调用前后分别包含 gettimeofday函数调用来实现这一点；该函数获取以经过的秒数和微秒表示的系统时钟时间，然后计算两次时间之间的差值，以获得以微秒为单位的运行时间。OpenMP Advisor

平台	#数据要点	运行时间范围 (ms)	Std. Dev.
峰会			
IBM POWER9（中央处理器）	13,023	[0.23 - 736,798]	48,502
NVIDIA V100（图形处理器）	26,040	[0.035 - 30,174]	3,708
Corona			
AMD EPYC7401 (中央处理器)	17,681	[0.024 - 291,627]	16,942
AMD MI50（图形处理器）	26,668	[0.448 - 46,913]	4,828

表 II：在每个加速器上收集的数据点。

还帮助我们生成了每个内核的全部六种变体，并且我们能够通过改变并行级别和所使用的数据，为每个应用程序生成多个变体。我们在本地集群上构建了每个内核，以确保使用自动化工具生成内核不会导致我们的代码出错。在成功构建所有内核后，我们在本地集群上运行每个内核，以验证我们收集的数据。一旦确认所有内核都能在本地集群上成功构建和运行，我们便创建作业，在 Summit 和 Corona 集群上构建并运行这些内核。不过，我们对这些集群的访问权限有限，而且由于节点故障或时间限制，我们的作业无法长时间运行。因此，我们不得不提交多个作业，并跟踪它们在数据收集方面的成功或失败。

获得每个内核的运行时间后，我们便可以轻松地将其与相应的 $ParaGraphs$ 结合起来，为我们的数据集收集数据点。表 II 展示了从每个加速器收集的数据点统计信息。

B. ParaGraph 模型

在 Summit 和 Corona 集群上收集到足够的数据后，我们便开始训练模型。我们使用 RGAT [27] 作为卷积层，实现了一个基于图神经网络的神经网络模型。模型采用 Pytorch-Geometric 库实现，以均方误差作为损失函数，并使用 Adam [33] 作为优化器。为了嵌入图，模型使用三个基于 RGAT 的图卷积层，随后接入两个采用 ReLU 激活函数的全连接层。如前所述，team 数量和线程数量被视为两个额外特征。我们使用另一全连接层来嵌入这两个特征。最后，将图的嵌入表示与这两个特征拼接起来，并输入最后一个全连接层，以预测运行时间。

边权重和另外两个特征使用 MinMaxScaler 进行归一化。数据集按照 a9:1 的比例划分为训练集和验证集。

V. 结果

在以下小节中，我们将解释用于评估 $ParaGraph$ 模型的指标。

A. 评估指标

为评估 $ParaGraph$ 的性能，我们使用$RMSE$，即均方根误差（公式 3）。

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N}(x_i - \hat{x}_i)^2}{N}} \tag{3}$$

其中，x_i 表示一个数据点的运行时间（以微秒为单位），$\hat{x}_i$ 是由 $ParaGraph$ 预测的运行时间，而 N 是样本总数。

由于不同平台之间的运行时间范围存在差异，因此也考虑了 $RMSE$ 的归一化版本。归一化$RMSE$ 是通过将 $RMSE$ 除以最小运行时间与最大运行时间之间的距离来计算的。我们还使用相对误差（即将绝对误差除以运行时间范围）来报告误差率。

平台	均方根误差（毫秒）	归一化均方根误差
Summit		
IBM POWER9（中央处理器）	4325	6×10^{-3}
NVIDIA V100（图形处理器）	280	9×10^{-3}
Corona		
AMD EPYC7401（中央处理器）	968	4×10^{-3}
AMD MI50（图形处理器）	510	1×10^{-2}

表 III：实验结果

B. 实验结果

表 III展示了每种加速器的实验结果。我们使用了NVIDIA V100、AMDMI50，以及具有 22 个核心的IBMPOWER9和具有24 个核心的 AMDEPYC 7401。如图所示，$RMSE$ 值的范围为280（毫秒）到 4325（毫秒）。之所以不同加速器具有不同的 $RMSE$ 值，例如POWER9 加速器的4325（毫秒），是因为不同加速器之间的运行时间离散程度不同。表II中的标准差让我们能够了解所收集数据的离散程度。接下来是归一化-均方根误差，它与运行时间的范围无关；我们看到，$ParaGraph$在不同加速器上的误差基本相同，因此可以应用于不同的加速器。

为了进一步分析我们的结果，我们计算了每个 10 秒分箱的相对误差。图 4 展示了所有四种加速器的 11 个分箱。除最后一个分箱外，每个分箱的范围均为 10 秒。图中显示，不同分箱和加速器的相对误差都很小（小于%10），这意味着我们的 $ParaGraph$ 模型在不同运行时间范围内表现稳定。

此外，我们在图 5 中分析了模型在训练过程中的稳定性。该图展示了四种加速器的验证$RMSE$ 。在最初的几个轮次中，$ParaGraph$ 模型不太稳定，导致 $RMSE$出现波动；然而，随着模型进一步训练，它能够从表示中提取并学习特征，降低每个轮次的 $RMSE$ 值，并最终收敛。

最后，我们计算每种应用程序的平均相对误差，以查看 $ParaGraph$ 模型是否能够对所有类型的应用程序进行低误差率预测。图 6 展示了每种应用程序的误差率。由此可见，该模型确实能够对各种应用程序进行准确预测，

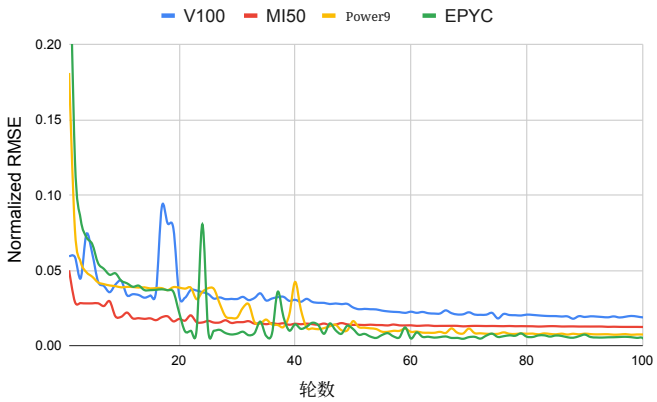


图 5：每轮次的归一化均方根误差。

从而实现较低的误差率。因此，该模型并未偏向某一种应用程序。在 AMD MI50 图形处理器上，Laplace 数据在采集过程中已损坏。因此，本研究和训练过程均未包含这些数据。

C. 消融实验

$ParaGraph$ 表示，如第三节所述，是它是在AST 的基础上应用一系列重大增强构建而成的。在本节中，我们量化这些增强的影响。首先，我们考虑不包含额外边和权重的AST，并将其称为原始抽象语法树。然后，我们向 AST 添加额外的边和边类型，并将其称为增强抽象语法树。最后，我们有 $ParaGraph$ ，它同时包含额外的边和边权重。表IV展示了消融研究的结果。我们看到，对于所有四种加速器，原始抽象语法树产生的误差最高。添加新边并引入新的边类型（增强抽象语法树）在一定程度上改善了预测。例如，通过添加这些新边，V100 的$RMSE$ 从 2114 (ms) 降至 786 (ms)。

我们提出的程序表示的关键特征之一是边权重。边权重传达了有关不同区域出现频率的重要信息。

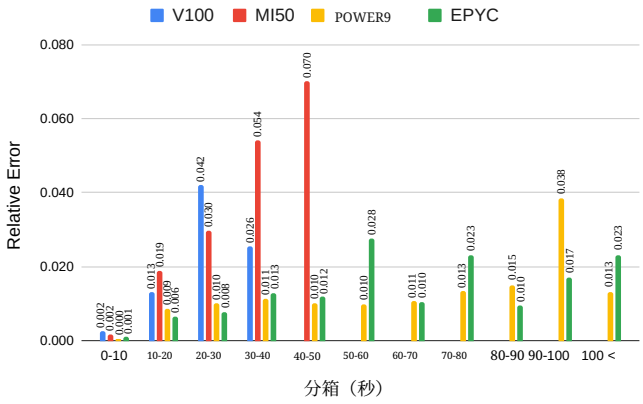


图 4：每 10 秒分箱的预测误差。

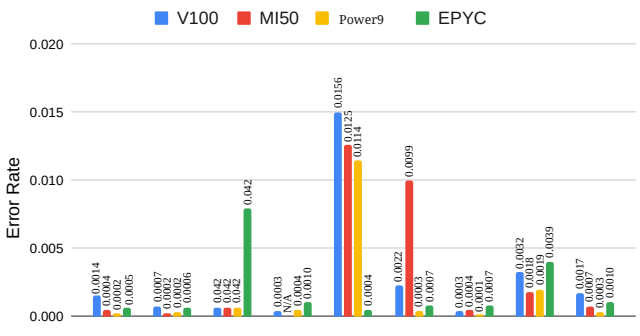


图6：每个应用程序的误差率。

抽象语法树执行。因此，我们看到在 $RMSE$ 添加边权重时有相当不错的提升。例如，V100 的$RMSE$ 进一步提升至 280 (ms)。

平台	Raw AST	Aug AST	ParaGraph
Summit IBM POWER9（中央处理器） NVIDIA V100（图形处理器）	 27593 2114	 26860 786	 **4325** **280**
Corona AMD EPYC7401 （中央处理器） AMD MI50（图形处理器）	 11911 2888	 9633 1177	 **968** **510**

表 IV：有边权重和无边权重训练的均方根误差。我们进一步分析了添加边及其权重对 $ParaGraph$ 模型训练过程的影响。

图7展示了模型在训练过程中的表现。该图描绘了在 MI50 加速器上，原始抽象语法树、增强抽象语法树和 $ParaGraph$ 每轮的 $RMSE$ 值。

仅使用未经任何增强的原始抽象语法树（即只有一种边类型）时，模型能够学习到应用程序的一些特征，并逐轮降低$RMSE$；然而，这种 $RMSE$的下降并不显著。增强抽象语法树包含 8 种不同类型的边。我们发现，添加这些边会使模型的训练过程不稳定。在最初几轮中，模型难以学习节点之间的不同关系。然而，经过若干轮后，模型的预测最终趋于稳定，均方根误差达到 1177（毫秒）。在为增强抽象语法树中的边添加权重，从而构建 $ParaGraph$ 后，我们看到模型的预测进一步改善。尽管验证 $RMSE$ 在最初几轮中有所波动，但最终以显著较小的误差收敛。

D. 与最先进工具的比较

据我们所知， $COMPOFF$ [2] 是唯一一个先进的 OpenMP GPU 卸载成本模型。我们进一步将 $ParaGraph$ 的结果与

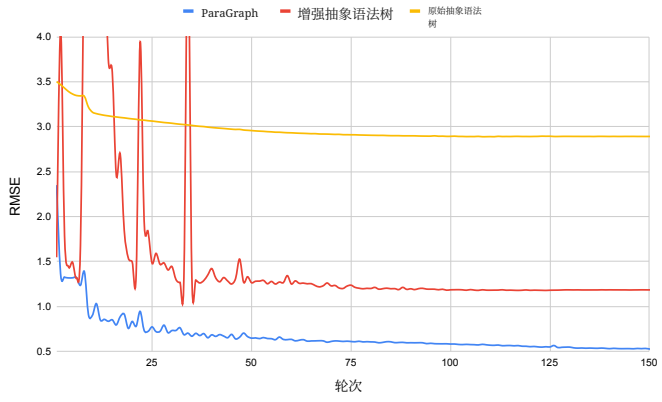


图 7：在 MI50 数据点上训练图神经网络模型期间验证集的均方根误差。

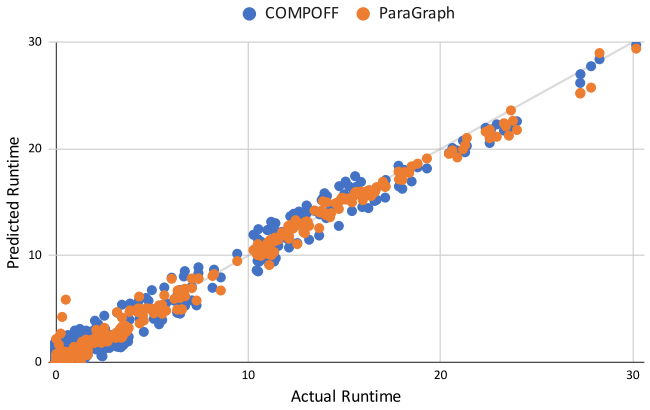


图8：对于每个数据点，ParaGraph和 COMPOFF 在 NVIDIA V100 上预测运行时间的比较。

$COMPOFF$ 进行了比较。如第二节-D所述，OpenMP Advisor 使用 $COMPOFF$ 来预测 OpenMP内核的运行时间。

虽然 OpenMP Advisor 最终需要一种能够预测所有可能底层架构的成本模型，但$COMPOFF$目前仅适用于图形处理器执行。相比之下， $ParaGraph$ 无论底层架构如何，都能对应用程序运行时间建模。在我们的实验中，$ParaGraph$ 用于预测中央处理器和图形处理器上的运行时间。因此，我们仅使用从 NVIDIA V100 图形处理器收集的数据点来比较 $ParaGraph$ 和$COMPOFF$。

对于 $ParaGraph$和 $COMPOFF$，图 9 展示了实际数据与预测数据之间的强相关性。 $COMPOFF$的结果以蓝点表示，而 $ParaGraph$ 的结果以橙点表示。尽管两者表现都相当出色， $ParaGraph$ 在预测运行时间与实际运行时间之间展现出强得多的相关性。在图 8 中，对于运行时间较短的内核，$COMPOFF$（蓝色）的误差率略高；但随着运行时间增加，误差率会降低（甚至完全消失）。然而，对于所有内核， $ParaGraph$（橙色）的误差率都显著更低。在 AMD 图形处理器上也观察到了类似结果。由于 $COMPOFF$不支持 CPU，因此我们无法比较在 CPU 上进行预测的结果。这是 $ParaGraph$ 相较于$COMPOFF$的一项重要优势。

E. 讨论

$ParaGraph$ 能够对应用程序的执行进行静态建模。我们的实验结果表明， $ParaGraph$ 能够有效预测应用程序在不同加速器上的运行时间。另一方面，有些应用程序的运行时间行为并不稳定，也就是说，它们在不同的执行过程中会表现出不同的行为。这些应用程序无法进行静态建模。因此，仅使用静态特征来预测其运行时间颇具挑战性。解决这一挑战的一种方法是开发一种结合静态特征和动态特征的混合方法（例如，性能

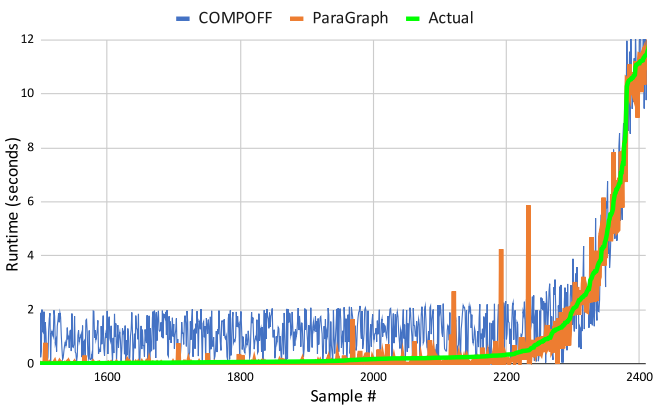


图 9：ParaGraph 和 COMPOFF 在 NVIDIA V100 上的预测与实际运行时间对比。

应用程序的计数器）。然而，收集应用程序的动态特征需要执行应用程序，这可能会增加预测成本。

六、结论与未来工作

在本文中，我们提出了 *ParaGraph*。一种表示高性能计算内核的新颖方法。 *ParaGraph* 基于抽象语法树；它向抽象语法树中融入了额外特征，以更好地表示应用程序。特别是， *ParaGraph* 通过向抽象语法树添加新的边，纳入了编译器的一些洞见。*ParaGraph* 还通过向相应的边添加权重，增加了有关循环和条件语句执行方式的信息。我们在针对四种不同加速器的一组应用程序上评估了 *ParaGraph* 。它实现了较低的误差率，突出了 *ParaGraph* 的有效性。 *ParaGraph* 的主要优势之一是其独立于底层硬件架构。因此，与之前的方法不同，它可以应用于更广泛的架构。

在这项工作中，我们使用 *ParaGraph* 来预测给定内核在OpenMPGPU 卸载问题中的执行时间。未来，我们计划探索和分析 *ParaGraph*如何帮助其他 OpenMP 优化策略，例如预测 SIMD 步长、调度策略、循环块大小等。另一个值得探索的研究方向是使用 *ParaGraph* 并捕获其他并行编程模型（如 OpenACC、Kokkos、HIP 等）中的并行性，以便为各种优化问题构建成本模型。

参考文献

[1] L.Dagum 和 R.Menon，“OpenMP：共享内存编程的行业标准 API”，*IEEE*《计算机科学与工程》，第5 卷，第 1 期，第 46–55页，1998 年。[2] A. Mishra、S. Chheda、C. Soto、A. M. Malik、M. Lin 和 B. Chapman，“COMPOFF：使用机器学习预测 OpenMP 卸载成本的编译器成本模型”，载于 *2022*年*IEEE*国际并行与分布式处理研讨会工作坊（*IPDPSW*），*2022*年*5*月*30*日至*6*月*3*日。IEEE，2022 年。[3] A. Ciurumelea、S. Proksch 和H.C. Gall，“使用神经语言模型为 Python提供注释补全建议”，载于 *2020*年*IEEE*第*27*届国际软件分析、演化与再工程会议（*SANER*）。IEEE，2020 年，第 456–467 页。[4] C. Cummins、H. Leather、P. Bramble、用于数据流分析和编译器优化的基于图的程序表示”，载于《机器学习国际会议》。PMLR，2021年，第 2244–2253 页。[5] V. Raychev、M.Vechev和 E. Yahav，“使用统计语言模型进行代码补全”，载于《第 35 届ACM SIGPLAN 编程语言设计与实现会议论文集》，2014年，第 419–428 页。[6] M. Allamanis、H. Peng 和C. Sutton，“用于源代码极端摘要的卷积注意力网络”，载于《机器学习国际会议》。PMLR，2016 年，第 2091–2100 页。[7] M. Allamanis、M.Brockschmidt和M. Khademi，“学习使用图表示程序”，arXiv预印本 arXiv:1711.00740，2017 年。[8] M.A. Heroux、R.Thakur、L.McInnes、J.S. Vetter、X.S. Li、J. Ah- erns、T.Munson和 K.Mohror，“ECP软件技术能力评估报告”，橡树岭国家实验室（ORNL），美国田纳西州橡树岭，技术报告，2020年。[9] Y.Kim、J. Lee、J.-S. Kim、H. Jei 和 H.Roh，“用于深度学习加速的多GPU内存优化综合技术”，《集群计算》，第23卷，第 3 期，第 2193–2204 页，2020 年。[10] K. Rzadca、P.Findeisen、J. Swiderski、P. Zych、P.Broniek、J. Kusmierek、P.Nowak、B. Strack、P.Witusowski、S. Hand 等，“Autopilot：Google 的工作负载自动扩缩容”，载于《第十五届欧洲计算机系统会议论文集》，2020 年，第 1–16 页。[11] M.Mirka、G. Sassatelli和 A.Gamati´e，“用于OpenMP工作负载动态控制的在线学习”，载于《2020年第九届现代电路与系统技术国际会议（MOCAST）》。IEEE，2020 年，第 1–6 页。[12] N. Denoyelle、B. Goglin、E. Jeannot 和 T. Ropars，“NUMA架构中的数据与线程放置：一种统计学习方法”，载于《第 48届并行处理国际会议论文集》，2019 年，第 1–10 页。[13] J. Alcaraz、A. TehraniJamsaz、A. Dutta、A. Sikora、A. Jannesari、J. Sorribes 和 E. Cesar，“使用平衡数据集预测 OpenMP 区域中的线程数”，《Computing》，第 1–19 页，2022 年。[14] A. Dutta、J. Alcaraz、A. TehraniJamsaz、A. Sikora、E.Cesar 和 A. Jan- nesari，“使用图神经网络对 OpenMP 循环进行基于模式的自动调优”，载于《2022 年IEEE/ACM 科学应用人工智能与机器学习国际研讨会（AI4S）》。IEEE，2022年，第 26–31 页。[15] A.M. Malik，“为编译器优化问题自动生成静态特征”，载于《澳大拉西亚人工智能联合会议》。Springer，2011年，第 769–778 页。[16] S. Dublish、V. Nagarajan和N.Topham，“使用机器学习在GPU中平衡线程级并行性与内存系统性能的 Poise”，载于《2019 年 IEEE高性能计算机体系结构国际研讨会（HPCA）》。IEEE，2019年，第 492–505 页。[17] A. Iranfar、W.S. De Souza、M. Zapater、K. Olcoz、S. X.de Souza和 D. Atienza，“用于多核服务器中时变应用程序吞吐量估计的基于机器学习的框架”，载于《2019年 IFIP/IEEE 第 27 届超大规模集成电路（VLSI-SoC）国际会议》。IEEE，2019年，第 211–216 页。[18] D. Grewe 和M.F. O’Boyle，“使用 OpenCL 的异构系统静态任务划分方法”，载于《编译器构造国际会议》。Springer，2011 年，第 286–305页。[19] H. Sayadi，“使用机器学习技术预测异构复合核心架构上多线程工作负载的能效”，arXiv 预印本 arXiv:1808.01728，2018 年。[20] A. Mishra、A. M. Malik、M. Lin 和 B.Chapman，“OpenMP Advisor”，arXiv 预印本 arXiv:2301.03636，2023 年。[21] R. B. Roy、T. Patel、V.Gadepally和 D. Tiwari，“BLISS：使用多样化轻量级学习模型池对复杂应用程序进行自动调优”，载于《第 42 届 ACM SIGPLAN编程语言设计与实现国际会议论文集》，2021年，第 1280–1295 页。[22] S. Falkner、A.Klein 和 F. Hutter，“BOHB：大规模场景下稳健且高效的超参数优化”，载于《机器学习国际会议》。PMLR，2018年，第 1437–1446 页。[23] H. Menon、A.Bhatele 和 T. Gamblin，“使用贝叶斯优化自动调优高性能计算应用中的参数选择”，载于《2020 年IEEE 国际并行与分布式处理研讨会（IPDPS）》。IEEE，2020 年，第 831–840页。[24] Y. Liu、W. M.Sid-Lakhdar、O. Marques、X.Zhu、C. Meng、J. W.Demmel 和 X. S. Li，“GPTune：用于自动调优的多任务学习

百亿亿次级应用”，载于 第*26* 届 *ACMSIGPLAN* 并行编程原理与实践研讨会论文集，2021 年，第 234–246 页。[25] X. Wu、M.Kruse、P. Balaprakash、H.Finkel、P.Hovland、V. Taylor 和 M. Hall，“使用贝叶斯优化，通过 llvm Clang/polly 循环优化编译指示对polybench 基准进行自动调优”，*Concurrency and Computation: Practice and Experience*，第34卷，6683，2022 20，第 no.p. e 。[26] M. Allamanis，“程序分析中的图神经网络”，载于 图神经网络：基础、前沿与应用。Springer，2022年，第 483–497页。[27] D. Busbridge、D. Sherburn、P.Cavallo 和 N. Y. Hammerla，“关系图注意力网络”，*arXiv* 预印本*arXiv:1904.05811*，2019 年。[28] ORNL，“橡树岭领导计算设施——Summit 超级计算集群”，2017 年。 [在线]。可用地址：https://www.olcf。ornl.gov/summit/ [29] L. L. N. 实验室，“LLNL - Corona”，2019 年。 [在线]。可访问：https://hpc.llnl.gov/hardware/compute-platforms/corona [30] S. Che、M. Boyer、J. Meng、D. Tarjan、J. W. Sheaffer、S.-H. Lee 和 K. Skadron，“Rodinia：异构计算基准测试套件”，载于 *2009 IEEE* 国际工作负载特征分析研讨会（*IISWC*）。Ieee，2009年，第44–54页。[31] P. Schober、C. Boer 和 L. A. Schwarte，“相关系数：适当的使用与解读”，麻醉与镇痛，第126卷，第5期，第1763–1768页，2018年。[32] A. K. Mitra，“用于求解拉普拉斯方程的有限差分法”，艾奥瓦州立大学航空航天工程系，2010年。[33] D. P. Kingma 和 J. Ba，“Adam：一种随机优化方法”，*arXiv* 预印本 *arXiv:1412.6980*，2014年。