arXiv:2304.08880v1 [cs.AR] 18 Apr 2023

NPS：使用图神经网络实现精准程序采样的框架

Yuanwei Fang[1], Zihao Liu[1], Yanheng Lu[1],Jiawei Liu[2],Jiajie Li[3],Yi Jin[4],Jian Chen[1],Yenkuang Chen[1],Hongzhong Zheng[1],和 Yuan Xie[1]

[1]达摩院，阿里巴巴集团[2]伊利诺伊大学厄巴纳-香槟分校计算机科学系[3]康奈尔大学电气与计算机工程系[4]复旦大学电气工程系{*yuanwei.fang, zihao.liu, yanheng.lyh, j.chen,y.k.chen, hongzhong.zheng, y.xie*}*@alibaba-inc.com*

摘要

随着摩尔定律走向终结，现代处理器对快速架构创新的需求日益增长，例如采用 *RISC-V* 自定义扩展，以持续提升性能。程序采样是微处理器设计中的关键步骤，它会为工作负载仿真选取具有代表性的模拟点。数十年来，*SimPoint* 一直是事实上的主流方法，但其依赖基本块向量（*BBV*），表达能力有限，需要耗时数月进行人工调优，从而阻碍了快速创新和敏捷硬件开发。本文介绍神经程序采样（*NPS*），这是一种利用图神经网络的动态快照学习执行嵌入的新型框架。*NPS* 使用 *AssemblyNet* 进行嵌入生成，利用应用程序的代码结构和运行时状态。*AssemblyNet* 是 *NPS* 的图模型和神经架构，能够捕捉程序在数据计算、代码路径和数据流等方面的行为。*AssemblyNet* 通过数据预取任务进行训练，以预测连续的内存地址。

在实验中，*NPS* 的性能最高比 *SimPoint* 提升了 *63%*，将平均采样误差降低了 *38%*。此外，*NPS* 展现出很强的鲁棒性，并具有更高的准确性，从而减少了高昂的精度调优开销。此外，在代码行为学习方面，*NPS* 比最先进的图神经网络方法具有更高的准确性和更强的泛化能力，能够生成高质量的执行嵌入。

1. 引言

在后摩尔定律时代，计算机体系结构创新已成为现代微处理器持续提升性能的关键途径 [17, 32]。RISC-V 指令集日益普及并得到广泛采用 [60]，促使众多微处理器设计融入新颖的体系结构特性 [18, 23, 25], ，例如定制指令和加速单元，从而拥抱快速创新。此外，计算机体系结构的黄金时代 [32]需要快速原型开发和敏捷的硬件开发。

详细的周期级仿真是计算机体系结构中的关键工具，但由于仿真速度缓慢，无法处理大规模应用程序。例如，在生产级模拟器上运行一个

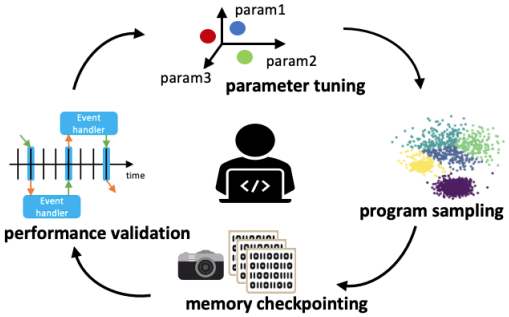


图 1：迭代式精度调优。

包含 1 万亿条指令的应用程序需要约三年时间 [6]。为了降低模拟成本，设计人员会进行程序采样，选取具有代表性的执行区间，即所谓的模拟点，以此近似整个应用程序的仿真 [47, 29, 26, 62, 55]。程序采样的准确性会直接影响微处理器设计：它决定了对各种拟议功能的评估结果。然而，准确的模拟点成本高昂，需要投入大量精力对设置进行精度调优 [47, 55]。准备准确模拟点的标准流程如图1所示，其中要经过多轮采样、检查点和验证，以调整并提高准确性。这种迭代调优需要人在回路，耗时很长，往往需要数月。过去，大公司依靠既有的知识库和丰富经验，降低其下一代设计的精度调优成本。然而，缺乏程序采样经验积累的新入行者（例如 RISC-V 初创公司）则要承担高昂的前期成本。因此，准确程序采样的高昂开销阻碍了微处理器开发中的快速创新和敏捷方法。

几十年来，SimPoint [47, 29] 一直是最先进的程序采样方法。经过仔细调优，SimPoint 在规模为数千万或数亿条指令的粗粒度模拟点上能够达到较高的准确性[55]。然而，由于其特征向量的分辨率有限，难以表征执行行为，SimPoint 的表现不稳定，准确性也缺乏稳健性 [62]。此外，设计人员往往会转而采用

平台	类别	速度	准确性
gem5 模拟器 [16]	基于事件的	~5K 指令/秒	中等
软件事务仿真 [6, 5]	事务-级	1K-50K 指令/秒	中等
现场可编程门阵列 RTL 仿真化 [1, 33]	RTL 级	50K-1M 指令/秒	high
软件 RTL 仿真 [56, 27, 2]	RTL 级	<10 指令/秒	high

表 1：性能建模平台的速度与准确性。

小规模或中等规模的模拟点（例如，1 万至 1000 万条指令），以满足低层级仿真对更高建模准确性的需求 [39, 51, 56, 62, 5]。例如，混合仿真采用基于事件的仿真与 RTL 级仿真相结合的方式 [39, 51], , 并针对 RTL 模型使用细粒度模拟点，以弥补仿真速度缓慢的问题。表 1 列出了大型 CPU 设计（数十亿个门）中常见性能评估和验证平台的速度与准确性。由于特征向量对分辨率的要求提高，SimPoint 难以选取小规模模拟点。本文聚焦于准确的程序采样，尤其是在细粒度场景下，力求提高准确性并减少调优工作。

准确程序采样的关键挑战在于设计一种高分辨率的执行特征刻画向量表示，在人工智能/机器学习领域中称为执行嵌入。SimPoint 使用基本块向量（BBV） [52] 作为执行嵌入。然而，BBV 在捕捉执行行为方面存在一些不足。首先，它缺少代码拓扑结构信息，因为没有对基本块连接关系进行编码。其次，BBV 不考虑指令语义，对不同类型和行为的指令一视同仁，因此无法保留基本块之间的相似性。第三，BBV 无法捕捉动态值。例如，它无法区分数据相关内存访问不同的顺序代码。最后，BBV 是针对单个应用程序定义的，缺乏跨应用程序的通用性。

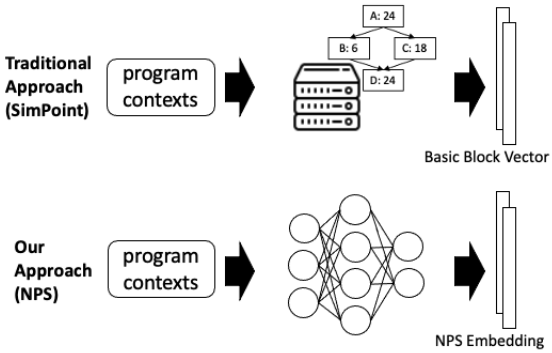


图 2：传统方法与我们的方法。

我们提出神经程序采样（NPS）来应对这些挑战，借助能够学习代码行为的图神经网络模型，为准确的程序采样生成高质量的执行嵌入。图 2 展示了这一思路：我们训练了一个名为 AssemblyNet 的“神经微处理器”，它能够利用深度学习执行代码，并将执行区间映射为嵌入。相比之下，传统方法在商用 CPU 上运行，以分析人工设计的特征向量（BBV）。NPS 嵌入能够捕捉代码拓扑结构，例如控制流图和数据流图、指令语义以及执行过程中的动态上下文，并具有跨应用通用性。因此，NPS 能够补充 BBV，借助通用化表示，以细粒度刻画代码行为。将 AI 应用于 NPS 时，我们利用的关键观察包括：**1**：设计者愿意牺牲机器资源来提高准确性，并减少高昂的调优工作；**2**：用于基准测试采样的嵌入生成只需执行一次；**3**：检查点和精度调优开销能够大幅摊销 AI 的计算成本；以及**4**：可以开发分层采样技术，在运行时开销与采样精度之间灵活权衡。

在本研究中，我们使用多种误差指标，在 SPEC2006 整数测试集上评估 NPS。结果表明，NPS 能够提供稳健且准确的程序采样，其表现远胜于最先进的 SimPoint 方法。NPS 为微处理器设计中的性能建模、模拟和基准测试开辟了新的机遇。本文的具体贡献包括：

- 我们提出神经程序采样（NPS），这是一种新颖的框架，可为准确的程序采样提供高分辨率的执行嵌入，并介绍图构建、应用程序跟踪、图快照、图神经网络和序列聚合等关键设计。
- 我们提出 AssemblyNet，这是一种图神经网络图模型和神经架构，能够高精度地学习代码执行的控制流和数据流行为。
- 在多种通用基准测试上对 NPS 进行评估。NPS 的表现比最先进方法最高提升 63%，平均误差降低 38%。
- 准确率鲁棒性评估表明，NPS 在各种设置下均能稳定提升准确性，从而减少对高成本精度调优的需求。
- 在一项具有挑战性的数据预取任务上评估 AssemblyNet。与最先进的图神经网络方法相比，AssemblyNet 将准确性从 18.2% 提升至 87.5%，展现出强大的代码执行行为捕获能力。

本文其余部分安排如下。首先，第2 节介绍背景。第3节描述 NPS 框架，包括图模型、神经架构和嵌入生成等关键设计。随后，我们在第5 节使用第4节所述的配置，在各种通用基准测试上评估 NPS。最后，我们讨论相关工作（第6节），并总结全文（第7节）。

2. 背景

程序采样[47, 62, 55] 将应用程序划分为一系列固定长度的执行区间，程序采样器利用一小组代表性模拟点的特征向量来选择这些区间。借助内存快照，模拟点可以在精度验证平台上运行，该平台通常是周期级软件模拟器[16, 6, 5], 或 RTL 仿真平台 [56, 27, 2, 1, 33]。为了提高采样精度，设计人员需要花费大量时间尝试各种采样器设置[47, 55] 并手动优化模拟点。这一过程既耗时又重复，需要进行多轮调整。SimPoint [47, 29] 是当前最先进的采样器，使用 K 均值聚类和贝叶斯信息准则来确定模拟点。

基本块向量 [52, 47]是表征程序执行的一种常用方式。在 SimPoint 中，BBV 作为特征向量来编码执行区间，从而支持相似度度量。基本块是一段从入口到出口完整执行的代码。每个维度表示应用程序在特定时间段内进入某个基本块的次数。代码块频率是一种高级执行信息，缺少详细的程序上下文。因此，BBV 能够捕捉程序行为随时间的变化，但粒度较粗。

图神经网络 [30, 35, 59] 从丰富的图数据样本中学习如何计算来自邻居节点的消息（公式 1）、如何聚合消息（公式 2），以及如何更新节点特征（公式 3）。因此，GNN 不依赖特定拓扑，可以泛化到各种未见过的图。具体而言，图可以表示为 $G = (V, E)$，其中 V 表示节点，E 表示边。在 GNN 中，我们将 E_i 记为类型 i 的边集合，将 $e_{u,v}$ 记为一条从节点 u 指向 v 的边。节点 v 在边类型 i 下的邻居定义为 $N_i(v) = \{u|(u, v) \in E_i\}$。我们用 h_v^k 表示第 k 层节点 v 的特征（$k = 0$ 表示初始节点特征）。我们可以根据公式 1 计算节点 v 在第 $k-1$ 层接收的边类型 i 的消息，其中 h_u^{k-1}是第 $k-1$ 层节点 u 的特征，f 是具有可学习参数 θ_i的非线性函数。

$$m_{i,v}^{k-1} = \sum_{u:N_i(v)} f(h_u^{k-1}, \theta_i) \tag{1}$$

由于每种边类型都有关联的可学习参数，图神经网络会聚合来自所有边类型的消息（公式 2）。其中g 是聚合函数（例如，逐元素求和）。

$$\tilde{m}_v^{k-1} = g(m_{i,v}^{k-1}|\text{for all edge type } i) \tag{2}$$

最后，我们使用可学习函数r更新第 k 层的h_v^k 节点特征h。

$$h_v^k = r(\tilde{m}_v^{k-1}, h_v^{k-1}, \alpha) \tag{3}$$

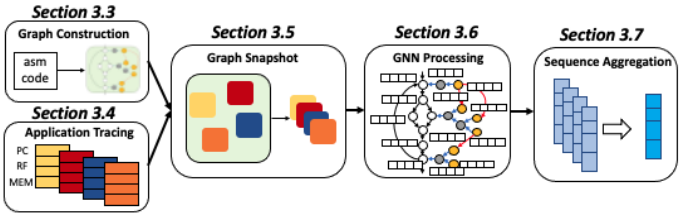


图 3：NPS 执行嵌入生成概览。

3. 神经程序采样

3.1. 概述

随着图神经网络领域近期取得突破 [30, 35,59], ，我们可以使用图神经网络对代码执行进行建模，在指令流之外，还能编码指令语义、代码结构和动态状态等丰富上下文。本文提出一种基于图神经网络的新型程序采样框架——神经程序采样（NPS），以便对细粒度模拟点进行准确采样，如图3所示。首先，NPS 根据应用程序的汇编代码构建其 AssemblyNet 图（**第3.3节**）。在运行时，将应用程序的动态状态和指令流收集为跟踪记录（**第3.4节**）。与此同时，根据跟踪记录从 AssemblyNet 中提取包含动态状态的图快照（**第3.5节**）。这些图快照是 AssemblyNet 图的子图，其初始节点值由动态状态赋予。随后，将这些快照输入图神经网络模块（**第3.6节**）以生成图嵌入。该模块会预加载一个经过训练的图神经网络模型（**第3.2节和第3.6节**），该模型能够推理代码执行过程。最后，NPS 汇总图嵌入，为相应区间生成最终的执行嵌入（**第3.7节**）。NPS 为所有执行区间生成嵌入后，经典的基于聚类的采样器[47, 29] 会选出最具代表性的模拟点。

3.2. 训练任务

NPS 在训练图神经网络模型（AssemblyNet）时采用自监督学习。这类似于 Transfomer [58],通过预测句子中的掩码词进行自监督训练，从而为下游自然语言处理任务提供词嵌入。这里，我们使用数据预取任务训练 AssemblyNet，以预测未来的内存访问。完成这项任务不一定需要标注训练数据；相反，可以通过应用程序跟踪（第 3.4节）轻松获取训练数据。数据预取任务使 AssemblyNet 能够推理并学习执行规则，例如代码路径选择、数据流传播和值计算。对于程序采样，NPS 会移除特定于任务的部分，并保留图神经网络模型，将执行区间编码为图嵌入。

数据预取任务要求首先预测正确的代码路径。程序的控制流图在经过少数几次内存引用后，就会迅速分化出数十条代码路径。

例如，400.perlbench 中一个典型的代码片段，仅经过 5 次内存引用后就会分支到 30 多条路径。AssemblyNet 需要按照正确顺序，准确预测代码路径上的所有内存地址，即每一位都必须正确。形式上，我们将数据预取任务定义如下。我们将指令 I 处的程序表示记为 S_I，其中 $S_I = f(G_I, V_I)$。这里，G_I 是图快照（第3.5节），V_I 表示动态状态集合（第3.4节）。我们希望训练一个神经网络（第3.6节），使其学习两个函数：f 和 h，从而在给定 G_I 和 V_I的情况下，预测指令 I 之后连续 D_I 个内存地址：$h(S_I) = \{addr_1, addr2, ..., addr_{D_I}\}$，其中 $addr_i$是第 i^{th} 次内存引用的 64 位地址。完成此任务意味着模型具备捕捉代码执行基本要素的能力，例如解析分支、激活路径、计算值和跟踪数据依赖。对于程序采样，我们只需要 f 对程序的执行进行编码。

3.3. AssemblyNet 图模型的离线构建

汇编代码是所有高级编程语言的通用基础，因此 AssemblyNet 基于汇编代码构建应用程序的图模型。该图模型捕获代码结构，并通过图快照纳入动态状态。它为图快照提供全局模板，而图快照则是融合了动态状态的子图。为了构建图模型，NPS 首先根据汇编代码创建骨干结构，为指令添加节点，并通过控制流边将它们连接起来。此外，图中还附加了各种类型的节点和边，用于表示数据流信息中的特定操作和数据值。

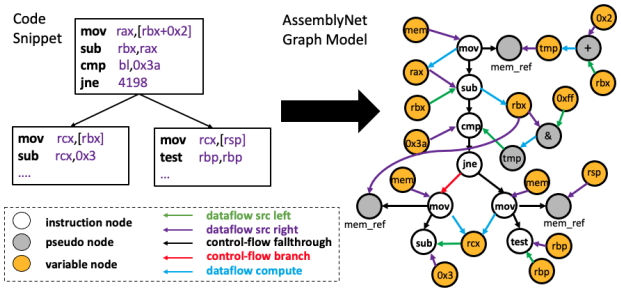


图 4：AssemblyNet 图模型示例。

AssemblyNet 的设计理念是利用专用节点和边来捕获计算规则，从而在不同程序间实现更好的通用性。例如，使用不同的边类型来区分指令中的操作数位置；处理具有不同位宽的寄存器（例如 *rax* and *eax*）时，会自动插入“按位与”节点。图 4 展示了一个由三个基本代码块和一个分支组成的小代码片段的图模型。接下来，我们将讨论图模型的详细设计。

图节点。 AssemblyNet 节点分为三类：指令、伪和变量。每类包

类别	type	描述	初始化
指令 *inst*		表示汇编指令（例如，mov、jne）	词汇表中的令牌 ID 词汇表
伪	+、−、*, *and*, *or*, 内存_引用	表示操作内指令（例如，an 内存访问）	词汇表中的令牌 ID 词汇表
变量	临时，寄存器，常量，*mem*	表示一个数据值（例如，常量、寄存器、内存）	值 from 应用程序跟踪或为零

表 2：节点类别和类型的描述。

含若干节点类型，用于编码执行规则的相关知识。每个指令节点表示汇编代码中的一条静态指令，例如 mov 或 cmp。伪节点表示指令内部发生的操作，例如地址计算或内存引用。例如，指令 mov rax, *[*rbx+*4]* 会将存储在 rbx+*4* 中的值加载到 rax，其源操作数关联着一个用于计算 rbx+*4* 的表达式，随后进行内存加载。在这种情况下，会用到两个伪节点：mem_ref 和+。变量类别中的节点表示动态值，可分为四种类型：临时结果（例如 rbx+*4*）、常量、寄存器值或内存中的值。例如，图 4 中所有黄色节点都是变量节点。我们在表 2 中总结了节点类别和类型，并说明其功能、用途和初始化方法。

图边。 AssemblyNet 有两种边类别：控制流 和 数据流，可细分为五种类型，以反映计算规则（见图 4）。我们使用控制流顺序执行 边连接连续的 指令 节点。如果某条条件分支从一个 指令节点分出（例如 ***jne***），我们会添加一条 控制流分支 边来连接目标 指令 节点。例如，***jne*** 节点有两条扇出边：分支未跳转（控制流顺序执行）和分支跳转（控制流分支）。这种设计有助于 AssemblyNet 区分顺序代码和条件代码，尤其是在执行期间选择代码路径。对于目标地址取决于运行时寄存器值的间接分支，AssemblyNet 会为离线分析确定的每个可能目标附加一条 控制流分支边。

数据流类别下有三种边类型：数据流左源、数据流右源和数据流计算。指令 和伪 节点使用前两种类型连接其数据源。这一特性使 AssemblyNet 能够识别操作数位置。此外，数据流计算 边能够捕捉 AssemblyNet 如何计算特定指令的数据。具有数据流计算边的节点可以计算输出，并将其发送给下游消费者。通常，数据流传播伴随着图神经网络中的多轮消息传递（称为图神经网络层）；值通过数据流边进行计算和传播。

对表达式的支持。 表达式用于计算汇编指令中内存引用的地

址。例如，***movsxd rdx, [rcx+rdx*4]*** 会从内存中加载一个值，其地址由表达式*[rcx+rdx*4]*，计算得出，然后将该值进行符号扩展并存储到*rdx* 中。对于指令中的每个表达式，AssemblyNet 都会递归构建一棵由变量节点和伪节点组成的表达式树。显然，表达式树的根始终是一个内存_引用节点。存在一条控制流顺序执行边，将相应的指令节点连接到表达式树的根。这样，表达式树便可从该边获取活跃性信息：如果对应的指令节点处于活跃状态，表达式树也处于活跃状态。此外，为了体现从内存加载值的语义，表达式树中始终连接着一个内存节点（即一种变量节点）（图4）；该节点稍后会在运行时初始化。

图构建。我们介绍 AssemblyNet 的图模型构建过程。首先，NPS 使用 gcc 对应用程序二进制文件进行反汇编，以获取汇编代码。然后，NPS 通过添加指令节点和控制流边来构建图骨架。此外，NPS 通过添加变量节点扩展图，并为指令添加表达式。为纳入数据依赖信息，NPS 标记每条指令的源寄存器对应的生产者指令。每个源寄存器可能对应多条生产者指令，因为代码路径可能分叉，并在不同目标处汇合。出现循环时，情况会变得复杂：如果两条指令都位于同一个循环中，那么汇编代码中较早出现的指令可能依赖于较晚出现的另一条指令。为解决这个问题，我们设计了一种跟踪每条指令传播情况的算法。为了标记传播情况，我们针对每条更新寄存器 R 的指令 I 执行广度优先搜索，遍历图中所有由 I 支配的指令节点。图遍历过程会持续进行，直到找到一条覆盖寄存器 R 的新指令。此外，每条读取寄存器 R 的指令 K 都会通过附加元组 (R, I) 来更新其簿记信息 U_K：$U_K = U_K \cup \{(R,I)\}$。我们重复这一过程，直到添加完所有元组。最后，我们再次遍历图，并根据簿记信息扩展表达式和数据流边。图 4 展示了运行该算法后，根据代码片段生成的图模型。

3.4. 应用程序运行时跟踪

图模型编码程序的静态信息，例如代码结构、指令语义和数据流。为捕获动态上下文，NPS 会执行应用程序跟踪，以收集寄存器值和内存值等运行时状态。应用程序跟踪模块使用标准插桩工具（如 Intel Pin [40]）记录运行时的微处理器架构状态，例如程序计数器和寄存器值，以及内存引用（地址和数据）。跟踪模块记录每条动态指令的状态，并将其保存到一个跟踪文件。最后，跟踪文件会发送到下游模块，用于生成图快照并进行图神经网络处理（图 3）。它还为训练任务提供数据标签。

算法 1图快照算法

要求： n_{cur} ▷ 当前指令节点
要求： *AsmGraph* ▷ AssemblyNet 图模型
要求： V ▷ 程序当前的动态状态
过程图_快照(n_{cur}，*AsmGraph*，V)
  图 ← 空图
  T_{access} ← {d| 所有最长代码路径中的内存访问次数（n_{cur} 无环）}
  最大值_允许_访问 ← 最小值(T_{access})
  指令集合 ← {v| 所有指令 AsmGraph 中的节点 使得 $D_{v,n_{cur}} \leq$ 最大值_允许 _访问}
  已访问的 ← **集合**(指令集合)，队列 ← **列表**(指令集合) ▷ 初始化簿记
  对于所有v 属于指令集合，**执行**
    图.add_节点(v) ▷ 先添加指令节点
  当队列 $\neq$ /0 **时** ▷ 扩展图快照 图
    n_i ←队列.pop()
    对所有 n_j 属于邻居(n_i) **执行**
      如果 $n_j \in$ 已访问，**则继续**
      否则如果$n\,j$ 是指令节点**且**$n_j \notin$ 指令集合 **则继续**
      否则如果 $n\,j$的 指令 节点 $v_j \notin$ 指令集合 **则** ▷ n_j 是 变量 或 伪 节点
        继续
      图.add_边(n_i, n_j)
      图.add_node(n_j)
      已访问.add(n_j), 队列.push(n_j) ▷ 更新簿记
  图 ← 节点_初始化(图，V) ▷ 参见第3.6
  返回 图

3.5. 图快照运行时生成

本节介绍图快照算法。已有工作 [54]表明，与当前指令 I 相关的代码行为由其附近的指令决定，而非整个GNN程序图。AssemblyNet利用这一观察，从AssemblyNet中提取子图，并将其与从执行过程中跟踪得到的动态值融合。图快照捕获代码拓扑结构、指令语义和数据流信息等静态信息，以及运行时寄存器值和内存值等动态上下文。为了让AssemblyNet能够推理多个连续的内存访问（从而增强能力），我们对图快照的拓扑定义了两个约束。**约束 1**：就指令节点和控制流边而言，图快照必须是有向无环图。**约束 2**：从当前指令 D 出发的任意代码路径，其内存引用数量必须相同

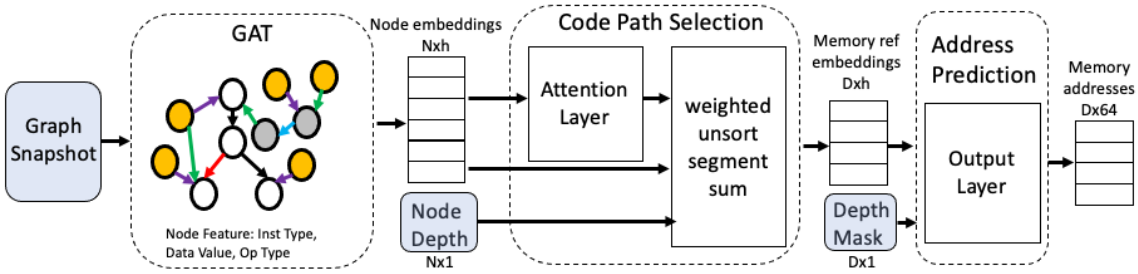


图5：AssemblyNet 神经架构（输入：图快照、节点深度和深度掩码）。

简而言之，约束 1 通过排除循环迭代来简化AssemblyNet的学习。相反，NPS使用多个快照来表示循环迭代的重复执行行为。约束 2 使候选代码路径保持一致，因为这些路径的内存访问次数可能不同。算法 1 简要描述了图快照算法。设 T_{access} 为从当前指令节点 n_{cur} 出发且不含任何环的所有最长代码路径 P 中内存访问次数的集合（约束 1）。P 可通过带循环检测的深度优先搜索遍历获得。此外，我们将节点 u 和节点 v 之间所有代码路径中的最小内存引用数记为 $D_{u,v}$。图快照 图 G 的最大允许访问数为 T_{access} 中的最小值。我们删除任意代码路径中其余的指令，以确保内存引用数不超过最大允许访问数_。因此，所有从 I 出发的代码路径都具有相同数量的内存引用（约束 2）。我们遍历图模型（第 5 行），将所有满足最大允许访问数_要求的指令节点收集到列表 Insts 中。最后，我们执行类广度优先遍历，通过向 G 添加节点和边来扩展子图，尽可能覆盖更多上下文（第 9-18 行）。确定拓扑后，通过节点初始化（第3.6节）将动态值融合到 G 中（第 19 行）。

3.6. AssemblyNet 神经架构

本节介绍 AssemblyNet 的神经架构。我们使用数据预取任务对其进行训练：给定一个图快照，预测连续的内存引用地址。代码路径选择和地址预测是专为该训练任务设计的。随后，我们应用训练好的图神经网络模型生成用于程序采样的执行嵌入。图5 展示了整体神经架构。

图神经网络模型。 AssemblyNet 使用图注意力网络（GAT）[59] 来建模程序执行的行为。GAT 假设邻居节点的贡献各不相同，并采用自注意力机制来学习消息的相对权重。具体而言，当前层的每个节点嵌入都由输入的加权消息以及该节点在前一层的节点表示计算得出。在训练期间，图神经网络会生成N 个节点嵌入（维度 $= h$），供下游模块使用，例如代码路径选择和地址预测。在运行时，图快照会被输入图注意力网络模型，以表征执行区间；每个图快照都会通过*readout* 函数映射为图嵌入。

代码路径选择。给定当前指令 I，预测 I 将沿哪条代码路径执行颇具挑战。例如，在 I 之后的五次内存引用范围内，可能有数十条候选代码路径。在 AssemblyNet 的图模型（第3.3节）中，我们为地址预测专门设置了任务节点（mem_ref），每条代码路径恰好包含 D 个任务节点。平均而言，$D = 6$，且可能的代码路径超过 30 条（第5.4节）。与此同时，神经网络必须先找出正确的路径，才能选择合适的一组任务节点。AssemblyNet 通过应用注意力层来解决这一难题：对所有潜在代码路径中的 i^{th}（$i \in D$）个任务节点进行聚合，并为其赋予权重，从而计算出最可能的 i^{th} 个任务节点。具体而言，注意力层根据每个任务节点的节点嵌入，为其学习一个权重。然后，将所有潜在任务节点的嵌入分别乘以这些权重。每个任务节点经加权后的嵌入会根据节点深度路由到特定位置（图5）。节点深度指定任务节点的路由目标，表示第 i^{th} 次内存访问的顺序。深度相同的节点会通过求和进行聚合（例如 TensorFlow [7]中的unsorted_segment_sum）。最后，我们采用节点掩码技术 [43]（深度掩码）来适应不同图快照中任务节点数量的变化，并屏蔽无效任务节点。我们将 D 的最大值限制为 20。AssemblyNet 中的注意力机制实质上是通过在每次内存访问（深度）处选取任务节点，来选择指令 I 最有可能遵循的代码路径。

地址预测。 代码路径选择模块利用注意力选择最可能的路径，并生成用于预测地址序列的嵌入。我们将计算得到的“内存引用”嵌入（维度 $= D * h$）输入地址预测模块，由多层感知机将这些嵌入转换为64位内存地址（维度 $= D * 64$）。我们将损失函数定义为对所有D个内存地址的Sigmoid交叉熵损失进行掩码求和。

节点初始化。 为了纳入动态状态，AssemblyNet 采用神经代码融合（NCF）技术 [54]，在运行时将动态值赋给图节点；我们修改了寄存器值赋值方式，以更好地捕获代码执行不变量。在 NCF 中，寄存器值赋

值方式对具有相同寄存器标识符（例如 *rax*）的所有变量节点都采用统一处理。不过，有些值只能在 GNN 消息传播期间计算。因此，AssemblyNet 仅为没有入向数据流边的寄存器变量节点赋值；其他变量节点则初始化为零，等待 GNN 计算。对于指令节点和伪节点，则使用词汇表中的词元值进行初始化，随后通过学习得到的嵌入层将其转换为初始节点嵌入。

3.7. 嵌入生成

程序采样将目标应用程序划分为连续的执行区间（例如，每个区间包含 1M 条动态指令）。我们为每个区间生成一个 NPS 嵌入。在嵌入生成过程中，AssemblyNet 会创建一系列图快照，每个快照覆盖一小段时间（通常为 50 条动态指令）。这些图快照会映射为节点嵌入，随后通过读出函数转换为图嵌入。这个读出函数可以是置换不变函数，例如求和或均值，也可以是更复杂的图级池化函数 [64, 63]。在 NPS 中，我们直接使用求和。

为支持较长的指令序列，NPS 会进行序列聚合，将图嵌入汇总到一个执行区间中。我们尝试了不同的序列聚合算法，例如均值聚合和序列自编码器 [21]。与均值聚合相比，自编码器在压缩序列时能够保留较高层次的时间信息。然而，自编码器的计算开销更大。在 NPS 中，我们结合均值聚合和自编码器，以支持大规模区间。NPS 首先对指令的“子序列”求均值，然后对其应用自编码器，以捕捉不同“子序列”平均行为的时间信息。这项技术帮助 NPS 为包含超过 100M 条指令的区间生成执行嵌入。最后，NPS 将所有执行嵌入发送给经典的基于聚类的采样器[29]，以选取具有代表性的模拟点。

4. 方法

4.1. 实验设置

评估平台。 我们使用 gem5 [16] 作为评估平台，以获取详细的性能统计数据。同时，我们增强了 gem5，使其能够在仿真期间报告每个执行区间的统计数据。在实验中，我们使用默认的 O3 模型和两种经过校准的商用 CPU 模型进行评估：一种开源 Haswell gem5 模型 [8]，以及一种基于 Intel Xeon 8163 CPU 校准的 Skylake 模型。

工作负载。 我们使用 SPEC2006 整数基准测试 [3,4]评估 NPS。选择 SPEC2006 可以直接、公平地比较 SimPoint [47, 29] 和 NCF [54], 这两种分别用于程序采样和代码执行学习的最先进方法。整个 SPEC2006 整数应用程序的规模从 300B 到 3800B 条指令不等。由于详细周期级仿真和 NPS 处理的开销，我们使用参考输入，让每个应用程序执行 20 billion 条指令（12 个应用程序共计 240B 条指令）。尽管包含 20B 条指令的区域仅占应用程序的一部分，但对于使用细粒度模拟点评估高分辨率程序采样而言，这个规模已足够大。

图神经网络。 我们使用一个 4 层图注意力网络（GAT） [59], 其超参数如下：节点维度为 256，包含 8 个注意力头、5 种边类型，以双曲正切函数 作为激活函数，词汇表大小为 300，隐藏层大小为 256。我们使用 Adam 优化器 [34] 训练 AssemblyNet，学习率固定为 10^{-5}。在训练和测试中，我们将从 2 亿条指令中收集的图样本划分为训练集和测试集，其中 70% 用于训练，30% 用于测试。整个训练过程在 Nvidia Tesla V100 上耗时不到 20 小时。模型训练完成后，我们将其部署到原型系统中，为数千亿条指令生成 NPS 嵌入。

系统原型。 我们的原型系统运行在 10 台 Intel Xeon Platinum 8163 服务器上。每台服务器配备 48 个以 2.5GHz 运行的 CPU 核心和 800GB 内存。我们使用 CentOS 7 作为操作系统，使用 Docker 19 进行部署，并使用 TensorFlow 2.2 [7]进行训练和嵌入生成。

4.2. 度量指标

程序采样器（例如 SimPoint）选择簇质心作为模拟点，以近似原始应用程序。因此，CPI 误差不可避免，且可能为正或负。我们使用两个 CPI 误差指标来衡量程序采样准确度： 平均绝对百分比误差（平均绝对百分比误差）和平均误差（ME）。平均绝对百分比误差是机器学习中常用的误差指标，它累加每个数据点的绝对误差（公式4） [22]。由于平均绝对百分比误差对每个区间的 CPI 误差取绝对值，因此在衡量 +/- 误差时较为保守。 平均误差 （公式5） [29] 则简单地对所有误差取平均，因此较为乐观，因为在累加误差时正负误差会相互抵消。 平均误差 常用于粗粒度程序采样；但若需要进行细粒度程序采样，它的误差分辨率就不够充分。在评估中，我们会同时给出这两个误差指标。为评估 AssemblyNet 的能力，我们使用数据预取准确度作为度量指标。请注意，对于多个连续访问的数据预取，只有当预测出的 64 位地址逐位准确且顺序正确时，预测才算正确。

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\frac{\left|CPI_{c(i)} - CPI_i\right|}{CPI_i} \tag{4}$$

$$ME = \left|\frac{\sum_{i=1}^{n} CPI_{c(i)} - CPI_i}{\sum_{i=1}^{n} CPI_i}\right| \tag{5}$$

其中，n 表示区间总数，$c(i)$表示区间$_i$ 所属的簇质心。

5. 评估

本节首先展示 NPS 的程序采样性能。随后，我们研究 NPS 嵌入的表征优势，并分析准确率鲁棒性。最后，我们评估 AssemblyNet 的有效性。

5.1. 程序采样准确度

我们将 NPS 嵌入应用于程序采样，并与业界事实标准 SimPoint [47, 29]方法进行比较。NPS 根据程序代码结构、指令类型、数据流图、控制流活跃度和动态数据值等信息学习执行行为，编码的执行上下文比 SimPoint 的 BBV 更丰富。NPS 和 SimPoint 使用相同的采样算法（采用贝叶斯信息准则的 K均值 [47]），但表征执行区间的向量表示不同：分别为 NPS 嵌入和 BBV。我们在随机选取的程序区域（第 *4*节）上，使用参考输入，在 Skylake gem5 CPU 上运行 SPEC2006整数测试集 应用程序。每个执行区间包含 1000 万条动态指令。我们将 MaxK 限制为 20，以便采样器为每个应用程序选取模拟点。图 6 展示了 12 个应用程序上的平均绝对百分比误差 （式4）。MAPE 保留了所有采样误差，相比平均误差 度量指标（式5），具有更高的误差分辨率。图6显示，在全部 12 个应用程序上，NPS 的准确性均优于 SimPoint；NPS 的平均 MAPE 为 6.8%，而 SimPoint 为 11%，采样误差降低了 38%。在许多应用程序中，NPS 大幅领先 SimPoint，误差降低超过 50%，最高达 63%（483.xalancbmk）。

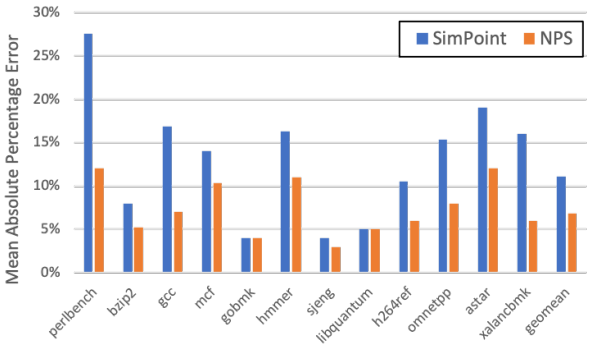


图 6：程序采样性能（平均绝对百分比误差）。

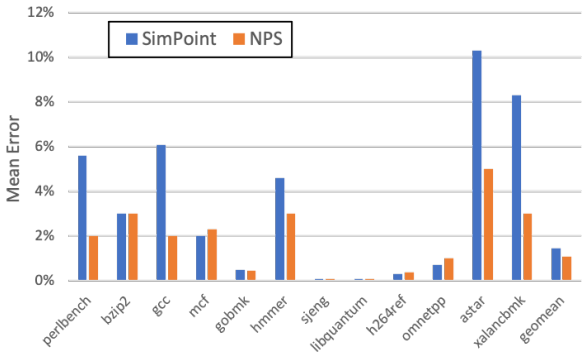
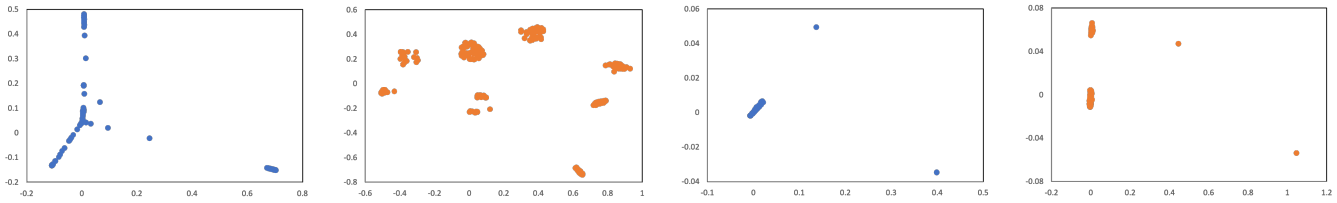


图 7：程序采样性能（平均误差）。

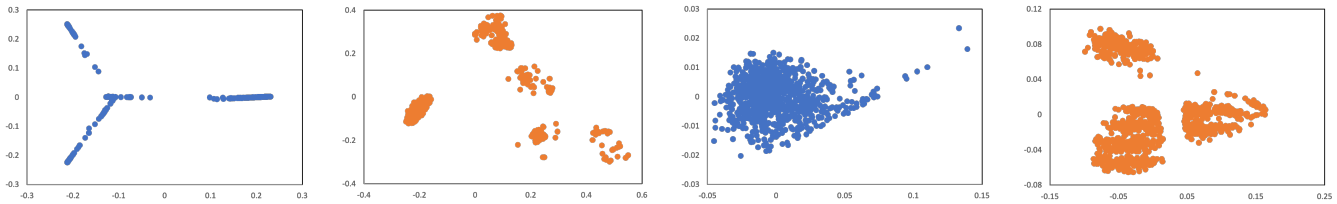
另一方面，图7 使用平均误差度量指标比较了 NPS 和 SimPoint 之间的采样准确性。对于 NPS 和 SimPoint，许多应用程序的采样误差都很小（小于 2%）。然而，由于正误差和负误差会相互抵消（第4.2节），平均误差是粗粒度且过于乐观的，会低估采样误差。因此，误差无法通过平均误差度量指标得到充分反映，这也会导致所选模拟点的准确性不稳定。尽管如此，在平均误差方面，NPS 仍然优于 SimPoint，平均误差达到 1.1%。两种度量指标的评估结果表明，NPS 通过采用高质量的执行嵌入，能够实现准确的程序采样。

5.2. 执行嵌入可视化

图8可视化展示了四个应用程序中第5.1节所述程序区域的 NPS 和 BBV 执行嵌入。我们利用主成分分析将执行嵌入投影到二维空间中。在 403.gcc 中，我们发现 NPS 嵌入能够更好地聚类相似的执行区间。这是因为 NPS 使用多种信息从执行过程中捕获特征，例如代码拓扑结构、指令语义、数据流、控制流和动态值，而不是基本块频率。BBV 通过比较不同基本块集合及其频率，能够很好地区分粗粒度执行区间（例如，100M 条指令）。然而，在细粒度区间（例如，500K 条指令）下，热点基本块变化不大，主要差异来自频率。因此，BBV 会遗漏有关执行的关键区分特征。此外，如图 8 所示，由于编码的执行信息不足，BBV 带来的自由度较少。例如，在 403.gcc 中，BBV 嵌入分布在 3 个方向上。456.hmmer 和 473.astar 也存在类似现象。在 456.hmmer 中，BBV 和 NPS 都能区分两个占比较小的执行区间集合（见右侧）。然而，NPS 还能将大多数执行进一步分为两个簇，而 BBV 因编码的上下文不足而无法区分它们。在 483.xalancbmk 中，BBV 将各次执行分散在整幅图中——执行区间涉及各种基本块，但 BBV 无法衡量它们之间的相似性。由于 BBV 无法准确表征细粒度执行，SimPoint 的准确性对采样器设置（例如，随机种子，*MaxK*）高度敏感，这大大增加了精度调优的风险。相比之下，NPS 利用代码拓扑结构、指令类型和数据值等不变量，生成高质量的执行嵌入，从而实现准确的程序采样。



(a) BBV 403.gcc (b) NPS 403.gcc (c) BBV 456.hmmer (d) NPS 456.hmmer



(e) BBV 473.astar (f) NPS 473.astar (g) BBV 483.xalancbmk (h) NPS 483.xalancbmk

图 8：执行嵌入（BBV 和 NPS）的散点图。

5.3. 采样准确性鲁棒性

本节研究采样鲁棒性——这是准确性调优的关键因素，以及其在仿真资源（*MaxK*[47]）变化时的表现。

，硬件微体系结构 ，和采样粒度 。**利用资源提升准确性**。如果采样器允许增加模拟资源以提高准确性，那么它就能随模拟资源扩展。良好的准确性可扩展性至关重要，因为工程师通常希望通过在精度调优过程中调整 *MaxK* ，以模拟资源换取更高的准确性。此外，如果增大 *MaxK* 不会损害准确性，那么这种采样就具有鲁棒性。在图 11中，我们调整*MaxK* ，并观察准确性变化趋势和鲁棒性。同时，采样器可以自行选择不超过 *MaxK* 的任意簇数，以最大限度地提高准确性。图 11a显示，SimPoint 在较大的 *MaxK* 下可能出现更大的误差，例如 401.perl- bench。随着 *MaxK* 增大，SimPoint 的准确性会波动，导致显著的调优开销。根本原因在于 BBV 无法清晰地对区间进行聚类：它会遗漏重要的执行特征，使采样器产生混淆。相比之下，NPS（图 11b 和图 11c）具有鲁棒性，不会因模拟资源增加而导致准确性下降。

图 9 概述了有利的 *MaxK* 截止值，它是精度扩展性的最佳点。超过此点后，准确性可能会下降，或者准确性不会显著提升。我们发现，与 BBV 相比，NPS 在更大的 *MaxK* 截止值下扩展性更好。此外，在某些情况下（bzip2 和 h264ref），NPS 中使用自编码器可以提高 *MaxK* 截止值。总之，得益于 NPS 的稳健性和准确性扩展性，工程师可以不必过于担心增加模拟点会损害准确性，从而省去繁琐的调优工作。

微体系结构影响。 设计人员通过程序采样来指导硬件设计，并基于仿真评估各种特性、参数和配置。因此，所选的模拟点必须保持准确

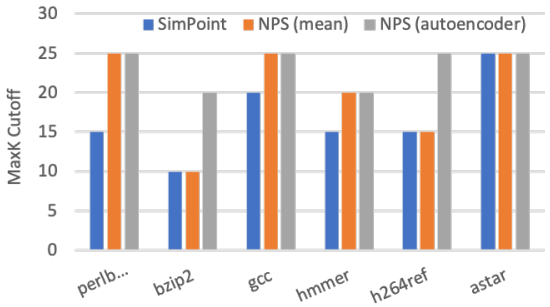
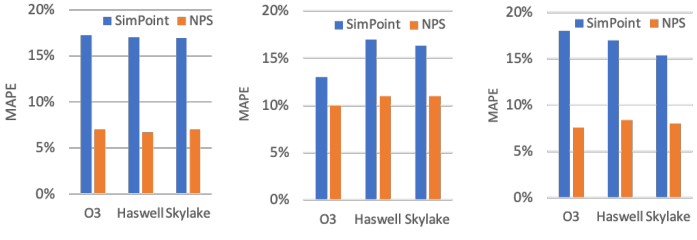


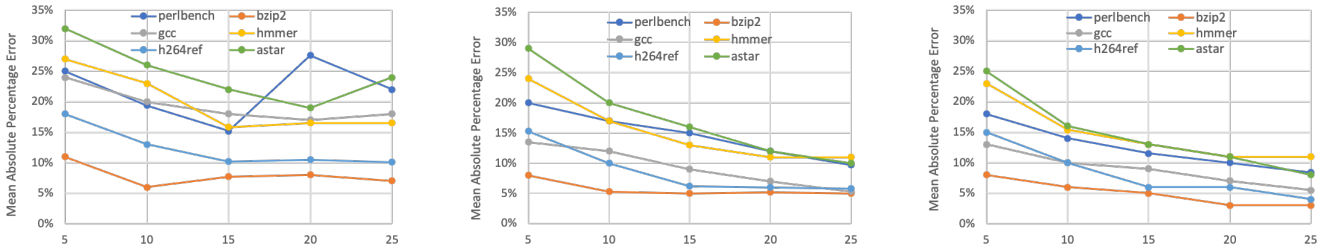
图 9：有利的 MaxK 截止值。

在不同微架构上保持鲁棒性。图10评估了三个应用程序（403.gcc、456.hmmer 和 471.omnetpp）在第4节所述的三种 CPU 模型上的采样精度敏感性。正如我们所见，NPS 始终实现了较低的采样误差，并且能够抵抗 CPU 模型变化，误差方差小于 1%。另一方面，SimPoint 对底层硬件平台更加敏感，导致模拟点不够理想。例如，在 456.hmmer 上，与 Haswell 和 Skylake 模型相比，O3 CPU 模型的模拟点误差率变化超过 20%。



(a) 403.gcc (b) 456.hmmer (c) 471.omnetpp

图 10：不同微架构变体下的准确性敏感度。



(a) SimPoint (b) NPS（平均值） (c) NPS（自编码器）

图 11：簇数量（MaxK）对准确性的扩展性。

采样粒度。我们研究了不同程序采样粒度下的准确率鲁棒性。图12进行了三个粒度级别的实验：**1**。包含 2B 条指令且区间为 1M 条指令的程序（细粒度），**2**。包含 20B 条指令且区间为 10M 条指令的程序（中等粒度），以及**3**。包含 200B 条指令且区间为 100M 条指令的程序（粗粒度）。该设置模拟了大规模应用程序中的程序采样：使用粗粒度模拟点，并在小型/中型区域上采用细粒度采样，以进行详细的低层级仿真。所有配置选择相同数量的模拟点，以保持采样复杂度一致（等效采样率）。在图12中，得益于高分辨率的执行嵌入，NPS 的准确性提升具有鲁棒性，在三个应用程序的三种粒度级别上均优于 SimPoint。此外，随着采样粒度变细，SimPoint 中的错误率显著上升，这是由于 BBV 的分辨率不足；而 NPS 则表现出更好的鲁棒性。简而言之，SimPoint 在细粒度下的表现不尽如人意。相比之下，NPS 在各种采样粒度下都能实现更高的准确性和鲁棒性。

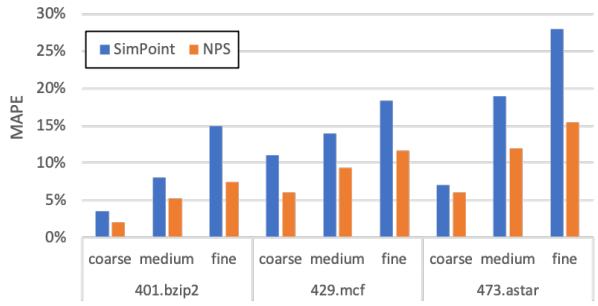


图 12：采样粒度对准确性的敏感性。

5.4. 评估 AssemblyNet

在图 13中，我们通过数据预取任务（第3.2节）评估 AssemblyNet 学习代码行为的有效性。我们采用最先进的 NCF [54] 方法作为基线。*NCF*（*general*）为所有应用程序使用统一的单一图神经网络模型。*NCF*（*specialized*）为每个应用程序使用一个图神经网络模型（GGNN[37]），这也是原论文 [54] 中的设计。对于 *NCF*（*general*）和 *AssemblyNet*，我们使用图注意力网络（GAT）[59]作为图神经网络模型。对这两者进行比较可见，AssemblyNet 在跨应用程序泛化方面表现更好，平均准确性为 87.5%，而 NCF 为 18.2%。对于 *NCF*（*specialized*），其平均准确性可达 57.4%，表现良好。然而，它需要为每个应用程序单独训练一个图神经网络模型。比较 *NCF*（*general*）和 *NCF*（*specialized*）后，我们发现，由于无法捕获应用程序不变量，NCF 的跨应用程序泛化能力受到严重影响。因此，NCF 无法以通用方式学习跨应用程序的代码执行。相比之下，AssemblyNet 在准确性和通用性方面表现出更优的性能。此外，在单个图快照中，AssemblyNet 生成的预取地址平均是 NCF 的 6 倍。因此，AssemblyNet 在捕获动态程序行为方面比 NCF 更强大，为生成高质量的执行嵌入铺平了道路。

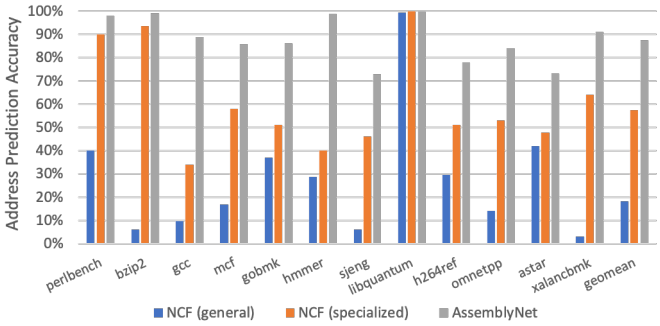


图 13：数据预取任务性能。

5.5. 按应用划分的执行嵌入

NPS 学习执行不变式，例如数据流传播、值计算、控制流活跃性和指令语义。因此，它可以将不同应用程序中的执行映射到同一向量空间。图 14展示了第 5.1 节中 12 个 SPEC 应用的程序区域散点图。得益于 NPS，我们可以直接比较不同应用程序中的执行；而由于基本块中编码的知识不可迁移，BBV 仅限于单个应用程序内。

。在图 14中，我们可以看到 483.xalancbmk 与其他应用程序截然不同。此外，458.sjeng 与 473.astar 非常相似，这两个应用程序实际上属于同一 人工智能 类别 [4] —— 458.sjeng 与国际象棋人工智能对弈有关，而 473.astar 则与游戏人工智能寻路有关。总之，NPS 为代码执行提供了一种通用表示。

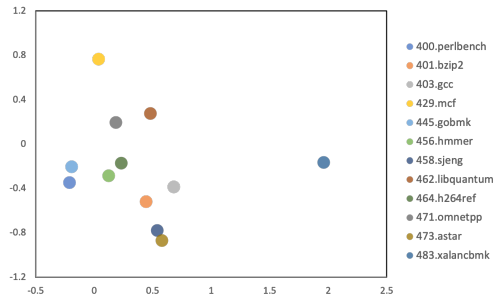


图 14：SPEC2006 整数应用程序散点图。

6. 讨论与相关工作

6.1. 微处理器设计性能评估

程序采样。 大规模基准测试在模拟器上运行成本高昂，有时甚至不可行。程序采样 [47, 29, 26, 62, 55, 48, 50] 是一种广泛使用的统计方法，可在评估微处理器的各种设计方案时降低工作负载模拟成本。业界标准采样器 SimPoint [47, 29], 使用 BBV [52] 对应用程序进行采样，以选出具有代表性的模拟点。接下来，检查点工具 [46, 45] 会对每个样本的内存状态进行快照，使设计人员无需快速前推即可并行模拟这些样本。随后，将模拟点的准确性与原始程序进行对比验证，设计人员再调整采样器的设置（例如 随机种子、*MaxK*、阈值等） [55], , 改变样本选择方式以微调准确性。此过程会反复进行，直到准确性达到要求。因此，传统的准确程序采样耗时很长：性能验证和检查点生成运行缓慢，而且需要多轮调优。相比之下，NPS 为高质量执行嵌入提供了一种新方法，能够以更低的调优开销实现准确的程序采样。由于 NPS 嵌入编码了更丰富的上下文，包括静态和动态信息，因此在作为执行区间的特征向量时，比 BBV 更有效，尤其适用于细粒度程序采样。此外，NPS 的稳健性消除了高昂的精度调优开销，为微处理器设计人员节省了大量时间。

工作负载合成。 降低模拟成本的另一类研究通过构建微基准测试来合成工作负载 [11, 12, 20, 26, 44]。微基准测试经过精心手工设计，用于突出工作负载的特定方面（例如内存访问模式），以便评估微处理器。例如，局部性度量指标包括重用距离和步长距离 [11, 12, 20, 44], , 控制流度量指标则包括指令组合、分支率和指令依赖 [26, 19, 44]。然而，工作负载合成需要对目标工作负载有深入的实践经验，并需要手工设计特征工程；为不同基准测试领域创建新的微基准测试时，这会产生一次性工程成本。相比之下，NPS 提供了一种通用方法，可从各个领域的目标工作负载中采样关键部分。

仿真基础设施。 出于性能评估和验证的目的，模拟器是微处理器开发中的重要工具，在速度和准确性之间提供了广泛的权衡选择。软件模拟器在建模支持操作系统的全系统工作负载方面具有很高的灵活性，既可以采用事件驱动方式 [16, 6] ，也可以采用基于事务的方式 [5]。然而，这种灵活性和相对较快的速度是以准确性下降为代价的。另一方面，RTL 级仿真能够提供准确的性能结果，但若在软件中运行，仿真速度会极其缓慢 [56, 27, 2]。因此，出现了混合模拟器，例如 *gem5+rtl* [39] 和 gem5-Aladdin[51] ，它们将 RTL 模型嵌入全系统仿真中，以平衡灵活性、仿真速度和建模准确性。基于 FPGA 的仿真 [1, 33]能够加速 RTL 建模，但受可用机器资源的限制。近期，研究人员积极开展了利用深度神经网络开发 AI 辅助模拟器 [49, 36] 的研究。NPS 利用人工智能技术帮助设计人员将工作负载划分为细粒度的仿真点，使其能够在最合适的仿真平台上以较低成本运行这些仿真。

6.2. 利用人工智能对软硬件系统建模

编译时。 利用人工智能对编译器行为建模正受到越来越多研究者的关注。例如，编译器研究人员将图神经网络模型应用于静态代码分析任务，如类型推断 [61], 错误修复 [24, 9], 算子调度[65], 和代码相似性比较 [10]。在这些任务中，抽象语法树或计算图被用作图神经网络图模型的骨架。除了图神经网络，研究人员还探索利用深度强化学习进行自动向量化 [28] ，以生成高性能的向量化代码。与这些研究类似，NPS 利用人工智能构建程序专用图模型，以捕获静态信息。此外，NPS 还专注于对运行时上下文进行编码，以生成高质量的执行嵌入。

执行时。 人们提出了各种深度神经网络模型，用于建模代码在运行时的控制流与数据流代码行为 [9, 15, 41, 54, 31]。例如，BranchNet [41]旨在通过卷积神经网络模型学习难以预测的分支。NCF[54] 使用图神经网络进行分支预测和数据预取。Voyager [53] 提出了一种分层神经数据预取器。NALU [57]训练神经网络来执行浮点数的算术运算。在这些工作中，数据预取

是一项广受欢迎的任务 [53, 31, 54]，它反映了学习代码执行的全方位能力。NPS 的灵感来自 NCF。然而，我们发现 NCF 无法在不同应用程序间很好地泛化，只在单个应用程序内有效。与 NCF 不同，NPS 采用跨应用程序的通用图表示（AssemblyNet），重点学习程序之间不变的计算规则（例如，运算符位置、数据流传播、控制流活跃性等）。此外，NPS 的训练任务更具挑战性，它将代码路径选择与连续内存引用预测结合起来，而非仅仅预测下一个后续内存访问 [54]。得益于这一困难的任务，NPS 生成了高质量的执行嵌入。

设计阶段。 在系统设计阶段，深度学习技术已在诸多方面取得显著进展。例如，研究人员提出使用循环神经网络 [14] 对云工作负载进行建模，以支持下一代云平台设计。此外，DiffTune [49] 利用 AI 学习模拟器的参数设置，从而减少其在给定工作负载集上的仿真误差。SimNet [36]构建了一个用于周期级微体系结构模拟的神经模拟器。BOOM Explorer[13] 利用高斯模型和深度核学习函数，高效搜索 RISC-V 核心的设计空间。Lin 等人 [38] 通过深度强化学习优化无路由器片上网络中的循环放置。Mirhoseini 等人 [42] 提出了一种 AI 工具：通过将网表编码为图神经网络，并采用深度强化学习进行优化，实现芯片布局规划和器件布局布线。类似地，NPS 是一种用于敏捷开发的 AI 辅助设计工具。NPS 提供稳健的细粒度程序采样，具有较高的准确性，从而简化微处理器设计的性能评估。

7. 结论

本文提出了神经程序采样（NPS），这是一种新颖的框架。它利用 GNN 的动态快照生成高质量的执行嵌入，从而实现针对细粒度模拟点的精准程序采样。在 NPS 中，Assem- blyNet 通过在数据预取任务上训练来学习代码执行。NPS 创建图快照并将其转换为执行嵌入，以用于程序采样，其表现远胜于事实上的 SimPoint 方法。NPS 生成执行嵌入的能力为微处理器性能建模、基准测试、仿真和评估开辟了令人振奋的研究机遇。

参考文献

[1]Palladium 仿真。 https://www.cadence.com/en-US/home/tools/system-design-and-verification/emulation-and-prototyping/palladium.html.[2] Synopsys VCS。 https://www.synopsys.com/verification/ simulation/vcs.html. [3] SPEC CPU 2006。 https://www.spec.org/cpu2006/，2006年。[4] SPEC2006 基准测试说明。 https://www.spec.org/cpu2006/publications/CPU2006benchmarks.pdf，2006年。[5] Arm SystemC 周期模型。 https://community.arm.com/arm-community-blogs/b/tools-software-ides-blog/posts/arm-systemc-cycle-models-now-available，2016年。

[6] 使用 gem5 进行体系结构探索。https://www.gem5.org/assets/files/ASPLOS2017_gem5_tutorial.pdf，2017年。[7] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard 等。TensorFlow：一种用于大规模机器学习的系统。载于第12届USENIX 操作系统设计与实现研讨会（OSDI 16），第265–283页，2016年。[8] Ayaz Akram 和 Lina Sawalha。针对 x86架构验证 gem5模拟器。载于2019年 IEEE/ACM高性能计算机系统性能建模、基准测试与仿真会议（PMBS），第53–58页。IEEE，2019年。[9] Miltiadis Allamanis、Marc Brockschmidt 和 Mahmoud Khademi。学习用图表示程序。ICLR 2018，2018年。[10] Uri Alon、Meital Zilberstein、Omer Levy 和 Eran Yahav。code2vec：学习代码的分布式表示。ACM编程语言学报，3（POPL）：1–29，2019年。[11] Amro Awad 和 Yan Solihin。STM：克隆时空内存访问行为。载于2014年 IEEE第20届高性能计算机体系结构国际研讨会（HPCA），第237–247页。IEEE，2014年。[12] Mario Badr、Carlo Delconte、Isak Edo、Radhika Jagtap、Matteo Andreozzi 和 Natalie Enright Jerger。Mocktails：捕获专有移动架构的内存行为。载于2020年 ACM/IEEE第47届年度计算机体系结构国际研讨会（ISCA），第460–472页。IEEE，2020年。[13] Chen Bai、Qi Sun、Jianwang Zhai、Yuzhe Ma、Bei Yu 和 Martin DF Wong。BOOM-Explorer：RISC-V BOOM微体系结构设计空间探索框架。载于2021年 IEEE/ACM计算机辅助设计国际会议（ICCAD），第1–9页。IEEE，2021年。[14] Shane Bergsma、Timothy Zeyl、Arik Senderovich 和 J Christopher Beck。使用循环神经网络生成复杂、逼真的云工作负载。载于 ACM SIGOPS 第28届操作系统原理研讨会论文集，第376–391页，2021年。[15] David Bieber、Charles Sutton、Hugo Larochelle 和 Daniel Tarlow。使用指令指针注意力图神经网络学习执行程序。第34届神经信息处理系统大会（NeurIPS 2020），2020年。[16] Nathan Binkert、Bradford Beckmann、Gabriel Black、Steven K Rein-hardt、Ali Saidi、Arkaprava Basu、Joel Hestness、Derek R Hower、Tushar Krishna、Somayeh Sardashti 等。gem5模拟器。ACM SIGARCH计算机体系结构新闻，39（2）：1–7，2011年。[17] Shekhar Borkar 和 Andrew A Chien。微处理器的未来。ACM通讯，54（5）：67–77，2011年。[18] Chen Chen、Xiaoyan Xiang、Chang Liu、Yunhai Shang、Ren Guo、Dongqi Liu、Yimin Lu、Ziyi Hao、Jiahui Luo、Zhijian Chen 等。玄铁910：一种商用多核、12级流水线、乱序执行、64位高性能 RISC-V 处理器，支持向量扩展：工业产品。载于2020年 ACM/IEEE第47届年度计算机体系结构国际研讨会（ISCA），第52–64页。IEEE，2020年。[19] Jian Chen、Lizy Kurian John 和 Dimitris Kaseridis。使用程序固有特征对程序资源需求建模。ACM SIGMETRICS性能评估评论，39（1）：1–12，2011年。[20] Jian Chen、Ying Zhang、Xiaowei Jiang、Li Zhao、Zheng Cao 和 Qiang Liu。DWT：面向数据中心的解耦工作负载跟踪。载于2020年 IEEE高性能计算机体系结构国际研讨会（HPCA），第677–688页。IEEE，2020年。[21] Kristy Choi、Curtis Hawthorne、Ian Simon、Monica Dinculescu 和 Jesse Engel。使用 Transformer自编码器编码音乐风格。载于国际机器学习会议，第1899–1908页。PMLR，2020年。[22] Arnaud De Myttenaere、Boris Golden、Bénédicte Le Grand 和 Fabrice Rossi。回归模型的平均绝对百分比误差。Neurocomputing，192：38–48，2016年。[23] Mike Demler。Andes发布 RISC-V 向量路线图。微处理器报告，2020年。[24] Elizabeth Dinella、Hanjun Dai、Ziyang Li、Mayur Naik、Le Song 和 Ke Wang。Hoppity：学习图变换以检测并修复程序错误。载于国际学习表征会议（ICLR），2020年。[25] David R Ditzel 等。使用 Esperanto 的 ET-SoC-1芯片上的逾1000个 RISC-V/张量处理器加速机器学习推荐。IEEE Micro，42（3）：31–38，2022年。[26] Lieven Eeckhout、John Sampson 和 Brad Calder。利用程序微体系结构无关特征和阶段行为，减少基准测试套件仿真。载于 IEEE 工作负载特征分析研讨会2005年会议论文集，第2–12页。IEEE，2005年。

[27] Tristan Gingold。Ghdl。http://ghdl.free.fr/，2017年。[28] Ameer Haj-Ali、Nesreen K Ahmed、Ted Willke、Yakun Sophia Shao、Krste Asanovic 和 Ion Stoica。Neurovectorizer：使用深度强化学习实现端到端向量化。载于第18届 ACM/IEEE国际代码生成与优化研讨会论文集，第242–255页，2020年。[29] Greg Hamerly、Erez Perelman、Jeremy Lau、Brad Calder、Timothy Sher-wood 和 Haym Hirsh。使用机器学习指导体系结构模拟。机器学习研究杂志，7（2），2006年。[30] Will Hamilton、Zhitao Ying 和 Jure Leskovec。大型图上的归纳式表示学习。神经信息处理系统进展，30，2017年。[31] Milad Hashemi、Kevin Swersky、Jamie Smith、Grant Ayers、Heiner Litz、Jichuan Chang、Christos Kozyrakis 和 Parthasarathy Ranganathan。学习内存访问模式。载于国际机器学习会议，第1919–1928页。PMLR，2018年。[32] John L Hennessy 和 David A Patterson。计算机体系结构的黄金新时代。ACM通讯，62 60，2019 2 48（）：–。[33] Sagar Karandikar、Howard Mao、Donggyu Kim、David Biancolin、Alon Amid、Dayeol Lee、Nathan Pe mberton、Emmanuel Amaro、Colin Schmidt、Aditya Chopra 等。FireSim：在公有云中进行 FPGA加速的周期精确横向扩展系统模拟。载于2018年 ACM/IEEE第45届年度计算机体系结构国际研讨会（ISCA），第29–42页。IEEE，2018年。[34] Diederik P Kingma 和 Jimmy Ba。Adam：一种随机优化方法。arXiv预印本 arXiv:1412.6980，2014年。[35] Thomas N. Kipf 和 Max Welling。使用图卷积网络进行半监督分类。载于国际学习表征会议（ICLR），2017年。[36] Lingda Li、Santosh Pandey、T homas Flynn、Hang Liu、Noel Wheeler 和 Adolfy Hoisie。SimNet：使用深度学习实现准确、高性能的计算机体系结构仿真。ACM测量与分析计算系统学报，6（2）：1–24，2022年。[37] Yujia Li、Daniel Tarlow、Marc Brockschmidt 和 Richard Zemel。门控图序列神经网络。国际学习表征会议（ICLR'16），2016年。[38] Ting-Ru Lin、Drew Penney、Massoud Pedram 和 Lizhong Chen。一种用于体系结构探索的深度强化学习框架：无路由器片上网络案例研究。载于2020年 IEEE高性能计算机体系结构国际研讨会（HPCA），第99–110页。IEEE，2020年。[39] Guillem López-Paradís、Adrià Armejach 和 Miquel Moretó。gem5+ rtl：一种支持在全系统模拟器中运行 RTL 模型的框架。载于第50 11届国际并行处理会议，2021年，第–页。[40] Chi-Keung Luk、Robert Cohn、Robert Muth、Harish Patil、Artur Klauser、Geoff Lowney、Steven Wallace、Vijay Janapa Reddi 和 Kim Hazelwood。Pin：使用动态插桩构建定制程序分析工具。ACM SIGPLAN 通讯，40（6）：190–200，2005年。[41] Siavash Zangeneh、Stephen Pruett、Sangkug Lym 和 Yale N Patt。BranchNet：使用离线深度学习预测难以预测的分支。2019年。[42] Azalia Mirhoseini、Anna Goldie、Mustafa Yazgan、Joe Wenjie Jiang、Ebrahim Songhori、Shen Wang、Young-Joon Lee、Eric Johnson、Omkar Pathak、Azade Nazi 等。一种用于快速芯片设计的图布局方法。《自然》，594（7862）：207–212，2021年。[43] Pushkar Mishra、Aleksandra Piktus、Gerard Goossen 和 Fabrizio Silvestri。节点掩码：使图神经网络具备更强的泛化能力并更好地扩展。arXiv预印本 arXiv:2001.07524，2020年。[44] Reena Panda 和 Lizy Kurian John。面向新兴大数据工作负载的代理基准测试。载于2017年第26届国际并行体系结构与编译技术会议（PACT），第105–116页。IEEE，2017年。[45] Harish Patil、Alexander Isaev、Wim Heirman、Alen Sabu、Ali Haji-abadi 和 Trevor E Carlson。ELFies：用于性能分析与仿真的可执行区域检查点。载于2021年 IEEE/ACM国际代码生成与优化研讨会（CGO），第126–136页。IEEE，2021年。[46] Harish Patil、Cristiano Pereira、Mack Stallcup、Gregory Lueck 和 James Cownie。PinPlay：一种用于并行程序确定性重放和可复现分析的框架。载于第8届 IEEE/ACM代码生成与优化国际研讨会论文集，第2–11页，2010年。[47] Erez Perelman、Greg Hamerly、Michael Van Biesbrouck、Timothy Sherwood 和 Brad Calder。使用 SimPoint 实现准确、高效的仿真。ACM SIGMETRICS性能评估评论，31（1）：318–319，2003年。[48] Pablo Prieto、Pablo Abad、Jose Angel Gregorio 和 Valentin Puente。通过异构采样基准测试快速、准确地评估处理器。*IEEE* 并行与分布式系统汇刊，32(12):2983–2995，2021年。[49] Alex Renda、Yishen Chen、Charith Mendis 和 Michael Carbin。Diff-tune：利用学习得到的可微代理模型优化 CPU 模拟器参数。载于*2020 53rd IEEE/ACM* 微体系结构国际年会 *(MICRO)*，第 442–455 页。IEEE，2020年。[50] Alen Sabu、Harish Patil、Wim Heirman 和 Trevor E Carlson。Loop-point：面向多线程应用程序的检查点驱动采样模拟。载于*2022 IEEE* 高性能计算机体系结构国际研讨会 *(HPCA)*，第 604–618 页。IEEE，2022年。[51] Yakun Sophia Shao、Sam Likun Xi、Vijayalakshmi Srinivasan、Gu-Yeon Wei 和 David Brooks。使用 gem5-Aladdin 协同设计加速器与 SoC 接口。载于*2016 49th IEEE/ACM* 微体系结构国际年会 *(MICRO)*，第 1–12 页。IEEE，2016年。[52] Timothy Sherwood、Erez Perelman 和 Brad Calder。通过基本块分布分析，在应用程序中寻找周期性行为和模拟点。载于*2001* 年国际并行体系结构与编译技术会议论文集，第 3–14 页。IEEE，2001年。[53] Zhan Shi、Akanksha Jain、Kevin Swersky、Milad Hashemi、Parthasarathy Ranganathan 和 Calvin Lin。数据预取的分层神经模型。载于第 *26* 届 *ACM* 编程语言与操作系统体系结构支持国际会议论文集，第 861–873 页，2021年。[54] Zhan Shi、Kevin Swersky、Daniel Tarlow、Parthasarathy Ranganathan 和 Milad Hashemi。通过神经代码融合学习执行。第八届国际学习表征会议 *(ICLR'20)*，2020年。[55] Sarabjeet Singh 和 Manu Awasthi。当代工作负载中统计采样的有效性：以 SPEC CPU2017 为例。载于*2019 IEEE* 工作负载特征分析国际研讨会 *(IISWC)*，第 70–80 页。IEEE，2019年。[56] W. Snyder。Verilator：快速、免费的 Verilog 模拟器。http://www.veripool.org，2012年。[57] Andrew Trask、Felix Hill、Scott E Reed、Jack Rae、Chris Dyer 和 Phil Blunsom。神经算术逻辑单元。神经信息处理系统进展，31，2018年。[58] Ashish Vaswani、Noam Shazeer、Niki Parmar、Jakob Uszkoreit、Llion Jones、Aidan N Gomez、Łukasz Kaiser 和 Illia Polosukhin。注意力就是你所需要的一切。神经信息处理系统进展，30，2017年。[59] Petar Veli ˇckovi ´c、Guillem Cucurull、Arantxa Casanova、Adriana Romero、Pietro Lio 和 Yoshua Bengio。图注意力网络。*ICLR2018*，2018年。[60] Andrew Waterman、Yunsup Lee、David Patterson、Krste Asanovic、第一卷《用户级指令集架构》、Andrew Waterman、Yunsup Lee 和 David Patterson。RISC-V 指令集手册。第一卷：用户级指令集架构'，版本，2，2014年。[61] Jiayi Wei、Maruth Goyal、Greg Durrett 和 Isil Dillig。LambdaNet：基于图神经网络的概率类型推断。*PLDI 2020*，2020年。[62] Roland E Wunderlich、Thomas F Wenisch、Babak Falsafi 和 James C Hoe。微体系结构模拟的统计采样。*ACM* 建模与计算机仿真汇刊 *(TOMACS)*，16(3):197–224，2006年。[63] Zhitao Ying、Jiaxuan You、Christopher Morris、Xiang Ren、Will Hamilton 和 Jure Leskovec。利用可微池化进行分层图表示学习。神经信息处理系统进展，31，2018年。[64] Muhan Zhang、Zhicheng Cui、Marion Neumann 和 Yixin Chen。用于图分类的端到端深度学习架构。载于第三十二届 *AAAI* 人工智能会议，2018年。[65] Yanqi Zhou、Sudip Roy、Amirali Abdolrashidi、Daniel Wong、Peter C Ma、Qiumin Xu、Ming Zhong、Hanxiao Liu、Anna Goldie、Azalia Mirhoseini 等。GDP：数据流图的广义设备放置。*ICLR2020*，2019年。