密级:_____

中国科学院大学
University of Chinese Academy of Sciences

硕士学位论文

基于排序学习算法的加速器设计空间探索

作者姓名: 鲍苗

指导教师: 陈云霁 研究员

中国科学院计算技术研究所

学位类别: 工学硕士

学科专业: 计算机系统结构

研究所 : 中国科学院计算技术研究所

2016 年 5 月

Accelerator Design Space Exploration based on learn to rank

A Dissertation Submitted to

The University of Chinese Academy of Sciences

in partial fulfillment of the requirement

for the degree of

Master of Science

in

Computer Architecture

by

Bao Miao

Dissertation Supervisor: Professor Chen Yunji

Institute of Computing Technology

Chinese Academy of Sciences

May, 2016

声　　明

我声明本论文是我本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了文中特别加以标注和致谢的地方外，本论文中不包含其他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示了谢意。

作者签名：鲍为　　日期：2016.5.30

论文版权使用授权书

本人授权中国科学院计算技术研究所可以保留并向国家有关部门或机构送交本论文的复印件和电子文档，允许本论文被查阅和借阅，可以将本论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编本论文。

（保密论文在解密后适用本授权书。）

作者签名：鲍为　　导师签名：　　日期：2016.5.30

摘 要

加速器设计空间探索是加速器设计中的首要问题，无论对函数加速器，亦或GPU，都具有重要的指导意义。加速器设计空间探索与体系结构设计空间探索类似，其基本目标是，在可选的由不同设计参数（如GPU设计中块并发数有5个不同取值以及常量缓存的大小有6个不同的取值等）构成的大规模参数空间中找到满足设计需求的参数组合。然而，可选的参数组合常常数以百万计，单纯依靠模拟的方法难以应对其所带来的设计参数灾难问题和仿真时间长的挑战。为了缓解该问题，研究人员提出了基于排序学习的方法，即通过建立一个排序模型，对不同体系结构设计参数的系统配置进行性能/功耗排序。

为了构建准确的排序模型，本文提出了基于决策树的排序学习算法RankBoostTree。算法的基本思想是，每轮迭代构建一个预测准确率并非十分高的决策树，通过在每轮迭代的过程中，加大前一轮决策树预测错误的训练集样本的权重，使得本轮迭代更加关注预测错误的样本对，从而达到快速降低预测错误率的目标。

为了验证本文方法的有效性，我们在函数加速器和GPU设计空间上比较了RankBoostTree和RankBoost以及ANN的预测准确率。相比RankBoost算法，我们的算法能够提升5.92%-11.99%的准确率，平均提高10.23%。相比ANN，我们的方法可以提升10.38%-30.37%的预测准确率，平均提高20.85%。同时，在迭代效率方面，相比RankBoost算法，在达到相同准确率的情况下，我们的方法所需要的迭代次数降低了至少5倍以上，所需要的训练数据集样本减少了66.5%。有效地验证了本文方法用于加速器设计空间探索的有效性。

关键词：加速器设计空间探索，函数加速器设计空间探索，GPU设计空间探索，排序模型，RankBoost，RankBoostTree，人工神经网络ANN

Abstract

Design Space Exploration for Accelerator is the primary problem in the field of accelerator design, it is significantly important not only for function accelerator, but also for GPU design. Similar with the basic target of architecture design space exploration, design space exploration for accelerator is to find parameter combinations to meet design requirements in the alternatives by different design parameters (such as for GPU design there are 5 optional values for threads of each block and 6 different optional values for the size of constant cache etc.) consisting of a large-scale parameter space as well. However, the optional parameter combinations often number in the millions, it is difficult to deal with the problem and challenge design parameters disasters and long simulation time bring in if only relying on simulation. To alleviate the problem and challenge mentioned above, researchers have proposed a method based on ranking, through the establishment of a model of the system , rank system performance / power order of different configuration represented by architecture design parameters.

In order to build an accurate ranking model, this thesis proposes a ranking algorithm based on Boosting decision tree called RankBoostTree. The basic idea of the algorithm is that each iteration construct a prediction accuracy rate not excellent enough, through each iteration process, increase the former decision tree prediction of the training set sample weight, so that the current round of iteration we will pay more attention to the samples predicted incorrectly, so as to achieve rapid reduction of target prediction error rate.

In order to verify the effectiveness of the method this thesis proposes, we compared the prediction accuracy of RankBoostTree , RankBoost and ANN on a general purpose processor core, core and GPU accelerator design space. Compared to RankBoost, our method can improve the prediction of 5.92%-11.99% accuracy rate, an average of 10.23%. Compared to ANN, our method can improve the prediction of10.38%-30.37% accuracy rate, an average of 20.85%. At the same time, the efficiency of the iteration, RankBoost algorithm, in the case to achieve the same accuracy of our method iterations required to reduce by more than at least five times, samples of the training data set required for 66.5% reduction. Effectively demonstrate the effectiveness of methods for architecture design space exploration.

Keywords: Design Space Exploration for Accelerator, Design Space Exploration for

function accelerator, GPU design space exploration, ranking model, RankBoost, RankBoostTree, ANN (Artificial Neural Network)

目 录

图目录

表目录

第1章 引言

1.1 本文的研究背景和意义

1.1.1 加速器设计空间探索的意义和挑战

随着登纳德缩放比例定律[1]（Dennard Scaling，指的是晶体管尺寸变小，功耗会同比变小）走到尽头，依靠传统技术方法提升性能降低功率已经不复存在。与此同时，随着晶体管密度不断增加，导致了暗硅问题，芯片集成更多的晶体管，在任何时间点都能够达到满功率状态[2][3]。为了克服这些挑战，在数据路径和定制的特定算法或应用程序的控制电路的硬件加速形态已经浮出水面，并且是一个很有前景的方法，这种方法相比通用的解决方案，能够带来几个数量级的性能和功耗提升比。由 CPU、加速器以及 GPU 组成的可定制结构，已经广泛运用于移动终端中，同时，也开始出现在服务器和台式机中。

这一解决方案的发展，自然会推动未来系统中加速器所占的比重以及丰富可定制加速器的多样性，如下图 1-1 所示，是未来的异构系统结构。按照这样的发展趋势，整个系统的全面评估和权衡对于系统设计者来说，是至关重要且难度很大的挑战。然而，目前的定制的体系结构只包含屈指可数的加速器，这是因为，庞大的设计空间探索目前还无法实现。

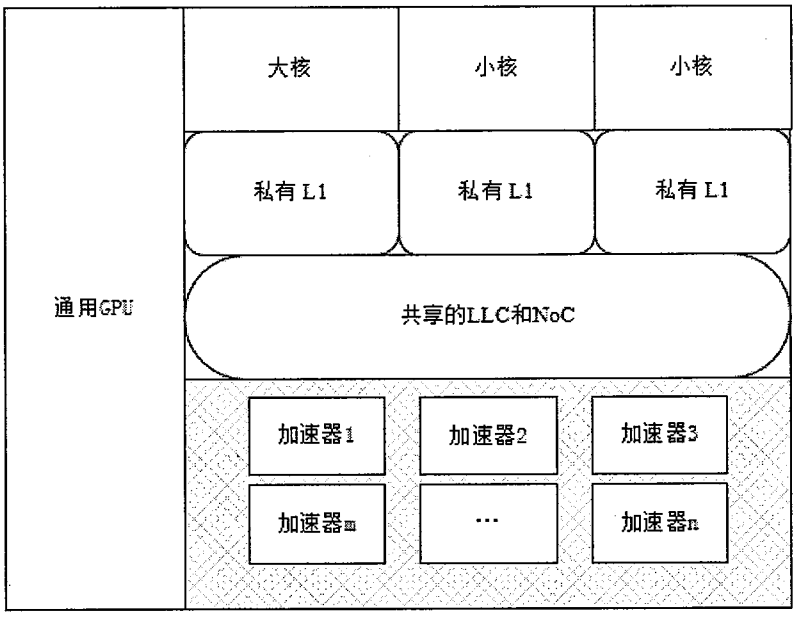


图 1-1 未来的异构系统[4]

设计空间探索通过量化设计参数对评估指标的影响来分析设计参数和评估指标之间

的关系，通常，这些分析都是利用时钟精确级的模拟器来完成的。通过在目标机上进行cycle-by-cycle 级别的仿真，预测体系结构参数变化对性能的影响进而寻找满足不同性能/开销/复杂度/功耗约束下的设计空间子集。然而，在实际运用过程中，会遇到以下两大挑战：

1）很多场景下，可变参数的数量和可取值使得设计空间太大难以完全探索评估。比如，在 C. Dubach 等人提到的微体系结构设计空间[5]，见下表 1- 1，最左边一列代表可变/可配置的参数名称，中间一列表示每个参数的可取值，第三列指的是每个参数在可取值范围内一共可以取多少个参数值。从表中数据可以看到，在一个有 13 个可配置参数的微体系结构设计空间中，一共有 630 亿组可探索的配置，如此庞大的设计空间探索，是非常费时费力的过程。

2）另外，时钟精确级的模拟仿真由于需要详细的微体系结构建模并且在真实负载下模拟大量的 benchmark 也同时加大了实现的难度。

表 1- 1　微体系结构设计空间[5]

参数	参数可取值	可取值数
Machine width	2,4,6,8	4
ROB size	32,40,48,…,160	17
IQ size	8,16,24, …,80	10
LSQ size	8,16,24, …,80	10
RF sizes	40,48,56, …,160	16
RF read ports	2,4,6, …,16	8
RF write ports	1,2,3, …,8	8
Gshare size	1K,2K,4K,8K,16K,32K	6
BTB size	1K,2K,4K	3
Branches allowed	8,16,24,32	4
L1 Icache size	8KB,16KB,32KB,64KB,128KB	5
L1 Dcache size	8KB,16KB,32KB,64KB,128KB	5
L2 Ucache size	0.25MB,0.5MB,1MB,2MB,4MB	5
Total		$6.3*10^{10}$

1.1.2 设计空间探索两大挑战的解决方案

为了解决上述设计空间探索的两大挑战，许多研究学者和相关机构进行了大量的研究。解决思路主要分为三类，第一类是不采用或者只用很少的时钟精确级的模拟器进行模拟仿真，改用其它方式建立设计参数与所研究体系结构行为的关系模型，这类典型方案的代表有分析模型；第二类是从模拟仿真本身入手，想方设法加快模拟仿真的速度，降低仿真的时间开销，比如，将一些统计采样技巧运用到模拟仿真中；第三类是把机器

学习的方法运用到设计空间探索中，仅需要少量的模拟仿真获得训练数据，基于训练数据建立机器学习模型，模型能够将设计参数映射到所研究体系结构行为值。下面，分别介绍这三类典型的解决方案：

(1) 分析模型

分析模型获取体系结构的知识并且允许用少量的甚至不用模拟仿真来评估体系结构行为。Sun 等人提出一种称为 Moguls 的性能分析模型来帮助体系结构设计者快速地探索存储层级的设计空间[6]。Chen 和 Aamodt 利用马尔可夫链来准确地模拟运行在多核体系结构上的多线程程序的吞吐量[7]。Chen 和 Aamod 后面进一步扩展他们的模型来预测多程序多核处理器的吞吐量[8]。Noonburg 和 Shen 通过概率地提取程序并行和机器并行特征，来评估超标量处理器的性能[9]。Karkhanis 和 Smith 提出了一种面向超标量处理器的一阶性能模型，一阶模型的思想是，在理想性能的基础上加上由于缺失事件引起的性能损失，缺失事件包括分支预测错误，指令 cache 缺失，数据 cache 缺失[10]。Chandra 等人提出一种概率模型来预测由运行在单芯片多处理器核上的两个不同线程的 cache 争用引起的 cache 缺失[11]。Eyerman 等人进一步扩展了一阶模型，将指令执行的流程划分为多个由不同缺失事件隔开的区间[12]。Eklov 等人提出了一种称为 StatCC 的统计 cache 争用模型，StatCC 能够预测一组协作运行的线程的性能[54]。Nair 等人提出一种一阶机械分析模型，这种模型能够通过低代价的分析，来评估正确路径状态下的占用，从而计算得到体系结构易受损因子[13]。分析模型并不是仿真密集型的，但是在分析模型中很难获取和集成大量的相互作用的参数。虽然如此，分析模型由于在集成领域专业知识方面的优势，仍然可以作为其他模型的补充。

(2) 快速仿真技巧

近些年来，一些基于统计采样的技术用于降低模拟器的仿真时间，譬如 SimPoint[14] 和 SMARTS[15]。另外，Iyengar 等人提出了一种表示改进轨迹（Refined Traces）的衡量指标，并且发展出一种创新的基于图的启发式算法产生更好的改进轨迹[16]。Nussbaum 和 Smith 提出从程序的指令轨迹提取程序的本质特征，用这些特征，就能够产生简单的合成指令轨迹，用作模拟仿真[17]。Eeckhout 等人提出了一种改进的统计模拟仿真框架，采用统计流图来精确地刻画程序的控制流行为[18]。为了减少单芯片多处理器核设计的模拟仿真开销，Genbrugge 和 Eeckhout 提出了多种统计量来获取 cache 存取行为并且共享对于在单芯片多处理器核上运行的多程序同时运行的程序至关重要的临界资源[19]。Hughes 和 Li 提出当构建合成多线程程序时，利用统计特征量来获取共享存储和线程间同步的行为[57]。同样，还有一类方法是，直接从初始程序提取几个短而具有代表性的指令阶段。Simpoint[20]把程序的执行阶段特征化为基本块向量，再把程序的各个执行阶

段进行聚类，然后将每个聚类中心作为每个模拟采样阶段的代表。SMARTS[21]基于统计采样理论，从原始程序中选择有代表性的指令片段，这种方法能够确定样本（比如，指令片段）的数量，确保能够实现用户指定的性能置信区间。虽然这些策略能够增加给定时间下的仿真数量，但是，在庞大的设计空间下，全面彻底的评估依然是无法实现的。

(3) 机器学习方法

为了解决设计空间探索空间维数灾难和仿真时间长这两大难题，一些研究提出了借助机器学习方法来评估庞大的设计空间[22][23][24][25]，为设计空间探索提供了新的解决方案。这种方法只需要少量的模拟仿真，通过运行基准程序来快速获取仿真数据，这些数据用来训练一个预测器。预测器能够预测模拟仿真没有覆盖到的设计空间，而不需要再做进一步的模拟仿真。同时，少量的模拟还可以配合快速仿真方法，降低每次仿真的开销，从而能够降低得到训练集的代价。基于机器学习方法的设计空间探索大体做法是通过训练一个机器学习模型来刻画设计参数和所研究的目标体系结构行为之间的复杂关系，并利用该模型来预测不同设计参数配置下的目标体系结构行为的具体数值。

1.2 相关研究

在过去的几年里，基于机器学习算法的设计空间探索取得了不错的进展。基于机器学习算法的设计空间探索主流思想分为两类，一类是基于回归的机器学习方法；另一类是将设计空间探索归结为排序问题。

基于回归模型的设计空间探索，利用训练好的回归模型可以快速预测没有被模拟仿真的体系结构配置下的性能，而不需要进行额外的模拟仿真。通过比较预测得到的不同配置下的性能，体系结构设计者就可以选择符合条件的配置。在过去的几年中，有很多出色的基于回归模型的设计空间探索研究工作。国内的相关研究，如郭崎等人受基于分歧的半监督学习范式[26]的启发，设计了一种类似于协作训练的算法来进一步加强基于回归设计空间模型的精确度[27]。在国际上，Ipek [28][29]等人提出利用 ANN（Artificial Neural Network，人工神经网络），Lee 和 Brooks 提出用曲线函数[30]来建模超标量或者单芯片多处理器核设计场景。和 ANN 相比，曲线函数的结果更容易解释，但是也意味着需要更多的人工干预；两种方法通过实验发现在预测准确率方面，表现相当。在另外一部分工作中，Lee 和 Brooks 通过 Pareto 前沿分析、流水线深度分析、以及多处理器层级分析，进一步证明了曲线回归模型的有效性[31]。Joseph 等人[32]采用线性回归方法构建线性预测模型，预测不同体系结构配置下的性能。然而，这些线性模型，不能刻画非线性的复杂设计空间的响应行为。因此，随后 Joseph 等人提出了采用径向基函数（RBF）网络[33]，RBF 是一种神经网络，能够构建处理器性能的非线性模型。Azizi 等人提出了基于多项式的面向联合电路-体系结构设计空间的方法来特征化并且预测性能-功耗的

平衡[34]。Dubach 等人采用 SVM（Support Vector Machine, 支持向量机）来建模联合体系结构-编译器设计空间，训练好的 SVM 模型能够预测不同体系结构配置下的编译器性能[35]。

上述针对设计空间探索所提出的回归方法已经被证实能够准确的预测处理器的响应，然而，这些工作无法提供明确的方法来预测不同配置之间的相对排序。下面介绍基于排序学习的设计空间探索研究。

陈天石等人突破传统体系结构设计空间探索的方法，将体系结构设计空间探索问题归结为一个回归问题，而是根据体系结构设计空间探索的实际需求，将该问题归结为一个不同体系结构设计配置下的排序问题。他们提出了 ArchRanker[36]，ArchRanker 是一个面向体系结构设计空间的排序器，这是将体系结构设计空间探索问题归结为排序问题的首次尝试。ArchRanker 实现了一个基于决策桩的 RankBoost 算法，并应用于多核以及众核设计空间探索，性能表现良好。这开辟了体系结构设计空间探索的新视角，为体系结构设计空间探索提供了新方法。

1.3 本文主要研究内容

本文基于 ArchRanker 的工作，作了进一步的探索和拓展。ArchRanker 采用的是基于决策桩作弱排序器的 RankBoost 算法，并应用于多核以及众核设计空间探索。本文实现了该算法，并应用于加速器设计空间探索，但是在实验中发现，基于决策桩的弱排序器，决策简单条件单一，每次只能运用一个维度的数据做决策，影响决策性能；同时，由于每次只能利用一个维度的数据，导致迭代效率较低。因而，本文改进了基于决策桩的排序学习算法 RankBoost，提出了一种基于决策树作弱排序器算法 RankBoostTree，并将基于决策树的弱排序器算法 RankBoostTree 应用于加速器设计空间探索中。本文的加速器设计空间探索，包括函数加速器设计空间探索和 GPU 设计空间探索。同时，本文分别对比了基于决策树作弱排序器的 RankBoostTree 算法和基于决策桩作弱排序器的 RankBoost 算法，以及基于决策树作弱排序器的 RankBoostTree 算法和传统回归方法中表现最为突出的神经网络模型 ANN 的排序关系预测准确率；并且，比较了 RankoostTree 算法和 RankBoost 算法的迭代效率。

本文组织结构：

第 1 章 引言。本章简单扼要地介绍了论文的研究背景与意义、当前相关领域的研究现状以及本文的主要研究内容。

第 2 章 相关背景知识介绍。本章第一节初步介绍了排序学习算法的概念，三大类排序学习算法，基于 Pointwise 的排序学习算法、基于 Pairwise 的排序学习算法和基于

Listwise 的排序学习算法，并简单比较了三大类排序学习算法的优缺点；然后介绍了基于 Pairwise 的排序学习算法 RankBoost 的原理和算法流程。第二节介绍了陈天石等人的工作 ArchRanker[36]，ArchRanker 是面向体系结构设计空间的排序学习器，实现了 RankBoost 算法。第三节介绍了常用的模拟仿真平台，包括 Gem5 模拟器、Sniper 模拟器、Aladdin 模拟器以及 GPGPU-Sim 模拟器。最后一小节总结了本章的主要工作。

第 3 章 基于 RankBoostTree 算法的加速器设计空间探索。第一节从获取仿真数据和基于RankBoostTree算法进行加速器设计空间探索两个阶段描述了RankBoostTree算法应用于加速器设计空间探索的工作原理和过程。第二节分析了 RankBoost 算法用决策桩作弱排序器的不足，引出用决策树作弱排序器的 RankBoostTree 算法。第三节介绍了 RankBoostTree 的实现原理；第四节介绍了 RankBoostTree 算法实现的整体框架并详细描述了 RankBoostTree 算法各个模块的设计、实现的功能以及核心功能点的实现细节。最后一节对本章作了小结。

第 4 章 实验和评估。本章在函数加速器设计空间上以及 GPU 设计空间上分别验证了 RankBoostTree 算法的预测准确率，并与 RankBoost 算法以及传统回归方法中表现最为突出的神经网络模型 ANN 进行了对比；同时，比较了改进后的 RankBoostTree 算法和 RankBoost 算法的迭代效率。

第 5 章 结束语。本章总结了本文的工作，并分析了未来可以探索的方向以及可以继续深入的工作。

第2章 相关背景知识介绍

本章将介绍排序学习算法的概念，三类主要的排序学习算法，基于 Pointwise 的排序学习算法、基于 Pairwise 的排序学习算法和基于 Listwise 的排序学习算法；然后介绍一种基于 Pairwise 的排序学习算法 RankBoost；最后介绍陈天石等人的研究工作 ArchRanker[36]，并详细描述 ArchRanker 如何将算法 RankBoost 应用于体系结构设计空间探索中。

2.1 排序学习算法

2.1.1 概念

LTR（Learning To Rank），即学习排序或排序学习，是一种有监督学习（Supervised-Learning）的排序算法，算法通过训练一个排序模型来完成排序任务[58]。学习排序算法在许多应用领域，比如，信息检索，自然语言处理和数据挖掘，都具有非常重要的作用，具体的应用例子有信息检索中排序返回的文档，推荐系统中的候选产品，用户排序，机器翻译中的排序候选翻译结果等[37]。同时，许多研究机构和组织团队已经开始研究排序学习问题，并取得了长足的进展[38]。

对于传统的排序模型，基本是由多种算法组合在一起实现的，然后再对其不断细化。但是，如果系统达到一定规模时，若模型中参数过多，调模型参数就会成为一项非常大的挑战，人工来调整参数已经几乎成为不可能完成的任务，同时，过分调整参数拟合数据，也极有可能导致过拟合的问题。因而，研究者很快想到，机器学习（Machine Learning）的方法也许可能用以解决此类问题。所以，学习排序（Learning To Rank）算法随之问世。

机器学习方法由于其本身的特性，很适合多种特征的组合；同时，机器学习理论基础发展得比较成熟，并且这门学科也有深厚的理论基础。机器学习学习到的参数一般都是通过迭代优化来调整的，它具有一套成熟的理论来解决在训练过程中可能出现的过拟合等问题。

2.1.2 三类排序学习算法

下面，本文主要介绍目前主流的三种排序方法，这三种排序算法分别是基于 Pointwise 的排序算法、基于 Pairwise 的排序算法和基于 Listwise 的排序算法。

(1) 基于 Pointwise 的排序算法[39]

基于 Pointwise 的排序学习算法最早由 Fuhr 和 Norbert 等人于 1989 年 提出[39]。基于 Pointwise 的排序方法，顾名思义，就是将训练样本作为单独的点（Point）一个一个分别计算的，借助这种思维策略，我们就可以将排序问题转变为机器学习中的典型问题，

如二值分类问题、回归问题和多值分类问题。

基于 Pointwise 排序算法的三类机器学习问题归纳如下：

表 2-1 基于 Pointwise 排序算法的三类机器学习问题

	基于 Pointwise 的排序方法		
	回归问题	分类问题	有序分类回归问题
输入空间	单个文档 Document x_j		
输出空间	实数数值	无序类	有序类
假设空间	假设函数 f(x)		
损失函数	回归损失函数	分类损失函数	有序分类回归损失函数
	L（f; x_j, y_j）		

基于 Pointwise 的排序方法普遍应用的典型机器学习模型是回归、分类和有序回归[59]。机器学习模型的重要组成包括输入空间、输出空间、假设空间和损失函数。确定了这几个重要的组成部分，也就构建了机器学习模型。

以文档排序为例，针对文档排序模型的输入是<Query，Document>对，但是因为对于一个 Document 集合，其 Query 是一样的，只是和 Document 集中的不同的 Document 组合，因而上表中将输入简写为单个 Document。回归问题，通过回归函数得到一个实数，因而，无论是线性回归还是逻辑回归，都能够通过模型预测得到一个实值预测值，用这个预测值来做排序。分类问题，通过模型将样本分成无序的类别，不同的类别得到相对的排序信息，比如分类的类别可以是{“相关”，“不相关”}。有序分类回归问题，通过模型计算，我们能够将样本分为{“Perfect”, “Excellent”, “Good”, “Bad”}中的某一类。

基于 Pointwise 排序算法的不足有以下两点：

(a) 基于 Pointwise 的排序方法，由于输入是<Query，Document>对，即输入样本包括查询信息，所以在建立模型时，也将考虑查询信息。这样带来的问题是，在训练集中，如果某个查询下的文档数量占绝对优势，则建立的模型就会过分拟合这个查询的样本，过度依赖某一个特定的查询，模型的推广性就比较差。

(b) 对于回归或分类问题，模型的损失函数会尽可能覆盖到全部样本，即尽可能拟合绝大部分数据，因而，有可能在顾全大局的时候，会提高本应当排在前面的样本的损失，而降低本应当排在后面的样本的损失，从而降低总体样本的损失。对于我们的排序问题，每个样本其实是带权重的样本，并不应该平等对待，所以，这种方法在学习排序中存在不足。

(2) 基于 Pairwise 的排序方法[40]

基于 Pairwise 的排序学习算法，在 2000 年由 Ralf Herbrich，Thore Graepel，和 Klaus Obermayer 首次提出[40]。基于 Pairwise 的排序方法，输入不再是<Query，Document>对，而是同一个 Query 下的所有文档之间两两组合形成文档对，<Document1，Document2>。对于每个文档对，<Document1，Document2>，进行排序，决策出哪个 Document 和 Query 更加接近，得到<Documenti，Documentj>的偏序关系对。算法不再与单个文档结合，而是与一个文档对，如<d1，d2>结合，计算出 d1 和 d2 哪个与 Query 最接近，然后对同一个 Query 下的所有文档进行两两排序，最终得到某个 Query 下的所有文档的全序关系。

举例说明，假设在同一个 Query 下，文档 d1 的相关性大于文档 d2，那么我们就可以这样处理数据，将 d1-d2 标记为+1，d2-d1 标记为-1，这样，我们就把排序问题转变成一个分类或回归问题了。

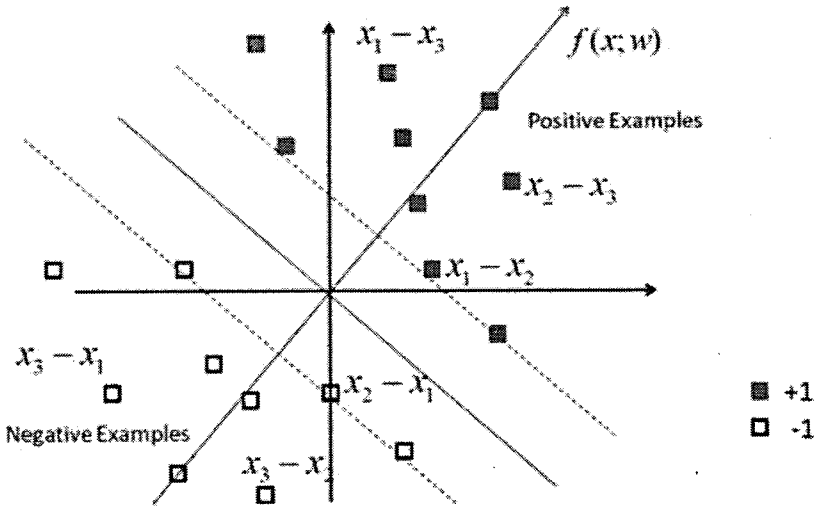


图 2- 1 SVM(Support Vector Machine, 支持向量机)分类[41]

如上图 2- 1 所示，对于分类或者回归问题，可以采用支持向量机或者其它机器学习算法处理。当采用 SVM (Support Vector Machine, 支持向量机) 进行最大间隔分类或者回归时，只需要把注意力放在支持向量上，而支持向量本质上只是两个文档的对比，并不依赖于查询信息，这样以来，也就消除了基于 Pointwise 的排序算法中对查询依赖的弊端。

基于 Pairwise 排序算法的缺点在于，由于算法是基于排序对的，因而要对输入数据进行两两组合构成偏序对，这样，会导致样本空间急剧膨胀，由原来的 n 个样本扩充到 n • (n-1)/2 个样本，大大增加计算量。

(3) 基于 Listwise 的排序方法[42]

基于 Listwise 的排序学习算法，在 2007 年首次出现在 Jun Xu，Hang Li 等人 AdaRank 的工作中[42]。基于 Listwise 的排序学习算法不再将排序问题形式化成典型的机器学习中的回归或者分类问题，而是专注于问题本身，通过优化排序结果本身来达到排序优化

的目标。基于 Listwise 的排序学习算法，顾名思义，算法的输入不像 Pointwise 是单个样本或者 Pairwise 是样本对，它的输入是一组文档列表。算法根据下面的公式计算得到每一个列表的分值，获得最大分值的列表为目标输出列表[60]。

$$P(\pi)=\prod_{i=1}^{n}\frac{S_{\pi(i)}}{\sum_{j=1}^{n}S_{\pi(j)}}$$
$$P(ABC)=\frac{S_A}{S_A+S_B+S_C}\bullet\frac{S_B}{S_B+S_C}\bullet\frac{S_C}{S_C} \quad (2.1)$$

其中，$S_A/(S_A+S_B+S_C)$ 表示A排在第一的概率；

$S_B/(S_B+S_C)$ 表示A排在第一位的条件下，B排在第二位的概率；

S_C/S_C 表示A排在第一位，B排在第二位的条件下，C排在第三位的概率。

算法的核心是排序函数 rank()，这个排序函数基于最小化损失得到，通过排序函数 rank()，将新输入的样本排序成合适的序列。

基于 Listwise 算法的不足之处在于，算法复杂度高，计算量大；同时，构造 rank() 函数的难度大。

2.1.3 RankBoost 算法[43]

RankBoost 排序学习算法是一种基于 Pairwise 的排序学习算法，本小节主要描述 RankBoost 的算法思想和算法流程。

以体系结构设计空间探索为例，P 代表体系结构设计空间，x 表示体系结构设计空间 P 中的一组参数配置，R(x)表示在体系结构参数配置 x 下的体系结构响应值，（如系统性能、功耗等）。在本文的算法流程中，给定一组模拟仿真参数配置 $p=\{x_1, x_2, \cdots\cdots, x_n\}$，其中，$x_1, x_2, \cdots\cdots, x_n$ 并不是无序的，它们是按照其体系结构响应值的偏序关系排序的。比如，对于功耗这个指标，若 $R(x_i)<R(x_j)$，则 x_i 排在 x_j 的前面，i，j 取值自(1，2，……，n)。特别地，如果 $R(x_i)=R(x_j)$，则 x_i 和 x_j 没有偏序关系。按照这种处理数据的方式，对于 n 个体系结构参数配置，可以得到 $n\cdot(n-1)/2$ 个配置对。训练集的样本格式如下：

$$Train=\{(x_1,x_2);R(x_1)<R(x_2)\} \quad (2.2)$$

上述偏序对（Train 集合的数据形式）作为输入，基于排序学习的设计空间探索的最终结果是得到一个排序函数 $H:\Phi\rightarrow R$，这个排序函数 $h(x)$ 给设计空间中的每个参数配置 x_i 计算一个相对数值 $h(x_i)$，根据每个 x 的 h(x)值作排序。即若 $h(x_i)<h(x_j)$，则 x_i 排在 x_j 的前面。由此，便可以得到所有 x 的全序排序列表，从而找到满足设计要求的设计参数配置。

下面介绍这个过程的细节。

RankBoost 算法通过一次次迭代训练得到多个弱排序器，h_1，h_2，……h_T，最后，将训练好的 T 个弱排序器线性组合得到最终的排序函数 H。下图 2-2 显示了这个算法的伪代码[43]。伪代码中，算法的输入是体系结构参数配置的偏序对，形如上述提到的训练集 Train。为了最小化训练集的排序误差，第 t 个弱排序器 h_t 在算法的第 t 轮迭代过程中更加关注于前一个弱排序器 h_{t-1} 没有排正确的偏序对。这样的在线学习通过动态调整离散概率分布，即带有权重的样本对的分布 D 来实现的，下图中的 D_t 代表的就是第 t 轮迭代样本对的权重分布。

输入：

体系结构不同设计参数配置样本集合 S

迭代次数 T

$D_t(t=1,2,\cdots,T)$ 第 t 轮迭代样本对的权重分布

$\alpha_t(t=1,2,\cdots,T)$ 第 t 轮迭代的控制参数

begin

for t=1,2,……,T do

基于样本对权重分布 D_t 训练弱排序器 $h_t(x)$

$$\alpha_t=\frac{1}{2}\ln\left(\frac{1+r_t}{1-r_t}\right)$$

$\forall(x_1,x_2)\in S\times S$：

$$D_{t+1}(x_1,x_2)=D_t(x_1,x_2)\exp\left(\alpha_t\left(h_t(x_2)-h_t(x_1)\right)\right)/Z_t$$

end

最终的排序模型：$H(x)=\sum_{t=1}^{T}\alpha_t h_t(x)$

end

图 2-2 RankBoost 算法的过程[43]

那么，下面的重点就在于如何初始化和更新第 t 轮迭代的样本对权重分布 D_t 了。设 S 表示体系结构参数配置样本的集合，R(x)表示参数配置样本 x 下的体系结构响应值，(x_1, x_2)表示参数配置样本 x_1 和 x_2。对于所有符合所研究的偏序关系的参数配置样本对 (x_1, x_2)，初始时都赋一致的均匀的权重，即如下公式所示[43]：

$$D_1=\begin{cases}1/g, & R(x_1)<R(x_2)\\ 0, & R(x_1)\geq R(x_2)\end{cases} \tag{2.3}$$

其中，$g=\left|\left\{(x_1,x_2)\in P\times P;\ R(x_1)<R(x_2)\right\}\right|$ 是归一化因子，归一化因子用来确保参数配置偏序对的正确概率分布。有了归一化因子，易得：

$$\sum_{x_1 \in P,\ x_2 \in P} D_1(x_1, x_2) = 1 \tag{2.4}$$

由上述参数配置偏序对的初始权重分布公式可以看出，如果我们已经考虑了样本对(x_1, x_2)，那么我们就不再重复地考虑样本对(x_2, x_1)。在第 t 轮迭代结束后，RankBoost 算法需要更新每个参数配置偏序对的分布权重 D_{t+1}，更新公式如下[43]：

$$D_{t+1}(x_1, x_2) = D_t(x_1, x_2) \exp\left(\alpha_t \left(h_t(x_1) - h_t(x_2)\right)\right) / Z_t \tag{2.5}$$

其中，Z_t 同样也是归一化因子，用来确保参数配置偏序对的权重分布 D_{t+1} 是一个正确的概率分布，同时，公式中的权重因子 α_t，它的定义见 RankBoost 算法的伪代码，表示的是弱排序器 h_t 的系数。参数配置偏序对的权重更新规则可以直观上这样解释。我们讨论当 $\alpha_t > 0$ 的时候，通常情况下，α_t 也的确是大于 0 的。如果第 t 轮迭代得到的弱排序器正确地预测了两个体系结构参数配置 x_1 和 x_2 的偏序关系，比如说，体系结构参数配置 x_1 在所研究的体系结构问题中排在 x_2 前面，第 t 个弱排序器排序的结果也是 $h_t(x_1) < h_t(x_2)$，那么，偏序对 (x_1, x_2) 的权重分布就要更新，即下一轮迭代的偏序对权重分布 $D_{t+1}(x_1, x_2)$ 会降低。否则，如果第 t 个弱排序器预测错了两个体系结构参数配置 x_1 和 x_2 的偏序关系，则该偏序对的权重分布会加大。

基于上述偏序对的权重更新公式的意义，下一轮迭代训练弱排序器 h_{t+1} 时，就会更多的关注排序错误的偏序对，对前一个弱排序器排序正确的偏序对，则降低其权重分布。

在第 t 轮迭代，RankBoost 算法基于权重分布 D_t 训练得到弱排序器 h_t，弱排序器 h_t 更多地关注于前一个弱排序器 h_{t-1} 没有排序正确的偏序对。RankBoost 算法最常用的最主流的做法，每个弱排序器都是一个简单的学习模型，这个学习模型一般是一个决策桩[43][44]，决策桩的数学定义形式如下[43]：

$$h(x) = \begin{cases} 1, & \text{若 } f_i(x) > \theta \\ 0, & \text{其他} \end{cases} \tag{2.6}$$

其中，$f_i(x)$ 表示体系结构设计参数配置 x 的第 i 个分量，θ 是一个常量阈值。为了训练得到一个弱排序器，按照 Freund 论文提出的方法[43]，我们需要一次一次地迭代寻找最佳的特征维数 i 和常量阈值 θ 的组合。在 RankBoost 算法迭代 T 次之后，就能够训练得到 T 个弱排序器，每个弱排序器就是一个上述的决策桩，分别用 $h_1, h_2, \cdots, h_T$ 表示，最终，得到的排序模型 H 为[43]：

$$H(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \tag{2.7}$$

H(x)能够为每一个输入设计参数配置 x 计算一个排序分值，这样，不同的设计参数

配置就能够根据其排序分值进行排序。多个弱排序器的线性组合能够胜任非线性映射所能解决的问题[45][46]。

2.2 ArchRanker[36]

传统的基于回归模型的体系结构设计空间探索方法专注于预测特定体系结构配置下的机器性能，而不关心两种不同体系结构配置下哪一种配置的性能更好。但是实际上，在体系结构设计空间探索中，我们更应该关心的，或者说更有价值、更直接的问题应该是解决不同体系结构参数配置下，哪种参数组合的性能更好的问题，即我们更应该关注的是不同体系结构参数配置下的性能排序关系，而不是致力于提高特定体系结构参数配置下性能预测的准确率。另外，准确预测特定体系结构参数配置下的性能是一个非常困难的问题，它需要基于大量的训练仿真才能够达到相当的准确率。尽管耗费如此的代价，预测得到的性能值也不能有效预测两种不同体系结构参数配置的性能偏序关系。比如说，考虑两组体系结构配置 x_1 和 x_2，x_1 的 IPC（Instructions Per Cycle）值是 2.00，x_2 的 IPC 值为 2.10，回归预测模型预测得到 x_1 的 IPC 值为 2.10，预测 x_2 的 IPC 值是 2.05，模型的总体误差仅为 3.66%，但是，模型预测的结果是，x_1 的性能比 x_2 好，这和真实的性能排序正好相反。而在实际应用中，显然，x_1 和 x_2 的性能排序关系对指导体系结构设计更加有意义。

基于上述的原因，陈天石等人提出 ArchRanker[36]。该工作考虑将体系结构空间探索问题归结为排序问题更加直接且更具有现实意义。排序学习算法对于已得到样本的全序或者部分有序为目标的排序问题有较好的效果，体系结构设计空间探索也正好是一个典型的排序问题。因而，用排序学习的方法来解决体系结构设计空间探索问题是一个行之有效的方法。

ArchRanker 是一个面向体系结构设计空间探索的排序学习器，ArchRanker 的工作主要分为两个阶段，训练阶段和测试阶段。其结构图如下图 2- 3所示：

ArchRanker 工作过程描述如下：

(1) 采集数据

将不同体系结构设计参数配置输入模拟器，通过模拟器仿真得到体系结构响应值，比如性能 IPC 或者功耗等。这一步是获取数据集。

(2) 划分训练集和测试集

(a) 将第一步得到的数据集，划分为两部分，一部分作训练集，另外一部分作测试集；

(b) 处理训练集的数据。把训练集的样本根据体系结构响应值的偏序关系构造基于 Pairwise 的偏序对。比如，训练集一共有 n 个样本，则构造 $n \cdot (n-1)/2$ 个样本对，每个样本对的偏序关系是一致的，即若 $Rank(x_i) < Rank(x_j)$（在

体系结构所研究的问题排序中，x_i 排在 x_j 的前面），则构造的偏序对记为 $<x_i, x_j>\left(i, j \in (1, 2, \cdots, n)\right)$；

(3) 针对训练数据集，用基于 Pairwise 的排序算法 RankBoost 算法训练排序模型；

(4) 将测试数据集输入训练好的排序模型中，通过排序模型，得到测试数据集中每组体系结构参数配置的相对排序分数，得到测试数据集设计空间的全序排序。

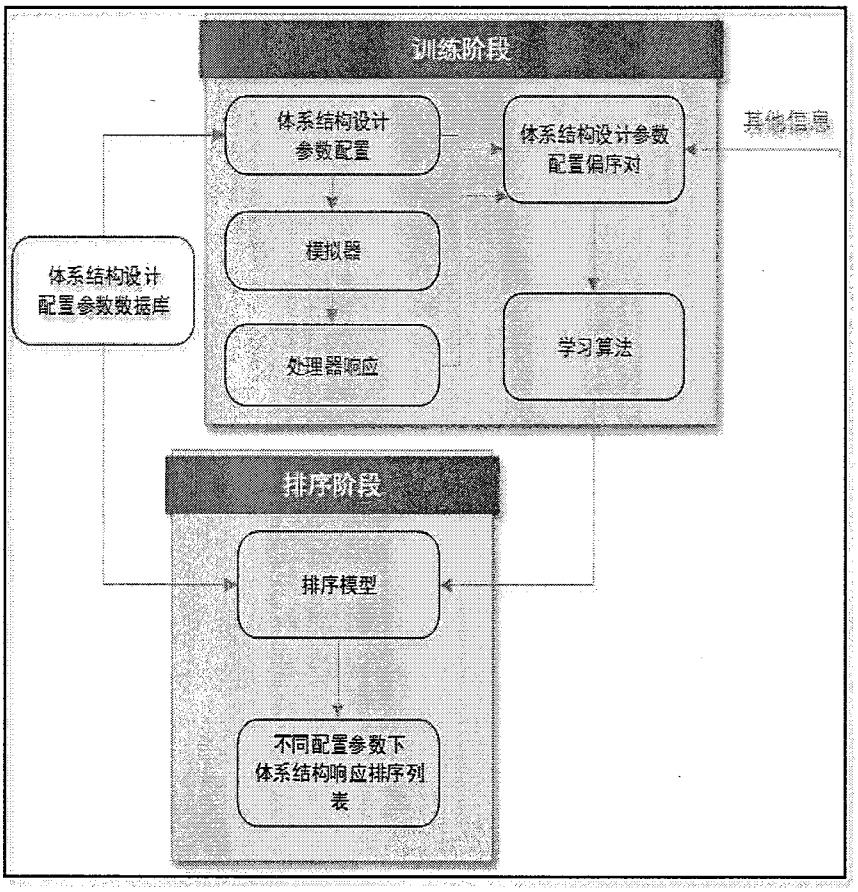


图 2-3 面向体系结构设计空间的学习排序器[36]

2.3 常用模拟器介绍

基于机器学习方法的设计空间探索可以用来解决设计空间探索设计空间灾难和仿真时间长的挑战，然而，基于机器学习的方法仍然需要进行少量的模拟仿真获得训练数据，基于训练数据建立机器学习模型。下面介绍体系结构领域常用的模拟器。

(1) Gem5

Gem5 是应用于计算机系统结构研究的模块化的工具平台，适用系统级架构以及处理器微体系结构[47]。Gem5 是由 Michigan 大学、普林斯顿大学以及麻省理工大学等学术领军团队以及 AMD、MIPS 和 Google 等这样的商业巨头共同开发的平台[48]。Gem5 的特点是，高可配置性，支持多种类型的 ISA（Instruction Set Architecture, 指令集架构），并且提供了多种 CPU 的模型。

Gem5 具有可配置性，通过基于 python 脚本的配置文件对体系结构设计参数配置，实现不同体系结构配置的模拟仿真。

Gem5 目前支持的 ISA 以及加载操作系统情况见下表 2-2：

Gem5 不仅提供了多样的 CPU 模型，还包括丰富的存储模型和系统模型。具体的模型见下表 2-3：

表 2-2 Gem5 支持的 ISA 及是否支持加载操作系统

ISA	是否支持加载操作系统
X86	是
MIPS	否
ALPHA	是
SPARC	否
ARM	是
Power	否

表 2-3 支持的 CPU 模型

CPU 模型	模型复杂程度
Atomic	简单
Timing	较简单
In-order	较复杂
O3（Out of Order）	复杂

支持的存储模型有 Classic 及 Ruby，支持的系统模型有 SE（System Emulation，系统调用模式）和 FS（Full System，全系统模式）。

(2) Sniper[49]

Sniper 是下一代高速、并行且具有较高准确性的 x86 模拟器[49]。该多核模拟器基于区间核心模型和 Graphite 仿真框架,允许快速准确的模拟仿真并支持模拟速度和精度之间的权衡，当探索不同的同构或者异构多核架构时，允许一定范围内模拟选项[49]。

Sniper 模拟器不仅允许多个程序负载的时间仿真，并且允许 10 到 100 个以上核的共享内存应用的时间仿真。Sniper 模拟器和现有的模拟器相比，速度有很大的提升。Sniper 最主要的特征是它基于区间仿真、快速高效的核心模型。区间仿真在体系结构层面提高了抽象层次，使得更快的模拟器开发和评价成为可能；它通过跳过缺失事件，也称为区间来实现。将 Sniper 模拟器和多套接字的英特尔 Core2 以及 Nehalem 系统作了对比，发现 Sniper 性能预测平均误差在 25%，并且仿真速度在几个 MIPS 数量级[49]。

Sniper 模拟器，以及区间核心模型，有助于 uncore 和系统级的研究，因为这些研究比典型的 One-IPC 模型需要更多的细节，时钟精确级的模拟器虽然能提供足够多的细节，但是有效的负载大小执行时钟精确级的模拟器太慢使得实际应用受到限制[49]。另一个好处是，区间核心模型允许 CPI 栈，可以显示由于系统的不同特性带来的性能损失，如缓存系统或分支预测器，并且能够帮助我们更好地理解每个组件对整个系统的性能影响。这使得 Sniper 在应用程序特性和软硬件协同设计中发挥额外的作用。

(3) Aladdin

Aladdin 是 Harvard 大学体系结构-电路-编译器联合团队开发的面向特定函数加速器的模拟仿真器[50]。

(a) 配置运行 Aladdin 模拟仿真器

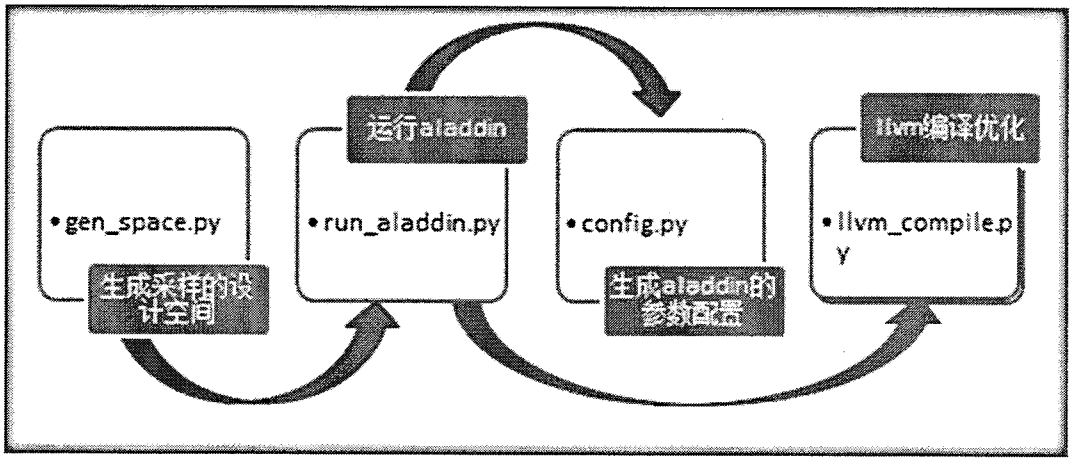


图 2-4 配置脚本源文件的调用关系

配置脚本源文件的调用关系如上图所示。gen_space.py 是用来产生采样到的设计空间子集；得到设计空间子集后，遍历设计空间子集中的每一种设计参数配置，调用 run_aladdin.py 来运行 Aladdin 模拟仿真器（python 脚本将调用 C++实现的 Aladdin 模拟仿真器源码）；在 run_aladdin.py 脚本中，调用 llvm_compile.py 和 config.py，进行编译优化以及生成 Aladdin 的配置参数。

(4) GPU 模拟器 GPGPU-Sim

GPGPU-Sim 是 Canada UBC 大学的 Admodt 教授所带领的课题组开发的世界上第一款时钟精度级别的 GPGPU 模拟仿真器[51]。GPGPU-Sim 是模拟 GPU 运行的软件，也称为模拟器，通过 GPU 模拟器来更好地观察 CUDA 程序在“模拟 GPU”中的运行情况。

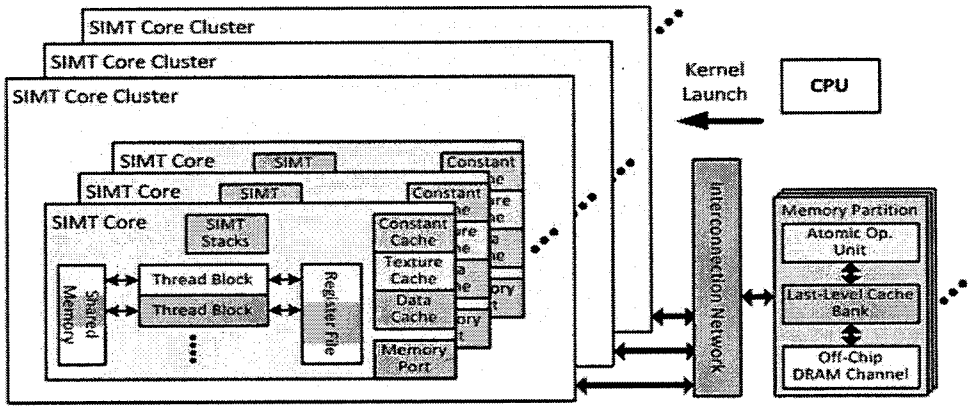


图 2- 5 GPGPU-Sim 总体框架[56]

GPGPU-Sim 的总体框架结构如上图 2-5 所示。SIMT 核心实质上是高并行流水的 SIMD 处理器，多个 SIMT 核心组成 GPGPU-Sim 的 GPU 模型，SIMT 核心之间通过片上网络和内存分区相连，实现通讯[55][56]。

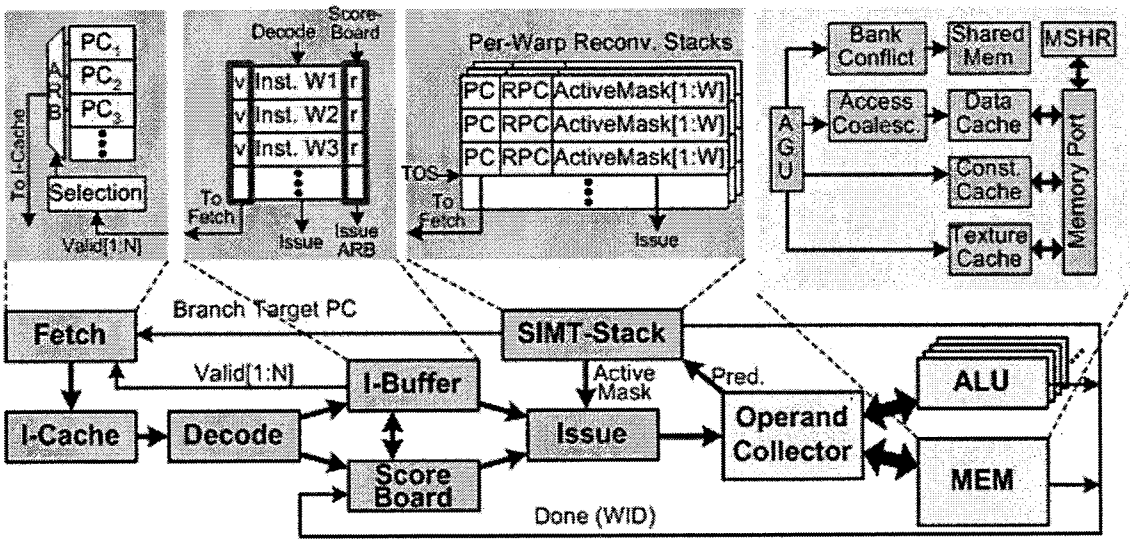


图 2- 6 SIMT 核心微体系结构[55]

图 2-6 显示了 SIMT 核心的微体系结构。这是最新版本的 GPGPU-Sim 新加入的很多特性，加入的新特性总结如下：

(a) 重新定义指令缓存的功能，同时将发射指令独立出来，与取指令和译码互相独立；

(b) 加入计分牌的功能，这样能够实现多发射机制；

(c) 建立能够使得逻辑指令和 IO 指令不在同一流水线进行的，支持多样的单指令多数据功能的模型。

2.4 本章小结

本章首先介绍了排序学习算法的相关概念，引出了排序学习算法的起源、发展和应

用；之后讲述了排序学习算法的分类，并分别对基于 Pointwise，基于 Pairwise 和基于 Listwise 的三类典型排序学习算法作以介绍，并分析了每种排序学习算法的优缺点；然后，引出基于 Pairwise 算法的排序学习算法 RankBoost，并对 RankBoost 算法的流程作了简单描述；然后，分析了陈天石、郭崎等人的工作 ArchRanker[36]，并解析了 ArchRanker 的工作原理和工作流程；介绍了常用的几种主流模拟仿真平台，包括 Gem5，Sniper，GPGPU-Sim 以及 Aladdin；最后，总结了本章的内容。

第3章 基于 RankBoostTree 算法的加速器设计空间探索

本章介绍基于 RankBoostTree 算法的加速器设计空间探索过程，描述 RankBoostTree 算法应用于加速器设计空间探索的工作原理和过程；分析 RankBoost 算法用决策桩作弱排序器的不足，引出决策树作弱排序器的 RankBoostTree 算法的优势并介绍 RankBoostTree 的实现原理以及 RankBoostTree 算法实现的整体框架和各个模块的设计、实现的功能以及核心功能点的实现细节。

3.1 基于排序学习算法 RankBoostTree 的加速器设计空间探索过程

基于排序学习算法 RankBoostTree 的加速器设计空间探索的流程如图 3-1 所示。基于 RankBoostTree 算法的加速器设计空间探索分为两个过程，第一，是获取仿真数据的过程；第二，是排序学习算法 RankBoostTree 基于仿真数据训练模型，预测结果的过程。下面分别介绍每个过程的工作原理和流程。

3.1.1 获取仿真数据

获取仿真数据是从加速器设计空间中基于一定的采样策略采样，并在模拟仿真平台上仿真得到采样点的体系结构目标行为。该过程分为 5 个步骤，第一步，确定影响加速器系统性能或功耗的设计参数；第二步，确定了设计参数后，分析每个设计参数的可取值，确定所研究的设计参数空间；第三步，基于一定的采样规则在设计参数空间中采样，获得采样点；第四步，按照采样点的设计参数对模拟仿真平台进行参数配置；第五步，在模拟仿真平台上进行模拟仿真，得到采样点的体系结构行为，如性能或功耗。

3.1.2 基于 RankBoostTree 算法进行加速器设计空间探索的过程

获取仿真数据后，将仿真数据按照一定的比例划分成训练集和测试集。RankBoostTree 算法遵循典型的机器学习算法的学习过程，算法分为训练和预测两个阶段。划分的训练集和测试集分别用于算法的训练阶段和测试阶段。下面具体描述每个阶段算法进行的设计空间探索的过程。

3.1.2.1 训练过程

对训练数据集的数据按照体系结构行为的偏好程度生成 RankBoostTree 算法所需要的数据形式，基于处理好的数据，训练弱排序器 Treei，(i = 1 ,2, 3, …, T)，T 表示迭代次数。判定模型是否达到准确率要求，若达到了准确率要求，则得到排序模型；若没有达到准确率要求，则继续迭代过程，训练弱排序器。

3.1.2.2 预测过程

当训练得到的模型达到了准确率要求时，多次迭代得到的弱排序器就构成了最终的排序模型。基于测试数据检验模型的预测准确率，若预测准确率达到要求，就能够用排序模型对设计空间中除采样点以外的其它体系结构设计参数配置作性能/功耗排序。

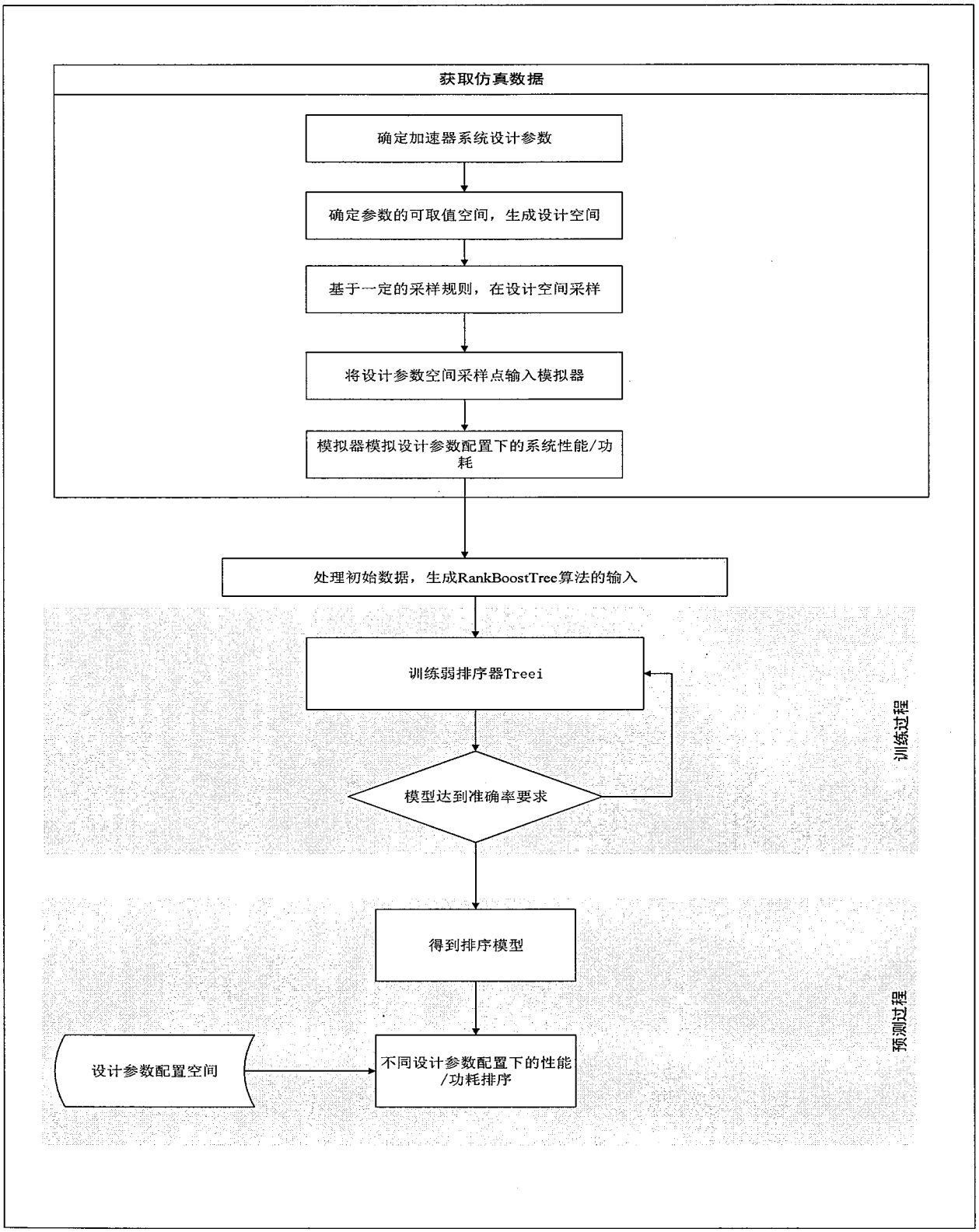


图 3- 1 基于 RankBoostTree 算法的加速器设计空间探索流程

3.2 基于决策桩的 RankBoost 算法的不足

RankBoost 算法的核心是弱排序器，构造一个什么样的弱排序器，如何构造弱排序器，都会影响 RankBoost 算法的性能。初始的 RankBoost 算法的弱排序器采用最简单的决策桩（Decision Dump）即如下公式来作排序[36]：

$$h(x) = \begin{cases} 1, & 若 f_i(x) > \theta \\ 0, & 若 f_i(x) \le \theta \end{cases} \tag{3.1}$$

决策桩属于单条件决策，决策过程如图 3-2 所示。基于向量 x 的第 i 维特征作决策，若第 i 维的特征值大于某个阈值 θ，则弱排序器 h 对 x 赋予偏好值 1；若第 i 维的特征值小于等于某个阈值 θ，则排序器 h 对 x 赋予偏好值 0。

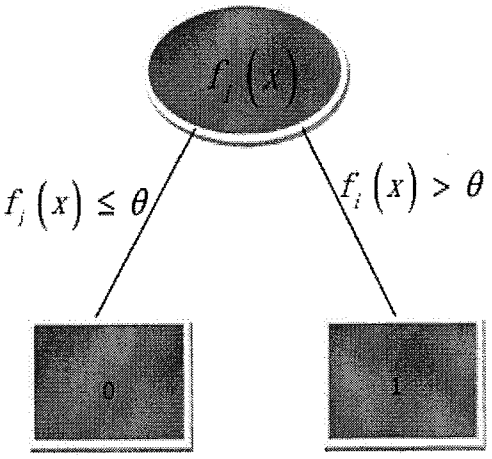


图 3-2 决策桩

由上述过程可以得到，决策桩的决策面是一个简单的平面，简单的平面作为决策面，在面对复杂的数据分布时，仅仅依靠有限的决策条件，效果不佳。因而，很自然地，想到赋予弱学习器更强大的决策能力。机器学习领域常用的分类/回归模型有决策树，神经网络，支持向量机等。这里，我们尝试与决策桩结构类似，但决策过程更为复杂的决策树来代替决策桩，作为算法的弱排序器。决策树的决策面可以将空间分成不规则的若干部分，对复杂的数据分布可以通过复杂的空间划分实现更精确的决策，决策树具有更强大的空间数据的划分能力。

另外，在使用决策桩作弱排序器时，每一轮迭代都需要遍历每一维特征，并且尝试所有体系结构参数配置（样本）这一维特征的取值，计算排序损失函数的值，取得所有可能性中使得排序损失函数达到最大的第 i 维特征以及该维特征所取得阈值。这是个非常耗时的过程，因而，在初始的 RankBoost 算法训练阶段，训练弱排序器十分慢。

3.3 RankBoostTree 排序学习算法的实现原理

RankBoostTree 算法改进了 RankBoost 算法的不足，每轮迭代训练一个决策树模型，用决

策树来作排序。下面介绍 RankBoostTree 算法的实现原理。

3.3.1 RankBoostTree 算法中的决策树

决策树模型可以看作是对实例进行分类的树状结构。构建决策树的过程实际上是基于某一种特征选择的准则，不断进行特征选择的过程。如图 3-3 所示，$f_i(x)$ 表示向量 x 的第 i 维特征，θ 表示第 i 维特征的分类阈值。一个样本 x 从根结点开始，沿着内部结点的决策条件对样本的特征值进行测试。根据测试结果，将样本归纳到相应的子结点。若子结点是叶子结点，则决策过程结束，该样本被分到某一个类别；若子结点是内部结点，则重复上述过程对样本的特征进行测试和决策。决策树算法是一种有监督的学习算法，每个训练样本都带一个类别标签，决策树模型通过不断学习，达到一定的分类正确率。

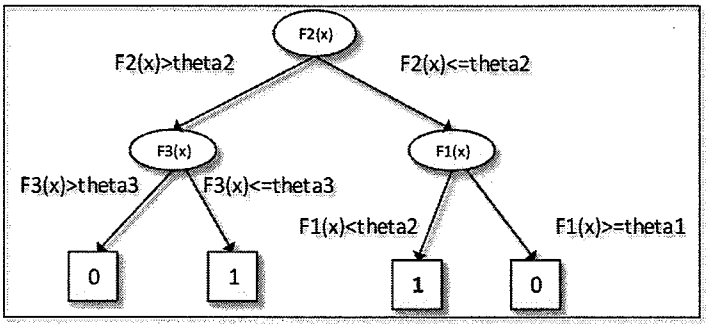


图 3-3 决策树的分类过程

对决策树有了基本的认识后，下面介绍排序学习算法 RankBoostTree 中的决策树。样本 X 由体系结构设计参数配置 x 和相应的体系结构目标行为组成。由于决策树属于机器学习中有监督的学习算法，所以，要为每一个样本 X 赋一个类别标签。RankBoostTree 算法是一种基于 Pairwise 的排序学习算法，输入的样本形式是体系结构参数配置偏序对，对于这样形式的样本对来说，单个样本实际的体系结构响应值不再有意义，有意义的是样本之间的偏序关系，所以，样本 X 的体系结构目标行为不能作为类别标签。由于决策树承担的角色是在每一轮迭代中，决策某一个样本 X 应当被赋值 1 还是赋值 0。这个是一个排序过程，赋值为 1 的样本比赋值为 0 的样本具有更高的偏好程度。当然，这也可以看成一个分类过程，将不同的样本分配到“1”类或者“0”类，分配到“1”类的样本比分配到“0”类的样本具有更高的偏好程度。

3.3.2 RankBoostTree 算法中样本的类别标签

由于决策树算法是一种有监督的机器学习算法，而本文所研究的加速器设计空间的数据并没有合适的类别标签。因此，下面先解决数据样本的类别标签问题。对于样本的类别标签，可以从以下两个方向来考虑，给样本赋类别标签：

(1) 对偏序对(x_i，x_j)构建类别标签（设 x_j 的偏好程度比 x_i 高）。这种方法建立类别标签比较容易，如 $X_1=[x_i-x_j]$的类别标签赋为 0，$X_2=[x_j-x_i]$的类别标签赋为 1。或者 $X_1=[x_i:x_j]$的类别标签赋为 0，$X_2=[x_j:x_i]$的类别标签赋为 1（其中，:表示样本特征之间的连接）。但

是，这种方法的弊端是，决策树的输入样本数呈平方级增加，导致生成决策树的时间大幅增加，降低算法的学习效率。

(2) 根据样本之间的偏好关系，对每个样本建立类别标签。这种方法不会增加决策树的输入样本数，但是建立标签比较复杂。下面，详细描述这种建立样本类别标签的方法。

首先，所有的训练集样本按照偏好程度的降序排序，如图 3-4 所示，x_1 表示偏好程度最高的样本，x_2 次之，……，x_n 表示偏好程度最低的样本。例如，在研究计算机系统结构性能，用 IPC（Instructions Per Cycle）来量化时，IPC 值越大，偏好程度越高（每个时钟周期执行的指令数越多，说明性能越好），则按照偏好程度的降序排序即为，按照 IPC 的降序排序；若研究计算机系统结构中的功耗问题，则功耗值越小，偏好程度越高，则按照偏好程度的降序排序即为，按照功耗的升序排序。所以，对不同的体系结构问题，应按照实际情况进行数据的排序处理。

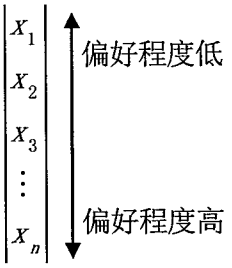


图 3-4 训练集样本按照偏好程度的降序排序

按照偏序程度排好序后，构建偏序对，如图 3-5 所示：

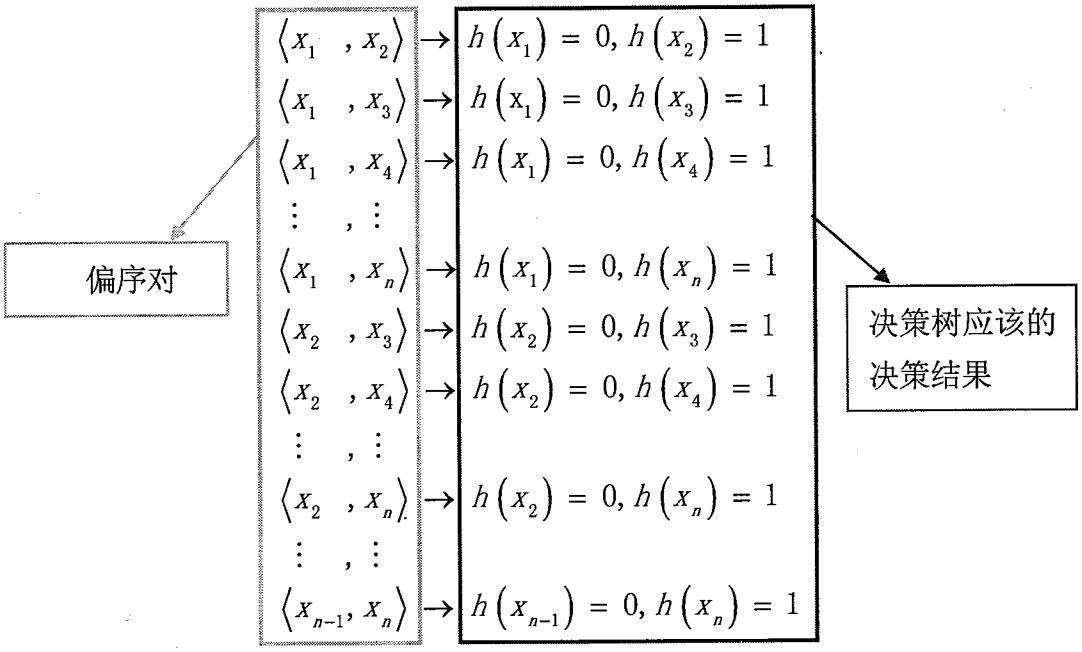


图 3-5 偏序对

从图 3-5 可以看出，除了 x_1 和 x_n 有确定的决策结果以外($h(x_1) \equiv 0$，$h(x_n) \equiv 1$)，其它样本的决策结果的域值为{ 0, 1 }。且观察图 3-5 中的偏序对，可知，出现在偏序对左边一

列的次数越多，其预测结果是 0 的概率就越大；出现在偏序对右边一列的次数越多，其预测结果是 1 的概率就越大。基于这样的观察结果，我们定义样本的类别标签公式如下：

$$\mathrm{Tag}\left(x\right) = \frac{\sum_{i} \mathrm{D}\left(x_i, x\right) \times \mathrm{Count}\left(x_i, x\right)}{\sum_{i} \mathrm{D}\left(x_i, x\right) \times \mathrm{Count}\left(x_i, x\right) + \sum_{j} \mathrm{D}\left(x, x_j\right) \times \mathrm{Count}\left(x, x_j\right)} \tag{3.2}$$

由上述公式计算每个样本的 Tag 值，设定一个阈值 θ，若样本的 Tag 值大于该阈值 θ，则样本的类别标签赋 1；如果样本的 Tag 值小于该阈值 θ，则样本的类别标签赋 0。如下公式所示：

$$\mathrm{TAG}\left(x\right) = \begin{cases} 1, & \text{if } \mathrm{Tag}\left(x\right) > \theta \\ 0, & \text{其它} \end{cases} \tag{3.3}$$

3.3.3 RankBoostTree 算法中决策树的生成原理

有了样本的类别标签公式，就有了决策树模型的输入。决策树模型基于一定的特征选择准则不断地进行特征选择，构建决策树的决策结点。特征选择，从训练样本的特征向量中选择具有分类能力的特征，具有分类能力的特征是指那些能够正确分类绝大部分样本的特征。如果基于一个特征的分类效果和随机分类的效果相当，那么，这样的特征作为分类条件就是没有意义的。一般情况下，特征选择的准则是信息增益或者信息增益比。本文采用的特征选择准则是信息增益比。

为了说明信息增益或者信息增益比，首先引入熵的概念[52]。下文信息增益与信息增益比的概念也都引自李航的《统计学习方法》。

熵是信息论中的术语，是对随机变量不确定性的量化。下面给出熵的数学定义[52]。

设 X 代表离散随机变量，X 的取值空间有限，且 X 的概率分布如下：

$$\mathrm{P}\left(X = x_i\right) = p_i, \quad i = 1,2,3,\mathrm{L},n \tag{3.4}$$

定义随机变量的熵[52]：

$$\mathrm{H}\left(X\right) = -\sum_{i=1}^{n} p_i \log p_i \tag{3.5}$$

特别地，当 $p_i = 0$ 时，$p_i \log p_i = 0$。

由熵的定义可得，熵仅和 X 的分布有关，而与 X 的取值没有关系。因而，X 的熵也可以写作[52]：

$$\mathrm{H}\left(p\right) = -\sum_{i=1}^{n} p_i \log p_i \tag{3.6}$$

熵的值越大，表示 X 的不确定性也越大。设有随机变量(X，Y)，联合概率分布定义如下

[52]：

$$P\left(X = X_i, Y = y_j\right) = p_{ij}, i = 1,2,3,\cdots,n \tag{3.7}$$

条件熵，指的是在条件随机变量下某一随机变量的不确定性。条件熵记作 $H(Y \mid X)$ ，意为在条件随机变量 X 下 Y 的不确定性，定义如下[52]：

$$H\left(Y \mid X\right) = \sum_{i=1}^{n} p_i H\left(Y \mid X = x_i\right) \tag{3.8}$$

其中，$p_i = P(X = x_i), i = 1, 2, 3, \cdots, n$。

由上述条件熵的定义可知，条件熵的实质是 X 条件下 Y 的条件概率分布的熵对 X 的数学期望。

下面引出信息增益的概念。信息增益一般都是某一特征 F 针对某一数据集 D 来说的，记为 $g(D, F)$。信息熵 $g(D, F)$定义为数据集 D 的熵 H(D)和特征 F 条件下数据集 D 的条件熵 $H(D \mid F)$的差，公式如下[52]：

$$g\left(D,F\right) = H\left(D\right) - H\left(D|F\right) \tag{3.9}$$

通常，我们定义互信息，互信息指的是熵和条件熵的差。由互信息的定义，可知，在决策树学习中，我们把训练数据集中的类和特征的互信息称为信息增益。

下面介绍信息增益在决策树中的作用。信息增益是用来选择特征的，在已知训练数据集 D 和特征 F 的条件下，数据集 D 的熵 H (D)的含义是，对数据集 D 进行分类的不确定性。条件熵 H(D | F)指的是在某一个特征 F 下，对数据集 D 进行分类的不确定性。因此，数据集 D 的熵 H (D)与条件熵 H(D | F)的差，定义为信息增益，这个信息增益表示的含义是，由这个条件特征 F 带来的数据集 D 的分类不确定性降低。从这个含义能够得到，对于某个数据集，它的信息增益取决于某一个特征，不同的特征会导致不同的信息增益。在决策树的学习过程中，应该选择那些能够导致大信息增益的特征，因为这样的特征能够更有效地分类。

有了上面选择特征的原则，我们不难得到选择特征的方法： 计算每个特征对某个训练数据集 D 的信息增益，并挑选结果最大的那个特征来进行分类。

我们记训练数据集为 D，用$|D|$来代表数据集 D 的样本容量，即数据集 D 中包含的样本总数。假设一共有 K 个类别，记为 C_k，其中，$k = 1, 2, 3, \cdots, K$, $|C_k|$表示分配到类别 C_k 的样本总数。假设特征 F 有 n 个不同的取值$\{f_1, f_2, f_3, \cdots, f_n\}$，那么基于特征 F 的取值把数据集 D 分成 n 个不同的子集 $D_1, D_2, D_3, \cdots, Dn$，同样的，$|D_i|$表示 D_i 的样本个数。设某个子集 D_i 里分配到类别 C_k 样本的集合记为 D_{ik}，即 $D_k = D_i \cap C_k$ ，同样，$|D_{ik}|$表示 D_{ik} 的样本个数，因此，我们可以得到下面的信息增益的计算流程[52]：

Input ：训练数据集 D 和特征 F:

Output ：特征 F 对训练数据集 D 的信息增益 g (D, F)

第一步：计算数据集 D 的熵 H(D)

$$H(D) = -\sum_{k=1}^{K} \frac{|C_k|}{|D|} \log_2 \frac{|C_k|}{|D|}$$

第二步：计算每一个特征 F 对数据集 D 的条件熵 $H(D \mid F)$

$$H(D \mid F) = \sum_{i=1}^{n} \frac{|D_i|}{|D|} H(D_i)$$

$$= -\sum_{i=1}^{n} \frac{|D_i|}{|D|} \sum_{k=1}^{K} \frac{|D_{ik}|}{|D_i|} \log_2 \frac{|D_{ik}|}{|D_i|}$$

第三步：计算每个特征的信息增益

$$g(D, F) = H(D) - H(D \mid F)$$

图 3-6 信息增益的计算流程[52]

上述信息增益的介绍，我们可以看出，计算得到的信息增益是某个特征对某个数据集来说的，是一个相对的概念，并没有绝对的含义。但是，如果考虑遇到难以分类的情况时，即某个数据集的熵本身就很大的情况下，信息增益的计算结果也会较大。这时，如果选择通过一种比值，来校正上面提到的问题，这就是本文采用的信息增益比。信息增益比也是用来挑选用以分类的特征。

下面给出信息增益比的定义。某一个特征 F 对数据集 D 的信息增益比，定义为它的信息增益 g (D, F)和数据集 D 的熵 H(D)之比，如下公式所示[52]：

$$g_R(D, F) = \frac{g(D, F)}{H(D)} \tag{3.10}$$

3.3.4 确定 RankBoostTree 算法中的阈值参数 θ

RankBoostTree 算法采用信息增益比作为决策树特征选择的准则。样本类别标签公式中的阈值参数 θ 的取值，有以下几种方案：

表 3-1 样本类别标签公式中阈值参数 θ 的取值方案

方案	参数 θ 的取值
1	所有样本 Tag 值处于第 25%的值
2	所有样本 Tag 值的中位数
3	所有样本 Tag 值处于第 75%的值
4	使得排序损失函数[定义在[43]]达到当前最小的 Tag 值

3.4 RankBoostTree 算法的实现

3.4.1 RankBoostTree 算法实现的整体框架

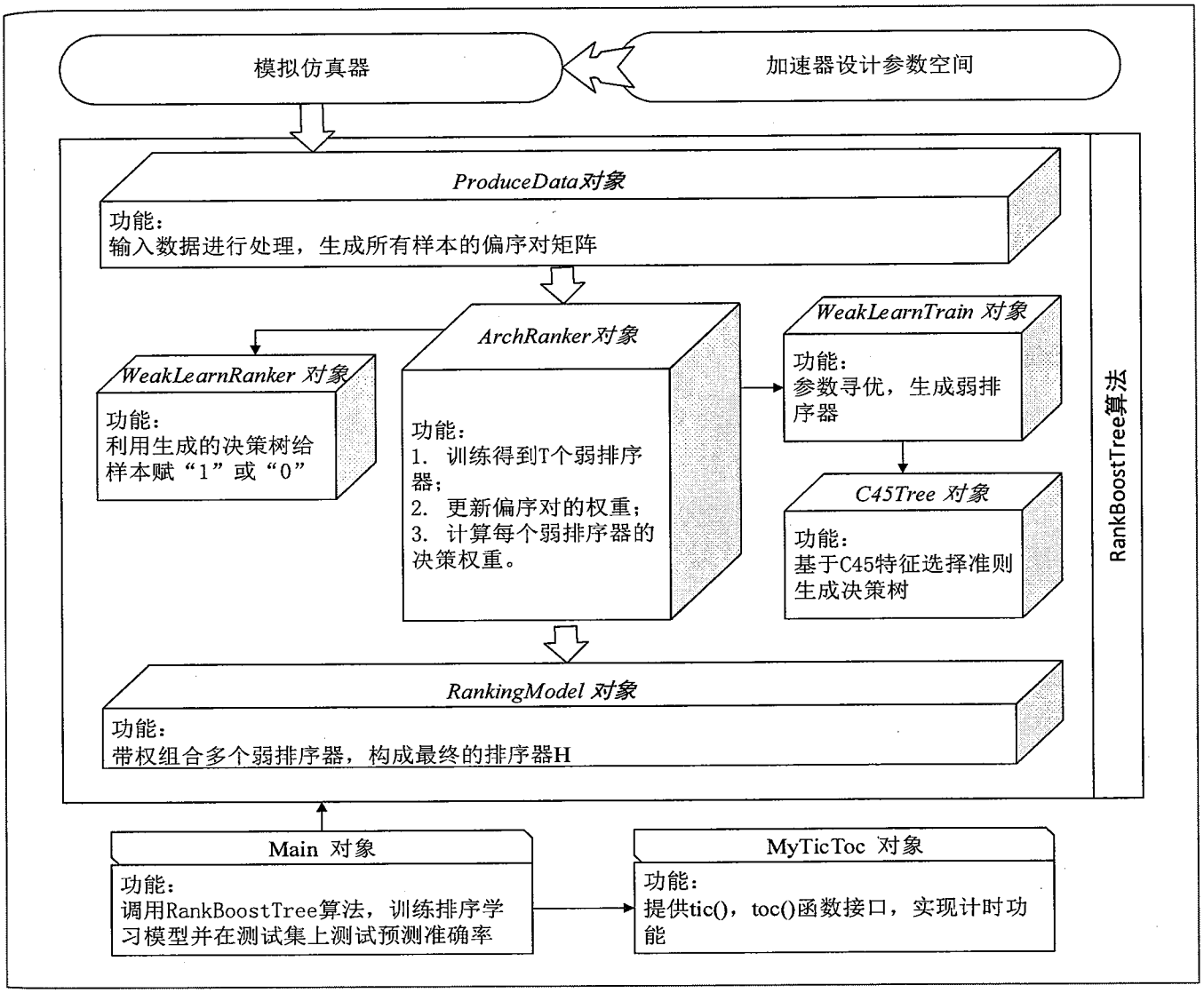


图 3-7 RankBoostTree 算法各模块的设计与功能实现

图例说明：

⇩ 表示数据传输关系

⟶ 表示类之间的函数调用关系

⇦ 表示部分数据传输关系

上图 3-7 是 RankBoostTree 算法实现的整体框架。由框架图可知，RankBoostTree 算法的核心功能主要由 ProduceData 类、ArchRanker 类、WeakLearnTrain 类、WeakLearnRanker 类、C45Tree 决策树类以及 RankingModel 类 6 个重要类实现。下面，详细介绍每个核心类的实现。

3.4.2 RankBoostTree 算法各模块的实现

(1) ProduceData 类

ProduceData 类主要承担的任务是将模拟器采样得到的数据进行处理，基于所研究的体系结构行为下的采样点的全序关系，生成所有采样点样本的偏序关系对，用偏序对矩阵来存储所有的偏序关系。其核心部分代码如下：

X,Y 规定了采样点样本的存储形式。prfrcMatrix (self) 函数用来生成偏序关系矩阵，偏序关系存储在矩阵 g 中。

ProduceData(X,Y)完成 ProduceData 类的主要工作。通过 Data(X,Y)函数处理初始数据，并调用 prfrcMatrix (self)生成所有的偏序关系对。

```
"""
Input
    X------- nSample x nFeature matrix, each row is one input.
    Y------- nSample x 1 vector of the corresponding class which each input belongs to.
"""
def prfrcMatrix(self):
    k=0;
    g=np.zeros((self.C,2),dtype=np.int);
    for i in range((self.nSample-1)):
        for j in range((i+1),(self.nSample)):
            g[k][0]=i+1;
            g[k][1]=j+1;
            k=k+1;
    return g;
def ProduceData(X,Y):
    data=Data(X,Y);
    data.g=data.prfrcMatrix();
    return data;
```

(2) ArchRanker 类

ArchRanker 类主要实现三个功能点。第一，通过 T 此迭代，训练 T 个弱排序器；第二，根据前一个弱排序器的预测结果，更新偏序对的权重分布，使得下一次迭代更关注排序错误的偏序对；第三，在每轮迭代过程中，计算每个弱排序器的决策权重。每个核心功能点的代码实现如下所示：

```
#每轮迭代训练一个弱排序器，xmlname[t]表示以 xml 文件的形式存储的第 t 个决策树
WkLearnTrain.wkLearnTrain(data,xmlname[t],D);

#每个弱排序器（决策树）的预测输出
o = WkLearnRanker.WkLearnRanker(data.obs,xmlname[t]);

#计算每个弱排序器（决策树）的决策权重
alpha[t] = 0.5*numpy.log((1+r[t])/(1-r[t]));

#更新偏序对的权重分布
D = D*numpy.exp( alpha[t]*( o[x2]-o[x1] ) )
```

(3) WeakLearnTrain 类

WeakLearnTrain 类实现训练弱排序器的功能，是 RankBoostTree 算法非常重要的一部分，完成模型参数寻优，计算采样点的类别标签，并调用决策树模型，训练弱排序器。其中，有一个非常重要的参数 pi(x)，这个参数是在参数寻优的过程中要用到的，它只和采样点 x 的权重分布有关，可以单独预先计算，它的推导以及表达式见[43]。计算每个采样点 x 的 pi (x)值，计算采样点的类别标签值以及参数寻优的核心代码如下：

```
"""*************计算 potential***************"""
    pi = numpy.zeros((data.nSample,1))
    for k in range(data.C):
        i = data.g[k][0];
        j = data.g[k][1];
        pi[i-1] = pi[i-1] - D[:,k]
        pi[j-1] = pi[j-1] + D[:,k]

"""*************计算数据对象的标签值***************"""
    count = numpy.zeros((len(data.obs)-1,2))
    for k in range(data.C):
        i = data.g[k][0]-1
        j = data.g[k][1]-1
        count[i][0] = count[i][0]+D[:,k]
        count[j][1] = count[j][1]+D[:,k]
    tag = [None]*(len(data.obs)-1)
    sortTag = [None]*(len(data.obs)-1)
    for k in range(len(data.obs)-1):
        tag[k] = (count[k][0]) / (count[k][0]+count[k][1])
```

```
"""***********参数寻优，寻找使得排序损失函数当前最小的参数**************"""
for i in range(1,len(sortTag)):
        for j in range(len(tags)):
            if tags[j] > sortTag[i] and tags[j] <= sortTag[i-1]:
                searchR = searchR + pi[index[j]]
        if math.fabs(searchR) > math.fabs(rStar):
            rStar = searchR
            theta = sortTag[i]

    for k in range(len(tag)):
        if tag[k] > theta:
            tag[k] = 1
        else:
            tag[k] = 0
```

(4) C45Tree 类

C45Tree 类实现了决策树算法。特征选择基于信息增益比实现，生成的决策树模型以 xml 标签的形式存储在 xml 类型的本地文件中。下面，主要列出决策树算法的几个重要函数如下所示：

表 3- 2 决策树算法的核心函数及其实现的功能描述

函数名	函数功能描述
entropy (x)	计算熵
gain(category,attr)	计算信息增益
gain_ratio (category, attr)	计算信息增益比
division_point(category,attr)	基于信息增益比的特征选择准则选择特征
grow_tree(data,category,parent,attrs_names)	递归地进行特征选择的过程，生成决策树
predict(xmldir,testing_obs)	用生成的决策树对测试数据作预测

(5) WeakLearnRanker 类

WeakLearnRanker 类的重要功能在于应用生成的决策树模型对训练数据的偏序关系进行预测，预测结果用以更新偏序对的权重分布，指导下一轮的训练过程。WeakLearnRanker 的功能主要通过调用决策树算法的决策树预测函数 predict (xmldir, testing_obs)来实现。

(6) RankingModel 类

RankingModel 类的主要工作是带权组合多个弱排序器，得到最终的排序器 H，每个弱排序器的权重即为 ArchRanker 类中计算得到的 alpha 因子。RankingModel 的核心代码如下：

```
def RankingModel(X,T,xmlname,alpha):
    nSample, nFeature = numpy.shape(X)
    nSample = nSample-1
    H = numpy.zeros((1,nSample))
    o = numpy.zeros((1,nSample))
    for t in range(T):
        o = WkLearnRanker.WkLearnRanker(X,xmlname[t])
        H = H + alpha[t]*o
    return H
```

3.5 本章小结

本章介绍了基于 RankBoostTree 算法的加速器设计空间探索过程。第一节从获取仿真数据和基于RankBoostTree算法进行加速器设计空间探索两个阶段描述了RankBoostTree算法应用于加速器设计空间探索的工作原理和过程。第二节分析了 RankBoost 算法用决策桩作弱排序器的不足，引出用决策树作弱排序器的 RankBoostTree 算法。第三节介绍了 RankBoostTree 的实现原理；第四节介绍了 RankBoostTree 算法实现的整体框架并详细描述了 RankBoostTree 算法各个模块的设计、实现的功能以及核心功能点的实现细节。

第4章 实验和评估

本章主要描述学习排序算法RankBoostTree在函数加速器设计空间探索以及GPU设计空间探索的应用，并对比了RankBoostTree算法和RankBoost算法的预测准确率和迭代效率，验证算法改进的有效性。

4.1 设计空间

4.1.1 函数加速器设计空间

可定制的硬件加速形态是未来系统的重要组成部分，Aladdin是一种面向特定函数的加速器模拟器，具有模块化设计、可配置的特点。Aladdin模拟器提供如表4-1所示的可配置参数partition, unroll, pipe以及clock_period，这些参数对高层次综合优化有着重要的作用，对性能/功耗有较大的影响，因此，本章基于这些设计参数进行函数加速器的设计空间探索。

Aladdin加速器提供的设计参数生成的设计空间大小如下：

$$\begin{aligned} designSize &= number_{partition} * number_{unroll} * number_{pipe} * number_{clock\_period} \\ &= 7 * 7 * 2 * 6 \\ &= 588 \end{aligned}$$

表4-1 Aladdin加速器设计空间[50]

参数	可取值	含义	取值个数
partition	1, 2, 4, 8, 16, 32, 64	分割因子	7
unroll	1, 2, 4, 8, 16, 32, 64	循环展开因子	7
pipe	0,1	使能循环流水	2
clock_period	1,2,3,4,5,6	时钟周期	6

表4-2 SHOC的8个benchmark描述

Benchmark	描述
bb_gemm	广义矩阵乘法，测量GEMMBLAS性能，单精度
fft	快速傅里叶转换，测量单精度和双精度快速傅里叶转换的性能
md	分子动力学，测量进行分子动力学中的兰纳-琼斯势性能
pp_scan	并行前缀求和，测量对大规模浮点矩阵执行并行前缀求和的性能
reduction	归约，用于测量大规模浮点加运算规约的性能
ss_sort	排序，测量无符号数矩阵基数排序性能
stencil	测量对一个二维的9点单精度stencil执行计算的性能（包括PCIe传输）
triad	把copy、Scale、Add三种操作组合起来进行测试，采用单精度执行计算

由于 Aladdin 模拟器本身能够提供的可配置参数生成的设计空间太小，并且 Aladdin 模拟器的可配置参数并没有包含 benchmark 的信息。所以，我们尝试增加 benchmark 的信息，将 benchmark 的参数引入 Aladdin 函数加速器模拟器中，这样，不仅增大了设计空间，并且由于引入 benchmark 信息使得函数加速器设计空间探索更具实际意义。

本文在 Aladdin 上执行了 8 个 benchmark，这 8 个 benchmark 来自 SHOC（ScalableHeterOgeneousComputing）性能测试包[61]，包括 bb_gemm，fft，md，pp_scan，reduction，ss_sort，stencil，triad。表 4-2 是这 8 个 benchmark 的简单介绍。

定义这 8 个 benchmark 的可配置参数，并将这 8 个 benchmark 的可配置参数加入 Aladdin 设计空间，这 8 个 benchmark 的可配置参数及其可取值如表 4-3 所示：

表 4-3 Benchmark 可配置参数设置

Benchmark	array_name	array_partition_type	array_size	array_wordsize
bb_gemm	x,y,z	{cyclic, complete}：3	arraySize_range:3	4:3
fft	'work_x', 'work_y', 'DATA_x','DATA_y', 'data_x','data_y', 'smem','reversed', 'sin_64','cos_64', 'sin_512','cos_512'	{cyclic, complete}:12	{512, 512, 512, 512, 8, 8, 576, 8, 488, 488, 488, 488}	4：12
md	'd_force_x', 'd_force_y', 'd_force_z', 'position_x', 'position_y', 'position_z','NL'	{cyclic, complete}：6	arraySize_range_md:6	4：6
pp_scan	'bucket','bucket2', 'sum'	{cyclic, complete}：3	arraySize_range:3	4：3
reduction	in	{cyclic, complete}：1	arraySize_range:1	4：1
ss_sort	'a','b','bucket','sum'	{cyclic, complete}：4	arraySize_range:4	4：4
stencil	'orig','sol','filter'	{cyclic, complete}：3	arraySize_range:3	4：3
triad	'a','b','c'	{cyclic, complete}：3	arraySize_range:3	4：3

注：其中，x：n 表示 n 个 x。

其中，$arraySize\_range = \{8, 16, 32, 64, 128, 512, 1024, 2048\}$

$arraySize\_range\_md = \{32, 1024\}$

所以，每个 benchmark 在 Aladdin 上执行的设计空间大小总结如下：

表 4- 4　SHOC 中 benchmark 在 aladdin 上的设计空间

Benchmark	设计空间大小
bb_gemm	$588*2^3*8^3$=2,408,448
fft	$588*2^3*8^3$=2,408,448
md	$588*2^7*8^7$=9,633,792
pp_scan	$588*2^3*8^3$=3,311,616
reduction	588*2*8=9,408
ss_sort	$588*2^4*8^4$=38,535,168
stencil	$588*2^3*8^3$=3,311,616
triad	$588*2^3*8^3$=3,311,616

4.1.2 GPU 设计空间

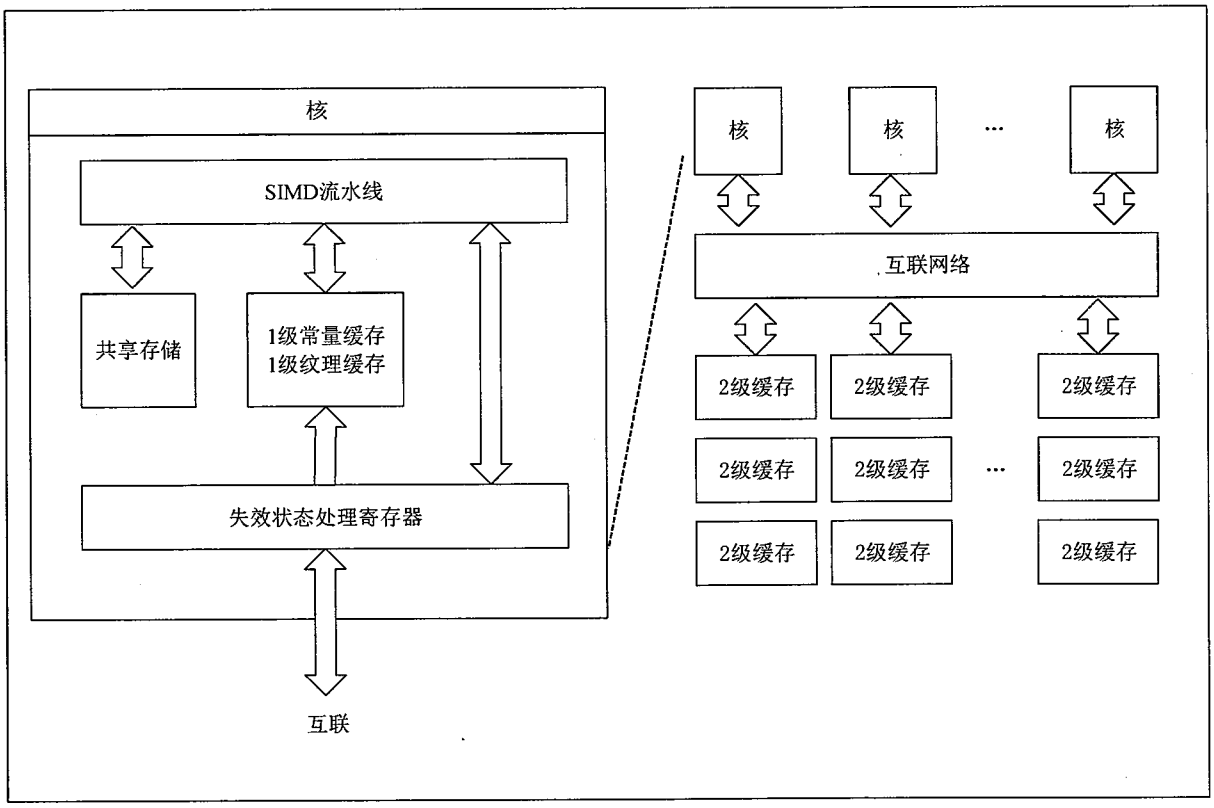


图 4- 1 NVIDIA GPU 结构图[62]

如图 4-1 所示是基于 NVIDIA GPU 的结构图。NVIDIA GPU 的核心部件是连接在同一互联网络并行工作的核，核可以看作一个 SIMD（Single Instruction Multiple Data，单指令多数据）的硬件部件，核内执行相同指令的线程组称为 Warp，即线程束，线程束的多个线程并行工作。由于核内资源有限，多个线程要共享寄存器组、缓存（包括指令

缓存、常量缓存、数据缓存以及纹理缓存）、纹理单元等存储资源，还要竞争有限的线程槽，线程块槽等。这些资源量的设置不同，核心并发的线程数不同，都会影响 GPU 的性能和功耗。因此，针对 NVIDIA GPU，选择表 4-5 所示的设计参数[62]生成设计空间。表 4-6 是在实验中不变的设计参数。

表 4- 5　GPU 设计参数[62]

设计参数	单位	可取值	参数含义
blk	块/核	1，2，4，8，16	块并发执行数
ccache	KB	1，2，4，8，16，32	常量缓存的大小
tcache	KB	1，2，4，8，16，32	纹理缓存的大小
smp	个	1，2，4，8	共享内存端口的个数
intra	个	1，2，4，8	Warp 内部之间的合并访问
ccp	个	1，2，4，8	只读常量高速缓存端口的个数
simd	个	8，16，32	SIMD 带宽
mshr	个/线程	1，2，3	失效状态处理寄存器的个数
dramq	项	16，32，64	DRAM 调度器队列的大小
inter	个	2，4，6	Warp 之间的合并访问

表 4- 6 不变参数[62]

参数	取值
核心数	30
处理器核的时钟周期	325 MHz
L1 数据缓存	无
L2 数据缓存	无
互联拓扑	蝶形
互联的 flit 大小	32 byters
DRAM 控制器的数量	8
每个控制器 DRAM 的芯片数	2
DRAM 的时钟周期	800 MHz
DRAM 的类型	GDDR3

从表 4-5 GPU 的设计参数可知，GPU 的设计空间大小为：

$$\begin{aligned} designSize &= number_{blk} * number_{ccache} * number_{tcache} * number_{smp} * number_{intra} \\ &\quad * number_{ccp} * number_{simd} * number_{mshr} * number_{dramq} * number_{inter} \\ &= 5 * 6 * 6 * 4 * 4 * 4 * 3 * 3 * 3 * 3 \\ &= 933,120 \end{aligned}$$

本文实验模拟 GPU，采用 NVIDIA Quadro FX 5800 GPU 的配置文件配置 GPU 模拟器。表 4-6 的不变参数也取自 NVIDIA Quadro FX 5800 GPU。

实验执行的 benchmark 取自 ispass2009-benchmark 以及 CUDA SDK。表 4-7 是实验所用的 benchmark：

表 4-7　GPU 设计空间探索 benchmark 描述[62][63]

benchmark	问题规模	线程并行数/block	指令数	描述
BFS	65536 nodes	512	17M	基于图的广度优先搜索
CP	256×256 grid	128	126M	计算分子动力学库仑势
Matrix	256×256 matrices	64	78M	矩阵乘法
RAY	256×256 image	128	71M	照明效果图形渲染

4.2 设计空间探索

4.2.1 训练集和测试集的划分和规模

(1) 函数加速器设计空间探索数据集

表 4-8　函数加速器设计空间探索训练集和测试集的划分和规模

Benchmark	训练集大小	测试集大小	训练集百分比%
bb_gemm	200	800	20
fft	200	800	20
md	200	800	20
pp_scan	200	800	20
reduction	200	800	20
ss_sort	200	800	20
stencil	200	800	20
triad	200	800	20

(2) gpu 设计空间探索数据集

表 4-9　gpu 设计空间探索训练集和测试集的划分和规模

Benchmark	训练集大小	测试集大小	训练集百分比%
matrix	100	160	38.5
CP	100	160	38.5
BFS	100	160	38.5
RAY	100	160	38.5

4.2.2 执行时间分布与功耗分布分析

(1) 执行时间分布

以函数加速器为例分析。下图所示分别是 bb_gemm、fft、md、pp_scan、reduction、

ss_sort、stencil 以及 triad 8 个 benchmark 的 1000 个采样点按照执行时间降序排序的执行时间分布图。

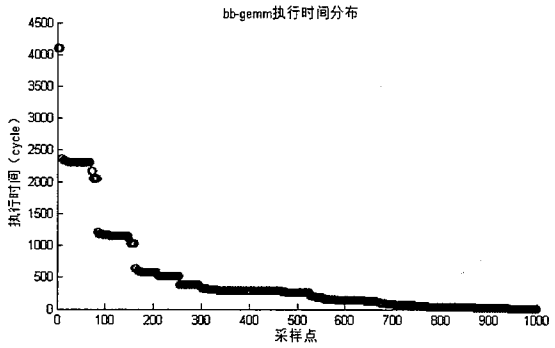


图 4- 2 bb_gemm 执行时间分布

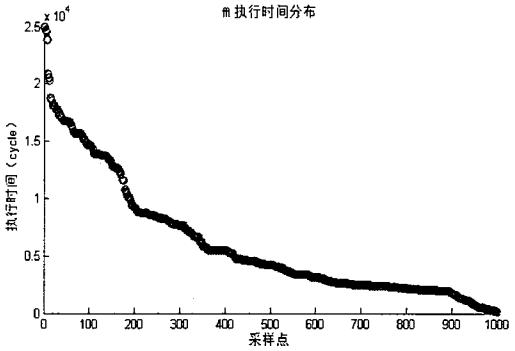


图 4- 3 fft 执行时间分布

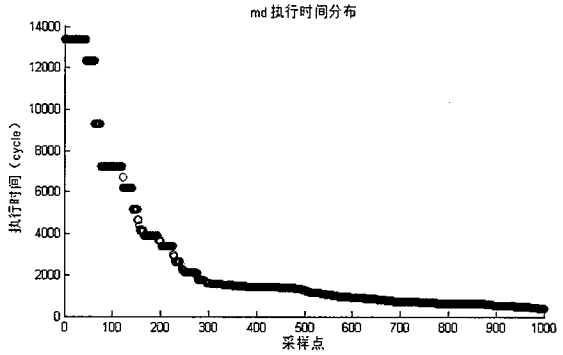


图 4- 4 md 执行时间分布

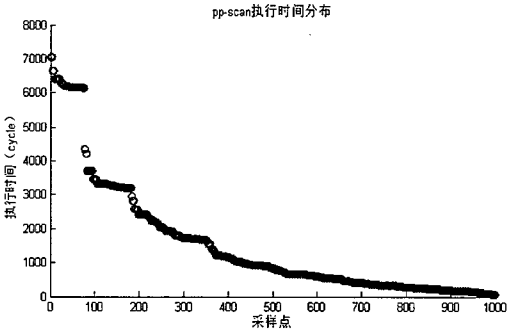


图 4- 5 pp_scan 执行时间分布

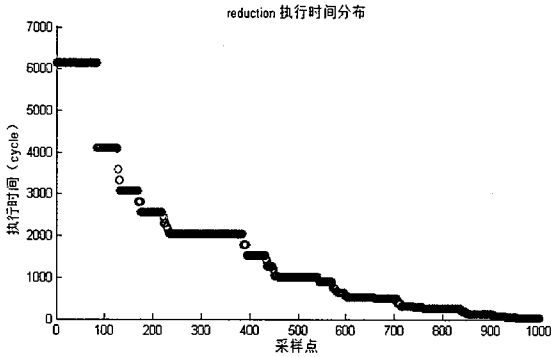


图 4- 6 reduction 执行时间分布

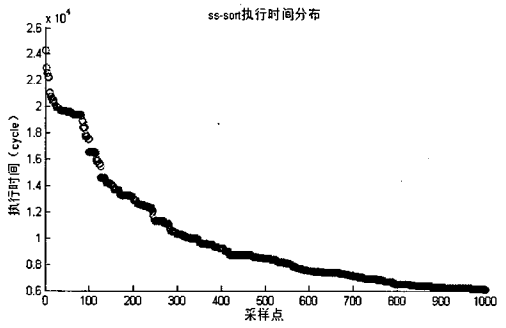


图 4- 7 ss_sort 执行时间分布

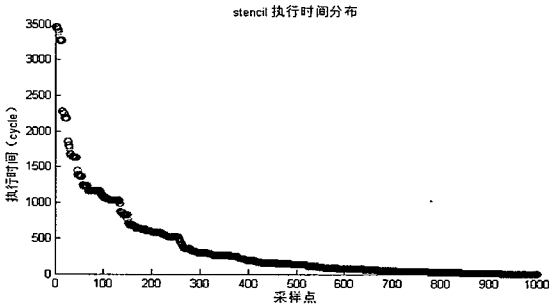


图 4- 8 stencil 执行时间分布

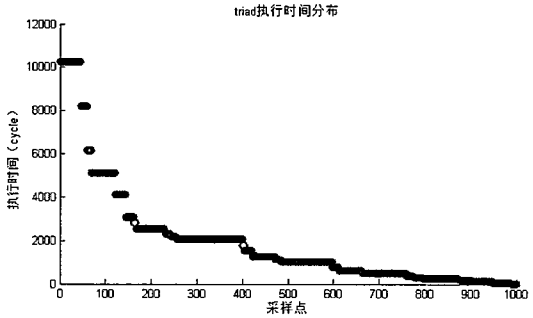


图 4- 9 triad 执行时间分布

由上图 4-2 到 4-9 的执行时间分布图可以看出，执行时间的分布有聚集的特征。如 bb_gemm，md，pp_scan，reduction 和 triad 表现得尤为突出。

(2) 功耗分布

依然以函数加速器为例分析。下图所示，分别是 bb_gemm、fft、md、pp_scan、reduction、ss_sort、stencil 以及 triad 8 个 benchmark 的 1000 个采样点按照功耗降序排序的功耗分布图。

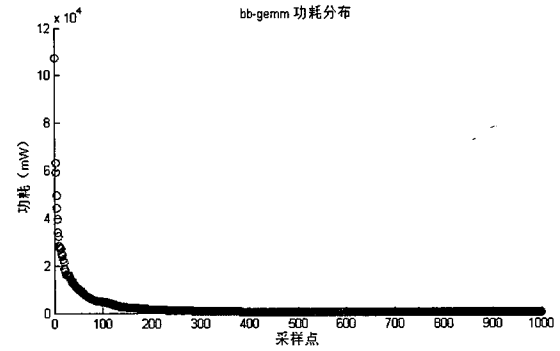


图 4- 10　bb_gemm 功耗分布

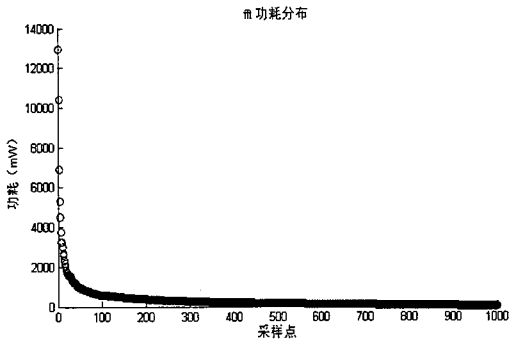


图 4- 11 fft 功耗分布

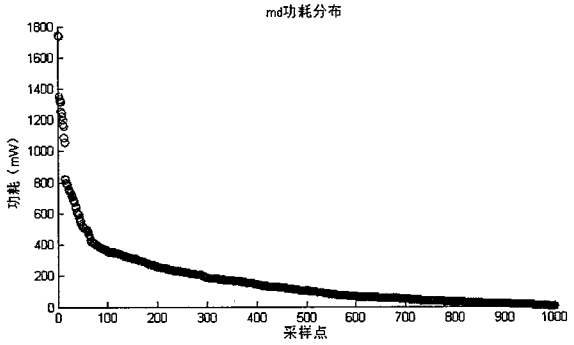


图 4- 12　md 功耗分布

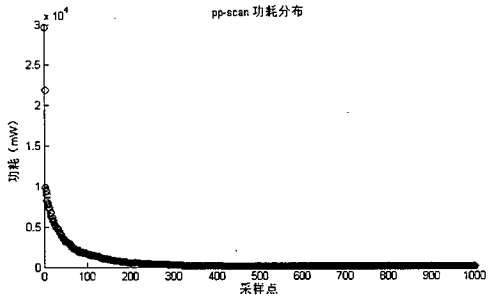


图 4- 13 pp_scan 功耗分布

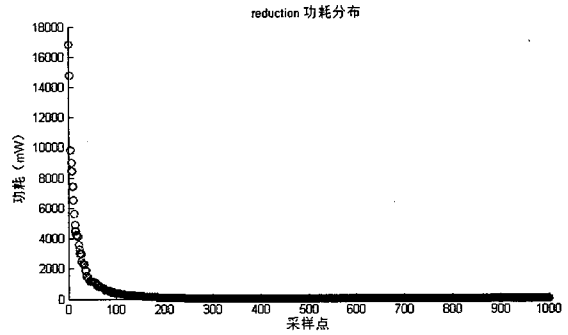


图 4- 14 reduction 功耗分布

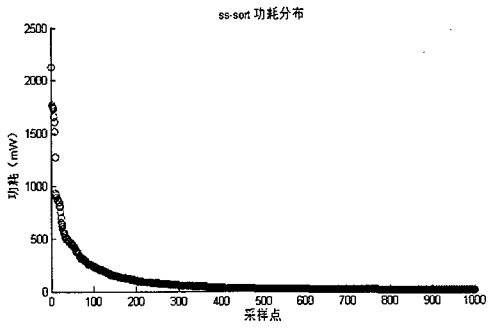


图 4- 15 ss_sort 功耗分布

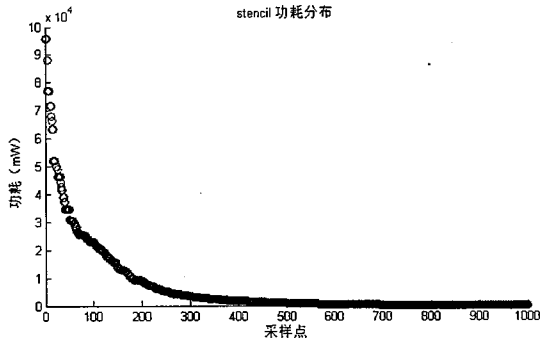


图 4- 16 stencil 功耗分布

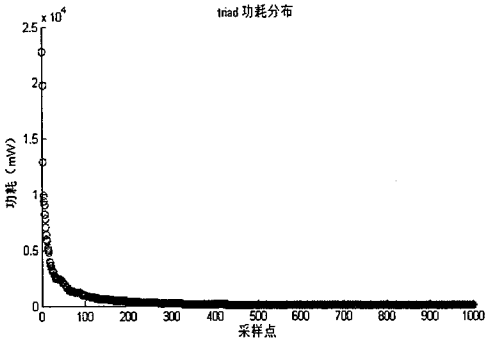


图 4- 17 triad 功耗分布

由图 4-10 到图 4-17 的功耗分析图可知，功耗分布没有聚集现象，功耗分布基本呈连续变化的过程。

(3) 执行时间预测和功耗预测的特点

已知上述执行时间和功耗的分布特征，分析函数加速器的设计参数，发现执行时间对设计参数的变化的敏感程度远不及功耗，并且不同设计参数对执行时间和功耗的影响程度也不同。以函数加速器的执行时间为例，pipe，clock_period，array_size 以及 array_partition_type 这几个参数的变化对 bb_gemm 的执行时间影响不大，而执行时间对 unroll 和 partition 参数的变化表现得比较敏感。这些参数的变化对功耗的影响比较大，基本不存在某个设计参数发生变化，而功耗没有变化的情况。

计算执行时间以及功耗的标准差，如表 4-10 以及 4-11 所示，功耗的标准差大于时间的标准差，说明设计参数对功耗的影响比对执行时间的影响大。

表 4-10 功耗标准差

Benchmark	标准差
bb_gemm	$6.35*10^3$
fft	$7.09*10^2$
md	$2.05*10^2$
pp_scan	$1.74*10^3$
reduction	$1.26*10^3$
ss_sort	$2.14*10^2$
stencil	$1.34*10^4$
triad	$1.41*10^3$
平均值	$3.16*10^3$

表 4-11 执行时间标准差

Benchmark	标准差
bb_gemm	$6.80*10^2$
fft	$5.12*10^3$
md	$3.31*10^3$
pp_scan	$1.70*10^3$
reduction	$1.73*10^3$
ss_sort	$4.13*10^3$
stencil	$5.65*10^2$
triad	$2.38*10^3$
平均值	$2.45*10^3$

4.2.3 RanBoostTree 实验结果

(1) 函数加速器设计空间探索实验

下图 4-18 是 RankBoostTree 预测函数加速器设计空间内的不同设计参数采样点下的性能以及功耗排序的准确率，实验分别统计了算法迭代 50 次、100 次和 200 次的结果。

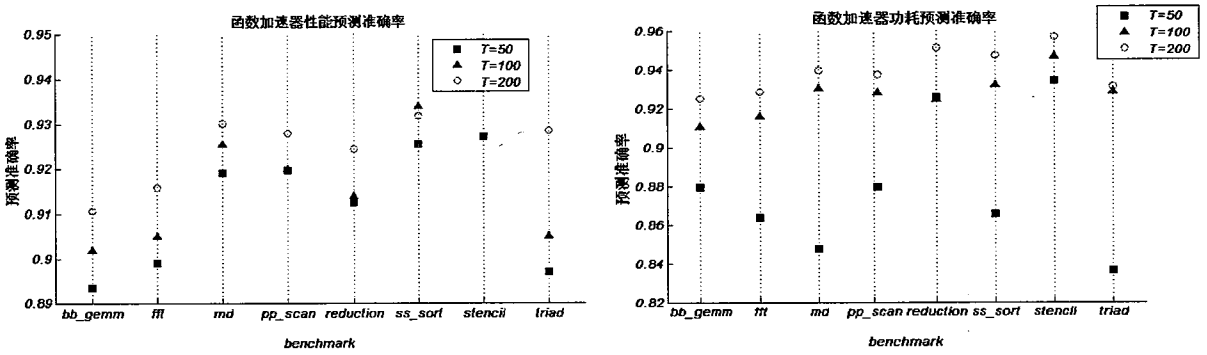
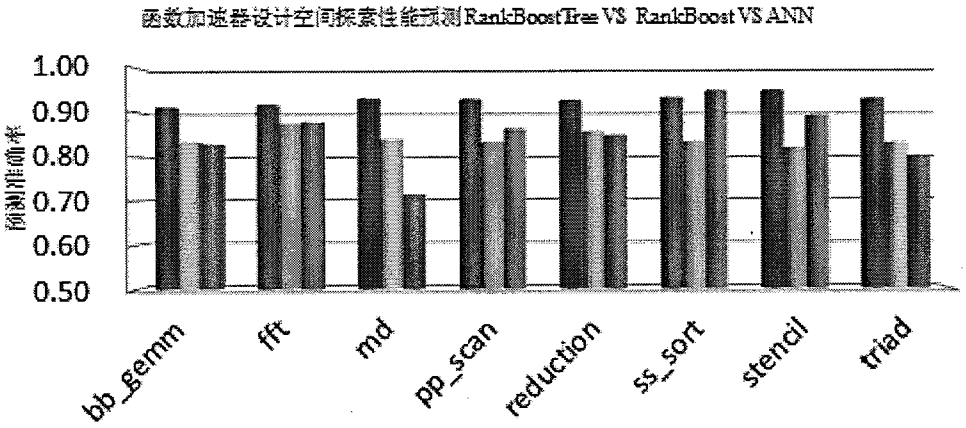


图 4-18　函数加速器设计空间性能排序关系预测准确率（左）功耗排序关系预测准确率（右）

由图实验结果可知，基于 RankBoostTree 的排序学习算法在函数加速器设计空间探索体系结构行为预测性能表现良好，并且随着迭代次数的增加，预测准确率稳步上升；同时，我们观察发现，随着迭代次数的增加，预测准确率的增速减缓。



	bb_gemm	fft	md	pp_scan	reduction	ss_sort	stencil	triad
RankBoostTree	0.91	0.92	0.93	0.93	0.92	0.93	0.95	0.93
RankBoost	0.83	0.87	0.84	0.83	0.85	0.83	0.82	0.83
ANN	0.83	0.88	0.71	0.86	0.85	0.95	0.89	0.80

图 4-19 函数加速器设计空间探索性能预测 RankBoostTree 对比 ANN

(a) RankBoostTree 和 RankBoost、ANN 性能对比：

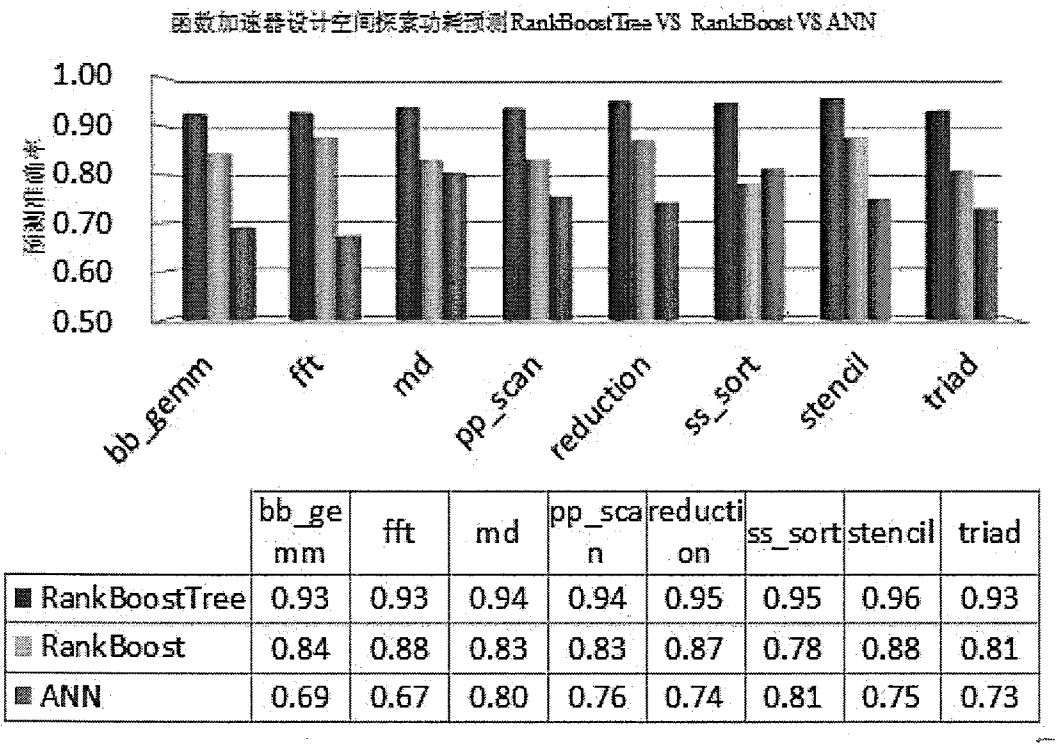
图 4-19，图 4-20，比较了 RankBoostTree 和 RankBoost 以及 ANN 在函数加速器设计空间探索性能和功耗的排序预测准确率比较。RankBoostTree 算法、RankBoost 算法以及 ANN 算法的实验条件如下：

- RankBoostTree 算法：迭代次数 T=200，训练样本 S1=200，测试样本 S2=800。

- RankBoost 算法：迭代次数 T=200，训练样本 S1=200，测试样本 S2=800。
- ANN 算法：迭代次数 T=200，训练样本 S1=200，测试样本 S2=800；隐藏层为 10，训练过程采用 Levenberg-Marquardt。

表 4- 12 性能预测 RankBoostTree 对比 ANN 性能提高百分比

Benchmark	相比 RankBoost 性能提高百分比（%）	相比 ANN 性能提高百分比（%）
bb_gemm	9.36	10.13
fft	4.81	4.51
md	10.98	30.47
pp_scan	11.81	7.80
reduction	8.07	9.27
ss_sort	12.02	- 1.59
stencil	15.91	6.36
triad	12.08	16.08



	bb_gemm	fft	md	pp_scan	reduction	ss_sort	stencil	triad
RankBoostTree	0.93	0.93	0.94	0.94	0.95	0.95	0.96	0.93
RankBoost	0.84	0.88	0.83	0.83	0.87	0.78	0.88	0.81
ANN	0.69	0.67	0.80	0.76	0.74	0.81	0.75	0.73

图 4- 20 函数加速器设计空间探索功耗预测 RankBoostTree 对比 ANN

表 4-12，4-13 统计了 RankBoostTree 预测性能和功耗相对 RankBoost 算法以及 ANN 算法准确率提高的百分比。由实验数据可得，RankBoostTree 相比 RankBosot 和 ANN，预测准确率大幅提高。

表 4- 13　功耗预测 RankBoostTree 对比 RankBoost 以及 ANN 性能提高百分比

Benchmark	相比 RankBoost 性能提高百分比（%）	相比 ANN 性能提高百分比（%）
bb_gemm	9.60	33.92
fft	5.82	37.80
md	13.00	16.75
pp_scan	12.62	24.13
reduction	9.18	27.99
ss_sort	21.10	16.73
stencil	9.37	27.48
triad	15.22	27.64

(2) GPU 设计空间探索实验结果

图 4-21 是 RankBoostTree 预测 GPU 设计空间内的不同设计参数配置下的执行时间预测准确率，实验分别统计了算法迭代 50 次、100 次、200 次的结果。

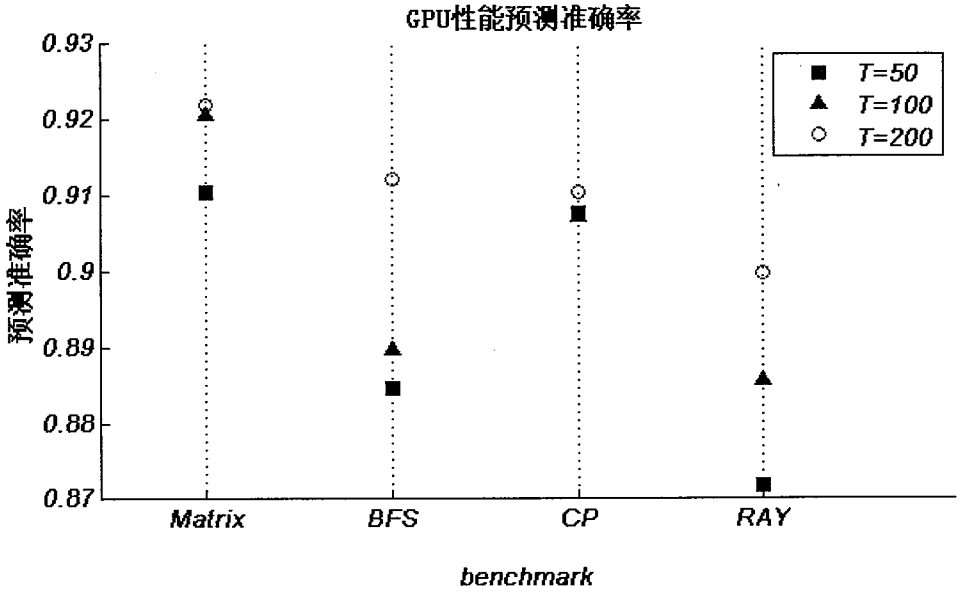


图 4- 21　GPU 设计空间执行时间排序关系预测准确率

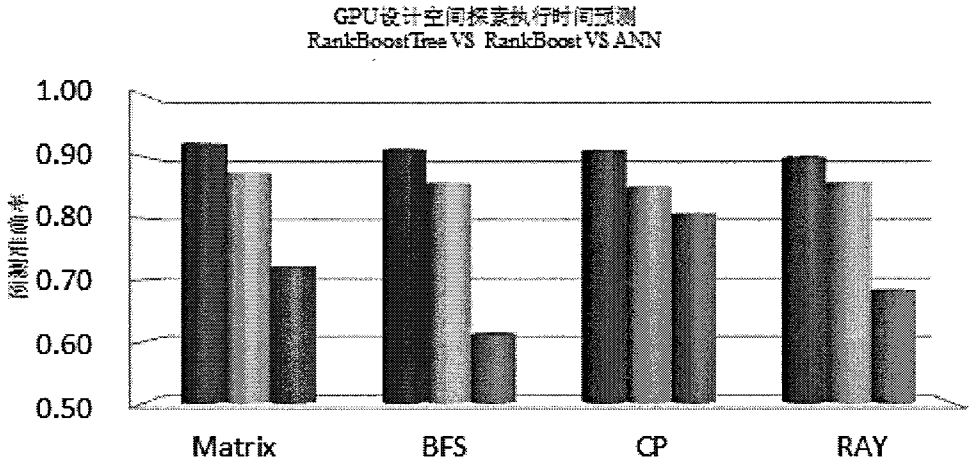
由实验结果可得，RankBoostTree 算法预测体系结构行为排序性能良好，并且随着迭代次数增加，准确率逐步提高。

(a) RankBoostTree 和 RankBoost、ANN 性能对比：

我们对比了相同实验条件下的 RankBoostTree 算法和 RankBoost 算法、ANN 算法的预测准确率。结果如下图 4-22 所示。RankBoostTree 算法、RankBoost 算法和 ANN 算法

的实验条件如下：

- RankBoostTree 算法：迭代次数 T=200，训练样本 S1=100，测试样本 S2=160。
- RankBoost 算法：迭代次数 T=200，训练样本 S1=100，测试样本 S2=160。
- ANN 算法：迭代次数 T=200，训练样本 S1=100，测试样本 S2=160；隐藏层为 10，训练过程采用 Levenberg-Marquardt。



	Matrix	BFS	CP	RAY
RankBoostTree	0.92	0.91	0.91	0.90
RankBoost	0.87	0.86	0.85	0.86
ANN	0.72	0.61	0.81	0.68

图 4- 22　GPU 设计空间探索功耗预测 RankBoostTree 对比 ANN

下表 4-14 是对比 RankBoostTree 算法和 RankBoost、ANN 预测准确率性能提高百分比。

表 4- 14　执行时间预测 RankBoostTree 对比 RankBoost 以及 ANN 性能提高百分比

benchmark	相比 RankBoost 性能提高百分比（%）	相比 ANN 性能提高百分比（%）
Matrix	5.53	27.79
BFS	6.37	49.26
CP	6.93	12.66
RAY	4.86	31.76

下图 4-23，分别统计了 RankBoostTree 算法相比 RankBoost 算法在函数加速器以及 GPU 设计空间探索所有的 benchmark 上的性能提高百分比；下表 4-15 分类统计了 RankBoostTree 算法对比 RankBoost 算法在函数加速器设计空间探索性能、功耗预测以及在 GPU 设计空间探索性能预测的提升百分比，并计算了平均预测准确率提高的百分比。

下图 4-24，分别统计了 RankBoostTree 算法相比 ANN 算法在函数加速器以及 GPU

设计空间探索所有的 benchmark 上的性能提高百分比；下表 4-16 分类统计了 RankBoostTree 算法对比 ANN 算法在函数加速器设计空间探索性能、功耗预测以及在 GPU 设计空间探索性能预测的提升百分比，并计算了平均预测准确率提高的百分比。

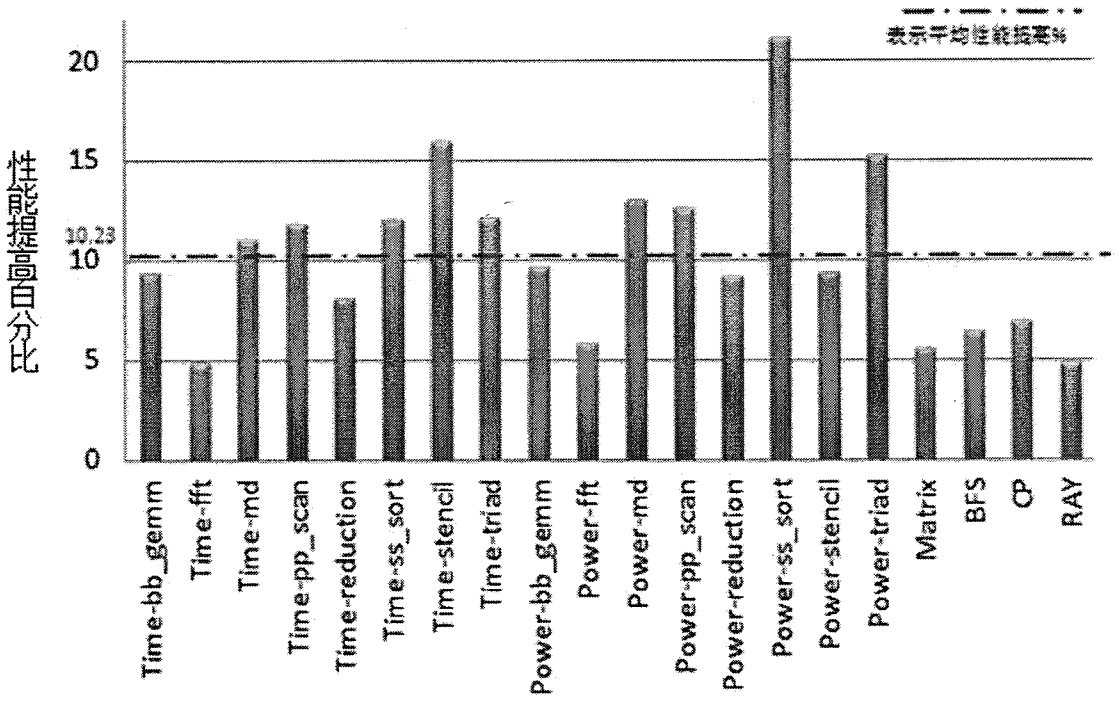


图 4- 23　RankBoostTree 对比 RankBoost 预测性能提高百分比

表 4- 15　RankBoostTree 对比 RankBoost 预测性能提高百分比汇总数据

研究项目	平均性能提高百分比（%）
函数加速器性能预测	10.63
函数加速器功耗预测	11.99
GPU 性能预测	5.92
平均性能提高%	10.23

表 4- 16　RankBoostTree 对比 ANN 预测性能提高百分比汇总数据

研究项目	平均性能提高百分比（%）
函数加速器性能预测	10.38
函数加速器功耗预测	26.56
GPU 性能预测	30.37
平均性能提高%	20.85

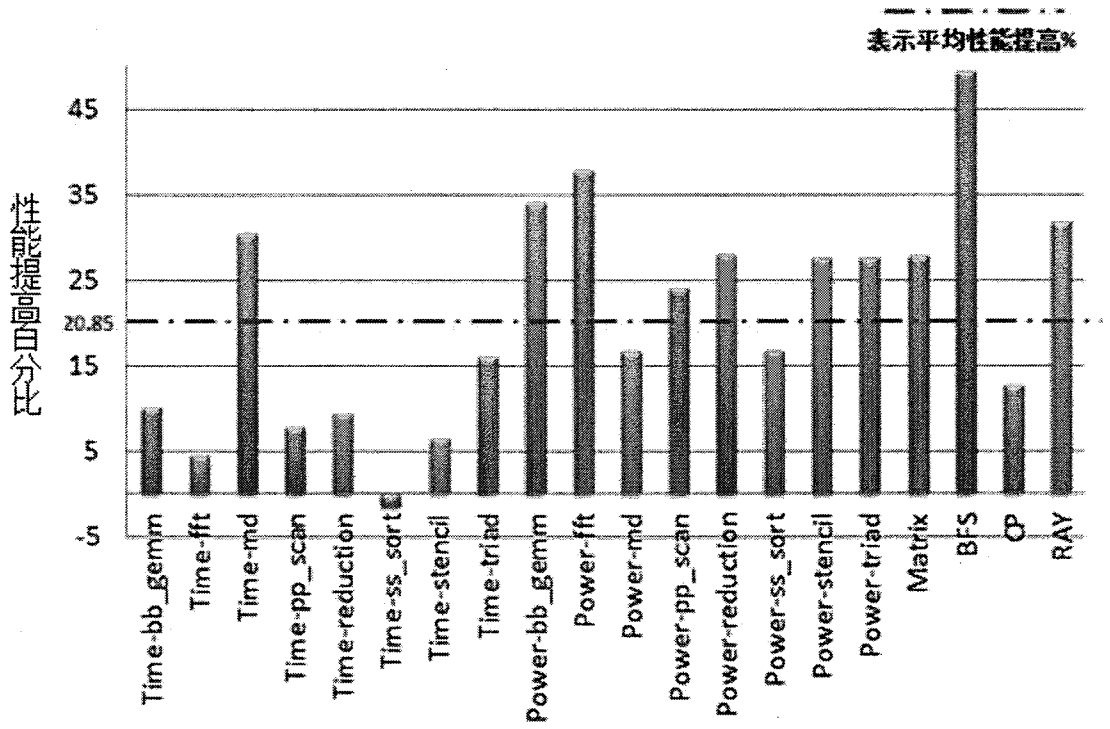


图 4-24 RankBoostTree 对比 ANN 预测性能提高百分比

4.2.4 RanBoostTree 实验结果总结

通过上述实验，总结得到以下几点结论：

1）分别在函数加速器设计空间探索数据集以及GPU设计空间探索数据集上验证了RankBoostTree 算法，证实了排序学习算法在加速器设计空间探索中的应用可行，并且性能表现良好；

2）同时，通过对比 RankBoostTree 算法和 RankBoost、ANN 算法在加速器设计空间探索的排序关系预测准确率，发现，RankBoostTree 相对于 RankBoost 算法，在函数加速器设计空间探索的性能预测准确率提高 10.63%，功耗预测准确率提高 11.99%，在GPU 设计空间探索预测准确率提高 5.92%，平均预测准确率提高 10.23%。RankBoostTree 相对于 ANN 算法，在函数加速器设计空间探索的性能预测准确率提高 10.38%，功耗预测准确率提高 26.56%，在 GPU 设计空间探索预测准确率提高 30.37%，平均预测准确率提高 20.85%。

4.3 RankBoost 算法改进前后性能对比

4.3.1 一次迭代运行时间对比

以函数加速器设计空间探索为例说明。表 4-17 统计了函数加速器设计空间探索数据集分别执行基于决策桩的 RankBoost 算法和基于决策树的 RankBoostTree 算法，迭代 10 次所用的时间：

表 4- 17　一次迭代的执行时间 RankBoost 对比 RankBoostTree

对比项	RankBoost(单位：秒)	RankBoostTree(单位：秒)
bb_gemm	43.82	46.16
fft	10.83	12.36
md	10.87	12.24
pp_scan	11.46	13.22
reduction	11.01	12.96
ss_sort	45.43	46.29
stencil	10.95	12.45
triad	43.20	44.89
平均迭代一次所用时间	2.34	2.50
RankBoostTree 相对于 RankBoost 算法迭代时间开销增加%		6.8%

4. 3. 2 达到相同准确率迭代次数比较

仍然以函数加速器设计空间探索为例说明。设相同的准确率为 85%，比较基于决策桩的 RankBoost 算法和基于决策树的 RankBoostTree 算法预测准确率达到 85%以上时所需要的迭代次数：

表 4- 18　达到相同准确率所需迭代次数比较

对比项	RankBoost 所需迭代次数	RankBoostTree 所需迭代次数
bb_gemm	90	20
fft	20	10
md	50	10
pp_scan	150+	20
reduction	10	10
ss_sort	80	10
stencil	150+	10
triad	50	10
平均所需迭代次数	75+	12.5
RankBoostTree 相对于 RankBoost 算法平均迭代次数效率提高		5x

4. 3. 3 达到相同准确率所需训练样本大小比较

以函数加速器设计空间探索举例说明。相同准确率设为 85%，比较基于决策桩的 RankBoost 算法和基于决策树的 RankBoostTree 算法在迭代次数 T＝50 时，预测准确率达到 85%以上所需要的训练样本大小比较：

表 4-19 达到相同准确率所需训练样本大小比较

对比项	RankBoost 所需训练样本大小	RankBoostTree 所需训练样本大小
bb_gemm	400	100
fft	60	50
md	50	60
pp_scan	500+	140
reduction	40	80
ss_sort	250	80
stencil	500+	70
triad	200	90
平均所需训练样本大小	250	83.7
RankBoostTree 相对于 RankBoost 算法平均所需样本大小减少%		66.5%

4.3.4 实验结论

通过对比优化前后的算法性能发现，决策树作弱排序器的 RankBoostTree 算法和决策桩作弱排序器的 RankBoost 算法相比，决策条件更加复杂，然而每次迭代带来的时间开销仅增加了 6.8%；另外，在算法迭代效率方面（用达到相同预测准确率所需要的迭代次数来衡量），优化后的 RankBoostTree 的算法迭代效率相比 RankBoost 提高了 5 倍，并且，在达到相同准确率情况下所需要的训练样本大小减少了 66.5%。

4.4 参数寻优实验讨论

4.4.1 RankBoostTree 算法阈值参数 θ 的取值策略

在第 3 章的 3.2 小节有提到过，样本类别标签公式中的阈值参数 θ 的取值，有以下几种方案：

表 4-20 样本类别标签公式中阈值参数 θ 的取值方案

方案	参数 θ 的取值
1	所有样本 Tag 值处于第 25%的值
2	所有样本 Tag 值的中位数
3	所有样本 Tag 值处于第 75%的值
4	使得排序损失函数[定义在[43]]达到当前最小的 Tag 值

我们针对这 4 种方案，以函数加速器设计空间探索性能预测为例，分别进行了实验。实验结果如下表所示：

表 4-21 不同参数设定方案预测准确率统计

Benchmark	方案 1 预测准确率%	方案 2 预测准确率%	方案 3 预测准确率%	方案 4 预测准确率%
bb_gemm	89.26%	90.12%	89.84%	90.69%
fft	89.28%	90.94%	89.84%	90.88%
md	89.89%	91.38%	90.07%	92.32%
pp_scan	90.03%	90.33%	90.54%	91.40%
reduction	88.92%	91.28%	90.01%	91.27%
ss_sort	88.86%	92.89%	90.30%	93.67%
stencil	90.87%	93.65%	90.34%	94.23%
triad	91.24%	90.57%	90.11%	90.52%

由上述实验发现，方案 4 的阈值参数 θ 的取参策略效果最好，方案 2 次之。因而，在 4.2 节和 4.3 节进行的所有 RankBoostTree 算法的实验，阈值参数 θ 都按照方案 4 来设定。

4.4.2 RankBoost 算法的参数 θ 取值策略

在 RankBoost 算法中，调整阈值参数 θ 也是一个费时的过程。实验过程中采取了两种方案思路，第一种是设定阈值参数 θ 的上下边界，并设定一个步长，从下边界开始，以所设定的步长为单位，依次递增步长个单位，直到到达上边界为止。如果步长设定的比较小，迭代过程慢，影响实验进度；如果步长设定较大，容易错过最佳阈值参数。因而，这种方案对实验者的经验技巧以及耐心都有比较高的要求，并且实验非常耗时。

另一种思路是不设定固定的上下边界，并且不设定固定的步长。这种方案是根据采样点的取值情况来定。所有的候选阈值参数 θ 都来自于采样点，候选阈值参数 θ 的可取值即为选定特征下的所有采样点的特征值。这种寻找最佳阈值参数 θ 的过程不需要人工来设定，并且整个过程不需要人工干涉，也不需要频繁更改步长。

RankBoost 算法采用第二种调整参数的方案可以达到与第一种方案相当的预测准确率，并且简单易实际操作，不需要太多的人工干涉。因而，本文实现 RankBoost 算法采用的是第二种调整参数方案。

4.5 本章小结

本章首先介绍了函数加速器以及 GPU 的设计空间，设计空间的容量以及实验执行的 benchmark 描述；然后介绍了实验采样得到的数据集规模以及训练集和测试集的划分，基于这样的数据集进行基于 RankBoostTree 算法的设计空间探索实验。分别进行了以下实验：

(1) 基于 RankBoostTree 算法实现函数加速器以及 GPU 设计空间的探索，并与回归

模型中性能表现良好的 ANN 算法作对比，比较预测偏序关系的正确率，验证 RankBoostTree 算法可以成功应用到加速器设计空间探索中。在函数加速器设计空间探索，实验测试了算法在 SHOC 测试包中 bb_gemm，fft，md，pp_scan，reduction，ss_sort，stencil，triad 上的性能以及功耗预测准确率；在 GPU 设计空间探索中，实验测试了算法在 ISPASS2006-benchmark 和 CUDA SDK 中 BFS，CP，RAY 和 Matrix 上的性能预测准确率。实验结果表明，与 ANN 算法相比，RankBoostTree 算法在加速器设计空间探索准确率平均提高 20.85%；

(2) 对比 RankBoostTree 算法和经典 RankBoost 算法的预测准确率，验证算法预测准确率的提升。实验结果表明，RankBoostTree 相对于 RankBoost 算法，在函数加速器设计空间探索的性能预测准确率提高 10.63%，功耗预测准确率提高 11.99%，在 GPU 设计空间探索预测准确率提高 5.92%，平均预测准确率提高 10.23%；

(3) 基于函数加速器设计空间对比了改进后的 RankBoostTree 算法和传统的 RankBoost 算法的迭代性能。比较了算法改进前后的平均迭代时间、达到相当预测准确率所需要的迭代次数以及达到相当准确率所需要的训练数据集大小。实验结果发现，决策树作弱排序器的 RankBoostTree 算法和决策桩作弱排序器的 RankBoost 算法相比，虽然决策条件更加复杂，但是每次迭代带来的时间开销仅增加了 6.8%，在可接受的范围内；另外，在算法迭代效率方面（用达到相同预测准确率所需要的迭代次数来衡量），优化后的 RankBoostTree 的算法迭代效率相比 RankBoost 提高了 5 倍，并且，在达到相同准确率情况下所需要的训练样本大小减少了 66.5%。

(4) 讨论了 RankBoostTree 算法和 RankBoost 算法在调整阈值参数 θ 的几种策略，并分析了几种策略的优缺点，选择较优的调整阈值参数 θ 的方案实现算法，提高工程实践效率。

第5章 结束语

5.1 本文工作总结

加速器设计空间探索，对于异构系统设计以及加速器设计意义重大，同时也充满挑战。本文受陈天石等人工作[36]的启发，将体系结构设计空间探索问题由传统的回归问题转化为排序问题，尝试将机器学习方法应用于加速器设计空间探索中。本文实现了基于决策桩的传统 RankBoost 算法，可能受决策桩决策规则简单的影响，在实验中发现传统 RankBoost 算法应用于加速器设计空间探索预测性能并不理想，本文采用基于决策树作弱排序器的 RankBoostTree 算法，改进传统 RankBoost 算法决策规则简单，性能过于依赖于数据集的分布，影响应用领域可扩展性的弊端。

本文将机器学习中排序学习的算法应用到加速器设计空间探索问题中来，验证了其可行性。本文首先介绍了基于决策桩的 RankBoost 算法；然后介绍了基于决策树作弱排序器的 RankBoostTree 算法，并详细描述了算法应用到设计空间探索中的细节处理；最后，我们将 RankBoostTree 算法应用到加速器设计空间探索的中，分别进行了以下实验：

1） 将 RankBoostTree 算法应用于加速器设计空间探索中函数加速器设计空间探索，GPU 设计空间探索两大类问题，证明了排序学习算法可以成功应用到加速器设计空间探索中。在函数加速器设计空间探索，实验测试了算法在 SHOC 测试包中 bb_gemm，fft，md，pp_scan，reduction，ss_sort，stencil，triad 上的预测性能；在 GPU 设计空间探索，实验测试了算法在 ISPASS2006-benchmark 和 CUDA SDK 中 BFS，CP，RAY 和 Matrix 上的预测性能；

2） 本文对比了 RankBoostTree 算法和经典 RankBoost 算法以及归回模型中性能表现优异的 ANN 算法在同等实验条件下的预测准确率，实验结果发现，与 RankBoost 相比，RankBoostTree 算法在加速器设计空间探索准确率平均提高 10.23%，与 ANN 算法相比，RankBoostTree 算法在加速器设计空间探索准确率平均提高 20.85%；

3） 本文基于函数加速器设计空间对比了改进后的 RankBoostTree 算法和传统 RankBoost 算法的性能。比较了算法改进前后的平均迭代时间、达到相当预测准确率所需要的迭代次数以及达到相当准确率所需要的训练数据集大小。实验结果发现，决策树作弱排序器的 RankBoostTree 算法和决策桩作弱排序器的 RankBoost 算法相比，虽然决策条件更加复杂，但是每次迭代带来的时间开销仅增加了 6.8%，在可接受的范围内；另外，在算法迭代效率方面（用达到相同预测准确率所需要的迭代次数来衡量），优化后的 RankBoostTree 的算法迭代效率相比 RankBoost 提高了 5 倍，并且，在达到相同准确率情况下所需要的训练样本大小减少了 66.5%。

5.2 下一步研究方向

本文用模拟器采集设计空间数据样本。然而，其实，在体系结构领域，获得不同体系结构设计参数配置下的体系结构响应有两种主流方法，且两种方法不分伯仲，一种是本文采用的模拟器模拟的方法；另外一种是分析模型的方法。最早提出分析模型的是 TS Karkhanis 等人在 2004 年提出的一阶超标量处理器模型[53]。因而，下一步工作可以基于分析模型采集样本，或者通过分析模型扩充样本，进一步提高学习排序算法在体系结构设计空间探索的预测准确率。