第 1 章　引　　言

1.1　本书背景及意义

过去的几十年，人类借助并行程序的强大计算能力，极大地推进了科学研究、军事国防和日常生活等许多领域的发展。在自然科学领域，研究人员通过并行程序实现高效准确的气象预报[1-3]、洋流模拟[4]和地震模拟[5-6]，在自然灾害来临时尽可能地保护人民生命财产安全；在医学领域，并行程序促进了基因、蛋白质的分析[7-9]和药物的研制，值得一提的是，它在新冠肺炎病毒 COVID-19 的感染机制分析和药物疫苗研制中发挥了重大作用[10-11]；在航空航天领域，基于并行程序的燃烧模拟和空气动力学模拟等应用[12]保障了航空航天飞机的安全航行和火箭的成功发射；在军事领域，研究人员使用并行程序协助军事装备与新一代核武器的研制[13]。此外，日常生活中诸如社交网络[14]、自媒体平台等现代科技产物都与并行程序密切相关。

性能①是并行程序最关键的指标之一，用于反映并行程序的运行效率。并行程序的高效运行对社会的稳定运转意义重大。例如，气象预报、洋流模拟和地震模拟等应用需要进行高效且精确的模拟，才能在自然灾害前作出及时预警，从而保障人民生命和财产安全。铁路票务系统需要快速响应购票请求并及时更新票务信息，才能在春节期间保障超过十亿人次的顺利出行。同时，随着人类科学的进步，许多领域对并行程序的性能需求也不断增大。例如，科学家试图通过蛋白质的折叠等结构特征，研究亨廷顿病等疑难杂症的病理，然而复杂蛋白质分子结构的精密分析需要并行程序具备强大的计算性能。

① 本书中的性能（performance）指并行程序的运行效率，即完成一定负载量所用的时间或单位时间内完成的负载量。一些文献中的性能含义为程序计算结果的精度。

世界各国的政府和企业投入大量资源建造大规模超级计算机，试图通过提升计算力以提升并行程序的性能。美国的 Summit 有超过 200 万的处理核，并达到 200PFLOPS[①]以上的理论峰值性能。日本的富岳（Fugaku）超级计算机的处理核的数量超过 700 万，理论峰值性能达到 500PFLOPS 以上。在国内，我国自主研制的天河 2A 超级计算机有近 500 万个处理核，理论峰值性能超过 100PFLOPS。“神威·太湖之光”超级计算机拥有 40960 个计算节点，超过 1000 万处理核，理论峰值性能为 125PFLOPS。近年来，我国陆续投入数十亿建造新一代 E 级超级计算机，可实现每秒百亿亿次数学运算。

超级计算机的规模扩增带来了计算力的迅速增长，但并非所有并行程序都能有效且高效地利用超级计算机的计算力。例如，高度共轭梯度基准测试（high performance conjudate gradient，HPCG）[15] 是一个评估高性能计算机性能的重要基准测试程序，图 1.1 展示了 TOP500 排行榜[16]②排名前十的超级计算机的理论峰值性能和 HPCG 的性能③对比。HPCG 在世界排名第一的超级计算机上的实测性能仅能达到理论峰值性能的 2.98%。HPCG 是高性能领域专家总结科学计算中的典型计算模式而提出的基准测试程序，其计算模式和数据都相对规整。规整的 HPCG

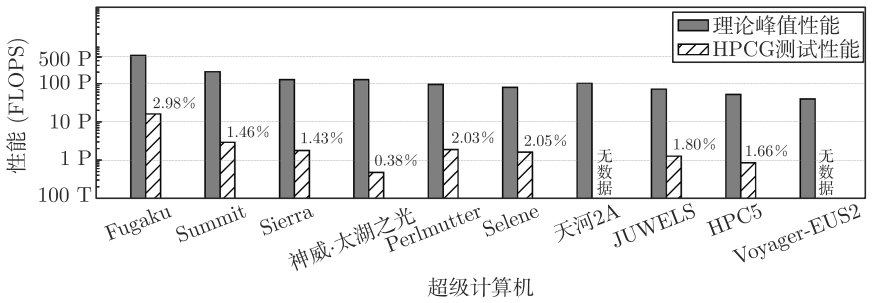

图 1.1 世界领先超级计算机峰值性能与 HPCG 性能对比

① 每秒浮点操作次数（floating-point operations per second，FLOPS）是一种衡量计算能力的标准，指每秒执行的浮点操作次数。PFLOPS（peta floating-point operations per second）表示每秒千万亿次浮点操作。

② TOP500 排行榜列举全世界计算力最强的 500 台超级计算机，并包含这些超级计算机的相关信息介绍。该排行榜每年分别于 6 月和 11 月更新两次。

③ 图中数据来源于 TOP500 排行榜 2021 年 11 月发布的榜单数据，其中天河 2A 和 Voyager-EUS2 未提交最新的 HPCG 性能测试结果。

在超级计算机上尚且无法有效利用计算资源，真实科学应用的效率更远不及此[1,5-6]。

并行应用的低效不仅浪费了大量资源，更延缓了科学研究和军事国防等各个领域的发展。因此，提高并行应用的性能刻不容缓。图 1.2 展示了通常情况下提高并行程序性能的示意流程。研究人员首先需要对并行程序进行性能分析以定位其性能瓶颈，然后针对该瓶颈进行特定的性能优化。优化后的并行程序继续通过性能分析定位性能瓶颈，而后进行进一步优化。反复上述过程，直至并行程序的性能达到相对较高的水平。上述过程中，性能分析和性能优化是提高并行程序性能的关键技术。目前已有大量的国内外研究工作针对性能分析和优化技术展开了深入研究（详见第 2 章）。性能分析主要通过对并行程序运行时性能行为进行分析，以定位程序的性能瓶颈，如负载不均、资源竞争、通信阻塞等。超级计算机和真实应用的复杂性导致性能瓶颈的定位十分困难，通常需要大量的分析开销。性能优化主要通过对并行程序的源码修改或运行时调度的方式，以尽可能消除或削弱程序中的性能瓶颈对性能的负面影响。然而，应用程序和硬件平台的复杂性导致优化策略的变体繁多，通常需要大量的人力开销选择并实施最佳优化策略。一些研究工作尝试通过性能分析技术指导性能优化策略选择，以降低部分人力开销。截至目前，国内外相关研究工作仍未完全解决性能分析和优化技术中的问题，其中主要的不足之处在于：①在性能分析方面，现有工作通常引入极大的运行时、存储开销或

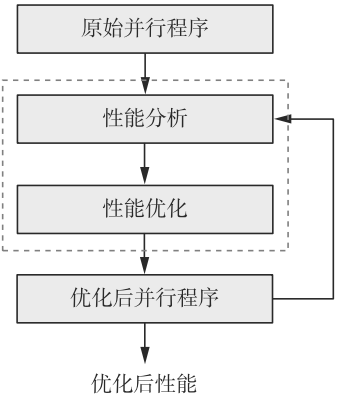

图 1.2　提升并行程序性能的流程图

手动分析开销，且只能针对简单或特定性能瓶颈进行分析。②在性能优化方面，现有工作难以同时考虑应用程序和硬件平台特性以实现全面且深入的优化，优化效果存在明显的提升空间。

本书针对上述问题进行了深入的研究，分别提出了有效的解决方案。第一，针对复杂大规模并行程序，通过结合编译技术和图分析技术，提出了一种轻量级可扩展性瓶颈检测技术；第二，面对繁杂的性能分析场景，设计了一个面向性能分析的领域特定编程框架，以降低性能分析工具的开发复杂性；第三，设计了一个异步策略感知的性能模型，用于指导性能优化策略的选择，并以典型科学应用程序——测试高性能计算集群系统浮点性能的基准程序（the high-performance linkpack benchmark，HPL）为案例进行深入分析；第四，设计并实现了面向领域的多层次性能优化框架，全面感知应用程序和硬件平台特性，借助编译和建模技术自动实现深度优化，并以计算流体力学为例实现对框架的初步验证。

1.1.1 性能分析技术的现状

性能分析技术对于理解和优化应用程序是必不可少的，具体指以收集程序运行时信息为手段研究程序行为的分析方法。自动分析并行应用程序的性能可以有效提高程序员理解和优化应用程序的效率。因此，研究人员开发了许多自动的通用性能数据采集和分析工具。现有的通用性能数据采集和分析工具主要分为两类，分别是基于程序概要的工具和基于事件轨迹的工具。

基于程序概要的工具在运行时定期中断程序并记录快照，采集程序的整体统计性能数据，例如函数的总执行时间、被调用次数等，因此它们的运行时开销和性能数据的存储开销都非常小。典型的基于程序概要的性能分析工具包括 VTune[17]、HPCToolkit[18]、mpiP[19] 和 Gprof[20] 等。例如，HPCToolkit 对于大规模并行程序进行性能数据采集时仅引入 1%∼5%的运行时开销。然而，统计性能数据仅记录了某些时间戳的程序快照，而时间戳之间的程序行为未被完整地记录下来，因此该类工具丢失了进行深入分析所需要的关键信息（例如进程内数据依赖、进程间通信模式等）。这些信息的缺失导致该类工具只支持简单的性能分析（例如热点分析、均衡性分析等），而对于复杂场景，则需要手动进行大量深入分析。

基于事件轨迹的工具跟踪并记录程序执行期间的所有事件，其日志中包含丰富的信息，包括计算、内存访问和通信模式等。典型的基于事件轨迹的工具包括 Scalasca[21]、ITC/ITA[22]、Vampir[23] 和 TAU[24] 等。事件轨迹中包含了所有性能分析所需的信息，研究人员可以基于这些数据进行各种各样的深入性能分析。此类方法的缺点是事件轨迹的记录通常需要非常高昂的运行时开销和数据存储开销。例如，Scalasca 的运行时开销可达数十倍至数百倍，其记录事件轨迹的日志文件通常可达 GB（Gigabyte，千兆字节）至 TB（Terabyte，兆兆字节）规模。

大规模并行程序的性能分析对开销要求很高，往往需要结合分析场景设计轻量级的分析方法[25-28]。此外，应用程序的复杂性（例如计算负载、通信模式和数据依赖等）和硬件平台的复杂性（如异构加速器件和网络架构等），导致了繁多而复杂的性能分析场景，需要程序员针对各种场景进行特定的深入分析，才能定位性能问题。一些研究工作针对某些特定场景，提出任务驱动的分析方法，可以实现精确的分析[29-32]。然而，这些研究工作提出的性能分析任务不能覆盖所有的分析场景。对于一个新的场景，仍然需要手动开发特定的性能分析工具，这给程序员带来了巨大的编码压力。

1.1.2 性能优化技术的现状

性能优化技术是提升并行程序性能的必要手段，具体指通过修改并行程序的计算、访存和通信等特征以加速程序的执行。一种常见的性能优化方式是程序员基于并行程序和目标硬件平台的特性手动进行移植和优化，这种方式需要程序员具有非常高超的性能优化技巧并投入大量的精力。例如，清华大学团队在“神威·太湖之光”上移植和优化美国大气研究中心开发的社区大气模式应用 CAM5，累计投入了 10 人/年以上的人力成本[33]。中国科学院大气物理研究所和计算机网络信息中心的联合团队在先导一号上移植和优化高分辨率海洋模式 LICOM3 预报系统的人力成本为 6~10 人/年[34]。实现大规模超级计算机上并行应用的移植和优化需要如此高昂的人力投入，对于任何研究团队都是一种沉重的负担。为了提高并行程序的优化效率，研究人员提出了一些框架以降低优化的人力开销，包括统一编程框架和自动优化框架。

统一编程框架对并行程序的数据结构和执行模式等概念进行抽象，并提供编程接口（或称原语）以降低程序员实现特定优化（例如归约、分块、循环划分等）的编码开销。典型的通用优化框架包括 Kokkos[35]、RAJA[36]、SYCL[37]、OpenCL[38] 和 DaCe[39] 等。然而，真实应用中通常存在非常复杂的依赖关系，程序员仍然需要针对应用设计特殊的复杂优化策略，并通过大量原语描述这些优化策略。

自动优化框架的目标是实现自动性能优化，其中最典型的一类是领域特定优化框架。它是针对特定领域设计的一种专用优化框架，通过抽象领域特征进行应用描述并自动实现相关优化，具有代表性的领域特定优化框架包括 Halide[40]、TACO[41]、Taichi[42]、TVM[43] 和 Stella[44] 等。领域特定优化框架通常侧重特定领域内计算模式的表达能力，不能同时兼备对目标硬件平台的表达和优化能力，导致无法全面结合应用和硬件的信息，错失一些性能优化机会。

现有的性能优化工作尝试结合编译技术降低优化的人力开销，但仍然难以实现自动且全面感知应用和硬件的优化策略选择。本书通过表达能力更强的编译框架，更全面地考虑应用程序和硬件平台的特性，并结合性能建模技术指导优化策略选择。

1.2 大规模并行程序性能分析与优化面临的关键问题

随着摩尔定律（Moore's law）和登纳德缩放定律（Dennard scaling）的逐渐失效，高性能计算机呈现出两种主要趋势，分别是规模扩增和引入异构加速器件。一方面，随着超级计算机规模的扩增，其网络架构愈加复杂，从而导致并行应用的通信也更加复杂。另一方面，异构架构带来的不仅是异构加速器件的卓越性能，还引入了主机端与异构加速器件之间协同工作的复杂性，在复杂化并行应用内部数据依赖的同时，也加重了程序员对并行程序的异步协同设计负担。

综上，超级计算机的发展使得并行程序更加复杂，包括并行程序的计算通信模式和数据依赖等。超级计算机的硬件复杂性和并行程序的应用复杂性给性能分析和优化带来了诸多挑战。目前性能分析和优化技术面临以下几个关键问题：

（1）复杂大规模并行程序的性能分析开销大。传统的性能分析技术主要分为基于事件轨迹的分析和基于程序概要的分析。基于事件轨迹的方法所记录的事件轨迹日志带来了高昂的运行时和数据存储开销。随着并行程序规模的扩大，开销也更加高昂。基于程序概要的方法的运行时和存储开销相对较低，但程序概要缺失了许多关键信息，导致该类方法难以在面对大规模并行程序中的复杂通信与计算模式时实现精确的性能瓶颈定位。传统方法无法以较低的成本分析复杂大规模并行程序的性能瓶颈。为了实现面向复杂大规模并行程序的轻量级性能瓶颈定位，需要研究有效且高效的技术以降低各类开销。

（2）性能分析场景繁杂且针对性分析工具的开发复杂性高。大规模并行应用中包含复杂的数据和控制依赖以及复杂的线程间锁和进程间通信模式，这些因素均会使性能问题更加复杂，需要程序员进一步深入分析以定位性能瓶颈。然而，现有的通用性能分析工具和特定功能的性能分析工具通常无法覆盖所有复杂场景下的性能分析，程序员需要手动开发针对特定场景的性能分析工具。但实现这些特定性能分析工具需要大量的专业知识和手动编码，导致程序员的开发效率十分低下。因此，需要研究相关技术，降低性能工具开发的复杂性，协助程序员进行高效分析。

（3）并行程序的优化策略繁多且难以选择。定位性能瓶颈后，程序员需要对并行应用作出针对性优化。然而，并行程序和大规模集群的复杂性导致性能优化策略的变体繁多且非常复杂，它们在超级计算机上运行时性能行为也难以预测。通常程序员需要手动实现所有策略后再依据测试结果进行择优，这将消耗程序员的大量精力并浪费巨大的资源。因此，需要研究相关的性能分析和预测技术协助程序员找到合适的优化策略，从而指导性能优化。

（4）复杂并行程序的全面深入优化需投入大量人力。手动分析和优化复杂并行程序通常需要极大的人力开销。一些研究工作借助编译技术实现自动优化框架，将程序员的优化压力降至最低。但目前的自动优化框架难以针对复杂并行程序全面挖掘应用程序和硬件平台的优化机会。它们通常将前端语言转换为单一的中间表达，并在该表达上进行各种优化操作，然后生成目标执行代码。然而，单一的中间表达不能兼顾对应用特征和底层硬件的完整描述，导致无法进行面向应用和硬件的全面优化。

此外，并行程序的一些优化策略需要结合应用和硬件的复杂特性进行选择，单一中间表达因缺失全面的信息，无法搜索最佳的优化策略。目前的自动优化框架仍然需要手动设计并编码实现部分优化策略，以实现全面且深入的性能优化。因此，需要研究有效的技术感知领域应用和硬件平台的性能优化机会，自动实现全面且深入的性能优化。

1.3 本书的主要研究内容与贡献

1.3.1 本书的主要贡献

本书结合编译技术和图分析技术，对大规模并行程序的性能分析和优化进行了深入研究。本书的创新和主要贡献包括：

（1）提出了一个基于图的并行程序可扩展性瓶颈检测系统 ScalAna。ScalAna 使用静态编译时分析获取程序的结构、控制流和数据流等信息，使得运行时能以极低的开销获取必要的性能数据。同时，本书发现并行程序的性能问题会通过控制流和通信传播至其他代码段。基于此发现，本书进一步提出了基于图的反向追踪技术，能够实现自动的可扩展性瓶颈检测。ScalAna 在自动定位可扩展性瓶颈的同时，显著地降低了运行时和存储开销。

（2）提出了一套面向性能分析的领域特定编程框架 PerFlow。PerFlow 能够分别对性能分析过程和程序性能行为进行抽象。首先，PerFlow 将性能分析任务的逐步分析过程抽象为数据流图。该数据流图由性能分析子任务组成，这些子任务可以由该框架的内置分析库提供，也可以由用户依据其特定分析需求实现。其次，PerFlow 以程序抽象图表示程序性能行为，允许用户利用各种图操作和图算法访问和分析程序性能，从而实现性能分析子任务。此外，PerFlow 基于二进制文件进行分析，适用于生产环境。实现结果表明，PerFlow 可以有效地降低程序员开发特定场景下性能分析工具的编码复杂性。本书成功将 PerFlow 部署于北京应用物理与计算数学研究所的生产环境中，指导 JASMIN 应用的软件参数调优。

（3）提出了一个异步策略感知的精确性能建模技术 ASMOD。ASMOD 可以预测不同异步策略下的程序性能，从而指导程序员进行性能

优化。ASMOD 将不同异步策略下的并行程序表示为有向图，图中的点代表并行程序的一个模块，边表示模块之间的依赖关系。通过解析—统计结合建模技术和层次化建模技术建立程序各个模块的模型，保障模型的准确性、高效性和可移植性。同时，ASMOD 提出一种基于图的硬件感知模拟技术以预测异步策略在特定硬件平台上的性能。本书以典型科学应用 HPL 为例，验证该技术的有效性和高效性。实验结果表示，ASMOD 可以准确地预测不同异步策略下 HPL 的性能。例如，ASMOD 对“神威·太湖之光”上超 400 万核规模的 HPL 进行性能预测，误差低至 1.09%。

（4）设计并实现了一个面向领域的多层次性能优化框架 Puzzle。Puzzle 采用多层次中间表达表示领域层、通用层和硬件层的特性，在各层中间表达上可以轻松实现对应优化。Puzzle 引入性能建模技术，结合多层次的特性，在各层次中指导优化策略选择。此外，Puzzle 支持生成多种硬件平台的可执行文件，具有良好的代码可移植性和性能可移植性。本书以计算流体力学为例，验证该框架的设计。本书提出了针对流体力学领域的领域中间表达，并提供一套领域特定语言降低程序员编程复杂性。实验表明，Puzzle 可以更有效地提升计算流体力学领域应用的性能。

1.3.2 本书的组织及各章内容简介

本书共分为 7 章，每一章的具体组织如下：

第 1 章概述了并行程序的性能分析和优化的研究背景与研究意义，介绍了目前性能分析和优化领域面临的主要问题与本书的主要贡献。

第 2 章介绍了国内外相关工作的研究现状。重点分析了国内外研究单位在并行程序性能分析和优化方面几项关键技术的研究现状。

第 3 章介绍了基于图的并行程序可扩展性瓶颈检测系统 ScalAna。首先介绍了如何通过结合静动态分析技术获取程序结构信息，构建了表示程序性能的图；其次介绍了如何通过图分析技术在表示程序性能的图上进行可扩展性瓶颈检测；最后通过多个基准测试程序和真实应用对 ScalAna 的时间高效性、空间高效性及准确性进行了验证。

第 4 章介绍了面向性能分析的领域特定编程框架 PerFlow。首先介绍了如何对程序性能进行抽象和统一表示；其次介绍了基于数据流图的性能分析任务抽象，并以多个示例展示如何通过 PerFlow 编写性能

分析任务；最后以多个基准测试程序和真实应用验证了 PerFlow 的高效性和准确性并可以显著减轻用户编码压力。

第 5 章介绍了异步策略感知的精确性能建模技术 ASMOD。首先介绍了如何通过结合解析—统计建模技术建立高效率的性能模型；其次介绍了模型如何感知异步策略及如何通过图分析技术实现高效性能预测；最后以高性能计算中典型科学计算程序 HPL 为例，验证该建模方法的高效性和准确性。

第 6 章介绍了面向领域的多层次性能优化框架 Puzzle。首先介绍了优化框架的整体架构，接着以计算流体力学为例，介绍了该框架的领域中间表达设计；其次介绍了中间表达的逐步递降过程；再次介绍了领域层、通用层和硬件层中的多层次优化，以及性能模型如何指导优化策略选择；最后测试多个计算流体力学应用在 Puzzle 框架上的优化效果。

第 7 章总结了全书并提出了进一步研究的方向。