



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

# 博士学位论文

大规模并行多层次不连续非线性可扩展理论研究及应用

作者姓名: 李琨

指导教师: 张云泉 研究员

中国科学院计算技术研究所

学位类别: 工学博士

学科专业: 计算机系统结构

培养单位: 中国科学院计算技术研究所

2022年6月



**Reserach and Application on Multi-level Discontinuous and  
Nonlinear Scalability for Massively Parallelism**

---

**A dissertation submitted to  
University of Chinese Academy of Sciences  
in partial fulfillment of the requirement  
for the degree of  
Doctor of Engineering  
in Computer Architecture**

**By  
Li Kun  
Supervisor: Professor Zhang Yunquan**

**Institute of Computing Technology, Chinese Academy of Sciences**

**June, 2022**



## **中国科学院大学**

### **学位论文原创性声明**

本人郑重声明：所呈交的学位论文是本人在导师的指导下独立进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的研究成果。对论文所涉及的研究工作做出贡献的其他个人和集体，均已在文中以明确方式标明或致谢。

作者签名：

日 期：

## **中国科学院大学**

### **学位论文授权使用声明**

本人完全了解并同意遵守中国科学院有关保存和使用学位论文的规定，即中国科学院有权保留送交学位论文的副本，允许该论文被查阅，可以按照学术研究公开原则和保护知识产权的原则公布该论文的全部或部分内容，可以采用影印、缩印或其他复制手段保存、汇编本学位论文。

涉密及延迟公开的学位论文在解密或延迟期后适用本声明。

作者签名：

导师签名：

日 期：

日 期：



## 摘要

高性能计算是计算机科学的重要分支，主要指从体系结构、并行算法和并行软件等多个方面研究和利用高性能计算机的技术。数十年来，高性能计算被广泛地应用在大规模科学和工程计算、人工智能、工业仿真等关键领域，对基础科学发现、国民经济发展和国防工业建设具有极高的应用价值，已成为国家科技综合实力的重要体现。

图灵奖得主 Jim Gray 指出，可扩展性问题是高性能计算中的核心问题，也是未来信息技术领域十二项研究目标中排在首位的重要挑战。针对高性能计算中的并行软件设计和并行系统结构一软一硬两个层次，可扩展性挑战也表现在架构移植难、算法设计难、应用优化难等诸多方面。本文对大规模并行多层次不连续非线性可扩展理论开展研究，深入分析可扩展性发展规律，提出物理模型、并行算法以及性能优化多层次协同设计方法，在多种硬件并行规模、不同软件并行粒度、各级交叉并行应用上开展可扩展性优化设计，具体贡献包括：

1. 提出了大规模并行多层次不连续非线性可扩展理论。基于高性能计算中的不连续非线性可扩展现象，通过对并行软件在同构多核到异构众核系统上可扩展性的分析，首次系统地对多个层次上的两种现象进行了深入的理论分析与内涵丰富，并提出了可扩展性的物理模型-并行算法-性能优化多层次协同设计理论。该可扩展理论考虑了从底层物理方法建模到上层应用性能优化完整的高性能计算研究链，为高性能计算领域可扩展性的研究，特别是大规模并行计算中的可扩展问题，提供了系统化、体系化的方法论层面指导。

2. 提出了百核量级的 Stencil 并行数值算法，利用新型向量化和分块技术设计并实现高可扩展单机多核 Stencil 计算。在并行算法层次，提出了转置布局计算和时空计算折叠两种 Stencil 向量化策略，提高数据在 CPU 内的并行度；同时在性能优化层次，提出了高效寄存器数据重用算法和缓存分块优化算法，提高数据的访存效率。实验结果表明，通过并行算法-性能优化的双层协同设计思想，最高超过 state-of-the-art 方法的绝对性能 4.39 倍，有效地提升了 Stencil 并行数值算法的多核可扩展性能。

3. 提出了万核量级的分布式机器学习框架，利用新型聚类和回归技术设计

**并实现高可扩展的多机众核机器学习预测模型。**本文在物理模型层次，提出了新型 Best Friend Graph 图数据结构及层次化的最小生成树网络模型，并设计了基于聚类的回归预测方法；在并行算法层次，提出了基于回溯的负载均衡算法和高效的并行通信算法，降低分布式系统的计算和通信开销。实验结果表明，通过物理模型-并行算法的双层协同设计思想，在保证聚类和回归方法准确性的同时，还能有效地将分布式机器学习框架的多机众核扩展性由已有工作的 1,536 核提升至 12,288 核。

**4. 针对科学计算软件应用优化，提出了百万核量级的大规模扩展方法并设计实现一套大规模高可扩展国产核材料辐照损伤模拟的软件应用 OpenKMC。**在物理模型层次，设计优化了高可扩展的新型势函数模型和分组反应策略，支撑应用高效动力学蒙特卡罗模型建立；在并行算法层次，提出了适应于大规模应用的并行同步象限算法和高效自适应通信算法，提高应用的负载均衡和通信效率；在性能优化层次，提出了访存优化技术、高效局部性算法、Athread 线程级异构并行和从核向量化加速方法，通过主存级-缓存级-寄存器级多层次化访存特征提取优化和轻量级的进程级-线程级-数据级多层次并行性挖掘对异构众核体系结构算力进行充分利用。实验结果表明，通过物理模型-并行算法-性能优化的多层次可扩展性协同设计思想，实现神威·太湖之光千亿原子的 520 万核大规模模拟，并行效率高达 80%，成为核材料模拟新的里程碑。

**关键词：** 大规模并行；可扩展性；科学计算应用；并行数值算法；机器学习框架

## Abstract

High performance computing is an important branch of computing science, which mainly refers to the research and utilization of high-performance computer technology from the aspects of architecture, parallel algorithms and parallel software. For decades, high performance computing has been widely used in crucial fields such as large-scale scientific and engineering computing, artificial intelligence, and industrial simulation. It has extremely high application values for basic scientific discovery, national economic development and national defense industry construction, and has become an important manifestation of the country's comprehensive scientific and technological strength.

Turing Award winner Jim Gray pointed out that scalability issues are at the heart of high performance computing, and it is also an important challenge that ranks first among the twelve research goals in the future information technology field. Parallel software design and parallel system architecture are two typical aspects of scalability, and the scalability challenges are reflected in the difficulty of architecture porting, algorithm design, and application optimization. This thesis conducts research on the theory of massively parallel multi-level discontinuous nonlinear scalability, deeply analyzes the development law of scalability, and proposes physical models, parallel algorithms and performance optimization multi-level collaborative design methods. The scalability optimization design is carried out at various hardware parallel scales, different software parallel granularities, and multi-level interdisciplinary parallel applications. The specific contributions include:

**1. A massively parallel multi-level discontinuous nonlinear scalable theory is proposed.** Through the analysis of the scalability on parallel software from homogeneous multi-core systems to heterogeneous many-core systems, it is found that there are two typical phenomena of scalability, namely discontinuity and nonlinearity, in the physical model, parallel algorithm and performance optimization of applications with different parallel granularity. The two phenomena at multiple levels are systematically analyzed, and the multi-level collaborative design theory of scalability is first proposed.

It provides methodological guidance for the research of scalability in the field of high-performance computing, especially the scalability problem in massively parallel computing.

**2. A hundred-core-level Stencil parallel numerical algorithm is proposed, using novel vectorization and tiling technology to achieve highly scalable single-machine multi-core Stencil calculations.** At the parallel algorithm level, two Stencil vectorization strategies, transposed layout calculation and spacial-temporal calculation folding, are proposed to improve the parallelism of data in the CPU; at the same time, at the performance optimization level, an efficient register data reuse algorithm and tiling optimization algorithm are designed to improve data access efficiency. The experimental results show that based on the collaborative design idea of parallel algorithm and performance optimization, the absolute performance is up to 4.39 times higher than the state-of-the-art method, and the multi-core scalable performance of the Stencil parallel numerical algorithm is effectively improved.

**3. A ten-thousand-core-level distributed machine learning framework is proposed, which uses novel clustering and regression techniques to achieve a highly scalable multi-machine many-core machine learning prediction model.** At the theoretical modeling level, this thesis proposes a new Best Friend graph data structure and a hierarchical minimum spanning tree network model, and designs a regression prediction method based on clustering; at the parallel algorithm level, a backtracking-based load balancing algorithm and an efficient parallel communication algorithms are proposed to reduce the computational and communication overhead of distributed systems. The experimental results show that through the physical model and parallel algorithm collaborative design idea, it ensures the accuracy of the clustering and regression methods, and effectively improve the scalability on multi-machine and many-core distributed machine learning framework from the existing work of 1,536 cores to 12,288 cores.

**4. Aiming at the optimization of scientific computing software applications, a million-core-level large-scale scaling method is proposed, and a large-scale and highly scalable domestic software application OpenKMC is designed to simulate the radiation damage of nuclear materials.** At the physical model level, a highly

scalable new potential function model and grouped reaction strategy are optimized to support the establishment of an efficient kinetic Monte Carlo model; at the parallel algorithm level, a parallel synchronous quadrant algorithm suitable for large-scale applications and efficient adaptive communication algorithm are proposed to improve load balancing and communication efficiency; at the performance optimization level, memory access optimization technology, efficient locality algorithm, Athread thread-level heterogeneous parallel strategy and vectorized acceleration method are proposed. Hierarchical memory-cache-register data access features are extracted and lightweight process-thread-data parallelism is tapped to optimize the utilization of the computing power on heterogeneous many-core architecture. The experimental results show that, through the multi-level collaborative design idea of physical model, parallel algorithm, and performance optimization, our OpenKMC achieves high accuracy and good scalability of applying hundred-billionatom simulation on 5.2 million cores with a performance of over 80.1% parallel efficiency, which becomes a new milestone in nuclear material simulation.

**Keywords:** Massively Parallelism; Scalability; Scientific Computing Applications; Parallel Numerical Algorithms; Machine Learning Frameworks



## 目 录

<b>第 1 章 绪论 .....</b>	<b>1</b>
1.1 研究背景 .....	1
1.2 研究问题 .....	3
1.2.1 国内外相关工作 .....	3
1.2.2 并行数值算法设计 .....	8
1.2.3 机器学习框架研发 .....	9
1.2.4 科学计算应用优化 .....	11
1.3 研究内容和主要贡献 .....	13
<b>第 2 章 多层次不连续非线性可扩展理论 .....</b>	<b>17</b>
2.1 基本概念 .....	18
2.1.1 性能指标 .....	18
2.1.2 并行粒度 .....	19
2.1.3 可扩展性定律 .....	20
2.2 可扩展性的不连续和非线性现象 .....	22
2.2.1 硬件层面 .....	23
2.2.2 软件层面 .....	25
2.3 多层次协同设计方法 .....	28
2.3.1 物理模型抽象 .....	28
2.3.2 并行算法设计 .....	32
2.3.3 性能优化方法 .....	35
2.4 本章总结 .....	41
<b>第 3 章 细粒度并行：百核量级 Stencil 并行数值算法设计 .....</b>	<b>43</b>
3.1 本章概述 .....	43
3.1.1 引言 .....	44
3.1.2 背景 .....	47
3.1.3 相关工作 .....	49
3.2 并行算法设计：适应多核架构的新型 Stencil 向量化算法 .....	51
3.2.1 局部转置布局算法 .....	51
3.2.2 空间计算折叠算法 .....	54
3.2.3 时空计算折叠算法 .....	59
3.3 性能优化方法：多核架构高效访存与计算优化 .....	63

3.3.1 层次化访存优化 .....	63
3.3.2 轻量级并行 .....	69
3.4 实验评估 .....	70
3.4.1 实验配置 .....	70
3.4.2 数据准备的影响 .....	72
3.4.3 串行无分块实验 .....	73
3.4.4 并行分块实验 .....	75
3.4.5 可扩展性 .....	78
3.4.6 实验讨论 .....	81
3.5 本章总结 .....	83
<b>第 4 章 中粒度并行：万核量级分布式机器学习框架研发 .....</b>	<b>85</b>
4.1 本章概述 .....	85
4.1.1 引言 .....	86
4.1.2 背景 .....	89
4.1.3 相关工作 .....	92
4.2 物理模型抽象：基于 Best Friend Graph 图结构的层次化最小生成树网 模型 .....	93
4.2.1 Best Friend Graph 数据结构 .....	93
4.2.2 模型理论特征 .....	95
4.2.3 层次化最小生成树网模型 .....	96
4.2.4 数据组织模型 .....	99
4.3 并行算法设计：适应分布式架构的新型机器学习算法 .....	99
4.3.1 快速距离计算 .....	99
4.3.2 并行化算法 .....	100
4.3.3 基于回溯的负载均衡算法 .....	101
4.3.4 基于 Best Friend 聚类的回归预测算法 .....	102
4.4 实验评估 .....	104
4.4.1 实验配置 .....	104
4.4.2 可视化 .....	105
4.4.3 收敛性 .....	105
4.4.4 准确度 .....	106
4.4.5 扩展性 .....	107
4.4.6 实验讨论 .....	108
4.5 本章总结 .....	109

---

<b>第 5 章 粗粒度并行：百万核量级核材料辐照科学计算应用优化</b>	111
5.1 本章概述	111
5.1.1 引言	112
5.1.2 背景	114
5.1.3 相关工作	117
5.2 物理模型抽象：动力学蒙特卡罗计算模型建立	118
5.2.1 Pair 势函数模型	118
5.2.2 分组反应模型	120
5.3 并行算法设计：适应复杂体系结构的并行计算和通信算法	121
5.3.1 并行 AKMC 计算	122
5.3.2 Ghost 晶格计算	123
5.3.3 非阻塞集合通信	123
5.3.4 自适应通信算法	124
5.4 性能优化方法：神威硬件特征与算法精准抽象的深度性能优化	127
5.4.1 访存优化设计	127
5.4.2 OpenACC 自动并行	128
5.4.3 从核转录-翻译-传输算法	128
5.4.4 向量化加速	130
5.5 实验评估	131
5.5.1 数据验证	131
5.5.2 辐照损伤可视化	132
5.5.3 叠加优化性能评估	133
5.5.4 大规模可扩展性	135
5.6 本章总结	138
<b>第 6 章 总结与展望</b>	139
6.1 工作总结	139
6.2 研究展望	140
<b>参考文献</b>	143
<b>致谢</b>	157
<b>作者简历及攻读学位期间发表的学术论文与研究成果</b>	159



## 图形列表

2.1 多层次不连续非线性可扩展现象 .....	22
2.2 可扩展性不同并行粒度的分层设计抽象 .....	28
2.3 大规模并行应用的可扩展性多层次协同设计方法 .....	29
2.4 主流处理器层次化存储抽象框图 .....	36
2.5 主流处理器轻量级多层次并行（进程级-线程级-数据级）抽象框图 ..	38
3.1 并行算法-性能优化双层协同设计 .....	43
3.2 DLT 中空间数据冲突的处理 .....	47
3.3 向量长度为 4 的局部转置布局 .....	52
3.4 局部转置布局的 Stencil 计算 .....	53
3.5 二维 Stencil 向量化中的空间数据对齐冲突 .....	54
3.6 二维标量 Stencil 计算中的计算折叠策略 .....	55
3.7 2D9P 盒形 Stencil 计算折叠向量化 .....	56
3.8 二维 Stencil 的内存存储布局 .....	58
3.9 二维 9 点盒形 Stencil 初始时标量算术表达式图解（时间展开因子 $m = 2$ ） .....	60
3.10 二维 9 点盒形 Stencil 折叠后标量算术表达式图解（时间展开因子 $m = 2$ ） .....	61
3.11 二维 9 点盒形 Stencil 时空计算折叠向量化过程（时间展开因子 $m = 2$ ） .....	62
3.12 基于局部转置布局的一维 Stencil 分块框架（2 个时间步） .....	64
3.13 循环融合后 Stencil 计算（2 个时间步） .....	67
3.14 偏移重用图解 .....	68
3.15 使用 AVX2 指令的寄存器内双精度浮点数向量集转置 .....	69
3.16 无分块串行实验中 Jacobi 类型 Stencil 计算时间百分比 .....	73
3.17 串行无分块实验中基于转置布局算法的向量化与其他方法的绝对性能比较（不同总时间步长的结果分别展示在不同子图中） .....	74
3.18 串行无分块实验中基于计算折叠算法的向量化与其他方法的绝对性能比较（不同机器的结果分别展示在不同子图中） .....	75
3.19 多核缓存分块实验中不同方法的绝对性能比较。（分块大小均固定在一级或二级缓存中，结果以不同的总时间步长分别展示。） .....	76
3.20 多核 Intel 机器上使用缓存分块技术不同方法的性能对比。（每组的加速度均与各组最低的性能作为基准进行比较，该基准加速度默认值为 1，用三角形标注。） .....	77

3.21 多核 AMD 机器上使用缓存分块技术不同方法的性能对比。(每组的加速度均与各组最低的性能作为基准进行比较, 该基准加速度默认值为 1, 用三角形标注。) .....	78
3.22 基于转置布局的向量化与其他方法不同维度多阶 Stencil 的多核可扩展性 .....	79
3.23 基于计算折叠的向量化与其他方法不同维度多阶 Stencil 的多核可扩展性 (Intel 机器) .....	80
3.24 基于计算折叠的向量化与其他方法不同维度多阶 Stencil 的多核可扩展性 (AMD 机器) .....	81
 4.1 物理模型-并行算法双层协同设计 .....	85
4.2 以全球知名城市为例的 Best Friend Graph .....	94
4.3 以全球知名城市为例的 Best Friend 聚类 .....	94
4.4 以全球知名城市为例的 Best Friend Forest .....	96
4.5 Best Friend 聚类生成的分层最小生成森林 .....	98
4.6 Best Friend 聚类中的数据组织 (不同类通过颜色和线框进行区分) ..	100
4.7 基于回溯的分布式负载均衡算法。图 4.7 (a) 描述了针对小任务的 MERGE 操作; 图 4.7 (b) 展示了基于回溯的大任务的 SPLIT 操作。为了便于展示, 城市的名称相应地进行了缩写。 .....	101
4.8 不同的聚类方法在 Zahn's Compound <sup>[1]</sup> (左图), Aggregation <sup>[2]</sup> (中图), 和 R15 <sup>[3]</sup> (右图) 数据集上的可视化结果 (标有 AMI 指数) .....	105
4.9 在真实数据集上不同方法相同配置下的准确度和可扩展性。我们的并行库实现了包括核岭回归 (KRR)、线性回归 (LR) 和支持向量回归 (SVR) 在内的回归技术。它们与 Best Friend 聚类方法相结合, 分别缩写为 BFCKRR、BFCLR 和 BFCSV. 与代表性聚类方法相结合的并行回归基准算法分别是 DCKRR (无聚类的分治方法) <sup>[4]</sup> , BKRR2 (K-Means) <sup>[5]</sup> , AHCKRR (凝聚层次聚类) <sup>[6]</sup> 和 MSTKRR (最小生成树聚类) <sup>[7]</sup> 。 .....	107
4.10 Cadata 和 Gas Sensor 数据集的时间剖析 .....	108
 5.1 物理模型-并行算法-性能优化多层协同设计 .....	111
5.2 体心立方 (BCC) 晶格的三种最近邻原子分布图 .....	114
5.3 基于空位跃迁的串行原子动力学蒙特卡罗 (AKMC) 方法 .....	116
5.4 神威 SW26010 多核处理器的基本架构 .....	116
5.5 事件选择和更新的分组反应模型 (红色三角形点落入的事件被选中, 与其相邻的黑色正方形点的事件被拒绝) .....	120
5.6 神威 SW26010 架构的并行 AKMC 算法 (4 个进程将模拟域细分为 16 个象限, 每个象限进一步分配到各个从核进行计算) .....	122

5.7 神威 SW26010 架构的 Ghost 晶格计算（左侧部分展示了蓝色进程的计算所依赖的周围环境（虚框所示），其中一些数据由其他进程持有。右侧部分展示了该进程将在 3 次通信后从其他进程（分别对应绿色、紫色、黄色进程）发送和接收的数据，以完成其第一象限的计算） ·	123
5.8 实现的 MPI 通信模式与传统通信模式的比较（左子图的是一对一交换通信算法 (PEX)，右子图是非阻塞集合通信 (NBX)） ······	124
5.9 DMA 中数据结构重构以使访问连续并对齐 ······	127
5.10 用于从核加速计算的转录-翻译-传输算法（绿色、紫色和黄色的核心分别是模板核、信使核和计算核。每个信使核将与同一行中的 7 个计算核配对，它们将使用快速行寄存器通信共享数据） ······	129
5.11 内层循环 $j$ - 粒子计算的向量化 ······	130
5.12 Fe-1.34 at.% Cu 合金的热老化析出演变（绿色方块对应 Lé 等人的真实物理实验结果 <sup>[8]</sup> ；黑色方块对应 Vincent 等人的动力学蒙特卡罗模拟结果 <sup>[9]</sup> ；其他曲线则为在不同架构上的 EAM 势或 Pair 势的 OpenKMC 模拟结果。括号中的“Comm”表示已进行通信优化的 OpenKMC 结果） ······	131
5.13 FeCu 合金热老化前后 Cu 簇析出可视化 ······	133
5.14 神威 SW26010 处理器上 OpenKMC 多种方法的叠加优化性能图（子图 (a) 和 (b) 为高浓度实验，即空位浓度为 12.8%，Cu 浓度为 12.0%；子图 (c) 和 (d) 为低浓度实验，即空位浓度为 $8 \times 10^{-4}$ %，Cu 浓度为 1.34%。子图 (a) 和 (c) 展示了物理方法优化的性能提升；子图 (b) 和 (d) 展示了计算技术优化的性能提升） ······	134
5.15 不同空位浓度下的性能变化（红线和蓝线分别代表 OpenKMC 模拟中使用和不使用众核优化（3T 算法和矢量化）的运行时间。括号中的数字表示分配给每个从核的空位数量。性能提升比使用黑色上三角进行标记） ······	134
5.16 OpenKMC 模拟 540 亿 ( $5.4 \times 10^{10}$ ) 原子的强可扩展性（ $x$ 和 $y$ 轴相应进行缩放；绿色直方条对应左侧主 $y$ 轴的模拟时间；黑色三角形散点对应右侧次 $y$ 轴的并行效率；模拟时间和并行效率值分别标注在直方条和三角形散点的顶部） ······	135
5.17 OpenKMC 模拟每个进程 1100 万个原子 ( $1.1 \times 10^7$ ) 的弱可扩展性（ $x$ 和 $y$ 轴相应进行缩放；绿色直方条对应左侧主 $y$ 轴的模拟时间；黑色三角形散点对应右侧次 $y$ 轴的并行效率；模拟时间和并行效率值分别标注在直方条和三角形散点的顶部） ······	136
5.18 不同空位浓度下的并行效率对比（括号中的数字表示平均分配给每个从核的空位数量） ······	137



## 表格列表

1.1 近年来“戈登贝尔奖”主要获奖情况概览 *	2
2.1 大规模可扩展性的多层次协同设计理论概览	17
2.2 表 2.1 中大规模可扩展性的多层次协同设计理论符号定义	18
3.1 Stencil 计算折叠算法中相关符号定义	54
3.2 求解 Jacobi 类型 Stencil 不同向量化方法寄存器行为统计（每向量）	59
3.3 实验中 Stencil 参数配置	70
3.4 不同 Stencil 优化工作所采用的向量化、缓存分块及并行化技术	71
3.5 串行无分块实验中基于转置布局算法的向量化与其他方法在不同存储层级上的性能提升对比	74
3.6 串行无分块实验中基于计算折叠算法的向量化与其他方法在不同存储层级上的性能提升对比	76
3.7 多核缓存分块实验中基于转置布局算法的向量化与其他方法在不同存储层级上的性能提升对比	77
3.8 可扩展性实验中内存带宽 (BW) 使用配额	79
4.1 经典形状数据集参数配置	104
4.2 真实数据集参数配置	105
4.3 Best Friend 聚类中的分层 <i>HCI</i> 指数	106
4.4 Best Friend 聚类中的分层聚类个数	106
4.5 Cadata 和 Gas Sensor 数据集上并行框架中不同操作的时间占比和加速比	108
5.1 本章工作及参考文献 <sup>[9]</sup> 中使用的 Pair 势键能 (eV)	120



# 第1章 绪论

## 1.1 研究背景

计算科学是从上世纪中叶开始伴随着计算机的出现而迅速发展并获得广泛应用的新兴交叉科学。到 80 年代后期，计算科学更被科学家们认为是继理论研究和实验科学后科学研究的第三范式<sup>[10]</sup>。在我国国民经济和国家安全的重大应用中，例如核材料科学、大气与生态环境、航空航天等各个科学领域，大规模计算科学都成为现阶段唯一的科学手段，对探索科学理论、促进应用突破、保障国家安全等方面都有深远的影响，起到了无法替代的基础性作用。在大规模科学与工程计算领域的应用中，性能是在实际研发中最受关注的重点问题，这一核心需求也是计算机并行软硬件平台发展的主要动力之一。

从近几十年高性能计算机体系结构的发展来看<sup>[11]</sup>，底层硬件特别是计算单元的多样性日渐丰富。从最早的向量处理机到多核处理器，再到如今的异构众核架构，多样性的底层体系架构极大地提高了硬件机器的性能，同时也是保持摩尔定律的重要方式。作为全球高性能计算机性能的风向标，TOP500 排行榜中超级计算机的发展也清晰地展示了这种趋势<sup>[12,13]</sup>。同样地，对于并行软件而言，其研发和设计的路线也随着底层硬件体系架构的更新而完善。设立于 1987 年的“戈登贝尔奖”（Gordon Bell Prize）是国际上高性能计算应用领域的最高学术奖项，其与 TOP500 排行榜均在每年的国际超算大会上进行颁发与发布，也被称为“超算界的诺贝尔奖”。表 1.1 中展示了近几年来“戈登贝尔奖”的主要获奖情况。目前，最新的“戈登贝尔奖”（2021 年）获奖应用 SWQSIM 在并行度上已经达到了新一代神威超算系统千万核的并行可扩展，单精度性能达到 1.2Eflops<sup>[14,15]</sup>。即便如此，这些高性能计算机的计算能力也仍旧不能满足复杂度更高的科学计算等大规模并行应用的需求。因此，很多科学家往往不得不继续采取减小体系、降低精度、简化模型等手段以削减计算量、保证扩展性。而未来超级计算机的“下一顶皇冠”将是 E 级超算，即每秒可进行百亿亿次运算的超级计算机。要实现这一目标，超算系统的可扩展性会成为最具挑战的重点问题之一<sup>[16]</sup>。因此，对当前百万核量级，甚至是更高量级的大规模并行算法和软件可扩展性问题的研究，仍然是我国及世界科技发展面临的一个亟待解决的前沿热点挑战性问题。

可扩展性问题是高性能计算领域的核心问题，即研究如何在增加资源的情况下继续保证大规模应用的性能。著名的图灵奖得主 Jim Gray 提出了未来信息技术领域需要重点解决的 12 个长远问题。其中，可扩展性问题排在第一位。可扩展性问题将会衍生出大型计算机系统包括存储、通信、算法等各种不同方面的子问题，是计算机系统结构方向中的一项关键而长期的研究目标。并行软件设计和并行系统结构是可扩展性一软一硬的两个方面，两者相辅相成，也是对立统一的。并行软件设计的高可扩展性意味着在更新底层硬件资源时能够获得更多“免费的午餐”，即在不需要更改应用的情况下性能也将保持可持续性的增长；而并行系统结构的高可扩展性则能够对更多的并行软件设计提供高性能的硬件支撑，以支持处理不同并行粒度的不同应用。

本文对高性能计算中软硬件的发展和以大规模并行软件优化为代表的各类典型应用中的痛点问题进行了调研与分析，总结了广泛存在于高性能计算（特别是大规模并行计算）可扩展问题中的一种现象，即多层次不连续非线性可扩展现象（Multi-level Discontinuous and Nonlinear Scalability, MDNS）<sup>[11,17]</sup>。可扩展性的不连续现象，指的是随着并行计算问题在不同属性更换或在不同硬件架构迁移，性能出现了不同性质的不可连续扩展，即必须更换新物理模型，新并行算法

**表 1.1 近年来“戈登贝尔奖”主要获奖情况概览 \***

时间	主要完成机构	超算机器	系统规模	方法应用	性能
2021	之江实验室；清华大学；无锡超算等	新一代神威超算	107,520 节点 (41,932,800 核心)	随机量子电路	4.4 EFLOPS (M**) 1.2 EFLOPS (S)
2020	UC Berkeley；中科院北京九所；北京大学等	Summit	4,560 节点 (27,360 GPUs)	深度势能分子动力学	162 PFLOPS (M) 91 PFLOPS (D)
2019	ETH Zurich	Summit	4,560 节点 (27,360 GPUs)	耗散量子输运模拟	90.89 PFLOPS (M) 85.45 PFLOPS (D)
2017	清华大学；山东大学；无锡超算等	神威太湖之光	10,400,000 核心	非线性地震模拟	18.9 PFLOPS (S)
2016	清华大学；中国科学院大学；无锡超算等	神威太湖之光	10,400,000 核心	非流体大气动力学	7.95 PFLOPS (D)

\* 数据来源：<https://awards.acm.org/bell/award-winners>。

\*\* 为使表达更清楚，单精度、双精度、混合精度分别简记为 S, D, 和 M。

设计或新并行软件实现方法以保证性能的现象。可扩展性的非线性现象，则意味着即使在某种物理模型、并行算法设计和并行软件实现等最适组合的解决方案范围内，性能也并不能随着问题规模或计算资源的增加而线性提高的现象。

## 1.2 研究问题

### 1.2.1 国内外相关工作

近年来，对大规模并行计算应用可扩展性的相关研究是高性能计算领域研究热点之一<sup>[17,18]</sup>。相关的研究工作主要分为两类：一类是侧重于理论层面的方法指导，典型的工作如国内国防科学技术大学的陈军、北京应用物理与计算数学研究所的莫则尧、袁国兴等人提出的近优可扩展性理论<sup>[19]</sup>，中国科学技术大学的陈国良、孙广中等人提出的分层并行计算模型<sup>[20,21]</sup>，以及中国科学院计算技术研究所的张云泉、冯晓兵，北京大学的陈一峯等人提出的两层协同设计方法<sup>[11]</sup>；另一类则是强调应用层面的优化策略，典型的工作如 IBM 研究院的 Lixiang Luo、美国橡树岭国家实验室的 Tjerk P. Straatsma、Dmytro Bykov 等人的 Summit 超级计算机上大规模并行计算应用的研发与优化研究<sup>[22]</sup>，以及国家并行计算机工程技术研究中心的刘鑫、陈左宁等人的神威·太湖之光计算机系统大规模应用特征分析与 E 级可扩展性的研究<sup>[23]</sup>。

国防科学技术大学的陈军，北京应用物理与计算数学研究所的莫则尧、袁国兴等人进行了大规模并行应用程序的可扩展性研究<sup>[18,19]</sup>。该项工作提出了数值可扩展性和并行可扩展性两个概念，分别用于定性地描述大规模并行应用在扩展时数值性能和并行性能的变化情况。在分析数值可扩展性和并行可扩展性的同时，该工作定义了计算比、计算含量比、数值效率比以及并行效率比等四项性能指标，以定量地对应用的扩展性能进行衡量。基于提出的可扩展性评价准则，该工作以二维等离子体粒子云网格法应用程序的可扩展性分析为例，对其进行了大规模可扩展性的详细分析。根据其近优可扩展性理论，提高该应用可扩展性的唯一的办法是改进并行算法，即进行分割网格计算并取适当的动态负载平衡方法来保证计算效率。经过优化后的程序完成了 32 节点上 6,389,760 个粒子的大规模模拟，并行效率达到了 43.03%。该项工作提出的近优可扩展性分析理论可以有效帮助开发者探究程序可扩展性差的原因，并以一个实际的大规模、粗粒度应用程序作为例证。然而，该项工作着重关注于单一的并行效率层面的算法分

析，并没有系统地建立起完整的高性能计算应用优化分析链。另外，该项工作的例证仅采用了单一的大规模粗粒度二维等离子体粒子云网格法应用程序，未对其他并行粒度的应用进行适应性探究，且该应用实际的并行度仅达到了 32 节点的百万粒子模拟，已经远远达不到现如今大规模计算所对应的粗粒度标准。

中国科学技术大学的陈国良、孙广中等人根据并行计算模型的发展和演变趋势，首次提出了分层并行计算模型<sup>[20]</sup>。该项工作指出，在并行计算机上可以构建并行算法设计、并行程序设计以及并行程序执行三个层次的模型来考虑程序的并行优化工作。其中，算法设计模型重点关注算法的设计原理，确保算法运行的正确性和较低的时间、空间复杂度。算法设计模型面向算法设计者，也是三层计算模型中最基础的部分，其他两层模型的设计需要根据算法设计模型而进行相应地研究与修改<sup>[21]</sup>。并行程序设计模型面向编程者，强调并行计算的正确编程实现，确保算法正确语义。如大粒度进程级并行采用消息传递接口（Message Passing Interface，MPI），细粒度线程级并行采用 OpenMP、Pthread 等。并行程序执行模型面向程序运行者，侧重考虑应用程序和运行时软件环境相关的问题，如存储和通信、线程调度、并行性级别划分等。虽然该工作通过理论分析表明，通过各阶段分工明确的分层计算模型，可以有效地为并行计算提供优化上的分析和指导，但是缺少实际的算例支撑和证明，也无法对分层并行计算模型的实际指导意义进行验证。分层并行计算模型认为，目前并行算法设计模型和并行程序设计模型比较成熟，应集中精力重点研究并行程序执行模型。然而，随着计算机体系结构和应用场景的日益复杂和多样化，实现算法和程序在不同架构处理器和不同应用场景间的性能优化和移植已经成为一个持续性的挑战。另外，该工作更多地关注于普通量级的并行计算优化，并未进行更高一层的大规模可扩展层面的分析和指导，而当前国内外处理器已经进入快速发展轨道，众核和异构化的大规模可扩展并行计算已经成为高性能计算发展的重要趋势。

中国科学院计算技术研究所的张云泉、冯晓兵，北京大学的陈一峯等人发现并总结了高性能计算几十年发展历史中在多个层次上出现的不连续和非线性两种可扩展性现象，并提出了两层协同设计方法<sup>[11]</sup>。该项工作特别强调了物理模型和数值算法层次上可扩展性研究的重要性，在已有的“算法-体系结构”协同设计方法的基础上，向上一层提出了“应用-算法”协同设计思路，形成了两层协同设计研究手段。为了清楚地阐释在大规模并行程序中所出现的不连续和非

线性现象，作者以全球气候模拟中的大气模式以及并行编程模型的设计研究为例进行了算法层面的数据分析。其中，该项工作重点介绍了在大气模式中使用高纬度自适应滤波算法以完美支持三维剖分的底层算法设计的思路，从算法设计的角度上详细论证了支持百万核量级并行的可能性，从而证明了“算法-体系结构”与“应用-算法”两层协同设计的有效性。该项工作首次发现并总结了高性能计算的不连续、非线性的可扩展现象，然而对该现象的总结缺乏系统地理论推导与实例支撑，其给出的大气模式应用实例更多地是从单一的滤波算法角度进行可扩展性的推演，并未从大规模应用优化的通信、计算等主要开销上完成全局的分析与例证。另外，其形成的“算法-体系结构”与“应用-算法”两层协同设计方法无形中将体系结构与应用优化隔离开，而随着超级计算机体系结构的复杂化、多样化，实现算法和程序在不同架构处理器上的性能优化和移植也是在可扩展性问题中需要完善考虑的一个重要挑战。

作为全球高性能计算机性能的风向标，TOP500<sup>[13]</sup> 排行榜中超级计算机的发展趋势表明，美国在最快超算上多次领跑，中国也在超算数量上继续领先，而且性能也在持续提升，全球超算格局逐渐形成中美两强“你超我赶”的形势。而作为“超算界的诺贝尔奖”，“戈登贝尔奖”近几年来的获奖应用也在中国神威·太湖之光超级计算机和美国 Summit 超级计算机之间来回易主。因此，在这两台全球顶尖超级计算机上可扩展性的研究也是值得关注的热点。

IBM 研究院的 Lixiang Luo，橡树岭国家实验室的 Tjerk P. Straatsma、Dmytro Bykov 等人进行了 Summit 超级计算机上大规模并行计算应用的研发与优化研究<sup>[22]</sup>。该项工作总结了将应用扩展到 Summit 时所需的移植经验，同时给出了一部分进行大规模扩展的应用移植实例来例证相关研究。该工作指出，针对 Summit 超级计算机的大规模并行计算应用扩展，需要从 12 个层面进行优化考虑：通过程序运行时间的测量建立性能评估模型，建立缓存一致性的统一内存访存模型，GPU 原子操作优化，GPU 胖节点通信优化，基于高 GPU-CPU 比率硬件架构特点的负载优化，线性代数相关计算的 GPU 加速，较大物理模型负载的分批切割，基于网格切分的算法优化，基于粒子模拟的算法建模，软件开发流程优化以及项目管理优化。同时，该项工作选取了 DIRAC (Direct iterative relativistic all-electron calculations, 直接迭代相对论全电子计算)、NAMD (Nanoscale Molecular Dynamics, 并行分子动力学软件包)、QMCPACK (Quantum Monte Carlo Simulation

Package, 量子蒙特卡罗模拟软件包) 等经典应用在 1/5 至全机规模的 Summit 超级计算机上进行了扩展性和加速比分析。该项工作通过大量实际的大规模应用开发优化经验, 较详细地分析了在 Summit 超级计算机上大规模扩展面对的各种问题和性能瓶颈, 并总结了应用高效移植和大规模扩展的方法技巧, 为后续各种应用扩展性能的提升提供了实用性的借鉴和指导。然而, 该项工作过多地侧重于与硬件结合的扩展性能优化和提升, 且多个优化层面是与 GPU 架构相结合的分析指导, 并未进行通用性的模型抽象与方法建模, 不具有高性能计算大规模应用扩展的通用性。另外, 该工作总结的 12 层优化方法相对比较分散, 且大多是以优化点的形式提出, 未能够系统地形成研究体系和方法论层面的指导, 给后续大规模扩展性研究带来较为困难的理解与应用。

国家并行计算机工程技术研究中心的刘鑫、陈左宁等人进行了神威·太湖之光计算机系统大规模应用特征分析与 E 级可扩展性的研究<sup>[23]</sup>。该项工作针对计算密集的大规模并行应用进行了计算特征和数据迁移行为的详细分析, 从编程环境和体系结构两大应用需求入手, 重点关注了算法特点、体系结构适应性、算法时间和空间复杂度、访存特点、通信复杂度等特征, 提出了下一代 E 级高性能计算机系统上大规模并行计算应用可扩展性的提升方法。该可扩展性提升方法指出, 大规模并行计算应用的可扩展性主要可以通过四类方法进行优化, 即计算方法优化、访存优化、计算优化和通信优化。基于其提出的可扩展性能模型, 该工作以美国加州大学伯克利分校总结的十三类高性能计算典型问题中的各个经典应用作为算例<sup>[24]</sup>, 在神威·太湖之光超级计算机上进行了可扩展性的研究。其中包括: 代表稠密线性代数问题的 LINPACK 应用、代表稀疏线性代数问题的 HPCG (High Performance Conjugate Gradients) 应用、代表谱方法问题的 FFT (Fast Fourier Transform, 快速傅里叶计算) 应用、代表结构化网格问题的 Stencil 计算应用、代表非结构化网格问题的计算流体力学求解应用、代表多体问题的宇宙演化模拟应用、代表 MapReduce 问题的托卡马克装置等离子体几何算法粒子模拟应用、代表图遍历问题的 Graph500 宽度优先搜索 (Breadth-First Search, BFS) 应用、代表动态规划问题的生物序列比对应用以及代表图模型问题的卷积神经网络应用。研究指出, 这十种不同应用的计算量、访存开销和通信开销是否随着问题规模的增加而线性增长, 成为各应用能否扩展到 E 级的关键问题, 也是影响应用整体并行效率和可扩展性的重要因素。该项工作选取了不同类型问

题的典型应用进行了可扩展性分析，具有相当高的代表性，且各个应用均达到了神威·太湖之光超算系统半机以上的大规模并行<sup>[15]</sup>。但目前的实验分析并未说明应用结果的准确性或收敛性是否会随着应用规模的扩展而发生变化，也并未对机器学习类的问题展开深入研究与讨论<sup>[23]</sup>。另外，该项工作更多地侧重于在神威·太湖之光超级计算机上结合神威处理器异构众核体系结构特点而提出可扩展性分析的性能模型。其多项性能分析与优化技术，例如单核组分析、片上带宽优化等均不具有普适性，未能够在更高的层面进行通用的可扩展性分析与模型抽象。

如上所述，目前大多数对于可扩展性的研究工作更多地只关注在单一层次上的问题。例如，大量算法可扩展性的研究虽然提出了一系列的优化技术，对核心函数达到了一定地加速效果和可扩展性，但仅仅依赖单一的算法优化和设计并不能满足整个应用程序对性能和可扩展性的需求。并且，在实际研发中，对各个单一层次上问题的优化，常常不能通过简单地堆叠组合而达到层次间的完美匹配，甚至可能出现性能退化，从而无法满足整个系统的性能和可扩展性的需求。再者，相关的可扩展分析模型大多停留在理论指导层面，缺少典型的实际应用进行例证或者例证应用的规模不够典型。另外，最近的相关工作则更多地关注在某一特定硬件架构的超级计算机上进行大规模应用扩展，如前文所述在Summit和神威·太湖之光上的工作。这些工作多针对特定的超算系统进行应用相关的分析研究，未能在更高层次进行系统地抽象出大规模并行计算应用扩展所面临的共性问题和挑战并形成方法论层面的借鉴和指导。

缓解或解决可扩展性的不连续非线性现象，需要重点突破算法、框架、应用在有限区间的可扩展性，在多个层次建立正确的理论解释并提出有效的联动解决方案。基于高性能计算中的不连续非线性可扩展现象<sup>[11]</sup>，本文首次系统地对多个层次上的两种现象进行了深入的理论分析与内涵丰富，并提出了细化的多层次研究方法。面对可扩展性问题，我们从设计初始就分别从物理模型、并行算法、性能优化三个层次对扩展性问题进行分析与研究。其中物理模型层次主要从软件应用的角度出发，考虑高扩展性的物理模型及数值计算方法；并行算法主要是从核心函数的角度出发，考虑高性能的算法设计及模块实现；性能优化主要是从硬件架构的角度出发，考虑硬件平台的高可用性能优化和提升算法。这一思路考虑了从物理模型到最后并行程序的完整高性能计算研究链，然后再利用多层

协同设计方法以缓解或解决可扩展性的不连续非线性现象，最终形成了系统化、体系化的大规模并行多层次不连续非线性可扩展理论。

本文选取了百核量级并行数值算法设计、万核量级分布式机器学习框架研发以及百万核量级科学计算应用优化三个涵盖不同并行粒度的典型可扩展性问题，从多个层次应用相应的协同设计方法，直面可扩展性的不连续非线性现象。对于细粒度并行问题，我们采用并行算法-性能优化双层协同设计，针对具体的底层架构特点耦合设计并行算法，更多地追求单机多核算法的“极致”性能，充分提升算法在特定架构下的扩展性；对于中粒度并行问题，我们采用物理模型-并行算法双层协同设计，针对具体的上层应用需求耦合设计并行算法，更多地追求分布式同构众核框架的“通用”性能，充分提升框架在各种应用中的扩展性；对于粗粒度并行问题，我们采用物理模型-并行算法-性能优化三层协同设计，更多地追求大规模异构众核应用的“组合”性能，充分提升应用在多种优化下的大规模扩展性。

### 1.2.2 并行数值算法设计

偏微分方程 (Partial Differential Equation, PDE) 求解器在热扩散、电磁学和流体动力学等不同领域构成大部分科学计算应用的核心计算引擎<sup>[11]</sup>。其通常会迭代地对空间网格中的格点及其最近邻进行有限差分计算，这种计算模式被称为 Stencil 计算。Stencil 是在科学计算和工程应用中广泛使用的最重要的算法之一。从物理模拟到机器学习的各个领域，均普遍涉及到 Stencil 算法的相关计算<sup>[25,26]</sup>。Stencil 算法也是 Berkeley View 中提出的高性能计算领域七大经典算法之一<sup>[24,27,28]</sup>，其作为高性能计算中浮点运算的主要算法而被大量采用为各种计算内核。

对并行数值算法 Stencil 的向量化一直是高性能计算领域研究的重点，而 Stencil 向量化中限制可扩展性能的瓶颈就是数据空间冲突问题。Stencil 计算中的数据空间冲突是由在现代高性能处理器架构下进行向量化加速而自然引起的问题，即一个网格格点的所有邻居出现在同一个向量寄存器中不同的位置上。由于数据空间冲突是特定算法在特定体系架构下所产生的问题，因此，必须通过对算法和架构层的耦合设计以处理可扩展性中的不连续非线性现象。

在并行数值算法设计的细粒度并行问题中，我们即以单机多核 Stencil 向量化算法的可扩展性研究为例，分析在计算问题和系统结构的可扩展性上各自面

对的不连续非线性现象。

从问题本身的角度来看，可扩展性的不连续性主要集中在不同方法之间的组合优化问题。传统 Stencil 的优化方法主要集中在向量化或缓存分块技术上，旨在分别提高数据并行性和数据局部性。这两种方法通常被视为在不同层次上的两种正交的优化方法。向量化利用 CPU 中的 SIMD (Single Instruction Multiple Data, 单指令多数据) 功能并行执行多个数据处理流，而缓存分块则增加对缓存中一组数据的重用性。但是，向量化的数据组织可能会破坏缓存分块技术的数据局部性；反之，缓存分块技术的分块形状和大小也会降低边界处向量化的效果。因此，不同方法所带来的优化不连续性需进行联动分析与耦合设计。可扩展性的非线性则更多地是考虑向量化本身所面临的主要问题，即数据空间冲突问题。不同的 Stencil 向量化算法设计会引入不同的访存规则和寄存器重排操作，而如何更好地在多核处理器上实现近线性的扩展性，则需要新型的算法设计以解决数据的空间冲突问题。

从系统结构的角度来看，提高性能的关键是充分利用算术运算指令的执行单元。并行数值算法受益于提供更高吞吐量数据移动的技术，这对于算法的可扩展性尤其重要。这些高级处理要求通过 SIMD 指令集进行优化和加速，而 SIMD 是现代 CPU 设计中的一项重要技术。在现代高性能处理器架构中，多数 CPU 均能提供对浮点计算的向量化支持，其可通过对向量运算进行有效地并行处理来提高性能。因此，硬件在底层设计的不连续性，给 Stencil 并行数值算法的多倍性能提升提供了有效支持。另外，随着进程数量的增加，如何在单机多核上有效地利用好多个核心的计算资源，实现 Stencil 数值算法的高效并行化，对于提升其在多核系统上的线性扩展也是非常重要的。

### 1.2.3 机器学习框架研发

作为信息技术领域的重要方向之一，人工智能 (Artificial Intelligence, AI) 及其应用已经渗透到人们生活的方方面面，成为现代科技的重要组成部分。随着人工智能技术日益成熟，其将瞄准更多不同领域的棘手问题并与不同技术进行融合，涌现出提高生产力的大量新型应用。近年来，高性能计算 (High Performance Computing, HPC) 正在加速这一科技变革。通过将人工智能的强大功能应用于现有的高性能计算任务负载 (AI for HPC)，以及充分利用高性能计算系统的计算资源以更好地对人工智能应用进行性能扩展 (HPC for AI)，人工智能和高性能计算

的技术融合正推动着科学、算法、软件和硬件实现飞跃式的发展，展示出非常光明的前景<sup>[29,30]</sup>。

无论是人工智能应用于高性能计算（AI for HPC）还是高性能计算应用于人工智能（HPC for AI），可扩展性都是两项技术融合的关键。只有满足在系统上高可扩展性这一要求，科学家们才能解决所面临的大数据和大计算的挑战，并从现有丰富的场景数据或科学应用中获益。目前，国内现有的可扩展性相关研究并未对机器学习类问题展开深入讨论<sup>[19,20,23]</sup>，而机器学习类问题又与经典的高性能数值算法、扩展的大规模科学计算应用紧密相关，在算法层面和应用层面起到了融合相承的重要作用。例如，机器学习中的卷积核作为一种重要的并行计算模式，在高性能计算中，其通常被称为 Stencil 计算；而 2020 年的“戈登贝尔奖”更是结合了物理建模、机器学习和高性能计算的相关方法，将具有从头算精度的分子动力学模拟的极限提升至了 1 亿个原子规模，开辟了新的计算范式。因此，在中粒度并行问题中，本文选取了分布式机器学习框架这一跨方向的应用，以开展对这类问题可扩展性研究的深入探索。

机器学习是指机器根据数据的属性特征等进行自我解析、自我学习的过程，也是人工智能的核心技术。在机器学习领域，聚类和回归是两种最基本和最关键的技术，它们也被分别认为是最具代表性的无监督和有监督学习方法<sup>[31,32]</sup>。因此，在分布式机器学习框架研发的中粒度并行问题中，我们即以多机分布式聚类和回归预测的研究为例，分析其在分布式系统的扩展性上所面临的多层次不连续非线性现象。

聚类和回归分别分属两种不同的机器学习方法。近年来，基于聚类分类的回归预测技术逐渐兴起，即先通过聚类训练出不同的模型，再通过不同的模型对测试集进行回归预测。从问题本身分析，这两种方法在技术上是不连续的。如何设计出聚类和回归更好的算法，并能够使二者结合时发挥出全局最佳的扩展性，是近期工作的一项热点研究。而将二者分开考虑，则每种方法在分布式系统上的扩展也都各自面临着性能的非线性增长问题。以最近的相关工作 Balanced KRR v2 (BKRR2) 中所使用的 K-Means 聚类为例<sup>[5]</sup>，由于  $K$  值是在 K-Means 中指定聚类数量的超参数，其训练时间会随着进程数的增加而大量增长，导致其在分布式系统上整体的扩展性较差。

再通过系统架构的角度来看，分布式机器学习框架的研发必然需要将设计

的聚类和回归方法应用到众核系统上。不同分布式系统上可用的百核、千核甚至是万核的计算资源都能够为框架提供不同程度的硬件加速。硬件平台的不连续性导致框架的设计必须将在不同系统的可扩展性进行深入分析和考虑，以保证分布式框架在面对系统的不连续性时仍能“通用”，保证高可扩展性。而随着硬件资源的增加，大多数机器学习算法在分布式系统上各进程之间的负载平衡与精度保障的统筹往往成为扩展时性能非线性增长的主要瓶颈。高精度的模型通常需要更多的数据置于本地进行训练，然而此时系统的整体负载则会逐渐倾斜失衡；分布式系统更好的负载平衡通常需要对模型大小进行均等划分，然而增加的数据通信则会造成性能的非线性增长，反之模型的精度则得不到保障。

#### 1.2.4 科学计算应用优化

材料科学是基础科学的重要分支，也是世界各国科技发展长期关注的重要领域。材料科学的发展，对国民经济、国防军工以及其他高新技术行业的发展和应用起着不可替代的关键性作用。传统上材料科学的发展依赖于理论研究和物理实验相结合，复杂的理论分析求解与耗时的实验操作验证增加了大量的人力、物力与时间成本，极大地阻碍了材料科学的发展。因此，以数值模拟方法为核心的计算物理学、计算化学等多领域应用的交叉学科逐渐兴起<sup>[33]</sup>。其特点是基于一定的物理化学原理，利用计算机甚至超算机器提供的强大算力，对特定的研究体系进行数值模拟计算，以研究材料的微观演化机制，从而在理论科学和物理实验之间建立起可靠的联系纽带。

动力学蒙特卡罗方法（Kinetic Monte Carlo, KMC）和分子动力学模拟方法（Molecular Dynamics, MD）是用于材料科学研究的两种最主要的数值模拟方法<sup>[34]</sup>。KMC 方法通过利用随机过程理论，实现对材料中微观尺度整个体系演化情况的模拟。其摈弃了与体系穿越势垒无关的微小的原子级别的振动，只着眼于体系的全局组态变化，因此，KMC 方法不能描绘原子的运动轨迹，其更关注的是体系的组态跃迁。通过 KMC 方法，可以实现在秒级时间尺度上对整个体系的模拟。MD 方法则是利用经典的牛顿力学方程，对材料中粒子的运动轨迹进行数值求解模拟。与 KMC 方法不同，MD 方法利用确定性的物理方程，能够更准确地记录和描述体系中各粒子的微观演化轨迹。相应地，由于大量复杂的计算，导致 MD 方法通常难以对材料体系进行长时间大规模的模拟<sup>[35]</sup>。

核能是核反应释放的能源，也是能源供应、保证国家安全的重要支柱之一。

然而，核反应堆组件的负载逐渐使它们产生损伤并导致其功能和效率的下降。核反应堆压力容器（Reactor Pressure Vessel, RPV）退化的主要原因可归于在中子辐照下 RPV 钢中富铜沉淀物的形成。这种杂质溶质簇的形成过程可以用动力学蒙特卡罗（KMC）方法在原子水平上进行模拟<sup>[36,37]</sup>。原子动力学蒙特卡罗方法（Atomic kinetic Monte Carlo, AKMC）是主导微观系统演化的一个精确模拟核材料辐照损伤的解决方案。基于空位跃迁的 AKMC 方法是以空位作为跃迁对象，研究因其跃迁而引起的金属溶质析出过程。

在科学计算应用优化的粗粒度并行问题中，我们以国产核材料辐照损伤大规模应用程序的可扩展性研究为例，通过分析在计算问题和系统结构的可扩展性上各自面对的不连续非线性现象，着重介绍了物理模型-并行算法-性能优化多层协同设计方法的有效性。具体而言，从计算问题本身的角度分析，可扩展问题的不连续性主要是不同势函数物理模型的适用性问题。传统上用于小体系模拟的 EAM 势能够很好地描述金属原子之间的相互作用，是描述金属体系最常用的一种势函数模型<sup>[38,39]</sup>。然而，从 *ab initio* 计算法产生的 EAM 势函数模型涉及到大量复杂的计算，且其不支持大规模复杂的 FeCuNiMnSi 合金体系和实际核工程模拟所需的间隙体模拟。因此，在面对大规模的体系模拟时，势必需要更换新的势函数物理模型。可扩展问题的非线性则主要是现有模拟中使用的数值计算算法不能很好地支持体系的线性扩展。对于  $N$  个事件的模拟体系，经典的 Gillespie 随机算法在每一个时间步的时间复杂度为  $O(N)$ ，显然不能满足在大规模模拟时对模拟事件计算速度的需求，阻碍了扩展性的进一步提升。我们再从大规模应用下的系统结构角度分析。国产神威系统的主从核架构理论上可以为单进程提供跨越式的 64 倍从核线程加速。同时，从核上丰富的 SIMD 架构扩展指令则可以为计算进一步提供至少 4 倍的向量化加速。因此，硬件不连续的多层级架构也造成了可扩展问题的不连续性。同时，随着硬件资源的增加，如不对通信模式进行相应地优化升级，大规模进程间的通信开销会成为非线性可扩展的瓶颈。

通过对大规模核材料辐照损伤计算应用的扩展性分析可见，不同层次的优化方法具有其独特的优缺点。在应用、算法和架构环境变化的情况下，优势和劣势往往能互相转化，需要考虑不同层级优化方法的协同设计，扬长避短，以发挥最优的可扩展性。

### 1.3 研究内容和主要贡献

高性能计算是计算科学的重要分支和具体实践。数十年来，高性能计算一直是学术研究和创新应用的关键技术，被广泛地应用在科学计算、工业仿真等重要领域，对国民经济发展和国防建设具有极高的应用价值，也是一个国家科技综合实力的集中体现。从数十年间对大规模并行软件在同构多核到异构众核系统上的研发历史来分析，我们发现在大规模并行应用软件的开发中在物理模型、并行算法以及性能优化等多个层次上存在可扩展性的两种有趣的现象，即不连续性和非线性。

本文从科学计算应用优化、并行数值算法设计和分布式机器学习框架研发这三种涵盖不同技术方向、不同并行粒度的层级出发，系统地总结了在高性能计算中面对不同粒度的并行需求时所遇到的可扩展性的不连续性和非线性现象。其中，细粒度并行问题主要针对的是高性能计算方向的算法可扩展，即高性能计算中经典的 Stencil 并行数值算法向量化的单机多核可扩展性；中粒度并行问题主要关注的是跨方向的框架可扩展，即高性能计算和人工智能两种方向的融合技术中基于聚类的回归预测框架同构众核的可扩展性；而粗粒度并行问题则主要研究的是跨学科的应用可扩展，即科学计算中核材料辐照模拟的大规模异构众核的可扩展性。

为了解决这两种现象所带来的挑战，我们提出了在多个层次上研究可扩展性的必要性。本文的主要贡献是通过在高性能计算中软硬件的发展以及不同并行粒度的问题中扩展性痛点的调研与分析，对大规模并行多层次不连续非线性可扩展理论进行了研究及应用，强调了多层级上可扩展性研究的重要性，并提出了物理模型、并行算法以及性能优化多层次协同设计方法，以缓解或解决可扩展性的不连续非线性现象。其中，具体的贡献点包括：

1. 提出了大规模并行多层次不连续非线性可扩展理论。基于高性能计算中的不连续非线性可扩展现象，通过对并行软件在同构多核到异构众核系统上可扩展性的分析，首次系统地对多个层次上的两种现象进行了深入的理论分析与内涵丰富，并提出了可扩展性的物理模型-并行算法-性能优化多层次协同设计理论。该可扩展理论考虑了从底层物理方法建模到上层应用性能优化完整的高性能计算研究链，为高性能计算领域可扩展性的研究，特别是大规模并行计算中的可扩展问题，提供了方法论层面指导。

2. 提出了百核量级的 **Stencil** 并行数值算法，利用向量化和分块技术实现高可扩展单机多核 **Stencil** 计算。在并行算法层次，提出了转置布局计算和时空计算折叠两种 **Stencil** 向量化策略，提高数据在 CPU 内的并行度；同时在性能优化层次，设计高效寄存器数据重用算法和缓存分块优化算法，提高数据的访存效率。实验结果表明，通过并行算法-性能优化的协同设计思想，最高超过 state-of-the-art 方法的绝对性能 4.39 倍，有效地提升了 **Stencil** 并行数值算法的多核可扩展性能。

3. 提出了万核量级的分布式机器学习框架，利用聚类和回归技术实现高可扩展的多机众核机器学习预测模型。本文在理论建模层次，提出了新型 Best Friend 图数据结构及层次化的最小生成树网络模型，设计基于聚类的回归预测方法；在并行算法层次，提出了基于回溯的负载均衡算法和高效的并行通信算法，降低分布式系统的计算和通信开销。实验结果表明，通过物理模型-并行算法协同设计思想，在保证聚类和回归方法准确性的同时，还能有效地将分布式机器学习框架的多机众核扩展性由已有工作的 1,536 核提升至 12,288 核。

4. 针对科学计算软件应用优化，提出了百万核量级的大规模扩展方法，设计实现一套大规模高可扩展国产核材料辐照损伤模拟的软件应用 **OpenKMC**。在物理模型层次，实现优化了高可扩展的新型势函数模型和分组反应策略，支撑应用高效动力学蒙特卡罗模型建立；在并行算法层次，提出了适应于大规模应用的并行同步象限算法和高效自适应通信算法，提高应用的负载均衡和通信效率；在性能优化层次，提出了访存优化技术、高效局部性算法、Athread 线程级异构并行策略和向量化加速方法，通过层次化访存特征提取和轻量级并行性挖掘对异构众核体系结构算力进行优化利用。实验结果表明，通过物理模型-并行算法-性能优化的可扩展性协同设计思想，实现神威·太湖之光上千万亿原子的 520 万核大规模模拟，并行效率高达 80%，成为核材料模拟新的里程碑。

根据我们所提出的多层次协同设计方法，针对 stencil 并行数值算法的细粒度并行问题，可采取并行算法-性能优化双层协同设计，高效地完成算法的实现与优化工作；针对机器学习框架研发的中粒度并行问题，可采用物理模型-并行算法双层协同设计方法，缓解在应用中存在的数据同步、负载不均等带来的通信和计算瓶颈；针对核材料辐照损伤大规模模拟的粗粒度并行问题，需要从物理模型、并行算法和性能优化三个层次进行联合研究，从研究初始就面向可扩展性问

题。

以上三个子贡献点的内容互相支撑，共同构成了处理高性能计算中多层次不连续非线性可扩展问题的新范式，为第一个贡献点的多层次协同设计理论提供了各种并行粒度由低到高、由简到繁的典型应用案例。本文总结并分析了可扩展性中不连续、非线性这一普遍存在现象，系统地梳理了不同技术方向、不同应用领域中的实际问题，特别是大规模百万核量级应用中的可扩展问题，为未来大规模并行计算领域发展提供方法论层面的借鉴和指导。



## 第2章 多层次不连续非线性可扩展理论

本章首先对高性能计算中可扩展性的主要基本概念进行了阐释与定义。然后，通过对计算机软硬件发展的深入分析，总结了一种在可扩展问题中广泛存在的现象，并将其定义为多层次不连续非线性现象（Multi-level Discontinuous and Nonlinear Scalability, MDNS）<sup>[11]</sup>。本文认为，多层次不连续非线性可扩展现象是高性能计算发展的一种普遍现象，也是大规模并行计算中提升可扩展性面临的主要挑战。

针对这一挑战，本章围绕可扩展性这一关键要素，在性能驱动的算法与应用负载统一模型、适应复杂体系结构的新型并行算法、硬件特征与算法精准抽象的深度性能优化三个方面获得突破，最终形成能够充分发挥计算硬件算力水平的物理模型-并行算法-性能优化多层次协同设计理论。

表2.1展示了大规模可扩展性的多层次协同设计理论概览。在可扩展性问题的研究中，物理模型抽象原则主要考虑适用于高可扩展的物理模型与数值方法，实现性能驱动的算法与应用统一建模。在物理模型的统一抽象基础上，并行算法设计原则研究适应复杂体系结构的新型高效并行计算和自适应并行通信算法，同时最大程度地探索计算和通信时间上的优化重叠。与并行算法设计相结合，程序性能优化原则通过主存级-缓存级-寄存器级层次化访存特征提取和进程级-线程级-数据级多层轻量级并行性挖掘，对高性能多核/众核体系结构算力进行更充分地利用并最终形成硬件特征与算法精准抽象的深度性能优化链。

本文第3章、第4至和第5章将以不同粒度的并行问题作为研究对象，论述

表2.1 大规模可扩展性的多层次协同设计理论概览

定理/原则名称	公式
物理模型抽象定理 (2.1)	$m_{OPT} = \arg \max \{ \sum_{i=1}^{n-1} C_p^P(m, m_i) \}$
并行算法设计原则 (2.2)	$T_p(p, N) = \frac{(pf_s + f_p)B_m T_1}{B_a p} + t_s + m_s t_w$
程序性能优化原则 (2.3)	$T_{OPT}(p, N) = \frac{(HT + MR \cdot AMP)T_{mem}(p, N)}{\alpha\beta} + \frac{T_{comp}(p, N)}{n_p n_t v_l}$
多层次协同设计理论 (2.4)	$T_p(m, A) = \frac{(pf_s + f_p)B_m T_1}{B_a p n_p n_t v_l} + (t_s + m_s t_w) + \frac{(HT + MR \cdot AMP)T_{mem}(p, N)}{\alpha\beta}$

可扩展性的多层次协同设计理论如何指导各类应用的可扩展性研究。该理论能够为高性能计算中不同并行粒度的可扩展性研究，特别是大规模并行计算应用的可扩展性研究，提供理论性的创新和方法性的指导。本章接下来将对多层次不连续非线性可扩展现象及可扩展性的多层次协同设计理论展开详细介绍。

## 2.1 基本概念

### 2.1.1 性能指标

可扩展性是高性能计算各类应用中的核心问题。通常情况下，可扩展性被认为是在扩大并行软件应用或并行体系结构规模时，对整体性能相应的提升能力<sup>[11]</sup>。为此，我们需要度量标准来定量地表征可扩展性。

表 2.2 表 2.1 中大规模可扩展性的多层次协同设计理论符号定义

符号	定义
$m_{OPT}$	最佳物理模型
$p$	并行系统计算单元的数量
$P$	并行计算问题
$N$	并行计算问题的规模
$C_p^P$	物理模型抽象指数
$T_p(p, N)$	并行算法主要的时间组成
$f_s$	并行算法串行计算部分所占比例
$f_p$	并行算法可并行部分所占比例（满足 $f_s + f_p = 1$ ）
$B_m$	并行子任务中最大计算负载量
$B_a$	并行子任务中理想化的平均计算负载量
$T_1$	串行算法的运行时间
$t_s$	通信启动时间
$m_s$	通信消息的大小
$t_w$	单条通信链路传输单位数据的时间
$T_{OPT}(p, N)$	性能优化主要的时间组成
$HT$	缓存命中时间
$MR$	缓存未命中频率
$AMP$	缓存未命中的平均时间成本
$T_{mem}(p, N)$	并行算法数据访问时间
$T_{comp}(p, N)$	并行算法计算部分时间
$\alpha$	分块技术的性能提升倍数
$\beta$	寄存器重用技术的性能提升倍数
$n_p$	进程级并行的性能提升倍数
$n_t$	线程级并行的性能提升倍数
$vl$	数据级并行的性能提升倍数
$A$	并行算法
$T_p(m, A)$	多层次协同设计主要的时间组成

**定义 2.1 (加速比).** 考虑一个由  $p$  个计算单元组成的并行系统和一个计算问题  $P$ 。假设  $P$  的实例串行算法的运行时间等于  $T_1$ ；在  $p$  个计算单元上并行运行实例的时间是  $T_p$ 。则在  $p$  个计算单元的并行系统上解决问题  $P$  的该实例时实现的加速比定义为：

$$S_p = \frac{T_1}{T_p}。 \quad (2.1)$$

计算单元的概念取决于底层机器的并行性，这些计算单元可能由核心、处理器甚至容器构成。为简单起见，本文将计算单元对应于处理器。另外，对于一个给定的计算问题  $P$ ，其可能会对应不止一种串行算法，这就导致了不同加速比  $S_p$  的定义。通常情况下，由并行算法运行在一个处理器上得到的执行时间  $T_1$  参与式 2.1 求解的  $S_p$  称为相对加速比；由求解计算问题  $P$  实例的最佳串行算法的执行时间参与求解的  $S_p$  称为绝对加速比。

在理想状态下，一个由  $p$  个计算单元组成的并行系统能提供的加速比等于  $p$ 。然而，现实情况下这是很难达到的。由负载不均引起的处理单元空转和计算单元之间的通信同步是造成可扩展性非线性增长难以达到  $p$  倍加速比的主要原因。因此，我们引入并行效率这一性能指标，用于衡量处理器在执行并行任务时的有效利用率。

**定义 2.2 (并行效率).** 在  $p$  个计算单元的并行系统上解决问题  $P$  的实例时的并行计算效率定义为：

$$E = \frac{S_p}{p} = \frac{T_1}{p \times T_p}, \quad (2.2)$$

通常情况下我们有  $E \in [0, 1]$ 。

当问题规模和处理器数量都增加时，如果有  $E = 1$ ，则该并行算法具有**线性加速比**。线性加速比通常会在由不需要通信且计算负载均衡的并行算法中观察到。因此，对于并行计算和并行通信的高效优化，是并行算法保持线性可扩展的关键。

### 2.1.2 并行粒度

在并行计算中，任务的粒度（或粒度大小）是对处理任务的工作量（或计算量）的度量。通常情况下，不同的应用实例均可划分为细粒度、中粒度和粗粒度三类并行问题<sup>[40]</sup>，而不同并行粒度的应用所面临的扩展性挑战也不相同。

**细粒度并行** 可扩展的细粒度并行问题一般是指向量级或循环级的并行。其一般具有以下特点：首先，并行应用中切分的子任务会再次进行细粒度的分割，使得工作量在处理器之间均匀分配。因此，细粒度并行易于实现负载均衡。其次，细粒度并行问题更适宜在支持快速通信的、规模相对较小的体系结构中进行处理。目前，具有低通信开销的共享内存架构最适合于细粒度并行。另外，在细粒度并行问题中，程序的并行性通常较难挖掘。大多数情况下，均由编译器负责细粒度的简单并行实现，而这也限制了应用扩展性的进一步提高。

**中粒度并行** 可扩展的中粒度并行问题一般是指较大的循环级并行或较小的子任务级并行。一般而言，中粒度并行性更多地是相对于细粒度和粗粒度并行之间的折中，在这种情况下，任务大小和通信时间大于细粒度并行性、小于粗粒度并行性。大多数通用分布式算法框架均属于中粒度并行。

**粗粒度并行** 可扩展的粗粒度并行问题一般是指较大的子任务级并行。在粗粒度并行中，一个程序将被分成若干大的子任务。因此，大量的计算将在各处理器进程中同步进行，极有可能会导致负载不平衡，即其中某些进程处理大量数据，而其他进程可能处于空闲状态。此外，进程之间一般采用的是消息传递接口（Message Passing Interface，MPI）进行数据通信。由于进程之间同步开销较大，高效的并行通信算法设计对程序的可扩展性至关重要。

### 2.1.3 可扩展性定律

**Amdahl 定律** 由 Gene Amdahl 于 1967 年提出的 Amdahl 定律是并行计算领域经典的强可扩展性理论计算模型，即固定软件层面的问题规模情况下，通过增加硬件资源而获得的性能加速比。

Amdahl 定律首先假设程序共有  $W$  次操作； $f_s$  是程序的串行部分所占比例， $f_p$  是程序可以并行的比例，使得  $f_s + f_p = 1$ ； $t_c$  表示基本操作的执行时间。则程序的串行运行时间为：

$$T_1 = (f_s + f_p) \cdot W \cdot t_c \circ \quad (2.3)$$

在并行计算中，给定  $p$  个处理器，Amdahl 定律的第二个假设是我们必须区分两种类型的计算：在单个处理器上执行的串行部分的计算；在所有处理器上执

行的并行部分的计算（理想情况下均衡分割）。则程序的并行运行时间为：

$$T_p = f_s \cdot W \cdot t_c + \frac{f_p \cdot W \cdot t_c}{p}。 \quad (2.4)$$

因此，使用  $p$  个处理器可以实现的最大加速比为：

$$S_p = \frac{T_1}{T_p} = \frac{1}{f_s + \frac{1-f_s}{p}}。 \quad (2.5)$$

随着  $p$  的增长，加速趋向于  $1/f_s$ 。例如，对于一个 10 小时的计算程序，如果我们可以并行化其中 9 小时的计算，而另外 1 小时不能并行化，那么我们最大的加速限制为 10 倍的加速比。即使继续增加计算资源，加速也将保持不变。

Amdahl 定律的强可扩展性分析结果指出，程序能达到的最大加速比被程序的串行部分所限制。这一度使得并行计算的研究热度陷入低潮。

**Gustafson 定律** Gustafson 通过使用 1024 个处理器，实现了超过 1000 倍的加速。这似乎“打破”了 Amdahl 定律并证明了大规模并行处理的合理性。据此，Gustafson 提出了另一种可扩展模型，通常被称为 Gustafson 定律。

与 Amdahl 定律类似，Gustafson 定律基于程序的串行部分和可并行部分的概念进行讨论。然而，Gustafson 的分析并没有在串行算法中考虑两个部分的任务比例，而是假设在并行算法中知道二者的关系。Gustafson 定律假设对于在  $p$  个处理器上进行的并行计算，串行和并行部分分别是  $f'_s$  和  $f'_p$ 。则程序的并行运行时间为：

$$T_p = (f'_s + f'_p) \cdot W \cdot t_c。 \quad (2.6)$$

对于等效的串行算法，程序的串行运行时间为：

$$T_1 = (f'_s + f'_p \cdot p) \cdot W \cdot t_c。 \quad (2.7)$$

假定  $\lambda = f'_s / (f'_s + f'_p)$ ，则  $p$  个处理器可以实现的加速比为：

$$S_p = \frac{T_1}{T_p} = p - \lambda(p - 1)。 \quad (2.8)$$

如果  $\lambda$  足够小, 即程序的串行部分比例足够低, 根据 Gustafson 定律, 我们有  $S_p = p$  的线性加速比。

Gustafson 定律指出可扩展性应同时扩大在并行处理器上所研究的问题规模, 即弱可扩展性。

## 2.2 可扩展性的不连续和非线性现象

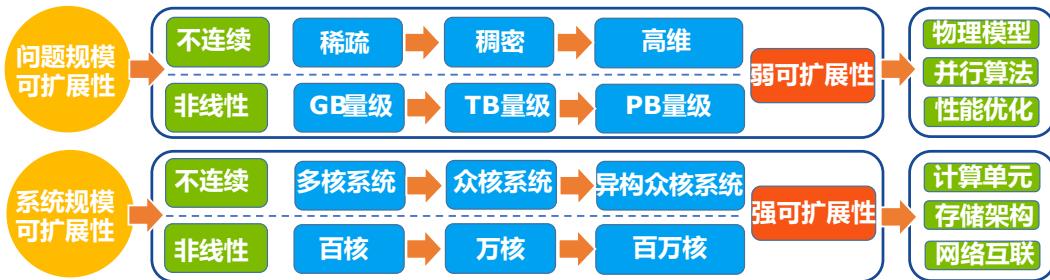


图 2.1 多层次不连续非线性可扩展现象

并行软件设计和并行系统结构是可扩展性一软一硬的两个方面, 两者相辅相成, 也是对立统一的。并行软件设计的高可扩展性意味着在更新底层硬件资源时能够获得更多“免费的午餐”, 即在不需要更改应用的情况下性能也将保持可持续性的增长; 而并行系统结构的高可扩展性则能够对更多的并行软件设计提供高性能的硬件支撑, 以支持处理不同并行粒度的不同应用。

随着系统硬件数量的扩展, 问题规模的扩展与否又分别对应的是弱可扩展性和强可扩展性。如图 2.1 所示, 弱可扩展性主要是在与底层硬件松耦合的物理模型、并行算法、性能优化等上层问题进行研究, 如典型的不连续问题包括各种稀疏、稠密、高维等不同性质的问题, 典型的非线性则包括所研究问题的数据规模在不同量级扩展时性能的非线性增长现象等。强可扩展性主要是在和底层硬件紧耦合的计算单元、存储架构、网络互联等方面的研究, 如典型的不连续问题包括在不同体系结构的硬件上对应用的迁移, 典型的非线性包括在同一体系结构不同硬件数量上的扩展等。事实上, 强弱可扩展性侧重的分别是对硬件和软件层面规模或者资源增加的性能分析。

本文对高性能计算中软硬件的发展和以大规模并行软件优化为代表的各类典型应用中的痛点问题进行了调研与分析, 总结了广泛存在于大规模可扩展问题中的一种现象, 即多层次不连续非线性可扩展性。图 2.1 中展示了在不同层次

的不连续非线性可扩展现象。可扩展性的不连续现象，指的是随着并行计算问题在不同属性更换或在不同硬件架构迁移，性能出现了不同性质的不可连续扩展，即必须更换新物理模型，新并行算法设计或新并行软件性能优化方法以保证可扩展性的现象。可扩展性的非线性现象，则意味着即使在某种物理模型、并行算法设计和并行软件优化实现等最适组合的解决方案范围内，性能也并不能随着问题规模或计算资源的增加而线性提高的现象。非线性主要是对问题量变的一种描述，而不连续性则更多关注的是问题的质变。

### 2.2.1 硬件层面

从硬件层面来看，计算系统的可扩展性存在着广泛的不连续和非线性现象。近几十年来，计算机的计算单元主要历经了从单处理器扩展到多核处理器，再到超级计算机中的众核处理器的转变。硬件上计算架构的发展趋势存在着典型的不连续现象，即其不能依赖单一计算架构的连续升级而获得持续的扩展性。从存储架构的角度来看，由于其与计算单元的发展相比较为缓慢，硬件层面的“内存墙”问题一直较为突出。不同存储介质所造成的不连续性及同一存储下带宽限制带来的非线性始终成为限制可扩展性能的主要瓶颈。从网络互联的角度来看，针对不同计算单元、存储架构所采取的不同网络连接和网络拓扑，是典型的不连续性现象。而在网络传输中的通信问题，始终是造成高性能计算非线性可扩展的核心原因。

**计算单元** 传统上，处理器芯片制造商通过增加处理器的时钟频率来提高处理器的性能。根据摩尔定律，通过在芯片上使用更多的晶体管的方式来提高处理器频率，处理器的性能每18个月便会翻一番。这便是带给软件应用开发者的“免费午餐”，即程序不需要经过太多改动即可获得更高的性能加速。然而，随着摩尔定律的失效，依赖单处理器提升主频以提高性能的时代已经结束。在硬件配置的物理极限到达之前，芯片上过高的频率对电力功耗和散热需求已经无法解决。这是典型的硬件计算架构的不连续。

由于不能继续依赖单一的单处理器获得持续的扩展性能，计算架构逐渐采用新型的多核体系结构，保持时钟频率不变或降低，同时控制能耗需求来增加性能，这也成为并行架构时代的标志。多核处理器通过在单个计算单元中嵌入多个中央处理单元来分别独立地运行不同指令，以提高并行计算的速度。一般地，他

们会共享计算机总线，有时也会拥有共享的存储、外设等。虽然计算能力较于单处理器有了倍式增长，但是此时总线的通信交换成为最明显的性能瓶颈。从吞吐量上来看，所有核心的通信传输均会经由总线，其网络带宽的传输能力无法满足处理器核心数量的扩展需求。

如图 2.1 所示，由于多核处理器性能的不可扩展，导致了计算单元新的不连续性。其中，以 GPU 为代表的异构众核架构因其在计算性能强、访存带宽高、应用范围广等特点逐渐受到开发和研究人员的重视。世界超算应用领域的风向标“戈登贝尔奖”在 2020 年的获奖应用便是采用了 CPU+GPU 异构众核架构的 Summit 超级计算机整机，完成了基于深度学习的大规模分子动力学模拟。而最新的“戈登贝尔奖”（2021 年），其获奖应用 SWQSIM 更是使用了国产异构众核计算单元的新一代神威超算系统，实现了千万核的并行可扩展<sup>[14]</sup>。

**存储架构** 相较于计算单元的快速发展，计算机存储架构的更新升级则缓慢了许多。长期以来，二者不均衡的发展速度导致了在大多数阶段内存的存取速度严重滞后于处理器的计算速度。受到这一“内存墙”的制约，高性能计算中的可扩展性增长缓慢，呈现出的典型的非线性现象。

在计算机存储系统的设计中，分层内存结构是一种内存组织的方法，其可以最大限度地减少数据的访问时间，也是缓解存储系统非线性现象的典型手段。在分层内存系统中，整个可寻址的存储空间从规模最大、访问最慢的存储逐层向处理器靠近成为规模更小、访问更快的存储单元，每个存储层级都是其上层的一个子集。例如，主存（或内存，Main Memory）用于暂时存放 CPU 中的运算数据，以及与硬盘等外部存储器进行数据交换；高速缓存（或缓存，Cache）是一个小而快的存储设备，它用来存储下一层更小、也更快的设备中数据的缓冲区域，也是主存内容的子集；寄存器（Register）则是存储数据最少、访问速度最快的存储单元，其位于处理器内部，是缓存内容的子集。这种内存的分层组织主要是依据**局部性原则**<sup>[41]</sup>，即最近刚被引用过的一个内存位置在不远的将来就会被再次引用（**时间局部性**），或最近刚被引用过的内存位置以及其周边的内存位置更容易再次被引用（**空间局部性**）。

同样，量变在一定程度上的累积就会催生质变。由于物理方面的限制与存储介质的升级，沿着原有单一的存储技术路线发展已经不能缓解内存墙问题，各

种不连续的新型复杂存储架构相继被提出。计算机不同存储架构的硬件升级是其不连续性的典型表现。从最初的单列直插式内存模块（Single In-line Memory Module, SIMM）到现在主流的 DDR（Double Data Rate）存储，极大地提高了硬件存储的扩展性能。

**网络互联** 从网络结构来看，高性能计算机的网络互联拓扑存在 Switch Fabric、Fat-tree 以及 Spine-Leaf 等多种架构；以实现技术而言，高性能计算机的网络实现主要有 Ethernet, Myrinet, Quadrics, Infiniband 和 Switch 等互联技术。这些技术的更新和发展呈现出了网络互联的不连续性。例如，最典型的 Switch Fabric 架构是 Crossbar 模型，其结构简单、易用，但是随着网络规模的扩展，Crossbar 模型交换机的开关密度呈平方级增长，相应的功耗、成本也急剧增加。而 Fat-tree 模型则通过以 Fat-Tree 向根部继续延伸的方式，更适用于网络规模的弹性扩展。

在同一网络互联的场景下，与算法紧耦合的通信模式则是各种高性能计算应用中非线性可扩展的痛点问题。从数据准备、通信准备和数据传输等整个数据通信的全过程中的每个阶段出发，积极找出提高数据通信效率和扩展性的优化算法，设计高效的数据通信模型，是实现系统整体扩展性能增长的关键。

### 2.2.2 软件层面

**物理模型** 物理模型是对实际研究问题的抽象，每一个物理模型的建立都有一定的条件和应用范围。针对具体的计算应用，从众多适用的物理模型中优选出高可扩展的物理模型，是不连续性的典型表现。而在同一物理模型下构建的应用，也会因处理的数据量、调节的参数域、叠加的限制项等各种因素而出现可扩展性的性能非线性问题。

我们以大规模材料科学的扩展性研究为例。在材料科学的研究中，分子动力学模型和动力学蒙特卡罗模型是两种最主要的物理建模方法。

分子动力学模型是利用经典的牛顿力学方程，对材料中粒子的运动轨迹进行数值求解模拟。分子动力学模型利用确定性的物理方程，能够更准确地记录和描述体系中各粒子的微观演化轨迹，提供高分辨率的系统动态模拟。相应地，由于计算精度需求高、计算量较大，导致分子动力学模型通常难以对材料体系进行长时间大规模的模拟<sup>[35]</sup>。

动力学蒙特卡罗模型则是利用随机过程理论，实现对材料中微观尺度整个

体系演化情况的模拟。相比于分子动力学模型，其摈弃了与体系穿越势垒无关的微小的原子级别的振动，只着眼于体系的全局组态变化。因此，动力学蒙特卡罗模型不能描绘原子的运动轨迹，其更关注的是体系的组态跃迁。通过动力学蒙特卡罗模型，可以实现在秒级时间尺度上对整个体系的模拟<sup>[34]</sup>。

由此可见，在材料科学领域的大规模应用模拟中，不同物理模型方法拥有各自的特点。当应用场景或者模拟需求发生变化时，原有的模型即不再适用，需要更换新的物理模型以满足应用的计算需求，体现了物理模型的不连续性。另外，在特定的物理模型中，不同物理参数的调节方案等因素也和应用的扩展性能息息相关，也是导致可扩展性非线性变化的重要原因。

**并行算法** 在并行算法设计层次，高性能计算专家通常将大规模应用中的几种典型算法抽象出来进行研究，并分为几种经典的“Dwarfs”。Dwarf 是一种并行计算和通信模式。Berkeley 从架构、算法和应用的角度明确定义了七种模板 Dwarfs。它们分别为：多体方法，蒙特卡罗方法，谱方法，结构化网格方法，非结构化网格方法，稠密线性代数方法，以及稀疏线性代数方法。根据 Berkeley 的描述，至少在未来几十年，这七种模板 Dwarfs 对科学和工程都是十分重要的算法<sup>[27]</sup>。

将模板 Dwarfs 算法相关的应用程序扩展到大型系统，一直面临着可扩展性不连续和非线性的极大挑战。以稠密线性代数的计算为例。70 年代实现的 LINPACK 和 EISPACK 基于向量处理器，分别包含了多种线性方程求解器及计算特征值的 Fortran 程序；80 年代实现的 LAPACK 则更注重于利用多层次的缓存架构提高数据的局部性；90 年代开发的 ScaLAPACK 则针对分布式系统，进一步实现了数据局部性和负载均衡的统一。随着 2000 年以后新型硬件的蓬勃发展，适于多核系统的 PLASMA 和适于众核系统的 MAGMA 实现了更细粒度的任务图并行。如上所述，沿着稠密线性代数核心算法库的发展来看，不同的并行算法面向于不同的问题应用、适用于不同的底层硬件。当应用的计算需求或者硬件的体系架构，特别是新型交叉计算应用或者关键计算单元的构件发生变化之后，就要以新的思路来设计并行算法，这展示了可扩展性不连续现象。

而在大规模并行计算系统上，特别是面向 E 级计算的应用中，这些并行算法的通信将成为性能非线性增长的主要瓶颈。在通信技术的优化上，一方面提出了通信避免算法降低通信的冗余度，另一方面也利用半整数维度划分策略，在并

行性、局部性、通信量等方面不断进行平衡。因此，随着对进程间并行通信和并行计算的优化，各种方法的可扩展性愈加呈现非线性，而且通常出现交替上升的趋势，即不同方法具有不同的最佳适用区间。这种不同并行算法在各种计算和通信上优化方法的发展可以看为是依赖应用而进行细粒度调优的一种非线性可扩展现象。

**性能优化** 性能优化的本质是实现算法性能特征到体系结构性能特征的高效映射，其与并行算法的主要差异是各自所处的研究环境不同。性能优化侧重的是在底层硬件特征抽取的基础上进行紧耦合的深度性能挖掘，一般情况下，一套完整的性能优化方法链可适用于相同硬件的不同应用，并都能够取得有效的扩展性提升。而并行算法则更多强调的是与应用相关紧耦合的算法设计。

在高性能计算中普遍应用的分块 (Tiling, 或 Cache Blocking) 这一编译技术应归为一种性能优化方法。分块方法的变化体现出其对底层硬件存储层次发展的不断改进和升级。最初的分块方法主要用于矩阵计算，即利用手工编写或自动编译等方式完成对循环的划分，以在每个内层循环的小子块上提高数据的局部性。由于分块方法能够非常有效地实现性能提升，后续大量的性能优化工作均围绕分块算法展开研究。1999 年 MIT 的研究团队利用分治的思想，提出了缓存无关分块的性能优化方法，并将之前的分块方法统称为缓存相关的优化方法<sup>[11,42]</sup>。缓存无关分块方法可以更好地适应不同硬件的存储层次及缓存大小，使得其总能获得较优的访存复杂度。以缓存相关方法和缓存无关方法等不同访存优化技术的发展而言，这种基于硬件特征而进行的算法变化和升级体现了分块方法在优化方法层面的不连续性现象；而分块方法根据缓存块大小、缓存行大小、相联度等特征进行参数调优而引起扩展性能的变化，则是典型的性能优化的非线性现象。

与缓存分块方法类似，多线程加速、向量化等并行化技术均为在硬件架构基础上进行深度性能挖掘的优化方法。其中，多线程背后的思路是给予并行调度器尽可能多的线程以供调度，从而最大化地利用所有计算资源；而向量化则是利用 SIMD 技术对 CPU 中的数据进行并行化处理，进一步提高计算的吞吐量。根据底层硬件提供的技术机制，不同性能优化方法的高效组合是典型的不连续问题，而每一种方法的实现方式（手工实现，自动编译，汇编优化等等）也存在着可扩

展性的非线性挑战。

如上，性能优化方法与底层的硬件架构特征息息相关。当底层硬件的体系结构发生更新变化之后，相应的性能优化方法也要进行新的变革，以进一步提高在相应系统上的可扩展性。

### 2.3 多层次协同设计方法

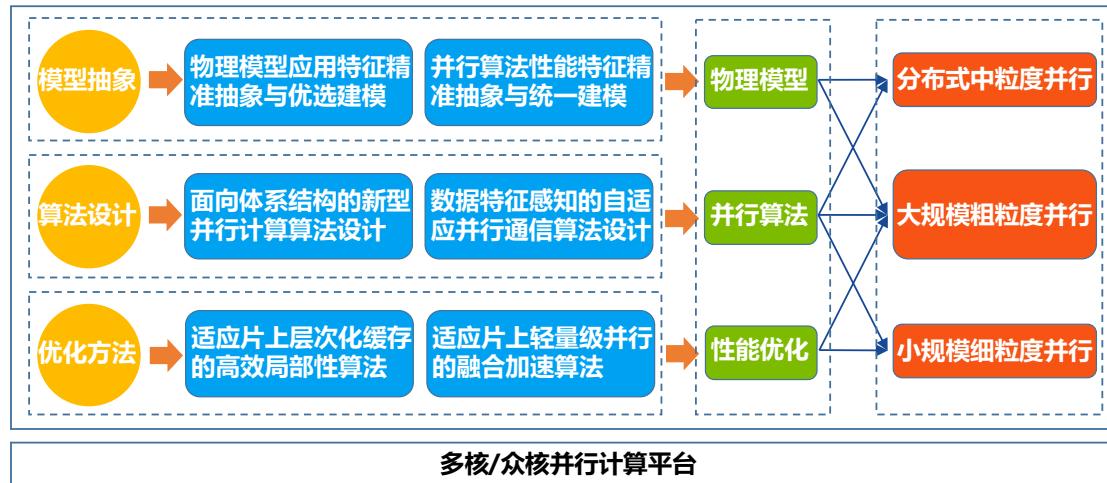


图 2.2 可扩展性不同并行粒度的分层设计抽象

前文从高性能计算中软硬件的发展及大规模应用软件的优化角度出发，分析了在可扩展性中不同方面广泛存在的不连续非线性现象。本节将基于上层物理模型、中层并行算法、下层性能优化等不同层级，提出一种各层级之间相互融合、相互配合的多层次协同设计理论。可扩展性不同并行粒度的多层次协同设计方法抽象如图 2.2 所示。根据我们所提出的多层次协同设计理论，针对并行数值算法的细粒度并行问题，可采取并行算法-性能优化双层协同设计，高效地完成算法的实现与优化工作；针对分布式框架研发的中粒度并行问题，可采用物理模型-并行算法双层协同设计方法，缓解在应用中存在的数据同步、负载不均等带来的通信和计算瓶颈；针对大规模科学计算的粗粒度并行性问题，则需要从物理模型、并行算法和性能优化三个层次进行联合研究，从研究初始就面向可扩展性问题。本节将具体展开可扩展性的多层次协同设计方法。

#### 2.3.1 物理模型抽象

物理模型是对实际研究问题的抽象，每一个物理模型的建立都有一定的条件和应用范围。在可扩展性问题的研究中，物理模型层次的可扩展性主要基于上

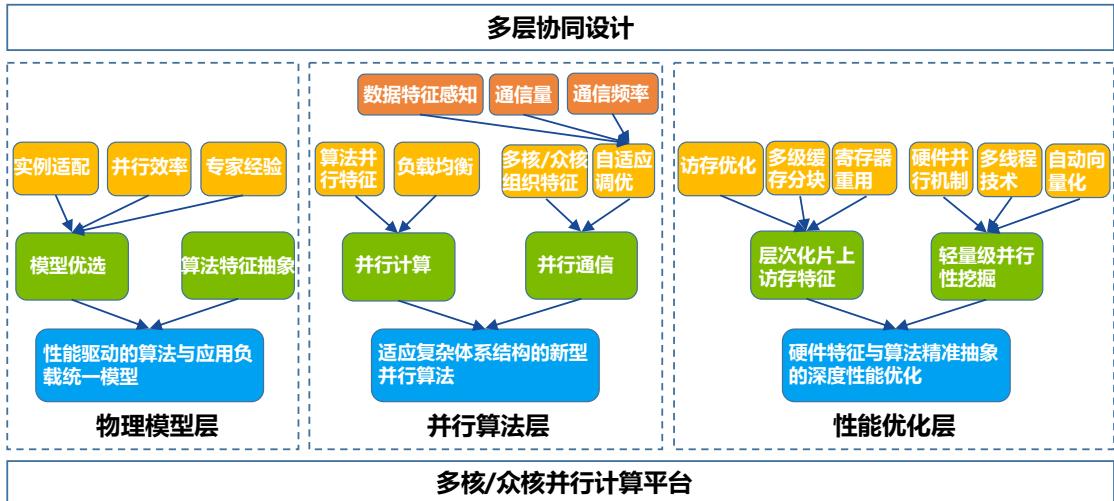


图 2.3 大规模并行应用的可扩展性多层次协同设计方法

层的软件应用，考虑适用于高可扩展的物理模型与数值方法。物理模型层属于较上层的研究，其与中间算法层紧耦合。一方面，不同物理模型适用的应用范畴不相同，而适用同一应用的多种物理模型也会带来扩展性能上的明显差异，这导致了模型抽象和择优选择上的复杂化；另一方面，每种模型中不同算法的并行特征又具有多变性，这种多变性又导致了模型并行化时扩展性的非线性现象。因此，实现性能驱动的算法与应用负载统一模型是实现应用高可扩展性的重要前提。

式 2.9 中的  $M^P$  描述了可用于求解  $P$  所有实例的典型物理模型的有限集合。对于计算问题  $P$  的每一个实例  $P_i$ ，如果可用于求解  $P_i$  的物理模型有限集  $M^{P_i}$  中存在模型  $m$ ，且其能够有效地适配到其他实例 ( $e_m > 0$ )，则所有  $\langle m, e_m \rangle$  的并集即为计算问题  $P$  的有效物理模型集合。

$$M^P = \bigcup \{ \langle m, e_m \rangle | \langle m, e_m \rangle \in M^{P_i}, e_m > 0 \} \quad (2.9)$$

**定义 2.3 (物理模型抽象原则)。** 考虑一个由  $p$  个处理器组成的并行系统和一个计算问题  $P$ 。假设  $P$  共有  $k$  个典型计算实例，则利用并行算法求解问题  $P$  的模型  $a$  和模型  $b$ ，在该系统上的物理模型抽象指数  $C_p^P$  定义为：

$$C_p^P(m_a, m_b) = \sum_{i=1}^k \text{sgn}(E_p^{P_i}(m_a) - E_p^{P_i}(m_b)) \quad (2.10)$$

其中， $m_a \in M^P, m_b \in M^P$ ， $E_p^{P_i}$  为在该  $p$  个处理器的系统上求解实例  $P_i$  的并行

效率。函数  $\text{sgn}$  为三值激活函数（符号函数），即：

$$\text{sgn}x = \begin{cases} -1 : x < 0 \\ 0 : x = 0 \\ 1 : x > 0 \end{cases} \quad (2.11)$$

根据式 2.10 中定义的物理模型抽象指数，我们可以很明确地对物理模型的选择做出判断：对于在  $p$  个处理器的系统上求解给定的计算问题  $P$ ，当  $C_p^P(m_a, m_b) > 0$  时，应选择有着更强可扩展性的物理模型  $m_a$ ；当  $C_p^P(m_a, m_b) < 0$  时，应选择有着更强可扩展性的物理模型  $m_b$ ；当  $C_p^P(m_a, m_b) = 0$  时，物理模型  $m_a$  与  $m_b$  有着相似的可扩展性，此时应根据专家经验选取物理模型  $m_a$ 。

**定理 2.1.** 给定一个物理模型  $m$ ，如果  $m$  在满足问题  $P$  实例的准确性要求下，相比其他模型，在  $p$  个处理器上并行计算时能够获得最多实例数量的高并行效率，即满足：

$$m_{OPT} = \arg \max \left\{ \sum_{i=1}^{n-1} C_p^P(m, m_i) \right\}, \quad (2.12)$$

则物理模型  $m$  为在  $p$  个处理器的并行系统上的最佳物理模型  $m_{OPT}$ 。

证明. 对于  $n$  个元素的有限集  $M^P$  ( $n = |M^P|$ )，当  $n = 1$  时， $M^P$  中的唯一元素  $m$  即为最佳物理模型；当  $n > 1$  时，任意取出两个模型  $m_a$  与  $m_b$ ，根据定义 2.3，必有一模型  $m_a$ ，使得  $C_p^P(m_a, m_b) \geq 0$ 。从  $M^P - m_b$  中重新任取一新物理模型  $m_c$ ，并将  $C_p^P(m_a, m_b)$  与  $C_p^P(m_a, m_c)$  累加，直到  $M^P = \emptyset$  为止。此时，得到了模型  $m_a$  相对其他模型的物理模型抽象指数累加和  $C_{sum}(m_a)$ 。重复上述操作并将物理模型抽象指数累加和两两比较取较大值，最后必有唯一的  $m$  相对其他所有模型有最大的  $C_{sum}(m)$  值，即  $m$  为在  $p$  个处理器的并行系统上的最佳物理模型  $m_{OPT}$ 。□

我们以大规模材料科学的数值模拟为例来阐释物理模型层次的可扩展性。材料科学是基础科学的重要分支，也是世界各国科技发展长期关注的重要领域。传统上材料科学的发展依赖于理论研究和物理实验相结合，复杂的理论分析求解与耗时的实验操作验证增加了大量的人力、物力与时间成本，极大地阻碍了材料科学的发展。因此，以数值模拟方法为核心的计算物理学、计算化学等多领域应用的交叉学科逐渐兴起。其特点是基于一定的物理化学原理，利用计算机甚至超

算机器提供的强大算力，对特定的研究体系进行数值模拟计算，以研究材料的微观演化机制，从而在理论科学和物理实验之间建立起可靠的联系纽带。

核材料的辐照损伤是材料科学中经典的数值模拟应用。诺贝尔物理学奖得主恩利克·费米曾指出，核技术的成败很大程度上取决于材料在反应堆中辐射场下的行为。受高能粒子的辐照，核反应堆压力容器会出现退化损伤，其主要原因可归于在中子辐照下钢中富铜沉淀物的形成。动力学蒙特卡罗方法（KMC, Kinetic Monte Carlo）和分子动力学模拟方法（MD, Molecular Dynamics）是用于材料科学的研究的两种最主要的数值模拟方法。对于求解大规模核材料辐照损伤的问题  $P$ ，可分别构建出动力学蒙特卡罗模型  $m_{KMC}$  以及分子动力学模型  $m_{MD}$ ，即  $M^P = \{m_{KMC}, m_{MD}\}$ 。

动力学蒙特卡罗模型  $m_{KMC}$  通过利用随机过程理论，实现对材料中微观尺度整个体系演化情况的模拟。其摈弃了与体系穿越势垒无关的微小的原子级别的振动，只着眼于体系的全局组态变化，因此，动力学蒙特卡罗模型  $m_{KMC}$  不能描绘原子的运动轨迹，其更关注的是体系的组态跃迁。通过动力学蒙特卡罗模型  $m_{KMC}$ ，可以实现在秒级时间尺度上对整个体系的模拟。

分子动力学模型  $m_{MD}$  则是利用经典的牛顿力学方程，对材料中粒子的运动轨迹进行数值求解模拟。与动力学蒙特卡罗模型  $m_{KMC}$  不同，分子动力学模型  $m_{MD}$  利用确定性的物理方程，能够更准确地记录和描述体系中各粒子的微观演化轨迹。相应地，由于大量复杂的计算，导致分子动力学模型  $m_{MD}$  通常难以对材料体系进行长时间大规模的模拟。

通常情况下，核材料的辐照损伤模拟具有较大的时间和空间跨度。此时适于长时间对大体系模拟的动力学蒙特卡罗模型  $m_{KMC}$  可以相对更准确地实现对全局组态的演化。因此，对于两种模型，我们有  $e_{m_{KMC}} > e_{m_{MD}} > 0$ 。另外，大量的数值模拟实验表明，与分子动力学模型  $m_{MD}$  相比，利用动力学蒙特卡罗模型  $m_{KMC}$  构建的解决方案通常具有更少的计算量、更小的通信需求，从而能获得更高的并行效率。根据式 2.10，我们有  $C_p^P(m_{KMC}, m_{MD}) > 0$ 。由定理 2.1 可知，大规模核材料辐照损伤问题 ( $P$ ) 的最佳物理模型为  $m_{KMC}$ ，即这种杂质溶质簇的形成过程应构建动力学蒙特卡罗模型以在原子水平上进行模拟。

### 2.3.2 并行算法设计

在多层次的协同设计方法中，算法设计是基于应用中核心函数的中间优化层，主要是负责高效地并行化。其可以与上层物理模型紧耦合，面向应用设计高可扩展性的物理模型；也可以与下层的性能模型紧耦合，面向架构完成高效的性能优化。然而，随着体系结构组织向复杂性和多样性演进，具备良好扩展性能的并行算法设计和实现日益困难。在如今主流的多核/众核体系结构下，算法的并行特征主要为算法的并行计算和并行通信特征。因此，本节将在统一抽象物理模型的基础上，研究适应复杂体系结构的新型并行算法，一方面充分挖掘算法的并行特征以重新定义算法并行任务划分，完成高效的并行计算；另一方面充分研究多核/众核的组织结构以实现数据特征感知型的自适应并行通信算法，同时最大程度地探索计算和通信时间上的优化重叠，最终完成并行算法在多核/众核处理器上的高可扩展。

**定义 2.4 (并行时间组成).** 令  $N$  为问题规模， $p$  为处理器数量，则并行算法的执行时间可以分解为计算时间 ( $T_{comp}(p, N)$ )、通信时间 ( $T_{comm}(p, N)$ )，访存时间 ( $T_{mem}(p, N)$ )，以及其他开销 ( $T_{other}(p, N)$ )，即：

$$T_p(p, N) = T_{comp}(p, N) + T_{comm}(p, N) + T_{mem}(p, N) + T_{other}(p, N)。 \quad (2.13)$$

其中，本节所讨论的并行算法设计主要针对的是计算时间 ( $T_{comp}(p, N)$ ) 和通信时间 ( $T_{comm}(p, N)$ ) 两项的算法层面优化。而计算时间 ( $T_{comp}(p, N)$ ) 和访存时间 ( $T_{mem}(p, N)$ ) 的进一步性能优化则更多地需要结合硬件特点进行，具体细节将在下一节展开讨论。因此，并行算法设计中主要的时间组成可简化为：

$$T_p(p, N) = T_{comp}(p, N) + T_{comm}(p, N)。 \quad (2.14)$$

如第 2.1 节中所述，在主要考虑开销较大的计算时间和通信时间的理想情况下，并行算法的运行时间  $T_p(p, N)$  中有  $T_p(p, N) = T_{comp}(p, N) + T_{comm}(p, N) = T_1(N)/p$ ，其中  $T_1(N)$  为算法的串行运行时间。然而，在并行化的过程中通常会引入额外的低效因素，例如负载不均、通信瓶颈等问题，这会导致  $(T_{comp}(p, N) + T_{comm}(p, N)) > T_1(N)/p$ 。

**定义 2.5 (并行算法计算时间).** 并行算法计算时间定义为对算法计算部分有效地并行实现所需要的运行时间，即

$$T_{comp}(p, N) = R_0 \times \frac{T_1(N)}{p}。 \quad (2.15)$$

其中， $R_0$  通常是一个大于 1 的常数，即在无通信开销的并行实现下常数倍的减速是合理并行化的结果。

某些因素可能会导致  $R_0$  小于 1，例如，相比串行算法，平行算法在每个进程上的访存可能会更接近处理器而带来额外的加速。由于这些因素通常是由结合硬件的性能优化引起，并不是算法设计本身的属性特征，所以在本小节中将暂不考虑这些因素的影响。

下面我们对  $R_0$  进行详细剖析来探索对并行算法计算性能的提升。定义 2.15 中并行算法计算部分的常数  $R_0$  是针对算法的并行特征进行抽取，并对其进行有效地并行化以实现负载均衡的计算优化系数。一般情况下，算法性能特征的抽象和建模是和具体算法相关的。在统一抽象模型的基础上，首先需要实现具体算法在不同粒度并行性的描述和定义，探索算法最大程度的并行化比例；然后通过对算法并行性以及并行子任务间的计算特点，完成算法负载均衡的实现，充分挖掘算法并行性。

**定义 2.6 (并行计算系数).** 令  $f_s$  是并行算法的串行计算部分所占比例， $f_p$  为抽取算法性能特征后算法可并行部分所占的最大比例，二者满足  $f_s + f_p = 1$ ； $B_m$ 、 $B_a$  分别为并行子任务中最大的计算负载量与理想化的平均负载量，则  $p$  个处理器上的并行计算系数  $R_0$  可由下式进行优化：

$$R_0 = \frac{f_s + \frac{f_p}{p}}{\frac{1}{p}} \times \frac{B_m}{B_a} = (pf_s + f_p) \times \frac{B_m}{B_a}。 \quad (2.16)$$

如式 2.14 所示，通信是并行算法设计中另一主要的时间开销来源。并行算法的通信特征在传统高性能计算软件生态中并没有被精准抽象，而是作为领域知识被隐藏在了架构相关的优化中，这就使得面对不同的应用或者不同的硬件时需要对通信重新进行分析优化。充分研究多核或众核的硬件组织结构，以实现数据特征感知型的自适应并行通信算法，是大规模并行应用中通信优化的必要手

段。

在大规模的并行算法设计中，众多进程之间一般采用的是消息传递的方式在整个计算网络进行数据通信。通过网络传输消息的总时间主要包括以下几部分：启动时间  $t_s$ ，即发送和接收节点建立通信所需要的准备时间；单跳时间  $t_h$ ，又称节点延迟，即包括交换机延迟、网络延迟等因素；单发时间  $t_w$ ，即经由单条通信链路传输单位数据的时间开销。

**定义 2.7(并行算法通信时间).** 令  $m_s$  为发送消息的大小， $l$  为通信链路的数量，则节点间  $l$  次通信链路并行通信时间组成为：

$$T_{comm} = t_s + (m_s t_w + t_h)l。 \quad (2.17)$$

由于在实际通信中，单跳时间  $t_h$  通常远小于启动时间  $t_s$  和单发时间  $t_w$ ，且通信链路  $l$  一般不由用户所控制，因此式 2.17 可进一步简化为：

$$T_{comm} = t_s + m_s t_w。 \quad (2.18)$$

**原则 2.2(并行算法设计原则).** 通过挖掘算法的并行特征，提高算法可并行计算的比例、有效并行化以实现负载均衡，可提升并行计算的可扩展能力；充分研究硬件组织结构和感知数据特征，减少数据通信总量、降低通信频率，可提升并行通信的可扩展能力；同时将二者结合考虑，探索计算和通信时间上的优化重叠，可提升并行算法整体的可扩展能力。

证明. 由式 2.15 中并行算法计算时间的定义及式 2.17 中并行算法通信时间的定义，可得并行算法的主要时间组成为：

$$\begin{aligned} T_p(p, N) &= T_{comp}(p, N) + T_{comm}(p, N) \\ &= R_0 \times \frac{T_1}{p} + (t_s + (m_s t_w + t_h)l) \\ &= \frac{(p f_s + f_p) B_m T_1}{B_a p} + t_s + m_s t_w。 \end{aligned} \quad (2.19)$$

根据式 2.1 中加速比的定义及式 2.19 中并行计算主要时间组成，我们可得到并行

算法在  $p$  个处理器上的加速比为：

$$S_p = \frac{pT_1}{\frac{(pf_s + f_p)B_m T_1}{B_a} + p(t_s + m_s t_w)} \quad (2.20)$$

以及并行算法在  $p$  个处理器上的并行效率为：

$$E_p = \frac{T_1}{\frac{(pf_s + f_p)B_m T_1}{B_a} + p(t_s + m_s t_w)} \circ \quad (2.21)$$

其中， $p$  为处理器的数量， $T_1$  为算法的串行运行时间， $B_a$  为并行子任务中理想化的平均负载量， $t_w$  为通信网络中的单发时间，这些变量通常均为并行算法设计中的不可控变量。因此，提高算法的可扩展性需对式 2.20 及式 2.21 中的其他变量进行分析。针对计算项，由于大规模计算中处理器的数量  $p$  通常较大，因此提高  $f_p$ 、减少  $f_s$ ，即增大算法中可并行部分的比例，可有效提升加速比  $S_p$  及并行效率  $E_p$ ；而减小  $B_m$ ，即对各并行子任务负载进行更均衡的分割，也可提高并行计算项的效率。针对通信项，减少  $m$ ，即对发送数据进行精准感知提取以降低通信总量，可有效提升算法的可扩展性；而减小  $t_s$ ，即降低通信频率，也可进一步提升通信效率。由于计算项和通信项是累加和计算，因此，实现二者实现时间的交叉重叠，可有效减少式 2.20 及式 2.21 中的分母项，提高并行算法的可扩展性。□

### 2.3.3 性能优化方法

性能优化层次主要是考虑适配硬件架构的高可扩展性能模型。处理器指令主要有两种：访存指令和计算指令。这两种指令在处理器核内部是独立调度在不同指令执行单元上运行的。因此，与算法设计相结合，性能优化方法通过层次化片上访存特征提取和轻量级计算并行性挖掘，对高性能多核/众核体系结构算力进行充分利用，并最终形成硬件特征与算法精准抽象的深度性能优化链。

如第 2.3.2 节中所述，式 2.13 中的计算时间 ( $T_{comp}(p, N)$ ) 和访存时间 ( $T_{mem}(p, N)$ ) 需要结合硬件特点进行进一步的性能优化， $T_{other}(p, N)$  则一般包含程序正常运行所需要的固定开销部分，因此，性能优化方法中主要的时间组成可简化为：

$$T_{OPT}(p, N) = T'_{mem}(p, N) + T'_{comp}(p, N) \circ \quad (2.22)$$

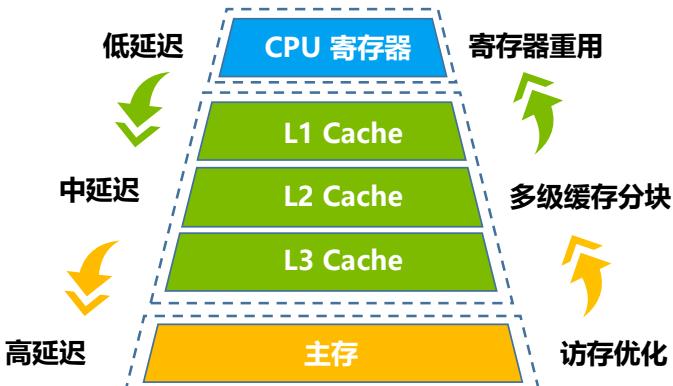


图 2.4 主流处理器层次化存储抽象框图

其中  $T'_{mem}(p, N)$  和  $T'_{comp}(p, N)$  分别为性能优化后的访存时间开销和计算时间开销。本节将首先从主存级-缓存级-寄存器级三个连续存储层级由低到高深入分析处理器层次化片上访存的特征，在此基础上实现对处理器读存性能优化的统一建模，并研究读存模型在性能优化中的制导方法和作用。

图 2.4 中展示了目前主流处理器层次化存储抽象框图。层次化片上访存特征主要是指处理器片上层次化缓存的组织与数据共享方式。由于缓存层次的容量大小差异巨大，随着数据在存储层次中不同的位置，导致访问数据的延迟时间和访问模式也存在很大差异。其中，寄存器是 CPU 内部有限容量的高速存储部件，它们可用来暂存指令、数据和位址。当 CPU 请求更多的数据访问时，会依次查找数据所在的 cache line 是否存在 L1 级、L2 级、L3 级甚至是主存中。如果查找缺失，数据所在的 cache line 将会从低级的内存层次依次传输到高层的内存层次，最终这行 cache line 会传输到 L1 cache 供寄存器访存指令使用。不同体系结构对于 Cache 的设计，比如 Cache 大小、Cache 结构、Cache 组织具有较大的差异性，需要进行统一的性能优化抽象。例如，华为鲲鹏 920 处理器单个处理器核独享 L1 Cache，核组共享 L2 Cache，所有处理器共享 L3 Cache；飞腾 2000 处理器单个处理器核独享 L1 Cache，核组中的簇共享 L2 Cache，无 L3 Cache。因此，对于层次化片上访存特征的提取抽象并充分利用是影响应用负载性能的关键要素。

在图 2.4 的读存模型中，访存的最底层是主存层。在计算机科学中，平均内存访问时间 (AMAT) 是通过使用命中时间、未命中率和未命中惩罚项三个因素来分析内存系统性能的常用指标。命中时间 (Hit Time, HT) 是在缓存中命中的时间，未命中率 (Miss rate, MR) 是缓存未命中的频率，未命中惩罚项是缓存未命中

的平均时间成本 (Average Miss Penalty, *AMP*)。具体可以用公式 2.23<sup>[43]</sup> 来定义。

$$AMAT = HT + MR \cdot AMP \quad (2.23)$$

为了减少 *AMAT* 以提高性能，我们可以针对这些指标逐一研究如何在主存层面降低这些因素带来的性能影响。例如，在神威 SW26010 处理器上，当 cache line 大小为 256 字节时，即使访问的地址是随机的，也会达到峰值带宽<sup>[44]</sup>。因此 256 字节是实现随机内存访问峰值带宽的最小宽度，更大的宽度会增加 *MR* 和 *AMP*。另外，系统的 cache 大小是有限的，充分压缩结构体大小，使得 cache 能缓存更多的被访问数据，无疑是提高内存平均访问速度的有效方法之一，而这需要我们对主存中的数据结构做好更合理的设计。

*Cache* 是属于层次化存储架构中间层的数据读存单元，对 *cache* 中数据的高效利用是建立读存模型重要的考虑因素。分块是一种在 *cache* 级别被广泛应用的性能优化技术，可以有效缓解大量应用程序中的内存带宽瓶颈，提高 *cache* 中数据的利用率。分块背后的关键思想是通过确保会被多次使用的数据保留在缓存中而不被换出，以实现缓存级别的数据重用。通常情况下，可以在一维、二维或三维的空间数据结构上进行分块，即空间分块技术 (Spatial Tiling)。近年来，时间分块技术 (Temporal Tiling) 也被提出以允许更多的缓存数据在时间维度进行重用，以进一步缓解带宽瓶颈。在具体实现方面，分块通常涉及数据重排和循环展开等进一步的细粒度组合优化方法。由于底层硬件种类以及分块算法实现千差万别，通常较难定量地对 *cache* 级别的优化进行抽象衡量。本节中，我们假定采用分块技术在 *cache* 级别可以带来  $\alpha$  倍的性能收益提升。

寄存器是位于 CPU 或 GPU 内部稀少的高速存储器，用于用来暂存指令、数据和位址。寄存器是计算机体系结构中的关键资源之一，也是读存模型中的最高层存储。在编译的代码生成阶段，程序中的变量会被编译器替换为寄存器。对于大规模并行计算应用而言，其使用的变量数量可以是几乎无限的，CPU 或 GPU 中的寄存器数量是远远不能满足所有变量的存储需求。因此，控制寄存器的分配和重用，一直是程序性能优化中最为重要的问题之一，也是编译理论中热点的研究问题。通常情况下，编译器会自动将经常使用的变量放在寄存器里，来避免反复地存取。而手动优化方法，例如公共子表达式抽取或者循环不变量识别等，往

往可以进一步带来更高的寄存器重用率，减少数据在 CPU 与 cache 之间的移动。同样地，本节中假定采用寄存器重用能够带来  $\beta$  倍的性能收益提升。

如图 2.4 所示，通过以上层次化片上访存特征的提取抽象，本节建立起从主存级-缓存级-寄存器级三个连续存储层级由低到高的处理器读存性能优化的统一模型，研究了该读存模型在性能优化中的制导方法和作用。

**定义 2.8 (层次化访存特征提取).** 层次化访存特征提取定义为对应用访存部分由低层到高层的高效性能优化，使得优化后的访存时间  $T'_{mem}(p, N)$  为

$$T'_{mem}(p, N) = \frac{1}{M_0} \times T_{mem}(p, N)。 \quad (2.24)$$

其中， $M_0$  通常是一个大于 1 的常数，即在层次化访存性能优化下常数倍的加速是有效的优化结果。

根据上文对主存级-缓存级-寄存器级三层访存特征提取的分析，我们对  $M_0$  进行进一步地详细剖析：

**定义 2.9 (访存性能优化系数).** 令  $\alpha$ 、 $\beta$  分别为采用分块技术和寄存器重用技术所带来的性能提升倍数，则  $p$  个处理器上的层次化访存性能优化系数  $M_0$  为：

$$M_0 = \frac{\alpha \cdot \beta}{AMAT} = \frac{\alpha\beta}{HT + MR \cdot AMP}。 \quad (2.25)$$

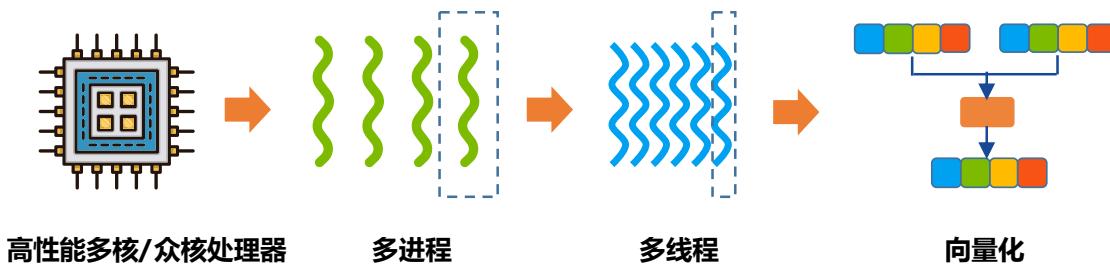


图 2.5 主流处理器轻量级多层次并行（进程级-线程级-数据级）抽象框图

多核/众核体系结构下，处理器核心组织方式的复杂性导致性能优化方法的设计和实现困难。充分挖掘硬件在进程级-线程级-数据级的多层次并行性、研究定义并行任务间的计算特征到硬件并行特征的高效映射成为高可扩展并行程序性能优化的关键。因此，对适应当前多核/众核复杂体系结构的轻量级并行性挖掘，可以充分发挥计算硬件的算力水平、提高不同粒度的并行性。

如上所述，在大规模并行计算应用中，众多处理器之间一般采用的是消息传递接口的方式进行数据通信。相较于这种同步开销较大的处理器间并行计算方式，共享片上信息的**轻量级**处理器内并行可在硬件层面提供更进一步的性能提升。图 2.4 中展示了目前主流处理器轻量级并行抽象框图。

高性能多核/众核处理器上通常配有一个或多个物理核心，多个处理器核心通过系统总线或交叉开关共享一个或多个内存模块。在众核处理器的架构中，通常采用计算核心和加速核心异构组合的方式进行硬件组织设计。作为处理器内的第一层并行，可采用多进程技术在同一处理器各个计算核心上进行轻量级的并行计算加速。实现方式上，可使用基于共享内存的消息传递功能完成异构众核处理器上的多进程计算。假定同一处理器上有  $n_p$  个计算核心，则理论上轻量的进程级加速可以对计算部分带来  $n_p$  倍的性能提升。例如，神威 SW26010 众核处理器分为 4 个核组，每个核组由 1 个计算核心（主核）和 64 个加速核心（从核）构成。因此，在神威 SW26010 众核处理器上轻量的进程级加速可以带来 4 倍的理论性能提升。相较于众核处理器异构架构的特点，多核处理器大多采用多个同构核心进行硬件的组织设计。当线程数与处理器核心数目相等时，也可以简单地认为多核处理器上的每个线程即为一个进程。因此，共享内存架构的多核处理器进程级和线程级并行一般以线程的数量作为分水岭，两级并行之间相互融合，差异较小。

作为处理器内的第二层并行，多线程技术则是为了充分利用处理器多个核心的计算能力，把单进程的计算任务再次切割为细粒度的线程并分配到处理器的每个核心上，达到并行计算，提高效率的目的。对于众核处理器的架构而言，则是将在计算核心进程上运行的任务切分到加速核心上。假定同一处理器上有  $n_t$  个加速核心，则理论上线程级加速可以对计算部分带来额外的  $n_t$  倍性能提升。例如，神威 SW26010 处理器每个核组配有 64 个从核，其可采用 OpenACC 自动并行技术，将主核的任务自动切分运行到从核计算，实现每核组 64 倍的理论加速。另外，神威 SW26010 处理器也提供了 Athread 线程库技术，以允许用户手工对各个加速核心进行细粒度的控制优化。而共享内存架构的多核处理器上的线程级并行则相对较为简单。实现方式上，OpenMP、Pthread 等跨平台的并行库均为实现线程级并行的优选工具。

向量化是处理器内的第三层并行，其利用 SIMD 技术对 CPU 中的数据进行

并行化处理，进一步提高计算的吞吐量。向量由具有相同标量数据类型的多个元素组成，向量长度（vector length,  $vl$ ）则指参与处理的向量元素个数，通常为 2、4、8 或 16 个元素，即：

$$Vector\ length = \frac{size\ of\ vector\ register\ (bits)}{size\ of\ data\ type\ (bits)} \quad (2.26)$$

根据底层硬件提供的技术机制，不同处理器架构支持的向量寄存器宽度也不相同。如 ARM 的 NEON 指令，支持 128 位的向量宽度操作；英特尔 X86 架构则大都支持 256 位或者更高的 512 位的向量宽度操作。256 位宽的向量寄存器可以处理 4 个 64 位的双精度浮点数，在这种情况下，向量长度即为 4。向量化是数据级并行，理论上可以为计算部分带来  $vl$  倍的加速。向量化与线程级并行性不同。向量化试图通过尽可能多地使用硬件上的向量通道来提高性能，即其在单个处理器内核上运行的每个线程之上又提供了额外的并行性。向量化的目的是最大程度地利用每个内核可用的向量寄存器。目前主流处理器都支持 SIMD 操作，且大多数编译器也都具有自动向量化的功能，能够将高级语言代码自动替换为 SIMD 指令。虽然通过自动向量化技术我们可以在一定程度上降低向量化编程难度，增强代码的可移植性，但是有时为了进行更高程度的性能优化，还需要通过与并行算法设计相结合进行手工编程，以设计出高效向量化算法进一步提升可扩展性能。

通过上文在计算部分结合硬件特点进行轻量级并行性的分析与挖掘，可以对高性能多核/众核体系结构算力进行更充分地利用，并最终形成硬件特征与算法精准抽象的进程级-线程级-数据级三个层次并行的深度性能优化链。

**定义 2.10 (轻量级并行性挖掘).** 轻量级并行性挖掘定义为结合硬件并行特点，对应用计算部分由粗粒度到细粒度的并行计算性能优化，使得优化后的计算时间  $T'_{comp}(p, N)$  为：

$$T'_{comp}(p, N) = \frac{1}{P_0} \times T_{comp}(p, N) \quad (2.27)$$

其中， $P_0$  通常是一个大于 1 的常数，即在轻量级并行计算优化下常数倍的加速是有效的优化结果。

根据上文对进程级-线程级-数据级三个层次并行性的分析，我们对  $P_0$  进行进一步地详细剖析：

**定义 2.11 (轻量级并行性能优化系数).** 令  $n_p$ 、 $n_t$ 、 $vl$  分别为采用进程级、线程级、数据级并行所带来的性能提升倍数，则  $p$  个处理器上的轻量级并行性能优化系数  $P_0$  为：

$$P_0 = n_p \cdot n_t \cdot vl。 \quad (2.28)$$

**原则 2.3 (程序性能优化原则).** 通过层次化片上访存特征提取，可从主存级-缓存级-寄存器级三个连续存储层级上由低到高进行读存优化，建立读存性能优化的统一模型；通过处理器上轻量级并行的算力挖掘，可以从进程级-线程级-数据级三个连续并行粒度上由粗到细进行计算优化，对高性能多核/众核体系结构的算力进行充分利用。基于读存模型和算力挖掘，形成硬件特征与算法精准抽象的深度性能优化链可有效实现可扩展性能的优化工作。

证明. 根据式 2.24 中层次化访存特征提取的定义及式 2.27 中轻量级并行性挖掘的定义，我们可得到在  $p$  个处理器上性能优化后的时间  $T_{OPT}(p, N)$  为：

$$\begin{aligned} T_{OPT}(p, N) &= T'_{mem}(p, N) + T'_{comp}(p, N) \\ &= \frac{1}{M_0} \times T_{mem}(p, N) + \frac{1}{P_0} \times T_{comp}(p, N) \\ &= \frac{(HT + MR \cdot AMP)T_{mem}(p, N)}{\alpha\beta} + \frac{T_{comp}(p, N)}{n_p n_t vl}。 \end{aligned} \quad (2.29)$$

其中， $N$  为问题规模， $p$  为处理器的数量， $T_{comp}(p, N)$  为计算时间， $T_{mem}(p, N)$  为访存时间，这些变量在性能优化中是相对不变量。因此，程序的性能优化需对式 2.29 中的其他变量进行分析。如前所述，针对访存项，提高  $\alpha$ 、 $\beta$ ，减少  $MR$ ，即优化 cache 分块、寄存器重用和主存访问的性能，可有效提升整体的访存性能；针对计算项，增大  $n_p$ 、 $n_t$  及  $vl$ ，即提升程序的进程级、线程级、数据级并行性，也可进一步提升计算效率。由于计算项和访存项是累加和计算，因此，二者性能优化的叠加实现可有效减少式 2.29 中优化后的整体时间开销，提高并行算法的可扩展性。□

## 2.4 本章总结

在大规模可扩展应用中，假定在  $p$  个处理器上对物理模型  $m$  ( $m \in M^P$ ,  $m = \arg \max \{\sum_{i=1}^{n-1} C_p^P(m, m_i)\}$ ) 使用并行算法  $A$  进行并行计算，则式 2.30 中总结

了由定理 2.1、原则 2.2 及原则 2.3 指导而得的大规模并行应用的可扩展性多层次协同设计理论：

$$\begin{aligned} T_p(m, A) &= T'_{comp}(p, N) + T_{comm}(p, N) + T'_{mem}(p, N) \\ &= \frac{(pf_s + f_p)B_m T_1}{B_a p n_p n_t v l} + (t_s + m_s t_w) + \frac{(HT + MR \cdot AMP)T_{mem}(p, N)}{\alpha\beta}. \end{aligned} \quad (2.30)$$

对于缓解或解决可扩展性的不连续非线性现象，我们认为要采取分层次协同研究的方法，如式 2.30 和图 2.3 所示。与以往根据应用性能再进行扩展优化不同，我们从应用的物理模型层、核心的并行算法层和硬件的性能优化层三个层次出发，从研究初始就直接面向可扩展性问题。通过对多个层次不连续非线性可扩展现象的描述与研究，我们将算法在有限区间内的并行可扩展性进行了突破与延伸，在多个层次建立了相关的理论解释。

**定理 2.4 (多层次协同设计理论).** 通过对应用模型和算法性能的统一分析，为物理模型层提供理论基础和建模指导；通过并行算法设计实现适应复杂体系结构的新型并行计算和并行通信算法，为可扩展性提供高效的算法层支撑；通过主存级-缓存级-寄存器级层次化访存特征提取和轻量级的进程级-线程级-数据级的多层次并行性挖掘，对硬件算力进行充分利用并形成硬件特征与算法精准抽象的深度性能优化链。物理模型、并行算法、性能优化三层相互支撑、相互融合，可形成大规模并行应用研究的可扩展多层次协同设计方法。

算法层作为核心层，其功能的延伸与拓展可以分别与物理模型集合，构建基于物理数学模型的应用-算法层协同设计方法；与性能模型结合，构建基于算法性能模型的算法-架构层协同设计方法。其中，应用-算法层协同设计的思想可以直接作用于中粒度并行问题；算法-架构层协同设计的思想可以直接作用于细粒度并行问题。由于在面对大规模应用时存在更典型、更复杂的扩展性挑战，我们需要将两层协同设计的思想综合应用于粗粒度的大规模并行问题进行研究。

目前大多数研究仅依赖于某个单一层的扩展优化。在并行应用的扩展性研究中，特别是针对粗粒度的大规模并行问题，单独考虑某一层的可扩展性无法满足整个应用的性能需求，且多层之间也未进行协同分析优化。通过该多层次协同设计方法，我们能够为后续不同并行粒度的可扩展性研究，特别是大规模并行系统上的应用研究，提供理论性的创新和方法性的指导。

## 第3章 细粒度并行：百核量级 Stencil 并行数值算法设计

### 3.1 本章概述

细粒度并行问题一般指向量级或循环级的并行，其一般具有以下特点：首先，并行应用中切分的子任务会再次进行细粒度的分割，使得工作量在处理器之间均匀分配。因此，细粒度并行易于实现负载均衡。其次，细粒度并行问题更适宜在支持快速通信的、规模相对较小的体系结构中进行处理。目前，具有低通信开销、共享内存架构的多核并行系统最适合于细粒度并行问题求解。另外，在细粒度并行问题中，程序的并行性通常较难挖掘。大多数情况下，均由编译器负责细粒度的简单并行实现，而这也限制了扩展性能的进一步提高。

Stencil 算法是 Berkeley 定义的七种高性能计算模板（Dwarfs）中结构化网格计算的代表性算法。作为经典的并行数值算法之一，Stencil 算法在单机多核系统上的向量化计算一直是研究的热点问题。因其在向量化实现中存在固有的数据空间冲突，导致 Stencil 的并行化算法在多核系统上的扩展面临较为严重的不连续和非线性现象。目前，Stencil 的多核优化方法主要分为向量化和缓存分块技术两类，旨在分别提高数据并行性和数据局部性。这两种在不同层次上的正交优化方法带来了耦合优化设计的不连续性。而为了缓解向量化的数据空间冲突，新的算法设计所引入的访存规则和寄存器重排操作，也会导致不同程度的可扩展性能非线性增长。

通过以上对 Stencil 并行数值算法的分析，我们可以发现这是一个典型的细

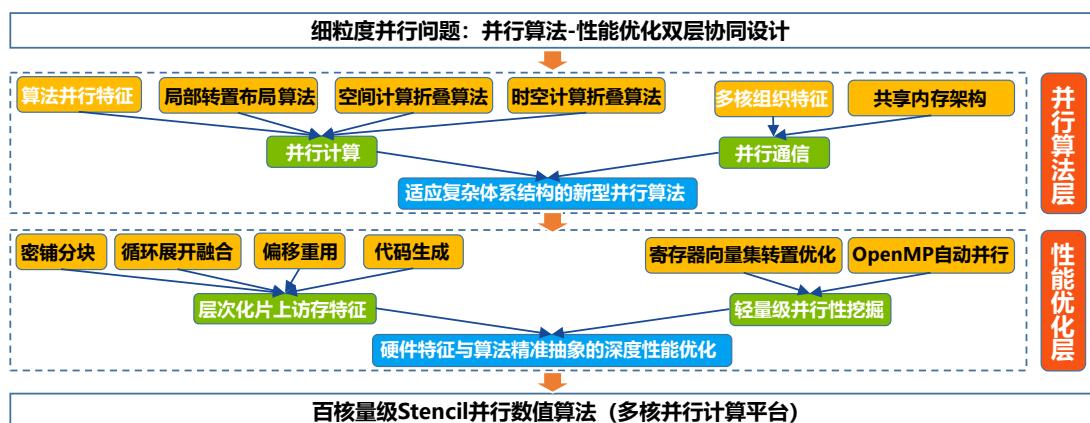


图 3.1 并行算法-性能优化双层协同设计

粒度并行问题。其一，Stencil 并行数值算法大多采用了向量级和循环级的并行。在向量级，需要设计新型向量化算法以缓解数据空间冲突；在循环级，通常使用 OpenMP 制导语句对循环完成多核之间的数据并行。其二，Stencil 并行数值算法采用了单机多核系统进行实现优化。大多数的单机多核系统均属于共享内存的体系结构。系统上计算核心的数量大约在百核量级，且可利用 OpenMP 等多线程技术实现快速的核间通信。最后，由于较难设计高效的 Stencil 向量化算法，目前多数 Stencil 并行数值算法直接使用了编译器优化技术来挖掘向量化加速的潜力。

如图 3.1 所示，根据我们所提出的多层次协同设计方法中的原则 2.2 和原则 2.3，针对 Stencil 并行数值算法的细粒度并行问题，可采取并行算法-性能优化双层协同设计以提升扩展性能。具体地，根据硬件 SIMD 架构的特点，从并行算法层设计适应高性能多核处理器的新型 Stencil 向量化算法；同时根据构建的硬件性能模型，高效地完成算法的实现与优化工作。由于不需要考虑上层的软件应用，采取并行算法-性能优化双层协同设计即可完成对 Stencil 并行数值算法的扩展性研究。不难看出，在可扩展的细粒度并行问题中，其存在的不连续和非线性现象相对容易解决。相关内容我们将在本章中详细展开。

### 3.1.1 引言

Stencil 是在科学计算和工程应用中广泛使用的最重要的算法之一。从物理模拟到机器学习的各个领域，均普遍涉及到 Stencil 算法的相关计算<sup>[25,26]</sup>。Stencil 也是 Berkeley View 中提出的高性能计算领域七大经典算法之一<sup>[24,27,28]</sup>，其作为高性能计算中浮点运算的主要算法而被大量采用为各种计算内核。

一次 Stencil 计算包含了一个预定义的计算模式，该模式沿时间维度迭代地更新  $d$  维空间网格中的每个格点。Stencil 的阶数定义了其在某个方向上邻居格点的依赖关系。如果对称的 Stencil 在一维中的阶数为  $r$ ，在时间步  $t$  时一个格点的值则是上一时间步  $(2r + 1)$  个邻居格点的加权和<sup>[45]</sup>。 $d$  维 Stencil 的实现包含了  $d + 1$  个循环，其中时间维度在最外层循环中进行遍历，所有的网格点则在内部循环中遍历更新。由于 Stencil 是以这种规则的计算结构为主要特点，所以它本质上是一个算术密度低、数据重用性差的带宽受限型算法<sup>[28,46]</sup>。

现有文献已对 Stencil 的性能优化进行了大量细致的研究。传统方法主要集中在向量化或缓存分块技术的优化上，旨在分别提高数据并行性和数据局部性。这两种方法通常被视为在不同层次上的两种正交的优化方法。向量化利用 CPU

中的 SIMD 功能单元并行执行多个数据处理流，而缓存分块则增加对缓存中一组数据的重用。

之前对于 Stencil 计算向量化的工作主要分为两类。第一类是利用相邻格点加权和计算的结合性。具体来说，可以通过重新安排 Stencil 计算的执行顺序以利用计算之间的公共子表达式结果<sup>[47-51]</sup>。因此，可以有效地减少数据加载和存储操作的数量，并以优化后的执行顺序减轻带宽的使用。第二类则是处理数据空间冲突<sup>[52,53]</sup>，这也是向量化中主要的性能限制因素。数据空间冲突是由向量化引起的问题，即一个网格格点的所有邻居出现在同一个向量寄存器中不同的位置上。一个里程碑式的方法是 DLT 方法（Dimension Lifting Transformation，维度布局变换）<sup>[52]</sup>，它是通过执行全局的维度变换改变数据的内存布局来解决空间数据冲突。

为了解决数据空间冲突的问题，现有的解决方案通常是采用两种常见的实现方法。第一种方法是以向量形式直接从内存中加载所有需要的元素（Multiple Loads）。与标量代码的实现相比，这种多次加载向量化方法大大增加了访存量。此外，在每次 Stencil 的计算中，至少存在两次未对齐的内存引用，即寄存器中第一个数据的地址不在 32 字节的边界处。由于 CPU 更支持对齐的数据加载和存储操作，因此该方法将大大降低性能。

第二种方法在 CPU 至内存的数据传输方面则类似于标量代码的实现。它将每个输入元素加载到向量寄存器中仅一次，再通过组装计算时所需的向量进行寄存器内的数据重排。与多次加载向量化方法相比，这种数据重排方法（Data Reorganization）减少了内存带宽的使用，并利用了 SIMD 架构所提供的丰富的数据整理指令。然而，CPU 内部数据整理的执行单元则变成了算法的瓶颈。

在 DLT 方法中，原始的一维数组长度为  $N$  的矩阵被视为大小  $vl * (N/vl)$  的矩阵，其中对于双精度浮点数的 256 位向量寄存器中  $vl=4$ 。然后 DLT 执行一个全局矩阵变换操作并组装输入向量以进行计算边界处的向量。DLT 的维度布局变换克服了输入空间的数据冲突问题。

DLT 方法有如下的一些缺点。首先，如果我们忽略边界的处理，DLT 可以被认为是  $vl$  次独立的 Stencil 计算。因此，当与缓存分块技术结合使用时，数据重用减少了  $vl$  倍。原因是  $vl$  次独立的 Stencil 计算之间不共享数据。另外，显式的全局维度变换操作会增加 DLT 的数据整理开销，这一点在高阶数或大维度的

Stencil 计算中尤其明显。科学计算中一维 Stencil 计算中的循环次数通常较大，足以分摊这种数据整理的开销。但是，用于其他应用（如图像处理）中的 3D 或高维 Stencil 的计算时间通常较小，因此此时的数据整理开销不容忽视。最后，DLT 方法数据整理时原地实现很困难，通常需要开辟额外的数据来存储中间变换的结果，这增加了代码的空间复杂度。

作为实现 Stencil 计算中并行化和数据局部性的关键技术之一，缓存分块（Cache Blocking, 或 Tiling）在几十年来被广泛研究。由于工作集的规模通常大于处理器上的缓存容量<sup>[54]</sup>，空间分块算法被提出，其通过在一个时间步内改变格点的遍历模式来探索数据重用。然而，这种分块技术受限于邻居格点的访问模式<sup>[42,46]</sup>。近年来，时间分块技术也被提出以允许更多的缓存数据在时间维度进行重用<sup>[28]</sup>。

上述两种 Stencil 的计算优化方法通常对彼此的实现没有影响。但是，向量化的数据组织开销可能会降低数据的局部性。此外，之前的大部分工作只关注在缓存级别的分块。这只能减少缓存和内存之间的数据传输量，而 CPU 和缓存之间通信的高带宽需求仍然未解决甚至在向量化时变得更糟。因此，沿时间维度仍有大量重复的 CPU 和缓存的数据传输，造成迭代地在同一个格点上进行冗余计算。

在本文中，我们首先设计了一种新型计算折叠策略来克服向量化的空间数据冲突，并保留数据局部性以同时进行分块。新的向量化方案基于我们优化的快速 CPU 内矩阵转置，该实现算法在数据整理操作和整体指令延迟上都达到了理论下限。与常规方法相比，计算折叠不需要额外的数据组织操作，在较大的问题规模下执行整个向量化流程非常高效。

基于所提出的空间上的计算折叠策略，我们也设计了时间上的计算折叠方法，合并称为时空计算折叠，以进一步减少算术计算的冗余。我们对多个时间步的计算扩展进行了深入分析，将同一点上的冗余操作进行折叠，并为其重新分配新的权重，直接实现寄存器内的多步更新。通过寄存器重用，我们获得了提升的 CPU 内 flops/字节比率，并且中间时间步的计算也被跳过以减轻寄存器数量的压力。时空计算折叠方法也可以推广到任意的 Stencil 计算。此外，我们也利用偏移重用技术来减少最内层循环内的冗余计算，集成缓存分块框架来保证数据的局部性，并设计了一个半自动的代码生成器来简化并行编程。

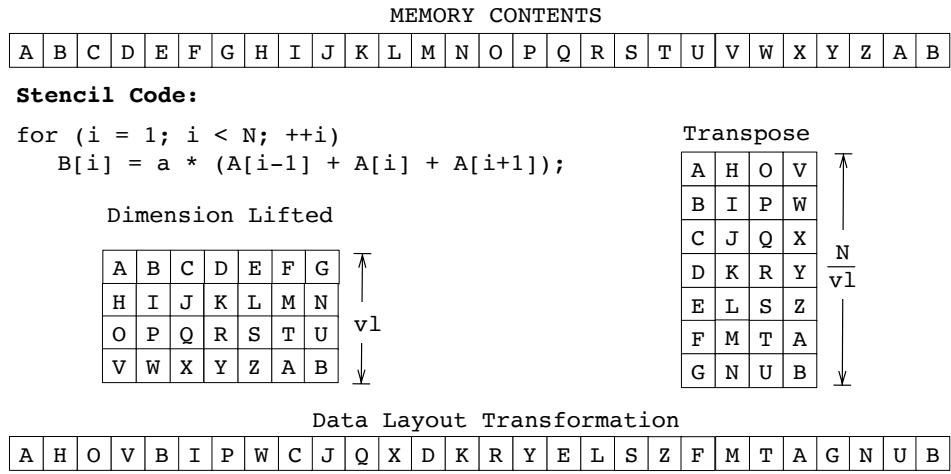


图 3.2 DLT 中空间数据冲突的处理

时空计算折叠方案使用 AVX2 和 AVX-512 向量指令集用于进行一维、二维和三维 Stencil 计算的实验评估。实验结果表明我们的方法与经典向量化方法（自动向量化<sup>[55]</sup> 和数据重组<sup>[28]</sup>）、state-of-the-art 编译器（Pluto<sup>[56,57]</sup> 和 SDS<sup>[28,52,53]</sup>）以及近期高度优化的工作（YASK<sup>[58]</sup> 和 Tessellation<sup>[42]</sup>）相比，具有明显的性能优势。

本章主要贡献如下：

- 我们为 Stencil 计算提出了一种有效的转置布局以消除数据空间冲突，并提出了相应的向量化算法。该算法使用寄存器重用、缓存分块等技术进行了性能优化。
- 我们提出了第二种高效的 Stencil 计算折叠策略和对应的向量化方法。该策略利用快速的寄存器内转置来数据消除空间冲突。
- 基于空间计算折叠策略，我们进一步提出了时空计算折叠方法，并使用偏移重用、密铺分块和半自动代码生成技术进行性能优化。该方法能够在时间迭代空间内减少算术计算的冗余度。
- 我们将我们提出的多种方法推广到各种 Stencil 内核上，并通过实验证明其在多核处理器上与各种高度优化的工作<sup>[28,52,53,55,57,58]</sup> 相比能够获得更优的性能。

### 3.1.2 背景

**空间数据冲突** 如 3.1.1 所述，向量化引起的输入数据的空间冲突是 Stencil 计算中一个关键的性能限制因素。在本小节中，我们以一维 3 点 Stencil 计算为例来

说明这一向量化引起的问题。

由于在大多数现有工作中，向量化仅限于最内层循环<sup>[59]</sup>，故图 3.2 中的代码仅展示了一维 3 点 Stencil 一个时间步的计算。在标量代码执行的第  $i$  次迭代中，它新加载  $A[i + 1]$  和  $B[i]$  到寄存器，并重用先前计算  $B[i - 1]$  所加载的寄存器数据  $A[i - 1]$  和  $A[i]$ 。通过观察 CPU 和内存之间的数据传输，这段代码的行为正类似于常见的数组复制，即 `memcpy` 函数<sup>[60]</sup>。而 CPU 内的计算实现起来是比较简单的，且如循环展开之类的循环优化操作也保持这些性质。

向量化则将一组数据分组到向量寄存器中以进行并行处理。标准的一维 3 点 Stencil 向量化计算连续的格点并将结果保存到输出数组  $B$ 。假设向量寄存器中包含 4 个元素（向量长度  $vl = 4$ ），则向量化代码使用向量运算进行计算并将计算结果  $(B[1], B[2], B[3], B[4])$  输出到一个向量寄存器。

向量化招致的一个问题是空间数据冲突。例如，要计算  $(B[1], B[2], B[3], B[4])$ ，它需要三个向量： $(A[0], A[1], A[2], A[3])$ ， $(A[1], A[2], A[3], A[4])$  和  $(A[2], A[3], A[4], A[5])$ 。元素  $A[2]$  出现在所有的这些向量寄存器，然而却在不同的位置，这被称为空间数据冲突。因此，此时的向量化没有像标量代码执行一样相对应的简单实现。

**常规向量化** 为了解决数据冲突的问题，Stencil 的计算通常采用两种常见的向量化实现方法。

第一种是以向量形式从内存中直接加载所有需要的元素，这也是现代编译器采用的自动向量化方法。与标量代码的实现相比，这种多次加载的向量化方法进一步增加了数据传输量。此外，在代码的每次迭代中，至少有两次未对齐的内存访问，即其第一个数据地址不是 32 字节对齐。由于对齐的数据加载和存储在 CPU 中有着更好的实现性能，这些未对齐的内存引用会相当地影响计算性能。

第二种常规解决方案在 CPU 和内存之间的数据传输方面类似于标量代码的实现。它仅将每个输入元素加载到向量寄存器一次，并通过寄存器间和寄存器内的数据排列指令组装计算时所需的向量。与多次加载方法相比，这种数据重排的方法减少了内存带宽的使用，并利用了大多数 SIMD 架构提供的丰富数据整理指令集的优势。但是，CPU 内部的数据整理的执行单元可能成为瓶颈。

**DLT 方法** 解决空间数据冲突的一个里程碑式方法是 DLT 方法<sup>[52]</sup>。

在 DLT 方法中，长度为  $N$  的原始一维数组被视为大小为  $vl*(N/vl)$  的矩阵，其中  $vl$  是向量寄存器中的向量长度。例如， $vl=4$  表示 256 位向量寄存器中可以存放 4 个双精度浮点数。然后 DLT 执行全局转置。图 3.2 展示了用于 28 个元素的一维数组的 DLT 方法。DLT 的全局数据重新布局克服了输入数据对齐冲突。例如，变换后的布局中的第二个  $vl = 4$  元素形成一个向量 ( $B[1], B[8], B[15], B[22]$ ) 和所有的三个必需的输入向量：左向量 ( $A[0], A[7], A[14], A[21]$ )，中心向量 ( $A[1], A[8], A[15], A[23]$ ) 和右向量 ( $A[2], A[9], A[15], A[23]$ )，它们彼此没有数据共享，并且连续地存储在内存中。DLT 需要组装输入向量来计算边界处的输出向量。

DLT 有以下缺点。首先，如果我们忽略边界处理，DLT 可以被视为  $vl$  次独立的 Stencil 计算。因此，当与分块技术结合时，数据重用减少了  $vl$  倍，因为  $vl$  次独立的 Stencil 计算之间没有数据重用。其次，DLT 在 Stencil 计算之前和之后执行的显式全局数据整理的开销较大。对于科学计算中的一维和二维的 Stencil 计算，时间维度的值通常较大，足以分摊数据整理的开销。但是对于图像处理等其他应用中的三维或高维 Stencil 计算，时间维度较小，这使得数据整理开销不可忽略。最后，通常很难原地实现 DLT 的全局数据整理，故需要使用额外的数据组来存储变换后的数据布局，这增加了代码的空间复杂度。

### 3.1.3 相关工作

对 Stencil 计算的相关优化已经被大量工作广泛地研究<sup>[61–65]</sup>，这些研究可以广义地归类为在提高计算能力、增强数据重用和改善数据局部性等方面的优化。

使用 SIMD 指令进行向量化是提高 Stencil 计算能力的有效方法。Henretty 等人提出了一种 DLT<sup>[52,53]</sup> 方法来克服空间数据冲突，但其代价是全局的数据变换操作。这使得由于其在空间上分离的数据元素而无法流畅地与分块技术相结合<sup>[46]</sup>。本质上，DLT 可以被视为条带挖掘技术（strip-mining，也可认为是一维的分块）和循环外向量化的组合<sup>[53]</sup>。在 DLT 中，循环被转换为深度为 2 的循环嵌套，其中外循环的大小等于向量长度  $vl$ ，而内循环则处理长度为  $N/vl$  的每个子序列。值得注意的是，向量化的相关工作中也引入了条带挖掘技术<sup>[66]</sup>。但是，常规实现思路应是将最内层循环的大小设为向量长度，并用向量代码代替。此外，我们工作中用于向量化的原地矩阵转置算法也已被广泛研究，且  $4\times 4$  的矩

阵转置内核基本上由两个阶段组成。Zekri 等人<sup>[67]</sup>仅对单精度浮点数 *float* 类型实现四个阶段的通道内指令。Hormati 等人将向量寄存器拆分为一些 128 位的指令通道<sup>[68]</sup>，然而用于双精度浮点数 *double* 类型的跨通道指令会产生较长的延迟，通常为 3 到 4 个时钟周期。Springer 等人<sup>[69]</sup>实现利用 SHUFFLE 和 PERMUTE2F128 指令进行双精度浮点数 *double* 类型的两阶段转置，然而它需要 8 个整数作为指令参数。

在数据重用方面也有大量的相关工作进行优化和探索。通过优化执行指令顺序的工作<sup>[48,50,51,70]</sup>可以减少数据加载或存储指令，以减轻寄存器的压力。然而，每个向量中只有单个元素可以被重用。Basu 设计了一个向量代码生成方案，以在计算过程中重用多个向量，但它受限于常数系数和对称的 Stencil 计算<sup>[49]</sup>。YASK<sup>[58]</sup> 基于他们提出的细粒度向量折叠方法，通过使用通用表达式消除和展开来提升数据重用性<sup>[71]</sup>，然而它仅针对高阶三维 Stencil 的计算进行了优化<sup>[48]</sup>。Zhao<sup>[48]</sup> 设计了一种贪心算法来判定具有高重用性的计算部分，并将重新排序的算子通过散射操作分组作为计算的公共输入部分。对于算法未识别的其他部分，它们仍然通过未优化的收集操作进行计算。Rawat<sup>[50]</sup> 利用具有共享叶子的有向无环树图来描述 Stencil 计算，并设计了一种调度算法，通过重新排序 GPU 上的指令来最小化寄存器的使用。Stock<sup>[70]</sup> 提出了一个框架来分析 Stencil 计算中的执行调度顺序，通过使用算数的结合性和交换性重新对操作进行排序以减轻寄存器压力。然而，这些方法并没有减少冗余的浮点计算。公共子表达式消除法 (CSE)<sup>[72]</sup> 被提出以通过重用子表达式来减少同一循环连续迭代中的冗余计算。然而，此方法严重依赖循环展开来查找特定的公共表达式。Deitz 将 CSE 方法扩展为数组子表达式消除法 (ASE)<sup>[73]</sup>，通过创建一个称为邻域表的数据结构进行抽象。抽象生成的邻域表及其子表记录了输入和系数的信息，这样可以重用数据点而不是平面，并允许多维数组的扩展。由于 ASE 通过临时变量重用邻域表的部分和，因此新引入了依赖并阻碍了编译器的指令级并行化。

缓存分块<sup>[74–78]</sup>是探索多个循环嵌套下数据局部性的最强大的模式变换技术之一。用于 Stencil 计算的著名的分块工作包括超矩形分块<sup>[79–82]</sup>，时间裁剪分块<sup>[83–85]</sup>，菱形分块<sup>[56,57]</sup>，缓存无关分块<sup>[45,65,86]</sup>，割裂分块<sup>[53]</sup>和镶嵌密铺分块<sup>[42]</sup>。Wonnacott 和 Strout 比较了许多现有的分块方案<sup>[87]</sup>的可扩展性，这些技术大部分是编译器转换技术。对于 Stencil 计算，现有工作已经提出了各种自动

调优的分块框架<sup>[88–91]</sup>，它们使用各种超矩形切片来实现缓存的数据重用。但是，这些工作中均引入了冗余计算以解决块间的依赖关系，这些依赖关系也阻碍了在不同核心上缓存块的并发执行。Pluto<sup>[56]</sup>能够生成菱形分块平铺一维 Stencil 计算。Bandishti<sup>[57]</sup>将其扩展到更高维度的 Stencil 计算。

## 3.2 并行算法设计：适应多核架构的新型 Stencil 向量化算法

### 3.2.1 局部转置布局算法

在本节中，我们首先讨论了我们的研究动机并阐释了我们方法的独特性。然后我们提出了一种新的转置布局及其对应的 Stencil 向量计算方案。随后我们对转置布局进行了优化，包括在多个时间步的扩展、与缓存分块框架的集成以及用于双精度浮点数的寄存器内快速矩阵转置算法。

#### 3.2.1.1 研究动机

从硬件的角度来看，提高性能的关键是充分利用算术运算指令的执行单元。由于 Stencil 计算在一个时间步的迭代中没有数据依赖性，其唯一的瓶颈是数据准备。相应地，向量化实现的关键技术则是解决数据准备时的空间对齐冲突。

DLT 是一种有效的 Stencil 向量化方法，其可以极大地减少寄存器内的数据重组操作。然而，DLT 向量化的数据格式本质上会损害数据局部性的特性。特别是在 DLT 方法中，同一个向量中的元素物理距离很远，因此它们之间没有数据重用。与此相反，多重加载和数据重组方法则将连续元素加载到同一个向量寄存器中。当它们与时间分块方案集成时，能够实现最佳的数据局部性。这两种方法似乎处于 CPU 中数据重组操作的次数和缓存中数据的复用能力之间未平衡的两个极端。我们在第3.2.1.2节中提出的方法旨在通过利用丰富的寄存器汇编指令集和现代 CPU 中的高效实现来保存寄存器内数据局部性的特性。

基于3.2.1.2节提出的向量化方案，我们然后设计了 3.3.1.2节中寄存器内的时间分块算法。据我们所知，现有工作尚未考虑对 Stencil 计算的多时间步长进行寄存器内数据复用优化。实验结果也展示了我们的算法对性能有显著改进。

此外，将 DLT 方法与缓存分块技术集成是非常低效的，特别是对于 SDSL<sup>[92]</sup>中展示的高维 Stencil 算法。除了数据局部性受损之外，由于边界处理和全局的转换布局，DLT 方法天然不适合时间分块。我们提出的向量化方法为 Stencil 计算的自然数据局部性保留了原始的数据布局，并在第 3.3.1.1节中集成了密铺分

块框架。

现有工作表明，DLT 方法所需的全局矩阵变换非常耗时并且需要额外的内存配额。我们在第 3.3.2.1 节中提出了一种高效的 CPU 内矩阵转置算法，其转置的变换是与 Stencil 的计算同步原地执行，因此开销较小。

### 3.2.1.2 局部转置向量化

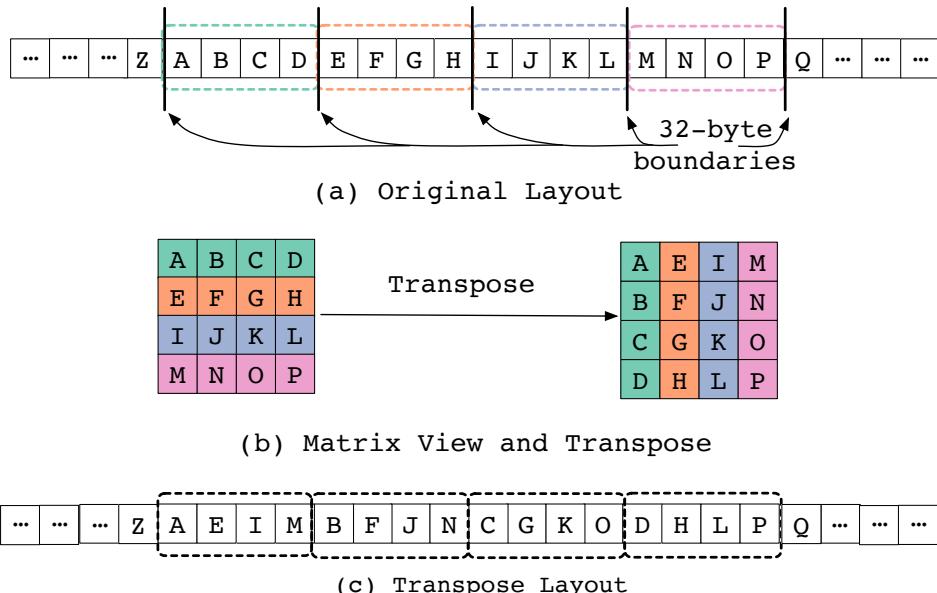


图 3.3 向量长度为 4 的局部转置布局

为了保持数据局部性并减少数据重组操作的数量，我们将矩阵转置应用于连续元素的小子序列。具体来说，类似 DLT 中的维度变换方法，子序列的一维视图也被二维矩阵视图取代。要在矩阵转置后执行向量化，矩阵的列大小应等于向量长度  $vl$ 。设行大小为  $m$ ，则矩阵大小为  $vl * m$ 。原始数据布局  $m = 1$  时，我们的局部转置布局相当于  $m = N/vl$  时的 DLT。在矩阵转置之后，它仍然需要一些数据重组来计算  $m$  个向量中的第一个和最后一个。例如，如果  $m = 1$ ，则必须为计算每个输出向量做好左右邻居向量的原始数据准备。这也是在数据局部性和上述的数据准备之间的性能平衡。

决定  $m$  的大小有以下几个考虑因素。首先， $m$  应该足够大，以通过中间向量的实际算术运算隐藏对第一个和最后一个向量进行数据重组的开销。假设一个 Stencil 运算的阶是  $r$ ，那么中间向量的算术运算次数则是  $(2r+1)*(m-1)+1$ 。数据重组的次数是  $4r$ ，因为第一个和最后一个向量需要  $2r$  个组装向量并且它们中的每一个都需要两个重组指令，这将在下文进行详细阐释。因此  $m$  应至少为 3。

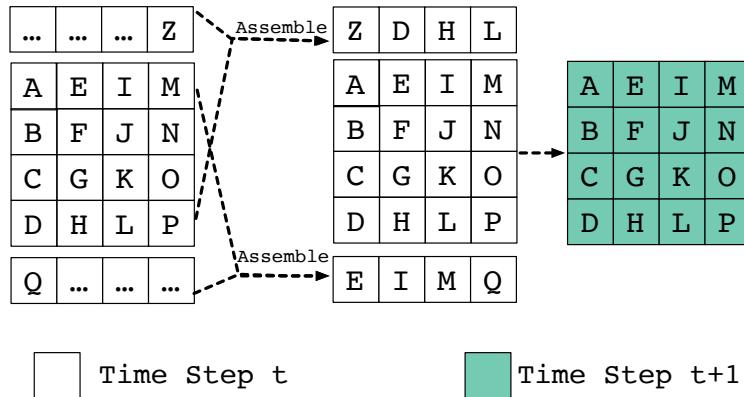


图 3.4 局部转置布局的 Stencil 计算

需要注意的是，这个限制与阶数  $r$  无关。其次，为了避免像 DLT 格式那样存储转置数据所需的额外数组，最好在 CPU 中完成矩阵的转置。因此， $m$  个输入向量和附加的辅助向量必须保存在 CPU 的向量寄存器中。在这项工作中，我们统一设置  $m = vl$ 。最后一个原因是转置大小为  $vl * vl$  的矩阵更容易在现代处理器上实现。下文 3.3.2.1 节我们也将提出一种高效的矩阵转置算法，用于大小为  $vl * vl$  的寄存器内矩阵转置。

图 3.3 展示了向量长度为 4 的一维 Stencil 的转置布局。每个长度为  $vl * vl$  的子序列的矩阵转置在 Stencil 计算之前和之后执行。参与运算的  $vl$  个向量称为**向量集**。值得注意的是，在实现中，每个向量集始终与 32 字节内存边界对齐。

更新一个一维 3 点的 Stencil 向量集需要两个组装向量。一个是其第一个向量的左依赖向量，另一个是其最后一个向量的右依赖向量。图 3.4 描述了这两个向量的数据重组操作。向量集的第一个向量是  $(A, E, I, M)$ ，它的左依赖向量是  $(Z, D, H, L)$ ，其数据分散存储在向量集的两个相距较远的向量中，即  $(*, *, *, Z)$  和  $(D, H, L, *)$ 。这两个向量由 blend 指令组合，然后通过 permute 操作以实现循环右移，完成组装向量。

如图 3.4 所示，得到组装向量之后，向量集的 Stencil 计算就比较简单了。通过使用开销较低的寄存器内矩阵转置和每个向量集平均两个组装向量，我们实现了一套有效的 Stencil 向量化方法。此外，与 Multiple Loads 方法相比，所提出的向量化方案避免了数据的冗余访存；而与 Data Reorganization 方法相比，则避免了频繁的寄存器重组开销。局部转置的布局也可以以相同的方式应用于高阶和多维 Stencil 计算。

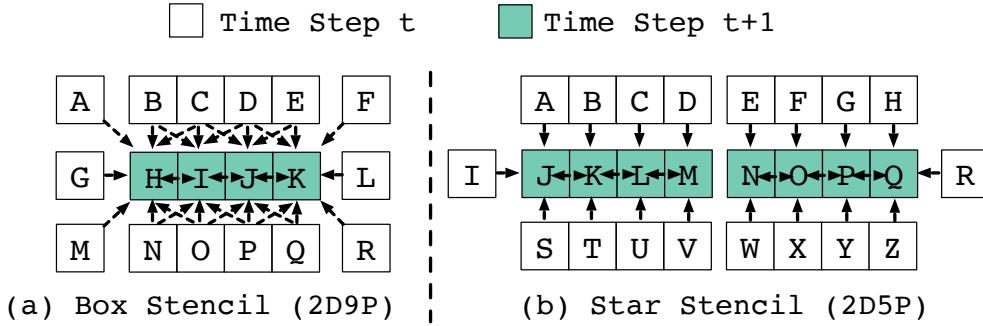


图 3.5 二维 Stencil 向量化中的空间数据对齐冲突

### 3.2.2 空间计算折叠算法

在本节中，我们首先讨论了现有方法存在的主要缺点。然后我们提出了一种新型计算折叠策略及其相应的向量化过程，以消除加载数据时的空间冲突。表 3.1 中总结了 Stencil 计算折叠算法中相关符号的定义。

#### 3.2.2.1 研究动机

Stencil 计算的常规向量化方法有两个主要缺点。首先，在单个向量计算中存在重复的内存访问。如图 3.5(a) 所示，为计算向量  $(H, I, J, K)$  下一个时间步的更新，需要向量  $(A, B, C, D), (B, C, D, E)$  与  $(C, D, E, F)$  作为输入，这些输入向量的共有元素出现在不同向量的不同位置。因此，这将导致至少存在两次未对齐的向量加载指令。

表 3.1 Stencil 计算折叠算法中相关符号定义

术语和符号	定义
$N$	一维 Stencil 问题的规模
$vl$	向量寄存器长度（双精度浮点数）
<b>VS</b>	向量集，即由 $vl$ 个向量组成参与算法运算的基本单元
$m$	时间展开因子
$t$	时间步
$E$	Stencil 的标量算术表达式
$C(E)$	Stencil 的标量算术表达式中使用的算术指令数
$P$	集合 $C(E)$ 与集合 $C(\mathbb{E}_\Lambda)$ 的势之比
$w$	Stencil 预定义的初始权值
$\lambda$	Stencil 重定义的分配权值
$v$	向量寄存器变量

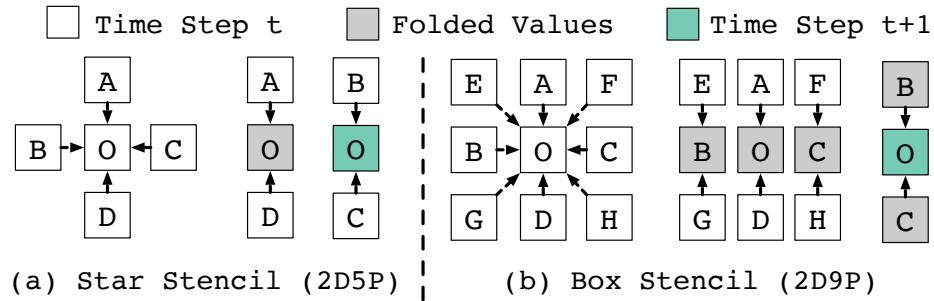


图 3.6 二维标量 Stencil 计算中的计算折叠策略

其次，常规的向量化方法阻碍了连续计算之间的数据重用。在 Stencil 算法的标量计算中，两次连续的计算可以共享 6 个输入格点。因此，每次新格点的计算只需要进行 3 次 load 和 1 次 store 指令。使用常规向量化方法后，引入的空间数据对齐冲突导致需要 9 次 load 和 1 次 store 指令。

此外，随着 Stencil 的阶或维度的增加，这些缺点也会更加突出。例如，对于一个 2D25P 的 Stencil，每次标量格点的计算可以重用 20 个格点，新读取 5 个格点，并存储一个更新后的格点。而向量化后的代码只能够重用 5 个向量，产生 20 次向量 load 和 1 次向量 store。因此，数据冗余度比 2D9P 的 Stencil 计算更严重，即  $20/5 > 9/3$ 。

### 3.2.2.2 标量空间计算折叠

为了保持数据的局部性并减少数据重组操作的数量，我们提出了一种折叠策略来减少空间对齐冲突。基于对星形 Stencil 的观察，我们发现数据对齐冲突只出现在最内层空间上的循环。图 3.5(b) 展示了一个 2D5P 的 Stencil 内核。两次连续的在向量  $(J, K, L, M)$  和向量  $(N, O, P, Q)$  上的 Stencil 计算，外层空间维度上不存在数据对齐冲突，即外层循环中的输入向量  $(A, B, C, D)$  和  $(E, F, G, H)$  并不存在共有格点。

我们的想法是先沿外层空间维度来计算 Stencil，然后进行数据布局转换以消除数据对齐冲突，之后再进行单位步长的循环内计算。沿着某一维度上的计算我们就称为一次折叠。图 3.6(a) 解释了在标量计算下 2D5P 的 Stencil 计算折叠方案。为了更新格点  $O$ ，首先沿着非单位步长维度（列方向）对邻居格点  $A$  和  $D$  进行折叠。折叠后的格点  $O$  的颜色被标记为灰色。然后，折叠的格点  $O$  被单位步长维度（行方向）上的格点  $B$  和  $C$  再次更新。注意，最后的折叠是通过一次

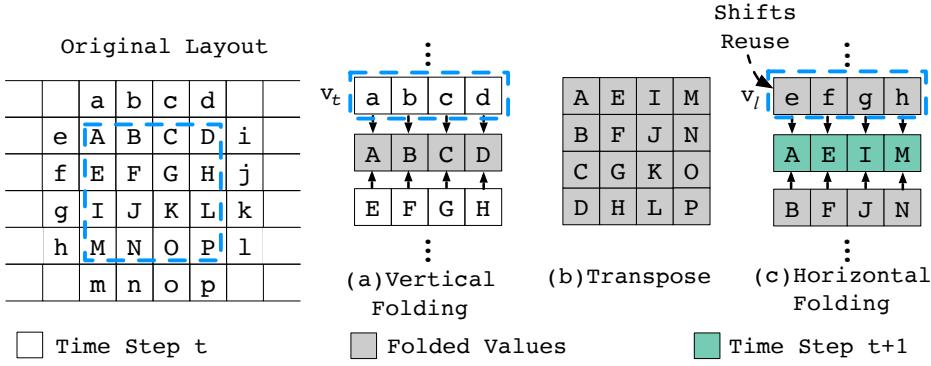


图 3.7 2D9P 盒形 Stencil 计算折叠向量化

转置的数据布局来执行的，其中格点  $B$  和  $C$  可以被视为非单位步长维度上的邻居。更新后的在时间步  $t+1$  上的格点  $O$  值被标记为绿色。

如图 3.6(b) 所示，对于盒形 Stencil 的计算，其所有空间维度都存在数据对齐冲突。尽管如此，折叠策略仍与处理星形 Stencil 计算时相似。图 3.6(b) 展示了 2D9Pstencil 的折叠方案。额外地，在拐角处的四个格点也要进行垂直折叠，例如格点  $E$ 、 $G$  折叠到  $B$  点，格点  $F$ 、 $H$  折叠到  $C$  点。然后将三个折叠后的格点值  $B$ 、 $O$  和  $C$  再次折叠成最终的结果。我们的折叠策略很容易扩展到高维的 Stencil 计算。至于一维的 Stencil 计算，我们可以将大小为  $N$  的数组视为  $(vl) * (N/vl)$  的二维矩阵，此时在外层维度上就没有数据依赖关系，因此，只需要在转置布局上做一次折叠操作即可完成 Stencil 计算。

### 3.2.2.3 空间计算折叠向量化

为了实现折叠策略的向量化，我们将每  $vl$  个向量进行分组，作为一个向量集。向量集是 Stencil 向量化方案的基本处理粒度。算法 1 给出了基于折叠策略的二维 9 点 Stencil 向量化算法伪代码。

算法入口包含在 STENCIL 函数中，它在时间维度进行遍历。两个内层空间循环的索引按  $vl = 4$  前向累加。因此， $x$  循环的每次迭代处理一个行块，而  $y$  循环则将每个行块切割拆分为向量集。算法第 10 行到第 15 行对每个行块的第一个向量集  $VS_1$  进行边界计算。首先， $VS_1$  由 LoadVS 函数加载到寄存器中（第 10 行）。图 3.7 中左子图被虚线包围的 16 个元素  $A$  到  $P$  展示了一个算例。通过 LoadVec 函数，向量集的顶向量和底向量分别被加载到寄存器  $v_t = (a, b, c, d)$ （第 11 行）和寄存器  $v_b = (m, n, o, p)$ （第 12 行）中。由于原始的数据布局具有垂直方向无冗余

**算法1** 基于计算折叠的 Stencil 向量化算法 (2D9P) $T \% 2 = 0, NX \% 4 = 0, NY \% 4 = 0$ 


---

```

1: function VFOLDVS( $\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_5$ )
2:   for  $i = 1 \rightarrow 4$  do
3:      $\mathbf{v}_{i-1} \leftarrow \text{VFOLD}(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$ 
4:   end for
5:    $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4 \leftarrow \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ 
6: end function

7: function STENCIL()
8:   for  $t = 1 \rightarrow T$  do
9:     for  $x = 1 \rightarrow NX$  by 4 do
10:       $\mathbf{VS}_1 \leftarrow \text{LoadVS}(A, x, 1)$ 
11:       $\mathbf{v}_t \leftarrow \text{LoadVec}(A, x - 1, 1)$ 
12:       $\mathbf{v}_b \leftarrow \text{LoadVec}(A, x + 1, 1)$ 
13:      VFOLDVS( $\mathbf{v}_t, \mathbf{VS}_1, \mathbf{v}_b$ )
14:      Transpose( $\mathbf{VS}_1$ )
15:       $\mathbf{v}_l \leftarrow \text{FoldandSetVec}(A, x, 0)$ 
16:      for  $y = 5 \rightarrow NY$  by 4 do
17:         $\mathbf{VS}_2 \leftarrow \text{LoadVS}(A, x, y)$ 
18:         $\mathbf{v}_t \leftarrow \text{LoadVec}(A, x - 1, y)$ 
19:         $\mathbf{v}_b \leftarrow \text{LoadVec}(A, x + 1, y)$ 
20:        VFOLDVS( $\mathbf{v}_t, \mathbf{VS}_2, \mathbf{v}_b$ )
21:        Transpose( $\mathbf{VS}_2$ )
22:        HFOLDVS( $\mathbf{v}_l, \mathbf{VS}_1, \mathbf{VS}_2[0]$ )
23:        Store( $B, \mathbf{VS}_1, y - 4, x$ )
24:         $\mathbf{v}_l \leftarrow \mathbf{VS}_1[3]$ 
25:         $\mathbf{VS}_1 \leftarrow \mathbf{VS}_2$ 
26:      end for
27:       $\mathbf{VS}_2[0] \leftarrow \text{FoldandSetVec}(A, x, y)$ 
28:      HFOLDVS( $\mathbf{v}_l, \mathbf{VS}_1, \mathbf{VS}_2[0]$ )
29:      Store( $B, \mathbf{VS}_1, y - 4, x$ )
30:    end for
31:     $A \leftrightarrow B$ 
32:     $NX \leftrightarrow NY$ 
33:  end for
34: end function

```

---

的特点，因此利用 VFOLDVS 函数在  $\mathbf{VS}_1$  上直接进行一系列的垂直折叠 (第 13 行)，如图 3.7(a) 中  $(a, b, c, d) + (A, B, C, D) + (E, F, G, H) \rightarrow (A, B, C, D)$ 。现在，向量集包含了四个使用同一列邻居格点更新后的折叠向量，转置后 (第 14 行) 进行

图 3.7(b) 中的水平折叠。

最内层的循环将 Stencil 进行流水线计算。 $y$  循环的每次迭代都加载一个新的向量集  $VS_2$ , 第 17 到第 21 行与第 10 行到 14 行相同, 均是进行垂直折叠。基于转置后的布局, 需要对之前的向量集  $VS_1$  利用 HFOLDVS 函数进行横向折叠以完成最后的计算, 如图 3.7(c) 中  $(e, f, g, h) + (A, E, I, M) + (B, F, J, N) \rightarrow (A, E, I, M)$ 。虽然横向折叠的函数在操作上类似于 VFOLDVS, 但在计算逻辑上, 邻居格点是沿着每一行进行收集的。因此, 我们将横向折叠的函数写作 HFOLDVS (第 22 行)。注意对向量集  $VS_1$  进行横向折叠时所需的右向量正是向量集  $VS_2$  转置后的第一个向量。同样对称地, 我们重用向量集  $VS_1$  的最后一个向量 (第 24 行) 作为下一个向量集  $VS_2$  计算的左向量。

每个向量集使用转置后的布局 (第 23 行到第 29 行) 存储到  $x - y$  交换的内存地址。因此, 如果我们忽略算法 1 当中的计算细节, 每次最外层的时间循环本质上是使用寄存器在全局数据空间上做了一次转置。图 3.8 展示了一个示例。时间循环的下一个迭代作用于转置后的数组, 该数组的指针和大小被交换 (第 31 行和第 32 行), 垂直折叠和水平折叠在物理空间上执行相对的操作。但是, 如果 Stencil 内核沿对角线对称, 则代码仍然是正确的。否则, 可以简单地对算法进行修改, 即手动交换 VFOLDVS 和 HFOLDVS 在偶数时间步上的迭代。总之, 我们的折叠策略使转置操作的次数减少了一半, 因为该算法不需要每次迭代后必须使用相同的数据布局。

这种空间计算折叠方法同样适用于高阶和高维的 Stencil 计算。所提出的向量化方案与多 load 向量化方法相比, 避免了数据的冗余加载, 而与数据重组向量化方法相比, 则减少了向量间/向量内的数据重排操作。一般来说, 对于一个

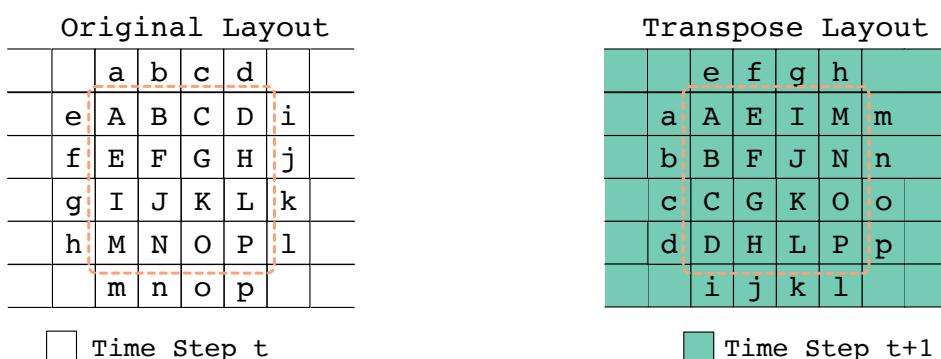


图 3.8 二维 Stencil 的内存存储布局

表 3.2 求解 Jacobi 类型 Stencil 不同向量化方法寄存器行为统计（每向量）

Kernel	1D5P			2D9P			3D27P			1D-Heat			2D-Heat			3D-Heat				
Operation <sup>1</sup>	C.	L.	S.	P.	C.	L.	S.	P.	C.	L.	S.	P.	C.	L.	S.	P.	C.	L.	S.	P.
AutoVec. <sup>2</sup>	5	5	1	0	9	9	1	0	27	27	1	0	3	3	1	0	5	5	1	0
Reorg.	5	1	1	4	9	3	1	6	27	9	1	18	3	1	1	2	5	3	1	2
S-Fold	5	1	1	0	5	1.5	1	1	15	4.5	1	3	3	1	1	0	5	1.5	1	1
S&T-Fold	4.5	0.5	0.5	0	5	1	0.5	0.5	10	2.5	0.5	1.5	2.5	0.5	0.5	0	5	1	0.5	1
																		10	1.5	0.5

<sup>1</sup> 为使表达清楚，算术运算、向量加载、向量存储和寄存器内数据重组操作分别简记为 C., L., S. 和 P.。

<sup>2</sup> 自动向量化，数据重组，空间计算折叠，以及时空计算折叠等方法也分别进行简记为 AutoVec., Reorg., S-Fold 和 S&T-Fold (下同)。

$vl \times vl$  的矩阵，寄存器内的转置只需要  $vl \log(vl)$  次通道内指令。因此，平均每个向量的计算仅会产生  $\log(vl)$  次重排操作，开销也与 Stencil 的阶数或维度无关。

表 3.2 列出了算术运算、向量加载、向量存储和寄存器内数据重组操作的数量对比。对于二维 9 点 Stencil 的空间计算折叠 (S-Fold)，算法 1 在最内层循环的一次迭代中，更新一个向量集需要加载 6 个向量，存储 4 个向量，并进行总共 5 次算术运算。对于其他的 Stencil 类型，我们也可以很容易地得到类似的结果。与自动向量化和数据重组方法相比，空间计算折叠算法可以减少所有 Stencil 的数据访问和高维盒形 Stencil 的运算指令。

### 3.2.3 时空计算折叠算法

#### 3.2.3.1 研究动机

一般情况下，在 Stencil 计算的下一轮新时间步开始更新之前，所有格点均只更新了一次（一个时间步长）。大多数现有的工作均只利用了分块技术<sup>[28,42,57]</sup>来减少主存和缓存之间的数据传输，而在  $m$  个连续的时间步之间并没有实现寄存器内的数据重用，其中  $m$  被称为时间展开因子。若沿着时间维直接简单地进行寄存器重用，则会在时间步  $t$  到  $t + m$  的计算中产生大量的中间结果，这会导致严重的寄存器溢出现象。

现有的工作和上述讨论的直接实现代表了寄存器在时间迭代空间中重用的两个极端。我们的想法是寻求一种平衡，使得能够在时间维上消除算法计算的冗余，同时减轻寄存器压力。为了实现这一目标，我们扩展了空间计算折叠算法，直接在寄存器中将计算格点更新  $m$  个时间步长。基于一系列的收益性分析，我们也提出了时空计算折叠算法并进行了详细的阐述。结合偏移复用等技巧，我们也将对时空计算折叠进行了优化，以获得更大的性能增益。

Naive approach for updating 2 time steps on the center point A(0,0)

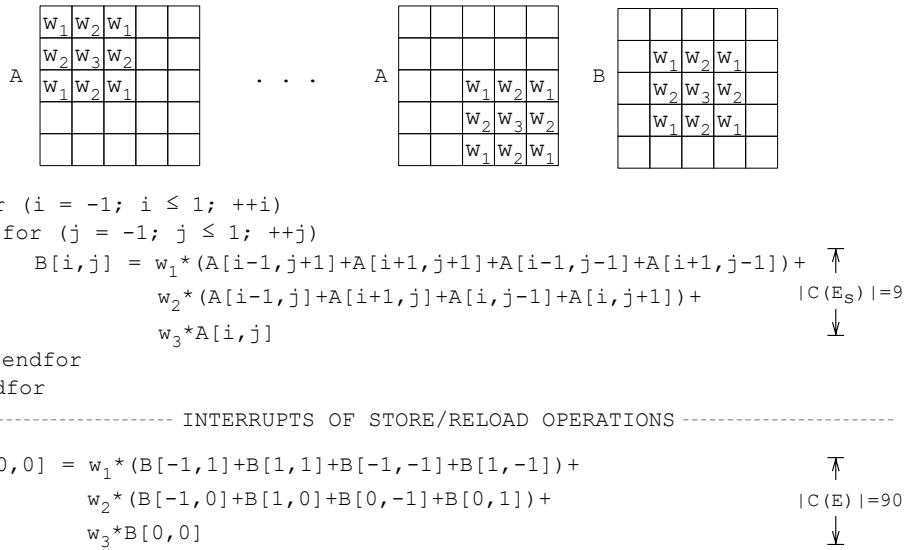


图 3.9 二维 9 点盒形 Stencil 初始时标量算术表达式图解（时间展开因子  $m = 2$ ）

### 3.2.3.2 标量时空计算折叠

图 3.9展示了在中心格点上时间展开因子为 2 时，典型的二维 9 点 Stencil 的标量算术表达。公式 3.1 中集合  $C(E)$  被定义用于描述在表达式  $E$  中进行 2 个时间步更新所需的算术运算指令 (加，乘，乘加，等等) 的数量。

$$C(E) = \bigcup_s \{ \langle g, w_g \rangle | \langle g, w_g \rangle \in C(E_s) \} \quad (3.1)$$

对于图 3.9中的原始表达式，首先将每个配有 8 个邻居的格点全部更新到  $t + 1$  个时间步，然后将这些更新后的值从寄存器写回到内存中。当下一次  $t + 2$  时间步迭代开始时，这些状态为  $t + 1$  时间步的格点再次从内存中被重新加载以完成新一轮的计算。为了获得一个中心格点上的 2 个时间步的更新，共计 10 个子表达式的运算指令均被计到集合  $C(E)$  中。在每个子表达式  $E_s$  中，中心格点均被分配了一个预定义的权重  $w_g$ ，然后得到一个 9 路加法运算的结果。由于每个子表达式中使用了 9 个不同的邻居格点，因此对于表达式  $E$ ，我们得到了  $|C(E)| = 10 \times |C(E_s)| = 90$ 。值得注意的是，在不同的子表达式中发生了多次对同一个格点的冗余算术操作，并且在计算过程中  $t + 1$  时间步的值所需进行的存储/重新加载操作产生了额外的中断开销。

Optimized approach for updating 2 time steps on the center point  $A(0,0)$

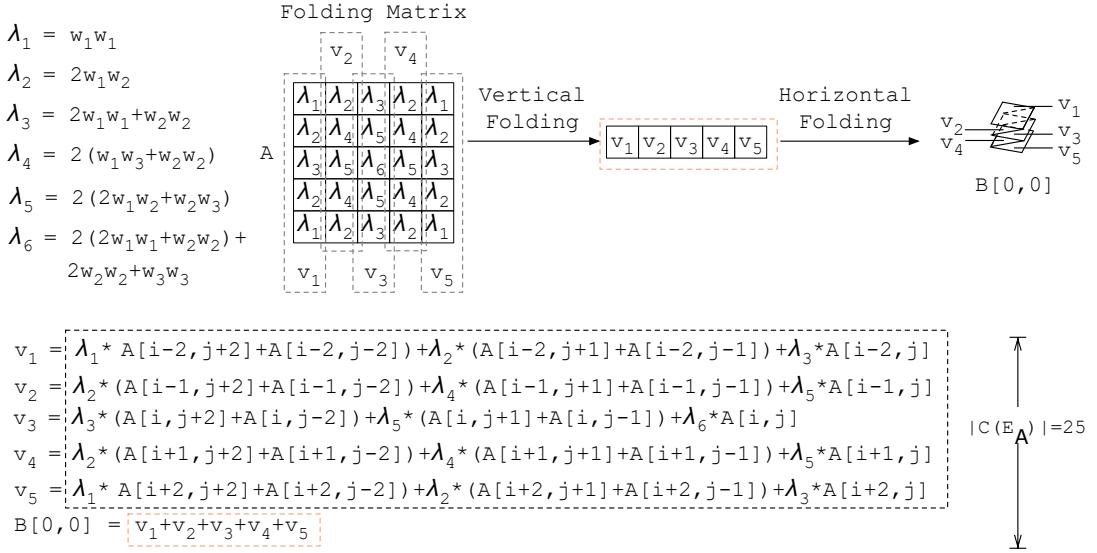


图 3.10 二维 9 点盒形 Stencil 折叠后标量算术表达式图解（时间展开因子  $m = 2$ ）

对于图 3.10 中优化后的表达式，根据  $m$  个时间步的展开项对权重进行重新分配。新的算术表达式  $E_\Lambda$  由新的权重  $\lambda$  组成的折叠矩阵决定。首先对同一列的 5 个邻居格点以权重  $\lambda$  进行垂直折叠，然后将得到的 5 个折叠值再进行水平折叠。每个中心格点的邻居格点  $g$  以权重  $\lambda_g$  进行计算折叠，构成邻居点集，因此，图 3.10 中集合以公式 3.2 得到其基数  $|C(E_\Lambda)|$  为 25。

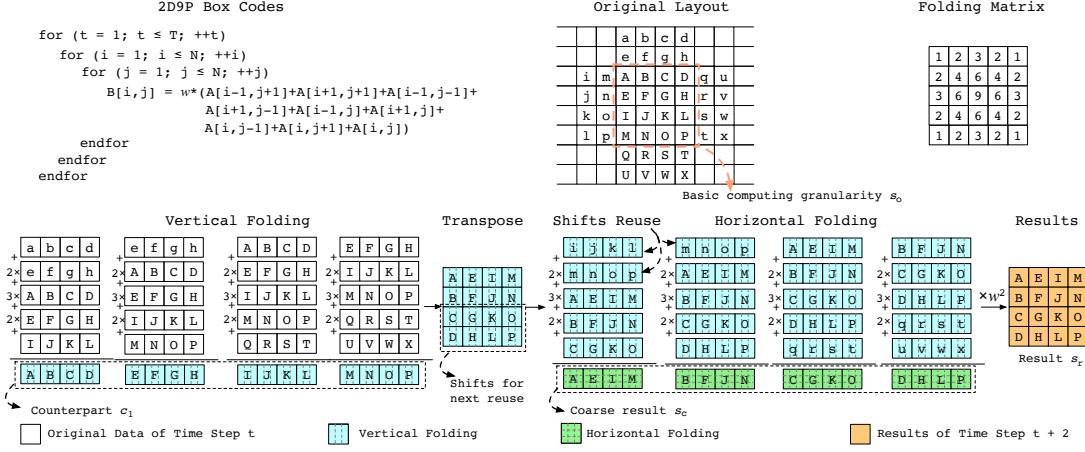
$$C(E_\Lambda) = \{\langle g, \lambda_g \rangle \mid \text{grid } g \text{ used in } E_\Lambda \text{ weighted with } \lambda_g\} \quad (3.2)$$

式 3.3 中定义了收益指数。一次有效折叠意味着两个集合基数的比值至少超过一个预设的阈值  $\theta$ （一般地， $\theta \geq 1$ ）。在本例中，由式 3.3 得到的收益指数为  $P(E, E_\Lambda) = 90/25 = 3.6$ 。此外，在  $E_\Lambda$  中也完全消除了存储和重新加载等操作产生的中断开销。

$$P(E, E_\Lambda) = \frac{|C(E)|}{|C(E_\Lambda)|} \geq \theta \quad (3.3)$$

### 3.2.3.3 时空计算折叠向量化

在本小节中，我们以 2 个时间步 2D9P 盒形 Stencil<sup>[28,42]</sup> 为例来说明图 3.11 中的时空计算折叠方法的细节，这也是在 3.2.3.2 节中讨论的优化算术表达式后相对

图 3.11 二维 9 点盒形 Stencil 时空计算折叠向量化过程（时间展开因子  $m = 2$ ）

应的向量化实现过程。

**数据准备** 图 3.11 描述了示例 Stencil 的代码和数据准备。基于前面 3.2.2 节的策略，时空折叠的基本粒度也被构造为  $4 \times 4$  的方形格点集，记为  $s_o$ 。它们分别在时间步  $t$  时刻从缓存中加载到四个寄存器  $v_0$  至  $v_3$  中。

**垂直折叠** 与 3.2.2.3 节中的操作类似，首先执行垂直折叠以收集同一列中的邻居格点。经垂直折叠，从  $s_o$  计算而来的折叠后的方形格点集被称为一个向量集副本。通常，计算  $m$  个时间步最多需要产生  $m+1$  个向量集副本。式 3.4 中描述了如何通过垂直折叠得到每一个向量集副本，其中  $\lambda$  是重新分配的权重；上标  $n$  是向量集副本的编号。

$$v_i^{(n)} = \sum_{t=-m}^m \lambda_t^{(n)} \cdot v_{i+t} \quad (3.4)$$

例如，图 3.11 所示的折叠矩阵将第一个向量集副本的权重重新分配为  $\lambda^{(1)} = \{1, 2, 3, 2, 1\}$ 。根据式 3.4，第一个向量集副本  $c_1$  中的每个  $v_i^{(1)}$  分别是对  $v_{i-2}, 2v_{i-1}, 3v_i, 2v_{i+1}$ ，和  $v_{i+2}$  进行累加而得。

**水平折叠** 完成垂直折叠后，对下一步的水平折叠首先进行一次本地转置，然后进行水平折叠以收集同一行的折叠值：

$$v_i^* = \sum_{t=-m}^m v_{i+t}^{(c-|t|)}。 \quad (3.5)$$

其中， $c$  是向量集副本的总数。由于重新分配给其他两个向量集副本  $c_2$  和  $c_3$  的权重有  $\lambda^{(2)} = 2\lambda^{(1)}=\{2, 4, 6, 4, 2\}$  and  $\lambda^{(3)} = 3\lambda^{(1)}=\{3, 6, 9, 6, 3\}$ ，式3.5可被展开为：

$$\begin{aligned} v_i^* &= v_{i-2}^{(1)} + v_{i-1}^{(2)} + v_i^{(3)} + v_{i+1}^{(2)} + v_{i+2}^{(1)} \\ &= v_{i-2}^{(1)} + 2v_{i-1}^{(1)} + 3v_i^{(1)} + 2v_{i+1}^{(1)} + v_{i+2}^{(1)}。 \end{aligned} \quad (3.6)$$

因此，对于一个方形格点集  $s_0$ ，通过利用  $c_1$  即可完成一个粗略结果  $s_c$  的计算。

**加权转置** 水平折叠之后，需要完成一次加权转置以完成最后的计算。通常情况下，Jacobi 类型的 Stencil 算法是需要使用两个数组来实现<sup>[28,56]</sup>，对应地，它们分别存储奇数和偶数时间步的值。因此，这里的本地转置可以进行优化省略，即其可以隐藏在水平折叠的转置中，交替地变换到原始的数据布局。经过向量化的时空折叠计算，式 3.3中的  $|C(\mathbb{E}_\Lambda)|$  进一步降为 9，理论上可以得到收益指数  $P(E, \mathbb{E}_\Lambda) = 10$ 。

表 3.2中最后一行也列出了时空计算折叠 (S&T-Fold) 的寄存器行为分析。对于二维 9 点的 Stencil 内核，时空计算折叠加载 9 个向量、存储 4 个向量用于两个时间步的向量集更新。平均每个向量的更新需要 4.5 次算术运算，且向量存储指令减少了一半、向量加载操作也相应减少。

### 3.3 性能优化方法：多核架构高效访存与计算优化

#### 3.3.1 层次化访存优化

本节中，我们将对 Stencil 并行数值算法进行由缓存级别到寄存器级别，再到代码生成的访存优化。其中，前两段为针对局部转置算法的性能优化方法，后三段则为针对时空计算折叠算法的性能优化方法。

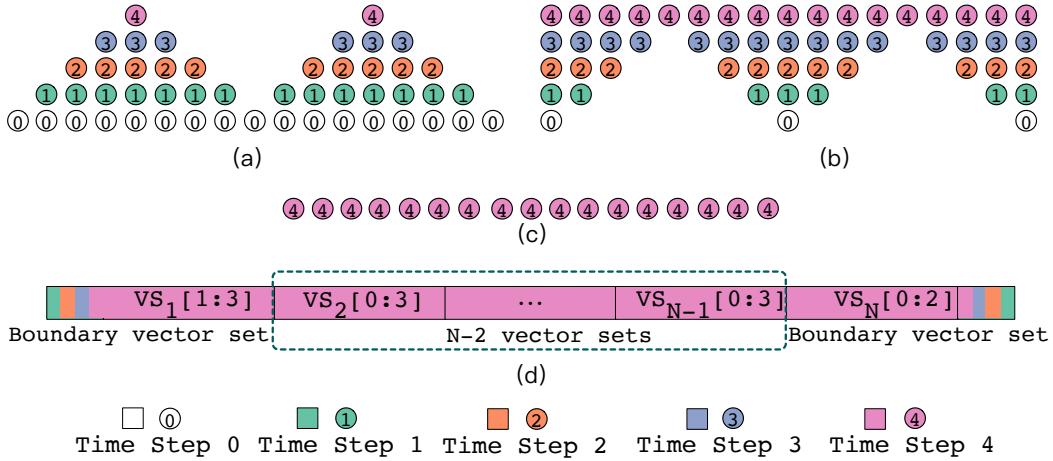


图 3.12 基于局部转置布局的一维 Stencil 分块框架 (2 个时间步)

### 3.3.1.1 分块框架集成

向量化和分块是两种针对不同层级的正交优化方法。向量化在执行级别使用数据并行性提高了计算能力，而分块则利用缓存级别的数据重用减少访存带宽。我们提出的局部转置布局使得向量化技术成为 Stencil 的空间对齐冲突解决方案，而循环展开融合则进一步提高了 CPU 内向量寄存器级别的数据重用能力。在本小节，我们将设计新的集成策略，实现我们的向量化方法和分块框架的正交优化。

密铺分块<sup>[42]</sup>可以视为通过利用各种形状的分块在数据迭代空间中进行密铺镶嵌。图 3.12(a) 和图 3.12(b) 展示了一维 Stencil 的密铺分块框架，迭代空间交替地由三角形和倒三角形分块进行密铺。因此，并发执行由两个阶段构成，这两个阶段首先在给定时间范围内的每个三角形中开始，随后则在相同时间范围内同时再更新倒三角形。不同时间步的更新用不同的颜色来区分，每个元素在时间维度上的状态在图 3.12 中用数字表示。对于图 3.12(a) 中的算例，每个三角形的新状态包含  $(0,1,2,3,4,3,2,1,0)$ ，其中中心格点更新了 4 个时间步，其他邻居格点更新的步数与中心格点的距离成正比。为了使所有格点以相同的时间步更新，相邻三角形邻近的一半构成新的倒三角形，格点更新为状态  $(4,3,2,1,0,1,2,3,4)$ 。如图 3.12(c) 所示，通过倒三角形和三角形分块叠加的投影，所有的格点更新到了 4 个时间步。集成密铺分块框架后，可以在给定的时间范围内利用不同分块并发执行，而无需其他冗余计算。

应用局部转置布局进行分块唯一的问题是每个块的两处边界计算。三角形分块的计算在空间上是一个“收缩”的过程，处理的格点范围随着时间步的推进

而减少。同样，在倒三角形分块的计算过程中也会发生“扩张”的过程。由于物理上相邻的格点彼此分散存储在同一个向量集中，边界处向量集的分块计算将过于复杂。因此，如图 3.12(d) 所示，我们将边界处的向量集在分块时转换回计算前的原始格式，并采用简单数据重组方法对其进行处理，以此来简化收缩和扩张的分块过程。集成的密铺分块也可以应用于多维 Stencil 的计算。对于  $d$  维 Stencil，迭代空间中的分块包含  $d + 1$  个阶段。与图 3.12 中的一维 Stencil 算例类似，阶段  $i$  的数据空间由  $tiles_i$  ( $0 \leq i \leq d$ ) 进行分块。 $tiles_1$  是一个超立方体（通常，在一维中是线段，二维中是正方形，三维中是立方体）。 $tiles_{i+1}$  则是通过从相邻的  $tiles_1$  沿某些维度拆分的子块重新组合构建的。将局部转置布局应用于高维 Stencil 与一维的情况完全相同，因为转置布局仅影响的是单位步长的维度。

### 3.3.1.2 循环展开融合

通常，Stencil 计算受限于其输入数据的空间对齐冲突，所有元素在下一个时间步的迭代开始之前仅能更新一次。尽管可以利用缓存分块技术<sup>[28,42,57]</sup> 来减少主存和缓存之间的数据传输，但在连续的时间循环之间仍未实现寄存器级别的数据重用。因此，CPU 内的 flops/byte 比率受到 Stencil 计算模式的限制。据我们所知，在现有工作中还没有探索寄存器中多个时间步的计算实现。

基于第 3.2.1.2 节中提出的局部转置布局，我们设计了一种寄存器内时间循环展开融合策略。它在时间步  $t$  加载一个元素，并在将其存储到内存之前直接更新  $k$  个时间步长。 $k$  称为展开因子。正常的向量化运算对应于  $k = 1$  的情况。如果  $k > 1$ ，则执行相当于将时间循环展开  $k$  次并对循环进行融合。因此，理论上这将使 CPU 内的 flops/byte 比率提高  $k$  倍。本小节将以一维 Stencil 示例，该算法也适用于多维 Stencil 的计算。

总的来说，算法的思想比较简单。当更新一个向量集后，我们将更新后的结果保存在寄存器中，处理并更新下一个相邻的向量集。然后，可以使用当前向量集更新后的值作为输入，前向反馈到上一个向量集将其沿时间维度再次进行更新。

算法 2 展示了我们的 CPU 内循环融合算法的伪代码。COMPUTE 函数接收一组向量集和由 ASSEMBLE 函数计算的组装向量，它按一个时间步更新当前向量集中的元素。需要注意的是，这是一次原地更新，寄存器中上次的值将被覆盖。

主函数遍历由展开因子  $k$  融合的时间循环。为简单起见，我们假设总时间步

---

**算法 2 循环融合算法**


---

```

1: function ASSEMBLE( $\mathbf{v}_a, \mathbf{v}_b$ )
2:    $\mathbf{v}_c = \text{mm256\_blend\_pd}(\mathbf{v}_a, \mathbf{v}_b)$ 
3:    $\mathbf{v}_c = \text{mm256\_permute4}\times64\text{-pd}(\mathbf{v}_c)$ 
4:   return  $\mathbf{v}_c$ 
5: end function

6: function COMPUTE( $\mathbf{v}_{left}, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4, \mathbf{v}_{right}$ )
7:    $\mathbf{v}_0 \leftarrow \text{ASSEMBLE}(\mathbf{v}_{left}, \mathbf{v}_4)$ 
8:    $\mathbf{v}_5 \leftarrow \text{ASSEMBLE}(\mathbf{v}_1, \mathbf{v}_{right})$ 
9:   for  $i = 1 \rightarrow 4$  do
10:     $\mathbf{v}_{i-1} \leftarrow \text{STENCIL}(\mathbf{v}_{i-1}, \mathbf{v}_i, \mathbf{v}_{i+1})$ 
11:   end for
12:    $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}_4 \leftarrow \mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ 
13: end function

14: function MULTIPLETIMESTEPS( $\mathbf{VS}_{1:k}, \mathbf{vrl}_{0:k-1, k}$ )
15:   for  $j = k + 1 \rightarrow N$  do
16:      $\mathbf{VS}_{k+1} \leftarrow \text{Load the } j\text{-th Vector Set}$ 
17:     for  $i = k \rightarrow 1$  do
18:        $\mathbf{vrl}_i \leftarrow \mathbf{VS}_i[3]$ 
19:       COMPUTE( $\mathbf{vrl}_{i-1}, \mathbf{VS}_i[0 : 3], \mathbf{VS}_{i+1}[0]$ )
20:     end for
21:     Store  $\mathbf{VS}_1$ 
22:     for  $i = 1 \rightarrow k$  do
23:        $\mathbf{VS}_i \leftarrow \mathbf{VS}_{i+1}$ 
24:        $\mathbf{vrl}_{i-1} \leftarrow \mathbf{vrl}_i$ 
25:     end for
26:   end for
27: end function

28: function MAIN( )
29:   while  $t < T$  do
30:     Booting computation.
31:     MULTIPLETIMESTEPS( $\mathbf{VS}_{1:k}, \mathbf{vrl}_{0:k-1, k}$ )
32:     Epilogue computation.
33:      $t+ = k$ 
34:   end while
35: end function

```

---

$T$  可以被  $k$  整除。在 while 循环的每次迭代中，每个元素都沿时间维度前向更新  $k$  个时间步。初始化代码段则准备好流水更新所需的数据。图 3.13 的上半部分展

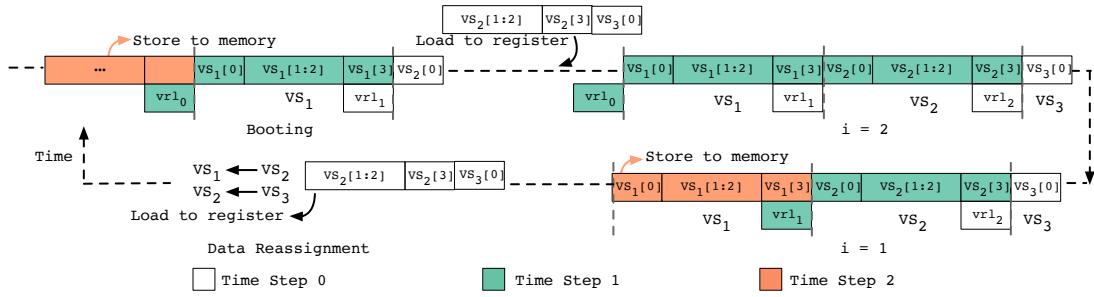


图 3.13 循环融合后 Stencil 计算 (2 个时间步)

示了流水计算后  $k = 2$  的情况。向量集  $\mathbf{VS}_1$  至  $\mathbf{VS}_k$  已经分别对应地更新了  $k - 1$  到 0 次。由于 COMPUTE 函数原地覆盖的属性，它需要将边界向量最后一次更新的值保存到每个向量集的左侧，记为  $\mathbf{vrl}_i$ 。例如，图 3.13 中  $\mathbf{vrl}_i$  和  $\mathbf{VS}_i[3]$  分别存储同一向量在时间步  $t - 1$  和  $t$  的值。

MULTIPLETIMESTEPS 函数将所有寄存器中的向量集从右至左依次更新一个时间步长。同时，它保留了  $\mathbf{vrl}$  中最右边向量的旧值。在每次循环结束时， $\mathbf{VS}_1$  则已经更新了  $k$  个时间步并写回存储在内存中，下一个循环就可以流水执行了。每次循环仅需访存和存储一组  $vl * vl$  个元素，完成  $k * vl * vl$  个 Stencil 计算。如上所述，循环融合算法将 CPU 内的 flops/byte 比率提高了  $k$  倍。

从算法中，我们可以看到其需要  $k$  个向量集和  $k$  个额外的组装向量来进行时间循环的展开融合，即除了系数向量寄存器之外，总共有  $(vl + 1) * k$  个寄存器。在现代 CPU 中，通常可用的向量寄存器的数量是  $vl * 4$ ，其中  $vl$  是一个寄存器中可承载的双精度浮点数的容量。因此，在我们的算法中我们设置了  $k = 2$ 。该算法还有另一个优点。按照惯例，Jacobi 类型的 Stencil 是用两个数组交替实现的，其存储的分别是在奇数和偶数时间步的值。如果我们设置  $k = 2$ ，则输入和输出值都是偶数时间步。因此，输入数据空间可以进行完全地重用并使整个计算原地运行，减缓存储空间的压力。

### 3.3.1.3 密铺分块

向量化和分块是两种正交的方法，他们分别作用在不同的存储层级上。分块技术作用在缓存级别实现数据重用，而向量化则是在寄存器级别利用数据并行性来提高计算能力。在我们时空计算折叠算法的工作中，我们选择密铺分块框架<sup>[42]</sup> 来利用缓存级别的数据重用。它提供了一个简单而清晰的并行框架和轻量级循环条件，以允许进行精细的内核优化。后续实验也展示了集成密铺分块和并

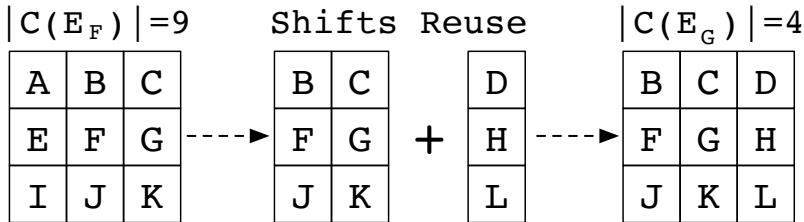


图 3.14 偏移重用图解

行化后我们方法的性能。

### 3.3.1.4 偏移重用

图 3.14 描述了数据空间中两个相邻格点 F 和 G 之间 1 个时间步 Stencil 的标量计算。从图 3.14 中可以看到，在从格点 F 到格点 G 连续的 Stencil 计算中，有一部分的计算结果中间值是可以被重用的，这使得式 3.3 中的收益指数可以再增加 2.25 倍。对于图 3.11 中提出的向量化方法，在每次迭代中，经过转置后的向量集副本  $c_1$  的最后两个向量可以作为偏移在计算下一个向量集时重用。因此，通过利用在上一轮中收集的中间计算结果作为下一次计算的输入，可以减少跨向量集之间的向量加载指令，有助于进一步提高性能。类似地，在图 3.7 中，最后一步折叠的值也可以暂时驻留在寄存器中，以避免冗余的算术计算和重复的数据传输。

### 3.3.1.5 代码生成

通过对各种情况进行数学分析，可以观察到所有的 Stencil 都需要在向量寄存器上执行很多相同且独立的操作，这对在不同架构上的有效实现提出了挑战。我们注意到，其实一些循环可以通过代码生成器进行自动创建来减少代码大小并实现高性能的可移植性。因此，我们将外部循环的算法骨架抽象为半自动的代码生成器，Stencil 表达式中的重复模式由代码生成器包装并生成。对于 Stencil 内核的特征描述则是在脚本文件中手动定义，并作为参数显式地传给代码生成器。代码生成器的输出则集成到整体的 C 代码中，以简化 Stencil 计算的并行编程。

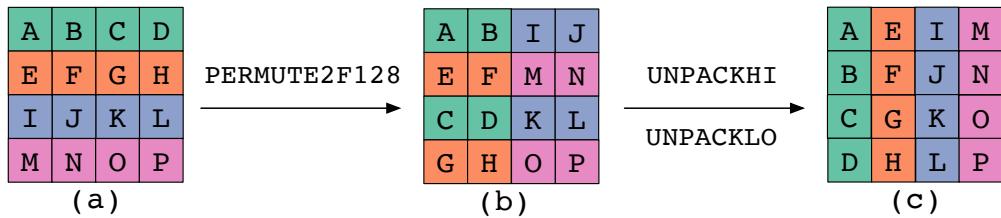


图 3.15 使用 AVX2 指令的寄存器内双精度浮点数向量集转置

### 3.3.2 轻量级并行

#### 3.3.2.1 寄存器向量集转置优化

与执行全局维度变换的 DLT 工作<sup>[52]</sup>不同，我们仅需要在整个过程中对每个向量集进行两次原地转置。完成大小为  $vl * vl$  的矩阵转置所需内存操作的下限是  $vl \log(vl)$ ，例如，对于  $vl = 4$ ，需要至少 8 条数据重组指令。在现代 CPU 的架构中，这 8 条指令可以在 8 个时钟周期内完成。然而，如 3.1.3 节中的相关工作所述，现有 CPU 内双精度矩阵转置算法的实现基本上采用的都是跨通道指令，这增加了 25% 的开销。

图 3.15 展示了我们设计的改进算法，其中高延迟的指令被单时钟周期的低延迟指令替换。整个算法可分为两个阶段。在第一阶段，距离为 2 的两个向量对，如  $(A, B, C, D)$  和  $(I, J, K, L)$ ，分别使用 permute2f128 指令进行数据交换。在第二阶段，相邻的向量对，例如  $(A, B, I, J)$  和  $(E, F, M, N)$ ，通过 unpackhi 或 unpacklo 指令分别再次进行数据交换。新转置算法的总开销由此减少到 8 个时钟周期。类似地，使用 AVX-512 指令集实现的转置算法则包含了三个阶段，其中最后一个阶段也由通道内的低延迟指令设计实现。

#### 3.3.2.2 OpenMP 自动并行

OpenMP 是一种用于共享内存并行系统的多线程程序设计方案，其提供了对并行算法的高层抽象描述，特别适合在多核 CPU 机器上的并行程序设计。编译器根据程序中添加的 pragma 指令，自动将程序并行处理，降低了并行编程的难度和复杂度。

OpenMP 采用了 fork-join 的执行模式。初始时系统中只存在一个主进程，当需要进行并行计算的时候，该进程会派生出若干个分支线程来执行并行任务。当并行代码执行完成之后，分支线程会合，并把控制流程交给单独的主进程。默认

情况下，并行区内线程数应等于系统中核心的个数。当派生的线程数量不超过核心数量时，我们也可以简单地将线程数认为是进程数。在本章中，我们均对单机多核的 Stencil 并行数值算法采用了 OpenMP 技术来实现轻量级的并行。因此，我们在共享内存的机器上使用了 OpenMP pragma parallel 的制导语句进行加速。制导语句 parallel for 用于生成一个并行域，并将计算任务在多个线程之间分配，从而加快并行计算的速度。

### 3.4 实验评估

为了高效地实现提出的各种向量化方案，我们采用高级向量扩展（AVX）指令集进行实现。AVX 是一种用于微处理器的指令集，由英特尔公司在 2008 年 3 月首次提出，它依赖于 x86 系列指令集架构<sup>[93]</sup>。AVX2 将大多数整数运算扩展到 256 位，并引入了融合的乘加 (FMA) 运算指令。现在，AVX-512 指令集使用新的增强向量扩展 (EVEX) 前缀编码将 AVX 扩展到了 512 位进行操作。在本节中，我们使用 AVX2 和 AVX-512 指令集在用于实际应用的各种 Stencil 上评估我们提出的向量化方法。

#### 3.4.1 实验配置

**硬件架构** 本节的实验结果是在两种不同的硬件机器上获得的。第一台机器由两个主频为 2.30GHz 的英特尔至强金牌 6140 处理器组成，它拥有 36 个物理内核，组织成两个卡槽。其配有 DDR4 DRAM 和 6 个内存通道，支持 127.5GB/s 的峰值内存带宽。每个内核包含一个 32KB 的私有一级缓存、一个 1MB 的私有二级缓存和共享的 24.75MB 三级缓存。该机器支持 AVX-512 的指令集扩展，能够以 SIMD 方式对 8 个双精度浮点数据进行运算，达到每核心 73.6GFlops 的

表 3.3 实验中 Stencil 参数配置

Type	Pts	Problem Size	Blocking Size
1D-Heat	3	10240000×1000	2000×1000
1D5P	5	10240000×1000	2000×500
APOP	6	10240000×1000	2000×500
2D-Heat	5	5000×5000×1000	200×200×50
2D9P	9	5000×5000×1000	120×128×60
Game of Life	8	5000×5000×1000	200×200×50
3D-Heat	7	400×400×400×1000	20×20×10
3D27P	27	400×400×400×1000	20×20×10

理论峰值性能（总计 2649.6GFlops）。另一台机器由两个主频 2.35GHz 的 32 核 AMD EPYC 7452 处理器组成。每个内核包含一个私有的 32KB 一级缓存、一个 512KB 的二级缓存和共享的 16MB 三级缓存。该机器支持 AVX2 的 SIMD 指令集和 4812.8GFLOPs 的理论峰值性能。同时，配有的 8 个 DDR4 内存控制器提供达 204.8 GB/s 的内存带宽。

**基准方法** 我们首先在3.4.3小节使用三种经典向量化方法（自动向量化<sup>[55]</sup>、数据重组<sup>[55]</sup> 和 DLT<sup>[52]</sup>）进行了串行无分块实验，以研究单个进程不同方法间的绝对性能对比。然后，在3.4.4小节中我们引用了一些较新的代表性相关工作（SDSL<sup>[53]</sup>、Pluto<sup>[57]</sup>、YASK<sup>[58]</sup> 和 Tessellation<sup>[42]</sup>）在多核架构进行进一步地并行分块实验比较。需要注意的是，SDSL 和 Tessellation 是分别在 DLT 和自动向量化方法上使用了缓存分块技术的两个扩展工作。最后，在第3.4.5小节中，我们与现在的一些高度优化的工作和最先进的 Stencil 编译器（SDSL<sup>[53]</sup>、Pluto<sup>[57]</sup>、YASK<sup>[58]</sup> 和 Tessellation<sup>[42]</sup>）进行了实验对比，对可扩展性进行了全面评估。在上述所有的实验中，我们均采用了 OpenMP 技术来实现并行。因此，我们在共享内存的机器上使用了 OpenMP pragma parallel 的制导语句进行加速。所有的程序均使用 ICC 19.0.3 编译器进行编译，使用“-O3 -xHost -qopenmp -ipo”选项进行编译及并行优化。表 3.4列出了不同 Stencil 优化工作所采用的各种具体的向量化、缓存分块及并行化技术。

表 3.4 不同 Stencil 优化工作所采用的向量化、缓存分块及并行化技术

Benchmarks	Vectorization	Register Tiling	Cache Blocking	Parallelization
SDSL <sup>[53]</sup>	DLT <sup>[52]</sup>	-	Split tiling	OpenMP
Tesselation <sup>[42]</sup>	AutoVec.	-	Tessellate tiling <sup>[28]</sup>	OpenMP
Our <sup>1</sup>	Locally Trans.	U.&J.	Integrated tessellate	OpenMP
Pluto <sup>[57]</sup>	AutoVec.	-	Diamond tiling <sup>[56]</sup>	OpenMP
YASK <sup>[58]</sup>	Vector Folding <sup>[71]</sup>	-	Loop tiling	OpenMP
Our <sup>2</sup>	Spatial Folding	Temporal Folding	Tessellate tiling	OpenMP

\* 为了使表达更简洁，基于转置布局的向量化方法简记为“Our<sup>1</sup>”，其中提出的局部转置布局、循环展开融合以及分块框架集成分别缩写为 Locally Trans.、U.&J. 和 Integrated tessellate；基于计算折叠的向量化方法简记为“Our<sup>2</sup>”，其中空间计算折叠和时空计算折叠分别缩写为 Spatial Folding 和 Temporal Folding。

**计算内核** 实验中计算所用的 Stencil 内核详细参数如表 3.3所示，其中包括了三个星形 Stencil 内核 (1D-Heat、2D-Heat、3D-Heat) 和三个盒形 Stencil 内核 (1D5P、

2D9P、3D27P)，分别对应相关工作<sup>[28,53]</sup>。星形和盒形 Stencil 可以代表大量的 Stencil 内核，其计算模式中的邻居格点是对称的。此外，我们也收集了一系列在真实应用中使用的经典 Stencil 内核<sup>[28,57,94]</sup>：

- APOP 是一个一维 3 点 Stencil 内核，其由两个不同的输入数组来计算美国股票的定价。
- Game of Life 是由康威提出的一个元胞自动机，每个格点的更新依赖于它周围所有的 8 个邻居格点。

默认情况下，相关工作中的实验总时长均设置在 200~1000 个时间步。因此，我们在实验中使用了一个较大的值 1000。总时长的影响也将在 3.4.3 节中讨论。每个 Stencil 内核的其他参数也依据相关工作进行微调，以保证所有方法都能达到各自的峰值性能。

**性能指标** 大多数 Stencil 计算的相关工作（如 DLT<sup>[92]</sup>, SDSL<sup>[95]</sup>, Tessellation<sup>[28,42]</sup>, 和 Pluto<sup>[57]</sup> 等）均使用算术性能（Stencils/s 或 Flop/s）作为度量指标。同样地，在本节中，我们也主要使用算术性能进行实验评估。通过采用式 3.7 中定义的每秒 Stencil 计算数 (Stencils/s)，我们对实验结果进行了定量分析。其中， $N_x$ 、 $N_y$ 、 $N_z$  是每个维度的 Stencil 尺寸； $T$  为总时间步长； $time$  为运算时间。由于 Stencil 计算是内存密集型算法，因此内存吞吐量也是衡量 Stencil 计算性能的参考指标，相关内容也在第 3.4.5 节中进行了讨论。

$$\text{stencils per second} = \frac{N_x \cdot N_y \cdot N_z}{time} \times T \quad (3.7)$$

### 3.4.2 数据准备的影响

在本小节中，我们首先研究了数据准备的开销对 Jacobi 类型的 Stencil 在时间维度上的影响。对于 DLT 方法，主要的数据准备开销是全局维度变换，而我们的计算折叠策略则是寄存器内的转置。我们采用了表 3.3 中的参数配置，并在保留和取消数据准备的情况下分别运行 Stencil 计算。图 3.16 展示了总时长从 1 到 1024 个时间步 Stencil 计算时间所占的百分比。

如图 3.16 所示，我们的方法可以持续地获得较高的计算密度。寄存器内的数据转置对整体性能影响不大。例如，在一维 5 点 Stencil 中其仅占据了可以忽略的 0.04% 的时间开销，在二维 5 点 Stencil 中则仅占据了 6.54% 的时间开销。而 DLT

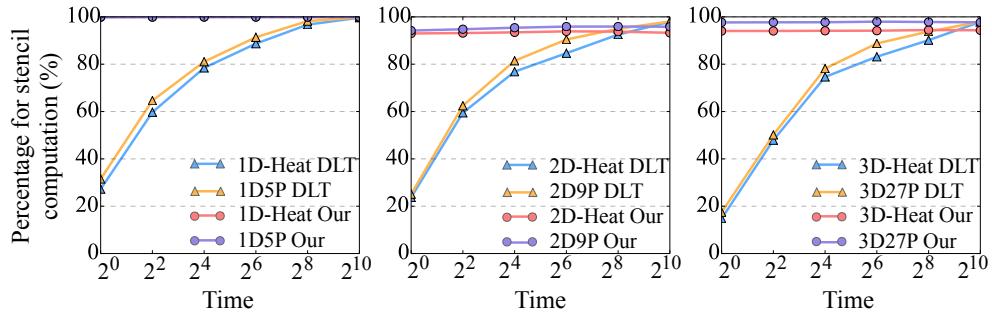


图 3.16 无分块串行实验中 Jacobi 类型 Stencil 计算时间百分比

方法对维数和时间大小具有明显的敏感性。在较低的维数或较高的时间步长下，使用 DLT 算法的计算百分比可以达到较好的结果。这主要是因为数据布局在较低维数下相对简单，而且转换成本也被较长的时间步长所均摊。这也说明了在使用 DLT 算法进行 Stencil 计算时，总时间步长需要足够大以平摊 DLT 算法所带来的数据准备开销。因此，在接下来的实验中，我们的总时长设置均使用了更大的时间步长 1000，以充分发挥 DLT 方法的性能。除了总时长的要求外，DLT 还需要一个额外的数组来存储维度变换后的数据布局，这也增加了存储的压力。

### 3.4.3 串行无分块实验

在本小节，我们接着展示了问题规模从一级缓存扩展到主存，不同方法的单进程性能结果。本小节的实验并未采用缓存分块技术，以研究向量化在不同存储级别上带来的绝对性能的改进。Multiple Loads 和 Data Reorganization 方法代表了现代编译器和近期工作中普遍使用的一类自动向量化方法<sup>[28,42]</sup>。DLT 是 Henretty<sup>[52]</sup> 设计的维度变换方法。本小节中参与计算的 Stencil 内核是经典的 Jacobi 类型的一维 Heat 方程。所有的方法都是通过手工编写的代码来实现，并通过适当的策略进行优化，以确保比较的公平性。例如，我们首先都将内存访问对齐到 256 位边界，然后最内层的循环按四步进行循环展开。此外，虽然我们的计算折叠策略支持原地实现，但仍像其他方法一样，使用两个数组分别存储奇数和偶数时间步的值。

**基于转置布局的向量化** 图 3.17展示了我们的方法与其他三种方法的性能比较。基于不用的总时间步长  $T$ ，结果在在两个子图中进行了说明。可以看出，我们实现 2 个时间步的循环展开融合方法在两个实验中都明显优于其他方法，这证明

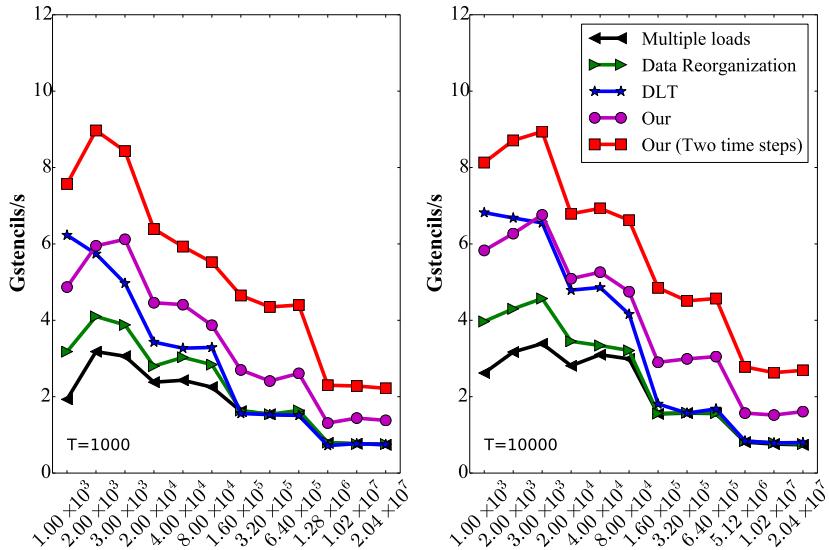


图 3.17 串行无分块实验中基于转置布局算法的向量化与其他方法的绝对性能比较（不同总时间步长的结果分别展示在不同子图中）

了 flop/byte 比率有明显的提升。并且在大多数情况下，即使没有应用 2 个时间步的循环展开融合，我们局部转置布局的方法也比 DLT 方法取得了更好的性能结果。另外，在一级缓存内 Stencil 大小为 1000 时，我们的方法性能稍有下降，这主要归因于与此对比的 DLT 方法中小规模的维度变换操作对性能影响较小。由于存在冗余的数据加载而引起大量的访存开销，Multiple Loads 方法表现出最差的性能。

为了进一步研究总时间步长  $T$  的影响，我们将默认的时间步迭代值增加十倍至  $T = 10000$ ，如图 3.17(b) 所示。可以看到， $T = 10000$  时的性能趋势仍与图 3.17(a) 中的结果基本一致。然而，我们方法的性能在一级缓存中略落后于 DLT，这种性能异常主要是由于过长的时间步长稀释了维度变换的成本。值得注意的是，在图 3.17 中，随着问题规模的增加，一级缓存中 DLT 的性能逐渐下

表 3.5 串行无分块实验中基于转置布局算法的向量化与其他方法在不同存储层级上的性能提升对比

Methods	Auto Vec.	Reorg.	DLT	Our	Our (2 steps)
L1 Cache	1.00x	1.28x	2.06x	2.16x	3.13x
L2 Cache	1.00x	1.11x	1.37x	1.67x	2.07x
L3 Cache	1.00x	1.01x	0.95x	2.02x	2.92x
Memory	1.00x	1.00x	1.01x	1.97x	2.96x
Mean	1.00x	1.11x	1.35x	1.98x	2.81x

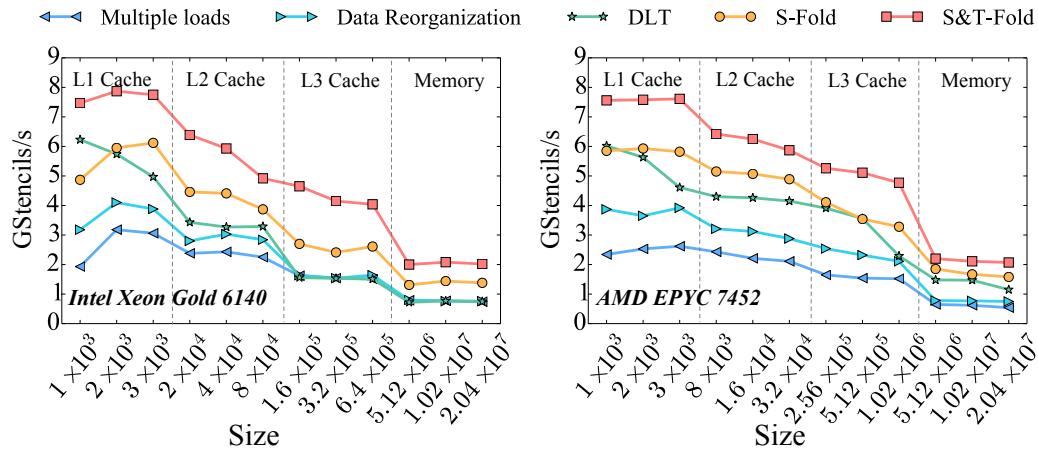


图 3.18 串行无分块实验中基于计算折叠算法的向量化与其他方法的绝对性能比较（不同机器的结果分别展示在不同子图中）

降。由于在数据访问速度较高的一级缓存中 DLT 的性能开销主要来自于维度变换，这也表明之后分块技术在 DLT 方法上的应用会存在潜在的性能瓶颈。

**基于计算折叠的向量化** 图 3.18展示了我们的方法与其他方法的性能对比。在两台机器的结果分别用两个子图加以说明。可以看出，我们的时空折叠方法在两台机器上的性能都明显优于其他方法。在大多数情况下，空间计算折叠也比 DLT 实现了更好的性能。由于冗余向量加载带来的访存开销，Multiple Loads 方法的性能最差。此外，随着问题规模从一级缓存扩展到主存，各方法的性能均存在明显下降，这主要是由数据在低层次的存储传输开销增大造成的。

表 3.5 和表 3.6 中分别报告了基于转置布局算法的向量化和基于计算折叠算法的向量化在不同存储层级上与基准方法相比绝对性能的相对提升情况。在每个算例下，在不同存储层级我们的方法性能提升均是最大的。以基于计算折叠算法的向量化为例，表 3.6 中的结果与图 3.18 中的结果相对应，也再次证明了我们方法的有效性。

#### 3.4.4 并行分块实验

在本小节中，我们展示了我们的方法结合缓存分块和并行化技术带来的性能提升。我们使用与 3.4.3 节相同的 Stencil 参数配置，并将我们的向量化方案与密铺分块框架<sup>[28]</sup> 相结合，并与 SDSL<sup>[53]</sup>、Pluto<sup>[57]</sup>、YASK<sup>[58]</sup> 和 Tessellation<sup>[42]</sup> 进行比较。

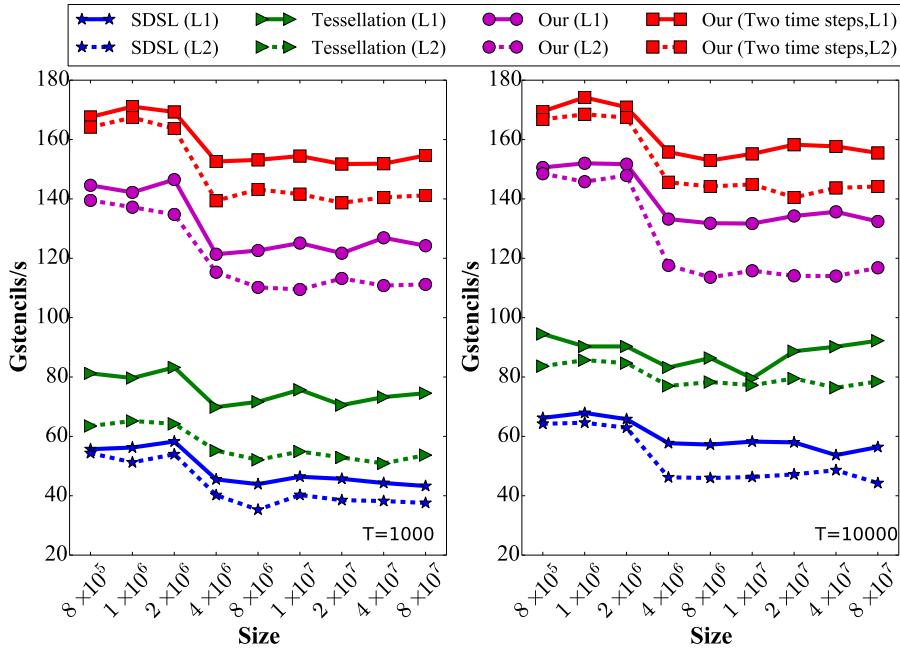


图 3.19 多核缓存分块实验中不同方法的绝对性能比较。(分块大小均固定在一级或二级缓存中, 结果以不同的总时间步长分别展示。)

**基于转置布局的向量化** 我们提出的基于转置布局的向量化方法结果如图 3.19(a) 和图 3.19(b) 所示, 时间步长分别为  $T = 1000$  和  $T = 10000$ 。所有的方法均采用了缓存分块技术, 我们逐一地对不同方法的性能进行了展示。如表 3.4所述, SDSL 采用了 Split 分块技术 (一维的嵌套分块, 高维度的混合分块) 来实现缓存的数据复用。Tesselation 则采用了镶嵌密铺分块技术, 利用编译器实现 Stencil 计算的自动向量化。基于转置布局的向量化曲线 (“Our”) 展示了局部转置算法 + 集成镶嵌密铺分块的性能, 而 “Our (Two time steps)” 曲线则展示了局部转置算法 + 集成镶嵌密铺分块 + 时间展开融合的结果。

表 3.6 串行无分块实验中基于计算折叠算法的向量化与其他方法在不同存储层级上的性能提升对比

Methods	AutoVec.		Reorg.		DLT		S-Fold		S&T-Fold		
	Machines	I. <sup>1</sup>	A.	I.	A.	I.	A.	I.	A.	I.	A.
L1 Cache	1.0x	1.0x	1.3x	1.4x	2.1x	2.2x	2.2x	2.2x	2.8x	3.0x	
L2 Cache	1.0x	1.0x	1.1x	1.2x	1.4x	1.7x	1.8x	2.1x	2.5x	2.9x	
L3 Cache	1.0x	1.0x	1.0x	1.3x	1.0x	1.8x	1.7x	2.1x	3.0x	3.1x	
Memory	1.0x	1.0x	1.0x	1.3x	1.0x	2.1x	1.8x	2.6x	2.7x	3.3x	
Mean		1.0x	1.0x	1.1x	1.4x	1.4x	1.9x	2.0x	2.4x	2.8x	3.0x

<sup>1</sup> 为了更清楚地进行说明, 表中英特尔至强金牌 6140 和 AMD EPYC 7452 这两款机器分别缩写为 I 和 A。

表 3.7 多核缓存分块实验中基于转置布局算法的向量化与其他方法在不同存储层级上的性能提升对比

	Blocking Level	Tessellation	Our	Our (Two time steps)
L3 Cache	L1	1.43x	2.54x	2.99x
	L2	1.21x	2.58x	3.01x
Memory	L1	1.62x	2.76x	3.42x
	L2	1.39x	2.92x	3.58x
Mean	L1	1.56x	2.69x	3.29x
	L2	1.32x	2.79x	3.48x

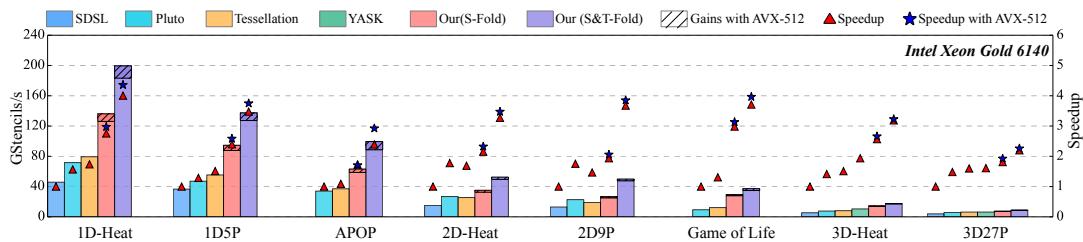


图 3.20 多核 Intel 机器上使用缓存分块技术不同方法的性能对比。(每组的加速度均与各组最低的性能作为基准进行比较, 该基准加速度默认值为 1, 用三角形标注。)

从图 3.19(a) 可以看出, 随着问题规模从三级缓存移动到主存, 性能明显下降, 这主要是由数据传输开销增大而引起。我们还进一步研究了分块大小对性能的影响。在分块大小固定在一级缓存内的条件下, 可以观察到整体的性能是高于分块大小固定在二级缓存内的结果。由于较小的 Stencil 可以直接预取到缓存中, 当问题规模溢出到内存中时, 不同分块大小之间的性能差距会进一步加剧。然而, 此时我们的方法仍然有着明显的性能优势。与 SDSL 相比, 在两种分块大小的情况下分别获得了大约 3.29 倍和 3.48 倍的性能提升。此外, SDSL 的性能整体上也不如 Tessellation 方法, 这是由于分块技术受限于其数据布局造成的。在图 3.19(b) 中我们评估了  $T = 10000$  的实验性能, 与图 3.19(a) 相比, 整体的性能趋势相似但结果略有增长。

表 3.7 展示了在不同存储层级上不同方法性能的相对提升。当问题规模在三级缓存内甚至在内存层级时, 我们的方法仍可以获得更好的优化性能。此时, 分块大小固定在一级缓存时加速比可达 2.54 至 2.76 倍, 这表明我们的向量化方法与集成的密铺分块技术可以在不同的问题规模下取得显著的优势。

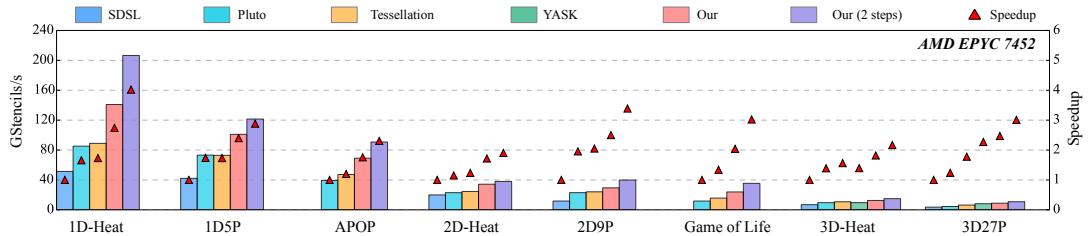


图 3.21 多核 AMD 机器上使用缓存分块技术不同方法的性能对比。(每组的加速度均与各组最低的性能作为基准进行比较, 该基准加速度默认值为 1, 用三角形标注。)

**基于计算折叠的向量化** 图 3.20 和图 3.21 展示了在两台机器上通过分块技术优化的不同方法的绝对性能和加速比的比较。由于 SDSL 不支持某些 Stencil 内核, 所以每个组的加速比是相对于各组最低的性能作为基准, 各组基准的加速比被标注为 1。在所有使用 AVX2 指令集的实验中, 我们的时空计算折叠方法收到了显著的性能提升, 这表明我们的向量化方案在较大的问题规模上比参考工作提供了更有效的加速。此外, 图 3.20 中使用 AVX-512 指令集的实验可以获得进一步的性能增益。SDSL 算法的性能不如和 Tessellation, 这是由于其数据布局限制了分块技术造成的。通过对图 3.20 和图 3.21 的分析, 可以发现实验的性能与 Stencil 内核的形状、尺寸和权重有关。对于星形 Stencil, 由于邻居格点更少, 与盒形 Stencil 相比, 其可以获得了更高的性能提升。对于低维的 Stencil 内核, 可以实现对寄存器内数据更多的数据重用, 从而展示出更好的性能。

#### 3.4.5 可扩展性

我们还评估了我们的方案和相关工作的可扩展性。实验的详细参数在表 3.3 中给出, 其中所有问题的规模都超过了三级缓存的大小。由于实验是在多种 Stencil 内核上进行的, 因此有些相关工作并非能够支持所有的 Stencil 内核。我们的分块框架与 Tessellation 相同, 所以我们的方法相对于它的性能提升完全来自向量化的收益。

**基于转置布局的向量化** 图 3.22 展示了使用 AVX2 和 AVX-512 指令集分别实现的一维、二维和三维 Stencil 计算的结果。SDSL 不支持 AVX-512 架构, 同时我们也省略了 Pluto 的结果, 因为它已被证明在性能上不如 Tessellation 技术。可以观察到, 在大多数情况下, 我们的方法可以获得最高的性能, 而 SDSL 的性能最低。在一维 Stencil 的计算中, 所有的方法在两个指令集上均实现了近乎线性的

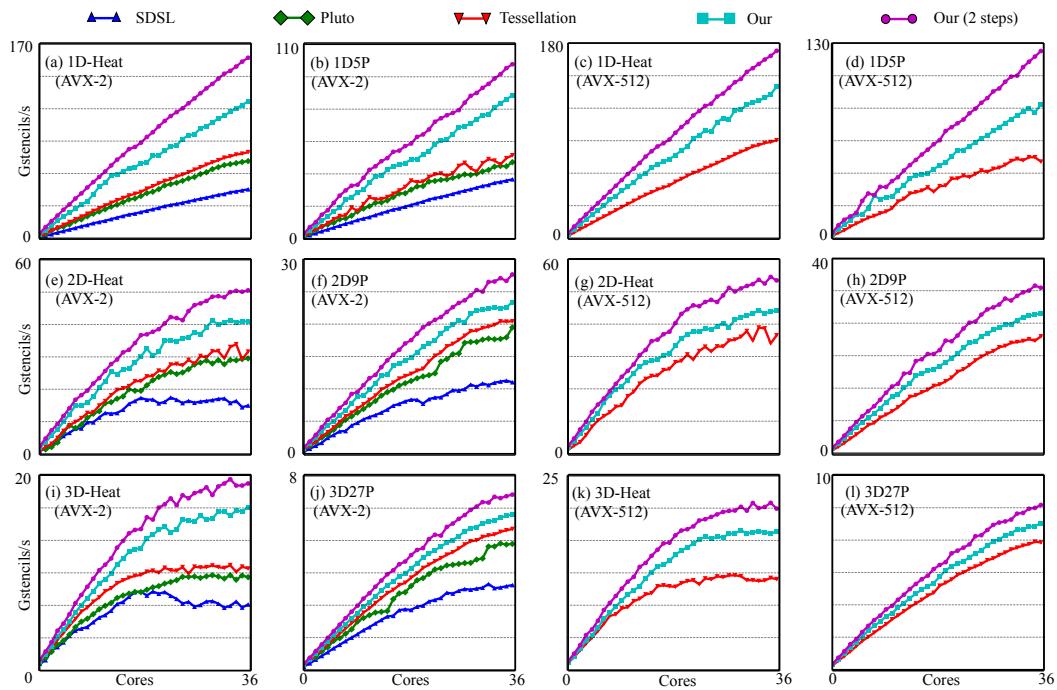


图 3.22 基于转置布局的向量化与其他方法不同维度多阶 Stencil 的多核可扩展性

表 3.8 可扩展性实验中内存带宽 (BW) 使用配额

Kernels	Pts	Stencil traffic (stencils/s)	Memory BW (GB/s)	BW utilization (%)
1D-Heat	3	2.58	41.28	16.13
1D5P	5	3.05	48.91	19.11
2D-Heat	5	3.67	58.74	22.95
2D9P	9	4.02	64.33	25.13
3D-Heat	7	4.77	76.38	29.84
3D27P	27	4.90	78.49	30.66
Theoretical Memory Bandwidth (GB/s)				127.96

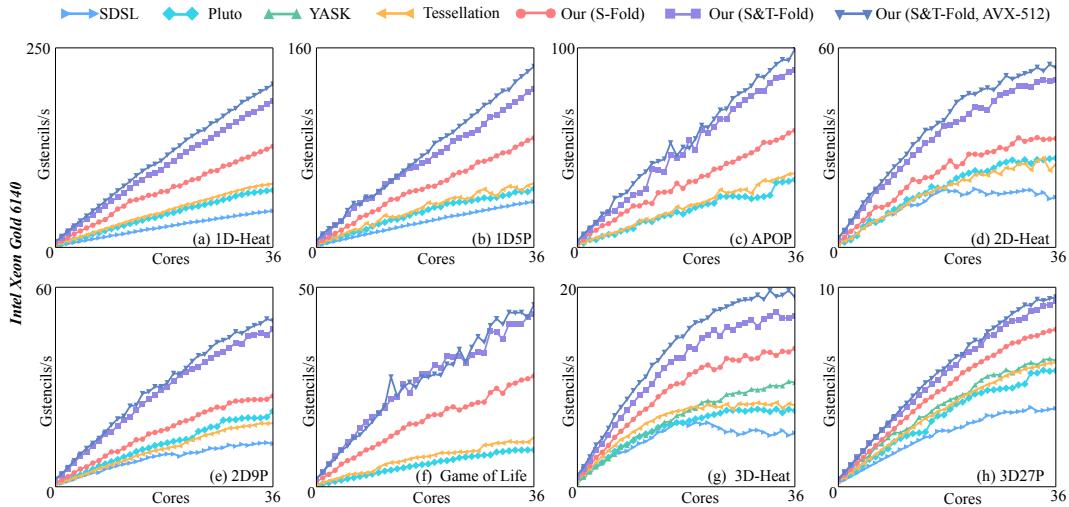


图 3.23 基于计算折叠的向量化与其他方法不同维度多阶 Stencil 的多核可扩展性 (Intel 机器)

可扩展性，并且我们所提出的时间循环融合策略提供了进一步的性能提升。随着问题维度的增加，由于多维 Stencil 固有的计算复杂性，所有方法的可扩展性都相应下降。然而，多维高阶 Stencil 的整体性能明显落后于低维 Stencil 的结果，这主要是由于高阶 Stencil 中复杂的数据访问模式造成的。使用 AVX2 指令集实现 Stencil 计算的情况下，我们的方法在一维 3 点和一维 5 点的实验中分别达到了 3.52 倍和 2.92 倍的性能提升。与使用 AVX2 指令实现的结果相比，图 3.22 中右半部分使用 AVX-512 指令集后的性能略有提高。

全局内存的吞吐量与传输的数据大小以及缓存的利用效率有关。在没有数据争用的情况下，全局内存的吞吐量大致与全局内存带宽饱和之前传输的数据量大小成正比。通过利用 Intel Processor Counter Monitor 测量工具<sup>[96]</sup>，我们的向量化方法在可扩展性实验中的内存带宽使用情况如表 3.8 所示。从表 3.8 中可以看出，内存吞吐量随着 Stencil 的阶数和维度的增加而增加。由于阶数或维度较小的 Stencil 计算速度较快，缓存需要更频繁地进行更新，导致传输的数据量较大。即使在三维 27 点的情况下，内存带宽仍然未达到饱和，此时内存利用率为 30.66%，这说明了我们的向量化方法结合缓存分块优化后数据复用的高效性。

**基于计算折叠的向量化** 从图 3.23 和图 3.24 中可以看出，我们的方法可以实现最高性能，而 SDSL 获得了最低性能。在一维 Heat 类型 Stencil 中，所有方法在两个指令集上都实现了近线性的可扩展性，并且我们提出的时空计算折叠提供了

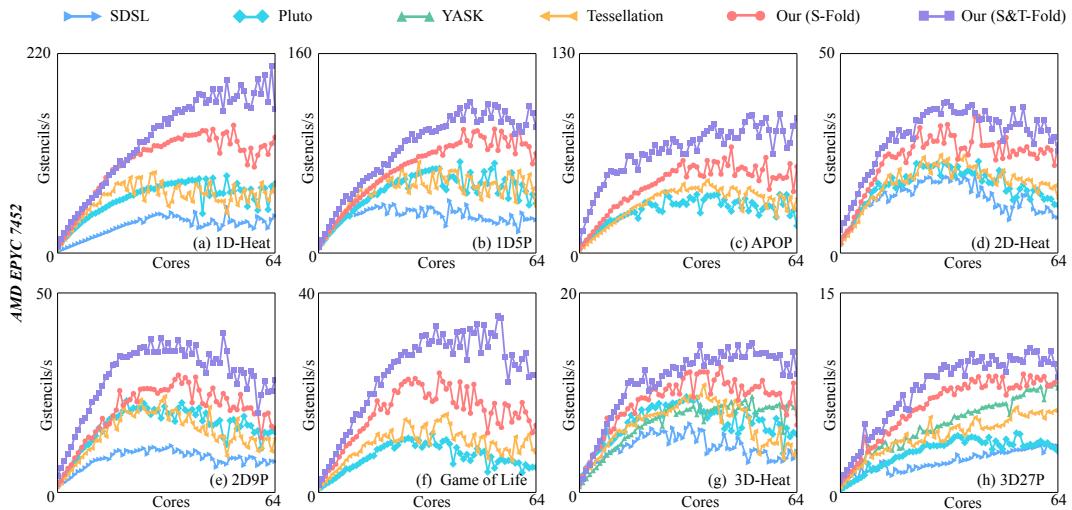


图 3.24 基于计算折叠的向量化与其他方法不同维度多阶 Stencil 的多核可扩展性 (AMD 机器)

显着的性能提升。随着问题维度的增加，由于多维 Stencil 计算的复杂性，所有方法的可扩展性都随之下降。同样，高阶 Stencil 的整体性能也落后于相应的低阶结果，这是由于高阶 Stencil 中复杂的数据访存模式造成的。与使用 AVX2 指令实现的结果相比，在 Intel 机器上 AVX-512 指令集优化后的性能也进一步提升。

### 3.4.6 实验讨论

在本小节中，我们对之前各种配置下的实验进行了分析，以从不同角度梳理我们提出的方法的贡献。

我们首先研究了数据准备对 DLT 的性能影响。DLT 是一种 state-of-the-art 的 Stencil 向量化方法。实验结果显然表明，DLT 方法的高性能是通过调节特定 Stencil 参数（即低维度或长时间步）获得的，而在实际应用中并不能保证这些特定的参数。相比之下，我们的向量化方案可以更有效地实现数据准备。

串行无分块实验则评估了各种向量化方法在单进程上的性能。与 Multiple Loads 方法相比，基于转置布局的向量化平均可以实现 2.81 倍的性能提升。随后，在较大的总时间步长的情况下，我们的方法性能增益仍然显着，并且与小时间步长的结果一致。而与基准方法相比，我们的时空计算折叠向量化在 Intel 和 AMD 机器上分别可以实现 2.8 倍和 3.0 倍的改进。此外，DLT 方法在相对较小的问题规模上更合适，这与其此时在内存中的维度变换相关的操作性能损失较小有关。随着问题规模从一级缓存扩展到主存中，我们可以清晰地看到整体性能趋势随

着两台机器上的存储层级扩展而下降。此外，AMD 机器在相同的内存结构中获得了稍高的性能，而 Intel 机器则在相同的问题规模下获得了更好的结果。这主要是由于 AMD 机器上配有更高的主频速度但更小的缓存大小。

多核实验则在使用缓存分块技术的基础上实现了各种 Stencil 并行计算测试。通常，新型的英特尔机器包含能够在更大的 512 位寄存器上执行的 AVX-512 指令集，因此，我们也研究了我们的方法在 AVX-512 架构上的性能。结果表明，使用 AVX-512 指令集可以为我们的方法带来更好的性能，尤其是在一维和二维的 Stencil 计算上。值得注意的是，当涉及大量 AVX2 和 AVX-512 指令的运算时，CPU 存在明显的频率降低现象（CPU throttling）。当使用更多内核时，计算速度的减缓甚至更明显。例如，当每个处理器上的活跃核心扩展到 18 个满核时，使用大量 AVX2 指令的 Intel 机器峰值主频将从 3.70GHz 动态下降到 3.00GHz<sup>[97]</sup>。相应地，使用 AVX-512 指令集则会进一步降低到 2.10GHz。这些因素导致了三维 Stencil 的性能提升不如低维低阶 Stencil。实验结果的总体趋势与串行无分块实验基本一致，相对于 SDSL，基于转置布局的向量化可以获得了平均 3.29 倍的加速。我们也进一步研究了分块大小对性能的影响。结果证明，适当的一级缓存或缓存内的分块大小可以为所有方法带来更好的性能。同时，我们的时空计算折叠方法也明显优于其他方法。

实验中最后的可扩展性评估表明，我们利用密铺分块技术的向量化方案在各种配置中均优于其他工作。受限于其特定的数据布局，SDSL 性能低于其他方法。由于多维或高阶 Stencil 的计算密集度更高，更多的邻居格点数据被加载到了缓存中，而它们却无法进行充分地复用。因此，每种方法的整体性能均随着维度或阶数的增加而逐渐下降。此外，在基于计算折叠向量化的可扩展性实验中，一个有趣的观察是在两台机器上达到峰值性能所需的核心数量。Intel 机器在所有核心都处于活跃状态时能够获得最佳性能，而 AMD 机器在略高于一半的核心数量时就已达到峰值性能。在两台机器上，达到峰值的实际核心数始终落在 30 到 40 左右的区间内。如上所述，节流现象（CPU throttling）导致了更多活跃核心参与计算时整体的主频降低。此外，随着计算核心的增加，更多的核间通信也是性能限制的因素。

### 3.5 本章总结

在本文中，我们提出了一种新型的转置布局，以有效地解决 Stencil 计算的向量化中输入数据的空间数据冲突。在所提出的转置布局的基础上，我们设计了一种具有寄存器内多时间步处理的时间循环展开融合策略。此外，我们描述了所提出的向量化方案如何与密铺分块框架集成以增强数据重用和并发性。

在本文中，我们提出了一种新型的空间计算折叠策略，以有效地克服空间数据冲突实现 Stencil 计算的向量化。然后在提出的空间计算折叠的基础上，我们设计了能够减少时间迭代空间中算术计算冗余的时间折叠算法。此外，我们通过偏移重用、密铺分块和半自动代码生成技术进行优化，进一步提升方法的性能。通过定性分析和定量实验，我们证明了与经典向量化方法（自动向量化<sup>[55]</sup>和数据重组<sup>[28]</sup>）、state-of-the-art 编译器（Pluto<sup>[56,57]</sup> 和 SDSL<sup>[28,52,53]</sup>）以及近期高度优化的工作（YASK<sup>[58]</sup> 和 Tessellation<sup>[42]</sup>）相比，我们的方法具有明显的性能优势。



## 第4章 中粒度并行：万核量级分布式机器学习框架研发

### 4.1 本章概述

中粒度并行问题一般指较大的循环级并行或较小的子任务级并行。一般而言，中粒度并行性更多地是相对于细粒度和粗粒度并行之间的折中，在这种情况下，任务大小和通信时间大于细粒度并行性、小于粗粒度并行性。大多数通用分布式算法框架均属于中粒度并行。目前，国内现有的可扩展性相关研究并未对机器学习类问题展开深入讨论<sup>[19,20,23]</sup>，而机器学习类问题又与经典的高性能数值算法、扩展的大规模科学计算应用紧密相关，在算法层面和应用层面起到了融合相承的重要作用。例如，机器学习中的卷积核作为一种重要的并行计算模式，在高性能计算中，其通常被称为 Stencil 计算；而 2020 年的“戈登贝尔奖”更是结合了物理建模、机器学习和高性能计算的相关方法，将具有从头算精度的分子动力学模拟的极限提升至了 1 亿个原子规模，开辟了新的计算范式。因此，在中粒度并行问题中，本文选取了分布式机器学习框架这一跨方向的应用，以开展对这类问题可扩展性研究的深入探索。

随着基于聚类分类的回归预测技术逐渐兴起，对高性能分布式机器学习算法框架的研发成为高性能计算和人工智能社区的共同关注点。而在分布式框架的研发过程中，存在着不同程度扩展性的不连续和非线性现象。例如，聚类和回归两种方法在技术上是不连续的。如何设计出聚类和回归更好的算法，并能够使二者结合时发挥出全局最佳的扩展性，是近期工作的一项热点研究。而将二者分

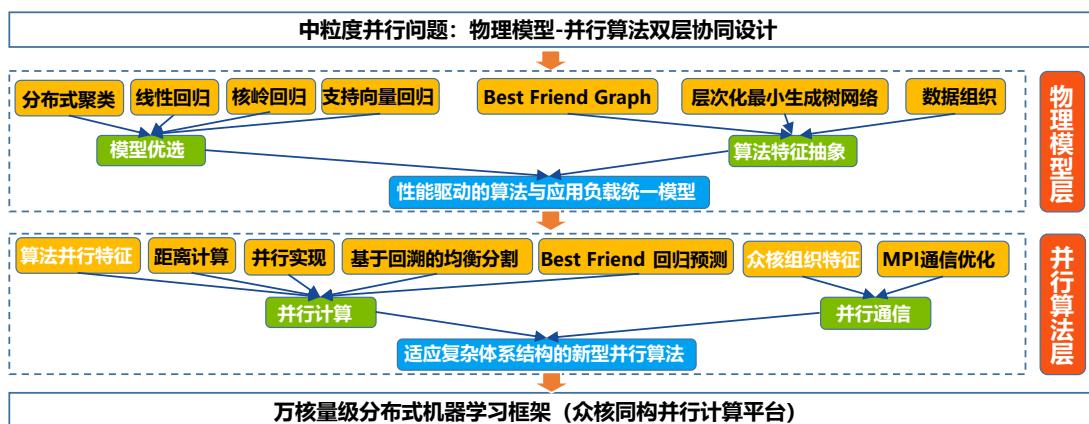


图 4.1 物理模型-并行算法双层协同设计

开考虑，每种方法在分布式系统上的扩展也都各自面临着性能的非线性增长问题。

机器学习框架的研发属于典型的中粒度并行问题。由于需要将不同的训练子任务分配到各计算节点，细粒度的共享内存体系结构已不能满足数据计算和通信的需求。其一般采用众核分布式系统的体系架构进行实现，经过优化后并行规模一般可达万核量级。相比于细粒度的高效核间通信，其分布式的通信同步将更多地要求手动设计算法以解决负载不均衡问题。另外，因为机器学习框架的研发主要是用于回归预测等不同的应用需求，不同分布式系统上可用的百核、千核甚至是万核的计算资源都能够为框架提供不同程度的硬件加速。因此，过多地考虑硬件架构的性能模型将对框架可扩展性的通用性造成破坏。因此，对于中粒度的机器学习框架研发，我们可采用物理模型-并行算法双层协同设计方法。

如图 4.1 所示，根据多层次协同设计理论中的定理 2.1 和原则 2.2，针对机器学习框架研发的中粒度并行问题，物理模型-并行算法双层协同设计方法主要是对模型和算法两层的联合优化。在物理模型层，我们根据需要解决的上层应用需求分别进行建模，设计新型的分布式聚类和回归方法。在算法模型层，需要设计高效并行算法，缓解在模型中存在的数据同步、负载不均等带来的通信和计算瓶颈。如上所述，由于不需要考虑下层的硬件架构，应用-算法层的协同设计即可完成对机器学习框架研发的扩展性研究。不难看出，在可扩展的中粒度并行问题中，其存在的不连续和非线性现象也相对较容易解决。相关内容我们将在第 4 章中详细展开。

#### 4.1.1 引言

机器学习在过去几十年中已渐渐成为信息技术领域的重要方向之一，其相关应用已渗透生活的方方面面，成为现代科技的重要组成部分。机器学习中的任务基本上可以分为两种主要类型：有监督学习和无监督学习。有监督学习是在事先知道样本的输出值应该是什么的情况下完成，然后通过学习一个函数来对其他未标记的数据进行预测。回归和分类是有监督学习中的两个重要类别<sup>[31,32]</sup>。无监督学习则不包含支持的标记样本，因此算法需要推断数据集中潜在的自然结构。聚类和主成分分析是两种流行的无监督学习技术<sup>[98]</sup>。随着人工智能等基于机器学习的技术显著崛起，研究人员需面对的数据量也在呈指数级增长<sup>[26,99]</sup>。这使得模型的训练时间从几小时到几周不等，给计算、网络和存储带来了巨大的

压力。因此，加速模型训练是机器学习领域的一项重要的研究挑战。

近年来，高性能计算和并行技术催化了机器学习的现代革命性进展。越来越多的科技公司将高性能计算技术作为机器学习的生产力引擎和支持人工智能创新的解决方案，如 Google Cloud TPU、Amazon AWS AIHPC 和 Microsoft Azure。在机器学习领域，聚类和回归是两种基本和关键的技术，它们被认为是最具代表性的无监督和有监督学习方法<sup>[31,32]</sup>。各种需要高性能训练的机器学习应用程序都涉及经典的回归预测技术，如线性回归、核岭回归和支持向量回归<sup>[100]</sup>。

聚类方法通过数据属性将数据分成不同的组，其目标是同一组中的样本彼此相似，且与其他组中的样本不同。组内样本相似度越高，组间样本差异性越大，则聚类效果越好。各种类型的聚类方法提供了一系列不同的用途<sup>[101–105]</sup>。基本上，它们可以区分为层次（嵌套）与划分（非嵌套）方法，其中凝聚层次聚类（AHC）和 K-Means 是两种被广泛使用的代表性聚类方法<sup>[106,107]</sup>。凝聚层次聚类及其变体生成一组嵌套的样本集并组织为分层树状结构<sup>[108]</sup>。K-Means 则在各种分类问题上比较简单高效<sup>[109,110]</sup>。近年来，图聚类（GC），例如最小生成树聚类（MSTC），也广泛地用于通过图理论检测具有不规则边界的聚类。这类方法首先使用 Prim 算法或 Kruskal 算法构建最小生成树（MST），对边进行排序以去除连通分量的不一致边，然后重复直到达到设定的循环值<sup>[111,112]</sup>。

回归方法在模型训练阶段涉及大量密集的计算。一种标准的分布式回归方法是将计算均匀地映射到  $p$  个处理器的系统上，然后规律地执行全局归约<sup>[113]</sup>。通常，由于存在频繁的同步开销，这些分布式回归方法在可扩展性方面效果不佳。另一种方法则是使用分治（DC）算法来优化并行过程。分治回归（DCR）的基本思想是将数据分成  $p$  个相似的部分，然后生成  $p$  个相似的训练模型，并平均化  $p$  个模型的结果以获得最终解。分治回归相对减少了内存和计算的开销，如核岭回归上的 DCKRR 和支持向量回归上的 DCSVM<sup>[114,115]</sup>。但是，在很多情况下，分支回归并不能保证算法的准确度<sup>[116]</sup>。

最近的相关工作 Balanced KRR v2 (BKRR2)，这是 Berkeley 研究团队对于分布式核岭回归的最新优化版本<sup>[5]</sup>。BKRR 利用 K-Means 将  $n$  个样本划分为  $p$  个集合，其中  $p$  是进程数量。每个样本集包含了  $n/p$  个分派的样本，然后相应地生成  $p$  个模型。为了保证负载均衡，如果某个样本集已经包含由 K-Means 分派的  $n/p$  个样本，则该集合不会再添加新的样本。然后再对每个分布式模型执行回归操作，

并收集结果以获得平均值。在 BKRR 的基础上，该团队提出了 BKRR2 的优化版本，其通过独立的训练模型来提高准确性，而不需要全局归约。虽然 BKRR2 证明它可以达到比目前最快的方法更高的准确率和效率<sup>[5,116]</sup>，但很多问题仍然没有得到明确地解决。

首先，K-Means 聚类在大型数据集上的准确性和收敛性并不总是令人满意。在 BKRR2 的实验中，它至少需要数千次的迭代才能收敛。由于聚类中心是动态变化的，因此在每次迭代中传输的数据量也较大。BKRR2 也没有直接解决这个问题，其只用了一个进程来实现 K-Means 的聚类过程。其次，由于  $K$  值是在 K-Means 中指定聚类数量的超参数，因此在 BKRR2 中它的值被简单地设置为等于进程的数量，这导致了其鲁棒性不高。随着进程数量的变化，聚类结果与真实分布有很大的偏差，这在许多情况下会导致聚类的准确性较差。更糟的是，训练时间会随着增加的进程数生成越来越多的聚类个数而增加，导致其不适用于更大规模的训练。此外，BKRR2 是以准确性为代价实现的负载均衡。在聚类过程中，样本被依次遍历并分派到离它们最近的聚类中心。然而，由于 BKRR2 的按序遍历设计，当后续出现一个更相关的样本时，它可能无法再添加到与其最相似的某个已“满”的聚类中。

在本章中，我们设计了一个高效的并行回归框架来解决现有方案中存在的问题。首先，我们提出了一种名为最好朋友的图结构来捕获数据样本中潜在的最重要的信息。基于所提出的数据结构，我们设计了一种新型聚类方法，即最好朋友聚类 (BFC)。通过层次化的最小生成树结构，来减少聚类的计算量，提高了聚类的准确性，并增强了算法的可扩展性。然后我们挖掘出图结构的内在属性，设计了一套优化的度量标准来自适应决定最优的聚类结果。

然后，基于回溯的思想，我们提出了一种数据并行的负载均衡算法，它可以在分布式节点上体现出真实分布上的更多特征信息。我们未采取基于最小生成树的相关工作中按非递增顺序连续删除边<sup>[112,117]</sup>的做法，而是通过简单地交换样本指针来聚合同一分组中的样本。之后通过回溯再拆分较大的聚类，并从最优层中合并较小的聚类，以使每个进程中的总样本数接近于  $n/p$ 。因此，边的相关信息不需要被存储，可以很容易地将整个过程并行化。

此外，我们将提出的最好朋友聚类应用于并行回归算法。尽管上述提出的方法均可以单独使用，但我们将它们与多种回归算法相结合，作为一个完整的并行

回归框架。每个进程根据聚类的结果在本地独立地生成单个或多个训练模型。对于每个测试样本，它在预测阶段从这些模型中选择一个最佳的进行计算。因此，在每个进程上我们也有效地实现了模型并行性。

最后，我们先将最好朋友聚类与三种不同的聚类方法进行了实验对比，然后通过与 state-of-the-art 工作对标，对我们的回归框架的准确性和可扩展性进行了细致的性能分析。实验结果证明了我们所提出的方法在分布式系统上具有明显的性能优势。

本章的主要贡献包括：

- 针对大规模的分布式聚类，我们通过层次化的最小生成树结构，提出了一种新型的最好朋友聚类方法，该方法具有准确、快速、无参数的特点。
- 我们对所提出的聚类方法的理论特性进行了深入研究，基于层次化的最小生成树结构特征设计了度量标准以自适应决定最佳聚类层。
- 我们优化了分布式系统上的聚类，提出了基于回溯的负载均衡算法，并将其应用于并行回归预测。
- 通过利用数据和模型并行的混合结构，我们将所提出的方法与回归技术相结合作为并行框架。它在收敛性、准确性和可扩展性方面表现出优秀的性能。

#### 4.1.2 背景

**回归技术** 回归是一种有监督的机器学习技术，其多用于研究一个或多个预测变量与响应变量之间的关系，以学习获得最佳拟合曲线。基于获得的曲线，可以对新样本点进行回归预测。目前有多种算法可用于构建回归模型，在我们的研究中，我们选取三种广泛使用的回归技术进行建模并实现。

线性回归是一种广泛使用的回归方法，其通过采用直线来找到因变量与一个或多个自变量之间的线性关系。给定一个  $d$  维训练样本  $\mathbf{x}_i$  和相应的测量回归值  $y_i$ ，训练阶段的目标是找到一个  $\mathbf{w}$  使得  $y_i \approx \mathbf{w}^T \cdot \mathbf{x}_i$ 。这可以表述为方程4.1中的最小二乘问题，以通过最小化残差和来找到最优解。因此，对于一个测试样本  $x^*$ ，我们可以利用训练模型通过  $y^* = \mathbf{w}^T \cdot \mathbf{x}^*$  来进行回归预测，并用均方误差 (MSE) 方法评估准确度。

$$\mathbf{w}_{opt} = \arg \min \frac{1}{2} \sum_i^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 \quad (4.1)$$

为了避免过拟合和解决一些不适定问题，方程 4.2 中使用了带正参数  $\lambda$  的 L2 正则化，这称为岭回归。

$$\mathbf{w}_{opt} = \arg \min \frac{1}{2} \sum_i^n (y_i - \mathbf{w}^T \mathbf{x}_i)^2 + \frac{1}{2} \lambda \|\mathbf{w}\|^2 \quad (4.2)$$

此外，核方法被广泛地用于通过非线性映射将样本映射到高维空间。因此，通过将岭回归与核方法相结合，产生了核岭回归（KRR），它在高维空间中学习模型能够获得更好的预测精度。解向量  $\alpha$  可以写成方程 4.3 中的闭式解，即：

$$\alpha = (\mathbf{K} + \lambda \mathbf{I}_N)^{-1} \mathbf{y}. \quad (4.3)$$

其中  $\mathbf{K}$  是由  $\mathbf{K}_{i,j} = \phi(x_i, x_j)$  构造的  $n \times n$  核矩阵， $\mathbf{y}$  是对应的  $n \times 1$  回归向量。然后使用  $\alpha$  来计算预测阶段的回归值，如式 4.4 中所示。

$$y^* = \sum_{i=1}^N \alpha_i k(x^*, x_i) \quad (4.4)$$

支持向量机 (SVM) 也可以用于称为支持向量回归 (SVR) 的回归方法，它包含算法所有关键特征，例如最具代表性的边际 (margin) 属性。KRR 和 SVR 都可以通过使用核技巧来学习非线性模型，但它们的损失函数不同，即分别为平方误差损失函数和 epsilon 不敏感损失函数。在 SVR 的算例中，需要为 SVM 设置一个容忍边际 (epsilon)，并且可以对其进行调整以获得模型所需的精度。式 4.5 中给出了解且其约束于式 4.6，其中  $\mathbf{w}$  是决策面的法向量。

$$\mathbf{w}_{opt} = \arg \min \frac{1}{2} \|\mathbf{w}\|^2 \quad (4.5)$$

$$|y_i - \mathbf{w}x_i| \leq \varepsilon \quad (4.6)$$

**聚类算法** 相关文献中提出了许多聚类方法来识别具有不同特征的类。本小节将简要介绍凝聚层次聚类 (AHC)、K-Means 和最小生成树聚类 (MSTC)。

凝聚层次聚类是一类重要的聚类方法，它将生成的嵌套样本集组织成一棵树状结构。基本算法包括以下三个关键步骤：以单个样本为集合开始，连续合并两个最近的集合，直到获得一个整个聚类集合。算法 3 简明地描述了凝聚层次聚类

算法。通常情况下凝聚层次聚类可以产生质量更好的聚类结果。然而，缓慢的收敛速度、大量的计算操作和昂贵的存储要求使得其在更大规模数据集上的扩展成为问题。此外，由于凝聚层次聚类没有直接的全局目标函数，因此无法指导算法的收敛性，且无法撤消上一步的聚类决策。

---

### 算法 3 Agglomerative Hierarchical Clustering Algorithm

---

- 1: Let each sample be a cluster.
  - 2: **repeat**
  - 3:     Merge the nearest two clusters as a new one.
  - 4:     Update the proximity matrix.
  - 5: **until** Only one cluster remains.
- 

K-Means 是最著名的划分聚类方法之一。算法 4 中描述了 K-Means 聚类的基本步骤。K-Means 在很多问题上简单而高效，但是，对于条状、环形状等非团簇状的样本集，其聚类效果一般。另外 K-Means 对于事先给定的  $K$  值、初始点敏感，不同  $K$  值、初始点可能导致聚类得到结果差异较大，也可能因为初始点分属同一类，导致最后结果陷入局部最优解，无法达到全局最优解。

---

### 算法 4 K-Means Algorithm

---

- 1: Choose  $K$  initial sample as centroids.
  - 2: **repeat**
  - 3:     Assign each sample to its nearest centroid to form  $K$  clusters.
  - 4:     Recompute  $K$  centroids.
  - 5: **until** Centroids remains unchange.
- 

最小生成树聚类是近年来颇受关注的新兴聚类算法，它的基本思想如下。首先，将所有的类维护成一组图结构中的连通分量，计算并记录类间的距离。然后使用经典的最小生成树算法构造最小生成树，直到获得  $k$  个聚类。该算法在具有不规则边界的聚类上可以获得相对较好的结果，然而它是高度串行化和计算密集型的算法<sup>[117]</sup>。

---

### 算法 5 MST-based Clustering Algorithm

---

- 1: Compute and sort the pairwise distances.
  - 2: **repeat**
  - 3:     Run MST algorithm to form clusters.
  - 4: **until**  $k$  clusters.
-

### 4.1.3 相关工作

本章的工作主要是对两条主线的深入研究。第一条主线是研究分布式系统上高质量的高效聚类方法。目前的工作中。凝聚层次聚类和 K-Means 基本上是两种广泛使用的方法<sup>[118]</sup>。在近期的一些研究中，也有一些工作将二者进行结构上的结合。Bahmani 等人通过使用分布式节点上的 MapReduce 算法扩展了 k-means 适用的问题<sup>[119]</sup>。Ene 等人优化了 MapReduce 上的贪心算法来解决 k-center 问题，并提出局部搜索策略以解决 k-median 问题<sup>[120]</sup>。之后，也有相关论文探索了结合 MapReduce 模型的研究<sup>[121–123]</sup>。Bouguettaya 等人基于一组由 K-Means 生成的质心构建了一个层次结构，以提高凝聚层次聚类<sup>[124]</sup> 的效率，而它只在单个节点上进行实现。另外，K 值在大多数基于 K-Means 的方法中是根据经验选择的<sup>[125,126]</sup>，而超参数调整本身就是一个更复杂的问题。至于在图聚类方面，也有相关团队投入了大量精力研究高效算法<sup>[127–130]</sup>。尽管如此，分层的特性在这些工作中几乎没有引起注意。Grygorash 等人提出了分层基于欧氏距离的最小生成树聚类算法 (HEMST)。给定聚类的数量作为输入，HEMST 通过合并多条边来加速收敛过程，而大于阈值的边则需要按顺序排序和移除<sup>[112]</sup>。Jin 等人通过 Prim 算法将聚类问题拆分为多个重叠的子问题，通过解决每个子问题，然后将它们合并为一个整体来进行聚类<sup>[131]</sup>。但是，这需要将最小生成树存储在 Map 端，然后混洗到 Reducer 端。Bateni 等人<sup>[117]</sup> 则使用了两种经典的最小生成树算法扩展了基于 MST 的方法，称为亲和聚类。然而，它仍然严重依赖 MapReduce 模型，并且需要在删除边后将所有边移动到一个节点上串行地生成最小生成树。此外，亲和聚类并没有给出他们实验中运行时间的详细数据和使用的节点数量<sup>[117]</sup>。Wang 等人利用分治法构建近似最小生成树，然而 MST 长边的检测过程也是高度串行化的<sup>[132,133]</sup>。因此，在分布式系统上仍需要一种能够对大规模数据进行并行计算的高效聚类算法，以提高 K-means 的准确性、凝聚层次聚类的效率和图聚类方法的可扩展性。

本章的第二个重点与分布式回归相关，现有方法在计算时间和内存使用方面都存在严重的可扩展性问题<sup>[134]</sup>。Mini-batch Gradient Descent (MBGD) 被用于批量数据的训练<sup>[135]</sup>，然而它是串行的实现。基于 MapReduce 的方法<sup>[136,137]</sup> 使得计算可以分布到各个节点，每个节点均包含了部分数据。然而，由于频繁的同步操作，计算和网络方面的总体开销较高。后来在分布式系统上对 SVM 和 KRR

采用分而治之的算法进行了优化<sup>[4,114,115]</sup>。最近并行 SVM(PSVM) 也被提出以减少内存和时间消耗<sup>[138,139]</sup>。Zhang 等人证明了核方法的回归比非核方法更准确，并且他们设计的 DCKRR 可以胜过之前的回归方法<sup>[115]</sup>。You 等人<sup>[116]</sup>提出了基于 K-Means 的核岭回归 (KKRR)，用于对聚类数据进行更有效地回归预测。最近的工作 BKRR2<sup>[5]</sup>则是在 KKRR 上进行了优化，在每个节点上尽量实现负载均衡，并且比 DCKRR 和 KKRR 具有更好的准确性，这被认为是并行核方法回归的 state-of-the-art 方法。因此，本章的重点是与这两项密切相关的工作 DCKRR 和 BKRR2 进行了比较，这两项工作也在第 4.1.1 节的引言部分进行了详细阐述。

## 4.2 物理模型抽象：基于 Best Friend Graph 图结构的层次化最小生成树网模型

在本节中，我们首先提出了一种简单、高效、准确的大规模聚类方法。第 4.2.1 节和第 4.2.2 节分别描述了聚类算法中 Best Friend Graph 结构的定义以及该图结构的理论属性，这也是我们方法的基本数据结构。

### 4.2.1 Best Friend Graph 数据结构

直观上，本节提出的聚类算法是受到最基本的社会关系，即朋友圈的启发，用于检测数据中的潜在分组属性。现有的聚类方法通常考虑所有样本对之间的关系，导致了较慢的收敛速度和较大的计算开销，这对于分布式系统上的大规模数据集来说是不合适的。实际上，许多样本之间的远层关系可以忽略不计。我们的聚类方法灵感上源于观察到人们日常经常会和最好的朋友一起做决定。因此，我们通过只考虑最重要的最好朋友的关系来简化朋友圈。

一个 **Best Friend Graph**  $G(V, E)$  是定义在一个用于聚类的  $n$  个输入样本的数据集上。该集合  $V$  由  $n$  个样本点组成，每个顶点  $i \in V$  都连有一条有向 **最好朋友边**  $(i, j)$ ，其中这条有向边的指向  $j$  是该点的最好朋友，即  $i$  的最近邻居。值得注意的是，如果存在多个与  $i$  距离相等的最近邻，我们只需选择具有最小字典顺序的顶点作为  $i$  的唯一最好朋友点。因此，含有  $|V| = n$  个顶点的 Best Friend Graph  $G(V, E)$  包含  $n$  条有向最好朋友边。图 4.2 展示了一个易于理解 Best Friend Graph。我们以全球城市指数<sup>[140]</sup> 中所提供的 12 个具有代表性的知名城市作为研究案例。通过计算出各城市之间的地理距离数据，我们用 12 条有向边来指示它们之间的最好朋友关系。

基于 Best Friend Graph 结构，一个聚类被定义为一组互相连接的顶点。显然，

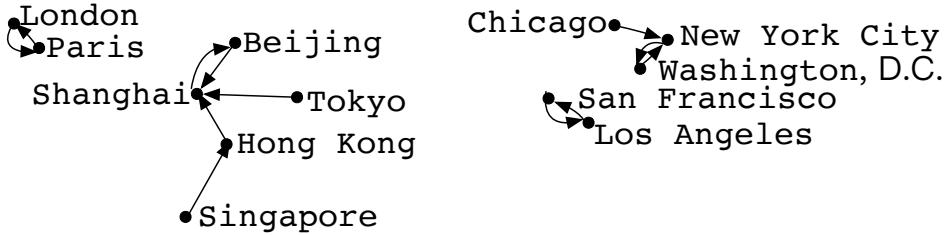


图 4.2 以全球知名城市为例的 Best Friend Graph

在第一步中，图 4.2 中的城市算例被朋友圈聚类为四组，而这经最好朋友聚类识别出的四组类与地理分类法相一致。例如，在地理学中，洛杉矶和旧金山被归类为美国西部地区；芝加哥、华盛顿特区和纽约市被归类为美国东部地区；伦敦和巴黎被归类为西欧地区；其他城市则被归类为亚洲地区。

令  $T_k(V_k, E_k)$  为  $G$  中的一个连通子图  $k$ ，且  $v_k = |V_k|$ 。具体来说，每个连通子图所表示的新聚类  $k$  都由一个质心  $\bar{x}_k$  表示。代表聚类  $T_k$  的质心  $\bar{x}_k$  则由式 4.7 对分配到该类中的所有顶点  $x_i$  进行求取平均值而得。在算术计算上，这需要为每个维度分别计算出该维坐标的算术平均值  $x_i'$ 。

$$\bar{x}_k = \frac{1}{v_k} \sum_{x_i \in V_k} x_i \quad (4.7)$$

然后，我们可以基于上一步中生成的各聚类中心再次构建一个新的 Best Friend Graph。在我们的城市算例中，四个聚类进一步分成两组，如图 4.3 所示。我们递归地将该方法应用于新一层的聚类，直到最后生成一个包含了所有样本的类。整个过程可在对数时间复杂度内快速收敛，因为在每个层中至少合并两个聚类。此外，随着聚类层的增加，计算和通信时间也随每一层聚类数量的减少而逐渐减少。



图 4.3 以全球知名城市为例的 Best Friend 聚类

#### 4.2.2 模型理论特征

在本小节中，我们对 Best Friend Graph 模型中的几个关键属性进行了理论分析，这些理论特征也是自适应确定最佳聚类层的基础。

**引理 4.1.** *Best Friend Graph* 中至少存在一个有向环。

证明. 如果我们用无向边替换所有的有向边，Best Friend Graph 则会成为一个有  $n$  个顶点和  $n$  条边的无向图。通过归纳，我们可以很容易地证明它至少包含一个无向环。将环上的边转换回有向边，我们就得到了一个有向环。否则，环上必有一个顶点  $i$  连有两条最好朋友边  $(i, *)$ ，这与 Best Friend Graph 的定义相矛盾。□

**引理 4.2.** *Best Friend Graph* 中的每个弱连通分量都包含一个且仅一个有向环。

证明. 根据定义， $\text{Best Friend Graph } G(V, E)$  中的每个弱连通分量  $G'(V', E')$  仍然是一个 Best Friend Graph。根据引理 4.1， $G'$  至少包含一个环。因此我们得到  $|E'| \geq |V'|$ 。我们必然有  $|E'| = |V'|$ ，即在  $G'$  中只存在一个环，否则会有  $|E| > |V| = n$ ，这与 Best Friend Graph 的定义相矛盾。□

**引理 4.3.** *Best Friend Graph* 中任一有向路径上边的权重值是非递增的。

证明. 从定义中可以清楚地推导出这一引理。例如，对于有向路径  $i \rightarrow j \rightarrow k$ ，我们有  $\omega(i, j) \geq \omega(j, k)$ ，否则顶点  $j$  的最好朋友应该是  $i$  而不是  $k$ 。□

**引理 4.4.** *Best Friend Graph* 中的有向环长度为 2，且环上边的权重值相等。

证明. 假设每个顶点只有一个最近邻，即最好朋友边  $(i, j)$  的权重  $\omega$  严格小于  $i$  与其他顶点的距离 ( $\omega(i, j) < \omega(i, k), k \neq i, j$ )。如果存在包含两个以上顶点的有向环，如  $i \rightarrow j \rightarrow k \rightarrow i$ ，我们有  $\omega(i, j) > \omega(j, k) > \omega(k, i) = \omega(i, k)$ 。因此  $i$  的最好朋友边应该是  $(i, k)$ ，与已知矛盾。

否则，我们有  $\omega(i, j) \geq \omega(j, k) \geq \omega(k, i) = \omega(i, k)$ 。由于  $(i, j)$  是一条最好朋友边，我们得到  $\omega(i, j) = \omega(i, k)$  和  $\omega(i, j) = \omega(j, k) = \omega(k, i) = \omega(i, k)$ 。根据 Best Friend Graph 定义中的最小字典序规则，我们得到了一个不一致的字典序列  $j < k < i < j$ 。□

**推论 4.5.** 从任意一个顶点开始，沿 *Best Friend Graph* 遍历必然会进入环。

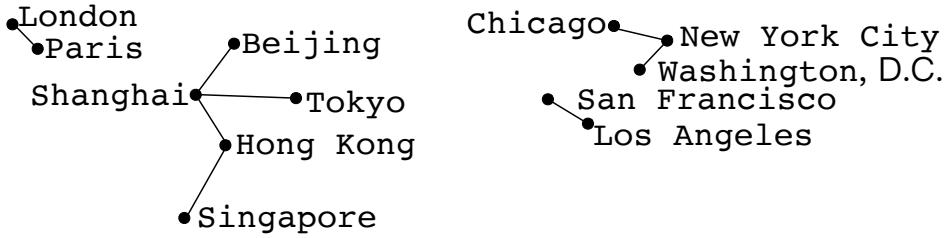


图 4.4 以全球知名城市为例的 Best Friend Forest

**定义 4.1.** 将 Best Friend Graph  $G(V, E')$  中所有有向边替换为无向边, 且在每个环中仅保留一条边, 此时的无向图被定义为  $G(V, E')$  的 Best Friend Forest  $F(V, E)$ 。

**定理 4.6.** 一个连通分量, 即 Best Friend Forest 中的一棵树  $T(V_T, ET)$ , 是其所在的完全图  $G(V_T, E)$  中的一棵最小生成树。

证明. 令  $T^*(V_T, E_{T^*})$  为完全图  $G(V_T, E)$  的一棵最小生成树。对于  $E_{T^*} - E_T$  中的一条边  $(i, j)$ , 我们在  $T(V_T, E_T)$  中得到一条无向路径。根据推论 4.5, Best Friend Graph 中连接  $i$  和  $j$  的对应的有向路径是以下三种情况之一:  $i \rightarrow \dots \rightarrow j$ ,  $i \leftarrow \dots \leftarrow j$  和  $i \rightarrow \dots \rightarrow k \leftarrow \dots \leftarrow j$ 。由于将  $(i, j)$  添加到  $T(V_T, E_T)$  会导致环, 所以路径中必然有一条边  $(u, v)$  不属于  $T^*(V_T, E_{T^*})$ 。根据引理 4.3, 在这三种任一路径下, 我们都有  $\omega(i, j) \geq \omega(u, v)$ 。而我们必有  $\omega(i, j) = \omega(u, v)$ , 否则通过将  $(i, j)$  替换为  $(u, v)$  我们会获得另一棵权重值更小的生成树, 这与  $T^*(V_T, E_{T^*})$  是最小生成树的假设相矛盾。对  $E_{T^*} - E_T$  中所有的边  $(i, j)$  重复该过程, 我们将得到与  $T^*$  具有相同权重值的  $T$ 。因此  $T(V_T, E_T)$  也是一棵最小生成树。  $\square$

图 4.4 中展示了与图 4.2 中算例相对应的 Best Friend Forest。根据定理 4.6, 图 4.4 中的每个连通分量都是最小生成树, 该图也是一个最小生成森林  $F(V, E)$ 。

### 4.2.3 层次化最小生成树网模型

针对大规模的分布式聚类, 我们在第 4.2.3 节中提出了层次化的最小生成树结构, 并设计了度量标准以自适应决定最佳聚类层。

聚类的目标是期望能够获得更高的类内紧密度和类间离散度<sup>[141]</sup>。由于聚类建立了一组层次化的结构, 我们设计了一套合理的度量标准以自适应地选择出最佳的聚类层作为回归模型的输入。如定理 4.6 中所述, Best Friend 聚类本质上在每个聚类层均生成了一个最小生成森林, 所有分散的类连在一起则为一个整

---

**算法 6 Best Friend Forest 构建**

---

**Require:**  $visited[] = 0, c[] = 0, e[] = 0, mst\_num = 0$

```

1: function INITIALIZEBESTFRIENDFOREST
2:   for  $i = 0 \rightarrow n - 1$  do
3:      $j = findBestFriend(i)$ 
4:     addEdge( $i, j, \omega(i, j)$ )
5:   end for
6: end function
7: function TRAVERSEMST
8:   for  $i = 0 \rightarrow n - 1$  do
9:     if  $!visited[i]$  then
10:      SEARCH( $i$ )
11:       $c[mst\_num] / = e[mst\_num]$ 
12:       $mst\_num = mst\_num + 1$ 
13:    end if
14:   end for
15: end function
16: function SEARCH( $i$ )
17:    $visited[i] = 1.$ 
18:    $e[mst\_num] += 1$ 
19:   while  $getNextNeighbor(i, &k)$  do
20:     if  $(!visited[k])$  then
21:        $c[mst\_num] += getEdgeWeight(i, k)$ 
22:       SEARCH( $k$ )
23:     end if
24:   end while
25: end function

```

---

体的聚类网。因此，基于最小生成树网，我们利用最小生成树的相关属性来量化聚类指标，进而分析聚类的有效性。

**定义 4.2.** 令  $T_i = (V_i, E_i)$  表示 Best Friend Forest  $F$  中的一棵最小生成树。 $T_i$  的类内紧密度  $c_i$  定义为：

$$c_i = \frac{1}{e_i} \sum_{(j,k) \in E_i} \omega(j, k), \quad (4.8)$$

其中  $e_i = |E_i|$ 。

算法 6展示了 Best Friend Forest 的构建过程和每个连通分量的距离计算。第一个函数找到所有的最好朋友边并将它们记录在全局数组中。第二个函数遍历

所有的连通分量，它找到一个未访问过的节点作为当前最小生成树的根节点，并调用第三个函数来提供给它返回值。第三个函数利用深度优先搜索 (DFS) 遍历最小生成树中的所有节点，并在两个数组  $c[]$  和  $e[]$  中更新数量和权重值。

**定义 4.3.** 对于一个由聚类生成的第  $k$  层的 Best Friend Forest  $F_k$ ，假定  $F_k$  中有  $m$  个类  $T_1, T_2, \dots, T_m$ ，且类  $T_i$  包含  $v_i$  个样本点，则  $T_i$  的类间离散度  $d_i$  定义为：

$$d_i = \min\{d(\bar{x}_i, \bar{x}_j) | 1 \leq j \leq m, j \neq i\}, \quad (4.9)$$

其中  $\bar{x}_i$  和  $\bar{x}_j$  分别是新的聚类中心， $d(\bar{x}_i, \bar{x}_j)$  是类  $T_i$  和  $T_j$  之间的欧式距离。

为了确定最优的聚类层，式 4.4 中定义了层次聚类指数 ( $HCI$ ) 作为衡量每一层聚类程度的测量指标，它结合了类内紧密度  $c_i$  和类间离散度  $d_i$ 。

**定义 4.4.** 对于一个由聚类生成的第  $k$  层的 Best Friend Forest  $F_k$ ，其中  $T_1, T_2, \dots, T_m$  分别表示该层  $F_k$  中的类，那么层次聚类指数  $HCI(k)$  被定义为类内紧密度和类间离散度的线性组合：

$$HCl(k) = \frac{1}{m} \sum_{i=1}^m \left( \frac{d_i - c_i}{d_i + c_i} \right), \quad (4.10)$$

最优的聚类层为：

$$k_{opt} = \arg \max \{ HCl(k) \}. \quad (4.11)$$

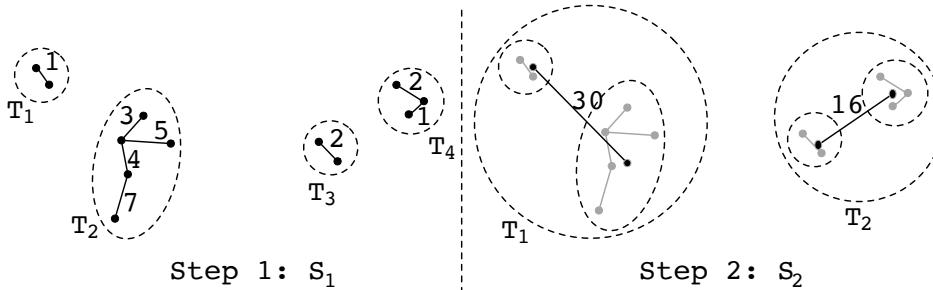


图 4.5 Best Friend 聚类生成的分层最小生成森林

图 4.5展示了全球知名城市算例中的  $HCI$  指数的计算过程。通过每条边上标有的放缩后的城市距离，我们在图 4.5中的两个子图得到了前两层聚类的最小生成森林。根据式 4.2中类内紧密度的定义，我们在  $S_1$  中有  $c_1 = 1, c_2 =$

$(3 + 5 + 4 + 7)/4 = 4.75$ ,  $c_3 = 2$ ,  $c_4 = (2 + 1)/2 = 1.5$ ; 在  $S_2$  中有  $c_1 = 30$ ,  $c_2 = 16$ 。由于在每一层中相似的顶点会被迭代地合并, 一个重要的发现是  $S_k$  中类  $i$  的  $d_i$  的值恰好是  $S_{k+1}$  中顶点  $i$  的最好朋友权重。例如, 在  $S_1$  中得到的  $d_1 = d_2 = 30$ ,  $d_3 = d_4 = 16$  是通过  $S_2$  中获取的信息计算而得的。因此, 根据式 4.10, 我们有  $HCI(1) = \frac{1}{4} \sum_{i=1}^4 \left( \frac{d_i - c_i}{d_i + c_i} \right) = 0.82$ 。类似地, 我们在  $S_2$  中通过新的  $c_1 = 30$ 、 $c_2 = 16$  和  $d_1 = d_2 = 70$  得到  $HCI(2) = 0.51$ , 而  $d_1$  和  $d_2$  的值则是这两个新合并的类之间的距离来测算的, 即是在  $S_3$  中计算的新质心之间的距离。为简洁起见, 这里我们直接给出了  $S_2$  中的值  $d_1 = d_2 = 70$ 。显然, 由于  $HCI(1) > HCI(2)$ ,  $S_1$  聚类的结果评估优于  $S_2$ 。值得注意的是,  $HCI$  不是新引入的参数, 而是衡量聚类质量的指标。它与 Best Friend 聚类结合, 用于简单地自动确定出具有最大  $HCI$  值的最优聚类层  $k_{opt}$ 。因此, 整个过程仍然是无参数的。

#### 4.2.4 数据组织模型

在每个聚类层次结构中的 Best Friend Forest 构建之后, 划分到同一个聚类中的样本指针则被交换到一起。每个样本实际上代表的是一个聚类, 该聚类由上一轮中的点聚类而成。这些点在低层次聚类中的相对位置是固定的, 即, 它们会作为一个的大指针进行整体的移动与交换。这种数据组织对于均衡分割非常重要, 因为它保证了在数组中样本的位置越接近, 则它们匹配的相似性就越多。图 4.6 中描述了每个聚类层结构中指针的交换过程。尽管伦敦、巴黎和香港在  $S_2$  层中被聚类到同一组, 但伦敦和巴黎在聚类数组中距离香港较远, 因此其包含更多相似之处。基于新设计的数据组织结构, 负载均衡方案可以简单地进行实现。数据组织的结构也随着 Best Friend 聚类算法层级的演进而进行更新, 这也只需要通过简单地指针交换完成而无需其他额外操作。

### 4.3 并行算法设计：适应分布式架构的新型机器学习算法

#### 4.3.1 快速距离计算

寻找最近邻点的过程占据了大部分的计算开销。对于含有由  $n$  个数据元素组成的顶点集  $V$  的 Best Friend Graph  $G(V, E)$ , 实践中完全不需要在大规模数据集上计算出完整的  $n \times n$  距离邻接矩阵。例如, 最好朋友可以通过具有  $O(\log(n))$  时间复杂度的快速近似最近邻方法轻松获得 (如 FLANN<sup>[142]</sup>、k-d 树<sup>[143]</sup> 等)。

我们在实现中采用了更通用的欧氏距离进行成对的距离计算, 而没有使用

这些现成的快速算法库。由于样本的数据量在每次递归时至少减少一半，我们进行平均化并使用平均向量来计算最好朋友。这也简化了计算，并将时间复杂度从  $O(n^2)$  优化到  $O(n \log(n))$ ，特别是通过我们常数级别的收敛性，时间复杂度可以达到最佳情况  $O(n)$ 。为了进一步利用现代 CPU 中向量处理单元的能力，我们将  $vl$  个数据样本分组到向量寄存器中，并以 SIMD 方式执行  $vl$  次计算，其中  $vl$  是寄存器中双精度浮点数 *double* 类型的最大存储数量。通过计算的向量化，显著地提高了计算效率。

#### 4.3.2 并行化算法

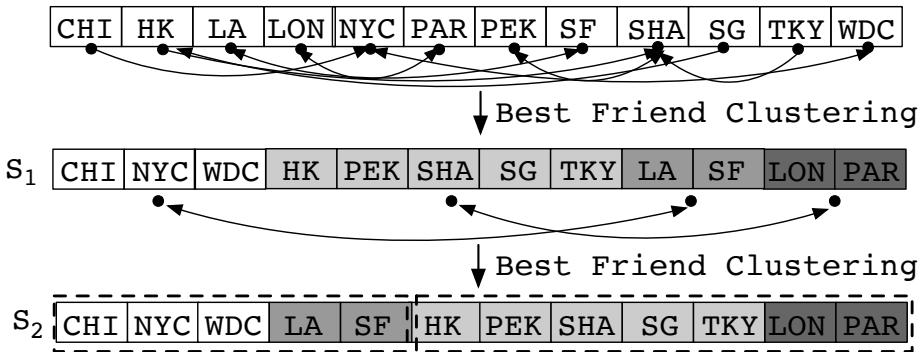


图 4.6 Best Friend 聚类中的数据组织（不同类通过颜色和线框进行区分）

为了减少计算、内存和通信的开销，我们对我们所提出的方法设计了高效的并行实现。算法 6 中第一个函数的并行化比较直接。样本的数据均匀地分布到所有处理器并相应地进行并行计算。其中的一个示例分布如图 4.6 所示，每个进程分配有 3 个样本，分别来计算它们各自的最近邻居。

Best Friend Forest 的遍历和紧密度的计算，即算法 6 中的第二个函数，本质上似乎是一个串行任务。但是，基于在第 4.2.2 节中提出的引理，我们能够在 Best Friend Forest 中利用边的信息来确定每棵单独的树。根据引理 4.2，我们可以得出每个环可以标识一棵树，而根据引理 4.4，通过搜索它的两条相等的边则很容易找到该环。

算法 7 描述了在算法 6 中第二个 TRAVERSEMST 函数的并行版本。第 1 行中所有的最好朋友边都被收集到所有的处理器上。然后，从第 4 行到第 12 行，每个处理器通过对所有  $|V|$  条边进行排序使用哈希方法来找到所有的相等边对。Best Friend Forest 中树的数量即为相等边对的数量。每个处理器遍历一组树，每

---

**算法7 算法6中的TRAVERSEMST函数并行化**


---

```

1: Gather the whole Best Friend Forest
2: hash.initialize()
3: mstNum = 0
4: for i = 0 → n - 1 do
5:   j = getBestFriend(i)
6:   if hash.exist((i, j)) then
7:     startNode[mstNum] = i
8:     mstNum += 1
9:   else
10:    hash.add((i, j))
11: end if
12: end for
13: parfor i = 0 → mstNum - 1 do
14:   SEARCH(startNode[i])
15: end parfor

```

---

棵树的遍历从环上两个节点中的任何一个开始（第14行）。总的来说，在我们的实现中通信只包含两个值的短消息即可：最近邻对和相应的最短距离。

#### 4.3.3 基于回溯的负载均衡算法

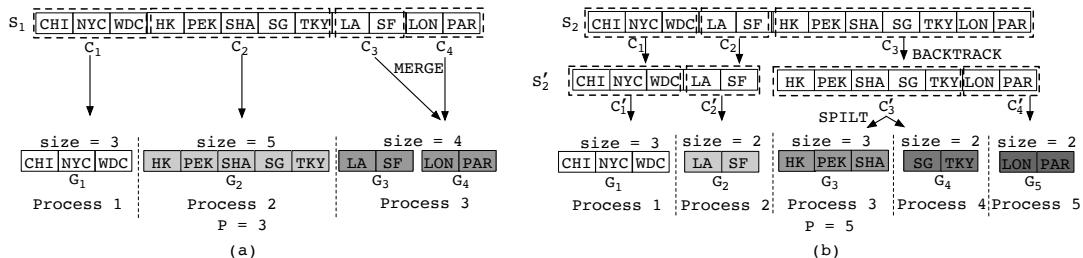


图4.7 基于回溯的分布式负载均衡算法。图4.7(a)描述了针对小任务的MERGE操作；图4.7(b)展示了基于回溯的大任务的SPLIT操作。为了便于展示，城市的名称相应地进行了缩写。

我们在第4.3.3节中提出了基于回溯的负载均衡算法，以在分布式系统上实现更优的性能。

基于对聚类结果的分析，我们观察到由聚类生成的各样本集合的划分通常是不规则且不均衡的，这使得在分布式系统上的计算也面临负载不平衡问题。因此，我们需要设计一种新的分布式负载分割算法来实现数据并行。在我们的设计中， $p$ 个计算节点上的均衡分割意味着每个节点上的样本数接近  $n_p = n/p$ 。基于聚

类的层级结构特点及 4.2.4 节中设计的数据组织模型，我们提出了一种由 MERGE 和 SPLIT 操作组成的基于回溯的分布式负载均衡算法。

对于小规模的聚类，我们执行 MERGE 操作将负载进行串联。图 4.7 (a) 展示了在聚类层  $S_1$  上使用 4 个进程的情况。由于我们有  $n_p = 12/3 = 4$ ，类  $C_3$  和  $C_4$  的规模与  $n_p$  相比过小。因此，我们按类大小对  $S_1$  进行排序，并将规模较小的类串联到同一个节点上，使得该节点的负载总大小接近  $n_p$ 。值得注意的是，在每个节点上各个模型仍然独立训练，这也说明了 MERGE 操作的目标是增大单节点上的负载至  $n_p$  的规模，而不是在各个节点上进行多个模型的混合。在图 4.7 (a) 中，聚类  $C_3$  和  $C_4$  合并到进程 3 中，作为模型  $G_3$  和  $G_4$ 。进程 1 到进程 3 上的负载规模此时分别为 3、5 和 4，从而实现了对  $S_1$  的均衡分割。

SPLIT 则用于将规模远大于  $n_p$  的大聚类进行合理分割。由于同一组聚类中的样本指针在每次迭代中作为一个整体进行移动，我们基于聚类数组的结构通过回溯机制来实现 SPLIT 操作。图 4.7 (b) 中展示了使用 5 个进程在  $S_2$  上的操作。在这种情况下，我们有  $n_p = 12/5 = 2.4$ ，然而原始的聚类  $C_3$  显然大于  $n_p$ 。因此，我们通过指针的移动轨迹对  $S_2$  的形成过程进行回溯。回溯后的  $S_2'$  包含了 2 个新形成的拆分类，其中  $C_3'$  的大小仍然大于  $n_p$ 。然后我们再次执行 SPLIT 操作，将  $C_3'$  继续分割成  $G_3$  和  $G_4$ 。因此， $G_1$  到  $G_5$  被均匀地分派到各个节点，且每组均作为独立的模型进行训练。通过利用回溯机制，我们始终能够保证经 SPLIT 操作后，彼此更相似的样本总会再次聚集到同一个类中，同时数据的空间局部性也得到了保护。

#### 4.3.4 基于 Best Friend 聚类的回归预测算法

分布式回归方法通常是构造  $p$  个独立的模型，其中  $p$  是处理器的数量（硬件并行度）<sup>[5,115]</sup>，并作为主要的输入参数对算法进行调节。

这种方法存在两个主要的缺点。首先， $p$  本质上与输入的数据无关，因此将  $p$  纳入算法的设计可能导致与数据集的内在结构不匹配。我们的方法通过使用层次化的 Best Friend Graph，有效地构建了一系列不依赖于任何预定义值的聚类结构。其次，现有方法可能需要重新组织数据以实现负载均衡。然而，在这个过程中，随着样本从一个聚类移动到其他聚类，数据之间的关系信息可能会因此丢失，损害最终模型的紧密度。

算法 8 中正式地总结了我们提出的分布式回归算法。首先，我们在伪代码的

---

**算法 8** 基于 Best Friend 聚类的分布式回归预测

---

**Require:**  $n$  samples for training,  $k$  samples for testing, best-clustered results  $A$

```

1:  $p \leftarrow$  rank of a process
2: Initialize best Mean Squared Error  $MSE^* \leftarrow \infty$ 
3: Balanced Partitions on  $n$  samples in  $A$  for  $A^p$ 
4: function REGRESSION
5:   TRAINING( $A^p$ )
6:   TESTING( $M^p$ )
7:   Reduce:  $MSE = (\sum_{p=1}^P e^p)/k$ 
8:   if ( $p = 0$ )&&( $MSE < MSE^*$ ) then
9:      $MSE^* = MSE$ 
10:    end if
11:   end function
12: function TRAINING( $A^p$ )
13:    $C^p \leftarrow$  Cluster Number in  $A^p$  on Rank  $p$ 
14:   for  $i = 0 \rightarrow C^p$  do
15:     Training Model  $M_i^p$  for Cluster  $i$  on Rank  $p$ 
16:   end for
17:   end function
18: function TESTING( $M^p$ )
19:   Initialize local MSE  $e^p \leftarrow 0$ 
20:   for  $i = 0 \rightarrow k - 1$  do
21:     if FindNearCLuster( $i$ ) =  $C^*$  then
22:       Select best  $M^*$  for prediction
23:       Update  $e^p$ 
24:     end if
25:   end for
26:   end function

```

---

第 5 行执行训练阶段。在我们的工作中，经过均衡分割后每个进程至少包含一个类，这意味着从第 12 行到第 17 行通过训练分别生成了相应的模型。然后，第 18 行到第 26 行则利用模型并行性在每个进程上进行独立地回归预测。对于给定的测试样本  $x^*$ ，如果离其最近的聚类中心  $C^*$  在进程  $p$  上，我们就使用对应的模型  $M^*$  来进行预测。如算法 8 中第 23 行所示，我们将误差在每个节点上暂时进行累积，而不是采取即时通信的模式。因此，只需要最后在第 7 行进行一次归约操作来进行统计分析。由于在 MPI 通信中，一次完整的消息比多次分散的消息开销更小，因此这种实现也进一步降低了通信开销。

## 4.4 实验评估

### 4.4.1 实验配置

**硬件架构** 我们使用 C++ 编程语言对软件库进行开发，实验运行在高性能的计算集群上。集群中的每台机器由两个时钟速度为 2.30 GHz（最高速度达 3.80 GHz）的 Intel Xeon Platinum 9242 处理器组成，它拥有 96 个物理内核，它们被组织成两个卡槽。每个处理器包含一个 71.5 MB 的智能缓存，支持 AVX512 指令集扩展，能够以 SIMD 的方式对 8 个双精度浮点数据进行向量化运算。

**基准方法** 实验分两部分进行。首先，我们使用三个经典的二维数据集对我们提出的 Best Friend 聚类进行性能评估。K-Means<sup>[116]</sup>、凝聚层次聚类（AHC）<sup>[6]</sup> 和最小生成树聚类（MSTC）<sup>[7]</sup> 方法的结果也作为不同的基准进行了对比参考。然后我们转向使用五个真实的更大规模的数据集对回归库的收敛性、准确性和可扩展性进行了性能评估。由于 DCKRR<sup>[4]</sup> 和 KRR2<sup>[5]</sup> 是两项非常相关的工作，因此本节的实验也将以它们作为两个参考基准。此外，我们也将 BKRR2 中的聚类方法用 AHC<sup>[6]</sup> 和 MSTC<sup>[7]</sup> 进行替换，作为 AHCKRR 和 MSTKRR。因此，并行回归的实验涵盖了代表性聚类方法的全部范围，即：BFCKRR (Best Friend 聚类)、BKRR2 (K-Means)、AHCKRR (凝聚层次聚类) 和 MSTKRR (最小生成树聚类)。

表 4.1 经典形状数据集参数配置

DATASET	#SAMPLES	#LABELS	#DIMENSIONS
ZAHN's COMPOUND <sup>[1]</sup>	399	6	2
AGGREGATION <sup>[2]</sup>	788	7	2
R15 <sup>[3]</sup>	600	15	2

**数据集** 表 4.1 中列出了三个经典的形状数据集用于实验中论证聚类的质量，它们的源文件可以在 Clustering Basic Benchmark (CBB)<sup>[144]</sup> 中获得。这些经典数据集代表了一系列著名的聚类问题，它们被广泛地作为基准用于论证聚类算法的有效性<sup>[1-3,144]</sup>。数据集 Million Song Data (MSD) 和 Cadata 被用作我们实验评估的两个回归数据集，因为它们都被用于 DCKRR 和 BKRR2<sup>[5]</sup> 的工作中。为了进一步证明我们方法的扩展性，我们使用了另外三个真实数据集，其包含了更高维、跨学科研究的数据。这五个数据集的详细信息按照训练集的规模，汇总在

表4.2中。所有的这些数据集源文件均可以在UCI机器学习库中找到<sup>[145]</sup>。另外，我们统一采用了欧氏距离来进行数据集中样本距离的度量，以保证比较的公平性。

表4.2 真实数据集参数配置

DATASET	#TRAIN	#TEST	#DIMENSIONS	FIELD
CADATA	18,432	2,208	8	HOUSING
PROTEINS	40,730	5,000	9	BIOMEDICINE
APS FAILURE	60,000	16,000	171	VEHICLE
MSD	463,715	51,630	90	MUSIC
GAS SENSOR	4,095,000	900,900	20	CHEMISTRY

#### 4.4.2 可视化

基于表4.1中的经典形状数据集，Best friend聚类的质量在图4.8中进行了更清晰地直观可视化展示。为了进一步定量地评估聚类的性能，我们使用调整互信息(AMI)<sup>[146]</sup>来衡量图4.8中通过不同方法获得的聚类分布与真实数据分类之间的相似性。由于K-means的性能受K值的影响较大，我们直接用真实的聚类标记数量来设置K值。结果显示，K-means生成的聚类结果较差，尤其是在Zahn的Compound<sup>[1]</sup>数据集上，它不能平滑地将非线性形状进行分离。在大多数情况下，AHC和MST聚类效果优于K-means。与上面提到的三种方法相比，我们可以观察到Best friend聚类能够比参考方法更好地对数据进行分类。

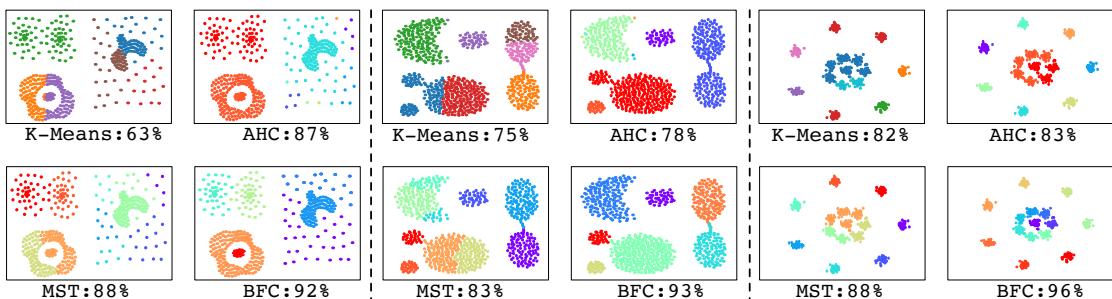


图4.8 不同的聚类方法在Zahn's Compound<sup>[1]</sup>(左图), Aggregation<sup>[2]</sup>(中图), 和 R15<sup>[3]</sup>(右图)数据集上的可视化结果(标有AMI指数)

#### 4.4.3 收敛性

假设硬件并行度包含 $c$ 个核心且所有样本的总特征量为 $N$ ,BKRR2中K-means的时间复杂度需为 $O(i \times c \times N)$ ，其中*i*是收敛所需的迭代次数。相应地，AHC<sup>[6]</sup>

表 4.3 Best Friend 聚类中的分层 *HCI* 指数

STEPS	1	2	3	4	5	6	7
CADATA	0.31	0.51	0.26	0.45	0.24	0	-
PROTEINS	0.65	0.44	0.28	0.14	0.13	0.11	0
APS FAILURE	0.30	0.56	0.27	0.35	0.58	0.50	0
MSD	0.09	0.25	0.19	0.02	0	-	-
GAS SENSOR	0.22	0.38	0.62	0.24	0.17	0.13	0

表 4.4 Best Friend 聚类中的分层聚类个数

STEPS	0	1	2	3	4	5	6	7
CADATA	18,432	4,737	326	64	8	2	1	-
PROTEINS	40,730	11,273	724	191	32	6	2	1
APS FAILURE	60,000	15,230	712	124	11	4	2	1
MSD	463,715	54,307	1,023	60	7	1	-	-
GAS SENSOR	4,095,000	231,541	13,115	1,721	202	10	2	1

和 MST<sup>[7]</sup> 的时间复杂度可以简单地概括为  $O(N^2/c)$ 。由于 Best Friend 聚类递归地每次至多留下一半的聚类数量完成分组，因此其可以在  $O(\log N)$  轮中达到快速收敛。表 4.3 和表 4.4 中分别给出了 *HCI* 和每层聚类个数的详细信息，其中突出显示了最佳收敛层。在 Gas Sensor 等大规模数据集上，Best Friend 聚类仍能实现常数级别的收敛，说明我们方法的时间复杂度即使是对较大的训练规模也不太敏感，适合大规模的扩展性算例。

#### 4.4.4 准确度

我们提出的均衡分割策略对模型准确度的影响较小，因为它将具有最多相似性的样本保留在同一分类中。尽管如此，为了保证我们实现的正确性，我们将实验的参数配置从 96 个核心扩展到了 12,288 个核心，通过报告的准确度（由 MSE 测量）进行比较。为了保证实验的公平性，我们从可选参数范围内进行微调，以在不同的方法中实现最低的 MSE。如图 4.9 所示，当采用 KRR 技术时，BFCKRR 实现了最低的 MSE。此外，BFCLR 和 BFCSV 在大多数情况下也可以做出更好的预测。在所有的方法中，DCKRR 和 BKRR2 的预测质量较差，这两者分别采用的是无聚类和 K-means 聚类方法。随着计算核心的增加，我们的回归库持续稳定地获得了较高的精度，而其他方法的 MSE 则波动明显。这说明我们的均衡分割的负载均衡算法极大地支持了分布式实验的准确性。

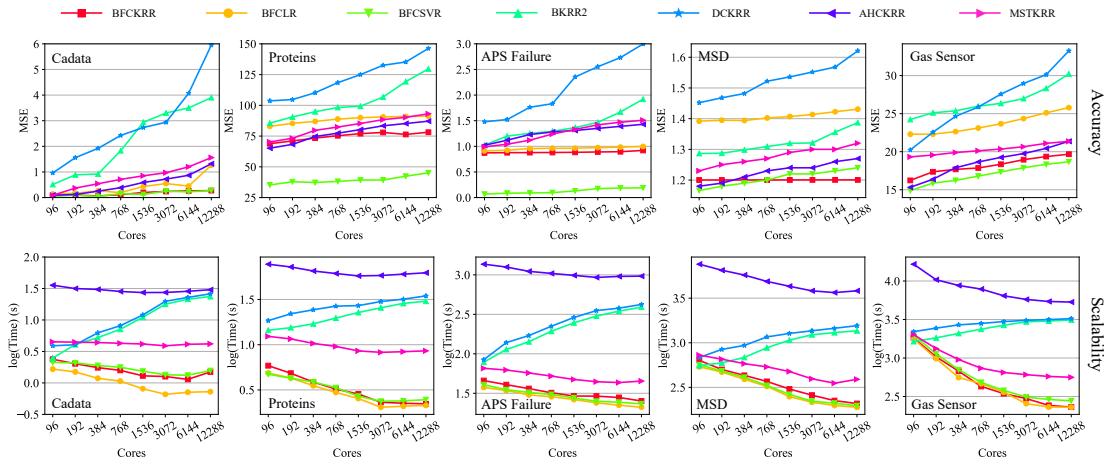


图 4.9 在真实数据集上不同方法相同配置下的准确度和可扩展性。我们的并行库实现了包括核岭回归 (KRR)、线性回归 (LR) 和支持向量回归 (SVR) 在内的回归技术。它们与 Best Friend 聚类方法相结合，分别缩写为 BFCKRR、BFCLR 和 BFCSVR。与代表性聚类方法相结合的并行回归基准算法分别是 DCKRR (无聚类的分治方法)<sup>[4]</sup>，BKRR2 (K-Means)<sup>[5]</sup>，AHCKRR (凝聚层次聚类)<sup>[6]</sup> 和 MSTKRR (最小生成树聚类)<sup>[7]</sup>。

#### 4.4.5 扩展性

图 4.9 也展示了不同方法在五个数据集上的可扩展性。我们观察到，与基准方法相比，我们的并行框架始终实现了较高的性能。对于 Gas Sensor 这样较大的数据集，不同方法之间的性能差距进一步扩大，此时我们的方法甚至比 BKRR2 快 13.6 倍以上。此外，随着核心数量翻倍，我们并行框架所需的时间相应地减少，而 DCKRR 和 BKRR2 方法的扩展性能都随着计算核心的增长而变差。这说明基于 K-means 或基于分治的方法在大规模回归中地应用性能较差，这分别由  $K$  值升高带来的聚类数量增加和开销较大的分治操作所致。通过类似的并行实现，MSTKRR 在大多数情况下都优于 BKRR2。然而，AHCKRR 的性能最低，因为凝聚式算法固有地具有平方时间复杂度。

为了进一步剖析我们的并行框架并了解它是如何随着数据集大小的增长而变化，我们对两个典型数据集的算例进行了定量研究，即实验中规模最小和最大的两个数据集。图 4.10 中比较了我们并行框架中聚类 I/O、聚类、回归 I/O、回归以及通信所占的对数时间。这里的 I/O 操作表示除通信之外的非计算部分的开销。很明显，增加的计算核心也加剧了通信和 I/O 成本。尽管通信和 I/O 带来了扩展性的压力，我们的并行框架仍然获得了持续的扩展性能。表 4.5 中展示了两个数据集的各项操作的运行时间所占比和加速比。有趣的是，虽然不同数据集上相同操作所占的平均比例差异不大，但我们的并行框架可以在 Gas Sensor 数据

集上获得更高的加速比。基于对图 4.10 的联合分析，我们可以推出关键点在于可并行化的操作（聚类和回归部分）在大规模数据集上有着更好的扩展效率。

表 4.5 Cadata 和 Gas Sensor 数据集上并行框架中不同操作的时间占比和加速比

Dataset	Cadata								Gas Sensor							
	Component	C.I/O(%) <sup>1</sup>	C.(%)	R.I/O(%)	R.(%)	M.(%)	S.(C.+R.) <sup>2</sup>	S.	C.I/O(%)	C.(%)	R.I/O(%)	R.(%)	M.(%)	S.(C.+R.)	S.	
96		2.80	63.98	0.34	28.28	0.84	1.00	1.00	1.88	64.84	0.35	31.51	0.89	1.00	1.00	
192		3.58	54.96	0.54	25.75	1.49	1.34	1.18	3.64	62.33	0.67	29.38	3.04	1.85	1.76	
384		4.54	46.88	1.54	25.15	4.00	1.74	1.36	5.97	53.34	1.22	27.59	8.67	3.21	2.70	
768		5.94	40.41	2.21	22.73	10.73	2.19	1.50	11.00	46.31	2.35	22.80	15.53	5.95	4.27	
1536		7.94	31.62	3.39	21.60	25.45	3.18	1.83	14.00	37.21	3.09	21.45	21.79	8.68	5.28	
3072		8.48	25.35	4.33	18.22	34.86	3.99	1.88	16.50	26.83	3.78	16.60	29.45	13.61	6.14	
6144		10.09	27.21	5.18	14.04	43.01	4.66	2.09	20.27	18.84	4.52	15.54	37.01	21.09	7.53	
12288		8.29	20.73	4.48	12.70	39.45	4.38	1.59	21.75	17.46	4.92	14.86	39.12	23.60	7.92	
Mean		6.46	38.89	2.75	21.06	19.98	2.81	1.55	11.88	40.90	2.61	22.47	19.44	9.87	4.57	

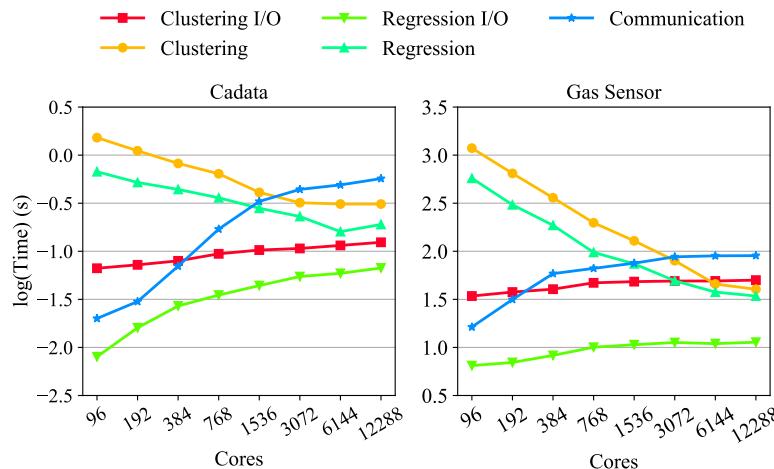
<sup>1</sup> 为了使表达更清楚，聚类、回归和通信过程分别缩写为 C.、R. 和 M.。<sup>2</sup> 加速比也相应地缩写为 S.。

图 4.10 Cadata 和 Gas Sensor 数据集的时间剖析

#### 4.4.6 实验讨论

在本节中，我们将对实验部分进行快速回顾，以从不同方面梳理出我们的贡献点。

我们首先在第 4.4.2 节研究了 Best Friend 聚类在经典形状数据集上的分类性能，并与三个经典基准算法进行了比较。通过直观的可视化对比，我们的方法可以取得更好的分类效果。然后第 4.4.3 节中的收敛性实验表明，即使面对数百万量级的数据集，我们的方法也可以稳定地实现常数级别迭代的收敛。第 4.4.4 节

中的准确度实验则将测试算例扩展到了 12,288 个计算核心。由于我们提出的负载均衡算法能够捕获了更多的样本之间相似性，因此在分布式系统上获得了更优的结果。最后，可扩展性实验表明，我们利用 Best Friend 聚类实现的并行回归框架在大规模算例中结果均优于其他方法，实现了更好的扩展性能。

#### 4.5 本章总结

在本章中，我们提出了一种更准确、更快速、同时无参数的 Best Friend 聚类方法。然后，我们对所提出的聚类方法的理论特性进行了深入研究，基于层次化的最小生成树结构特征设计了度量标准以自适应决定最佳聚类层。此外，受回溯机制的启发，我们设计了均衡分割算法用于并行实现中的负载平衡。最后，我们将提出的方法与回归技术集成为一个并行回归框架，在收敛性、准确性和可扩展性方面表现出卓越的性能。



## 第5章 粗粒度并行：百万核量级核材料辐照科学计算应用优化

### 5.1 本章概述

粗粒度并行问题一般指较大的子任务级并行。在粗粒度并行中，一个程序将被分成若干大的子任务。因此，大量的计算将在各处理器进程中同步进行，极有可能会导致负载不平衡，即其中某些进程处理大量数据，而其他进程可能处于空闲状态。此外，进程之间一般采用的是消息传递的方式进行数据通信。由于进程之间同步开销较大，高效的并行通信算法设计对程序的可扩展性至关重要。

近年来，以数值模拟方法为核心的计算物理学、计算化学等多领域应用的科学计算交叉学科逐渐兴起，而动力学蒙特卡罗方法和分子动力学模拟方法是用于材料科学计算研究的两种最主要的数值模拟方法。材料科学计算应用优化属于典型的粗粒度并行问题。首先，科学计算中高精度、长时间的数值模拟带来了大量的计算，这需要将其切割成子任务，在大规模的超算系统上运行。近年来，世界超算应用领域的风向标“戈登贝尔奖”的大部分科学计算应用多数已超过万核量级，最新的“戈登贝尔奖”（2021年）获奖应用SWQSIM使用了国产异构众核计算单元的新一代神威超算系统，更是实现了千万核的并行可扩展。其次，通信问题一直是大规模并行应用的扩展性瓶颈。由于大量进程间存在的数据通信同步需求，通信的时间往往占据了程序运行的主要时间，造成计算资源的严重

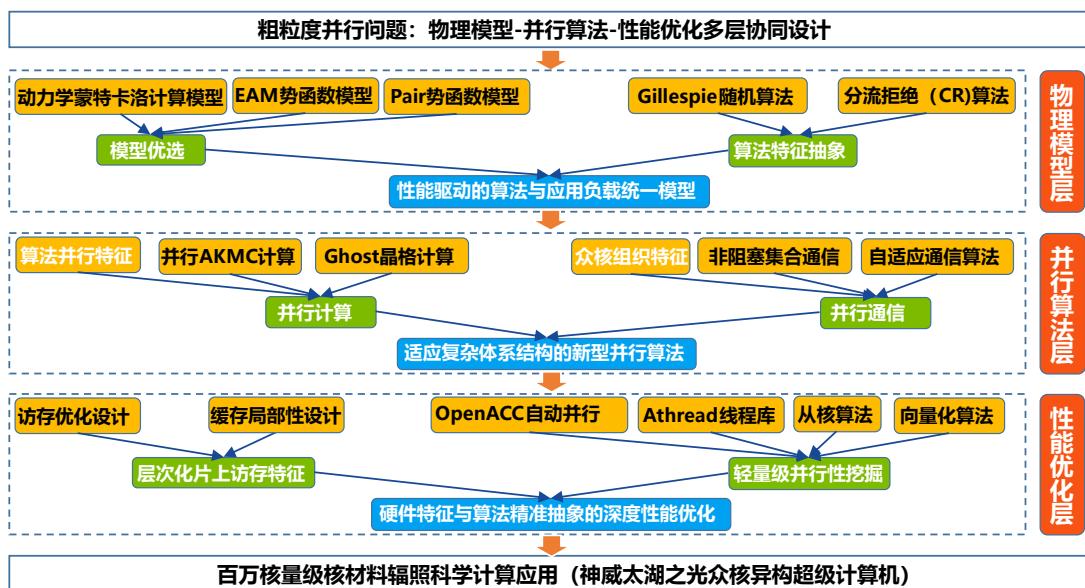


图 5.1 物理模型-并行算法-性能优化多层协同设计

浪费。如何提升计算通信比，增强程序的可扩展性，一直是大规模并行应用的痛点问题。

如前所述，对于 stencil 并行数值算法的细粒度并行问题和机器学习框架研发的中粒度并行问题，可分别采取并行算法-性能优化或物理模型-并行算法的双层协同设计方法，以提升程序或应用的扩展性能。相较于细粒度和中粒度的并行性问题，粗粒度的大规模材料模拟应用所面临的可扩展性挑战更典型、更复杂。如图 5.1 所示，根据多层次协同设计理论中的定理 2.1、原则 2.2 和原则 2.3，对于可扩展的粗粒度并行问题，需要从物理模型、并行算法和性能优化三个层次进行联合研究，从研究初始就面向可扩展性问题。

在本章中，我们将以国产核材料辐照损伤大规模应用为例进行可扩展性的优化研究。图 5.1 中展示了其不连续非线性可扩展的协同设计方案。物理模型层主要考虑高可扩展的理模型和数值算法；中间层的并行算法则更要考虑面向大规模分布式系统上的高效并行计算和通信算法设计；性能优化针对架构特点，进行读存模型和轻量级并行的实现优化。这一思路从上到下完整地考虑了从物理模型建立到最后并行程序优化的高性能计算大规模应用优化研究链，系统地研究并分层次考虑了应用的可扩展性，从理论上支持了其百万核量级的并行性设计。

### 5.1.1 引言

核能是核反应释放的能源，其常被核电站用于提供全天候、无排放的清洁环保电力。目前世界上大约 13% 的电力是由核能提供的<sup>[9,147]</sup>。然而，核反应堆组件的负载逐渐使它们产生损伤并导致其功能和效率的下降。核反应堆压力容器退化的主要原因可归于在中子辐照下 RPV 钢中富铜（Cu）沉淀物的形成。在轻水反应堆中，中子辐照是造成压力容器钢脆化的原因。核辐照技术表明，钢的硬化与中子辐射诱导原子团簇的形成同时发生。虽然这些原子团簇的组成成分仍然存在争议，但大多数原子团簇中均富含铜、镍（Ni）、锰（Mn）和硅（Si）等溶质原子。当使用小角度中子散射和场发射扫描透射电子显微镜进行观察时，这些原子团簇又类似于沉淀物。

模拟这些团簇在主体铁（Fe）基质中的形成需要更好地了解这些元素在体心立方 Fe 中的能量特性以及它们的迁移能，以便获得有关力学的信息。这些团簇元素原子水平的基本性质大部分在实验上是未知的，其可以通过从头计算

(Ab Initio) 方法获得。因此，大量相关工作已经利用从头计算来研究由位移级联产生的点缺陷与 Cu、Ni、Mn 和 Si 溶质原子之间存在的相互作用类型，这些计算对于研究辐照下压力容器钢微观结构的长期演变至关重要。现在，这些从头计算的数据正用于使用动力学蒙特卡罗 (kinetic Monte Carlo, KMC) 方法构建的物理模型，以研究点缺陷与 Cu、Ni、Mn 和 Si 溶质原子之间存在的相互作用<sup>[9]</sup> 和模拟稀 FeCuNiMnSi 复合合金的微观结构演变<sup>[148,149]</sup>。

蒙特卡罗方法包括原子动力学蒙特卡罗 (Atomic Kinetic Monte Carlo, AKMC)<sup>[150]</sup>、物体动力学蒙特卡罗 (Object Kinetic Monte Carlo, OKMC)<sup>[151–153]</sup> 和事件动力学蒙特卡罗 (Event Kinetic Monte Carlo, OKMC, EKMC)<sup>[154]</sup> 三类不同的模拟方法。本章对杂质溶质簇形成过程的研究采用了在原子水平上进行模拟的原子动力学蒙特卡罗方法<sup>[36,37]</sup>，并在模拟系统中引入了点缺陷。如上所述，在这样的模拟中，空位的跃迁能量非常重要，而这些基本的性质在原子水平上的实验上是未知的，其可以通过从头计算而得<sup>[155,156]</sup>。

许多研究团队一直致力于开发不同的 AKMC 软件来模拟 RPV 中金属合金的微观结构演变<sup>[9,157,158]</sup>。然而，此类软件大多只有串行版本，且均广泛地采用常规经验势模型。因此，它们的计算效率在大规模模拟中成为严重的性能瓶颈。我们提出了一个新型并行 AKMC 模拟软件，OpenKMC，以加速对核材料演化的研究进程。我们在能量计算中使用了优化的 Pair 势模型，并将每个进程的工作负载划分为更细粒度的计算象限，以避免边界粒子的跃迁冲突。此外，我们还提出了一系列的优化策略来减少通信时间、提高访存效率。

在实际模拟中，许多研究需要使用超级计算机对 KMC 模拟进行计算加速，以实现更大体系的核材料损伤模拟。因此，我们在神威 SW26010 处理器上实现并进一步优化了我们的 OpenKMC 软件。神威 SW26010 处理器也是世界领先的国产超级计算机神威·太湖之光的主要构建单元。

本文的主要贡献包括：

- 我们采用了优化的对势模型来获得相互作用能而不涉及冗余计算。
- 我们将数据划分为细粒度的象限以在共享边界处解决跃迁冲突，并使用分组反应模型来加速每次模拟中的事件选择。
- 设计缓存优化策略，提升访存的效率。
- 我们对通信进行了一系列优化，以消除通信冗余、减少同步时间以及自适

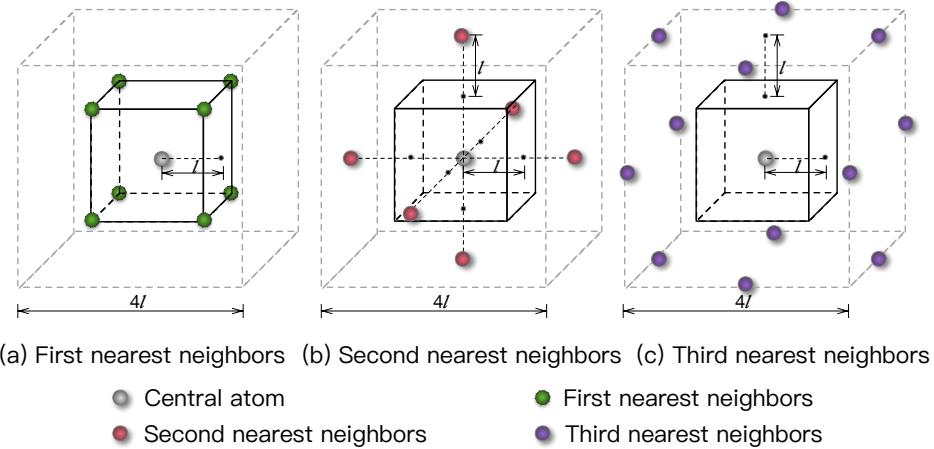


图 5.2 体心立方 (BCC) 晶格的三种最近邻原子分布图

应地调节通信频率。

- 我们设计了从核转录-翻译-传输算法并实现计算的向量化,以加速 SW26010 处理器上的计算过程。

### 5.1.2 背景

**原子动力学蒙特卡罗方法** 原子动力学蒙特卡罗方法是主导微观系统演化的一个精确模拟方法。基于空位跃迁的 AKMC 方法是以空位作为跃迁对象, 研究因其跃迁而导致材料中金属溶质析出的过程。从物理的角度出发, 金属晶体大致上都可认为具有紧密排列、高对称性的简单物理结构。通常情况下, 这些金属晶体的结构可以分为面心立方 (FCC)、体心立方 (BCC)、简单六方 (HCP) 等主要的结构类型。912°C (1,674 °F) 以下, 纯铁为体心立方结构, 称为  $\alpha$ -Fe, 为核材料辐照损伤中的主体溶剂原子。在材料的微观视角下, 模拟中的所有事件都可以认为是发生在 Fe 的体心立方结构中, 而点缺陷的扩散跃迁则可以看作是一个空位与其周围 8 个最近邻原子的碰撞交换。图 5.2 展示了在 BCC 晶格 (单位长度为  $l$ ) 中对于一个中心原子, 其第一、第二和第三最近邻原子的分布示意图。如图 5.3 所示, AKMC 算法一般可以通过四个基本步骤来构建。对于每个空位, 计算从当前状态衍生的各种可能的跃迁概率是 AKMC 算法的第一步, 这直接依赖于相对应的转移速率。在演化中, 一次跃迁事件  $X$  被表示为玻尔兹曼因子 (Boltzmann Factor) 概率, 即:

$$\Gamma_X = \Gamma_0 \cdot \exp\left(-\frac{E_a^X}{kT}\right). \quad (5.1)$$

$\Gamma_0$  是一个初始尝试频率 ( $s^{-1}$ )， $T$  是绝对温度， $E_a^X$  为事件跃迁能 (eV)。根据物理学的相关定义，初始尝试频率  $\Gamma_0$  被设置为  $6 \times 10^{12} s^{-1}$ 。将计算时间和结果的精度考虑进去，通过使用环境依赖模型，我们可以得到事件的跃迁能，其满足以下的平衡规则<sup>[153]</sup>：

$$E_a^X = E_a^0 + \frac{1}{2} \cdot (E_f - E_i)。 \quad (5.2)$$

使用的参考活化能  $E_a^0$  (Fe:0.65 eV, Cu:0.56 eV) 只取决于与空位交换位置的迁移原子的化学性质。 $E_i$  和  $E_f$  分别为空位跃迁前、跃迁后反应系统的总能量<sup>[150,159]</sup>。

论文使用式 5.3 中的时间驻留算法<sup>[160]</sup> 完成模拟中时间增量  $\Delta t$  的计算，其成比例于依据其发生概率生成的随机数所激发的所有可能的跃迁事件频率之和的倒数，因此，事件的传播也与每个独立的事件相关联。

$$\Delta t = \frac{-\ln r}{\sum_{X=1,8} \Gamma_X} \quad (5.3)$$

下一步是依据概率为每个空位随机赋值一个权重，以确定每个空位的跃迁，并且更新由这些空位跃迁所引起的体系能量的改变。与此同时，时间增量 ( $\Delta t$ ) 也积累到模拟的时间中。最后，重复 AKMC 方法的上述步骤，直到模拟时间已达到预设的时间阈值。

为了获得实际的物理时间尺度，蒙特卡罗模拟的时间尺度需要被重新进行缩放<sup>[9]</sup>：

$$t_{real} = \frac{C_{V,MC-sim}}{C_{V,real}} t_{MC-sim}, \quad (5.4)$$

其中

$$C_{V,real} = \exp\left(-\frac{E_{for}(V^{Fe})}{kT}\right) \exp\left(\frac{\delta S}{k}\right) \\ \times \left[ 1 - 8x_{Cu} - 6x_{Cu} + 8x_{Cu} \exp\left(\frac{E_{b(V-Cu)}^{(1)}}{kT}\right) + 6x_{Cu} \exp\left(\frac{E_{b(V-Cu)}^{(2)}}{kT}\right) \right]。 \quad (5.5)$$

式中  $E_{for}(V^{Fe})$  为纯铁中的空位形成能，其计算公式如下：

$$E_{for}(V^{Fe}) = 8\varepsilon_{(Fe-V)}^{(1)} + 6\varepsilon_{(Fe-V)}^{(2)} - 4\varepsilon_{(Fe-Fe)}^{(1)} - 3\varepsilon_{(Fe-Fe)}^{(2)}。 \quad (5.6)$$

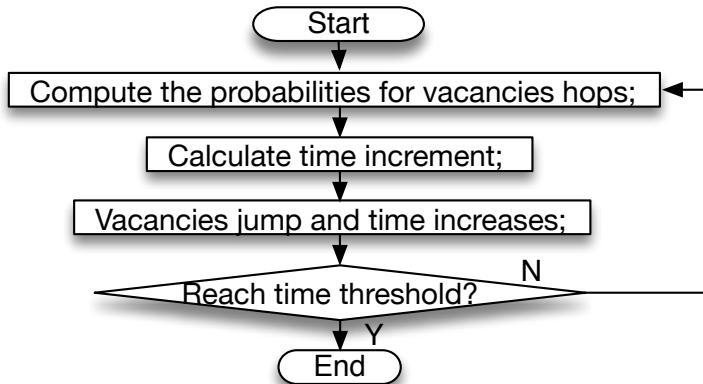


图 5.3 基于空位跃迁的串行原子动力学蒙特卡罗 (AKMC) 方法

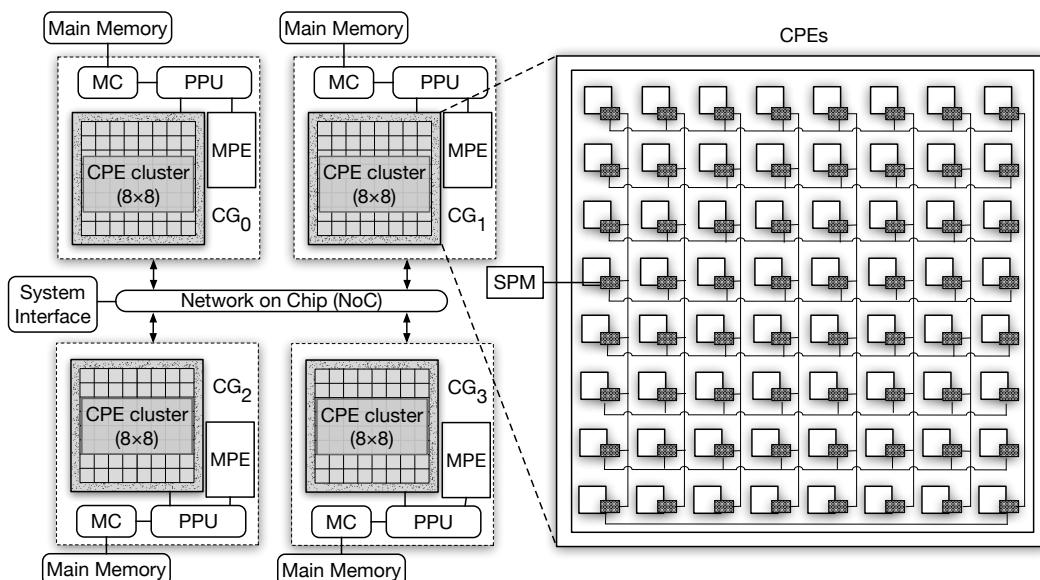


图 5.4 神威 SW26010 多核处理器的基本架构

本章的工作中将上式  $E_{\text{for}}(\text{V}^{\text{Fe}})$  的值根据物理模型进行了微调设置为 1.995 eV，而参考文献中<sup>[9]</sup>该项的值设置为 2.02 eV。 $E_{\text{b}(\text{V}-\text{Cu})}^{(1)}$  (本章工作中设置为 0.17 eV) 和  $E_{\text{b}(\text{V}-\text{Cu})}^{(2)}$  (本章工作中设置为 0.19 eV) 分别是第一或第二最近邻相互作用中的空位-Cu 结合能<sup>[9]</sup>； $x_{\text{Cu}}$  为溶质铜浓度； $\frac{\delta S}{k}$  则根据 Mathon 的工作在 Fe-Cu 系统上设置为 2<sup>[161]</sup>。

**神威·太湖之光众核架构** 世界领先的超级计算机神威·太湖之光，目前性能世界排名第四，其峰值性能超过 125 Pflops。神威·太湖之光主要搭载中国定制的 SW26010 众核处理器<sup>[162]</sup>。

图 5.4 展示了 SW26010 处理器的基本架构。每个处理器由四个核组 (Core

Group, CG) 组成，它们通过片上网络 (Network-on-Chip, NoC) 相互连接。每个核组包括 65 个核心：1 个管理处理单元 (Management Processing Element, MPE, 简称主核) 和 64 个计算处理单元 (Computing Processing Elements, CPE, 简称从核)，它们被组织为  $8 \times 8$  的网格。处理器通过系统接口 (System Interface, SI) 连接到其他外部设备。主核和从核的时钟频率均为 1.45GHz 且支持 256 位宽度的向量指令。每个核组的理论峰值内存带宽约为 34.1 GB/s，双精度峰值性能约为 765 GFlops<sup>[163]</sup>。

神威 SW26010 架构与其他多核或众核处理器有很大不同。在内存层次结构方面，主核上一级数据或指令缓存均为 32 KB，且配置了统一 256 KB 的二级缓存。每个从核都有一个 16 KB 的一级指令高速缓存和一个 64 KB 的本地存储器 (Scratch Pad Memory, SPM, 用户控制暂存器)。对于从核阵列中的内部通信，其提供了寄存器通信机制，通过 8 列和 8 行的通信总线进行数据的快速传输。从核阵列中的寄存器通信为建立快速数据传输通道提供了可能性，从而可以在从核层面获得显著的数据共享能力<sup>[162,164]</sup>。

### 5.1.3 相关工作

近年来，国内外不同的研究团队一直致力于开发更准确地用于模拟反应堆压力容器金属合金材料的微观结构演变的方法。在法国拥有 58 座发电厂的 Électricité de France (EDF) 团队开发了一款名为 LAKIMOCA<sup>[9,154]</sup> 的核材料辐照损伤模拟软件。金属合金中原子级别的行为可以通过该软件进行准确模拟，并适用于长期动力学行为的预测。然而，该款 AKMC 软件只有串行版本，因此对于大体系的模拟可能非常耗时。KMCLib<sup>[36]</sup> 支持自定义的 KMC 模型。为了更好地实现负载均衡，KMCLib 在每一步都将格点与进程进行重新匹配，所以在运行大规模模拟时开销较大。SPPARKS<sup>[165]</sup> 是一款并行 KMC 软件，其使用同步象限算法<sup>[166]</sup> 作为其并行化的算法支持。SPPARKS 中包括几个蒙特卡罗的算法模型，如空位扩散模型、铒晶格上的 H/He 扩散模型、Ising 模型、薄膜模型和晶粒生长的 Potts 模型。Crystal-KMC<sup>[167]</sup> 也是一款并行 KMC 软件，但其只能模拟纯 Fe 体系中的空位扩散过程。此外，他们的工作没有验证正确性，800 核下的 Crystal-KMC 可扩展性仅为 56%。SPPARKS 和 Crystal-KMC<sup>[167]</sup> 都无法处理金属合金体系中的空位扩散，例如 Fe 晶格中的空位促进型 Cu 沉淀过程。Jiménez 等人设计了一款基于 GPU 的 OKMC 软件来模拟辐照金属材料的演化<sup>[168]</sup>。尽管

使用了 GPU 硬件进行了优化加速，但是 AKMC 相关的模型不受支持，无法模拟金属合金中点缺陷的扩散。E. Martínez 等人开发了同步并行动力学蒙特卡罗 (spKMC)<sup>[169]</sup> 及对离散晶格支持的相关扩展包<sup>[170]</sup>。Ising 模型基于同步时间分解算法在 spKMC 中实现并可扩展到 256 个核，模拟体系达到 11 亿个原子，并行效率约为 82%。N.J. van der Kaap 和 L.J.A. Koster 开发了一种并行动力学蒙特卡罗模拟<sup>[171]</sup>，它在通用图形处理器 (GPGPU) 上运行。然而，它的应用也仅限于 GPGPU 上的载流子传输的模拟。在他们的工作中总共使用 4,096 个核心<sup>[171]</sup>，并行效率在不同浓度下从 85% 到 0.4% 显著变化。尽管各种 KMC 模拟软件可以帮助我们理解并行化的机制，并且现代计算集群提供了丰富的计算资源，但设计一整套并行 AKMC 模拟软件仍然具有相当大的挑战性，尤其是在具有复杂众核架构的神威处理器上。

在并行化 KMC 模拟时，每个进程必须与周围邻居进程同步以进行数据通信完成相关依赖项的计算。在现代处理器上，尤其是多核架构，多次同步的成本非常昂贵，甚至这种通信会占据整个模拟的大部分时间。此外，神威的众核架构上并没有高速缓存，每个众核仅有 64 KB 的数据暂存器。如果不相应地设计算法进行优化，这种硬件架构会使得应用频繁地进行访存，加剧带宽的压力。最后，神威 SW26010 处理器上的直接内存访问 (DMA) 操作只有在访问的地址在 128 字节对齐的情况下才能达到其峰值带宽<sup>[44,164]</sup>。因此，需要设计时考虑到每个缓存行的大小应是原子数据结构大小的倍数以完成最大化的对齐，否则将进一步降低访存的性能。

## 5.2 物理模型抽象：动力学蒙特卡罗计算模型建立

为了在大规模分布式集群中实现高效的 KMC 模拟，我们研发了 OpenKMC 核材料辐照损伤应用，并从物理学角度首先进行动力学蒙特卡罗计算模型建立，提出了优化的 Pair 势函数计算模型和分组反应模型。

### 5.2.1 Pair 势函数模型

在 5.1.2 节中第一步计算能量时，大量的相关文献利用原子间的经验势能提出了很多的计算模型。其中一类经典的经验势模型是嵌入式原子方法 (embedded-atom Method, EAM)<sup>[39]</sup>，方法中原子的相互作用能由对势（最多可计算到第三近邻

层) 和多体势(环境依赖项) 所叠加产生:

$$E_{\text{EAM}} = \sum_{\mu} \left[ \frac{1}{2} \sum_{i=1}^3 \sum_{\nu} \epsilon_{(\mu\nu)}^{(n)} + F(\rho_{\mu}) \right], \quad (5.7)$$

公式中  $\mu, \nu$  是原子索引,  $F$  是多体嵌入能,  $\rho_{\mu}$  是在原子  $\mu$  处插值产生的体系电子总密度。EAM 势很好地描述了金属原子之间的相互作用, 是描述金属体系最常用的一种势函数模型<sup>[38,39]</sup>。但是, EAM 势需要从 *ab initio* 计算法产生, 且目前对于复杂的 FeCuNiMnSi 合金和实际核工程模拟所需的间隙体没有相应的 EAM 势。

另一方面, 出现在 20 世纪 80 年代左右的 Pair 势模型(对势)<sup>[172]</sup>可以比较好地描述除金属和半导体以外的几乎所有无机化合物, 也包括可以处理这样的复杂合金<sup>[9]</sup> 和间隙体<sup>[148]</sup>。有些对势是经过一定的理论分析而得到的, 但其中一些参数则需要根据宏观实验参数用经验方法来确定, 这些宏观实验参数主要有弹性常数、平衡点阵常数以及内聚能、空位形成能和层错能等, 这些称为半经验势。Pair 势模型在计算原子间能量时考虑两层近邻的位置, 构成系统的所有实体之间的化学相互作用能的计算方法如式 5.8 所示:

$$\begin{aligned} E_{\text{pair}} = & \sum_j \epsilon_{(\text{Fe-Fe})}^{(i)} + \sum_k \epsilon_{(\text{V-V})}^{(i)} + \sum_l \epsilon_{(\text{Fe-V})}^{(i)} \\ & + \sum_m \epsilon_{(\text{Fe-Cu})}^{(i)} + \sum_n \epsilon_{(\text{V-Cu})}^{(i)} + \sum_p \epsilon_{(\text{Cu-Cu})}^{(i)}. \end{aligned} \quad (5.8)$$

在式 5.8 中, 当  $i = 1$  或  $2$  时, 其分别对应于第一近邻或第二近邻,  $j$  为 Fe-Fe 键的个数、 $k$  为 V-V 键的个数、 $l$  为 Fe-V 键的个数、 $m$  为 Fe-Cu 键的个数、 $n$  为 V-Cu 键的个数、 $p$  为 Cu-Cu 键的个数。本章工作中 Pair 势函数模型的式 5.8 使用的优化参数见表 5.1。

Pair 势模型的局限性在于缺乏多体势能, 因此该模型不能考虑局部环境的影响。此外, 对势参数也需要通过实验或热力学数据仔细进行对照调整<sup>[9,148]</sup>。本工作中使用的对势参数取自经典物理文献<sup>[148]</sup> 中的数据, 它可以重现 EAM 模型的模拟结果并能够对稀 FeCuNiMnSi 合金的相关研究进行扩展。

由于物质系统的复杂性以及原子间相互作用类型的不同, 很难得到满足各种不同体系和物质的一般性而又精度较高的势函数模型。针对核材料的辐照损伤

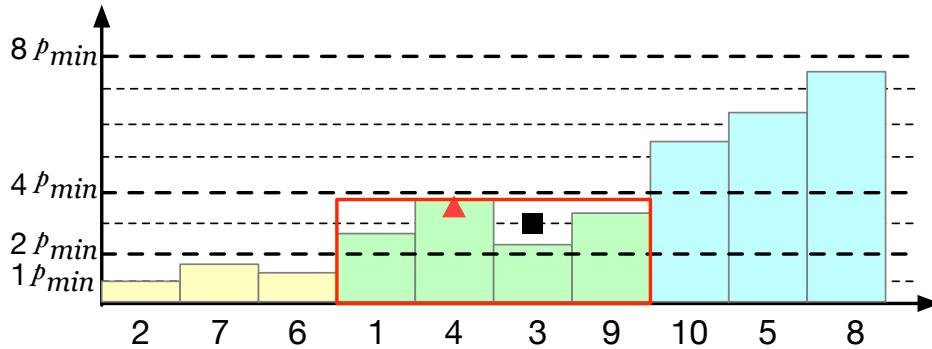


图 5.5 事件选择和更新的分组反应模型（红色三角形点落入的事件被选中，与其相邻的黑色正方形点的事件被拒绝）

模拟，大量的实验表明，本节工作通过实现 Pair 势模型对 EAM 势进行替换，可以在保证精度的前提下，大大提高模拟的效率。

### 5.2.2 分组反应模型

#### 算法 9 Gillespie 随机模拟算法

- 1: Initialize simulation.
- 2: **repeat**
- 3:     Generate random number  $r$  bounded by 0 and 1.
- 4:     Find the smallest  $m$  such that  $r \cdot p_s < \sum_{i=1}^m p_i$ , where  $p_i$  the propensity for each reaction,  $p_s$  the sum of  $p_i$  for all  $N$  reactions.
- 5:     Perform  $m$ th reaction and recompute  $p_i$  and  $p_s$ .
- 6: **until** Reach the threshold.

在 KMC 模拟的每次迭代中，都需要根据事件的相对概率从事件列表中随机选取一个事件。在核材料的模拟中，通常是需要使用动态离散概率分布中的随机

表 5.1 本章工作及参考文献<sup>[9]</sup> 中使用的 Pair 势键能 (eV)

第一近邻			第二近邻		
键能	优化参数	参考文献 <sup>[9]</sup>	键能	优化参数	参考文献 <sup>[9]</sup>
$\varepsilon_{(V-V)}^{(1)}$	0.126	0.315	$\varepsilon_{(V-V)}^{(2)}$	-0.014	-0.214
$\varepsilon_{(V-Fe)}^{(1)}$	-0.163	-0.161	$\varepsilon_{(V-Fe)}^{(2)}$	-0.163	-0.161
$\varepsilon_{(V-Cu)}^{(1)}$	-0.102	-0.103	$\varepsilon_{(V-Cu)}^{(2)}$	-0.180	-0.206
$\varepsilon_{(Fe-Fe)}^{(1)}$	-0.611	-0.778	$\varepsilon_{(Fe-Fe)}^{(2)}$	-0.611	-0.389
$\varepsilon_{(Fe-Cu)}^{(1)}$	-0.480	-0.609	$\varepsilon_{(Fe-Cu)}^{(2)}$	-0.571	-0.344
$\varepsilon_{(Cu-Cu)}^{(1)}$	-0.414	-0.581	$\varepsilon_{(Cu-Cu)}^{(2)}$	-0.611	-0.389

变量生成 (RVG) 算法来确定在下一个时间增量中选取具体的跃迁事件进行模拟。这里的“动态”是指每次发生事件时，概率的分布都会发生变化。通常，随机模拟算法 (SSA) 通过对可能发生事件的动态概率分布进行正确地采样而被用于模拟体系的反应网络中。

如算法 9 所示，一个常见的随机模拟算法是使用 Gillespie 随机算法<sup>[173]</sup>，其对所有事件执行线性扫描后根据全局概率选择一个。对于  $N$  个事件<sup>[165]</sup>，它的时间复杂度为  $O(N)$ 。事件的列表也可以组织为二叉树，此时的时间复杂度为  $O(\log_2 N)$ 。然而，在应用到大规模的体系时，线性以及对数时间复杂度下的性能仍不能满足要求。因此，我们转而使用  $O(1)$  的扩展性更高的分组反应模型，通过设计新的数据结构，将事件依据概率值重新分组。然后利用分流拒绝 (composition and rejection, CR) 算法，使事件选择的时间复杂度降为常数级<sup>[165,174]</sup>。

图 5.5 展示了分组反应模型。首先，从事件概率的最小值  $p_{min}$  到事件概率的最大值  $p_{max}(8p_{min})$  所覆盖的总计  $N$  个事件被重新分到  $G$  个组（以 2 的幂次大小作为每组的分割界限）。在这个算例下，每个组的上限分别为  $2p_{min}$ 、 $4p_{min}$  和  $8p_{min}$ 。然后，在选择步骤中随机选择一个事件组。在所选的事件组内，我们生成一个从 1 到  $N_{local}$  的随机整数  $i$  和一个从 0 到  $p_{localmax}$  的随机值  $r$ ，其中  $N_{local}$  为该组的事件数量， $p_{localmax}$  为该组事件概率值的上界。如果  $p_i < r$ ，则拒绝该次反应事件  $i$ ，如图 5.5 中的黑色方块点所代表的事件。重复上述操作，直到出现  $p_i \geq r$ ，如图 5.5 中红色三角点所代表的事件，这意味着选择了概率值为  $p_4$  的反应事件。由于每个事件组中的概率分布面积大于该组的总面积的一半，采用随机模拟的分组反应模型可以在时间复杂度为  $O(1)$  的情况下快速完成。

### 5.3 并行算法设计：适应复杂体系结构的并行计算和通信算法

在本节中，针对 OpenKMC 应用的大规模分布式模拟，我们从三个不同的方面进行高效通信模式的设计。首先，计算 Ghost 晶格以减少消息总量并消除通信中的冗余数据信息；然后，采用非阻塞集合通信模式通过点对点操作减少同步时间；最后，我们提出了一种自适应通信算法，根据对模拟中的事件发生状态进行检测来自动降低通信频率。

### 5.3.1 并行 AKMC 计算

为了扩展 KMC 模拟的时间和体系的尺度，需要并行化 KMC 算法。本节我们实现了 SPPARKS<sup>[175]</sup> 中的优化算法，并采用了 Shim 和 Amar<sup>[166]</sup> 提出的同步象限算法来解决边界更新冲突的问题，减少全局通信开销。如果事件发生在距离当前格点较远的位置<sup>[165]</sup>，则该次事件对此格点的影响可以忽略。因此，可以利用在空间上对事件进行解耦的思想来实现并行化。

基于神威处理器的架构，我们从三个层面对该算法进行了重新设计。如图 5.6 所示，首先，二维模拟域中所有数据都被并行划分到所有可用的处理器上。对于单个神威 SW26010 处理器，其分成 4 个进程并被组织为  $2 \times 2$  的进程网格。需要注意的是，在传统算法的实现中，当两个或多个进程在共享边界附近同时执行事件时，它们可能会存在边界冲突，导致不正确的跃迁概率计算。

第二个层次则通过将模拟域划分为更细粒度的象限来解决这个问题，通常在二维模拟域中划分为 4 个象限、三维模拟域中为 8 个象限<sup>[165]</sup>。如图 5.6，所有进程在一段时间  $\Delta t_{syn}$  内独立地在自己的子域上按照象限号依次执行事件。在进程移动到下一个象限的计算时，必须提前更新边界区域中的格点。图 5.7 给出了我们采用的 Ghost 晶格计算的数据传输路线。周围的进程准备好针对当前象限所需数据的 Ghost 晶格，以发送到当前象限所在的进程。同样，本地进程的 Ghost

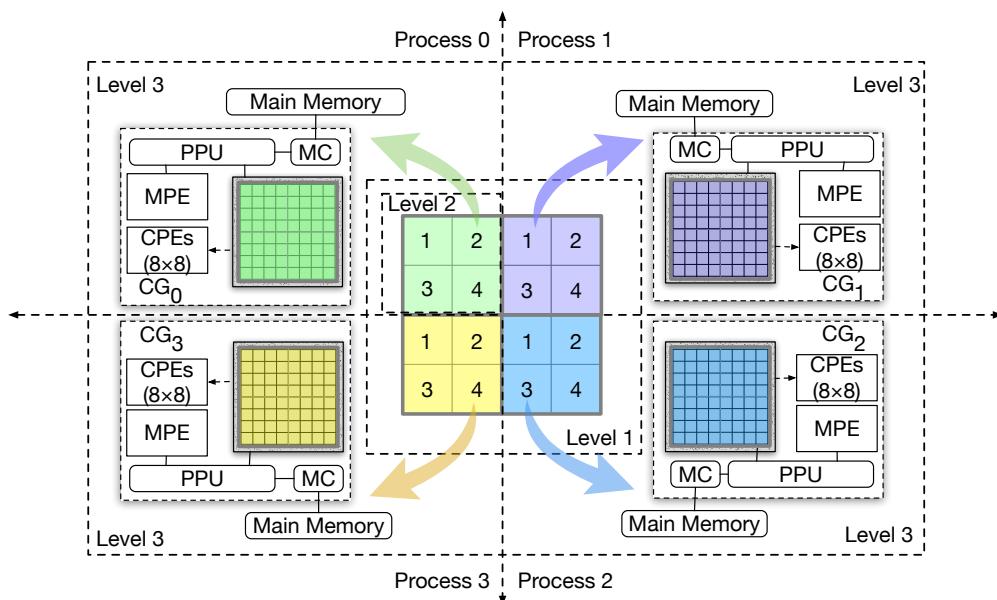


图 5.6 神威 SW26010 架构的并行 AKMC 算法（4 个进程将模拟域细分为 16 个象限，每个象限进一步分配到各个从核进行计算）

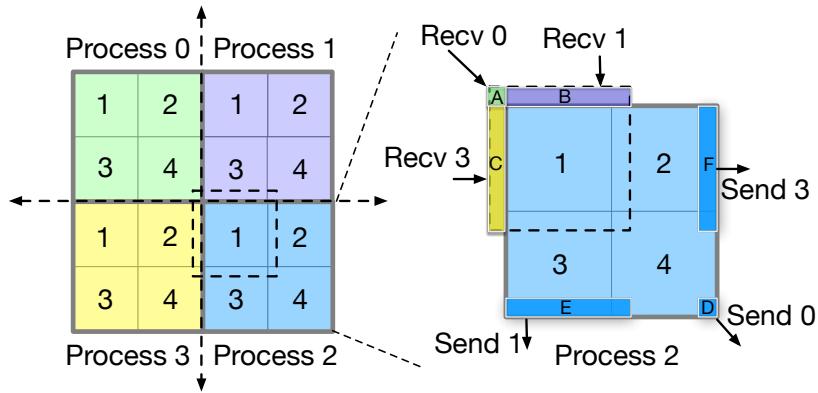


图 5.7 神威 SW26010 架构的 Ghost 晶格计算（左侧部分展示了蓝色进程的计算所依赖的周围环境（虚框所示），其中一些数据由其他进程持有。右侧部分展示了该进程将在 3 次通信后从其他进程（分别对应绿色、紫色、黄色进程）发送和接收的数据，以完成其第一象限的计算）

晶格也会相应发送到其他邻居进程。当最后一个象限的计算完成之后，系统的总执行时间相应增加  $\Delta t_{syn}$ 。由于每次的模拟域被限制在一个象限中，我们实现了更好的缓存分块，以获得更高的访存性能。

第三，基于神威 SW26010 的架构，我们可以通过将分布在各个进程上的计算任务进一步划分，以在核组上使用从核阵列来进一步加速象限的计算。在我们的设计中，一部分从核扮演着主核与其他从核之间的信使角色进行信息整理和传递，其余从核则执行分配的计算任务。这里涉及了一个我们针对神威从核架构所提出的转录-翻译-传输算法，相关工作在第 5.4.3 节中进行详细展开。

### 5.3.2 Ghost 晶格计算

在计算分配到每个进程上的象限时，必须提前更新其边界区域中的 Ghost 晶格。物理学上，传统的方法是将周围象限中的所有数据直接传输到本地进程。然而，由于物理学中截断半径的存在，大量的数据传输是冗余的，因为它们对于本地象限中格点的更新影响可以忽略。因此，Ghost 晶格计算的关键是消除这些 Ghost 区域外的冗余数据传输。我们通过分析截断半径的影响范围，将本地数据划分为更细粒度的数据块进行 Ghost 晶格的传输，以消除冗余的非作用晶格的数据通信。

### 5.3.3 非阻塞集合通信

当达到同步时间  $\Delta t_{syn}$  时，进程之间需要进行通信以完成对 Ghost 晶格粒子的更新。如图 5.8 中的左子图所示，传统的通信模式是一对一交换通信方法

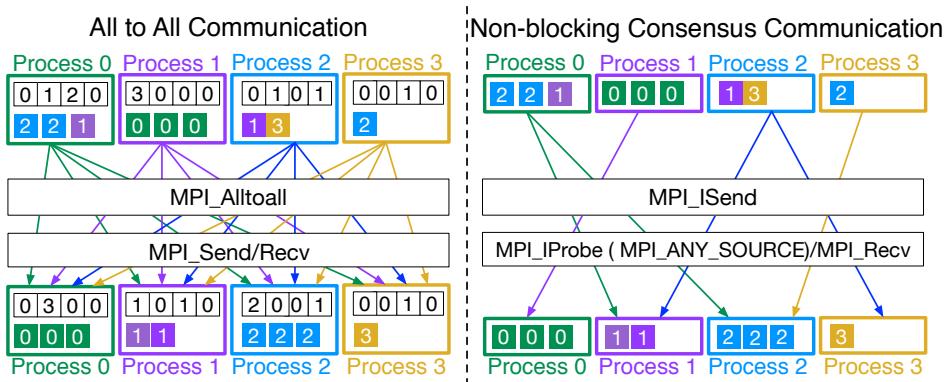


图 5.8 实现的 MPI 通信模式与传统通信模式的比较(左子图的是一对一交换通信算法(PEX), 右子图是非阻塞集合通信(NBX))

(Personalized Exchange Algorithm, PEX)<sup>[176,177]</sup>。针对一个  $P$  进程的分布式系统, 每个进程将要发送给其他各进程的数据量写入大小为  $P$  的向量中, 然后通过调用 `MPI_Alltoall` 函数对各个消息接收源和消息大小进行明确。最后, 调用 `MPI_Send/Recv` 函数建立通信通道分别进行消息传输。

虽然传统的一对一交换通信方法直观, 易于实现, 但是该算法涉及到的通信操作比较昂贵, 而且扩展性较差。一旦进行大规模模拟, 性能瓶颈急剧凸显。对于  $P$  个进程的分布式系统, 完成一对一交换通信方法的平均时间复杂度为  $\Theta(P)$ <sup>[178,179]</sup>。因此, 对于大规模模拟, 我们采用了扩展性较好的非阻塞集合通信 (Nonblocking Consensus Communication) 方法<sup>[180]</sup>。图 5.8 中的右子图描述了该通信方法。该方法利用非阻塞的集合操作, 它们可以相互独立地发起和完成通信而无需调用全局 `MPI_Alltoall` 通信函数。首先, 每个进程独立地调用 `MPI_ISend` 进行消息发送, 然后使用 `MPI_IProbe` 探针进行查询, 获取通信所需的必要信息, 最后调用 `MPI_Recv` 接收来自 Ghost 进程中的数据。由于使用非阻塞的操作, 节省了大量的传输和同步时间。对于  $P$  个进程, 完成非阻塞集合通信方法的平均时间复杂度为  $\Theta(\log(P))$ 。

### 5.3.4 自适应通信算法

通过自适应通信算法, 在同步子域算法的基础上, 论文实现了更精准地通信操作。发送的粒子数据得到了进一步的精简, 且不必要的通信也被删减, 通信的消息大小和数量均得到了进一步的优化。

如前所述, 每个进程上象限的同步时间  $\Delta t_{syn}$  是影响通信的关键因素。 $\Delta t_{syn}$  的值越小, 意味着 Ghost 晶格更新地越频繁, 得到的仿真结果越准确。然而, 频

繁的通信带来了较差的并行效率，因为并非所有的通信步都是必须的。如果仅是简单地通过增加  $\Delta t_{syn}$  增加同步的时间间隔，虽然通信的次数会减少、并行效率会提高，然而模拟结果的误差也会越大。由于在模拟过程中跃迁事件发生的概率可能会随机发生变化，所以很难找到一个最佳固定的  $\Delta t_{syn}$  值同时来满足长时间 KMC 模拟准确性和高效率的要求。因此， $\Delta t_{syn}$  的设定需要进行细致地权衡调节。

首先，我们考虑一个由大量相同事件组成的理想化物理系统，每个事件在单位时间内发生的概率为  $p$ ，事件的总数为  $N$ ，则对于  $n$  个事件的时间可由式 5.9 计算：

$$t = \frac{n}{N \times p} = \frac{n/N}{p}。 \quad (5.9)$$

在该计算域内的活跃事件比例为  $E$ ，其等于  $(n/N) \times 100\%$ ，这给出了对真实模拟的事件数量的一个估计。我们将  $p_s$  定义为在当前计算域内事件的总概率，通过与活跃事件的数量的比值来探究与步进时间的关系，则计算域内的初始同步时间  $\Delta t_{syn}$  由下式获得：

$$\Delta t_{syn} = \min(E/p_{max}, threshold - T), \quad (5.10)$$

其中  $p_{max}$  是所有进程中  $p_s$  的最大值，threshold 是预设的模拟总阈值时间， $T$  是当前模拟的时间。

在 SPPARKS<sup>[165]</sup> 中，跃迁事件的概率被认为是动态变化的，因此依赖于它的  $\Delta t_{syn}$  足以调整模拟的进度。然而在 AKMC 模拟中，各事件的跃迁概率基本相差不大，需要更精确的自适应同步通信算法。如算法 10 所示，在我们的设计中，通信时首先对于每次粒子的跃迁及其影响的粒子进行判定。只有当事件发生在截断范围内，真实地执行了一次跃迁并影响到了邻居进程，受影响的粒子才会被选择并添加到发送队列。否则，即使达到了同步时间  $\Delta t_{syn}$ ，AKMC 模拟也将跳过通信继续进行计算。通过将物理规律和通信技术叠加进行优化，自适应算法可以在准确性和效率之间取得更好的平衡。

---

**算法 10** 自适应通信算法
 

---

**Require:**

$E$ : The fraction of the events in the sector.  
 $threshold$ : A preset total simulation time.  
 $T$ : The current simulation time.  
 $p_{threshold}$ : The propensity threshold for jumping events.  
 $t_{stop}$ : A default synchronization interval.

```

1:  $p_{local\ max} \leftarrow 0$ 
2: for  $i = 0$  to  $sectors$  do
3:    $n_{active} \leftarrow$  Number of active sites in current sector.
4:    $p_s \leftarrow$  Total propensity of events in current sector.
5:    $p_s \leftarrow p_s/n_{active}$ 
6:    $p_{local\ max} = \max(p_s, p_{local\ max})$ 
7: end for
8:  $p_{max} \leftarrow$  maximum  $p_{local\ max}$  across all processors.
9: if  $p_{max} > 0$  then
10:   $\Delta t_{syn} \leftarrow E/p_{max}$ 
11: else
12:   $\Delta t_{syn} \leftarrow threshold - T$ 
13: end if
14:  $\Delta t_{syn} = \min(\Delta t_{syn}, threshold - T)$ 
15: while  $T \leq threshold$  do
16:   for  $j = 0$  to  $sectors$  do
17:     while  $t < \Delta t_{syn} \&& jumps_b \leq 0$  do
18:        $p_{local\ max} \leftarrow$  The total propensity of events.
19:       Execute jumping events.
20:        $jumps_b \leftarrow$  The number of boundary events.
21:       Calculate  $dt$ 
22:        $t += dt$ 
23:     end while
24:     Communication.
25:   end for
26:    $T += \Delta t_{syn}$ .
27:    $p_{max} \leftarrow$  maximum  $p_{local\ max}$  across all processors.
28:   if  $p_{max} > 0$  then
29:      $\Delta t_{syn} \leftarrow t_{stop}/p_{max}$ 
30:   else
31:      $\Delta t_{syn} \leftarrow threshold - T$ 
32:   end if
33:    $\Delta t_{syn} = \min(\Delta t_{syn}, threshold - T)$ 
34: end while

```

---

## 5.4 性能优化方法：神威硬件特征与算法精准抽象的深度性能优化

### 5.4.1 访存优化设计

由于模拟域需要在分布式的集群上进行划分，因此每个进程都拥有其子域内的计算晶格和周围的 Ghost 晶格依赖。每个进程上的 Ghost 晶格在参与边界格点的计算时不会执行事件。然而，与计算晶格的连续存储不同，Ghost 晶格是随机进行更新的。这些对不同数组的离散和随机的访存导致更少的缓存命中率。

在计算机科学中，平均内存访问时间 (*AMAT*) 是通过使用命中时间、未命中率和未命中惩罚项三个因素来分析内存系统性能的常用指标。命中时间是在缓存中命中的时间，未命中率 (*MR*) 是缓存未命中的频率，未命中惩罚项是缓存未命中的平均时间成本 (*AMP*)。具体可以用公式 5.11<sup>[43]</sup> 来定义。

$$AMAT = Hit\ time + Miss\ rate \cdot Miss\ penalty \quad (5.11)$$

为了减少 *AMAT* 以提高性能，我们可以针对这些指标逐一研究如何降低这些影响。在神威 SW26010 处理器上，当 cache line 大小为 256 字节时，即使访问的地址是随机的，也会达到峰值带宽<sup>[44]</sup>。因此 256 字节是实现随机内存访问峰值带宽的最小宽度，更大的宽度会增加 *MR* 和 *AMP*。

系统的 cache 大小是有限的，充分压缩结构体大小，使得 cache 能缓存更多的被访问数据，无疑是提高内存平均访问速度的有效方法之一，而这就需要我们对应用逻辑做好更合理的设计。默认情况下，内存地址会按照数据大小对齐，这样有些原子的数据信息可能会跨 cache line 存放，从而影响性能。图 5.9 展示了原子

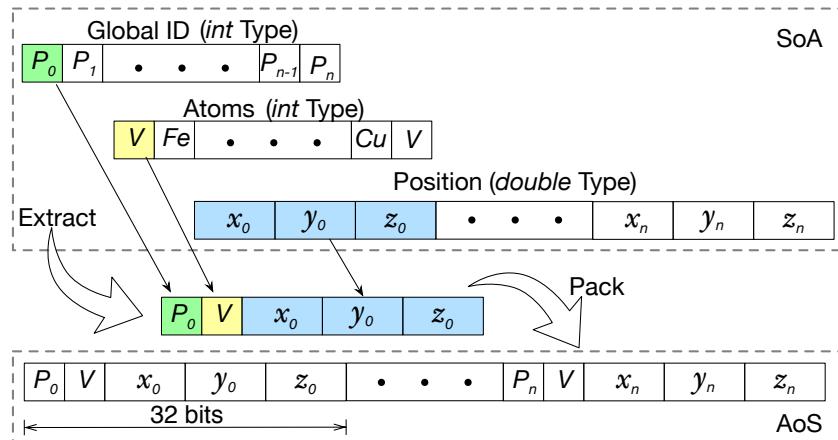


图 5.9 DMA 中数据结构重构以使访问连续并对齐

数据从数组结构（SoA）形式到结构体数组（AoS）形式的重组。我们提取了一个格点的基本属性，包括 ID、类型和位置，并将它们打包成一个新的 32 字节的原子数据结构，以便在通信之前使用。当我们从内存中获取 cache line 时，内存访问总是对齐的，因为 cache line 是由 8 个原子数据结构（总共 256 个字节）组成的整条数据。同时，我们也可以充分利用缓存的空间局部性，因为 Ghost 晶格一般是连续访问的。

### 5.4.2 OpenACC 自动并行

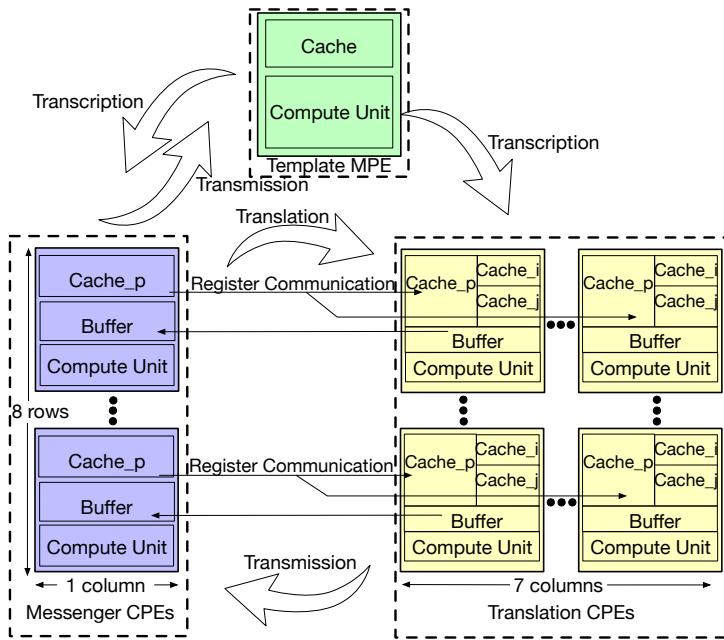
采用 OpenACC 制导语句对程序中的 for 循环部分并行，是神威·太湖之光超算系统第一种众核并行的方法。OpenACC 并行也是神威·太湖之光超算系统最主要的并行方法。大规模 KMC 模拟的 OpenACC 制导加速，分为以下几个步骤：

- 首先使用神威编译器分析 KMC 串行程序，记录 KMC 模拟串行版本的运行时间，找到串行版本的热点函数。
- 找到空位跃迁代码段的核心循环，将其单独重写成为 C 核心段文件。
- 在改写的 C 文件中的核心循环前添加 #pragma acc parallel loop 和其他制导语句指示。
- 使用“swacc -priv”选项编译 KMC 程序，根据私有变量自动分析结果，添加 private 子句。私有化的变量即 SW26010 处理器每个从核线程均可能会去改写，并且改写的内存区域有重叠的变量。
- 使用神威 sw5gcc 编译改写的 C 文件，使用 mpicxx 编译器进行链接，生成 KMC 可执行程序后，添加“-cgsp 64”选项以使用 1 个主核搭配 64 个从核的形式提交到神威高速队列运行。

### 5.4.3 从核转录-翻译-传输算法

针对神威 SW26010 处理器主从加速编程模型的特点，神威·太湖之光超算系统内置了加速线程库 (Athread)，以使用户可以精准、快捷地控制和调度核组内的线程，从而能够更加充分地发挥主从核架构的加速性能。

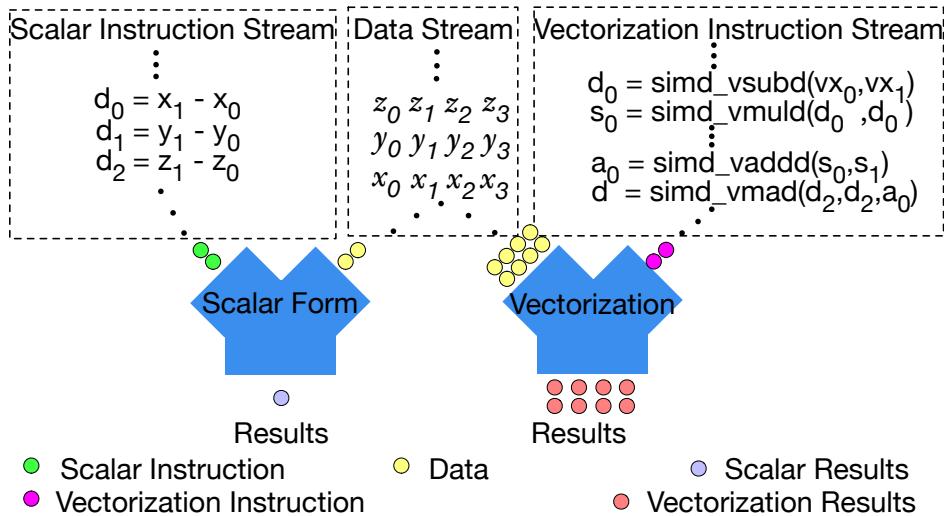
在基于神威架构的移植过程中，鉴于 OpenACC 编译器自动并行效果有限，尚无法解决性能瓶颈，我们提出了从核转录-翻译-传输算法 (Transcription-Translation-Transmission, 3T)，在完成 OpenACC 加速的基础上继续使用 Athread 加速线程



**图 5.10** 用于从核加速计算的转录-翻译-传输算法（绿色、紫色和黄色的核心分别是模板核、信使核和计算核。每个信使核将与同一行中的 7 个计算核配对，它们将使用快速行寄存器通信共享数据）

库进行了更细粒度的加速优化。设计的 3T 算法描述了 KMC 大规模模拟中实现的线程级并行方法。图 5.10 展示了我们算法的整个过程，其中绿色的主核被称作是模板核（Template MPE），紫色的 8 个核心被称作是信使核（Messenger CPE），另外 56 个黄色的核心被称作是计算核（Translation CPE）。算法首先将数据从主核复制到从核（Transcription），然后数据经从核计算后（Translation），通过快速寄存器通信，将数据传输到指定的从核进行汇总整理，最终的结果再次发送到主核（Transmission）。

以粒子  $i$  与  $j$  之间欧式距离的计算为例：首先，原子  $i$  和  $j$  的原始数据，包括类型，原子 ID 和位置，从主核分别打包发送至相应的从核。同时，信使从核获得来自主核的包括截断半径、边界等一系列参数，并对参数进行打包通过快速寄存器通信发送至相应的计算从核。当所有数据都发送至负责计算的从核时，在这些从核上立即进行一列相关的计算。计算产生的结果存储在局部的缓冲区中，以进行进一步的判断。如果它们是相邻的，结果将通过快速寄存器通信被发送到相应的信使从核，经信使从核统一整理、打包传输回主核。因此，通过将计算和写回的时间进行重叠，进一步减少了主从核间的通信开销。

图 5.11 内层循环  $j$ -粒子计算的向量化

#### 5.4.4 向量化加速

为了利用神威 SW26010 处理器提供的 256 位 SIMD 矢量寄存器，我们也在大规模 KMC 模拟中实现了向量化加速。在从核上向量化实现的过程中也存在一些阻碍和挑战。例如，当从核从主核获取数据时，内存访问是不规则的，造成向量化指令的访存开销较大。对于这种情况，我们提出了主核中数据结构的访存优化策略。原子的数据被重新组织成 AoS 形式，每个原子占用 32 字节。由于更好的数据局部性，数据结构中的相邻数据也以高概率被访问。然后，在将数据传输到从核进行加速之前，仅需要对部分需要从核计算的数据进行重新打包操作。以 Athread 线程库优化加速的 3T 算法中欧式距离计算为例。对于每个中心粒子  $i$ ，成对粒子的距离计算需要遍历大量的候选粒子  $j$ ，因为大量粒子所在的晶格在粒子  $i$  的截断半径范围内。因此，对于同一个中心粒子  $i$ ，我们使用第 5.4.1 节中的访存优化策略将相关的原子数据以 3T 算法 DMA 到从核后，再使用神威从核上的 SIMD 扩展来高效地同时执行 8 次  $j$  粒子的计算。图 5.11 展示了欧式距离计算中标量和向量化后的运算比较。通过采用 SIMD 操作，我们利用了数据级并行性，实现了  $j$  粒子间的并行计算，从而在神威的从核架构上实现了更高效的计算。

## 5.5 实验评估

在本节中，我们将对我们研发的核材料模拟软件 OpenKMC 从正确性、单节点性能和多节点扩展性能等方面进行详细的实验评估。同时，我们也完成了经典的铜富集沉淀析出的可视化，作为实验的进一步补充。

我们将设计的 SWMD 硅晶体分子动力学模拟软件在神威·太湖之光超级计算机上进行了性能评估。考虑到硅的热和应力变化，我们使用 Tersoff III 势函数<sup>[181]</sup> 进行能量的计算。代码由神威架构定制的 SWCC Compilers (Version 5.421-sw-500) 编译器进行编译。为便于描述，通信缩写为“Comm.”；传统邻居表算法缩写为“NBLList”；“MC acceleration” 代指使用 OpenACC 进行众核加速；“PC” 算法则指生产者-消费者配对算法。

### 5.5.1 数据验证

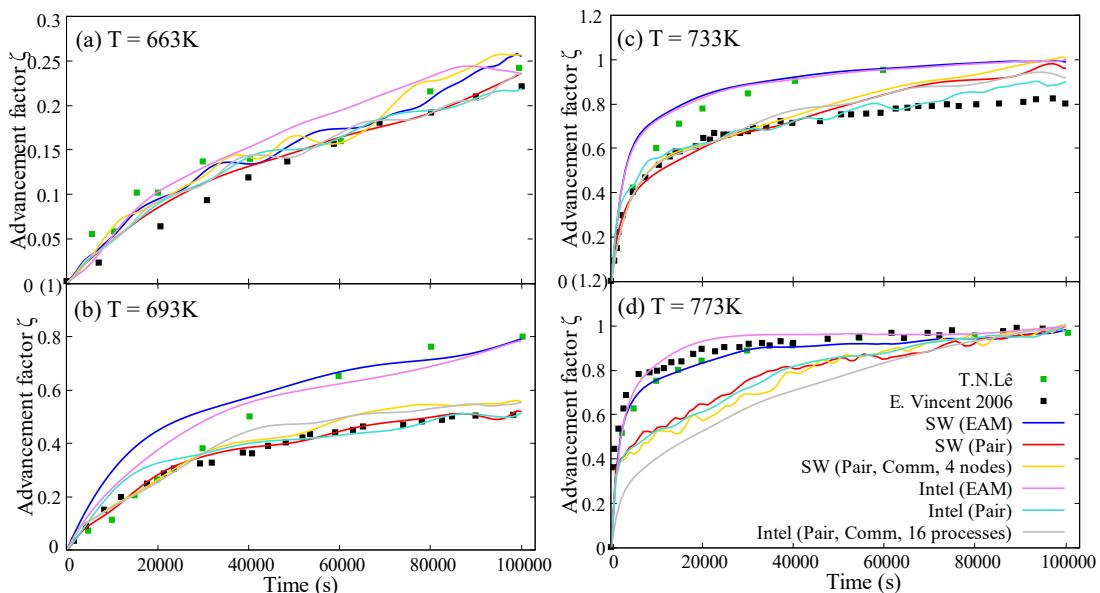


图 5.12 Fe-1.34 at.% Cu 合金的热老化析出演变（绿色方块对应 Lé 等人的真实物理实验结果<sup>[8]</sup>；黑色方块对应 Vincent 等人的动力学蒙特卡罗模拟结果<sup>[9]</sup>；其他曲线则为在不同架构上的 EAM 势或 Pair 势的 OpenKMC 模拟结果。括号中的“Comm”表示已进行通信优化的 OpenKMC 结果）

为了验证 OpenKMC 模拟的准确性，我们对二元 Fe-1.34.%Cu 合金在 663K 至 773K 的温度下进行了一系列热老化模拟，并与 Lé 等人真实的物理实验<sup>[8]</sup> 以及 Vincent 等人的经典模拟结果<sup>[9]</sup> 相对比。我们以  $10^5$  秒（放缩时间）跟踪模拟的演变过程，并在此模拟空间内引入  $8 \times 10^{-4}$  % 的空位浓度。模拟的时间基于 Vincent 等人的工作<sup>[182]</sup> 重新进行放缩调整，以获得相应的物理时间尺度。对于

每个模拟，我们使用 40 个晶胞的 BCC 晶格，且空间维度中的每一维均应用周期性边界条件。沉淀促进因子以式 5.12<sup>[8,9]</sup> 进行计算，其中  $C_X(t)$  为  $t$  时刻固溶剂中的 Cu 浓度，其在模拟中会趋向于溶解度极限  $C_X(\infty)$ 。OpenKMC 对沉淀析出动力学的模拟分别是在神威·太湖之光的单个节点（4 个主核和 256 个从核）和 Intel Xeon E5-2670 的单进程上使用 EAM 势和 Pair 势进行的，图 5.12 中展示了模拟的结果分别与真实的物理实验以及经典模拟结果的对比。

$$\zeta(t) = (C_X(0) - C_X(t))/(C_X(0) - C_X(\infty)) \quad (5.12)$$

很明显，在热老化模拟的过程中，Cu 逐渐析出。整体上，我们的 OpenKMC 对沉淀析出动力学的模拟在两种架构都很好地再现了 Vincent 的工作<sup>[182]</sup>。至于与真实物理实验结果的对比<sup>[8]</sup>，OpenKMC 的结果曲线可以定性地描述模拟系统随时间的演化，尽管仍然存在一定的定量偏差。这种误差主要来自于简化模型（例如，仅考虑一种缺陷，即点缺陷）以及原子间缺乏多体相互作用和长程力相互作用。

在执行通信优化时我们也评估了模拟的准确性。我们在 Intel Xeon E5-2670 和 SW26010 上的分布式实验均采用了 16 个进程。如图 5.12 所示，在进行通信优化后，OpenKMC 仍可以获得足够准确的模拟，且此时两种架构的总运行时间节省至一半。因此，通过我们高效通信模式的优化后，OpenKMC 可以同时保证高模拟精度与高运行效率。

### 5.5.2 辐照损伤可视化

为了进一步证明 OpenKMC 对于核材料辐照损伤长期演化的有效性，我们在 663K 的温度下对 Fe-1.34Cu(%) 的辐照热老化进行了放缩物理时间 100 年的模拟。在该算例中，体系三个维度的每个维度上均排列了 40 个刚性 BCC 单位晶格，这与第 5.5.3 节中的实验和 Vincent 的相关工作<sup>[9]</sup> 配置相同。模拟前后体系的原始状态和最终状态可视化成图展示在图 5.13 中。

在模拟开始时，很明显，所有 Cu 原子随机分布在  $\alpha$ -Fe 铁氧体中，该体系并没有明显成核的 Cu 析出物。然而，经过 100 年辐照热老化后，体系的微观结构与模拟开始时相比发生了明显的变化。此时，大量成核的 Cu 析出物富集且最大富铜核由 542 个原子组成。这种由于辐照热老化所引起的不同尺寸 Cu 沉淀的形

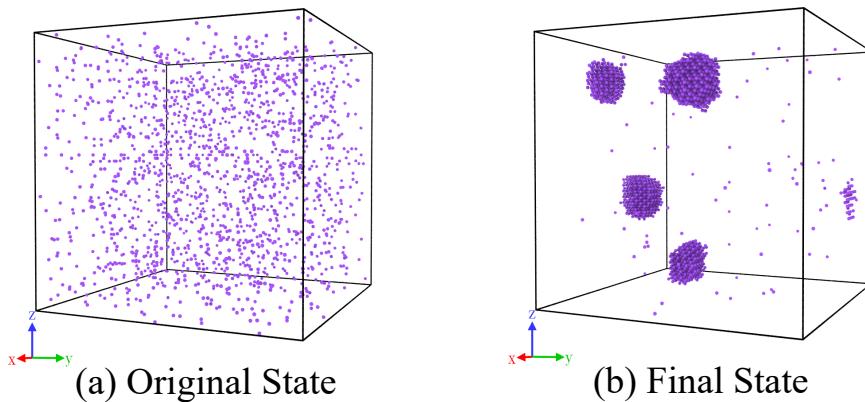


图 5.13 FeCu 合金热老化前后 Cu 簇析出可视化

成与热力学理论一致<sup>[9,148]</sup>。因此，长期演化得到的可视化结果是 OpenKMC 模拟准确、有效的证明。

### 5.5.3 叠加优化性能评估

图 5.14展示了在使用 260 个核心（4 个 MPE 和 256 个 CPE）的 SW26010 节点上，不同类型的优化方法分别在高浓度和低浓度下的性能提升。高浓度实验采用了较高的 12.0.%Cu 原子浓度和 12.8% 的空位浓度，其他配置与 Vincent 中的研究案例相同<sup>[9]</sup>。

如图 5.14(a) 和图 5.14(c) 所示, 初始时, 能量计算内核的性能主要由 EAM 势所影响, 在从物理角度利用优化的 Pair 势模型之后, 模拟有明显的性能加速。图 5.14(b) 和图 5.14(d) 则是从计算技术的角度在使用 Pair 势模型计算之后使用提出的多种方法的叠加优化的比较。首先, 我们通过分组反应模型减少了事件的执行时间; 然后访存优化设计减少了未对齐的内存访问。对于通信模式, 我们的自适应通信算法则进一步将通信时间减少了一半。接下来, 我们通过 3T 算法和向量化来加速从核上的计算过程。在线程并行中, 空位事件的模拟被加载到从核进行计算加速。当采用高浓度时, 我们保证每个从核上均有足够的计算任务, 并且图 5.14(b) 中在此时获得了明显的加速。但是, 如果引入较低的  $8 \times 10^{-4}\%$  空位浓度, 则模拟体系中仅存在一个空位。因此, 如图 5.14(d) 所示, 从核的计算资源潜力没有被充分挖掘, 从核的使用反而会带来了额外的主从核通信开销。在较高的空位浓度下, 最终的 OpenKMC 设计相对于 5.14(b) 中的 Pair 势模型版本提速 7.76 倍, 相对于与图 5.14(a) 中的原始 EAM 势模型版本相比, 整体加速则

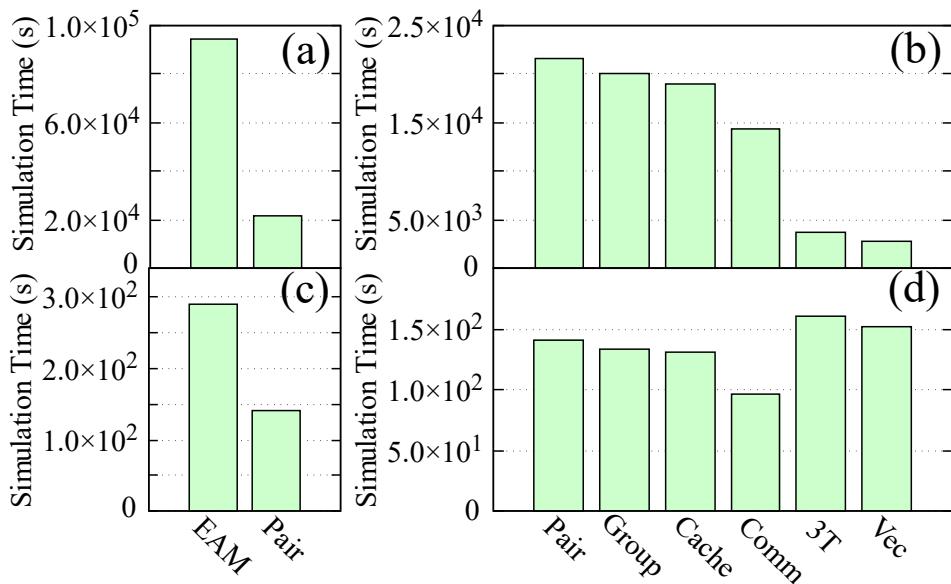


图 5.14 神威 SW26010 处理器上 OpenKMC 多种方法的叠加优化性能图（子图 (a) 和 (b) 为高浓度实验，即空位浓度为 12.8%，Cu 浓度为 12.0%；子图 (c) 和 (d) 为低浓度实验，即空位浓度为  $8 \times 10^{-4}$ %，Cu 浓度为 1.34%。子图 (a) 和 (c) 展示了物理方法优化的性能提升；子图 (b) 和 (d) 展示了计算技术优化的性能提升）

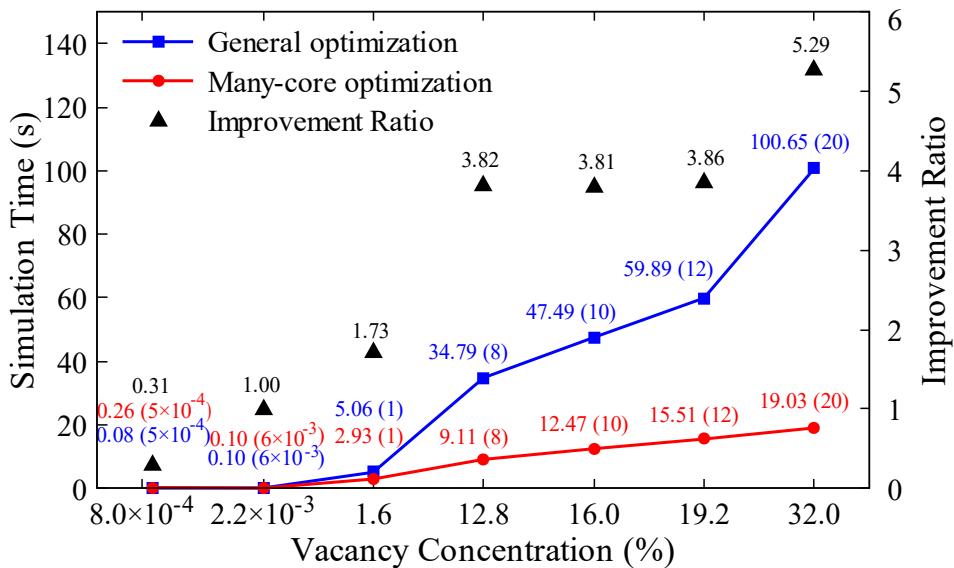


图 5.15 不同空位浓度下的性能变化（红线和蓝线分别代表 OpenKMC 模拟中使用和不使用众核优化（3T 算法和矢量化）的运行时间。括号中的数字表示分配给每个从核的空位数量。性能提升比使用黑色上三角进行标记）

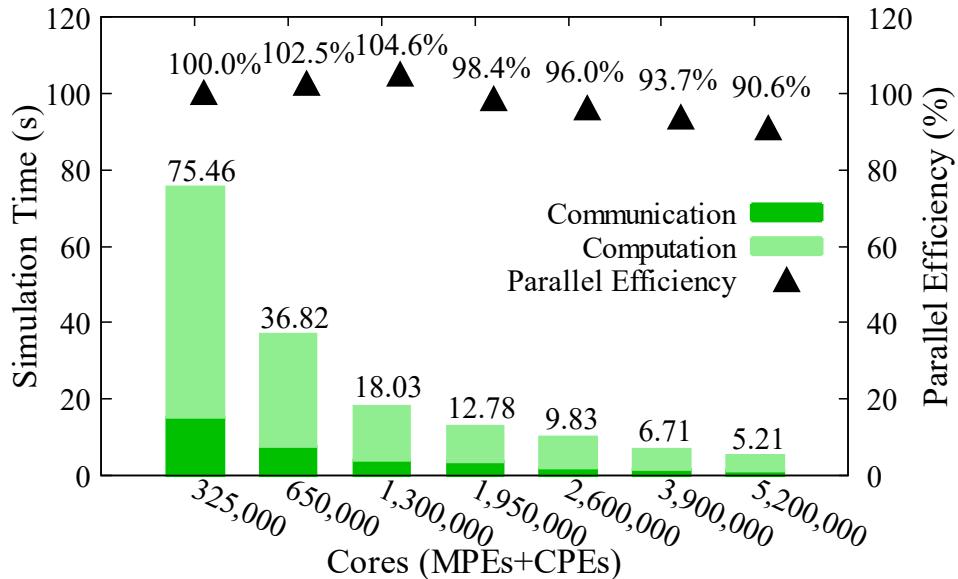


图 5.16 OpenKMC 模拟 540 亿 ( $5.4 \times 10^{10}$ ) 原子的强可扩展性 (x 和 y 轴相应进行缩放；绿色直方条对应左侧主 y 轴的模拟时间；黑色三角形散点对应右侧次 y 轴的并行效率；模拟时间和并行效率值分别标注在直方条和三角形散点的顶部)

提高了 33.88 倍以上。

为了研究空位浓度对性能的影响，我们在图 5.15 中详细地进行了不同空位浓度下的实验对比，其中红线和蓝线分别代表 OpenKMC 模拟中使用和不使用众核优化（3T 算法和矢量化）的运行时间。由于空位浓度设置为  $8 \times 10^{-4}\%$ （Cu 浓度为 1.34%，温度为 573K），当应用众核优化时，模拟的运行时间反而增加，而不是减少。这主要是因为当空位计算量过少时，分配给每个从核的计算任务量很小，多核启动和数据通信的时间占据了模拟时间很大的比例，从而使得性能显著下降（从 12.8% 浓度时的 3.82 倍加速到  $8 \times 10^{-4}\%$  浓度时的仅 0.31 倍加速）。然而，当空位浓度增加到 32.0% 时，性能加速比（5.29 倍）甚至比 12.8 at.% 浓度时更高。

值得注意的是，图 5.15 中的性能加速比取决于每个从核上分配到的空位数量，而不是空位的浓度。这是因为并行化的对象针对的是模拟体系中的空位跃迁计算。因此，当模拟系统中的原子总数足够大时，即使是在非常低的空位浓度下，众核优化仍然能够取得较好的性能。

#### 5.5.4 大规模可扩展性

在本小节中，我们将评估我们的 OpenKMC 在神威·太湖之光上的可扩展性。为了实现强可扩展性测试中体系的大小和 SW26010 处理器上的内存（每个

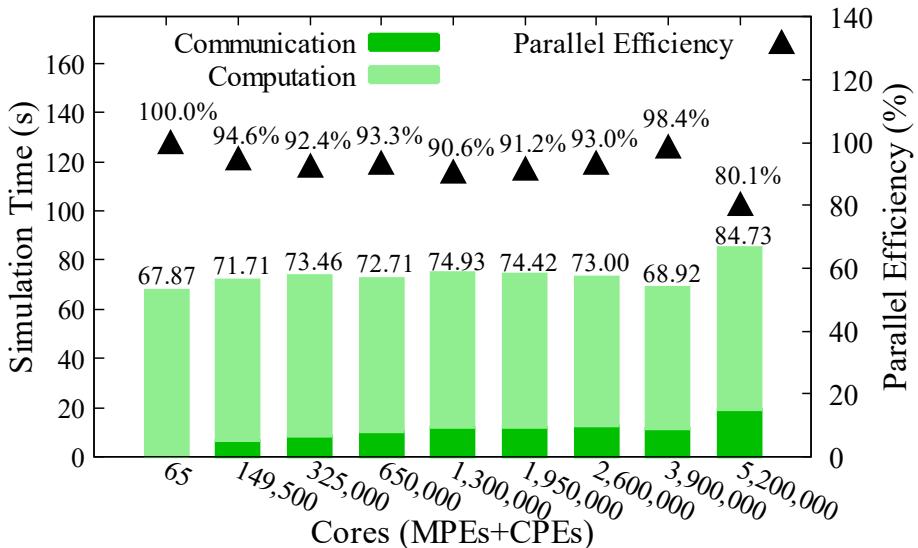


图 5.17 OpenKMC 模拟每个进程 1100 万个原子 ( $1.1 \times 10^7$ ) 的弱可扩展性 (x 和 y 轴相应进行缩放; 绿色直方条对应左侧主 y 轴的模拟时间; 黑色三角形散点对应右侧次 y 轴的并行效率; 模拟时间和并行效率值分别标注在直方条和三角形散点的顶部)

核组总计 8GB) 之间的平衡, 我们将 5,000 个核组 (5,000 个主核搭配 320,000 个从核) 的性能作为基准, 每个进程模拟大约 1,100 万个原子 ( $1.1 \times 10^7$ )。在此配置中, 模拟的每个原子所需的内存大约为 0.72KB。相应地, 弱可扩展性也从一个以 1,100 万个原子为基准的单核组 (1 个主核搭配 64 个从核) 进行初始模拟。

对于强可扩展性, 我们研究的算例配置为 540 亿 ( $5.4 \times 10^{10}$ ) 个原子、热老化温度为 300°C (573K)。由于 Cu 簇是由空位发生跃迁事件形成的, 我们在此算例中引入了浓度为  $1.5 \times 10^{-4}.$ % 的 Cu 原子和浓度为  $8 \times 10^{-5}.$ % 的空位。Cu 原子总数和空位总数分别为 43,200 和 81,000, 这保证了在使用 520 万个核心时平均每个进程至少含有一个杂质元素。模拟时间也重新进行了调整, 以获得相应的物理时间尺度。在当前算例下,  $2.26 \times 10^{-2}$  的模拟时间对应放缩的物理时间尺度为 3 个月。图 5.16 展示了当核心数量从 325,000 增加到 5,200,000 时, OpenKMC 模拟的性能仍具有很强的可扩展性。我们可以看到, 计算时间在整个模拟中占据了较大的比例, 计算和通信时间都随着进程数量的增加而减少。如图 5.16 所示, 即使进程数量增加到 520 万, 并行效率仍然在 90% 左右, 这表明 OpenKMC 应用具有良好的强可扩展性。在这种大的模拟算例下, 实际加速几乎与理想加速相当, 尤其是当进程数小于 130 万时, 分组反应模型和高效通信模式带来的双重优化效果显著, 导致了出现超线性的加速。

图 5.17 展示了 OpenKMC 以一个核组为基准 (1 个主核搭配 64 个从核) 的

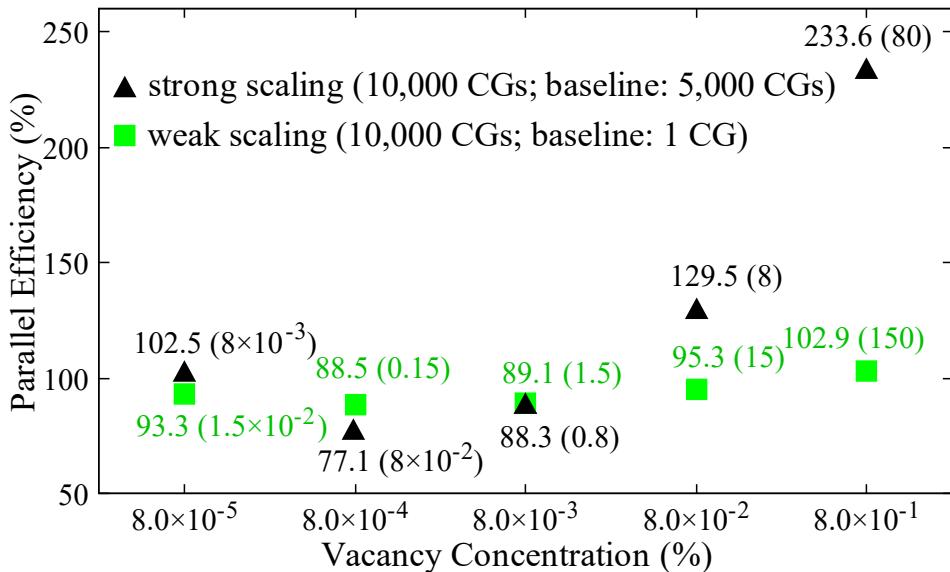


图 5.18 不同空位浓度下的并行效率对比（括号中的数字表示平均分配给每个从核的空位数量）

弱可扩展性模拟。由于每个核组指定一个进程，因此核组内主核和从核之间的快速 DMA 操作不被视为通信时间。我们在每个进程使用平均约 1,100 万个原子 ( $1.1 \times 10^7$ ) 来初始化模拟体系。可以看出，随着进程数量的增长，OpenKMC 仍保持了较高的并行效率。最终，OpenKMC 模拟达到了 520 万个进程的规模，大约模拟了 8,400 亿个原子 ( $8.4 \times 10^{11}$ )，并达到了 80.1% 的并行效率。值得注意的是，当进程数量少于 390 万时，几乎所有模拟的并行效率均接近于 100%，这证明了 OpenKMC 具有模拟大规模物理系统同时保证大规模并行性的能力。如图 5.17 所示，模拟的计算时间在不同进程数量上几乎保持不变。但是，在 520 万进程上模拟的通信时间稍高，这主要是由于大规模下的通信争用造成的。

另外，我们也在较大的的模拟体系（使用 30,000 个核组）上对另一算例分别进行了强可扩展性和弱可扩展性的研究。该算例参照 Vincent 的相关工作<sup>[9]</sup>，采用了相同的空位浓度 ( $8 \times 10^{-4}\%$ )、Cu 浓度 (1.34%) 和温度 (573 K)。实验结果也表明，OpenKMC 能够在大规模模拟的情况下保持较高的计算效率与准确率。

此外，在固定 Cu 浓度 (1.34%) 和温度 (573K) 的情况下，我们也进行了不同空位浓度下的扩展性研究，结果如图 5.18 所示。可以看出，在较宽的浓度范围内 ( $8 \times 10^{-5}\%$  到  $8 \times 10^{-1}\%$ )，并行效率也始终保持着较高的结果。特别是在  $8 \times 10^{-1}\%$  的空位浓度下，由于大量空位（每个从核 80 个）被分配到从核加速计算，强可扩展性的效率达 233.6%。

## 5.6 本章总结

在核材料的辐照损伤模拟工作中，我们设计了一种新的并行 AKMC 应用软件，OpenKMC，来支持国产神威·太湖之光超级计算机上的千亿原子体系的模拟。OpenKMC 支持传统的 EAM 势模型和我们优化的 Pair 势模型。为了在神威·太湖之光上进行更高效的实现，我们实现分组反应模型来加速事件的选择和更新过程，并重新设计数据结构以降低内存访问成本。此外，针对神威处理器的架构，我们提出了一系列高效通信模式和从核转录-翻译-传输算法，进一步提高了模拟的性能。同时，我们也使用了向量化技术在从核上进行计算加速，带来进一步的性能提升。实验表明，OpenKMC 能够提供高精度的 AKMC 模拟。在神威·太湖之光上使用 520 万个核心时，540 亿个原子的强可扩展性和 8,400 亿个原子的弱可扩展性模拟分别获得了 80.1% 和 90.6% 的高并行效率。

值得注意的是，我们工作中的许多优化策略，例如分组反应模型和从核转录-翻译-传输算法，尽管它们是专门针对神威 SW26010 处理器的优化，但都是大规模材料模拟中较为通用的优化算法。我们相信我们提出的方法可以为其他架构的算法设计带来启发性的经验，例如异构集群和下一代神威超级计算机。

## 第6章 总结与展望

### 6.1 工作总结

高性能计算是计算科学的重要分支和具体实践。数十年来，高性能计算一直是学术研究和创新应用的关键技术，被广泛地应用在工业仿真、科学计算、人工智能等重要领域，对国防工业建设、基础科学发现和国民经济发展具有极高的应用价值。图灵奖得主 Jim Gray 指出，可扩展性问题是高性能计算中的核心问题。从近几十年高性能计算机体系结构的发展来看，我们发现不同并行粒度的应用开发中在物理模型、并行算法以及底层硬件等多个层次上存在可扩展性的两种有趣的现象，即不连续性和非线性。

可扩展性的不连续现象，指的是随着并行计算问题在不同属性更换或在不同硬件架构迁移，性能出现了不同性质的不可连续扩展，即必须更换新物理模型，新并行算法设计或新并行软件实现方法以保证性能的现象。可扩展性的非线性现象，则意味着即使在某种物理模型、并行算法设计和并行软件实现等最适组合的解决方案范围内，性能也并不能随着问题规模或计算资源的增加而线性提高的现象。

并行软件设计和并行系统结构是可扩展性一软一硬的两个方面。可扩展性的不连续和非线性难题包括架构移植难、算法设计难、应用优化难等诸多表现形式。本文对大规模并行多层次不连续非线性可扩展理论开展研究，深入分析可扩展性发展规律，提出物理模型、并行算法以及性能优化多层次协同设计方法，在多种硬件并行规模、不同软件并行粒度、各级应用优化层次上开展可扩展性优化设计，具体贡献包括：

1. 提出了大规模并行多层次不连续非线性可扩展理论。通过对并行软件在同构多核到异构众核系统上可扩展性的分析，发现不同并行粒度的应用在物理模型、并行算法以及性能优化等多个层次上存在可扩展性的两种典型现象，即不连续和非线性。系统地对多个层次上的两种现象进行了分析，提出了可扩展性的多层次协同设计理论，为高性能计算领域可扩展性的研究，特别是大规模并行计算中的可扩展问题，提供了方法论层面指导。

2. 提出了百核量级的 **Stencil** 并行数值算法，利用新型向量化和分块技术实

现高可扩展单机多核 **Stencil** 计算。在并行算法层次，提出了转置布局计算和时空计算折叠两种 Stencil 向量化策略，提高数据在 CPU 内的并行度；同时在性能优化层次，设计高效寄存器数据重用算法和缓存分块优化算法，提高数据的访存效率。实验结果表明，通过并行算法-性能优化的协同设计思想，最高超过 state-of-the-art 方法的绝对性能 4.39 倍，有效地提升了 Stencil 并行数值算法的多核可扩展性能。

3. 提出了万核量级的分布式机器学习框架，利用新型聚类和回归技术实现高可扩展的多机众核机器学习预测模型。本文在理论建模层次，提出了新型 Best Friend 图数据结构及层次化的最小生成树网络模型，设计基于聚类的回归预测方法；在并行算法层次，提出了基于回溯的负载均衡算法和高效的并行通信算法，降低分布式系统的计算和通信开销。实验结果表明，通过物理模型-并行算法协同设计思想，在保证聚类和回归方法准确性的同时，还能有效地将分布式机器学习框架的多机众核扩展性由已有工作的 1,536 核提升至 12,288 核。

4. 针对科学计算软件应用优化，提出了百万核量级的大规模扩展方法并设计实现一套大规模高可扩展国产核材料辐照损伤模拟的软件应用 **OpenKMC**。在物理模型层次，实现优化了高可扩展的新型势函数模型和分组反应策略，支撑应用高效动力学蒙特卡洛模型建立；在并行算法层次，提出了适应于大规模应用的并行同步象限算法和高效自适应通信算法，提高应用的负载均衡和通信效率；在性能优化层次，提出了访存优化技术、高效局部性算法、Athread 线程级异构并行策略和向量化加速方法，通过层次化访存特征提取和轻量级并行性挖掘对异构众核体系结构算力进行优化利用。实验结果表明，通过物理模型-并行算法-性能优化的可扩展性协同设计思想，实现神威·太湖之光上千万亿原子的 520 万核大规模模拟，并行效率高达 80%，成为核材料模拟新的里程碑。

## 6.2 研究展望

本文从科学计算应用优化、并行数值算法设计和分布式机器学习框架研发这三种涵盖不同技术方向、不同并行粒度的层级出发，系统地总结了在高性能计算中面对不同粒度的并行需求时所遇到的可扩展性的不连续性和非线性现象。其中，细粒度并行问题主要针对的是本方向，即高性能计算中经典的 stencil 并行数值算法向量化的单机多核可扩展性；中粒度并行问题主要关注的是跨方向，

即高性能计算和人工智能的融合技术中基于聚类的回归预测框架同构众核的可扩展性；而粗粒度并行问题则主要研究的是跨学科，即科学计算中材料模拟的大规模异构众核的可扩展性。

近年来，机器学习与物理建模的结合正在改变着科学的研究的范式，而人工智能和高性能计算的技术融合则有望革命性地改变科学计算领域。例如，2020年的“戈登贝尔奖”即是以科学计算 + 机器学习 + 高性能计算为交叉研究的第一个新型科学范例。不同领域的科学家们正在以前所未有的新方式集结起来，希望通过 AI+HPC 的范式突破科学边界、解决困难问题。相比传统的高性能计算，智能科学计算在计算能效上有成量级的提升。然而，人工智能的需求超越了任何一款芯片单独处理的能力，追求高性能集群上更大规模应用的高可扩展性是未来的必然趋势。为此，未来可以在以下几个方面对于本文的多层次协同设计理论进行进一步地扩充研究：

1. 高性能异构资源管理器。随着高性能计算和人工智能技术的互相融合，具有 GPU 或其它加速器的高并行、高密集计算能力的异构架构将成为支撑复杂应用的必然的选择。然而，现有大部分应用的硬件资源调度模式中使用 MPI 技术单独管理 CPU，GPU 则作为辅助的计算加速设备。从**硬件架构的性能优化**出发，在 HPC+AI 的融合场景中，如何实现 CPU 作业和 GPU 作业的融合调度，提升资源利用率，是提高架构层可扩展性的核心。目前，微软亚洲研究院从提升内存资源利用率的角度，率先提出了 ZeRO 模型，使得无需重构应用代码即可支持在内存中运行大型深度学习应用；从提升计算资源利用率的角度，华为则新推出了昇腾 910 人工智能处理器，从底层设计直接将芯片上的部分核心部署为 AI CPU，承担了部分 AI 计算功能。而对于高性能异构资源管理器的整体设计，目前还尚未有相关工作提出。

2. 高通用智能基础算法库。由于高性能计算和人工智能计算在矩阵类型、数据结构、计算精度以及底层架构优化上都存在较大差异，现有大部分高性能计算基础数学库无法直接与人工智能计算相融合。从**并行算法设计的层级**出发，如何改进现有的张量结构，增强张量的表达能力，以实现更为通用的高性能基础算法库，是提高算法层可扩展性的关键。目前，华为率先提出了科学计算套件 MindScience，通过对多种基础算法的封装来实现对科学计算应用的模拟。然而，MindScience 对算子的抽象层级比较简单，且其更多地面向自研处理器架构进行

优化，移植性较差。

3. 高扩展自动并行化框架。通信是影响并行计算可扩展性非线性增长的关键因素之一。同样地，HPC+AI 应用的性能也会严重受到通信性能的制约。然而，复杂的通信模式设计使得科学家们并不能高效地将程序进行大规模扩展。因此，高扩展自动并行化框架设计则显得尤为重要。从物理模型的抽象设计层级出发，将更多的通信算法进行高效设计与集成，使得上层应用开发者能够更加关注应用本身，甚至只关注串行代码的逻辑实现，由并行化框架完成高扩展的自动分布式计算，对高性能计算和人工智能的真正融合具有重要意义。目前，微软亚洲研究院提出的 DeepSpeed 深度学习优化库通过张量切片和 ZeRO-Offload 等技术来优化通信效率，以支持大规模应用的处理。然而，其并未实现自动并行，复杂的接口设计对应用开发者并不友好；华为 MindSpore 实现了专为科学计算而优化的自动并行框架。然而，其张量切分策略较为简单，性能有待进一步优化提升。因此，对高扩展自动并行化框架的设计，仍是 HPC+AI 技术融合中的痛点问题。

## 参考文献

- [1] Zahn C T. Graph-theoretical methods for detecting and describing gestalt clusters [J]. IEEE Transactions on computers, 1971, 100(1): 68-86.
- [2] Gionis A, Mannila H, Tsaparas P. Clustering aggregation [J]. Acm transactions on knowledge discovery from data (tkdd), 2007, 1(1): 4-es.
- [3] Veenman C J, Reinders M J T, Backer E. A maximum variance cluster algorithm [J]. IEEE Transactions on pattern analysis and machine intelligence, 2002, 24(9): 1273-1280.
- [4] Zhang Y, Duchi J, Wainwright M. Divide and conquer kernel ridge regression: A distributed algorithm with minimax optimal rates [J]. The Journal of Machine Learning Research, 2015, 16(1): 3299-3340.
- [5] You Y. Fast and accurate machine learning on distributed systems and supercomputers [D]. Ph. D. Dissertation. UC Berkeley, 2020.
- [6] Sun T, Shu C, Li F, et al. An efficient hierarchical clustering method for large datasets with map-reduce [C/OL]//2009 International Conference on Parallel and Distributed Computing, Applications and Technologies. 2009: 494-499. DOI: 10.1109/PDCAT.2009.46.
- [7] Li J, Wang X, Wang X. A scaled-mst-based clustering algorithm and application on image segmentation [J]. Journal of Intelligent Information Systems, 2019: 1-25.
- [8] Lê T, Barbu A, Liu D, et al. Precipitation kinetics of dilute fecu and fecum alloys subjected to electron irradiation [J/OL]. Scripta Metallurgica et Materialia, 1992, 26(5): 771 - 776. <http://www.sciencedirect.com/science/article/pii/0956716X9290436I>. DOI: [https://doi.org/10.1016/0956-716X\(92\)90436-I](https://doi.org/10.1016/0956-716X(92)90436-I).
- [9] Vincent E, Becquart C, Domain C. Solute interaction with point defects in  $\alpha$  fe during thermal ageing: A combined ab initio and atomic kinetic monte carlo approach [J]. Journal of nuclear materials, 2006, 351(1-3): 88-99.
- [10] 臧大伟, 曹政, 孙凝晖. 高性能计算的发展 [J]. 科技导报, 2016, 34(14): 22-28.
- [11] 张云泉, 袁良, 陈一峯, 等. 高性能计算多层次不连续非线性可扩展现象研究 [J]. 计算机学报, 2020, 43(6): 973-989.
- [12] 张云泉, 袁国兴, 孙家昶, 等. 中国高性能计算机 TOP100 十周年回顾与展望 [J]. 计算机工程与科学, 2012, 34(8): 11-16.
- [13] project T. Top500 list [EB/OL]. 2021. <https://www.top500.org/lists/top500/2021/11/>.
- [14] Liu Y, Liu X, Li F, et al. Closing the " quantum supremacy" gap: achieving real-time simulation of a random quantum circuit using a new sunway supercomputer [C]//Proceedings

- of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2021: 1-12.
- [15] 杨广文, 赵文来, 丁楠, 等. “神威·太湖之光”及其应用系统 [J]. 科学, 2017, 69(3): 12-16.
- [16] 孙凝晖, 谭光明. 高性能计算机发展与政策 [J]. 中国科学院院刊, 2019, 34(6): 609-616.
- [17] 张云泉, 袁良, 袁国兴, 等. 2021 年中国高性能计算机发展现状分析与展望 [J]. 数据与计算发展前沿, 2021.
- [18] 莫则尧. 实用的并行程序性能分析方法 [J]. 数值计算与计算机应用, 2000, 21(4): 266-275.
- [19] 陈军, 莫则尧, 李晓梅, 等. 大规模并行应用程序的可扩展性研究 [J]. 计算机研究与发展, 2000, 37(11): 1382-1388.
- [20] 陈国良, 苗乾坤, 孙广中, 等. 分层并行计算模型 [J]. 中國科學技術大學學報, 2008, 38(7): 841-847.
- [21] 陈国良, 孙广中, 徐云, 等. 并行计算的一体化研究现状与发展趋势 [J]. 科学通报, 2009(8): 1043-1049.
- [22] Luo L, Straatsma T P, Suarez L A, et al. Pre-exascale accelerated application development: The ornl summit experience [J]. IBM Journal of Research and Development, 2020, 64(3/4): 11-1.
- [23] 刘鑫, 郭恒, 孙茹君, 等. “神威·太湖之光”计算机系统大规模应用特征分析与 E 级可扩展性研究 [J]. 计算机学报, 2018.
- [24] Asanovic K, Bodik R, Catanzaro B C, et al. The landscape of parallel computing research: A view from berkeley [J]. 2006.
- [25] Sawdey A, O'Keefe M, Bleck R, et al. The design, implementation, and performance of a parallel ocean circulation model [C]//Proceedings of 6th ECMWF Workshop on the Use of Parallel Processors in Meteorology: Coming of Age. 1995: 523-550.
- [26] Li K, Shang H, Zhang Y, et al. Openkmc: a kmc design for hundred-billion-atom simulation using millions of cores on sunway taihulight [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019: 1-16.
- [27] Asanovic K, Bodik R, Demmel J, et al. The parallel computing laboratory at uc berkeley: A research agenda based on the berkeley view [J]. EECS Department, University of California, Berkeley, Tech. Rep, 2008.
- [28] Yuan L, Zhang Y, Guo P, et al. Tessellating stencils [C/OL]//SC '17: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. New York, NY, USA: Association for Computing Machinery, 2017. <https://doi.org/10.1145/3126908.3126920>.
- [29] 张云泉. 对当前人工智能热的冷思考 [J]. 高科技与产业化, 2018(2): 14-17.

- [30] 张云泉, 徐葳, 龙桂鲁. 数据科学: 问题导向的交叉学科创新 [J]. 科学通报, 2015(5): 425-426.
- [31] Draper N R, Smith H. Applied regression analysis: volume 326 [M]. John Wiley & Sons, 1998.
- [32] Kuhn M. Caret: classification and regression training [J]. ascl, 2015: ascl-1505.
- [33] 金钟, 陆忠华, 李会元, 等. 高性能计算之源起——科学计算的应用现状及发展思考 [J]. 中国科学院院刊, 2019, 34(6): 625-639.
- [34] 杨文, 胡长军, 刘天才, 等. 数值反应堆原型系统开发及示范应用研究进展 [J]. 原子能科学技术, 2021, 55(9): 1537.
- [35] 田卓, 陈一峯. 神威太湖之光上分子动力学模拟的性能优化 [J]. Journal of Software, 2021, 32(9): 2945-2962.
- [36] Leetmaa M, Skorodumova N V. Kmclib: A general framework for lattice kinetic monte carlo (kmc) simulations [J]. Computer Physics Communications, 2014, 185(9): 2340-2349.
- [37] Blackman J, Mulheran P. Growth of clusters on surfaces: Monte carlo simulations and scaling properties [J]. Computer physics communications, 2001, 137(1): 195-205.
- [38] Fang F, Shu X, Deng H, et al. Modified analytic eam potentials for the binary immiscible alloy systems [J]. Materials Science and Engineering: A, 2003, 355(1-2): 357-367.
- [39] Domain C, Becquart C, Van Duyse J. Kinetic monte carlo simulations of fecu alloys [J]. MRS Online Proceedings Library Archive, 1998, 538.
- [40] 李学干. 计算机系统的体系结构 [M]. 清华大学出版社有限公司, 2006.
- [41] 袁良, 张云泉, 白雪瑞, 等. 并行程序设计语言中局部性机制的研究 [J]. 计算机科学, 2020, 47(1): 7-16.
- [42] Yuan L, Huang S, Zhang Y, et al. Tessellating star stencils [C]//Proceedings of the 48th International Conference on Parallel Processing. 2019: 1-10.
- [43] Hennessy J L, Patterson D A. Computer architecture: a quantitative approach [M]. Elsevier, 2011.
- [44] Duan X, Chen D, Meng X, et al. Redesigning lammps for peta-scale and hundred-billion-atom simulation on sunway taihulight [C]//Redesigning LAMMPS for Peta-Scale and Hundred-Billion-Atom Simulation on Sunway TaihuLight. IEEE, 2018: 0.
- [45] Tang Y, Chowdhury R A, Kuszmaul B C, et al. The pochoir stencil compiler [C/OL]//SPAA '11: Proceedings of the Twenty-Third Annual ACM Symposium on Parallelism in Algorithms and Architectures. New York, NY, USA: Association for Computing Machinery, 2011: 117-128. <https://doi.org/10.1145/1989493.1989508>.
- [46] Krishnamoorthy S, Baskaran M, Bondhugula U, et al. Effective automatic parallelization of

- stencil computations [J/OL]. SIGPLAN Not., 2007, 42(6): 235-244. <https://doi.org/10.1145/1273442.1250761>.
- [47] de la Cruz R, Araya-Polo M. Algorithm 942: Semi-stencil [J/OL]. ACM Trans. Math. Softw., 2014, 40(3): 23:1-23:39. <http://doi.acm.org/10.1145/2591006>.
- [48] Zhao T, Basu P, Williams S, et al. Exploiting reuse and vectorization in blocked stencil computations on cpus and gpus [C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 2019: 1-44.
- [49] Basu P, Hall M, Williams S, et al. Compiler-directed transformation for higher-order stencils [C]//2015 IEEE International Parallel and Distributed Processing Symposium. 2015: 313-323.
- [50] Rawat P S, Rastello F, Sukumaran-Rajam A, et al. Register optimizations for stencils on gpus [C]//Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. 2018: 168-182.
- [51] Rawat P S, Sukumaran-Rajam A, Rountev A, et al. Associative instruction reordering to alleviate register pressure [C]//SC18: International Conference for High Performance Computing, Networking, Storage and Analysis. 2018: 590-602.
- [52] Henretty T, Stock K, Pouchet L N, et al. Data layout transformation for stencil computations on short-vector simd architectures [C]//International Conference on Compiler Construction. Springer, 2011: 225-245.
- [53] Henretty T, Veras R, Franchetti F, et al. A stencil compiler for short-vector simd architectures [C/OL]//ICS '13: Proceedings of the 27th International ACM Conference on International Conference on Supercomputing. New York, NY, USA: Association for Computing Machinery, 2013: 13-24. <https://doi.org/10.1145/2464996.2467268>.
- [54] Malas T M, Hager G, Ltaief H, et al. Multidimensional intratile parallelization for memory-starved stencil computations [J/OL]. ACM Trans. Parallel Comput., 2017, 4(3). <https://doi.org/10.1145/3155290>.
- [55] Tian X, Bik A, Girkar M, et al. Intel® openmp c++/fortran compiler for hyper-threading technology: Implementation and performance. [J]. Intel Technology Journal, 2002, 6(1).
- [56] Bondhugula U, Hartono A, Ramanujam J, et al. A practical automatic polyhedral parallelizer and locality optimizer [C]//Proceedings of the 29th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2008: 101-113.
- [57] Bandishti V, Pananilath I, Bondhugula U. Tiling stencil computations to maximize parallelism [C]//SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2012: 1-11.
- [58] Yount C, Tobin J, Breuer A, et al. Yask-yet another stencil kernel: A framework for hpc stencil

- code-generation and tuning [C]//2016 Sixth International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing (WOLFHPC). IEEE, 2016: 30-39.
- [59] Datta K, Kamil S, Williams S, et al. Optimization and performance modeling of stencil computations on modern microprocessors [J]. SIAM review, 2009, 51(1): 129-159.
- [60] Falke S, Merz F, Sinz C. Extending the theory of arrays: memset, memcpy, and beyond [C]//Working Conference on Verified Software: Theories, Tools, and Experiments. Springer, 2013: 108-128.
- [61] Datta K, Murphy M, Volkov V, et al. Stencil computation optimization and auto-tuning on state-of-the-art multicore architectures [C]//SC'08: Proceedings of the 2008 ACM/IEEE conference on Supercomputing. IEEE, 2008: 1-12.
- [62] Kamil S, Datta K, Williams S, et al. Implicit and explicit optimizations for stencil computations [C]//Proceedings of the 2006 workshop on Memory system performance and correctness. 2006: 51-60.
- [63] Meng J, Skadron K. Performance modeling and automatic ghost zone optimization for iterative stencil loops on gpus [C]//Proceedings of the 23rd international conference on Supercomputing. 2009: 256-265.
- [64] Venkatasubramanian S, Vuduc R W, none n. Tuned and wildly asynchronous stencil kernels for hybrid cpu/gpu systems [C]//Proceedings of the 23rd international conference on Supercomputing. 2009: 244-255.
- [65] Strzodka R, Shaheen M, Pajak D, et al. Cache oblivious parallelograms in iterative stencil computations [C]//Proceedings of the 24th ACM International Conference on Supercomputing. 2010: 49-59.
- [66] Allen R, Kennedy K. Optimizing compilers for modern architectures: a dependence-based approach [M]. Taylor & Francis US, 2002.
- [67] Zekri A S. Enhancing the matrix transpose operation using intel avx instruction set extension [J]. International Journal of Computer Science & Information Technology, 2014, 6(3): 67.
- [68] Hormati A H, Choi Y, Woh M, et al. Macross: macro-simdization of streaming applications [J]. ACM SIGARCH computer architecture news, 2010, 38(1): 285-296.
- [69] Springer P, Hammond J R, Bientinesi P. Ttc: A high-performance compiler for tensor transpositions [J]. ACM Transactions on Mathematical Software (TOMS), 2017, 44(2): 1-21.
- [70] Stock K, Kong M, Grosser T, et al. A framework for enhancing data reuse via associative reordering [C]//Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation. 2014: 65-76.
- [71] Yount C. Vector folding: improving stencil performance via multi-dimensional simd-vector

- representation [C]//2015 IEEE 17th International Conference on High Performance Computing and Communications, 2015 IEEE 7th International Symposium on Cyberspace Safety and Security, and 2015 IEEE 12th International Conference on Embedded Software and Systems. IEEE, 2015: 865-870.
- [72] Aho A V, Johnson S C, Ullman J D. Code generation for expressions with common subexpressions [C]//Proceedings of the 3rd ACM SIGACT-SIGPLAN symposium on Principles on programming languages. 1976: 19-31.
- [73] Deitz S J, Chamberlain B L, Snyder L. Eliminating redundancies in sum-of-product array computations [C]//Proceedings of the 15th international conference on Supercomputing. 2001: 65-77.
- [74] Irigoin F, Triolet R. Supernode partitioning [C]//Proceedings of the 15th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. 1988: 319-329.
- [75] McKellar A C, Coffman E G, Jr. Organizing matrices and matrix operations for paged memory systems [J]. Commun. ACM, 1969, 12(3): 153-165.
- [76] Lam M D, Rothberg E E, Wolf M E. The cache performance and optimizations of blocked algorithms [J]. ACM SIGOPS Operating Systems Review, 1991, 25(Special Issue): 63-74.
- [77] Wolf M E, Lam M S. A data locality optimizing algorithm [C]//Proceedings of the ACM SIGPLAN 1991 conference on Programming language design and implementation. 1991: 30-44.
- [78] Wolfe M. More iteration space tiling [C]//Proceedings of the 1989 ACM/IEEE conference on Supercomputing. 1989: 655-664.
- [79] Ding C, He Y. A ghost cell expansion method for reducing communications in solving pde problems [C]//SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing. IEEE, 2001: 55-55.
- [80] Rastello F, Dauxois T. Efficient tiling for an ode discrete integration program: Redundant tasks instead of trapezoidal shaped-tiles. [C]//ipdps. 2002.
- [81] Rivera G, Tseng C W. Tiling optimizations for 3d scientific computations [C]//SC'00: Proceedings of the 2000 ACM/IEEE conference on Supercomputing. IEEE, 2000: 32-32.
- [82] Nguyen A, Satish N, Chhugani J, et al. 3.5-d blocking optimization for stencil computations on modern cpus and gpus [C]//SC'10: Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE, 2010: 1-13.
- [83] Song Y, Li Z. New tiling techniques to improve cache temporal locality [J]. ACM SIGPLAN Notices, 1999, 34(5): 215-228.

- [84] Wonnacott D. Achieving scalable locality with time skewing [J]. Int. J. Parallel Program., 2002, 30(3): 181-221.
- [85] Jin G, Mellor-Crummey J, Fowler R. Increasing temporal locality with skewing and recursive blocking [C]//SC'01: Proceedings of the 2001 ACM/IEEE Conference on Supercomputing. IEEE, 2001: 57-57.
- [86] Frigo M, Strumpen V. Cache oblivious stencil computations [C]//Proceedings of the 19th annual international conference on Supercomputing. 2005: 361-366.
- [87] Wonnacott D G, Strout M M. On the scalability of loop tiling techniques [J]. IMPACT 2013, 2013.
- [88] Christen M, Schenk O, Burkhardt H. Patus: A code generation and autotuning framework for parallel iterative stencil computations on modern microarchitectures [C]//2011 IEEE International Parallel & Distributed Processing Symposium. IEEE, 2011: 676-687.
- [89] Gysi T, Grosser T, Hoefer T. Modesto: Data-centric analytic optimization of complex stencil programs on heterogeneous architectures [C]//Proceedings of the 29th ACM on International Conference on Supercomputing. 2015: 177-186.
- [90] Kamil S, Chan C, Oliker L, et al. An auto-tuning framework for parallel multicore stencil computations [C]//2010 IEEE International Symposium on Parallel & Distributed Processing (IPDPS). IEEE, 2010: 1-12.
- [91] Zhang Y, Mueller F. Auto-generation and auto-tuning of 3d stencil codes on gpu clusters [C]// Proceedings of the Tenth International Symposium on Code Generation and Optimization. 2012: 155-164.
- [92] Henretty T, Veras R, Franchetti F, et al. A stencil compiler for short-vector SIMD architectures [C]//Proceedings of the 27th international ACM conference on International conference on supercomputing. 2013: 13-24.
- [93] Guide I I. Url: <https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html> [J]. IntrinsicsGuide (access date: 12.6.2021), 2021.
- [94] of University of Oregon C D. Stencil pattern [EB/OL]. 2014. <https://ipcc.cs.uoregon.edu/lectures/lecture-8-stencil.pdf>.
- [95] Rawat P, Kong M, Henretty T, et al. Sdslc: A multi-target domain-specific compiler for stencil computations [C]//Proceedings of the 5th International Workshop on Domain-Specific Languages and High-Level Frameworks for High Performance Computing. 2015: 1-10.
- [96] Intel. Intel processor counter monitor [EB/OL]. 2021. <https://github.com/opcm/pcm>.
- [97] Wikichip.org. Wikichip of intel xeon gold 6140 [EB/OL]. 2019. [https://en.wikichip.org/wiki/intel/xeon\\_gold/6140](https://en.wikichip.org/wiki/intel/xeon_gold/6140).

- [98] Ding C, He X. K-means clustering via principal component analysis [C]//Proceedings of the twenty-first international conference on Machine learning. 2004: 29.
- [99] Chen G, Song X, Zeng H, et al. Scene recognition with prototype-agnostic scene layout [J]. IEEE Transactions on Image Processing, 2020, 29: 5877-5888.
- [100] Coates A, Huval B, Wang T, et al. Deep learning with cots hpc systems [C]//International conference on machine learning. 2013: 1337-1345.
- [101] Xue Z, Wang H. Effective density-based clustering algorithms for incomplete data [J]. Big Data Mining and Analytics, 2021, 4(3): 183-194.
- [102] Tian Y, Zheng R, Liang Z, et al. A data-driven clustering recommendation method for single-cell rna-sequencing data [J]. Tsinghua Science and Technology, 2021, 26(5): 772-789.
- [103] Li J, Jiao H, Wang J, et al. Online real-time trajectory analysis based on adaptive time interval clustering algorithm [J]. Big Data Mining and Analytics, 2020, 3(2): 131-142.
- [104] Hu J, Pan Y, Li T, et al. Tw-co-mfc: Two-level weighted collaborative fuzzy clustering based on maximum entropy for multi-view data [J]. Tsinghua Science and Technology, 2020, 26 (2): 185-198.
- [105] Pu G, Wang L, Shen J, et al. A hybrid unsupervised clustering-based anomaly detection method [J]. Tsinghua Science and Technology, 2020, 26(2): 146-153.
- [106] Zhao X, Wang Z, Gao L, et al. Incremental face clustering with optimal summary learning via graph convolutional network [J]. Tsinghua Science and Technology, 2021, 26(4): 536-547.
- [107] Wang N, Guo G, Wang B, et al. Traffic clustering algorithm of urban data brain based on a hybrid-augmented architecture of quantum annealing and brain-inspired cognitive computing [J]. Tsinghua Science and Technology, 2020, 25(6): 813-825.
- [108] Murtagh F, Contreras P. Algorithms for hierarchical clustering: an overview [J]. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 2012, 2(1): 86-97.
- [109] Likas A, Vlassis N, Verbeek J J. The global k-means clustering algorithm [J]. Pattern recognition, 2003, 36(2): 451-461.
- [110] Li K, Yuan L, Zhang Y, et al. Reducing redundancy in data organization and arithmetic calculation for stencil computations [C/OL]//SC '21: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. St. Louis, Missouri, 2021. <https://doi.org/10.1145/3458817.3476154>.
- [111] Kleinberg J, Tardos E. Algorithm design [M]. Pearson Education India, 2006.
- [112] Grygorash O, Zhou Y, Jorgensen Z. Minimum spanning tree based clustering algorithms [C]// 2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06). IEEE, 2006: 73-81.

- [113] Sun Z, Fox G. Study on parallel svm based on mapreduce [C]//Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA). The Steering Committee of The World Congress in Computer Science, Computer ..., 2012: 1.
- [114] Hsieh C J, Si S, Dhillon I. A divide-and-conquer solver for kernel support vector machines [C]//International conference on machine learning. 2014: 566-574.
- [115] Zhang Y, Duchi J, Wainwright M. Divide and conquer kernel ridge regression [C]//Conference on learning theory. 2013: 592-617.
- [116] You Y, Demmel J, Hsieh C J, et al. Accurate, fast and scalable kernel ridge regression on parallel and distributed systems [C]//Proceedings of the 2018 International Conference on Supercomputing. 2018: 307-317.
- [117] Bateni M H, Behnezhad S, Derakhshan M, et al. Affinity clustering: Hierarchical clustering at scale [C]//NIPS. 2017: 6867-6877.
- [118] Tan P N, Steinbach M, Kumar V. Introduction to data mining [M]. Pearson Education India, 2016.
- [119] Bahmani B, Moseley B, Vattani A, et al. Scalable k-means++ [J/OL]. Proc. VLDB Endow., 2012, 5(7): 622-633. <https://doi.org/10.14778/2180912.2180915>.
- [120] Ene A, Im S, Moseley B. Fast clustering using mapreduce [C]//Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining. 2011.
- [121] Balcan M F F, Ehrlich S, Liang Y. Distributed  $k$ -means and  $k$ -median clustering on general topologies [J]. Advances in Neural Information Processing Systems, 2013.
- [122] Bateni M, Bhaskara A, Lattanzi S, et al. Distributed balanced clustering via mapping coresets. [C]//NIPS. 2014: 2591-2599.
- [123] Anchalia P P, Koundinya A K, Srinath N. Mapreduce design of k-means clustering algorithm [C]//2013 International Conference on Information Science and Applications (ICISA). 2013.
- [124] Bouguettaya A, Yu Q, Liu X, et al. Efficient agglomerative hierarchical clustering [J]. Expert Systems with Applications, 2015, 42(5): 2785-2797.
- [125] Bradley P S, Fayyad U M. Refining initial points for k-means clustering. [C]//ICML: volume 98. Citeseer, 1998: 91-99.
- [126] Kodinariya T M, Makwana P R. Review on determining number of cluster in k-means clustering [J]. International Journal, 2013, 1(6): 90-95.
- [127] Ahn K J, Guha S, McGregor A. Analyzing graph structure via linear measurements [C]//Proceedings of the twenty-third annual ACM-SIAM symposium on Discrete Algorithms. SIAM, 2012: 459-467.
- [128] Andoni A, Nikolov A, Onak K, et al. Parallel algorithms for geometric graph problems [C]//

- Proceedings of the forty-sixth annual ACM symposium on Theory of computing. 2014: 574-583.
- [129] Chitnis R, Cormode G, Esfandiari H, et al. Kernelization via sampling with applications to finding matchings and related problems in dynamic graph streams [C]//Proceedings of the twenty-seventh annual ACM-SIAM symposium on Discrete algorithms. SIAM, 2016: 1326-1344.
- [130] Esfandiari H, Hajiaghayi M, Liaghat V, et al. Streaming algorithms for estimating the matching size in planar graphs and beyond [J]. ACM Transactions on Algorithms (TALG), 2018, 14(4): 1-23.
- [131] Jin C, Patwary M M A, Agrawal A, et al. Disc: A distributed single-linkage hierarchical clustering algorithm using mapreduce [J]. work, 2013, 23: 27.
- [132] Wang X, Wang X, Wilkes D M. A divide-and-conquer approach for minimum spanning tree-based clustering [J]. IEEE Transactions on Knowledge and Data Engineering, 2009, 21(7): 945-958.
- [133] Zhong C, Malinen M, Miao D, et al. A fast minimum spanning tree algorithm based on k-means [J]. Information Sciences, 2015, 295: 1-17.
- [134] Chen C P, Zhang C Y. Data-intensive applications, challenges, techniques and technologies: A survey on big data [J]. Information sciences, 2014, 275: 314-347.
- [135] Hinton G, Srivastava N, Swersky K. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent [J]. Cited on, 2012, 14(8).
- [136] He Q, Shang T, Zhuang F, et al. Parallel extreme learning machine for regression based on mapreduce [J]. Neurocomputing, 2013, 102: 52-58.
- [137] Yang H, Luan Z, Li W, et al. Mapreduce workload modeling with statistical approach [J]. Journal of grid computing, 2012, 10(2): 279-310.
- [138] Mitra P, Murthy C, Pal S K. A probabilistic active support vector learning algorithm [J]. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2004, 26(3): 413-418.
- [139] Bekkerman R, Bilenko M, Langford J. Scaling up machine learning: Parallel and distributed approaches [M]. Cambridge University Press, 2011.
- [140] Wikipedia. Global city [Z]. 2020.
- [141] Asano T, Bhattacharya B, Keil M, et al. Clustering algorithms based on minimum and maximum spanning trees [C]//Proceedings of the fourth annual symposium on Computational geometry. 1988: 252-257.
- [142] Muja M, Lowe D G. Fast approximate nearest neighbors with automatic algorithm configuration. [J]. VISAPP (1), 2009, 2(331-340): 2.

- [143] Friedman J H, Bentley J L, Finkel R A. An algorithm for finding best matches in logarithmic expected time [J]. ACM Transactions on Mathematical Software (TOMS), 1977.
- [144] Fränti P, Sieranoja S. K-means properties on six clustering benchmark datasets [J]. Applied Intelligence, 2018, 48(12): 4743-4759.
- [145] Dua D, Graff C. Uci machine learning repository [EB/OL]. University of California, Irvine, School of Information and Computer Sciences, 2019. <http://archive.ics.uci.edu/ml>.
- [146] Vinh N X, Epps J, Bailey J. Information theoretic measures for clusterings comparison: is a correction for chance necessary? [C]//Proceedings of the 26th annual international conference on machine learning. 2009: 1073-1080.
- [147] Milford J B, Kreith F, Kutscher C F. Nuclear energy [M]//Principles of Sustainable Energy Systems, Third Edition. CRC Press, 2018: 247-268.
- [148] Vincent E, Becquart C, Domain C. Microstructural evolution under high flux irradiation of dilute fe–cunimnsi alloys studied by an atomic kinetic monte carlo model accounting for both vacancies and self interstitials [J]. Journal of Nuclear Materials, 2008, 382(2-3): 154-159.
- [149] Messina L, Castin N, Domain C, et al. Introducing ab initio based neural networks for transition-rate prediction in kinetic monte carlo simulations [J]. Physical Review B, 2017, 95(6): 064112.
- [150] Castin N, Messina L, Domain C, et al. Improved atomistic monte carlo models based on ab-initio-trained neural networks: Application to fecu and fecr alloys [J]. Physical Review B, 2017, 95(21): 214117.
- [151] Becquart C S, Domain C, Sarkar U, et al. Microstructural evolution of irradiated tungsten: Ab initio parameterisation of an okmc model [J]. Journal of nuclear materials, 2010, 403 (1-3): 75-88.
- [152] Domain C, Becquart C S, Malerba L. Simulation of radiation damage in Fe alloys: an object kinetic Monte Carlo approach [J/OL]. J. Nucl. Mater., 2004, 335(1): 121-145. <http://www.sciencedirect.com/science/article/pii/S0022311504006385>. DOI: <https://doi.org/10.1016/j.jnucmat.2004.07.037>.
- [153] Domain C, Becquart C, Malerba L. Simulation of radiation damage in fe alloys: an object kinetic monte carlo approach [J]. Journal of Nuclear Materials, 2004, 335(1): 121-145.
- [154] Lazo J. Numerical simulations of microstructural evolution of iron alloys under irradiation [Z]. 2013.
- [155] Domain C, Becquart C. Ab initio calculations of defects in fe and dilute fe-cu alloys [J]. Physical Review B, 2001, 65(2): 024103.
- [156] Domain C, Becquart C, Foct J. Ab initio study of foreign interstitial atom (c, n) interactions with intrinsic point defects in  $\alpha$ -fe [J]. Physical Review B, 2004, 69(14): 144112.

- [157] Becquart C S, Domain C. Introducing chemistry in atomistic kinetic Monte Carlo simulations of Fe alloys under irradiation [J/OL]. *Phys. status solidi*, 2010, 247(1): 9-22. <http://doi.wiley.com/10.1002/pssb.200945251>.
- [158] Soisson F, Becquart C, Castin N, et al. Atomistic Kinetic Monte Carlo studies of microchemical evolutions driven by diffusion processes under irradiation [J/OL]. *J. Nucl. Mater.*, 2010, 406(1): 55-67. <http://dx.doi.org/10.1016/j.jnucmat.2010.05.018><https://linkinghub.elsevier.com/retrieve/pii/S0022311510002308>.
- [159] Olsson P, Klaver T, Domain C. Ab initio study of solute transition-metal interactions with point defects in bcc fe [J]. *Physical Review B*, 2010, 81(5): 054102.
- [160] Young W M, Elcock E W. Monte carlo studies of vacancy migration in binary ordered alloys: I [J/OL]. *Proceedings of the Physical Society*, 1966, 89(3): 735-746. <https://doi.org/10.1088%2F0370-1328%2F89%2F3%2F329>. DOI: 10.1088/0370-1328/89/3/329.
- [161] Mathon M, Barbu A, Dunstetter F, et al. Experimental study and modelling of copper precipitation under electron irradiation in dilute fecu binary alloys [J]. *Journal of nuclear materials*, 1997, 245(2-3): 224-237.
- [162] Fu H, Liao J, Yang J, et al. The Sunway TaihuLight supercomputer: system and applications [J]. *Science China Information Sciences*, 2016, 59(7): 072001.
- [163] Fang J, Fu H, Zhao W, et al. swdnn: A library for accelerating deep learning applications on sunway taihulight [C]//Parallel and Distributed Processing Symposium (IPDPS), 2017 IEEE International. IEEE, 2017: 615-624.
- [164] Li K, Li S, Huang S, et al. FastNBL: fast neighbor lists establishment for molecular dynamics simulation based on bitwise operations [J]. *The Journal of Supercomputing*, 2019: 1-20.
- [165] Plimpton S, Battaile C, Chandross M, et al. Crossing the mesoscale no-man's land via parallel kinetic monte carlo [J]. *Sandia Report SAND2009-6226*, 2009.
- [166] Shim Y, Amar J G. Semirigorous synchronous sublattice algorithm for parallel kinetic monte carlo simulations of thin film growth [J/OL]. *Phys. Rev. B*, 2005, 71: 125432. <https://link.aps.org/doi/10.1103/PhysRevB.71.125432>.
- [167] Li J, Wei P, Yang S, et al. Crystal-kmc: parallel software for lattice dynamics monte carlo simulation of metal materials [J]. *Tsinghua Science and Technology*, 2018, 23(4): 501-510.
- [168] Jiménez F, Ortiz C. A gpu-based parallel object kinetic monte carlo algorithm for the evolution of defects in irradiated materials [J]. *Computational Materials Science*, 2016, 113: 178-186.
- [169] Moura A, Esteves A. Synchronous parallel kinetic monte carlo simulation of al3sc precipitation [J]. 2015.
- [170] Martínez E, Monasterio P R, Marian J. Billion-atom synchronous parallel kinetic monte carlo

- simulations of critical 3d ising systems [J]. Journal of Computational Physics, 2011, 230(4): 1359-1369.
- [171] van der Kaap N, Koster L J A. Massively parallel kinetic monte carlo simulations of charge carrier transport in organic semiconductors [J]. Journal of Computational Physics, 2016, 307: 321-332.
- [172] Soisson F, Barbu A, Martin G. Monte carlo simulations of copper precipitation in dilute iron-copper alloys during thermal ageing and under electron irradiation [J]. Acta Materialia, 1996, 44(9): 3789-3800.
- [173] Garcia Cardona C, Wagner G J, Tikare V, et al. Crossing the mesoscale no-mans land via parallel kinetic monte carlo. [R]. Sandia National Laboratories (SNL), Albuquerque, NM, and Livermore, CA …, 2009.
- [174] Slepoy A, Thompson A P, Plimpton S J. A constant-time kinetic monte carlo algorithm for simulation of large biochemical reaction networks [J]. The journal of chemical physics, 2008, 128(20): 05B618.
- [175] ISteve Plimpton A S, Aidan Thompson. SPPARKS Kinetic Monte Carlo Simulator [EB/OL]. 2009. <https://spparks.sandia.gov/>.
- [176] Gropp W D, Gropp W, Lusk E, et al. Using mpi: portable parallel programming with the message-passing interface: volume 1 [M]. MIT press, 1999.
- [177] Wu B, Li S, Zhang Y, et al. Hybrid-optimization strategy for the communication of large-scale kinetic monte carlo simulation [J]. Computer Physics Communications, 2017, 211: 113-123.
- [178] Li S, Wu B, Zhang Y, et al. Massively scaling the metal microscopic damage simulation on sunway taihulight supercomputer [C]//Proceedings of the 47th International Conference on Parallel Processing. ACM, 2018: 47.
- [179] Li S, Zhang Y, Hoefler T. Cache-oblivious mpi all-to-all communications based on morton order [J]. IEEE Transactions on Parallel and Distributed Systems, 2018, 29(3): 542-555.
- [180] Hoefler T, Siebert C, Lumsdaine A. Scalable communication protocols for dynamic sparse data exchange [J]. ACM Sigplan Notices, 2010, 45(5): 159-168.
- [181] Tersoff J. Empirical interatomic potential for silicon with improved elastic properties [J]. Physical Review B, 1988, 38(14): 9902.
- [182] Vincent E, Becquart C S, Pareige C, et al. Precipitation of the FeCu system: A critical review of atomic kinetic Monte Carlo simulations [J/OL]. J. Nucl. Mater., 2008, 373(1-3): 387-401.  
DOI: 10.1016/j.jnucmat.2007.06.016.



## 致 谢

少年与爱永不老去，即便披荆斩棘，丢失怒马鲜衣。

我出生在一个并不和善的大家庭里，奶奶不疼，姥姥不爱。因为父母工作的原因，我从小便被寄养在大家庭里。受人脸色是父母的常事，而被视为累赘的我也并未得到些许亲情的照顾。天不眷怜，一岁那年，高烧的我在身体多处落下病根。因为肺部病情多年的反复，中考和高考均因为紧急手术而相继缺席。高考失利，我在自己的房间哭了很久很久，最后做出了复读的决定。大概是上天对我的考验未曾停歇，第二次高考前夕，我又不得不接受了人生中的第三次手术。母亲在手术室门口哭成泪人，而我却对自己的境遇无能为力。

命运愈坎坷，我应愈坚强。出院后的一个月，我没有放弃这来之不易的机会，考入了山东大学。人情冷暖，成人后的世界远比我想像得复杂。我羡慕于在篮球场上酣畅爽快的他们，倾心于在舞蹈室里恣意曼妙的她们，然而我却始终无法与很多人成为我们。因为动辄缺席体育活动，明面儿上的偏见始终存在，背后的猜疑也从未停息。路灯微亮，晚风轻柔地让人无可抱怨，一闪而过的是我匆匆去往自习室的背影。夜铃脆响，朦胧的月色洒下治愈的微光，披在身上就是我收获的行囊。几经波折，四年的披荆斩棘终得以化作一张船票，而中科院计算所就是承载着我的希望与梦想的下一站。

最穷无非讨饭，不死终会出头。在外求学的十年，我可能是班里唯一一个没有打过游戏的男生。都说笨鸟先飞，而一只病鸟可能要花更多的力气才能和笨鸟并肩。身体或许并不完美，但总有一些可做的事和成功的机会。即便我被关在果壳之中，仍自以为是无限宇宙之王。有些自己在内心幽处的坚持和热爱，不能说，不能常想，却也不能忘。

学习中的每一次新发现都让我兴奋，实验中的每一个新想法也令我欢愉。冬去春来，我又一次地完成了暗中对自己说的事儿。衷心感谢我的恩师张云泉研究员，是您给了我的科研的船票，教引我叩响学术的大门。我攻读博士期间的工作全程是在您的悉心指导下逐渐获得凝练、升华的，本文的顺利完成更是离不开您逐字逐句悉心细致地审阅和修改。每次当我迷惘困惑时，您春风化雨般的教诲和鼓励总是能令我重拾信心与勇气。有幸得您培养的六年博士生涯无疑是我最

幸福的时光，您“攻坚莫畏难”的钻研精神深深地感染和影响着我，将使我受益终生。目前的我虽然仍有许多不足，但我会秉承您的教导，继续前进。感谢那个一路坚持读书的自己，是他将我从老家带出县城，来到省城，走到京城，一路生花。生命之美不在于一副完美的身躯，也不在乎生与死。若能凭借自己的知识，对国家、对社会做出一点儿贡献，所有的苦难也是值得了。

追风赶月莫停留，平芜尽处是春山。愿我们一路走来，想起的不再是生命的荆棘苦痛，记住的是陪伴着我们彻夜赶路的漫天星光。历经磨难，初心不改。出走半生，归来仍是少年。

行文至此，我要向所有其他曾在科研、生活上帮助过我的人们表达感谢。

衷心感谢袁良老师。生病时，有您温暖关怀；无助时，有您坚定支持；迷茫时，有您耐心指引。您正直善良、踏实做事的风格深深影响着我，亦师亦友。

衷心感谢贾海鹏老师。您是我在推免时认识的实验室里第一位老师。千里马常有，而伯乐不常有。自认不敢居千里马，然而您却于我有伯乐之恩。

衷心感谢在我博士初期给予指导、在我博士后期给予帮助的曹婷老师。难忘在您的指引下完成了我的第一篇学术论文，耐心带我从零开始探索博士之路。

感谢苏黎世联邦理工学院的李士刚老师、中科院计算所的商红慧老师、北京大学的陈一峯老师、微软亚洲研究院的高孝天老师在科研上提供的指导和帮助；感谢实验室的李希代、张广婷、赵艳洋和金琳老师多年来的支持和帮助；感谢计算所教育处的冯钢、周世佳老师在学业生活中对我的关照和指导；感谢实验室同学们日常生活中的帮助。

我要特别感谢我的室友陈恭巍。结交在相知，骨肉何必亲。作为一起生活最久的人，每晚与你的夜谈是我最快乐、最受益的时光。你的正直、博闻与宽容，使我也逐渐成长起来。与你相遇相知，是我的幸运。

我要特别感谢我的室友黄若然。有友如斯，夫复何求。作为一同约饭最多的人，你的乐观、豁达与幽默，给我无尽温暖的支持与力量，丰富了我的博士时光。

我要特别感谢我的同窗陈暾。作为一道学习最长的人，你的善良、真诚与低调，使我曾经一度孤独迷茫的日子重新感受到了温暖与信任。

最后，非常感谢我的父亲李修湖、母亲徐立华和姐姐李洁。感谢你们二十多年来对我的养育、教育和陪伴，毫无保留地一直为我默默付出。我的每一次进步，带给你们的喜悦和骄傲甚至远胜于我自己。谨以此文献给深爱着我的你们。

## 作者简历及攻读学位期间发表的学术论文与研究成果

### 基本情况

李琨，男，汉族，党员，山东青岛人。中国科学院计算技术研究所计算机系统结构专业博士研究生。

### 教育背景

2012年9月--2016年6月，就读于山东大学计算机科学与技术学院，计算机菁英班专业，获工学学士学位；

2016年9月--2022年6月，推免至中国科学院计算技术研究所直接攻读博士学位（直博），计算机系统结构专业，高性能计算方向。

### 攻读博士学位期间发表的学术论文

#### 第一作者论文列表

- [1] [IPDPS'22, CCF-B] **Kun Li**, Liang Yuan, Yunquan Zhang, Yue Yue. An Efficient Vectorization Scheme for Stencil Computation. Proceedings of the International Parallel and Distributed Processing Symposium, 2022.
- [2] [IEEE TPDS, CCF-A] **Kun Li**, Liang Yuan, Yunquan Zhang, and Gongwei Chen. An Accurate and Efficient Large-scale Regression Method through Best Friend Clustering. IEEE Transactions on Parallel and Distributed Systems, 2022.
- [3] [SC'21, CCF-A] **Kun Li**, Liang Yuan, Yunquan Zhang, and Yue Yue. Reducing Redundancy in Data Organization and Arithmetic Calculation for Stencil Computations. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2021.
- [4] [SC'19, CCF-A] **Kun Li**, Honghui Shang, Yunquan Zhang, Shigang Li, Baodong Wu, Dong Wang, Libo Zhang, Fang Li, Dexun Chen, and Zhiqiang Wei. Open-KMC: a KMC design for hundred-billion-atom simulation using millions of cores on Sunway Taihulight. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis, 2019.

- [5] [TJSC, CCF-C] **Kun Li**, Shigang Li, Shan Huang, Yifeng Chen, and Yunquan Zhang. FastNBL: fast neighbor lists establishment for molecular dynamics simulation based on bitwise operations. *The Journal of Supercomputing*, 2019.
- [6] [ISPA'19, CCF-C] **Kun Li**, Shigang Li, Bei Wang, Yifeng Chen, and Yunquan Zhang. swMD: Performance Optimizations for Molecular Dynamics Simulation on Sunway Taihulight. *International Conference on Parallel & Distributed Processing with Applications*, 2019.
- [7] [JCST'17, CCF-B 中文] 李琨, 贾海鹏, 曹婷, 张云泉. 大规模集群上多维 FFT 算法的实现与优化研究. *计算机科学与探索*, 2017.
- [8] [HPCChina'16] 李琨, 李焱, 曹婷, 贾海鹏, 张云泉. 基于 MPI 的 CPU+GPU 异构集群上三维 FFT 的高效实现. *中国计算机学会全国高性能计算学术年会*, 2016.

## 合作论文列表

- [1] [SC'21, CCF-A] Liang Yuan, Hang Cao, Yunquan Zhang, **Kun Li**, Pengqi Lu, and Yue Yue. Temporal Vectorization for Stencils. *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2021.
- [2] [CS'19, CCF-B 中文] 王栋, 商红慧, 张云泉, 李琨, 贺新福, 贾丽霞. 原子动力学蒙特卡洛程序 MISA-KMC 在反应堆压力容器钢辐照损伤研究中的应用. *计算机科学*, 2019.
- [3] [ICPP'18, CCF-B] Junmin Xiao, Shigang Li, Baodong Wu, He Zhang, **Kun Li**, Erlin Yao, Yunquan Zhang, and Guangming Tan. Communication-Avoiding for Dynamical Core of Atmospheric General Circulation Model. *Proceedings of the 47th International Conference on Parallel Processing*, 2018.
- [4] [To be appeared] Hang Cao, Liang Yuan, He Zhang, Yunquan Zhang, Baodong Wu, **Kun Li**, Shigang Li, Yongjun Xu, Minghua Zhang, Pengqi Lu, and Junmin Xiao. AGCM-3DLF: Accelerating Atmospheric General Circulation Model via 3D Parallelization and Leap-Format.

### 攻读博士学位期间已获得的专利

- [1] 李士刚, 李琨, 陈一峯, 张云泉。一种分子动力学模拟中邻接表快速建立方法和系统。授权专利号: CN109032667B。
- [2] 李士刚, 吴保东, 李琨, 张云泉。一种基于递归的并行快速傅里叶变换通信优化方法和系统。授权专利号: CN109840306B。

### 攻读博士学位期间参加的科研项目

- [1] 国家重点研发计划, 数值核反应堆 E 级计算性能优化关键技术研究。
- [2] 国家重点研发计划, 百万处理器核可扩展并行共性算法与性能优化关键技术研究与开发。
- [3] 国家重点研发计划, E 级系统存储层次计算模型研究。
- [4] 面上项目, 一种新型 Stencil 并行算法研究与优化实现。
- [5] 北京市自然科学基金, 面向深度学习的 GPU 虚拟化关键方法与技术研究。

### 攻读博士学位期间的主要获奖情况

- [1] 中科院计算所所长特别奖 (2022)
- [2] 国家奖学金博士生奖 (2021)
- [3] 中国科学院大学必和必拓联合奖学金 (2021)
- [4] 中科院计算所曙光博士生奖学金 (2020)
- [5] 中国科学院大学一等博士生奖学金 (2020)
- [6] 中国科学院大学国际学术会议奖学金 (2019)
- [7] 中科院计算所国重一等博士生奖学金 (2019, 2020, 2021)
- [8] 中国科学院大学优秀大学生奖学金 (2017)
- [9] 微软亚洲研究院“明日之星”称号 (2022)
- [10] 中国科学院“优秀共青团员”称号 (2020)
- [11] 中国科学院大学“三好学生”称号 (2016, 2017)
- [12] 中国科学院大学“优秀学生干部”称号 (2016, 2017)
- [13] 中国科学院大学“优秀共产党员”称号 (2016)
- [14] 中科院计算所“优秀志愿者标兵”称号 (2017, 2018)

