上海超级计算中心
Shanghai Supercomputer Center

高性能计算
发展与应用

DEVELOPMENT & APPLICATION
OF HIGH PERFORMANCE COMPUTING

[总第 61 期]

4

2017 年

目 录

深度学习发展现状及算法实现

郭培卿 上海超级计算中心 上海 201203

摘要：

随着数据规模不断增大，深度学习在人工智能领域扮演着越来越重要的角色。本文从深度学习基本概念出发，介绍了近年来深度学习领域发展现状，展望深度学习的未来方向，以及算法最终实现方案。

概述

传统学习算法中，数据规模的增加往往会导致计算性能的下降。深度学习（Deep Learning）又称为深度神经网络，是指使用包含复杂结构，或由多重非线性变换构成的多个处理层对数据进行高层抽象的算法，深度学习算法的性能随着数据规模增加而增加，通过深度学习进行特征表示已经成为数据挖掘社区迅速崛起的方法之一。

作为一种具有多个隐藏层、可以实现多层特征学习和表示的深层学习结构，深度学习中每一层神经元均基于前一层神经元所学习到的特征表示再进行学习，并逐层形成更加抽象的高层属性特征表示，从而获得数据特征的分布规律。深度学习以受限玻尔兹曼机（Restricted Boltzmann Machine，RBM）模块为基础，构建多层深度可信网络（Deep Belief Networks，DBN）的学习模型，对各个层次进行逐层进行无监督预训练并完成初始化，一个典型的DBN可以看成是由多个RBM堆叠组成。为了实现更好的训练和学习效果，DBN的学习过程包括由低层到高层、逐层独立的预训练以及误差回传调优，其基本工作流程包括数据采集、聚合管理、模型开发、部署评分、结果更新等各阶段。

深度学习算法有诸多分类体系，如按学习方式、主动性、训练启动时间等标准分类，较为常见的则是依据数据是否获得特定标签可分为有监督学习、无监督学习以及半监督学习。

有监督学习（supervised learning）

有监督学习通过已有训练样本（即已知数据及其对应输出），去训练得到一个最优模型，这个模型属于某个函数的集合，最优表示在某个评价准则下最佳，再利用这个最优模型将所有的输入映射为相应的输出。有监督学习过程可以类比为：有一些问题和它们的答案，学习这些已知答案的问题，从而具备经验（学习成果）的过程，当遇到新的未知答案的问题时，可以依据学习成果，得出这个新问题的答案。

典型的有监督学习包括回归分析以及分类：

1.回归分析（Regression Analysis）

回归分析中，给定一个函数的一些坐标点，通过回归分析算法来估计原函数模型，求出一个最符合已知数据集的函数解析式，然后用来预估其它未知输出的数据。输入一个自变量（特征向量），它会根据这个模型解析式输出一个因变量（标签），并且标签值范围连续。

2.分类（Classification）

输入有特征（feature）和标签（label）的训练数据，找到特征和标签间的关系（mapping），这一过程的算法实现即为分类（classification）。经过学习后，当有特征无标签的未知数据输入时，可以通过已有关系得到未知数据的标签。深度学习能够将网络中某一层的输出当作数据的另一种表达，从而将其认为是经过网络学习到的特征，基于该特征可以进一步利用大规模数据，使算法得以充分训练和验证，并且准确性随着训练数据量的增加逐渐提高。

无监督学习（unsupervised learning）

与监督学习的不同，无监督学习事先没有任何训练样本，直接对数据进行建模，所有数据只有特征向量没有标签，相似类型的数据聚集在一起，将没有标签的数据分类组合的过程即为聚类（Clustering）。无监督学习可以类比为：有一些问题，但是答案未知，无监督学习将其自动分组，每组的问题具有相似的性质和内容结构，当有新问题

出现时，可以实现自动聚类。

无监督学习本身的特点决定了聚类过程难以获得如分类一样近乎完美的结果，但由于在实际应用中，标签的获取常常依赖相当程度的人工介入，而无监督学习则不存在人工计算量的问题，因此出现了第三类学习方法——半监督学习。

半监督学习（semi-supervised learning）

有监督学习和无监督学习的中间带就是半监督学习。对于半监督学习，其训练数据的一部分有标签，另一部分则没有，并且没有标签数据的数量远大于有标签数据的数量。隐藏在半监督学习下的基本规律在于：数据的分布不是完全随机的，通过一些有标签数据的局部特征，以及更多的无标签数据的整体分布，就可以得到可以接受的分类结果。

半监督学习的出现，代表了基本认知规律从有监督学习开始、发展到半监督学习、最后再到无监督学习的发展方向。

发展现状

自2006年Hinton等人提出深度学习的概念以来，几乎每年都有具有影响力的标志性事件发生：2010年，美国国防部DARPA计划首次资助深度学习项目；2011年，微软研究院和谷歌的语言识别研究人员先后采用DNN技术降低语音识别错误率20%-30%，是该领域10年来最大突破；2012年，Hinton将ImageNet图片分类问题的Top5错误率由26%降低至15%，同年Andrew Ng与Jeff Dean搭建Google Brain项目，用包含16000个CPU核的并行结算平台训练超过10亿个神经元的深度网络，在图像识别领域取得突破性进展；2013年，Hinton创立的DNN Research公司被Google收购，Yann LeCun加盟Facebook人工智能实验室；2014年，谷歌将语言识别的精准度从2012年的84%提升到98%，移动端Android系统的语言识别正确率提高了25%，同时，谷歌的人脸识别系统FaceNet在LFW上达到99.63%的准确率；2015年，微软采用深度神经网络的残差学习方法将ImageNet的分类错误率降低至3.57%，已低于同类试验中人眼识别的错误率5.1%，其采用的神经网络已达到152层；2016年，DeepMind使用由1920个CPU集群和280个GPU支持的深度学习围棋软件AlphaGo战胜人类围棋冠军李世石；2017年，升级完善后的AlphaGo三局完胜柯洁，下半年AlphaGo Zero、Alpha Zero相继问世，不断挑战人类对于棋类项目的认知极限。

深度学习近年来获得长足进步的因素主要有三个方面：

（1）海量数据。学习一个有效的表示需要大量训练数据，根据2013年的数据[1]，Facebook每天收到超过3.5亿张图片，沃尔玛每小时产生2.5PB的用户数据，YouTube每分钟有300小时视频被上传，云服务商和相关企业拥有海量数据来训练算法。

（2）充足的计算资源。摩尔定律依然有效，半导体工业和计算机架构的进步提供了充足的计算能力，同时获取计算资源途径越来越多，针对深度学习芯片的不断问世，计算成本不断下降，使得在合理的时间内训练算法成为可能。

（3）算法进步。算法技术的进化极大地提高了准确性并拓宽了深度学习的应用范围，早期的深度学习应用打开了算法发展的大门，激发了许多深度学习框架的发展、开源和共享，使得众多研究者和从业者能够更为便捷地使用深度学习网络。

神经网络复兴的主要的三个发起人Lecun、Bengio以及Hinton，2015年在Nature上联合发表深度学习综述《Deep Learning》[2]，总结了深度学习的三大发展方向：

（1）无监督学习

无监督学习在深度学习复兴最初几年具有重要作用，主要利用无监督学习进行预训练，以得到一个较好的初始值，随后再使用有监督训练进行微调。但是随着计算能力的发展，人们发现只要在数据集足够大的情况下使用纯有监督学习也能得到较好性能，因此近几年无监督学习发展速度有所放缓，鉴于人类和动物的学习在很大程度上都是无监督学习，无监督学习的特点更接近人工智能的本质。

（2）深度强化学习

深度强化学习的主要思想是将深度学习与强化学习相结合，是一种从感知到动作的端到端学习。简而言之，和人类一样，输入感知信息（比如视觉），通过深度神经网络直接输出动作，中间没有任何人工特征的工作。深度增强学习具备使机器人实现真正完全自主学习一种甚至多种技能的潜力。

（3）自然语言理解

使用深度学习技术的各种应用比如神经机器翻译、问答系统、文摘生成等均取得了不错的效果，效果提升主要归功于注意力机制和循环神经网络相结合的强大能力，自然语言理解是深度学习在未来几年有所作为的领域之一。

算法实现

一项深度学习工程的搭建，可分为训练（Training）和推断（Inference）两个环节。训练环节通常需要通过大量的数据输入，或采取增强学习等非监督学习方法，训练出一个复杂的深度神经网络模型，由于涉及海量的训练数据和复杂的深度神

经网络结构，训练需要的计算规模非常庞大。推断环节则利用训练好的模型，使用新的数据去获得结论，虽然计算量比训练环节少，但推断环节仍然涉及大量的矩阵运算。

由于深度学习与传统计算的模式和特征有着本质区别，前者无须执行复杂逻辑指令，同时需对海量数据并行处理，传统架构的处理器无法有效支撑深度学习大规模并行计算的需求，深度学习面临着前所未有的计算能力的挑战。

传统架构芯片

CPU

传统计算架构一般由中央运算器（执行指令计算）、中央控制器（让指令有序执行）、内存（存储指令）、输入（输入编程指令）和输出（输出结果）五个部分构成，其中中央运算器和中央控制器集成一块芯片上构成了通用中央处理器（CPU）。

在CPU内部，仅单独的逻辑运算单元（ALU模块）用以完成指令数据计算，其他模块都是为了保证指令的有序执行，这种通用性结构适用于传统的编程计算模式，然而面对并不需要太多的程序指令，却需要海量数据运算的深度学习，这种结构就显得比较笨拙，因此，需要更适应深度学习算法的新的底层硬件来加速计算过程，目前主要实现方式包括GPU、FPGA以及ASIC专用定制芯片。

GPU

GPU具备数量众多的计算单元和超长流水线，具备强大的并行计算能力与浮点计算能力，其海量数据并行运算的能力与深度学习的需求不谋而合，因此被最先引入深度学习，可以大幅加速深度学习模型的训练速度，相比CPU，GPU能够提供更快的处理速度、更少的服务器投入和更低的功耗。

随着GPU可编程性不断增强，变成环境的不断成熟，GPU通用计算编程的复杂性不断降低，GPU已逐渐演化为一个新型可编程高性能并行计算资源，通过CPU、GPU组成异构平台，将算法程序中任何密集计算所对应的代码移植到GPU上处理，其余部分代码连同I/O、磁盘网络访问等操作依靠CPU辅助进行，从而实现通用GPU（General Purpose GPU，GPGPU）计算。目前主要通用计算标准架构有NVIDIA的CUDA（Compute Unified Device Architecture）、APPLE、AMD、IBM、INTEL、NVIDIA共同支持的OpenCL（Open Computing Language）、微软的Direct Compute等。

GPU 作为图像处理器，设计初衷是为了应对图像处理中需要大规模并行计算。因此，其在应用于深度学习算法时，有三个方面的局限性：

第一，推断环节中无法充分发挥并行计算优势。GPU在深度学习算法训练环节非常高效，但在推断时一次只能处理一个输入样本，无法发挥其并行度的优势。

第二，硬件结构固定不具备可编程性。深度学习算法还未完全稳定，如果深度学习算法发生变化，GPU无法灵活配置硬件结构，以适应新的算法需要。

第三，运行深度学习算法能效远低于FPGA。研究[3]表明，要在深度学习算法中实现同样性能，GPU所需功耗要远高于于FPGA。

FPGA

FPGA（Field Programmable Gate Array）是一种集成大量基本门电路及存储器的芯片，可通过硬件写入FPGA配置文件来来定义这些门电路及存储器间的连线，从而实现特定功能。由于硬件写入的内容是可配置的，通过配置特定的文件可将FPGA转变为不同的处理器，因此FPGA可灵活支持各类深度学习计算任务，此外FPGA具备低延迟的特点，适合在深度学习的推断环节支撑海量用户实时计算请求。

目前的FPGA市场由Xilinx和Altera主导，两者共同占有85%的市场份额，其中Altera在2015年被INTEL以167亿美元收购[4]，Xilinx则选择与IBM进行深度合作。

实际使用中，FPGA也存在一定的局限性，为了实现可重构特性，FPGA内部有大量极细粒度的基本单元，但是每个单元的计算能力（主要依靠LUT查找表）都远远低于CPU和GPU中的ALU模块，同时，FPGA处理速度和功耗与专用定制芯片存在一定差距，在量产规模超过一定水平后，成本优势也将逐渐变为劣势。

ASIC专用处理芯片

ASIC（Application Specific Integrated Circuit）则是不可配置的高度定制专用芯片，需要大量的研发投入，一旦流片后，芯片功能无任何更改余地，也就意味着ASIC具有较大的市场风险。但从另一方面看，ASIC作为专用芯片的性能要高于FPGA，如果实现高出货量，其单颗成本可做到远低于FPGA。ASIC定制芯片具备以下特点，以适应人工智能算法的发展需要：

首先，定制芯片的性能提升非常明显。性能提升意味着可以把深度学习逐步推广到实时理解和识别各种图像、语音和文本信息，如果进一步实现实时训练，将可以不间断通过观察人类行为而不断提升人工智能。

其次，下游设备的高需求量有助于摊薄定制

芯片投入的成本。人工智能的市场空间将不仅仅局限于计算机、手机等传统计算平台，从无人驾驶汽车、无人机再到智能家居的各类家电，远大于智能手机体量的设备需要引入感知交互能力，出于对实时性的要求以及训练数据隐私等考虑，这些能力不可能完全依赖云端，必须要有本地的软硬件基础平台支撑。

第三，通过算法切入人工智能领域的公司希望通过芯片化、产品化来盈利。通过算法切入人工智能领域的公司，包括采用语音识别、图像识别、ADAS（高级驾驶辅助系统）等算法，能够提供高频次、基础性的功能服务，然而，仅仅通过算法来实现商业盈利往往难度很大，以英特尔153亿美元收购Mobileye[5]为代表，通过将深度学习核心算法芯片化、产品化，不但有利于充分发挥算法性能，也有望为商业盈利铺平道路。

人工智能专用芯片的涌现，表明了从芯片层面开启的新一轮计算模式变革拉开帷幕，也意味着人工智能产业进入了新的发展阶段。

TPU芯片（谷歌）

谷歌2016年发布与Open Power Foundation合作开发的TPU（Tensor Processing Unit），该芯片由Open Power Foundation提供芯片设计，匹配谷歌TensorFlow框架。

每个TPU封装了矩阵乘法单元（MUX，包含65536个8位乘法和加法单元）、统一缓冲（UB，24MB容量SRAM作为寄存器）、激活单元（AU，硬件连接的激活函数）。TPU以复杂指令集CISC为基础，为了控制MUX、UB和AU进行计算，谷歌定义了十几个专门为神经网络推理而设计的高级指令，从而实现针对各种神经网络模型的编程，同时谷歌还提供编译器和软件栈，可将来自TensorFlow的API调用，直接转化成TPU指令。

与传统架构CPU或GPU相比，单用途的TPU就是一个单线程芯片，不需要考虑缓存、分支预测、多道处理等问题，同时TPU的控制单元更小，更容易设计，给片上存储器和运算单元留下了更大的空间，并且良品率高，有利于降低成本。此外，由于CPU和GPU需要考虑各种任务上的性能优化，因此会有越来越复杂的机制，带来的副作用就是这些处理器的行为非常难以预测，而在TPU计算时，则可以能轻易预测运行一个神经网络需要多长时间，从而让芯片以吞吐量接近峰值的状态运行，同时严格控制延迟。

与同期CPU+GPU相比，经过专门针对机器学习进行裁减后，TPU能够提供15-30倍的性能提升，以及30-80倍的能效提升（每瓦计算能力）[6]。在谷歌搜索、街景、照片、翻译等谷歌服务，在AlphaGo战胜李世石、柯洁的围棋系列赛中，都用到了TPU来加速背后的神经网络计算。

TESLA P100芯片（NVIDIA）

2016年4月，NVIDIA发布支持深度学习的新型芯片Tesla P100。这是NVIDIA首次设计一款专门用于深度学习领域的计算芯片。

每块Tesla P100芯片包含150亿个晶体管，芯片面积为600平方毫米，双精度运算速度5.3万亿次，单精度运算速度10.6万亿次，半精度运算速度21.2万亿次。同时发布的还有搭载了八个P100芯片、用于深度学习的计算机DGX-1。

Knights Mill（INTEL）

2016年8月，INTEL在IDF上推出专为机器深度学习设计的芯片——代号Knights Mill 的XeonPhi家族新成员，计划于2017年第四季度正式上市。

Knights Mill被认为将用来填补目前Knights Landing处理器（用于HPC的Xeon Phi系列加速芯片）和未来基于Nervana的产品之间的空白，因此Knights Mill将从Knights Landing中继承大部分设计。为了获得更精确的数学计算性能以适应神经网络计算，Knights Mill用一个较小的双精度端口和四个向量神经网络指令（VNNI）端口，取代了Knights Landing的矢量处理单元（VPU）上的两个大的双精度/单精度浮点（64位/ 32位）端口，与Knights Landing相比，Knights Mill将提供一半的双精度浮点性能，两倍的单精度浮点性能，根据计划，Knights Mill将提供4倍于深度学习应用的性能，以1.5GHz处理器频率预测，一块Knights Mill芯片可以提供超过27万亿次的计算性能。

Knights Mill代表了INTEL第一个专门针对机器学习训练环节深层神经网络的Xeon Phi产品。在推断环节，INTEL已推出了基于Altera的FPGA产品，此外，INTEL也在基于收购的Nervana开发新一代用于深度学习训练的产品。

TrueNorth芯片（IBM）

True North是IBM参与DARPA(美国国防部先进研究项目局)研究项目SyNapse（Systems of Neuromorphic Adaptive Plastic Scalable Electronics，自适应可塑可伸缩电子神经系统）的最新成果，其终极目标是开发出打破冯诺依曼体系的硬件。

TrueNorth芯片集成有54亿个晶体管，构成网络具有100万个数字神经元规模，这些神经元由数量庞大的模拟神经突触相连结。TrueNorth芯片使用IBM神经突触系统（IBM Neuromorphic System），采用类脑的神经网络设计，在执行图像识别与综合感官处理等复杂的认知任务时，效率远远高于传统芯片。

2016年3月，美国劳伦斯利佛莫尔国家实验室（

LLNL）宣布购买IBM研究院开发的基于TrueNorth神经突触芯片的同类首创类脑超级计算平台，用于执行深度学习推理任务。这台价值100万美元的计算机集成了16个True North芯片处理器，该计算机的处理能力相当于1600万个神经元和40亿个神经突触，而能耗仅与平板电脑相当——16个TrueNorth芯片的功耗仅为2.5瓦，同时IBM还将给LLNL提供端到端生态系统，助其创建并编程能够模拟人脑感觉、行动和认知能力的高能效设备，该生态系统包括模拟器、编程语言、集成式编程环境、算法和应用库、固件、用于构建神经网络以支持深度学习的工具，及教学课程表和云平台等组件。

其他

国内已发布的专用芯片包括中星微电子的“星光智能一号”芯片NPU[7]、中科院计算技术研究所的“寒武纪1号”（英文代号Dian Nao，面向神经网络的原型处理器结构）、“寒武纪2号”（Da Dian Nao，面向大规模神经网络）以及“寒武纪3号”（Pu Dian Nao，面向多种机器学习算法）等芯片[8]。

发展路线

设计专用芯片的目的是从加速深度学习算法到从底层结构模拟人脑来更好实现人工智能，从目前发展趋势来看，人工智能芯片发展路线可以分为三个阶段。

第一阶段，基于FPGA 的半定制人工智能芯片。在芯片需求还未成规模、深度学习算法暂未稳定需要不断迭代改进的情况下，利用具备可重构特性的FPGA 芯片来实现半定制的人工智能芯片是最佳选择。

第二阶段，采用针对深度学习算法的全定制ASIC芯片，如谷歌TPU、中科院计算所寒武纪处理芯片。这类芯片是完全采用ASIC设计方法全定制，性能、功耗和面积等指标面向深度学习算法都做到了最优。

第三阶段：类脑计算芯片，如IBM的TrueNorth芯片。这类芯片的设计目的不再局限于仅仅加速深度学习算法，而是在芯片基本结构层面上开发出新的类脑计算机体系结构，这类芯片的研究离成为市场上可以大规模广泛使用的成熟技术还有很大的差距，甚至有很大的风险，但是长期来看类脑芯片有可能会带来计算体系的革命。

市场生态

有观点认为，目前深度学习芯片的需求可归纳为三个层次[9]：一是面向各大企业及实验室研发阶段的训练（Training）；二是面向数据中心的云端推断（Inference on Cloud），主流人工智能应用需要通过云端提供服务；三是面向智能手机、智能安防摄像头、机器人、无人机、自动驾驶、VR等设备的终端推断（Inference on Device），需要高度定制化、低功耗的芯片产品。

训练层

在训练层，目前GPU已成为实际通用标准，在GPU加速市场，又呈现出NVIDIA一家独大的局面，面对深度学习训练这一市场，谷歌、AMD等巨头纷纷发起挑战。在训练环节，各大厂家竞争的核心并不在于芯片本身，而是基于芯片加速背后的生态圈，取决于能否提供足够友好、易用的工具环境，让开发者迅速获取深度学习加速算力，从而降低深度学习模型研发和训练加速的成本和周期。

为了确保其市场份额，NVIDIA在以下三个方面增加了投入。一是加速产品研发与更新，一年内接连推出两代Tesla系列的专用芯片P100和V100，主打工业级超大规模深度网络加速；二是加强人工智能软件堆栈体系的生态培育，即提供易用、完善的GPU深度学习平台，不断完善CUDA、CuDNN等套件以及深度学习框架、深度学习类库来保持对NVIDIA系列GPU加速方案的粘性；三是推出NVIDIA GPU Cloud云计算平台，除了提供GPU云加速服务外，以NVDocker方式提供全面集成和优化的深度学习框架容器库，以其便利性进一步吸引研究开发者使用。

谷歌2017年5月发布TPU2.0，在仅用于推断、不可用于训练的第一代TPU的基础上，除了增加推断功能以外，还高效支持训练环节的深度网络加速[10]。在谷歌深度学习翻译模型计算中，8个TPU 2.0仅需6个小时，就能实现相当于在32块顶级GPU上并行训练一整天的训练效果。同时，谷歌并未急于推进TPU芯片的商业化，关于TPU芯片谷歌的整体规划，是基于目前在深度学习框架领域排名第一的TensorFlow，结合谷歌云服务推出TensorFlow Cloud，通过TensorFlow加TPU云加速的模式为开发者提供服务。如果一旦谷歌将来能为开发者提供相比购买GPU更低成本的TPU云加速服务，借助TensorFlow生态毫无疑问会对NVIDIA构成重大威胁。另一方面，让众多已经熟悉GPU加速的研究开发者转到TPU云计算平台需要大量的生态系统培育工作，并且TPU作为一种ASIC芯片方案，如果将来TPU云服务无法获得巨大的市场份额从而降低单颗TPU的成本，或深度学习主流框架发生重大变化，TPU芯片的价格优势将不复存在甚至失去应用价值。

除了谷歌以外，作为唯一同时拥有GPU和x86硅芯片技术的公司，AMD也在奋起直追。2016年

12月，AMD发布三款基于Radeon Instinct的深度学习加速器MI6、MI8、MI25，用于深度学习推理和训练[11]。同时发布的还有用于GPU加速器的免费开源库MIOpen，用于实施高性能的机器智能，为AMD ROCm软件提供全新优化的深度学习框架，为机器智能持续进化提供基础。MIOpen GPU加速库可以为卷积、池化、激活函数、归一化和张量格式等标准例程提供GPU优化，ROCm平台则可用于加速Caffe、Torch 7和Tensorflow等主流深度学习框架。

云端推断层

相比训练市场中NVIDIA的一家独大，云端推断芯片领域由于当前的需求并未进入真正的高速爆发期，多数人工智能应用尚未在消费级市场形成巨大需求，在爆发点来临之前，各大厂商纷纷布局自己的云端FPGA应用生态。

由于一般训练得到的深度神经网络模型复杂度很高，其推断（Inference）过程仍是计算密集型和存储密集型，使其难以被部署到资源有限的终端用户设备上。另一方面，即使在相对简单的算法中，单次推断计算量远远小于训练，但如果有大量请求同时使用这项，计算量总和足以对服务器带来巨大压力，由于海量的推断请求属于计算密集型任务，CPU在推断环节再次成为瓶颈，FPGA体系结构决定了面向与海量用户高并发的云端推断，FPGA芯片相比GPU具备更低计算延迟优势。因此，云端推断（Inference On Cloud）就变得非常必要。

微软2015年开始通过Catapult项目在数据中心实验CPU+FPGA方案；亚马逊AWS于2016年推出基于FPGA的云服务器 EC2 F1；百度与FPGA巨头Xilinx（赛思灵）合作，在百度云服务器中部署KintexFPGA，用于深度学习推断。此外，INTEL通过多桩并购进入这一市场：2015年INTEL以167亿美元收购 FPGA界排名第二的Altera，整合Altera多年FPGA技术以及INTEL自身的生产线，推出CPU+FPGA 异构计算产品主攻深度学习的云端推断市场；2016年，收购拥有为深度学习优化的硬件和软件堆栈的初创公司Nervana，补全了深度学习领域的软件服务能力；2017年以153亿美元收购ADAS服务商Mobileye，将人工智能芯片的触角延伸到了设备端市场。

终端层

在终端层，深度学习应用远未成熟，各大服务商基本遵循由软件应用拓展到软件+芯片解决方案的发展路线，由此形成了丰富的芯片产品方案，意图形成端到端的综合解决方案体系，实现各层次资源联动。在不能单纯依赖云端推断的设备的应用场景中，要求终端本身需要具备足够的推断计算能力，而当前ARM等架构芯片的计算能力并不能满足终端设备本地深度神经网络的推断需求，需要有新的芯片架构来应对逐渐增加的人工智能应用场景。例如在智能手机领域，华为在麒麟970芯片中搭载寒武纪IP，为Mate10带来较强的深度学习本地端推断能力，高通在新一代芯片中加入骁龙神经处理引擎，用于本地端推断，ARM推出了针对深度学习优化的DynamIQ技术；在ADAS（高级辅助驾驶系统）领域，海量由激光雷达、摄像头等传感器采集的实时数据需要处理，被INTEL以153亿美元收购的Mobileye、被高通以470亿美元收购的NXP、以及NVIDIA的Drive PX2方案都是ADAS的有力竞争者；其他有需求的领域包括计算机视觉（Computer Vision，CV）、VR设备、语音交互设备以及机器人等人工智能解决方案。

结语

在深度学习领域，各大厂商纷纷涉足应用软件、开发平台、计算芯片、网络服务等领域，INTEL、谷歌、NVIDIA等科技巨头致力于打造满足上下游各方需求的生态圈，旨在人工智能时代爆发之时拥有充分的技术优势与话语权，推动产业进一步发展，而在另一方面，前沿性研究方兴未艾，也为希望进入深度学习领域的企业与研究机构提供了充分的可能性。深度学习的发展前景，受到基础理论、训练算法、数据样本、计算资源、应用场景、行业需求、资本市场等因素的共同影响。

参考资料：

[1] http://36kr.com/p/206304.html

[2] Y LeCun, Y Bengio, G Hinton, Deep Learning, Nature 521, 436 - 444, 2015

[3] http://blog.csdn.net/gavinlib/article/details/72458595

[4] http://news.163.com/15/1229/12/BC0KTIN200014SEH.html

[5] http://www.cs.com.cn/xwzx/hwxx/201708/t20170821_5432451.html

[6] http://36kr.com/p/5069563.html

[7] http://36kr.com/p/5048372.html

[8] http://news.ifeng.com/a/20171107/53020460_0.shtml
[9] http://news.zol.com.cn/654/6542666.html
[10] http://technews.cn/2017/05/18/google-io-2017-new-tpu/
[11] https://instinct.radeon.com/en/

要闻集锦

新“极光”有望成为美国首台E级超级计算机

据www.top500.org网站2017年10月4日消息报道，之前美国透露将在2021年完成首台百亿亿次（E级）超级计算机“极光（Aurora）”，近日又有了一些更加详细的信息。这一消息是美国能源部先进科学计算顾问委员会在一次会议上透露的，据悉新的“极光”超级计算机将于2020年底开始制造，从2021年第一季度开始交付，并于2022年全部完成。系统的软硬件研发将从明年开始，同期还将进行一项应用准备项目。

事实上，重新规划的“极光”超级计算机充分反映了美国能源部近期提出的要在2023年左右实现一台“全新架构”E级超级计算机的理念。“全新架构”指的是什么目前尚不得而知，但可以肯定的是，新“极光”肯定不会再使用当前典型的CPU-GPU混合架构。之前的“极光”拟使用Cray公司的“shasta”超级计算机配合Intel公司的第三代至强Phi处理器。但是，由于第三代至强Phi处理器需使用10nm工艺，而Intel转向10nm工艺的进度已然大幅落后，因此也拖后了第三代至强Phi的研发，造成了“极光”无法按时完成。

未来的E级超级计算机必须支持深度学习任务，这意味着其底层架构必须支持单精度（32位）、半精度（16位）和四分之一字（8位）运算。这一点与Intel公司处理器的发展方向是相符的。未来的至强Phi将支持64/32/16/8位浮点和整数指令，其中很多已经被包含在Intel最新的先进向量指令集（AVX-512）中了。

除此之外，“极光”还将使用多项未来有望成为主流的全新技术，比如Intel的硅光电技术和3D XPoint存储器等。去年，Intel已经发布了第一代硅光电产品，全球首款100G硅光电收发器。今年5月，Intel又展示了3D XPoint存储器模块，有望大幅增加E级系统的存储容量。而在互连方面，“极光”也有可能用上第三代Omni-Path互连技术。

（李 苏）

GTC-P在大规模节点基于OpenACC的移植优化研究

韦跃明[1] 王一超[1] WILLIAM Tang[2, 3] BEI Wang[2] 林新华[1, 4]

[1]上海交通大学高性能计算中心 上海 200240

[2]普林斯顿大学计算科学与工程研究所 美国

[3]普林斯顿大学等离子体物理实验室 美国

[4]东京工业大学 日本

摘要：

近年来，随着GPU等加速器不断发展，基于加速器的异构计算正逐渐成为高性能计算的主流。然而集群架构越来越复杂，同一应用，运行在不同体系架构上往往需要开发多个版本，这给代码的开发和维护都带来了很大的挑战。OpenACC是基于指令的并行编程模型，为应用在多种平台上（包括GPU，x86多核处理器）提供了可移植性。GTC-P是基于particle-in-cell（PIC）算法，模拟粒子和等离子通过托卡马克装置时运动的科学应用。由于其极佳的可扩展性，GTC-P 现已在Top500排名前10的6台超级计算机上进行了性能测试[1]，还入选了美国能源部下属的NERSC国家超算中心的基准测试集[2]。我们在原有的OpenMP版本GTC-P基础上，使用OpenACC移植和优化GTC-P，并在多平台、大规模节点上进行测试分析。通过移植和一系列优化工作，包括数据局部性优化，线程映射优化和CUDA局部代码优化等工作，我们在单节点上实现了4.2倍加速。我们仅仅用了300行左右的OpenACC指导语句，就实现了CUDA代码90%以上的性能。而在大规模节点测试中，我们在Titan上4096个计算节点进行了拓展性实验，并对实验结果展开了分析。

关键词：高性能计算，OpenACC，PIC

1. 引言

Gyrokinetic Toroidal Code at Princeton（GTC-P）是由普林斯顿大学打开，具有高度可扩展、可移植的particle-in-cell（PIC）代码。它主要通过求解5D的Vlasov-Poisson方程，模拟粒子和等离子在托克马克装置中的运动。GTC-P在千万亿级的超级计算机上表现出很高的运行效率。在当前TOP500的榜单上，GTC-P在TOP10中的6台进行了性能测试，包括Titan, Mira等等。

我们在原有的OpenMP版本GTC-P基础上，使用OpenACC移植和优化GTC-P，并在多平台、大规模节点上进行测试分析。通过移植和一系列优化工作，包括数据局部性优化，线程映射优化和CUDA局部代码优化等工作，我们在单节点上实现了4.2倍加速。我们仅仅用了300行左右的OpenACC指导语句，就实现了CUDA代码90%以上的性能。而在大规模节点测试中，我们在Titan上4096个计算节点进行了拓展性实验，并对实验结果展开了分析。

本研究主要贡献如下：首先，我们首次通过OpenACC完成GTC-P的移植和优化工作。通过数据局部性、线程映射等优化手段，OpenACC单节点实现了4.2倍加速。我们发现OpenACC的原子操作对性能影响很大，针对GPU和x86多核，我们分别提出了两种不同的优化手段减小原子操作影响。其次，据我们所知，这是首次在大规模节点上对OpenACC移植的实际应用进行测试分析。我们对算法进行了调整，通过重复计算减少GPU内存使用，从而使模拟的问题规模进一步增大。我们在Titan上超过4000个计算节点对OpenACC的性能进行了测试，实验结果显示，在大规模节点上，OpenACC实现了和CUDA基本一致的拓展性。

2. 相关工作

OpenACC自发布以来，引起了很多研究者的

关注，有许多研究围绕着OpenACC的性能进行了展开。Hoshino等研究了OpenACC对在两个基准测试和一个真实世界的CFD应用上的性能[3]。作者首先探索了矩阵乘法，7点stencil和UPACS应用程序的基线移植策略。然后，研究了OpenACC和CUDA中通用和特定的应用优化技术。评估表明，目前的OpenACC编译器大约达到了CUDA版本性能的50%，根据编译器之间的差异，最高可以达到CUDA性能的98%。但是，与使用共享内存优化的CUDA代码相比，片上内存的限制造成OpenACC和CUDA代码存在显著的性能差距。文中所用的CFD应用代码最初是用Fortran编写的，并通过MPI实现并行化。作者首先确定程序瓶颈，然后在CUDA和OpenACC中重写GPU执行的一部分应用程序。另外，由于UPACS的性能通常受内存吞吐量的限制，因此作者应用了一系列典型的访存优化，包括几个阻塞转换和循环合并。在整个优化研究过程中，作者还分别尝试优化CUDA和OpenACC每个版本的代码，然后公平权衡比较它们的适用性和效率。

Liang Deng等研究了内部代码CFD应用TH-CFD的ADI求解器在英特尔Ivy Bridge CPU服务器，K20 GPU和英特尔至强融核7110P上的并行化和优化[4]。考虑到ADI解算器的特点，他们探索了一系列有效的性能优化技术。结果显示，两个Ivy Bridge CPU的加速比为2.5，而Intel Xeon Phi协处理器的加速比较以前的优化版本提高了1.7倍。为了解释Ivy Bridge和Xeon Phi之间的性能差异，他们对硬件性能指标进行测量和分析。而基于OpenACC的最佳版本性能可以达到经过优化后的CUDA版本79%的性能。作者最后从性能和可编程的角度对多核和多核架构进行了系统的比较。研究结果表明，GPU在引入OpenACC编程模型之后，是一个移植成本较低，效率很高的并行架构，可以大幅提升应用程序的性能。

Christopher P. Stone等人通过OpenACC在超级计算机的多个NVIDIA Kepler GPU上加速SP-MZ基准测试[5]。在单节点上对预定义的A类和C类基准测试结果与之前研究报告中的结果一致。通过重构标准的托马斯算法，改版后的代码显现出强大的性能优势，多个矩阵系统可以在每个网格线上同时进行求解。此外作者还发现异步执行的内核在较小的区域情况下可以获取性能的提升。在A类网格情况下，在NVIDIA Kepler GPU上可以同时启动多个区域计算内核，从而提高整体吞吐量。此外，主机和设备之间数据传输的影响在单节点和多节点上都有体现。通过在GPU设备上直接设置区域边界条件，避免了主机和设备之间数据拷贝，减少了23%（对于A级网格）的通信开销。这对现实世界的CFD应用具有很大有影响，因为物理边界条件（在SP-MZ基准测试中并不是必需的）有事可能会更耗时。使用OpenACC可以很方便地将代码加载到GPU上，与使用CUDA这样的相对底层的语言方法相比，可以减轻复杂的边界条件计算。此外，作者还通过MPI-OpenACC基准评估了NVIDIA的GPUDirect通信。结果发现，当每个节点的计算量变小时，使用NVIDIA GPUDirect节点间通信（相对于Device-BC方法）提高了SP-MZ代码的可扩展性。

3. OpenACC

随着GPU和多核架构在高性能计算中的出现，编码者希望能够使用熟悉的高级编程模型进行编程，从而为各种计算体系结构提供高性能、可移植的程序。OpenACC作为一种编程模型在2011年出现，它使用高级编译器指令来实现代码中并行。

为了确保OpenACC可以移植到所有可用的计算架构中，OpenACC定义了一个加速计算的抽象模型。这种模式的目标是确保OpenACC不仅适用于特定的体系结构，而是适用于当前广泛的可用体系结构，同时还确保OpenACC可以在未来的设备上使用。

OpenACC主要将主机的数据和计算加载到加速器上执行。这些设备可能是相同的，或者是完全不同的体系结构，例如CPU主机和GPU加速器。这两个设备也可能有独立的内存空间或统一的内存。在这两个设备有不同内存的情况下，OpenACC编译器将分析代码并负责主机和设备内存之间的数据传输。

OpenACC的高级API基于指令或编译指令语句（分别应用在Fortran或C/C++中），这些指令用于注释要转换为在加速器或协处理器（例如GPU）上运行的代码段[8]。与OpenMP一样，这些指令用于指定适合并行化的代码块。理想情况下，不需要修改源代码情况下，只需在适当的区域内加上指导语句，编译器就可以识别顺序结构（如循环）中的可并行性，并自动将该段可并行代码转换为加速器特定语言。

4. 移植优化

本部分将首先介绍GTC-P程序的关键特性以及性能瓶颈，然后详细讲述我们所应用的优化方法。

4.1 热点分析

在我们移植工作开始之前，我们首先对GTC-P代码进行了热点分析。我们的基本思路是：先找到整个应用最耗时的部分，即计算密度最高部分的代码，然后将这部分代码通过OpenACC指导语句从CPU移植到GPU上进行计算。如图2所示，我们在Intel E5-2670型号CPU对OpenMP CPU版本的GTC-P代

码进行了热点分析，我们使用了最小的测试算例A进行了分析。从热点图中，我们可以发现主要的热点函数是charge和push，这两个函数运行时间占整个应用总运行时间85%以上。

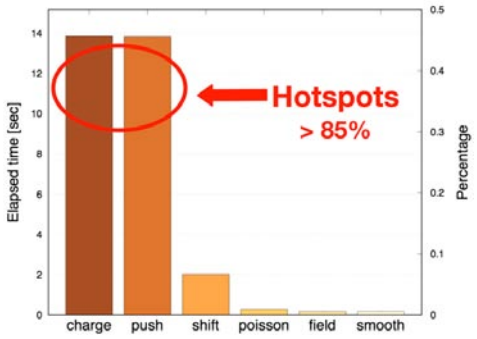


图2 GTC-P 在 x86 多核 CPU(Intel E5-2670)上测试算例 A 的热点函数分析结果

4.2 数据局部性优化

在我们完成了上面OpenACC版本GTC-P的基础版本移植工作之后，我们发现尽管已经将应用程序中计算密集程度最高的部分移到了加速器上，但是数据从主机复制到加速器并返回的过程将比计算本身更耗时。这是因为编译器很难确定程序何时以及是否需要数据，因此必须很谨慎地处理数据，以确保在需要的情况下复制数据。为了改善这一点，我们将利用应用程序的数据局部性进行优化。数据局部性意味着，在设备或主机内存中使用的数据应该尽量保持在本地，只要它将来还要被使用。即实现数据重用或者减少主机和设备存储器之间不必要的数据拷贝传输。

在实现了程序重要区域的并行化之后，我们需要向编译器提供关于并行区域使用的数据的局部性的信息。正如前面所述，编译器会对数据移动采取谨慎的态度，总是复制可能需要的数据，以确保程序可以产生正确的结果。对于开发者而言，可以明确知道什么数据什么时候是真正需要的。我们同时也知道如何在两个函数之间共享数据，编译器是很难确定这些信息的。因此，加速过程的下一步是向编译器提供有关数据局部性的附加信息，以最大限度地重用设备上的数据并最大限度地减少数据传输。这一步主要对主机和加速器内存分离的设备有效。

我们通过添加数据区域data region便于在多个并行区域之间共享数据。数据区域可以在同一个函数内的一个或多个并行区域外围添加，或者可以在程序调用时候添加，从而使数据在多个函数区域之间共享。

4.3 线程映射优化

OpenACC编程模型将目标体系结构抽象为三个并行级别的抽象模型：gangs，worker，vector。OpenACC编译器有默认方式将这种抽象并行模型映射到具体的体系架构上。但是由于编译器对程序和体系架构的信息了解有限，因此开发人员手动指定映射模式对于提升程序性能具有很重要的意义。在Nvidia的GPU上，gangs与CUDA线程块对应，而worker和vector和每个线程块内的线程相对应。在我们的代码中，我们主要通过设置gangs和vector大小来调整映射关系。

对于gangs和vetor的大小设置，不同的代码有不同的实际硬件资源利用情况，因此没有统一的最佳设置方式，我们需要手动不断调整他们的大小，找出最适合的设置参数。

对于多层循环，OpenACC编译器有时会做出一些优化，包括自动将独立的两层循环做并行分解。但是对于我们的程序，有些地方并不需要这样的自动并行。在push函数中，主循环中有一个自循环，其迭代次数为三。编译器分析出子循环是独立的，因此自动进行了并行展开，在内层循环自动进行并行计算。但实际上由于内层循环和外层循环相比很小，只有三，而外层循环可达到几十万以上。因此，对内层循环进行自动并行后大大降低了硬件资源的利用效率，导致程序运行速度大幅度降级。因此，我们通过手动添加知道语句 seq强制串行执行内部循环，减少线程间的硬件资源竞争，提升整体效率。

4.4 内存优化

和CPU相比，GPU显著优势就是计算能力强。GPU的缺点之一是显存大小和CPU内存大小相比小很多。因此，相同规模的测试算例在CPU上可以正常运行，在GPU上却因为显存不足而无法运行。为了提高OpenACC的代码在GPU上可以运行问题的规模，我们对CPU的代码进行了调整优化，减少在GPU上的内存开销。

在CPU的代码中，在charge中我们使用四点近似算法计算每个粒子的作用。我们使用辅助的数组，将四点近似算法中计算得到的值存储起来，在push中重新使用这些值，不需要重新再计算一遍。对于CPU这样内存比较大，而计算能力相对弱的体系架构而言，这样做的确能够提高程序的运行效率。但是对于GPU设备，由于显存本身就不大，再使用这些辅助数组记录这些计算得到的值会造成很大的内存浪费。由于GPU计算能力较强，我们完全没有必要将这些值预先存储起来，GPU完全可以重新计算这些值，

再使用它们。受到这点启发，我们对代码进行了修改，通过重新计算将GPU使用的内存减少了一半，从而大大提升了OpenACC版本代码可以模拟的问题规模。经过我们的优化，将GPU的使用降低了50%左右。

4.5 通过CUDA进一步优化

在我们代码经过OpenACC优化之后， chagre仍和CUDA代码有一定差距，其原因主要是CUDA可以使用较底层的共享内存而OpenACC无法通过知道语句直接使用它们。由于OpenACC的设计初衷就是能够在不同硬件之间移植的通用可移植编程语言，因此对于像GPU的专有特性共享内存，OpenACC并没有提供相关指导语句。虽然OpenACC不支持使用底层硬件资源共享内存，但是我们可以通过在OpenACC代码中添加CUDA代码，然后混合编译实现chagre内的一些关键操作通过共享内存实现。

我们插入CUDA代码，替代charge中之前通过OpenACC atomic操作避免数据冲突的方法。在charge函数中，属于同一个超级单元的所有点由相同的CUDA线程处理，并且两个相邻超级单元中的点由两个连续的CUDA线程处理。每个线程块都在共享内存上保存一个本地的网格副本，用于电荷充电，所有的操作都通过快速共享内存的原子操作完成。此外，可以通过在共享存储器中提供部分网格的两个副本来避免原子操作，一个用于具有偶数ID的线程进行电荷充电，而另一个用奇数ID进行电荷充电。共享内存中的电荷在充电结束之后被累加到全局内存中。通过插入CUDA代码使用GPU共享内存优化之后，charge函数性能得到了两倍以上性能提升 。

5. 结果讨论

5.1 单节点优化效果

在前面章节中，我们对OpenACC版本GTC-P针对GPU做了一系列的优化工作，包括数据局部性优化，线程映射优化以及插入CUDA代码混合编译优化方法。经过一系列的优化，只使用OpenACC指令情况下，在单节点上，对比CPU上OpenMP代码，我们实现了3倍加速。再进一步使用CUDA代码对charge函数做进一步优化之后，单节点上我们实现了4.2倍加速。和原代码相比，纯OpenACC版本代码只使用了216行OpenACC相关指令，对比完全用CUDA进行移植整个工作量大大减少了。具体单节点性能和原OpenMP性能对比如图3所示。其测试节点信息为两个Intel Xeon E5-2695V3 CPU和两块Nvidia Tesla K80 GPU加速器。

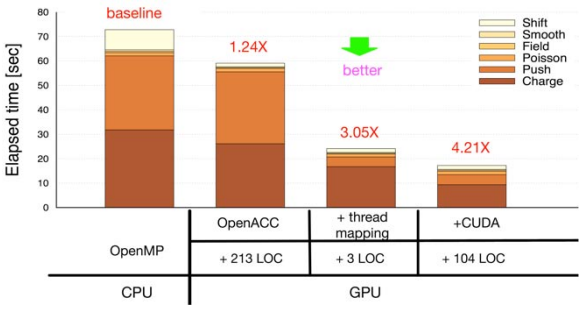


图3 经过一步一步优化之后，OpenACC 版本 GTC-P 性能

根据上图结果，我们发现经过数据局部性优化，内存优化等操作之后的OpenACC代码在单节点上实现了比OpenMP的CPU代码1.2倍加速，其加速效果并不明显。我们进一步通过设置OpenACC代码中gangs，vector等参数来修改线程映射设置。通过线程映射优化之后，修改了GPU中CUDA线程块以及线程块内线程的设置，对GPU硬件资源做了更合理的分配利用，从而得到了很好的优化效果。而经过CUDA代码对charge函数中的原子操作做了进一步优化，直接使用GPU的共享内存对原子操作进行优化，也实现了较大的性能提升。

总体来说，仅仅使用OpenACC对CPU程序进行移植加速，在GPU上可以实现很好的加速效果。而移植工作量对比传统相对底层语言，例如CUDA和OpenCL减少了不少。由于OpenACC支持和CUDA混合编译，对于像\textbf{chagre}这样只需要一小段CUDA代码就可以给程序性能带来显著提高的函数，使用CUDA加速是一个不错的选择。我们仅仅使用了104行CUDA代码，就将整个应用的加速比提升到4.2。

5.2 OpenACC可移植性

OpenACC和CUDA，OpenCL等相比具有较好的可移植性，同一套代码可以运行在不同的体系架构上。这和传统的针对底层硬件开发的专门语言相比具有很大的优势。以前，开发人员想在不同的体系架构上运行同一个应用，往往需要开发不同的版本代码，例如我们现在的GTC-P应用，分别有针对Intel x86多核的OpenMP版本代码和针对GPU的CUDA版本代码，这给代码的开发和维护带来了很大的难度。我们基于OpenMP版本的GTC-P开发的OpenACC代码，则解决了这个问题。由于OpenACC支持多体系架构，同一套代码可以运行在不同的体系架构上。我们针对本版本代码，分别在x86多核，x86+Nvidia GPU，OPENPower+GPU平台上对代码进行了测试。

可移植性测试结果如图4所示。从测试结果中我们可以看到，同一套代码，只是对编译选项进行了

简单的修改，就可以运行在三个具有完全不同体系架构的平台上，而且均达到了预期的效果。这说明使用OpenACC开发的代码具有很好的移植性，对于需要在不同体系架构运行的应用，使用OpenACC进行开发是一个很不错的选择。

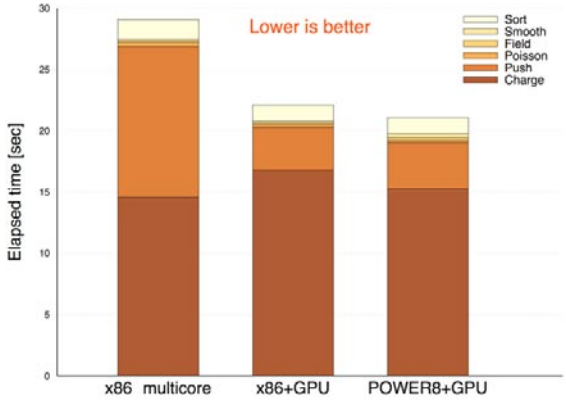


图4 OpenACC 版本 GTC-P 在各个平台的可移植性

5.3 拓展性实验

GTC-P模拟现实世界的等离子体运动，往往需要运行在大规模节点，进行大规模问题模拟。以上我们对OpenACC在单节点的优化效果和不同单节点平台上的可移植性做了测试和分析，接下来我们将对OpenACC版本的GTC-P在大规模节点的可拓展性进行测试和分析。我们将从强拓展性，弱拓展性两个方面进行测试和分析。

我们在上海交通大学高性能计算中心的超级计算机π上进行了强拓展性和弱拓展性测试。我们在32个节点，总计64块Nvidia Tesla K20M GPU上进行了强拓展性和弱拓展性测试。

1) 强拓展性 我们在π上使用了32个节点，

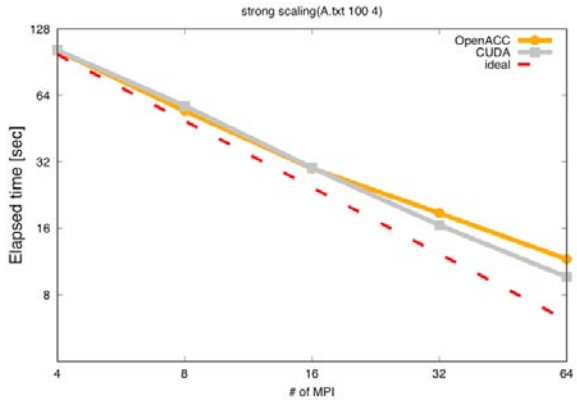


图5 OpenACC 和 CUDA 版本代码在多 GPU 节点强拓展性测试比较

64块Nvidia Tesla K20M进行了强拓展性测试。强拓展性测试中，我们保持模拟的问题规模大小不变，不断增加计算资源（即GPU节点数），然后观察加速效果。在理想情况下，运行时间应该和GPU节点数成反比。

从强拓展性测试结果来看，我们发现OpenACC版本的GTC-P具有很好的强拓展性，在测试实验规模下实验结果趋近于一条直线，和GPU节点数目成正比。和CUDA拓展性相比，OpenACC在节点数目在16以上，性能有所下降，比CUDA稍弱。但是考虑到OpenACC是比较高级的语言，在可以大大减轻开发工作量的同时保持了良好的可移植性，具有和CUDA底层语言相近的性能是非常难得的。

2) 弱拓展性 我们在π上使用了32个节点64块GPU进行了弱拓展性实验。我们分别使用了A，B，C，D四个不同测试算例，对应运行在1，4，16，64块GPU上。

我们在π上对OpenACC和CUDA两个版本的GTC-P做了弱拓展测试，其结果如图6所示。从结果图中我们可以看出，虽然每块GPU计算量没有改变，但是随着节点数目增加，整体的运行时间均有提升。这对于一般应用而言是正常的，因为程序必然有一部分不可加速部分，节点数目的上升并不会对这部分代码有影响。对比来看，我们的OpenACC代码和CUDA基本持平，这说明OpenACC移植到GTC-P应用在弱拓展性方面达到了和CUDA相同的水平。考虑到OpenACC是比较高级语言，CUDA是底层语言，OpenACC的表现是十分出色的。

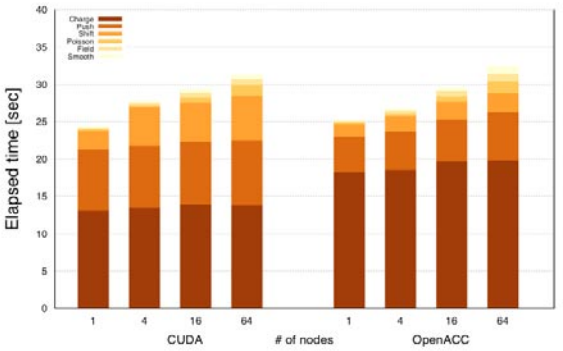


图6 OpenACC 和 CUDA 版本代码在多 GPU 节点弱拓展性测试比较

6. 结论及未来工作

GTC-P是基于PIC算法，模拟实际物理粒子碰撞的科学应用。我们在原有的OpenMP版本GTC-P基础上，使用OpenACC移植GTC-P并在多平台、大规模节点进行测试分析。

我们首先将GTC-P的主要热点函数移植到GPU上得到了最初的版本代码。经过数据局部性的优化、线程映射等优化后，OpenACC版本代码对比

OpenMP代码单节点实现了3倍加速。我们进一步使用了CUDA对charge函数核心部分进行了优化，然后再和OpenACC代码进行混合编译。插入的CUDA代码主要利用了CUDA的共享内存对原子操作进行优化。经过CUDA代码进一步优化之后，单节点OpenACC实现了4.2倍加速。

我们在单节点优化完成之后，对在多节点上进行了测试。我们在上海交通大学高性能计算中心π集群上进行了拓展性实验。我们在32各节点，64块Nvidia Tesla K20上进行了强拓展性实验和弱拓展性实验。我们同时对OpenACC和CUDA版本GTC-P进行了测试，并对性能进行了对比分析。经过测试，我们发现OpenACC的拓展性非常好，和CUDA拓展性基本一致。

综合我们移植，优化和测试结果来看，我们对OpenACC做出如下评价：OpenACC在设计的最开始，就是一个高级的，独立于平台的编程模型。因此，开发一套单一的代码，就可以在一系列设备上运行，并且获得良好的性能。然而，OpenACC的编程模型提供的简单性和可移植性是以性能损失为代价的。OpenACC抽象加速器模型的定义是基于各种加速器设备的共有属性，不能因为某些设备的体系结构具体化而降可移植性。在较底层的编程模型（如CUDA或OpenCL）中总有一些优化，这些优化无法在高层次上表现出来。例如，尽管OpenACC具有缓存指令，但在NVIDIA GPU上使用共享内存的一些用法通过CUDA更容易实现。对于任何主机或加速器而言也是如此：对于像OpenACC这样的高级指令编程语言，某些优化过于低级。开发人员需要确定在关键部分的代码，选择使用较底层的编程语言来获取较好的优化效果。综合来看，考虑到移植开发成本和实际性能，对于一般应用而言，如果不是追求硬件的极致性能，使用OpenACC是一个不错的选择。

参考文献：

[1] TANG W, WANG B, ETHIER S. Scienti c Discovery in Fusion Plasma Turbulence Sim- ulations at Extreme Scale[J]. Computing in Science Engineering,2014,16(5):44-52.

[2] ETHIER S, CHANG C S, KU S H, et al. NERSC's Impact on Advances of Global Gy- rokinetic PIC Codes for Fusion Energy Research[J]. Computing in Science Engineering, 2015, 17(3): 10-21.

[3] HOSHINO T, MARUYAMA N, MATSUOKA S, et al. CUDA vs OpenACC: Perfor- mance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application[C]//2013 13th IEEE/ACM International Symposium on Cluster, Cloud, and Grid Computing.

[4] DENG L, FANG J, WANG F, et al. Evaluating Multi-core and Many-Core Architectures through Accelerating an Alternating Direction Implicit CFD Solver[C]//2016 15th In- ternational Symposium on Parallel and Distributed Computing (ISPDC).

[5] STONE C P, ELTON B H. Accelerating the Multi-zone Scalar Pentadiagonal CFD Al- gorithm with OpenACC[C]// Proceedings of the Second Workshop on Accelerator Pro- gramming Using Directives. WACCPD '15. Austin, Texas: ACM, 2015

[6] WANG W X, LIN Z, TANG W M, et al. Gyro-kinetic simulation of global turbulent transport properties in tokamak experiments[J]. Physics of Plasmas, 2006, 13(9): 1089.

[7] Z L, S E, TS H, et al. Size scaling of turbulent transport in magnetically con ned plas- mas.[J]. Physical Review Letters, 2002, 88(19): 195004.

[8] Version. The OpenACC Application Programming Interface[J]. 2011.

基于申威众核处理器的RNNLM加速方法

王福全　李波
北京航空航天大学计算机学院 北京 100191 wangfuquan@huaa.edu.cn

摘要：

近些年，深度学习模型在处理分类和回归的问题上展现了巨大的潜力。其中，由于循环神经网络（RNN）的处理序列的优势，所以其被广泛用到解决自然语言处理（NLP）的相关问题中。由于传统NLP中n-gram语言模型存在很多弊端，几年前RNN被尝试构建语言模型（RNNLM），取得了不错的效果。然而模型的训练需要大量的计算资源。为解决这一问题，本文基于申威众核处理器的硬件特性和RNNLM的运算特点，设计并实现了一系列针对RNNLM的优化和并行化的方法。其中主要的三个手段是：1、RNNLM中的矩阵向量乘在申威众核处理器的优化策略；2、适配申威众核处理器的多核组的多路训练算法；3、基于RNNLM特点的流水线加速方法。对比于基于因特尔至强处理器的运行时间，我们优化后的训练效率有了明显提升。

关键词：神威处理器，循环神经网络，语言模型，数据并行，流水线

1. 引言

语言模型(LM)[1]是语音识别以及其他自然语言处理(NLP)任务中的关键部分。研究人员对如何提高LM的能力和效率做了广泛研究[2, 3]。决策树和最大熵等有效的模型算法都被用于解决这一问题。神经网络模型也在解决这一问题上表现出很大的潜力。之前的研究和实验表明基于神经网络的语言模型[4]的效果比主要的先进模型都好。RNN是一种特殊的神经网络，其在处理时间序列问题有优势,因此RNN被用于构建有效的LM。基于RNN的LM在实际的应用中，已经被证明优于传统的n-gram模型。然而RNNLM的训练过程需要消耗大量的计算资源。假设隐层有1000个节点，那么其大约需要1000*10000个参数去表示隐层和输出层之间的权值。

在训练时的这些问题严重增加了RNN实现的复杂性。在互联网数据急剧增加的现在，构建能够快速训练的RNNLM显得尤为重要。为了解决这一问题，基于硬件的加速扮演了重要的角色。之前的研究人员针对各种平台对RNNLM进行了优化，其中有GPU，ASICs，FPGSs。本次实验是基于申威众核处理器(Suyway CPU)进行RNNLM的加速方法的研究，使其能够在大规模语料上更快速的训练语言模型。

申威众核处理器在处理需要大量计算的任务时表现优异。在本次实验中我们在申威众核处理器上针对其硬件特性实现了并行的RNNLM。

针对其并行化工作，主要存在如下两方面的挑战：1.申威众核处理器拥有自己独有的异构体系结构。我们需要重新设计各种有效的方法去充分运用申威众核处理器的计算单元。2. 申威众核处理器的优势在于其主从核组合的多变多级的并行计算能力。所以我们需要分析RNN的网络结构使用并行手段加快其训练速度。为了完成这些挑战，我们设计了一系列的并行优化策略。

经过这一系列想法的实现，我们把RNNLM部署在一个申威众核处理器上。与原始RNNLM部署在一个因特尔至强处理器上对比，目前得到的加速比是8X。

本文的结构如下，第2章介绍了申威众核处理器的体系结构以及RNNLM的结构特点。第3章详细介绍本文提出的策略和各部分的实现过程。第4章展示了实验结果。第5章总结全文。

2. 相关背景知识

2.1 申威众核处理器体系结构

申威众核处理器在2016年发布，其主要搭载在神威太湖之光超级计算机上，该超级计算机在世界最快的超级计算机TOP500的名单上名列前茅。

每个申威众核处理器上包含四个核组(CGs)，每个核组包括一个MPE,一个MC以及一个的CPE集群。这四个核组通过片上的网络连接。每个核组有自己的内存空间，并且通过MC连着MPE和CPE集群。申威众核处理器通过系统接口与外部设备相连[5]。每一个核组都是可以独立运行的以及拥有一个8G大小的内存。每个集群中轻量级的CPE拥有64KB的缓存大小。申威众核处理器的体系结构如图1。

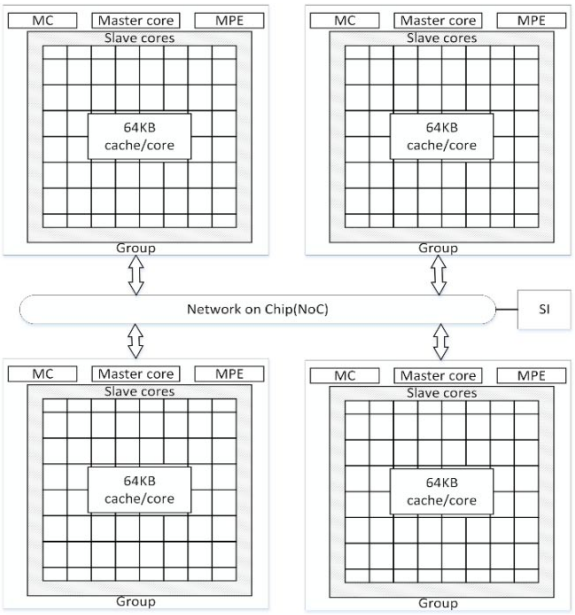


图1 神威众核处理器的基本结构

CPE集群具有强大的计算能力，通常用于执行耗时的计算。当使用CPE集群时，需要进行计算的数据应该通过DMA从核组的主存中被传输到其轻量级CPE的缓存中。申威众核处理器每个CPE只有64KB的缓存空间，在实际实验过程当中最多只能使用56KB大小。所以数据传输也成为程序运行的瓶颈。

2.2 循环神经网络语言模型（RNNLM）介绍

不像前馈神经网络当中各层之间的连接是朝着一个方向，循环神经网络（RNN）[6]在中间隐层之间有额外的循环连接。在t时刻，输入为x(t)，隐藏层的临时状态h(t)，输出y(t)就可以如下表示：

$$h(t)=f\left(w_{ih}x(t)+w_{hh}h(t-1)+b_h\right) \quad (1)$$

$$y(t)=g\left(w_{ho}h(t)+b_o\right) \quad (2)$$

这里，W_{ih}是输入层和隐藏层之间连接权重，W_{ho}是隐藏层和输出层之间的连接权重，W_{hh}代表着隐藏层两个时间连贯的时间步之间的循环权重矩阵。f(x)和g(x)各自代表隐藏层和输出层的激活函数。

基于RNNLM的输入输出层对应于全部或者被压缩的词表[7]，所以这些层的每一个神经元代表一个或者一类词。在计算句子概率的时候，这些词将按照序列的形式输入到神经网络。具体来说，x(t)代表t时刻的词，y(t)代表基于x(t)和存储在h(t-1)中的历史信息计算出来的下一词的概率分布。

RNNLM使用隐藏层的内部状态来存储历史信息，

这种做法不受限于输入历史的长度。比较起n-gram模型，RNNLM处理句子这种序列更占优势。一般在构建模型时，隐藏层的神经元的个数要比输入/输出层要少很多，隐藏层神经元的个数应该根据训练集的大小去调整，一般来说，训练样本越多，隐藏层的神经元个数应当适当增加。此外，即使增加了隐藏层也不会造成n-gram模型带的数据稀疏问题，这种种表明RNNLM具有更强的学习能力。

2.3 RNNLM的训练

当训练RNNLM的模型时，所有数据按照语料库顺序呈现。在这个过程中我们利用back-propagation through time(BPTT)[8]，如图2显示，按照时间步循环的架构可以展开成一个有向的前馈结构，然后可以通过常规遵循规则进行训练。

对于给定的输入数据网络的实际输出应该是先被计算出来，然后网络中的每一个权重通过反向传播回来的实际输出和期待输出所带来的偏差进行更新。当前时间节点的隐藏层的某一个神经元i与下一个时间节点的隐藏层的某一个神经元j之间的权值更新过程可以有如下表达：

$$w_{ji}\leftarrow w_{ji}+\eta\cdot\sum_{t=1}^{T}\delta_j(t)\cdot x_i(t) \quad (3)$$

其中$X_i(t)$表示输入层神经元i；η是学习率；$\delta_j(t)$是从j节点反向传播的残差，T是BPTT所需要传播的时间步。

在输出层，我们采用softmax激活函数$g(z)=\frac{e^z}{\sum k^{e^{z_k}}}$作为交叉验证的损失函数，神经元p的残差$\delta_p(t)$可以简单地通过实际输出$O_p(t)$和期望输出$t_p(t)$获得，公式如下：

$$\delta_p(t)=t_p(t)-O_p(t) \quad (4)$$

隐藏层所用的激活函数是Sigmoid函数$f(z)=\frac{1}{1+e^{-z}}$，所以神经元k的残差的计算公式为

$$\delta_k(t)=f'(x)|_{f(x)=h_k(t)}\bullet\delta_{BPTT}(t) \quad (5)$$

其中，$h_k(t)$是t时刻隐藏层第k个神经元的状态。$\delta_{BPTT}(t)$是通过时间步向后传播的残差的累积。确切的表达式为

$$\delta_{BPTT}(t)=\sum_{o\in output}w_{ok}\delta_o(t)+\sum_{h\in hidden}w_{hk}\delta_h(t+1) \quad (6)$$

上式中$\delta_O(t)$代表t时刻输出层的残差，同时

$\delta_h(t+1)$是t+1时刻的后向传播而来的隐藏层残差。RNNLM的训练过程如算法1。

算法 1：RNNLM的训练过程

```
1.  装载训练语料库得到词表V
2.  构建RNNLM运算空间并初始化
3.  遍历语料中的词按序列读入t时刻的词(Wt)
4.      前馈计算，计算每一个神经元激活值
5.      得到输出层实际输出，计算残差
6.      BPTT更新各参数
7.      计算一定的次数后，运行一次验证集
8.  遍历结束
9.  保存模型到文件中
```

3. 循环神经网络语言模型的并行化方法

我们基于申威众核处理器的体系结构对RNNLM进行了并行优化，并将其部署到一个申威众核处理器上运行。本章将详细描述我们做的3个方面的优化研究工作。1、RNNLM中的矩阵向量乘在申威众核处理器的优化策略；2、适配申威众核处理器的多核组的多路训练算法；3、基于RNNLM特点的流水线加速方法。

3.1 矩阵向量乘优化

在神经网络的训练过程中，需要耗费大量的运算资源，而矩阵向量乘在这其中占了很大一部分。在CPU和GPU上，基础线性代数子程序库(BLAS)库在计算这类问题上表现很好。许多深度学习框架比方说caffe都用了BLAS去提高自己的训练速度。申威众核处理器提供了类似BLAS的快速进行矩阵向量乘的接口，我们在可用的地方用了该接口。

然而，有些类似矩阵向量乘的地方，其就无能为力了，例如sigmoid、softmax函数。我们利用CPE集群的计算能力设计了一些操作。然而在主存和本地内存之间的DMA传输相对效率低下，并且由于轻量级CPE的本地内存的64K的限制，我们得频繁的执行DMA传输。为了除了这一情况，首先尽可能减少DMA传输的次数和数据的数量，其次确保每个轻量级CPE的本地存储空间的充分利用。最后，查找可重用数据。

为了实现以上所述，我们对需要进行运算的数据进行分块的操作。我们将数据切分成大小略小于一个轻量级CPE的本地存储空间。将数据较为规整的划分了为若干数据块。以图为例，数据分块大小为A*B，本地内存的可用空间P，对于M*N的数据我们将数据分块的长宽分别设为近似$(P*M/N)^{0.5}$和$(P*N/M)^{0.5}$,然后将数据分块一次分配给每个计算单元。

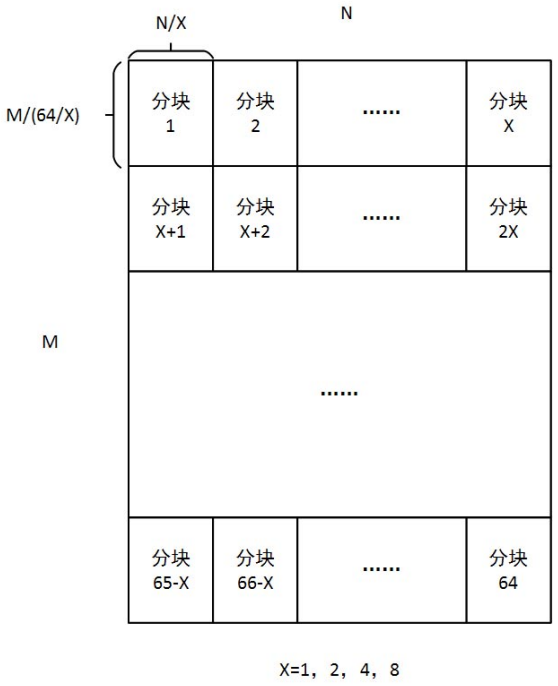


图2 分块示意图

算法2较为详尽的表述了这一思想的实现过程。

算法 2：基于矩阵分块的计算框架

```
输入：data[m], para
输出：result
1. 设置可用空间resultSize和use[m];
2. index=id得到分块序号
3. for i: 1 to m
4.   设定分块大小及设定行列分块个数row[m], col[m]
5. for i: 1 to row[0]
6.   for j: 1 to col[0]
7.     if (index + 64<row[0]*col[0])          index += 64;
8.     else          return;
9.     计算分块的地址偏移量及实际分块大小
10.    DMA_get从内存读取数据
11.    if (m>1)
12.      按数据块个数遍历
13.        计算数据块偏移
14.        DMA_get将数据从主存督导本地内存
15.        func计算结果
16.    else          func计算结果
17.    计算slaveResult在result中的偏移量
18.    DMA_put结果写回内存
```

算法中m为单位运算载入数据块的个数、data[m]为内存中算法所需数据变量、size[m]为data的大小（包括x、y两个维度）、slaveData[m]和slaveSize[m]为计算单元中的数据块变量和大小、result为内存中存储结果的变量、slaveResult和

resultSize为计算单元cache存储结果的变量和大小、para为算法参数、id为计算单元序号、func为slaveData[i]运算函数。我们以一个数据块为算法基础，将数据按分块大小划分为若干块，并把数据块分配给对应计算单元；其次将另一个数据按算法需求分块载入；然后调用函数func实现计算操作；最后将结果写回内存。其中数据在计算单元阵列的cache与内存中的传输采用了申威众核处理器支持的跨步传输技术，减少数据传输的次数和开销。

基于矩阵的计算框架和基于向量的计算框架是类同的。基于这些我们就解决了sigmoid、softmax等函数的并行优化问题。

3.2 适配申威众核处理器的多核组的多路训练算法

我们要在一个申威众核处理器上部署RNNLM，一个处理器上有4个采用片上融合技术的相对独立的核组。因此我们提供了一套在多个核组上进行模型训练的方法。这种方法主要是对之前基于数据并行模型训练的方法的改进。

在数据并行模型中，训练数据被分成了n块。每一个训练单元拥有一份完整网络模型的备份并且用自己拥有的那份数据进行训练。在训练到一定时间，将自己的有过更新的权值参数传输到参数服务器进行组合更新，然后参数服务器将参数返回到训练单元继续进行训练。相比于单核组版本，其在训练过程中要同步参数。这种机制[9]的框架如图3：

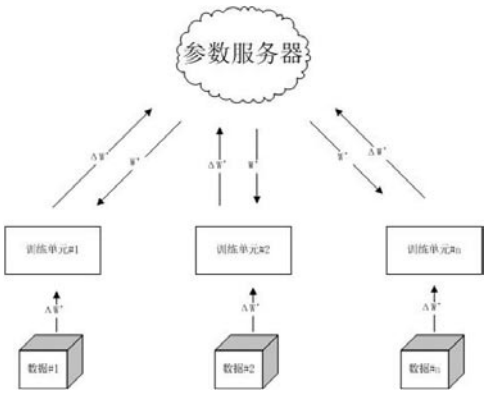


图3 带有参数服务器的权值同步

在我们的实现中，每个核组是一个训练单元，我们用MPI接口进行数据在训练单元之间的参数传输。

图4展示了我们改进的数据并行框架。

我们设置了多个参数服务器。但是数据在核组之间进行传递是非常费时的。所以我们让每个参数服务器对所有参数的一部分负责，比起只用一个参数服务器，这种框架能够减少传输，并且相当于增加了一个计算节点，减少了平均的工作量。

结合上节的向量矩阵乘法的优化，我们把这两个技术手段实现在一起。把它做成一种申威众核处理器支持的一种核组私有模式的两级并行模式。计算框图如算法3。

算法 3：核组私有模式的两级并行模式

1. 输入：训练集、验证集和测试集
2. 输出：模型保存文本
3. 装载训练语料库得到词表V
4. MPI_Init()
5. 构建RNNLM运算空间并初始化
6. 执行算法2
7. 执行到一定时间利用MPI传输梯度数据，
8. 每个训练单元更新参数
9. 计算一定的次数后，运行一次验证集
10. MPI_ MPI_Finalize()
11. 保存模型到文件中
12. 运行测试集

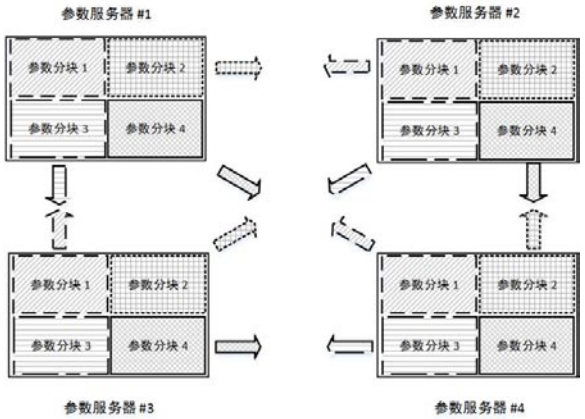


图4 多参数服务器分发参数过程

3.3 基于RNNLM特点的流水线加速方法

本节我们提出一种基于RNNLM结构特点的流水线加速优化策略。

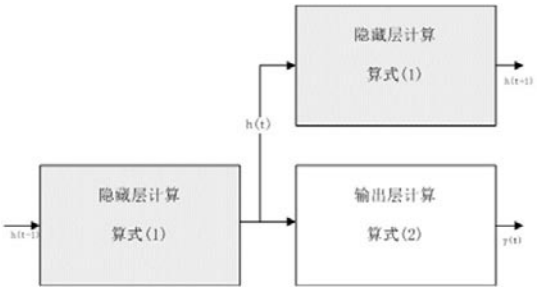


图5 两段式流水线

一般RNN流水线加速[10]如图5所示，将前馈阶段为两个阶段：从输入层到隐藏层和从隐藏层到输出层。将这两阶段做流水线。

然而实际情况却不是这么理想，因为这两个阶段的计算量机器不对等，流水线效果出不来。假设输入层和输出层的神经元个数是V，隐藏层的神经元的个数是H。输入层输入的是one-hot形式向量，每次只激活一个神经元，所以$W_{ih}x(t)$相当于提取W_{ih}中的某一行。因此h(t)的计算主要集中在$W_{hh}x(t-1)$，复杂度大体为O(H*H)。相对于y(t)的O(H*V)，其计算量很小。因为V是基于训练数据的词表，其大小是十到百万不等，然而H的大小是百千计的。因此，第二阶段的计算量远高于第一阶段的计算量。这样图示的两段式流水就没有意义。

为了解决这个问题，我们抛弃第一阶段的优化，针对第二阶段实现了优化。

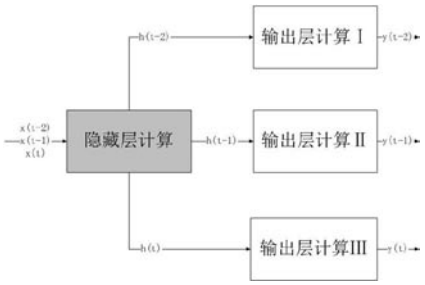


图6 隐藏层和输入层流水线设计

如图6所示，我们复制了几份输出层的处理单元，简单讲就是我们让处理隐藏层运算的单元持续性运行，同时让输出层的运算并行的执行，每个执行单元运算的是不同时刻的。例如，有三个输出层运算单元，在t-2时刻，隐藏层的计算结果传输到输出层运算单元1，当输出层1在t-2时刻进行计算，那么隐藏层在t-2时刻的计算结果将传递到下一个输出层运算单元2。这样的流水线设计比图5展示的两段式流水线更有效率。我们可以通过简单的公式去看一下加速效果。

$$S=\frac{(t_V+t_H)+t_V*(B-1)}{t_H*B+t_V} \quad (7)$$

其中，t_V和t_H分别代表输出层和隐藏层计算的时间消耗，B表示输出层运算的节点数。

这个方法的关键是基于RNNLM的规模输出层运算单元的数量B的设置。B设置的很大，我们需要更多的运算资源来充当输出层运算单元，而且从理论上能获得更高的加速比。我们想把模型部署在一个申威众核处理器上，而只有4个核组进行选择。所以本次实验我们把B设置成3。

本节的流水线优化算法结合3.1节中的优化策略可以组合成一种核组私有模式的两级并行模式。

4. 实验结果

在我们的实验中，我们主要专注于RNNLM的训练速度。为了测试我们提出的优化策略，我们在WSJ语料库上进行了不同优化策略的测试。

本文对比实验运行平台是因特尔至强E5-2420处理器，并在其上运行串行程序，最终与我们部署在一个申威众核处理器上的并行算法做比较。表1给出两个处理器的简单参数对比。

表1 两硬件平台参数对比

	申威26010	至强E5-2420
频率（GHz）	1.45	2.2
设备内存(GB)	8/核组	8
核组数(个)	4	略

第三章中阐述了我们的优化的三个策略。其中3.1节和3.2节构成一种完整的两级优化策略，为了方便，我们以下称为1+2策略。3.1节和3.3节构成另一种完整的两级优化策略，同样我们称其为1+3策略。针对这两种策略我们调整模型中超参数进行实验。

首先我们调节模型中隐层的数量和BPTT的步长在这两个平台进行了测试，实验中BPTT步长分别2、4和8，隐藏层神经元数量分别为128、256和512，测试结果表2。

表2 1+2策略运行时间(s)对比

隐层数量	BPTT=2		BPTT=4		BPTT=8	
	至强	申威	至强	申威	至强	申威
128	1012	172	1808	300	4121	679
256	1771	229	3893	492	7992	960
512	3210	366	4930	554	13990	1550

为了更直观的展示本次加速结果，我们将该策略下加速度随BPTT和隐藏层神经元的个数变化的以图表的形式展现。如图7。

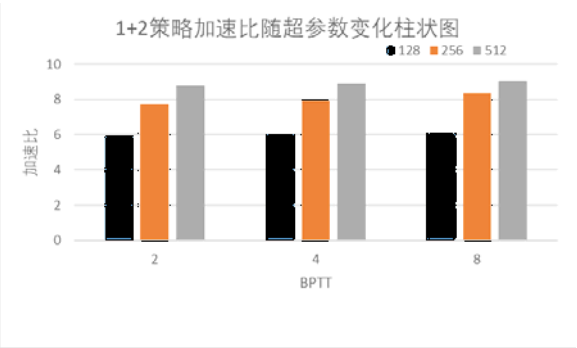


图7 1+2策略加速比

同样的，1+3策略测试结果表3和更直观的加速比图8。

表3 1+3策略运行时间(s)对比

隐层数量	BPTT=2		BPTT=4		BPTT=8	
	至强	申威	至强	申威	至强	申威
128	1012	333	1808	590	4121	1369
256	1771	387	3893	854	7992	1755
512	3210	635	4930	973	13990	2780

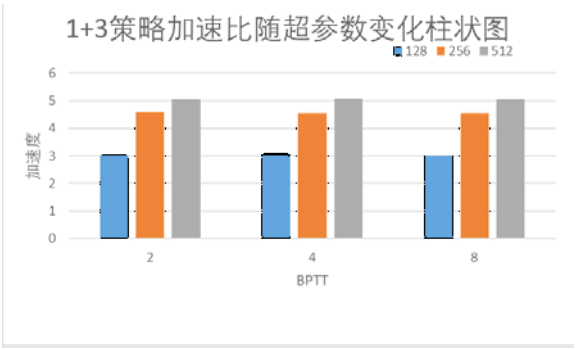


表8 1+3策略加速比

结合实验结果来看，策略1-2比策略1-3运行起来更有效率。其中两种策略运行在申威众核处理器的一个核心(4个核组)与模型在至强上串行时间相比，最终加速比分别为8.89和5.05左右。

5. 总结和未来工作

本次实验提出了用申威众核处理器去加速RNNLM。从结果上来看，通过优化改进，模型的运算效率确实得到了提升。

本文首先分析申威众核处理器的硬件特点和RNNLM的运算特点，提供了并行地对矩阵向量乘以及类似的运算进行处理的方案。紧接着又将数据并行进行改造使其适应申威众核处理器的硬件特点，达到提升性能的作用。最后提出一种在RNNLM结构上进行流水的机制。

本次实验虽然只部署在一个申威众核处理器上，但是因为核组之间相对独立，所以该框架相对容易进行扩展，不过要达到最优，得重新调整负载和数据传输等。

不足，1、数据并行会使最终训练出来的模型的困惑度略高于基准。不过普遍是可以接受的；2、本次实验后半段，我们投入很大精力去将策略2和策略3进行整合，数据并行和模型并行整合。可是效果并不理想。

未来希望能够将这次实验不足的地方进行更深入的研究。

参考文献：

[1] 邢永康, 马., 统计语言模型综述. 计算机科学, 2003. 30(9): p. 22-26.
[2] Rosenfeld, R., Two decades of statistical language modeling: Where do we go from here? 2000.
[3] Xu, W. and A.I. Rudnicky, Can artificial neural networks learn language models? 2000.
[4] Bengio, Y., et al., A neural probabilistic language model. journal of machine learning research, 2003. 3(Feb): p. 1137-1155.
[5] 国家超级计算无锡中心, “神威·太湖之光”系统快速使用指南. 2016.
[6] Mikolov T, K.M., Burget L, et al, Recurrent neural network based language model, in Interspeech. 2010. p. 3.
[7] Mikolov T, K.S., Burget L, et al, Extensions of recurrent neural network language model, in 2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). IEEE. 2011. p. 5528-5531.
[8] Elman, J.L., Finding structure in time. Cognitive science, 1990. 14(2): p. 179-211.
[9] Zinkevich, M., et al. Parallelized stochastic gradient descent. in Advances in neural information processing systems. 2010.
[10] Huang, Z., et al. Accelerating recurrent neural network training via two stage classes and parallelization. in Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on. 2013. IEEE.

面向国产SW26010众核处理器的科学计算核心深度优化研究

● 许志耿　林新华
上海交通大学高性能计算中心 上海 200240

摘要：

神威·太湖之光是我国首台自主设计研发并登顶TOP500排行榜的超级计算机系统，搭载了国产SW26010众核处理器。该处理器的架构与传统众核处理器、通用图形处理器相比有很大区别，其浮点运算理论峰值高达3.06TFlops，但主存理论峰值带宽仅为134GB/s，导致浮点运算与访存能力极不平衡。为充分发挥科学计算应用在SW26010处理器上的性能，同时评估该处理器在科学计算领域的应用潜力，我们研究了基于SW26010架构的科学计算核心深度优化方法。首先，我们使用自主研发的汇编微基准测试集，全面评估了SW26010众核处理器流水线、访存和片上通信的性能和架构特点。然后，针对计算密集型核心——稠密矩阵乘法，我们设计了基于片上寄存器通信的多级并行算法，并且在底层汇编指令层面实现了高效计算内核，使单、双精度矩阵乘法的性能均达到了处理器理论峰值性能的90%以上；针对访存密集型核心——Stencil计算，我们采用了对空间与时间维度分块、重叠数据传输与计算、基于寄存器通信提高计算强度等优化策略，将性能提升到了理论最高优化性能的70%以上。基于架构研究、计算核心优化的结果，我们发现：（1）对于访存密集型应用，以及部分计算强度低于33.84的计算密集型应用，设计基于寄存器通信的数据共享和并行计算方法是打破SW26010的访存性能瓶颈、获取更高性能的重要途径；（2）对于高计算强度的科学应用，底层汇编指令重排和提高指令双发射率是发挥处理器计算性能的核心优化手段。

关键词：神威·太湖之光，国产SW26010众核处理器，稠密矩阵乘法，Stencil计算，寄存器通信，汇编优化，访存优化

神威·太湖之光是我国首台自主设计研发并登顶TOP500排行榜的超级计算机系统，搭载了国产SW26010众核处理器。该处理器的架构与传统众核处理器、通用图形处理器相比有很大区别，其浮点运算理论峰值高达3.06TFlops，但主存理论峰值带宽仅为134GB/s，导致浮点运算与访存能力极不平衡。为充分发挥科学计算应用在SW26010处理器上的性能，同时评估该处理器在科学计算领域的应用潜力，我们研究了基于SW26010架构的科学计算核心深度优化方法。目前在太湖之光上的大规模科学应用移植优化工作主要侧重于应用的可扩展性、整机效率上，因此上层算法设计是研究重点。在我们的研究工作中，我们侧重于“神威·太湖之光”超级计算机单节点上科学计算核心的深度优化，以进一步提升大型应用的性能。针对SW26010架构，提出了基于寄存器通信的并行优化算法，同时针对底层存储访存与流水线架构特性进行了汇编级别调优，取得了较为理想的优化效果。我们工作的主要科研贡献是：1. 深入分析了SW26010众核处理器的架构特性，并提供了全面、准确的性能量化参数。2. 提出了适应SW26010架构的稠密矩阵乘法、Stencil计算的深度优化方案。3. 基于性能架构分析与科学计算核心深度优化结果，提出了面向SW26010众核处理器的科学计算应用深度优化策略。

1. SW26010 架构简介

神威众核处理器采用了独特的主从核异构架构，和传统的商用多核/众核处理器不同的是，每个众核处理器具有4个核组(Core Groups，CG)，这4个核组通过片上网络连接，提供了高达3TFlops左右的

浮点峰值运算能力。每个核组包含一个主核，称为管理单元（Manage Processing Elements，MPE）和64个从核，称为计算处理单元（Computing Processing Elements，CPE）。主核和从核均采用了64-bit RISC指令集，每个核心均仅支持单线程，工作频率为1.45GHz，支持256位向量寄存器计算（包括乘加指令FMA）。但由于流水线、指令发射等微架构存在差别，因此在实际应用中分别承担不同部分的计算任务。另外，每个核组通过128-bit的内存控制器可以访问8GB DDR3主存，理论峰值带宽为34GB/s，4个核组的聚合带宽为136GB/s[2]。与3TFlops的浮点运算峰值相比，访存带宽较为受限，这对计算加速优化有较大影响，在接下来的章节我们会进一步进行讨论。

1.1 主核架构

主核可以运行于用户态和系统态，支持函数中断、乱序执行，因此适用于作业调度、通信管理等任务。每个主核中含有两条浮点运算流水线，支持乘加复合指令，因此，在完全流水的情况下，每条浮点运算流水线每个时钟周期可以产生个8浮点运算操作的吞吐量，4个核组中的4个主核一共提供了0.0928TFlops的峰值浮点运算能力。在存储器层次方面，每个主核分别含有32KB的一级数据缓存和一级指令缓存，以及256KB的二级数据、指令unified缓存。

1.2 从核架构

与主核不同的是，从核仅支持用户态模式，不支持函数中断。从核的设计目的在于，在有限的芯片空间上，以尽可能精简的微架构，聚合更多的核心以提供尽最高的浮点运算能力。每个核组的64个从核被组织成了8x8的从核阵列，通过片上阵列网络进行连接。

每个从核含有2条流水线，其中一条为浮点运算流水线，另一条为访存流水线。浮点运算流水线每个时钟周期可以产生8个浮点运算操作的吞吐量，因此4个核组中的256个从核一共提供了2.969TFlops的浮点运算能力，占整个处理器浮点运算峰值的97%。

在存储器层次方面，从核和主核有很大的区别。每个从核含有一个私有的局部片上Scratch Pad Memory（SPM），没有数据缓存，但是有16KB的一级指令缓存。不存在片上共享存储供不同从核使用，因此无法直接进行共享式的数据访问。但是，SW26010架构提供了轻量、快速的寄存器级别从核数据通信方式——寄存器通信。

表1 指令延迟测试结果

指令类型	指令	延迟 (时钟周期数)
算术运算	vmuld, vmuls, vaddd, vadds, vsubd, vsubs, vmad, vmas	7
	vsqrtd/vsqrts	32/18
	vdivd/vdivs	34/19
SPM 访存	vldd, vlds	4
混洗	vinsw, vinsf, vextw, vextf, vshff, vshfw	1
同步	sync, synr	14

2. 基于汇编微基准测试集分析SW26010众核处理器性能和架构

2.1 汇编微基准测试集

为了对SW26010众核处理器的体系架构和性能特性进行全面、精准和定量的分析，我们编写了一套基于汇编语言的微基准测试集。我们的测试分析工作有三个目标：（1）提出衡量各架构组件的性能的方法；（2）提供定量、准确的分析结果；（3）分析推导可能的底层硬件机制。为此，我们采用了综合的分析策略，不仅对延迟和吞吐量进行分析，也编写了有针对性的微基准测试集来推测未知的微架构实现，例如流水线上指令发射逻辑等。

在指令延迟测量方面，我们运行一个包含大量相同操作的长指令序列，且指令执行时不存在流水重叠（通常每个迭代包含1000条指令，满足从核16KB一级指令缓存大小），然后通过平均运行时间来估计指令的延迟。另外，同步指令也被用来实现通信隔离，以暴露寄存器通信的延迟；在指令吞吐量测量方面，我们构建了足够的工作负载，例如通过发射足够的独立的指令流、存储器访问请求或者寄存器通信等来充分使用硬件资源。最后，通过分析数值结果，以及专门的微基准测试程序结果，我们可以推断出影响性能的关键架构组件的潜在实现。

为保证数据准确度，除了用C语言和intrinsics编写的内存性能测试集，其余微基准测试程序内核均通过手写汇编代码实现。另外，我们使用Athread线程库来实现从核间的细粒度并行。

2.1.1 流水线分析

SW26010每个从核包含两条流水线：负责浮点运算的流水线P0，以及负责存储访问的流水线P1。定点运算均可在两条流水线上执行。SW26010支持的向量指令可以归为3类：算术运算指令(Arithmetic Instrution)、访存指令(Memory Access Instruction)、混洗指令（Permutation Instruction）。我们通过微基准测试集对流水线上的指令延迟、指令吞吐、指令发射逻辑进行了评估。

我们利用了写后读（Read After Write，RAW）相关性来构建指令延迟测试序列：va = op(va，vb，...),

其中va和vb是向量寄存器操作数，op是指令操作码，并且寄存器va即是源操作数也是目的操作数，保证了指令序列的RAW相关性。

为测试吞吐量，我们构建了一个由相互独立（不存在数据依赖）的指令流组成的长指令序列，通过逐步增加独立指令流的数目，测量运行时间得到每个时钟周期吞吐的指令数（Insruction Per Cycle，IPC）来分析流水线上不同指令能达到的最大吞吐量。

结合指令延迟的测量结果，我们发现：

1) 浮点运算指令的IPC在独立指令流达到7之后达到了最高值1，也即达到浮点运算峰值。

2) 结合浮点运算指令7个时钟周期的延迟，可以得出每个时钟周期都能有一条指令的吞吐，也即浮点运算指令可以完全流水。

3) 除法和开方指令不能完全流水。双（单）精度除法、开方指令分别需要等前一条出发指令发射之后30（15）、28（14）个时钟周期后才能发射出去。

2.2 存储分析

SW26010的存储器层次包含三层：8GB的片外DDR3主存、64KB的从核片上局部存储SPM以及寄存器文件。

与主流处理器架构不同的是，从核上放弃了数据缓存，转而采用可显示控制的草稿纸存储器（SPM）作为主存和寄存器文件之间的高速中间存储介质。在本节中我们对主存、SPM的存储访问性能和优化策略进行了评估。

2.2.1 主存访问

SW26010提供了3类DMA模式：点对点模式（PE Mode）、广播模式（BCAST Mode）、行模式（Rank Mode。我们针对不同模式的DMA进行了带宽测试。为了去除函数调用、不对齐访问的额外开销，我们保证数据128字节对齐（主核缓存行大小），并且使用intrinsic编写。

点对点模式适用于任何数据传输场景。每个从核读写各自所需的主存数据。因此我们全面地 测试了在从核数量不同、传输单位数据块大小不同的情况下DMA的读写带宽，并且使用stream基准测试集进行了实际带宽测试。所示。我们发现：

1. 读带宽最高为27.9GB/s，并且需要64个从核完全参与数据传输才能达到最大带宽；

2. 写带宽峰值为24.1GB/s；

3. 读写混合时，带宽峰值为23.4GB/s；

4. 带宽测试峰值为22.6GB/s；

广播模式适用于主存向从核广播数据，因此是单向传输。与点对点模式不同的是，广播模式下任意一个从核发起广播请求后，数据会从主存传输到从核阵列上，然后再通过从核阵列网络将数据传输到所有从核上。因此，与点对点模式相比，广播模式下数据无需进行多次重复传输。实测发现广播模式带宽为6.97GB/s，考虑到广播模式避免了63次重复的数据传输，因此有效带宽为446.1 GB/s(6.97 x 64)。

行模式实现了核组上同行从核的集合通信。与广播模式不同的是，在行模式下，任意一个从核发起行模式传输请求后，数据会从主存上循环地将数据块依次传输到请求发起核心所在行的所有从核。该模式也可以通过点对点模式下跨步传输形式实现，因此我们比较了两者的带宽。我们发现，当单次传输的数据块大小小于2048字节（256个双精度浮点数）时，行模式带宽高于点对点模式。

2.2.2全局离散访存

全局离散访存提供了较为便捷的编程模式，用户无需考虑DMA传输的显示调用，可以以访问片上存储的方式直接读写主存。我们测试了全局离散访存的Stream带宽，发现Copy、Scale、Add、Traid四种访存模式下带宽仅为3.88GB/s、1.61GB/s、1.45GB/s和1.48GB/s，与DMA模式相比非常低。

另外，我们也是用指针追逐（Pointer-chasing）方法测试了全局离散访存的数据读取延迟。我们使用一个长度为S的数组A，数组内容初始化为：A[k]=(long)&A[(k+stride)%S]，然后用指针p（p初始化为A）对数组A进行重复访问：p = (long*)(*p)。我们发现，全局离散访问的延迟和同时发起数据读取请求的从核个数直接相关：当从核个数n小于2时，每个从核的访存延迟为134-138ns（194-200个时钟周期）；当从核个数大于2时，每个从核的访存延迟为58n ns（84n个时钟周期）。因此，我们可以推测，当多个从核同时发起全局离散访存时，访存行为被串行化处理，假如64个从核同时发起访问，则访存带宽高达5376个时钟周期。

2.2.3 SPM访问

尽管SPM可以被配置为软件模拟的缓存，其性能仍然较差。因此，用户显示控制读写的使用模式仍然是高效利用片上存储的使用方式。我们对SPM进行了带宽也延迟的分析测试。

在访存延迟方面，我们在汇编语言层面进行了指针追逐测试。结果显示访存延迟为2.76ns，与之前测试的访存指令4个时钟周期的延迟一致。

在带宽方面，我们首先通过读写指令的吞吐量计算得到SPM理论带宽上限。测量结果为每个时钟周期可以有32字节的吞吐量，因此SPM理论峰值带宽为46.4GB/s。除此之外，我们将stream基准测试集

通过Athread线程库以及从核SIMD intrinsic进行了移植，测量得到了SPM的实际带宽。我们在程序中进行了循环展开以提高指令吞吐量，并且保证避免寄存器溢出的情况。结果显示，Copy， Scale， Add，以及Triad四种访存模式对应的带宽为43.74GB/s，29.36GB/s， 31.71GB/s， and 30.9GB/s。

2.3 寄存器通信分析

片上寄存器通信提供了从核阵列网络上CPE之间点对点/广播256位数据的通信方式，并向编程人员提供了一组intrinsic接口用于编程。直接的寄存器通讯只允许在同一行或同一列内的从核之间，遵循包含先进先出（First-In First-Out， FIFO）发送/接收缓冲区的匿名生产者-消费者协议，也即发送指令是异步的，并且如果发送（接收）缓冲区是满（空）的情况下，发送方（接收方）将被阻塞。

为了评估寄存器通信性能并支持基于寄存器通信的数据共享方案的设计，我们设计了一套微基准测试集，测试了点对点和广播模式的通信延迟。

为了测量点对点通信延迟，我们使用同步指令保证发送方等待数据被接受后才进行下一次发送，否则，异步发送指令会导致连续的寄存器通信之间存在延迟重叠。值得注意的是，我们并未使用乒乓测试（ping-pong test，通常用于MPI基准测试）测量通信延迟，因为我们不应在测试前假定同一对从核间发送与接收的延迟是相等的。尽管如此，我们可以使用乒乓测试来验证延迟测试结果。

我们对同一行/列中的任意从核对（从核对由发送核和接收核组成）进行了通信延迟测试。通过去除synr同步指令的执行时间（显示测量结果为14个周期）来计算寄存器通信的延迟。需要说明的是，测量的寄存器通信延迟由三部分组成：发送（putr）和接收（getr）指令的执行延迟以及通过网状网络的数据传输延迟。可以观察到，从核之间的通信延迟不尽相同，并且与网格中的接收方的位置有关，与发送方的位置无关。具体来说，要通过寄存器通信获得数据，从核4/5/6/7在任何一行花费11个周期，而从核0/1/2/3需要10个周期;在任一列中，从核2/3/6/7花费11个周期，而从核0/1/4/5需要10个周期。我们在大量的SW26010处理器上重复了这类测试，所得结果均相同。

3. 稠密矩阵乘法在 SW26010 上的深度优化

3.1 SW26010上矩阵乘法初始优化版本

我们可以将核组中的每个计算核心作为单独的计算资源，通过矩阵分块将计算任务划分为多个小规模任务，由64个计算核心分别完成。

我们将三个矩阵A，B和C分成若干个48×48的数据块，以适应64KB的从核局部存储SPM的容量，因为3 x 48 x 8 = 55926 < 64000。具体来说，我们通过Athread库将三层循环的计算内核offload到从核，通过DMA intrinsic接口传输将主存储器中的矩阵块数据传输到从核的局部存储上，并利用SIMD intrinsic将最内层的k循环进行向量化。在单个核组上实测性能为76GFlops，约为单个核组峰值性能的10%。

3.2 基于寄存器通信机制的多级并行优化算法

根据SW26010的存储器、计算核心层次架构，我们设计了相应的多级并行算法。各并行层次的计算任务与数据划分如图\ref{3layergemm}所示。首先，我们在主存上进行了一级数据划分，以应对大规模计算任务；其次，在核组内部，我们结合寄存器通信进行了二级数据划分，实现从核间协同计算；接着，在从核内我们进行了三级数据划分，实现寄存器级别通信和计算；最后，在计算内核进行Register Blocking和汇编指令重排，实现流水线上实现指令级并行。

3.2.1 数据划分层次一：主存数据分块

考虑到每个核组拥有8GB DDR3内存，整个核组一共64*64KB的局部存储空间无法容纳超过4 MB的数据，因此在进行实际的大矩阵运算时，必须对矩阵进行初步分块，在不超过SPM容量的前提下，保证每个从核能够容纳尽可能多的数据。我们采用的分解方法如图中层次一所示。

另外，由于DMA传输可以异步进行，因此可以在DMA传输的同时进行浮点运算。这可以通过双缓冲技术实现：在存储区域预留两倍的读写缓冲区；在多次迭代时，先读取第一次迭代所需数据到缓冲区0，然后开始计算；在计算缓冲区0的同时，以异步的方式同时读取第二次迭代所需数据到缓冲区1，通过交替使用缓冲区0和1，实现计算与数据传输的重叠。

3.2.2 数据划分层次二：核组内数据分块，从核间基于寄存器通信进行协作计算

在初始版本中，我们将每个从核作为单独的计算核心进行计算，不同计算核心片上的数据无法共享，每次数据更新均需要从主存获取，因此主存带宽需求大，成为了性能瓶颈。为提高数据复用率，降低主存带宽需求，我们通过寄存器通信进行从核间数据复用，突破主存带宽的瓶颈。

具体来说，我们将8x8从核阵列作为一个整体，主存分块数据传输到从核阵列上之后，每个从核分别存储分块数据的1/64，也即将数据块映射散布到整

个从核阵列上。接下来，我们以列乘以行向量（也即外积）的计算方式进行整个从核阵列的计算。由于被映射到了从核阵列上，整个计算过程分为8个轮次，第i轮次将A数据块的第i列与B数据块的第i行相乘，得到结果矩阵C数据块的部分和。在8个轮次计算完毕后，累加得到的矩阵C数据块也即A数据块和B数据块矩阵相乘的结果。

在实现上，我们使用寄存器通信将每一轮次的计算数据发送到对应位置的从核进行计算。举例来说，在第一轮次中，我们将计算A的第一列与B的第一行子矩阵的乘法。此时，从核阵列第一列的从核通过行广播，将数据发送给同一行的其余从核；第一行的从核通过列广播，将数据发送给同一列的其余从核。因此，以位于第三行第四列的从核（2，3）为例，将通过寄存器通信获得所在行的第一列从核（2，0）的矩阵A的数据，以及所在列的第一行从核（0，3）的矩阵B的数据。在获得数据之后，便可进行计算，将结果存储在本地，因为最终A的第三行和B的第四列子矩阵的乘积都将存储在这一从核上。

3.2.3 数据划分层次三：从核内部数据分块，支持寄存器粒度的通信和计算

经过前两步的划分，每个从核上均容纳了2个m x k的A子块，2个k x n的B子块，以及2个m x n的C子块（双缓冲要求A、B、C均需要2个存储区域），并且尽量保证占满整个片上局部存储SPM。考虑到寄存器通信每次只能发送4个double类型（256位）数据，我们需要多次寄存器通信才能将从核上的子矩阵数据全部发送到负责计算的从核上。因此，我们需要在寄存器通信的同时进行计算，将通信的延迟和计算开销重叠，才能最终达到峰值性能。为了达到这一目的，我们需要将从核上的子矩阵进一步分块，使得每次寄存器通信能够传输分块中的一小部分数据，并且在数据传输到目的从核后能够立即进行计算，在计算的同时接收下一轮次的数据，也即以流水化的形式实现计算与通信的重叠。

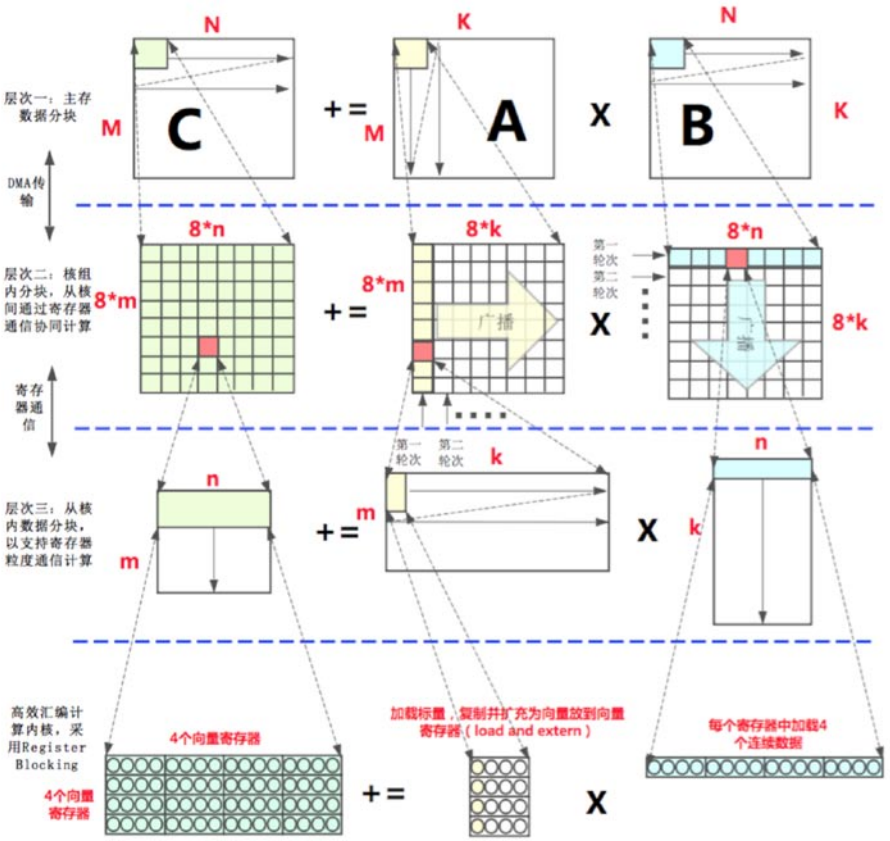


图1 矩阵乘法多级并行算法

3.2.4 矩阵乘法内核的汇编指令级性能调优

我们在C语言层面进行了前述章节优化之后，性能为518GFlops，与从核阵列的742GFlops峰值仍然有一定距离。根据处理器性能和架构特性得知，SW26010流水线内部为顺序发射，导致编译器生成的汇编代码质量对运行时实际性能有直接影响。因此，我们对编译器生成的汇编代码进行了进一步分析，发现了如下导致流水线运行效率低下的问题：

1. 相关指令排列过于紧凑，导致指令序列间存在大量气泡，导致流水线停顿。

2. 寄存器使用不充分，32个寄存器仅适用20个左右，导致寄存器分配时存在寄存器溢出的问题，引起大量的SPM访存加载行为，进一步导致流水线停顿。

3. 浮点运算、访存指令匹配不恰当，双发射率低。

为解决上述汇编代码问题，我们重新编写了矩阵乘法计算内核的汇编代码。对于双精度矩阵乘法（Double prcision General Matrix Multiplication，DGEMM），我们使用8个向量寄存器分别用于存储矩阵A和B每次传入的数据，以及另外16个寄存器用于存储每次迭代的计算结果。由于充分地使用了不同的寄存器，16条乘加指令彼此不存在相关性，因此可以以完全流水的方式发射；同时，数据加载与寄存器通信指令与乘加指令进行了配对，从而实现双发射；最后，由于加载指令与乘加指令存在相关性，我们在第一次迭代开始前加载了所需数据，在计算过程中，当该存储矩阵A或B的寄存器不会再被使用之后，我们立马进行了下一轮次的数据加载/收发，并且数据的加载/收发可以和计算指令双发射出去，因此可以重叠计算指令和访存/寄存器通信指令的耗时，并且寄存器通信的延迟隐藏在了计算耗时中。在基于数据分块、重新设计寄存器通信并行算法以及汇编代码级别的调优之后，我们将矩阵乘法的运算性能从76GFlops提高到了679GFlops，达到了理论峰值性能的92%。

4. Stencil 计算在 SW26010 上的深度优化

Stencil计算是许多科学计算应用程序的核心，例如地球物理学，天体物理学，核物理学或海洋学，常出现在超级计算机涉及的数值模拟领域。具体来说，Stencil计算的输入通常是中心网格元素u(t，i，j，k)和多个i，j，k维度的相邻值，输入元素的数量被称为Stencil的点数。

4.1 二维Stencil优化分析

我们以二维Jacobian Stencil作为研究对象。这是一个二维5点2轴的计算模式，在计算过程中遍历二维空间中的点，取X轴与Y轴上的相邻点求和再取平均值进行更新。我们将二维Jacobian Stencil计算移植到SW26010从核阵列上，并进行向量化，使用DMA点对点模式进行跨步数据传输。这一初始实现版本的性能仅为5.2GFlops。

4.1.1 不对齐访存优化与循环展开

由于Stencil计算需要对相邻空间点的数据，因此在向量化之后，当中心点向量满足256位数据对齐条件时，以距离中心点左右64位的两点为起始点的向量必定不能满足256位数据对齐条件，产生访存不对齐问题。在SW26010上，不对齐的向量载入需要执行两次加载指令，然后再进行数据高地位混合操作，因此其时间开销是对齐访问的两倍以上。我们发现，由于中心向量已经被载入用于计算，因此为了获取左右两点起始的向量，我们可以加载中心点往左256位的向量，以及中心点往右256位的向量，然后通过混洗指令得到目标向量。另外，为了配合不对齐访存优化过程中对寄存器复用，我们对循环最内层进行了循环展开，展开次数为6次，保证最大限度降低寄存器与SPM之间的冗余读写。

4.1.2 双缓冲优化

为了将计算与通信进行重叠，我们需要双缓冲技术。考虑到计算过程中，不能直接将计算结果覆盖到原先的网格上，因此需要同时保留原始网格数据和新网格数据，导致从核的SPM局部存储上需要两倍的空间。而双缓冲技术需要额外的缓冲区用于传入下一轮次计算所需数据，同时传出当前轮次计算结果，因此所需空间再次加倍。因此一共需要1936x4x8Byte也即60.5KB的存储空间，64KB的SPM局部存储恰好满足要求。

4.1.3 基于寄存器通信的时间步优化（Temporal Blocking）

上述优化均基于计算单次时间步的情况。每当当前时间步计算结束之后，从核阵列上的数据需要通过DMA传输回主存，然后再将当前时间步中，下一块求解域的数据通过DMA传输到从核阵列上进行计算。当整个网格当前时间步计算完毕之后，再进行下一时间步的计算。该计算模式导致计算强度过低，迫使流水线大部分时间等待数据传输和加载，其最高性能无法超过峰值的5%。

为了解决这一问题，我们需要增加Stencil计算强度，提高数据复用率和流水线运转效率。因此，可以通过在从核上计算多个时间步，也即在时间t维度上进行分块（Temporal Blocking），增加计算量与数据传输量的比例。简单的Temporal Blocking方法在单个从核上进行，但是由于从核所能容纳的数据块较小，并且每增加一个时间步计算会使得有效输出网格区域每次向内缩减一个单位（边缘数据无法更新），为保证一定规模的输出网格面积，可一次性计算的时间步数量受到了限制。另外，在相邻从核上，由于不进行数据交换，边缘区域数据的计算会重复进行，增加冗余的计算时间。因此，在从核上进行简单的Temporal Blocking并不能真正提高计算强度。为此，我们设计了结合寄存器通信的Temporal

Blocking方法，提高整个从核阵列的有效输出网格面积，并且可以在一次数据传输之后，进行更多次的时间步计算。

在具体实现上，由于我们在Y轴进行数据拆分，保证行方向相邻的从核上的数据在问题空间上也是相邻的。这样一来，每次在进行边缘数据计算时，可以通过寄存器通信进行数据交换，从而保证边缘数据可以随时间步一同更新，从而提高从核阵列的有效计算面积，减少不同从核由于数据不共享产生的重复计算。

另外，我们将分支判断从循环最内层移至最外层，以减少分支转移的开销。虽然这样会导致不同分支中需要嵌入部分重复的代码，导致代码量增长，但在保证不超过指令缓存大小的情况下，性能提升不会受到影响。

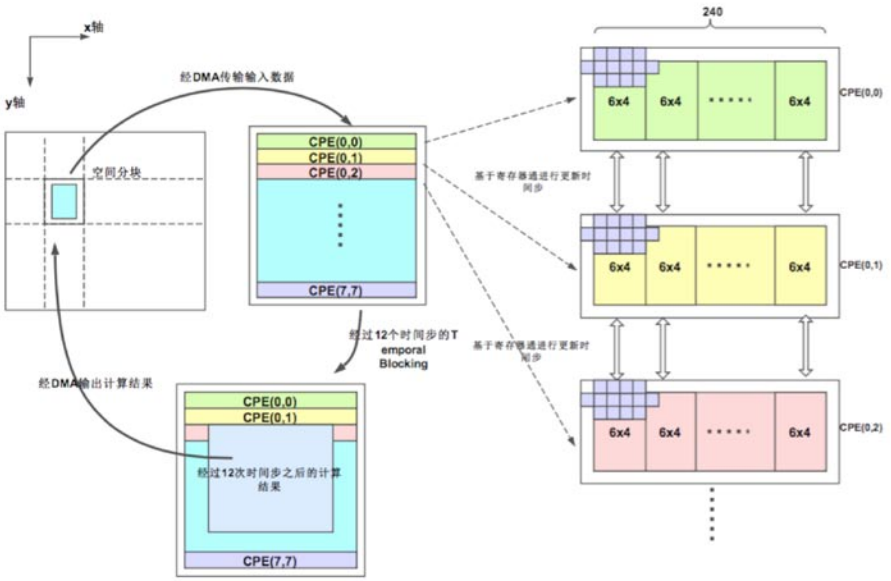


图2 stencil优化算法流程图

4.1.4 优化结果分析

经过一系列优化之后，我们将性能提高到了94.7GFlops，进一步减少分支转移开销之后，我们最终将性能提高到了104.1GFlops，也即14%的处理器理论峰值性能。与初始并行版本5.2GFlops的性能相比，加速比为21倍。我们在单个从核上进行了小规模二维Stencil计算，经过向量化、不对齐访问优化、循环展开等手段进行充分优化之后，单个从核上的性能为1.99GFlops，考虑到从核阵列含有64个从核，因此在从核阵列上进行大规模二维Stencil计算所能达到的性能极限是1.99 x 64也即 127.4GFlops。我们优化后的实测性能达到了104.1GFlops，与最优优化性能127.4GFlops相比，达到了最佳优化性能的82%。

4.2 三维Stencil优化分析

我们以三维Jacobian Stencil计算作为三维Stencil优化研究对象。与二维Jacobian Stencil计算类似，其计算模式为求取x，y，z三个维度相邻7个空间点均值以更新中心点。三维Stencil与二维相比同样具有计算强度低的问题，而其三个维度的数据访问使得在64KB的从核局部存储空间上的访存优化存更加困难。本节我们将基于二维Stencil的平面网格优化算法，设计z轴方向的优化方法。

4.2.1 三维Stencil优化分析

由于三维Stencil需要z轴的两个相邻点作为输入，因此每次计算至少需要传输3个平面，从而计算得到中心平面作为输出。为了实现三维的Temporal Blocking，我们需要在从核的局部存储SPM上存放多个平面，而每多计算一个时间步，所需平面就增加两个，64KB容量的SPM限制了时间步的增加，并且空间使用十分紧张。另一方面，当我们将计算得到的平面传输回主存后，为计算下一个网格空间，需要再次传输多个平面，这将导致上一次计算过程中大部分计算被重复执行，且重复传输大量数据。与二维Stencil类似的初始实现版本仅达到4.7GFlops的性能。

为解决上述问题，我们采用一种暂存各时间步平面的z轴方向的Temporal Blocking方法，同时在x，y轴上采用二维Stencil的基于寄存器通信的Temporal Blocking方案。Z轴优化方法如下：

1. 从主存上传入T0 时间步对应的 7 个平面。

2. 开始计算往后三个时间步直到 T3，并且保留计算过程中z轴高处的平面。

3. 从主存传入一个 T0 时间步对应的平面。由于我们此时有暂存的各个时间步平面，可以直接根据新传入的平面，计算出下一时间步对应的平面，同时放弃位于底层的平面(直接覆盖以保证存储平面数不超过8个)，直至T3 时间步，并且将T3 时间步内位于底层的计算完毕的平面传回主存。

4. 重复第三步，继续传入新的平面，计算得到更多的 T3 时间步平面。

4.2.2 三维Stencil优化分析

与二维Stencil优化结果不同的是，三维Stencil的Temporal Blocking时间步较少，导致性能加速比低于二维测试结果。最终我们将三维Stencil的性能提高到了67.4GFlops的计算性能，与初始并行版本相比加速比为14.3倍。我们在单个从核上同样进行类似的小规模三维Stencil计算，用于分析优化所能达到的最优性能。测试发现，三维小规模Jacobian Stencil所能达到的最优性能为1.42GFlops，因此整个核组的优化极限性能为1.42 x 64也即90.88GFlops。而我们优化后的实测性能为67.4GFlops，达到了最优优化性能的74%。

5. 结论

我们自主设计并实现了一套针对SW26010众核处理器的微基准测试集，从流水线、访存、片上通信三个方面全面评估了架构性能和特点，并且得到了大量精确的性能量化结果，包括指令延迟吞吐、片上/片外访存带宽和延迟，以及寄存器级别通信延迟和带宽等。基于对SW26010众核处理器的架构特点和性能的评估，我们对稠密矩阵乘法、Stencil计算这两类分属计算密集型、访存密集型的科学计算核心进行了深度优化。针对矩阵乘法问题，我们设计了多级并行算法，在核组间、核组内从核间、从核内部三个层次进行了数据划分以满足各层次数据传输带宽的要求，并且结合寄存器通信设计了从核间协同计算、数据共享的并行算法，降低了主存带宽的压力，打破主存带宽的瓶颈。最终经过底层汇编代码的重写编写，消除指令气泡、提高双发射率、实现指令完全流水化之后，将稠密矩阵运算的单、双精度性能均提升到了峰值性能的90%以上。针对Stencil计算，我们以二维和三维Jacobian的Stencil计算核心为研究对象，设计了空间、时间步维度的分块（Blocking）算法，并且对数据传输、不对齐访存、分支转移等方面进行了优化，最终将性能提升到理论优化极限性能的70%以上。

根据微基准测试和科学计算核心深度优化结果，我们认为，对于计算强度高于33.4的计算密集型科学计算核心，在SW26010众核处理器上使用传统的并行优化方法，加以汇编指令级别调优后可以充分发挥处理器性能。另一方面，对于计算强度低于33.84的计算密集型应用，以及访存密集型应用，SW26010处理器的架构特性、访存性能要求编程人员设计数据传输优化方法，并且必须充分利用寄存器通信机制实现从核间数据共享，以降低主存带宽压力，打破访存性能瓶颈，以获得更高的性能。因此，对于大多数科学计算核心而言，基于寄存器级别通信机制的并行算法设计是充分发挥SW26010处理器性能的重要条件。

参考文献:

[1] EIJKHOUT V. Introduction to High Performance Scienti c Computing[M]. [S.l.]: Lulu. com， 2014.

[2] FU H， LIAO J，YANG J， et al. The Sunway TaihuLight supercomputer: system and applications[J]. Science China Information Sciences，2016， 59(7): 072001.

[3] AARSETH S J. N - Body Simulations[C]// Dynamics of the Solar Systems. Vol. 69. [S.l.]: [s.n.]，1975: 57.

[4] De la CRUZ R，ARAYA - POLO M. Modeling stencil computations on modern HPC ar - chitectures[C]//International Workshop on Performance Modeling， Benchmarking and Simulation of High Performance Computer Systems. Springer. [S.l.]: [s.n.]，2014: 149 - 171.

[5] SCHALLER R R. Moore ’ s law: past， present and future[J]. IEEE spectrum，1997，34(6): 52 - 59.

[6] GEPNER P，KOWALIK M F. Multi - core processors: New way to achieve high system performance[C]// Parallel Computing in Electrical Engineering， 2006. PAR ELEC 2006. International Symposium on. IEEE. [S.l.]: [s.n.]， 2006: 9 - 13.

[7] HENNESSY J L，PATTERSON D A. Computer architecture: a quantitative approach[M]. [S.l.]: Elsevier，2011.

[8] Top500 List. 2017. http://www.top500.org.

[9] KUMAR R，TULLSEN D M，JOUPPI N P，et al.Heterogeneous chip multiprocessors[J].Computer，2005，38(11): 32 - 38.

[10] Intel. Intel Xeon Phi Processor Overview. https://software.intel.com/en-us/xeon-phi/x200-processor.
[11] NSCCWX. Sunway TaihuLight Compiler user guide. 2016. http://www.nsccwx.cn/.
[12] DONGARRA J, HEROUX M A, LUSZCZEK P. HPCG benchmark: a new metric for ranking high performance computing systems[J]. Knoxville, Tennessee, 2015.
[13] ZHANG J, ZHOU C, WANG Y, et al. Extreme-scale phase eld simulations of coarsen- ing dynamics on the sunway taihulight supercomputer[C]//Proceedings of the Interna- tional Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press. [S.l.]: [s.n.], 2016: 4.
[14] YANG C, XUE W, FU H, et al. 10M-core scalable fully-implicit solver for nonhydro- static atmospheric dynamics[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press. [S.l.]: [s.n.], 2016: 6.
[15] QIAO F, ZHAO W, YIN X, et al. A highly e ective global surface wave numerical sim- ulation with ultra-high resolution[C]//Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. IEEE Press. [S.l.]: [s.n.], 2016: 5.
[16] HEINECKE A, VAIDYANATHAN K, SMELYANSKIY M, et al. Design and imple- mentation of the linpack benchmark for single and multi-node systems based on intel® xeon phi coprocessor[C]//Parallel & Distributed Processing (IPDPS), 2013 IEEE 27th International Symposium on. IEEE. [S.l.]: [s.n.], 2013: 126-137.
[17] JIN G, ENDO T, MATSUOKA S. A parallel optimization method for stencil computation on the domain that is bigger than memory capacity of GPUs[C]//Cluster Computing (CLUSTER), 2013 IEEE International Conference on. IEEE. [S.l.]: [s.n.], 2013: 1-8.
[18] WILLIAMS S, SHALF J, OLIKER L, et al. Scienti c computing kernels on the cell processor[J]. International Journal of Parallel Programming, 2007, 35(3): 263.
[19] GROPP W, LUSK E, SKJELLUM A. Using MPI: portable parallel programming with the message-passing interface[M]. Vol. 1. [S.l.]: MIT press, 1999.
[20] DAGUM L, MENON R. OpenMP: an industry standard API for shared-memory pro- gramming[J]. IEEE computational science and engineering, 1998, 5(1): 46-55.
[21] ANDREWS L P, ARIAS D, MANDALIA B D, et al. Direct memory access unit for transferring data between processor memories in multiprocessing systems. US Patent 5, 634, 099. 1997.
[22] MANO M M, KIME C R, MARTIN T, et al. Logic and computer design fundamen- tals[M]. Vol. 3. [S.l.]: Prentice Hall, 2008.
[23] FANG J, SIPS H, ZHANG L, et al. Test-driving intel xeon phi[C]//Proceedings of the 5th ACM/SPEC international conference on Performance engineering. ACM. [S.l.]: [s.n.], 2014: 137-148.
[24] AUSTIN T, LARSON E, ERNST D. SimpleScalar: An infrastructure for computer system modeling[J]. Computer, 2002, 35(2): 59-67.

有限元结构分析的层级负载均衡并行计算方法

苗新强[1, 2] 金先龙[1, 2] 丁峻宏[3]
[1]上海交通大学 机械系统与振动国家重点实验室 上海 200240
[2]上海交通大学 机械与动力工程学院 上海 200240
[3]上海超级计算中心 上海 201203
jxlong@sjtu.edu.cn

摘要：

由于性价比高、计算能力强，多核机群已经成为当今高性能计算的主流工具。然而，多核机群环境下不同的存储机制和通信延迟特点也为高效并行算法的设计带来了挑战。传统区域分解法采用直接剖分的方法实现负载均衡，即直接根据参与并行计算的处理器核的数目将结构剖分为与之相等的若干个子区域。由于多核机群单个节点内处理器核数目的成倍增加，这就导致子区域的数目会随着处理器核数目的增加而大幅度增加。子区域数目的大幅度增加一方面会导致界面方程的规模和条件数急剧扩大，从而降低系统的数值收敛性；另一方面会导致参与并行计算的进程数目大幅度增加，从而加剧对网络端口和网络带宽的竞争。系统数值收敛性的降低和网络通信开销的增加严重影响了界面方程的求解效率，并大幅度降低了区域分解法的整体并行效率。为充分利用多核机群的硬件资源获取最优性能，本文设计了一种有限元结构分析的层级负载均衡并行计算方法。该方法建立在对计算任务的层次性和粒度性充分挖掘的基础上。为与多核机群的硬件拓扑体系结构相适应，本文将有限元结构分析的计算任务划分为三个层次：节点间并行，片间并行和核间并行。其中，节点间并行和片间并行采用粗粒度并行计算方法，而核间并行采用细粒度并行计算方法。通过将计算任务映射到多核机群的不同硬件层面执行，该方法不仅有效实现了不同层面的负载均衡，而且大幅度降低了系统的通信开销。此外，它还大幅度减少了子区域的数目，有效提高了界面方程的数值收敛性。为验证算法的有效性，在“天河二号”超级计算机上进行了两个有限元结构线性静力分析大规模并行计算的数值仿真测试。每个模型都分别采用传统区域分解法和本文设计的层级负载均衡法进行求解，启动的并行计算节点数依次为：50，100，150，200。结果表明：同传统区域分解法相比，层级负载均衡并行计算方法能够获得较高的加速比和并行效率。本文算法具有广泛的通用性，它能够有效求解各种有限元结构分析问题，包括线性静力分析、线性动力分析、非线性静力分析和非线性动力分析等。在本文中，作者目前的研究主要集中在线性静力学问题上。对于非线性问题或者动力学问题，由于涉及多个迭代步求解，因此可以将本文算法封装为一个子函数进行调用。

关键词：多核机群，有限元分析，并行计算，负载均衡

直接法和迭代法是有限元结构分析并行计算的两种基本方法。直接法通过排序、三角分解和回代求解等能够给定线性系统的精确解，并且具有良好的数值稳健性[1-3]。然而，它往往需要较大的内存空间[4]。迭代法是基于试验和误差的方法，它通过多次迭代求解对结果进行改善以尝试收敛到问题的精确解[5-6]。相对直接法来说，迭代法具有所需内存空间小的优点。但迭代法不一定在合理的时间内收敛，对条件数很大的病态问题也可能是不收敛的[7]。区域分解法可以融合直接法和迭代法求解的优点，使并行计算既具有良好的数值稳健性，又具有所需内存空间小的优点[8-10]。它首先将结构有限元网格划分

资助项目：国家863高技术研究发展计划（2012AA01A307），国家重点研发计划项目（2016YFB0201800）和国家自然科学基金(11272214,51475287）资助

为若干个子区域，然后利用直接法消去每个子区域的内部自由度，再通过迭代法求解界面方程，最后根据求得的边界位移回代各子区域内部自由度。利用区域分解法进行有限元结构分析并行计算时，各子区域仅在求解界面方程的过程中需要相互通信，而其它计算过程均可在各子区域内部独立完成。因此，在传统单核处理器构建的超级计算机上，区域分解法能够获得较高的并行计算效率。

随着处理器和超级计算机体系结构的发展，多核机群日益成为大规模科学计算和工程分析的主流工具[11]。一方面多核机群通过在单个节点内集成更多的计算核心大幅度提升了系统的计算性能，另一方面它在节点内和节点间不同的存储机制和通信延迟差异也为高效并行算法的设计带来了挑战[12]。传统区域分解法在进行区域分割时没有考虑多核机群的体系结构特点，它采用直接剖分的方法实现负载均衡，即直接根据参与并行计算的处理器核的数目将结构剖分为与之相等的若干个子区域。由于多核机群单个节点内处理器核数目的成倍增加，这就导致子区域的数目会随着处理器核数目的增加而大幅度增加。子区域数目的大幅度增加一方面会导致界面方程的规模和条件数急剧扩大，从而降低系统的数值收敛性；另一方面会导致参与并行计算的进程数目大幅度增加，从而加剧对网络端口和网络带宽的竞争。此外，传统区域分解法没有考虑多核机群节点内和节点间不同的存储机制和通信延迟差异，造成系统的通信开销较大。系统数值收敛性的降低和网络通信开销的增加严重影响了界面方程的求解效率，并大幅度降低了区域分解法的整体并行效率。为充分利用多核机群的硬件资源获取最优性能，本文设计了一种有限元结构分析的层级负载均衡并行计算方法。它在区域分解法的基础上，通过将并行算法与多核机群的硬件拓扑体系结构融合在一起有有效提高了系统的并行计算效率。

1. 多核机群与区域分解法

1.1 多核机群

如图1所示，多核机群由多核计算节点通过高速网络相互连接构成[13]。它的每个计算节点都配置有两个或多个多核处理器。节点内的内存和高速缓存基于树形层次化结构进行组织，存储器的大小从叶节点到根节点逐渐增大。每个计算核心都配置有尺寸较小的二级缓存，位于相同处理器芯片上的所有计算核心共享尺寸较大的三级缓存，位于相同节点内的所有计算核心共享尺寸最大的系统主存。在各个节点间，多核机群采用分布式存储的体系结构，要访问其它节点的内存必须通过消息进行传递。可见，多核机群具有两级存储机制：节点内共享存储和节点间分布式存储。

从通信层面看，多核机群具有三层通信模式：片内通信、片间通信和节点间通信[14]。相同芯片上不同计算核心间的通信称为片内通信；相同节点内不同芯片的计算核心间的通信称为片间通信；不同节点的计算核心间的通信称为节点间通信。由于通信通道和机制的不同，多核机群上的通信延迟是非均匀的：片内通信延迟最小，片间通信延迟稍大，而节点间通信延迟最大。因此，要提高多核机群的并行计算效率就要设法减少不同模式下通信延迟差异对应用程序性能造成的影响。

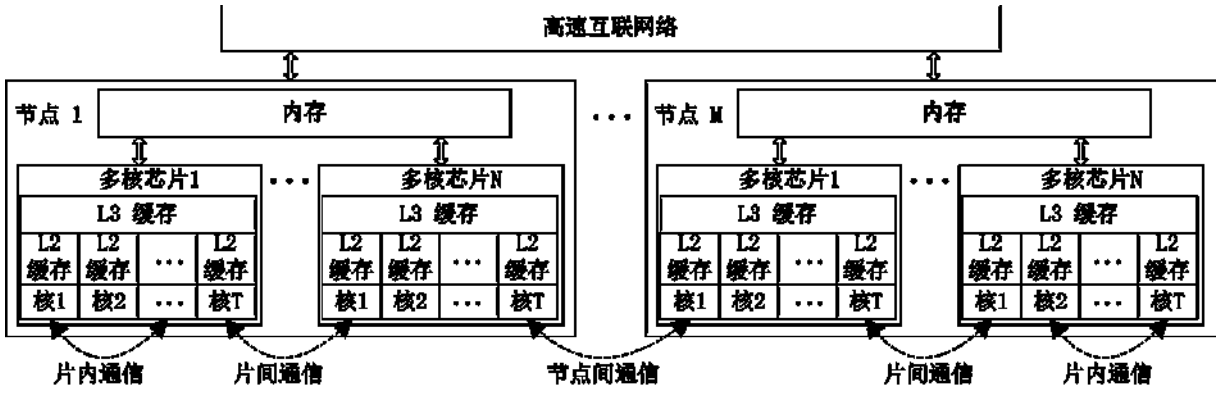


图1 多核机群体系结构

1.2 区域分解法

区域分解法是一种粗粒度的并行计算方法。它既适合运行在具有分布式存储体系结构的超级计算机上，又适合运行在具有共享存储体系结构的超级计算机上。利用区域分解法进行有限元结构分析并行计算的基本原理如下[15]：

首先将结构有限元网格划分为若干个子区域，然后按照先内部自由度后边界自由度的编号原则同

时独立形成每个子区域的系统方程:

$$\begin{bmatrix} K_{II} & K_{IB} \\ K_{BI} & K_{BB} \end{bmatrix} \begin{Bmatrix} x_I \\ x_B \end{Bmatrix} = \begin{Bmatrix} P_I \\ P_B \end{Bmatrix} \quad (1)$$

式中，xI, xB, PI, PB分别为内部自由度和边界自由度对应的位移和外部载荷向量；K**为刚度矩阵的分块矩阵，其中下标I和B分别代表内部自由度和边界自由度。

接着通过缩聚同时独立消去各子区域内部自由度，得到只含边界自由度未知量的界面方程

$$\tilde{K}x_B = \tilde{P} \quad (2)$$

式中有效刚度矩阵

$$\tilde{K} = K_{BB} - K_{BI}(K_{II})^{-1}K_{IB} \quad (3)$$

有效载荷向量

$$\tilde{P} = P_B - K_{BI}(K_{II})^{-1}P_I \quad (4)$$

然后将所有子区域的界面方程联立求解，得到各子区域边界位移xB。本文采用并行预条件共轭梯度算法求解界面方程[16]。求得边界位移后，各子区域内部位移xI可由下式回代求解

$$x_I = (K_{II})^{-1}(P_I - K_{IB}U_B) \quad (5)$$

最后同时独立计算各子区域的应变和应力，并输出计算结果。

由以上过程可见，区域分解法仅在求解界面方程的过程中需要通信，而其它计算过程如形成子区域系统方程、缩聚和回代内部自由度等均可在各子区域内部独立完成。因此，当子区域数目较少时，界面方程的求解开销较小，系统能够获得较高的并行计算效率。然而，利用多核机群进行大规模并行计算时，子区域的数目会随着处理器核数目的成倍增加而大幅度增加，从而严重影响界面方程的求解效率，并大幅度降低系统整体的并行效率。

2. 层级负载均衡并行计算方法

从硬件体系结构看，多核机群由多核、多处理器和多节点的层次性元素构成。如1.1节所述，位于不同层次的元素在数据存储和通信延迟上都有很大差异。因此，要提高多核机群的并行计算效率就要实现不同层面的负载均衡和最小化系统的通信开销。本文在区域分解法的基础上将并行算法与多核机群的硬件拓扑体系结构融合在一起，提出了一种有限元结构分析的层级负载均衡并行计算方法。

层级负载均衡并行计算方法建立在对计算任务的层次性和粒度性充分挖掘的基础上。为与多核机群的硬件拓扑体系结构相适应，本文将有限元结构分析的计算任务划分为三个层次：节点间并行，片间并行和核间并行。其中，节点间并行和片间并行采用粗粒度并行计算方法，而核间并行采用细粒度并行计算方法。如图2所示，通过将计算任务映射到多核机群的不同硬件层面执行，层级负载均衡并行计算方法不仅有效实现了不同层面的负载均衡，而且大幅度降低了系统的通信开销。此外，它还大幅度减少了子区域的数目，有效提高了界面方程的数值收敛性。因此，它能够充分利用多核机群的硬件资源获取最优性能。

2.1 节点间并行

考虑到多核机群在节点间采用分布式存储体系结构，并且节点间的通信延迟远远大于节点内的通信延迟，因此节点间适合采用粗粒度并行计算方法。这样一方面能够实现大规模数据的分布式存储，

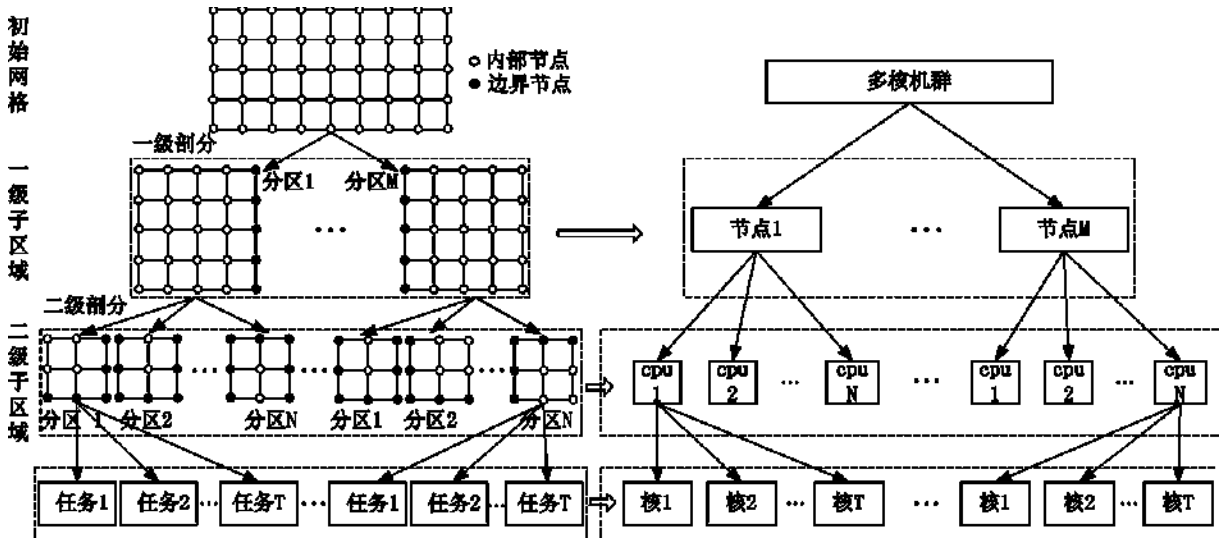


图2 任务映射

另一方面能够有效减少系统的通信开销。

如图2所示，节点间并行基于区域分解法实现：首先将结构有限元网格均等剖分为M个一级子区域，然后将每个一级子区域分别分配给一个节点机单独处理。其中，M为参与并行计算的节点机总数。由于各一级子区域的规模都相同，并且各节点机具有相同的计算性能，这就实现了节点间的负载均衡和并行计算。

2.2 片间并行

为提高数据访问速度，现代超级计算机一般都配置了二级和三级缓存。由于位于不同处理器芯片上的三级缓存是相对独立的，并且片间通信开销比片内通信开销稍大，因此片间也适合采用粗粒度并行计算方法。为尽量将不同处理器芯片间的通信限制在同一节点内，应将相邻的子区域分配在一起。

如图2所示，片间并行建立在上文一级分区的基础上：首先将每个一级子区域进一步均等剖分为N个二级子区域，然后将派生于相同一级子区域的各二级子区域分别分配给相同节点的不同处理器芯片单独处理。其中，N为单个节点机内处理器芯片的总数。由于位于相同节点的各二级子区域均由同一个一级子区域派生而来，这就将相邻的子区域分配在了一起，从而将不同处理器芯片间的通信尽量限制在了同一节点内。

通过两级分区就将直接参与并行计算的子区域的数目降低为传统区域分解法的1/T。其中，T为单个处理器芯片上的计算核心总数。子区域数目的大幅度降低一方面有利于减少界面方程的规模和条件数，从而有效提高系统的数值收敛性；另一方面有利于大幅度减少参与并行计算的进程数目，从而减少通信对有限的网络端口和带宽的竞争，进而提高系统的通信效率。

2.3 核间并行

随着多核处理器的发展，在单个芯片上集成的计算核心数目也越来越多。若采用粗粒度并行计算方法利用这些多核资源，难免会面临因分区过多而导致系统数值收敛性降低和通信开销增加的问题。因此，核间并行适合采用细粒度并行计算方法。这样一方面有利于提高系统的数值收敛性，另一方面可以利用单个芯片上所有计算核心共享三级缓存的特点有效提高数据的访问速度和通信效率。

如图2所示，核间并行涉及到最底层的计算模块，是对每个二级子区域计算任务的进一步分解。基于区域分解法进行有限元结构分析并行计算的主要步骤包括：组集子区域系统平衡方程、缩聚、求解界面方程、回代内部自由度以及计算子区域应变和应力。核间并行要做的工作就是针对二级子区域的每个计算步骤寻找出里面的热点计算程序，即大的循环结构，将其分解为大量互不相干且能够独立执行的子任务，然后将每个子任务分配给多核处理器的一个计算核心去执行。下面以缩聚为例介绍核间并行的实现：

缩聚是利用区域分解法进行并行计算的一个核心环节，它通常需要消耗大量的计算时间。为了充分利用多核资源缩短计算时间，本文将缩聚计算转换为一系列相互独立的线性方程组的求解过程，并通过OPenMP进行并行化处理以将每个子任务分配给多核处理器的一个计算核心去执行。

由式（3）可见，缩聚刚度矩阵 $\tilde{K}$ 的计算需要用到 $(K_{II})^{-1}K_{IB}$ 的结果。为了得到这两个矩阵相乘的每一列的值，可将 K_{IB} 的元素逐列提取出来并赋值给向量b，然后再通过稀疏直接求解器依次求解线性方程组 $K_{II}t_1=\bar{b}$ 的解。在t1的基础上，$K_{BI}(K_{II})^{-1}K_{IB}$ 每一列的结果可通过计算 $t_2=K_{BI}t_1$ 得到。最后，将 K_{BB} 相应列的元素提取出来并与t2相减就可得到缩聚刚度矩阵 $\tilde{K}$ 对应列的计算结果。考虑到缩聚刚度矩阵的列数与子区域边界自由度的个数z相等，因此上述操作过程共需重复执行z次。对每次操作来说，它都是一个能够独立执行的子任务，因此可以通过OPenMP实现对这些子任务的并行化处理。同理，缩聚外部载荷向量 $\tilde{P}$ 也可采取同样的策略求解。缩聚外部载荷向量只有一列，在进程里操作一次即可。本文设计的缩聚算法的伪代码如图3所示，图中2～9行为通过OPenMP利用多核资源进行并行计算的部分。

1.//calculate the condensed stiffness matrix
*2.**!$omp parallel do***
*3.**do** i=1,z*
4. $b=K_{IB}(:,i)$
5. solve $K_{II}t_1=\bar{b}$ with a direct sparse solver
6. compute $t_2=K_{BI}t_1$
7. compute $\tilde{K}(:,i)=K_{BB}(:,i)-t_2$
*8.**end do***
*9.**!$omp end parallel do***
10. //calculate the condensed load vector
11.b=P_I
12. solve $K_{II}t_1=\bar{b}$ with a direct sparse solver
13. compute $t_2=K_{BI}t_1$
14. compute $\tilde{P}=P_B-t_2$

15.//solve interface equations
16. solve $\tilde{K}x_B = \tilde{P}$ with the parallel PCG *algorithm*
17. //calculate internal degrees of freedom
18. $b = P_I - K_{IB}x_B$
19. solve $K_I x_I = b$ with a direct sparse solver

图3 缩聚算法

图中，$K_{IB}(:,i)$，$\tilde{K}(:,i)$ 和 $K_{BB}(:,i)$ 分别表示相应矩阵的第i列元素构成的列向量；z为二级子区域边界自由度的个数；b，t_1，t_2均为临时列向量。

2.4 层级负载均衡并行计算流程

有限元结构分析的层级负载均衡并行计算方法在编程模式上采用MPI+OPenMP混合编程模式。整个并行计算的顶层基于MPI进程构建，其中每个MPI进程负责控制一个二级子区域的处理。并行计算的底层基于OPenMP线程实现，通过在每个MPI进程内部派生大量线程以充分发挥多核处理器高度并发计算的优势。如图4所示，有限元结构分析的层级负载均衡并行计算流程为：

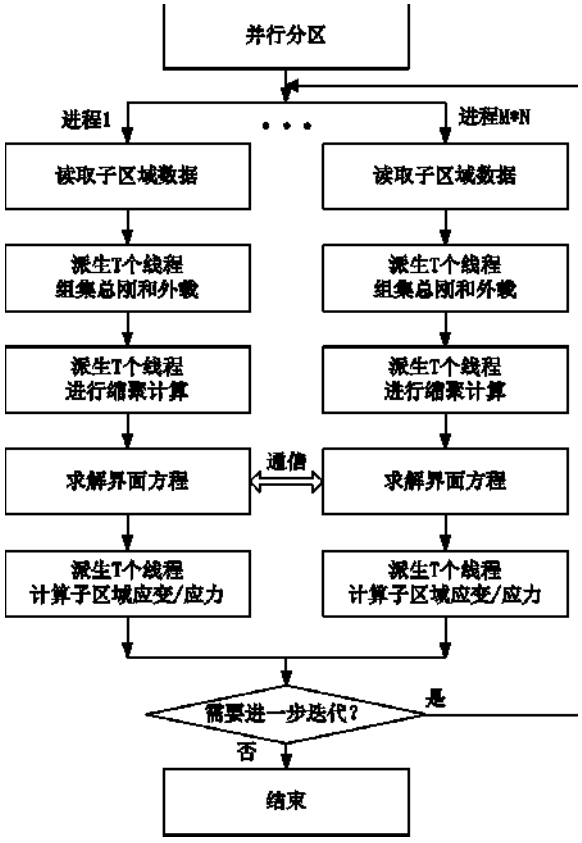


图4 层级负载均衡并行计算流程

第一步，通过两级分区生成并行计算所需的数据文件，包括各子区域的单元、节点、载荷、边界条件和相邻分区信息等。

第二步，同时在每个节点机内启动N个MPI进程，其中每个进程负责一个二级子区域数据文件的读入。

第三步，每个MPI进程内部分别派生出T个线程，利用多核资源完成相应子区域总体刚度矩阵和外部载荷向量的计算。其中，T为单个处理器芯片上的计算核心总数。

第四步，每个MPI进程内部分别派生出T个线程，利用多核资源完成缩聚计算。

第五步，各进程共同利用并行预条件共轭梯度算法求解界面方程，得到各子区域边界节点位移后再回代求解内部位移。

第六步，每个MPI进程内部分别派生出T个线程，利用多核资源计算相应子区域应变/应力。

第七步、若需要进一步迭代则跳转至第二步重新开始执行，否则结束。

综上所述，层级负载均衡并行计算方法一方面通过两级分区大幅度降低了子区域的数目，有效提高了界面方程的数值收敛性；另一方面，它通过将计算任务映射到多核机群的不同硬件层面执行，不仅有效实现了不同层面的负载均衡，而且大幅度降低了系统的通信开销。因此，它能够充分利用多核机群的硬件资源有效提高有限元结构分析大规模并行计算的效率。

3. 数值算例

为了验证本文算法的有效性，作者在广州超算中心的“天河二号”超级计算机上进行了测试。并行算法评估的数值模型如图5和图6所示。图5为左端固定，右端受竖直向下载荷作用的悬臂梁计算模型。梁长5m，宽0.6m，高0.8m。采用四面体单元进行网格剖分后，该模型具有35,898,476单元，6,846,069节点，20,538,207自由度。

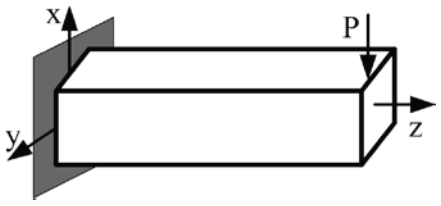


图5 悬臂梁计算模型

图6为某拱桥计算模型，主要分析该模型在重力和车载作用下的变形和应力。模型总长16m，总宽10m，总高5.4m。采用四面体单元进行网格剖分后，该模型具有44,081,196单元，8,461,996节点，25,385,988自由度。

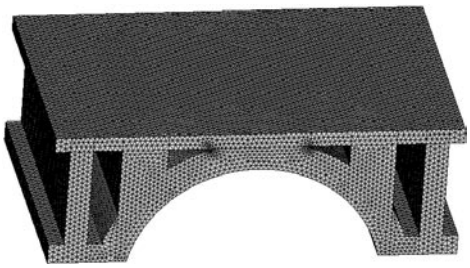


图6 拱桥计算模型

3.1 并行计算环境

“天河二号”超级计算机采用目前国际主流的机群架构构建。它由16000个计算节点组成，每个节点配置2颗基于Ive Bridge-E Xeon E5 2692多核处理器芯片和64 GB内存。每颗多核处理器芯片具有12个计算核心，运作时钟频率为2.2GHZ，峰值性能为0.2112TFLOPS。每个计算核心都配置有独立的256KB二级缓存，位于相同处理器芯片上的所有计算核心共享30 MB三级缓存。在节点内不同处理器核间通过片上网络或高速缓存互联，通信速度很快。而各节点间采用TH Express-2主干拓扑结构网络联接，通信速度相对较慢。

3.2 计算数据准备

并行计算前首先需要通过两级分区准备数据。根据前面2.1和2.2节的内容知，为与多核机群的硬件拓扑体系结构相适应两级分区时一级子区域总数应等于每次启动的节点机总数，并且每个一级子区域的派生二级子区域总数应等于单个节点内处理器芯片的总数。本次测试中，每次启动的节点机总数依次为：50，100，150，200，因此经一级剖分得到的一级子区域总数也应依次为：50，100，150，200。由于“天河二号”超级计算机每个节点内配置2颗多核处理器芯片，因此二级剖分时每个一级子区域将被进一步剖分为2个二级子区域。

3.3 并行算法性能评估

对上述各计算模型分别采用传统区域分解法和本文层级负载均衡并行计算方法进行求解，得到的测试结果如表1和表2所示。表中，并行计算总时间从读取数据文件开始到计算各子区域应变/应力结束；缩聚时间指消去各子区域内部自由度的时间；求解界面方程时间指利用并行预条件共轭梯度算法求得各子区域边界位移的时间。

表1 悬臂梁模型并行计算时间和性能统计

核数	传统区域分解法					层级负载均衡并行计算法				
	缩聚 (s)	求解界面方程 (s)	总时间 (s)	加速比	并行效率	缩聚 (s)	求解界面方程 (s)	总时间 (s)	加速比	并行效率
1200	112	714	837	1	100%	565	171	809	1	100%
2400	30	951	986	0.85	42.44%	218	165	417	1.94	97.00%
3600	11	1159	1174	0.71	23.76%	123	176	316	2.56	85.34%
4800	6	1260	1268	0.66	16.50%	84	187	281	2.88	71.98%

表2 拱桥模型并行计算时间和性能统计

核数	传统区域分解法					层级负载均衡并行计算法				
	缩聚 (s)	求解界面方程 (s)	总时间 (s)	加速比	并行效率	缩聚 (s)	求解界面方程 (s)	总时间 (s)	加速比	并行效率
1200	153	1373	1535	1	100%	895	318	1303	1	100%
2400	40	1631	1677	0.92	45.77%	332	286	659	1.98	98.86%
3600	18	2053	2074	0.74	24.67%	172	294	491	2.65	88.46%
4800	9	2202	2213	0.69	17.34%	95	321	428	3.04	76.11%

由表1和表2可见，相对传统区域分解法本文提出的层级负载均衡并行计算方法能够获得较高的加速比和并行效率。采用传统区域分解法利用大量处理器内核进行并行计算时，随着子区域数目的增多界面方程求解时间也大幅度增加，从而严重影响了系统总体的并行效率。例如在表2中，核数由1200增加到4800的过程中，传统区域分解法的界面方程求解时间由1373s迅速增加到了2202s，增加幅度高达829s。这就在很大程度上直接造成了系统总体并行效率由100%下跌到了17.34%。界面方程求解时间的大幅度增加主要由两个因素导致：首先随着子区域数目的大幅度增加，界面方程的规模和条件数也急剧增大从而降低了系统的数值收敛性；其次，传统区域分解法没有考虑多核机群节点内和节点间不同的存储机制和通信延迟差异，造成系统的通信开销较大。

采用本文层级负载均衡并行计算方法进行有限元结构分析大规模并行计算时能够大幅度缩减界面方程求解时间，并取得良好的加速比和并行效率。例如在表2中，核数为3600时传统区域分解法和层级负载均衡并行计算方法的界面方程求解时间分别为2053s和294s，后者比前者减少了1759s。此时，传统区域分解法的加速比和并行效率仅为0.74和24.67%，而层级负载均衡并行计算方法的加速比和并行效率则高达2.65和88.46%。这主要归功于两个方面的因素。第一，同传统区域分解法相比，本文设计的两级分区方法大幅度减少了子区域的数目。子区域数目的大幅度降低一方面有利于减少界面方程的规模和条件数，从而有效提高系统的数值收敛性；另一方面有利于大幅度减少参与并行计算的进程数目，从而减少通信对有限的网络端口和带宽的竞争，进而提高系统的通信效率。第二，本文算法通过将计算任务映射到多核机群的不同硬件层面执行不但有效实现了不同层面的负载均衡，而且大幅度降低了系统的通信开销。因此，层级负载均衡并行计算方法能够大幅度减少界面方程求解时间以及系统求解总时间，从而有效提高系统总体的并行计算效率。

本文算法具有广泛的通用性，它能够有效求解各种有限元结构分析问题，包括线性静力分析、线性动力分析、非线性静力分析和非线性动力分析等。在本文中，作者目前的研究主要集中在线性弹性力学问题分析上。对于非线性问题，由于涉及多个迭代步求解，因此可以将本文算法封装为一个子函数进行调用。在每个迭代步的求解过程中，同传统区域分解法相比本文算法都能大幅度减少界面方程求解时间，有效提高系统的加速比和并行效率。因此，经过多个迭代步求解之后，本文算法累计削减的界面方程求解时间会更加明显，本文算法相对传统区域分解法的优势也将更加突出。作者下一步的工作将致力于将本文算法推广应用到求解非线性问题上。

4. 结论

随着处理器和超级计算机体系结构的发展，多核机群日益成为大规模科学计算和工程分析的主流工具。然而，多核机群节点内和节点间不同的存储机制和通信延迟差异为高效并行算法的设计带来了挑战。本文根据多核机群的硬件拓扑体系结构特点，设计了一种有限元结构分析的层级负载均衡并行计算方法以提高系统的并行效率。

该方法将计算任务划分为三个层次以与多核机群的硬件拓扑体系结构特点相适应：节点间并行，片间并行和核间并行。其中，节点间并行和片间并行采用粗粒度并行计算方法，而核间并行采用细粒度并行计算方法。通过将计算任务映射到多核机群的不同硬件层面执行，层级负载均衡并行计算方法不仅有效实现了不同层面的负载均衡，而且大幅度降低了系统的通信开销。此外，它还大幅度减少了子区域的数目，有效提高了界面方程的数值收敛性。为验证算法的有效性，在“天河二号”超级计算机上进行了有限元结构分析大规模并行计算的测试。结果表明：同传统区域分解法相比，层级负载均衡并行计算方法能够获得较高的加速比和并行效率。

参考文献：

[1] Paszynski M, Pardo D, Calo V M. A direct solver with reutilization of LU factorizations for h - adaptive finite element grids with point singularities. Comput Math Appl. 2013,65: 1140 - 1151

[2] Wang S, Xia J L, de Hoop M V, et al. Massively parallel structured direct solver for equations describing time - harmonic qP - polarized waves in TTI media. Geophysics. 2012,77: 69 - 82

[3] Gupta A. A shared - and distributed - memory parallel general sparse direct solver. Algebra Eng Commun Comput, 2007. 18: 263 - 277

[4] Paszyński M, Pardo D, Torres - Verdín C, et al. A parallel direct solver for the self - adaptive hp finite element method. J Parallel Distr Com. 2010,70: 270 - 281

[5] Ullmann E, Elman H, Ernst O. Efficient iterative solvers for stochastic galerkin discretizations of log - transformed random diffusion problems. SIAM J Sci Comput, 2012. 34: 659 - 682

[6] Yao J. An easy method to accelerate an iterative algebraic equation solver. J Comput Phys. 2014,267: 139 - 145

[7] Mitin I, Kalinkin A, Laevsky Y. A parallel iterative solver for positive - definite systems with hybrid MPI - OpenMP parallelization for multi - core clusters. J Comput Sci, 2012. 3: 463 - 468

[8] 王建炜，金先龙，曹源. 列车与结构动态耦合分析的并行计算方法. 计算力学学报，2012，29：352 - 356

[9] Xicheng W, Baggio P, Schrefler B. A multi-level frontal algorithm for finite element analysis and its implementation on parallel computation. Eng Computation, 1999, 16: 405-427
[10] Yang Y S, Hsieh S H, Asce M, et al. Improving parallel substructuring efficiency by using a multilevel approach. J Comput Civil Eng, 2012, 26: 457-464
[11] Zavala-D í az J C, P é rez-Ortega J, Salazar-Res é ndiz E, et al. Matrix multiplication with a hypercube algorithm on multi-core processor cluster. Dyna. 2015, 82: 240-246
[12] Dongarra J, Faverge M, Herault T, et al. Hierarchical QR factorization algorithms for multi-core clusters. Parallel Comput, 2013. 39: 212-232
[13] Seelam S, Fong L, Tantawi A, et al. Extreme scale computing: Modeling the impact of system noise in multi-core clustered systems. J Parallel Distr Com. 2013, 73: 898-910
[14] Hamid N, Walters R, Wills G. Understanding the impact of the interconnection network performance of multi-core cluster architectures. J Comput, 2016. 11: 132-139
[15] Antoine X, Lorin E, Bandrauk A D. Domain decomposition method and high-order absorbing boundary conditions for the numerical simulation of the time dependent schrödinger equation with ionization and recombination by intense electric field. J Sci Comput. 2015, 64: 620-646
[16] Rao A. MPI-based parallel finite element approaches for implicit nonlinear dynamic analysis employing sparse PCG solvers. Adv Eng Softw. 2005, 36: 181-198

要闻集锦

美能源部投资千万进行量子计算研究

据www.hpcwire.com网站2017年10月20日消息报道，近日，美国能源部科学办公室向两个研究小组投资超过1000万美元，进行为期五年的量子计算研究，旨在评估量子架构的可行性，进而开发适用于量子计算系统的算法。这两个研究小组都是由橡树岭国家实验室（ORNL）的量子信息科学组领导的，一个将负责用于科学应用的量子加速方法研究，这也是更大型量子计算试验开创项目的一部分；另一个获得了750万美元的投资，将与IBM、IonQ、乔治亚工学院和弗吉尼亚工学院合作，旨在对量子架构的应用性能开展为期五年的研究。

ORNL将聚焦科学应用的三个领域的研究：量子场理论、量子化学和量子机器学习。第一个小组的工作是确定能否利用现有的或不久后更新的量子硬件运行想要的量子应用程序。很多应用从未在量子架构上运行过，因此面临很大的挑战。但现有的量子计算机规模相对很小，应用必须要适应硬件，这样才能最大化性能和准确度。这就需要深度理解独特的量子程序，需要在不同的量子架构上运行程序来评估其有效性，并最终确定可行性。

第二个小组将致力于算法设计方法，将参考数字和模拟量子计算的最优特点，最终将匹配量子模拟算法的复杂性，用于量子架构。同传统计算平台相比，由于量子硬件的开发和部署是新兴领域，因此研究小组将利用异构量子系统的方案，采用量子计算机和传统处理器相结合的方法。该小组的研究人员集合了计算机、应用数学、科学应用、量子计算领域的科学家共同攻关。量子模拟算法汇集了量子和传统计算的不同特点，掌握好二者的平衡才能实现超越传统方式的新的科学发展。

（肖 湄）

绕组损耗分布对油浸式变压器温升的影响分析

● 李德波 冯永新
广东电网有限责任公司电力科学研究院 广州 510060

摘要：

在绕组温升计算过程中，绕组损耗分布直接影响油流和温度分布。为了更准确地研究油浸式变压器温度场分布，建立了一台油浸式电力变压器的低压绕组二维流体场-温度场计算模型，对绕组热源的处理：用每饼的欧姆损耗和涡流损耗代替传统的整体平均损耗，并且采用Fluent UDF函数在每次迭代时动态地校正温度依赖变量。同时对比分析了损耗温度效应、涡流损耗以及热源分布对绕组油道中油流速度及温度分布的影响。分析结果表明：考虑损耗的温度效应后，各个分区和整体绕组的热点均发生上移，整体绕组的热点温度增加24.8K，热点位置偏差为7个线饼；考虑涡流损耗使得各个线饼温度增大，尤其在各个分区的局部热点附近，温度的增加更加显著；热源分布对绕组温度分布的影响主要集中在绕组端部。

关键词：油浸式变压器，饼式绕组，Fluent UDF，损耗，温度分布；

引言

油浸式电力变压器是电网中能量转换、传输的重要的电力设备，其故障将会给电力系统带来重大经济损失[1-3]。电力变压器的预期寿命主要取决于绕组的绝缘老化，而影响变压器绝缘老化率的主要因素是绕组温度。因此，准确地分析和计算绕组温升对于变压器的设计和绝缘寿命的预测十分重要[4-9]。

近年来，随着计算能力的不断提高，研究人员和工程师对电力变压器的热行为进行了大量的数值模拟[10-22]。谢裕清等人[10]分别采用最小二乘有限元和迎风有限元计算油流速度场和整个场域的温度场提出了一种计算绕组温升的多物理场耦合有限元计算方法。许永明等人[15]根据油路结构特点和绕组冷却分析，分别采用CFD方法和流体网络方法计算流场和绕组温度，提出了一种基于流体网络的绕组温度计算方法。Kranenborg等人[18]对饼式变压器绕组进行了2维计算，研究了冷却模式对流量和温度分布的影响。Lee等人[19]通过研究确定了用于OD和ON冷却模式的饼式变压器绕组中的热传递的相关性。在绕组温升计算过程中，绕组损耗分布直接影响油流和温度分布。然而，在以上研究中，对绕组温度场的热源大都采用平均热源的形式且进行了不同程度的简化。如，忽略涡流损耗[10-12]；采用损耗占比计算涡流损耗[13]；线饼表面热负荷以匝数最多的计算[14]；在温度场的迭代求解过程中忽略了欧姆损耗或涡流损耗随温度的非线性变化等[15-17]。热源的简化和整体平均化处理虽然给温度场仿真计算带来了方便，但是可能会使仿真得到的局部热点与实际有所偏差。

基于上述情况，本文根据一台油浸式电力变压器的低压绕组结构，建立了该变压器二维流体场-温度场计算模型。在绕组温升计算过程中，考虑变压器油物性参数随温度变化的函数关系式；对绕组热源的处理，用每饼的欧姆损耗和涡流损耗代替传统的整体平均损耗，并且考虑温度场迭代过程中损耗随温度的非线性变化。同时本文还对比分析了损耗温度效应、涡流损耗以及热源分布对绕组油道中油流速度及温度分布的影响，讨论了绕组损耗分布对变压器温升的影响机理，为工程应用中更准确地定位绕组热点温度提供参考。

1. 控制方程

在油浸式电力变压器运行过程中，由于绕组和油流之间存在温度梯度，绕组中的热量通过热传导和对流换热传递至循环的油中，然后通过油流将热量输运至外界空气中。整个计算场域满足质量、动量和能量守恒定律。在实际工程计算中，常将变压

基金项目：中国南方电网科研项目资助（GDKJQQ20153004）

器油近似等效为不可压缩流体，控制方程式为：

$$\frac{\partial \rho}{\partial t}+\nabla\cdot(\rho \boldsymbol{U})=0, \tag{1}$$

$$\frac{\partial(\rho \boldsymbol{U})}{\partial t}+\nabla\cdot(\rho \boldsymbol{U}\times\boldsymbol{U})= -\nabla p+\mu(\nabla^2\boldsymbol{U})+g(\rho-\rho_{ref}), \tag{2}$$

$$\frac{\partial(\rho c_p T)}{\partial t}+\nabla\cdot(\rho c_p \boldsymbol{U}T)= \nabla\cdot(k\nabla T)+S_E. \tag{3}$$

式中U为流体的速度矢量，ρ为流体密度，T为油流温度，c_p为定压比热容，κ为热传导系数，S_E为热源。方程式（2）的右边项分别是压力，粘性力和浮升力。式（3）为变压器油中的对流扩散传热方程。

为了确定绕组线饼内的温度场，需求解固体中的热传导方程：

$$\frac{\partial(\rho c_p T)}{\partial t}=\nabla\cdot(k\nabla T)+S_E, \tag{4}$$

其中方程式右端源项SE为体积欧姆损耗和体积涡流损耗的总和。由于漏磁场分布的不均匀性导致各个线饼上产生的涡流损耗不同。同时考虑各线饼损耗分布的不均匀和损耗随温度的非线性变化，在每次迭代时使用以下公式动态地校正每个线饼的损耗值[18]：

$$\frac{P_{ohmic}}{P_{0\,ohmic}}=\frac{T_{avg\,disc}+T_k}{T_0+T_k}, \tag{5}$$

$$\frac{P_{eddy}}{P_{0\,eddy}}=\frac{T_0+T_k}{T_{avg\,disc}+T_k}. \tag{6}$$

式中$P_{0\ ohmic}$为绕组在温度为T_0时的欧姆损耗，这里取75℃，$T_{avg\ disc}$为绕组线饼的平均温度，T_k为电阻随温度变化的温度因子，这里取234.5℃；$P_{0\ eddy}$为绕组在温度为T_0时的涡流损耗。

在计算场域稳态温度过程中，Fluent求解器在物性参数、边界条件等初始化之后就对网格节点进行求解。通过有限体积法对控制方程离散，把微分方程转化代数方程组，然后迭代求解各网格节点上的函数值，最终得到稳态情况下油流的速度分布及整个场域的温度分布。

2. 计算模型

2.1 物理模型

本文根据一台油浸式电力变压器的低压绕组结构，建立了该变压器二维流体场-温度场计算模型。在建立低压绕组温升计算模型时，对模型进行如下假设：

1）忽略绕组底部的一、二号线饼，油流从外部轴向油道流入；

2）忽略各线饼扁导线间的匝绝缘，每个线饼均按铜扁线处理。

变压器绕组温升计算模型如图1所示。低压绕组包含78个线饼，构成4个分区。第一分区包含21个线饼，二号线饼和三号线饼之间存在额外的垫圈，其他三个分区各包含19个线饼。模型总尺寸为。每一个线饼的宽为50.8mm，高为15.0mm。绕组之间的水平油道高度为4.1mm，内外部轴向油道宽度分别为8.9mm和6.4mm。油流从第一分区的内部和外部轴向油道流入绕组，但是在流经一、二号线饼之后，由于油垫圈的存在，导致油流被重定向到外部油道。每个分区的油流入口为上一个分区的油流出口，油流出口为下一个分区的油流入口。

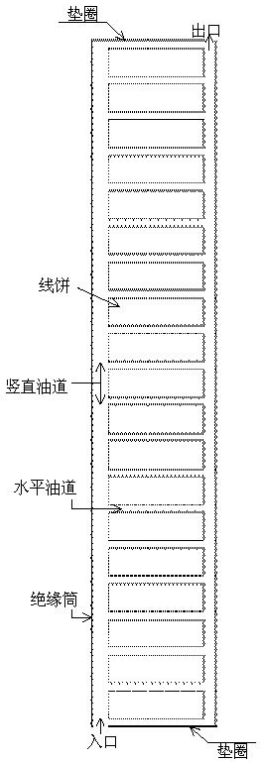


图1 低压绕组温升计算模型（第二分区）

2.2 物性参数及边界条件

油浸式变压器的低压绕组温升计算结构包括变压器油、线饼、内外绝缘筒及油垫圈几种材料，变压器油的物性参数随温度的变化而变化，在每次迭代时需对物性参数进行动态修正；线饼由绝缘纸包裹的铜扁线制成，这些材料的物性参数的设置如表1所示[23]。

图1所示的绕组计算模型为整体绕组模型的一

部分，由于绝缘筒和油垫圈由厚纸板构成，热导率非常低，因此这些壁面可近似为绝热边界条件，速度边界条件为不滑移边界条件。线饼和油流交界面设置为耦合边界条件。绕组入口为速度入口边界条件，油流入口流速为0.069m/s,方向垂直于入口平面，入口温度为300K。出口为压力出口边界条件,平均压力为0Pa。

表1 变压器物性参数

材料	参数	参数值
变压器油	密度/(kg · m^{-3})	$1098.72-7.101\times10^{-5}T_{oil}+5.0\times10^{-7}T_{oil}^{2}$
	定压比热容c_p/[J · (kg · K)$^{-1}$]	$807.163+3.58T_{oil}$
	导热系数/[W · (m · K)-1]	$0.1509-7.101\cdot10^{-5}T_{oil}$
	动力粘度系数/(Pa · s)	$0.08467-4.0\times10^{-4}T_{oil}+5.0\times10^{-7}T_{oil}^{2}$
扁线	密度/(kg · m-3)	8933
	定压比热容c_p/[J · (kg · K)-1]	385
	导热系数/[W · (m · K)-1]	401
绝缘纸及垫圈	密度/(kg · m-3)	930
	定压比热容c_p/[J · (kg · K)-1]	1340
	导热系数/[W · (m · K)-1]	0.19

对于各个线饼的热源，本文分别采用平均热源和非平均热源两种形式。各个线饼的欧姆损耗和涡流损耗[19]如下图2所示。由图2可知绕组两端线饼的涡流损耗较大，中间线饼的涡流损耗较小。各线饼的单位体积热源可以通过下式计算：

$$V_{each}=\mathrm{p}\cdot(R_{out}^{2}-R_{in}^{2}).H_{disc}, \tag{7}$$

$$Q=\frac{Q_{total}}{V_{total}}=\frac{Q_{total}}{76\cdot V_{each}}. \tag{8}$$

式中，Rin和Rout分别为线饼的内径和外径，Hdisc为线饼的高度，Veach为每个线饼的体积，Qtotal为低压绕组的欧姆损耗和涡流损耗的总和，Vtotal为所有线饼的总体积，Q为平均热源密度。

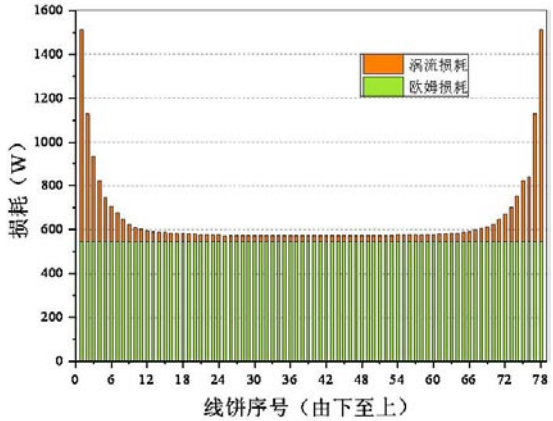


图2 欧姆损耗和涡流损耗分布

对于非平均热源的处理，每个线饼的单位体积损耗包含单位体积欧姆损耗和单位体积涡流损耗，考虑温度场迭代过程中损耗随温度的非线性变化，并且由式（5），（6）可知，两种损耗随温度变化的关系式是不同的，因此，每个线饼的单位体积欧姆损耗和单位体积涡流损耗需分离并单独施加。

2.3 网格划分

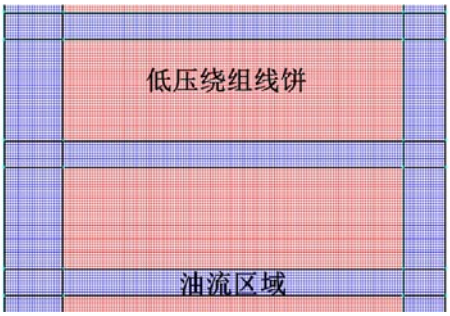
模型网格划分直接关系到温度场仿真结果的准确与否。在网格划分过程中考虑以下因素：

1）流体与固体交界面处的温度梯度变化较大，该处网格应适当加密；

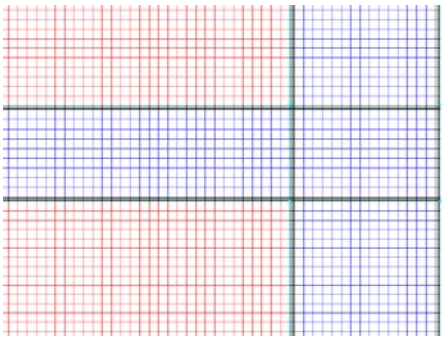
2）对网格独立性进行验证，找出热点温度和热点位置不再变化的网格数量。

3）在满足计算机内存的情况下，综合考虑网格数量和计算时间之间的平衡；

采用Gambit对模型进行网格划分，流体与固体交界处的壁面采用边界层网格，部分区域的网格如图3所示。取第二分区进行网格独立性验证，采用2.2节所述的边界条件，网格独立性验证结果如表2所示。



（a）网格划分图



（b）局部放大图

图3 局部网格划分图

表2 网格独立性验证

网格节点数	热点温度/K	热点位置
5万	发散	发散
8.8万	发散	发散
10万	330.1411	线饼12
12万	329.9477	线饼12
29万	329.9477	线饼12
36万	329.0704	线饼12
50万	328.9213	线饼12
74万	328.8616	线饼12
125万	328.828	线饼12

从表2的结果可以看出，当网格节点数量小于9万时，计算是发散的；当网格节点数量大于10万，随着网格数目的增加，热点温度变化很小，几乎不受网格数目的影响，热点位置均出现在线饼12。

综合考虑网格数量和计算时间之间的平衡，本文采用每个分区50万节点的网格剖分方案，整个低压绕组网格节点总数为200万。

2.4 计算方法

本文通过Fluent软件求解控制方程，考虑油流浮升力的作用，将油流的重力加速度设置为9.81m/s^2，方向沿y轴负方向。由于冷却通道内的流速较低，油粘度高，所以流态为层流。控制体表面的对流和扩散通量均采用二阶迎风离散化方案，温度场求解时将压力-速度耦合方式设置为SIMPLE算法，求解模式为稳态求解，所有变量均采用双精度模式求解，且规定收敛残差为10^{-6}。

为了确定绕组每个线饼内的温度分布，因此必须在流体和固体域之间进行耦合。Fluent求解器自动生成流固耦合边界，同时根据绕组结构特点和周围流体特征，计算出任意时刻耦合边界的综合传热系数。对于温度依赖的变压器油物性参数以及非平均热源的各线饼损耗，采用Fluent User defined Function (UDF)函数在每次迭代时动态地校正,整个仿真计算流程如图3所示。

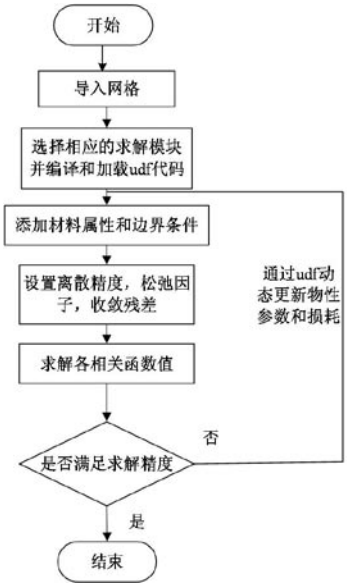


图4 仿真计算流程图

3. 仿真结果与分析

3.1 损耗的温度效应对油流及温度分布影响

应用Fluent软件计算该绕组模型的速度及温度分布。对于平均热源，保证所有边界条件和几何参数相同，采用2.2节所述的边界条件，分别对考虑温度场迭代过程中损耗随温度的非线性和忽略温度对损耗影响两种情况进行仿真，研究损耗的温度效应对绕组温度分布的影响,仿真结果如图5、6所示。

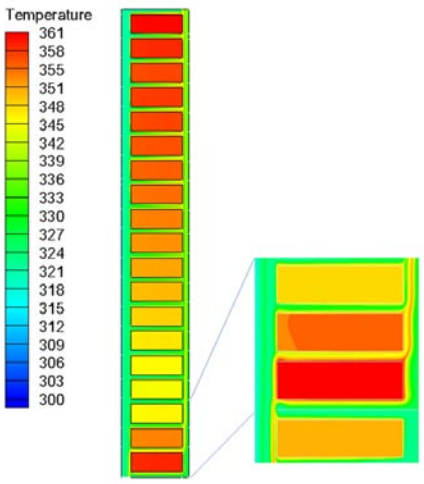
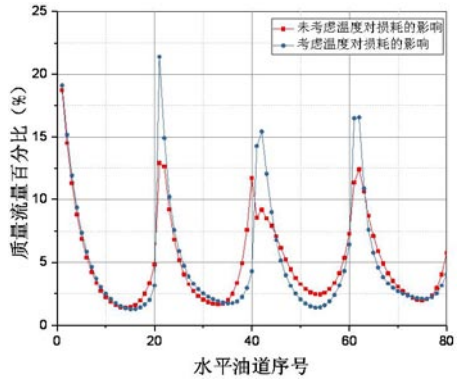


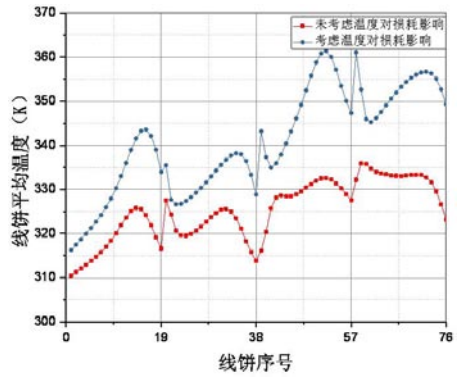
图5 考虑损耗温度效应下绕组温度云图（第四分区）

从图6（a）中可以看出，两种情况下，水平油道的质量流量分布变化趋势在一、二分区基本一致，三、四分区存在差异。在每个分区内，各水平油道内的质量流量由下至上呈现先减后增的趋势，局部最小质量流量发生在每个分区的中上部水平油道，这是由于在计算过程中考虑了油流的浮升力的作用；考虑温度效应的每个分区的最小质量流量低

于未考虑下的，而最大质量流量高于未考虑下的。



（a）水平油道质量流量分布



（b）各线饼平均温度

图6 损耗温度效应对油流及温度分布影响

相应地，从图6（b）中 ，从整个绕组模型的温度分布来看，绕组温度分布是不均匀的，下部温度低，上部温度高。对于各个绕组分区，各分区内线饼平均温度整体呈现先增后减的趋势，局部最热点与局部质量流量最小点基本一致，出现在每个分区的中上部分。由于三、四分区质量流量分布的差异直接导致三四分区的温度变化趋势的不同。二、三和四分区的绕组温度分布呈现“ Z形 ”轮廓，考虑了温度效应下更为明显。

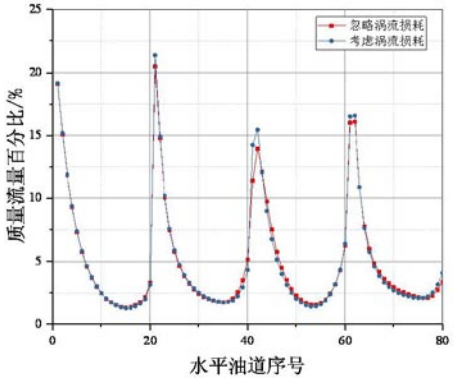
从图5局部放大图中可以明显观察到热条纹从上一分区的出口被吸入下一分区的第二、三水平油道。热条纹在水平油道中对流，且入口处的质量流量较大，大量的热油流造成两个底部线饼的局部过热。考虑损耗的温度效应后，各个线饼的平均温度值相差很大，最大差值为28.5K。

结合表3，可以看出，考虑损耗的温度效应后，各个分区和整体绕组的热点均发生上移，整体绕组的热点温度增加24.8K，热点位置偏差为7个线饼。因此，损耗的温度效应对绕组内部的温度分布影响很大。

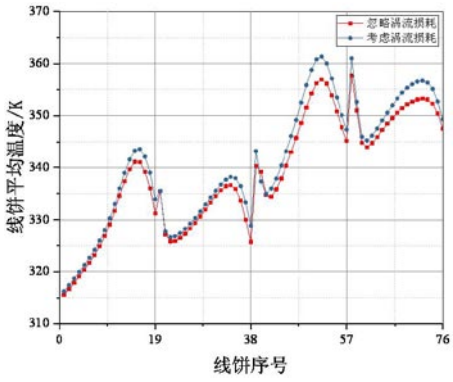
3.2 涡流损耗对油流及温度分布影响

针对当前一些论文在计算绕组温升过程中忽略涡流损耗的处理方式，本节对比分析了涡流损耗对油流及温度分布影响。

两个研究中的所有边界条件和几何参数都是相同的，采用2.2节所述的边界条件，均采用平均热源且考虑了损耗的温度效应。计算结果如图7所示。



（a）水平油道质量流量分布



（b）各线饼平均温度

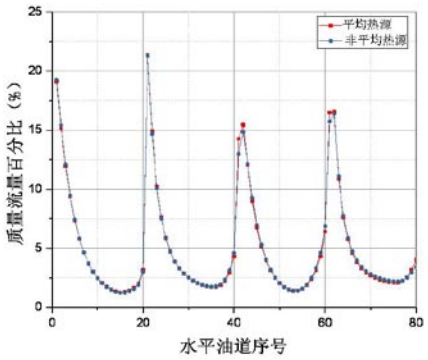
图7 涡流损耗对油流及温度分布影响

图7（a）两种情况下，水平油道的质量流量分布变化趋势在各个分区基本一致。从图7（b）中可以看出，考虑涡流损耗使得各个线饼温度增大，尤其在各个分区的局部热点附近，温度的增加更加显著，最大的温度插值为4.7K，位于52号线饼。

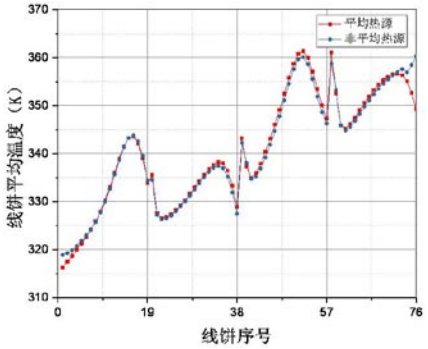
表4结果表明，在各个分区内，两种情况下，局部热点的位置基本一致，表明涡流损耗不影响局部热点的定位，但是考虑了涡流损耗后，各分区的线饼平均温度及局部热点温度均升高。整体绕组热点位置发生改变，热点温度升高3.6K。

3.3 热源分布对油流及温度分布影响

本节主要研究了线饼损耗均匀分布和非均匀分布对绕组温度分布的影响。两个研究中的所有边界条件和几何参数都是相同的，采用2.2节所述的边界条件，且均考虑了损耗的温度效应。在平均热源下，总的欧姆损耗和涡流损耗在各线饼之间均匀分布。 对于非平均热源下，单独施加每个线饼的欧姆损耗和涡流损失。计算结果如图8所示。



（a）水平油道质量流量分布



（b）各线饼平均温度

图8 热源分布对油流及温度分布影响

从图8（a）中可以看出，两种情况下，水平油道的质量流量分布变化趋势在各个分区基本一致，油流的质量流量似乎不受热源分布的影响，这是由于油的密度和动力粘度随温度变化很小。图8（b）中线饼平均温度分布表明，与平均热源相比，采用非平均热源时产生的温差主要发生在绕组的两端，这是由于绕组的涡流损耗分布在两端较大导致的，端部温度最大差值为11.1K。

表5结果表明，两种热源情况下，第一、二、三分区的温度分布基本一致，线饼平均温度及热点温度的差值很小，热点位置基本一致。第四分区内，考虑到第四分区位于整个绕组油流出口端，由于热条纹和绕组顶端的散热性能增强共同作用，在平均热源下，局部热点出现在分区入口的第一个线饼；考虑了损耗不均匀分布后，由于绕组的最后一个线饼的损耗大约是比中央部分高2.5倍，非平均热源下的绕组热点温度位于绕组顶端。对于整个绕组，第四分区的散热性能由于第三分区，因此热点温度位于第三分区中上部分的52号线饼，但由于绕组端部较大的涡流损耗，非平均热源下的整体绕组热点依然位于绕组顶端。

表3 损耗温度效应对绕组特征温度分布影响

	模型（损耗的温度效应）	*线饼平均温度*	*热点温度*	*热点位置*
分区一	*考虑*	*330.322*	*343.816*	*线饼16*
	忽略	*318.429*	*325.937*	*线饼14*
分区二	*考虑*	*332.394*	*338.412*	*线饼15*
	忽略	*321.757*	*327.624*	*线饼1*
分区三	*考虑*	*348.709*	*361.635*	*线饼15*
	忽略	*328.617*	*332.748*	*线饼14*
分区四	*考虑*	*352.405*	*361.242*	*线饼1*
	忽略	*332.422*	*336.027*	*线饼2*
整体	*考虑*	*340.957*	*361.635*	*线饼52*
	忽略	*325.306*	*336.027*	*线饼59*

表4 涡流损耗对绕组特征温度分布影响

	模型（涡流损耗）	*线饼平均温度*	*热点温度*	*热点位置*
分区一	*考虑*	*330.322*	*343.575*	*线饼16*
	忽略	*328.799*	*341.067*	*线饼15*
分区二	*考虑*	*332.394*	*338.213*	*线饼15*
	忽略	*331.059*	*336.622*	*线饼15*
分区三	*考虑*	*348.709*	*361.393*	*线饼14*
	忽略	*345.924*	*356.918*	*线饼14*
分区四	*考虑*	*352.404*	*361.044*	*线饼1*
	忽略	*349.983*	*357.756*	*线饼1*
整体	*考虑*	*340.957*	*361.393*	*线饼52*
	忽略	*338.942*	*357.756*	*线饼58*

表5 热源分布对绕组特征温度分布影响

	模型	线饼平均温度	热点温度	热点位置
分区一	平均热源	330.322	343.816	线饼16
	非平均热源	330.691	344.064	线饼16
分区二	平均热源	332.394	338.412	线饼15
	非平均热源	331.773	337.61	线饼15
分区三	平均热源	348.709	361.635	线饼15
	非平均热源	347.637	360.408	线饼14
分区四	平均热源	352.405	361.242	线饼1
	非平均热源	352.999	360.558	线饼19
整体	平均热源	340.957	361.635	线饼52
	非平均热源	340.775	360.558	线饼76

4. 结论

本文根据一台油浸式电力变压器的低压绕组结构，建立了该变压器二维流体场-温度场计算模型，分析了损耗的温度效应及热源分布对绕组油道中油流速度及温度分布的影响。分析结论如下。

1）损耗的温度效应对绕组的温度分布影响很大。考虑损耗的温度效应后，各个分区和整体绕组的热点均发生上移，整体绕组的热点温度增加24.8K，热点位置偏差为7个线饼。

2）考虑涡流损耗使得各个线饼温度增大，尤其在各个分区的局部热点附近，温度的增加更加显著。

3）热源分布对绕组水平油道油流流动影响较小，对绕组温度分布的影响主要集中在绕组端部。

4）综合考虑损耗的温度效应及非平均热源，使得建立的绕组数值计算模型与实际运行变压器内部的热行为更为接近，从而为工程应用中更精确分析变压器热行为提供参考。

参考文献：

[1] 王珊珊, 肖黎, 廖才波. 110kV 环氧浇注干式变压器流体-温度场的有限元仿真计算[J]. 变压器, 2016, 53(1): 1-5.

[2] 李洪奎，井永腾，李 岩，等. 超高压双器身结构的变压器设计与电磁分析[J]. 高电压技术,2016,42(7)：2322-2328.

[3]Ebenezer M, Ramachandralal R M, Pillai Sarasamma C N P. Study and analysis of the effect of harmonics on the hot spot temperature of a distribution transformer using finite-volume method[J]. Electric Power Components and Systems, 2015, 43(20): 2251-2261.

[4] 兰贞波，文 武，阮江军，等. 基于有限元法的干式变压器多物理场分析计算[J]. 高压电器，2015，51(8)：107-113.

[5] 傅晨钊, 汲胜昌, 王世山,等. 变压器绕组温度场的二维数值计算[J]. 高电压技术, 2002, 28(5):10-12.

[6] 肖 强，王红艳，陈 晴，等. 油浸式电力变压器内部温度场与流场特性分析[J]. 南京工程学院学报：自然科学版,2016,14(1):60-64.

[7] 李剑,姚舒瀚，杜斌,等.植物绝缘油及其应用研究关键问题分析与展望[J]. 高电压技术， 2015,41(2)：353-363.

[8] 朱海兵，李晓健，吴 奕,等.变压器温度场分布的热流耦合分析[J].南京工程学院学报：自然科学版,20151,3(3):74-78.

[9] XIE B, II S, IKEBATA A, et al. A multi-moment finite volume method for incompressible Navier-Stokes equations on unstructured grids: volume-average/point-value formulation[J]. Journal of Computational Physics, 2014, 277: 138-162.

[10] 谢裕清， 李琳， 宋雅吾,等. 油浸式电力变压器绕组温升的多物理场耦合计算方法[J]. 中国电机工程学报， 2016, 36(21):5957-5965.

[11] 李琳, 谢裕清, 刘刚,等. 油浸式电力变压器饼式绕组温升的影响因素分析[J]. 电力自动化设备, 2016, 36(12):83-88.

[12] 刘国坚, 王丰华. 油浸式电力变压器温度场分布的计算分析[J]. 科学技术与工程, 2015(32):83-89.

[13] 熊兰, 赵艳龙, 杨子康,等. 树脂浇注干式变压器温升分析与计算[J]. 高电压技术, 2013, 39(2):265-271.

[14]夏彦卫，张建涛，谢庆. 一种油浸式电力变压器绕组温升工程计算方法[J]. 高压电器，2017，53(9)：176-180.

[15]徐永明, 刘飞, 齐玉麟. 基于流体网络的电力变压器绕组温度预测[J]. 高电压技术, 2017, 43(5):1509-1517.

[16] 陈伟根, 苏小平, 孙才新,等. 基于有限体积法的油浸式变压器绕组温度分布计算[J]. 电力自动化设备, 2011, 31(6):23-27.

[17]王宇翔, 王建民, 刘建新,等. 油浸式变压器绕组涡流损耗及温升分布研究[J]. 电力科学与工程, 2011, 27(1):28 - 31.
[18] E.J. Kranenborg, C.O. Olsson, B.R. Samuelsson, L. - Å. Lundin, R.M. Missing, Numerical study on mixed convection and thermal streaking in power transformer windings[C], in: Proceedings of the 5th European Thermal - Sciences Conference, The Netherlands (2008).
[19]J.Y. Lee, S.W. Lee, J.H. Woo, I.S. Wang, CFD analyses and experiments of a winding with zigezag cooling duct for a power transformer, CIGRE Conference, Paris, France, (2010) A2 - 310.
[20] A. Skillen, A. Revell, H. Iacovides, W. Wu, Numerical prediction of local hotspot phenomena in transformer windings, Applied Thermal Engineering 36(2012) 96 - 105.
[21] IEEE Std C57.91 - 1995, Guide for Loading Mineral - Oil - Immersed Transformers(1995) (Annex G).
[22]Wijaya J, Guo W, Czaszejko T, et al. Temperature distribution in a disc - type transformer winding[C].Industrial Electronics and Applications. IEEE, 2012:838 - 843.
[23] Wakil N E, Chereches N C, Padet J. Numerical study of heat transfer and fluid flow in a power transformer [J]. International Journal of Thermal Sciences, 2006, 45(6):615 - 626.

要闻集锦

英国政府发布人工智能发展报告

据www.gov.uk网站2017年10月27日消息报道，英国政府网站10月15日公布了《英国发展人工智能产业》报告。作为英国“数字化战略”的一部分，该报告旨在为英国商务能源与产业战略部和文化传媒体育部提供政策建议，推动英国人工智能新兴领域和领先技术的发展与应用。

该报告指出，人工智能的迅猛发展源于三项关键要素：新的、更大规模的数据集；越来越多的特定高层次技能专家；越来越强大的计算能力。针对如上要素，提出四个方面建议，以指导政产学界共同努力，使英国成为人工智能全球领导者。这些建议主要为：（1）提升数据可获性，包括推动人工智能数据拥有机构和数据利用机构之间的合作和共享，增强“数据信任”；资助研究以机器可读的格式发布基础数据，并提供明确的权利信息。（2）改善人才技能培养，包括认同人工智能多元化劳动力的价值和重要性；针对学生发起资助项目，鼓励学生攻读人工智能硕士课程；在领先高校至少设立200个专业博士学位名额；鼓励发展人工智能大规模网络开放课程和在线持续专业发展课程；设立国际图灵人工智能奖学金；（3）增强人工智能研究，包括将艾伦•图灵研究所建成国家级人工智能和数据科学研究所；鼓励高校授权创建衍生公司；政府与学术和产业界共同努力，协调人工智能研究计算能力的需求。（4）支持人工智能应用，包括建立英国人工智能委员会，协调发展英国人工智能应用；制定框架解释人工智能过程、服务和决策，提高透明度并建立问责制；制定一系列行动计划，推广人工智能应用的最佳实践；确保产业战略挑战基金和小企业研究计划能支持解决人工智能领域的各种挑战。

（柯　庆）

大数据存储与处理关键技术研究

孙大为 张广艳 舒继武 郑纬民
清华大学计算机科学与技术系 北京 100084

关键词：大数据、存储架构、处理技术

1. 引言

云计算、物联网、移动互连、社交媒体等新兴信息技术和应用模式的快速发展，促使全球数据量急剧增加，推动人类社会迈入“大数据时代”[1-4]。近年来，大数据迅速发展成为科技界和企业界甚至世界各国政府关注的热点。Nature和Science等相继出版专刊来专门探讨大数据带来的挑战和机遇。著名管理咨询公司麦肯锡声称，“数据已经渗透到当今每一个行业和业务职能领域，成为重要的生产因素。人们对于大数据的挖掘和运用，预示着新一波生产力增长和消费者盈余浪潮的到来”，大数据已成为社会各界关注的新焦点。

一般意义上，大数据是指无法在可容忍的时间内用现有IT技术和软硬件工具对其进行感知、获取、管理、处理和服务的数据集合。但近年来大数据的飙升主要来自人们的日常生活，特别是互联网公司的服务。据著名咨询公司IDC的统计，2011年全球被创建和复制的数据总量为1.8ZB（10的21次方），其中75%来自于个人（主要是图片、视频和音乐），远远超过人类有史以来所有印刷材料的数据总量（200PB）。

与传统规模的数据工程相比，大数据具有以下主要特性：（1）数据集合的规模不断扩大，已经从GB到TB再到PB，甚至已经开始以EB和ZB来计数。IDC的研究报告称，未来十年全球大数据将增加50倍，管理数据仓库的服务器的数量将增加10倍以便迎合50倍的大数据增长。（2）数据类型繁多，包括结构化数据、半结构化数据和非结构化数据。现代互联网应用呈现出非结构化数据大幅增长的特点，至2012年末非结构化数据占有比例将达到整个数据量的75%以上。（3）产生速度快，处理能力要求高。根据IDC的“数字宇宙”的报告，预计到2020年，全球数据使用量将达到35.2ZB。在如此海量的数据面前，处理数据的效率就是企业的生命。大数据往往以数据流的形式动态、快速地产生和演变，具有很强的时效性，只有把握好对数据流的掌控才能有效利用这些数据。（4）数据真伪难辨，可靠性要求更严格。大数据的集合和高密度的测量将令“错误发现”的风险增长。斯坦福大学的统计学教授特来沃尔-哈斯迪（Trevor Hastie）称，如果想要在庞大的数据“干草垛”中找到一根有意义的“针”，那么所将面临的问题就是“许多稻草看起来就像是针一样”。（5）价值大，但密度低，挖掘难度大。价值密度的高低与数据总量的大小成反比。如何通过强大的机器算法更迅速地完成数据的价值“提纯”成为目前大数据背景下亟待解决的难题。此外，大数据往往还呈现出了个性化、不完备化、交叉复用等特征。

大数据计算基础研究目前还没有成体系的理论成果，这其中涉及到：大数据的感知与表示、大数据的组织与存储、大数据的计算架构与体系、大数据的模式发现与效应分析等内容。并最终构建一个完整的大数据计算体系。大数据计算体系的研究，一方面，需要关注大数据如何存储，提供一种高效的数据存储平台。另一方面，在数据合理存储平台的基础上，为了应对大数据的快速以及高效可靠处理，需要建立大数据的计算模式以及相关的优化机制。本文则介绍了我们在大数据存储方面所开展的重复数据删除和大数据编码优化的研究工作，以及在大数据计算方面所开展的大数据快速Range-sum查询、大数据流式计算和大数据图计算的研究工作。

2. 大数据存储

高效的大数据存储架构涉及到大数据重复数据删除问题和大数据编码优化问题，我们针对这两个问题进行了初步的研究。

2.1 大数据重复数据删除技术

大数据时代，数据的体量和增长速度大大超过之前，这其中因重复数据导致的部分在不断增大，国际数据公司IDC通过研究发现在数字世界中有近75%的数据是重复的[5]，企业战略集团指出在备份和

归档存储系统中数据的冗余度超过90%[6]。为此，为了缓解存储系统的空间增长问题，缩减数据占用空间，降低成本，高效的重复数据删除技术(Cluster Deduplication)在大数据时代显得尤其重要。但重复数据删除技术是计算密集型，也是I/O密集型的技术，特别是重复删除运算相当消耗运算资源，读写数据时都要进行大量的I/O处理，对存取性能消耗会造成相当程度冲击。

在大数据存储环境中，将集群重复数据删除技术有效的融入分布式集群存储架构中，使得存储系统在数据存储过程中实现对重复冗余数据的在线去重，同时在存储性能、存储效率以及去重率等方面进行了优化。

(1) 具有重复数据删除功能的分布式存储架构

为了构建适用于大数据存储的分布式重复数据删除系统，通过在分布式文件系统中设计并实现具有重复数据删除功能的分布式文件系统，来实现分布式系统的高去重率、高可扩展、高吞吐等特征。其架构如图1所示，包括客户端(Client)、元数据服务器(Meta Data Server)以及数据服务器(Data Server)三部分组成。

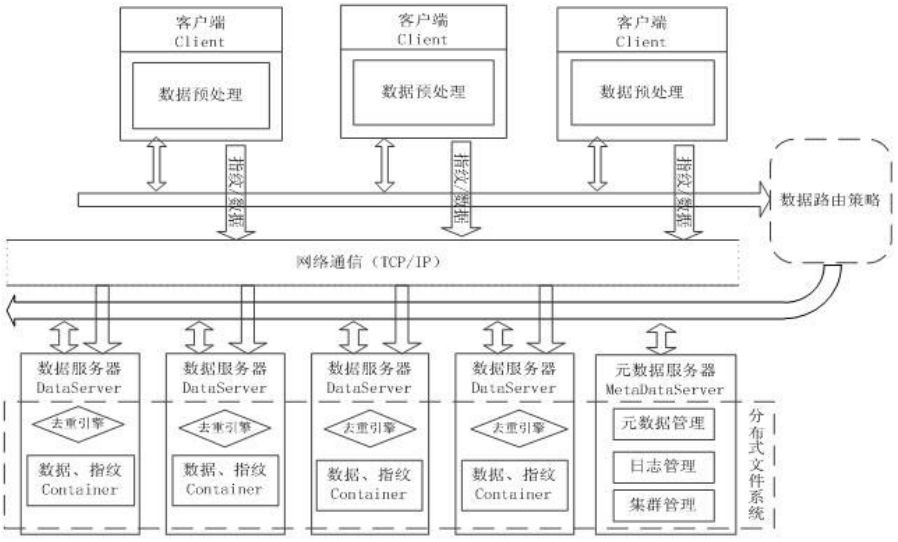


图1 重复数据删除系统架构

客户端主要提供集群重复数据删除系统对外的交互接口，并在所提供的文件操作接口中实现基于重复数据删除的存储逻辑，实现对数据的预处理，如数据块的划分与指纹的提取，并组织为超块(SuperBlock)，通过网络与分布式文件系统中各节点进行交互确定超块的路由地址，最后负责将数据与指纹发送到数据服务器。

元数据服务器实现了对元数据的存储、集群的管理与维护。实现了对数据存储过程中整个会话的管理，保存与管理分布式文件系统中的元数据，管理和维护系统存储状况，指导数据路由并满足系统存储的负载均衡。

数据服务器主要包含数据去重引擎以及数据的存储和管理。数据服务器通过网络与客户端进行通信，响应客户端的读写请求，与元数据服务器通过网络异步更新数据服务器的数据接收状况以及节点存储状况。当接受到客户端的写请求时，数据服务器负责接送数据并在节点内进行冗余数据的去重。

网络通信模块主要是提供一种能够在客户端与分布式文件系统各节点进行通信的高效机制。通信方式有通过远程过程调用交互元数据以及少量控制信息，通过Socket流式网络连接传输大量的数据与指纹信息。

(2) 数据路由策略

基于单节点内局部去重，即在一个节点内对数据进行去重，保证了存储环境中系统的整体性能与存储带宽。这其中数据的存储位置将是关键，因为数据路由位置将直接影响数据的去重率，需要根据数据的相似性，以及数据局部性的相关理论，基于超块的高效局部相似路由算法来保证全局数据去重的可靠性。

在数据路由粒度方面，将连续切分的小分块数据组成大的超块，将超块作为路由数据分发的基本单位。超块是对上传数据通过分块算法(如可变分块(Content-Defined Chunking，CDC)算法、固定分块(Fixed-Sized Partition，FSP)算法进行分块后，如图2所示，由连续的几个小分块拼接成的大的局部块。文件由连续的超块组成，并将超块作为数据路由的单位，发送到选定的节点中进行节点内的冗余数据去重。

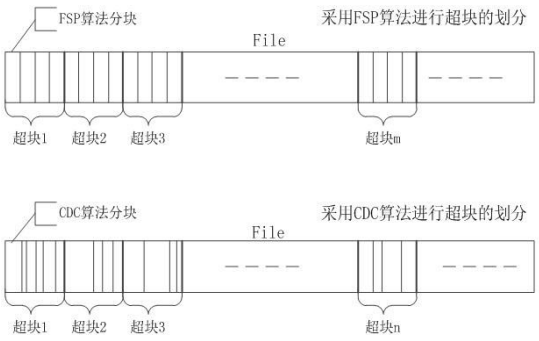


图2 超块的划分与组织

在数据相似性方面，数据相似就转变为判读节点中已有超块与新来的超块之间的相似度。可以通过使用Jaccard距离[7]来衡量两个超块的相似度。通过有状态的局部相似路由算法，实现数据的路由。

2.2 大数据编码优化技术

纠删码技术是不同于多副本技术的另外一种容灾策略，它的基本思想是：通过纠删码算法对k个原始数据块进行数据编码得到m个纠删码块，并将这k + m个数据块存入到不同的数据存储节点中去，完成容灾机制的建立。当k + m个元素中任意的不多于m个元素出错(包括数据和冗余出错)时，均可以通过对应的重构算法恢复出原来的k块数据。基于纠删码的数据冗余方法，具有冗余度低、磁盘利用率高等特点。

在大数据存储平台中利用纠删码作为容灾策略，相比多副本容灾策略对存储空间和网络带宽的需求有所降低，但是同时引进了纠删码计算，提高纠删码编码的计算速度是关键。为了提高以纠删码为容灾策略的大数据存储系统中数据存储速度，最有效的办法就是减少纠删码计算过程的异或次数。

目前求调度的算法都是启发式的，如CSHR，UBER-CSHR，X-Sets等，用它们对一个柯西矩阵求取调度时，各自所得到的调度都无法保证是所有调度方法中最优的，并且柯西矩阵配置参数(k，m，w)组合下会有$\binom{2^w}{k+m}\binom{k+m}{k}$个柯西矩阵，究竟哪一个柯西矩阵会产生比较好的调度，目前为止没有发现好的规律。

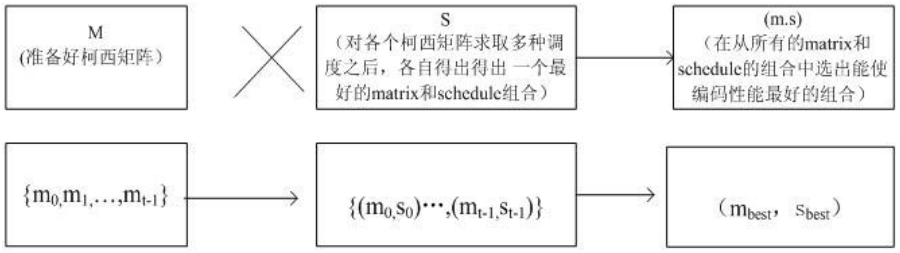


图3 选择框架简单示意图

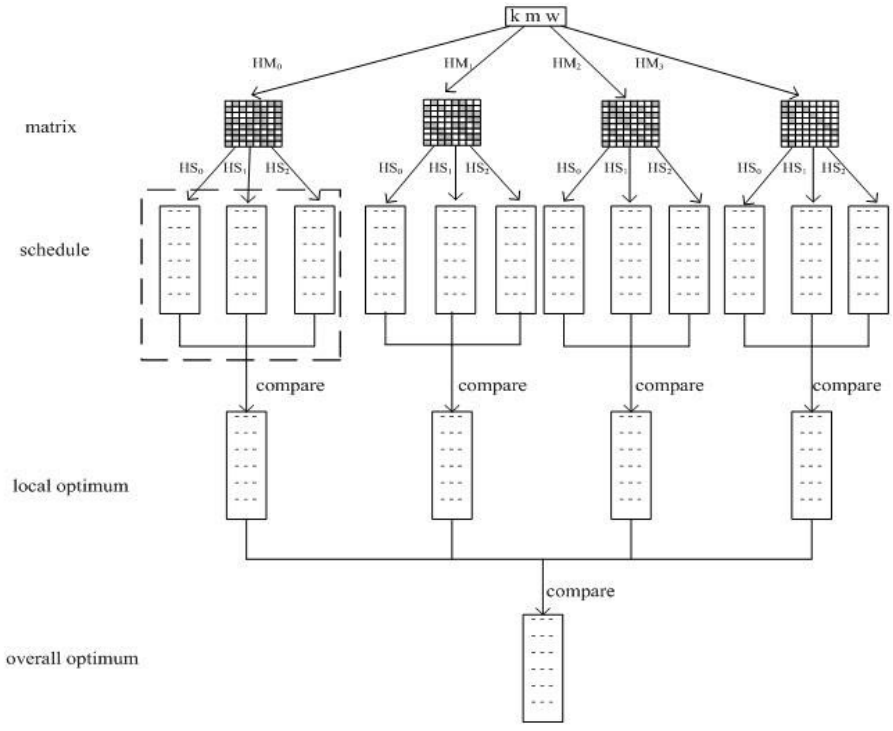


图4 整体选择框架描述

针对该问题，为了能提高数据编码效率，提出了关于纠删码求调度的选择框架思想，利用该选择框架能够求出目前技术水平下高效的数据编码方案——优化调度方案。该方案为柯西矩阵配置参数(k, m, w)选择出目前能实现高效编码效率的柯西矩阵和相应的调度，以用于大数据存储的数据编码。选择框架如图3所示，它包括三部分：

(1) 首先准备柯西矩阵，根据多种生成柯西矩阵的算法生成柯西矩阵集合{m_0，m_1，……，m_{t-1}}。考虑到更新性能(柯西矩阵中“1”的个数越少越好)，倾向于选择柯西矩阵中“1”的个数较小的柯西矩阵。

(2) 对第一步准备好的柯西矩阵求取调度。对每个柯西矩阵运行多种求取调度的启发式算法之后都得出各自最好的柯西矩阵和调度组合(m，s)，所以第二阶段的结果为{(m_0，s_0)，(m_1，s_1)，……，(m_{t-1}，s_{t-1})}。

(3) 从第二步的结果中，选出所有调度中异或操作次数最少的调度，得到能使编码性能最高的柯西矩阵和调度组合(m_{best}，s_{best})。

整体的选择框架描述如图4。

3. 大数据处理

在合理存储平台的基础上，为了实现对大数据的快速、高效、可靠处理，需要实现对大数据处理机制的优化，其中涉及到大数据快速Range-sum查询技术、大数据流式计算技术和大数据图计算技术。

3.1 大数据快速Range-sum查询技术

Range-sum 问题最早在文献[8]中进行定义，主要是对满足区间查询条件的数据进行求和，问题具体形式描述如下：

Select exp(AggColumn), other ColName where

li_1<ColNamei< li_2 opr

lj_1 < ColNamej <lj_2 opr

…;

其中:exp包括SUM，COUNT(*)两类基本函数，其他的聚合函数如AVG，STD等可以通过扩展SUM和COUNT(*)来实现；AggColumn是聚合属性列；li_1<ColNamei< li_2 ，lj_1 < ColNamej <lj_2是区间查询条件；opr 表示逻辑运算符号，如逻辑AND或OR等。通常将支持聚合计算的属性AggColumn称为聚合属性；支持区间查询条件的属性ColNamej，ColNamej称为索引属性。

分布式环境下Range-sum的计算开销主要是由节点之间和节点内部两类计算时间延迟导致。节点之间的时间延迟是由于聚合计算模式中由于数据分布不同节点，在计算聚合结果时需要多个节点之间进行数据通信、同步等带来的时间延迟；节点内部时间延迟主要是由于区间查询条件包括的数据量大，在每个节点上在处理随时间不断膨胀的原始数据或索引数据而产生的时间延迟。只有有效优化两类延迟，才能充分提高大数据Range-sum的计算效率。

FastRAQ[9]是一种根据聚合计算特点，设计面向聚合计算模式的大数据分区方法，在每个分区内独立获得近似聚合计算结果，避免分区之间数据交互和等待产生的延迟。分区内设计支持区间查询条件的基数估算方法，结合分区内聚合属性无偏样本获得满意精度的聚合计算结果，避免扫描原始数据带来的计算开销。FastRAQ的计算思想如式（1）所示。

$$\sum_{i=1}^{M} Count_i \times Sample \quad (1)$$

其中，M表示分布式环境下的分区数目；Counti是第i个分区内满足区间条件的基数估算值；Samplei是第i个分区内聚合属性的样本值。

FastRAQ的分区方法是通过聚合属性值的运算实现的，根据计算结果把数据划分成相互独立的多个分区，每个分区负责一个固定的聚合属性值的范围，所有分区负责的属性值范围构成了整个聚合属性值的值域空间。在每个分区内计算无偏估计样本作为整个分区内聚合属性的近似值。分区算法可以限定样本值的相对误差。FastRAQ在每个分区内采用基数估算直方图支持区间条件下记录基数的统计。区间基数估算直方图在恒定索引数据量条件下以Hash运算为基础，实现区间基数估算。图5给出了FastRAQ分区组织结构和Range-sum查询流程。

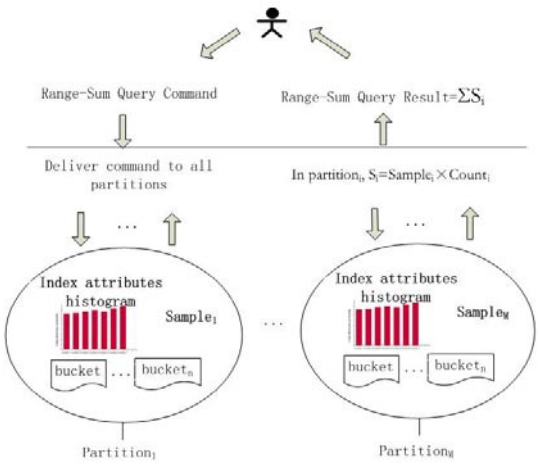


图5 FastRAQ分区组织结构和Range-sum查询流程

FastRAQ在查询前需要组织好分区和每个分区内的索引结构。FastRAQ结合计算模式设计一种schema-less的column-family描述方法，在column-family中定义计算模式下不同的索引属性簇，主要包括三类：聚合属性簇，定义为“aggregation”；

索引属性簇，定义为“index”，其他属性簇定义为“default”。FastRAQ基于schema建立类SQL的DML与DDL。FastRAQ中一个具体Range-sum查询语句与属性簇关系如图6所示。

FastRAQ可以支持多维索引属性下的近似Range-sum查询。FastRAQ首先获得每个区间条件下的基数值，然后对多个基数根据记录的ID做集合运算，获得多区间条件下的基数估算结果。在每个分区内利用多区间基数估算结果和聚合属性值样本相乘产生分区内的近似聚合计算结果。最后把多个分区内所有的计算结果相加，得到最终的Range-sum近似计算结果。

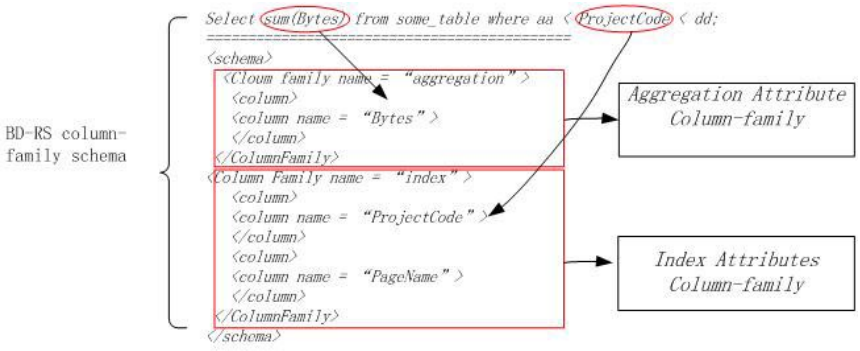


图6 FastRAQ中面向聚合计算的属性簇描述方法

3.2 大数据流式计算技术

在大数据流式计算中，其系统架构如图7所示，无法确定数据的到来时刻和到来顺序，也无法将全部数据存储起来。因此，不再进行流式数据的存储，而是当流动的数据到来后在内存中直接进行数据的实时计算。如Twitter的Storm、Yahoo的S4就是典型的流式数据计算架构，数据在任务拓扑中被计算，并输出有价值的信息。对于无需先存储，可以直接进行数据计算，实时性要求很严格，但数据的精确度往往不太重要的应用场景，大数据流式计算具有明显优势。大数据流式计算中数据往往是最近一个时间窗口内的，因此数据延迟往往较短，实时性较强，但数据的精确程度往往较低。

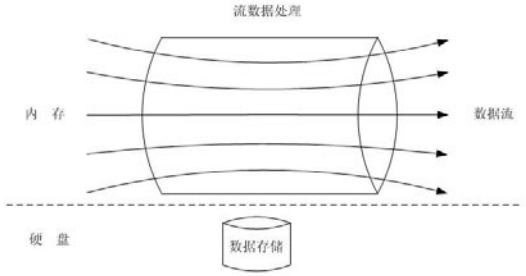


图7 大数据流式计算架构

大数据流式计算中的数据流主要体现了五个方面的特征[10, 11]。（1）实时性：流式大数据是实时产生，实时计算，结果反馈往往也需要保证及时性。流式数据价值的有效时间往往较短，大部分数据到来后直接在内存中进行计算并丢弃，只有少数数据才被长久保存到硬盘中。（2）易失性：在大数据流式计算环境中，数据流往往是到达后立即被计算并使用。数据的使用往往是一次性的、易失的，即使重放，得到的数据流和之前的数据流往往也是不同的。（3）突发性：在大数据流式计算环境中，数据的产生完全由数据源确定，由于不同的数据源，在不同时空范围内的状态不统一且动态变化，导致数据流的速率呈现出了突发性的特征。前一时刻数据速率和后一时刻数据速率可能会差异巨大。（4）无序性：在大数据流式计算环境中，各数据流之间、同一数据流内部各数据元素之间是无序的。一方面，由于各个数据源之间是相互独立的，所处的时空环境也不尽相同，因此无法保证数据流间的各个数据元素的相对顺序。另一方面，即使是同一个数据流，由于时间和环境的动态变化，也无法保证重放数据流和之前数据流中数据元素顺序的一致性。（5）无限性：在大数据流式计算中，数据是实时产生、动态增加的，只要数据源处于活动状态，数据就会一直产生和持续增加下去，可以说，潜在的数据量是无限的，无法用一个具体确定的数据实现对其进行量化。

针对具有实时性、易失性、突发性、无序性、无限性等特征的流式大数据，理想的大数据流式计算系统应该表现出低延迟、高吞吐、持续稳定运行和弹性可伸缩等特性。这其中离不开系统架构、数据传输、编程接口、高可用技术等关键技术的合理规划和良好设计[12-16]。

系统架构是系统中各子系统间的组织方式，属于大数据计算所共有的关键技术，大数据流式计算需要选择特定的系统架构进行流式计算任务的部署。当前，大数据流式计算系统采用的系统架构可

以分为无中心节点的对称式系统架构（如S4系统）以及有中心节点的主从式架构（如Storm系统）。

数据传输是指完成有向任务图到物理计算节点的部署之后，各个计算节点之间的数据传输方式。在大数据流式计算环境中，为了实现高吞吐和低延迟，需要更加系统的优化有向任务图以及有向任务图到物理计算节点的映射方式。在大数据流式计算环境中，数据的传输方式分为主动推送方式（基于Push方式）和被动拉取方式（基于Pull方式）。

编程接口是方便用户根据流式计算的任务特征，通过有向任务图来描述任务内在逻辑和依赖关系，并编程实现任务图中各节点的处理功能。用户策略的定制、业务流程的描述和具体应用的实现需要通过大数据流式计算系统提供的应用编程接口。良好的应用编程接口可以方便用户实现业务逻辑，可以减少用户的编程工作量，并降低用户系统功能的实现门槛。当前大多数开源大数据流式计算系统均提供了类似于MapReduce的类MR用户编程接口。

大数据流式计算系统高可用是通过状态备份和故障恢复策略实现的。当故障发生后，系统根据预先定义的策略进行数据的重放和恢复。按照实现策略可以细分为：被动等待策略（Passive Standby），主动等待策略（Active Standby）和上游备份策略（Upstream Backup）。

此外，大数据流式计算系统也离不开其他相关关键技术的支持，包括：系统故障恢复、系统资源调度、负载均衡策略、数据在任务拓扑中的路由策略等。

3.3 大数据图计算技术

大数据图计算是大数据计算的热门领域，主要是用来分析数据节点之间的关系和相似度。已应用于用户分析、欺诈检测、生命科学等多个领域，其巨大的商业价值已经凸现出来。如：发现有影响力的用户（PageRank）和社区、欺诈检测和推荐系统（GraphLab[17]）。一个领域的工具开发出来后常常会被应用到其他领域，除了GraphLab，分布式计算还被应用到Giraph、GraphX、Faunus和Grappa。其中，GraphLab是由CMU开发的一个并行的图挖掘分布式系统。解决了传统的Map-reduce中机器学习处理中存在的频繁迭代计算和大量节点通信导致计算效率低下的问题。在GraphLab中，以点计算单元，并将机器学习算法抽象为三个过程，分Gather、Apply、Scatter三个步骤。在每一个迭代过程中，点的计算都要经过这三个过程。并且，Graphlab是基于共享内存的，各机器异步、动态并行的执行计算任务，比起BSP计算效率更高，并且能够很好的保证数据的一致性。

在大数据时代，大图的分割问题是大数据图计算最为突出的问题[18]。由于对整个图的访问是随机进行的，那么在图划分时需要考虑三个方面的因素：（1）通信代价，访问跨机器边的通信量。（2）负载均衡，使得每一台机器的问题规模基本接近。（3）存储冗余，为了减少通信量在机器上复制其他机器的存储信息（存在数据一致性问题），主要考虑冗余程度，使综合开销最佳。

此外，大数据图计算还存在以下问题：（1）图数据的局部性差，由于节点众多，两个相连接的点（连接的点对也是随机无法预知的）可能存储的位置相隔很远，即不在同一个存储块。这样需要随机访问节点及边。而访问磁盘的效率又极其低，严重影响计算效率。（2）数据及图结构驱动，对于不同的图形结构需要不同的计算方法，需要设计一个通用的方法。（3）存储和效率，大图处理的规模基本上是十亿级的点的数量，单台PC进行存储似乎不太可能，所以大多数的图计算系统是分布式系统，这样把可以把存储容量和计算分摊到每一个机器上，但问题也出现了，就是怎样进行划分才能使得各机器负载均衡，以及减少个划分之间的通信。

4. 总结

大数据已成得到了社会各界的广泛关注，并将影响着人们生活中的各个方面。但大数据计算基础研究目前还没有成体系的理论成果，这其中涉及到：大数据的感知与表示、大数据的组织与存储、大数据的计算架构与体系等多个方面的内容。我们在大数据计算体系的研究方面，开展了两个层面的工作。在大数据存储层面上，重点关注如何实现大数据的高效、快速的存储，针对大数据重复数据删除问题和大数据编码优化问题，提出了相应的解决方案。在大数据计算层面上，重点关注在合理存储平台的基础上，为了实现对大数据的快速、高效、可靠处理，如何实现对大数据计算机制的优化，其中涉及到大数据快速Range-sum查询技术、大数据流式计算技术和大数据图计算技术。在大数据领域，会不断有很多新的问题和挑战，有待我们进一步深入的研究和解决。

参考文献

[1] Lynch, C. Big data: How do your data grow? Nature, 2008, 455(7209): 28-29.

[2] Lim L, Misra A, Mo T L. Adaptive data acquisition strategies for energy - efficient, smartphone - based, continuous processing of sensor streams, Distributed and Parallel Databases, 2013, 31(2): 321 - 351.
[3] 李国杰, 程学旗. 大数据研究:未来科技及经济社会发展的重大战略领域——大数据的研究现状与科学思考. 中国科学院院刊, 2012, 27(6): 647 - 657.
[4] 孙大为, 张广艳, 郑纬民. 大数据流式计算：关键技术及系统实例. 软件学报, 2014, 25(4): 839 - 862.
[5] Gantz J, Reinsel D. The digital universe decade—are you ready? IDC White Paper, May 2010.
[6] Biggar H. Experiencing data de - duplication: Improving efficiency and reducing capacity requirements. White Paper, the Enterprise Strategy Group, Feb. 2007.
[7] Reina, D G, Toral, S L, Johnson, P, Barrero, F. Improving discovery phase of reactive ad hoc routing protocols using Jaccard distance. Journal of Supercomputing, 2014, 67(1): 131 - 152.
[8] Chun, S J, Chung, C W, Lee, J H, Lee S L, Dynamic Update Cube for Range - Sum Queries. The 27th VLDB Conference, Roma, Italy, 2001.
[9] Yun X C, Wu G J, Zhang G Y, Li K Q, Wang S P, FastRAQ: A Fast Approach to Range - Aggregate Queries in Big Data Environments, IEEE Transactions on Cloud Computing, 27 Aug. 2014.
[10] Hoi S C H, Wang J L, Zhao P L, et al. Online feature selection for mining big data, Proc. ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, SIGKDD 2012, Beijing, China, ACM Press, Aug. 2012, 93 - 100.
[11] Michael K, Miller K W. Big data: new opportunities and new challenges, Computer, 2013, 46(6): 22 - 24.
[12] Scalosub G, Marbach P, Liebeherr J. Buffer management for aggregated streaming data with packet dependencies, IEEE Transactions on Parallel and Distributed Systems, 2013, 24(3): 439 - 449.
[13] Cugola G, Margara A. Processing flows of information: from data stream to complex event processing, ACM Computing Surveys, 2012, 44(3): 15:1 - 62.
[14] Lim L, Misra A, Mo T L. Adaptive data acquisition strategies for energy - efficient, smartphone - based, continuous processing of sensor streams, Distributed and Parallel Databases, 2013, 31(2): 321 - 351.
[15] Chatziantoniou D, Pramatari K, Sotiropoulos Y. Supporting real - time supply chain decisions based on RFID data streams, Journal of Systems and Software, 2011, 84(4): 700 - 710.
[16] Zhang Z, Gu Y, Ye F, et al. A hybrid approach to high availability in stream processing systems, Proc. 30th IEEE International Conference on Distributed Computing Systems, ICDCS 2010, Genova, Italy, IEEE Press, Jun. 2010, 138 - 148.
[17] GraphLab, http://graphlab.org/projects/index.html.
[18] Furedi, Z, Kostochka, A, Kumbhat, M. Choosability with separation of complete multipartite graphs and hypergraphs, Journal of Graph Theory, 2014, 76(2): 129 - 137.

高性能计算
发展与应用（内部刊物）

DEVELOPMENT & APPLICATION
OF HIGH PERFORMANCE COMPUTING

主　办：上海超级计算中心

协　办：江南计算技术研究所
上海科学技术情报研究所

编辑部：上海张江高科技园区郭守敬路585号

邮　编：201203

电　话：61872222

传　真：61872288

投　稿：news@ssc.net.cn

电子版：http://www.ssc.net.cn/magzine.aspx