计算机研究与发展
Journal of Computer Research and Development
DOI: 10.7544/issn1000-1239.202111185
60(5): 1177−1190, 2023

大规模海洋数据同化的并行优化

蔡 迪 洪学海 肖俊敏 谭光明
(中国科学院计算技术研究所 北京 100190)
(caidi19s@ict.ac.cn)

Parallel Optimization for Large-Scale Ocean Data Assimilation

Cai Di, Hong Xuehai, Xiao Junmin, and Tan Guangming
(*Institute of Computing Technology*, *Chinese Academy of Science*, *Beijing* 100190)

Abstract Ocean data assimilation is an effective method to process ocean data by using ocean observation data and ocean numerical model simultaneously, and the processed ocean data is closer to the real ocean situation. Under high resolution, parallel assimilation programs based on LASG/IAP climate ocean model (LICOM) often involve a lot of file reading, communication and calculation. Although these aspects have been optimized in previous studies, these optimization algorithms only remain at the upper level. Without considering the underlying file system and the architecture of the supercomputer cluster, the optimization algorithm has great limitations, so the effect of optimization is not obvious. In this paper, the data characteristics and computing characteristics of ocean data assimilation are combined with the architectural characteristics of the used supercomputer platform . On this basis, combining the temporal locality and spatial locality, a load-balancing strategy based on computing topology, a parallel optimization strategy based on the storage architecture of Lustre parallel file system and the characteristics of supercomputer clusters, and a three-layer overlapping strategy of computing, reading and communication, and writing back are proposed. Finally, we test our algorithm on Tianhe-2 supercomputer cluster using high-resolution datasets. Compared with the existing ocean assimilation program, the overall performance of our algorithm improves by 18 times under 4 000 cores. In addition, we also test on the Sugon 7000 supercomputer cluster. The maximum number of DCU cards used in this paper is 4000. Compared with the existing program, the overall performance is improved about 8 times.

Key words data assimilation; load balance; I/O optimization; parallel optimization; overlapping computation with communication

摘 要 海洋数据同化是一种同时利用海洋观测资料和海洋数值模式对海洋数据进行修正的有效方法，经过处理的海洋数据更加接近海洋的真实情况. 在高分辨率下，基于中国科学院大气物理研究所（Institute of Atmospheric Physics, Chinese Academy of Sciences, IAP）和大气科学和地球流体力学数值模拟国家重点实验室（State Key Laboratory Modelling for Atmospheric Sciences and Geophysical Fluid Dynamics, LASG）发展的 LASG/IAP 气候系统海洋模式（LASG/IAP climate ocean model, LICOM）的同化并行程序

收稿日期: 2021-11-30; **修回日期**: 2022-06-19
基金项目: 国家重点研发计划项目 (2016YFC1401706); 国家自然科学基金青年科学基金项目 (61802369); 国家自然科学基金项目 (62172391,61972377,62032023,T2125013)
This work was supported by the National Key Research and Development Program of China(2016YFC1401706), the National Natural Science Foundation of China for Young Scientists (61802369), and the National Natural Science Foundation of China (62172391, 61972377, 62032023, T2125013).
通信作者: 洪学海(hxh@ict.ac.cn)

往往涉及大量的文件读取、通信和计算,以往的研究虽然对这些方面进行了优化,但是由于优化只是停留在上层算法层面,没有考虑底层的文件系统以及超算集群的架构,因此优化的效果不太明显.针对以往研究存在的问题,进一步将海洋数据同化的数据特性、计算特性与所使用的超算平台的架构特性相结合,在此基础上结合时间局部性和空间局部性,提出了基于计算拓扑图的负载均衡策略、基于 Lustre 文件存储架构和超算集群特性的并行优化策略,以及计算、读取通信、写回 3 层重叠策略.最后,使用高分辨率数据集,在天河 2 号超算集群上对所提算法进行了测试.相比于现有算法,所提的算法在 4 000 核下对总体同化性能上提升了 18 倍.另外,还在曙光 7 000 超算集群上开展了测试.在 4 000 块 DCU 加速卡上,相比于已有算法,所提算法提升总体计算性能 8 倍左右.

关键词 数据同化;负载均衡;I/O 优化;并行优化;计算与通信重叠

中图法分类号 TP302

海洋数据同化是一种从气候数值预测领域发展而来的,将物理模型与观测资料相结合的预测校正方法[1-3],广泛运用于大气[4]、海洋[5]和地表[6]等诸多领域.目前最为常见的数据同化方法分别为集合卡尔曼滤波算法(FnKF)[7]、集合最优插值算法[8],以及三维、四维变分方法等[9-12].其中集合最优插值由集合卡尔曼滤波发展而来.

随着航海事业的不断发展,捕鱼、军事、资源开采和生物研究等海洋相关的活动日益增多.这些日益增多的海洋活动对海洋预报的准确性和实时性提出了更高的要求,因此许多准实时的海洋观测系统被开发出来[13].随着海洋模式的不断发展,海洋数据同化的分辨率也不断提高.在其他因素不变的情况下,水平方向的分辨率每提高 10 倍,对应的计算量和数据量将提高 100 倍,这大大增加了同化系统对内存、I/O 以及计算能力的要求.海量的数据读取和计算任务对系统的实时性带来了很大的挑战.我国自主研发的天河 2 号超算平台凭借其优越的 I/O 性能以及计算的性能曾多次登顶世界超算的榜首,可以很大程度上满足同化程序对于内存、I/O 以及计算性能的需求,因此许多大数据学者都以天河 2 号作为研究和开发平台[14-24].如何开发一个高效的并行算法提取出海洋数据同化的特性,并与天河的存储以及执行架构相结合,最大程度地提升天河系统的内存、I/O 以及计算资源的利用率显得尤为重要.

本文算法主要针对基于中国科学院大气物理研究所(Institute of Atmospheric Physics, Chinese Academy of Sciences, IAP)、大气科学和地球流体力学数值模拟国家重点实验室(State Key Laboratory Modelling for Atmospheric Sciences and Geophysical Fluid Dynamics, LASG)发展的 LASG/IAP 气候系统海洋模式(LASG/IAP climate ocean model, LICOM)[25-27],在局部集合最优插值的算法框架下,对大规模高分辨率的海洋数据同化程序进行并行优化.由于本文采用的是高分辨率的海洋观测数据,因此同化时存在计算量大、内存需求高、I/O 时间长等问题,在文献 [28] 中已经针对此这个模式下的数据同化程序做了一些优化工作,取得了不错的效果,但是其优化时没有深入理解海洋数据同化的数据特性、计算特性,没有考虑所使用的超算平台架构,以及没有考虑算法的时间局部性和空间局部性的问题,因此仍存在很大的优化空间.本文在文献 [28] 的基础上,用其优化后的程序作为本文的对比程序(下文称之为 I/O 代理程序),并进一步将海洋数据同化的数据特性、计算特性与所使用的超算平台的架构特性相结合,并结合时间局部性和空间局部性,设计了一款高效的大规模海洋数据同化并行算法.

1 算法及程序流程

1.1 同化方法

局部集合最优插值算法是由集合最优插值进行可并行性优化后发展而来[29],具有更好的并行性.

本文局部集合最优插值的实现过程表示为

$$\boldsymbol{\varphi}^{\alpha}=\boldsymbol{\varphi}+\alpha\boldsymbol{A}'\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}(\alpha\boldsymbol{H}\boldsymbol{A}'\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}+\boldsymbol{\gamma}\boldsymbol{\gamma}^{\mathrm{T}})^{-1}\boldsymbol{d}'. \tag{1}$$

其中,$\boldsymbol{\varphi}^{\alpha}$为分析场数据,$\boldsymbol{\varphi}$为背景场数据,$\alpha$为一个在区间(0, 1]上的随机因子;$\boldsymbol{A}$为静态样本数据,$\boldsymbol{A}'$为扰动预测矩阵,由 $\boldsymbol{A}$ 中心化而来,$\boldsymbol{A}'\boldsymbol{A}'^{\mathrm{T}}$为背景误差协方差的估计;$\boldsymbol{H}$为测量算子,代表真实模型与预测模型之间的映射;$\boldsymbol{\gamma}$为扰动测量误差向量,$\boldsymbol{\gamma}\boldsymbol{\gamma}^{\mathrm{T}}$为观测误差协方差;$\boldsymbol{d}'=\boldsymbol{d}-\boldsymbol{H}\boldsymbol{\varphi}$为观测增益,其中 $\boldsymbol{d}$ 为扰动测量向量.

令

$$\boldsymbol{W}=\alpha\boldsymbol{H}\boldsymbol{A}'\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}+\boldsymbol{\gamma}\boldsymbol{\gamma}^{\mathrm{T}}, \tag{2}$$

由于集合扰动误差和测量误差无关，所以由式(2)得

$$\boldsymbol{W}=(\sqrt{\alpha}\boldsymbol{HA}'+\boldsymbol{\gamma})(\sqrt{\alpha}\boldsymbol{HA}'+\boldsymbol{\gamma})^{\mathrm{T}}. \tag{3}$$

令

$$\sqrt{\alpha}\boldsymbol{HA}'+\boldsymbol{\gamma}=\boldsymbol{U\Sigma V}^{\mathrm{T}}, \tag{4}$$

其中$\boldsymbol{U\Sigma V}^{\mathrm{T}}$由$\sqrt{\alpha}\boldsymbol{HA}'$进行 SVD 分解得到. 结合式(3)可得

$$\boldsymbol{W}=\alpha\boldsymbol{HA}'\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}+\boldsymbol{\gamma\gamma}^{\mathrm{T}}=\boldsymbol{U\Sigma V}^{\mathrm{T}}\boldsymbol{V\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}=\boldsymbol{U\Sigma\Sigma}^{\mathrm{T}}\boldsymbol{U}^{\mathrm{T}}, \tag{5}$$

则

$$\boldsymbol{W}^{-1}=(\alpha\boldsymbol{HA}'\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}+\boldsymbol{\gamma\gamma}^{\mathrm{T}})^{-1}=\boldsymbol{U}(\boldsymbol{\Sigma\Sigma}^{\mathrm{T}})^{-1}\boldsymbol{U}^{\mathrm{T}}=\boldsymbol{U\Lambda}^{-1}\boldsymbol{U}^{\mathrm{T}}, \tag{6}$$

结合式(1)可得

$$\boldsymbol{\varphi}^{\alpha}=\boldsymbol{\varphi}+\alpha\boldsymbol{A}'\boldsymbol{A}'^{\mathrm{T}}\boldsymbol{H}^{\mathrm{T}}\boldsymbol{U\Lambda}^{-1}\boldsymbol{U}^{\mathrm{T}}\boldsymbol{d}'. \tag{7}$$

SVD 分解以及式(7)便是本文各个海洋格点的主要计算部分，其中最为耗时的便是 SVD 分解过程. 在 SVD 分解完成之后，将式(7)分为 5 个步骤计算：

$$\boldsymbol{X}_1=\boldsymbol{\Lambda}^{-1}\boldsymbol{U}^{\mathrm{T}}, \tag{8}$$

$$\boldsymbol{X}_2=\boldsymbol{X}_1\boldsymbol{d}', \tag{9}$$

$$\boldsymbol{X}_3=\boldsymbol{U}\boldsymbol{X}_2, \tag{10}$$

$$\boldsymbol{X}_4=(\alpha\boldsymbol{HA}')\boldsymbol{X}_3, \tag{11}$$

$$\boldsymbol{\varphi}^{\alpha}=\boldsymbol{\varphi}+\boldsymbol{A}'\boldsymbol{X}_4. \tag{12}$$

在局部最优插值的算法中，整个同化数据区域按照经纬度被划分成很多个格点，对于每一个格点只要可以获取到周围(2×*prep_rx*)×(2×*prep_ry*)范围内的数据便可以对该节点进行数据同化，其中在本文的分辨率下 *prep_rx*=105, *prep_ry*=19. 这样良好的并行特性十分便于计算的并行化，但是这也导致了每个格点在数据分发时需要带边数据，在进行通信时会产生大量的通信量，增加数据通信的时间，大大降低程序的并行效率. 因此为了提升程序的并行效率、提高并行系统的带宽利用率以及减少通信时间，提升程序的并行效率尤为重要，这些将在下文做详细介绍.

1.2 程序流程

文献 [28] 的程序的流程图如图 1 所示. 并本文以此作为对比程序，并在此基础上进行优化.

1.3 存在的挑战

1)负载均衡的挑战

由图 2 可以看出，I/O 代理程序各个进程的计算负载极不均衡，大量的进程没有计算任务，大部分计算任务被分配到了少量的进程上，且这些进程内部负载也十分的不均衡，存在极大的进程浪费. 在 I/O 代理程序中，同化程序以同化数据中海洋格点的数目为负载均衡的依据，对进程进行计算任务的划分.

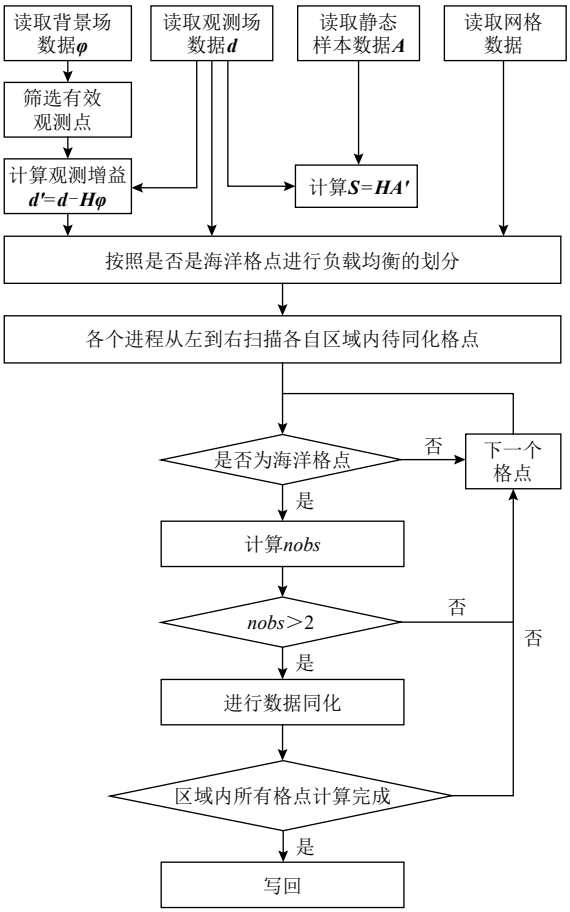


Fig. 1 Flowchart of LICOM data assimilation pragram based on the I/O agents strategy

图 1 基于 I/O 代理策略的 LICOM 数据同化程序流程图

这样的划分方式只是从同化流程的角度理解海洋数据同化，没有深入理解同化算法以及进程的计算过程. 因此在这种负载均衡方式下，虽然各个进程表面上看起来负载均衡，实际上执行时存在很大的负载不均衡性. 虽然每个进程获得的可同化格点的数目是一样的，但并不是每个海洋格点都是可以同化的，当格点周围的 *nobs*≤2 时(其中 *nobs* 代表同化格点周围有效的观测数据点的数量)，此部分格点是不可以同化的. 最坏的情况是，一些进程获取的海洋格点全部需要同化，而一些海洋格点获取的海洋格点完全不需要同化，这要导致极大的负载不均衡性，从而造成大量计算节点的浪费.

2)读取的挑战

由图 3 可以看出 I/O 代理程序在读取 64.8 GB 的数据时较为耗时，并行读取效率较低. I/O 代理程序读取时采用代理节点对数据进行读取，之后由代理节点对计算进程进行数据分发. 这样的设计方式仅在上层算法层面对同化程序的读取进行了优化，并

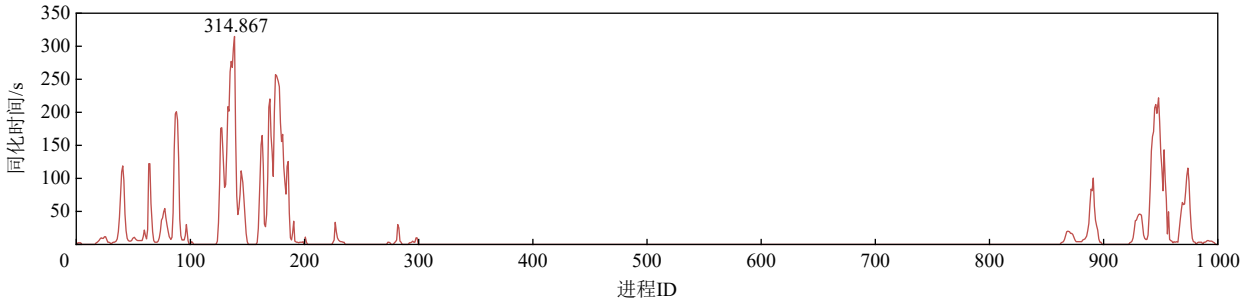


Fig. 2 The assimilation time of processes under 1 000 cores for the I/O agents program

图 2 I/O 代理 1 000 核下各个进程的同化时间

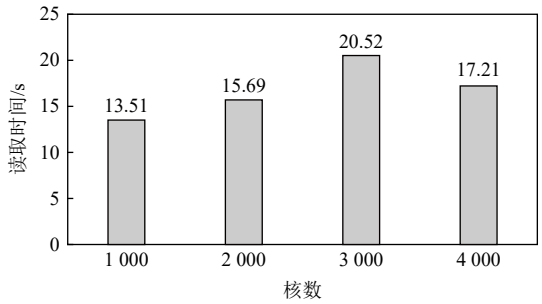


Fig. 3 The time for the I/O agents program to read 64.8 GB of data under different numbers of cores

图 3 不同核数下 I/O 代理程序读取 64.8 GB 数据的时间

未考虑到超算集群的文件系统以及通信架构, 因此并未充分挖掘 Lustre 并行文件系统的并行读取能力, 并行带宽利用率较低.

3)通信的挑战

由图 4 可以看出 I/O 代理程序的通信不仅耗时较长且随着核数增加显著增长. 正如读取的挑战所述, I/O 代理程序并未考虑超算集群的节点通信架构, 只是简单地调用函数 *MPI_Scatter* 进行数据分发, 存在很大的通信负载不均衡, 超算集群通信通道利用率非常低.

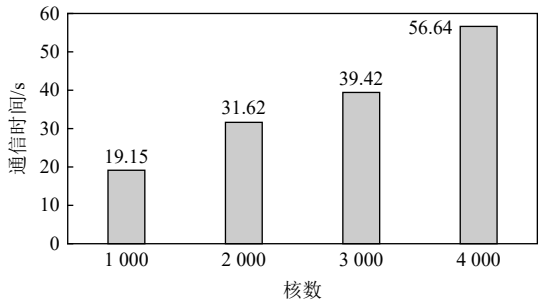


Fig. 4 Communication time of I/O agents program under different numbers of cores

图 4 不同核数下 I/O 代理程序的通信时间

4)写回的挑战

由图 5 可以看出 I/O 代理程序的写回十分的耗时, 且随着核数增加十分的不稳定. I/O 代理程序写回时采用各个进程直接写回对应数据块的策略, 这样的策略会导致海量的进程同时密集地访问同一个对象存储目标(object storage target, OST), 而 OST 并不能同一时间处理这么多 I/O 进程的请求, 造成极大的冲突, 特别是核数达到千以上时. 此外海量的进程同时访问同一块磁盘也会导致严重的磁头争用和海量的寻址.

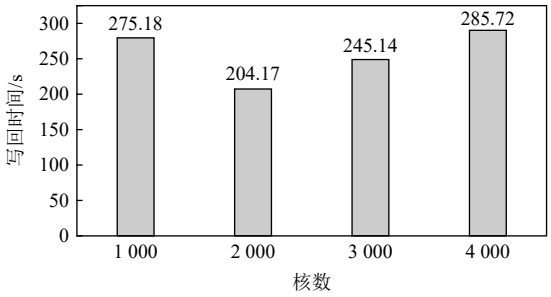


Fig. 5 Write back time of I/O agents program under different cores

图 5 不同核数下 I/O 代理程序的写回时间

1.4 本文贡献

本文针对 1.3 节所述的 I/O 代理程序存在的 4 个挑战, 通过分析同化程序整体的算法流程, 对算法进行重组并结合超算集群的存储架构、节点分布特性以及通信特性, 提出了基于计算拓扑图的 2 层负载均衡策略以及基于 Lustre 文件存储架构和超算集群特性的并行优化策略, 并在此基础上进一步提出了计算、读取通信、写回 3 层重叠策略. 本文的总技术路线如图 6 所示.

2 负载均衡优化

2.1 基于计算拓扑图的负载均衡优化

本文对海洋数据同化算法和进程实际的计算流程进行深入的分析, 观察到 *nobs* 才是各个进程实现

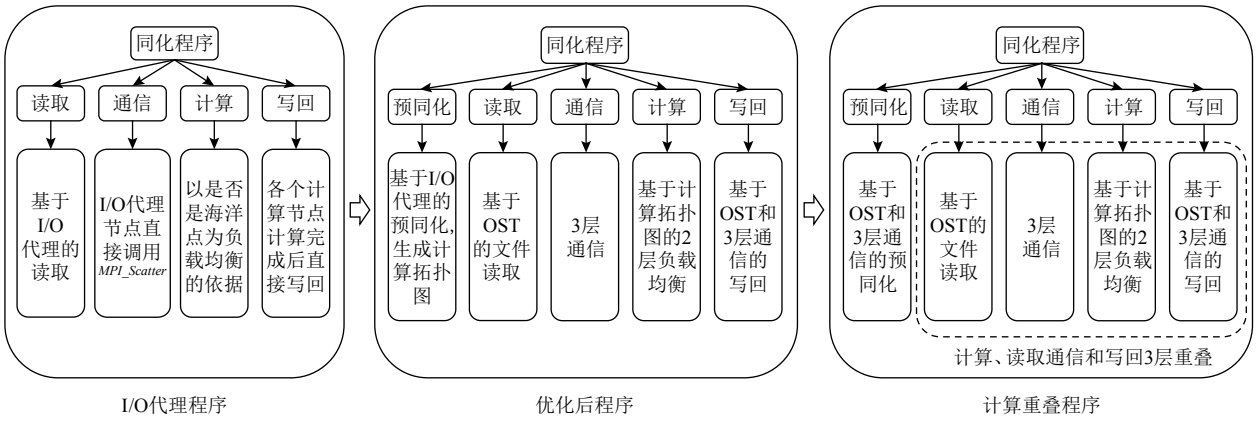


Fig. 6 General technical route

图 6 总体技术路线

负载均衡的关键. 而且实际同化过程中最耗时的部分是 SVD 分解过程, 其计算复杂度为 $nobs^3$, 其中 *nobs* 的取值范围为 3~1 000. 由此也可以看出基于海洋格点的负载均衡方法中, 有的海洋格点 *nobs* 很大, 有的海洋格点 *nobs* 很小, 计算复杂度最大相差 $1\,000^3/27$, 为 370 多万倍, 这样的负载均衡方法十分不合理. 本文算法针对实际执行流程中的这一特性, 在实际执行同化前加入了一个预同化步骤, 分别计算出每一个海洋格点进行数据同化的计算复杂度, 在原本海洋格点分布图的基础上生成基于计算复杂度的计算拓扑图, 并以此作为负载均衡划分的依据, 在最大程度上实现了各个进程的负载均衡. 不仅如此, 实现的计算拓扑图还可以很大程度上反映出本次数据同化计算任务的分布特性, 显示出整个海洋同化数据中的对应海洋格点是否需要同化, 以此实现了针对海洋同化数据计算分布特性的读取写回以及通信优化, 这部分将在 2.2 节进行介绍.

2.2 2 层负载均衡

如图 7 所示, 假设整个海洋同化数据的计算量为 90, 每个节点(Split 组)内 5 个核, 其中 0 号核为该 Split 组的 Master. 生成计算拓扑图后, 为了方便展示, 假设将整个同化数据分为 4 组(25, 20, 30, 15)并计算出每个组的计算量. 本文先在组的维度按照经度进行 1 层的负载均衡, 以第 1 组为例, 其计算量为 25, 本文算法为这个组分配 5 个 Master, 由于地理区域上的观测点分布是不均匀的, 因此每个 Master 负责的地理区域大小可能有大有小, 每个 Master 负责的计算量为 5. 在每个 Split 组内部, 本文算法再进行了 1 层的负载均衡, 将计算量再次平均划分给节点内的计算进程. 这里的 Split 组属于不同的节点, 分别使用节点间和节点内通信通道, 这样 2 层的负载均衡方式不仅可以最大限度地实现计算进程的负载均衡, 而且更为第 3 节的读取和通信分组做了铺垫, 第 3 节的读取和通信优化策略也充分考虑节点间和节点内的通信通道问题, 并根据负载均衡的分组, 实现了相应的读取和通信的优化策略. 通过这样的分组方式, 每个组的 Reader 对 Master 的数据分发, 以及 Master 对 Split 组的数据分发分别使用的是节点间和节点内的通信通道, 互不影响, 可以极大地提升通信效率.

3 基于 Lustre 文件存储架构和超算集群特性的并行优化

3.1 基于 OST 的读取优化

基于 LICOM 模式的数据同化本身涉及大量的文件读取, 在同化时间没有充分优化时, 读取时间可

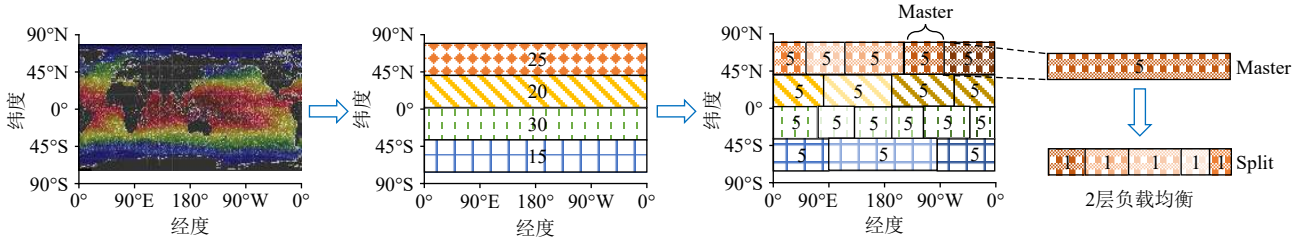


Fig. 7 Two-layer load balancing

图 7 2 层负载均衡

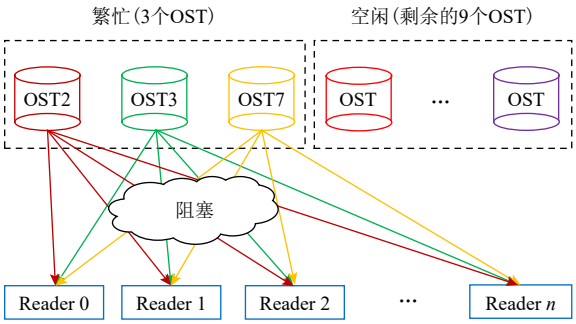
以忽略不计，例如在文献 [28] 的 Fortran 版本中，程序千核的运行时间为 8 000 s 左右，而读取时间占总时间的比重很小，可以忽略不计. 但在同化时间充分优化后，读取时间将在程序整体运行时间中占据很大的比重，特别是在本文负载均衡优化后，千核下程序其他部分的执行时间只有几十秒，而程序读取 64.8 GB 的数据需要 10 多秒，因此读取时间的优化也是提升并行同化程序性能的关键.

在传统的读取算法优化中，大多数的优化算法都是通过将数据重组，保证数据的成块连续读取，减少文件的寻址次数和 I/O 的请求次数，从而减少文件的读取时间. MPI-IO 是一种读取通信优化策略，采用的就是这个思想，许多读取密集型的算法以此作为读取的优化策略，并在 MPI-IO 的基础上进行了优化[30-34].

传统的读取优化大多是在算法层面对 I/O 进行优化，并没有考虑过读取进程是如何和底层文件存储系统进行交互的，也没有考虑超算集群的特性. 因此这些优化算法往往只能一定程度上优化程序的读取时间，如果旧的读取策略已经是成块连续读取的，新优化算法的性能提升将十分的有限，而且随着平台网络环境的不同有很大的波动. 本文提出的读取优化，将超算平台的文件存储架构和超算集群的特性结合起来，充分挖掘了 Lustre 文件存储架构的 I/O 带宽和超算集群的节点通信通道的分配特点[35-42]，程序读取 64.8 GB 文件的读取时间缩小到 1.6 s 左右，I/O 性能显著提升.

本文程序所使用的天河 2 号超算平台使用的文件系统为 Lustre 文件系统. 文件在存储时系统按照分片的方式，将文件存储到不同的 OST 中，其中对象存储服务器(object storage server, OSS)负责处理上层的 I/O 读写请求以及调度 OST. 在忽略 OSS 调度时间的影响下，如果 OST 的数目为 m，且 OST 的 I/O 带宽为 n(单位为 GBps)，那么理论上整个超算平台的带宽可以达到 mn(单位为 GBps). 在天河 2 号超算平台上，OST 的数目为 12，OSS 的数目为 2. 如果忽略掉 OSS 的调度时间，天河系统的整体带宽可达 $12n$(单位为 GBps). 在实测时，I/O 代理程序的吞吐量较低，远远没有达到天河的峰值吞吐量.

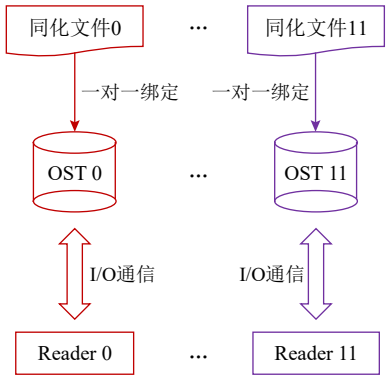
首先，如图 8，9 所示，I/O 代理程序的 I/O 读取算法存在大量的读取进程同时访问少量 OST 的情况，并且由于未对文件存储方式进行设置，默认情况下同一个文件可能无序地存在几个不同的 OST，这就导致了虽然读取的是不同的文件但不同的进程可能会访问相同的 OST. 这样的读取方式使得同一时间只有少量 OST 超负荷运行，其他 OST 则一直空闲，没有充分利用 OST，而且同时对少量 OST 进行密集访问会存在大量的 I/O 请求冲突，增加了 OSS 的调度难度，大大增加了 OSS 的调度时间. 在本文的算法中，对于 64.8 GB 也就是 12 个文件，每个文件 5.4 GB 的数据，本文通过 Lustre 的用户接口，将每个文件分别存到不同的 OST 中. 在文件读取时，本文只指定 12 个进程作为 Reader 进行文件读取. 对于这 12 个进程本文也做了特别的设定，本文将这 12 个进程分别绑定到不同的天河节点上. 2.2 节中提到，天河上的节点在通信时分别使用节点间和节点内的通信通道.



只保证算法上连续读取，但实际执行时I/O通信十分繁忙、无序且相互阻塞，OST利用率不高(n>11).

Fig. 8 Reading strategy for the I/O agents pragram

图 8 I/O 代理程序的读取策略



I/O通信有序，且互不影响，OST得到充分利用，实现完全并行读取.

Fig. 9 Reading strategy based on OST

图 9 基于 OST 的读取策略

如图 10 所示，天河上每个节点有 24 个核，每个节点独自占用 1 个节点间通信通道，节点内部则走节点内的通信通道. 这样的设计方式可以充分利用每个节点的通信通道的带宽，最大限度地避免对 OST 请求的冲突. 当 OSS 数目足够多时，每个读取进程和 OST 的 I/O 请求完全并行，不受其他读取进程的影响，实现完全并行的通信. 但是正如前文提到的天河的 OSS 只有 2 个，因此无法实现理论上 I/O 请求完

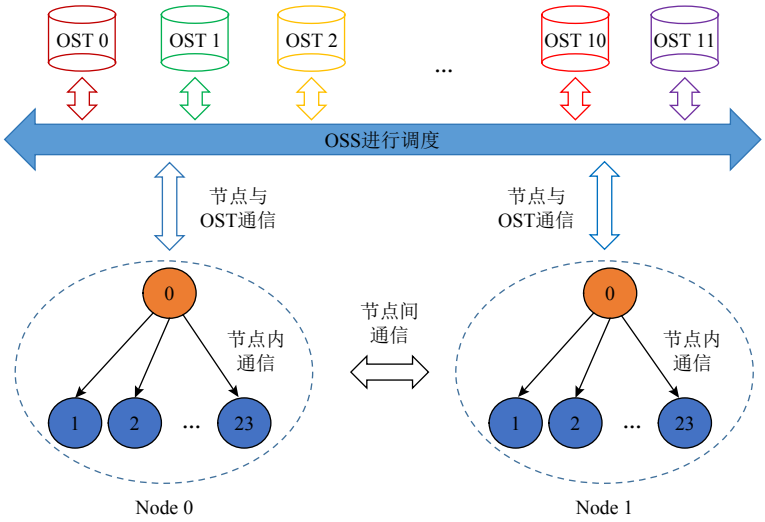


Fig. 10 The layout and communication mode of supercomputing cluster computing node

图 10 超算集群计算节点布局与通信方式

全没有冲突的情况，但是这种设计方式也可以最大限度地避免 I/O 请求的冲突，并且同一时间充分利用天河可用的 OST，实现完全并行读取.

3.2 3 层通信

1）3 层通信算法

本文的算法在读取时设置了 12 个 Reader 对 12 个文件进行完全并行的读取，接下来要解决 Reader 对读取到的数据进行分发的问题. 在 I/O 代理程序中，读取进程读取到相应的数据之后，就直接对所有组内的进程进行数据分发. 这样的实现方式虽然简单，但是效率却十分的低下. 在实际执行时 1 个读取进程可能要同时将数据分发给海量的进程，虽然 MPI 在底层优化时，采用二分的方法进行数据分发的优化，但是并没有关注图 10 中超算集群上节点内和节点间的通信通道问题. 这样可能存在一种情况，即在 *MPI_Scatter* 优化算法中，分发进程集中到 1 个或几个节点上，同时使用 1 个节点的通信通道向外分发数据，这样就会造成很大的网络拥塞，分发的效率十分的低下. *MPI_Scatter* 只是一种通用性的方法，实际执行的效果不佳.

本文根据图 10 所示的超算集群上节点间和节点内通信通道的配置特点设计了 3 层通信算法. 如图 11 所示，首先设置 12 个进程作为 Reader 与 OST 进行通信，获取到相应的数据，其中每个 Reader 分别分配到不同的节点上，独立使用自己的通信通道与自己对应的 OST 进行交互，与其他 Reader 互不影响实现并行的 OST 交互. 其次在 Reader 获取相应的数据之后，将计算进程按照节点为单位分为 Split 组，其中每个节点的 0 号进程为这个 Spilt 组的 Master. 0 号进程负责节点间的收发数据，独占当前节点的通信通道. 最

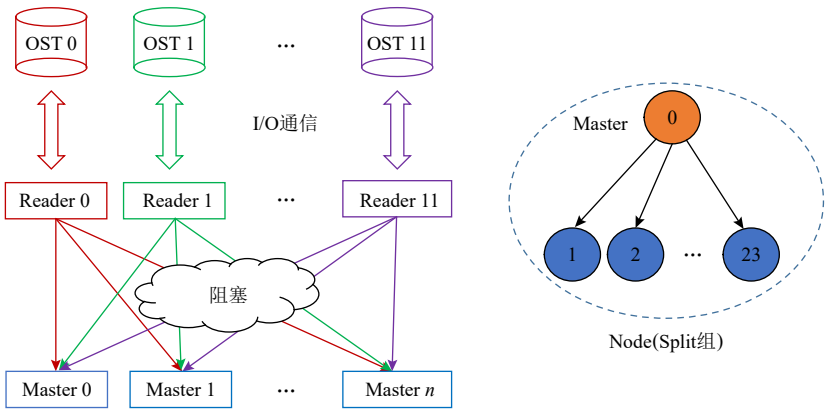


Fig. 11 Three layers of communication

图 11 3 层通信

后为了并行处理,本文将同化数据按照纬度 *ny* 方向横向切块,分成了 *group* 个 Master 组. 每个 Master 组都需要 12 个文件中对应的 1 块数据块,因此本文为每个 Master 组配置了 12 个 Reader,每轮由 12 个 Reader 去并行读取 12 个文件的对应数据块,并行分发给对应的 Master. 这样的设计方式可以将 Reader 和 Master 间的通信集中到节点间的通信通道上,充分利用各个节点通信通道的同时,实现和下文节点内通信的隔离. 在 Master 获取到相应的数据后,各个 Master 通过节点内的通信通道向自己负责的 Split 组分发数据. 这样设计方式可以实现节点间和节点内在不同通信通道的并行通信.

2)基于环的分发策略

每个 Master 组的 12 个 Reader 对各自的 Master 进行分发数据时,如果直接采用数据分发的方式,同一时间 12 个 Reader 同时对所有 Master 发起通信,可能会出现“hot spot”,导致严重的通信拥塞. 因此本文设计了图 12 所示的基于环的分发策略.

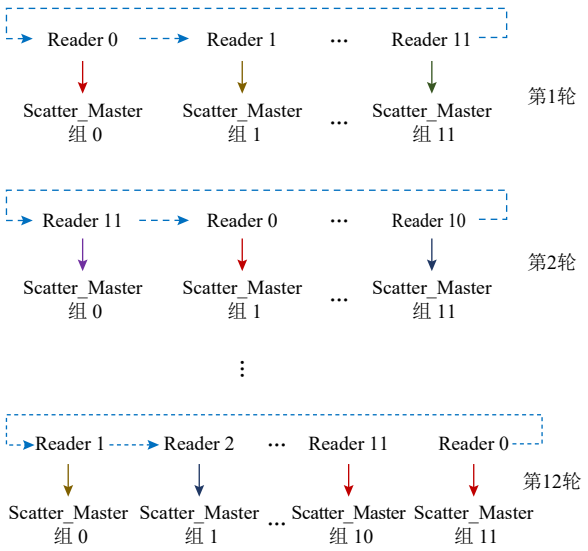


Fig. 12 Ring-based distribution strategy

图 12 基于环的分发策略

本文首先将每个 Master 组内的 Master 再次平均分组,生成 Scatter_Master 组,初始情况下每个 Reader 对应 1 个 Scatter_Master 组,负责向这部分的 Master 分发数据. 在 1 轮分发完成后,每个 Reader 进行轮转,如图 12 所示,依次循环直到所有的数据分发完毕. 这样的设计方式可以实现 Reader 间完全并行的数据分发,不会产生“hot spot”和 Reader 间的通信拥塞,极大地提升了通信的效率.

3.3 预同化优化和写回优化

1)预同化优化

第 2 节提到的基于计算拓扑图的负载均衡策略中需要进行一步预同化的操作,这部分需要先读取背景场的数据,然后根据背景场、观测场和网格数据进行预同化,计算出每个海洋格点的计算复杂度,以此生成计算拓扑图. 这个过程涉及一张背景场文件读取,这个背景场文件的大小为 5.4 GB. 在 1 000 核下,这部分的读取和通信时间为 6 s 左右,且随着核数呈线性增长. 本文算法在 4 000 核下优化到最后的整体运行时间为 34 s 左右,因此预同化部分的优化也十分的重要.

本文将 5.4 GB 的文件分别等分到 *group*(*group*<12)个 OST 中,然后每个 Master 组设置 1 个 Reader 对自己负责的 OST 进行并行读取. 在读取完成后仿照 3.2 节提到的 3 层通信,将数据分发给不同的组,各个组收到数据后进行预同化过程.

2)写回优化

I/O 代理程序中,写回时间占 I/O 代理程序运行时间的 27%,其时间为 200~300 s 之间,且随着超算集群用户数量和网络拥塞情况出现很大的波动. 有时写回时间甚至大于 1500 s,不仅十分的耗时且十分的不稳定. 这主要是由于 I/O 代理程序采用各个计算进程独自写回所同化的数据,这样的写回策略会对 OSS 发起数百万次的写回请求,同一时间 OSS 需要处理几千次的文件写回请求,这对 OSS 的调度和超算集群的网络通信产生非常大的压力,当超算集群用户增多、通信网络变得拥堵时,写回的性能会受很大的影响. 本文根据 Lustre 文件存储架构的特性,将写回的文件分别根据 Master 组等分到不同的 OST 中,然后根据 3 层通信,实现了 3 层的 gather 操作,将每个计算进程计算完成的数据经由 Split→Master→Reader 收集到每个 Master 组的 Reader 0 中,由每个组的 Reader 0 负责集中的写回,将对应的数据写回对应的 OST 中,实现完全并行的写回. 最后的写回请求的次数为 *group* 次,写回时间稳定在 2~3 s 且不随核数的增加而增加,十分的稳定,不随超算集群的用户数量、网络环境的波动而波动.

4 计算、读取通信、写回 3 层重叠

4.1 读取通信与计算重叠

由于在具体的数据同化过程中,每个计算进程只要获取到相应的数据,同化过程便可以进行,不需要其他的通信操作,各个计算进程完全并行. 针对这种同化过程,本文实现了对读取通信和计算的重叠.

为了实现读取通信与计算的重叠，Reader将需要读取的数据进行纬度*ny*方向的切片，生成*overlap*片数据块. 每次Reader只读取1个小块的数据，在读取到相应的数据后，再进行Reader到Master、Master到节点内计算进程的通信，各个计算进程在获取到数据后便进行同化计算. 在其他进程进行同化计算的同时，Reader再读取下一小片的数据，并和Master进行通信，这样便实现读取通信和计算的重叠.

由于同化的数据都是带边的，其中经度*nx*方向的*prep_rx*=105，纬度*ny*方向的*prep_ry*=19，假设每个计算进程所需数据块的*nx*方向子区域的长度为*sub_nx_len*，*ny*方向的区域分组的长度为*ny_len*，那么每个Reader需要读取的数据为((*nx*+2×*prep_rx*)×(*ny_len*+2×*prep_ry*))×*sizeof*(*type*)，每个计算进程所需的通信数据大小为(*sub_nx_len*+2×*prep_rx*×(*ny_len*+2×*prep_ry*))×*sizeof*(*type*). 因此，*ny*方向上数据切得越细，需要的带边数据也就越多，所产生的读取和通信冗余量就越大.

为了解决读取和通信数据冗余的问题，本文采用在读取时先提前开辟((*nx*+2×*prep_rx*)×(*ny_len*+2×*prep _ry*))×*sizeof*(*type*)的内存空间，其大小为不分片时的整体数据块的内存大小加上带边数据的内存大小，其中*ny_len*为区域分组中数据块在*ny*方向上的总长度. 然后设置了*psi_read_index*作为Reader已经读取的数据偏移，在Reader进行读取时根据*psi_read_index*计算下一次需要读取的数据量，即逻辑上需要的数据偏移减去*psi_read_index*代表的已经读入内存的数据偏移. 在实际执行时，分片后数据读取所读入的数据量和以不分片方式读入的数据量相等，完全解决了带边数据导致的读取数据冗余问题，实现了高效的数据复用.

解决完读取数据冗余的问题，下面介绍通信数据冗余问题的解决. 本文算法在通信之前，和读取相似，先提前开辟(*sub_nx_len*+2×*prep_rx*×(*ny_ len*+2×*prep_ry*))×*sizeof*(*type*)的内存空间，并设置*psi_index*为已经通过通信获取到的同化数据的数据偏移. 在通信时利用*psi_index*计算需要通信的数据量，即将需要通信的数据偏移减去已经通过通信的获取到的数据偏移. 通过这种方式，最终的通信数据量和未分片的数据量相等，完全解决了带边数据所导致的通信数据冗余问题.

具体的读取、通信与计算重叠算法和避免读取、通信冗余策略如图13、图14所示.

4.2 计算、读取通信、写回重叠

2.1节中提到计算拓扑图可以反映出本次数据同化计算任务的分布特性. 同化数据显示的是全球的海洋气候信息，所以包含大量的陆地以及大量没有观测点的海洋区域，而且这些区域多是成块且连续的. 因此根据海洋同化数据计算量的分布特性，设计了读取和通信避免策略.

在生成计算拓扑图后，对整个海洋同化数据进行分类，分为需要同化区域与不需要同化区域. 对于需要同化区域则根据上文的负载均衡算法进行基于负载均衡的任务划分以及同化数据读取、分发和计算. 对于不需要同化的区域，则可以设置一些Reader进行读取和写回，这部分区域虽然不用计算，但是为了生成全球海洋气候的分析场，这些数据也应该写回到最后的分析场中.

在本文的实验中，不需要同化的区域占总海洋同化数据的60%左右. 对于负责这部分读取和写回的Reader与其他进程相互独立，可以独立执行. 因此在生成计算量分布图后，这部分进程可以在其他进

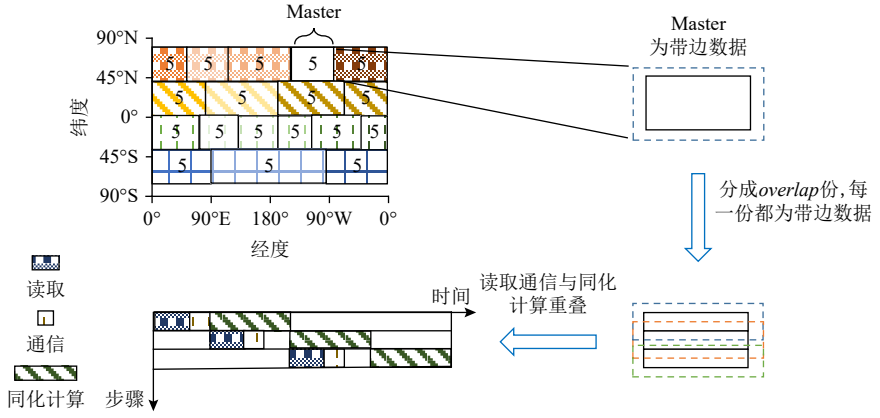


Fig. 13 The overlap between reading, communication and computation

图13 读取、通信与计算重叠

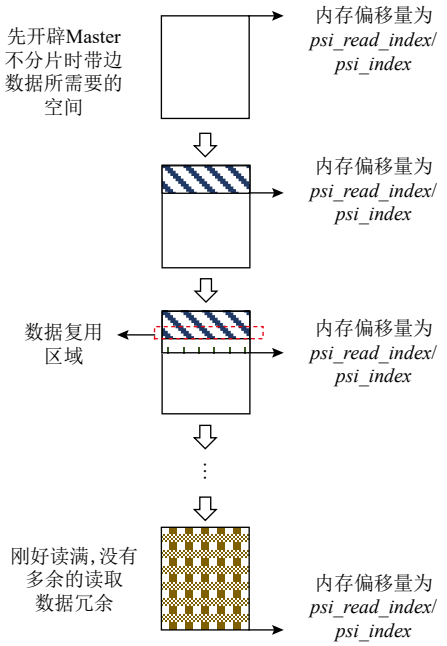


Fig. 14 The strategy of avoiding reading and communication redundancy

图 14 避免读取、通信冗余策略

程进行计算和读取通信的同时，并行地对不需要同化的数据进行写回. 这样的数据读取和写回，可以完全被其他进程的同化计算所掩盖掉，可以减少整体程序大量的数据读取通信和写回的时间. 经过上述的优化，计算进程由于只需要读取通信需要同化的40% 的区域，因此整体的读取和通信量大大减少，读取和通信时间也大大减少. 与此同时，虽然有一些海洋区域不需要进行同化，但是如果将其分发给计算节点，由于计算节点需要对海洋格点逐个进行遍历因此仍会存在一些判定处理的时间，在将这部分不需要同化计算的区域筛选出来后，只将需要同化计算的部分划分给计算节点，计算节点整体的同化计算效率也大大提升，同化计算时间也大大减少.

5 实验分析

本文采用的实验平台分别是天河 2 号超算集群和曙光 7000 超算集群，这 2 个平台所使用的数据集参数分别如表 1 和表 2 所示.

5.1 天河上各部分性能对比测试

1)读取时间测试

本文分别对 I/O 代理程序的读取通信策略和基于 Lustre 文件存储架构的、基于 3 层通信的读取通信策略进行了对比测试. 如图 15 所示，I/O 代理程序的读取较为耗时，本文程序的读取时间不仅远小于

Table 1 Parameters of Data Set Used on Tianhe

表 1 天河上使用的数据集参数

分辨率/(°)	网格精度	静态样本数	观测数据数	数据大小/GB
0.1	3 600×1 800×55	12	391 017	64.8

Table 2 Parameters of Data Set Used on Sugon 7000

表 2 曙光 7000 上使用的数据集参数

分辨率/(°)	网格精度	静态样本数	观测数据数	数据大小/GB
0.1	3 600×1 800×55	120	391 017	648

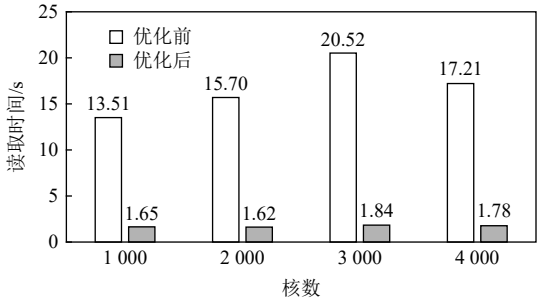


Fig. 15 Comparison of reading time between the two algorithms

图 15 2 种算法读取时间对比

I/O 代理程序的读取时间，且随核数的增加而基本保持在 2 s 左右.

2)通信时间测试

从图 16 可以看出 I/O 代理程序的通信时间随核数的增加而线性增加，而本文程序的通信时间随着核数的增加基本保持不变.

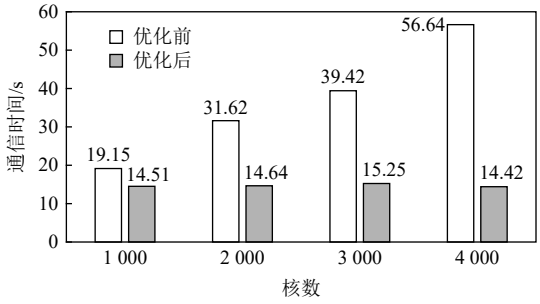


Fig. 16 Comparison of communication time between the two algorithms

图 16 2 种算法通信时间对比

3)数据同化时间测试

本文负载均衡优化前后的数据同化时间对比如图 17 所示. 基于计算拓扑图的负载均衡算法充分利用了可用的核数，避免了少部分核被分配大量计算量而大部分核只负责少部分甚至没有分配计量的情况，充分利用了超算集群的计算性能，使得同化时间大大缩短.

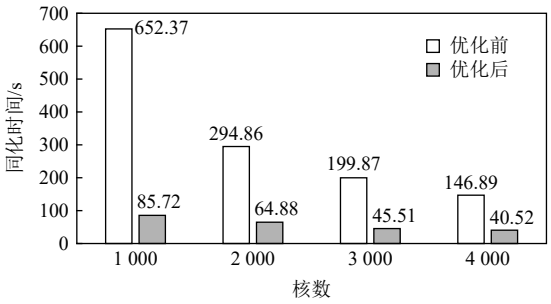


Fig. 17 Comparison of assimilation time between the two algorithms

图 17 2 种算法同化时间对比

4)写回时间测试

本文分别对 2 种算法的写回时间做了测试，如表 3 所示. I/O 代理程序的写回时间十分的长，而且在实际测试时表现得十分不稳定，本文程序的写回时间不仅远小于 I/O 代理程序的写回时间且随着核数的增加基本保持稳定.

Table 3 Comparison of Write Back Time Between the Two Algorithms

表 3 2 种算法的写回时间对比

核心数	优化前写回时间/s	优化后写回时间/s
1 000	275.18	2.58
2000	204.17	2.50
3 000	245.14	2.69
4 000	285.72	2.51

5)预同化时间测试

本文预同化过程涉及在千核下 5.4 GB 文件的读取和分发. 从图 18 可以看出. 在未优化前，预同化时间随着核数的增加呈线性增长；优化后随着核数的增加预同化时间基本保持稳定.

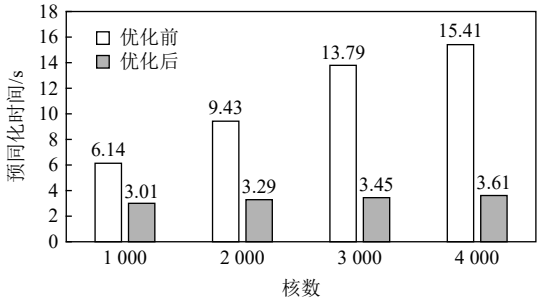


Fig. 18 Comparison of pre-assimilation time before and after optimization

图 18 优化前后预同化时间对比

6)整体性能测试

本文将展示整体并行同化程序优化前后的运行时间对比. 如图 19 所示，分别显示了 I/O 代理程序、本文非计算重叠版本和本文最终优化版本的程序整体运行时间对比. 从图中可以看出 I/O 代理程序的计算、读取和写回时间都比较大. 虽然计算时间随着核数的增加而明显减少，但是文件的 I/O 读取通信时间随着核数的增加而逐渐增加. 同时 I/O 代理程序的写回时间随着核数增加也十分不稳定. 在非计算重叠算法中，可以看出程序的写回时间和预同化时间仍占据一定的比重，且预同化时间随着核数的增加而增加. 在本文最终的优化版本中除了计算、读取通信和写回重叠的运行时间以外，其他部分的运行时间占总时间的比例不大. 而且对比前后 2 种优化算法，

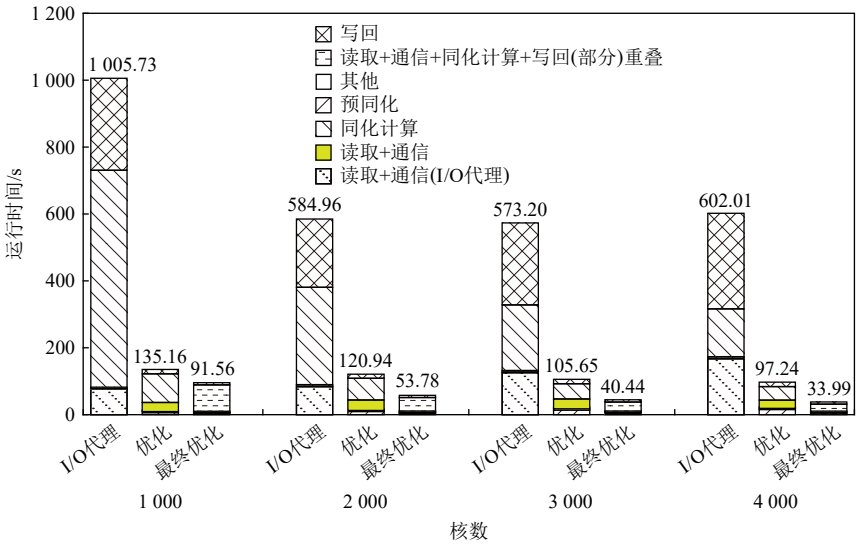


Fig. 19 The overall running time of thousand cores program on Tianhe

图 19 天河上千核程序整体运行时间

可以明显看出读取通信以及写回的时间，在计算时间足够长时实现了完全的重叠. 同时也可以看出由于筛选出了 60% 不需要同化计算的区域，读取和通信的数据量大大减少，本文最终优化版本的读取和通信时间大大减少. 而且在同化计算中由于省去了 60% 同化区域判定处理的时间，整体海洋数据同化程序的同化计算时间也大大减少. 总体来说经过重叠优化后本文最终优化版本相比于之前优化的版本性能得到了很大的提升. 在 4 000 核下本文程序的整体性能相比 I/O 代理程序提升了 18 倍.

5.2 曙光 7000 上整体性能对比测试

在曙光 7000 超算集群上本文算法仍然表现优异. 由于节点布局与天河不同，本文在节点分配上做了调整. 曙光 7000 是 Parastor 文件系统，并且未提供用户处理节点(类似于 Lustre 文件系统的 OST)的接口，但节点(OST)、节点池(OSS)众多，更加偏向于随机散列的读取，因此本文在文件读取和通信优化上取消了 Reader 这一层，采用 Master 直接和 OST 进行交互，以最大限度地利用 Parastor 节点 OST 的独立通道，提高并行读取效率. 取消 Reader 这一层，Master 直接和 Split 组进行交互，也将提升 Master 到计算进程的通信效率. 在图 20 中分别列出了预同化、读取、通信、同化计算和写回时间，其他比重过小的时间这里未做展示. 由于曙光 7000 超算集群上的数学库还不完善，本文使用的 MAGMA 数学库计算效率很低，虽然使用了 DCU 卡进行加速，但是其计算速度仍慢于天河上的 MKL 库函数，但这并不影响本文算法程序与 I/O 代理程序性能的对比. 在使用相同库函数的情况下本文算法程序性能明显优于 I/O 代理程序，在 4 000 块卡下整体性能提升 8 倍左右.

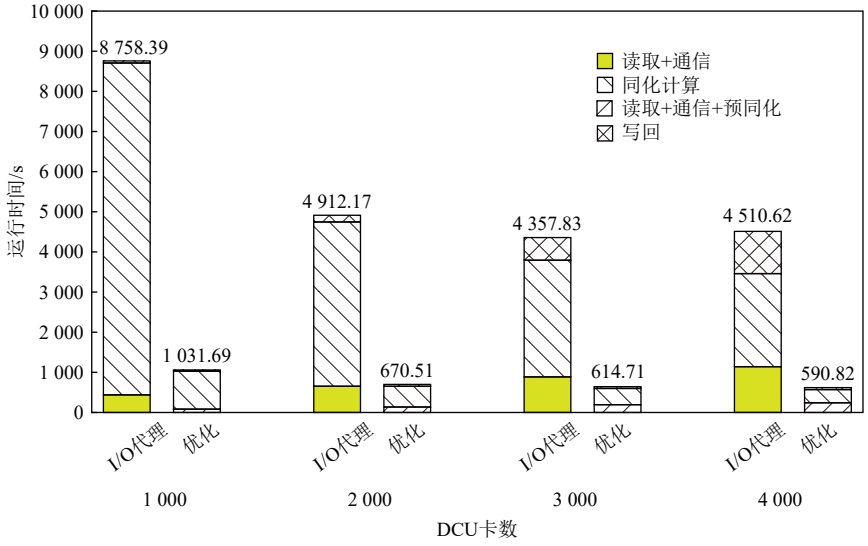


Fig. 20 The overall running time of thousand DCU card program on Sugon

图 20 曙光上千块 DCU 卡程序整体运行时间

6 结束语

本文主要介绍了基于 LICOM 模式的海洋数据同化程序的优化算法. 分别提出了基于计算拓扑图的负载均衡策略，基于 Lustre 文件存储架构的读取、写回和通信策略，并在此基础上实现了计算、读取通信、写回的 3 层重叠. 相比于 I/O 代理程序，本文程序的各项性能都得到了显著的提升. 在天河上，4 000 核下本文程序整体性能提升 18 倍. 在曙光 7000 上，4 000 块卡下，整体性能提升了 8 倍左右.

但是本程序仍有可优化的空间，由于随着核数的增加计算时间不断地缩短，而 I/O 时间随着核数的增加不会减少，因此当核数足够大时计算的时间会小于 I/O 时间，计算时间无法再掩盖 I/O 时间，这样会限制程序的可拓展性上限. 由于曙光 7000 的存储架构以及节点分布特性与天河不同，再加上曙光 7000 文件系统用户接口的限制，虽然本文针对其做了相应的优化，尽可能地将 Parastor 文件系统的节点(OST)利用起来，但是其仍达不到像天河 Lustre 文件系统一样极限的 I/O 带宽利用率. 因此在未来的工作中准备进一步优化 I/O 时间以提升程序的可拓展性上限，并对曙光 7000 的文件存储架构做进一步的研究，对曙光 7000 上的程序做进一步的优化.

作者贡献声明：蔡迪负责论文相关的实验设计、代码实现、实验测试以及论文撰写等工作；洪学海负责前期实验方案的设计讨论与论文修改；肖俊敏负责论文结构的讨论和修改；谭光明负责实验补充和设计讨论，并指导论文修改.

参 考 文 献

[1] De Luca P, Galletti A, Giunta G, et al. Recursive filter based GPU algorithms in a data assimilation scenario[J]. Journal of Computational Science, 2021, 53: 101339

[2] Teruzzi A, Di Cerbo P, Cossarini G, et al. Parallel implementation of a data assimilation scheme for operational oceanography: The case of the MedBFM model system[J]. Computers & Geosciences, 2019, 124: 103−114

[3] Yan Hongxiang, Zarekarizi M, Moradkhani H. Toward improving drought monitoring using the remotely sensed soil moisture assimilation: A parallel particle filtering framework[J]. Remote Sensing of Environment, 2018, 216: 456−471

[4] Molteni F. Atmospheric simulations using a GCM with simplified physical parametrizations. I: Model climatology and variability in multi-decadal experiments[J]. Climate Dynamics, 2003, 20(2): 175−191

[5] Emerick A A, Reynolds A C. Ensemble smoother with multiple data assimilation[J]. Computers & Geosciences, 2013, 55: 3−15

[6] Rodell M, Houser P R, Jambor U E A, et al. The global land data assimilation system[J]. Bulletin of the American Meteorological Society, 2004, 85(3): 381−394

[7] Houtekamer P L, Mitchell H L. A sequential ensemble Kalman filter for atmospheric data assimilation[J]. Monthly Weather Review, 2001, 129(1): 123−137

[8] Evensen G. The ensemble Kalman filter: Theoretical formulation and practical implementation[J]. Ocean Dynamics, 2003, 53(4): 343−367

[9] Bei Naifang, Foy B, Lei Wenfang, et al. Using 3DVAR data assimilation system to improve ozone simulations in the Mexico City basin[J]. Atmospheric Chemistry and Physics, 2008, 8(24): 7353−7366

[10] Li Yongzuo, Wang Xuguang, Xue Ming. Assimilation of radar radial velocity data with the WRF hybrid ensemble-3DVAR system for the prediction of hurricane Ike (2008)[J]. Monthly Weather Review, 2012, 140(11): 3507−3524

[11] Clayton A M, Lorenc A C, Barker D M. Operational implementation of a hybrid ensemble/4D-Var global data assimilation system at the met office[J]. Quarterly Journal of the Royal Meteorological Society, 2013, 139(675): 1445−1461

[12] Powell B S, Arango H G, Moore A M, et al. 4DVAR data assimilation in the intra-Americas sea with the regional ocean modeling system[J]. Ocean Modelling, 2008, 25(3/4): 173−188

[13] Clark C, Harrison D E, Johnson M, et al. An overview of global observing systems relevant to GODAE[J]. Oceanography, 2009, 22(3): 22−33

[14] Wu Junjie, Liu Yong, Zhang Baida, et al. A benchmark test of boson sampling on Tianhe-2 supercomputer[J]. National Science Review, 2018, 5(5): 715−720

[15] Xue Wei, Yang Chao, Fu Haohuan, et al. Ultra-scalable CPU-MIC acceleration of mesoscale atmospheric modeling on Tianhe-2[J]. IEEE Transactions on Computers, 2014, 64(8): 2382−2393

[16] Zhang Xianyi, Yang Chao, Liu Fangfang, et al. Optimizing and scaling HPCG on Tianhe-2: Early experience[C]//Proc of the 14th Int Conf on Algorithms and Architectures for Parallel Processing. Cham, Switzerland: Springer, 2014: 28−41

[17] Xue Wei, Yang Chao, Fu Haohuan, et al. Enabling and scaling a global shallow-water atmospheric model on Tianhe-2[C]//Proc of the 28th IEEE Int Parallel and Distributed Processing Symp. Piscataway, NJ: IEEE, 2014: 745−754

[18] Chen Cheng, Du Yunfei, Jiang Hao, et al. HPCG: Preliminary evaluation and optimization on Tianhe-2 CPU-only nodes[C]//Proc of the 26th IEEE Int Symp on Computer Architecture and High Performance Computing. Piscataway, NJ: IEEE, 2014: 41−48

[19] Liao Xiangke, Xiao Liquan, Yang Canqun, et al. MilkyWay-2 supercomputer: System and application[J]. Frontiers of Computer Science, 2014, 8(3): 345−356

[20] Li Dali, Xu Chuanfu, Wang Yongxian, et al. Parallelizing and optimizing large-scale 3D multi-phase flow simulations on the Tianhe-2 supercomputer[J]. Concurrency and Computation: Practice and Experience, 2016, 28(5): 1678−1692

[21] Chow E, Liu Xing, Misra S, et al. Scaling up Hartree-Fock calculations on Tianhe-2[J]. The International Journal of High Performance Computing Applications, 2016, 30(1): 85−102

[22] Xiao Junmin, Wang Shijie, Wan Weiqiang, et al. S-EnKF: Co-designing for scalable ensemble Kalman filter[C]//Proc of the 24th Symp on Principles and Practice of Parallel Programming. New York: ACM, 2019: 15−26

[23] Guan Zhibing, Xiao Junmin, Ji Tongkai, et al. The impact of different matrix decomposition methods on ocean data assimilation[J]. Journal of Frontiers of Computer Science and Technology, 2019, 13(1): 147−157 (in Chinese)
(管志斌, 肖俊敏, 季统凯, 等. 不同矩阵分解方法对海洋数据同化的影响[J]. 计算机科学与探索, 2019, 13(1): 147−157)

[24] Xiao Junmin, Zhang Guizhao, Gao Yanan, et al. Fast data-obtaining algorithm for data assimilation with large data set[J]. International Journal of Parallel Programming, 2020, 48(4): 750−770

[25] Liu Hailong, Lin Pengfei, Zheng Weipeng, et al. A global eddy-resolving ocean forecast system in China-LICOM forecast system (LFS)[J/OL]. Journal of Operational Oceanography, 2021: 1−13 [2021-05-19].https://www.tandfonline.com/doi/full/10.1080/1755876X.2021.1902680

[26] Lin Pengfei, Yu Zhipeng, Liu Hailong, et al. LICOM model datasets for the CMIP6 ocean model intercomparison project[J]. Advances in Atmospheric Sciences, 2020, 37(3): 239−249

[27] Zou Liwei, Zhou Tianjun, Tang Jianping, et al. Introduction to the regional coupled model WRF4-LICOM: Performance and model intercomparison over the western north pacific[J]. Advances in Atmospheric Sciences, 2020, 37(8): 800−816

[28] Wan Weiqiang, Xiao Junmin, Hong Xuehai, et al. Parallel implementation and optimization of large scale ocean data assimilation algorithm[J]. Computer Engineering & Science, 2019, 41(5): 765−772 (in Chinese)
(万威强, 肖俊敏, 洪学海, 等. 面向大规模海洋数据同化算法的并行实现及优化[J]. 计算机工程与科学, 2019, 41(5): 765−772)

[29] Nino-Ruiz E D, Sandu A, Deng X. A parallel implementation of the ensemble Kalman filter based on modified Cholesky decomposition[J]. Journal of Computational Science, 2019, 36: 100654

[30] Thakur R, Gropp W, Lusk E. On implementing MPI-IO portably and with high performance[C]//Proc of the 6th Workshop on I/O in Parallel and Distributed Systems. New York: ACM, 1999: 2332

[31] Thakur R, Lusk E, Gropp W. Users guide for ROMIO: A high-performance, portable MPI-IO implementation[R]. Chicago, Illinois: Argonne National Lab, 1997

[32] Prost J P, Treumann R, Hedges R, et al. MPI-IO/GPFS, an optimized implementation of MPI-IO on top of GPFS[C]//Proc of the 15th ACM/IEEE Conf on Supercomputing. Piscataway, NJ: IEEE, 2001: 58−58

[33] Liu Weifeng, Ureña I A C, Gerndt M, et al. Automatic MPI-IO tuning with the periscope tuning framework[C]//Proc of the 28th IEEE Int Parallel & Distributed Processing Symp Workshops. Piscataway, NJ: IEEE, 2014: 352−360

[34] Puri S, Paudel A, Prasad S K. MPI-Vector-IO: Parallel I/O and partitioning for geospatial vector data[C/OL]//Proc of the 47th Int Conf on Parallel Processing. New York: ACM, 2018[2021-04-15].https://dl.acm.org/doi/abs/10.1145/3225058.3225105

[35] He Weilie. MPI point to point communication on Linux cluster architecture[J]. Journal of Shenyang Aerospace University, 2007, 24(3): 61−63 (in Chinese)
(何卫列. 基于 Linux 集群架构的 MPI 点对点通信研究[J]. 沈阳航空工业学院学报, 2007, 24(3): 61−63)

[36] Wang Wei, Li Wang. Improvement of MPI-IO programming interface based on Lustre file system[J]. Application of Electronic Technique, 2012, 38(5): 128−131 (in Chinese)
(王巍, 李旺. 基于 Lustre 文件系统的 MPI-IO 编程接口改进[J]. 电子技术应用, 2012, 38(5): 128−131)

[37] Tsujita Y, Hori A, Ishikawa Y. Striping layout aware data aggregation for high performance I/O on a Lustre file system[C]//Proc of the 17th Int Conf on High Performance Computing. Cham, Switzerland: Springer, 2015: 282−290

[38] Tsujita Y, Hori A, Kameyama T, et al. Topology-aware data aggregation for high performance collective MPI-IO on a multi-core cluster system[C]//Proc of the 4th Int Symp on Computing and Networking (CANDAR). Piscataway, NJ: IEEE, 2016: 37−46

[39] Yu Weikuan, Vetter J, Canon R S, et al. Exploiting Lustre file joining for effective collective IO[C]//Proc of the 7th IEEE Int Symp on Cluster Computing and the Grid (CCGrid'07). Piscataway, NJ: IEEE, 2007: 267−274

[40] Dickens P M, Logan J. A high performance implementation of MPI-IO for a Lustre file system environment[J]. Concurrency and Computation: Practice and Experience, 2010, 22(11): 1433−1449

[41] Logan J, Dickens P. Towards an understanding of the performance of MPI-IO in Lustre file systems[C]//Proc of 2008 IEEE Int Conf on Cluster Computing. Piscataway, NJ: IEEE, 2008: 330−335

[42] Dickens P, Logan J. Towards a high performance implementation of MPI-IO on the Lustre file system[C]//Proc of OTM Confederated Int Conf on the Move to Meaningful Internet Systems. Berlin: Springer, 2008: 870−885

Cai Di, born in 1995. Master. His main research interest is high performance computing.
蔡　迪, 1995 年生. 硕士. 主要研究方向为高性能计算.

Hong Xuehai, born in 1967. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include high performance computing, big data, and cloud computing.
洪学海, 1967 年生. 博士, 教授, 博士生导师, CCF 高级会员. 主要研究方向为高性能计算、大数据和云计算.

Xiao Junmin, born in 1989. PhD. His mian research interests include high performance computing, parallel optimization, and supercomputing for AI.
肖俊敏, 1989 年生. 博士. 主要研究方向为高性能计算、并行优化和 AI 的超级计算.

Tan Guangming, born in 1980. PhD, professor, PhD supervisor. Senior member of CCF. His main research interests include high performance computing and parallel computing.
谭光明, 1980 年生. 博士, 教授, 博士生导师. CCF 高级会员. 主要研究方向为高性能计算和并行计算.