研究论文

海洋环流模式 NEMO 的代码现代化

周生昌[1,2]，刘卫国[1,3]，宋振亚[2,3,4]，杨晓丹[2,3*]

（1. 山东大学 软件学院，山东 济南 250101；2. 自然资源部 第一海洋研究所，山东 青岛 266061；
3. 青岛海洋科学与技术试点国家实验室 区域海洋动力学与数值模拟功能实验室，山东 青岛 266237；
4. 海洋环境科学和数值模拟自然资源部重点实验室，山东 青岛 266061）

摘 要：海洋数值模式是精准海洋环境预报的核心手段。随着计算分辨率的不断提高，海洋数值模式对计算性能的要求也越来越高。为了提高模式计算性能，充分发挥现代计算机的特点，选取海洋环流模式 NEMO 开展了代码现代化优化方案在海洋环流模式中的应用研究。首先使用 Intel 性能分析工具对模式的计算性能进行了分析；随后，针对热点函数，开展了编译选项优化、标量串行代码优化、SIMD 优化、内存带宽优化以及多进程扩展。结果显示：经过以上优化步骤，在不增加任何硬件成本的前提下，模式整体性能可提升 31%，且在多进程下表现出了较好的负载均衡性。这表明，本研究采用的优化策略是一种切实可行的方法。在此基础上，进一步对代码现代化过程中出现的显著影响计算效率的问题，如大量指针的使用阻止矢量化、循环嵌套过多、内存带宽占用过高等，开展了分析和讨论，为未来模式的设计和改进提供了参考和建议。

关键词：代码现代化；海洋环流模式；NEMO；Intel 性能分析工具；SIMD

中图分类号：P73 **文献标志码**：A **文章编号**：1671-6647(2021)01-0062-11

doi：10.3969/j.issn.1671-6647.2021.01.007

引用格式：ZHOU S C, LIU W G, SONG Z Y, et al. Code modernization optimization of ocean general circulation model NEMO[J]. Advances in Marine Science, 2021, 39(1): 62-72. 周生昌，刘卫国，宋振亚，等. 海洋环流模式 NEMO 的代码现代化[J]. 海洋科学进展，2021，39(1)：62-72.

研究海洋的手段主要有理论研究、观测试验和数值模拟，其中观测试验是数值模拟和理论研究的基础，理论研究为数值模拟和观测试验设计提供指导，而数值模式则集成了人们从观测数据和理论研究中所获得的知识和理解，不仅可以用来验证理论假说和弥补观测资料的不足，而且是能够最终用于预测未来变化的唯一工具[1]。随着海洋和气候变化研究的不断深入，高性能计算机的发展，海洋数值模式正逐步朝着更高分辨率、更多物理过程和更快计算速度的方向发展[2]。一般来说，水平分辨率每提高 1 倍，计算量会增大到原来的 8～10 倍[3]。同时，随着分辨率的提高，会有越来越多的原次网格不能描述的物理过程被包含进来，而新的次网格过程的参数化也需要通过将多组数值试验结果与实测数据进行比较来进行评估和调整，这显然也提高了对计算量和计算能力的要求[4]。因此，充分利用和发挥高性能计算机的性能来提高海洋数值模式计算速度已成为模式发展的一个必要条件。

目前高性能海洋数值模拟的研究主要采用以下几种方案提高模式计算能力：1）基于分布式内存并行化程序。主要采用 MPI（Message Passing Interface）消息传递接口使计算任务合理分布在多个独立进程的计算单元内，并通过高速网络进行数据通信。该方法实现复杂，需考虑边界交换、负载均衡等问题，虽然可扩展

收稿日期：2019-12-02

资助项目：国家自然科学基金项目——海量数据驱动下的高分辨率海洋数值模式关键算法研究（U1806206）和新型海洋与气候模式的发展（41821004）；自然资源部基本科研业务费专项资金项目——ENSO，PDO 和 AMO 的非线性调制机理研究（GY0219Q08）

作者简介：周生昌（1997—），男，硕士研究生，主要从事高性能计算方面研究. E-mail：sc.zhou@mail.sdu.edu.cn

* **通信作者**：杨晓丹（1988—），女，助理研究员，博士，主要从事气候变化和高性能计算方面研究. E-mail：yangxiaodan@fio.org.cn

（李 燕 编辑）

性好，但节点之间交换会带来通讯开销。2）基于共享式内存并行化程序。主要采用 OpenMP 等接口将计算任务分配给多个线程，并通过共享内存地址的方式实现数据交换。该方法实现相对容易，无通讯开销，但一般来说无法跨节点，这限制了其并行规模。3）基于进程/线程并行化程序。该方法是以上 2 种并行方式的结合，实现较为复杂，在异构计算机（如 CPU ＋ GPU、国产申威处理器等）上效果特别显著。

上述 3 种方案主要从并行扩展方面提高模式计算性能，这需要大量的计算资源，且程序在设计和实现中并未考虑代码现代化。特别是随着高性能计算机的发展，这些方案已经无法充分利用和发挥现代计算机的优势。杨晓丹等[3]在海浪模式 MASNUM 上开展了代码现代化优化方面的尝试，通过检查代码的规范性、合理性，并针对语言特性对代码进行改进，使之更好地适应计算机基础架构，充分发挥了现代计算机的计算特点。但是在此海浪模式中，计算量最大的源函数与相邻点无关，无分区间数据交换通讯，并且由于分支判断少，cache 命中率高[5]，这与海洋环流模式 NEMO（Nucleus for European Modeling of the Ocean）[6]计算特点有较大的差异。NEMO 不但存在大量的数据交换，而且分支判断较多。因此如何使用代码现代化方法优化和提高海洋环流模式的计算性能，非常具有代表性和必要性。

为了充分利用和发挥现代高性能计算机的性能，进一步促进海洋环流数值模式的应用和发展，本文基于代码现代化的概念，提出了海洋环流模式代码现代化优化的方案，并在此基础上对海洋环流模式 NEMO 进行了优化，分析和探讨了代码现代化优化方案在海洋环流模式应用中的前景和局限性，为今后海洋环流数值模拟的研究和发展提供参考和建议。

1　优化方案设计

1.1　优化策略

现代高性能计算机是由多核和众核处理器、高速缓存和内存、高带宽处理器以及它们之间的通信结构和高速 I/O 组合而成的[7]。为了充分适应平台的现代化结构，充分发挥平台的性能，需要发展高性能和现代化的软件，包括对原始代码进行改进，以及为现代计算机重新设计应用程序以获得最大性能，但这些均需要充分利用高性能计算机硬件资源，这是代码现代化的基础。

本文基于代码现代化优化策略，并结合 NEMO 海洋环流模式的特点以及现有平台架构，制定了 6 个阶段的优化策略，如图 1 所示。为了保证结果的准确性，排除平台因素引起的误差，每次均取 2 次试验结果的平均值作为最终结果。

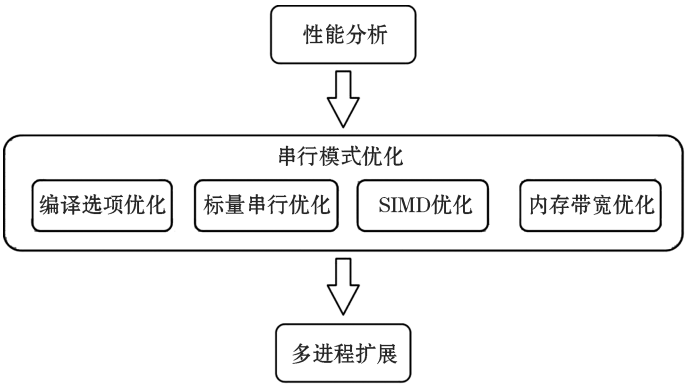

图 1　代码现代化优化策略

Fig.1　Code modernization optimization strategy

1.2　试验算例

试验算例为全球海洋环流真实算例 ORCA1_LIM，该试验算例只包含海洋动力学、热力学（OPA 模块）以及海冰动力学（LIM 模块）。模式共积分 120 个时间步长，水平网格数目为 362×292，垂向网格数目为 75。为了减小 I/O 对扩展性的影响，本研究中的测试是针对模式计算性能的无 I/O 测试。

1.3　测试环境

试验测试环境如表 1 所示，单节点内共有两颗 Intel Xeon Gold 6252 处理器，每颗处理器有 24 核心，因

此单节点共有 48 物理核心，96 逻辑核心，内存为 384 GB DDR4。

表 1 测试环境

Table 1 The testing environment

项 目	参 数
CPU	2×Intel Xeon Gold 6252，每颗 24 核心，主频 2.10 GHz
内存	384GB DDR4
硬盘	4TB
操作系统	CentOS 7
编译器	Intel version 18.0.3
性能分析工具	Intel Vtune Amplifier 2018.3，Intel Trace Analyzer Collector (ITAC) 2018.3
MPI	mpif90 for the Intel MPI Library 2018 Update 3 for Linux
默认编译选项	-i4 -r8 -O2

2 NEMO 优化

2.1 定位热点函数

为了更准确地获取影响 NEMO 模式计算效率的瓶颈，从而更有针对性地对 NEMO 模式进行优化，首先使用 Intel Vtune Amplifier 工具定位出模式的热点函数，即耗时较多的函数，如图 2 所示。结果显示 NEMO 的热点函数极为分散，即热点函数较多，耗时分布较为均衡，本研究选取前五个热点函数(tra_ldf_iso，lim_rhg，tra_adv_tvd，ldf_slp、nonosc)进行重点优化。

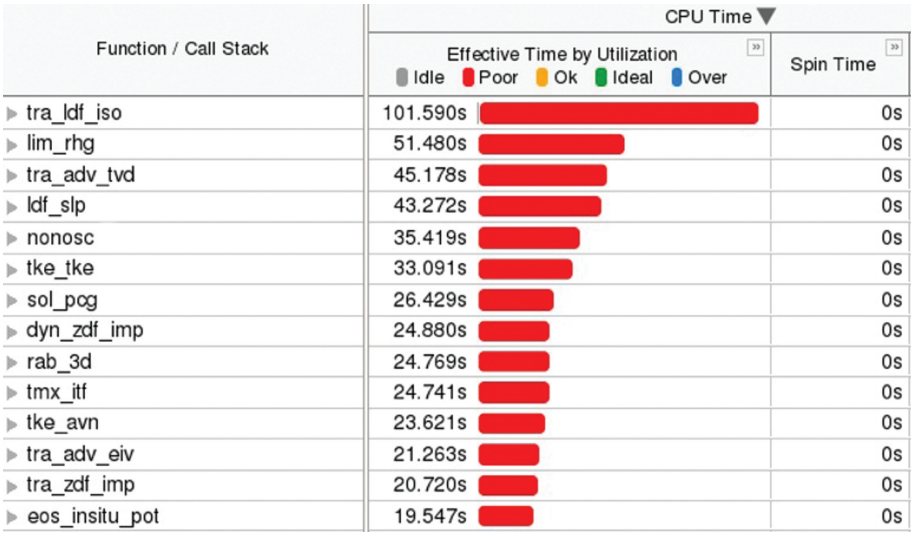

图 2 串行模式各函数计算时间

Fig.2 Serial mode function calculates time

2.2 编译选项优化

在性能分析的基础上，本研究首先使用编译选项对模式进行优化。使用 xHOST 选项使编译器生成处理器支持的最高指令集，该选项在本文测试环境下等效于 xMIC-AVX512；使用 ipo 启用文件之间的过程间优化，使编译器对单独文件中定义的函数执行内联函数扩展；使用 no-prec-div 选项对模式的浮点数除法进行优化，将除法改为乘倒数的形式进行计算，从而提高运算效率；此外，还使用了 fp-model fast=2，qopt-dy-

namic-align，qopt-prefetch 等优化选项，具体如表 2 所示。在经过编译选项优化后，NEMO 的整体加速比可以达到 1.21 倍。通过对输出结果进行分析，以海表面温度为例，得出 O1 和 O3 两种编译选项所引起的误差绝对值不超过 9.3×10^{-6}℃，占仅占平均海表面温度的 0.000 05%，且误差绝对值在 1.0×10^{-8}℃以下的网格点占比超过 99.8%，满足精度要求。同样，对其它变量的检查也满足精度要求。因此编译选项优化是有一种有效的优化手段。

表 2　编译选项优化列表

Table 2　The list of compilation option optimization

配置文件	作　用
O3	启用最高优化级别
xHOST	生成主机支持的最高指令集
ipo	启用过程间优化
no-prec-div	优化浮点数除法
fp-model fast＝2	启用更积极的浮点数运算优化
qopt-dynamic-align	数组动态对齐
qopt-prefetch	启用预取插入优化

2.3　标量串行优化

标量串行优化的目的是通过对热点函数代码进行调整修改，如去除重复计算、减少条件分支、降低循环嵌套等，确保代码使用最少的计算量和合适的精度获得正确的结果。下面以 traldf_iso 热点函数为例介绍此优化过程。

在如下例子中（图 3a），zdk1t 和 zdkt 计算公式是相同的，即 zdk1t(ji,jj,jk－1) ＝ zdkt(ji,jj,jk)，因此通过将 zdk1t 从循环中移除，在计算完 zdkt 后再将结果赋值给 zdk1t，可以减少重复性计算，如图 3b 所示：

```
DO jk=2,jpkm1
 DO jj=1,jpj
  DO ji=1,jpi
   zdk1t(ji,jj,jk)=(ptb(ji,jj,jk,jn) &
    & -ptb(ji,jj,jk+1,jn))*wmask(ji,jj,jk+1)
   zdkt(ji,jj,jk)=(ptb(ji,jj,jk-1,jn) &
    & -ptb(ji,jj,jk,jn))*wmask(ji,jj,jk)
  END DO
 END DO
END DO
... ...
```

(a)优化前

```
DO jk=2,jpkm1
 DO jj=1,jpj
  DO ji=1,jpi
   zdkt(ji,jj,jk)=(ptb(ji,jj,jk-1,jn) &
    & -ptb(ji,jj,jk,jn))*wmask(ji,jj,jk)
  END DO
 END DO
END DO
DO jk=2,jpkm1
 zdk1t(1:jpi,1:jpj,jk)=zdkt(1:jpi,1:jpj,jk+1)
END DO
... ...
```

(b)优化后

图 3　重复计算优化前和优化后

Fig.3　Repeat calculation before optimization after optimization

当从内存中访问数组元素时，两次访问的寻址间隔会影响 Cache 命中率，降低计算效率。由于 Fortran 中的数组是按照列优先的方式存储，因此调整数组元素访问顺序，使得相邻两次访问的寻址间隔变小，可以有效提高 Cache 命中率。调整数组元素访问顺序前后分别如图 4a 和 4b 所示。

另外，减少循环嵌套也能有效提高模式计算效率。在图 5a 的循环中，z2d 是临时数组，目的是存储 zftu 在三维空间沿 K 轴方向的积分。由于三重循环计算效率较低，因此利用 Fortran 提供的 forall 命令将该循环减少为一重循环，如图 5b 所示。经过上述串行与标量优化后，NEMO 模式整体计算效率提升 1.22 倍。

```
DO jj=1,jpjm1
 DO ji=1,fs_jpim1
  ... ...
  zmsku=1./MAX(tmask(ji+1,jj,jk) &
    & +tmask(ji,jj,jk+1)+tmask(ji+1,jj,jk+1)
    & +tmask(ji,jj,jk),1.)
  zmskv=1./MAX(tmask(ji,jj+1,jk) &
    & +tmask(ji,jj,jk+1)+tmask(ji,jj+1,jk+1)
    & +tmask(ji,jj,jk),1.)
  ... ...
 END DO
END DO
```

(a)优化前

```
DO jj=1,jpjm1
 DO ji=1,fs_jpim1
  ... ...
  zmsku=1./MAX(tmask(ji,jj,jk) &
    & +tmask(ji+1,jj,jk)+tmask(ji,jj,jk+1) &
    & +tmask(ji+1,jj,jk+1),1.)
  zmskv=1./MAX(tmask(ji,jj,jk+1) &
    & +tmask(ji,jj+1,jk+1)+tmask(ji,jj+1,jk)&
    & +tmask(ji,jj,jk),1.)
  ... ...
 END DO
END DO
```

(b)优化后

图 4 调整数组元素访问顺序优化前和优化后

Fig.4 Adjust access order of array element before optimization after optimization

```
DO jk=1,jpkm1
 DO jj=2,jpjm1
  DO ji=fs_2,fs_jpim1
   z2d(ji,jj)=z2d(ji,jj)+zftu(ji,jj,jk)
  END DO
 END DO
END DO
```

(a)优化前

```
forall(ji=fs_2:fs_jpim1,jj=2:jpjm1)
 z2d(ji,jj)=SUM(zftu(ji,jj,1:jpkm1),dim=1)
endforall
```

(b)优化后

图 5 循环嵌套优化前和优化后

Fig.5 Loop nesting before optimization after optimization

2.4 SIMD 优化

SIMD (Single Instruction Multiple Data) 即单指令多数据流,以同步方式,在同一时间内对多个数据执行同一条指令。矢量化是现代计算机的一个标志,利用自动矢量化,可以显著提高软件的计算性能。在本测试环境下,利用 xHOST 选项编译时,已包含了自动矢量化功能。然而自动矢量化需要遵循矢量化规则,如循环中不能存在数据前后依赖以及跳转语句等。以下是自动矢量化失败的例子及采用的优化策略。

在图 6a 所示的代码中,由于 zdit 和 zdjt 都是由指针方式实现的动态数组,Fortran 指针可以作为别名指向一块内存地址,因此编译器不能判断 zdit 和 zdjt 指针之间是否存在数据依赖,从而无法实现自动矢量化。通过将假定依赖的语句拆分成 2 个循环,使每个循环满足自动矢量化的条件,可以实现自动矢量化(图 6b)。

```
DO jk=1,jpkm1
 DO jj=1,jpjm1
  DO ji=1,fs_jpim1
   zdit(ji,jj,jk)=(ptb(ji+1,jj,jk,jn) &
    & -ptb(ji,jj,jk,jn))*umask(ji,jj,jk)
   zdjt(ji,jj,jk)=(ptb(ji,jj+1,jk,jn) &
    & -ptb(ji,jj,jk,jn))*vmask(ji,jj,jk)
  END DO
 END DO
END DO
```

(a)优化前

```
DO jk=1,jpkm1
 DO jj=1,jpjm1
  DO ji=1,fs_jpim1
   zdit(ji,jj,jk)=(ptb(ji+1,jj,jk,jn) &
    & -ptb(ji,jj,jk,jn))*umask(ji,jj,jk)
  END DO
  DO ji=1,fs_jpim1
   zdjt(ji,jj,jk)=(ptb(ji,jj+1,jk,jn) &
    & -ptb(ji,jj,jk,jn))*vmask(ji,jj,jk)
  END DO
 END DO
END DO
```

(b)优化后

图 6 SIMD 数据依赖优化前和优化后

Fig.6 SIMD data dependency before optimization after optimization

另外，由于代码中存在大量的指针作为临时数组，因此在不影响物理过程的前提下，通过将指针修改为动态数组，可以使代码安全地进行自动矢量化，如图 7 所示。经过 SIMD 优化后，NEMO 模式整体计算效率提升至 1.23 倍。

```
REAL(wp),POINTER,DIMENSION(:,:,:) :: a
CALL wrk_alloc(jpi,jpj,a)
Use a
CALL wrk_dealloc( jpi, jpj, a)
```

(a)优化前

```
REAL(wp),ALLOCATABLE,DIMENSION(:,:,:) :: a
ALLOCATE(a(jpi,jpj))
Use a
```

(b)优化后

图 7　SIMD 指针优化前优化后

Fig.7　SIMD pointer before optimization after optimization

2.5　内存带宽优化

当 CPU 需要读取内存中的数据时，CPU 将首先发出一个由内存控制器执行的请求，然后再将数据由内存返回中央处理器，此过程的时间称为延时周期。为了缩短延时周期，中央处理器和内存之间设置了 L1 和 L2 缓存。由于 L1 和 L2 均采用静态随机访问的原理，可以将其理解为内存带宽[8]。对内存带宽进行优化，提高缓存命中率，也是提升软件计算效率的有效方式。

由于海洋环流模式复杂的物理过程，模式中使用大量的临时数组保存计算过程的中间结果，这就造成了运算时参与计算的数组过多的问题。当 CPU 计算时，如果一条语句需要同时读取大量数组，由于内存带宽限制，则会减少缓存命中率，降低计算效率。因此减少或避免使用临时数组，是缓解内存带宽占用过高的一种优化手段。

如图 8a 所示，数组 A，B 为临时数组，运算时首先通过计算为 A，B 数组赋值，并在之后的循环中访问该数组。优化策略为将临时数组 A、B 去掉，而把计算赋值的功能重构为子程序 cal_a，cal_b，并在循环中需要此中间结果时调用。

```
Temporary Array A(i,j)
Temporary Array B(i,j)
Do loop
 compute A(i,j)
 compute B(i,j)
End loop
Do loop1
 Read A(i,j)
 Read B(i,j)
End loop1
.......
Do loop2
 Read A(i,j)
 Read B(i,j)
End loop2
....
```

(a)优化前

```
SUBROUTINE cal_a(temp_a)
 ... ...
end SUBROUTINE cal_a
SUBROUTINE cal_b(temp_a)
 ... ...
end SUBROUTINE cal_b
Do loop1
 call cal_a(temp_a)
 call cal_b(temp_b)
 Use temp_a
 Use temp_b
End loop1
.......
Do loop2
 call cal_a(temp_a)
 call cal_b(temp_b)
 Use temp_a
 Use temp_b
End loop2
....
```

(b)优化后

图 8　内存带宽优化优化前和优化后

Fig.8　Memory bandwidth before optimization after optimization

对热点函数进行分析后，发现 traldf_iso、nonosc 和 lim_rhg 中均存在大量临时数组，可以使用上述策略进行优化。图 9 显示了优化前后各个函数的计算时间以及其加速比，可以看出优化后单个热点函数的计算效率有效提升 10%～40%，效果显著。另外，由于 NEMO 的热点函数呈现分散的特点，此优化使得模式整体计算效率加速至 1.31 倍。

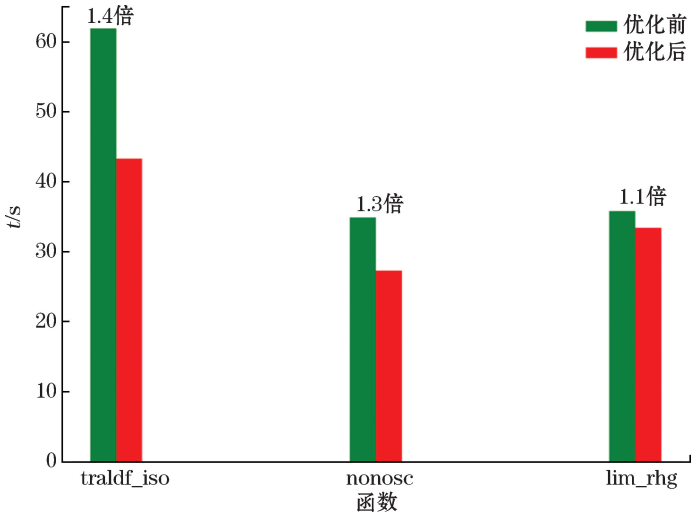

图 9 内存带宽优化前后函数时间对比

Fig.9 Comparison of function time before and after memory bandwidth optimization

2.6 MPI 多进程扩展

NEMO 模式总体是一个循环过程，其中时间是循环的主序列。循环从数据读取开始，以结果输出为结束。循环部分包含模式中的所有核心计算，总计算量占总循环过程的 95%及以上[9]。本文使用的 NEMO 3.6 版本基于 MPI 消息传递接口编写，使用 MPI 分布式内存并行计算可以有效的进行任务划分，从而提高计算效率。在经过以上串行优化之后，接下来利用 Intel Trace Analyzer Collector 工具对 NEMO 进行多进程扩展测试，以观察模式的负载均衡性。

图 10 显示了模式在 48 进程时的负载均衡性，从结果可以看出，计算（蓝色）和通信（红色）时间占比比较合理，且分配给每个进程的计算任务基本一致，其中进程间的计算时间占比最大相差不超过 8%，有 40 个进程相差在 1%以内，基本实现了负载均衡，不需要进行进一步优化。

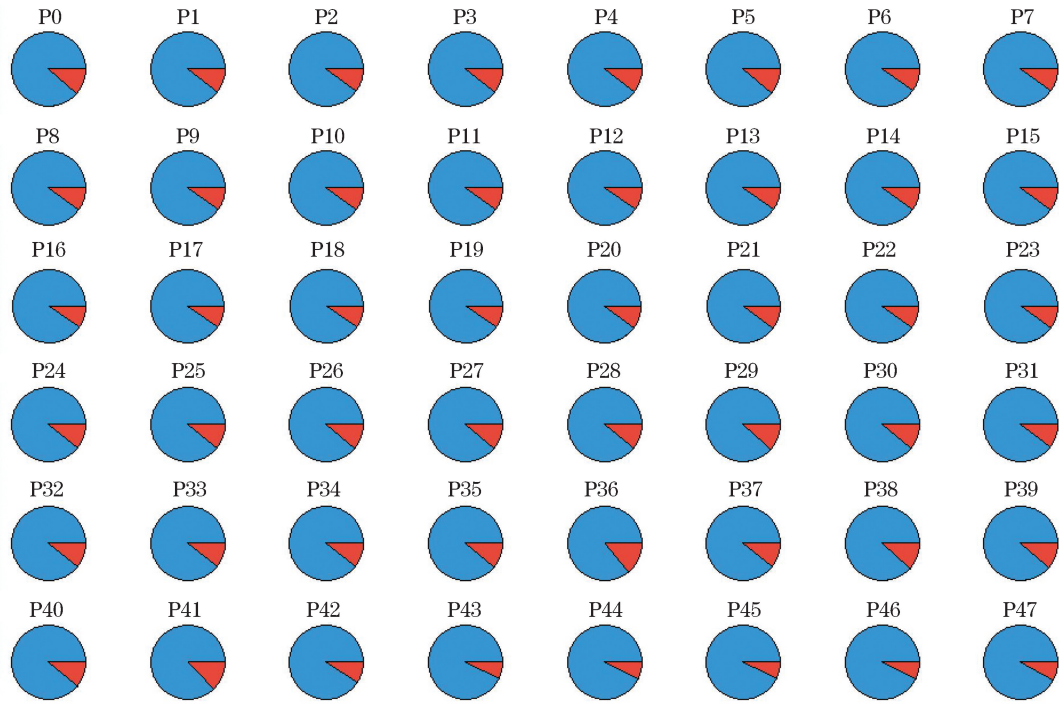

图 10 48 进程下负载均衡分析

Fig.10 Load balancing analysis of 48 processes

为了更全面地对模式的并行能力进行检验，我们进一步测试了模式的扩展性。图 11 为海洋环流模式在不同进程数下的加速比。在单节点 48 进程内，随着进程数的增加，加速效率逐渐降低，这是由于增加进程引起的代价不可消除，如进程间通信量增加，带宽限制等。另外，程序中必定有不能并行的串行部分，即使进程无限增多，通过并行所产生的加速比都是受限的，即阿姆达尔定律[10]。因此模式无法随着进程的增加获得线性加速比。此外，当进程数超过 48 时，由于测试环境没有开启超线程，进程可能会被挂起等待，同时进程间通信增加，加速效率会继续降低。

在经过编译选项优化、标量串行优化、SIMD 优化、内存带宽优化几个步骤后，模式整体性能得到了有效提升，如图 12 所示。选取合适的编译选项可以充分发挥计算机的性能，使得计算效率提升了 21%；通过规范代码书写，去除重复计算，减少循环嵌套等方法，实现串行代码优化，以及通过修改代码，使之符合编译器对矢量化的要求，实现编译器自动矢量化，这两步优化共实现了 2%的性能提升；另外，针对模式占用内存带宽过高的问题，提出了去掉临时数组的解决方案，使得优化后的单个热点函数加速比最高可以达到 1.4 倍，模式整体加速了 8%。经过上述优化，最终模式的整体计算效率共提升了 31%，节省了接近 1/3 的计算资源。同时，48 进程下的 MPI 并行模式计算效率相较于串行模式提升了 15.8 倍，虽然单节点内扩展性较弱，但负载均衡性较好。

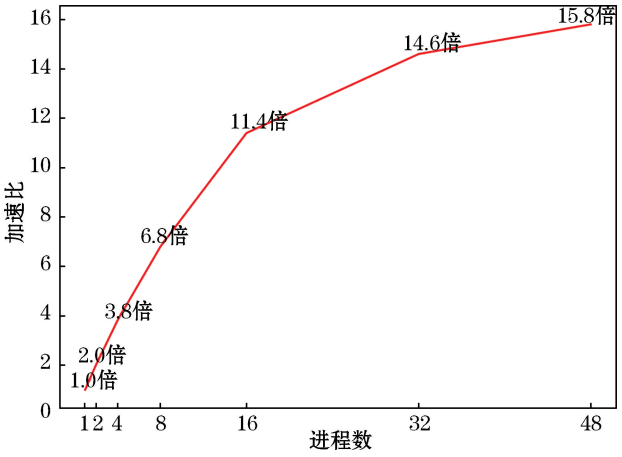

图 11　不同进程下的加速比

Fig.11　Acceleration ratio of different processes

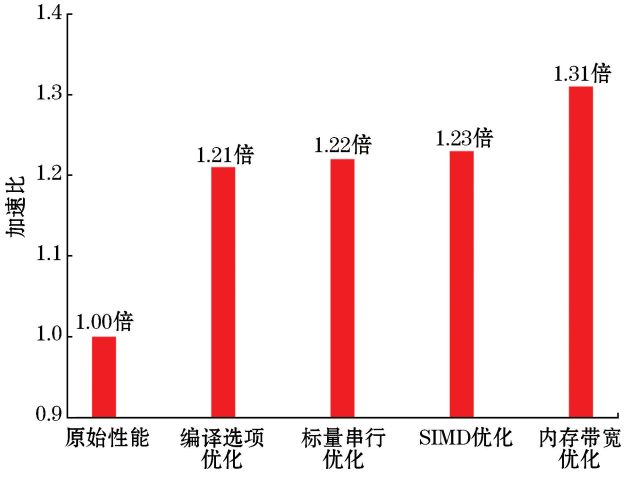

图 12　各优化步骤加速比

Fig.12　Acceleration ratio of each optimization step

3　试验结果分析

3.1　大量指针被假定数据依赖

矢量化有许多限制，例如存在数据依赖、函数调用(数学库调用除外)、在同一循环中混合可矢量化类型以及存在与数据相关的循环退出条件等，则不能进行矢量化。其中数据依赖分为写后读依赖和读后写依赖。写后读依赖即计算一个数组元素时需要依赖该数组已经计算的元素，如果强制将其矢量化，可能后面需要读取的依赖元素并没有被计算完成，因此造成读取数据错误。除了存在不可避免的计算中的前后依赖之外，NEMO 的矢量化报告显示模式源代码中存在大量的指针，而由于 Fortran 中的指针可以作为别名指向内存地址，因此编译器通常无法判断包含指针的代码是否存在数据依赖，从而无法自动矢量化。本文采取了两种方法来避免因指针造成的无法矢量化问题，包括拆分循环和将指针修改为动态数组。但需指出的是，这只是针对热点函数的局部优化，因此效果有限。若要全局优化此问题，则需要将代码重构。

3.2 循环嵌套过多

NEMO 的核心计算都包含在循环嵌套中，源代码中的函数多是四重循环，包括一维时间循环以及三维空间循环。当算例规模达到一定程度时，会不可避免地出现效率低下。另外，由于编译器仅会对最内层循环进行自动矢量化，因此循环嵌套过多也会造成 SIMD 优化效率低下。针对此问题，一种优化思路是遵循内大外小的原则，即将迭代次数多的循环放在最内层，提高 SIMD 优化效率。但更有效的解决方案则是减少循环嵌套层数，但是需要从根本上改变算法问题，这将是非常复杂的工程。

3.3 内存带宽过高

本研究中的串行与标量优化以及 SIMD 优化效果有限，而在经过内存带宽分析及优化后，热点函数的性能提升十分明显，这说明 NEMO 海洋环流模式内存带宽占用过高是个比较显著的问题。这是由于 NEMO 计算时涉及大量数组运算，因此会出现大量缓存未命中的情况，需要频繁从内存中读取数据。而当内存带宽占用过高时，处理器从内存获取数据的速度会出现瓶颈。去除临时数组是一种解决方案，但却会增加大量的重复计算，增大计算压力，这与串行代码优化中的去除重复计算优化策略恰恰相反。但针对具体的 NEMO 内存带宽瓶颈问题，去除临时数组后减少的时间远远大于增加重复计算带来的计算时间，因此去除临时数组是非常有效的。

4 结 语

在本文中，首先介绍了 NEMO 模式特征，并根据代码现代化指导步骤制定了 NEMO 的优化策略。随后给出了试验算例以及测试环境，并详细介绍了 NEMO 代码现代化的优化过程。结果表明，通过对选取的热点函数进行编译选项优化、标量串行优化、SIMD 优化以及内存带宽优化，串行的海洋环流模式的整体加速比可以达到 1.31 倍，其中热点函数加速效果非常明显，例如 traldf_iso 相较于原始性能提升了 2.3 倍。但因模式热点函数分散，选取的热点函数并不是模式的全部计算瓶颈，因此整体加速比有限。最后对 NEMO 在 MPI 多进程下进行了扩展性测试，测试结果表明在 48 进程(测试环境最大核心数)的加速比为 15.8 倍，计算和通信时间占比合理，负载均衡。但可能是由于内存带宽占用过高，导致单节点内扩展性较差，这需要进行进一步分析验证。另外，我们还对优化后的模式结果进行了验证，以确保优化后结果的准确性。需要注意的是，虽然对于海洋模式 NEMO 来说，我们的优化对模拟结果影响不大，在误差允许范围内，但是在海气耦合模式中，由于存在强的非线性海气相互作用，误差可能会被放大，因此在优化时需要注意并检验。总的来说，代码现代化在不增加任何硬件成本的前提下是一种行之有效的优化方案。

此外，在对 NEMO 进行代码现代化优化的过程中，分析出其主要存在以下几点影响计算效率的问题，如模式大量使用指针、循环嵌套过多、内存带宽占用过高等。针对这些问题，本文给出了相应的解决方案，如将指针修改为动态数组，减少循环嵌套次数以及去除临时数组等，可以为今后海洋模式的设计和改进提供参考。

另外，MPI 并行化仅在二维方向上进行分解(沿着 jpi 和 jpj)[11]。当 MPI 进程数增多时，每个进程间的通信量会成倍增加，通信开销会越来越大，导致模式性能下降。为了减少通信时间，通过引入 MPI/OpenMP 混合并行将会是一个切实可行的方法。OpenMP 可以实现共享内存层面的任务并行，它是一种简单有效的并行方式[11-12]。这种混合并行方法通过减少 MPI 进程数，可以减小进程间通信开销，同时利用 OpenMP 打开超线程又能充分利用计算资源，使得并行结构更加合理[13]。由于部分 NEMO 算例的特征是在三维数组上沿着 jpi、jpj 和 jpk 执行的操作，因此可以将 OpenMP 共享式内存并行化应用在 jpk(此试验中为 75)方向上，提高进程内的计算能力。在热点函数中引入 MPI/OpenMP 并行化后，由于进程间通信数量减少，将有利于提高模式的扩展性。这是接下来研究的方向之一。

参考文献(References)：

[1] SONG Z Y, LIU W G, LIU X, et al. Research progress and perspective of the key technologies for ocean numerical model driven by the mass data[J]. Advances in Marine Science, 2019, 37(2): 161-170. 宋振亚，刘卫国，刘鑫，等. 海量数据驱动下的高分辨率海洋数值模式发展与展望[J]. 海洋科学进展，2019，37(2)：161-170.

[2] ZHAO W, LEI X Y, CHEN D X, et al. Porting and application of global eddy-resolving parallel ocean mode pop to SW supercomputer [J]. Computer Applications and Software, 2014, 31(5): 42-45. 赵伟，雷晓燕，陈德训，等. 全球涡分辨率并行海洋模式 POP 在神威蓝光上的移植和应用[J]. 计算机应用与软件，2014，31(5)：42-45.

[3] YANG X D, SONG Z Y, ZHOU S, et al. Code modernization optimization of MASNUM wave model[J]. Advances in Marine Science, 2017, 35(4): 473-482. 杨晓丹，宋振亚，周姗，等. MASNUM 海浪模式的代码现代化优化[J]. 海洋科学进展，2017，35(4)：473-482.

[4] SONG Z Y, LIU H X, LEI X Y, et al. The application of GPU in ocean general circulation mode POP[J]. Computer Applications and Software, 2010, 27(10): 27-29. 宋振亚，刘海行，雷晓燕，等. GPU 在海洋环流模式 POP 中的应用[J]. 计算机应用与软件，2010，27(10)：27-29.

[5] ZHAO W, SONG Z Y, QIAO F L, et al. High efficient parallel numerical surface wave model based on an irregular quasi-rectangular domain decomposition scheme[J]. Science China: Earth Sciences, 2014, 44(5): 1049-1058. 赵伟，宋振亚，乔方利，等. 基于非规则类矩形剖分的高效并行海浪数值模式[J]. 中国科学：地球科学，2014，44(5)：1049-1058.

[6] MADEC G. THE NEMO TEAM. NEMO ocean engine (Version v3.6)[EB/OL]. [2017-10-13]. http://doi.org/10.5281/zenodo.1472492.

[7] PEARCE M. What is code modernization?[EB/OL]. (2015-07) [2016-09-01]. https://software.intel.com/content/www/us/en/develop/articles/what-is-code-modernization.html.

[8] ZHANG Y Y, SUN L C. On the impact of memory on computer performance[J]. Value Engineering, 2018, 37(24): 200-202. 张源源，孙连春. 浅谈内存对计算机性能的影响[J]. 价值工程，2018，37(24)：200-202.

[9] WANG Y Q, ZHANG Y, LIN B, et al. Computing performance optimization of global marine environment prediction model in supercomputing cluster based on NEMO[J]. Marine Forecasts, 2018, 35(3): 41-47. 王延强，张宇，林波，等. 基于 NEMO 的全球海洋环境预报模式在超算集群的计算性能优化[J]. 海洋预报，2018，35(3)：41-47.

[10] PACHECO P. An Introduction to Parallel Programming[M]. USA: Morgan Kaufmann Publishers, 2011.

[11] EPICOCO I, MOCAVERO S, ALOISIO G. The NEMO oceanic model: computational performance analysis and optimization[P]. High Performance Computing and Communications (HPCC), 2011 IEEE 13th International Conference on, 2011.

[12] ZHANG X, JI Z Z, WANG B. Some study on application of OpenMP in mesoscale meteorological model-MM5[J]. Climatic and Environmental Research, 2001, 6(1): 84-90. 张昕，季仲贞，王斌. OpenMP 在 MM5 中尺度模式中的应用试验[J]. 气候与环境研究，2001(1)：84-90.

[13] SU M F, El-KADY I, BADER D A, et al. A novel FDTD application featuring OpenMP-MPI hybrid parallelization[C]// ICPP 2004. Montreal, Que: IEEE, 2004: 373-379.

Code Modernization Optimization of Ocean General Circulation Model NEMO

ZHOU Sheng-chang[1,2], LIU Wei-guo[1,3], SONG Zhen-ya[2,3,4], YANG Xiao-dan[2,3]

(1. *School of Software*, *Shandong University*, Jinan 250101, China;
2. *First Institute of Oceanography*, *Ministry of Natural Resources*, Qingdao 266061, China;
3. *Laboratory for Regional Oceanography and Numerical Modeling*, *Qingdao Pilot National Laboratory for Marine Science and Technology*, Qingdao 266237, China;
4. *Key Laboratory of Marine Science and Numerical Modeling*, *Ministry of Natural Resources*, Qingdao 266061, China)

Abstract: The ocean general circulation model (OGCM) is the key tool for ocean environment simulation and forecast. With the ocean resolution finer, the demand for improving computational performance is more and more urgent. To improve the calculation performance of OGCM by taking full advantage of modern computers, a code modernization optimization scheme is carried out in this paper using an OGCN named NEMO as an example. The Intel performance analysis tool is used to evaluate the computing performance of the model at first. Then, several optimization steps, which are compiler options, serial and scalar optimization, SIMD, memory bandwidth optimization and extending to multi-cores, are applied to hotspot functions. After optimization, the model's overall performance can be improved by 31% without increasing any hardware cost and load balance have a good performance in multi-process. The results indicate that the optimization strategy used in this study is very effective and useful. Furthermore, the problems that significantly affect the computational efficiency in the model, such as the heavy use of pointers that can prevent vectorization, multiple loop nesting, high memory bandwidth usage, are discussed in this paper, to provide reference and suggestion for the OGCM in the future design and improvement.

Key words: code modernization; Ocean General Circulation Model; NEMO; intel performance; analysis tools; SIMD

Received: December 2, 2019