DOI：10.7652/xjtuxb201812012

计算流体力学程序单核指令级优化方法

刘闯，何峰，肖兮，董小社，张兴军
（西安交通大学计算机科学与技术系，710049，西安）

摘要：针对目前大多数计算流体力学程序对系统的单核计算能力利用不足，提出一种针对计算流体力学程序的单核指令级优化方法。该方法首先分析程序的性能指标存在潜在的性能不足，根据分析结果进行优化；依据容器的存储特性和系统的访存特性，对程序的存储结构和访存顺序进行调整，以优化空间开销和访存性能；对 CPU 的流水机制进行分析，在循环和分支中消除指令的控制相关和数据相关从而达到减少流水中断率的目的；分析编译器对高级语言的处理特点并结合系统中的运行时栈在指令级作出分析，优化指令结构从而减少指令冗余和降低指令复杂度。实验结果表明，在 TIANHE-1A 超级计算机系统上进行测试，与优化前程序相比，优化后的程序执行时间约减少 68.34%，空间消耗约减少 55.43%。通过对程序性能各项指标进行分析的结果表明，程序在流水中断率、缓存命中率及机器指令数等性能指标上均有大幅地提升，该方法优化覆盖范围多于目前其他优化方法，有较好的优化效果，在计算流体力学程序优化研究中具有一定的借鉴价值。

关键词：流体力学程序；指令级优化；访存优化；流水优化
中图分类号：TP302.7 **文献标志码**：A **文章编号**：0253-987X(2018)12-0077-07

A Single-Core Instruction-Level Optimization Method for Computational Fluid Dynamics Programs

LIU Chuang, HE Feng, XIAO Xi, DONG Xiaoshe, ZHANG Xingjun
(Department of Computer Science and Technology, Xi'an Jiaotong University, Xi'an 710049, China)

Abstract: A single-core instruction-level optimization method for computational fluid dynamics (CFD) programs is proposed to overcome the shortage of most current CFD programs in utilizing the single-core computing power of a system. The method first analyzes the performance indicators of a program to find potential performance deficiencies, and then optimizes them according to the analysis results. Then the memory structure and memory access sequence are adjusted according to the memory access characteristics of the system and the container to optimize memory access performance and space overhead. The pipeline mechanism of CPU is analyzed, and the control correlation and data correlation of instructions in loops and branches are eliminated to reduce the pipeline interruption rate. Both the characteristics of a compiler processing the high-level language and the runtime stack at the instruction level are analyzed to optimize the instruction structure and to reduce instruction redundancy and duplication. Experimental results show that the performance of the optimized program is greatly improved. Testings on TIANHE-1A supercomputer system show that the execution time of the program reduces by 68.34% and the space consumption reduces by 55.43%. Analyses show that the

收稿日期：2018-06-13。 作者简介：刘闯(1993－)，男，硕士生；董小社(通信作者)，男，教授，博士生导师。 基金项目：国家重点研发计划资助项目(2016YFB0200902)。
网络出版时间：2018-10-25 网络出版地址：http://kns.cnki.net/kcms/detail/61.1069.T.20181023.1652.012.html

performance of the program is greatly improved in pipeline interruption rate, cache hit rate and number of machine instructions. It shows that the proposed method has more coverage than other existing optimization methods and better optimization effect, and has a good reference value.

Keywords: computational fluid dynamics program; instruction-level optimization; memory-access optimization; pipeline optimization

通过计算流体力学程序模拟真实情景是提高流体机械的制造精度和节约成本的重要手段,计算流体力学程序具有计算密集、处理数据量大的特性,因此常需要大量的计算资源。尽管计算机的计算能力有了飞速的发展,计算流体力学程序对计算性能的利用率仍然存在不充分的现象,导致计算资源的浪费。

目前对于流体计算程序性能提升的研究主要体现在并行化或异构加速上[1-3],例如采用多核系统并行处理或采用协处理器加速计算。超级计算机上部署的大型计算流体力学程序大多都采用了上面两种计算方式,但是作为一个重要的影响因素,程序的单核执行效率并没有得到足够的重视,很多程序的单核性能只能达到系统理论峰值的 5%~10%[4],导致大量的计算资源被浪费,因此对程序单核性能的优化应引起重视。

针对流体计算程序的优化,一些文献提出了综述性的优化方案和思路[5-7],或针对具体程序进行了优化[8],但是这种优化主要集中于对云计算、网格计算等并行化计算的优化,对单核级优化的工作不够深入;针对单核级的程序优化的研究中,主要优化方案是流水优化或访存优化,优化步骤一般是首先分析程序特性或测试程序热点以找到瓶颈,然后通过调整代码结构流水性能[9]或调整数据的组织方式优化访存性能[4,10-12];从处理器的角度,贺爱香、Oyarzun 等根据 ARM 处理器的计算特性对问题提出优化方案[13-14],这些研究具有一定的通用价值,但是对于大型计算程序常用的 Intel 处理器并没有深入研究;申小伟等通过对编译器等运行环境的优化来提升程序性能[15],但是这种方法对于跨专业的程序优化人员具有一定的门槛,且在公用的计算环境中难以部署。根据上述背景,本文针对轴流压气机模拟的流体计算程序,提出一种针对计算流体力学程序单核的优化方法。总体优化思路有 3 个方面:①优化指令结构,减少由相关引起的指令流水断流,使程序充分发挥 CPU 的计算能力;②调整存储结构和访存结构并进行容器优化,从而增强存储和访存的性能;③减少冗余指令,优化底层代码数量和结构从而减少程序执行所需的时间空间。本文在轴流压气机模拟程序上实现优化方法,在 TIANHE-1A 超级计算机和商用服务器上进行测试实验,实验结果表明,本文提出的优化方法可以有效地提高程序的缓存命中率、优化流水连续性、减少目标代码数,从而大幅提升程序单核执行效率。

1 流体计算程序简介

1.1 流体计算程序背景

本文采用轴流压气机模拟程序[16-17]对轴流压气机转子进行模拟,程序采用有限体积法进行空间离散,3 步 Runge-Kutta 法进行时间推进,控制方程组采用笛卡尔坐标系下的相对流动控制方程组,形式如下

$$\frac{\partial \boldsymbol{U}}{\partial t}+\frac{\partial \boldsymbol{F}}{\partial x}+\frac{\partial \boldsymbol{G}}{\partial y}+\frac{\partial \boldsymbol{H}}{\partial z}=\boldsymbol{I}+\frac{\partial \boldsymbol{F}_v}{\partial x}+\frac{\partial \boldsymbol{G}_v}{\partial y}+\frac{\partial \boldsymbol{H}_v}{\partial z} \quad (1)$$

湍流模型采用 SA 湍流模型,为加速收敛程序采用三层网格,通过在不同网格层上轮流迭代从而使不同频率的误差分量得到有效地衰减。

程序模拟 36 叶片压缩机工作时的流体状态,采用 36 个进程,每个进程对一个叶片进行模拟,单叶片网格数约为 42 万,总计约 1 535 万。在边界处理时需进行通信,通信采用 MPI 完成。程序主要开销是在单个进程的计算上,通信等待时间相对整个程序时间占比较少,且优化不涉及通信时间的优化,因此并行开销可忽略不计。

1.2 流体计算程序实现细节及性能分析

考虑程序的缓存命中率对程序的存储和计算特性进行分析。存储方面,程序中网格的存储采用 STL 库中的 Vector 容器进行存储,采用该容器的主要原因是 Vector 容器是顺序存储容器,有较好的随机存取性能,以及可以利用 STL 中提供的丰富的操作,功能强大。根据物理网格模型,容器的维度为四维,存储维度顺序为 x、y、z、n,其中 x、y、z 为空间维度,n 为多层网格的层号。

计算方面,程序计算主体的运行时间约占整个运行时间的 95%以上,剩余时间做程序流程控制、数据的通信与保存等操作,因此程序优化的主体在计算部分。在计算部分中,程序对整个网格层采用 for 循环进行更新,大多数循环深度为 3 层,由于边

界上存在虚拟网格，循环次数约为 43.5 万次，略多于网格数，每次迭代中更新一个或多个变量。此外，为实现复杂计算，程序还调用了 math 库中的部分计算函数，考虑到计算函数计算开销较大，在某些调用中可采用等价短指令替换。

为进一步分析程序计算性能，本文采用性能分析软件对程序进行分析。结果显示，程序的流水线性能未充分发挥，双向中断率均超过 50%，暂停周期数较高，导致执行某个程序的指令平均时钟周期(CPI)与理论值有差距。因此，在优化时应考虑流水优化；缓存命中率约为 50%，访存上也有优化空间；此外程序执行指令数为 $38\ 534\times10^9$ 条，在优化中需寻找潜在的指令优化来减少冗余代码。

2　优化策略研究

本文所述方法应用在 TIANHE-1A 超级计算机系统中 Intel Xeon X5670 CPU 上，采用 Intel CPU 作为示例是考虑到大多数的高性能计算环境都采用 Intel 处理器，且 Intel CPU 中采用的加速部件在其他处理器中有类似部件设计，因此以 Intel CPU 为基础设计的优化方案具有较好的普适性。

图 1 给出了 Intel CPU 的架构示意图，该示意图基于 Intel Xeon X5670 CPU 所采用的 Nehalem 架构，主要体现 CPU 针对单核程序的加速方式，对其余部件做了省略和简化。图 1 中 CPU 包含两个核，每个核内设置独立的计算部件，从逻辑上分为取值部件、解码部件、顺序回退部件、计算单元，图 1 中的箭头代表了部件间的数据通路。基于上述架构，Nehalem CPU 架构部署了取指(IF)、译码(ID)、执行(EXE)、写回(WB)四阶段流水机制。此外，Nehalem 架构还采用了超标量技术，在 CPU 中建立了 4 条流水线，因此理论上可以同时执行 4 条流水指令。每个核设置独立的 L1 和 L2 缓存，多个核间共享 L3 缓存，方便进行数据缓存，提升访存性能。其中一级缓存分为数据缓存和指令缓存，每个缓存容量为 32 KB，采用 8×64 组相连的映射方式，访存速度为 4 个时钟周期；L2 缓存容量为 256 KB，采用 8×512 组相连映射方式，访存速度为 10 个时钟周期；L3 缓存容量为 8 MB，采用 16×8 192 组相连映射方式，访存速度为 40～75 个时钟周期。

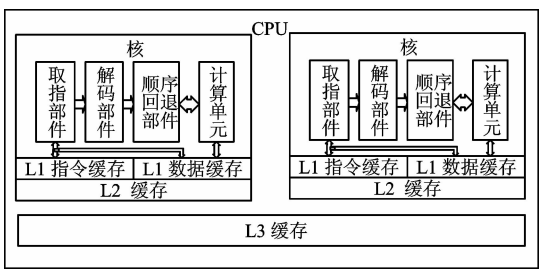


图 1　Intel CPU 架构示意图

基于对 Nehalem 架构的分析，程序优化的思路主要集中在使程序充分发挥流水性能和访存性能上。

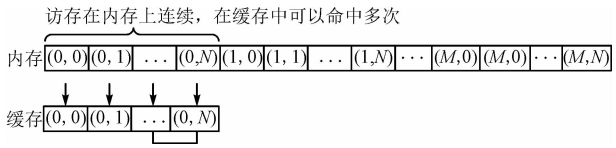
除针对 CPU 架构的优化外，还需考虑程序本身的简洁性。对于不同形式的程序，通过编译器编译得到的目标代码有很大差距，若有冗余和过多同义代码会导致性能下降。为保证程序性能，在编码时需考虑高级语言被编译后得到的目标代码是否简洁高效。

结合上述分析，本文提出的流体计算程序单核优化方案主要考虑 3 个方面：①优化存储结构和访存结构，发挥高速缓存效率；②调整程序的指令结构，发挥流水性能；③优化程序指令，使程序底层指令数得到削减和优化。前两个方面面向 CPU 对单核指令执行的高速访存和流水优化部件展开，目的是充分发挥 CPU 加速部件的加速效果。第 3 个方面针对程序代码本身优化，考虑到不同指令在底层执行时采用不同的汇编指令翻译，程序的目标代码应足够精简高效。

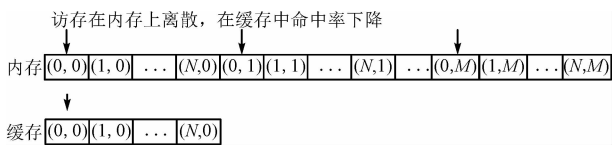
2.1　访存优化

访存优化的目的是使程序在数据访问时缓存的利用率更高，主要优化手段是调整数据的存储结构和访存结构，首先考虑存储结构。

程序的访存方式是通过循环遍历的方式对网格进行访问，优先访问网格层号 n，其次为 x、y、z 轴，这种存储结构和访存方式的不匹配造成了缓存效率较低。图 2 给出了不同存储方式下访存情况的对比，对于 $M\times N$ 矩阵，规定访存顺序为从(0，1)到(0，N)，若矩阵采取 $a_{M,N}$ 的形式存储，则矩阵在内存中的物理分布如图 2a 所示，在访存时数据分布在连



(a)访存在内存上连续



(b)访存在内存上离散

图 2　不同存储方式下访存情况的对比

续的内存中，缓存会将连续内存存入，在下次访存时可直接取用缓存，减少内存访问。若矩阵采取 $a_{N,M}$ 的形式存储，则矩阵在内存中分布如图 2b 所示，在访存时会产生跳跃，因此缓存无法命中，需要进行内存访问，影响程序运行效率。

为解决访存问题，将数据的存储结构调整为 $a_{n,z,y,x}$，这样调整的好处有：①存储结构更改后 x 网格内相邻各点在内存中存储的更近，更有利于程序的连续访问；②对于多维容器来说，由于 $x>y>z>n$，更改后的存储形式减少了容器在较低维度所创建的数量，减少了目标代码的指令数。例如，对于容器 $a_{20,10}$，需先创建一个容量为 20 的容器，每个容器中再创建一个容量为 10 的容器，总共创建 21 个，若改为 $a_{10,20}$，则仅需创建 11 个容器。

下面根据存储结构调整访存结构，访存顺序的控制由 for 循环语句进行。为使访存顺序更适合存储结构，将循环结构和存储维度调整为一一对应的关系。即对于 $a_{n,z,y,x}$ 形式的数据，处理循环为最外层循环处理 n、次外层处理 z、内层处理 y、最内层处理 x。对不满足这种处理方式的操作，考察操作的相关性，若无相关性可通过拆分循环来优化访存顺序。

访存方式的优化和存储结构的优化有类似的优点：调整访存结构有利于高速缓存；调整后循环的控制变量初始化次数减少，消减底层指令数。

2.2 流水优化

考虑影响流水线性能的问题，在程序一级影响因素主要有：由分支转移引起的控制相关会导致流水失效；由写后读引起的数据相关，这些相关会导致流水暂停，使流水性能下降。Nehalem CPU 架构部署了四阶段流水机制，虽然该架构加入了分支预测和数据旁路的方式减少流水线中断，但是仍不足以完全消除相关性引起的性能损失，因此在优化时仍可消除相关性。

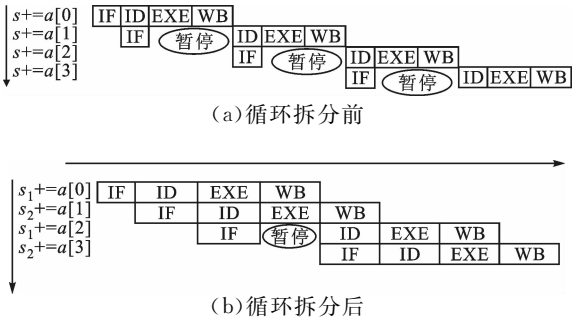
考虑计算中的循环部分，由于循环在每次迭代时需进行一次判断，若循环内指令较少会导致计算和分支的比例过低，产生控制相关导致性能下降，同时循环中对同一变量的反复读写可能引起数据相关。针对控制相关和数据相关问题，一般采用循环展开并引入多中间变量的方法进行优化，循环展开可以在以下方面得到优化：①减少分支数；②减少循环控制的指令开销；③减少循环内计算的数据相关。循环展开示例如下

```
for(i=0;i<k;i++)
  s+=a[i];
```

改为：

```
for(i=0;i<k;i+=2)  {
  s1+=a[i];
  s2+=a[i+1];
  }
s=s1+s2;
```

展开因子 $k=2$，即每次循环计算次数为两次。两种代码的流水情况如图 3 所示，流水性能得到了提升。



(a)循环拆分前

(b)循环拆分后

图 3 循环拆分前后流水示意图

理论上，在展开足够多循环的情况下可以完全消除流水中断，展开因子应大于硬件标量数与执行周期的乘积，如对于具有 3 个浮点乘器件，每个浮点乘需 5 个时钟周期的指令集，展开因子应大于 15；但是实际操作中需考虑寄存器溢出，即中间变量过多导致寄存器不够，触发访存。为此还需保证最优循环因子 k 满足

$$\max\{k\};\ F(k)\leqslant r,\quad k\in\mathbf{Z} \tag{3}$$

式中 $F(k)$ 为 k 次展开需要的寄存器数。

此外，解除数据相关还可以采用长计算拆分为短计算、创建临时变量的方式解除相关、多个连续的写后读相关语句交叉等方法。

对 if 语句引起的控制相关，分支预测部件往往不能正确预测走向，预测错误的惩罚往往达到十几个到几十个时钟周期，带来流水性能损失。在 X86 指令集中，条件数据传输指令可以被编译为普通指令流水线的一部分，在判断失败时没有惩罚，因此采用条件数据传送代替条件控制转移能提升程序性能。常用的条件数据传输指令是三目运算符，条件传输形式如 $s=a<b?\ a:b$，编译器会将这种类型的语句编译为不具有条件分支的形式，以下给出一种编译后的伪代码示例

```
x=a
x'=b
bool T=(a<b)
if (! T) x=x'
```

可以看出，这种编译方式将分支转移操作放在伪代码最后一行的 if 语句中，在目标代码中这个 if 语句通过条件拷贝指令 cmov 实现，分支的结果只在这一条指令中体现，与上下文无关，从而消除了分支开销。通过这种转换可以提升分支在流水线中的稳定性，减少预测错误的惩罚。

2.3　指令优化

指令优化的目的是从指令一级减少冗余代码。冗余代码主要指程序编译后的目标代码中产生的冗余，一般主要产生于如下两种情况。

(1)完成相同功能的不同语句在经过编译器编译后会产生不同的目标代码，若选择不合适的语句会引起冗余的目标代码。这种冗余包括两种，一种是目标代码量的增加，指对于同样功能的代码由于实现方法不同导致编译后代码条数不同；另一种是代码复杂度的增加，CISC 指令集中的指令执行时间不同，因此同样的指令条数下指令执行时间有所不同。

(2)为实现方法调用，操作系统需保存当前运行时信息，并在运行时栈开辟新的空间，在调用结束时恢复信息并释放空间。这种调度工作会体现在目标代码中，虽然一般认为正常的调用过程对程序性能影响不大，但当调用在程序中占比过大时会影响程序性能。调用引起代码冗余的一种示例如下。

非调用版指令如下

```
s=a+b;
```

目标代码为

```
1   mov %rdi,%rbp
2   mov %rax,rbp
3   add %rbx,%rax
```

调用版指令如下

```
Sum (int a,int b)
{return a+b;}
```

目标代码为

```
1   push %rbp
2   push %rbx
3   sub %8,%rsp
4   mov %rdi,%rbp
5   mov %rax,rbp
6   add %rbx,%rax
7   add %8,%rsp
8   pop %rbx
9   pop %rbp
10  ret
```

调用版的第 1、2 行用来实现现场保存，第 3 行为新调用栈分配空间，然后执行加法功能，第 7 行释放调用栈，第 8、9 行恢复现场，最后返回。可以看出，当实现同样加法功能的 $s=a+b$ 被替换为方法调用 Sum(a,b)后，运行时栈调度的开销超过了方法本身。虽然编译器中有对高级语言的自动优化功能或调用产生冗余代码的问题，如自动将短调用进行内联处理，但是编译器优化的前提是保证程序的安全，因此面对具有内存别名使用或变量类型未知等问题的代码不能进行自动优化，需手动对高级语言进行优化。

分析本程序得出，导致性能下降的主要问题是：①部分库函数调用时产生的隐藏代码增加了代码量，增加包括调用栈引起的代码开销和库函数本身复杂性引起的代码开销；②代码及代码结构本身的计算复杂度较高，因为有较多耗时的除法计算需求。为解决上述问题，考虑使用简单指令代替复杂指令，并结合访存优化及流水优化考虑汇编指令在系统中的运行状态进行优化。

下面讨论指令优化的具体实现。针对库函数调用的问题，结合程序分析发现，程序主要调用的库函数为数学库和容器类库。数学库的乘方、开方等操作一般不可替代，但是对于低阶乘方，可以直接使用数乘代替。如 pow(a,2)可以改写为 $a^*=a$，这种替换可以减少调用开销，同时简化方法本身的复杂度。

对容器类库函数的修改可考虑对 C++语言自带容器数组进行二次封装代替库容器，在本文的程序中为采用数组代替 Vector 容器。从可行性分析，Vector 和数组有较为类似的存储结构，因此在存取功能上具有可替代性。Vector 相对数组的优势在于灵活的操作方式和内存分配方式，程序中对容器的主要操作方式为随机访问，此外有少量的动态内存分配操作，因此在二次封装数组时需要为数组增加随机访存函数，从而实现对 Vector 的替代，所以用数组代替 Vector 具有可行性。从必要性分析，Vector 是 STL 封装的容器库，在使用时涉及库调用会增加底层代码量，容器的存储结构上，Vector 为实现动态增长，在分配空间时会分配额外的内存，在多维空间上，这种存储空间的浪费变得更加明显，在访存时也会增加额外开销，高速缓存的命中率也会下降。

针对代码结构结合访存特性考虑编译器对语句的处理方式，由于对变量的访问常被编译为寄存器访问操作，对引用的访问被编译为内存访问操作，因此为优化访存性能，需将循环中频繁引用访问修改

为变量访问。针对代码本身,考虑科学计算中常出现的乘除法指令,在 Intel 提供的参考机中部分算数指令性能见表 1,其中延迟表明实际运算周期数,发射表明两次运算最小间隔周期数,容量表明该功能的运算单元数。

表 1 Intel 参考机算数指令性能

运算类型	周期数				容量	
	整型延迟	整型发射	浮点延迟	浮点发射	整型	浮点型
加法	1	1	4	3	1	1
乘法	3	1	1	5	1	2
除法	3～30	3～30	1	3～15	3～15	1

由表 1 可以看出,乘除法运算执行时间较长,除法尤为明显,因此对乘除法的优化也应加以考虑。针对非浮点数的乘除法指令,采用移位运算代替乘除法的方法进行处理。该方法在乘除法数量较多且乘数或除数较为规律时可产生较好的优化效果。对于浮点数除法,可采用公共除法移出的方法将除法转换为乘法,减少执行时间。

3 实验结果

对第一节中的流体计算程序采用上述优化方案进行优化,并在商用服务器对各版本程序采用 perf 性能检测工具对程序进行小规模性能分析。商用服务器采用 Intel Xeon CPU E7-8850 @ 2.00 GHz(8×10 核)CPU,内存 125 GB,编译器采用 gcc 4.8.5。在三层网格上迭代轮次为 5、5、10。程序优化前后的主要指标的分析结果和执行时间见表 2。由于对容器以及对指令的优化,程序所使用的运行时空间的大小也有所下降,内存占用实验结果见表 3。

表 2 程序主要指标优化前后的比较

程序版本	指令条数/10^9	每周期指令数	指令暂停周期	前向流水中断率/%	后向流水中断率/%	缓存命中率/%	执行时间/s
优化前	38 534	0.66	1.16	76.25	51.36	52.46	710.5
优化后	13 976	0.82	0.86	70.76	44.97	64.79	209.2
相对浮动比例/%	−63.73	24.24	−25.86	−8	−12	27	−70.56

表 3 优化前后占用内存情况

程序版本	占用内存/MB
优化前	1 537
优化后	815
相对下降比例/%	49.97

由表 2、3 可以看出,程序性能的主要提升在于优化后导致指令数的减少,在系统级优化后缓存性能和流水性能也有所提升。在规模为 5、5、10 的程序中,经过优化的程序每周期指令数提升 24.24%,指令暂停周期下降 25.86%,流水中断率下降 6%,缓存命中率提升 14%,执行时间下降 70.56%,空间消耗下降 49.97%。

将程序部署在 TIANHE-1A 超级计算机系统中测试性能,采用 4 节点运行 36 进程。采用的 CPU 为 Intel Xeon X5670(2×6 核),内存 24 GB,编译器为 icc_xe_2013.0.07。三层网格迭代轮次采用 50、50、10 000 作为程序完整规模运行的实验,测试结果见表 4。

由表 4 结果可以看出,在 TIANHE-1A 系统上,三层网格迭代轮次分别为 50、50、10 000 计算规模时,程序耗时下降 68.34%,空间消耗减少 55.43%,与商用服务器上测试结果类似。根据测试结果可以看出,本文的优化方法能有效提升程序对 CPU 计算能力的发挥,大幅减少程序的底层代码,在测试程序中取得较好的优化效果。

表 4 程序完整规模运行的实验测试结果

程序版本	计算时间/s	占用内存/GB
优化前	309 555.06	1.60
优化后	97 976.52	0.71
相对下降比例/%	68.34	55.43

4 结 论

根据计算机系统单核级指令执行的加速特性和编译特性,提出一种针对计算流体力学程序的单核优化方案,主要优化方面包括:①优化指令结构,减少指令流水的断流,使程序充分发挥 CPU 的计算能力;②调整存储结构和访存结构,增强存储和访存的性能;③减少冗余指令,优化指令结构,从而减少程序执行所需的时间开销和空间消耗。

将本文方法应用在压气机转子模拟程序中,测试结果显示优化方案能有效减少程序指令数、增加缓存命中率,减少指令流水断流,在 TIANHE-1A 计算机系统中进行测试,结果显示优化后时间性能提升约 68.34%,空间性能提升约 55.43%,加速比

提升 3 倍以上，具有较好的优化效果。

参考文献：

[1] ASTORGA D D R, DOLZ M F, SANCHEZ L M. Discovering pipeline parallel patterns in sequential legacy C++ codes [C]// Proceedings of the International Workshop on Programming Models and Applications for Multicores and Manycores. New York, USA: ACM, 2016: 11-19.

[2] TAYLOR B, MARCO V S, WANG Zheng, et al. Adaptive optimization for OpenCL programs on embedded heterogeneous systems [C]// Proceedings of the 18th ACM SIGPLAN/SIGBED Conference on Languages, Compilers, and Tools for Embedded Systems. New York, USA: ACM, 2017: 11-20.

[3] BIFERALE L, MANTOVANi F, PIVANTI M, et al. Optimization of multi-phase compressible lattice Boltzmann codes on massively parallel multi-core systems [J]. Procedia Computer Science, 2011, 4(4): 994-1003.

[4] 车永刚，张理论，王勇献，等. 一个结构网格并行 CFD 程序的单机性能优化 [J]. 计算机科学，2013，40(3)：116-120.
CHE Yonggang, ZHANG Lilun, WANG Yongxian, et al. Single-machine performance optimization of a structured grid parallel CFD program [J]. Computer Science, 2013, 40(3): 116-120.

[5] NGUYEN A T, REITER S, RIGO P. A review on simulation-based optimization methods applied to building performance analysis [J]. Applied Energy, 2014, 113(6): 1043-1058.

[6] ALKHANAK E N, LEE S P, REZAEI R, et al. Cost optimization approaches for scientific workflow scheduling in cloud and grid computing: a review, classifications, and open issues [J]. Journal of Systems & Software, 2016, 113(5): 1-26.

[7] XU Jie, HUANG E, CHEN C H, et al. Simulation optimization: a review and exploration in the new era of cloud computing and big data [J]. Asia-Pacific Journal of Operational Research, 2015, 32(3): 11-34.

[8] CHE Y. Optimization of a parallel CFD code and its performance evaluation on Tianhe-1A [J]. Computing & Informatics, 2015, 33(6): 1377-1399.

[9] 罗红兵，张晓霞，王伟，等. 科学计算应用程序单核指令级优化研究 [J]. 计算机研究与发展，2014，51(6)：1263-1269.
LUO Hongbing, ZHANG Xiaoxia, WANG Wei, et al. Instruction level parallel optimizing for scientific computing application [J]. Journal of Computer Research & Development, 2014, 51(6): 1263-1269.

[10] 张宝印，莫则尧，曹小林. 基于计算缓存方法的分子动力学程序性能优化 [J]. 计算机工程与科学，2009，31(11)：77-79.
ZHANG Baoyin, MO Zeyao, CAO Xiaolin. Molecular dynamics program performance optimization based on computational buffer method [J]. Computer Engineering and Science, 2009, 31(11): 77-79.

[11] SIDDIQUE N A, GRUBEL P A, BADAWY A H A, et al. A performance study of the time-varying cache behavior: a study on APEX, Mantevo, NAS, and PARSEC [J]. Journal of Supercomputing, 2018, 74(2): 665-695.

[12] DING W, KANDEMIR M. CApRI: Cache-conscious data reordering for irregular codes [J]. ACM Sigmetrics Performance Evaluation Review, 2014, 42(1): 477-489.

[13] 贺爱香，顾乃杰，苏俊杰. 基于多核 ARM 体系结构的基础函数优化方法 [J]. 计算机工程，2018，44(5)：47-52.
HE Aixiang, GU Naijie, SU Junjie. Fundamental function optimization method based on multi-core ARM architecture [J]. Computer Engineering, 2018, 44(5): 47-52.

[14] OYARZUN G, BORRELL R, GOROBETS A, et al. Efficient CFD code implementation for the ARM-based Mont-Blanc architecture [J]. Future Generation Computer Systems, 2017, 79(3): 786-796.

[15] 申小伟，叶笑春，王达，等. 一种面向科学计算的数据流优化方法 [J]. 计算机学报，2017，40(9)：2181-2196.
SHEN Xiaowei, YE Xiaochun, WANG Da, et al. A data flow optimization method for scientific computing [J]. Journal of Computer Science, 2017, 40(9): 2181-2196.

[16] ZHANG Chuhua, MIAO Yongmiao, GU Chuangang. Numerical simulations of three dimensional turbulent flows in a shrouded backswept impeller at design and off-design flow rates using unstructured grid method [C]// Proceedings of the ASME Turbo Expo: Power for Land, Sea, and Air. New York: ASME, 2000: V001T03A033.

[17] ZHAO Lei, ZHANG Chuhua. A parallel unstructured finite volume method for all speed flows [J]. Numerical Heat Transfer: Part B Fundamentals, 2014, 65(4): 336-358.

（编辑　武红江）