计算机研究与发展 DOI:10.7544/issn1000-1239.2018.20160872
Journal of Computer Research and Development 55(2):400-408,2018

基于 OpenMP 4.0 的发动机燃烧模拟软件异构并行优化

杨梅芳[1] 车永刚[1,2] 高 翔[1]
[1](国防科技大学计算机学院 长沙 410073)
[2](国防科技大学并行与分布处理重点实验室 长沙 410073)
(1520241161@qq.com)

Heterogeneous Parallel Optimization of an Engine Combustion Simulation Application with the OpenMP 4.0 Standard

Yang Meifang[1], Che Yonggang[1,2], and Gao Xiang[1]
[1](*College of Computer, National University of Defense Technology, Changsha* 410073)
[2](*Science and Technology on Parallel and Distributed Processing Laboratory, National University of Defense Technology, Changsha* 410073)

Abstract LESAP is a combustion simulation application capable of simulating the chemical reactions and supersonic flows in the scramjet engines. It can be used to solve practical engineering problems and involve a large amount of computations. In this paper, we port and optimize LESAP with the OpenMP 4.0 accelerator model, targeting the heterogeneous many-core platform composed of general CPU and Intel Many Integrated Core (MIC). Based on the application characteristics, a series of techniques are proposed, including OpenMP 4.0 based task offloading, data movement optimization, grid-partition based load-balancing and SIMD optimization. The performance evaluation is done for a real combustion simulation configuration, with 5 320 896 grid cells, on one Tianhe-2 supercomputer node. The results show that the resulting heterogenous code significantly outperforms the original CPU only code. When the heterogenous code runs on two Intel Xeon E5-2692 CPUs and three Intel Xeon Phi 31S1P coprocessors, the runtime per time-steep is reduced from 64.72 seconds to 21.06 seconds. The heterogeneous computing achieves a speedup of 3.07 times over the original code that only runs on the two Intel Xeon E5-2692 CPUs.

Key words combustion simulation; heterogeneous many-core platform; Intel MIC; OpenMP 4.0; performance optimization

摘 要 LESAP是一个超燃冲压发动机燃烧数值模拟软件,可模拟发动机燃烧室内的燃烧化学反应与超声速流动,具有实际工程应用价值,其计算量巨大.面向通用CPU与Intel集成众核协处理器(many integrated core, MIC)构成的新型异构众核平台,使用新的OpenMP 4.0编程标准,实现了LESAP软件面向异构并行平台的移植,并采用SIMD向量化、数据传输优化、基于网格块划分的负载均衡等技术

收稿日期:2016-11-21;**修回日期**:2017-02-21
基金项目:国家自然科学基金国际合作与交流项目(61561146395);国家自然科学基金项目(11502296);国家"八六三"高技术研究发展计划基金项目(2012AA01A301)
This work was supported by the Joint NSFC-ISF Research Program, Jointly Funded by the National Natural Science Foundation of China and the Israel Science Foundation (61561146395), the National Natural Science Foundation of China (11502296), and the National High Technology Research and Development Program of China (863 Program) (2012AA01A301).
通信作者:车永刚(ygche@nudt.edu.cn)

进行了性能优化.性能测试结果表明异构版本比纯 CPU 版本性能更佳.在天河二号超级计算机的 1 个结点(含 2 个 12 核的 Intel Xeon E5-2692 CPU 加 3 块 Intel Xeon Phi 31S1P 协处理器)上,对一个实际超燃发动机燃烧数值模拟问题,网格规模为 532 万单元时,每时间步的平均执行时间从原来纯 CPU 版的 64.72 s 减少到 21.06 s,性能加速比达到约 3.07.

关键词 发动机燃烧数值模拟;异构众核平台;Intel 集成众核; OpenMP4.0;性能优化

中图法分类号 TP391

近年来,高超声速飞行器成为世界各航空航天大国关注和研究的热点[1],其马赫数超过 5(马赫数为飞行速度与当地声速之比,是无量纲数,记为 Ma.从马赫数可以大致了解飞行器速度的状况——是亚声速,还是超声速,甚至是高超声速.马赫数小于 1 者为亚声速,近乎等于 1 为跨声速,大于 1 为超声速;一般情况下,若马赫数大于 5 为高超声速),在军事和民用方面都具有巨大的潜在价值[2].实现高超声速飞行的关键是发动机,超声速燃烧冲压发动机(简称超燃冲压发动机)是高超声速飞行器主要推进装置之一.超燃冲压发动机内部的流动与燃烧机制非常复杂,对发动机内高速、复杂的非定常流的测试极为困难,并且测试的装置也非常复杂和昂贵,目前世界上只有少量可用的测试装置.基于计算流体动力学(computational fluid dynamics, CFD)数值模拟的方法成为研究超燃冲压发动机内部燃烧与流动问题的一种重要方法.

超燃冲压发动机数值模拟软件 LESAP(large eddy simulation for air-breathing propulsion)[3-6]采用 CFD 技术模拟超燃冲压发动机中的流动与燃烧.超燃冲压发动机内的超声速燃烧是复杂的多物理、多尺度问题,湍流、物质混合及燃烧反应相互耦合,流动与燃烧发生在极大时空尺度范围内,对其进行数值模拟需要精细的网格及大量时间步,导致巨大的计算开销,因此如何利用现代高性能计算机系统实现高效的超燃冲压发动机燃烧模拟软件成为关键问题.

在高性能计算机体系结构方面,以"主处理器+加速器或协处理器"方式进行协同工作的众核异构型高性能计算机日益成为主流[7].在 2016 年的 TOP500[8]中,位于第 2 的以 Intel 集成众核处理器 MIC (many integrated core)作为加速部件的天河二号[9]超级计算机就是典型的众核异构型计算机.针对这种 CPU+MIC 的异构并行[10]体系结构,目前主要有 5 种编程模式:CPU 原生模式、CPU 为主 MIC 为辅模式(也称 offload 模式)、CPU 和 MIC 对等模式、MIC 为主 CPU 为辅模式、MIC 原生模式.其中 CPU 为主 MIC 为辅模式是 MIC 编程中最常用的模式[11],目前 MIC 上的 offload 异构移植实现方式主要有 2 种:一种是由 Intel 研发的语言扩展 LEO(language extensions for offload)[12];另一种则是 2013 年 7 月由 OpenMP 架构评审委员会(OpenMP ARB)发布的 OpenMP 4.0[13].目前国内外基于 CPU+MIC 异构的 CFD 应用[14-18]移植方式都采用 Intel's LEO.

OpenMP 4.0 规范是用于共享内存并行系统的多线程程序设计的一套指导性注释编程接口.和旧版本的 OpenMP 相比,OpenMP 4.0 新增了对加速器、SIMD 的编程等功能.OpenMP 4.0 是业界的工业标准,独立于厂商,相较于 Intel 提出的 Offload 语法更具有可移植性和可继承性,编程也更为方便.本文尝试采用 CPU 为主 MIC 为辅模式,使用MPI+OpenMP 4.0 编程模型,对 LESAP 软件进行面向 CPU+MIC 混合异构平台的并行移植和性能优化.

1 发动机燃烧模拟软件 LESAP 分析

1.1 软件介绍

超燃冲压发动机燃烧模拟软件 LESAP 是典型的燃烧模拟软件,通过 2 阶时间分裂方法将化学反应与流场模拟解耦,实现超燃冲压发动机内湍流燃烧的数值模拟.在计算模型上,可采用混合雷诺平均 N-S 模拟/大涡模拟(navier-stokes/large eddy simulation, RANS/LES)[19]、大涡模拟或直接数值模拟(direct numerical simulation, DNS).对燃烧化学反应模拟采用了详细的化学反应模型,在流场求解时对无粘通量计算采用 5 阶 WENO(weighted essentially non-oscillatory)格式[20],对粘性通量采用 2 阶中心差分格式.时间格式采用隐式双时间步方法,其中的子迭代采用上下对称高斯-赛德尔(lower-upper symmetric Gauss-Seidel, LU-SGS)算法.

LESAP软件是典型的CFD应用软件,包含10 000余行Fortran-90代码,计算流程复杂,计算量巨大.原始版本实现了初步的MPI+OpenMP 2层混合并行,是高度并行化的应用,适合做CPU+MIC异构移植.整个LESAP软件数值模拟流程图如图1所示,输入文件主要有网格数据、边界条件和控制参数,网格输入文件按照区域分解原则被均匀划分成多个网格区块,每个MPI进程处理一个网格区块.主要计算部分在一个时间步循环中,包含燃烧化学反应计算(reaction)和流场通量计算(march),该时间步循环要迭代数万步才能获得可用的结果.

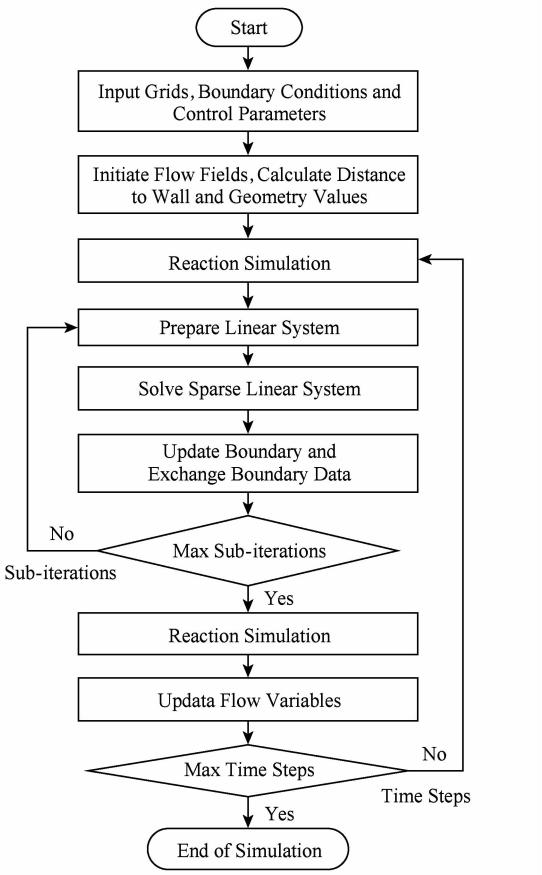

Fig. 1 Flowchart of LESAP

图1 LESAP软件数值模拟流程图

在现实模拟中,LESAP需要大规模精细的网格来捕获燃烧现象的详细情况,计算量巨大.因此,LESAP的计算开销很高,对其进行优化和异构移植具有重要意义.

1.2 计算热点分析

本文在至强E5-2692 v2处理器(主频2.20 GHz)上进行测试以分析软件的计算热点.采用原始的串行程序,测试数据为单块网格(含1 492 400个网格单元),使用Vtune测得各个子程序执行时间百分比,如表1所示.由表1可知,程序时间开销主要集中在与燃烧化学反应计算相关的子程序*ws*、与流场通量计算相关的子程序*f_weno*上.

Table 1 The Execution Time Percent of Subroutines in LESAP

表1 各个子程序占总执行时间的百分比

No.	Subroutine	Execution Time Percent/%
1	*ws*	89.76
2	*f_weno*	7.71
3	*wvalue*	2.41
4	*lusgs*	1.43
5	*hei*	0.93
6	*viscidflux*	0.72
7	*abc*	0.45
8	*reaction*	0.25
9	Other	0.80

2 面向CPU+MIC异构计算平台的并行优化

相比多核的英特尔至强处理器,英特尔MIC架构具有更小的内核和更多的硬件线程,以及更宽的矢量单元,可以满足高度并行化的应用需求,提高整体性能[11].MIC编程有许多编程模式,比较实用的模式有:1)native原生模式,程序只在MIC上执行计算任务,CPU处于空闲状态;2)offload加载模式,即从CPU端启动主函数,将部分计算核心通过offload模式加载到MIC上执行,Intel's LEO和OpenMP 4.0都可以完成加载任务;3)symmetric对等模式,即从CPU和MIC各自启动主函数,各自执行自己所负责的计算任务.

本文采用CPU为主MIC为辅的加载模式,使用MPI+OpenMP 4.0编程模型实现CPU+MIC异构协同并行,充分利用MIC众核的性能优势.CPU+MIC混合并行的难点主要是任务分配和负载均衡的问题,并且加载模式需要使用PCIe接口在两种设备之间传输大量数据,是主要的性能瓶颈[21].下面将介绍如何使用OpenMP 4.0实现CPU+MIC异构协同并行、SIMD向量化、CPU与MIC间数据传输优化以及任务分配和负载均衡问题.

2.1 OpenMP 4.0实现CPU+MIC异构协同并行

我们采用了一种异构并行计算模式,如图2所示.单结点上运行多个MPI进程,每个进程处理一个网格区块,每个进程使用OpenMP多线程方式

利用结点内的一组 CPU 核. MPI 进程有 2 种:一种是 P1 类型,完全在 CPU 上运行;另一种是 P2 类型,用一个 CPU 核发起主函数,通过使用 OpenMP 4.0 将计算量大、需要高度并行的计算核心加载到 MIC 上计算.

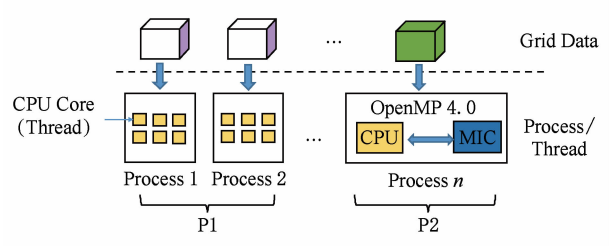

Fig. 2 The parallel model of running configuration

图 2 CPU+MIC 异构混合并行的并行模式

为了减少线程在核上的动态迁移带来的额外开销,我们采用了进程/线程绑定的方式. 对于 P1 类型的进程,根据进程数和 CPU 核数均匀分配多个 CPU 核,每个线程绑定一个 CPU 核;对于 P2 类型的 MPI 进程,则只需要一个线程,绑定到一个 CPU 核上.

我们设计灵活的输入参数,以便于充分利用硬件资源,如表 2 所示. 用户可以结合自身的硬件环境,动态地设置用于计算的 CPU、MIC 卡数以及每个结点用于计算的网格数,并且可以调整每个结点上的 CPU 与 MIC 卡的任务分配,从而解决负载不均衡的问题.

Table 2 The Input Control Parameters of LESAP

表 2 LESAP 运行时输入的控制参数

No.	Parameter	Meaning
1	*nstep_max*	The number of time step
2	*substp_max*	The number of sub-steps in each time step
3	*Nnode*	The number of total nodes
4	*NCoreperNode*	The number of CPU cores per node
5	*NBperNode*	The number of blocks per node
6	*NMIC*	The number of MICs per node
7	*NBperMIC*	The number of blocks per MIC
8	*NTperMICProc*	The number of threads per MIC MPI process

在用户设置了相应的参数后,LESAP 再用这些参数根据本文设计的变量计算方法,见表 3,得到相应的进线程绑定参数.

Table 3 The Variables Needed to Compute When the LESAP Runs

表 3 LESAP 运行时通过计算得到的变量

Variable	Value	Meaning
myid	0,1,2,…	The numbr of MPI processes
myLocalProcID	mod (*myid*,*NProcperNode*)	The ID of local processes
Process_Offload	if (*myLocalProcID*. LT. *Nb4CPU*) *Process_Offload*=. false. else *Process_Offload*=. true.	If *Process_Offload* is true, the process will run on CPU and MIC, else will run on the CPU only.
Nproc4MIC	*NBperMIC*×*NMIC*	The number of MPI processes for MIC
Nproc4CPU	*NProcperNode*-*NBperMIC*×*NMIC*	The number of MPI processes for CPU
NcoreCPUperNode	*NCoreperNode*-*Nproc4MIC*	The number of CPU cores per node
NTperCPUProc	*NcoreCPUperNode*/*Nproc4CPU*	The number of CPU threads per node
myMicID	mod ((*myLocalProcID*-(*NProcperNode*-*NBperMIC*×*NMIC*)),*NMIC*)	The ID of MIC device needed offloading in this process

Nproc4MIC 是每个结点控制 MIC 卡的进程数,为每块 MIC 卡处理的网格块数 *NBperMIC* 与使用的 MIC 卡数 *NMIC* 之积. 进程号按块分配,每个结点上编号靠后的 *Nproc4MIC* 个进程用于控制 MIC. 该类进程即为 P2 类型进程,即 offload 进程,每进程绑定一个 CPU 核进行控制. 剩下的每个结点上纯 CPU 计算的进程数 *Nproc4CPU*=*NProcperNode*−*Nproc4MIC*. 该类进程即为 P1 类型进程,即纯 CPU 计算,每进程绑定的线程数 *NTperCPUProc*=(*NCoreperNode*−*Nproc4MIC*)/*Nproc4CPU*.

为了更好地理解本文并行异构计算模式,这里给出一个例子,参数输入格式如下:

mpirun -n〈*RankNum*〉 -N 〈*NodeNum*〉 ./LESAP &〈*nstep_max*〉〈*substp_max*〉〈*Nnode*〉〈*NCoreperNode*〉 &〈*NBperNode*〉〈*NMIC*〉〈*NBperMIC*〉〈*NTperMICProc*〉.

假设输入为:

mpirun -n 8 -N 1 ./LESAP 1 10 1 24 8 3 2 224.

表示在单结点内运行 8 进程,处理的网格块数为 8. 单步迭代 10 次,单结点内有 24 个 CPU 核,处理 8 块

网格数据,使用 3 块 MIC 卡,每块 MIC 卡分别处理 2 个网格块,每个 MIC 进程开启 224 个线程. 那么 P2 类进程数 $Nproc4MIC = NBperMIC \times NMIC = 2 \times 3 = 6$,使用了 6 个 CPU 核来控制 offload. P1 类进程数 $Nproc4CPU = NProcperNode - Nproc4MIC = 8 - 6 = 2$,每 P1 进程绑定的 CPU 核数为

$$NTperCPUProc = (NCoreperNode - Nproc4MIC)/Nproc4CPU = (24-6)/2 = 9.$$

OpenMP4.0 编译指导语句中增加了! $omp target,可以将 CPU 上的数据和代码加载到 MIC 上运行. 如图 3 所示,OpenMP 4.0 可以将部分高并行化的计算任务交给 MIC 卡计算,比如 LESAP 中的燃烧反应模块中的 *ws* 子程序,实现以 CPU 为主 MIC 为辅的异构协同计算. 其中 to,from,tofrom 指定了 CPU 与 MIC 之间的数据传输方式,使用布尔变量 *use_coprocessor* 标记 MPI 进程的计算核是在 CPU 上运行还是在 MIC 上运行,device 可以指定使用的 MIC 卡设备号.

```
! $ omp declare target (list1 ,list2 ,list3 )
! $ omp declare target(ws)
! $ omp target if (use_coprocessor) device (MicID) &
   map (to:list1 ) &. / * Translate list1 from CPU to MIC * /
   map (from:list2 ) &. / * Translate list2 from MIC to CPU * /
   map (tofrom:list3 ) / * Translate list3 between CPU and
      MIC * /
      call ws ( ) / * The subroutine runs on MIC * /
! $ omp end target
```

Fig. 3 The implementation of offloading to MIC using OpenMP 4.0 device constructs

图 3 支持 MIC 设备加载模式计算的 OpenMP 4.0 实现

LEASP 软件计算流程复杂,函数调用关系复杂,异构移植的原则是根据表 1 中的子程序执行时间比例,尽量将运行时间开销大的子程序进行移植,在移植变量或者子程序到 MIC 之前,需要用 OpenMP 4.0 的编译指导语句! $omp declare target 对变量或者子程序进行声明.

2.2 CPU 与 MIC 之间的数据传输优化

CPU 与 MIC 之间大量的数据传输,是 CPU+MIC 混合异构并行的性能瓶颈. 本文通过以下方法对数据传输进行了优化:1)MIC 设备上的数据复用,不仅可以减少数据的传输时间开销,还可以减少 MIC 卡的内存开销;2)CPU 与 MIC 间按需进行数据传输,每次在 CPU 和 MIC 间进行数据传输时,只传递相应计算步中需要使用到的变量,以此来降低 CPU 与 MIC 之间的数据传输开销.

OpenMP4.0 提供了 2 种编译指导语句可以实现数据传输优化:1)! $omp target data,可以创建大的 device data 数据环境,当一个变量在一个封闭的 device data 数据环境内时,新的数据环境可以从这个封闭的环境中使用这个变量,也即复用这个变量,而不需要分配额外的内存,也不需要初始化和赋值. 2)! $omp target update,可以保持 MIC 端和 CPU 端的变量的值一致.

本文使用! $omp target data 尽早建立一个 device data 环境,将需要多次使用的数据传输到 MIC 上,并将需要加载到 MIC 上的代码段放到这个 device data 数据环境中. 在这个环境内可以通过! $omp target map 传输新数据,并将代码加载到 MIC 上. 如果出现数据更新时,则使用! $omp target update to/from 来传输数据,保证数据的一致性. CPU 与 MIC 之间数据传输优化的 OpenMP 4.0 实现方式如图 4 所示:

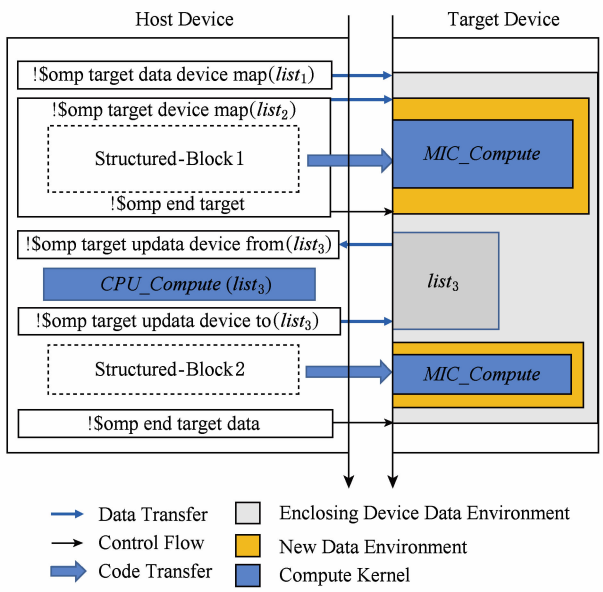

Fig. 4 Optimizing the data movement by OpenMP 4.0. the structured-block can be loops and or subroutines

图 4 CPU 与 MIC 之间数据传输优化的 OpenMP4.0 实现

2.3 CPU/MIC 协同计算负载均衡优化

由于主频与核数的不同,CPU 与 MIC 之间的处理能力存在差异. 为了达到不同平台之间的负载均衡,需要根据 CPU 和 MIC 的任务处理能力进行任务分配. 本文主要做了以下 2 个层次的负载均衡:1)CPU/MIC 协同计算时,单结点内 CPU 设备与 MIC 设备之间的负载均衡;2)设备(CPU 或者 MIC)内部各线程之间的负载均衡.

2.3.1 CPU/MIC 设备间的负载均衡

由于 CPU 与 MIC 的计算能力不同,所以 CPU 与 MIC 划分的网格数不能相同.动态分配的方式虽然在负载均衡性能上效果很好,但是会增加额开销,MPI 进程间通信的开销比进程内通信开销大很多.并且 LESAP 软件属于数据密集型应用,网格数据划分只能采用静态数据划分的方法,为了减少通信开销,我们采用静态任务划分,每个进程处理一个网格块.

为了充分利用 CPU 和 MIC 的任务处理能力,提高 LEASP 软件的最佳模拟性能,我们增加了 MIC 卡数、每个 MIC 卡的网格计算块数和 MIC 卡上的线程数等控制参数.在运行 LESAP 软件时,设置不同的控制参数可以对 MIC 和 CPU 进行任务分配.

假定整个模拟流场区域由 K 个均匀划分的网格块组成,单结点内创建 a 个 P1 类型的 MPI 进程,创建 b 个 P2 类型的 MPI 进程,整个计算的总进程数为 $P=a+b$.每个进程处理一个网格块,则 $P=K$,实现 CPU 与 MIC 设备间的负载均衡的关键是使用多少块 MIC 卡处理多少网格数据,下面建立其数学描述.

假设 M 为 CPU 端网格块数,i 为使用的 MIC 卡数,N 为每块 MIC 卡处理的网格块数,则有

$$a=M,\ M\in\{1,2,\cdots,K-1\},\tag{1}$$

$$b=i\times N,\ b\in\{1,2,\cdots,K-1\},\tag{2}$$

$$K=M+i\times N.\tag{3}$$

假设 t 为 MIC 线程数,T 为模拟运行时间,本文的负载平衡方案是通过调节 $M,i,N,t,MIC\_KMP\_AFFINITY$,寻找一个最佳负载分配方案,使得模拟运行时间 T 最小,即大致达到了负载均衡.图 5 为 8 块网格算例在单结点 CPU+MIC 异构平台上的任务分配图,8 块网格算例,2 块网格由 CPU 端处理,剩下 6块网格由 3 块 MIC 卡处理,每块 MIC 处理 2 块网格.

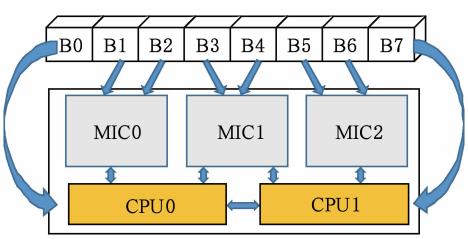

Fig. 5 8-grid calculation example allocated in a single-node CPU+MIC heterogeneous platforms

图 5 8 块网格算例在单结点 CPU+MIC 异构平台上的任务分配

2.3.2 CPU/MIC 设备内的负载均衡

设备内的负载均衡分 2 个层次,一个是循环迭代分配到线程上的方式,另一个是线程分配到硬件计算核心上的方式.在每个设备(CPU 或 MIC)内,通过 OpenMP 实现了多线程并行.对于第 1 个层次,OpenMP 提供了 schedule 设置线程分配方式,主要有 static 静态分配和 dynamic 动态分配.由于 LESAP 计算过程中每个进程所计算的网格单元数固定,并且每次迭代任务量相同,我们选择了 static 静态分配.

对于第 2 个层次,在 CPU 设备内,为了减少线程在核上的动态迁移带来的额外开销,我们采用一个线程绑定一个 CPU 核的方式.在 MIC 卡内,我们通过环境变量 *MIC_KMP_AFFINITY* 来设置 MIC 上的线程与 MIC 计算核心的绑定方式,可取的值有:1)scatter 分散模式,将线程优先分配到负载最轻的物理核心上;2)compact 紧密模式,将线程按顺序分配至逻辑核心上;3)balanced 平衡模式,尽量将线程分配到负载较轻的物理核心,也会尽量将相邻线程分配到同一个物理核心上.这 3 种方式都是静态划分方式,因此需要根据程序运行的实际负载情况,有针对性地进行设置,才能达到比较好的效果.

MIC 卡包含众多物理核,每个核上可以开启 4 个线程,只有让 MIC 卡上的核充分利用起来才能发挥 MIC 的最大性能.但如果线程数设置太多,线程开销比较大,也会造成负载不均衡,所以需要寻找一个合适的线程数,可以保证程序的并发度和 MIC 核的高可利用率.

2.4 *ws* 优化

由第 1.2 节的热点分析可知,与燃烧化学反应计算相关的子程序 *ws* 占据了主要的计算时间.子程序 *ws* 计算部分主要集中在一个 3 重循环中,如图 6(a)所示.由于 *ws* 子程序在 *reaction* 函数的循环中被调用,源代码为了提高 OpenMP 的并行粒度,直接对 *reaction* 中的循环做 OpenMP 并行化,子程序 *ws* 中的循环则不需再做 OpenMP 并行.由图 6(a)可以看出,IF 条件语句的作用是根据 *chemf* 变量和 *CC* 变量计算出中间变量 *rPow*,中间变量 *rPow* 在这个 3 层循环中的计算量是 $(ns-1)\times Nrec\times ns$ 次.而 $Chemf(ir,is)$,$CC(is)$ 与最外层循环无关,中间变量 *rPow* 的计算量实际上存在大量的冗余.这部分代码中的 IF-ELSE 分支语句导致编译器无法自动向量化.本文对其进行循环拆分,将 *rPow* 单独提前计算出结果,存放在一个二维数组

中,如图 6(b)所示,计算量则变成 $Nrec\times ns$ 次,计算量减少$(ns-2)\times Nrec\times ns$ 次,剩余的代码则可以通过增加!$omp simd private($is$) reduction(*:$pif$)编译指导语句实现向量化. ws 计算过程中存在很多类似图 6 中的 $rPow$ 中间变量,我们对这些中间变量都进行了循环拆分优化,计算量较大幅度降低,性能得到较大的提升.

```
① do ialpha=1, ns-1
②   do ir=1, Nrec
③     do is=1,ns
④       IF (chemf(ir,is).EQ.0) THEN
⑤         rPow=1.0;
⑥       ELSE IF (chemf(ir,is).EQ.1) THEN
⑦         rPow=CC(is);
⑧       ELSE
⑨         rPow=CC(is)×CC(is);
⑩       ENDIF
⑪       pif(ir)-pif(ir)×rPow.
⑫     enddo
⑬   enddo
⑭ enddo
```

(a) Before

```
① REAL::yPow(Nrec,ns+1)
② do ir=1,Nrec
③   do is=1,ns
④     IF (chemf(ir,is).EQ.0) THEN
⑤       yPow(ir,is)=1.0;
⑥     ELSE IF (chemf(ir,is).EQ.1) THEN
⑦       yPow(ir,is)=CC(is);
⑧     ELSE
⑨       yPow(ir,is)=CC(is)×CC(is);
⑩     ENDIF
⑪   enddo
⑫ enddo
⑬ do ialpha=1,ns-1
⑭   do ir=1,Nrec
⑮     !$omp simd private(is) reduction(*:pif)
⑯       do is=1,ns
⑰         pif(ir)=pif(ir)×yPow(ir,is);
⑱       enddo
⑲   enddo
⑳ enddo
```

(b) After

Fig. 6 The optimization for ws

图 6 ws 中的循环拆分优化

3 测试结果与分析

3.1 测试平台与环境

为了评估上节异构并行优化方案的有效性,我们在天河二号单个结点上进行了性能测试. 测试平台的环境与参数如表 4 所示,系统由 2 个 12 核的至强 E5-2692 v2 的 CPU 和 3 块 57 核的 Xeon Phi 31S1P (MIC)加速器组成. CPU 内存共 128 GB,单 MIC 卡共 8 GB 内存. 由于 LESAP 软件采用 Fortran90 语言开发,编译器使用 Intel ifort v14,支持 OpenMP 4.0. 使用"-O2-axAVX-fno-alias"的编译器优化选项,$MIC\_USE\_2MB\_BUFFERS$=32 KB. 性能计时仅针对图 1 流程中的时间步迭代,不计读取数据和初始化的时间,测试平均每个时间步的执行时间,每一个时间步含 10 次子迭代. 采用网格规模约 532 万(5 320 896)单元的 9 块网格算例进行性能测试.

Table 4 The Test Platform and Environment of One Node

表 4 单结点测试平台与环境

Items	Values
2×CPU	Intel Xeon CPU E5-2692 v2@2.20 GHz (12 cores)
3×MIC	Intel Xeon Phi 31S1P (57 cores)
Main Memory	128 GB (2×CPU)+24 GB(3×MIC)
Operating System	Linux 2.6.32-279-aftms-TH
Compiler	Intel ifort version 14.0.2
Optimization Options	-O2-fno-alias
MPI Version	MPICH Version 3.1.3

3.2 测试结果与讨论分析

我们将 LESAP 软件分为 3 个版本:1) MPI-OMP 表示实现了 MPI+OpenMP 两级并行,运行于 CPU 平台的原始版本;2) HYB-OMP4 表示使用 OpenMP 4.0 对原始版本进行了 CPU+MIC 异构并行移植,并经过数据传输优化和负载均衡优化后的运行于 CPU+MIC 混合平台的版本;3) HYB-OPT 表示对上述异构版本进行了向量化优化后的最终优化版本.

3.2.1 CPU 与 MIC 的负载均衡效果及分析

我们用规模约 532 万网格单元的 9 块网格算例对 LESAP 软件的原始版本 MPI-OMP 只使用 CPU 进行并行数值模拟,在单结点上单步迭代执行时间约为 64.72 s,占用 CPU 内存约 9.9 GB. 之后对 HYB-OMP4 版本进行负载均衡效果测试. 测试目标是找出对于该网格算例最佳的 CPU/MIC 的网格分配,MIC 线程数以及 MIC 线程绑定模式,使得 LESAP 软件负载均衡,达到最好的性能. 测试平台为含有 2 个至强 CPU 和 3 块 57 核的 MIC 组成的单结点,则 MIC 卡数最大值为 3. MIC 卡上的 57 个物理核心中包含 4 个硬件线程,因为其中 1 个物理核心通常用于运行操作系统及相关服务,所以余下的 56 个核含 224 个线程. 由于应用程序必须很好地扩展出

超过 100 个线程才能达到高度并行的要求[21],以充分利用 MIC 多核的优势,我们在每块 MIC 卡上使用的线程数分别为 112(每核上开启2 个线程)或者 224(每核上开启 4 个线程),并使用 9 块网格在不同 MIC 卡数、MIC 卡处理的网格块数以及 *MIC_KMP_AFFINITY* 情况下进行对比.

测试结果如表 5 所示,其中 i 为 MIC 卡数,N 为每块 MIC 卡处理的网格块数,t 为 MIC 线程数,*MIC_KMP_AFFINITY* 为设置 MIC 线程绑定模式的环境变量.根据实验环境的硬件条件进行不同的条件设置,使得 CPU 与 MIC 端的负载尽可能平衡,从而得到一个最佳的性能.由表 5 可知,对于 HYB-OMP4,线程的绑定方式对 LESAP 的性能没有较大的提升.线程数为 112 与 224 在不同的硬件配置下,性能差异不大.HYB-OMP4 的性能随着使用的 MIC 数 i 和每块 MIC 卡处理的网格块数 N 的增加而增加.

Table 5 The Execution Time of HYB-OMP4 on Different Cases

表 5 不同条件下 HYB-OMP4 版本的单步迭代执行时间

s

$i\times N$	Compact		Scatter		Balance	
	224	112	224	112	224	112
1×1	64.661 6	64.356 9	64.718 4	64.521 2	64.270 7	64.568 3
1×2	43.913 9	43.917 1	43.748	43.644 7	43.722 2	43.658 9
1×3	43.895 9	43.731 3	43.806 6	43.864 8	43.682 5	44.071 1
1×4	42.063 4	36.995	43.413 2	37.546 2	41.639 1	35.896
2×1	43.755 8	43.748 3	43.850 6	43.741 7	43.841 7	43.809 1
2×2	33.167 9	33.257 2	33.156 2	33.285 2	33.220 8	33.223 2
2×3	33.028 4	28.341 8	33.815 6	27.873 5	34.472 9	28.961 7
3×1	43.729 7	43.566 2	43.690 5	43.956 6	43.786 9	43.738 5
3×2	25.102 6	**23.284 7**	24.876 9	23.320 9	24.932 9	23.486 3

可以看出,LESAP 软件的 HYB-OMP4 版本在天河二号单结点异构平台处理约 532 万网格单元的 9 块网格算例时,MIC 线程数 $t=112$,绑定方式 *MIC_KMP_AFFINITY*=compact 时,选择分配给 CPU 端处理的网格块数 $M=3$,分配给 MIC 处理的网格块数 $i\times N=3\times 2=6$,单步迭代执行时间 T 最短,约 23.28 s,与原始版本相比,加速比达 2.78 倍.

3.2.2 HYB-OPT 最终版本的效果及分析

在 HYB-OMP4 的基础上进行 *ws* 优化之后,得到 HYB-OPT 版本,我们使用该 9 块网格算例进行测试以检验向量化优化的效果.取 CPU 端处理的网格块数 $M=3$,MIC 处理的网格块数 $i\times N=3\times 2=6$,MIC 线程数 $t=112$,绑定方式 *MIC_KMP_AFFINITY*=compact,测得的单步迭代执行时间约为 21.06 s.与 HYB-OMP4 相比,性能提升 9.5%,与原始的纯 CPU 版本 MPI-OMP 相比,加速比达到了 3.07 左右.LESAP 软件不同版本的单步迭代执行时间及加速比情况如图 7 所示:

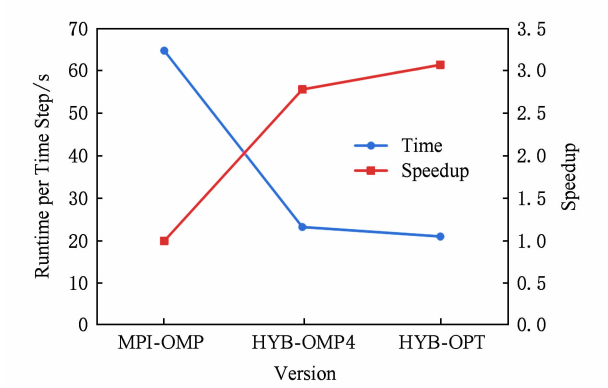

Fig. 7 The execution time and speed-up of different LESAP versions

图 7 LESAP 软件不同版本的单步迭代执行时间及加速比

4 结论与未来工作

本文面向 CPU+MIC 异构并行平台,针对一个实际的已经实现 MPI/OpenMP 两级并行的超燃冲压发动机燃烧数值模拟软件,使用 OpenMP4.0 实现以 CPU 为主 MIC 为辅模式的并行移植,并采用了数据传输优化、负载均衡优化及向量化优化等性能优化技术.针对异构平台计算能力的差异,进行了一系列的负载均衡分析和测试.性能测试表明,对于约 532 万网格单元规模的实际应用算例,LESAP 软件每时间步的平均执行时间从 64.72 s 减少到 21.06 s,相对于原始版本加速比达到 3.07.

本文工作充分利用异构并行计算平台的特点,显著缩短了超燃冲压发动机燃烧数值模拟的执行时间,节省了机时费用开销.后续工作主要是进一步提高 CPU+MIC 的协同并行计算能力,探究更为合理的 CPU 与 MIC 负载均衡模型,以取得更好的异构并行性能加速效果.

致谢 感谢国防科学技术大学航天科学与工程学院汪洪波提供 LESAP 原始代码!

参 考 文 献

[1] Niu Dongsheng, Hou Lingyun, Pan Pengfei, et al. Three-dimensional combustion numerical simulation of scramjet internal and external flow fields [J]. Journal of Aerospace Power, 2014, 29(4): 763-769 (in Chinese)
(牛东圣, 侯凌云, 潘鹏飞, 等. 超燃冲压发动机内外流场三维燃烧数值模拟[J]. 航空动力学报, 2014, 29(4): 763-769)

[2] Baidupedia. Hypersonic aircraft [OL]. [2016-04-12]. http://baike.baidu.com/view/3564607.htm

[3] Wang Hongbo, Wang Zhengguo, Sun Mingbo, et al. Combustion characteristics in a supersonic combustor with hydrogen injection upstream of cavity flameholder [J]. Proceedings of the Combustion Institute, 2013, 34(2): 2073-2082

[4] Wang Hongbo, Sun Mingbo, Qin Ning, et al. Characteristics of oscillations in supersonic open cavity flows [J]. Flow, Turbulence and Combustion, 2013, 90(1): 121-142

[5] Wang Hongbo, Wang Zhengguo, Sun Mingbo, et al. Large eddy simulation based studies of jet-cavity interactions in a supersonic flow [J]. Acta Astronautica, 2014, 93(1): 182-192

[6] Wang Hongbo, Wang Zhengguo, Sun Mingbo, et al. Simulations of combustion with normal and angled hydrogen injection in a cavity-based supersonic combustor [J]. Proceedings of the Institution of Mechanical Engineers Part G Journal of Aerospace Engineering, 2014, 228(4): 530-541

[7] Liu Li, Yang Guangwen. A highly efficient GPU-CPU hybrid parallel implementation of sparse LU factorization [J]. Chinese Journal of Electronics, 2012, 21(1): 7-12

[8] TOP500.org. Top 500 supercomputer sites [OL]. [2016-11-25]. http://www.top500.org/

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

[10] Wu Qiang, Yang Chanqun, Tang Tao, et al. Exploiting hierarchy parallelism for molecular dynaMICs on a petascale heterogeneous system [J]. Journal of Parallel & Distributed Computing, 2013, 73(12): 1592-1604

[11] Wang Endong, Zhang Qing, Shen Bo, et al. The Programming Guide for MIC High Performance Computing [M]. Beijing: China Water&Power Press, 2012

[12] Intel Corporation. User and Reference Guide for the Intel C++ Compiler 14.0 [OL]. [2016-04-12]. http://www.spec.org/mpi2007/flags/EM64T_Intel140_flags.20140908.html

[13] OpenMP Architecture Review Board. OpenMP application program interface version 4.0 [OL]. [2016-04-12]. http://www.openmp.org/

[14] McIntosh-Smith S, Boulton M, Curran D, et al. On the performance portability of structured grid codes on many-core computer architectures [C] //Proc of the 29th Int Conf on ISC 2014. Berlin: Springer, 2014: 53-75

[15] Liu Yang, Deng Liang. Acceleration of CFD engineering software on GPU and MIC [M] //Algorithms and Architectures for Parallel Processing. Berlin: Springer, 2015: 835-848

[16] Deng Liang, Bai Hanli, Zhao Dan, et al. Kepler GPU vs Xeon Phi: Performance case study with a high-order CFD application [C] //Proc of IEEE Int Conf on Computer and Communications. Piscataway, NJ: IEEE, 2015: 87-94

[17] Che Yonggang, Xu Chuanfu, Fang Jianbin, et al. Realistic performance characterization of CFD applications on Intel many integrated core architecture [J]. The Computer Journal, 2015, 58(12): 3279-3294

[18] Xu Chuanfu, Deng Xiaogang, Zhang Lilun, et al. Collaborating CPU and GPU for large-scale high-order CFD simulations with complex grids on the TianHe-1A supercomputer [J]. Journal of Computational Physics, 2014, 278: 275-297

[19] Spalart P, Jou W, Strelets M, et al. Comments on the feasibility of LES for wings, and on a hybrid RANS/LES approach [J]. Advances in DNS/LES, 1997, 1: 4-8

[20] Jiang Guangshan, Shu Chiwang. Efficient implementation of weighted ENO schemes [J]. Journal of Computational Physics, 1996, 126(1): 202-228

[21] Jeffers J, Reinders J. Intel Xeon Phi Coprocessor High Performance Programming [M]. San Francisco: Margan Kaufmann, 2013

Yang Meifang, born in 1992. Master of the National University of Defense Technology. Her main research interests include high performance computing and parallel optimization.

Che Yonggang, born in 1973. PhD and associate professor in the National University of Defense Technology. Senior member of CCF. His main research interests include parallel computing and performance evaluation of computer architectures.

Gao Xiang, born in 1991. Master of the National University of Defense Technology. His main research interests include high performance computing.