雅可比迭代的 CPU/GPU 并行计算及在 CFD 中的应用*

李大力，张理论，徐传福，刘巍

（国防科学技术大学 计算机学院，湖南 长沙 410073）

CPU/GPU Hybrid Parallel Algorithm of Jacobi Iteration and it's Application in CFD*

Li Da-Li, Zhang Li-Lun, Xu Chuan-Fu, Liu Wei

(School of Computer Science, National University of Defense Technology, Changsha 410073, China)

Abstract: In this paper, the characteristics of multi-core and many-core architecture are integrated to achieve the collaborative parallel computing of CPU and GPU for a real CFD application. Based on nested OpenMP thread, we implement OpenMP/CUDA hybrid parallization of Jacobi iterative method, and also use it for the aerodynamics_simulation of structural grid flow field. For a single-zone test case with 2 million grids, the speedup of GPU acceleration of computation for Right-Hand-Side, Left-Hand-Side matrix and it's inverse, and Jacobi iteration, is 11.35, 13.83 and 8.34 respectively, and the whole speedup is 9.86. For the test case with the same total grids and 4 data zones , the whole speedup of GPU computing and the CPU/GPU collaborative computing is 5.56 and 7.49 respectively.

Key words: Jacobi iteration; GPU; CFD; TH-1A; Heterogeneous hybrid parallel

摘要： 本研究从 CFD 实际应用背景出发，综合多核与众核的特点，采用 OpenMP 线程嵌套以实现 CPU/GPU 的协同并行计算，实现了雅可比迭代法的 GPU 加速以及 OpenMP/CUDA 混合并行算法，并将其用于结构网格气动外流场的数值模拟。对单区 200 万结构网格数据，右端项、左端项矩阵及矩阵求逆、雅可比迭代等核心计算部分的 GPU 并行分别取得了 11.35、13.83 和 8.34 倍的加速比，整个求解过程取得了 9.86 倍的加速比；对 200 万 4 区结构网格数据，整个求解过程的 GPU 并行加速比为 5.56，CPU/GPU 协同并行的加速比为 7.49。

关键词： 雅可比迭代；图形处理器；计算流体力学；天河-1A；异构混合并行

1 引言

以 NVIDIA 公司 Fermi 架构为代表的通用图形处理器（GPGPU）支持双精度浮点计算，大大拓宽了图形处理器在大规模科学计算中的应用。近几年来 TOP500 排行榜中利用通用图形处理器加速计算的系统数量不断增长。截至 2012 年 6 月，使用加速器的超级计算机数量为 58 台，使用 NVIDIA GPU 加速的有 53 台[1]；其中包括我国的“天河-1A”系统[2]，其计算结点采用一颗 Tesla M2050 的 GPU 和两颗 Intel Xeon X5670 的 CPU。

Table 1 Statistical chart of TOP500 Supper-Computer accelerator (June, 2012)

表格 1 2012 年 6 月 TOP500 的超级计算机中使用加速器的统计表

加速器类别	数量	所占比例（%）	总的最大持续性能 (GFlops)	总的峰值性能 (GFlops)	总核心数
N/A	442	88.4	107103147.87	140616821.21	12004293
Nvidia Fermi	53	10.6	14663388.84	28888263.12	1250784
IBM Cell	2	0.4	1168500	1537632	136800
ATI Radeon	2	0.4	364150	647436.2	26268
Intel MIC	1	0.2	118600	180992	9800

* Supported by the National Natural Science Foundation of China under Grant No.11272352 (国家自然科学基金);the National Grand Fundamental Research 973 Program of China under Grant No.G2009CB723803 (国家重点基础研究发展规划(973));the open project of State Key Laboratory of Aerodynamics(空气动力学国家重点实验室开放课题).

作者简介：李大力(1983－),男,湖北广水人,硕士,学员,主要研究领域为高性能计算,GPU 并行加速;张理论(1975-),男,河南人,博士,副研究员,主要研究领域为高性能计算;徐传福(1980-),男,安徽六安人,博士,助理研究院,主要研究领域为高性能计算,并行计算机性能评测;刘巍(1980-),吉林人,博士,主要研究领域为计算流体力学,高性能计算.

近年来，国内外已有较多基于 CPU/GPU 异构平台的 CFD 并行工作，将研究型 CFD 软件或 In-house 软件移植到异构平台上。例如 Dominik Goddeke 和 Robert Strzodka 等人在 GPU 集群上实现了对二维 Poisson 方程的隐式多重网格求解算法，其 GPU+CPU 并行实现相对于纯 CPU 实现加速了 2 倍左右[3]。Andrew Corrigan 等人针对无粘可压流的三维欧拉方程非结构代码，实现了显式 Runge-Kutta 的 GPU 并行加速算法，相对 OpenMP 实现和串行实现都取得了很好的加速效果；同时采用冗余计算来减少 GPU 线程通信，相对于无冗余计算的 GPU 并行实现取得了 3.9 倍加速[4]。S.J.Pennycook 和 G.R.Mudalige 等人通过 CFD 中的 NAS-LU 这一基准测试来评价 GPU 并行性能，采用不同规模数据上对包含几类 CPU、GPU 的工作站、集群进行了性能测试。在单工作站时，两颗 GPU 相对 CPU 分别加速 3 倍和 7 倍，集群实现也有很好的加速效果和性能功耗比[5]。张兵等利用 Roe 迎风格式实现了欧拉方程的隐式 GPU 并行求解算法，该算法可以很好的适用于结构网格和非结构网格。在 CPU 和 GPU 上，隐式雅可比迭代法明显快于显式方法（5 步龙格库塔）。随着数据规模的增大，GPU 并行实现（显式和隐式）的加速效果均有一定程度增加[6]。唐滔等基于 AMD 流处理 GPU，采用雅可比迭代法求解线性方程组，雅可比迭代的核心计算部分 SPMV（矩阵向量乘）在 GPU 上取得了 18.92 的加速比效果[7]。卢风顺提出了基于加速比的 GPU/CPU 负载分配分配模型，使得全局计算时间最短，该模型对未知数据的应用有一定的指导意义，但 GPU 启动时间等因素导致 GPU 上最佳负载分配比例往往小于模型预测结果，因此在实用中需根据经验进行调整[8]。

本文结合实际的 CFD 计算应用背景,研究雅可比迭代法的 GPU 并行优化,以及 CPU/GPU 的异构协同并行计算。研究应用背景是多区结构网格外流场的气动力学数值模拟。主要采用高阶精度格式（WCNS）和有限差分法并结合系列解法器来求解完全 N-S 方程,其中雅可比迭代法是该应用的一种求解方法。通过对不同规模数据的计算测试，验证了雅可比算法在 CFD 应用中的可行性；分析了雅可比迭代算法在 CFD 实际应用中相关计算瓶颈，包括右端项、左端项矩阵及矩阵求逆、迭代计算等几部分,并实现了 GPU 并行；在此基础上研究了雅可比迭代法 GPU/CPU 并行计算的简明性能分析模型；通过测试不同的 GPU 线程组织方式对 GPU 计算的影响，得到一些有实际意义的经验参数。相对于 CPU 单核，对单区 200 万结构网格数据，右端项、左端项矩阵及矩阵求逆、雅可比迭代等核心部分计算的纯 GPU 并行的加速比分别为 11.35、13.83 和 8.34，整个求解过程加速比为 9.86；对 4 分区的 200 万结构网格数据，整个求解过程的纯 GPU 加速比为 5.56，CPU/GPU 协同并行的加速比为 7.49。

2 雅可比迭代法和 CFD 计算瓶颈分析

2.1 雅可比迭代的数学原理[9]

用雅可比迭代法求解大型线性方程组：

$$Ax = b \tag{1}$$

其中 A 为系数矩阵，b 为右端向量，x 为解向量。

将矩阵 A 分解成矩阵 R 和对角矩阵 D，即：

$$A = D + R \tag{2}$$

对公式（1）和（2）进行变形，从而可以得到雅可比迭代法的公式如下：

$$x^{(k+1)} = D^{-1}(b - Rx^{(k)}) \tag{3}$$

由（3）式可知，在一个时间步内，各个点的迭代计算是相互独立互不影响的，具有很好的并发特点。

雅可比迭代法的收敛条件是矩阵 A 为严格或不可约的对角占优矩阵。严格行对角占优对应的条件如下：

$$|a_{ii}| > \sum_{i \neq j} |a_{ij}| \tag{4}$$

当然这条件是雅可比迭代法收敛的充分条件，有时候不满足此条件雅可比迭代法也可能收敛。本文主要是针对无湍流、定常问题进行研究，有稳定的收敛解，能够用雅可比迭代法进行求解。

2.2 可行性分析

原程序中实现了多种隐式求解器，包括点松弛（PR-SGS）[10,11,12]和 LU-SGS[13]等。LU-SGS 方法具有很好的稳定性和较长的时间推进步长，但是其计算存在很强的数据依赖关系，每一个网格点的计算都要严格依赖当前时间步的邻近点计算的数值，从而很难在 GPU 上实现大规模细粒度的并行[14]，目前国际上还没有很有效的实现方法和例子。点松弛法采用逐点松弛的方法进行求解，数据依赖关系和 LU-SGS 方法类似，较难实现高效的 GPU 并行。显式的龙格-库塔算法虽然有极好的数据独立性和计算并行性，但是其计算稳定性差、时间推进步长短、收敛效果不佳，在实际的应用中很少采用。

本文之所以选取雅可比迭代法作为研究对象主要是其相对于其它几类隐式算法对GPU特殊的众核体系结构有较好的适应性。雅可比迭代法在兼顾计算稳定性的同时也具有很好的数据独立性和计算并行性，能很好的适应 GPU 众核的计算体系结构。我们将雅可比迭代法与程序中现有的点松弛法在几个不同规模网格的算例上做了对比验证，二者的残差对数衰减过程基本相符合，故其 GPU 实现能很好适合实际应用需求。

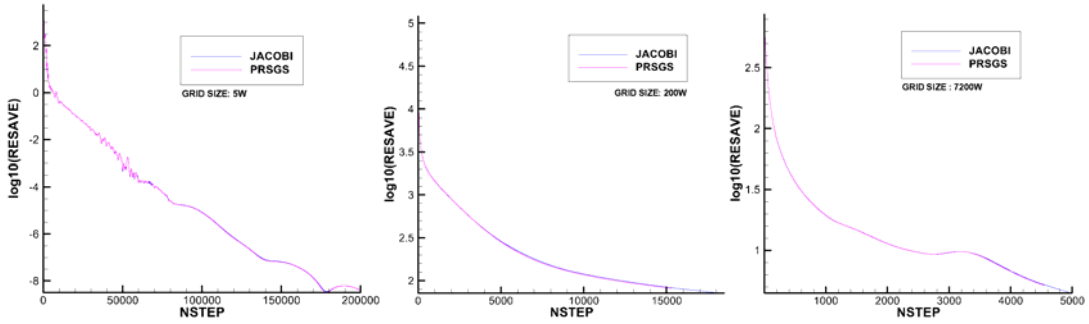

Fig.1 The log10(RESAVE) die-away curve of different scale data
图 1 不同规模网格点平均残差对数的衰减曲线

2.3 CFD的计算瓶颈分析

算法 1 是雅可比迭代法求解 CFD 的实际算法流程。雅可比迭代法本身主要的计算量是矩阵向量乘（即 R*x 以及 D^{-1} 与 b-R*x 的乘法运算）。但是在 CFD 实际的应用中，公式（3）中的右端项 b（Jacobian 矩阵的计算、矩阵向量乘），左端项矩阵 D（Jacobian 矩阵的计算）以及矩阵 D 的逆，雅可比迭代中矩阵 R（Jacobian 矩阵的计算）这些量在每个时间步推进中都是需要计算才能得出。本文研究中主要对其中的右端项计算（rhside）、左端项矩阵及其求逆（lhside_mat_matrix）以及雅可比迭代的计算（jacobi_iteration）这几个部分进行了 GPU 并行和 CPU/CPU 异构混合并行的实现。

算法1(雅可比迭代求解 CFD)

```
call pre_process
do ite = nstepst , nsteped
   call calc_other_variables
   call rhside
   call lhside_mat_matrix
      do iter = 1 , nsubmax
         call jacobi_iteration
         call residual
         call stop_subiter
         if ( nstop /= 0 ) exit
      end do
      call update
end do
call post_process
```

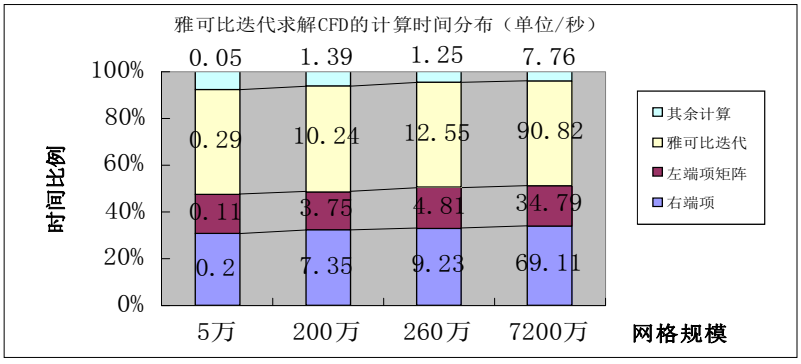

Fig.2 The CFD time distribution in different scale data
图 2 不同网格规模下雅可比迭代求解 CFD 的时间分布

图 2 是对不同网格规模下 CPU 串行计算的各主要部分占用时间进行的测试。图中的测试结果显示，随着网格规模的改变，各部分计算的耗时在迭代计算中所占的比例保持不变，计算时间和网格点规模近似成线性关系。本研究是针对定常问题，程序中雅可子迭代步数设置是 2，右端项、左端项矩阵和雅可比迭代大概分别占 33%、17%和 45%。如果将子迭代步数设置的更大一些，那么雅可比迭代过程所占的时间比例将更大。针对非定常问题，每个雅可比子迭代步中都要求解右端项和左端项矩阵及其逆，这两者所占的计算比例也会更大。

3 雅可比迭代的 GPU 实现

3.1 GPU并行流程和并行粒度

GPU 并不支持原程序中的指针数组嵌套，需要将一部分数据调整为高维数组。整个 GPU 并行计算的简要过程如图 3。先将计算所需的数据传输到 GPU，存在显存上，然后在时间步进中的每个核心计算部分调用 GPU 核函数进行并行计算，每个时间步迭代中还包含若干个雅可比子迭代步，一个时间步完成之后会输出守恒量增量等数据，在 CPU 上更新流场。

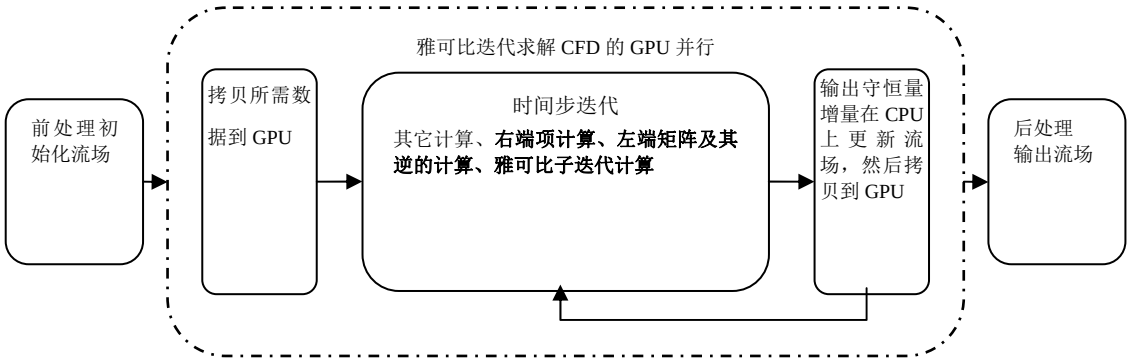

Fig.3 GPU parallel process of Jacobi iteration method in solving CFD
图 3 雅可比迭代求解 CFD 的 GPU 并行流程

上述流程图中，黑体部分（包括右端项计算、左端矩阵及其逆的计算、雅可比子迭代计算）实现 GPU 并行的，其余部分仍然是在 CPU 上计算。

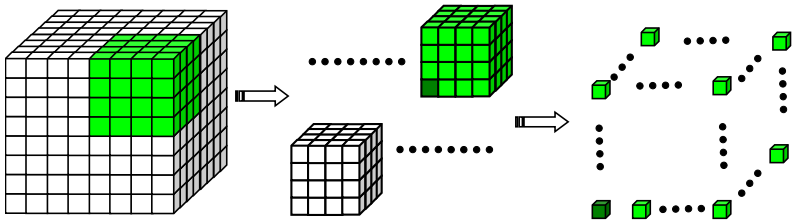

Fig.4 Fine grain parallel of Grid-Point level on GPU
图 4 GPU 上网格点级别的细粒度并行

如图 4 所示，GPU 上的并行是细粒度并行（网格点级），每个 GPU 线程处理单格点计算。将每个区块的数据划分为若干个并行的子数据块，其中每个子数据块对应一个 GPU 线程块，每个子数据块中的一个网格点对应一个 GPU 线程，在 GPU 的流多处理器上，程序和数据是以线程块为单位进行调度和切换的。

3.2 GPU并行性能模型

研究平台采用国家超算长沙中心的“天河-1A”超级计算机。单计算结点包含一颗 Tesla M2050 的 GPU 和两颗 Intel Xeon X5670 的 CPU。每颗 GPU 双精度浮点峰值性能为 515Gflops[15]，每颗 X5670 的峰值浮点性能是 70.32Gflops[16]。

由于 CPU 上的宿主语言采用 Fortran，用 Linux 环境下的 PGI 64 位 12.5 版本的 Fortran 编译器[17]，该编译器支持 CUDA 架构的 GPU 调用，便于直接用 Fortran 编写 CUDA 代码，编译时候需加入“-Mcuda”编译开关。从 12.3 之后的 PGI 版本支持 OpenMP 线程嵌套，从而可以很方便的编写 CPU 与 GPU 协同并行程序。但是目前 PGI 还只支持运行时调用的 OpenMP 线程嵌套，通过配置 OMP_NESTED 和 OMP_MAX_ACTIVE_LEVELS 环境变量来激活该功能。

根据阿姆达尔定律可以得到，整个迭代计算过程的加速比 Sg 和雅可比迭代中并行部分加速比 Sp 以及并行部分所占的计算时间比例 R 关系如下：

$$S_g = \frac{1}{\frac{R}{S_p} + (1-R)} \tag{5}$$

由于主要的数据传输在进入计算之初已经完成，期间 GPU 和 CPU 只有较少的数据交换，故这里将通信开销视为其它计算时间（串行计算）。经初步分析，影响雅可比迭代并行部分的 GPU 并行加速比 Sp 的主要因素为 CPU 核的浮点性能 Fc；GPU 浮点性能 Fg 以及 GPU 的实际性能效率 E，E 为一个无量纲数，可以根据算例实测出来。

$$S_p = \frac{F_g}{F_c} * E \tag{6}$$

综合（5）和（6），可以得到，迭代计算过程的加速比 Sg 的公式为：

$$S_g = \frac{1}{\frac{R}{\frac{F_g}{F_c} * E} + (1-R)} \tag{7}$$

影响 E 的因子主要有以下几个方面，GPU 核函数的占用率（occupancy），每个流多处理器上活跃的线程块数（不大于 8），GPU 的访存等因素。一般来讲，当占用率越大的时候，表明运行时活跃的线程数越多，线程并行度也就越高，GPU 的性能效率也就越高；流多处理器上活跃的线程块越多，GPU 通过线程块的调度，从而对访存开销的隐藏也就越好，GPU 的性能效率也就越高；GPU 上的访存带宽为 148G/s，访存延迟很小，但是如果在 kernel 中有较多零碎的访存操作时，当线程块规模较大时，也会成为限制 GPU 性能效率的瓶颈。

对本研究而言，Fg 为 515Gflops；Fc 的大小为 70.32Gflops/6，即 11.72Gflops；由图 2 的测试结果可以得到，可并行部分所占的计算比例大约等于 0.96；占用率表示 kernel 函数运行时 GPU 上最大活跃线程数占器件可支持的最大活跃线程数（1536）的比值，本实现中受资源（GPU 上的寄存器和共享内存）限制，最大的线程块大小只能为 512，可推知运行时每个流多处理器上的最大活跃线程数也只为 512，即占用率最优可为 0.333；最大活跃线程块数可由网格块规模以及占用率计算得出来；访存性能和每个线程访问全局内存的数量相关，当线程块越大的时候，一个线程块的访存开销也就越大；本实现的 GPU 实际性能效率大约为 1/3。综合以上数据，结合公式 6 和 7，可以计算出预期的并行部分加速比大约能达到 14.65，整个迭代求解过程的加速比大约能达到 9.48。

3.3 GPU并行的测试结果

GPU 上的并行程序性能影响因素较为复杂，尤其是 GPU 的线程分配方式，对 GPU 上核函数运行的占用率、活跃线程块数以及访存都有影响，往往需要与实际应用和具体实现相结合，并经过实测获得最佳的分配方式。我们对雅可比迭代的 GPU 版本的实现，分别在单区 200 万网格规模的算例上进行了测试，选取了几种不同的线程分配方式（见下图），以考察每个并行部分的最佳线程分配方式。图中的数值是每个并行部分在不同的线程分配模式下计算时间相对于该部分最短计算时间的比值。

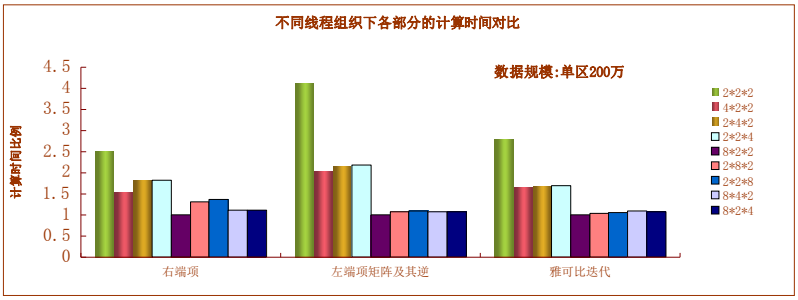

Fig.5 Time contrast of each computation part under different thread organization

图 5 不同线程组织下各部分的计算时间对比

由上述图表对比可以看出，针对本单区 200W（180*181*61）的网格数据，较优的线程块组织形式为 8*2*2，即一个 thread block 由 32 个线程组成，x 方向为 8，y 方向为 2，z 方向为 2。我们还可以看到，当线程块太小时候（小于 32），性能会有很大的降低，而且差不多线程块缩小一半，性能就降低一半。这是因为在 GPU 的核上，线程是以 warp 为单位执行的，一个 warp 由 32 个线程组成，他们相互之间可以共享数据。当线程块大于等于 32 的时候性能都比较好，但是由于我们 kernel 的寄存器占用较多，而且在流多处理器上是以线程块为单位调度的，当线程块较大，而且寄存器占用较多的时候就会限制一个流多处理器上可调度的线程块数，导致线程块规模再大的时候（大于 32），计算性能并没有提升，反而略有下降；当然这个可以通过优化 kernel，减少寄存器占用等手段来加以改善。结合本实现和算例特点，在 8*2*2 的线程配置下，得到的算法各部分的加速比以及总体的加速比如下图所示。迭代计算的总体加速效果可以达到 9.86 倍，当然这是在单区的情况下，多区时候由于区块间的 GPU 数据的通信，会导致性能有所降低，在后面的 GPU 与 CPU 协同计算部分就可以体现出来。

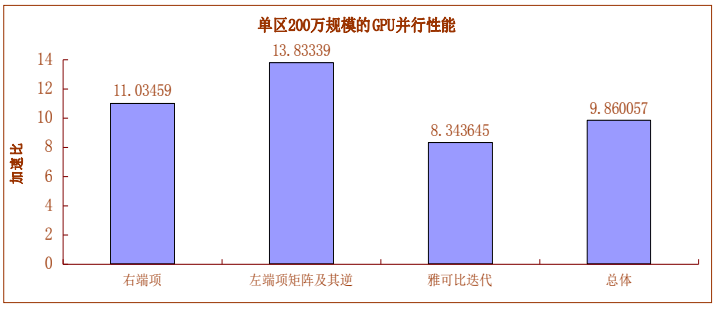

Fig.6 Speedups of core parts and entire solver by GPU acceleration

图 6 各核心部分以及总体求解计算的 GPU 并行加速比

4 CPU 与 GPU 的协同并行

4.1 协同并行实现流程

如上测试结果，可见 GPU 上的并行实现有较好的加速效果，考虑到每个结点上还有两颗 X5670 的 CPU，亦有很可观的计算资源，需要使用 GPU 和 CPU 的协同并行计算，更大限度的利用计算资源。

使用 OpenMP 线程嵌套来实现 GPU 和 CPU 的协同并行以及 CPU 上的 OpenMP 多核并行。GPU 和 CPU 的协同并行，主要是体现在网格区块级的粗粒度并行，在每个区块循环处通过 OpenMP 线程并行控制 GPU 和 CPU 分别对各自的负载部分进行计算，如图 7 所示。GPU 上的并行属于细粒度的并行（网格点级），过程和并行方式与上一章节的基本一致，区别在于数据准备阶段只拷贝部分网格块计算所需的数据到 GPU 上。而 CPU 上的并行是网格区块切面级的中等粒度并行，对每个网格快中 i、j、k 三个方向（三维情况下）中的 k 方向做 OpenMP 并行，如图 8 所示，一个线程处理一层的格点。CPU 上的 OpenMP 并行是共享存储的线程并行，并行的通信开销较小。

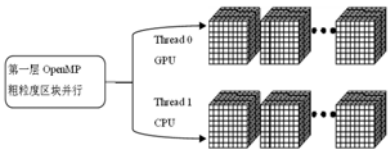

Fig.7 Coarse-grain grid-block level parallel
图 7 CPU/GPU 的区块级粗粒度协同并行

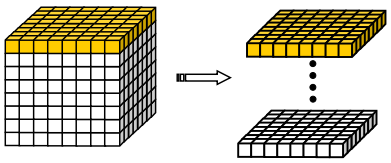

Fig.8 Middle-grain grid-layer level parallel
图 8 CPU 上中等粒度的网格层并行

4.2 协同并行的性能

先对原有的 CPU 串行程序的右端项计算、左端矩阵及其逆的计算、雅可比子迭代计算等核心计算部分做了 OpenMP 并行。图 9 是对单区 200 万的网格数据，通过不同的线程数（CPU 计算核心数）得到的总体求解计算部分的加速比对比图。采用 OpenMP 线程嵌套的方式将 CPU 上的 OpenMP 并行程序与前面实现的 GPU 并行程序结合起来，在网格区块、网格层以及网格点等三个不同的数据并行粒度上进行异构混合的协同并行计算。

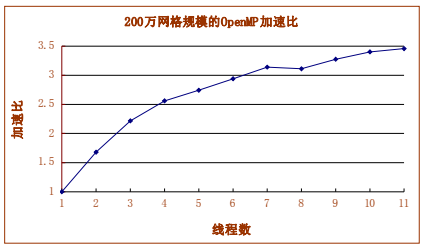

Fig.9 OpenMP Speedup curve
图 9 不同线程数下的 OpenMP 加速比

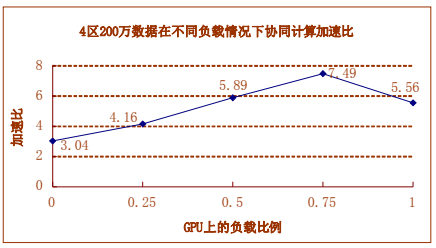

Fig.10 Collaborate parallel speedup curve
图 10 不同负载下的 CPU/GPU 协同并行加速比

在协同并行中，使用一个 CPU 核与 GPU 交互，用 10 个 CPU 核进行 CPU 上的 OpenMP 并行计算。图 9 测试结果，在 10 个 CPU 核（10 个 OpenMP 并行线程）时候，CPU 部分的加速比大约为 3.46，而 GPU 的实现的总体加速比效果大概是 9.86。按照加速比的大小来均衡负载，则 GPU 和 CPU 的负载比例大概是 2.85 左右的时候，协同并行的性能最佳。为此我们将 200 万的单区结构网格数据剖分成 4 个等规模区块的多区结构网格数据来进行测试。按照上面的负载比例，当 GPU 上处理 3 个分区、CPU 上处理 1 个分区时，理论上性能最优。

图 10 是实测的不同负载情况下的加速比结果，由图中可以看出在 GPU 的负载比例为 0.75 的时候，加速比效果较好，为 7.49。同时还可以看到，纯粹采用 GPU 的并行实现时，多区情况比单区情况的加速比要低，这主要是多区网格区块之间需要进行数据交换和通信造成的。

图 10 还可以得出，在同等规模数据的多区网格的 CPU/GPU 协同并行的加速比没有前面单区网格的纯 GPU 加速比效果好，这同样是因为通信和数据交换的增加导致的。但是由于 GPU 存储的限制（M2050 为 3GB），GPU 上测试的数据网格块的规模也有限，本实现将雅可比迭代计算部分所需的数据均对齐成高维数组拷贝到 GPU 的显存上，实测在 GPU 上同时计算的网格规模最大只能在 250 万左右。如果算例规模较小，则可以将其全部放于 GPU 上进行运算，将得到较好的加速效果；但一般来说，算例的网格规模都会比较大，无法全部在 GPU 上完成，这时候就可以将原来的网格剖分成多区网格进行 CPU/GPU 协同并行来加速计算。另外还可以恰当的剖分网格，兼顾负载均衡，减少区块数量，从而降低 GPU 和 CPU 间的通信开销，提高 GPU 和 CPU 协同并行效果。

5 总结与展望

本文分析了雅可比迭代法求解 CFD 应用的相关计算瓶颈，并对其进行 GPU 并行加速，同时结合“天河-1A”计算机结点的 CPU/GPU 异构的体系特点，采用 OpenMP 的线程嵌套功能实现了 CPU 和 GPU 的 OpenMP/CUDA 协同并行计算。同时还简单讨论了雅可比迭代法求解 CFD 应用的 GPU 并行的加速比性能模型，分析了影响 GPU 并行性能的主要因素。在单区 200 万结构网格数据上，右端项、左端项矩阵及矩阵求逆、雅可比迭代等核心计算部分的 GPU 并行分别取得了 11.35、13.83 和 8.34 倍的加速比，整个求解过程取得了 9.86 倍的加速比；对 200 万 4 区结构

网格数据，整个求解过程的 GPU 并行加速比为 5.56，CPU/GPU 协同并行的加速比为 7.49。

本文中的 GPU 并行实现和 CPU/GPU 协同并行实现还有优化空间，通过减少 GPU 寄存器占用，利用共享内存优化访存，提升核函数的占用率，来优化 GPU 并行的性能和效率。网格剖分的时候如果可以按照适当的比例对网格剖分，兼顾负载均衡，降低网格区块间的通信，提升 GPU 的并行效果。采用混合的求解方法，比如在 GPU 上使用易于并行的雅可比迭代法，在 CPU 上则采用较快速的 LU-SGS 方法，来提高 CPU/GPU 协同并行效率。结合 MPI，研究多 GPU、多节点的 CPU/GPU 并行计算，实现更大规模算例的异构混合并行计算。

致谢 感谢国家超算长沙中心提供的研究平台和技术支持，感谢中国空气动力研究与发展中心提供的 HOSTA 程序以及测试网格数据。

References:

[1] "top500 supercomputer sites", www.top500.org (2012/7/12).

[2] Yang, X.J., Liao, X.K., Hu, Q.F., Song, J.Q., 2011. The TianHe-1A supercomputer: its hardware and software, Journal of Computer Science and Technology 26(3), 344-351.

[3] Dominik Goddeke , Robert Strzodka , Jamaludin Mohd-Yusof , Patrick McCormick , Hilmar Wobker , Christian Becker , Stefan Turek, Using GPUs to improve multigrid solver performance on a cluster, International Journal of Computational Science and Engineering, v.4 n.1, p.36-55, November 2008.

[4] A. Corrigan, F. F. Camelli, R. Löhner, and J. Wallin. Running Unstructured Grid-based CFD Solvers on Modern Graphics Hardware. International Journal for Numerical Methods in Fluid, 2010.

[5] Pennycook, S., Hammond, S., Mudalige, G. and Jarvis, S.: Performance Analysis of a Hybrid MPI/CUDA Implementation of the NAS-LU Benchmark. In: 1st International Workshop on Performance Modeling, Benchmarking and Simulation of High Performance Computing Systems (PMBS 10) held in conjunction with IEEE/ACM Supercomputing 2010 (SCI10). New Orleans, LA, USA, 2010.

[6] 张兵,韩景龙. 基于 GPU 和隐式格式的 CFD 并行计算方法[J], 航天学报, 2010（2）, 249-256.

[7] Tang, T., Lin, Y.S., 2009. Design and Implementation of Jacobi and Laplace Algorithms on GPU Platform, Computer Engineering & Science 31(A1), 93-96.

[8] Lu, F., et al., 2011. CPU/GPU computing for long-wave radiation physics on large GPU clusters. Computers & Geosciences 41, 47-55

[9] 颜庆津, 数值分析, 北京: 北京航空航天大学出版社, 2000. 55-60..

[10] 张毅锋,邓小刚,毛枚良,陈坚强. 一种可压缩流动的高阶加权紧致非线性格式(WCNS)的加速收敛方法[A], 计算物理, 2007. 24(6), 698-704.

[11] 张毅锋,邓小刚,毛枚良,陈坚强. 高阶加权紧致非线性格式(WCNS)在二维流动计算中的加速收敛研究[A], 空气动力学报, 2008. 29(3), 249-355.

[12] 吕英华, 计算电磁学的数值方法, 北京: 清华大学出版社, 2006. 128-133.

[13] 阎超, 计算流体力学方法及应用, 北京: 北京航空航天大学出版社, 2006. 151-154.

[14] Zhang, L.P. and Wang, Z.J. “A Block LU-SGS Implicit Dual Time-Stepping Algorithm for Hybrid Dynamic Meshes,” Computer & Fluids Vol. 33, pp. 891-916, 2004.

[15] "Tesla M2050 / M2070 GPU 计算模块", http://www.nvidia.cn/object/product_tesla_m2050_m2070_cn.html (2012/7/12).

[16] "Intel Xeon Processor X5670", http://ark.intel.com/products/47920/Intel-Xeon-Processor-X5670-(12M-Cache-2_93-GHz-6_40-GTs-Intel-QPI) (2012/7/12).

[17] "The Portland Group", http://www.pgroup.com/ (2012/7/12).

雅可比迭代的CPU/GPU并行计算及在CFD中的应用

作者： Li Da-Li， 李大力， Zhang Li-Lun， 张理论， Xu Chuan-Fu， 徐传福， Liu Wei， 刘巍

作者单位： Li Da-Li,Zhang Li-Lun,Xu Chuan-Fu,Liu Wei(School of Computer Science,National University of Defense Technology,Changsha 410073,China)， 李大力,张理论,徐传福,刘巍(国防科学技术大学 计算机学院,湖南 长沙 410073)