DOI: 10.7511/jslx201704018

基于 FTM 算法的 GPU 加速

曾 良, 杜煜昊, 张 莹*, 胡 昱, 洪 瑶, 陈 虎

(南昌大学 机电工程学院,南昌 330031)

摘 要:为了解决 FTM(Front Tracking Method)算法在计算机中计算耗时长的问题,利用 CUDA(Compute Unified Device Architecture)来实现 FTM 算法在 GPU 中的并行计算。结合 GPU 并行计算架构的特性以及 FTM 算法的特点,本文通过共享内存的引入、线程块划分和线程块共享内存边界元素的纳入、迭代方法的改进和迭代过程中存储结构的变换等方法,提出了将 FTM 算法中的网格计算以及界面标记点处理方法在 GPU 中的实现方式。最后,通过模拟单气泡在静止液体中的自由上升运动,验证了 FTM 在 GPU 中计算的可行性与计算效率的提升。

关键词:FTM;CUDA;GPU;并行计算;计算流体动力学;数值模拟

中图分类号:O351.2;O242.1 **文献标志码**:A **文章编号**:1007-4708(2017)04-0511-06

1 引 言

随着计算机技术日趋成熟,计算机数值模拟方法逐渐应用于科学研究中,并得以高速发展。而伴随着数值模拟技术研究的不断深入,求解的问题趋于复杂,方程数量变多,方程求解的难度越来越大。近年来,图像处理器(GPU)发展迅速,其强大的浮点运算能力逐渐得到开发。相比之前的数值计算所用的 CPU,GPU 有着强计算能力、高精度、可移植以及并行能力强等特点,能够比较理想地解决上述问题,越来越受到科研机构和大型企业的重视。现在市场上能利用 GPU 进行运算的软件主要有 OPENCL 和 CUDA 等。其中,CUDA 因其以 C 语言为开发基础,编程人员不再需要掌握复杂的图形知识,并具有较强的通用性的特点[1],广泛运用到声波、热场、电磁场和页岩气[2-6]等领域中。在此之前,CUDA 已经在计算流体力学(CFD)[7,8]上得到广泛的应用,特别是在格子 Boltzmann(LBM)[9,10]和光滑粒子流体动力学(SPH)[11-13]。LBM 因其具有很高的并行性和较好的编程性,早期就开发运用到 CUDA 中,因此相关理论已经比较完善,应用效果也相当不错。而 SPH 需要模拟大量的粒子,计算量大,引入了并行计算之后,效率也大幅度提升。

在多相流问题的研究中对相界面的追踪至关重要,而 FTM[14] 以其数学方程物理理解的特性,具有能够精确捕捉运动界面以及通过界面重构来合理分配计算资源等特点。与其他直接的数值模拟方法相比,FTM 的计算稳定性在捕捉气液两相流方面具有一定的优势。因此,本文基于 GPU 并行计算架构的特性以及 FTM 算法的特点,提出了将 FTM 在 GPU 中实现的方法以及优化思路,从而达到提升计算效率的目的。

收稿日期:2016-04-05;**修改稿收到日期**:2016-08-24.

基金项目:国家自然科学基金(11562011);江西省自然科学基金(20151BAB202002)资助项目.

作者简介:张 莹*(1970-),女,博士,教授
(E-mail:yzhan2033@163.com).

2 Front tracking 算法

2.1 N-S 控制方程及处理

当控制方程应用到整个计算区域时,力都集中在界面上,比如表面张力,故必须引入一项并将它们看成体积力乘 δ 函数,该 δ 函数只有在界面上时才不为 0。通过上述修正,二维不可压缩流动的动量方程可写为

$$\frac{\partial}{\partial t}(\rho \boldsymbol{u})+\nabla(\rho \boldsymbol{u}\boldsymbol{u})=-\nabla p+\rho \boldsymbol{g}+\nabla\mu(\nabla \boldsymbol{u}+\nabla \boldsymbol{u}^{\mathrm{T}})+\sigma\kappa n\delta(x-x^{f}) \tag{1}$$

式中 x^f 为界面位置,ρ 和 μ 分别为非连续的密度场和粘度场,κ 为平均曲率,其他量遵循习惯约定。

为了求解上述方程,将式(1)分解,把其中的压力项单独分离出来,并引入一个临时速度 $\boldsymbol{u}^*$,得到计算无压力的速度离散方程,然后得到包含压力的速度通用离散方程,

$$(\boldsymbol{u}^{n+1}-\boldsymbol{u}^{*})/\Delta t=-(\nabla_h p/\rho^{n}) \tag{2}$$

式中 ∇_h 为取步长 h 的离散形式的哈密顿算子。

2.2 压力方程

对于不可压缩的流动,其离散形式的连续性方程可写为

$$\nabla_h \cdot \boldsymbol{u}^{n+1} = 0 \tag{3}$$

把式(2)代入式(1),获得一个简单的压力泊松方程:

$$\nabla_h \cdot \left(\frac{1}{\rho^n} \nabla_h p \right) = \frac{1}{\Delta t} \nabla_h \cdot \boldsymbol{u}^* \tag{4}$$

2.3 表面张力

界面追踪法对于界面的计算尤为重要,因此界面上表面张力的处理至关重要,其表达式为

$$F_\sigma = \sigma \kappa \boldsymbol{n} \delta(x - x^f) \tag{5}$$

式中 $\delta(x - x^f)$ 为狄拉克函数,当在界面上时为1,不在则为0。

取单位面元上的表面张力,对于二维流动有

$$\kappa \boldsymbol{n} = \partial \boldsymbol{t} / \partial s \tag{6}$$

因此得到单位界面元上的表面张力:

$$\delta F_\sigma^l = \sigma \int_{\Delta s} \frac{\partial \boldsymbol{t}}{\partial s} \mathrm{d}s = \sigma(\boldsymbol{t}_{l+1/2} - \boldsymbol{t}_{l-1/2}) \tag{7}$$

式中 $\boldsymbol{t}$ 为界面上的切向量,s 为界面元,l 为界面上的节点。

2.4 初始条件

算例中,流体计算区域为1×1的矩形区域,为了满足稳定性,指数函数的迭代精度为 10^{-4},压力迭代的精度为 10^{-3},时间步长取同时满足对流项和扩散项收敛的最小值,边界为周期性边界。初始时刻为0,初始速度为0,在浮力的作用下于静止流场中开始运动。

3 CUDA 基本架构

CUDA 程序架构有两部分,一部分在 CPU 上叫做主机(Host),另一部分在 GPU 上称为设备(Device)。主机用一个内核函数(kernel)完成一次从 CPU 到 GPU 的调用,在此架构下,CPU 主要负责串行计算和逻辑处理,而 GPU 则负责进行大规模并行运算。CUDA 中 kernel 执行任务时的最小单位是线程(thread)。数个 thread 可以组成一个线程块(block)。完成相同任务的 block 组成一个 grid。

每一个 block 有一个仅供自己 thread 读取的共享内存,而其他 block 中的 thread 无法读取此共享内存。因此,不同 block 中 thread 相互交换数据的效率较低,实际操作时需要尽量避免。并且每个 thread 都有各自的寄存器(register)和本地内存(local memory)的空间。此外,所有的 thread(即使不是同一个 block)都共享一份全局寄存器(global memory)、常量寄存器(constant memory)和纹理内存(texture memory)。而不同的 grid 则有各自的全局寄存器、常量寄存器和纹理内存。寄存器是显卡上最快的缓存器,但其存储大小有限,其次共享存储器、常量存储器和纹理存储器都有较快的速度但其存储大小都有限,而全局存储器虽然有较大存储空间,访问延迟却比寄存器高数百倍。因此,编程者需要灵活利用各种内存的存储特点,以达到良好的运行性能。CUDA 基本架构如图1所示。

4 FTM 算法在 GPU 中的实现

FTM 采用欧拉法描述的计算网格和拉格朗日法描述的表面点。本文把欧拉法描述的网格计算和拉格朗日描述的表面点的计算通过 GPU 实现。

4.1 网格的处理

FTM 算法通过差分将 N-S 方程的压力项单独分离出来,并引入一个临时速度,对流项和扩散项。首先,得到计算无压力的速度离散方程,并将这些量在交错网格上计算。为了达到在 GPU 中并行的目的,往往是一个 thread 处理对应一个节点,当计算网格较大时,一个 thread 计算若干个节点。为方便说明和处理,本文采用的是一个 thread 计算一个与其对应的节点。CUDA 中 thread 组成线程块,所以自然把计算区域分解为若干个与线程块对应的区域。关于线程块的大小与维度是由用户指定的,且对计算性能有着较大影响。

4.1.1 共享内存的引入和区域划分

以五点差分为例,每计算一个节点(i,j),thread 需要访问节点(i,j)和其周围的四个元素$(i,j+1)$,$(i,j-1)$,$(i-1,j)$和$(i+1,j)$,大部分节点的数据都需要访问5次。如果这些数据都是直接通过访问全局内存,效率显然是很低的,因为全局内存的读写速度很慢且受合并访存的影响较大。相比于全局内存,共享内存的访存速度明显要快。所以引入共享内存,将每个线程块的元素一次性读入共

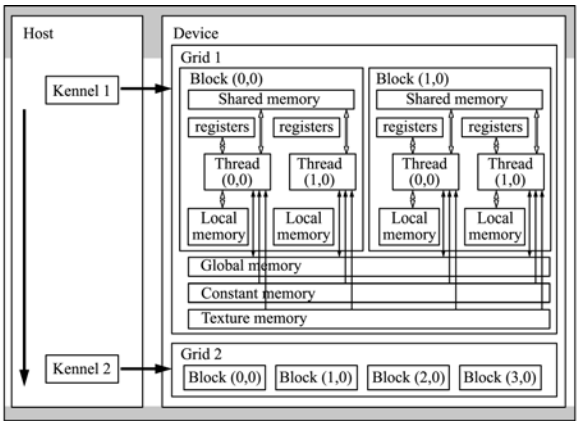

图1 CUDA 架构示意图

Fig. 1 Schematic diagram of CUDA architecture

享内存，每个线程块再访问其对应的共享内存，使效率大大提高。

如果共享内存只存储该线程块对应的区域节点的数据，当计算该线程块边界的节点时，需要访问其四周的元素数据，但是线程块之间的共享内存之间不可以相互通信，故只能访问全局内存，且需要通过逻辑语句来判断是否属于线程块的边界节点，从而产生分支。在 GPU 实际运算时，threads 按线程束(warp)为单位执行，一个 warp 由 32 个 threads 组成。由每个 warp 控制 threads 进行运算，即 warp 之间的关系是串行关系。一般情况同一个 warp 中的 thread 执行同样的指令，效率很高，但是当 warp 中的 thread 出现分支时，程序只能依次执行各个分支，效率很低。所以如果共享内存只存储该线程块对应的区域节点的数据，效率不好。为了减少这种分支的产生，本文设定每个线程块的共享内存不仅仅只存储该线程块对应区域节点的数据，而且还包括其周围的一层数据。如图 2 所示，深灰块为该 block 需计算的点，浅灰块为共享内存需额外存储的数据点，使程序的效率大大提高。

4.1.2　迭代处理

FTM 算法中，压力泊松方程以及密度场和粘度场的重构一般是通过逐次超松弛迭代法(SOR)求解。但是 SOR 迭代法本身并行性不好，一般将 SOR 迭代转换为红黑迭代或者多色迭代法求解[15]以提高并行性。本文将 SOR 迭代法转化为红黑迭代法在 GPU 中实现。本文中红黑迭代由黑白二色代替，如计算压力场时，需把数据分成黑白两组(本文为了便于说明，仅拿压力分组说明，其他参数以此类推)如图 3(a)所示，黑白块相互包围，每次黑色块所代表压力组 P_1 的更新是由它周围四个白色块和该黑色块之前的物理参数计算修正，白色块表示的压力组 P_2 同理。通过这样的处理，每次 P_1(P_2)的计算都是可并行的，这样求解压力的过程可在 GPU 并行完成。

为了提高访存效率，将数据分组，如图 3 所示。通过分组使黑色块和白色块具有连续的地址，使效率得到提升，且不会增大存储空间。

每一轮计算完黑块和白块的数据后，需要判断迭代是否满足精度以决定是否继续迭代。如图 4 所示，黑白块的计算是在 GPU 中完成，计算完成后判断是否满足精度要求，将判断后的结果数据从 GPU 传输到 CPU，从而决定是否继续迭代。但是 GPU 和 CPU 的通信是比较耗时的。为了减少这部分耗时，本文在每迭代一定的次数后再判断是否继续迭代，减少了 GPU 和 CPU 的通信次数，从而使效率提高。如果次数选取过大，GPU 和 CPU 的通信次数减小较多，可能导致计算在 GPU 的耗时增大；次数选取过小则 GPU 和 CPU 的通信次数减小不明显。其中，具体每迭代几次后再进行判断由实际需求决定，当计算的精度要求较高或者处理的问题迭代次数较多时(如气液两相密度比较大)，可以每多迭代几次再进行判断是否继续迭代。可以通过一些计算测试得到达到精度要求所需要的迭代次数的大致范围来选取。本文算例所选取的次数为 10 次。

4.2　相界面处理

在 FTM 算法中，为了准确捕获相界面，相界面通过一定数量的标记点表示，如图 5(a)所示，界

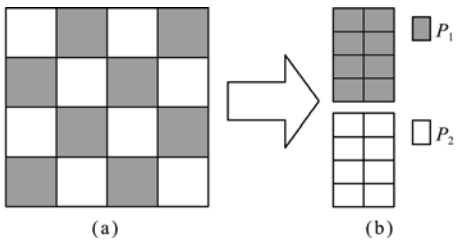

图 3　红黑迭代原理

Fig. 3　Red and black Iterative Methods

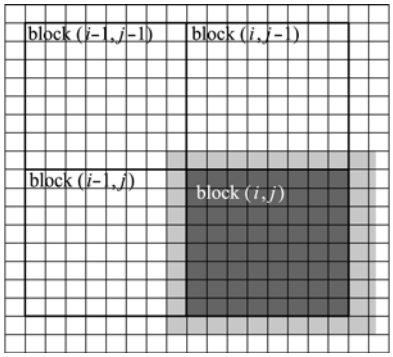

图 2　共享内存的存储方式改进

Fig. 2　Improved storage mode of shared memor

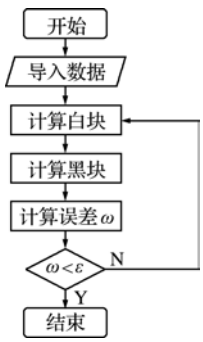

图 4　迭代过程

Fig. 4　Iterative process diagram

面捕获的精度越高，界面标记点的数量也越大，所以对界面的处理计算量通常很大。

界面的计算主要有表面张力计算和界面重构。表面张力的计算采用 Brackbili 的 CSF(Continous-Surface-Force)方法[16]，取单位面元上的表面张力来研究，得到单位界面元上的表面张力。通过双线性插值(图 5(b))将界面上的表面张力光滑到界面附近的网格上。界面的重构首先通过双线性插值从界面附近的网格得到界面标记点的移动速度，从而获得下个时刻的界面标记点位置，然后通过判读界面标记点之间的距离，增加或删除一些过于稀疏或密集的界面标记点。

关于界面计算在 GPU 中的处理，本文是将 thread 与界面标记点或单位界面元对应，从而达到并行的效果。当涉及一些需要两个或两个以上界面标记点的计算时，如计算相邻界面点之间的距离，每个界面标记点的数据会访问两次以上，将界面点的数据写入共享内存，使得效率提高。当网格和界面的信息进行交流时，是需要注意的。以表面张力光滑到网格为例，界面单元计算得到的表面张力需要光滑到附近的网格上，界面附近的某些网格节点的表面张力由多个界面元光滑后的张力叠加。这些节点可能受到多个 thread 同时访问，为了避免计算错误，叠加的过程在 GPU 中需要进行原子操作。

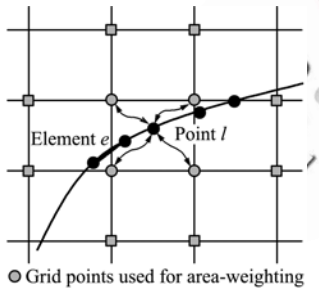

图 5 相界面示意图

Fig. 5 Schematic diagram of phase interface

5 计算结果与分析

为了检验 FTM 算法在 GPU 上实现的效果，本文模拟单个气泡在静止液体中自由上升的过程。基于的实验平台为 i7-4790，CPU 处理单元的频率是 3.6 GHz。GPU 为 GTX 960，拥有 2.3T Flop/s 的单精度浮点运算能力，1024 个流处理器(SP)，2 GB的全局内存，显存带宽最高可达 148.8 GB/s。

Clift 等[17]在研究单个气泡在液体中上升运动时，根据雷诺数(Re)、莫顿数(Mo)和厄特沃什数(Eo)得到了完整的气泡形状体系图，后来 Bhaga 等[18]对该研究做了进一步扩展。本文模拟了 Bhaga 等[18]实验中几种典型气泡形态的情况，得到了如图 6 所示的对比结果，与实验结果吻合，验证了算法的正确性。

本文主要分析 Eo 数对气泡形状的影响。通过 GPU 模拟了 Re＝28 时，不同 Eo 数(表面张力系数不一样，其他参数一样)时气泡在静止液体中的运动，分别模拟了网格规模为 128×128，256×256，512×512，1024×1024 和 2048×2048 五种情况，并且这五种网格都满足冯·纽曼稳定性条件。图 7 为网格规模为 256×256 时的计算结果。从图 7 可以看出，气泡在自由上升过程中从初始的圆形慢慢变扁，当 Eo 数较大时(Eo＝64)，气泡从圆形变成球帽形，最后形状达到稳定；而 Eo 数较小时(Eo＝32，Eo＝21.2，Eo＝16)，气泡从初始的圆形变成椭圆形，最后形状达到稳定。为了更清晰地描述气泡的变形，用横纵比来度量气泡的形状。图 8 为图 7 情况下，气泡在不同 Eo 数时横纵比随时间的变化，可以看出气泡的横纵比从 1 增大，最后增大到一个稳定值也就是形状达到了稳定，且 Eo 数越大横纵比最后达到的稳定值也越大，也就是气泡的形状越扁平，得到的结果符合 Bhaga 等[18]的结论。

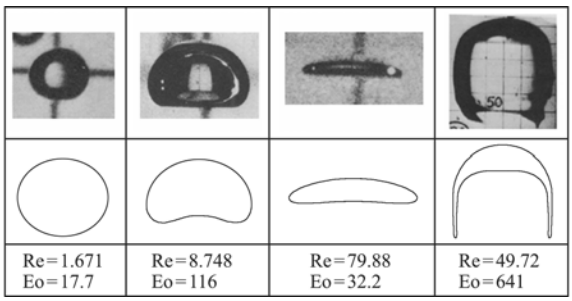

图 6 模拟与实验的对比

Fig. 6 Comparison of simulation and experiment

以 Eo＝64 情况进行 GPU 的计算效率分析，网格规模为 256×256 时，线程块大小分别为 4×4，16×16 和 32×32，得到的加速比分别为 1.814，6.234 和 3.184。这是因为当线程块过小时，每个线程块的 warp 数较少，不足以通过 warp 切换来隐藏访存时间；线程块过大时，线程块的数量较小，不足以使 GPU 的 SM 全部驻扎，效率也不高。表 1 为 Eo＝64 时，不同网格规模下，线程块选取最优大小，GPU 和 CPU 的计算耗时以及加速情况。可以看出，随着网格规模的增大，也就是计算密集程度的提升，GPU 的优势更加明显，当网格规模达到 2048×2048 时，CPU 计算耗时是 GPU 耗时的 28.9 倍。由于硬件的限制，GPU 的计算精度为单精度，所以与 CPU 的双精度相比存在计算误差，网格规模为 256×256 时得到的横纵比分析，误差为 0.3%。

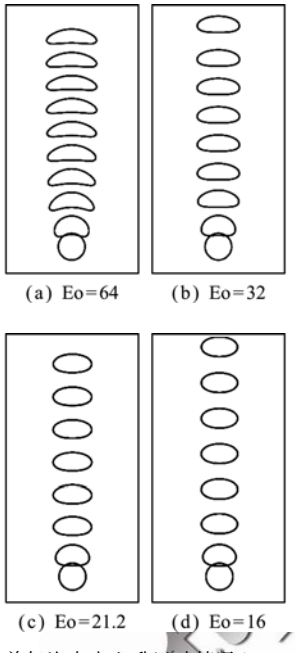

图 7　单气泡自由上升运动情况(Re＝28.3)

Fig. 7　Single bubble rising motion

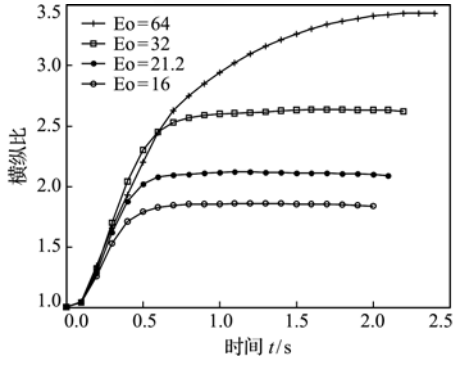

图 8　横纵比与时间的关系

Fig. 8　Horizontal and vertical ratio time diagram

表 1　网格大小与加速比的关系

Tab. 1　Relationship between grid size and speedup

网格规模	GPU/ms	CPU/ms	加速比
128×128	1070	3932	3.674
256×256	2956	18428	6.234
512×512	10141	122152	12.045
1024×1024	39317	747566	19.013
2048×2048	67922	1962987	28.900

6　结　论

本文结合 GPU 并行计算架构的特性以及 FTM 算法的特点，提出了一些处理方法，实现了 FTM 算法在 GPU 中的并行计算，并且通过模拟单气泡在静止液体中的自由上升运动，验证了 FTM 在 GPU 中计算的可行性。由于本文计算所用的 GPU 不是专业的计算卡，双精度的计算效率较低，所以没有采用双精度而是单精度，但其误差仍在合理的范围之内。根据此次试验可以看出，虽然 FTM 在算法层面上不具有较高的并行性，但是通过在计算密集的地方进行局部并行处理，并使用一些方法使算法的并行性提高，使得 FTM 也能到达很好的加速效果。

参考文献(References)：

[1]　NVIDIA Corporation. NVIDIA CUDA C Programming guide Version 6.5 [EB/OL]. [2014-08-01]. http://docs. nvidia. com/cuda/pdf/CUDA-C-Programming-Guide. pdf.

[2]　刘红伟，李　博，刘　洪，等. 地震叠前逆时偏移高阶有限差分算法及 GPU 实现[J]. 地球物理学报，2010，**53**(7)：1725-1733. (LIU Hong-wei，LI Bo，LIU Hong，et al. The algorithm of high order finite diffe-rence pre-steak reverse time migration and GPU implementation [J]. *Chinese Journal of Geophysics*，2010，**53**(7)：1725-1733. (in Chinese))

[3]　Rong F，Zhang W，Shi B，et al. Numerical study of heat transfer enhancement in a pipe filled with porous media by axisymmetric TLB model based on GPU [J]. *International Journal of Heat & Mass Transfer*，2014，**70**(3)：1040-1049.

[4]　Molnár F，Szakály T，Mészáros R，et al. Air pollution modelling using a Graphics Processing Unit with CUDA[J]. *Computer Physics Communications*，2010，**181**(1)：105-112.

[5]　杜刘革. 基于多 GPU 的 FDTD 并行算法及其在电磁仿真中的应用[D]. 山东大学，2011. (DU liu-ge. Multiple GPUs Based FDTD Parallel Algorithm and Its Applications in Electromagnetic Simulation[D]. Shandong

University,2011.(in Chinese))

[6] 牛 聪,黄 涛,王 磊,等.页岩气开采中的井底压力并行计算[J].计算力学学报,2014,**31**(3):396-401.(NIU Cong,HUANG Tao,WANG Lei,et al. Parallel computing for bottom-hole pressure in shale gas reservoirs[J]. *Chinese Journal of Computational Mechanics* 2014,**31**(3):396-401.(in Chinese))

[7] Kampolis I C,Trompoukis X S,Asouti V G,et al. CFD-based analysis and two-level aerodynamic optimization on graphics processing units[J]. *Computer Methods in Applied Mechanics & Engineering*,2010,**199**(9):712-722.

[8] Julien C T,Inanc S. CUDA implementation of a Navier-Stokes solver on multi-GPU desktop platforms for incompressible flows [A]. Proceedings of Aiaa Aerospace Sciences Meeting [C]. 2009.

[9] Kuznik F,Obrecht C,Rusaouen G,et al. LBM based flow simulation using GPU computing processor[J]. *Computers & Mathematics with Applications*,2010,**59**(7):2380-2392.

[10] Obrecht C,Kuznik F,Tourancheau B,et al. A new approach to the lattice Boltzmann method for graphics processing units [J]. *Computers & Mathematics with Applications*,2011,**61**(12):3628-3638.

[11] Hérault A,Bilotta G,Dalrymple R A. SPH on GPU with CUDA[J]. *Journal of Hydraulic Research*,2010,**48**(s1):74-79.

[12] Crespo A C,Dominguez J M,Barreiro A,et al. GPUs,a new tool of acceleration in CFD:efficiency and reliability on smoothed particle hydrodynamics methods [J]. *PLOS ONE*,2011,**6**(6):e20685.

[13] 王 珏,邱流潮.应用基于GPU的SPH方法模拟二维楔形体入水砰击问题[J].计算力学学报,2013,**30**(s1):174-177.(WANG Jue,QIU liu-chao. Application of the GPU-based SPH method to the simulation of water entry of 2D wedge bodies[J]. *Chinese Journal of Computational Mechanics*,2013,**30**(s1):174-177.(in Chinese))

[14] Unverdi S O,Tryggvason G. A front-tracking method for viscous, incompressible, multi-fluid flows [J]. *Journal of Computational Physics*,1992,**100**(1):25-37.

[15] 张宝琳.数值并行计算原理与方法[M].北京:国防工业出版社,1999.(ZHANG Bao-lin. *Principles and Method of Numerical Parallel Computation*[M]. Beijing:National Defence Industry Press ,1999.(in Chinese))

[16] Brackbill J U,Kothe D B,Zemach C. A continuum method form modeling surface tension [J]. *Journal of Computational Physics*,1992,**100**(2):335-354.

[17] Clift R,Grace J R,Weber M E. *Bubbles, Drops, and Particles*[M]. Academic Press,1978.

[18] Bhaga D,Weber M E. Bubbles in viscous liquids:shapes,wakes and velocities[J]. *Journal of Fluid Mechanics*,1981,**105**(1):61-85.

FTM algorithm based on GPU acceleration

ZENG Liang, DU Yu-hao, ZHANG Ying*, HU Yu, HONG Yao, CHENG Hu

(School of Mechanical and Electrical Engineering,Nanchang University,Nanchang 330031,China)

Abstract:In order to solve the problem of low computational effective of the FTM algorithm,this paper used CUDA to realize the parallel implementation of the FTM algorithm in GPU. Combining with the GPU parallel computing architecture and the characteristics of the FTM algorithm, and through introducing shared memory and bringing thread partition and thread block shared memory boundary elements into use,we proposed a method to implement the grid implementation of the FTM algorithm and a way of processing interface marked point in the GPU. At last,the feasibility and efficiency of FTM in GPU were verified by simulating the free ascending motion of single bubbles in a stationary liquid.

Key words:FTM;CUDA;GPU;parallel computing;computational fluid dynamics;numerical simulation

引用本文/Cite this paper:

曾 良,杜煜昊,张 莹,等.基于FTM算法的GPU加速[J].计算力学学报,2017,**34**(4):511-516.

ZENG Liang,DU Yu-hao,ZHANG Ying,et al. FTM algorithm based on GPU acceleration[J]. *Chinese Journal of Computational Mechanics*,2017,**34**(4):511-516.