CUDA 体系结构上的一层浅水系统的模拟

张 哲
(辽宁师范大学 计算机与信息技术学院计算机系,辽宁 大连 116029)

摘 要：对于使用支持 NVIDA CUDA 程序设计模型的 GPU 的二维一层浅水系统，给出了如何加速平衡性良好的有限体积模式的数值解，同时给出并实现了在单双浮点精度下使用 CUDA 模型利用潜在数据并行的算法。数值实验表明，CUDA 体系结构的求解程序比 CPU 并行实现求解程序高效。

关键词：GPU；浅水系统；OpenMP；CUDA
中图分类号：TP39 **文献标识码**：A **文章编号**：1674-7720(2012)10-0085-04

Simulation of one-layer shallow water systems on CUDA architectures

Zhang Zhe
(Computer Department, College of Computer and Information Technolage, Liaoning Normal University, Dalian 116029, China)

Abstract: The paper addresses how to speed up the numerical solution of a first order well-balanced finite volume scheme for 2D one-layer shallow water system by using modern graphics processing units (GPUs) supporting the NVIDIA CUDA programming model. Analgorithm which exploits the potential data parallelism of this method is presented and implemented using the CUDA model in single and double floating point precision. Numerical experiments show the high efficiency of this CUDA solver in comparison with a CPU parallel implementation of the solver.

Key words: GPU; shallow water systems; OpenMP; CUDA

以具有源项的守恒定律形式表达的浅水方程广泛用于引力影响下的一层流体建模，这些模型的数值解有许多用途，有关地面水流，如河流或溃坝水流的数值模拟。由于范围广，这些模拟需要大量的计算，因此需要很有效的求解程序在合理的执行时间内求解这些问题。

因为浅水系统的数值解有许多可开发的并行性，使用并行硬件可增加数值模拟速度。参考文献[1]给出了对于 PC 集群模拟浅水系统的数值模式和该模式的有效并行实现。参考文献[2]通过使用 SSE 优化的软件模块，改进了这个并行实现。尽管这些改进能在更快的计算时间内获得结果，但模拟过程仍需要很长的运行时间。

现代图形处理单元 GPUs 为以并行方式进行大规模浮点操作提供了数以百计的优化处理单元，GPUs 也可成为一种在复杂计算任务中大大提高性能的简便而有效的方法[3]。

曾有人建议将浅水数值求解程序放在 GPU 平台上。为了模拟一层浅水系统，在一个 NVIDIA GeForce 7800 GTX 显卡上实现了一个明确的中心迎风方法[4]，并且相对于一个 CPU 实现，其加速比从 15 到 30。参考文献[1]在 GPUs 上所给数值模式的有效实现在参考文献[5]中有相关描述。相对于一个单处理器实现，在一个 NVIDIA GeForce 8800 Ultra 显卡上得到了两个数量级的加速。这些先前的建议是基于 OpenGL 图形应用程序接口[6]和 Cg 着色语言(动画着色语言)的[7]。

近来，NVIDIA 开发了 CUDA 程序设计工具包，以 C 语言为开发环境，对一般目的的应用，使 GPU 的程序设计更为简便。

本文目标是通过使用支持 CUDA 的 GPUs 加速浅水系统的数值求解，特别是为了取得更快的响应时间，修改参考文献[1]和参考文献[5]中并行化的一层浅水数值求解程序，以适应 CUDA 体系结构。

1 数值模式

一层浅水系统是一个具有源项的守恒定律系统，该系统可以为在引力加速度 g 作用下的占有一定空间的均匀流动的浅水层建模。系统形式如下：

$$\frac{\partial W}{\partial t}+\frac{\partial F_1}{\partial x}(W)+\frac{\partial F_2}{\partial y}(W)=\begin{bmatrix}0\\ gh\\ 0\end{bmatrix}\frac{\partial H}{\partial x}+\begin{bmatrix}0\\ 0\\ gh\end{bmatrix}\frac{\partial H}{\partial y} \quad (1)$$

其中，

$$W=\begin{pmatrix}h\\q_x\\q_y\end{pmatrix},F_1(W)=\begin{pmatrix}q_x\\\frac{q_x^2}{h}+\frac{1}{2}gh^2\\\frac{q_xq_y}{h}\end{pmatrix},F_2(W)=\begin{pmatrix}q_y\\\frac{q_xq_y}{h}\\\frac{q_y^2}{h}+\frac{1}{2}gh^2\end{pmatrix}$$

而 $h(x,y)\in R$ 表示在 t 时刻点 (x,y) 处的水层厚度，$H(x,y)\in R$ 是从一个固定参考水平测量的深度函数，$q(x,y,t)=(q_x(x,y,t),q_y(x,y,t))\in R^2$ 是 t 时刻点 (x,y) 的水层质量流。

为了离散化系统(1)，将计算区域 D 分成 L 单元或有限体积 $V_i\in R^2$，并假定其为四角形。给定一个有限体积 V_i，$N_i\in R^2$ 是 V_i 的中心，$\mathcal{N}_i$ 是下标 j 的集合，使单元 V_i 和 V_j 是邻居；Γ_{ij} 是两个相邻单元 V_i 和 V_j 的公共边，而 $|\Gamma_{ij}|$ 是它的长度；$\boldsymbol{\eta}_{ij}=(\eta_{ij,x},\eta_{ij,y})$ 是单位向量，而该单位向量对于边 Γ_{ij} 和有关单元 V_j 的点是正规的[1]。假设在 t^n 时刻近似值 W_i^n 已经计算，为了提前完成，以 Δt^n 作为时间步长，应用如下的数值模式[1]：

$$W_i^{n+1}=W_i^n-\frac{\Delta t^n}{|V_i|}\sum_{j\in\mathcal{N}_i}|\Gamma_{ij}|\boldsymbol{F}_{ij}^{-} \tag{2}$$

其中 $|V_i|$ 是 V_i 的面积，$\boldsymbol{F}_{ij}^{-}\in R^3$ 是一个向量，此向量的计算涉及许多取决于 W_i^n 和 W_j^n 的线性代数操作[1]，使用如下条件计算 Δt^n：

$$\Delta t^n=\min_{i=1,\cdots,L}\left\{\left[\frac{\sum_{j\in\mathcal{N}_i}|\Gamma_{ij}|\|\boldsymbol{D}_{ij}\|_\infty}{2\gamma|V_i|}\right]^{-1}\right\} \tag{3}$$

其中 $\gamma(0<\gamma\leqslant 1)$ 是 CFL(柯朗—弗里德里希斯—卢伊)参数，$\boldsymbol{D}_{ij}\in R^{3\times 3}$ 是一个对角矩阵[1]。

2 CUDA 实现

这部分主要描述数值模式的潜在数据并行和它在 CUDA 体系结构上的实现。

2.1 并行源

以下给出了从数值模式的数学描述中获取并行源的主要计算步骤，其中每步都具有高度并行性，因为在每个边或体积所完成的计算与其他边或体积完成的计算无关。

起初有限体积网格从输入数据构建，然后重复处理时间步长直到到达最终的模拟时间。

(1)基于边的计算：对于连接两个单元 V_i 和 $V_j(i,j\in\{1,\cdots,L\})$ 的每个边 Γ_{ij} 需完成两个计算：

①计算向量 $\boldsymbol{M}_{ij}=|\Gamma_{ij}|\boldsymbol{F}_{ij}^{-}\in R^3$，它有助于计算每个边对其邻近单元 V_i 和 V_j 的最新状态[2]。并且还要将其分别加到与 V_i、V_j 相关联的部分和(M_i 和 M_j)中。

②必须计算 $Z_{ij}=|\Gamma_{ij}|\|\boldsymbol{D}_{ij}\|_\infty$ 的值，它有助于计算每个边对其邻近单元 V_i 和 V_j 的 Δt 值[3]，并且还要将其分别加到与 V_i、V_j 相关联的部分和(Z_i 和 Z_j)中。

(2)对于每个体积，局部 Δt_i 的计算[8]：$\Delta t_i=2\gamma|V_i|Z_i^{-1}$。

(3)Δt^n 的计算：对于每个体积 V_i，所有已计算的局部 Δt_i 的最小值就是 Δt^n。

(4)W_i^{n+1} 的计算：从第 n 个状态和以前阶段计算的数据中可以计算出每个体积 W_i^{n+1} 的第 $n+1$ 个状态。

由于数值模式显示了高度潜在的数据并行性，在 CUDA 体系结构上实现是一个好的选择。

2.2 算法实现细节

本文考虑的主要问题是双维正规有限体积网格。这个算法的一般步骤如图 1 所示。

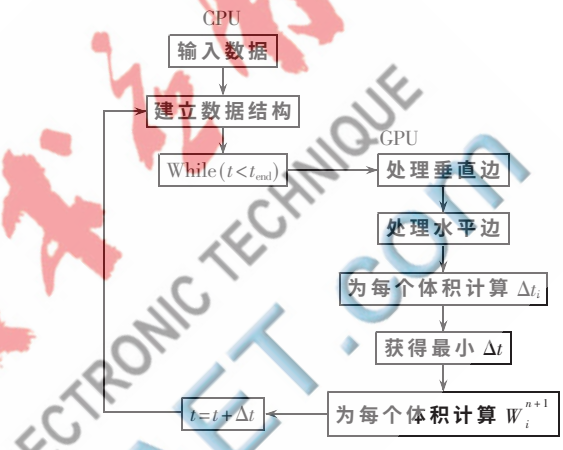


图 1 在 CUDA 体系结构上实现的并行算法的一般步骤

将在 GPU 上执行的每个处理步骤分配到同一个 CUDA 核，核是在 GPU 上执行的一个函数，其在执行中形成并行运算的线程块[9]，算法实现步骤如下：

(1)建立数据结构

对于每个体积，存储它的状态(h、q_x、q_y)和高度 H。定义一个 float4 类型的数组，其中每个元素表示一个体积并且包含了以前的参数，因为每个边(线程)仅需要它的两个相邻的体积的数据，将数组存储为一个二维纹理，纹理存储器特别适合每个线程在它周围环境进行存取。另一方面，当每个线程需要存取位于全局内存的许多邻近元素时，每个共享内存块更适合，每个线程块将这些元素的一小部分装入共享内存。欲实现两个版本(使用二维纹理和使用共享内存)，用二维纹理可使执行时间更短。

体积区域和垂直水平边的长度需要预先计算，并传递到需要它们的 CUDA 核。运行时通过检查网格中线程的位置，能知道一个边或体积是否是一个边界和一个边的 $\boldsymbol{\eta}_{ij}$ 值。

(2)处理垂直边和水平边

将边分成垂直边和水平边进行处理。垂直边 $\eta_{ij,y}=0$，水平边 $\eta_{ij,x}=0$。在垂直和水平边处理中，每个线程分别代表一个垂直和水平边，计算它对邻近体积的贡献在 2.1 节已描述。

当借助于存储在全局内存的两个累加器驱动一个特定的体积时，边（即线程）彼此相互同步，并且每个边都是 float4 类型的数组元素。每个累加器的大小就是体积值。累加器的每个元素存储边对体积的贡献（一个 3×1 向量 M_i 和一个 float 值 Z_i）。在垂直边的处理中，每个边将贡献写入第一个累加器的右面体积和第二个累加器的左面体积。水平边的处理与垂直边处理类似，只是加入累加器的贡献不同。图 2 和图 3 分别显示了垂直边和水平边的处理情况。

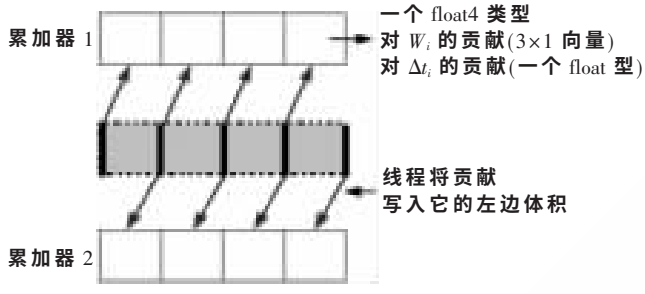


图 2 垂直边处理

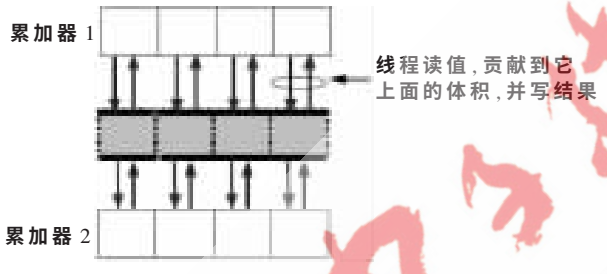


图 3 水平边处理

(3)为每个体积计算 Δt_i

每个线程代表一个体积，正像 2.1 节描述的计算体积 V_i 的局部值 Δt_i。最终将通过把存储在两个累加器中的相应体积 V_i 位置的两个 float 值相加获得 Z_i 值。

(4)获得最小 Δt

通过在 GPU 上应用规约算法寻找体积的局部 Δt_i 的最小值，所应用的规约算法是 CUDA 软件开发工具中所包含的规约样例的核 7（最优化的一个）。

(5)为每个体积计算 W_i^{n+1}

在这一步，每个线程代表一个体积，正如 2.1 节描述的修改体积 V_i 的状态 W_i。通过两个 3×1 向量相加求出 M_i 的最终值，这两个向量存储在与两个累加器的体积 V_i 相称的位置。因为一个 CUDA 的核函数不能直接写进纹理，需要通过将结果写进一个临时数组来修改纹理，然后将这个数组复制到绑定纹理的 CUDA 数组。

这个 CUDA 算法的双精度（double）版本已实现，以上所描述的有关实现的差别是需要使用两个 double2 类型元素的数组存储体积数据，并需要使用 double2 类型元素的 4 个累加器。

3 实验结果

在[-5，5]×[-5，5]区域内，考虑循环溃坝问题。深度函数是 $H(x,y)=1-0.4e^{-x^2-y^2}$，并且初始条件是：

$$W_i^0(x,y)=\begin{bmatrix} h^0(x,y) \\ 0 \\ 0 \end{bmatrix}$$

其中 $h^0(x,y)=\begin{cases} 2 & \text{if } \sqrt{x^2+y^2}>0.6 \\ 4 & \text{otherwise} \end{cases}$

数值模式运行在不同的网格。在[0，1]时间间隔内，执行模拟系统。CFL 参数是 $\gamma=0.9$，并且需要考虑墙壁边界条件（$q\cdot\eta$）。

已经实现了 CUDA 算法的一个串行和四核 CPU 并行版本（使用 OpenMP[8]），两个版本都使用 C++实现，且使用矩阵操作本征库[10]，在 CPU 上已经使用了 double 数据类型。将 CUDA 实现与参考文献[5]中描述的 Cg 程序进行比较。

所有程序都是在一个具有 4 GB 内存的酷睿 i7920 处理器上执行，所使用的显卡是 GeForece GTX260 和 GeForce GTX280。表 1 显示了所有网格和程序的执行时间（由于没有足够的内存，有些情况不能执行）。

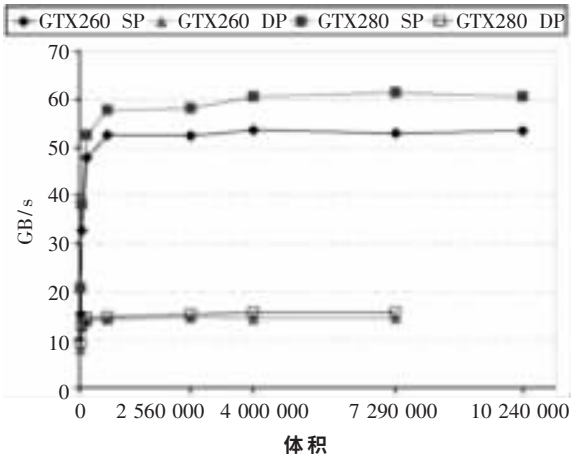
在具有两种显卡的所有情况下，单精度 CUDA 程序（CUSP）的执行时间超过 Cg 程序的执行时间。使用一个 GeForce GTX280，相对于单核版本，CUSP 实现了超过 140 的加速。对于复杂问题在两种显卡下，双精度 CUDA 程序（CUDP）比 CUSP 程序的执行速度慢 7 倍。正如预期的一样，相对于单核版本，OpenMP 版本仅实现了不足 4 倍的加速（OpenMP 版本的执行速度比单核版本快不到 4 倍）。

表 1 所有网格和程序的执行时间 (s)

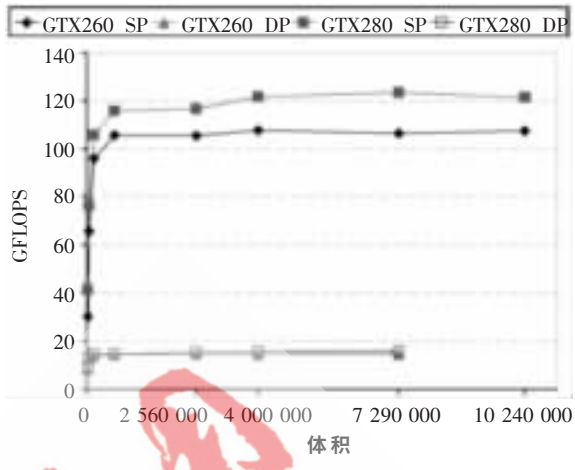
体积	CPU		GTX260			GTX280		
	1 core	4 cores	Cg	CUSP	CUDP	Cg	CUSP	CUDP
100×100	0.7	0.2	0.11	0.02	0.07	0.08	0.01	0.06
200×200	5.4	1.5	0.26	0.07	0.37	0.20	0.06	0.36
400×400	44.6	13.9	0.84	0.40	2.75	0.68	0.36	2.63
800×800	358.7	112.3	4.42	2.92	21.45	3.75	2.66	20.81
1 600×1 600	2 883.5	898.1	30.72	23.49	167.5	26.14	21.22	161.4
2 000×2 000	5 639.6	1 755.6	58.54	44.97	335.6	49.48	39.83	307.3
2 700×2 700	13 902.4	4 340.0	–	111.9	819.7	–	96.63	755.2
3 200×3 200	23 290.0	7 240.0	–	184.9	–	–	163.4	–

图 4 以图形的方式显示了两种显卡在 CUDA 实现中取得的 GB/s 和 GFLOPS 值。用 GTX280 显卡，对于大的网格，CUSP 达到了 61 GB/s 和 123 GFLOPS。理论最大值是：对于 GTX280 显卡，GB/s 和 GFLOPS 值分别为 141.7 GB/s、933.1 GFLOPS（单精度）、77.8 GFLOPS（双精度）；对于 GTX260 显卡，GB/s 和 GFLOPS 值分别为 111.9 GB/s、804.8 GFLOPS（单精度）、67.1 GFLOPS（双精度）。

比较了在单核和 CUDA 程序求得的数值解，计算出了全部网格（$t=1.0$ 时）在 CPU 和 GPU 中得出的解之间



(a)GB/s 值



(b)GFLOPS 值

图 4 在所有网格中两种显卡在 CUDA 实现中取得的 GB/s 和 GFLOPS 值

的不同 L1 范数。使用 CUSP 的 L1 范数的数量级在 10^{-2}~10^{-4} 之间变化，而使用 CUDP 所得到的数量级在 10^{-13}~10^{-14} 之间变化，这反映了在 GPU 上使用单双精度计算数值解的不同精度。

使用 CUDA 框架为一层浅水系统建立和实现了一个有效的一阶良好有限体积求解程序。为了在 CUDA 体系结构上有效地并行数值模式，这个求解程序实现了优化技术。在一个 GeForce GTX280 显卡上使用单精度执行的模拟达到了 61 GB/s 和 123 GFLOPS，比一个单核版本的求解程序快了 2 个数量级，也比一个基于图形语言的 GPU 版本速度快。这些模拟也显示了用求解程序所得到的数值解对于实际应用是足够精确的，用双精度比用单精度求得的解更精确。对于未来的工作，我们提出了在不规则网格上进行有效模拟的扩展策略，给出两层浅水系统的模拟。

参考文献

[1] CASTRO M J, GARCFA-RODRIGUEZ J A. A parallel 2D finite volume scheme for solving systems of balance laws with nonconservative products: application to shallow flows[J]. Comput Methods Appl Mech Eng, 2006, 195(19): 2788-2815.

[2] CASTRO M J, GARCFA-RODRIGUEZ J A. Solving shallow-water systems in 2D domains using finite volume methods and multimedia SSE instructions[J]. Comput Appl Math, 2008, 221(1): 16-32.

[3] RUMPF M, STRZODKA R. Graphics processor units: new prospects for parallel computing[J]. Lecture notes in computational science and engineering, 2006, 51(1): 89-132.

[4] HAGEN T R, HJELMERVIK J M, LIE K-A. Visual simulation of shallow-water waves[J]. Simul Model Pract Theory, 2005, 13(8): 716-726.

[5] LASTRA M, MANTAS J M, URENA C. Simulation of shallow-water systems using graphics processing units[J]. Math Comput Simul, 2009, 80(3): 598-618.

[6] SHREINER D, WOO M, NEIDER J. OpenGL programming guide: the official guide to learning OpenGL, Version2.1[M]. ddison-Wesley Professional, 2007.

[7] FERNANDO R, KILGARD M J. The Cg tutorial: the definitive guide to programmable real-time graphics[M]. Addison-Wesley, 2003.

[8] CHAPMAN B, JOST G, DAVID J. Using OpenMP: portable shared memory parallel programming[M]. The MIT Press, Cambridge 2007.

[9] NVIDIA. CUDA Zone[EB/OL]. [2009-09-10]. http://www.nvidia.com/object/cuda_home.html.

[10] Eigen2.0.9[EB/OL]. [2009-09-10]. http://eigen.tuxfamily.org.

(收稿日期：2011-12-22)

作者简介：

张哲，女，1965年生，副教授，硕士，主要研究方向：并行计算。