面向异构架构的混合精度有限元算法及其CUDA实现

刘建华 王朝尉 任江勇 田 荣
(中国科学院计算技术研究所高性能计算机研究中心 北京 100190)

摘 要 长期以来,单精度似乎与科学计算无缘,然而从体系结构看,混合精度计算可以充分发挥向量部件、GPGPU设备的单精度性能,提供更高的效能,如降低通讯带宽要求、提高数据传输和通讯效率等。混合精度显格式有限元算法,结合材料强非线性多尺度有限元程序msFEM,实现了GPGPU上的有效加速。实验结果表明:混合精度显格式有限元程序实现了90%以上的计算通过单精度完成,其计算结果与全部使用双精度的结果相一致。该算法可以使得在不支持双精度格式的加速卡上实现科学计算功能。在支持双精度浮点格式的GPU上,混合精度算法与全部采用双精度计算相比其加速效果提高了1.6~1.7倍。

关键词 GPGPU,混合精度算法,有限元,并行计算
中图法分类号 TP391.7 **文献标识码** A

Mixed Precision Finite Element Algorithm on Heterogeneous Architecture

LIU Jian-hua WANG Chao-wei REN Jiang-yong TIAN Rong
(High Performance Computer Research Center, Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100190, China)

Abstract For a long time, single precision has been giving away to double precision in scientific computing. However, on computer architectures, mixed-precision computing, can take full advantages of excellent computing compatibilities of vector components, GPGPU, offering merits such as reducing communication bandwidth requirements, improving data movement efficiency etc. A mixed-precision explicit finite-element algorithm was proposed and implemented on nVidia GPU for strongly nonlinear multi-scale material simulation. The developed mixed-precision finite-element method gives the same results as that of the fully double-precision calculation, while keeping a 90% portion of finite element calculations to be done by single precision float calculation. As a result, on the device that does not support native double precision float format, the mixed-precision algorithm makes it possible to fulfill double precision finite element simulation, while on the device that supports the native double precision, the mixed-precision algorithm is 1.6~1.7 times faster than the full double precision calculation.

Keywords GPGPU, Mixed precision algorithm, Finite element method, Parallel computing

1 引言

在现代体系结构上,32位浮点运算通常至少要比64位浮点运算快两倍。人们很早就发现在求解线性方程组时可以采用低精度计算获得计算加速,而只用少量的高精度计算达到与全部用高精度一致的计算结果。1948年威尔金森设计制造图灵计算机时提出求解线性方程组的迭代改进(iterative refinement)[1]。求解线性方程组 $Ax=b$ 的迭代改进思想可以描述为:

$x^{(0)}=0$

$d^{(s)}=b-Ax^{(s)}$ compute residual in *high* precision

$Ac^{(s)}=d^{(s)}$ solve equation system in *low* precision

$x^{(s+1)}=x^{(s)}+c^{(s)}$ accumulate solution in *high* precision

威尔金森和他的合作者们证明,如果矩阵 A 不是高度病态,连续对 x 用混合精度求解会收敛到全部用高精度计算的结果。对于线性方程组求解,计算90%以上的浮点可以通过低精度来完成,并且不会影响解的精度。

混合精度计算的主要思想是对计算密集部分采用低精度,对少量关键步骤采用高精度,从而达到计算加速。

混合精度计算的有效性在于必须保证最终的计算精度没有损失。Langou等[2]在Cell[3]及主流CPU上应用稠密矩阵求解法对混合精度算法的有效性进行了全面测试。Goddeke等[4,5]对大规模矩阵的多重网格求解法进行了测试。这些测试证实了混合精度计算结果与全部用高精度计算结果的精度完全一致。

本文第2节首先介绍显格式有限元算法和有限元算法增

到稿日期:2011-08-05 返修日期:2011-10-13 本文受国家自然科学基金(11072241)资助。
刘建华(1986-),男,硕士生,主要研究方向为并行计算;王朝尉(1984-),男,硕士,工程师,主要研究方向为数值模拟、并行计算;任江勇(1982-),男,硕士,工程师,主要研究方向为数值模拟、并行计算;田 荣(1973-),男,博士,研究员,主要研究方向为并行计算、数值模拟、大规模并行软件、计算力学。

量格式的混合精度算法；第 3 节介绍 msFEM 程序在 GPU 上的 cuda 实现和优化；第 4 节对 msFEM 有限元程序中针对混合精度 GPU 算法、双精度 GPU 算法和双精度 CPU 算法做了综合性能对比；最后总结全文。

2 显格式有限元及其混合精度计算

目前在混合精度计算方面，对于线性方程组的混合精度计算(迭代精化)的研究较多[6,7]。很多科学与工程问题可归结为牛顿第二定律的求解，如微观尺度上的分子动力学，宏观尺度上的显式粒子方法、显式有限元等。这类问题中，力向量的求解时间约占到整体计算的 90%。显式算法最大的特点是容易并行化，并且算法对宏观尺度上的碰撞等瞬态问题、多尺度计算中各种物理化学过程的微细观演化和由于椭圆形丧失等导致材料强非线性问题等都具有极强的适应能力[8]。下面以线弹性问题为例来说明混合精度显式有限元计算的基本思想。

2.1 显式有限元算法

显格式有限元方法的基础是牛顿第二定律：

$$Ma+f_{int}=f_{ext} \tag{1}$$

式中，a 为加速度矢量，f_{int} 为内力矢量，f_{ext} 为外力矢量，M 为质量矩阵。在动态显式有限元中，用集中法使质量矩阵成为对角矩阵，从而避免矩阵求逆运算。

对每个网格单元求解内力 f_{i_e}，组装得到内力 f_{int}。

$$f_{i_e}=\iiint_{\Omega_e} B^{\mathrm{T}}\sigma \mathrm{d}\Omega=\sum_{g=1}^{n_g} B^{\mathrm{T}}(x_g)\sigma(x_g)\omega_g \tag{2}$$

式中，n_g 为单元高斯点个数，ω_g 为数值积分权重。

利用几何方程和物理方程来求单元的应变 ε 和应力 σ。由位移得到单元应变 ε 为：

$$\begin{aligned}\varepsilon &= Lu=LNa^e=L[N_1 \quad N_2 \quad \cdots \quad N_m]a^e \\ &=[B_1 \quad B_2 \quad \cdots \quad B_m \quad]a^e=Ba^e\end{aligned} \tag{3}$$

式中，n_g 为单元高斯点个数，B 为应变矩阵，L 为三维空间问题微分算子，单元应力可以根据物理本构方程求得：

$$\sigma=D\varepsilon=DBa^e=Sa^e \tag{4}$$

式中，S 为应力矩阵，D 为弹性矩阵。

2.2 混合精度算法

2.2.1 全量格式

$$ma=f=f_{ext}-f_{int} \tag{5}$$

由加速度进而求节点的速度和位移：

$$v=v_0+a\Delta t \tag{6}$$

$$u=u_0+v\Delta t \tag{7}$$

式中，f_{ext} 为外部载荷，f_{int} 为内力。v 为节点速度，u 为节点位移。在有限元中求解每个网格单元节点上的内力 f_{i_e}，然后将其组装成内部力 f_{int}。

2.2.2 增量格式

$$m(a^k+\Delta a^{k+1})=f^k+\Delta f^{k+1} \tag{8}$$

式中，上标 k 表示上一时步，$k+1$ 表示当前时步。由于

$$ma^k=f^k \tag{9}$$

当前求解的牛顿第二定律变为增量格式：

$$\beta m\Delta a=\beta\Delta f \tag{10}$$

加速度更新格式为：

$$a^{k+1}=a^k+\Delta a \tag{11}$$

混合精度计算的基本思路是，通过单精度求解增量格式的牛顿第二定律，然后通过双精度完成加速度的累加以防止精度丢失。同时，为了防止小浮点数 Δa 和 Δf 的下溢，引入缩放参数 β(β 取为 f 的 1-范数/N 的倒数，N 为向量长度)[9]。

混合精度计算的具体算法如下所述，其中“(s)”表示变量及计算由单精度浮点完成，“(d)”表示变量及计算由双精度来完成[10]。

Step 1 $m\Delta a^{k+1}=\beta\Delta f^{k+1}$
(s) (s)

Step 2 $a^{k+1}=a^k+\Delta a^{k+1}/\beta$
(d) (d) (s)

Step3 $v^{k+1}=v^k+a^{k+1}\Delta t$
(d) (d) (d)

Step 4 $u^{k+1}=u^k+v^{k+1}\Delta t$
(d) (d) (d)

3 双精度和混合精度有限元算法在 GPGPU 上的实现与优化

在 GPU 上进行计算就必须在 CPU 和 GPU 间传送数据，在 CPU 程序中使用类对象＋动态数组结构的方式来组织数据，使用类对象是为了使数据组织更加具有意义，而使用数组是为了利用 GPU 的存储耦合特性加快数据的访存，从而提高计算效率。为了能够充分地在 CPU 上使通信和计算重叠，将网格点分为内部点和边界点两部分来分别计算，在 CPU 中计算内部点的同时进行边界点通信。将 GPU 所需数据通过中间结构体在 CPU 端进行收集，然后传输给 GPU。

如果不做数据的中间结构转化，而直接对数据做 CPU 与 GPU 间传输，就可以减去数据的中间转换时间。不过由于数据的不连续性，每个网格单元的应力、结点信息中的每项数据都要做一次传输，这显然是不可行的。

3.1 GPU 优化

3.1.1 异步并行执行

在程序中使用了 GPU 异步流，异步执行主要有两个方面的作用：首先，处于同一个流内的计算与数据拷贝是依次顺序进行的，但一个流内的计算可以和另一个流内的数据传输同时进行，因此异步能够使 GPU 中的执行单元与存储器控制单元同时工作，提高了资源利用率，加快了计算速度；其次，当 GPU 在进行计算或者数据传输时就返回主机线程，主机线程不必等待 GPU 运行完毕就可以继续进行计算(所需数据与 GPU 计算结果无数据关联)，使 CPU 和 GPU 同时工作[11]。

在我们的应用中，使用 GPU 异步并行，数据结构的主机内存需要采取分页锁定格式(pinned memory)。下面是我们的程序异步 GPU 计算的时间对比(见表 1)。其中，试验设备为 CPU：Intel(R) Xeon(R) CPU L5420 2.50GHZ(8 核)，GPU：NIVDIA Tesla C2050，GPU 驱动：cuda3.0 编译器：nvcc。

表 1　GPU 异步执行时间(单位:ms)

单元数	同步	异步流数			
		2	4	8	16
2541	3.80	3.36	3.36	3.37	3.91
12221	19.82	13.20	13.72	12.70	15.70
112211	199.21	127.58	134.78	123.09	127.31

从表 1 可以看到,使用异步并行明显减少了 GPU 计算时间。在异步执行流数为 8 时,我们的程序所花时间相对较少。当网格结点数为 12221 时,性能加速比为 19.82/12.70＝1.56;当网格结点数为 112211 时,性能加速比为 1.62。

3.1.2　寄存器使用优化

寄存器(register)是 GPU 片上的高速缓存,执行单元以极低的延迟访问寄存器(1 个时钟周期)。每个流式多处理器(SM)中寄存器的大小与设备相关,具体到我们的设备为 32768。寄存器文件数量总体虽然有 32k,但是平均分给并行执行的每个线程后,每个线程拥有的数量非常有限。例如,如果每个 Block 块有 64 个线程,那么每个线程可用的 32bit 寄存器数量为 32768/64＝256,即 256 个 float 数据或者 128 个 double 数据,如果 Block 块有 128 个线程,那么就只能存 128 个 float 数据或者 64 个 double 数据。所以寄存器的使用需要合理分配。

针对程序主要使用了 3 种控制优化方法:第一种方法是控制 Grid 和 Block 块大小,具体见表 2。从表 2 可以看到,当 Block 块的大小为 32 时,所需时间最少。经过分析认为因为使用的是 block 内共享寄存器,所以在线程为 32 时,每个线程中的变量能够较多地使用寄存器,从而减少了访存时间。

表 2　Block 块大小对程序影响(单位:ms)

网格数	Block 块			
	16	32	64	128
2541	4.94	3.36	3.47	3.76
12221	18.42	11.54	13.33	14.67
112211	181.18	123.25	137.29	143.32

第二种方法是尽量减少中间变量的使用,简单地说,就是用计算来减少存取时间,在一定程度上加大计算量来减少中间变量的使用,对一些中间变量不进行保存当需要时重新计算以减少寄存器的使用。这跟在 Cpu 上编程使用中间变量来减少计算时间是不同的,例如我们对 GPU 上的求解应变阵函数优化的所得结果(见表 3)。求解应变阵时间占 GPU 计算时间比例从 20%左右降到 3%。

表 3　GPU 上对求解应变阵优化的性能(单位:ms)

单元数	2541	12221	112211
优化前	2.59	8.78	84.64
优化后	2.50	8.35	80.73
加速比	1.04	1.05	1.05

第三种方法是使用共享存储器代替部分寄存器的使用。位于 GPU 片内的共享存储器是一块低延迟(1 个时钟周期延迟)、Block 块内线程共享的可读写存储器。Tasla C2050 卡上每个 SM 的共享存储器大小为 48KByte。使用共享存储器优化,主要有两个方面:一是使用共享存储器代替寄存器存储中间变量,二是访问频繁的数据从 global memory 先传输到 shared memory 上再使用。具体效果见表 4。

表 4　共享存储器优化后时间(单位:ms)

单元数	2541	12221	112211
优化前	3.59	16.88	169.84
优化后	3.37	13.17	127.43
加速比	1.07	1.28	1.33

3.1.3　常量存储器与纹理存储器的使用优化

常量存储器和纹理存储器在设备端为只读存储。相对于全局存储器它们都有各自的缓存,用以节约带宽,加快访问速度。纹理存储器相比常量存储器还有一些特殊的功能,如寻址模式、类型转换、滤波等。

相比之下,二者各有优缺点:首先,常量存储器存储空间较小(只有 64kB),但是可以存储各种类型的数据结构(包括自定义结构体),使用较为灵活,访问速度快;而纹理存储器可声明的内存比常量存储器要大,可以多次进行数据绑定纹理,但访问速度较慢。

在程序中我们使用常量储存器存储传输数据量较小的只读的物质点信息、高斯点信息。由于常量存储器用来存储的数据量所占比例很小,因此实际效果并不明显,使用带来的性能具体见表 5。而对于结点信息,由于其数据量很大,使用常量存储器明显不可能,因此可以使用纹理存储器,具体效果见表 6,可以看到有一定的效果。

表 5　常量存储器的使用效果(单位:ms)

网格单元数	使用前 GPU 时间	使用后 GPU 时间	性能对比(时间比值)
2541	3.34	3.36	0.9940
12221	13.22	13.17	1.0038
112211	128.13	126.76	1.0108

表 6　纹理存储器使用效果对比(单位:ms)

网格单元数	使用前 GPU 时间	使用后 GPU 时间	性能对比(时间比值)
2541	2.51	2.36	1.064
12221	8.68	7.45	1.165
112211	81.70	70.95	1.152

CPU 向 GPU 传输的数据有:高斯点、物质点、结点信息、应力。其中高斯点和物质点所占比例很小(当网格单元为 2541 时,二者所占比例为 0.1%),通过表 5 可以看出,常量存储器加速效果不明显。而结点信息占总体比例为 64%,纹理存储器加速效果较好。

4　测试与结果分析

4.1　GPU 双精度性能测试

对有限元 GPU 程序与原 CPU 程序做各个部分性能对比,包括:网格结点单元内力 f 求解时间、内力 f(GPU 程序包含数据打包)求解时间、合力 F(内力＋组装)求解时间,总时间(不含输出:中间结果不进行输出)、总时间(含输出)5 部分,如表 7 所列。

表 7　CPU 双精度与 GPU 双精度算法性能

时间	CPU 双精度	GPU 双精度	CPU 双/GPU 双
求内力	17.17s	0.128s	134.1
组装力	17.68s	0.892s	19.8
总时间	18.61s	1.305s	14.3

其中,在输出部分包含纯输出和后处理两部分。在我们的测试中,对输出的后处理部分使用 openmp 共享存储并行。

从表 7 可以看出，GPU 双精度算法相比 CPU 双精度算法局部加速比为 134，整体加速比为 14.3。

4.2 GPU 混合精度性能测试与分析

对程序在 GPU 上使用双精度和混合精度算法做性能测试，得到表 8 中数据。由表 8 可以看到，当局部 GPU 并行时，混合精度与双精度加速比比值为 1.6～1.8，整体效果为 1.3 左右。

针对程序进行结构优化，对混合精度 GPU 有限元算法整体性能及精度进行测试。表 8 是我们测得的性能实验结果。

在数值精度方面，我们测得的结果是有 12 位有效数值，精确度在 10 到 11 位，满足精度需要，与上述精度结果一致。

表 8 GPU 双精度混合精度算法时间

时间	GPU 双精度	GPU 混合精度	GPU 双/GPU 混合
求内力	128	76.7	1.67
求合力	892	662	1.35
总时间	1305	971	1.34

由表 8 可得：与 CPU 双精度相比，在网格结点为 112211 时，GPU 双精度算法的总体性能加速比（不含输出）为 14～15，GPU 混合精度算法的总体性能加速比（不含输出）为 19～20。而单就计算内力 F（不含组装）部分，亦即采用 GPU 计算对应部分（包括 GPU 计算，GPU 传输），在网格结点数为 112211 时，GPU 双精度加速比为 134 倍，GPU 混合精度加速比高达 224 倍。

结束语 本文提出了一种混合精度显格式有限元算法，用于材料强非线性多尺度模拟。混合精度显格式有限元程序实现了 90%以上的有限元计算通过单精度完成，其计算结果与全部使用双精度一致。

参 考 文 献

[1] Strzodka R, Göddeke D. Mixed precision methods for convergent iterative schemes[C]// Proceedings of the 2006 Workshop on Edge Computing Using New Commodity Architectures. May 2006: 59-60

[2] Langou Ju-lie, Langou Ju-lien, Luszczek P, et al. Exploiting the performance of 32 bit floating point arithmetic in obtaining 64 bit accuracy (revisiting iterative refinement for linear systems) [C]// Proceedings of the 2006 ACM/IEEE conference on Supercomputing. 2006

[3] Kurzak J, Dongarra J J. Implementation of mixed precision in solving systems of linear equations on the CELL processor[M]. Concurrency Computat. Pract. Exper. to appear

[4] Göddeke D, Wobker H, Strzodka R, et al. Co-processor acceleration of an unmodified parallel solid mechanics code with FEASTGPU[M]. Accepted for publication in the International Journal of Computational Science and Engineering, 2008

[5] Göddeke D, Strzodka R. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations (part 2: Double precision GPUs)[R]. Technical University Dortmund, 2008

[6] Göddeke D, Strzodka R, Turek S. Performance and accuracy of hardware-oriented native-, emulated- and mixed-precision solvers in FEM simulations[J]. International Journal of Parallel, Emer-gent and Distributed Systems, Special Issue: Applied Parallel Computing, 2007, 22(4): 221-256

[7] Strzodka R, Göddeke D. Pipelined mixed precision algorithms on FPGAs for fast and accurate PDE solvers from low precision components[C]// Proceedings of the 14th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'06). 2006: 259-270

[8] Li X S, Demmel J W, Bailey D H, et al. Design, implementation and testing of extended and mixed precision BLAS[J]. ACM Transactions on Mathematical Software (TOMS), 2002, 28(2)

[9] Cecka C, Lew A J, Darve E. Assembly of Finite Element Methods on Graphics Processors[M]. Int. J. Numer. Meth. Engng., 2000: 1-6

[10] Kurzak J, Dongarra J. Implementation of mixed precision in solving systems of linear equations on the Cell processor[J]. Concurrency and Computation: Practice and Experience, 2007, 19(10): 1371-1385

[11] Taiji M, Narumi T, Ohno Y, et al. Protein Explorer: A Petaflops Special-Purpose Computer System for Molecular Dynamics Simulations[C]// Proc. Supercomputing. 2003

(上接第 260 页)

[5] Yoo H W, Ryoo H J, Jang D S. Gradual shot detection using localized edge blocks [J]. Multimedia Tools and Application, 2006, 28(3): 283-300

[6] 曹建荣，蔡安妮. 压缩域中基于支持向量机的镜头边界检查算法[J]. 电子学报，2008，36(1)：203-208

[7] Zhang N, Xiao G Q, Jiang J M, et al. BPNN Algorithm towards Shot Boundary Detection [J]. Journal of Computer Application, 2009, 29(5): 1369-1372

[8] Zhang H J, Kan kan halli A, Smoliar S W. Automatic partitioning of full motion video [J]. Multimedia System, 1993, 1(1): 10-28

[9] Wang J Y, Luo W. A Self-adapt in g Dual-threshold Method for Video Shot Transition Detection[C]// IEEE International Conference on Networking, Sensing and Control. 2008, 4: 704-707

[10] 张玉珍，杨明，王建宇，等. 基于运动补偿和自适应双阈值的镜头分割[J]. 计算机科学，2010，37(9)：282-286

[11] 薛亮，于敏，张正炳. 一种改进的运动计算法——新三步搜索法[J]. 电子技术，2004，4：22-27

[12] Zheng J, Zou F M, Shi M. An Efficient Algorithm for Video Shot Boundary Detection[C]// IEEE Proceeding of 2004 International Symposium on Intelligent Multimedia, Video and Speech Processing. Hong Kong, 2004: 266-269