CPU/GPU 集群上求解偏微分方程的可扩展混合算法

罗 力[1] 杨 超[2] 赵宇波[1] 蔡小川[3,1]

[1]（中国科学院深圳先进技术研究院 深圳 518055）

[2]（中国科学院软件研究所 北京 100080）

[3]（美国科罗拉罗大学博尔德分校 博尔德 CO 80309）

摘 要 当前世界上排前几位的超级计算机都基于大量CPU和GPU组合的混合架构，它们对某些特殊问题，譬如基于FFT的图像处理或N体颗粒计算等领域可获得很高的性能。但是对由有限差分（或基于网格的有限元）离散的偏微分方程问题，于CPU/GPU 集群上获得较好的性能仍然是一种挑战。本文提出并测试一种基于这类集群架构的混合算法。算法的可扩展性通过区域分解算法实现，而GPU的性能由基于光滑聚集的代数多重网格法获得，避免了在GPU上表现不理想的不完全分解算法。本文的数值实验采用32 CPU/GPU求解用差分离散后达三千万未知数的偏微分方程。

关键词 PDEs；CPU/GPU集群；区域分解；代数多重网格；可扩展算法

A Scalable Hybrid Algorithm for Solving Partial Differential Equations on a Cluster of CPU/GPU

LUO Li YANG Chao ZHAO Yu-bo CAI Xiao-chuan

[1] (*Shenzhen Institutes of Advanced Technology, The Chinese Academy of Sciences, Shenzhen 518055*)

[2] (*Institute of Software, The Chinese Academy of Sciences, Beijing 100080*)

[3](*Department of Computer Science, University of Colorado at Boulder, Boulder CO 80309, USA*)

Abstract Several of the top ranked supercomputers are based on the hybrid architecture consisting of a large number of CPUs and GPUs. High performance has been obtained for problems with special structures, such as FFT-based imaging processing or N-body based particle calculations. However, for the class of problems described by partial differential equations (PDEs) discretized by finite difference (or other mesh based methods such as finite element) methods, obtaining even reasonably good performance on a CPU/GPU cluster is still a challenge. In this paper, we propose and test an hybrid algorithm which matches the architecture of the cluster. The scalability of the approach is implemented by a domain decomposition method, and the GPU performance is realized by using a smoothed aggregation based algebraic multigrid method. Incomplete factorization, which performs beautifully on CPU but poorly on GPU, is completely avoided in the approach. Numerical experiments are carried out by using up to 32 CPU/GPUs for solving PDE problems discretized by FDM with up to 32 millions unknowns.

Keywords PDEs; CPU/GPU cluster; domain decomposition; algebraic multigrid; scalable algorithm

1 引 言

很多工程与科学计算问题可以归纳为求解基于有限元和有限差分方法离散的偏微分方程问题。在纯CPU上已有相当多成熟和通用的算法及高性能软件包，比如可扩展并行计算包PETSc[1]。

当前世界上前几位的超级计算机已逐渐转向为由大量CPU与GPU组成。在某些领域比如快速Fourier变换[2]快速多极算法[3]等,相比于混合架构，纯CPU计算

罗力，研究助理，主要研究方向是计算流体力学和并行算法。E-mail: li.luo@siat.ac.cn。**杨超**，副研究员，研究方向为并行计算。**赵宇波**，正高级工程师，研究方向为计算力学。**蔡小川**，教授，研究方向为科学与工程的并行算法。

已得到不错的加速比。尽管在稠密线性代数方面使用GPU已得到成效[4]，然而绝大多数的基于稀疏矩阵的并行求解算法在GPU上表现不佳。这是由于问题的非结构和非规则特性，特别是预条件迭代算法中大量使用的不完全分解算法在GPU上没有良好的表现。目前在GPU上进行稀疏矩阵计算的方面已有一定探索，比如新版本的PETSc已可支持GPU加速[5]。Rocha等将Jacobi预条件共轭梯度法应用于求解心电生理学中的稀疏线性问题[6]。[7]中探讨了更高效的GPU预条件技术如代数多重网格方法等。

为了避免在预条件中使用不完全分解算法，我们提出并测试一种基于区域分解和代数多重网格的混合算法。假设有相同个数的CPU核和GPU卡，首先将整体求解区域划分为多个重叠的子区域，每个子区域映射至一对CPU核和GPU卡的组合。我们将子区域内的预条件子计算任务交给GPU卡完成，而将所有其它计算任务交给CPU核完成。为了发挥GPU的优势，采用基于光滑聚集的多重网格法进一步将子区域网格衍生构造出多层更小的粗网格层次，每层的计算由GPU加速效果明显的操作完成。

本文余下内容组织如下：第二部分介绍基于加性Schwarz预条件技术和子区域多重网格法的混合算法。第三部分给出本文方法在NVIDIA Tesla S1070集群上的数值结果。第四部分对全文作总结。

2　混合算法

在很多应用领域，对偏微分方程在某计算区域上进行有限元和有限差分方法离散后得到一个稀疏线性方程组

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

我们考虑是对称正定的情形，比如自伴椭圆问题。目前在CPU集群上已有一些软件包提供了高效的并行算法[1][8][9]，但在CPU/GPU集群上的高效算法仍然欠缺。我们考虑预条件迭代法求解如下方程组

$$M^{-1}Ax=M^{-1}b \tag{2}$$

其中预条件子M^{-1}是A^{-1}的近似。求解该方程组的性能相当大程度依赖于M^{-1}的定义和实现，因此我们提出将所有与M^{-1}有关的计算分配至GPU，而将所有其它计算分配至CPU完成，目的使该混合算法在CPU/GPU集群上所需计算时间最少。

对本文的模型问题，采用一种加性Schwarz共轭梯度法(CG)[8]进行求解，如算法1所示。

算法 1

Preconditioned CG for $Ax=b$

1. $r_0=b-Ax_0, z_0=M^{-1}r_0, p_0=z_0$
2. do j = 0,1,…until convergence
3. $\alpha_j=(r_j,z_j)/(Ap_j,p_j)$
4. $x_{j+1}=x_j+\alpha_j p_j$
5. $r_{j+1}=r_j-\alpha_j Ap_j$
6. $z_{j+1}=M^{-1}r_{j+1}$
7. $\beta_j=(r_{j+1},z_{j+1})/(r_j,z_j)$
8. $p_{j+1}=z_{j+1}+\beta_j p_j$
9. end do

其中x_0是给定初始值，M^{-1}是Schwarz预条件子。

记CPU/GPU为一对CPU核和GPU卡的组合，np为这些组合的数目。首先将计算区域Ω划分至np个非重叠子区域，重叠的子区域可由非重叠的子区域向外拓展δ个网格层次得到，记为Ω_k, $k=1,2,...,np$，由一个MPI进程分配至一对CPU/GPU。加性Schwarz预条件子[10]的计算过程如算法2所示，其中R_k和R_k^T分别是限制算子和插值算子。

算法 2

Additive Schwarz: $z\leftarrow AS(r)$

1. For k = 1,2,..., np
   Restriction : $r_k=R_k r$
   Solve the subdomain problem $z_k=B_k^{-1}r_k$
   End for
2. Interpolation : $z\leftarrow\sum_{k=1}^{np}R_k^T r_k$

加性Schwarz预条件子是一种典型的区域分解算法，多个子区域问题同时进行求解，并且每个子区域问题可看成一个黑箱操作。首先将子区域矩阵$C=B_k$和右端向量$d=r_k$从CPU内存复制至GPU全局存储器，各GPU获取数据后开始求解，然后将结果向量$x=z_k$从GPU复制回CPU，这个过程需要同步以保证全局向量已完全组装完毕。一个MPI进程内CPU内存与GPU全局存储器之间的数据通过PCI-Express通道传输，多个MPI进程间通信通过InfiniBand高速交换机互联。图1显示本文的区域分解算法在一个二维矩形区域上的实现。

我们采用基于NVIDIA CUDA实现的光滑聚集多重网格法(smoothed aggregation:SA-AMG)[11]求解子区域问题$C_x=d$。记拓展算子为，可定义相应于粗网格上的矩阵为$C_c=P^TCP$，考虑迭代法

$x\leftarrow x\text{-}Py$

其中y由求解以下粗网格问题得到

$C_c y=P^T(Cx-d)$,

令$n=n_1$为C的维数，并记细网格层上线性方程组$C_x=d$

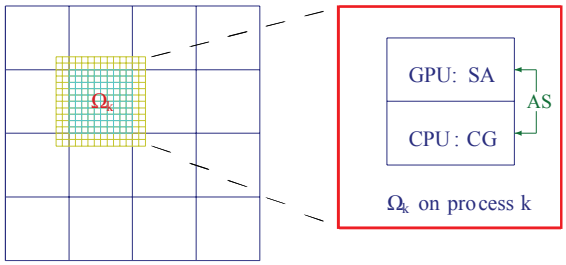

图1 CPU/GPU集群上区域分解算法。左端：二维矩形区域上有重叠的子区域划分。右端：每个子区域由一个MPI进程管理一对CPU/GPU

为$C_l x=d_l$。 引入一系列粗网格层

$$C_{l+1} = (I_{l+1}^l)^T C_l I_{l+1}^l,$$

其中拓展算子 I_{l+1}^l 由一个给定的光滑子 S_l 和一个张量拓展子 P_{l+1}^l 乘积得到。

$$I_{l+1}^l = S_l P_{l+1}^l, \quad 1,\ldots,L-1.$$

光滑子的一种选择是

$$S_l = I - \frac{4}{3\lambda_l} C_l,$$

其中 λ_l 是 C_l 的谱半径上界。

对每一层上的C_l所涉及的多个自由度集根据连接强度进行分组，每组选择一个自由度集作为粗网格层，从而得到张量拓展子 P_{l+1}^l，具体过程可见[11]。

SA-AMG算法过程如算法3所示。

算法 3

Smoothed Aggregation: $x_l = AMG_l(x_l, d_l)$

0. If on the coarsest level, then:
   Solve $C_l x_l = d_l$ by direct LU, else:
1. Apply μ steps of smoothing to $C_l x = d_l$ of the form
   $x \leftarrow (T_l C_l)x + T_l d_l$
   where T_l is an approximate inverse of C_l
2. Coarse grid correction:
   (a). Set $d_{l+1} = (I_{l+1}^l)^T (d_l - C x_l)$ and $x_{l+1} = 0$
   (b). Solve the coarse problem $B_{l+1} x_{l+1} = d_{l+1}$
   by γ applications of $x_{l+1} = AMG_{l+1}(x_{l+1}, d_{l+1})$
   (c). Then correct the solution on the level l
   by $x_l \leftarrow x_l + I_{l+1}^l x_{l+1}$
3. Apply μ steps of smoothing to $C_l x = d_l$ of the form
   $x \leftarrow (T_l C_l)x + T_l d_l$

SA-AMG算法是一种代数多重网格算法，无需显式知道问题的几何网格信息，通过线性方程组本身即可构造所需的粗网格层次，尤其适合于子区域问题的黑箱求解。由于SA-AMG只作为本文预条件子的一部分，因此并不需要精确求解，在本文的实现中，我们只对每层作数次(μ)光滑处理(smoothing)，对SA-AMG作数次(γ)循环操作。根据数值结果，多次的光滑处理或多次循环操作可使子区域求解更为精确，从而使总的迭代数减少，然而可能导致计算时间有所增加。对每一层上的光滑处理我们采用Jacobi迭代法或Chebyshev多项式近似光滑，前者的并行特性非常适合GPU的高密度计算模式，后者的实现依赖于重要的计算核心稀疏矩阵向量乘运算(SPMV)。

SA-AMG算法的GPU实现在性能上主要依赖于三种运算： Blas-axpy(2.(a),2.(c)), SPMV(1.,3.), 及稠密三角求解 (0.)。在编程上将CUDA模型的线程块分配至矩阵和向量的多个行，合理选择线程块的大小和采用合适的存储方式是提高SA-AMG算法性能的关键。比如SPMV，上千个线程同时执行矩阵的多个行中的非零元加乘运算，并对同一行中的分量规约操作利用CUDA提供的shared memory进行加速。对稠密三角求解我们选择MAGMA包的LU求解器[12]。

3 数值实验

本文的数值结果于NVIDIA S1070 GPU集群“深腾7000”上的8个结点计算得到。每个结点由2个4核2.26 GHz Intel Xeon E5520 CPU处理器和4个1.3 GHz NVIDIA Tesla C1060 GPU卡组成。结点间由20 Gb InfiniBand DDR网络互联。GPU编程模型使用NVIDIA CUDA Toolkit 3.2，并通过参数-arch_sm 13确保双精度。CPU代码使用O3水平优化。

记“Iter.”为CG法总迭代次数，“TSolve”为总计算时间，“TData”为CPU与GPU之间数据复制时间，“Eff.”为并行效率。

本文研究Dirichlet边界条件下Poisson方程于区域$\Omega=[0, 1]^2$上的数值解。在一致NxN矩形网格上对微分方程进行五点差分离散，所得的稀疏方程组是对称正定的。迭代求解终止条件为相对误差小于10^{-6}。

3.1 优化参数分析

首先讨论本文算法的参数并选取最优值，测试Jacobi或Chebyshev多项式两种光滑算子。表1显示总迭代次数与总计算时间受Jacobi法迭代次数(sweeps)或Chebyshev多项式阶数(degree)的影响. MPI进程数固定为32，网格规模为8193×4097。SA-AMG的循环次数为1，加性Schwarz预条件子的重叠数(overlap)设为1。可以观察到总迭代次数随着Jacobi法迭代次数或Chebyshev多项式阶数的增加而减少，然而，从总计算时间上考虑，1次Jacobi法迭代即为最优。

多次SA-AMG的循环操作可使子区域求解更为精确，以减少CG法的总迭代次数。表2显示总迭代次数如何受循环次数的影响。所有的循环都在GPU上完成，以避免CPU与GPU间多次复制数据。光滑算子采用

表1 Jacobi光滑算子和Chebyshev多项式光滑算子的参数选优，网格8193X4097，*np*=32，时间为秒

Jacobi smoother			Polynomial smoother		
Sweeps	Iter	TSolve	Degree	Iter	TSolve
1	249	26.687	-	-	-
2	242	28.175	2	306	36.719
3	241	29.579	3	316	39.557
4	240	31.631	4	304	39.489
5	236	33.267	5	285	39.517
6	230	34.093	6	283	40.399

1次Jacobi迭代，加性Schwarz预条件子的重叠数设为1。从表中可见，总迭代次数随循环次数增多明显地减少，而总计算时间当循环次数为3时最少。

表2 SA-AMG的循环次数对总迭代次数的影响，网格193×4097，*np*=32，时间为秒

Cycles	Iter	TSolve
1	249	26.687
2	209	26.277
3	188	26.919
4	182	29.497
5	178	31.586

接下来选取最优加性Schwarz预条件子的重叠数。一般情况下大的重叠数可以减少总迭代次数，但将导致通信时间增多，同时子区域问题规模增大，使得总计算时间不一定减少。表3显示重叠数大小的影响，当中光滑算子采用1次Jacobi迭代，SA-AMG循环次数为3。从表中可以看出，总迭代次数确实随重叠数的增大而减少，但总计算时间却表现为先增大后减少。

表3 加性Schwarz预条件子的重叠数对总迭代次数的影响，网格8193×4097，*np*=32，时间为秒

Overlap	Iter	TSolve
0	185	25.85
1	188	26.92
2	191	27.71
3	177	26.53
4	161	24.46
5	153	23.78

根据以上实验结果，本文算法的参数最优选择为：Jacobi法迭代次数Sweeps=1，SA-AMG循环次数Cycles=3，加性Schwarz预条件子的重叠数Overlap=1，我们余下数值实验均采用此最优参数选择。

3.2 加速比分析

表4给出本文算法应用于CPU/GPU混合架构相对于纯CPU架构的加速比分析。由于混合算法是对局部计算进行GPU加速，为衡量加速效果，引入以下几个定义。为加性Schwarz预条件子区域问题CPU计算时间占总CPU计算时间的比率，反映本文算法中可用GPU加速的潜力。为理论加速比，为实际加速比，定义为纯CPU计算时间与CPU/GPU混合计算时间之比。MPI进程数固定为16。从表中可看出，随着网格规模的增大，加速比不断增加，达到理论加速比的73%。

表4 CPU/GPU混合算法加速比分析，*np=16*，时间为秒

Mesh	*np*	P_{acc}	S_{max}	S_{tol}
513×513	16	76.22%	4.206	2.489
1025×1025	16	87.94%	8.292	4.101
2049×2049	16	90.38%	10.40	4.475
4097×4097	16	91.20%	11.36	8.361

3.3 网格可扩展性分析

我们测试本文算法的网格可扩展性，即固定MPI进程数，并不断增加网格规模。作为比较，引入纯CPU集群表现最优的不完全分解Cholesky算法(ICC)作为加性Schwarz预条件子区域求解器，其填充级别为3。数值结果如表5所示，本文混合算法的总计算时间自网格规模起则少于加性Schwarz-ICC(3)算法。两种算法的总迭代数均随着网格规模的增大而增加，但本文混合算法的迭代数增加速度明显慢于加性Schwarz-ICC(3)算法，体现较好的网格可扩展性。另外根据表中“TData”的数据，可见CPU与GPU间的数据复制时间相对于总计算时间非常小。

表5　网格可扩展性分析，*np*=16，时间为秒

		Local solver				
		ICC(3)(CPU)		SA-AMG(CPU/GPU)		
Mesh	*np*	Iter	TSolve	Iter	TData	TSove
513×513	16	150	0.290	59	0.011	0.615
1025×1025	16	260	1.961	82	0.050	1.454
2049×2049	16	467	13.755	104	0.199	6.826
4097×4097	16	814	94.918	132	0.940	18.890

3.4 弱可扩展性分析

弱可扩展性分析从4个MPI进程，网格规模开始，增加进程数目的同时增大网格规模，使得每个处理器计算规模不变。在不考虑通信开销的理想情况下，总计算时间应该保持不变。从表6看出，本文混合算法从总迭代次数，总计算时间和并行效率上要比加性Schwarz-ICC(3)算法要好，然而两种算法的弱可扩展性都不够理想。

表6 弱可扩展性分析，时间为秒

		Local solver					
		ICC(3)(CPU)			SA-AMG(CPU/GPU)		
Mesh	*np*	Iter	TSolve	Eff.	Iter	TSolve	Eff.
2049×2049	4	497	56.33	-	98	12.09	-
4097×2049	8	645	72.84	77%	159	18.40	66%
4097×4097	16	814	94.92	59%	132	18.89	64%
8193×4097	32	1214	144.33	39%	188	26.92	45%

4 总 结

本文提出并测试一种基于CPU/GPU集群求解偏微分方程的可扩展混合算法。算法的可扩展性通过区域分解算法实现，而GPU的性能由基于光滑聚集的代数多重网格法获得，避免了在GPU上表现不理想的不完全分解算法。我们将预条件子的相关计算分配至GPU，而将其它所有计算分配至CPU。数值实验表明本文混合算法从迭代次数、计算时间和可扩展性上要比现有的纯CPU算法表现要好。今后将推广至更大规模的CPU/GPU集群及更复杂的偏微分方程模型。

参 考 文 献

[1] Balay S, Buschelman K, Eijkhout V, et al. PETSc User's Manual,Tech. Rep. ANL-95/11-Revision 3.1. Argonne National Laboratory, 2010. http://www.mcs.anl.gov/petsc/petsc-as/snapshots/petsc-current/docs/manual.pdf.

[2] Chen Y, Cui X, Mei H. Large-scale FFT on GPU clusters [C]//In ICS '10: Proceedings of the 24th ACM International Conference on Supercomputing. New York(USA), 2010: 315-324.

[3] Hamada T, Narumi T, Yokota R, et al. 42 TFlops hierarchical n-body simulations on GPUs with applications in both astrophysics and turbulence [C] //In SC'09: Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis. New York(USA), 2009: 62:1-62:12.

[4] Tomov S, Dongarra J, Baboulin M. Towards dense linear algebra for hybrid GPU accelerated manycore systems [J]. Parallel Comput, 2010, 36:232-240.

[5] Minden V, Smith B, Knepley M G. Preliminary implementation of PETSc using GPUs [C] //In Proceedings of the 2010 International Workshop of GPU Solutions to Multiscale Problems in Science and Engineering. 2010.

[6] Rocha B M, Campos F O, Amorim R M, et al. Accelerating cardiac excitation spread simulations using graphics processing units [J]. Concurr. Comput. Pract. Exper, 2011, 23:708-720.

[7] Douglas C C, Lee H, Haase G, et al. Parallel algebraic multigrid method with GP-GPU hardware acceleration [J]. J. Comput. Appl. Math. 2011, Submitted.

[8] Saad Y. Iterative methods for sparse linear systems [M]. PWS, 1996.

[9] Smith B, Bjørstad P, Gropp W. Domain decomposition: parallel multilevel methods for elliptic partial differential equations [M]. New York: Cambridge University Press, 1996.

[10] Cai X-C, Sarkis M. A restricted additive schwarz preconditioner for general sparse linear systems [R]. Tech. Report CU-CS-843-97, Department of Computer Science, University of Colorado at Boulder, 1997.

[11] Brezina M, Falgout R, MacLachlan S, et al. Adaptive smoothed aggregation (α-SA) multigrid [J]. SIAM Rev., 2005, 47:317-346.

[12] MAGMA. Matrix algebra on GPU and multicore architectures [EB/OL]. http://icl.cs.utk.edu/magma/