一种基于实测的自动负载建模算法

刘 旭 莫则尧 安恒斌 曹小林 张爱清

(北京应用物理与计算数学研究所 北京 100094)

摘 要 负载平衡是影响大规模并行计算效率的一个关键因素,准确的负载建模是负载平衡的基础。提出了一种基于实测的自动负载建模算法。该算法无需用户提供信息,具有良好的理论保证以及近似线性的计算复杂度和完全的并行性。2400个进程上的分子动力学模拟表明,该算法执行速度快,同时能够保证60%以上的负载平衡效率。

关键词 并行计算,负载平衡,负载建模,粒子模拟,JASMIN框架

中图法分类号 TP301 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2015.1.014

Automatic Load Modeling Algorithm Based on Real Time Measuring

LIU Xu MO Ze-yao AN Heng-bin CAO Xiao-lin ZHANG Ai-qing

(Institute of Applied Physics and Computational Mathematics, Beijing 100094, China)

Abstract Load imbalance can cause significant performance degradation in large-scale simulations. Accurate load modeling is fundamental for load balancing. An automatic load modeling algorithm based on real time measuring was presented. It eases the burden of the user from providing a load model. The algorithm possesses several good properties mathematically, and runs totally distributed with a sublinear computational complexity. Experiment result of molecular dynamics simulation on 2400 processes demonstrates its speed and efficiency.

Keywords Parallel computing, Load balancing, Load modeling, Particle simulation, JASMIN

1 引言

大规模并行计算面临的一个难题是如何保持或提高并行性能。例如,对于某些应用程序,在数百个处理器上,并行效率可以达到80%以上,但是在数千个处理器上可能只有50%,在数万个处理器上甚至更低。影响并行性能的一个关键因素就是负载不平衡。而且,通常情况下,处理器数越多,负载不平衡的影响越大。

负载平衡是并行计算的一个重要研究内容。在基于网格的科学计算中,负载平衡主要包括3个部分:

(1)负载建模。预估每个网格单元上的计算量。

(2)负载剖分。根据网格单元上预估的计算量,将网格均分成一系列子网格。

(3)负载迁移。将网格单元从原来所属的进程迁移到新的进程上。

其中,准确的负载建模是基础。本文针对负载建模开展研究。

负载建模方面的研究较少。在实际应用中,主要有3类负载建模算法:

(1)用户根据计算过程的特点,设置负载模型(例如,在分子动力学模拟中,根据分子个数设置负载模型)。

(2)根据模拟中实测的计算时间,预测未来时间步的计算量(例如,文献[1,2]中,用过去相邻的几个时间步统计的时间外推未来的计算量)。

(3)结合(1)和(2)的混合型方法。使用该类方法的宇宙学模拟应用曾在SC12会议上获得了代表并行应用最高水平的Gordon Bell奖[3]。

然而,在实际应用中,已有的负载建模算法还不能够满足需求。(1)、(3)类的建模算法一方面会增加用户负担,另一方面还可能过于粗糙。对于(2)类建模算法,由于每个网格单元上的计算量很小,因此一般不能统计出网格单元上的计算时间,仅能统计出每个进程的计算时间。(2)类已有的建模算法缺乏理论保证,而且对于一些模拟,模型不稳定,增加了不必要的数据迁移。但是,(2)类的建模算法可以自动建模,无需用户提供信息,因此有利于减轻用户负担。

由于(2)类的负载建模算法可以减轻用户负担,因此本文提出了一种属于该类的基于实测的自动负载建模算法。该算法具有良好的理论保证,同时具有 $O(N_{C_p}\log N_{C_p})$ 的计算复杂度。其中 N_{C_p} 表示平均单个进程上的网格单元数目。

2 问题分析

为了分析简单清晰,下文只考虑基于结构网格的数值模

到稿日期:2013-12-26 返修日期:2014-03-15 本文受国家自然科学基金(61033009,61003083,11171039),国家重点基础研究发展计划(2011CB309702)资助。

刘 旭(1981-),男,博士,副研究员,主要研究领域为并行算法研究、并行应用软件研制,E-mail:liu_xu@iapcm.ac.cn;莫则尧(1971-),男,博士,研究员,主要研究领域为并行计算;安恒斌(1974-),男,博士,研究员,主要研究领域为并行计算;曹小林(1974-),男,博士,研究员,主要研究领域为并行计算;张爱清(1976-),女,博士,副研究员,主要研究领域为并行计算。

拟应用，并且在整个模拟过程中网格保持不变。模拟中，计算逐网格单元进行，一个区域上的计算量等于每个网格单元计算量之和。

随着模拟时间步进，每个网格单元的计算量可能不变，也可能变化（例如粒子模拟，粒子在网格单元间迁移会造成网格单元计算量的变化）。

假定，计算环境完全同构，计算时间与计算量成正比。

首先，将计算量的变化分成两种：

(1)计算量渐变，即在相邻两个时间步中，计算量不变或变化较小；

(2)计算量突变，即在相邻两个时间步中，计算量发生较大变化（特别地，模拟的初始时刻也归为此类）。

对于计算量突变的情况，只有用户有能力预测，因此需要用户负责建模。

下面针对计算量渐变的情况进行分析，设计算法。

根据实际应用的需求，需要处理两类负载建模问题：

A. 用户不提供负载模型，算法根据实测的计算时间预测负载；

B. 用户提供粗糙的负载模型，算法根据实测的计算时间修正负载模型。

对于问题 A，此时，仅有实测的计算时间。我们希望负载模型的计算量与实测的计算时间吻合，即

$$\begin{cases} \sum\limits_{c\in C_1} l_c = t_1 \\ \vdots \\ \sum\limits_{c\in C_{N_p}} l_c = t_{N_p} \end{cases}$$

其中，l_c 表示网格单元 c 上的计算量，C_p 表示进程 p 上的网格单元集合，N_p 表示进程总数，t_p 表示进程 p 的计算时间。下标从 1 开始。

对于问题 B，此时有用户提供的负载模型和实测的计算时间。我们希望负载模型的计算量与实测的计算时间吻合，且接近用户提供的负载模型，即

$$\begin{cases} \sum\limits_{c\in C_1} l_c = t_1 \\ \vdots \\ \sum\limits_{c\in C_{N_p}} l_c = t_{N_p} \\ l_c \approx l_c^u, c\in C_p \end{cases}$$

其中，l_c^u 表示用户负载模型网格单元 c 上的计算量。

3 算法设计

对于问题 A，其解不唯一。因此利用计算量渐变这个性质，再加上一个条件——“负载模型最接近于上一时刻的负载模型”，即 $\min \| L-L^o \|$。其中 L 是由 l_c 组成的向量，L^o 是由上一时刻 l_c 组成的向量（两者相同分量所对应的网格单元必须一致），$\| \cdot \|$ 表示 $p(p\geqslant 1)$ 范数。

这样就得到了一个优化问题：

$$\begin{cases} \min \| L-L^o \| \\ \sum\limits_{c\in C_1} l_c = t_1 \\ \vdots \\ \sum\limits_{c\in C_{N_p}} l_c = t_{N_p} \\ l_c \geqslant 0, c\in C_p \end{cases}$$

下面求解该优化问题。

首先，上述优化问题等价于如下优化问题：

$$\begin{cases} \begin{cases} \min \| L_1 - L_1^o \| \\ \sum\limits_{c\in C_1} l_c = t_1 \\ l_c \geqslant 0, c\in C_1 \end{cases} \\ \vdots \\ \begin{cases} \min \| L_{N_p} - L_{N_p}^o \| \\ \sum\limits_{c\in C_{N_p}} l_c = t_{N_p} \\ l_c \geqslant 0, c\in C_{N_p} \end{cases} \end{cases}$$

其中，L_{C_p} 代表进程 p 上 l_c 组成的向量。

这样，原先全局的优化问题可以通过求解一系列单个进程上的子优化问题解出。

可以证明，子优化问题 $\begin{cases} \min \| L_{C_p} - L_{C_p}^o \| \\ \sum\limits_{c\in C_p} l_c = t_p \\ l_c \geqslant 0, c\in C_p \end{cases}$ 的解，可以通过如下算法得到。

算法 1 $L_{C_p} = \text{alg } 1(t_p, L_{C_p}^o)$

```
(1)N_Cp = |C_p|;
   Δt = t_p − Σ_{c∈C_p} l_c;
   L^s_Cp = sort(L^o_Cp, ascending);
(2)if(Δt + N_Cp × l^s_1 ≥ 0){
     Δl = Δt / N_Cp;
     L_Cp = L^o_Cp + Δl;
     return L_Cp;
   }
(3)for(c=1; c≤N_Cp; c++){
     if(Δt + (N_Cp − c + 1) × l^s_c ≥ 0){
       l_thresh = l^s_{c−1};
       Δl = Δt / (N_Cp − c + 1);
       break;
     }
     else{
       Δt = Δt + l^s_c;
     }
   }
(4)for(c=1; c≤N_Cp; c ++ ){
     if(l^o_c ≤ l_thresh){
       l_c = 0;
     }
     else{
       l_c = l^o_c + Δl;
     }
   }
   return L_Cp;
```

从计算量的角度看，算法 1 中(1)sort 的计算量是 $O(N_{C_p}\log N_{C_p})$，其他部分都为 $O(N_{C_p})$，因此整体为 $O(N_{C_p}\log N_{C_p})$。由于每个子问题所需的数据都在相应的进程上，无需数据通信，因此每个子问题可以完全独立地并行求解。

进一步地，对于一些模拟，大量网格单元上的计算量变化很慢。此时，可以在每个进程上增加判断“if($|t_p-\sum_{c\in C_p} l_c^o|<\alpha\frac{\sum_{p=1}^{N_p} t_p}{N_p}$)”，其中一般设置阈值$\alpha\in(0,0.1)$。如果判断为真，则不修正该进程上的子负载模型。这样处理后，可以解决粒子模拟中粒子仅分布在局部区域的模拟问题；否则，在无粒子区域更新模型既无必要，又有较大开销。

对于问题B，可以用类似的方法处理。加上一个条件，负载模型最接近于用户提供的负载模型，即$\min\|L-L^u\|$。其中L^u是由用户提供的l_c组成的向量。

这样就得到了一个优化问题：

$$\begin{cases}\min\|L-L^u\|\\ \sum_{c\in C_1} l_c=t_1\\ \vdots\\ \sum_{c\in C_{N_p}} l_c=t_{N_p}\\ l_c\geqslant 0,c\in C_{N_p}\end{cases}$$

同样可以用算法1求解该优化问题（用户模型与实测时间的单位可能不统一，需要先做单位转换）。

4 数值试验

为了验证本文算法的有效性，进行了并行测试。

负载平衡效率（LBE）[4]定义为：$\frac{\text{mean}\{t_p\}}{\max\{t_p\}}$。

根据负载模型预测的负载平衡效率（LBE_m）定义为：$\frac{\text{mean}\{\sum_{c\in C_p} l_c\}}{\max\{\sum_{c\in C_p} l_c\}}$。

针对分子动力学模拟进行了试验。分子动力学模拟在计算物理学、计算化学和计算生物学等领域有着广泛应用[5]。在分子动力学模拟中，粒子分布不均匀，或者粒子间相互作用计算不均匀，可能引起负载不平衡[6]。

MD[7]是一个基于JASMIN框架[8,9]实现的分子动力学程序，它使用单层结构网格，模拟粒子短距离作用。程序采用了区域分解的并行方式，首先进行负载建模，然后根据文献[10]中的算法将计算区域划分成若干网格片，接着使用Morton空间填充曲线将网格片排成一维序列，最后使用文献[11]中的负载平衡算法将网格片序列分配到各处理器上进行计算。

我们选取二维两个矩形物体高速碰撞模拟作为测试模型。图1反映了物理时刻1时该模型的粒子分布。在该时刻，两个物体（密度为灰色的部分）发生小范围碰撞，碰撞区域密度明显高于其他部分（密度为白色的部分）。根据计算过程的特征可知，在碰撞区域，单元中的计算量最大，物体未碰撞部分单元中的计算量次之，其他无粒子区域计算量接近0。测试从模拟推进到物理时刻1开始考察（因为在模拟时刻0附近，粒子分布较均匀，计算量可以用粒子个数较为准确地刻画，所以从物理时刻1开始考察。测试通过使用重启动数据，从物理时刻1开始）。程序使用动态负载平衡，每隔5个时间步进行一次负载建模和负载调整。模型有1.3×10^8个网格单元和5.5×10^8个粒子。

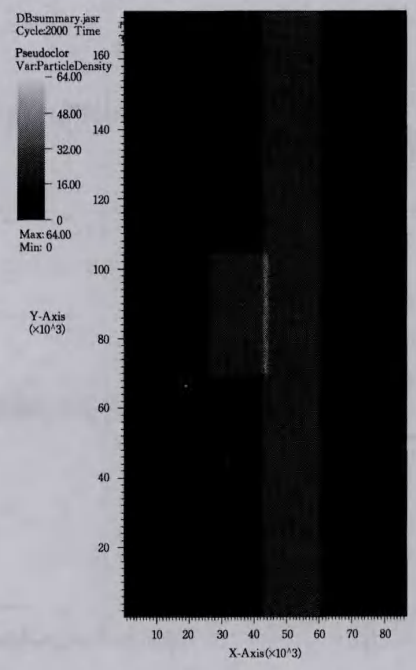


图1 物理时刻1的粒子密度分布

算法代码使用C/C++实现，纯MPI方式并行。编译使用icc，加上O2优化选项。测试在TianHe-1A上进行，使用400个结点，每个结点启用6个进程，共2400个进程。

测试比较了6种建模方法：

(1)时间平均。将一个进程统计的计算时间平均到该进程计算的各个网格单元上，作为该网格单元的计算量。

(2)加权滑动平均法[1]。

(3)粒子个数。将一个网格单元上的粒子数目作为该网格单元的计算量。

(4)混合法[3]。

(5)A1。使用算法1求解优化问题A。

(6)B1。使用算法1求解优化问题B。使用(3)作为用户的负载模型。

测试1 时间步长设为很小值1×10^{-13}，模拟1000个时间步，比较6种算法的负载平衡效率。

测试目的：检验在负载非均匀分布且基本保持不变的情况下，负载建模算法能否收敛到准确的负载分布以及收敛的速度如何。

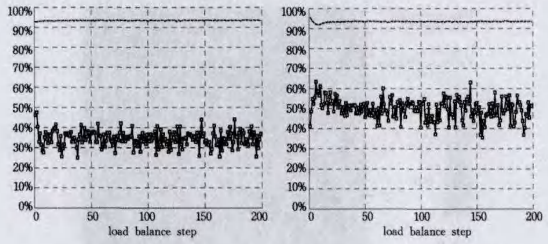
图2表现了使用6种方法的负载平衡效率随负载调整进行的变化。

首先，在所有的测试中，LBE_m明显高于LBE，说明负载平衡算法可以较好地剖分各种负载模型，测试不会因为负载平衡算法受到影响。

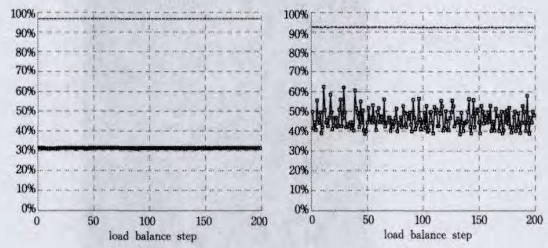
其次，时间平均法、加权滑动平均法、混合法LBE波动较大，B1法波动较小。A1法收敛，收敛到约88.5%。粒子个数法基本保持不变（由于时间步长很小，粒子基本保持不动，因此该模型基本不变）。

最后，时间平均法整体$LBE=34.2\%$，加权滑动平均法整体$LBE=49.4\%$，粒子个数法整体$LBE=31.4\%$，混合法整体$LBE=45.7\%$，A1法整体$LBE=84.1\%$，B1法整体$LBE=74.2\%$。时间平均法、粒子个数法负载较不平衡，加权

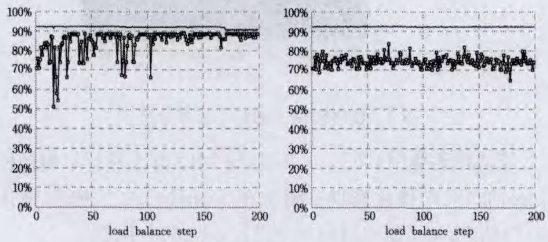
滑动平均法、混合法中等，本文提出的 A1、B1 法负载较平衡。



(a)时间平均法　(b)加权滑动平均法



(c)粒子个数法　(d)混合法



(e)A1 法　(f)B1 法

点折线，LBE；虚线，LBE_m

图 2　极小步长试验中各种算法得到的负载平衡效率

图 3 表示了通过 A1 法计算得到的负载模型中的负载分布。对照图 1 可以看出，粒子密度越高的部分，模型负载也越重，但是两者并不是简单的线性关系(图 3 中白色部分对应的负载值与灰色部分对应的负载值之比，明显大于图 1 中白色部分对应的负载值与灰色部分对应的负载值之比)。

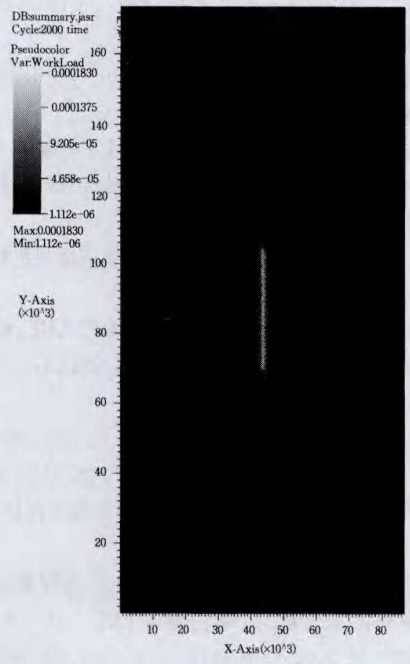


图 3　A1 法，经过 200 次负载调整后，负载模型中的负载分布

测试 2　时间步长设为正常值 1×10^{-3}，模拟 1000 个时间步，比较 6 种算法的负载平衡效率。

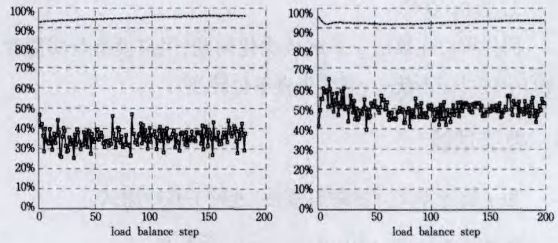
测试目的：检验在负载非均匀分布、慢速动态变化情况下，使用各种负载建模算法能否进行有效的负载平衡。

图 4 表现了使用 6 种方法的负载平衡效率随负载调整进行的变化。其中时间平均法由于测试超时，因此没有收集到 1000 步的数据，但是已有的数据足够说明趋势。

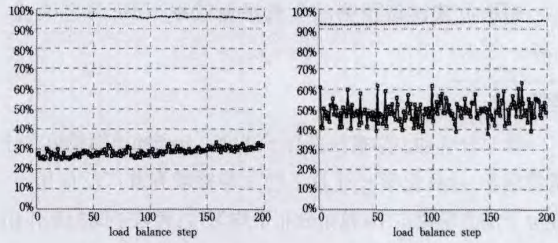
首先，在所有的测试中，LBE_m 远高于 LBE，说明负载平衡算法可以较好地剖分各种负载模型，测试不会因为负载平衡算法受到影响。

其次，时间平均法、加权滑动平均法、混合法、A1 法、B1 法 LBE 波动较大，粒子个数法波动较小。

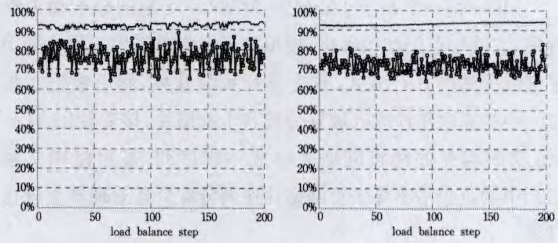
最后，时间平均法整体 $LBE=35.1\%$，加权滑动平均法整体 $LBE=49.6\%$，粒子个数法整体 $LBE=28.6\%$，混合法整体 $LBE=48.5\%$，A1 法整体 $LBE=76.0\%$，B1 法整体 $LBE=72.3\%$。时间平均法、粒子个数法负载较不平衡，加权滑动平均法、混合法中等，本文提出的 A1、B1 法负载较平衡。



(a)时间平均法　(b)加权滑动平均法



(c)粒子个数法　(d)混合法



(e)A1 法　(f)B1 法

点折线，LBE；虚线，LBE_m

图 4　正常步长试验中各种算法得到的负载平衡效率

在所有测试中，所有算法平均每次的建模开销均小于 0.1 秒，执行速度较快，能够满足一般应用的需求。

结束语　针对结构网格并行应用提出了一类基于实测计时的自动负载建模方法。这类方法具有良好的理论保证，同时具有 $O(N_{C_p}\log N_{C_p})$ 的计算复杂度，并且易于实现。在 2400 个进程的分子动力学模拟测试中，这类方法执行快速，同时能够保证 60%以上的负载平衡效率。

(下转第 78 页)

未调优至最佳性能。第二，EPCC 测试集中的算例的算法强度分布均匀，再结合所有算例已涵盖了“十三个小矮人”中的5个，这些都表明该测试集具备优良的测试性。其次，通过对比同一个算例在 Intel Knights Corner 和 NIVIDIA Kepler 架构下的性能，构建了针对这两个平台的相对性能模型。本实验中，仅一个算例在 Knights Corner 上的性能达到了 Kepler 上的约 70%，其余所有的算例性能在 Knights Corner 上的均未达到 Kepler 上的 40%。这也就表明了未经优化的 OpenACC 代码在 Intel Knights Corner 和 NVIDIA Kepler 架构下的性能可移植性不佳。

参 考 文 献

[1] Kurkure N, Das A, Valmiki M, et al. Evaluation of Rodinia Codes on Intel Xeon Phi[C]// 4th International Conference on International Conference on Intelligent Systems, Modelling and Simulation, 2013. Bangkok: IEEE, 2013: 415-419

[2] Aoki T. Application Performances on Many-core Processors Xeon Phi versus Kepler GPU[OL]. 2013-12[2014-3]. http://www.ocw.titech.ac.jp/index.php?module=General&action=DownLoad&file=20131226717065-477-1-45.pdf&type=cal&JWC=20131226717065

[3] OpenMP Architecture Review Board. OpenMP Application Program Interface[OL]. 2013-7[2014-4]. http://www.openmp.org/mp-documents/spec30.pdf

[4] CAPS entreprise. OpenACC Reference Manual CAPSCompilers 3.3[OL]. 2012-12[2014-4]. http://www.caps-entreprise.com/products/caps-compilers/

[5] Khronos OpenCL Working Group. The OpenCL Specification [OL]. 2008-12[2014-4]. https://www.khronos.org/registry/cl/specs/opencl-1.0.29.pdf

[6] OpenACC Group. The OpenACC Application Programming Interface_v1.0[OL]. 2011-11[2014-4]. http://www.openacc.org/sites/default/files/OpenACC.1.0_0.pdf

[7] David A. Patterson John L. Hennessy and et al. Computer Architecture: A Quantitative Approach(第5版)[M]. 北京：机械工业出版社，2012：285-288

[8] Johnson N. EPCC OpenACC benchmark suite [OL]. 2013-5 [2014-4]. https://www.epcc.ed.ac.uk/research/computing/performance-characterisation-and-benchmarking/epcc-openacc-benchmark-suite

[9] Kaltofen E L. The "Seven Dwarfs" of Symbolic Computation [C]// Numerical and Symbolic Scientific Computing, 2012. Wien: Springer Vienna, 2012: 95-104

[10] McCalpin J D. Stream: Sustainable memory bandwidth in high performance computers[OL]. 2013-2[2014-4]. http://www.cs.virginia.edu/stream/ref.html

[11] md rezaur rahman. The scalable heterogeneous computing benchmark suite (shoc) for intel xeon phi[OL]. 2013-4[2014-4]. https://software.intel.com/en-us/blogs/2013/03/20/the-scalable-heterogeneous-computing-benchmark-suite-shoc-for-intelr-xeon-phitm

[12] NVIDIA. CUDA C Programming Guide [OL]. 2014-2(5.5) [2014-4]. http://docs.nvidia.com/cuda/cuda-c-programming-guide/#axzz342yBEw4Q

[13] Lin H, Scogland T, Zhang J, et al. OpenCL and the 13 Dwarfs: A Work in Progress[C]// ICPE'12 Proceedings of the 3rd ACM/SPEC International Conference on Performance Engineering, 2012. New York: ACM, 2012: 291-294

[14] Hoshinom T, Maruyama N, Takaki R. CUDA vs OpenACC: Performance Case Studies with Kernel Benchmarks and a Memory-Bound CFD Application[C]// 13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), 2013. Delft: IEEE, 2013: 136-143

[15] Yang You, Fu Hao-huan, Huang Xiao-meng, et al. Accelerating the 3D Elastic Wave Forward Modeling on GPU and MIC[C]// the 27th International Symposium on Parallel and Distributed Processing Workshops and PhD Forum. Washington, 2013: 1088-1096

(上接第 66 页)

这类方法可以推广到不是基于结构网格的其他并行应用。希望将来在各种应用中都能使用这类方法。

参 考 文 献

[1] Luitjens J, Berzins M. Improving the performance of Uintah: A large-scale adaptive meshing computational framework[C]// Proceedings of the 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS10). 2010: 1-10

[2] Sheng Di, Kondo D, Cirne W. Host Load Prediction in a Google Compute Cloud with a Bayesian Model[C]// International Conference for High Performance Computing, Networking, Storage and Analysis (SC12). 2012

[3] Ishiyama T, Nitadori K, Makino J. 4.45 Pflops Astrophysical N-Body Simulation on K Computer -The Gravitational Trillion-Body Problem[C]// International Conference for High Performance Computing, Networking, Storage and Analysis (SC12). 2012

[4] 张林波，迟学斌，莫则尧，等. 并行计算导论[M]. 北京：清华大学出版社，2006

[5] Frenkel D, Berend S. Understanding Molecular Simulation[M]. Academic Press Inc., 2001

[6] Hess B, Kutzner C, Spoel D V D, et al. GROMACS 4: Algorithms for highly efficient, load balanced, and scalable molecular simulation[J]. Journal of Chemical Theory and Computation, 2008, 4(3): 435-447

[7] 曹小林，刘旭. 基于 JASMIN 框架的粒子模拟并行计算[J]. 科研信息化技术与应用，2010，1(2)：28-33

[8] 莫则尧，张爱清. 并行自适应结构网格应用支撑软件框架 JASMIN 用户指南[R]. T09-JMJL-01. 北京应用物理与计算数学研究所，2009

[9] Mo Z Y, Zhang A Q, Cao X L, et al. JASMIN: a parallel software infrastructure for scientific computing[J]. Front. Comput. Sci. China, 2010, 4(4): 480-488

[10] 刘旭，张爱清. 一种空间矩形剖分的负载平衡算法[C]// 高性能计算年会(HPCC12). 2012

[11] 刘旭，莫则尧，曹小林. 基于内存约束的一维负载平衡方法及其应用[J]. 计算物理，2009，26(2)：184-190