第 36 卷 第 9 期 计 算 机 工 程 2010 年 5 月
Vol.36 No.9 Computer Engineering May 2010
·软件技术与数据库· 文章编号：1000—3428(2010)09—0073—04 文献标识码：A 中图分类号：TP301

基于多区域多代码问题的自动分块算法

张 娟，陆林生
(江南计算技术研究所，无锡 214083)

摘 要：针对现有分块算法并行度低、负载不平衡的缺陷，提出一种基于多区域多代码问题的自动分块算法。通过循环分配算法实现计算区域间的处理器分配，基于 Block 的递归二分法对无向图进行剖分，实现计算区域内的任务分配。实验结果表明，该算法可使整个计算空间分配到的处理器量大致相等，处理器间的通信量最小。

关键词：多区域多代码；并行计算；自动划分；负载平衡

Automatic Partition Algorithm Based on Multi-region and Multi-code Problem

ZHANG Juan, LU Lin-sheng
(Jiangnan Institute of Computing Technology, Wuxi 214083)

【Abstract】 Aiming at the shortage of low parallelism degree and load imbalance in existing partition algorithm, this paper presents an automatic partition algorithm based on multi-region and multi-code problem. This algorithm uses loop distribution algorithm to realize processor distribution between computational domain, uses recursion dichotomy based on Block for splitting undirected graph to realize task distribution in computational domain. Experimental results show that this algorithm can make allocated number of processor equation all of the computational space, and the communication traffic between processors is least.

【Abstract】 multi-region and multi-code; parallel computing; automatic partition; load balance

1 概述

计算流体力学(Computational Fluid Dynamics, CFD)数值模拟的实际应用课题大多为复杂的多区域多代码大规模问题。所谓多区域是指 CFD 数值模拟中，整个计算空间被划分为多个计算区域，各区域拥有不同控制方程模型、物理化学模型、计算格式等。各区域之间的这些不同点使程序在执行时选择不同的控制参数，程序执行路径发生改变，在各区域中执行不同代码，因此，属于多代码问题。各区域实际执行的代码不一样会引起不同区域之间在单个网格单元上浮点计算量的差异，这增加了并行计算中分块算法研究的难度。

在 CFD 数值模拟中，为了简化核心程序开发过程，限定某一处理器处理的网格单元具有相同的控制方程模型、物理化学模型、计算格式等，因此，在研究自动分块算法时要保证某一处理器处理同一计算区域中网格单元。

在采用结构化网格时，计算区域可能由多个区块构成，称这些区块为物理块(下文若不另加说明，物理块均作此解释)。并行计算分块算法研究面临 2 个问题：(1)整个计算空间的物理块数多，甚至达到几百上千个；(2)物理块与物理块之间网格单元数差异大，某些物理块的网格单元数甚至小于并行计算中处理器分配到的平均网格单元数。自动分块算法需实现以下 2 点：(1)允许一个处理器处理多个物理块；(2)允许一个物理块由多个处理器处理。

由于 CFD 数值模拟的实际应用要进行大规模并行计算，一般会采用几千个，处理器并行处理同一问题，甚至有可能达到几万个。随着处理器规模加大，相应的网格单元分配到处理器的时间增加，并行度降低，因此，有必要研究高效的自动分块算法实现通信优化。

目前，国际上有许多分块软件，如 MB-Split；也有多种分块算法，如 Scatter Decomposition[1], Orthogonal Recursive Bisection(ORB)[2], BoundedNeighbours[3]等。上述软件和算法都没有考虑多区域多代码的问题。国内关于任务划分和处理器分配的研究成果集中在人工划分或计算区域较简单的自动划分。

2 自动分块

假设整个计算空间的计算区域集 $R=\{r_i \mid i=1,2,\cdots,N_R\}$，计算区域 r_i 的网格单元数为 $Ncell_i$，计算区域 r_i 中网格单元的权值为 W_i (权值表示该网格单元的浮点计算量)，处理器集合为 $P=\{p_i \mid i=1,2,\cdots,N_P\}$，$f_i$ 表示分配到处理器 p_i 的浮点计算量，处理器与处理器间通信量集合 $C=\{<p_i,p_j> \mid i\neq j, \forall 1\leqslant i,j\leqslant N_P\}$，处理器 p_i 与 p_j 间负载差异量为 $L(i,j)=\left|f_i-f_j\right|, \forall 1\leqslant i,j\leqslant N_P$。自动分块的问题描述为：寻求一个合适的分块算法，使

$$\max_{\forall 1\leqslant i,j\leqslant N_P}(L(i,j)) \tag{1}$$

为极小且满足：

$$\sum_{i=1,2,\cdots,N_P} f_i=\sum_{i=1,2,\cdots,N_R}(Ncell_i\times W_i)/N_P \tag{2}$$

并尽量使

作者简介：张 娟(1979－)，女，博士研究生，主研方向：并行算法及应用；陆林生，教授、博士生导师

收稿日期：2009-11-13 **E-mail**：zjtzcn@163.com

$$\sum_{i=1,2,\cdots,N_p}\ \sum_{\substack{j=1,2,\cdots,N_p\\ j\neq i}} < p_i, p_j > \qquad (3)$$

达到最小。式(1)描述的是任意 2 个处理器之间计算量差异的最大值，自动分块算法的目的是使该最大值达到极小，并且满足式(2)即所有处理器的计算量总和等于整个计算空间总的计算量，同时在满足式(1)的情况下，尽量使式(3)即总的通信开销达到最小。

本文算法是个任务分配问题，在数学上已证明，分配问题是个 NP 完全问题，因此，只能采用启发式解法求近似解。由于多区域多代码问题的特点以及核心程序开发的实际需要，本文算法规定某一处理器只能处理来自同一计算区域中的网格单元，因此在算法设计时首先将处理器分配到各计算区域，然后将计算区域包含的网格单元映射到该计算区域拥有的处理器。

3 算法设计

本文算法可分为 2 步：(1)计算区域间循环分配算法；(2)计算区域内基于 Block 的递归二分法。

3.1 区域间循环分配算法计算

3.1.1 问题描述

计算区域间处理器数分配问题，将 N_P ($N_P \geqslant N_R$)个处理器分配到 N_R 个计算区域，使计算负载在各处理器间分配比较均衡。这是个典型的分配问题，属于 NP 完全问题，采用近似解法。其问题的数学描述是：寻找在 N_R 个计算区域上对处理器数的一个分配 $p(i), set.i = 1,2,\cdots,N_R$ ，满足：

$$\sum_{i=1,2,\cdots,N_R} p(i) = N_P$$

并使 $\max\limits_{\forall 1\leqslant i,j\leqslant N_R}(\left|L_R(i,p(i)) - L_R(j,p(j))\right|)$ 为极小，其中，$L_R(i,p(i)) = Ncell_i \times W_i / p(i)$ 。

3.1.2 循环分配算法

本文采用循环分配算法解决计算区域间处理器数的分配问题。在循环分配算法开始前，根据计算区域的计算量由小到大对计算区域排序，确保待分配的处理器数大于未分配到处理器的计算区域数。循环分配算法的基本思想是以尚未分配到处理器的计算区域和待分配的处理器数为对象，计算应分配到当前待处理计算区域的处理器数。

循环分配算法具体步骤如下：

输入 计算区域数 N_R，网格单元权值 W_i,set.i=1,2,⋯,N_R，网格单元数 $Ncell_i$,set.i=1,2,⋯,N_R ，处理器数 N_P ，根据计算区域计算量由小到大对计算区域进行排序，将排序后的计算区域号存放于数组 {$Array(i)$,set.i=1,2,⋯,N_R} 中

输出 数组 {$Array(i)$,set.i=1,2,⋯,N_R} 对应的各区域分配到的处理器数位 $p(i)$,set.i=1,2,⋯,N_R

1 DO i=1,N_R

2 j=Array(i)

3 $p_0 = \dfrac{W_j \times Ncell_j}{\sum\limits_{k=1,N_R}(W_{Array(k)} \times Ncell_{Array(k)})/(N_P - \sum\limits_{k=1,i-1} p(k))}$

4 IF (p_0==AINT(p_0)×1.0) THEN

5 p(i)=p_0

6 ELSE

7 $p_1 = \lfloor p_0 \rfloor$

8 $p_2 = \lceil p_0 \rceil$

9 ENDIF

10 $AveGrid_1 = \dfrac{W_j \times Ncell_j}{p_1}$； $AveGrid_2 = \dfrac{W_j \times Ncell_j}{p_2}$

11 $AveR_1 = \dfrac{\sum\limits_{k=i+1,N_R} W_{Array(k)} \times Ncell_{Array(k)}}{N_R - (\sum\limits_{k=1,i-1} p(k) + p_1)}$

12 $AveR_2 = \dfrac{\sum\limits_{k=i+1,N_R} W_{Array(k)} \times Ncell_{Array(k)}}{N_R - (\sum\limits_{k=1,i-1} p(k) + p_2)}$

13 IF (ABS(AveGrid1-AveR1)> ABS(AveGrid1-AveR1)) THEN

14 p(i)=p_2

15 ELSE

16 p(i)=p_1

17 ENDIF

18 ENDDO

19 输出 p(i),set.i=1,2,⋯,NR，根据 Array(i),set.i=1,2,⋯,NR，得到每个计算区域的处理器数

在目前实际应用中，CFD 数值模拟多区域问题的计算区域数较少，在计算区域间使用循环分配算法可行。

3.2 基于Block的递归二分法

每个物理块各方向的网格点数、边界面的边界特性(物理边界、通信边界)和邻居特性都是已知的，分配给该计算区域的处理器数也是已知的。在同一计算区域中，各网格单元浮点计算量相同。计算区域内任务分配需要把网格单元映射到属于该计算区域的处理器上，使每个处理器的网格单元数相等，处理器之间需要交换的信息量最少。

3.2.1 数学描述

将问题限定为计算区域 r_i 内包含的物理数 $m>1$，处理器数 $p(i)>1$。

数学描述为：寻找对 m 个物理块 $\{p_b(i)\,|\,1\leqslant i\leqslant m\}$ 的最佳划分得到：$\{s_b(i,j)\,|\,1\leqslant i\leqslant m, 1\leqslant j\leqslant n(i)\}$ 和 $\{s_b(i,j)\,|\,1\leqslant i\leqslant m, 1\leqslant j\leqslant n(i)\}$ 在 $p(i)$ 个处理器上的最佳分配，其中，$n(i)$ 表示划分的 i 个物理块 $p_b(i)$ 得到更小的块数目，称这个更小的块为子块；$s(i)$ 为分配到第 i 个处理器 p_i 的小块集合满足：

$$\{s(k)\,|\,s(k)\subset\{s_b\}; s(k)\bigcap_{k\neq j} s(j)=\phi; \bigcup_{k=1,p(i)} s(k)=\{s_b\}; 1\leqslant j,k\leqslant p(i);\}$$

寻求最佳划分和分配实现：

$$\begin{cases}\min\{\max\limits_k(abs(N_{cal,k}-N_{ave})/N_{ave})\}\\ \min\{\max\limits_k(N_{comm,k}/N_{cal,k})\}\end{cases}$$

其中，$\forall k = 1,2,\cdots,p(i)$；第 k 个处理器代表计算区域 r_i 拥有的任一个处理器；$N_{cal,k}$ 是第 k 个处理器包含的计算网格单元数，即 $\{s_b\,|\,s_b\in s(k)\}$ 中包含的计算网格单元数总和；$N_{ave} = \dfrac{Ncell_i}{p(i)}$ 是计算区域内 r_i 处理器平均计算网格单元数；$N_{comm,k}$ 是第 k 个处理器需要通信的网格单元数。

3.2.2 任务分配与无向图剖分

计算区域内任务分配是本文算法的重点。当参与计算的处理器体系结构相同时，可将计算区域内任务分配问题等同于对应的无向图剖分，剖分得到的各子域的顶点权之和相等。由于本文需解决的问题采用结构网格，整个计算区域由物理块构成，因此可构造无向图用于描述物理块之间的关系。每个物理块对应图中的一个顶点，当 2 个物理块之间有通信边界面时，相应的顶点之间存在边。具体描述如下：

用无向图 $G=(V,E,\rho,\sigma)$ 来描述待划分区域内物理块及物理块间的关系，其中，顶点集 $V=\{v_1,v_2,\cdots,v_n\}$ 表示物理块；边 $E=\{e_1,e_2,\cdots,e_n\}\subseteq V\times V$ 表示物理块与物理块间的通信关系；顶点权值 $\rho: V\to \mathrm{R}$ ；边的权值 $\sigma: E\to \mathrm{R}$ ，R 为实数集。与顶点 v_i 关联的边数叫做 v_i 的度数，记为 $d(v_i)$，如果图 G 的

2 个顶点 v_i，v_j 之间存在一条道路，则称 v_i 和 v_j 是连通的。若图 G 的任意 2 个顶点连通，则称图 G 是连通的[4]。

本文采用基于 Block 的递归二分法剖分无向图。在介绍基于 Block 的递归二分法之前，说明以下 2 个问题：(1)无向图的顶点不仅局限于物理块，还包括在划分过程中经物理块一次或多次分裂得到的小块，相关关系依次类推；(2)无向图的顶点对应于一个子块，顶点可分裂。

3.2.3　基于 Block 的递归二分法的实现

基于 Block 的递归二分法不同于一般递归二分法，它剖分的无向图顶点对应由连续网格单元构成的子块。在某次二分过程中，首先利用无向图的搜索策略将图分为 2 个子域，在无法实现负载平衡的情况下，再利用分裂子块方法实现。根据计算区域间处理器的分配结果，当前待划分计算区域 r_i 拥有的处理器数为 $p(i)$，采用基于 Block 的递归二分法将计算区域 r_i 对应的图剖分为 $p(i)$ 个子域，这需要 $p(i)-1$ 次二分[5]。如果对分过程中出现将子图剖分为 M 个子域，则 M 不是 2 的倍数，得到的 2 个中间子域的顶点权之和比值为 $\frac{\lfloor M/2 \rfloor}{M-\lfloor M/2 \rfloor}$。

采用基于 Block 的递归二分法对图进行剖分，递归初始条件为子图被划分成的子域数，即计算区域分配到的处理器数，同时要解决以下 3 个问题：(1)停止递归调用条件；(2)二分无向图时采用的搜索策略；(3)子块分裂方法。分裂子块改变了无向图顶点数、顶点权值、边数和边权值，因此，采用基于 Block 的递归二分法对图进行剖分过程中处理的是动态图，在程序开发中大大增加了数据结构设计的难度。图 1 是由 9 个物理块构成的某计算区域简单示意图。图 2 是图 1 中计算区域对应的无向图 G。

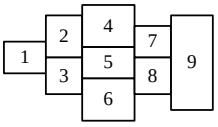

图 1　计算区域简单示意图

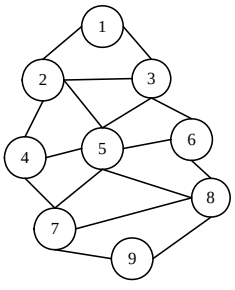

图 2　图 1 中计算区域对应的无向图 G

本文称下一个将被二分的子图为当前待处理子图。停止递归调用的条件有 2 个：

(1)当前待处理子图仅需被划分成 1 个子域；

(2)当前待处理子图仅含有一个顶点。

首先判断是否符合条件(1)，若符合则停止递归调用；否则继续判断是否符合条件(2)，若符合需在停止递归调用前做以下处理：当前待处理子图仅含有一个顶点，被划分成 M_1 个子区域，即将该顶点分裂成 M_1 个顶点。可见，子图顶点对应一个由连续网格单元组成的子块，采用基于 Block 的递归二分法对顶点进行分裂，初始条件为 M_1，终止递归调用条件为当前待处理子图需被划分成 1 个子域。对顶点进行某次二分时要注意分裂方向(沿某一计算平面分裂顶点得到 2 个新顶点，本文称这个计算平面为分裂面，分裂面的方向为分裂方向)选取。为避免出现某计算方向网格单元数较小，影响计算精确度，分裂方向一般选取网格单元数最多的方向。

在二分过程中，对无向图进行搜索，确定顶点所属子域。本文无向图搜索由某顶点(称该点为入口顶点)开始，采用广度优先策略进行搜索。

无向图搜索策略具体描述如下：

(1)入口顶点的选取。选择入口顶点的标准为(优先级由高到低)：1)无向图中度数最小的顶点；2)权值最小的顶点。为了避免二分连通图后得到 2 个非连通子图，采用标准 1)。为了避免分裂权值小的顶点影响计算精确度，采用标准 2)。

在实际工程应用中，可能出现无向图非连通，因此，在数据结构设计时要允许一个图有多个入口顶点分别对应可能出现的非连通图中的多个连通区域。

(2)通过广度优先搜索将无向图分成 2 个子域。其目的是由入口顶点开始，通过广度优先策略遍历无向图顶点，将无向图划分成 2 个子域。2 个子域权值比要处于限定范围 $Range=\left[\frac{\lfloor M/2 \rfloor}{M-\lfloor M/2 \rfloor}-\varepsilon,\frac{\lfloor M/2 \rfloor}{M-\lfloor M/2 \rfloor}+\varepsilon\right]$ 内，ε 为某一定值，若不能满足则考虑采用分裂子块方法实现。

假定无向图 G 经一次二分得到的 2 个中间子域 G_1 和 G_2，以此为例描述通过广度优先搜索将无向图分为 2 个子域的过程：1)由入口顶点开始搜索；2)在利用广度优先策略搜索过程中，当搜索到某一顶点的邻点(邻点是指 2 个顶点之间有边相连)时，首先将排除已加入子域 G_1 中的邻点，然后将剩余邻点先度数由大到小再权值由小到大进行排序，最后将排序的邻点逐个加入子域 G_1 中。定义子域总的权值为该子域包含的顶点权值之和，记为 WG。终止加入顶点到子域 G_1 的条件为

①在加入某顶点到子域 G_1 后，$\frac{WG_{G_1}}{WG_{G_2}}\in Range$；

②在加入某顶点到子域 G_1 前，$\frac{WG_{G_1}}{WG_{G_2}}\leqslant\frac{\lfloor M/2 \rfloor}{M-\lfloor M/2 \rfloor}$；

③在加入某顶点到子域 G_1 后，$\frac{WG_{G_1}}{WG_{G_2}}\geqslant\frac{\lfloor M/2 \rfloor}{M-\lfloor M/2 \rfloor}+\varepsilon$。

在遇到理想终止条件①后，无向图 G 中剩余的顶点构成子域 G_2，称该顶点为临界点。以图 2 中的无向图 G 为例说明当遇到终止条件为②时的处理办法。假定遇到临界点前，无向图 G 顶点已被分为 2 组 $G_1=\{v_1,v_2,v_3,v_4,v_5\}$ 和 $G_2=\{v_6,v_7,v_8,v_9\}$，待处理顶点 v_5 的邻点构成临时数组 $G_t=\{v_6,v_7,v_8\}$，按照顶点度数由大到小再权值由小到大对 G_t 中的顶点进行排序，选定 v_7 加入 G_1，此时出现终止条件②，处理方法为：将 v_7 退回到 G_2，按照顶点度数由小到大再权值由大到小重新对 G_t 中顶点排序，按顺序选择顶点进行分裂。为最小化切割边的权值之和以及避免权值小的顶点被分裂影响计算精确度，要重新按照度数由大到小再权值由小到大排列 G_t 中顶点。图 2 中的无向图 G 经广度优先搜索以及顶点 v_6 的二分得到图 3 中的图 G'，其中虚线表示被切割的边，标出剖分图 G 得到的 2 子域。相应地，图 1 中第 6 块物理块分裂后的计算区域如图 4 所示。

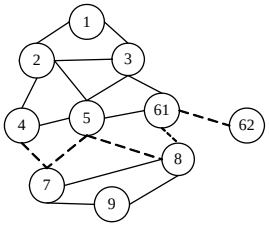

图 3　图 G 经一次二分后得到的图 G'

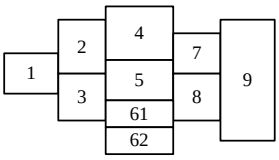

图 4　图 1 中第 6 块物理块分裂后示意图

根据 3.2.2 节中无向图与计算区域的对应关系可知，顶点分裂相当于对计算区域中与该顶点对应的由连续网格单元构成的子块进行划分，具体方法如下：

(1)选择子块中网格单元数最多的方向为候选分裂方向，避免得到的新子块在某一方向网格单元数过少。若存在多个候选分裂方向，查找被划分的子块与 G_1 中顶点对应的子块间的相邻面，判断该相邻面的法向方向是否与某一候选分裂方向重合，若重合则选择该方向为分裂方向，否则任选一个候选分裂方向为分裂方向。

(2)计算子块中将加入子域 G_1 的部分在分裂方向的网格单元数(将加入子域 G_1 的部分在分裂方向的网格点数为 $Grid_{G_1}$，另外 2 个方向的网格单元数分别为 $Grid_1, Grid_2$)，定义 $Grid_T = \dfrac{WG \times \dfrac{\lfloor M/2 \rfloor}{M} - WG_{G_1}}{Grid_1 \times Grid_2}$，若 $Grid_T$ 为整数，则 $Grid_{G_1} = Grid_T$；否则 $Grid_{G_1}$ 有 2 种选择：$Grid_{G_1} = \lfloor Grid_T \rfloor + 1$ 或 $Grid_{G_1} = \lfloor Grid_T \rfloor$。分别根据上述 2 种方案，计算 2 个新子域中处理器平均计算量与整个计算区域处理器平均计算量差值的较大值，选择该值较小的方案为最佳方案。

在图的划分过程中，当达到负载平衡时，为了得到较优解逐个分裂临时数组 G_t 中的所有顶点，本文利用分裂子块的方法对其进行调整，如图 2 中，分裂顶点 v_6 的同时，对 v_7 和 v_8 进行分裂。由于实际应用过程中，每次二分临时数组中的顶点数不多，因此可采用优化通信的切割边的权值总和最小方法。

4　实验结果

本文选用 2 个实例进行说明，其中，例 1 是多区域问题；例 2 是单区域数据块较多的问题。

例 1 模拟发动机的内部流场分 2 个计算区域，分别标为区域 Region1 和区域 Region2。每个区域由多个数据块组成，Region1 含有 6 个数据块，分别为 PhyBlock2, PhyBlock3, PhyBlock4, PhyBlock5, PhyBlock6 和 PhyBlock13；Region2 含有 8 个数据块，分别为 PhyBlock1, PhyBlock7, PhyBlock8, PhyBlock9, PhyBlock10, PhyBlock11, PhyBlock12 和 PhyBlock14。2 个区域的相同点为：控制方程采用粘性 NS 方程、MUSCL 外插；无粘通量采用 Steger-Warmming 模式；时间推进采用 LU-SGS；湍流模型采用 K-Omega 模型；组分数为 10。不同点为：Region1 是完全气体，Region2 是带有 10 个组分数的化学方应。计算方法采用隐式有限体积法，根据网格单元数和网格单元权值得到区域计算量。鉴于上述 2 个区域的差异，网格单元权值分别取 0.5, 1.0。

计算数据块的具体情况见表 1，NI, NJ,NK 分别表示 i, j, k 方向的最大网格单元数。

表 1　例 1 计算数据块的具体情况

物理块号	NI	NJ	NK
1	41	90	1
2	257	40	1
3	60	40	1
4	85	80	1
5	257	50	1
6	90	90	1
7	12	90	1
8	150	140	1
9	39	169	1
10	109	140	1
11	30	350	1
12	102	90	1
13	139	90	1
14	85	80	1

定义衡量负载平衡的标准 $\phi = \max_i \dfrac{N_i - \bar{N}}{\bar{N}}$，其中，$N_i$ 为每个处理器分得的网格数；$\bar{N} = (\sum_{i=1}^{p} N_i)/p$ [6]，p 是处理器数。采用上述自动分块算法，结果见表 2。

表 2　例 1 中不同处理器数的负载平衡情况

处理器数	4	8	16	32	64	128
ϕ/(%)	5.28	5.36	5.48	8.18	3.09	7.67

例 2 本例模拟了磁悬浮列车的外部流场情况，仅有一个计算区域，该计算区域共包含 34 个物理块。计算数据块的具体情况见表 3。

表 3　例 2 计算数据块的具体情况

物理块号	NI	NJ	NK
1	52	15	13
2	52	10	39
3	19	5	253
4	6	66	444
5	7	444	62
6	13	5	253
7	52	39	10
8	52	15	13
9	13	26	53
10	7	5	444
11	70	95	38
12	13	26	38
13	70	95	53
14	444	95	19
15	119	444	29
16	52	15	13
17	52	19	95
18	52	10	39
19	19	5	198
20	59	19	253
21	13	26	19
22	62	7	444
23	52	39	10
24	52	15	13
25	52	19	95
26	19	5	56
27	444	7	5
28	6	66	444
29	13	26	53
30	13	5	253
31	319	35	444
32	13	26	20
33	119	444	36
34	20	444	129

负载平衡的定义参照例 1 中的定义。采用上述自动分块算法，结果见表 4。

表 4　例 2 中不同处理器数的负载平衡情况

处理器数	1	4	8	16	32	64	128
ϕ/(%)	0.0	0.15	0.52	0.57	1.02	2.73	2.02

(下转第 79 页)