

**《形式语言与计算理论》**

**课程报告**

**题目： 并行计算模型**

**姓名： 董玲玉**

**学号： B20200339**

**专业： 计算机科学与技术**

**学院： 计算机与通信工程**

**成绩：**

**任课教师：段世红**

**2020年11月18日**

目录

[1 引言 3](#_Toc56782381)

[2 相等子集最小割——NPC问题 5](#_Toc56782382)

[2.1 相等子集最小割问题描述 5](#_Toc56782383)

[2.2 图的最大割问题描述 6](#_Toc56782384)

[2.3 相等子集最小割的NPC证明 8](#_Toc56782385)

[3 区域分解常用方法 9](#_Toc56782386)

[3.1贪婪算法 10](#_Toc56782387)

[3.2多级K-Way算法 11](#_Toc56782388)

[3.3遗传算法 13](#_Toc56782389)

[3.4递归二分法 15](#_Toc56782390)

[3.4.1递归坐标二分法 15](#_Toc56782391)

[3.4.2递归索引二分法 15](#_Toc56782392)

[3.4.3递归几何二分法 16](#_Toc56782393)

[3.4.4递归图二分法 16](#_Toc56782394)

[3.4.5递归谱二分法 17](#_Toc56782395)

[参考文献 18](#_Toc56782396)

# 1 引言

MIMD（多指令流多数据流）多处理器编程设计思路为：将一个大的问题分解为许多小的子问题，考虑子问题之间的通信链路有哪些，将这些子问题根据连通性和负载均衡两个方面的条件聚合成为更大的子问题，最后将聚合的子问题再以一定的组合映射到处理器上。这四个步骤分别称为分解（Partitioning）、通信（Communication）、聚合（Agglomeration）以及映射（Mapping）。

1）分解（Partitioning）：

将要执行的计算和计算所用的数据分解成小任务。诸如目标计算机中的处理器数量之类的实际问题被忽略，注意力集中在识别并行执行的机会上。

2）通信（Communication）：

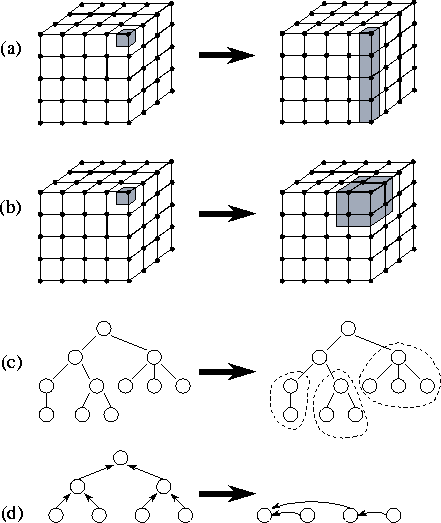
确定了协调任务执行所需的通信，定义了适当的通信结构和算法。

3）聚合（Agglomeration）：

根据性能需求和实现成本评估前两个阶段的任务分解和通信结构，如果需要，将任务合并成更大的任务，以提高性能或降低开发成本。

4）映射（Mapping）：

每个任务都以一种试图满足最大化处理器利用率和最小化通信成本的竞争目标的方式分配给处理器。映射可以通过负载平衡算法静态指定或在运行时确定。



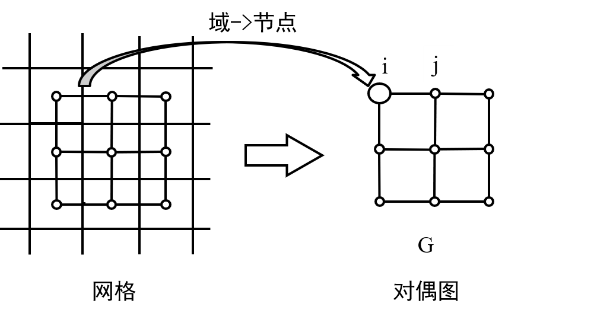
**区域分解简单示例**

区域分解技术对应四个步骤中的聚合步骤。一般来说，由于复杂问题的求解往往需要经过将一个大的问题分解为许多子问题的过程，专注于问题的求解而忽略计算机中的处理器数量等实际问题，导致子问题的粒度，也就是计算复杂度，与处理器的性能往往是不匹配的。将子问题直接作为计算的基本单元，将带来大量的调度与通信成本，无法直接形成有效的并行算法。

其中影响并行性能的一个关键问题就是通信成本，在大多数并行计算机上，在发送和接收消息过程中必须停止计算，而通常通信时间成本比计算高得多，减少通信占用的时间可以提高性能，既可以通过发送更少的数据来实现，也可以通过减少通信次数但增加每次通信的数据量的方式实现。

而计算任务的负载失衡会导致计算资源的浪费，将会出现有的处理器早早完成了计算，却因为其他负担任务繁重的节点尚未完成计算，而一直处于等待状态中。同样的，由于负载不同，处理器运行的同步性变得更差，变相增加了通信成本。

因此，有必要根据性能和实现成本的需求对划分的子任务进行聚合得到更大的粗粒度的任务，降低通信频率。在CFD（Computational fluid dynamics，计算流体力学）求解程序中体现为对网格单元进行分区，使得各个分区内的网格数量相近以满足计算负载的均衡、各个分区之间割的数量较少以满足通信频率的降低。



**网格到对偶图转换**

为了方便进行聚合，使用对偶图来表示网格。对偶图是用顶点表示一个网格，如果网格元素共享一条边（三维是共享一个面），就在顶点之间放置一条边，通过计算链接不同子域内的顶点数估计总通信量，对偶图提出的目的是减少编码的复杂性，减小编码难度，即用更少的信息来表示图，其意义也在于关于域的问题可以转换成关于节点的问题。

于是关于网格节点的聚合问题就转换成求图的**相等子集最小割**问题，也称为**区域分解**问题，后文就该问题展开论述。

# 2 相等子集最小割——NPC问题

## 2.1 相等子集最小割问题描述

**输入**：图G=(N,A)，两个不同的节点s和t，正整数W

**性质**：节点集，，，并且

该问题是一个NP-complete问题，如何证明？

在实际中，我们判断一个问题是不是NP-complete，通常不会去根据这个定义来判断，而是使用Reduction来判断，就是找到一个已经被证明是NP-complete的问题，然后尝试reduce。

总的来说，判断一个NP问题是不是NP-Complete的两个方法

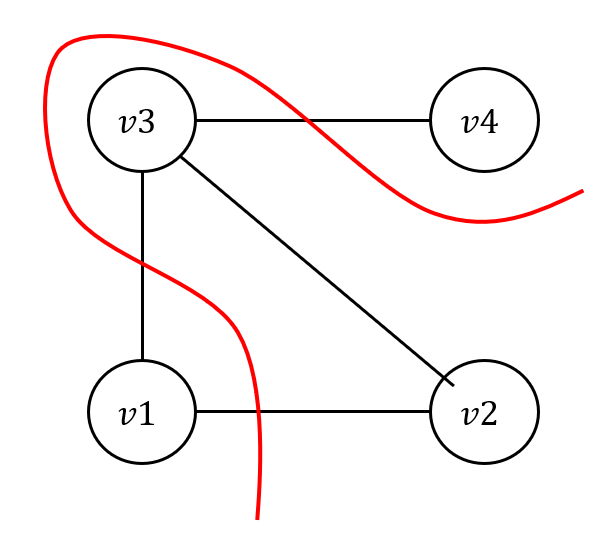
1. 找到一个NP-Complete问题，经过证明可以reduce to 你的问题，这意味着你的方法可以解决这个NP-Complete问题，那很显然，这个解决方法也是NP-Complete的。
2. 所有的NP问题都可以reduced到你的问题

很显然，方法1简单多的，我们只要找到一个现成的 NP-Complete问题就可以了。而在相等子集的最小割问题中，需要借助简单的图的最大割来辅助证明NP-complete。

## 2.2 图的最大割问题描述

最大割问题（Maximum Cut）是NPC问题。给定一张图，求一种分割方法，讲所有定点（Vertex）分割成两群,同时使得被切断的边（Edge）数量最大。

例如G（E,V）是如下图的无向图，包含四个点，我们给出这个问题的解为3，验证这个答案对不对。显然这个答案是正确的，比如我们将顶点集合V分成S、T两个子集S={}，T={}，显然我们找不到割为4的解法。



**四节点无向连接图**

要解决这个问题，首先我们分析它的复杂度。这个问题实际上是将一个图的顶点集合分成两个子集的问题，分成两个子集之后再看这两个顶点子集之间存在多少条边，最大边数的分割方式就是问题的解。一个顶点可能属于子集A也可能属于子集B，每个顶点就有两个可能性，若这个图有n个顶点，这个问题复杂度至少为O（2^n），为指数级。但是给出这个问题的解来验证解的正确性是多项式范围内的。G（E,V）可以表示为：

M是邻接矩阵，表示点与点之间存在边，则不存在。已知一个解S={}，T={}，验证这个解正确与否转换为统计等于1的个数，若结果为3，则解正确。验证答案正确性相当于遍历邻接矩阵M,因此复杂度为O（n^2）,是多项式时间复杂度。

图的最大割形式化语言描述如下：

**输入**：图，权值函数（非负整数），正整数

**性质**：有子集，满足

其中N是点集，A是边集。

## 2.3 相等子集最小割的NPC证明

2.2节中简单介绍了图的最大割问题的NPC问题特征，事实上Karp在1972年的论文[1]中提出的21个NPC问题中包含了最大割问题的详细证明。

与图的最大割问题一样，相等子集最小割同样是一个NPC问题，要判断一个NP问题是不是NPC问题的最常用方法是利用可归约性。即找到一个NPC问题，经过证明可以规约到相等子集最小割问题，因为被规约的问题更复杂，能用来解决被规约问题的算法也能用来解决该NPC问题，那么显然相等子集最小割也是NPC问题。而这个事先找到的NPC问题就是图的最大割问题。下面是证明归约性的简单过程。

**定理**：简单图的最大割相等子集最小割[2]

**证明：**给定图以及正整数W作为图的最大割的输入，令n=|N|，U={}满足U∩N=。相应的，相等子集最小割的输入是：图以及正整数，定义如下：

假设有分割,那么。因为W是正数，都是非空集。令，有是图的一种分区，有,并且：

现在假设有一种分区：，且那么，其中是图G的一种分区方法，其满足：

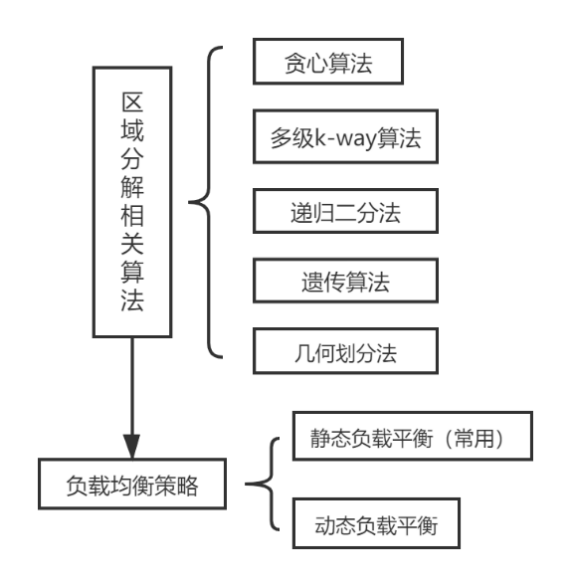
因此只有当存在带权割边小于等于，G才会有带权割边大于等于W。如此就将分隔开并将图的节点分开到两个相等大小的子集，规约得证。

# 3 区域分解常用方法

既然相等子集最小割是一个NPC问题，那么也就是说它无法在多项式时间内找到最优解，因此我们一般使用图划分算法或启发式算法完成分区操作。

尝试使用的算法可能不是每次都能给出最优解，但至少在大多数情况下都能给出一个好的解。好的区域分解算法需要在执行时间与解决方案的质量之间做好权衡。

常用的有图上列出的这几种，贪心算法一般按网格的几何形状进行划分，几何划分算法属于图划分算法，遗传算法和多级K-Way算法属于启发式算法。这些算法在运行速度、分区效果、内存占用等各方面存在差异。下面简单列出区域分解常用方法的算法描述以及优缺点。

****

**区域分解常见策略**

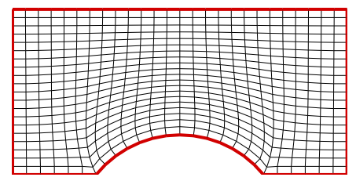
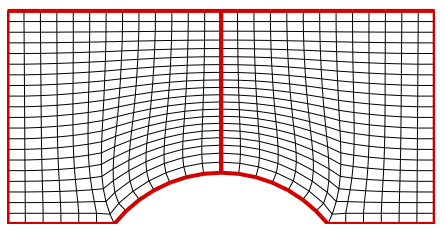
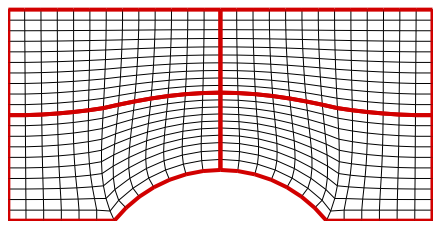
3.1贪婪算法

贪婪算法一般用于多块结构网格的区域分解处理。通过估计进程的期望负载和网格块包含的计算量，优先分割计算量大的网格块，分配到期望负载大的进程上。一般流程如下：

1）估计各个网格块的计算量以及各进程的期望负载、计算能力。

2）从队列中找出计算量最大的网格块以及计算能力最强的进程，通过分割算法拆分网格块，尽可能使得进程达到期望负载，若有多余的网格块则回收到队列中。

3）更新期望负载信息，重复上述过程直到队列中的所有网格块完成分配。

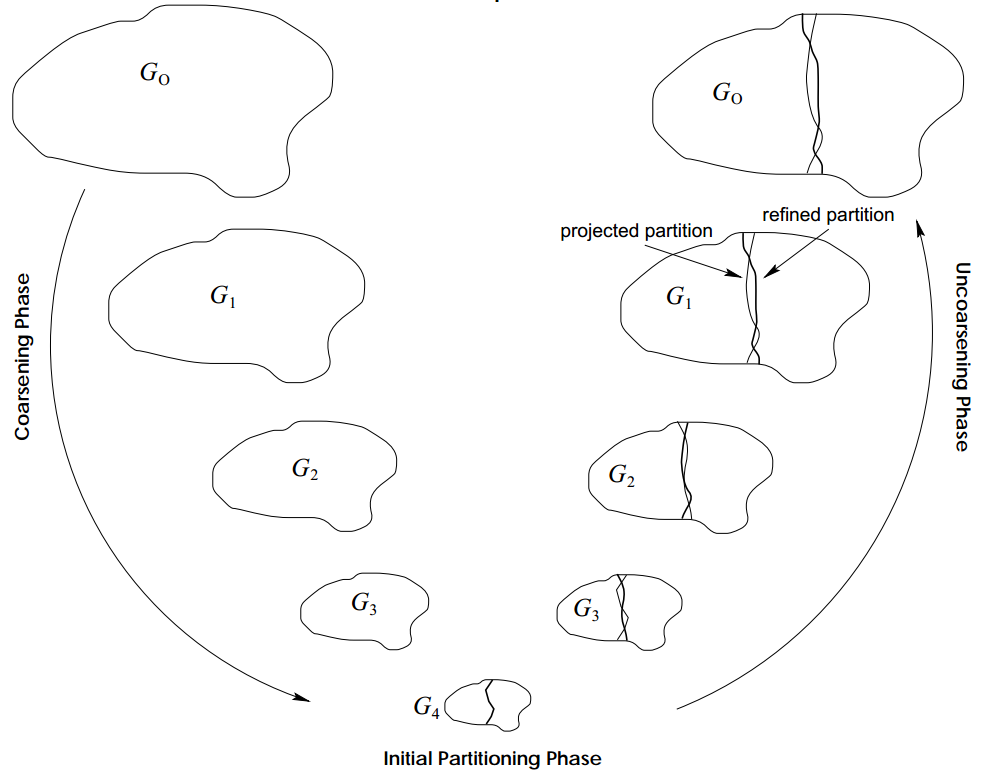
**初始网格 一分为二 二分为四**

**优势：**过程简单，易于实现，执行速度快。

**劣势：**一般用于多块结构网格的区域分解，并且无法有效控制分区间的进程通信开销。

3.2多级K-Way算法

多级K-Way算法需要经过粗化、划分、反粗化三个阶段来完成网格区域的分解。分区之前首先将网格转换为对偶图。



**多级K-Way划分流程**

**粗化阶段：**

将初始图经过一系列转换，得到一系列较小的图、、…、 。其中图的顶点数量记作||，并且满足||>||>…>||。图的转换可以通过一系列方式实现，基本原理是通过选取图中相邻的顶点进行折叠，常用的方法有以下几种：

Random Matching (RM)

以随机的顺序访问各个顶点，随机查询其邻居顶点的匹配情况，若从未经过匹配，则将两个顶点均标记为同一匹配状态，拥有同一匹配状态的顶点将产生折叠。若不存在未匹配的邻居，则该顶点在本次粗化中保持不变。

Heavy Edge Matching (HEM)

以随机的顺序访问各个顶点，查询其邻居顶点的匹配情况，选出其中边权重最大的未匹配的邻居顶点，将两个顶点标记为同一匹配状态，拥有同一匹配状态的顶点将产生折叠。若不存在未匹配的邻居，则该顶点在本次粗化中保持不变。

Light Edge Matching (LEM)

与HEM方法类似，但选择边权重最小的未匹配邻居顶点进行折叠。

Heavy Clique Matching (HCM)

与HEM方法类似，但通过顶点权重、边权重、已粗化的边权重等参数构造边密度值函数，选择边密度最大的未匹配邻居顶点进行折叠。

**划分阶段：**

划分阶段需要将粗化得到的图进行区域分解操作，得到对应处理器数量的分区。该阶段通常借助其他图划分算法进行实现，下面列出几种：

1）Kernighan-Lin Algorithm (KL)

KL算法首先随机初始化得到一个分区方案，通过选择分区内的一系列顶点尝试与另一分区进行交换，当可以平衡负载及减小通信时确定交换，最终得到效果较好的划分结果。由于经过粗化的图内顶点数量少，分区的时间成本也比较低。

2）Graph Growing Algorithm (GGP)

图增长算法通过顶点集合的不断扩展完成图的分区。集合初始只包含一个顶点，这个顶点由随机方式获得。此后以一定规则，不断将集合的邻居顶点收入集合中，直到顶点数量达到顶点总量一半时完成划分。

由于该方法初始顶点的选择将对结果产生非常大的影响，一般通过选取多个初始点生成划分结果，取其中最好的。

**反粗化阶段：**

对于图的分区操作将得到分区结果,将分区结果根据上一级子图的粗化过程进行映射，得到分区结果，在此时调用KL算法对分区结果进行调整，得到最终的。

逐级向上映射最终得到图的分区结果。

**优势：**多级K-way算法拥有非常快的执行速度以及良好的分区结果。

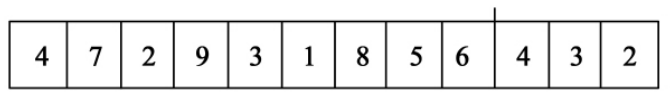
**劣势：**由于需要存储一系列粗化的子图以及粗化过程中的相关信息，对内存的占用比较严重。

3.3遗传算法

遗传算法的一般流程为：

1. 设计染色体形式，形成初始种群
2. 交叉、变异
3. 计算适应度，更新最优个体
4. 重复2）、3）过程直到满足条件

遗传算法在区域划分问题中应用时，一般采用两段式的染色体编码，染色体的前段为网格编号的序列，后段为每个区域拥有的网格数量，每一个染色体就代表了一种网格分区的方案。



**两段式染色体示例**

由于不允许同一网格被划分到多个分区，染色体前段的网格编号不允许出现相同的情况。这一约束影响了后续的交叉、变异操作，一般采用**部分匹配交叉**的方式进行新染色体的生成，具体的操作方式为：

1. 在两父代中随机选出两个交叉点
2. 子代1和子代2分别继承父代1、2交叉点间的片段
3. 根据交叉点间的片段得到网格编号的映射关系
4. 子代1和子代2分别继承父代1、2的剩余片段，根据映射关系调整网格编号

对于变异操作，一般使用**逆位算子**或**交换算子**来操作，即随机选出两个变异点，交换两点的编号或使得两点间的编号逆序排列。

对于后段染色体，不参与交叉过程，只参与变异过程。变异方式为随机选出两个变异点，对其求平均或其他的数值操作。只需保证所有位的总和与网格总数相等即可。

适应度函数的设计必须考量负载平衡和进程通信两个方面的因素，因此一般分为两个部分，例如第一部分为平均网格数与各区域网格数的差值的绝对值之和。第二部分为各个分区的边界面积之和。

整个遗传算法运行的目标就是求得拥有最小适应度函数值的个体。

**优势：**综合考虑负载平衡和最小通信，逼近最优划分。

**劣势：**随着网格规模和种群数量的增大，内存的需求和时间成本将非常巨大。

3.4递归二分法

递归二分法是一类算法的总称，通过将网格转换得到的对偶图以一定规则划分为两个子图，并对子图递归的进行划分操作来得到最终的分区结果[3]。不同的递归二分法主要的差异在于分区的规则，常用的有以下几种。

3.4.1递归坐标二分法

递归坐标二分法的分区规则是根据网格的坐标值进行排序，根据排序结果将网格划分为两个部分。根据不同维度坐标值的范围，选择范围最宽的某一维度坐标进行排序。

|  |
| --- |
| **算法 递归坐标二分法** |
| 1.确定定义域范围最宽的维度 |
| 2.对该维度的坐标进行排序 |
| 3.根据排序结果为每个区域分配一半顶点  4.重复1-3步直到达到目标分区数量 |

**优势：**递归坐标二分法实现简单，运行速度快，划分得到的分区形状规则。

**劣势：**该方法并不考虑网格之间的连接关系，同一分区内可能存在不连通的两个部分，无法控制通信成本。

3.4.2递归索引二分法

递归索引二分法的分区规则是对网格坐标值按一定方式进行映射，得到一个索引值。对索引值进行排序，根据排序结果将网格划分为两个部分。

|  |
| --- |
| **算法 递归索引二分法** |
| 1.确定索引规则，将网格坐标转换为索引值 |
| 2.对网格的索引值进行排序 |
| 3.根据排序结果为每个区域分配一半顶点  4.重复1-3步直到达到目标分区数量 |

**优势：**递归索引二分法的实现同样简单，运行速度快。

**劣势：**分区结果较差，与递归坐标二分法相近。

3.4.3递归几何二分法

递归几何二分法的数学原理非常复杂，但实现过程比较直观。

|  |
| --- |
| **算法 递归几何二分法** |
| 1.将输入的d维网格坐标点p映射到d+1维下以原点为球心的单位球面上。 |
| 2.在球内找出一个特殊点，使得经过该点的任意空间平面可以将球上映射点划分为两个部分。 |
| 3.通过旋转和映射变换使得特殊点成为新坐标系下的原点。  4.在映射球上任选一个圆，反映射回d维，进行一定的转化，将网格坐标划分为两个部分 |
| 5.重复1-4步直到达到目标分区数量 |

**优势：**执行速度快，综合考虑负载平衡和最小通信。

**劣势：**对网格单元的形状有一定的要求，单元纵横比以及顶点角度都有限制。比较适合二维问题的处理。

3.4.4递归图二分法

递归图二分法[5]的分区规则是选出转换得到的对偶图中距离最大的两个顶点，计算其他顶点到其中一个顶点的最短距离，对这些距离进行排序，将顶点划分为两个部分。该方法的一大难点在于如何求得图的直径至少是伪直径，有一些启发式算法可以用于该难点解决，如SPARSPAK RCM 算法。

|  |
| --- |
| **算法 递归图二分法** |
| 1.找出图中相隔最远的两个顶点 |
| 2.计算其他顶点到其中一个顶点的最短距离，排序 |
| 3.根据排序结果为每个区域分配一半顶点  4.重复1-3步直到达到目标分区数量 |

**优势：**递归图二分法一定程度上考虑了图的连接关系，在一次划分的过程中至少可以保证得到的其中一个分区是连通的，同时满足了负载均衡。

**劣势：**时间成本比较大，可能出现质量差的划分结果。

3.4.5递归谱二分法

递归谱二分法[4]的分区规则是计算转换得到的对偶图的费德勒向量，根据向量各个维度分量的正负情况将对应的网格划分到两个分区。费德勒向量是图的拉普拉斯矩阵的次小特征值对应的特征向量，求解过程非常困难，一般需要使用Lanczos[9]迭代方法，将对称稀疏矩阵转换为三对角阵并降低矩阵维度，继而求解三对角阵的费德勒向量。

|  |
| --- |
| **算法 递归谱二分法** |
| 1.计算图的费德勒向量 |
| 2.对费德勒向量各个维度的分量进行排序 |
| 3.根据排序结果为每个区域分配一半顶点  4.重复1-3步直到达到目标分区数量 |

**优势**：根据费德勒向量的相关数学原理[6,7,8]，可以保证根据其划分得到的两个分区是连通的，并且网格的数量相近。该方法可以满足负载平衡和通信成本两个方面的考量。

**劣势**：由于分区过程中矩阵的转换以及费德勒向量的求解计算复杂度高，在面对大规模网格时花费的时间成本非常高。

# 参考文献

[1] Karp R M. Reducibility among combinatorial problems[M]//Complexity of computer computations. Springer, Boston, MA, 1972: 85-103.

[2] Johnson D S, Stockmeyer L. Some simplified NP-complete graph problems[J]. Theoretical Computer Science, 1976, 1: 237-267.

[3] Simon H D. Partitioning of unstructured problems for parallel processing[J]. Computing systems in engineering, 1991, 2(2-3): 135-148

[4] Pothen A， Simon H D， Liou K P. Partitioning sparse matrices with eigenvectors of graphs[J]. SIAM journal on matrix analysis and applications， 1990， 11(3): 430-452.

[5] A. George and J. Liu, Computer Solution of LargeSparse Positive Definite Systems, Prentice-Hall, Englewood Cliffs, New Jersey, 1981.

[6] Fiedler M. A property of eigenvectors of nonnegative symmetric matrices and its application to graph theory[J]. Czechoslovak Mathematical Journal， 1975， 25(4): 619-633.

[7] Slininger B. Fiedlers theory of spectral graph partitioning[J]. 2013.

[8] Chan T, Gilbert J R, Teng S H. Geometric spectral partitioning[M]. Xerox Corporation, Palo Alto Research Center, 1994.

[9] Parlett B N， Simon H， Stringer L M. On estimating the largest eigenvalue with the Lanczos algorithm[J]. Mathematics of computation， 1982， 38(157): 153-165.