相似驱动的细粒度并行任务重构算法

郝水侠[1,2,3] 曾国荪[1,2] 马小信[1,2] 许金超[1,2]
(同济大学计算机科学与技术系 上海 201804)[1]
(国家高性能计算机工程技术中心同济分中心 上海 201804)[2]
(江苏师范大学数学科学学院 徐州 221116)[3]

摘 要 异构计算是高性能计算技术的发展趋势,计算任务与体系结构匹配成为异构计算亟待解决的问题。重构技术为实现两者匹配带来了契机,要么任务重构适应体系结构,要么体系结构重构适应任务。提出基于相似驱动的并行任务重构算法以实现异构计算匹配。通过给出任务和系统匹配度量机制定义了图重构操作和图重构基本问题。根据问题给出细粒度重构算法,该算法主要有3个过程:任务图节点对融合、节点和边重构及重构精化过程。用格林威治大学典型实例图作为并行任务及典型体系结构测试了该算法。实验表明它在给定的误差范围内能保证计算任务和体系结构匹配。

关键词 并行任务,体系结构,异构计算,图相似,重构
中图法分类号 TP338 **文献标识码** A

Similarity-driven Fine-grained Parallel Task Reconfigurable Algorithm

HAO Shui-xia[1,2,3] ZENG Guo-sun[1,2] MA Xiao-xin[1,2] XU Jin-chao[1,2]
(Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)[1]
(Tongji Branch, National Engineering & Technology Center of High Performance Computer, Shanghai 201804, China)[2]
(School of Mathematical Science, Jiangsu Normal University, Xuzhou 221116, China)[3]

Abstract Heterogeneous computing is a novel trend of high-performance technology. Mismatch between parallel tasks and architecture leads to realize high performance hardly. Reconfiguration technique brings an opportunity for it. In order to realize their matching, this paper provided similarity-driven reconfigurable algorithm. Firstly the paper gave definitions of graph-similarity and graph-reconfiguration. Secondly through building a measurement between task graph and fixed architecture, the paper provided a graph-reconfiguration problem and a similarity-driven reconfigurable algorithm. The algorithm includes three processes: nodes-fused, node_edge_reconfiguration and node_edge_refined. Finally the paper gave a test on Greenwich benchmark with the typical architecture. The results show that the algorithm can realize the optimal match between the parallel tasks and the architectures on heterogeneous computing.

Keywords Parallel task graph, Architecture, Heterogeneous computing, Similarity-driven, Reconfiguration

1 引言

异构计算是高性能计算技术的发展趋势。例如美国洛斯阿拉莫斯国家实验室研制了“走娟”(Roadrunner),中国曙光公司研发了“星云”(Nebulae)。异构计算指性能和功能各异的计算机(如:PC、工作站群、向量机、SIMD、MIMD计算机、GPU、DSP、CELL、ASIC专用机等)通过高速网络,在一定的耦合方式连成的并行计算环境中,充分利用程序和处理部件的异构性,各尽潜能,合理分治,协同计算一个应用任务,使其完成时间最小的过程。它是应用任务存在异构和计算机系统存在异构的必然产物,是高性能并行计算发展的必然结果。在异构计算中,由于处理器的计算能力和连接方式各不相同,并行任务分配也随之有所变化,因此如何保证计算任务和体系结构匹配是异构环境下实现最大效能的关键要素。

可重构计算指硬件的结构配置信息可以像软件程序一样被动态调用或修改。目前可重构技术主要体现在3个方面:节点内重构、节点间重构和网络间重构。通过可重构计算,一方面可以提高计算机的性能,使之能适应不同的要求;另一方面可以节省软硬件的开发费用,尽量使用已有的资源来构造新的系统,减少不必要的浪费。可重构计算使得系统能兼有硬件的效率和软件的灵活性。重构过程是客观存在的,重构性为充分发挥异构性能提供了空间。

到稿日期:2012-11-02 返修日期:2013-02-25 本文受国家863高技术研究发展计划(2009AA012201),国家自然基金项目(61103068),NSFC-微软亚洲研究院联合资助项目(60970155),教育部博士点基金项目(20090072110035),上海市优秀学科带头人计划项目(10XD1404400),高效能服务器和存储技术国家重点实验室开放基金项目(2009HSSA06)资助。

郝水侠(1973—),女,博士生,副教授,主要研究领域为异构计算、重构计算,E-mail:sxhaotj@gmail.com;曾国荪(1964—),博士,教授,博士生导师,主要研究领域为异构计算、信息安全。

在实现与体系结构结合的并行任务重构时，借助了图划分的思想。图划分问题是一种 NP 完全问题，所以近年来发展了一些启发式算法[1]，如遗传算法、模拟退火、A * 算法、蚁群算法等。但这些算法往往以节点的权值大小为划分目标，辅助割边权值最小化，如最常见且较成功的方法是多级划分[2,3]，这种方法把图划分分为 3 步，即粗化、初步划分和精化还原。但这往往不能满足目前应用对通信提出的一些具体的要求。例如割边最小化，意味着当割边最小时整体通信量最小，然而这种割边最小化在有些场合是不适合的。如 George Karypis 在文献[4,9,10]中提到图划分中割边和通信量并不完全相同，实际划分中应以任务之间的通信量而不是割边来衡量通信大小。另外，C. Walshaw 在文献[3]中提到即使在割边相同的情况下，若目标图的内网和外网之间通信速度有差别，则划分应以映射到处理器的总体通信量最小为划分目标。从这些文献可以发现，图划分不仅仅要考虑待划分并行任务图本身的特征，而且要与实际体系结构相联系。文献[5]利用异构应用任务划分的通信特征来设计网络拓扑结构和路由算法。文献[6]根据体系结构处理器可以通信的最大跳数来设计应用任务的划分方法。这些文献说明，在异构计算中，无论是均衡划分[7]还是非均衡划分[8]，图划分都应考虑实际的体系结构特征。文献[11,12]考虑了体系结构是全连接下的任务划分方法，但实际问题中体系结构大都不是全连接的，各个处理器与其他处理器的通信不完全相同。基于此，本文考虑任意结构上细粒度任务的重构方法。

异构和重构相辅相成，它们可以共同提高整个系统运行的效能。重构是为了实现异构，使并行任务图与体系结构更好地匹配。异构的最终目的是系统高效能运行。因此我们借助重构思想，提出相似驱动的细粒度并行任务重构算法，目的是通过并行任务细粒度重构，实现异构计算中计算任务和体系结构完全匹配，最终实现异构计算的高效能。

2 并行任务图重构的需求和目的

2.1 并行算法设计的一般过程

并行算法一般过程可分为 4 步，即任务划分、通信分析、任务组合、任务映射，具体如图 1 所示。划分，即将整个计算分解成一些小的任务，其目的是尽量开拓并行执行的机会。通信，即确定诸任务执行中所需交换的数据和协调诸任务的执行，由此可检测上述划分的合理性。组合，即按体系结构要求和实现的代价来考察前两阶段的结果，必要时可将一些小的任务组合成更大的任务来提高性能或减少通信开销。映射，即将每个任务分配到一个处理器。并行任务图是并行编译的基础，但是并行任务图中节点的个数远远大于处理器的个数，若将其直接分配给处理，会有很多通信开销，不利于任务并行。因此在进行并行计算前先对并行任务图的节点进行适当的合并和划分，以便在减少通信开销的同时使得任务最大并行化。

实际上，根据体系结构特征将并行任务图中计算节点划分和组合的过程就称为重构。任务划分主要是充分开拓并行任务的并发性和可扩放性。组合是由抽象到具体的过程，其主要目的是合并小尺寸任务，减少任务数，借此通过增加任务的粒度来减少通讯成本。划分和组合是以并行任务图为处理对象，而重构虽然也包括划分和组合，但是它还充分考虑了体系结构特征。

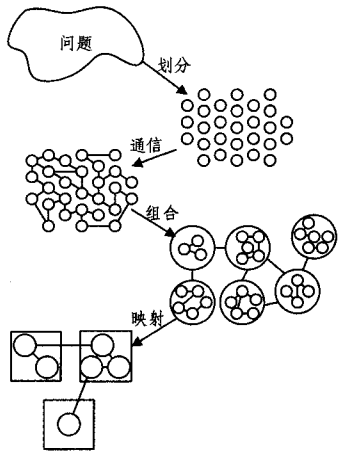

图 1 并行算法的一般过程

2.2 并行任务图的生成和描述

从并行算法设计过程可以看出，问题描述是并行算法研究的起始点，通常是将源程序转变成中间表示形式。常见的中间表示形式有抽象语法树结构、视图和图的表示形式。

本文使用图来表示程序的中间形式，它一方面保留了抽象语法树的结构信息，另一方面更容易实现重用。在并行计算中，常用并行任务 DAG 来表示并行化的中间过程。

并行任务 DAG 图生成过程较复杂，它不仅要考虑程序的数据依赖关系，还要考虑控制依赖关系，此外还得考虑程序中循环等的特殊关系。具体的生成过程如图 2 所示，首先根据程序之间的执行顺序，由源程序生成相应的控制流图(CFG 图)；然后通过对源程序进行数据依赖分析，得到程序数据相关图，同时根据控制依赖关系和前向控制树得到程序控制相关图；将程序数据相关图和控制相关图进行合并，生成程序依赖图；最后通过一定的合并和拆分方法将程序依赖图转换成任务 DAG 图。具体的并行任务可以定义如下。

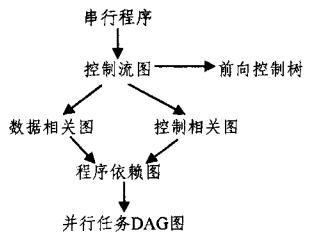

图 2 DAG 图生成过程

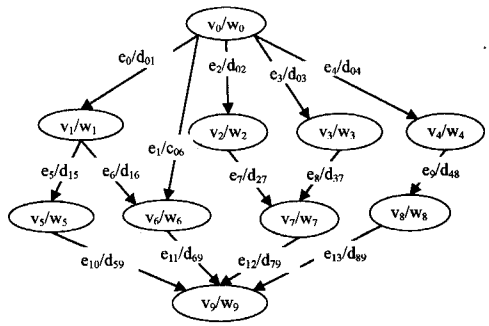

图 3 并行任务

定义 1(并行任务) 一个并行任务可定义为一个有向无环图(DAG)，可用四元组表示，记为 $PTG=(V,E,W,D)$，其中：$V=\{v_1,v_2,\cdots,v_n\}$ 是子任务节点集合，节点 $v_i\in V$ 代表计

算任务中的一个子任务。$E=\{e_1,e_2,\cdots,e_m\}$是子任务节点间有向边集合，$E=V\times V$，$e_k=\overrightarrow{v_iv_j}\in E$表示子任务$v_i$和子任务$v_j$之间存在依赖关系，子任务$v_j$必须等待任务$v_i$完成后才能开始执行。$W=\{w_1,w_2,\cdots,w_n\}$是任务计算量的集合，$w_i$表示子任务$v_i$的计算量。$D=(d_{ij})_{n\times n}$是数据通信量矩阵，$d_{ij}$表示子任务$v_i$和子任务$v_j$的数据通信量，如图3所示。

2.3 固定系统结构假定

目前在并行计算中，尤其是异构计算中，算法的设计和体系结构有密切关系，有的系统结构是固定的，有的是可变的，如可重构系统、按需的云环境。在本文中，我们假定系统结构是固定的。

定义2(固定系统结构) 它可以定义为简单图FAG，用四元组来表示，记为$FAG=(P,L,S,C)$。其中：P是处理器集合，顶点$p_i\in P$代表固定系统结构中的一个处理器。L是处理器之间通信链路的集合，$L=P\times P$，$l_k=\overrightarrow{p_ip_j}\in L$表示处理器$p_i$和$p_j$有直接通信链接。$S=\{s_1,s_2,\cdots,s_m\}$是所有处理器速率的集合，$s_i$表示处理器$p_i$的速率。$C=(c_{ij})_{m\times m}$是所有处理器之间通信带宽(速率)构成的矩阵，即$c_{ij}$表示单位时间内处理器$p_i$和$p_j$之间传输的数据量。

图4是一些典型体系结构图，节点内标号分别是处理器编号和处理器的速率，边上标识了相邻处理器之间的通信速率。图4(a)是树形结构，除了根节点和叶子节点之外，每个内节点只与父节点和子节点相连，它们可以直接通信。图4(b)是二维网孔网络，每个节点只与其上、下、左、右的近邻相连。图4(c)是超立方体，每个顶点的度为3。这些体系结构是典型网络异构的例子。

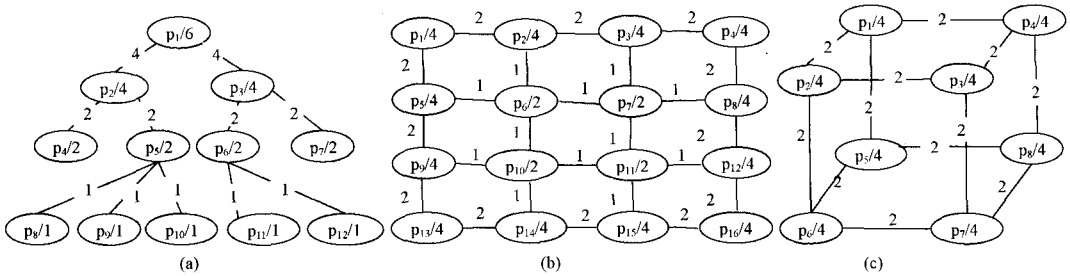

图4 常见的体系结构

2.4 并行任务重构的目的

在异构计算中，并行算法的设计与体系结构有直接的联系。对于同样的问题，不同的算法设计执行的效果有很大的差异。因此在异构计算中针对体系结构的并行任务的划分和组合的需求应用而生。在这里，我们把这种过程称为重构。重构的目的是为实现并行任务和体系结构匹配提供条件，从而更方便达到按需分配，各尽所能，最终实现异构计算的高效能。例如矩阵-向量乘法，设计成串行矩阵-向量乘法，其运算时间为$O(n^2)$；设计成带状划分的矩阵-向量乘法在处理器个数和划分带数相同时，在超立方体和网孔结构上的运行时间为$O(n)$；设计棋盘划分的矩阵-向量乘法在超立方体上的时间是$O(n)$，而在网孔上的时间是$O(\log n)$[13]。根据前面分析，在网孔上用同样多的处理器，棋盘划分的矩阵-向量程序比带状划分时要快。在并行算法调度过程中也有这样的经验，当并行计算任务和系统结构相适应时，算法的效率会大大提高。因此在异构计算中，实施并行任务重构势在必行，有非常重要的现实意义。

异构和重构相辅相成，重构是手段，异构是目的，通过重构，可以实现异构匹配，以便最终实现高效能。并行任务DAG的每个节点都可以看成一个细粒度的并行任务，细粒度有较高的灵活性，可以根据体系结构特征，重构并行任务粒度，以适应体系结构特点，实现并行任务和体系结构的最大匹配。

3 任务与系统匹配的度量机制

前面分析得到，重构为按需分配、各尽所能提供条件，重构的目的是实现并行计算任务和体系结构匹配。如何描述计算任务和体系结构的匹配情况，自然会联想到图同构，当两个图同构时，两个图之间的性质完全相同。若并行任务图和体系结构同构，则可以直接把并行任务映射到体系结构上，实现两者完全匹配。这样就可以实现按需分配、各尽所能，使系统效率达到最高。两个图同构条件要求比较高，很多并行任务图通过重构无法满足条件。因此我们欲借助图同构概念，引入并行计算任务图和体系结构相似的概念，来刻画并行计算任务和体系结构的匹配关系。

3.1 图相似定义

在进行任务重构时，我们需要考虑影响系统性能的几个重要特征：(1)任务的计算量不同，(2)任务之间的通信量不同，(3)处理机的计算速率不同，(4)处理机之间的通信速率不同。文献[8]利用分布向量给出了异构计算的非均衡划分算法，它根据异构计算中处理器计算能力将任务划分成对应的比例。异构计算中，不仅要考虑处理器计算能力的特征，还要考虑系统的结构特征，因此，我们给出异构计算中计算节点分布向量和通信出度权和分布向量。

定义3(图节点权值分布向量) 在一个有权图$G=(V,E,W,D)$中，假设节点个数为m，每个节点对应权值为$w_i(v_i\in V)$，则所有节点的分布向量定义为$\vec{v}_G=(\alpha_1,\alpha_2,\cdots,\alpha_m)$，其中$\alpha_i=\dfrac{w_i}{\sum_{j=1}^{m}w_j}$，且$\sum_{j=1}^{m}\alpha_i=1$。

特别当G为PTG时，$\vec{v}_G$为任务负载分布向量；当G为FAG时，$\vec{v}_G$为计算速率分布向量。

当$\alpha_1=\alpha_2=\cdots=\alpha_m$时，PTG相应的划分是等划分。当$\alpha_i\neq\alpha_j(i,j\in 1,2,\cdots,m)$时是非等分重构，任务负载根据计算速率负载重构，PTG可以按照FAG的计算速率分布向量重构，实现两者对应节点权值分布向量相等。

定义4(图出边权和分布向量) 在一个有权图$G(V,E,$

$W,D)$ 中，假设节点个数为 m，边对应权值为 d_{ij}，记 $d_i=\sum_{(v_i,\forall v_j)\in E} d_{ij}$，$d_G=\sum_{v_i\in V} d_i$。则所有出边权值和分布向量定义为 $\vec{e}_G=(\beta_1,\beta_2,\cdots,\beta_m)$，其中，$\beta_i=\frac{d_i}{d_G}$，且 $\sum_{j=1}^{m}\beta_i=1$。

特别当 G 为 PTG 时，$\vec{e}_G$ 为数据通信需求负载分布向量；当 G 为 FAG 时，$\vec{e}_G$ 为系统通信速率分布向量。

定义 5(图同构)　设图 $G=(V,E)$ 及图 $G'=(V',E')$，其中 V,V' 是节点集，E,E' 是边集，如果 $|V|=|V'|=n$，并且存在一一对应的映射 $g:V\to V'$，且 $e=(v_i,v_j)(v_i,v_j\in V)$ 是 G 的一条边当且仅当 $e'=(g(v_i),g(v_j))$ 是 G' 的一条边，则称 G 与 G' 同构，记作 $G\cong G'$。

定义 6(任务图和体系结构图相似)　设任务图 $PTG=(V,E,W,D)$ 和体系结构图 $FAG=(P,L,S,C)$ 的分布向量分别为 $\vec{v}_{PTG}$、$\vec{v}_{FAG}$、$\vec{e}_{PTG}$、$\vec{e}_{FAG}$，如果 $|V|=|P|=m$，且存在映射 $f:V\to P$，满足，$\vec{v}_{PTG}=\vec{v}_{FAG}$，$\vec{e}_{PTG}=\vec{e}_{FAG}$，则称任务图和体系结构图相似，记作 $PTG\sim FAG$。

命题 1　设任务图为 $PTG=(V,E,W,D)$，体系结构图为 $FAG=(P,L,S,C)$，如果 $PTG\sim FAG$，则图节点权值成比例，即 $\frac{w_1}{s_1}=\frac{w_2}{s_2}=\cdots=\frac{w_m}{s_m}=\alpha$，$\sum w_i=\alpha\sum s_i$。

证明：若 $PTG\sim FAG$，则 $\vec{v}_{PTG}=\vec{v}_{FAG}$。

若两个图向量相等，则对应分量相同，即 $\alpha_i^{PTG}=\alpha_i^{FAG}(i\in\{1,2,\cdots,m\})$。

若 $\alpha_i^{PTG}=\alpha_i^{FAG}(i\in\{1,2,\cdots,m\})$，则有 $\frac{w_i}{\sum_{j=1}^{m} w_j}=\frac{s_i}{\sum_{j=1}^{m} s_j}$，也可以写成 $\frac{w_i}{s_i}=\frac{\sum_{j=1}^{m} w_j}{\sum_{j=1}^{m} s_j}$。

若令 $\frac{\sum_{j=1}^{m} w_j}{\sum_{j=1}^{m} s_j}=\alpha$，则有 $\frac{w_1}{s_1}=\frac{w_2}{s_2}=\cdots=\frac{w_m}{s_m}=\alpha$，$\sum w_i=\alpha\sum s_i$。证毕。

命题 2　设任务图为 $PTG=(V,E,W,D)$，体系结构图为 $FAG=(P,L,S,C)$，它们各自的出边权值和为 $d_i=\sum_{(v_i,\forall v_j)\in E} d_{ij}$ 和 $c_i=\sum_{(p_i,\forall p_j)\in E} c_{ij}$。如果 $PTG\sim FAG$，则图出边权和成比例，$\frac{d_1}{c_1}=\frac{d_2}{c_2}=\cdots=\frac{d_m}{c_m}=\beta$，$\sum d_i=\alpha\sum c_i$。

证明：若 $PTG\sim FAG$，则 $\vec{e}_{PTG}=\vec{e}_{FAG}$。

若两个图出边权和向量相同，则对应分量相同，即 $\beta_i^{PTG}=\beta_i^{FAG}(i\in\{1,2,\cdots,m\})$。

若 $\beta_i^{PTG}=\beta_i^{FAG}(i\in\{1,2,\cdots,m\})$，则有 $\frac{d_i}{\sum d_i}=\frac{c_i}{\sum c_i}$，也可以写成 $\frac{d_i}{c_i}=\frac{\sum d_i}{\sum c_i}$。

若令 $\frac{\sum d_i}{\sum c_i}=\beta$，则有 $=\frac{d_1}{c_1}=\frac{d_2}{c_2}=\cdots=\frac{d_m}{c_m}=\beta$，$\sum d_i=\beta\sum c_i$。证毕。

通俗地说，任务图和体系结构图相似指两个图顶点个数相同，对应节点计算权值和通信出边权和成比例。根据定义可证明两个命题的逆命题也成立。

定义 7(图重构操作)　给定源图 $G=(V,E,W,D)$，其中 $V=\{v_1,v_2,\cdots,v_n\}$，$E=\{e_1,e_2,\cdots,e_m\}$，$W=\{w_1,w_2,\cdots,w_n\}$，$D=(d_{ij})_{n\times n}$，假设相邻的两个节点 $v_i,v_j\in V$，对应的权值分别为 $w_i,w_j\in W$，对应的相邻节点集分别为 $adv(v_i)$，$adv(v_j)$，对应的边权值为 d_{ix}，d_{jy}，其中 $x\in adv(v_i)$，$y\in adv(v_j)$，则图重构操作定义为 v_i,v_j 两节点融合为一个新节点 $v_z'=\{v_i,v_j\}$，融合后 v_z' 的权值为 $w(v_z')=w_i+w_j$；v_i 和 v_j 之间的权值看作为 $d_{ij}'=0$；v' 的相邻节点集为 $adv(v_z')=adv(v_i)+adv(v_j)-\{v_i,v_j\}$，对应的边权值为 $d_{zx}'=d_{ix}$，$d_{xy}'=d_{iy}(x\neq y)$，或者 $d_{zx}'=d_{ix}+d_{jy}(x=y)$。

不相邻节点对不容许重构融合，具体的操作如图 5 所示，顶点内数字代表节点权值，边上数字代表边权值。

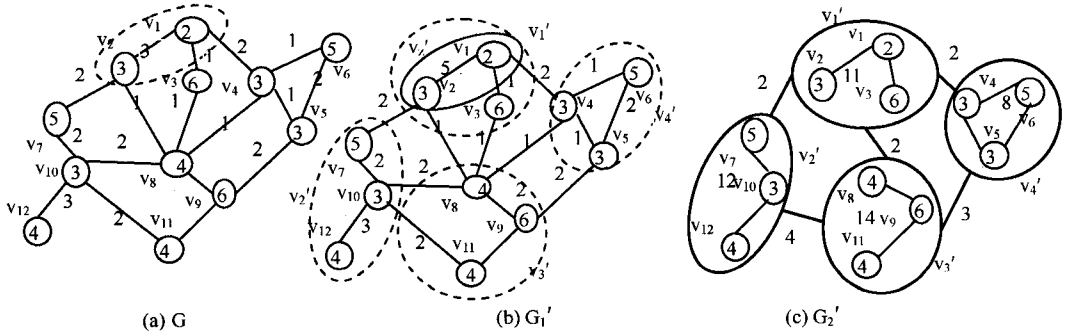

图 5　图重构操作

顶点外标号是顶点编号。图 5(a)是源图，虚线部分是待融合节点。图 5(b)是实施$\{v_1,v_2\}$融合的图。图 5(c)是对图 5(b)所有虚线部分融合重构后形成的图。图 5(a)虚线部分 v_z' 由节点$\{v_1,v_2\}$组成，图 5(b)将其融合为 v_z'。其中 $adv(v_1)=\{v_2,v_3,v_4\}$，$adv(v_2)=\{v_1,v_7,v_8\}$。则融合后 $w(v_z')=w_1+w_2=5$，$adv(v_z')=adv(v_1)+adv(v_2)-\{v_1,v_2\}=\{v_3,v_4,v_7,v_8\}$。对应的边权值 $d_{z3}'=d_{13}=1$，$d_{z4}'=d_{14}=2$，$d_{z7}'=d_{27}=2$，$d_{z8}'=d_{28}=1$。若需要多点融合，则采用逐步增加节点的方法，如在 v_z' 中增加$\{v_3,v_4\}$融合到 v_1'，则 $v_1'=\{v_z',v_3\}$，其中 $adv(v_z')=\{v_3,v_4,v_7,v_8\}$，$adv(v_3)=\{v_1,v_3,v_8\}$。融合后其节点权值等于 $w(v_1')=w(v_z')+w_3=11$。$adv(v_1')=adv(v_z')+adv(v_3)-\{v_1,v_2,v_3\}=\{v_4,v_7,v_8\}$。如图 5(b)所示，$v_1'$ 共有 3 条边(v_1',v_2')，(v_1',v_3')，(v_1',v_4')，其对应边权值分别为 $d_{12}'=d_{27}=2$，$d_{13}'=d_{28}+d_{38}=2$，$d_{14}'=d_{14}=2$。其余重构节点类似。整个图 5(b)的重构结果如图 5(c)所示。

定义 8(任务图重构问题)　给定并行任务图 $PTG=(V,E,W,D)$ 和体系结构图 $FAG=(P,L,S,C)$，其中 $|V|=n$，$|P|=m$，$n>>m$。对 PTG 采用多次重构操作，使得 $PTG\to PTG_1\to PTG_2\to PTG_3\to\cdots\to PTG_k$，使得 $PTG_k\sim FAG$。

在并行任务重构过程中，既考虑了体系结构处理器的能力，也考虑了处理器之间的通信能力，这符合实际应用情况，通过对并行任务图重构，使得并行任务图和体系结构相似。下面在定义的基础上给出具体的细粒度并行任务重构算法。

4 细粒度并行任务重构算法

细粒度并行任务重构算法主要包括3部分：任务图节点对融合过程，即将任务图通过合并节点的方法逐步缩小其节点及边个数；节点和边重构过程，即对缩减后的任务图实施重构操作，使得重构后的任务图和体系结构图初步相似；节点和边重构精化过程，即从缩减图的重构还原到源任务图，在还原的过程中进一步调整各重构图，使得重构图与目标图体系结构相似。下面详细介绍每个过程。

4.1 任务图节点对融合过程

在任务图节点对融合过程中，针对PTG的源图 $G^{(0)}=(V^{(0)},E^{(0)},W^{(0)},C^{(0)})$，利用重构操作，构造一系列节点和边个数减少的图 $G^{(1)},G^{(2)},\cdots,G^{(k)}=(V^{(k)},E^{(k)},W^{(k)},C^{(k)})$，其中 $|V^{(i-1)}|>|V^{(i)}|,i\in\{1,2,\cdots,k\}$，图融合过程利用了'匹配'的思想。$G^{(i)}$ 的匹配 M_i 为边集合，$\forall e_1=(v_1,u_1),e_2=(v_2,u_2)\in M_i$，满足 $v_1\neq v_2,v_1\neq u_2,u_1\neq v_2,u_1\neq u_2$。当算法构造图 $G^{(i)}$ 的一个匹配后，可以把已匹配的两个节点合并成 $G^{(i+1)}$ 的一个节点，未匹配的点就直接复制到 $G^{(i+1)}$，从而减少 $G^{(i)}$ 的节点个数。

算法1 节点对融合算法 nodes_fused_algorithm()

输入：任务图 $G^{(i)}=(V^{(i)},E^{(i)},W^{(i)},D^{(i)})$

输出：任务图 $G^{(i+1)}=(V^{(i+1)},E^{(i+1)},W^{(i+1)},D^{(i+1)})$，映射集合 $Map^{(i)}=\{(v_j,k)|\ v_j\in V^{(i)},k\in N\}$

k=1，$Map^{(i)}=\phi$；// 节点块编号k初始化，节点块映射集合初始化

while $\exists v\in V^{(i)}$ is unmatched

v←get_unmatched_vertex()；// 随机找到未匹配的节点

if $\forall u\in$ adj(v)is matched// Adj(v)为v的邻接节点

matched[v]=v；

else

find an unmatched u←$arg_u(\max d(v,u))\wedge u\in$ adj(v)；//找数据量最大的邻接节点u

matched[u]=v，matched[v]=u；//标记已匹配的节点

$Map^{(i)}$←$Map^{(i)}+\{(u,k)\}$；

set u to be matched；

endif

$Map^{(i)}$←$Map^{(i)}+\{(v,k)\}$；

set v to be matched；

k←k+1；

endwhile

$G^{(i+1)}$←rebuild($Map^{(i)}$，$G^{(i)}$)；//利用 $Map^{(i)}$，重建 $G^{(i)}$，获得融合后的新图 $G^{(i+1)}$

在融合过程中重复执行上述算法，直到融合后图的节点数小于 λk，k 为体系结构中处理器的个数，λ 为常数，根据实验经验，取 $\lambda=15$ 到20。若相继两个图 $G^{(i)}$ 和 $G^{(i+1)}$ 的节点数比 $|V^{(i)}|/|V^{(i+1)}|>0.8$，结束融合过程。

4.2 节点和边重构过程

源任务图通过节点对融合形成新的任务图 $G=(V,E,W,D)$，以系统图 $FAG=(P,L,S,C)$ 为目标，将 G 通过重构操作形成 $G'=(V',E',W',D')$，满足 $G'\sim FAG$。根据命题1和命题2，若 $G'\sim FAG$，则存在 α 和 β 使得对应向量成比例。因为总节点权和在图变换时不会变换，所以 α 比较好计算，即 $\alpha=\frac{\sum w_i'}{\sum s_i}=\frac{\sum w_i}{\sum s_i}$。但总边权和不断在变化，即不断减少，也就是说 $\sum d_i'<\sum d_i$，因此不能简单用 $\frac{\sum d_i}{\sum c_i}$ 代替 $\frac{\sum d_i'}{\sum c_i}$ 来计算 β。假设在任务图变换中边权平均值不变，这符合实际，因此，我们可定义 $\beta\approx\frac{\sum d_i}{|E|}/\frac{\sum c_i}{|L|}$。

重构过程实质上是根据目标图对 G 进行重构操作，是一个 m 步循环过程，最终将得到 G'，并且形成重构映射 Map'。若令 k 为目标块号，它的计算权值为 s_k，出边权和等于 $edge\_cuts(p_k)$，则重构块的目标计算权值等于 $\alpha\times s_k$，目标出边权和等于 $\beta\times edge\_cuts(p_k)$。算法从一点开始，按加入节点影响权值最小的顺序将邻接点逐一加入该重构块，不断地修改 Map'，直到重构块实际权值等于目标值。当所有节点都处理完毕，算法结束。

算法2 节点和边重构算法 node_edge_reconfigure_algorithm()

输入：任务图 G=(V,E,W,D)，系统图 FAG=(P,L,S,C)，任务图和系统图节点权之比 α，边权之比 β，误差 $\epsilon_\alpha,\epsilon_\beta$

输出：G'=(V',E',W',D')，Map'=$\{(v,k)|v\in V,k\in N\}$

k←1，Map'=ϕ；//k为重构节点块编号

while $k\leqslant m\wedge(\exists v\in V$ is undealed)

v←get_undealed_vertex()；//随机选择未处理节点

v_k'←v，w_k'←w(v)，c_k'←edge_cuts(v)；//edge_cuts(v)函数返回节点v的边权和

set v to be dealed；

w_obj=$\alpha\times s_k$，c_obj=$\beta\times$edge_cuts(p_k)；//分别求FAG的对应节点和边权值之和

while $w_k'<w\_obj\times(1-\epsilon_\alpha)\wedge c_k'<c\_obj\times(1-\epsilon_\beta)$

Q←adj()；//在G中求 v_k' 全部邻接节点集

$u\leftarrow arg_u(\min(\frac{w(v_k')+w(u)}{w\_obj}+\frac{edge\_cuts(v_k'+\{u\})}{d\_obj}))\wedge u\in Q$；

//选择影响最小节点

$v_k'\leftarrow v_k'+\{u\}$，$w_k'\leftarrow w_k'+w(u)$，$c_k'\leftarrow edge\_cuts(c_k'+\{u\})$；

set u to be dealed；

Map'←Map'+$\{(u,k)\}$；

endwhile

Map'←Map'+$\{(v,k)\}$；

k←k+1；

endwhile

G'←reconfig(Map'，G)；//利用Map'和G，可以重构基本符合要求的新G'。

通过上述算法，可以得到一个较好的图初始重构，其各个部分的权值与预先指定的值相近。

4.3 节点和边重构精化过程

重构精化过程主要是将融合后的任务图重构映射精化还原到源任务图的重构映射，并且根据最终的重构映射结果形成并行任务重构图，保证重构图和目标图的节点权之比与出边权和比相等，从而使两个图相似。它包括还原和精化两个过程。

还原实质上是算法1的逆过程，即若 $(u,k)\in Map'$，$(x,u),(y,u)\in Map^{(i)}$，$u\in V^{(i+1)}$，k 是重构块标识，$x,y\in V^{(i)}$，则 $(x,k)\in Map''$，$(y,k)\in Map''$，其中 $Map^{(i)}$ 是任务图 $G^{(i)}$ 到 $G^{(i+1)}$ 的映射集合，Map' 是 $G^{(i+1)}$ 的重构映射集合，Map'' 是

$G^{(i)}$的重构映射集合。但还原过程中存在进一步优化的可能，即在$G^{(i+1)}$中移动一个节点不能改善重构图的节点大小，但是还原到$G^{(i)}$后，有进一步优化的可能。因此在每步还原过程中执行一次精化过程。

精化过程是完善重构块实际权值，减少与目标权值的误差。调整方法是对处在重构块边缘的节点移动相邻重构块。调整时选择相邻重构块中误差最大的重构块，计算调整后的重构块实际权值，若误差减少，则进行调整。调整后的收益函数记为$gain(v,p,q)$。调整前重构块p、q的计算权值、出边权值和分别等于w_p,c_p,w_q,c_q。将p重构块的边缘节点v调整到q后，其计算权值、出边权值和分别等于w_p',c_p',w_q',c_q'，p、q的目标权值分别为$w_obj_p,c_obj_p,w_obj_q,c_obj_q$。若调整前误差$\varepsilon_{前}=|w_p-w_obj_p|+|c_p-c_obj_p|+|w_q-w_obj_q|+|c_q-c_obj_q|$，调整后$\varepsilon_{后}$计算方法类似。则$gain(v,p,q)=\varepsilon_{前}-\varepsilon_{后}$，若$gain(v,p,q)$大于零，则进行调整。

另外精化中两个重要参数可根据命题直接得到，即$\alpha=\frac{\sum w_i^{(i)}}{\sum s_i}=\frac{\sum w_i^{(i+1)}}{\sum s_i}$，$\beta=\frac{\sum d_i^{(i)}}{\sum c_i}\approx\frac{\sum d_i^{(i+1)}}{\sum c_i}$，虽然$\sum d_i^{(i+1)}$和$\sum d_i^{(i)}$有差异，但总体变化不大，因为这部分主要是节点在邻接块的调整，产生新的融合点较少，不会使得总边权发生很大变化。

算法 3 节点和边精化算法 node_edge_refine_algorithm()

输入：$G^{(i+1)}=(V^{(i+1)},E^{(i+1)},W^{(i+1)},D^{(i+1)})$，$Map^{(i)}$，$Map'$，系统图 FAG=(P,L,S,C)，任务图和系统图节点权之比 α，边权之比 β，误差 $\varepsilon_\alpha,\varepsilon_\beta$

输出：$G^{(i)}=(V^{(i)},E^{(i)},W^{(i)},D^{(i)})$，$Map''$

$G^{(i)}$←reconfig($Map^{(i)}$,$G^{(i+1)}$);//用 $Map^{(i)}$和 $G^{(i+1)}$还原为 $G^{(i)}$

Map''=remap($Map'^{(i)}$,$Map^{(i)}$)//利用重构映射 Map'和 $Map^{(i)}$构造 Map''初始值

while ∃p∈P is unrefined do //精化 $G^{(i)}$的重构映射 $Map'^{(i)}$

p←get_unrefined_partition ();//得到未精化的顶点 p

$w_p=\sum w_j, c_p=\sum$edge_cuts(v_j);//p 的实际权值等于映射到 p 的所有节点 v_j 的权和

$w\_obj_p=\alpha * s_p$, $c_obj_p=\beta\times$edge_cuts(p);//p 的目标权值

while ($|w_p-w\_obj_p|)>\varepsilon_\alpha \vee |c_p-c\_obj_p|>\varepsilon_\beta$)//实际权值不在误差范围之内

Q←get_unrefined_adj (p)

$w_q=\sum w_j, c_q=\sum$edge_cuts(v_j);//q∈Q 的实际权值

$w\_obj_q=\alpha * s_q$, $c_obj_q=\beta\times$edge_cuts(q);//q 的目标权值

$q=arg_q(\max(|w_q-w\_obj|+|c_q-c\_obj|))\wedge q\in Q$;//选择误差最大的邻接处理器

$v=arg_v(\max(gain(v,p,q)))\wedge v\in$BoundaryVeterx(q);//边界点 v 满足收益最大

modify w_p,w_q,c_p,c_q;

Map''←$Map''+\{(v,p)|(v,q)\in Map''\}$;//将(v,q)重构映射改为(v,p)

Endwhile

set p is refined;

endwhile

其中，adjpartition(p)返回 p 的邻接重构块，Boundary - Vertex(p)返回 p 的边界点。在精化过程中，我们还考虑了边界点选择时边权的条件。另外在还原过程中，若$G^{(i+1)}$对应 P 部分在$G^{(i)}$中有匹配的节点，则在还原过程中，匹配节点也有同样的对应 p 重构。至此，经过一步步精化还原，最终达到了重构目标。

4.4 算法分析

本文提出在异构系统中的相似驱动重构算法包括节点对融合过程、节点和边重构过程、重构精化过程 3 个过程，输入并行任务图以及体系结构图分布向量后，通过这 3 个过程便能得到所需的重构结果。

节点对融合过程的目的是合并源图节点及边，从而减少节点和边的个数，例如 20000 个节点、10000 条边的图，重构为 10 块，经过融合，一般只有 400 个左右的节点，这样既能减少重构时间也能减少精化还原次数。节点对融合过程的复杂度为 O(|E|)。节点和边重构过程的重构算法是广度优先搜索算法的一种变形，在搜索时考虑了节点权值和边权值与总目标权值比例，以保证重构 k 块。其复杂度也是 O(|E|)。精化还原过程还需要得到每个点的邻接点情况，所以复杂度也是 O(|E|)。因此整个算法的复杂度与边数存在线性关系。

5 重构算法的效果评估

我们实验所用的并行任务图均来自于格林威治大学 Chris Walshaw 教授所收集的一些开放的测试用图。表 1 描述了各图的属性，包括节点数、边数以及图的相关描述信息。可以从网址 http://staffweb. cms. gre. ac. uk/~c. walshaw/partition 获得详细信息。

表 1 测试用图属性

图名	节点数	边数	描述
add20	2395	7462	20-bit adder
data	2851	15093	Map data
3elt	4720	13722	2D Finite element mesh
4elt	15606	45878	2D Finite element mesh
uk	4824	6837	2D dual graph
bcsstk33	8738	291583	3D Stiffness matrix
whitaker3	9800	28989	2D Finite element mesh
crack	10240	30380	2D nodal graph
cs4	22499	43858	3D dual graph
cti	16840	48232	3D semi-structured graph
add32	4960	9462	32-bit adder
brack2	62631	366559	3D Finite element mesh

体系结构采用图 4，选择编号 p1－p8 处理器。首先通过实验给出 add20 在 3 种体系结构下重构图的邻接矩阵，如图 6 所示，每个矩阵代表 add20 在一种体系结构的重构图。计算误差、出边权和误差分别为 1. 89%，7. 143%，1. 28%，5. 60%，1. 72%，3. 84%。实验表明，算法设计满足了基本要求，各重构块的节点和出边权和在误差范围之内。

$$\begin{pmatrix}330&108&90&122&59&224&73&114\\108&332&45&113&206&136&83&47\\90&45&336&212&97&55&31&54\\122&113&212&365&165&98&51&54\\59&206&97&165&339&74&97&18\\224&136&55&98&74&340&99&131\\73&83&31&51&97&99&182&52\\114&47&54&54&18&131&52&171\end{pmatrix}$$

(a)

$$\begin{pmatrix}619&387&255&108&182&106&172&61\\387&438&142&69&175&82&153&69\\255&142&424&74&66&4&86&11\\108&69&74&193&78&0&51&3\\182&175&66&78&235&30&94&22\\106&82&4&0&30&199&78&44\\172&153&86&51&94&78&187&42\\61&69&11&3&22&44&42&100\end{pmatrix}$$

(b)

$$\begin{pmatrix}302&6&76&20&111&106&40&110\\6&316&84&56&95&95&148&55\\76&84&292&90&195&123&105&76\\20&56&90&321&87&134&180&74\\111&95&195&87&306&200&137&105\\106&95&195&87&306&200&137&105\\40&148&105&180&137&153&299&108\\110&55&76&74&105&95&108&283\end{pmatrix}$$

(c)

图 6 对应体系结构各自重构的并行任务图邻接矩阵

接着我们给出该算法在所有测试图中根据不同的体系结构所产生的计算权值误差、出边权和误差和割边值。为了描述方便，并行任务图记为 $G^{(0)}$，体系结构为 G'，重构并行任务图为 G，其中 $G \sim G'$，顶点标号相对应。则重构节点误差比定义为 $\Delta_w = \frac{\sum |\frac{w_i}{w_G} - \alpha_i|}{|V|}$，割边 $\Phi = |E_G| = \sum c_{ij}$，重构边权和误差比 $\Delta d = \frac{\sum |\frac{d_i}{d_G} - \beta_i|}{|V|}$。用这 3 个参数测试重构算法。另外传统的方法采用以割边最小化为划分目标的方法，体系结构采用图 4(c)。

并行任务图在不同情况下的重构效果如表 2 所列。

表 2　并行任务图在不同情况下的重构效果

$G^{(0)}$ \ G'	(a)			(b)			(c)			传统上方法		
	Δ_w	Φ	Δ_d	Δ_w	Φ	Δ_d	Δ_w	Φ	Δ_d	Δ_w	Φ	Δ_d
add20	2.25%	2644	7.143%	1.28%	2708	5.60%	1.72%	2605	3.84%	2.43%	2770	4.69%
data	3.90%	1554	7.94%	4.14%	1447	5.79%	5.24%	1590	5.48%	1.87%	2051	9.04%
3elt	3.25%	733	7.46%	1.72%	710	4.17%	2.19%	814	4.63%	4.65%	919	10.71%
4elt	4.29%	1429	9.48%	1.83%	1296	6.42%	2.18%	1418	4.83%	3.34%	1398	4.54%
uk	4.58%	188	9.83%	3.97%	187	8.7%	1.17%	182	7.51%	4.74%	202	9.46%
bcsstk33	2.32%	105097	7.19%	1.49%	113315	4.20%	2.27%	106792	3.91%	2.98%	136545	3.40%
whitaker3	4.64%	1578	5.59%	2.85%	1264	6.72%	3.48%	1369	7.04%	5.21%	1623	7.48%
crack	1.29%	1629	6.05%	3.50%	1403	6.92%	4.22%	1512	4.33%	3.93%	1811	7.74%
cs4	1.68%	4755	8.10%	2.12%	4242	4.16%	1.82%	4306	2.16%	1.36%	5944	3.50%
cti	2.07%	5828	4.28%	1.49%	6889	4.24%	1.67%	6591	4.06%	1.22%	8068	5.52%
add32	4.05%	212	9.6%	7.60%	185	8.56%	4.14%	233	6.35%	2.18%	244	9.9%
brack2	4.64%	26852	9.26%	3.88%	23559	6.07%	1.32%	23503	4.71%	1.78%	34347	3.50%

实验结果表明，计算误差和通信误差分别在 5%和 10%之内。不同的体系结构并行任务图重构的误差相差比较大，这与并行任务图的结构有关，若并行任务图和体系结构图的特征基本相同，则误差较小。若误差较小，则并行任务图和体系结构图相似，从而异构计算中并行任务和体系结构匹配，因而达到执行时间最小化。一般来说，计算节点的误差要小于通信误差，这跟算法设计有关，通信误差以出边权和来度量。在不同的并行任务图中，重构误差往往不同，若将计算误差限制在 2%，通信误差限制在 5%，add20 在体系结构(c)下的重构能满足要求，3elt 在体系结构(b)下的重构能满足要求(见图 2)。这说明并行任务图的重构结果与体系结构紧密相连。与传统方法相比较，本文所给的相似驱动的重构算法也降低了割边值。通过相似驱动的重构算法，使得在任务划分阶段就保证了异构计算的并行任务的异构特征能适应体系结构特征，从而在任务执行时可发挥异构机器的特长，各尽所能，最终实现性能最优化。

结束语　本文提出了相似驱动细粒度重构算法，其本质是以重构为手段，使并行任务图和体系结构相似，从而实现异构匹配，最终使异构系统能发挥优势，各尽所能，实现按需分配，从而实现系统高效能。这种思想在本质上为实现系统和并行任务的最大匹配提供了可能。其思想可以应用在异构计算中相关并行任务调度的负载均衡及通信均衡，特别是为研究体系结构处理器间的通信不均衡提供了很好的方法，也为研究并行任务和体系结构结合进行划分和调度做了新思路。这种重构方法为在异构环境中调度提供了很好的准备工作。但是目前在如何实现量和结构完全相似理论方面仍然有待改进，这也是我们今后研究的重点。

参 考 文 献

[1] Kernighan B W, Lin S. An efficient heuristic for partitioning graphs[J]. Bell System Technology Journal, 1970, 49: 291-308

[2] Hendrickson B, Leland R. A multilevel algorithm for partitioning graphs[C]// Proceedings of the ACM/IEEE Supercomputing Conference. 1995: 626-657

[3] Walshaw C, Cross M. Multilevel mesh partitioning for heterogeneous communication networks[J]. Future Generation Computer System, 2001, 17(5): 601-623

[4] Schloegel K, Karypis G. Wavefront diffusion and LMSR : algorithms for dynamic repartitioning of adaptive meshes. Multilevel k-way partitioning scheme for irregular graphs[J]. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(5): 451-466

[5] Need C, Wehn N. Designing efficient irregular networks for heterogeneous system-on-chip[J]. Journal of System Architecture, 2008, 54(2): 384-396

[6] Bambha N K, Bhattacharyya S S. Joint application mapping/interconnect synthesis techniques for embedded chip-scale mutliprocessors[J]. IEEE transaction on parallel and distributed systems, 2005, 16(8): 99-112

[7] Kramer R, Gupta R, Soffa M L. The combining DAG: a technique for parallel data flow analysis[J]. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(8): 805-813

[8] 沈轶伟，曾国荪. 异构计算中一种图的非均衡划分算法[J]. 计算机科学，2006，33(6)：260

[9] Selvakkumaran N, George K. Multiobjective hypergraph partitioning algorithms for cut and maximum subdomain-degree minimization[J]. IEEE Transactions on Computer Aided Design of Intergrated Circuits and System, 2006, 25(3): 504-512

[10] Karypis G, Kumar V. Multilevel k-way partitioning scheme for irregular graphs[J]. Journal of Parallel Distribution Computer, 1998, 48(1): 96

[11] Verbauwhede I, Schaumont P. The happy marriage of architecture and application in next-generation reconfigurable systems [C]// Proceedings of the 1st conference on computing frontiers. Ischia, Italy, 2004: 363-376

[12] Moulitsas I, Karypis G. Architecture aware partitioning algorithms[C]// Proceedings of the 8th international conference on Algorithms and Architectures for Parallel Processing. Agia Napa, Cyprus. , 2008: 42-53