分类号 TP316.4
UDC
密级
编号

中南大学

CENTRAL SOUTH UNIVERSITY

硕士学位论文

论文题目 基于DAG模型的高效并行任务调度算法研究

学科、专业 计算机应用技术

研究生姓名 华强胜

导师 陈志刚 教授

MS THESIS

Efficient Algorithms for Scheduling of Parallel Tasks based on the DAG Model

Speciality: Computer Application Technology

Master Degree Candidate: Hua Qiang-Sheng

Supervisor: Prof.Chen Zhi-Gang

School of Information Science & Engineering

Central South University

ChangSha Hunan P.R.C

原 创 性 声 明

本人声明，所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。尽我所知，除了论文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得中南大学或其他单位的学位或证书而使用过的材料。与我共同工作的同志对本研究所作的贡献均已在论文中作了明确的说明。

作者签名：华强胜　　日期：2004年4月28日

关于学位论文使用授权说明

本人了解中南大学有关保留、使用学位论文的规定，即：学校有权保留学位论文，允许学位论文被查阅和借阅；学校可以公布学位论文的全部或部分内容，可以采用复印、缩印或其它手段保存学位论文；学校可根据国家或湖南省有关部门规定送交学位论文。

作者签名：华强胜　导师签名：陈志刚　日期：2004年4月28日

摘 要

在当今的网络并行计算环境中，并行任务调度已经成为并行处理和高性能计算领域中极其重要的关键技术，不恰当的调度甚至会抵消任务并行化所带来的收益。基于此，本文研究了一种以带节点权值和边权值的有向无环图来表示并行任务的 DAG 调度问题。一般情况下，这种 DAG 调度是个 NP 完全问题。

国内外的研究学者在该领域进行了广泛而深入的研究。但是随着网络硬件技术和处理器技术的飞速发展，该领域仍然存在不少亟待解决的关键问题。在这种新形势下，本文主要研究了其中的三个关键技术，它们是：其一，针对非线性聚簇下怎样高效调度独立任务问题，本文提出了一种基于最大并行度的独立任务调度算法 MPD；其二，针对某些调度算法虽然性能良好但复杂度高，或者某些调度算法虽然复杂度低但性能不佳的问题，本文提出了一种在性能和复杂度间进行折衷的 DAG 调度算法 EZDCP，从而使得 DAG 调度算法更加实用；最后本文还提出了一个较为系统性的 DAG 粒度理论，并且根据 fork 和 join 图，本文用数学方法严格证明了细粒度下非线性聚簇要优于线性聚簇。该粒度理论对指导调度算法的选取和进行调度算法的性能评估起着重要的作用。通过对一些基准测试 DAG 图的实例分析，本文的两个调度算法性能要优于已有的同类 DAG 调度算法。

关键词 DAG 调度，非线性聚簇，线性聚簇，独立任务，DAG 粒度

ABSTRACT

In the network and parallel computing environment, task scheduling has become a nontrivial problem in the parallel processing and high performance computing area. An improper scheduling method will counteract the benefits from the parallelism of the tasks. So this paper focuses on the parallel task scheduling problem which is represented by the node-labeled and edge-labeled Directed Acyclic Graphs(DAGs). The DAG scheduling problem has shown to be NP complete in general.

The DAG scheduling problem has drawn many researchers' attention in the past two decades. Many outstanding achievements have been made in this area. But with the rapid development of the network hardware technology and the processor technology, there still exist lots of unsolved problems in the DAG scheduling area. This paper mainly aims to solve the three problems. Firstly, in the nonlinearly clustering method, the paper proposes an efficient independent task scheduling algorithm named MPD which overcomes the shortcomings of the current peer algorithms. The MPD algorithm means it is based on the maximized parallelism degree among the independent tasks; Secondly, because some current DAG scheduling algorithms posses a good performance while with a high complexity, and some algorithms posses a low complexity while with an incomparable performance, the paper then presents a new efficient algorithm called EZDCP. This algorithm has a comparable performance while with a low complexity thus makes it very attractive in practice; Thirdly, A systematic granularity theory of the DAG scheduling is given in this paper. Based on the fork and join graphs, it has been proved that the nonlinearly clustering is better than linearly clustering for the fine grain DAGs which is defined by the paper. The granularity theory can be very effective in choosing the appropriate scheduling algorithms and in performance evaluation. The two proposed algorithms have been shown superiority over the previous algorithms with the same assumption through the peer set graphs which have been introduced as useful benchmark graphs.

KEY WORDS DAG scheduling, nonlinearly clustering, linearly clustering, independent tasks, DAG granularity

目 录

第一章 绪论

1.1 本文的研究背景

网络并行计算环境下的调度问题是一类组合优化问题,在计算机及通信等许多领域有着广泛的应用,在理论上又与算法设计、复杂性理论关系密切,因而引起了许多学者的研究兴趣[1-3]。根据实际应用中的不同要求,存在很多种调度模型,目前被研究过的就有几百种[4]。Graham 等人在 1979 年给出了一个任务调度模型的标准描述[5],即 α | β | γ，其中 α 代表机器环境描述， β 代表模型限制条件，γ 代表优化目标或目标函数。下面分别对这三个参数作个简单介绍，具体分析请参考文[2]。

对机器环境 α 而言，它分三种情况，分别是:

(1) 单机模型 (single machine);

(2) 并行机模型 (parallel machine)，该模型又可细分为三类 (分别以 P, Q, R 表示)，一是同构并行机模型(identical parallel machine),二是一致相关并行机模型(uniformly related machine),三是非相关并行机模型(unrelated parallel machine);

(3) shop environment 模型，它又可以分为 open shop, job shop environment 和 flow shop 三种情形 (分别以 O, F, J 表示)。

模型限制条件 β 有以下几种:

(1)属于抢占式还是非抢占式(pmtn);

(2)任务之间有无前驱后继关系(prec);

(3)各任务有无最早开始时间(release date, R_j);

(4)各任务有无最迟结束时间(due date, D_j)。

目标函数 γ 包括:

(1) 最小化并行任务的最早结束时间 [C_{max} or makespan];

(2) 最小化任务的平均(加权)完成时间 [$\sum C_j$ or $\sum W_j C_j$];

(3) 最小化 L_{max}[$L_{max}=\max(L_j), L_j=C_j-D_j$, C_j=completion time of task j];

(4) 最小化 $\sum U_j$ 或者 $\sum W_j U_j$ [if $C_j<=D_j$ then $U_j=0$ else $U_j=1$]

在以上不同组合所表示的调度模型中，最早被研究的问题是同构并行机(identical parallel machine)上最小化停机时间(makespan)问题。除了极少数情况下这种调度问题有最优解外，其它绝大部分都属于 NPC(NP 完全)问题，所以不存在一个多项式时间的解决方案，除非 P=NP[2]。

虽然迄今为止已经存在众多的调度问题,但随着并行处理领域的飞速发展以

及现实需求,近年来依然有更为实际的新模型被研究者们陆续提出,并加以研究。这其中针对以有向无环图 DAG (Directed Acyclic Graph)来表示的并行任务在多处理机上进行调度的研究在近 20 年得到了迅速发展。这种模型的机器环境是完全互连的同构并行处理机，并且可以假定处理机数目无限；其限制条件包括：任务执行具有非抢占性,即处理机只有在执行完某个任务之后才能处理另外一个任务，另外这些任务之间具有前驱后继的依赖关系，某个子任务只有在其所有的父节点任务处理完毕后才能开始执行；该模型的调度目标就是要使得整个 DAG 图的调度长度最短。用 $\alpha|\beta|\gamma$ 来表示这种 DAG 调度模型就是 $Pm|prec|C_{max}$。早期的这种 DAG 调度模型只引入了各个任务的执行时间（用每个节点权值表示），而并没有考虑任务之间的通信时间。但即使是在这种情况下，这种 DAG 调度仍然被证明是 NP 完全问题[6]。Oregon State University 的 B. Kruatrachue 博士在 1988 年提出了包含计算通信延迟在内的新的 DAG 并行任务调度模型（Node-labeled and Edge-labeled DAG）[7]，并且已经证明这种一般的 DAG 调度仍然是个 NP 完全问题。本论文的研究对象就是这种节点和边均带权值的 DAG 并行任务模型。

为了使得这种 DAG 图的调度长度最短，需要考虑两个方面的目标：一是要尽量最大并行化执行各个任务(Maximize the parallelism)，另一方面要最小化各个任务间的通信延迟(Minimize the communication latency)。但如果并行执行的任务太多,任务间的通信延迟必然增大,反之如果要最小化任务间的通信延迟,那必然会降低任务的并行化程度。所以这种 DAG 调度问题就是在这两者之间寻求一个最优或近优(optimal or near optimal)均衡点，也就是各种文献当中所描述的 Max-Min 问题[7]。既然一般情况下找到一个最优的 DAG 调度算法是个 NP 完全问题，故这二十年来国内外的研究学者都采用了各种各样的启发式调度算法(Heuristics)[8]。譬如分支定界法（branch-and-bound），整数规划(interger programming),局部搜索（local search），图论（graph theory），遗传算法(genetic algorithm)和进化方法(evolutionary methods)等等。

1.2 国内外研究动态

在上一小节当中我们提到了 DAG 调度模型也分成两种情况,其一就是不考虑任务间的通信延迟，其二就是把任务间的通信时间作为一个重要因素来进行 DAG 调度。在共享存储的并行计算模型上，进行启发式任务调度时一般不需要考虑任务通信延迟，但近些年来，由于网络和计算机技术的飞速发展，在网络并行计算环境中（譬如工作站网络 Now 中）任务间的通信延迟越来越成为一个不可忽略的现实影响因素。文[7]早在 1988 年就提出了包含计算通信延迟在内的新的 DAG 并

行任务调度模型，并且已经证明一般的 DAG 调度是个 NP 完全问题[8]，也就是说它的最优解不能在多项式时间内解决，除非 P=NP。目前也仅发现在以下三种情况下才有多项式的最优解：

(1)节点权值为单元时间，任务图为树型结构，调度到 N 个处理机上[9]；

(2)节点权值为单元时间，任务图形状任意，调度到 2 个处理机上[10]；

(3)节点权值为单元时间，任务图为“Interval-Order”，调度到 N 个处理机上[11]。并且以上三种情况均没有考虑通信延迟，即假设边权值为 0。

最近 Ali 和 El-Rewini 也证明，针对“Interval-ordered”DAG 图，如果其所有的边权值都等于节点权值，那么这种 DAG 调度也可以在多项式时间内求出最优解[12]。

在实际应用中，对于那些任意结构的 DAG 图（arbitrary structured），如果节点权值和边权值可取任意值，那么在这种一般情况下，用启发式方法来求解 DAG 调度问题就成为一个首要选择。几十年来，国内外的学者们已经对启发式 DAG 调度进行了广泛而深入的不懈研究，产生了很多的 DAG 调度算法，但归纳起来，这些各个时期的算法可以分成如下四种[13]：其一是表调度算法(List Scheduling)，其二是聚簇调度算法(Clustering)，其三是基于任务复制的调度算法(Task Duplication)，其四是遗传算法(Genetic Algorithm)和随机化搜索技术方法(Randomized Search Techniques)。这里表调度算法有两个阶段，其一是确定各任务的优先级阶段，其二是处理机的选择阶段，从而选择合适处理机来运行当前优先级最高的就绪任务。表调度算法主要是针对有限完全互连的处理机网络，并且现有各种各样的表调度算法的调度结果显示，它比其它各类启发式算法更加切合实际，并且复杂度低而且性能较好。但是表调度算法在某些情况下可能会产生某些反常现象，譬如改变该算法中某些参数，如增加处理机数目，减少各任务执行时间或者删除某些任务间的偏序关系时，整个并行任务的执行时间不但不会减少反而还会增加[14]。然而理论验证证明所有这些情况下，这些整个任务的执行时间与最优算法相比存在一个比率小于 2 的上界(Upper Bound)[14]。目前基于表调度的算法有 MCP, ETF, DLS, LAST, HLFET 以及 ISH[15-20]。基于聚簇的算法因为是事先假定把各任务映射到无限数目处理机上，所以和一般算法比，它需要另外一个合并步骤，即把由任务图划分的簇进行再合并，直到和当前可用的处理机数目相同时为止。它的每一个簇中的所有任务只能映射到同一个处理机上，然后才在该处理机上指定簇中各个任务的执行序列。必须指出，同簇中的任务之间的通信延迟可以忽略不计。聚簇算法可以分成两大类，线性聚簇和非线性聚簇。如果一个簇中至少含有两个互相独立的任务（任务之间不存在依赖关系），那么这便是一个非线性聚簇，反之便为线性聚簇。Tao Yang 曾证明对于其所定义的粗

粒度 DAG，线性聚簇要优于非线性聚簇[21]。目前的聚簇算法有 EZ, LC,MD, DSC,DCP,EZDCP 和 MPD[15,21-26]。基于任务复制的调度算法主要是为了消除不同处理机间任务间通信时延而设计。它的应用对象也主要为无限数目的同构处理机网络。因为这种算法不仅要在其它处理机上复制任务，还要复制大量父亲节点任务需要的数据，所以它比一般的算法需要大得多的复杂度，故一般仅局限于理论研究，较少应用于实际。基于任务复制的算法有 DSH，CPFD, Bottom-UP-Down Duplication Heuristic,TCSD 和 Duplication First and Reduction Next[20,27-31]。随机化搜索技术方法近些年才流行起来，它主要结合由前面的搜索结果所获取的知识和一些随机特征来产生新的调度结果。用随机化搜索技术方法进行调度的结果比较好，但是它们的调度时间要比其它类的算法高的多。另外譬如遗传算法(Genetic Algorithms)[8]，它针对不同的 DAG 图，其最优控制参数也不相同，所以在实际调度过程中必须随时变更这些控制参数，从而造成不便。随机化搜索技术方法包括遗传算法，模拟退火(Simulated Annealing)以及局部搜索技术(Local Search Technique)[32-34]等。

国内对 DAG 调度的研究起步比较晚，而且也都局限于国外研究的调度算法所属范围之内，基本上还是处于跟踪发展阶段。譬如文[35]提出了一种名为 BDCP 的调度算法，它也是一种基于动态关键路径技术的表调度算法；文[36]针对在非全互连工作站网络环境中通信之间不能并行执行的问题,提出了一个基于任务复制的静态调度算法 TSA_FJ；文[37]提出了基于 DAG 的、优化的分代任务调度算法 OGS，该算法综合考虑机器就绪时间和每个任务全部先导的完成时间，从而减少全部参与调度节点的空闲时间，达到优化调度长度(makespan)的目的；文[38]是 DAG 调度在网格计算中的应用，它提出了一种基于有向无环图的两层资源监测系统(DTGMS)；文[39]提出了一个基于 DAG 图的指令调度优化算法；文[40]针对人们往往只注重找到一个调度路径最短的算法，却忽略了要节省处理器的问题，提出了一个基于任务复制的算法 TSA-OT；文[41]提出了在处理机网络为超立方体时的基于 LU 分解的任务图调度方法。文[56-58]针对那些传统的不考虑任务通信的调度算法进行了分类，并且提出了一些改进的调度算法。以上文献都是从各个侧面提出了现有算法的不足从而加以改进并应用到实际领域中去。

在 DAG 调度领域中，作者也相继发表了 3 篇英文论文[25,26,42]。文[25]综合了动态关键路径思想和 EZ 算法的优势，提出了一种 EZDCP 算法。该算法时间复杂度相当于 EZ 算法，但其性能却能够和 DSC 算法媲美。文[26]提出了一种新的非线性聚簇下的独立任务调度算法 MPD。文[42]提出了一种新的细粒度 DAG 图，并证明了一条相关定理。下面将进一步阐述本文的这些研究内容及研究意义。

1.3 本文的研究内容及研究意义

本文的研究内容主要包含四个方面。其一，在DAG图的粒度划分方面，Tao Yang定义了一种粗粒度（coarse grain）DAG图[43]，也就是所有任务的执行时间均大于任务间的通信时间，并且证明了对于这种粗粒度DAG，线性聚簇要优于非线性聚簇。不仅如此，Tao Yang还给出了线性聚簇DAG的调度长度的上下界分析（近优比）。既然粗粒度DAG有如此良好的性质，本文将重点讨论是否存在某种细粒度DAG图，并且是否也存在某种相对应的调度性质。其二，在非线性聚簇调度算法中，目前针对独立任务的调度基本上都是基于节点的 tlevel 值或blevel值或者这两者的综合信息来进行。譬如先调度tlevel值较小的任务，或者优先调度blevel值较大的任务，再或者优先调度blevel和tlevel的差值较大的任务等等[8]。但在任意形状的DAG图中，这些调度策略并不能满足调度目标（也就是使得DAG调度长短最短），实际上这些策略有时候会大大增加任务调度长度[26]。基于此，本文将讨论一种新的能够克服现有调度策略缺陷的独立任务调度算法。其三，由于目前的DAG调度算法中存在这样一种现象，那就是或者算法性能很好，但是复杂度很高，或者算法复杂度低，但性能不佳。针对此，本文将研究一种在复杂度和性能之间进行折衷的调度算法。最后本文还将讨论一种评价DAG调度算法性能的基准测试DAG图“Peer Set Graphs”。

在并行处理领域中，针对以有向无环图 DAG (Directed Acyclic Graph)来表示的并行任务在多处理机上进行处理的研究已经由来已久[8]。这里带权值的DAG主要定义了各个子任务间的前驱后继依赖关系和每个子任务的运行时间。但是在早期的研究中，它并没有包含每个任务之间的通信时间。而最近二十余年来，随着网络和处理器性能的迅速提高，高性能计算领域进入了一个全新的阶段，从而也给并行调度领域中的启发式DAG调度带来了新的问题，此时的并行调度不仅要考虑任务之间的通信代价还要考虑调度环境的异构性带来的影响。而现有各种各样的启发式DAG调度算法在这种新形势下存在这样或者那样的不足，譬如链路竞争和处理机的异构性以及网络拓扑结构的松散易变性都给怎样产生DAG图，怎样选取调度策略带来了挑战。除此之外，传统DAG调度领域在理论和实际中还存在一些诸如怎样进行DAG粒度的划分，怎样验证各种启发式算法的近似比以及怎样结合实际综合权衡 DAG 调度算法的各种目标而设计出低复杂度高性能的可扩展DAG调度算法等问题。本论文的研究就是要着力解决这些关键技术，从而在这种新形势下提出DAG调度领域中的新理论，新方法和新技术，从而为组合优化领域中设计高效的启发式调度算法和网络并行计算环境中的静态和动态负载均衡提供理论和技术支持。

此外，由于网络计算环境日益成为一种不受地域限制的廉价的超级计算环

境，网络分布式计算是当今高性能计算领域中一个十分活跃的研究方向。其中的任务调度又是该领域中极其重要的关键技术。任务调度策略的选取将会对网络计算的效率和性能产生重要影响，不恰当的调度甚至会抵消任务并行化所带来的收益。本文就针对网络分布式计算环境产生的一些新问题，以及当前调度算法中存在的一些缺陷，研究 DAG 调度中的一些关键技术。该研究具有重要的研究价值和实际意义，譬如求解这样一类 NP 难问题的高效率的近似算法在大型软件项目开发，分布式计算，车间作业排序，资源约束排序以及光纤网络通信中都具有实际应用价值。

1.4 本文的内容组织与安排

本论文分为七大部分。第一部分为绪论，主要介绍工作的研究背景和国内外研究现状分析。第二章将详细介绍 DAG 调度模型以及相关任务调度算法。第三章将介绍并行任务的粒度划分以及相关定理。第四章将介绍 DAG 调度算法的评价目标以及评价 DAG 调度性能的一些基准测试 DAG 图。第五章和第六章将主要介绍本文提出的两个调度算法 MPD 和 EZDCP。最后是本文的工作总结以及一些研究展望。

第二章 基于 DAG 模型的并行任务调度算法

2.1 DAG 调度模型

一个并行任务可以用一个节点和边均带权值的有向无环图来表示。它的详细定义如下。

定义 1: 一个节点和边均带权值的 DAG(node-labeled and edge-labeled)图是一个四元组 G=(V,E,W,C),其中 $V=(V_1,V_2,\ldots,V_{|V|})$ 表示任务的集合，$|V|$ 表示任务的个数，$E=\{e_{ij}|v_i,v_j\in V\}\subseteq V\times V$，表示通信边的集合，同样 $|E|$ 用来表示共有多少条边。集合 C 表示边上通信量的集合，W 表示各个任务的本身执行时间的集合。值 $C_{ij}\in C$ 表示由边 $e_{ij}\in E$ 所引起的通信代价，如果这两个任务被映射到同一个处理机上，那么 C_{ij} 就变为 0。对 DAG 中的每个节点，有 $C(V_i, V_i)\equiv 0$。值 $W_i\in W$ 表示节点 $V_i\in V$ 的执行代价。我们称 $PRED(V_i)$ 为节点 V_i 的直接前驱节点集合，$SUCC(V_i)$为结 V_i 的直接后继节点集合。如果 $PRED(V_i)=\Phi$,那么节点 V_i 便称作入节点（entry node），相对应的，如果 $SUCC(V_i)=\Phi$,那么节点 V_i 便被称作出节点(exit node)。如果两个节点间没有任何存在依赖关系的路径，那么这两个节点被称作独立节点。

图 2-1 所示便为一个 DAG 图。DAG 图以及相关调度算法中用到的一些符号释义请参见表 2-1。其中 DAG 图有时候也称为宏数据流图(macro data-flow graph)，这是因为某个任务只有在接收到所有的输入之后才开始立即执行，并且只有在完成所有的输出之后才终止执行。

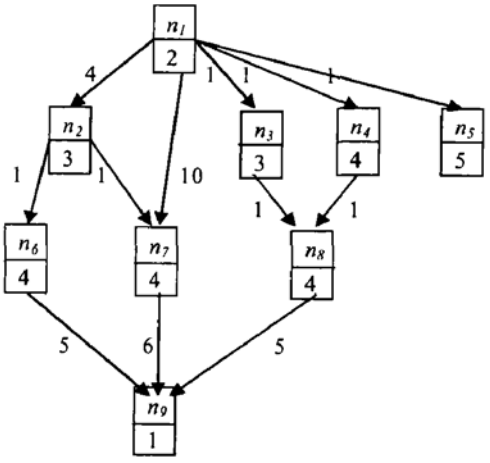


图 2-1 一个 DAG 图实例

表 2-1 DAG调度算法中一些名词释义

符号	定义
DAG	有向无环图，本文中一般指节点和边均带权值的有向无环任务图
v_i, n_i	DAG任务图中某个节点（或称任务）的编号
$w(n_i)$, w_i	某个节点n_i的任务计算代价
(n_i, n_j), e_{ij}	一条从n_i到n_j的边
$c(n_i, n_j)$, c_{ij}	(n_i, n_j)这条边上的通信代价
$\|V\|$	DAG图的节点数（任务数）
$\|E\|$	DAG图的边数目
CP	DAG图中的关键路径(Critical Path)
DCP	DAG图中的动态关键路径（Dynamic Critical Path）
tlevel	从入节点到该节点的最长路径长度，不包括该节点本身权值
blevel	从该节点到出节点的最长路径长度，包括该节点本身权值
sbl	不包含通信代价的blevel值
televel	包含所在节点计算代价的tlevel值
belevel	不包含所在节点计算代价的blevel值
UNC	无限数目虚拟处理机(Unbounded Number of Clusters)
BNP	有限数目处理机（Bounded Number of Processors）
TDB	基于任务复制的调度算法(Task-Duplication Based)
MPD	基于最大并行度的独立任务调度算法(Maximized Parallelism Degree)
EZDCP	基于边消除和动态关键路径的静态任务调度算法(Edge-zeroing and Dynamic Critical Path)
APN	任意处理机网络(Arbitrary Processor Network)

2.2 DAG图的生成和调度目标系统

一个并行程序通常是通过一个DAG来模型化的。尽管程序中的循环不能通过DAG模型来显示地表示，但可以通过循环分解[50-51](loop-unraveling)技术来挖掘循环中的数据流计算并行性，并将循环分解成多个任务。其主要思想是循环中的所有迭代是同时启动的，不同迭代中的操作在其输入数据准备就绪时就可以执行了。而且，对于一大类数据流计算问题和大多数的数值算法（例如矩阵乘）来说，程序中的条件分支和不确定性是非常少的。因此DAG模型能够精确地表示这些应用程序，从而使得调度技术能够得到应用。并且在很多数值应用程序中，

譬如高斯消去(Gaussian Elimination)和快速傅立叶变换(FFT)，循环边界在编译时都是知道的。这样，循环中的一个或者多个迭代就可以被确定性地封装在一个任务当中，因而该任务就可以被表示为DAG中的一个节点。节点和边的权值通常可以通过诸如数值操作，存储器访问操作和消息传递原语等操作来估计获得。

在DAG调度中，目标系统被假定为由处理单元（PE）组成的网络，每个PE由一个处理机和一个本地存储器单元组成，因此PE不是共享内存的，并且通信是完全依赖于消息传递。这样的PE可以是工作站甚至是个人计算机。处理机可以为同构的或者是异构的。处理机的异构性是说它们有不同的速度或处理能力。但是假定并行任务的每个模块都可以在任一处理机上执行，即使在不同处理机的完成时间可能是不同的。PE是通过具有某种拓扑结构的互联网络互相连结的。拓扑结构可以是全互连或者由特殊结构构成，譬如超立方体等。

2.3 基于DAG模型的任务调度算法

自从上个世纪60年代以来，国内外的学者就开始研究怎样调度DAG任务图，并且40多年内诞生了很多调度算法。参考文[8]，本文对DAG调度算法做了一个详细分类，每个子类里面都给出了相对应的调度算法，并且附有参考文献，参见图2-2。

根据所调度的DAG图是任意形状还是特殊结构，该图最上一层把调度算法分成两大类。在早期的研究中，人们对任务图都做了一些假设，譬如他们考虑的都是些树形或者fork-join图等。但是一般而言，一些并行程序表示成DAG图后，其形状并没有这么规则，因此后期的一些调度算法都致力于处理那些任意结构DAG图。这种调度算法又可以进一步划分成两大类。其中有些调度算法假设DAG图中的所有任务计算代价均为单元值，而另外一部分又假设任务的计算时间任意。对于那些任务计算代价均为单元值的调度算法，它们都不考虑任务间的通讯代价，而对于后者，它们又可以分为两大类，分别是包含任务间的通讯代价和不包含任务间的通讯开销。

对于那种共享存储的并行计算模型而言，任务间的通讯代价可以忽略不计。但在网络并行计算环境中，通信延迟成为影响计算性能的一个至关重要的因素，所以近期的DAG调度算法又主要集中在这个研究领域。

在考虑处理机数目的基础上，DAG调度算法又可以分为两大类，其一是假设处理机数目无限，其二从实际角度出发只考虑有限数目处理机。前者本文称为UNC调度算法(Unbounded Number of Clusters)，或者称无限虚拟处理机调度算法，而后者就称为BNP算法（Bounded Number of Processors），也即有限数目

处理机算法。这两类调度算法都假设处理机完全互连(fully-connected)，不考虑链路竞争(link contention)，也不考虑采用什么路由策略(routing strategies)。UNC 调度算法中所采用的方法也被称为聚簇法(Clustering)。在调度的开始阶段，每个DAG图的任务节点都被认为是一个簇。接下来就是一个簇的合并步骤，当然这些合并要以不增加整个任务的完成时间为前提。最后当没有任何簇可以合并时，合并步骤才得以终止。UNC算法所采用的一个原理就是：充分利用更多数目的处理机来尽量减少任务的调度长度(schedule length)。然而经过UNC算法所产生的聚簇还需要另外一个执行步骤，那就是把这些聚簇调度到实际的处理机中。这是因为现有的处理机个数要小于簇的个数。因此，聚簇算法最后的调度结果和簇的映射步骤息息相关。相比而言，BNP算法就不需要另外一个后续处理步骤。

如果并不是采用完全互连的处理机网络，而是采用诸如超立方体Hypercube等连接，那么就有可能存在链路竞争。这些调度算法称APN(Arbitrary Processor Network)方法。因此它们除了调度任务外，还需要把消息(message)调度到通信链路中(communication link)。这种消息的调度和不同网络拓扑中所采用的路由策略有很强的联系。这种应用环境给DAG调度带来了更多的挑战性。本文将不考虑这种APN情形。

那种基于任务复制的调度算法(TDB)也是假设处理机数目无限，但是为了减小调度长度，这种调度方法将把某些任务进行复制来调度到多个处理机上。该调度方法的原理就是通过在多个处理机上调度同一个任务的方法来减少任务间的通信延迟。不同的TDB调度策略会选择不同的任务节点进行复制，有的只选择那些直接前驱节点，而有的会选择复制所有的祖先节点。本文将重点研究处理机数目无限的UNC类的DAG调度方法。根据图2-2的启发式DAG调度方法的分类，我们还可以把它分成四大类。它们分别是表调度算法,聚簇调度算法，基于任务复制的调度算法，基于遗传算法和随机搜索技术的调度算法。下面分别予以阐述。

2.3.1 表调度算法

目前大部分的DAG调度算法都是基于一种所谓的表调度技术(list scheduling)[7,15-17,19]。表调度算法的一个基本思想就是通过给每个任务节点赋予优先级，从而确定一个调度序列，然后重复执行以下两步直到图中所有节点被调度：

(1)从调度序列中移除第一个节点；

(2)把该任务节点调度到可以最先执行该任务的处理机。

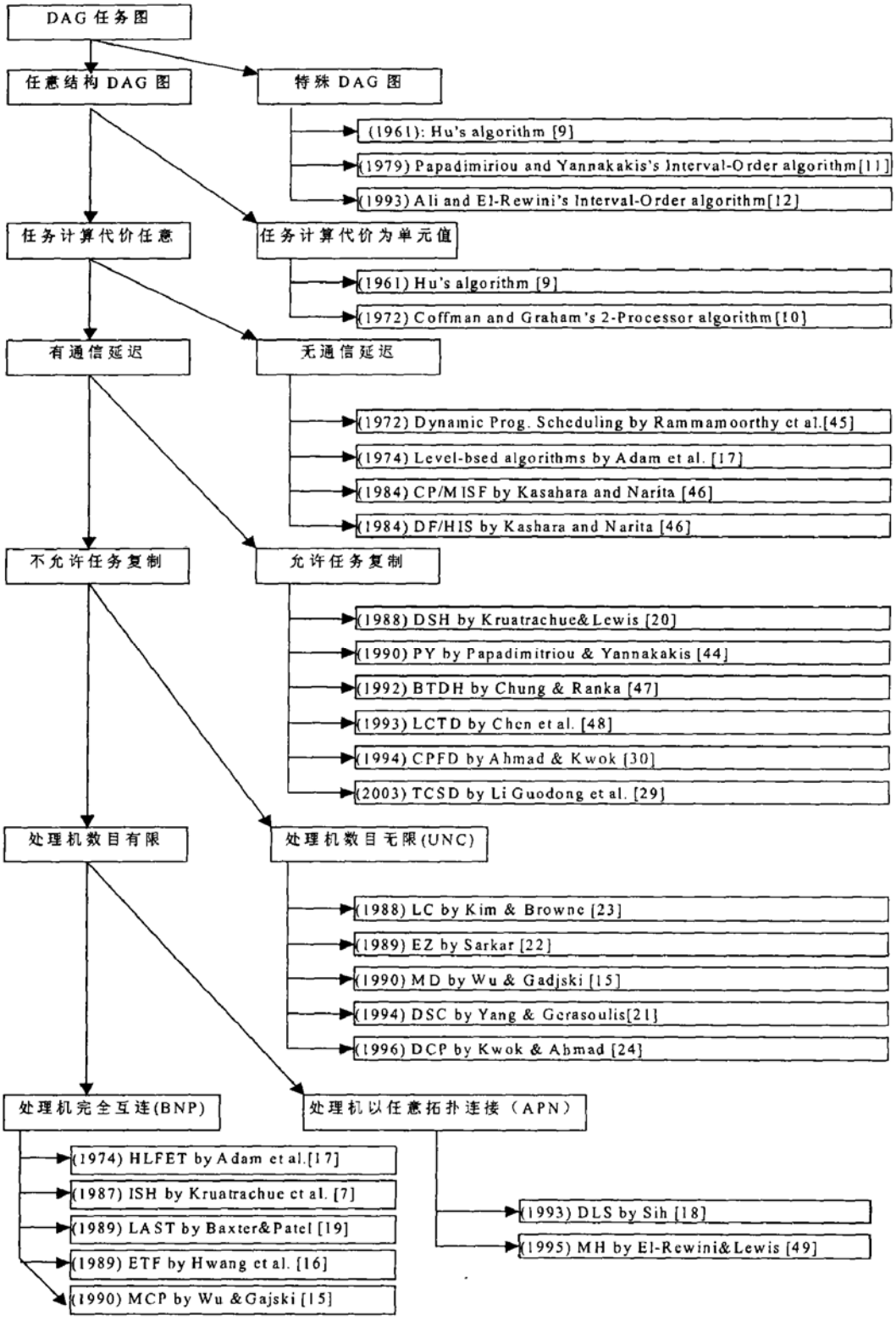


图 2-2 DAG 调度算法分类

由于在传统的表调度算法中，那些调度表在任务节点调度到处理机之前就已经确定好执行序列（即静态创建），而且更重要的是，这种调度表在任务调度过程中不再更改，这样可能在调度过程中造成那些本应优先执行的任务节点因其优

先级别低而迟迟不能执行的情况。针对此，近来人们提出了一种基于动态表调度(dynamic list scheduling)的方法。该方法在每次执行一个任务后，它就会重新计算所有未被调度节点的优先级别，然后重新对调度表进行排序。也即采用下面三个步骤：

(1)决定所有未被调度节点的新的优先级；

(2)从调度表中选择优先级最高的节点进行调度；

(3)把该任务节点调度到可以最先执行该任务的处理机。

一般来说这种动态表调度算法可以产生更短的调度长度，但是此方法也会大大增加算法的时间复杂度。

在给任务节点设置优先级时，有两个属性经常用到，那就是tlevel和blevel值。它们的定义如下；

定义2：在一个DAG图中，tlevel(V_i)是指从一个入节点到节点V_i的最长距离，它不包括节点V_i本身的计算量的值。相应的，blevel(V_i)是指从节点V_i到某个出节点的最长距离。因为在聚簇过程中，某些边的权值可能变成0，所以tlevel(V_i)和blevel(V_i)的值都是动态变化的。如果我们在计算blevel(V_i)值的时候不考虑通信边的代价在内，那么这样的blevel我们就称为静态blevel(static blevel)，简称为sbl。令入点的tlevel值为0，出点的blevel值为该出点的权值。

下面分别给出计算这两个属性的算法：

1) tlevel值的计算

(1)Construct a list of nodes in topological order. Call it TopList.

(2)For each node n_i in TopList do

(3)max=0

(4)For each parent n_x of n_i do

(5)If tlevel(n_x)+w(n_x)+c(n_x, n_i)>max then

(6)max=tlevel(n_x)+w(n_x)+c(n_x, n_i)

(7)endif

(8)endfor

(9)tlevel(n_i)=max

(10) endfor

2) blevel值的计算如下

(1)Construct a list of nodes in reversed topological order. Call it RevTopList.

(2)For each node ni in RevTopList do

(3)max=0

(4)for each child n_y of n_i do

(5)if $c(n_i, n_y)$+blevel(n_y)>max then

(6)max=$c(n_i, n_y)$+blevel(n_y)

(7)endif

(8)endfor

(9)blevel(n_i)=w(n_i)+max

(10)endfor

从上面两个算法的执行过程可以看出，它们的时间复杂度都是O(|V|+|E|)。对于这tlevel和blevel这两个属性值的应用，本文在第五章的非线性聚簇下的独立任务调度中还会进行进一步阐述。对图2-1而言，它的每个节点的三个属性值如表2-2所示。

表2-2　图2-1中每个节点的3个不同属性值

节点	sbl值	tlevel值	blevel值
n_1	11	0	23
n_2	8	6	15
n_3	8	3	14
n_4	9	3	15
n_5	5	3	5
n_6	5	10	10
n_7	5	12	11
n_8	5	8	10
n_9	1	22	1

目前基于表调度的算法有HLFET, ISH, LAST, ETF以及MCP算法等[7, 15-17, 19]。

2.3.2 聚簇调度算法

正如前文所说，聚簇调度算法都假设处理机个数无限（虚拟处理机）。它的基本调度思想如下：在调度的开始阶段，每个DAG图的任务节点都被认为是一个簇。接下来便进行簇的合并，当然这些合并要以不增加整个任务的完成时间为前提。该种合并操作要在最后没有任何簇可以合并时为止。聚簇算法的思想是充分利用更多数目的处理机来最大程度减少任务的调度长度(makespan)。因为该方法都假设有无限个虚拟处理机，但实际上处理机数目总是有限的。所以聚簇算法通

常需要一个簇（或者称虚拟处理机）到现实处理机的一个映射过程。当然映射完成后还要决定每个任务在各个处理机上的调度时间。

根据簇中有没有独立任务，聚簇算法又可以分为线性聚簇和非线性聚簇两大类。本文在下面的章节中还会详细阐述这两个概念，进而研究非线性聚簇下独立任务的调度问题。因为聚簇算法具有阶段性的特点，所以它特别适用于目前的一些任意处理机网络和异构系统的调度。因为当 DAG 图被聚簇后，根据簇到处理机的映射不同，还可以重新计算任务间的通讯延迟和计算量的大小。这是因为在任意处理机网络和异构系统中，每个处理机的处理能力和处理机间的通讯速率都会有所不同。这一点将会在本文的研究展望一节详细阐述。现有的聚簇算法有 EZ, LC, MD, DSC, DCP 和 EZDCP 等[15,21-25]。

2.3.3 基于任务复制的调度算法

DAG 调度的实质也就是一方面减少任务之间的通讯延迟，另外一方面就是要最大化任务之间的并行度。如果调度策略选取不当，可能会抵消任务并行化所带来的收益。所以 DAG 调度算法必须要在这两者之间进行折衷。基于任务复制的调度算法的思想是：通过在不同处理机上复制某个或某些任务，以此来减少任务间的通讯延迟。这是因为如果任务被映射到同一个处理机上，它们的通讯延迟可以忽略不计。不过这样也大大增加了算法的空间复杂性，另外一方面正如前文所说，基于任务复制的调度算法还存在一个怎样选取复制节点的问题。有的算法会全盘复制所有的祖先节点，而有的只是选择复制那些父亲节点。

近年来，为了寻求最小化并行任务的调度时间，人们开始不断研究怎样选取恰当的任务复制算法。该算法对 fork 图和 join 图可以产生最优的调度。这是显然的，因为对于 fork 图而言，我们只要在每个处理机上复制根节点，那么每个孩子节点都可以有最早的任务执行时间。而对于 join 图，虽然没有必要复制每个父亲节点从而让孩子节点最早执行，但是任务复制算法可以采用一个类似递归的调度策略来最小化每个节点的开始执行时间，这样就可以产生一个最优调度结果。

因为要进行向后搜索(backtracking)和存储更多的任务，任务复制算法的时间复杂度和空间复杂度都比同类算法要高出很多。所以任务复制算法更多的用于理论研究，而较少用于实际调度工具中。任务复制调度算法也不是本文讨论的重点。现有的任务复制调度算法有 DSH[20],PY[44],BTDH[47],LCTD[48],CPFD[30]和 TCSD[29]等。

2.3.4 随机化搜索技术方法

随着处理机技术和网络硬件技术的发展，并行处理所应用的平台也已经从紧耦合的MPP机转向了松耦合的自治工作站网络。为了适应这种新形式，DAG调度算法的性能要求也越来越高，它必须要考虑下面四个问题：算法性能，时间复杂度，可扩展性以及可应用性。

(1) 性能：一方面，一个调度算法必须总是产生高效的结果，譬如调度长度最短，用到的处理机个数最少等等。另一方面，该算法必须具有鲁棒性，这是指算法必须对任何形式的输入，譬如任意形状的DAG和任意数目的处理机都能够产生好的结果。

(2) 时间复杂度：一方面，因为实际的输入往往是具有成千上万个处理节点的大型输入数据。如果时间复杂度较高，将使得算法具有不可用性。另一方面在实时调度系统中，只有快速的算法才能产生高效的结果。

(3) 可扩展性：一方面指算法具有问题规模的可扩展性，即随着规模的扩大，算法总能保证有效的输出，另一方面指算法具有处理能力的可扩展性，即随着处理机数目的增多，输出结果会更有效。

(4) 可应用性：指算法必须考虑其实际运行环境。譬如任意大小的任务执行时间和通信时间，链路竞争以及处理机拓扑结构的易变性等。

很明显，以上几个调度目标互相冲突，为了应对这种挑战，人们提出了一些新的调度思想，譬如遗传算法，随机化方法和并行调度技术等。这些新方法主要结合由前面的搜索结果所获取的知识和一些随机特征来产生新的调度结果。用随机化搜索技术方法进行调度的结果比较好，但是它们的调度时间要比其它类的算法高的多。另外譬如遗传算法(Genetic Algorithms)，它针对不同的DAG图，其最优控制参数也不相同，所以在实际调度过程中必须随时变更这些控制参数，从而造成不便。随机化搜索技术方法包括遗传算法[33]，模拟退火(Simulated Annealing)[34]以及局部搜索技术(Local Search Technique)[32]等。

第三章 并行任务的粒度划分及其相关定理

我们知道，并行程序的有效执行在很大程度上取决于程序到模块的有效划分和这些模块在一组处理机上的调度执行。影响程序并行开发的因素有很多，其中主要的因素包括程序的划分，计算负载在各个处理机上的平衡和数据通信所产生的额外开销等。对于给定的一个并行程序，程序的划分指定了程序的串行单元，它被称为簇(cluster)，这些单元可以被处理机并行执行，以使得程序的全部并行计算时间在包括处理机间通信开销的情况下是最小的。在本文中，术语簇和聚簇(clustering)是指分别指“一组在同一个处理机上运行的任务”和“将 DAG 图中任务节点进行合并”，而不是传统意义上的 PC 或者工作站集群(clusters of PCs/workstations)。

本章我们将重点研究什么是并行任务的粒度，粒度对调度策略的影响以及分析线性聚簇调度算法性能。

3.1 聚簇中的两个调度策略

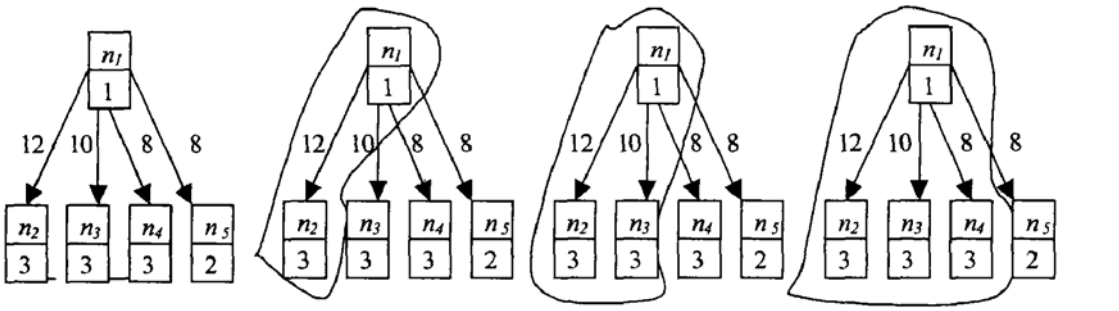
聚簇是将一个 DAG 中的任务映射到 m 个簇上。在聚簇中可以使用两个调度策略：

(1)将不相关的任务映射到一个簇中；

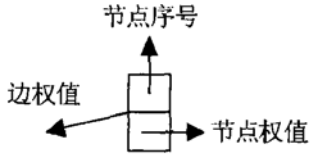
(2)将 DAG 中一条有依赖关系路径上的任务映射到一个簇中。

前者称为非线性聚簇(nonlinearly clustering)，它通过串行化不相关任务来减少并行性，以防止较大通信开销的产生。后者称为线性聚簇(linearly clustering)，它完全利用了 DAG 中的并行性。为了最大化并行性，任务应该尽量地均匀分散到各个处理机上，以使得计算平衡。从这一点上讲，应该使用线性聚簇。然而，进行通信的任务应该被分配到相同或邻近的处理机上，以保持数据的局部性和减少通信开销。那么从这个角度来说，就应该使用非线性聚簇。图 3-1 给出了这两个聚簇的例子。如果计算的粒度太大，就要限制并行性；如果粒度的太小，又会因为通信延迟而降低性能。因此就需要对任务粒度这一因素进行折衷性分析。任务粒度一般被定义为任务计算量与通信量的比值。假设 R 表示执行时间，C 表示任务所产生的通信延迟时间，那么 R/C 就是任务的粒度。当通信开销很大时，任务粒度(granularity)就会很小，这时就不适宜进行并行化。

众所周知，找到最优的聚簇结果是一个 NP 完全问题，但是对于线性聚簇，计算并行时间的问题是比较容易解决的，它具有多项式时间的复杂度。知道什么时候使用线性聚簇策略是非常重要的。



(a)原始 DAG 图　(b)线性聚簇后的 DAG (c)第一种非线性聚簇后的 DAG (d)第二种非线性聚簇后的 DAG



(e) DAG 图中每个节点上各标识的含义

图 3-1 一个细粒度 DAG 图以及其三种不同聚簇后的 DAG 图

任务图的粒度对于结构参数和程序划分都是很敏感的。很明显，非线性聚簇应被使用于细粒度的任务。在启发式算法中，选择一个合适的策略时应该考虑局部粒度的大小以及它对全句并行时间的影响。而且对于固定大小的问题来说，处理机的数量也是与粒度相关的。因为处理机个数 N 增大时，执行于每个处理机上的子问题的粒度就要减小。有研究表明，对于通信额外开销能够在 N 个处理机上被完全分解的算法，加速比对于除了最坏情况下的所有带宽值都能构随着处理机个数的增长而增大。对于通信额外开销不能被分解的算法，加速比在某一个 N 值时会接近最大值，然后它就下降到接近 0，这个 N 值是由处理与通信(processing to communication)的比值决定的。因此就必须对程序模块间并行的个数和关联的额外开销进行平衡。在将任务聚成簇以后，这些划分就将被调度到处理机上。因此划分是调度的预处理步骤。本章将着重研究以上提出的种种问题。

3.2 并行任务的粒度概述

本文中，我们以“grain”来表示某个任务的粒度，以“granularity”来表示 DAG 图的粒度。一般而言，无论是任务的粒度还是整个 DAG 的粒度，它都表示了一种任务的计算代价和不同任务间的通讯延迟之间的关系。但是对于怎样定义这种粒度，不同的文献都有各自表述。

在文[7]中，任务的粒度被定义为一个或者多个并发执行的程序模块。该程序模块在接收到所有的输入之后立即开始执行，而且仅仅在其所有的输出都计算之后才终止计算。该任务的粒度大小定义为程序模块中原子操作的个数。这其中

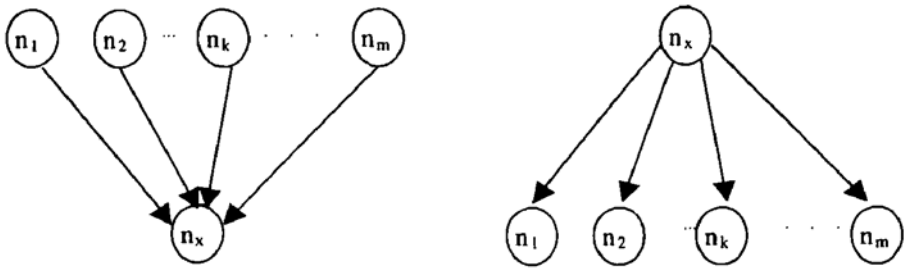
存在的“grain packing”问题就是一个在程序模块的计算和通信时间中进行折衷的问题。在文章[52]和[53]中，粒度被定义为单一处理机顺序执行的一系列程序步骤或者指令。这些步骤或指令必须要顺序串行执行。在文[54]中，任务粒度还北定义为同步操作中的执行时间。因为这些任务的粒度定义都只是考虑了DAG中单个任务的计算代价，所以它们并不能对带边权值DAG的调度产生指导作用。正是因为这个原因，文章[22]把任务的粒度定义为计算代价R和通讯代价C的商，也即R/C，并且该文认为正是这种粒度的大小决定了DAG中任务串行化和并行执行之间的均衡问题。文[55]扩展了上述粒度定义，他们引入了一个新的DAG图粒度表述，那就是把DAG的粒度g定义为：

$$g=\min_{x=1..v}\{t_x/\max_{j}c_{x,j}\} \qquad \text{（公式 3-1）}$$

这里的$c_{x,j}$表示任务n_x和任务n_j之间的通信代价，t_x表示n_x的计算代价，v表示节点的个数。以上所有的粒度方面的定义都有它的局限性。最近被广大研究人员采用的是Tao Yang教授在文献[43]中引入的基于fork图和join图所定义的DAG粒度。

3.3 Tao Yang的并行程序粒度定义及相关定理

正如上一节所说，Tao Yang教授提出的粒度定义是基于fork图和join图的，请参见图3-2。他的粒度定义思想如下：首先以这两个图来定义某个任务的粒度，然后根据所有任务的粒度来得出整个DAG图的粒度。



(a) join 图 J_x　　(b) fork 图 F_x

图 3-2 join 图和 fork 图

我们知道，DAG图都是由多个join或者fork图来组合。对于某个任务n_x而言，如果我们仅看它的所有的父亲节点，那么就如图3-2(a)所示为join图；如果我们仅看它的所有孩子节点，那么就如图3-2(b)所示为fork图。那么怎么定义任务n_x的粒度？它可以由其fork和join图的粒度来得到。我们定义join集合J_x包含了任务节点n_x和它的所有父亲节点，定义fork集合F_x包含了任务节

点 n_x和它的所有孩子节点。那么 $J_x=\{n_1, n_2, \cdots, n_m\}$，并且 $F_x=\{n_1, n_2, \cdots, n_m\}$。我们有：

$$g(J_x)=\min_{k=1..m}\{t_k\}/\max_{k=1..m}\{c_{k,x}\} \quad \text{（公式 3-2）}$$

和

$$g(F_x)=\min_{k=1..m}\{t_k\}/\max_{k=1..m}\{c_{x,k}\} \quad \text{（公式 3-3）}$$

然后我们引入任务 n_x的粒度 g_x为：

$$g_x=\min\{g(J_x), g(F_x)\} \quad \text{（公式 3-4）}$$

从而一个 DAG 图的粒度 g(G)为

$$g(G)=\min_{x=1..v}\{g_x\} \quad \text{（公式 3-5）}$$

如果 $g(G)\geq 1$，我们称该 DAG 图为粗粒度 DAG 任务，反之我们便称之为细粒度 DAG 任务。实际上这种粗粒度 DAG 任务也即那种图中所有任务的计算代价均大于每两个任务之间的通讯延迟。很显然这种粗粒度 DAG 图对那种通信延迟比较小的并行任务十分适用。Tao Yang 教授在文[43]中证明了两个对粗粒度任务非常有用的两条定理，分别是：

(1)对于粗粒度 DAG 任务而言，线性聚簇结果总是优于或者等同于非线性聚簇。也即对于一个粗粒度 DAG 的任何一种非线性聚簇，总会找到一个线性聚簇，它的调度长度要等于或者小于非线性聚簇的调度长度；

(2)不仅如此，对任意 DAG 的任何线性聚簇算法，它的调度长度 PT_{lc}满足下列不等式

$$PT_{opt}\leq PT_{lc}\leq (1+1/g(G))PT_{opt} \quad \text{（公式 3-6）}$$

这里的 PT_{opt}代表最优的调度长度，而 PT_{lc}代表线性聚簇后的调度长度。并且对于那种粗粒度 DAG，也即当 $g(G)\geq 1$ 时，以上不等式变为

$$PT_{lc}\leq 2*PT_{opt} \quad \text{（公式 3-7）}$$

很明显，以上两条定理对粗粒度 DAG 选择调度算法有很强的指导意义，并且它还给出了线性聚簇 DAG 的调度性能上下界分析。至于以上定理的证明过程，本文不给出详细的描述，有兴趣的读者可以参见文[43]。

Tao Yang 的这篇文章中把 $g(G)<1$ 的 DAG 定义为细粒度任务图，本文认为这样定义细粒度 DAG 有它的片面性。既然粗粒度 DAG 有这么好的性质，那么我们何不定义一种真正意义上的细粒度 DAG，并且研究它是否也具有某种令人鼓舞的调度定理？答案是肯定的。下一节将给我本文定义的细粒度 DAG 和它的一些相关定理。

3.4 本文提出的新的粒度定义

3.4.1 新粒度定义的描述

本文提出的新的粒度定义也是基于 fork 图和 join 图。我们仍然定义 join 集合 J_x 包含了任务节点 n_x 和它的所有父亲节点，定义 fork 集合 F_x 包含了任务节点 n_x 和它的所有孩子节点。那么 $J_x=\{n_1, n_2, \cdots, n_m\}$，并且 $F_x=\{n_1, n_2, \cdots, n_m\}$。我们有公司 3-8 和公式 3-9。

$$G(J_x)=\max_{k=1..m}\{t_k\}/\min_{k=1..m}\{c_{k,x}\} \quad \text{(公式 3-8)}$$

$$G(F_x)=\max_{k=1..m}\{t_k\}/\min_{k=1..m}\{c_{x,k}\} \quad \text{(公式 3-9)}$$

然后我们引入任务 n_x 的粒度 G_x 为：

$$G_x=\max\{G(J_x), G(F_x)\} \quad \text{(公式 3-10)}$$

从而一个 DAG 图的粒度 G(G) 为

$$G(G)=\max_{x=1..v}\{G_x\} \quad \text{(公式 3-11)}$$

当 $G(G)\leq 1$ 时间，我们称该 DAG 为细粒度 DAG 任务图。这和 Tao Yang 所定义的细粒度 DAG 任务图截然不同。本文定义的这种细粒度 DAG 也是指那种 DAG 图中所有节点的计算代价均小于每条边的通信延迟。所以它更加适合那种大通信延迟的并行任务。既然对那种粗粒度 DAG 图，线性聚簇要优于非线性聚簇，那么对于那种细粒度 DAG 图，是否可以证明非线性聚簇要优于线性聚簇？答案是肯定的。下面我们给出其数学证明过程。

3.4.2 细粒度 DAG 调度性质证明

在此我们仍然考虑那种 join 集合 J_x，它的每条边的通信代价 $c_{k,x}$ 以 w_j 表示，也即 $c_{k,x}=w_j$, j=1..m。不失一般性，我们假定 J_x 中的节点和边按照以下次序排列，即

$$t_j+w_j\geq t_{j+1}+w_{j+1},\ \ j=1..m-1 \quad \text{(公式 3-12)}$$

这里的 t_j 表示节点 n_j 的计算代价。这样的 join 任务图请参见图 3-3(a)。
对这样的任务图，它的最优调度长度为：

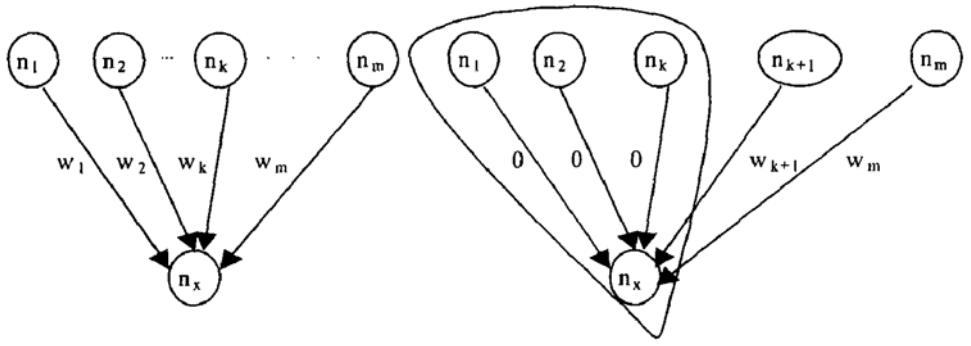
$$\max\{t_x+\sum_{j=1}^{k}t_j,\ \ t_x+w_{k+1}+t_{k+1}\} \quad \text{(公式 3-13)}$$

这里的 k 值必须要满足下列两个条件，参见图 3-3(b)：

$$(1)\ \sum_{j=1}^{k}t_j \leq w_k+t_k \quad \text{(公式 3-14)}$$

(2) $\sum_{j=1}^{k+1} t_j > w_{k+1}+t_{k+1}$　　(公式 3-15)

从公式 3-14，文[43]证明了对于一个粗粒度 DAG，线性聚簇要优于非线性聚簇。本文将从公式 3-15 出发，证明对于一个细粒度 DAG 而言，非线性聚簇要优于线性聚簇，并且 DAG 图的粒度越小，非线性聚簇中就会有更多的独立任务。



(a) join 图 J_x　　(b) 聚簇后的 DAG

图 3-3　join 图的一个最优聚簇算法示例

公式 3-15 经过化简如下：

$$\sum_{j=1}^{k} t_j > w_{k+1} \qquad \text{(公式 3-16)}$$

上式经过缩放变换可得出：

$$k*\max_{j=1..m}\{t_j\} \geq \sum_{j=1}^{k} t_j > w_{k+1} \geq \min_{j=1..m}\{w_j\} \qquad \text{(公式 3-17)}$$

对公式 3-17 经过进一步化简得到：

$$k \geq 1/\{\max_{j=1..m}\{t_j\}/\min_{j=1..m}\{w_j\}\}=1/G(G) \qquad \text{(公式 3-18)}$$

这样当 G(G)<1 时，我们可以从公式 3-18 得出 $k_{opt} \geq 2$。因为 k 必须取正整数，所以这里的 k 必须为大于或者等于 2 的一些数值，并且 DAG 图的粒度越小，k 值将越大，也就会有更多的独立任务被放到同一个聚簇中。

结合本文的粒度定义和 Tao Yang 的粒度定义，我们可以把并行任务 DAG 进行重新划分。如果 G(G) ≤1，那么该 DAG 就是一个细粒度任务图(fine granularity)；如果 g(G) ≥1，那么该 DAG 就是一个粗粒度任务图(coarse granularity)，如果 G(G) >1 并且 g(G)<1，那么该 DAG 就是一个中粒度任务图(median granularity)。对不同粒度的 DAG 图，其调度策略也不尽相同。根据这些粒度定义，本文下面将对线性聚簇的性能界限进行扩展。

3.4.3 扩展的线性聚簇性能界限分析

从本文的两个粒度定义，我们可以看出 G(G) ≥g(G)。

对一个 DAG 图 G 而言，我们假定其关键路径 cp={$n_1, n_2, \cdots, n_k$}，并且 L_{cp} 代表这条关键路径的长度，它包含了路径上所有节点的计算代价和通信延迟，如下所示：

$$L_{cp}=\sum_{j=1}^{k} t_j + \sum_{j=1}^{k-1} c_{j,j+1} \qquad \text{(公式 3-19)}$$

从 G(G)和 g(G)的粒度定义，我们可以得出公式 3-20 和公式 3-21：

$$\sum_{j=1}^{k-1} c_{j,j+1} \leqslant \sum_{j=1}^{k} t_j / g(G) \qquad \text{(公式 3-20)}$$

$$\sum_{j=1}^{k} t_j \leqslant \sum_{j=1}^{k-1} c_{j,j+1} * G(G) \qquad \text{(公式 3-21)}$$

现在我们考虑经过最终聚簇的 DAG 图 G_n，它和任务图 G 结构相同，只是因为经过了聚簇，那些同簇中的任务之间的通讯代价可以忽略不计。现在线性聚簇后的调度长度 PT_{lc} 就等同于任务图 G_n 的关键路径长度。通过以上分析，我们可以得出：

$$\sum_{j=1}^{k} t_j \leqslant PT_{opt} \leqslant PT_{lc} \leqslant L_{cp} \qquad \text{(公式 3-22)}$$

从而有如下推导：

$$PT_{lc} \leqslant L_{cp}=\sum_{j=1}^{k} t_j + \sum_{j=1}^{k-1} c_{j,j+1} \leqslant (1+G(G))* \sum_{j=1}^{k-1} c_{j,j+1} \leqslant \sum_{j=1}^{k} t_j \{(1+G(G))/g(G)\}$$

$$\leqslant \{(1+G(G))/g(G)\} * PT_{opt}$$

也即：

$$PT_{lc} \leqslant \{(1+G(G))/g(G)\} * PT_{opt} \qquad \text{(公式 3-23)}$$

这样，当 g(G)=G(G)时，公式 3-23 变为：

$$PT_{opt} \leqslant PT_{lc} \leqslant \{1+1/g(G)\} * PT_{opt} \qquad \text{(公式 3-24)}$$

当 g(G)＝1 时，公式 3-23 变为：

$$PT_{opt} \leqslant PT_{lc} \leqslant \{1+G(G)\} * PT_{opt} \qquad \text{(公式 3-25)}$$

当 g(G)=G(G)=1 时，公式 3-23 变为：

$$PT_{opt} \leqslant PT_{lc} \leqslant 2\ PT_{opt} \qquad \text{(公式 3-26)}$$

这样，结合本文的粒度定义和文[43]的粒度定义，本文便扩展了线性聚簇性能的上下界分析。从以上特例也可以看出，这些结果和前人所总结的一些上下界分析结果相同。

3.4.4 细粒度 DAG 的调度实例分析

为了更好地理解本文提出的细粒度 DAG 图和它的调度性质，本节将给出一个实例分析，请参见图 3-1。

很明显，这个 DAG 图的粒度 G(G)=3/8，g(G)=1/12。所以这是一个细粒度 DAG 图。根据细粒度 DAG 图的调度性质，其最优调度应该是非线性聚簇，而且根据 $k \geq 1/G(G)$，我们可以推出 $k \geq 8/3$，因此最优的 k 值应该为 3，也就是该最优的非线性聚簇中要有 3 个独立任务。

图 3-1(a)是原始细粒度 DAG 图，其每个节点的计算代价都小于任意一条边的通信延迟。图 3-1(b)是一个线性聚簇，它的调度长度为 14；图 3-1(c)是一种非线性聚簇，不过该簇中只有两个独立任务，也即 k=2，此时的调度长度为 12，它比线性聚簇的调度结果要好。图 3-1(d)是另外一种线性聚簇，该簇中有三个独立任务，也即 k=3，根据调度性质，此时为最优调度，调度长度为 11。而如果当 k=4，也即所有任务聚成一个簇的时候，它的调度长度为 12，显然不是最优的调度。所以这个例子不仅体现了针对细粒度 DAG，非线性聚簇要优于非线性聚簇，而且粒度越细，非线性聚簇中的独立任务也随之增多。当然本文对以上性质的证明主要是从 fork 和 join 图入手，所以从任意 DAG 图来证明某个细粒度任务的非线性聚簇要优于线性聚簇是今后的一个研究方向。

第四章 DAG 调度基准测试图

从前文的阐述，我们可以看到，DAG 调度问题在这二十余年来已经得到了长足的发展，也诞生了各种各样的 DAG 调度算法。绝大部分的这些调度算法都宣称自己提出的算法是高效(efficient)的，但是怎么和其它算法进行性能的比较问题还不是非常的清晰明了。这是因为一方面不同的调度算法都是基于不同的假设，譬如任务是否允许复制，是否为抢占式任务等等，这就使得算法性能的比较显得相对没有什么太大的意义，另外一方面，目前还没有一个完完全全被所有人公认的一系列基准测试程序，这也就使得很多算法的比较大多自成一家。本章就主要讨论 DAG 调度性能的评价体系和给出大部分研究人员公认的一些基准测试 DAG 图，从而为本文的算法比较分析奠定基础。

4.1 DAG 调度算法的评价目标

为了使得调度算法之间有一定的可比性，本文一般在相同或者相似的假设条件和目标系统进行算法性能的比较。譬如一般目标系统都是指那些完全互连的同构系统，而且任务不允许复制，任务执行为非抢占式等。一般评价 DAG 调度算法的指标包括任务的调度长度和算法的时间和空间复杂性，有时候所用到的处理机个数也是一个重要的性能评价参数。前两个性能评价目标间经常存在一个折衷的问题，因为一般为了获得更好的调度性能（更短的调度长度），常常会带来更高的时间复杂度。一方面人们可以使用那些分支定界技术或者其它的非常耗时的全局搜索技术来达到最优或者近优的调度结果，另一方面，人们会采用快速方法来获取一个较满意的解决方案。进一步而言，采用更多的处理机一般情况下可能产生更好的调度结果。

影响 DAG 调度性能的参数包括任务图的结构，任务图的粒度大小，任务的个数以及边的条数等，除此之外，目标系统的处理机个数也是一个比较重要的影响因素。至于那些基准测试图(Benchmark Graphs)，目前还不存在一个完完全全大家都公认的标准[51]。不少研究学者都采用一种随机 DAG 产生图(Random Graphs)。但是这些随机 DAG 图的产生也是不同文献采用不同的产生标准，人们对怎样产生随机 DAG 图也没有一个标准的约定。此外，还有文献以那些诸如高斯消去等程序产生的 DAG 图(traced graphs)来作为评价标准。近年来，在研究 DAG 调度领域的学者们所撰写的研究论文中，不同的人都采用了各自具有代表性的 DAG 图来作为实例分析和测试标准，它们已经成为事实上的基准 DAG 图。虽然这些 DAG 图一般而言规模都不是很大，但是它们对跟踪算法的运行步骤和测试算法的性能有一

定的代表性。这些 DAG 图一般被称为“peer set graphs”，简称 PSG。所以本文也将以这些事实上的基准测试图来进行算法的性能比较分析，下面将对其进行详细介绍。

4.2 基准测试 DAG 图

正如上文所述，本文将采用那些事实上的“peer set graphs”(PSG)来作为基准测试 DAG 图(benchmark graphs)。下面本文将列出 11 个这种 PSG 图的出处及相关信息，如表 4-1 所示。这些 PSG 图可以从网上下载，网址是 http://www.eee.hku.hk/~ykwok/publications.html。

表 4-1 11 种基准测试 DAG 图 PSG 的相关信息

序号	该 PSG 的节点数	作者	文章名称	发表时间
1	13	Ahmad, Kwok	On parallelizing the multiprocessor scheduling problem	1999
2	16	Al-Maasarani	Priority-Based Scheduling and Evaluation of Precedence Graphs with Communication Times	1993
3	17	Al-Mouhamed	Lower bound on the number of processors and time for scheduling precedence graphs with communication costs	1990
4	11	Shirazi et al.	Comparative study of task duplication static scheduling versus clustering and non-clustering techniques	1995
5	9	Colin and Chretienne	C.P.M. scheduling with small computation delays and task duplication	1991
6	7	Gerasoulis and Yang	A comparison of clustering heuristics for scheduling DAGs on multiprocessors	1992

7	15	Kruatrachue and Lewis	Duplication scheduling heuristic (DSH): A new precedence task scheduler for parallel processor systems	1987
8	9	McCreary & Gill	Automatic determination of grain size for efficient parallel processing	1989
9	11	Chung & Ranka	Application and performance analysis of a compile-time optimization approach for list scheduling algorithms on distributed-memory multiprocessors	1992
10	18	Wu & Gajski	Hypertool: a programming aid for message-passing systems	1990
11	7	Yang and Gerasoulis	List scheduling with and without communication delays	1993

至于其它的一些测试图，譬如 traced graphs 和 random graphs,也可以从以上网址进行下载。

第五章 非线性聚簇中的高效独立任务调度算法 MPD

我们知道聚簇算法主要分为两大类，其一是一种线性聚簇方法，其二是非线性聚簇方法。线性聚簇不会产生独立任务，它们每个簇中的任务的执行次序可以按照箭头顺序依次执行。而对于非线性聚簇中的那些任务，除了有依赖关系的任务之外，还存在那种没有依赖关系的任务，也即独立任务，它们的调度次序将对最后的调度长度产生至关重要的影响。下面我们首先阐述什么是线性聚簇。

5.1 线性聚簇调度算法

我们在第三章中简要阐述了什么是线性聚簇算法，本节将详细对其进行阐述。线性聚簇的思想比较简单，每次合并关键路径上的所有节点为一个簇。该算法首先找出关键路径上的节点，然后立即把它们调度到处理机上。这些被调度的节点和边将从 DAG 图中进行移除。该算法的执行步骤描述如下：

线性聚簇算法 LC(Linearly Clustering)：

(1)首先，把所有的边标记为“未读”;

Repeat

(2) 确定那些仅包含“未读”边的关键路径

(3) 通过把该关键路径上的所有边权值赋 0 来进行聚簇

(4)把该路径上的所有边标记为“已读”，并且把和该路径上任意节点相连的边也标记为“已读”

Until 所有的边都变成“已读”

对图 2-1 所示的 DAG 图，应用 LC 算法的调度步骤如下表 5-1 所示：

表 5-1 对图 2-1 的 LC 调度步骤

步骤	被聚簇的关键路径（簇）
1	(n_1, n_7, n_9)
2	(n_4, n_8)
3	(n_2, n_6)
4	(n_5)
5	(n_3)

这时该任务图被划分成 5 个簇，此时的调度长度为 19。

LC 算法的时间复杂度为 O(|V|*(|E|+|V|))。因为 LC 算法并不能调度不同路径上的节点到同一个处理机中，所以它并不能对 fork 图或者 join 图产生最优的调度结果。

5.2 非线性聚簇下独立任务调度算法 MPD

5.2.1 非线性聚簇

从第三章的讨论我们可以看出，目前也只是对那种粗粒度 DAG 而言，线性聚簇要优于非线性聚簇。而在目前的网络并行分布式计算环境中，任务间的通信延迟将会变得很大，我们并不能保证所有任务的通讯代价要大于任意一条边的通讯延迟。也因为线性聚簇算法并不能调度不同路径上的任务到同一个处理机中，所以目前的聚簇算法绝大部分都属于非线性聚簇。非线性聚簇对那种细粒度 DAG 和大部分中粒度 DAG 任务图的调度性能要优于线性聚簇。当然，这里的非线性聚簇是指至少某个聚簇中存在不同路径上的任务，或者说存在独立任务。这里的独立任务也是指不存在前驱后继依赖关系的任务。

对于线性聚簇来说，每个划分中的任务只要按照 DAG 图中箭头的顺序依次调度即可，如图 5-1(d)所示，按照线性聚簇算法 LC，节点 n_1、n_2和 n_4被聚成一簇，节点 n_3被单独聚成一簇，每个簇内任务的调度按照箭头顺序即可。但对非线性聚簇来说，因为每个划分中可能存在相互独立的节点，它们之间不存在任何依赖关系的路径，所以很难直接决定它们的调度顺序，但在实际中它们的调度次序总是对整个并行任务的调度长度产生很大的影响。所以决定这些独立节点的调度次序对一个并行任务的有效执行至关重要。但是目前各种任务调度算法对此都没有一个明确的阐述，即使偶尔有一些算法，它们的调度策略也都不能满足实际应用。本文正是在这种背景下提出了一个基于节点最大并行度的任务调度算法 MPD，它克服了现有各种算法的很多缺陷，对独立任务调度结果性能良好。

5.2.2 现有独立任务调度策略的缺陷

我们在第二章中定义了任务的一些属性值 tlevel,blevel 和 sbl，这些值都是前人从 DAG 任务图中抽取出来的节点信息。以后的所有基于 DAG 图任务划分与调度算法基本上都用到了这些节点属性值。因为 tlevel 值直接决定了一个节点的最早开始执行时间，所以很多算法（如 MCP 算法）对 tlevel 值小的节点赋予较高的优先级，以使得此节点能够优先执行，这也意味着按照 DAG 图的拓扑顺序(Topological Order)来执行各个任务（包括独立任务）；因为 blevel 值和关键路径紧密相关，所以为了先调度关键路径上的节点，一些算法（如 Sarkar 算法）对 blevel 值大的节点赋予较高的优先级；因为边权值在聚簇的过程中可变，所以又有一些算法(如 EZ 边消除算法和 ETF 最早开始时间优先算法)对 sbl 值较大的节点赋予较高的优先级；又为了综合考虑一个节点的最早开始时间和关键路径

节点，还有一些算法对 btlevel(blevel-tlevel)值较大的节点赋予较高优先级。下面我们用一些例子来说明这些节点信息在非线性任务聚簇 DAG 图调度中的应用。

对图 5-1(a)所示的任务划分，任务 n_1、n_2和 n_3被聚成一簇，任务 n_4被单独聚成一簇，其中 n_2和 n_3为独立节点。此时这四个节点的信息如表 5-2 所示。因为 $tlevel(n_2)=5, tlevel(n_3)=5$,所以不能依据这两个节点的 tlevel 值能决定其执行先后次序。而 $blevel(n_2)=29, blevel(n_3)=28$,如果按照 blevel 值大的节点先执行，即先调度 n_2，那么整个并行任务的调度长度为（必须注意在同一个簇中任务之间的通讯时间为 0）：5+20+10+10+8=53。如果按照它们的 sbl 值或者 btlevel(blevel-tlevel)值来进行调度，因为 $sbl(n_2)=28, sbl(n_3)=18$，$btlevel(n_2)=24, btlevel(n_3)=23$，所以仍然先调度节点 n_2,其调度长度也为 53。但是如果我们先调度节点 n_3，那么此时的调度长度变成：5+10+20+1+8=44。这两种不同的调度的差别是明显的，先调度 n_3要优于先调度 n_2。但是赋予 blevel、sbl 和 btlevel 值较高的节点高优先级的策略都不能保证最优调度。那么是否对这些值较高的节点赋予较低的优先级能够保证最优调度？答案同样是否定的。如下例所示。

如图 5-1(b)所示，我们把节点 n_3的权值改为 20。表 5-3 给出了此时各节点的信息。如果按照这些节点信息值高的节点赋予较低优先级的原则，因为：

$blevel(n_2)<blevel(n_3)$；

$btlevel(n_2)<btlevel(n_3)$。

所以仍然先执行节点 n_2。此时调度长度为：5+20+20+10+8=63。同样如果先调度节点 n_3，则此时的调度长度变为：5+20+20+1+8=54。很明显，先调度节点 n_3仍然要优于先调度节点 n_2。

如图 5-1(c)所示，我们看一下先调度 tlevel 值较小的节点能否产生最优调度。此时 $C(n_1, n_3)$变为 5，任务被聚成三个簇，同样独立节点 n_2和 n_3仍然被聚成一簇。各节点信息如表 5-4 所示。因为：

$tlevel(n_2)<tlevel(n_3)$；

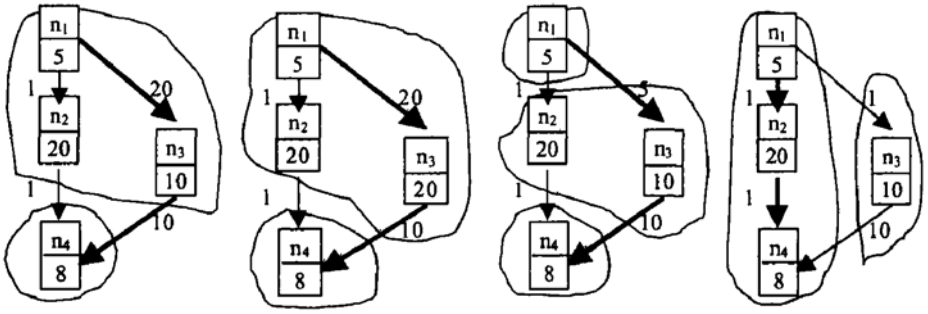
$blevel(n_2)>blevel(n_3)$；

$sbl(n_2)>sbl(n_3)$；

$btlevel(n_2)>btlevel(n_3)$。

按照 tlevel 值较小的节点赋予较高优先级的原则，节点 n_2要先于节点 n_3执行，如果按照其它三个值较高的节点也先执行的原则，n_2同样也先于 n_3执行。此时调度长度为：5+1+20+10+10+8=54。但是如果先执行节点 n_3，则此时调度长度为：5+5+10+20+1+8=49。由此可见，tlevel 值较小的节点赋予较高优先级的策

略同样不能保证最优调度。



(a) 原始 DAG 图 (b)W(n_3) 变为 20 (c)C(n_1,n_3)变为 5 (d) 线性聚簇后的 DAG

图 5-1 四个聚簇后的 DAG 图

表 5-2 图 5-1(a)中 DAG 图的各节点属性值

Nodes / Node value	n_1	n_2	n_3	n_4
tlevel	0	5	5	26
blevel	34	29	28	8
(btlevel) blevel-tlevel	34	24	23	-18
Static_blevel	33	28	18	8
tlevel	5	25	15	34
blevel	29	9	18	0

表 5-3 图 5-1(b)中 DAG 图的各节点属性值

Nodes / Node value	n_1	n_2	n_3	n_4
tlevel	0	5	5	35
blevel	43	29	38	8
(btlevel) blevel-tlevel	43	24	33	-27
Static_blevel	33	28	28	8
tlevel	5	25	25	43
blevel	38	9	18	0

表 5-4 图 5-1(c)中 DAG 图的各节点属性值

Nodes / Node value	n_1	n_2	n_3	n_4
tlevel	0	6	10	30
blevel	38	29	28	8
(btlevel) blevel-tlevel	38	23	18	-22
Static_blevel	33	28	18	8
tlevel	5	26	20	38
blevel	33	9	18	0

从以上的任务调度分析可以看出，在这三个 DAG 图中，先调度节点 n_3 均优于先调度节点 n_2。究其原因就是先调度节点 n_3 时，使得节点 n_2 的计算时间以及节点 n_2 和节点 n_4 的通讯时间与节点 n_3 与 n_4 之间的通讯得到了最大限度的并行化。因为此时任务已经被聚簇，各个任务的粒度、计算量以及通讯大小都已经确定，要减少整个任务的执行时间，途径只有一个，即使各任务的通讯时间和计算时间整体上得到最大程度并行。在上面的 DAG 图中，如果先调度节点 n_2，那么就不能保证上面的最大并行度。显而易见，这些传统的节点信息不能保证最大化任务的通信与计算的并行程度。我们有必要扩展并抽取新的节点信息。

5.2.3 基于最大并行度的独立任务调度算法 MPD

为了保证独立节点的执行顺序能够最大化任务通讯与任务计算的并行度，同时也为了测量并行度的大小，本文引进了以下新的节点信息。

定义 3：在一个 DAG 图中，$tlevel(V_i)$是指从一个入节点到节点 V_i 的最长距离，它必须包括该路径上所有节点包括 V_i 本身的计算量的值。相应的，$blevel(V_i)$是指从节点 V_i 到某个出节点的最长距离，但节点 V_i 本身的计算量的值必须去除。对于入节点，其 tlevel 值为其计算代价本身，而对出节点，其 blevel 值为 0。

设 V_i 和 V_j 为某簇中的两个独立节点，这两个节点的调度时序由以下算法决定（其中函数 min(a,b)返回 a 和 b 中的最小值）：

MPD(Maximized Parallelism Degree)算法：

Step 1：如果某个簇中存在独立任务 V_i 和 V_j，那么计算簇中节点的 blevel 和 tlevel 两个属性值；

If

$$\min[tlevel(V_i),tlevel(V_j)]+\min[blevel(V_i)+blevel(V_j)] >= \min[tlevel(V_j),tlevel(V_i)]+\min[blevel(V_j)+blevel(V_i)]$$

then　　首先调度节点 V_i

else　　首先调度节点 V_j；

Step 2：从第一个被调度的节点向后一个被调度的节点之间加一条虚拟箭头，表示这两个独立任务之间的依赖关系；

Step 3：重新计算该簇中的节点的 tlevel 和 blevel 两个属性值，转向 step1。

在这个 MPD 算法中，其中 $\min[tlevel(V_i),tlevel(V_j)]$表示先调度 V_i 节点时前半部分的并行度，$\min[blevel(V_j)+belevel(V_i)]$表示先调度 V_i 节点时后半部分的并行度，$\min[tlevel(V_i),tlevel(V_j)]+\min[blevel(V_j)+belevel(V_i)]$表示先调度 V_i 节点时整个任务的并行度；同样 $\min[tlevel(V_j),tlevel(V_i)]$表示先调度 V_j 节点时前半部分的并行度，$\min[blevel(V_i)+belevel(V_j)]$表示先调度 V_j 节点时后半部分的并行度。

$\min[tlevel(V_j),tlevel(V_i)]+\min[blevel(V_i)+belevel(V_j)]$表示先调度 V_j 节点时整个任务的并行度。以上独立任务的调度策略保证了并行度高的节点先调度，这样就最大化了任务执行时的并行程度。在图 5-1 所示的前三个 DAG 中，应用此策略均能保证节点 n_3 要先于节点 n_2 调度，此时各个节点的新的节点信息分别如表 5-2、表 5-3 和表 5-4 所示。

(1) 对图 5-1(a)，其调度步骤如下：

因为：

Sum1=min[televel(n_3), tlevel(n_2)]+min[belevel(n_3)+blevel(n_2)]
= min[15, 5]+ min[18, 29]=5+18=23

Sum2=min[televel(n_2), tlevel(n_3)]+min[belevel(n_2)+blevel(n_3)]
= min[25, 5]+ min[9, 28]=5+9=14

所以节点 n_3 应该比节点 n_2 先执行。

这里 sum1 和 sum2 的差值为 9（23-14），它也代表分别先调度这两个节点后的调度长度的差值(53-44=9)。

(2) 对图 5-1(b)，其调度步骤如下：

因为：

Sum1=min[televel(n_3), tlevel(n_2)]+min[belevel(n_3)+blevel(n_2)]
= min[25, 5]+ min[18, 29]=5+18=23

Sum2=min[televel(n_2), tlevel(n_3)]+min[belevel(n_2)+blevel(n_3)]
= min[25, 5]+ min[9, 38]=5+9=14

所以节点 n_3 应该比节点 n_2 先执行。

这里 sum1 和 sum2 的差值为 9（23-14），它也代表分别先调度这两个节点后的调度长度的差值(63-54=9)。

(3) 对图 5-1(c)，其调度步骤如下：

因为：

Sum1=min[televel(n_3), tlevel(n_2)]+min[belevel(n_3)+blevel(n_2)]
= min[20, 6]+ min[18, 29]=6+18=24

Sum2=min[televel(n_2), tlevel(n_3)]+min[belevel(n_2)+blevel(n_3)]
= min[26, 10]+ min[9, 28]=10+9=19

所以节点 n_3 应该比节点 n_2 先执行。

这里 sum1 和 sum2 的差值为 5（24-19），它也代表分别先调度这两个节点后的调度长度的差值(54-49=5)。

5.2.4 MPD 算法的复杂度分析

和传统的采用 tlevel, blevel 和 sbl 属性值来进行调度独立任务的算法相比，因为 MPD 算法中有一个额外的重新计算 televel 和 belevel 属性值的操作，所以该算法的时间复杂度为 O(|V|*(|V|+|E|))。对于一个中粒度或者粗粒度 DAG 任务而言，因为独立的任务并不是很多，所以该算法的时间复杂度在该情况下和传统的算法调度时间相差并不是很大，但是性能却比传统的调度算法要优。传统的采用这三个属性值或者它们属性值的组合的算法的时间复杂度为 O(|V|+|E|)。

5.2.5 实验结果分析

并行 PSG 任务的 DAG 图如图 5-2(a)所示，它已经被非线性聚成两簇 PE0 和 PE1。此时各个节点的信息如表 5-5 所示。由图 5-2 可知，在 PE0 任务划分中，n_3和 n_4为独立节点，在 PE1 任务划分中，n_1和 n_2以及 n_5和 n_2均为独立节点。根据基于最大任务并行度的独立节点调度策略：

(1) 在 PE0 中的调度步骤如下，因为：

Sum1=min[televel(n_3),tlevel(n_4)]+min[belevel(n_3),blevel(n_4)]
=min[3,3]+min[2,4]=3+2=5

Sum2=min[televel(n_4)+tlevel(n_3)]+min[belevel(n_4),blevel(n_3)]
=min[5,2]+min[2,3]=2+2=4

所以节点 n_3应该先于节点 n_4调度；

这里 sum1 和 sum2 的差值为 1（5-4），它也代表分别先调度这两个节点后的调度长度的差值(8-7=1)，如图 5-2(b)和图 5-2(d)所示。

(2) 而在 PE1 中的调度步骤如下，因为

Sum1=min[televel(n_1),tlevel(n_2)]+min[belevel(n_1),blevel(n_2)]
=min[1,0]+min[6,5]=0+5=5

Sum2=min[televel(n_2),tlevel(n_1)]+min[belevel(n_2),blevel(n_1)]
=min[4,0]+min[1,7]=0+1=1

所以节点 n_1应先于 n_2调度；

这里 sum1 和 sum2 的差值为 4(5-1)，它也代表分别先调度这两个节点后的调度长度的差值(12-8=4)，如图 5-2(c)和图 5-2(d)所示。

从节点 n_1向节点 n_2增加一个依赖关系的箭头；

因为

Sum1=min[televel(n2),tlevel(n5)]+min[belevel(n2),blevel(n5)]
=min[5,1]+min[1,2]=1+1=2

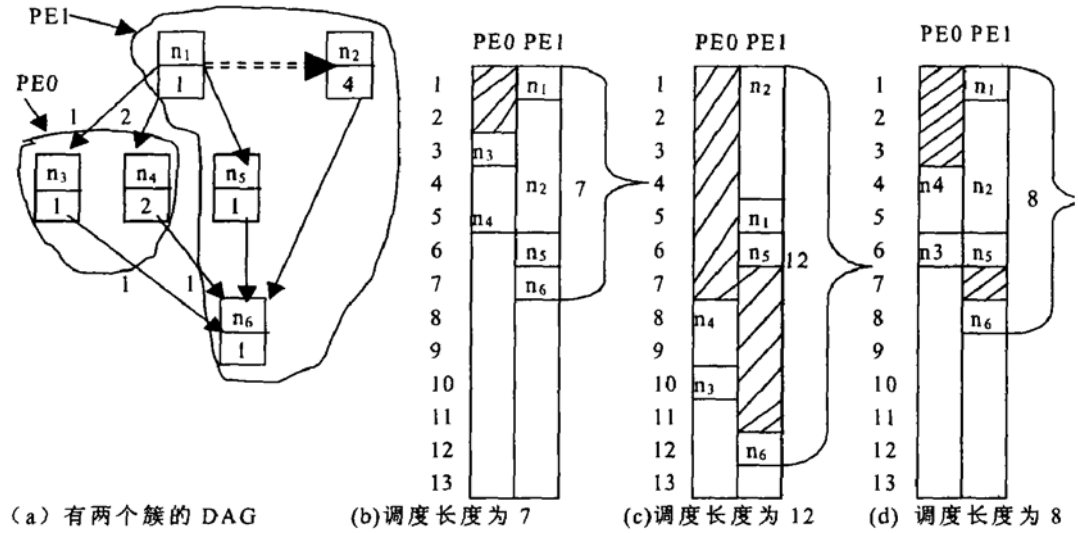
Sum2=min[televel(n5),tlevel(n2)]+min[belevel(n5),blevel(n2)]
=min[2,1]+min[1,5]=1+1=2

故节点 n_2和节点 n_5可以任意选择一个进行调度。

按照此顺序调度的 Gantt 图如图 5-2(b)所示，其调度长度为 7。如果对独立节点不采用任何调度策略，那么它的调度长度将达到 12，如图 5-2(c)所示。如果采用其它一些传统的调度策略，那么节点 n_4将先于节点 n_3调度，其调度长度将变为 8，如图 5-2(d)所示。

本章针对传统的 DAG 图中非线性聚簇下独立任务调度算法的缺陷，重新抽取并扩展了 DAG 图中各节点的信息值，指出了传统算法对独立任务调度的性能不佳

是由于没有最大化任务的并行程度所造成的。为了保证独立任务的执行顺序能够最大化任务的并行度，本文提出了一个基于最大任务并行度的独立任务调度算法 MPD，它克服了现有的任务调度算法的很多缺陷。试验结果和分析表明，本文提出的算法要优于其它传统任务调度算法，并且该策略的提出对今后 DAG 上任务划分算法的研究也具有一定的指导意义。



（a）有两个簇的 DAG　(b)调度长度为 7　(c)调度长度为 12　(d) 调度长度为 8

图 5-2 聚簇后的 DAG 以及其不同调度结果的 Gantt 示意图

表 5-5 图 5-2(a)中 DAG 图的各节点属性值

Nodes / Node value	n_1	n_2	n_3	n_4	n_5	n_6
tlevel	0	0	2	3	1	6
blevel	7	5	3	4	2	1
(btlevel) blevel-tlevel	7	5	1	1	1	-5
Static_blevel	4	5	2	3	2	1
televel	1	4	3	5	2	7
belevel	6	1	2	2	1	0

第六章 基于边消除和动态关键路径的 EZDCP 调度算法

正如前文所说，现有的调度算法中，有的性能很好，但是复杂度较高；而有的调度算法复杂度不高，但是性能却不及那些高复杂度的调度算法。怎样在这两者之间进行折衷，提出一个复杂度较低（可以实际应用）和较高性能的调度算法是本文追求的一个目标。正是基于这个思想，本章提出了一个基于边消除和动态关键路径的 EZDCP 算法。顾名思义，该算法结合了边消除算法（EZ）和动态关键路径(Dynamic Critical Path，简称 DCP)思想，不仅复杂度较低，而且性能良好。

6.1 边消除调度算法 EZ

边消除算法（Edge-zeroing）[32]是根据那些边权值来进行聚簇。在每一个执行步骤中，EZ 算法都从 DAG 图中搜索具有最大边权值的边。然后 EZ 算法尝试把这条边权值赋值为 0（也即把这条边连接的两个簇进行合并），如果这种赋 0 操作不会增加 DAG 任务的调度长度，则合并成功。在两个簇合并后，节点的调度序列是基于节点的 sbl 属性值。该算法的简要描述如下：

表 6-1 对图 2-1 的采用 EZ 算法的调度步骤

Step	Edge examined	*PT* if zeroed	Zeroing	PT_i
1	(n_1,n_7)	21	yes	21
2	(n_7,n_9)	20	yes	20
3	(n_6,n_9)	19	yes	19
4	(n_8,n_9)	22	no	19
5	(n_1,n_2)	18	yes	18
6	(n_1,n_4)	18	yes	18
7	(n_4,n_8)	22	no	18
8	(n_1,n_3)	21	no	18
9	(n_3,n_8)	20	no	18
10	(n_1,n_5)	23	no	18
11	(n_2,n_6)	18	yes	18
12	(n_2,n_7)	18	yes	18

EZ 算法：

(1)对 DAG 图中的所有边权值进行由高到低的排序；

(2)把 DAG 图中的所有边标记为“未被访问”；

Repeat

(3)从那些未被访问的边中选择一条权值最大的边。标记该边为“已访问”。尝试把这条边权值赋值为 0（也即把这条边连接的两个簇进行合并），如果这种赋 0 操作不会增加 DAG 任务的调度长度，则合并成功。在这两个簇合并后，把新簇中所有边权值赋值为 0，并且标记为“已访问”。簇中节点的调度序列依照节点的 sbl 属性值。

Until 所有的边均被访问。

对图 2-1 所示的 DAG 图，应用 EZ 算法的调度步骤如上表 6-1 所示，PT 代表 DAG 调度长度(Parallel Time)。此时该任务图被划分成 4 个簇，分别为 $\{n_1, n_2, n_4, n_6, n_7, n_9\}$，$\{n_3\}$，$\{n_5\}$和$\{n_8\}$，此时的 DAG 调度长度为 18。

EZ 调度算法的时间复杂度为 O(|E|*(|V|+|E|))。因为 EZ 调度算法在进行调度决策时仅仅考虑那些节点任务之间的通信代价，所以它并不能保证对 fork 图和 join 图产生最优调度结果。

6.2 其它一些调度算法

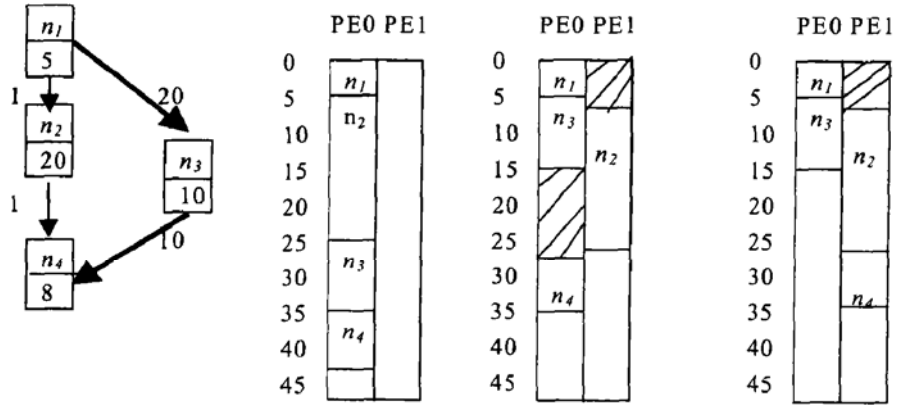
除了上文提到的 EZ 和 LC 这两种较早出现的算法以外，还有很多其它的一些调度算法，譬如 DSC[21]算法和 DCP[24]算法等。虽然 DCP 算法能够产生比其它算法相比较少的调度长度，但是它是以提高算法的时间复杂度为代价的，其时间复杂度达到了 $O(|V|^3)$。DSC 算法虽然时间和空间复杂度都比较低，但实现起来却比较复杂，所以本文就不加以详细介绍。至于其它一些调度算法，本文已经在第二章中进行了简要介绍，譬如 HLFET(Highest Level First with Estimated Times), MCP(Modified Critical Path), ETF(Earliest Time First), DLS(Dynamic Level Scheduling) 和 MD(Mobility Directed) 等，它们在论文[8]中有较详细的描述，本章我们仅对给定 DAG 图给出其调度结果，具体算法过程从略。

图 6-1(a)给出了一个 4 个节点的 DAG 任务图。为了更好地阐述 EZDCP 算法的实现机理，本文先给出各种调度算法对该图的调度结果。其中图 6-1(b)采用的是 HLFET, MCP, ETF 和 DLS 调度算法, 6-1(c)采用的是 EZ，LC 和 MD 调度算法，而图 6-1(d)采用的是 EZDCP 和 DSC 调度算法。

EZ 算法产生的调度图如图 6-1(c)所示。很明显，从这个例子中可以看出，在每步的调度过程中，EZ 算法用来选择被调度结点的标准并不能总是很正确地标识那些影响调度长度的最重要的结点；不仅如此，簇内任务调度次序仅仅按照 sbl 属性值来进行也不能保证每步都能正确地执行那些重要性高的结点。对这个 DAG 图用 EZ 算法来进行调度的结点次序是(n_1, n_3, n_4, n_2)。在n_1和n_3被调度后，权

值最大的边是(n_3, n_4)。这样 n_4就被调度到处理机 PE0 上。然而在此之后为了不增加程序的执行时间，n_2不能被调度到 PE0 上。调度结果表明此种调度算法不是一个高效的方法。其调度长度为 35，而图 6-1(d)所示的调度长度仅为 34。

因为 LC 算法不能调度不同路径上的结点到同一个处理机上，所以对一个任意结构的 DAG 图来说，它并不能得到最优甚至次优解。对图 6-1(a)中的 DAG 图，用 LC 算法产生的调度结果如图 6-1(c)所示。很明显这个调度结果也不是一个高效的调度。实际上，如果在一个 DAG 图中存在几个关键路径，那么它的调度性能会比其它调度算法(包括 EZ 算法)的性能更差。其它算法的调度结果性能也可以从图 6-1 中看出，这里就不详细介绍了。



(a)原始 DAG 图　　(b) 调度长度(43)　　(c) 调度长度(35)　　(d) 调度长度(34)

图 6-1 对给定 DAG 图采用不同调度算法后的调度结果 Gantt 示意图

6.3 EZDCP 调度算法

6.3.1 动态关键路径介绍

在介绍 EZDCP 算法之前，我们先给出两个定义。

定义 4：一条 DAG 图的关键路径(Critical Path)是这样一些结点和边的集合：它们从入结点开始到出结点结束，其上所有结点的计算代价和边上的通信代价的和为最大值。

举例来说，图 6-1(a)上的关键路径便是(n_1, n_3, n_4)，它们已经用粗箭头标了出来。但是因为该关键路径的长度在聚簇的过程中会不断变化，所以我们又有下面的动态关键路径的定义。

定义 5：一条动态关键路径(Dynamic Critical Path，简称 DCP)是指当前已被部分调度的 DAG 图的关键路径。这里必须要指出，包含在同一个簇中的结点间

的通信代价已经变为 0。并且由于非线性聚簇的关系，这条动态关键路径可能包含那些互相独立的结点。一个 DAG 图的调度长度由它的动态关键路径 DCP 决定。调度图的长度(PT)由下面的公式决定 ：$PT=\max_{V_i \in V}\{tlevel(V_i)+blevel(V_i)\}$。如果 DCP 上的所有边都被访问过，那么 sub-DCP（次关键路径）指 DAG 上至少有一条边没有被访问的最长路径。

前面我们已经讨论过，EZ 算法和 LC 算法都存在它们各自的缺陷。并且一般来说，并行程序的执行时间是由调度 DAG 图的动态关键路径决定。譬如在图 6-1(a)中，如果边(n_1, n_3) 已经被赋为 0，下一步我们选择边 (n_1, n_2)进行赋 0 操作，那么此刻的关键路径便是 (n_1, n_3, n_2, n_4)而不是 (n_1, n_2, n_4)，并且它的长度是 5＋10＋20＋1＋8＝44。这样如果我们能够在一定的约束条件下保证每一步都能最大限度地减少动态关键路径的长度(譬如我们可以在此关键路径上选择那条权值最大的边来进行赋 0 操作)，那么就有可能在最后达到最小化程序并行执行时间的目的。总之，EZDCP 算法的目标就是要尽量在每一步操作之后使得动态关键路径的长度减少和动态关键路径的数目减少，并且最后还应该最小化处理机的个数。

6.3.2 EZDCP 算法的描述

基于前面所述的主要思想，我们设计了 EZDCP 算法，它能够满足我们前面的要求。其算法描述如下所示：

EZDCP 算法：

(1)初始化每条边均为“未被访问过”。

Repeat

(2)扫描此刻的被部分调度的 DAG 图，找出那些至少含有一条未被访问边的关键路径（可能不止一条），否则找出那些至少含有一条未被访问边的次关键路径。

(3)把关键路径上边的权值按降序进行排列。

(4)从被选择的 DCP 上挑选一条未被访问的具有最大权值的边。如果这样的边不止一条，那么则按如下的规则选取，即选取那些层次比较高的边，也即：如果 $C(V_i, V_j)=C(V_k, V_l)$ 并且 $tlevel(V_i)<tlevel(V_k)$ ，那么选择边 e_{ij}。把这条边设置为“访问过”的标志。如果把这条边的权值赋为 0 能够满足如下条件，也就是此赋 0 操作不会产生另外一条不同的关键路径，它的长度不会等于甚至大于当前关键路径的长度。合并这两个簇。重复此步骤直到此 DCP 上的所有边都已经访问过或者其中一次赋 0 操作成功。

until 该 DAG 上的所有边都已经被访问过。

(5)检查这些不同的簇,看它们是否还能够进一步合并但不会增加程序的并行执行时间，从而使得处理机的个数最少并且使利用率提高。

对簇中的任务的调度，EZDCP 采用在第三章提出的 MPD 算法。

算法中的限定条件首先保证了在每一步不会增加程序的执行时间；其次，它保证不会因为此合并而产生另外一个与当前 DCP 长度相同的关键路径(包括本来就有多条关键路径的情况)，也即如果出现这种情况，则此次聚簇不成功。否则，关键路径的长度应该在每次聚簇后都能够最大限度地减少。

6.3.3 EZDCP 算法复杂度分析

定理 1:

对 EZDCP 算法调度的每一步,有 $PT_{i+1}<=PT_i$，并且如果 $PT_{i+1}=PT_i$，那么此时必定有多条关键路径或者表明此次聚簇不成功。

证明:

设在第 i 步，程序的执行时间为 PT_i,

(1)如果此时只存在一条关键路径，那么如果下一步的聚簇使程序执行时间增长或者这次聚簇产生另外一条关键路径，那么根据预定条件，这次聚簇便不成功，这样 $PT_{i+1}=PT_i$,否则 $PT_{i+1}<PT_i$;

(2)如果多于一条关键路径，那么如果下一步聚簇成功，便有 $PT_{i+1}=PT_i$,否则 PT_{i+1} 仍然等于 PT_i。

综合两种情况，定理 1 得证。

因为对算法的每一步都要寻找当前的关键路径，这样就需要在 O(|E|+|V|)时间里遍历该调度 DAG 图。又因为这样的步骤需要重复小于或等于 |E| 步，所以算法的整体时间复杂度仍为 O(|E|*(|E|+|V|))。在一个 DAG 图中，因为该算法需要存储|V|个结点和|E|条边的信息，所以其空间复杂度为 O(|E|+|V|)。

6.3.4 实验结果分析

一个 DAG 基准测试图如图 2-1 所示。根据 EZDCP 聚簇调度算法产生的簇如图 6-2 所示。其调度步骤如表 6-2 所示。并且分别采用 EZ 算法，LC 算法和 EZDCP 算法对该图的调度 Gantt 示意图如图 6-3 所示。

由于 EZDCP 聚簇算法开始规定每个任务处于不同的簇中,并且一个簇中的任务必须被调度到同一个处理机上，所以当所有边都已经被标为“访问过”后仍然存在某些簇只含有较少任务的情况。但是这些簇仍然占据了不同的处理机资源，

这样就有可能对它们再次合并从而使处理机的个数最少，但同时又不会使程序运行时间增长。这种合并由下面的原则决定。假设 DAG 图已经被划分成 p 簇 $(C_1, C_2, \cdots, C_p)$。tlevel(C_i)是该簇中具有最小 tlevel 值结点的 tlevel 值。W(C_i)是该簇中所有结点中计算代价的和。blevel(C_i)是该簇中具有最大 blevel 值结点的 blevel 值。设 tlevel(C_i)<=tlevel(C_j)。

如果 tlevel(C_i)+W(C_i)+blevel(C_j)<=PT(parallel time)，那么 C_i 和 C_j 就可进行再次合并。譬如在表 6-1 中，在第 10 步，结点 n_3 和 n_5 原本处于不同的簇中。在第 11 步，检查它们是否可以再次合并。因为：

tlevel(n_3)+W(n_3)+blevel(n_5)=3+3+5=11<17

所以 n_3 与 n_5 可以被再次合并成一簇。这样处理机的个数就由 4 个减少到了 3 个，并且处理机的利用率也得到了很大提高。

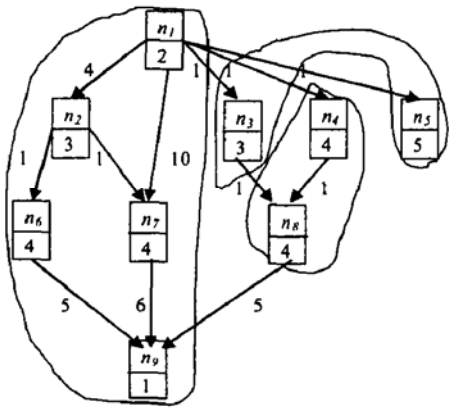


图 6-2　EZDCP 算法对图 2-1 基准测试 PSG 图的聚簇调度结果示意图

表 6-2 EZDCP 算法对图 2-1 基准测试 PSG 图的详细调度步骤

Step	*DCP* or sub-*DCP* which has at least one unexamined edge	Edge examined	*PT* if zeroed	Zeroing	PT_i
1	(n_1,n_7,n_9)	(n_1,n_7)	21	yes	21
2	(n_1,n_2,n_7,n_9)	(n_7,n_9)	20	yes	20
3	(n_1,n_2,n_6,n_9)	(n_6,n_9)	19	yes	19
4	(n_1,n_2,n_6,n_7,n_9)	(n_1,n_2)	18	yes	18
5	(n_1,n_4,n_8,n_9)	(n_8,n_9)	18	no	18
6	(n_1,n_4,n_8,n_9)	(n_1,n_4)	18	no	18
7	(n_1,n_4,n_8,n_9)	(n_4,n_8)	17	yes	17
8	(n_1,n_3,n_8,n_9)	(n_1,n_3)	17	no	17
9	(n_1,n_3,n_8,n_9)	(n_3,n_8)	20	no	17
10	(n_1,n_5)	(n_1,n_5)	19	no	17
11		(n_3,n_5)	17	yes	17

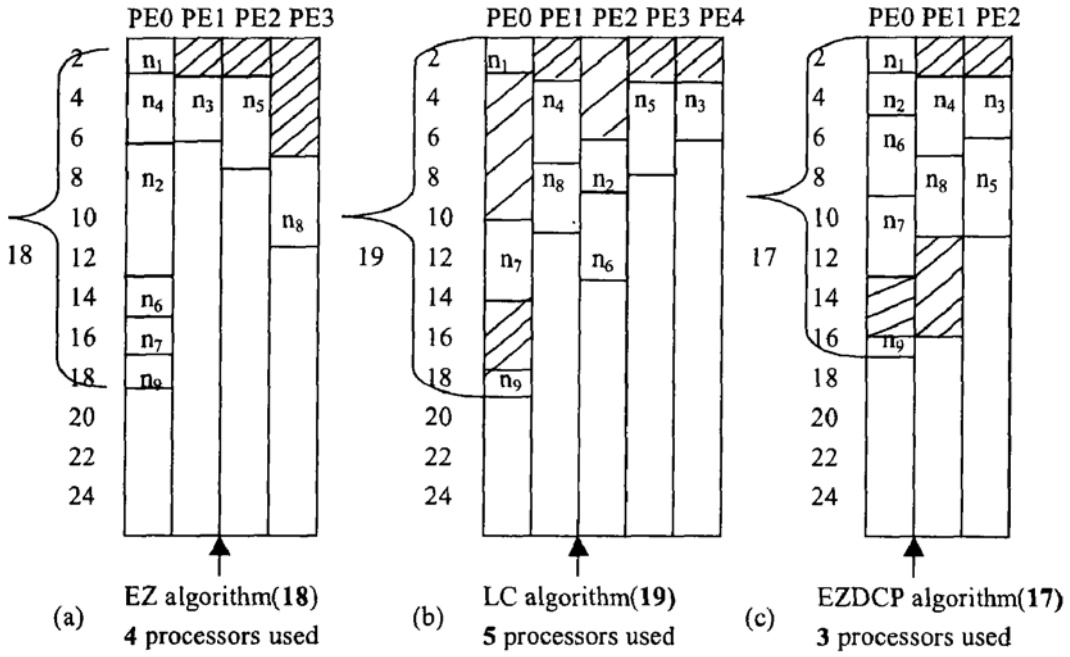


图 6-3　三种调度算法对图 2-1 基准测试 PSG 图的不同调度结果的 Gantt 示意图

从以上的分析，我们可以看出，EZDCP 算法具有如下几个特点：

(1)该算法可以保证在每步减少 DCP 的长度，除非那一步本来就存在多条关键路径；

(2)该算法根据节点的最大并行度 MPD 算法来动态决定每个节点的执行次序，而不是孤立地根据每个结点的 tlevel 或者 blevel 值来进行调度；

(3)它比同类具有相似甚至更高复杂度的算法具有更优的调度结果。譬如，在图 6-1 中，虽然 MD 算法的时间复杂度是 $O(|V|^3)$，它的调度长度仍然大于 EZDCP 算法。在图 6-3 中，虽然 EZ 算法具有和 EZDCP 算法相同的时间复杂度，但它的调度长度却比 EZDCP 算法要长；

(4)它在保证不增加调度长度的同时能够最小化处理机的个数，从而进一步提高了处理机的利用率。

第七章 本文的工作总结及研究展望

7.1 全文总结

本文主要研究了一种以带边权值和节点权值的有向无环图 DAG 来表示的并行任务的调度问题。文章首先介绍了这种 DAG 并行任务模型，总结了现有的几种 DAG 调度算法，对国内外几十年来 DAG 调度的发展进行了介绍，并对它们进行了详细的分类。接着本文主要讨论了四个方面的问题。

其一，本文讨论了 DAG 任务的的粒度理论问题。DAG 粒度的大小对选择有效的调度算法至关重要，并且它还可以用来对 DAG 调度算法的性能进行合理评估。本文在这一点上提出了一种新的任务粒度定义，并且用数学方法定义了一种细粒度 DAG 图(fine grain DAG)，这种 DAG 图中所有任务间的通信延迟均大于任务的执行时间。在此基础上，本文还基于 Fork-Join 图用严格的数学方法证明了对于这种细粒度 DAG，非线性聚簇要优于线性聚簇。不仅如此，本文还扩展了线性聚簇 DAG 的调度长度近优比；

其二，为了对各种调度算法的性能进行合理而有效的评估，本文根据前人的研究成果引入了已经被国内外研究人员公认的一些基准测试 DAG 图。它可以对各种同类型的 DAG 调度算法进行合理的评估。不仅如此，本文还给出了评价 DAG 调度算法性能的三个指标，分别是算法的复杂度，调度长度的大小以及利用处理机的个数。

其三，现有各种 DAG 调度算法中独立任务的调度基本上都是基于节点的 tlevel 或者 blevel 属性值或者这两者的综合信息来进行，本文从理论和实例两个方面分析了这种独立任务调度策略性能不佳的原因，指出这是因为没有最大化任务的执行和任务间通信的并行执行程度。基于此,本论文重新提取了 DAG 图的任务节点信息，并在此基础上提出了一个基于最大并行度的 MPD(Maximized Parallelism Degree)调度算法，该算法能够有效解决非线性聚簇下独立任务的调度问题；

最后，在现有的各种调度算法中，针对某些算法（譬如 DCP）调度性能很好，但是复杂度太高（使得较难用于实际），或者某些算法（譬如 EZ 和 LC）复杂度较低（易于使用），但是性能不佳的特点，本文设计了一种称为 EZDCP 的 DAG 调度算法。该算法基于动态关键路径和边消除思想，不仅具有较低的复杂度，而且其性能和 DSC 算法相当。由于 EZDCP 算法在性能和复杂度中进行了折衷，故它更加实用。

7.2 研究展望

最近二十余年来，随着网络硬件技术和处理器性能的迅速提高，并行处理领域进入了一个全新的阶段，从而也给并行调度领域中的代表 DAG 调度带来了新的问题。此时的并行调度不仅要考虑任务之间的通信代价还要考虑以下因素：

(1)新形式下的处理机网络并非完全互连(fully-connected)，而是采用任意形状的处理机网络(Arbitrary Processor Network,简称 APN)，譬如超立方体等。

(2) 调度环境的异构性带来的影响。目前绝大部分的 DAG 调度算法都是假设处理机同构，也即处理机的处理能力一致。

这些并行处理领域的新发展给传统的 DAG 调度带来了巨大挑战。因为处理机网络拓扑任意，所以连接处理机间的通信链路变为和处理机同等重要的可用资源。此时我们不仅需要调度任务给处理机，而且还要把通信消息调度到通信链路上从而最小化链路竞争。另外这里消息的调度策略还要考虑路由策略的影响，譬如存储转发(store and forward)技术和虫孔路由(wormhole)技术的不同选择会对消息调度策略产生重要的影响。此外因为处理机间的异构性，DAG 调度必须要考虑不同处理机间的处理能力不同，而且处理机间的通信速率也可能不同。在这样的情况下，原先 DAG 图中的一些已经预测和估计的任务通信延迟和计算时间在映射到物理处理机上执行时会有所变化。而现有的这些 DAG 调度算法均没有合理有效地考虑这些问题。除此之外，传统的 DAG 调度理论中还存在不少亟待解决的关键问题。在 DAG 聚簇中，众所周知，找到最优的聚簇调度是一个 NP 完全问题，但是对于线性聚簇，计算并行时间的问题是比较容易解决的，它具有多项式时间的复杂度。那么什么时候使用线性聚簇策略和什么时候使用非线性聚簇策略？我们在进行 DAG 调度时，必须选择一个合适的考虑 DAG 粒度的调度算法，也就是要进行 DAG 粒度的有效划分，从而指导有效选择调度策略。而且现有的启发式 DAG 调度算法种类繁多，那么怎样验证各种启发式算法的近似比以及怎样结合实际综合权衡 DAG 调度算法的各种目标而设计出低复杂度高性能的可扩展 DAG 调度算法等问题也是急需解决的。

当然这种新形式下的 DAG 调度和传统的 DAG 调度并不是互相分割的目标，以上问题的解决将会促进两者融合发展。传统的 DAG 调度中一些关键技术的解决必将推动面向 APN 和异构环境下 DAG 调度的发展，因为面向异构系统的调度就包含了面向同构系统的调度，面向 APN 网络的调度必然包含了面向完全互连网络的调度。反过来这些新形势下的调度技术的解决必将丰富传统 DAG 调度理论，而且必将大大推动 DAG 调度理论应用于实际系统。

异构计算平台和任意处理机网络是现今并行和分布式领域发展的必然趋势，那么针对 APN 网络和异构计算平台的特点，我们拟采取一个“四步骤”

(four-stage)的DAG调度策略，它结合了聚簇调度和表调度等其它调度算法的优势，其大体思想如下：

(1)对原始DAG任务图进行高效的聚簇。当然此时的簇的个数可能大于现实处理机的个数。这里的聚簇算法可以选取我们的EZDCP算法。

(2)聚簇的合并步骤。这个步骤我们需要把第一步的各个簇映射到和物理处理机数目等同的P个虚拟处理机中去。在此我们首先计算每个簇的负载大小，然后进行排序，最后选用一种合适的负载平衡策略进行簇到虚拟处理机的映射。

(3)进行虚拟处理机到物理处理机（实际存在的处理机）的映射。这一步对APN和异构计算平台非常重要。因为在这一步我们可以由处理机间的计算速度矩阵和通信速率矩阵来重新进行任务的计算和通信代价的评估。与此同时，根据不同的路由策略，我们还必须把那些消息调度到不同链路中去从而避免或者最小化链路竞争。譬如如果采用存储转发策略，通信延迟不仅要考虑消息的启动时间，通信带宽，消息大小，还要考虑处理机间的距离。而虫孔路由策略仅需要考虑通信链路竞争的影响，至于处理机间的距离则无需考虑。当然不同拓扑结构所采用的路由策略也不尽相同。目前不同的路由策略总可以归结为两类：其一是一种确定性路由方法，其二就是一种适应性路由机制。确定性路由方法在本项目中不太适用。因为当两条消息使用同一个链路时它们也不能改变路由策略，即使此时还有其它空闲链路可用，这样就不可避免地带来链路上的竞争。所以我们将采用适应性路由机制。在确定了消息路由之后，我们可以调度消息到链路中去。这可以通过时间通信图模型(TGG)来进行。TGG 模型的目标是最小化链路上的竞争。利用该模型可以计算消息的最早启动时间和最迟启动时间，这些值然后被用于启发式地调度链路上的消息。

(4)进行每个处理机上的任务的调度选择步骤(ordering of each task)。这一步骤类似于传统的”one stage”的表调度策略。当然在此我们可以采用本文提出的MPD算法。

另外为了使得DAG调度算法更能实际应用，我们还需要对DAG中任务和通信延迟进行有效合理且精确的估计，这也是DAG调度领域中未来的一个研究方向。

参 考 文 献

[1] Hall, L.A. Approximation algorithms for scheduling. In: Hochbaum, D., eds. Approximation Algorithms for NP-Hard Problems. Boston: PWS, 1996. 1-45

[2] Karger D. Stein, C. Wein. J. Scheduling algorithms. In: Atallah, M.J., eds. Handbook on Algorithms and Theory of Computation. CRC Press, 1998

[3] 陈华平，黄刘生等. 并行分布计算中的任务调度及其分类. 计算机科学，2001，28(1):45-47

[4]http://www.mathematik.uni-osnabrueck.de/research/OR/class/, 03/2004

[5]R. L. Graham, E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. Annals of Discrete Mathematics, 1979: 287-326

[6] M. R. Garey and D. S. Johnson. Computers and Intractability: A Guide to the Theory of NP-Completeness, Freeman, San Francisco, CA, 1979

[7] B. Kruatrachue and T. Lewis. Grain Size Determination for Parallel Processing. IEEE Software, 1988:23-32

[8] Yu-Kwong Kwok, Ishfaq Ahmad. Static scheduling algorithms for allocating directed task graphs to multiprocessors. ACM Computing Surveys, 1999, 31(4): 406-471

[9] T. C. Hu. Parallel sequencing and assembly line problems. Oper. Res, 1961, 19(6):841-848

[10] E. G. Coffman and R. L. Graham. Optimal scheduling for two-processor systems. Acta Information, 1972:200-213

[11] C. H. Papadimitriou and M. Yannakakis. Scheduling interval-ordered tasks. SIAM J. Comput, 1979, 8(3): 405-409.

[12] H. H. Ali and H. El-Rewini. The time complexity of scheduling interval orders with communication is polynomial. Parallel Processing Letter, 1993, 3(1):53-58

[13] Haluk Topcuoglu, Salim Hariri, Min-You Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(3): 260-274

[14]R. L. Graham. Bounds on Multiprocessing Timing Anomalies. SIAM J. Appl. Math., 1969, 17(2):417-429

[15] Min-You Wu, D. Gajski. Hypertool: A Programming Aid for Message-Passing Systems. IEEE Transactions on Parallel and Distributed Systems, 1990, 1(3):330-343

[16] J. J. Hwang, Y. C. Chow, F. D. Anger, and C. Y. Lee. Scheduling Procedence Graphs in Systems with Interprocessor Communication Times. SIAM Journal on Computing, 1989, 18(2):244-257

[17] T. L. Adam, K. M. Chandy, and J. Dickson. A Comparison of List Scheduling for Parallel Processing Systems. Communications of the ACM, 1974, 17(12):685-690

[18] G. C. Sih and E. A. Lee. A Compile-Time Scheduling Heuristic for Interconnection Constrained Heterogeneous Processor Architectures. IEEE Transactions on Parallel and Distributed Systems, 1993, 4(2):75-87

[19] J. Baxter and J. H. Patel. The LAST Algorithm: A Heuristic-Based Static Task Allocation Algorithm. Proceedings of International Conference on Parallel Processing, 1989, V2: 217-222

[20] B. Kruatrachue and T. G. Lewis. Duplication Scheduling Heuristics(DSH):A New Precedence Task Scheduler for Parallel Processor Systems. Technical Report, Oregon State University, 1987, Corvallis, OR97331

[21] Tao Yang, Apostolos Gerasoulis. DSC: Scheduling Parallel Tasks on an Unbounded Number of Processors. IEEE Transactions on Parallel and Distributed Systems, 1994, 5(9): 951-967

[22]V. Sarkar. Partitioning and Scheduling Parallel Programs for Multiprocessors. MIT Press, Cambridge, MA, 1989

[23] S. J. Kim and J. C. Browne. A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures. Proceedings of International Conference on Parallel Processing, 1988, V2:1-8

[24] Yu-Kwong Kwok, Ishfaq Ahmad. Dynamic Critical-Path Scheduling: An Effective Technique for Allocating Task Graphs to Multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1996, 7(5): 506-521

[25] Chen Zhi-gang, Hua Qiang-Sheng. EZDCP: A new static task scheduling algorithm with edge-zeroing based on dynamic critical paths. Journal of Central South University of Technology(English Edition), 2003, 10(2):140-144

[26] Qiang-Sheng Hua, Zhi-Gang Chen, Francis C.M. Lau. A new method on Independent Task Scheduling in Nonlinearly DAG Clustering. to appear in the seventh International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2004), IEEE Computer Society Press, Hong Kong, May 2004

[27] Chan-IK Park, Tae-Young Choe. An optimal scheduling algorithm based on task duplication. IEEE Transactions on Computers, 2002, 51(4):444-448

[28] Savina Bansal, Padam Kumar and Kuldip Singh. An Improved Duplication Strategy for Scheduing Precedence Constrained Graphs in Multiprocessor Systems. IEEE Transactions on Parallel and Distributed Systems, 2003, 14(6):533-544

[29] Li Guodong, Chen Daoxu, Wang Daming, Zhang Defu. Task Clustering and Scheduling to Multiprocessors with Duplication. Proceedings of the International Parallel and Distributed Processing Symposium(IPDPS'03):April 22-26, 2003, Nice, France

[30] I. Ahmad and Y. Kwok. A New Approach to Scheduling Parallel Programs Using Task Duplication. Proc. Int'l Conf. Parallel Processing, 1994, V2:47-51

[31] Andrei Radulescu, Arjan J.C. van Gemund. Low-Cost Task Scheduling for Distributed-Memory Machines. IEEE Transactions on Parallel and Distributed Systems, 2002, 13(6):648-658

[32] Min-You Wu, Wei Shu, Jun Gu. Efficient Local Search for DAG Scheduling. IEEE Transactions on Parallel and Distributed Systems, 2001, 12(6): 617-627

[33] H. Singh and A. Youssef. Mapping and Scheduling Heterogeneous Task Graphs Using Genetic Algorithms. Proc. Heterogeneous Computing Workshop, 1996. 86-97

[34] P. Shroff, D.W. Watson, N.S. Flann and r. Freund. Genetic Simulated Annealing for Scheduling Data-Dependent Tasks in Heterogeneous Environments. Proc. Heterogeneous Computing Workshop, 1996. 98-104

[35] 郑纬民, 石威. 相关任务图的均衡动态关键路径调度算法. 计算机学报, 2001, 24(9):1-8

[36] 赵宏, 张毅, 姜誉, 方滨兴, 刘振英. 一个调度 Fork-Join 任务图的新算法. 软件学报, 2002, 13(4):693-697

[37] 桂小林,钱德沛.元计算环境下的支持依赖任务的 OGS 算法研究.计算机学报,2002,25(4):584-588

[38] 刘东华,徐志伟,李伟.基于有向无环图的两层网格监测系统.计算机研究与发展, 2002,39(8):937-942

[39] 尹宝林,陆伯鹰.一个基于 DAG 图的指令调度优化算法.计算机工程与应用, 2001, 37(12):121-124

[40] 张毅，方滨兴，刘振英.TSA_OT:一个调度 Out-Tree 任务图的算法.计算机学报, 2001,24(4):1-5

[41] 韩承德，冯秀山，章军.基于超立方体的静态任务调度.软件学报,1999,10(12):238-244

[42] Qiangsheng Hua, Zhigang Chen. Efficient granularity and clustering of the Directed Acyclic Graphs. Proceedings of the fourth parallel and distributed computing, applications and technologies (PDCAT' 03), Chengdu, China, IEEE Press, 2003. 625-628

[43] Apostolos Gerasoulis, Tao Yang. On the Granularity and Clustering of Directed Acyclic Task Graphs. IEEE Transactions on Parallel and Distributed Systems, 1993, 4(6): 686-701

[44] C. H. Papadimitriou and M. Yannakakis. Towards an architecture-independent analysis of parallel algorithms, SIAM J. Computing, 1990, 19(2):322-328

[45] C.V. Ramamoorthy, K.M. Chandy and M.J. Gonzalez. Optimal Scheduling Strategies in a Multiprocessor System. IEEE Transactions on computers, 1972, C-21:137-146

[46] H. Kasahara and S. Narita. Practical Multiprocessor Scheduling Algorithms for Efficient Parallel Processing, IEEE Transactions on Computers, 1984, C-33:1023-1029

[47] Y.C. Chung and S. Ranka. Application and Performance Analysis of a Compile-Time Optimization Approach for List Scheduling Algorithms on Distributed-Memory Multiprocessors. Proceedings of Supercomputing' 92, Nov. 1992. 512-521

[48] H. Chen, B. Shirazi and J. Marquis. Performance Evaluation of A Novel Scheduling Method: Linear Clustering with Task Duplication. Proceedings of International Conference on Parallel and Distributed Systems, Dec. 1993. 270-275

[49] H.El-Rewini and T.G. Lewis. Scheduling Parallel Programs onto Arbitrary Target Machines. Journal of Parallel and Distributed Computing, 1990,9(2):138-153

[50] I. Ahmad, Y. -K. Kwok, M. -Y Wu, and W. Shu. Automatic Parallelization and Scheduling of Programs on Multiprocessors using CASCH. Proceedings of the 1997 International Conference on Parallel Processing, 1997. 288-291

[51] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and Comparison of the Task Graph Scheduling Algorithms. Journal of Parallel and Distributed Computing, 1999,59(3):381-422

[52]C. McCreary and H. Gill. Automatic Determination of Grain Size for Efficient Parallel Processing. Communication of the ACM, 1989, 32(9):1073-1078

[53] Yiqun Ge and David Y.Y. Yun. A Method that Determines Optimal Grain Size and Inherent Parallelism Concurrently. Proceedings of the 1996 International Symposium on Parallel Architectures, Algorithms and Networks(ISPAN' 96), 1996. 200-206

[54] Apostolos Gerasoulis, Sesh Venugopal and Tao Yang. Clustering Task Graphs for Message Passing Architectures. Proceedings 1990 International Conference on Supercomputing: 447-456

[55]A. Gerasoulis and S. Venugopal. Linear Clustering of Linear Algebra Task Graphs for Local Memory Systems. Preliminary Report.

[56] 黄金贵，陈建二，陈松乔. 并行环境下基于多处理机任务的调度模型与调度算法. 计算机科学，2002,29(4):1-3

[57] Jianer Chen, Jingui Huang. Semi-Normal Schedulings: Improvement on Goemans' Algorithm. Proceedings of 12th Annual International Symposium on Algorithm and Computation (ISAAC' 01, New zealand), Lecture Notes in Computer Science, Springer, 2001. 48-60

[58] Songqiao Chen, Jingui Huang, Jianer Chen. Approximation Algorithm for Multiprocessor Parallel Job Scheduling. Journal of Central South University of Technology (English Edition), 2002, 9(4): 267-272

[59] Tao Yang and Apostolos Gerasoulis. PYRROS: Static Task Scheduling and Code Generation for Message Passing Multiprocessors. Proceedings of 6th ACM International Conference on Supercomputing (ICS 92), Washington D.C :ACM, 1992. 428-437

致 谢

从2000年10月进入实验室以来，到现在已经有3年半左右的时间了。回顾这一段日子的本科加研究生生活，我首先要衷心感谢导师陈志刚教授。正是陈老师的帮助，才使得我提前加入了网络计算这个研究小组。在此我要特别感谢陈老师几年来对我所做研究的大力支持， 他不仅提供良好的实验室环境，而且积极支持我参加各种学术交流，这些都使我受益颇深。当然除了学习和研究方面的帮助，陈老师在生活和工作中也教会了我们很多做事的方法，这些宝贵的经验也使我获益匪浅。

正是在网络计算小组中师兄师姐的一些前期工作基础的指引下，我才逐步选择了并行与分布式计算这个研究领域，所以我要特别感谢曾志文老师，李登师兄，感谢合作伙伴许伟同学，和你们在研究上的合作与交流使我倍感愉快。

感谢我的本科导师赵跃龙教授。虽然研究生阶段和赵老师平时的交流不是很多，但是我能感受到赵老师对我未来道路的选择非常关心，感谢您对我生活上的一些支持与鼓舞。

感谢实验室和研究生班的其他同学在生活和工作中的支持与帮助，感谢七年来信息工程所和信息科学与工程学院的其他老师对我的谆谆教诲。

最后我还要诚挚地感谢那些在我困难的时候给予我支持和帮助的同学和朋友们，他们是老朋友朱峰同学，陈永超同学，曾志文老师，何成同学。在此我要向你们道声“谢谢”。

作者攻读硕士学位期间的主要研究成果

1）作者攻读硕士学位期间参加的科研项目情况

[1]湖南省自然科学基金（02JJY2097）“基于DS的普适性DLB负载行为预测及评估”，主要参加人员，06/2002-2004

[2]中南大学学生科技创新项目“证件照片中多姿态多表情下多方法融合人脸识别系统”，01/2002-2003

（该项目曾获第八届全国大学生挑战杯课外学术科技作品竞赛二等奖，湖南省“天一银河杯”研究生科技创新奖等多项奖励）

[3]撰写高等学校博士学科点专项科研基金申请书一份，课题名称为“三层客户/服务计算中构件挖掘及负载均衡技术研究”，03/2002

[4]撰写广东省自然科学基金申请书一份，课题名称为“网络并行计算中的负载预测及评估”，02/2003

[5]撰写国家自然科学基金申请书一份，课题名称为“启发式DAG调度理论中关键技术研究”，03/2003

[6]撰写高等学校博士学科点专项科研基金申请书一份，课题名称为“面向APN和异构系统的DAG调度理论关键技术研究”，03/2004

2）作者攻读硕士学位期间已经发表的论文(6 篇)

[1] Chen Zhi-gang, Hua Qiang-Sheng. EZDCP:A new static task scheduling algorithm with edge-zeroing based on dynamic critical paths, Journal of Central South University of Technology(English Edition), 2003,V10(2):140-144 [SCI 和 EI 检索]

[2] Qiangsheng Hua, Zhigang Chen. Efficient granularity and clustering of the Directed Acyclic Graphs, Proceedings of the 4th parallel and distributed computing, applications and technologies (PDCAT' 03), Chengdu,China,2003(IEEE Press):625-628 [ISTP 和 EI 检索]

[3] Qiang-Sheng Hua, Zhi-Gang Chen, Francis C.M. Lau. A new method on Independent Task Scheduling in Nonlinearly DAG Clustering, to appear in the 7th International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2004)(IEEE Computer Society Press), Hong Kong, May 2004 [该会议论文将被 ISTP 和 EI 检索]

[4] 华强胜,陈志刚. 一种基于主负载信息表动态负载模型及其均衡算法研究,计算机科学,2002,29(12):183-185

[5] 华强胜,陈志刚. 证件照片的一种混合人脸识别方法，计算机工程，2003,29(10):65-67

[6] 许伟，陈志刚，曾志文，华强胜. 分布式系统中主机负载预测的一种普适性方法，计算机工程与应用，2004,40(6):181-183