ISSN1000-9825 Journal of Software 软件学报 2000,11(8):1097~1104

基于DAG图解-重构的机群系统静态调度算法*

周佳祥 郑纬民

(清华大学计算机科学与技术系 北京 100084)

E-mail: zjx@est4.cs.tsinghua.edu.cn/zwm-dcs@mail.tsinghua.edu.cn

摘要 机群系统静态任务调度是NP-完全问题,通常的算法是通过一些启发式算法得到多项式次优解.该文提出的图解-子图重构算法实现了对分布在有向无环图(directed acyclic graph,简称DAG)上的并行任务的快速有效调度.该算法的复杂性为$O(\log^{|V|}\times(|V|+|E|))$,采用递归方法实现了对任务图的有效分解和子图重构,生成任务群,完成任务调度,并且初步实现了对处理机的优化.通过实例分析以及与其他启发式调度算法的性能比较,证明该算法是一种快速、有效、可行的任务调度算法.

关键词 任务调度,有向无环图,任务群,前驱任务,最优前驱任务,机群系统.

中图法分类号 TP316

随着VLSI技术、网络通信技术的发展,机群系统在并行领域的地位越来越突出,它在流体力学、气象模型、并行数据库系统、实时系统、有限元处理等领域中都取得了重大突破.为了能在机群系统中实现快速存取,提高加速比,需要一种有效的任务划分和调度策略.任务划分是将一个作业分成粒度合适的子任务,它们的关系可用具体的任务关系图——有向无环图(directed acyclic graph,简称DAG)来表示.任务的调度策略就是将DAG上的任务按一定的调度策略分配到机群系统的计算结点去执行.

静态任务调度是一个NP-完全问题,各种优化算法都是基于一定的限制条件得到的次优解.通常,可以通过启发式算法得到相应的次优解,得到复杂度为多项式的时间函数.最早的静态任务调度算法是基于优先级的调度算法[1~3].在该算法中,将任务按优先级放入任务池中,当有空闲处理机时,将当前优先级最高的任务提交运行.但它存在着致命的缺点,即没有考虑处理机之间的通信开销(在实际情况中,每一个作业被划分成若干个任务,各个任务之间可能存在通信和消息传递).而基于阈值调度算法[4]的思想,即设定任务结点的最早执行时间和最迟执行时间的间隔为阈值,用任务调度算法求得需要的处理机的个数,通过不断的调整来最后确定处理机的个数和调度方案.它的缺点是算法复杂性高,并且不一定得到次优解.

图解-子图重构算法是一种基于机群系统(network of workstations,简称NOW)的快速静态任务调度算法.其基本思想是,根据并行任务的相关性,利用搜索最佳前驱任务求得DAG主任务群,然后通过子图算法实现子图重构,再采用递归方法实现快速静态任务调度,最后,初步实现了处理机优化.该算法的特点是,在不增加算法复杂性的基础上,有效地减少了任务调度长度,实现了快速调度.

1 基本概念

1.1 概念和定义

任务关系图(tasks dependent graph)是将并行程序划分后得到的,在静态任务调度中可用图论的有向无环图来表示.为方便起见,将任务的有向无环图用四元组$TG=(V,E,G,D)$来表示.其中:

---

* 本文研究得到国家自然科学基金(No.69873023)、国家863高科技项目基金(No.863-306-ZD-02)资助.作者周佳祥,1973年生,博士,助教,主要研究领域为并行/分布系统的任务调度和负载平衡技术.郑纬民,1946年生,教授,博士生导师,主要研究领域为并行/分布处理技术.

本文通讯联系人:郑纬民,北京 100084,清华大学计算机科学与技术系应用教研组

本文1999-06-16收到原稿,1999-08 30收到修改稿

$V=[v_i]$表示任务图中结点(任务图中的结点是指任务)的集合,v_i表示第i个任务,$|V|$表示图中结点数目.

$E=[e_{i,j}]$表示任务图中边(任务图中的有向边是指两个任务之间存在消息传递)的集合,$e_{i,j}$是指结点i指向结点j的有向边,表示任务i和任务j存在相关性,$|E|$表示图中边的数目.

$G=[g_i]$表示任务的粒度(granularity),即子任务计算量的集合,g_i表示任务v_i的计算量.

$D=[d_{i,j}]$表示任务之间通信的数据量的集合,$d_{i,j}$表示任务i发送给任务j的数据量.

1.2 结点间关系

合结点(joint node):有多个后继结点的任务结点.

分结点(folk node):有多个前驱结点的任务结点.

孤结点(single node):图分解后剩下的孤独结点.

伪入(pseudo-entry)、伪出(pseudo-exit)结点:当前子图中的入、出结点.

请参见图6,结点6是合结点,结点1是分结点.在图5(c)子图3中,结点5是孤结点.在图5(b)子图2中,结点4是伪入结点,结点9是伪出结点.

结点i的前驱结点集 $\mathrm{pred}(i)=\{j|e_{j,i}\in E\}$;

结点i的后继结点集 $\mathrm{succ}(i)=\{j|e_{i,j}\in E\}$;

结点i的最早执行时间 $\mathrm{est}(i)=0$,if $\mathrm{pred}(i)=\varphi$或$\mathrm{est}(i)=\min\limits_{j\in \mathrm{pred}(i)}\max\limits_{\substack{k\in \mathrm{pred}(i)\\ k\neq j}}(\mathrm{ect}(j),\mathrm{ect}(k)+d_{k,i})$;

结点i的最早完成时间 $\mathrm{ect}(i)=\mathrm{est}(i)+g_i$;

结点i的最优前驱结点 $\mathrm{fpred}(i)=j|\mathrm{ect}(j)+d_{j,i}\geqslant(\mathrm{ect}(k)+d_{k,i})\ \forall\ j\in\mathrm{pred}(i),k\in\mathrm{pred}(i),k\neq j$;

结点i的次优前驱结点 $\mathrm{spred}(i)=j|\mathrm{ect}(j)+d_{j,i}\leqslant(\mathrm{ect}(k)+d_{k,i})$

$\forall\ j\in\mathrm{pred}(i),k\in\mathrm{pred}(i),k\neq j,k,j\notin\mathrm{fred}(i)$.

当存在两个或两个以上的最佳(或次优)前驱结点时,优先考虑通信量最大的结点,再考虑合结点,从而决定最佳(次优)前驱结点.

上述的任务结点关系可用深度优先搜索[5]方法得到.最优前驱结点的特征就是当处理机的个数足够多时,将有最优前驱关系的任务放到同一个处理机上调度,满足并行调度长度最短.

任务群(task cluster):经调度后分配在各个结点上的任务的集合.例如,机群系统由处理结点$P_1,\ldots,P_n$组成,提交的并行作业共派生任务$T_1,\ldots,T_m$,则当处理机1上分得的任务为$T_{1,1},\ldots,T_{1,k}$时,将$\{T_{1,1},\ldots,T_{1,k}\}$称为分配到处理机$P_1$上的任务群.

内部空闲时间片(intra idle time slots):处理机在系统调度长度范围之内的所有空闲时间片.

例如,在如图1所示的Gantt图中,阴影部分表示内部空闲时间片,结点1是初始输入结点,结点7是最终输出结点,而结点2是伪入结点,结点4是伪出结点.任务1、3、5、6、7构成主任务群,任务1、2、4构成了另一个任务群(统称为次任务群).

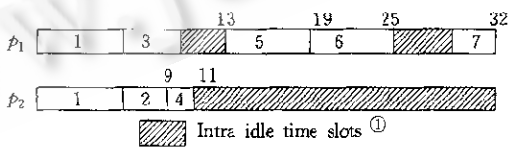


Fig. 1 Gantt chart in task scheduling

图1 任务调度的Gantt图

1.3 假设条件

• 在同构机群系统中,每个结点的计算能力完全一致,结点在机群拓扑中的地位也完全相同,并且暂时考虑到机群系统内的结点数足够被用来执行各任务群.

• 每个任务不可再分割,且每个任务是非抢先式的(nonpreemptive),任务一旦提交,就必须在运行完毕后才能由其他任务接着运行.

• 任务群一旦分配好,就需要全部放入结点,不可再被迁移.

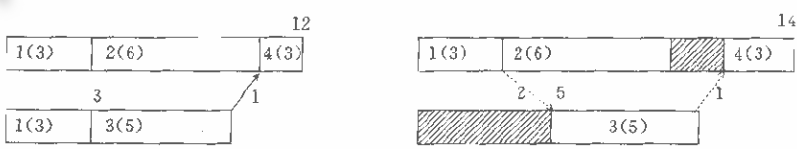
• 在同构机群系统中,考虑每个结点的计算能力完全一致,结点在机群拓扑中的地位也完全相同.因此,我们在下面的计算中用 g_i 表示任务 i 的执行时间,$d_{i,j}$ 表示任务 i 和任务 j 在不同机器上运行时存在的通信延时,在同一个结点运行时,通信延时为0.

• 每个DAG只有一个入结点,一个出结点.如果出现多个入结点时,可以加入一个空结点,它的相关后继结点是原图所有的入结点,并且相应的数据延时为0.同理,当出现多个出结点时,可以加入一个空结点,它的相关前驱结点是原图中所有的出结点,其相应的数据延时为0.

2 算法说明

图解-子图重构算法(简称图解-重构算法)是在系统提供足够处理机的基础上,采用Bottom-up方法进行搜索,从DAG的最终出结点开始,匹配当前任务的最佳前驱任务,生成主任务群;针对其子图的拓扑结构,通过复制某些已分配的前驱结点副本作为当前子图的元素(目的在于减少任务间的通信延时[4])来实现子图的补全和重组,再采用递归方式实现对各个连通子图的调度,生成次任务群,分配到新的处理机上,最后,初步实现处理机的优化,达到了快速而有效地调度静态任务的目的.

复制任务副本的优点是,通过某些任务副本在处理机的前端空闲时间片内运行,有效地减少了任务间的通信延时,优化了系统调度长度.例如,$m(n)$表示任务结点 m 的长度为 n,⟶表示任务之间存在通信.如图2(a)所示,采用任务副本调度方案得到的调度长度为12,"任务3"的执行时刻为3;如图2(b)所示,一般调度方案的调度长度为14,"任务3"的执行时刻为5.



(a) Scheduling on duplication tasks
(a) 引入任务副本的调度

(b) Common task scheduling
(b) 一般任务调度

Fig. 2
图2

2.1 图解-重构算法中提出的算法说明

2.1.1 子图重构算法

从图解后的某连通子图的伪入结点(pseudo_entry_node)出发,复制其最佳前驱结点(已分配)到当前子图,直到初始入结点,重构成一个完整的连通子图(指带有初始入结点的DAG子图).例如,在图2(a)中,在将结点1加入到结点3之前,构成了完整的连通了图,将该连通图依据Bottom-up搜索算法即可求得次任务群.

2.1.2 Bottom-up搜索算法

从当前(伪)出结点开始,考虑(子)图的某些结点受到其他输入结点的时间制约,在(子)图范围内寻找最佳前驱结点,递归搜索,直到初始入结点为止.

子图时间制约的确定:当某结点在DAG中的最佳前驱结点不在当前子图中时,该结点的调度存在着时间制约问题,即它的调度时刻受到来自于子图以外的其他任务的时间限制.例如,在如图5(b)所示的连通子图中,结点9受到子图以外的结点10的时间制约,而结点10是结点9在DAG中的最佳前驱结点.

2.1.3 结点优化算法

当拓扑结构中存在孤结点时,暂时不将它补全为子图,而是将它放入优化队列opt_queue之中.当其他子图都调度完毕时,将opt_queue中的元素插入到合适的任务群内部空闲时间片(intra idle time slots)中,达到优化处理机、充分利用系统资源的目的.

根据各任务群中任务的起始执行时间和完成时间,得到调度队列中存在的内部空闲时间片.采用下列步骤实现初步优化:

(1) 分配足够长的内部空闲时间片执行孤结点的运行.

(2) 当步骤(1)不满足时,若能在相关处理机中插入少许延时可实现同步执行,而对系统性能(调度长度)影响不大时,插入少许延时,即牺牲系统调度长度以获取整个系统的运行效率.例如,当插入两个内部空闲时间片可以换取节省12个时间片,且能为系统节省一台处理机时,对提高系统效率是可观的,在实际应用中是值得的.

(3) 若步骤(1)和步骤(2)不可行,则只能分配新的计算结点.

2.2 算法分析

2.2.1 算法复杂性的证明

由文献[5]可知,主任务群的搜索算法是一种深度优先搜索算法的延伸,它的算法复杂性为$O(|V|+|E|)$,子图采用的是递归算法,它的算法复杂性小于$O(|V|+|E|)$,DAG分解后的子图的个数为$O(\log|V|)$.所以,子图重构算法的复杂性为$O(\log|V|\times(|V|+|E|))$.

2.2.2 子图重构的意义

重构算法通过复制任务副本而不影响系统任务调度,确保了系统的任务调度长度尽可能地缩短.

证明:由图论知识可知[5],本算法在DAG中的主任务群的选取是一种类似关键路径的搜索算法,它的调度长度绝对小于在DAG中的最长路径的长度.前驱结点匹配算法保证了主任务群的执行时间就是系统的调度长度,而次任务群的搜索算法采用子图重构算法,充分利用处理机调度时存在的前端空闲时间片(用于执行任务副本),有效地减少了因任务间的通信开销所带来的对系统调度的延时,有助于尽早实现任务的调度. □

2.2.3 处理机优化算法分析

在图解-子图重构算法中,我们提出了处理机优化算法.它有重要的实际应用意义,对于提高系统的并行效率具有十分显著的作用.

优化算法的可行性证明

根据算法划分原则,由任务划分得到的任务群的(伪)出结点有很大一部分在原始任务结点图中是出度为1的结点,这在实际的任务划分中是客观存在的,符合实际并行模型.同时,各个并行子任务之间的关系比较复杂,从任务群的Gantt图表中可以看到,每个任务群中存在可观的内部空闲时间片,若能充分利用,将明显提高系统性能.可以证明,当系统的处理结点数减少m个,而系统的调度长度保持不变时,系统的效率将提高$\frac{100m}{n-m}\%$.

证明:因为该优化算法充分利用系统调度的内部空闲时间片,优化结果基本上不影响系统的调度长度,所以,优化前后的系统的加速比基本保持不变.

$$Speedup(n)=Speedup(n-m)=Speedup,$$

$$Eff(n)=\frac{100Speedup}{n}\%,$$

$$Eff(n-m)=\frac{100Speedup}{n-m}\%.$$

得证:

$$\Delta eff(n-m)=100\,\frac{Eff(n-m)-Eff(n)}{Eff(n)}\%=\frac{100m}{n-m}\%.$$ □

例如,当$n=4,m=1$时,$\Delta eff=25\%$.

2.2.4 算法描述

图解-重构算法伪代码

```
main ()
{
Input:DAG(B,E,G,D);
计算相关参数: pred(i),succ(i),fpred(i),spred(i),est(i),ect(i);
/* 利用 Bottom_up 方式进行任务调度 */
Task_scheduling(DAG);
```

```
处理机优化算法:
}
Task_scheduling(*Graph)   /*该算法是递归算法*/
{
 Task_clustering(*Graph);              /*To find the domestic cluster*/
 Reconfigure_to_subgragh(*Graph);   /*g_sum=number of the subgraghs*/
 for (i=1 to g_sum)
 {
       while (some task nodes left in subgragh_i)
       {
        Reconfigure_to_subgragh(*subgragh_i);
        Task_scheduling(*subgraph_i);
       }
  }
利用内部空闲时间将队列 opt_queue 的元素插入其他任务群;  /*实现处理机的初步优化*/
}
Task_clustering(*Graph)
{
 x=当前子图中的(伪)出结点;
 if (x 是孤结点)
   {将 x 放入队列 opt_queue;
        return();
   }
 {   y=fpred(x);
     if (y has been allocated)
       在当前子图中按"前驱结点原则"确定前驱结点;
         endif
         将结点 y 分配到任务群 C_i;  /*C_i 的任务最终被分配到处理机 P_i 上*/
         x=y
 }
 do while (x=(伪)入结点)
}
Reconfigure_to_subgragh (*Gragh)
{
在当前子图中删除已调度的任务结点,对某些时间受约的合结点表明受约时间及其关系;
根据当前拓扑结构形成子图;  /*采用子图形成算法*/
}
```

3 实验和比较

3.1 DAG 图和相关参数的计算

如图 3 和表 1 所示,结点$\left(\frac{m}{n}\right)$表示结点的任务号为 m,计算长度为 n,有向边表示结点之间存在通信,边上的数值表示通信长度.

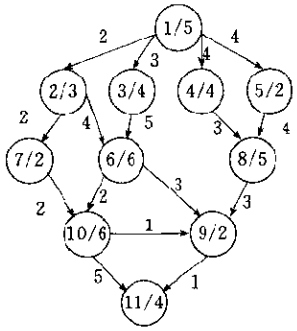


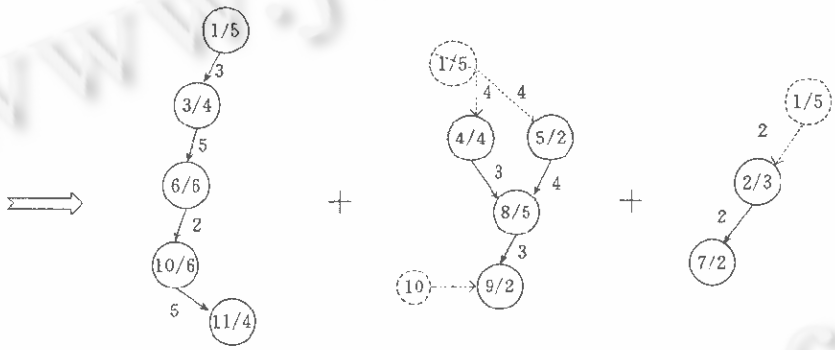
Fig. 3 DAG
图3 DAG图

Table 1 DAG argument table
表 1 DAG参数表

	est(i)	ect(i)	fpred(i)	spred(i)
1	0	5		
2	5	8	1	
3	5	9	1	
4	5	9	1	
5	5	7	1	
6	12	18	3	2
7	8	10	2	
8	11	16	4	5
9	24	26	10	8
10	18	21	6	7
11	27	31	10	9

3.2 图的分解和子图的生成过程

分解 1(如图 4 所示):



(a) Sub-graph 1
(a) 子图1

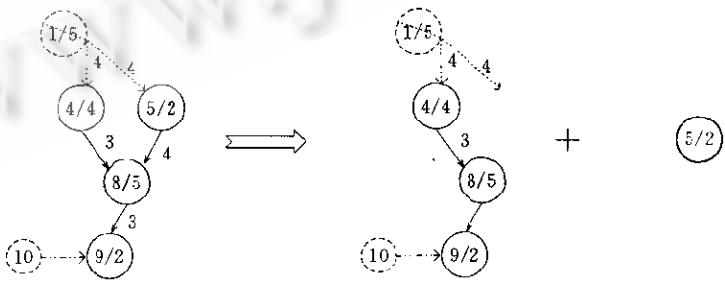
(b) Sub-graph 2
(b) 子图2

(c) Sub-graph 3
(c) 子图3

Fig. 4
图4

其中,图 4(a)表示主任务群,图 4(b)和图 4(c)表示两个重构后的子图,实线部分表示拓扑结构,虚线(·····>)表示采用重构法添加的子图,(—··—··→)表示受时间制约的任务结点.

分解 2(如图 5 所示):



(a) Sub-graph 2
(a) 子图2

(b) Sub-graph 2.1
(b) 子图2.1

(c) Sub-graph 2.2
(c) 子图2.2

Fig. 5
图5

图5表示图5(a)中子图2(即图4(b))的再重构过程.图5(b)是一个受时间制约的子图,图5(c)是孤结点,放入opt_queue队列以等待最后优化.

3.3 Gantt图说明时间调度结果

如图6所示,P_i表示机群系统的处理结点,……>表示任务间通信延时.

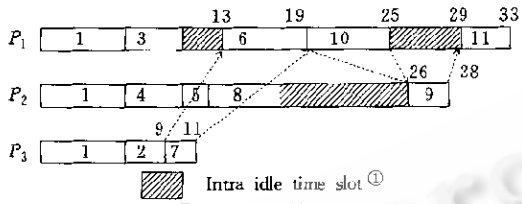


Fig. 6 Gantt chart in task scheduling
图6 任务调度的Gantt图

由Gantt图可知,主任务群(P_1)的调度长度是系统的调度长度;P_2,P_3上的任务结点1是副本任务,有效地利用了处理机的前端空闲时间片,保证了副本后继结点尽早执行;孤任务结点5放入次任务群(P_2)中,既充分利用了内部空闲时间片,又不对系统的调度长度产生影响,达到了优化处理机、提高系统性能的目的.但需要注意的是,任务结点9受到了结点10的时间制约,故而其执行时间比表1中的最早执行时间有延时.

3.4 性能评价与比较

将本算法与其他任务调度算法进行性能比较,见表2,其中

加速比 $Speedup=\frac{T_s}{T_p}$, T_s表示串行执行时间,T_p表示并行执行时间.

效率 $Efficiency=\frac{Speedup}{n\_proc}$, n_proc表示调度所用到的处理机个数.

该任务图的串行计算时间长度为44.

Table 2 Performance contrast among several scheduling algorithms
表2 调度算法的性能比较

Algorithm①	Length②	Processors③	Speedup④	Efficiency⑤	Algorithm complexity⑥
Linear clustering⑦	36	4	1.22	0.30	$O(\|V\|\times(\|V\|+\|E\|))$
Threshold scheduling[5]⑧	35	4	1.26	0.31	$O(\|V\|\|E\|+\|V\|PM)$*
Our algorithm⑨	33	3	1.37	0.44	$O(\log\|V\|\times(\|V\|+\|E\|))$
The optimum⑩	31	/	/	/	NP Complete

*: P means the number of processor, M means the number of threshold
*:P表示处理机个数,M表示阈值个数

①算法,②调度长度,③处理机数,④加速比,⑤效率,⑥算法复杂性,⑦线性分细算法,⑧阈值调度算法,⑨本算法,⑩理论最优.

加速比在很大程度上受制于通信时间在整个调度长度中的比重.考虑到任务的执行是非抢先式的,尤其是子任务的执行需要其前驱任务结点的数据全部到达后才能执行,在静态分配的子任务之间存在通信延时的制约和同步要求.这些因素制约了系统加速比的进一步提高.在上述例子中,由于通信开销和子任务的执行时间同属于一个数量级,故而其调度长度受到制约,加速比远小于处理机的个数.

在DAG中,$\log|V|\ll|V|<|E|\leqslant|V|^2$,所以,图解-子图重构算法在时间复杂性上小于线性算法和阈值算法.实验结果显示,本算法加速比也有了提高,尤其是在采用处理机优化之后,系统的并行效率有了明显的改进.

4 结束语

图解-子图重构算法利用了图论知识和DAG图的性质,从另一个角度考虑了静态调度的问题.本算法的实验结果显示,它在调度长度、并行效率和时间复杂性等并行性能方面有了明显的提高,是一种快速而有效的启发式调度算法.在实际应用中,该算法将为机群系统上的实时任务调度提供有力的理论基础和思路.

参考文献

1 Adam T L, Chandy K M, Dickson J R. A comparison of list schedules for parallel processing systems. Communications of the ACM, 1974,17(12):685~690

2 Gerasoulis A, Yang T. A comparison of clustering heuristics for scheduling directed acyclic graphs of multiprocessors. Journal of Parallel and Distributed Computing, 1992,16(4):276~291

3 Pande S S, Agrawal D P, Mauney J. A scalable scheduling method for functional parallelism on distributed memory multiprocessors. IEEE Transactions on Parallel and Distributed Systems, 1995,6(4):388~399

4 Kruatrachue B. Static task scheduling and grain packing in parallel processing systems [Ph. D. Thesis]. Oregon State University, 1987

5 Yan Wei-min. Data Structure. Beijing: Tsinghua University Press, 1996
(严蔚敏. 数据结构. 北京:清华大学出版社,1996)

6 Santoshkumar S P, Agrawal D P, Mauney J. A threshold scheduling strategy for sisal on distributed memory machines. Journal of Parallel and Distributed Computing, 1994,9(2):223~236

A Static Scheduling Algorithm on DAG Partition-Reconfiguration in the Network of Workstations

ZHOU Jia-xiang ZHENG Wei-min

(*Department of Computer Science and Technology Tsinghua University Beijing* 100084)

Abstract Static task scheduling on network of workstations is well-known to be an NP-complete problem in a strong sense. Some heuristic algorithms have been proven to be sub-optimal under some restrictive conditions. In this paper, the authors present a heuristic algorithm named DAG (directed acyclic graph) partition and subgraph reconfiguration algorithm, which is a fast and effective one used in parallel task scheduling. The complexity of this algorithm is $O(\log^{|V|}\times(|V|+|E|))$. It adopts recursion to implement DAG partition and sub-graph reconfiguration, then builds task clusters to carry out the task scheduling. At the same time, it even optimizes the number of processors to some degree for it has not been solved before. The performance has been observed in a representative example compared with other existing scheduling schemes in terms of several valuable factors. The experimental results show that this algorithm is feasible.

Key words Task scheduling, DAG (directed acyclic graph), task cluster, predecessor task, optimal predecessor task, NOW (network of workstations).