1000-9825/2003/14(05)0891 ©2003 Journal of Software 软 件 学 报 Vol.14, No.5

背包问题的最优并行算法*

李庆华+, 李肯立, 蒋盛益, 张 薇

(国家高性能计算中心(武汉),湖北 武汉 430074)
(华中科技大学 计算机科学与技术学院,湖北 武汉 430074)

An Optimal Parallel Algorithm for the Knapsack Problem

LI Qing-Hua+, LI Ken-Li, JIANG Sheng-Yi, ZHANG Wei

(National High Performance Computing Center (Wuhan), Wuhan 430074, China)
(School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China)
+ Corresponding author: Phn: 86-27-87544471, Fax: 86-27-87544471, E-mail: liqh@263.net
http://www.nhpcc.hust.edu.cn
Received 2002-10-28; Accepted 2002-12-11

Li QH, Li KL, Jiang SY, Zhang W. An optimal parallel algorithm for the knapsack problem. *Journal of Software*, 2003,14(5):891~896.
http://www.jos.org.cn/1000-9825/14/891.htm

Abstract: A new parallel algorithm for the knapsack problem is proposed, in which the method of divide and conquer is adopted. Based on an CREW-SIMD machine with shared memory, the proposed algorithm needs $O(2^{n/4})^{1-\varepsilon}$ processors, $0\le\varepsilon\le1$, and $O(2^{n/2})$ memory to find a solution for the *n*-element knapsack problem in $O(2^{n/4}\cdot(2^{n/4})^{\varepsilon})$ time. The cost of the algorithm is $O(2^{n/2})$, which is optimal and an improved result over the past researches. The wrong results in corresponding literatures are also pointed out in this paper.
Key words: knapsack problem; NP-complete; parallel algorithm; method of divide and conquer

摘 要: 利用分治策略,提出一种基于 SIMD 共享存储计算机模型的并行背包问题求解算法.算法允许使用$O(2^{n/4})^{1-\varepsilon}$个并行处理机单元,$0\le\varepsilon\le1$,$O(2^{n/2})$个存储单元,在 $O(2^{n/4}(2^{n/4})^{\varepsilon})$时间内求解 n 维背包问题,算法的成本为$O(2^{n/2})$.将提出的算法与已有文献结论进行对比表明,该算法改进了已有文献的相应结果,是求解背包问题的成本最优并行算法.同时还指出了相关文献主要结论的错误.
关键词: 背包问题;NP 完全;并行算法;分治法
中图法分类号: TP301 **文献标识码**: A

背包问题(也称子集和问题)可以描述如下:给定 n 个正整数 $W=(w_1,w_2,\ldots,w_n)$和正整数 M,要求寻找这样一个子集 $I\subseteq\{1,2,\ldots,n\}$,使得$\sum w_i=M,i\in I$.背包问题属于 NP 完全问题,直接的枚举搜索可能遍历问题的所有 2^n 个解

* Supported by the National Natural Science Foundation of China under Grant No.60273075 (国家自然科学基金); the National High-Tech Research and Development Plan of China under Grant No.863-306ZD-11-01-06 (国家高技术研究发展计划(863)); the National High Performance Computing Foundation under Grant No.99313 (国家高性能计算基金)

第一作者简介: 李庆华(1940－),男,湖北武汉人,教授,博士生导师,主要研究领域为并行处理,网格计算,组合优化.

空间,因此,直接搜索最坏情况下的时间复杂性为 $O(2^n)$.由于求解该问题的指数时间复杂性,背包问题在信息密码学领域和数论研究中有着极为重要的应用[1,2].因此,使用算法设计方法来降低求解该问题过于庞大的计算量,具有重要的理论和实际意义.

本文中,令 $N=2^{n/2},e=N^{1/2}=2^{n/4}$.分枝限界算法对于某些实例表现出较好的时间性能,但其最坏情形的时间复杂性仍为 $O(N^2)$,例如,文献[3]中提出的由市场共享问题产生的背包问题,使用分枝限界算法几乎不可能对它们求解.Horowitz 和 Sahni[4]利用分治方法提出了著名的二表算法,算法的时间和空间复杂性被分别降至 $O(N\log N)$ 和 $O(N)$.文献[4]基于二表算法,采用动态产生两个排序表中子集和元素的方式,使算法对存储空间的需求降为 $O(e)$,时间需求则保持不变.另外,使用归并方法,可以将其时间复杂性进一步减少至 $O(N)$,但空间复杂性则仍为 $O(N)$[3,5].上述算法虽然至今仍然是串行求解背包问题的最有效的算法,但对于实践应用中维数稍大的问题实例,尚不能在合理的时间内对其求解.

随着并行处理技术的日臻成熟,对背包问题的有效的并行算法的研究一度成为该领域的研究热点[5~8].Karnin[6]提出了一种基于 CREW-SIMD 共享存储计算模型的并行二表算法,算法使用 $O(N^{1/3})$的处理机和共享存储,在 $O(N)$的时间求解背包问题,因此,该算法的成本为 $C=T\times P=O(N^{4/3})$.1991 年,Ferreira[5]提出一种对共享存储器的空间需求为 $O(N)$、成本为 $O(N\log N)$的并行算法.文献[7]则使用 $O(N^{1/4})$的处理机,将对共享存储的需求减少至 $O(N^{1/2})$,但算法的成本却变为 $O(N^{5/4})$.其后,文献[6]利用文献[7]中第 1 阶段并行算法,对二表算法中表面上不可并行的第 2 阶段巧妙地并行化,改进了文献[7]的结果,并宣称其成本已达到串行背包算法的最低时间复杂性 $O(N)$,从而断定背包问题成本最优的并行算法已经找到.遗憾的是,由于文献[7]中对算法第 1 阶段时间复杂性的分析是错误的,使得文献[6]的结论同样是不可信的,即该算法并不是背包问题的成本最优算法.

本文利用最优并行归并算法,提出一种基于 CREW-SIMD 共享存储并行计算模型的新算法,算法中每个处理机都有 $O(1)$的局部存储空间和 $O(N)$的全局共享存储空间.算法允许使用 $O((2^{n/4})^{1-\varepsilon})$个处理机,$0\le\varepsilon\le 1$,在 $O(2^{n/4}(2^{n/4})^{\varepsilon})$时间内求解背包问题,其成本为 $O(N)$,因此,本文提出的算法是背包问题的最优并行算法.

本文第 1 节介绍作为本文算法基础的二表算法.第 2 节提出成本最优的并行算法.第 3 节是算法性能分析与比较,其中对文献[6,7]中主要结论的错误进行了深入分析.最后是对本文的总结和今后研究工作的展望.

1 二表算法

由于最优并行算法基于串行二表算法,因此首先简单叙述二表算法[4,6].

二表算法(the two list algorithm).

I. 生成阶段:

(1) 将 W 平均分成两部分 W_1 和 W_2,$W_1=(w_1,w_2,\ldots,w_{n/2})$,$W_2=(w_{n/2+1},w_{n/2+2},\ldots,w_n)$.

(2) 生成 W_1 的所有 N 个可能的子集之和,并将它们按非降序存放在表 $A=[A_1,A_2,\ldots,A_N]$中.

(3) 生成 W_2 的所有 N 个可能的子集之和,并将它们按非升序存放在表 $B=[B_1,B_2,\ldots,B_N]$中.

II. 搜索阶段:

(1) $i=1,j=1$.

(2) If $A_i+B_j=M$, then stop: a solution is found.

(3) If $A_i+B_j<M$, then $i=i+1$; else $j=j+1$.

(4) If $i>N$ or $j>N$ then stop: there is no solution.

(5) Go to (2).

实际上,可以将二表算法分成 3 个步骤((a),(b)属于第I阶段,(c)属于第II阶段):

(a) 生成表 A,B 中的所有子集之和;

(b) 对 A,B 中的所有元素排序;

(c) 对 A,B 进行搜索,以得到问题的解.

显然,像文献[6]严格执行上述步骤,算法在最坏情况下的时间复杂性为 $O(N\log N)$,但如果将第I阶段的步骤(a),(b)依下述方法一并执行,则算法的时间复杂性可降为 $O(N)$[3,5].从自然排序表$[0,w_1]$开始,将 w_2 依次加到表的

两个元素中得到表$[w_2,w_1+w_2]$,然后用与两个表的长度之和相同的时间对它们进行归并,再将 w_3 加到归并后的排序表中各元素上,再归并,…,最后就可以得到所需要的两个排序表 A,B.这样,其运行时间将减少为

$$\sum_{0\le i\le n/2-1} O(2^i)=O(N)\ .$$

步骤(c)显然能在 $O(N)$的时间内完成.因此,算法总的运行时间可达到 $O(N)$.

2 并行算法

Ferreira[5]的算法在很大程度上实现了对二表算法两个阶段的并行,但其两个阶段的成本都是 $O(N\log N)$,因此其计算成本不能达到最优.文献[7]对二表算法第I阶段使用动态产生表中元素的方法,使得并行算法对共享存储的需求降为$O(N^{1/2})$.虽然该文算法总的时间复杂性$O(N)$是正确的,但其对第I阶段时间复杂性的分析($O(n^2)$)则值得商榷.文献[6]基于文献[7]中第I阶段并行算法,对第II阶段实现了最优并行化.不过,由于存在与文献[7]同样的问题,而且其并行算法也不是最优算法,对该文结论的详细分析将在下一节给出.以下采用分治策略,借鉴文献[6]中第II阶段并行算法和 Ferreira 算法[5]的设计思想,提出背包问题的最优并行算法.

2.1 $P=O(N^{1/2})$

为了便于表达,我们先假定处理机数为 $P=O(N^{1/2})=O(e)$,依次考虑算法的两个阶段,即表 A,B 的生成阶段和搜索阶段.

2.1.1 第I阶段

因为二表算法中两个表 A,B 的生成在本质上完全相同,以下只描述按归并方式生成非降序排序表 A 的过程.考虑表 A 被生成的最后一步:设共享存储器中已保存的含 $N/2$ 个子集和元素的排序表 $A_{n/2-1}$ 为 $A_{n/2-1}=[A_{n/2-1.1},A_{n/2-1.2},\ldots,A_{n/2-1.N/2}]$,将 $w_{n/2}$ 利用 P 个处理机并行地加到该表的所有元素上,每个处理机内含有表 $A^1_{n/2-1}=[A_{n/2-1.1}+w_{n/2},A_{n/2-1.2}+w_{n/2},\ldots,A_{n/2-1.N/2}+w_{n/2}]$中的 $N/(2P)$个元素,设这样的 $N/(2P)$个元素组成表 $A^1_{n/2-1}$ 的一个块.使用文献[9]中类似的方法,对两个表 $A_{n/2-1},A^1_{n/2-1}$ 先划分,再归并.

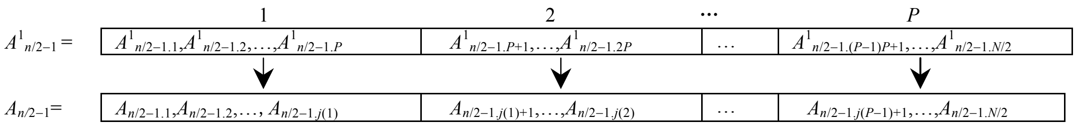

Fig.1 The partition and merger of list $A_{n/2-1}$ and $A^1_{n/2-1}$

图 1 表 $A_{n/2-1},A^1_{n/2-1}$ 的划分归并

如图 1 所示,首先利用表 $A^1_{n/2-1}$ 的 P 个块,将表 $A_{n/2-1}$ 对应地划分成 P 块,每块约含 $N/(2P)$个元素(如果出现某一块对中的一块的元素数超出 $N/(2P)$很多,则还必须对元素数较多的块继续划分),然后将 P 对彼此对应的块对并行归并,各处理器将归并后的结果写入共享存储器的不同地址单元,便可得到含 N 个子集和元素的排序表 $A_{n/2}$,即表A.由于对两个表进行划分时的折半查找允许对共享存储器的同一地址空间同时读取数据,因此执行这一过程的并行操作时间为$N/(2P)+2\log(N/2)+2N/(2P)=3N/(2P)+2log(N/2)$(虽然可能出现一次划分不成功的情况,即出现某一块对中的一块的元素数超出 $N/(2P)$很多的情况,但即便如此,其时间界将仍然保持不变[9]).如前所述,在排序表A,B生成过程中,最开始时为只有 1 个元素 0 的表,即 $A_0=[0]$,其次是两个元素的表 $A_1=[0,w_1]$,然后依次是 4 个元素的表 $A_2,\ldots,N/2$ 个元素的表 $A_{n/2-1},N$ 个元素的表 $A_{n/2}$,对前面的 $n/4$ 个表 $A_1,A_2,\ldots,A_{n/4}$,在与其对应的表(即加了相应 w_i 的表)归并时,可不必对它们划分,而使用折半查找,直接求其中一个表中的每一元素在另一表中的位序(rank).对后面的 $n/4$ 个表,则采用划分归并表 $A_{n/2-1}$ 和 $A^1_{n/2-1}$ 的方式进行,即先将对应的两个表并行划分,然后再将对应块并行归并.因此,排序表A,B的生成算法可描述如下:

表生成算法(the list generating algorithm).

```
begin
  for i = 1 to n/4 do
    do in parallel
      produce all of the subset sums of Ai in nondecreasing order.
    end do
  end for
```

for $i = n/4 + 1$ to $n/2$ do
  do in parallel
    produce all the subset sums of A^1_{i-1} by adding the value of w_i to the subset sums of A_{i-1}.
    partition two lists A_{i-1} and A^1_{i-1} into P blocks almost evenly.
    merge the corresponding blocks to obtain the sorted list A_i.
  end do
end for
end

2.1.2 第II阶段

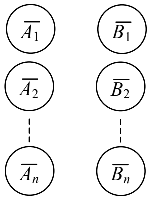

Fig.2 The block structure of the list A and B
图 2 表 A 和表 B 的分块结构

首先,如图 2 所示,将排序表 A,B 平均分成 P(或 e)块,使得每块都含有 $N/P=e$ 个按非降或非升序排列的元素,不妨将表 A,B 分别设为

$A=(\overline{A}_1,\overline{A}_2,\dots,\overline{A}_i,\dots,\overline{A}_e)$,其中 $\overline{A}_i=(\overline{A}_{i.1},\overline{A}_{i.2},\dots,\overline{A}_{i.e})$,$1\le i\le e$,$B=(\overline{B}_1,\overline{B}_2,\dots,\overline{B}_j,\dots,\overline{B}_e)$,其中 $\overline{B}_j=(\overline{B}_{j.1},\overline{B}_{j.2},\dots,\overline{B}_{j.e})$,$1\le j\le e$.

然后,将表 A 的第 i 块分配给处理机 i,$1\le i\le e$,如果不对表 A 和 B 的每一块作任何处理,则为求解背包问题,处理机 i 就应对表 B 的每一块进行搜索,但是,通过下述剪块规则,处理机 i 可只需对 B 的一小部分块搜索.为此,引入与文献[6]中引理 1 和引理 2 相似的下述引理,其证明与文献[6]中引理 1 和引理 2 的证明类似,本文从略.

引理 1. 对于表 A,B 的任意块对$(\overline{A}_i,\overline{B}_j)$,$1\le i,j\le e$,$\overline{A}_i=(\overline{A}_{i.1},\overline{A}_{i.2},\dots,\overline{A}_{i.e})$,$\overline{B}_j=(\overline{B}_{j.1},\overline{B}_{j.2},\dots,\overline{B}_{j.e})$,若 $\overline{A}_{i.1}+\overline{B}_{j.e}>M$,则子集和组合$(\overline{A}_{i.r},\overline{B}_{j.s})$不可能是背包问题的解,其中 $\overline{A}_{i.r}\in\overline{A}_i$,$\overline{B}_{j.s}\in\overline{B}_j$.

引理 2. 对于表 A,B 的任意块对$(\overline{A}_i,\overline{B}_j)$,$1\le i,j\le e$,$\overline{A}_i=(\overline{A}_{i.1},\overline{A}_{i.2},\dots,\overline{A}_{i.e})$,$\overline{B}_j=(\overline{B}_{j.1},\overline{B}_{j.2},\dots,\overline{B}_{j.e})$,若 $\overline{A}_{i.e}+\overline{B}_{j.1}<M$,则子集和组合$(\overline{A}_{i.r},\overline{B}_{j.s})$不可能是背包问题的解,其中 $\overline{A}_{i.r}\in\overline{A}_i$,$\overline{B}_{j.s}\in\overline{B}_j$.

应用引理 1 和引理 2 的结论,可得到如下剪块算法:

剪块算法(the prune algorithm).

begin
  Divide the two ordered lists, A and B, into e blocks, respectively.
  for all P_i where $1\le i\le P$ do
    for $j = 1$ to e do
      begin
        $X=\overline{A}_{i.1}+\overline{B}_{j.e}$;
        $Y=\overline{A}_{i.e}+\overline{B}_{j.1}$;
        if ($X=M$ or $Y=M$)
        then stop, a solution is found
        else if ($X<M$ and $Y>M$)
          then write $(\overline{A}_i,\overline{B}_j)$ to the shared memory;
      end
end

显然,剪块算法的时间复杂性为 $O(e)$,即 $O(2^{n/4})$.在剪块算法开始之前,所有可能的块对数为 $e\times e=N$,但在执行完该算法之后,下面的定理 1 则可以保证剩下的块对数最多为 $2e$.

定理 1. 在执行剪块算法之后,共享存储器中剩下的块对数最多为 $2e$.

证明:同文献[6]中定理 2 的证明. □

这样,由定理 1,在执行完剪块算法之后,将共享存储器中的块对平均分配到各处理机单元,此时每个处理机单元中的块对数最多为 2;最后,每个处理机对本处理机上的块对执行单方向的顺序搜索,以最终求出问题的解.

综合前面的表生成算法和剪块算法,背包问题的最优并行算法概略描述如下.

最优并行算法(the optimal parallel algorithm).

(1) Perform the list generating algorithm.

(2) Perform the prune algorithm.

(3) Equally assign the picked block pairs to each processor.

(4) for all P_i where $1 \le i \le P$ do

P_i performs the search routine of the two-list algorithm.

2.2 $P<O(N^{1/2})$

为了能用本文的算法处理维数较大的背包问题,我们将上述算法推广到处理机数 $P<O(N^{1/2})$的情形.设处理机数为 $P=e^{1-\varepsilon}$,$0 \le \varepsilon \le 1$.在步骤(1)的表生成阶段,首先,每个处理机并行地将 w_i 加到前一个已排序表 A_{i-1} 中,得到新表 A^1_{i-1},其中 $1\le i\le n/2-1$.其次,对于表对 A^1_i 与 A_i, $1\le i\le n(1-\varepsilon)/4$, $0<\varepsilon\le 1$,直接将它们并行归并得排序表 A_{i+1};而对于表对 A^1_j 与 A_j, $n(1-\varepsilon)/4<j\le n/2-1$,则必须先将它们并行划分成 P 块,,然后将划分后的 P 个块对并行归并,以生成排序表 A_{j+1}.将两个排序表 A,B 平均分成 P 块以后,步骤(2)利用 P 个处理机对所有可能的 P^2 个块对执行剪块算法;再将剩下的块对(其数量最多为 $2P$)平均分配给每一处理机(步骤(3));最后各处理机执行串行二表搜索,以最终确定问题的解(或判定问题无解).

3 算法性能分析与比较

3.1 性能分析

首先考虑处理机数为 $P=O(N^{1/2})$的情形,执行步骤(1)的并行表生成算法第 1 个循环的时间为 $O(n^2)$,第 2 个循环所需的并行操作时间为 $O\left(\sum_{i=n/4}^{n/2-1}(3\cdot 2^i/(2P)+2i)\right)=O(N/P)=O(2^{n/4})$;步骤(2)即剪块算法的运行时间为 $O(P)=O(2^{n/4})$;步骤(4)的执行时间为 $O(N/P)=O(2^{n/4})$.因此,算法总的时间复杂性为 $O(2^{n/4})$,并行计算的成本为 $C=T\times P=O(2^{n/4})\times O(2^{n/4})=O(2^n)$.

其次,若 $1\le P<N^{1/2}$,通过对本文并行算法中的表生成算法、剪块算法以及串行的二表搜索算法这 3 个组成部分的分析可知,此时并行背包算法的时间复杂性为 $O(N/P)+O(P)+O(N/P)$,注意到 $1\le P<N^{1/2}$,从而 $N/P>P$,故此时的时间复杂性将囿界于 N/P,即 $O(N/P)$,而算法的成本依然保持为 $C=O(N/P)\times O(P)=O(2^{n/2})$.

因此,对 $1\le P\le 2^{n/4}$ 的任意情形,本文算法的计算成本均可达到最优.

3.2 性能比较

文献[6]基于文献[7]中第 1 阶段的并行算法,声称已得到求解背包问题的成本最优并行算法,因此,首先分析文献[7]算法中主要结论的错误.文献[7]中利用 $2^{n/8}$ 个并行处理机单元动态地产生各含 $2^{n/2}$ 个子集和元素的表 A,B,却只用 $O((n/8)^2)$的时间,这显然是不可能的,因为当 n 较大时,其计算成本 $O((n/8)\cdot 2^{n/8})$将远远小于串行生成表 A,B 的最低运算量 $O(2^n)$.实际上,在深入分析文献[7]中的并行算法后不难发现,该算法除了将对共享存储器的空间需求减少至 $O(2^{n/4})$以外,在生成两个排序表 A,B 的第 1 阶段,即使使用 $2^{n/8}$ 个并行处理机单元,也不能减少算法的运行时间,即其时间复杂性仍为 $O(n2^{n/2})$,因此,文献[6]中并行背包算法的时间复杂性和成本也应分别为 $O(n2^{n/2})$和 $O(n2^{5n/8})$,而不是如该文所述的分别为 $O(2^{3n/8})$和 $O(2^n)$.因此,尽管文献[6]实现了二表算法中第 2 阶段的最优并行化,但总体上,该文中的算法仍不是成本最优的并行算法.

Karnin[6]用 $O(2^{n/6})$个处理机单元,在 $O(2^{n/2})$时间内求解背包问题,其成本为 $O(2^{2n/3})$;Ferreira 在文献[8]中提出的平衡三表算法和非平衡三表算法的处理机数、时间复杂性、并行成本分别为 $O(2^{n/3})$,$O(2^{n/3})$,$O(2^{2n/3})$与 $O(2^{n/4})$,$O(2^{3n/8})$,$O(2^{5n/8})$,在能见诸文献的背包问题并行算法中,Ferreira 在文献[5]中提出的算法是这些算法中成本最低的,仅为 $O(n2^{n/2})$.显然,即使和 Ferreira 的这一并行算法[5]相比,本文算法的计算成本也只及该算法成本的 $1/n$,而且同时保持了 Ferreira 的算法[5]中处理机个数可根据实际需要调整的优点,从而为利用本文算法并行求

解维数较大的背包问题提供了可能.例如:对维数 n=100 的背包问题,如果如像文献[6,7]中所描述的算法那样,就需要一个至少具有 $2^{12.5}$ 个处理机单元的 SIMD 并行计算机,显然,这将比串行求解该问题更加困难.

上述算法的性能和本文算法性能的比较结果见表 1.从表 1 可见,本文算法的性能明显优于已有的各种并行背包算法的性能.

Table 1 Comparisons of the parallel algorithms in solving the knapsack problem ($0\le\varepsilon\le1$)

表 1 背包问题各并行算法的性能对比 ($0\le\varepsilon\le1$)

Algorithms	Processor	Time	Memory	Cost	Technique type	Adaptability
Karnin[6]	$O(2^{n/6})$	$O(2^{n/2})$	$O(2^{n/6})$	$O(2^{2n/3})$	Dynamic generation	Not adaptable
Ferreira[8]1	$O(2^{n/3})$	$O(2^{n/3})$	$O(2^{n/3})$	$O(2^{2n/3})$	Dynamic generation and parallel search	Not adaptable
Ferreira[8]2	$O(2^{n/4})$	$O(2^{3n/8})$	$O(2^{3n/8})$	$O(2^{5n/8})$	Dynamic generation and parallel search	Not adaptable
Ferreira[5]	$O((2^{n/2})^{1-\varepsilon})$	$O(n(2^{n/2})^{\varepsilon})$	$O(2^{n/2})$	$O(n2^{n/2})$	Parallel generation and parallel search	Adaptable
Chang[7]	$O(2^{n/8})$	$O(n2^{n/2})$	$O(2^{n/4})$	$O(n2^{5n/8})$	Dynamic generation	Not adaptable
Lou[6]	$O(2^{n/8})$	$O(n2^{n/2})$	$O(2^{n/4})$	$O(n2^{5n/8})$	Dynamic generation and parallel search	Not adaptable
Ours	$O((2^{n/4})^{1-\varepsilon})$	$O(2^{n/4}(2^{n/4})^{\varepsilon})$	$O(2^{n/2})$	$O(2^{n/2})$	Parallel generation and parallel search	Adaptable

4 结 论

本文基于 CREW-SIMD 共享存储并行计算模型,利用分治策略,提出了一种求解背包问题的并行算法.算法允许使用$(2^{n/4})^{1-\varepsilon}$个并行处理机单元,$0\le\varepsilon\le1$,在 $O(2^{n/4}(2^{n/4})^{\varepsilon})$时间内求解背包问题,算法的成本为 $O(2^{n/2})$.若如 Schroeppel 和 Shanir[4]所断言:$T=O(2^{n/2})$是顺序求解背包问题的时间下界,则本文的算法就是求解此问题的最优并行算法.文中对已有求解背包问题的并行算法进行了深入的分析和对比,指出了相关文献中主要结论的错误.

另一方面,到目前为止,可使用 $O(2^{n/4})$的存储空间动态地产生两个排序表中的子集和元素,以 $O(n2^{n/2})$的时间串行求解背包问题.这样,一个自然的问题是,能否设计一种既可保持本文算法的处理机数可视实际需要调整的优点,又可将共享存储器的空间需求降低为$O(2^{n/4})$,同时算法的成本为$O(n2^{n/2})$或$O(2^{n/2})$的并行背包算法.尽管这样的算法不一定是背包问题的最优并行算法,并且事实上单元存储器的成本也远低于单元处理器的成本,但对于求解时间和空间都呈指数增长的 NP 完全问题而言,研究这一问题仍然具有理论和实际意义.

References:

[1] Chor B, Rivest RL. A knapsack-type public key cryptosystem based on arithmetic in finite fields. IEEE Transactions on Information Theory, 1988,34(5):901~909.

[2] Laih C-S, Lee J-Y, Harn L, Su Y-K. Linearly shift knapsack public-key cryptosystem. IEEE Journal of Selected Areas Communication, 1989,7(4):534~539.

[3] Dantchev S. Improved sorting-based procedure for integer programming. Mathematical Programming, Serial A, 2002,92:297~300.

[4] Schroeppel R, Shamir A. A $T=O(2^{n/2})$, $S=O(2^{n/4})$ algorithm for certain NP-complete problems. SIAM Journal of Computing, 1981, 10(3):456~464.

[5] Ferreira AG. A parallel time/hardware tradeoff $T\cdot H=O(2^{n/2})$ for the knapsack problem. IEEE Transactions on Computing, 1991, 40(2):221~225.

[6] Lou DC, Chang CC. A parallel two-list algorithm for the knapsack problem. Parallel Computing, 1997,22:1985~1996.

[7] Chang HK-C, Chen JJ-R, Shyu S-J. A parallel algorithm for the knapsack problem using a generation and searching technique. Parallel Computing, 1994,20(2):233~243.

[8] Ferreira AG. Efficient parallel algorithms for the knapsack problem. In: Cosnard M, Barton MH, Vanneschi M, ed. Proceedings of the IFIP WG 10.3 Working Conference on Parallel Processing. North-Holland: Elsevier Science Publishers, 1988. 169~179.

[9] Cheng GL. The Design and Analysis of Parallel Algorithms. Beijing: Higher Education Press, 1994. 35~37 (in Chinese).

附中文参考文献:

[9] 陈国良.并行算法的设计与分析.北京:高等教育出版社,1994.35~37.