多处理器任务调度算法TDS的建模与验证

李召妮 雷丽晖 李永明
(陕西师范大学计算机科学学院 西安 710062)

摘 要 在多处理器系统中,一个应用所要完成的任务可以分配给同一个处理器处理,也可以分配给多个处理器处理,所以传统的测试方法难以满足多处理器任务调度算法的验证。在此,提出一个基于扩展Büchi自动机的形式化模型,并用该模型来描述多处理器任务调度算法TDS(Task Duplication based Scheduling);用线性时序逻辑描述出算法TDS期望的一些性质;最后在该模型上验证了这些性质。该方法有效地克服了传统测试的局限性,保证了多处理器任务调度的可靠性。

关键词 多处理器调度算法,线性时序逻辑,模型检测,扩展Büchi自动机
中图法分类号 TP301.1 **文献标识码** A

Modeling and Verifying Scheduling Algorithm TDS for Multiprocessor Systems

LI Zhao-ni LEI Li-hui LI Yong-ming
(School of Computer Science, Shaanxi Normal University, Xi'an 710062, China)

Abstract In multiprocessor systems, the tasks which need to be completed by an application can be distributed onto the same processor or a group of processors. Therefore, software testing is unable to verify the reliability of multiprocessor scheduling algorithms. Thus a formal model based on extended Büchi automata which is used to describe the TDS (Task Duplication based Scheduling) algorithm was presented. The expected properties of the algorithm were described by linear temporal logic formulas and verified on the above model. This method overcomes the limitations of software testing and can be used to check the reliability of the multiprocessor task scheduling TDS.

Keywords Multiprocessor scheduling algorithm, Linear temporal logic, Model checking, Extended büchi automata

1 引言

随着科技的发展,目前多处理器已应用到我们的计算机中,特别是已广泛应用于服务器领域,而多处理器的任务调度问题是影响系统性能的关键问题。多处理器任务调度的目标是:按照某种策略将每个任务合理分配到各个处理器上并行有序地执行,以减少并行运行任务间的通信开销和等待时间,使任务的执行时间最短。已经证明,基于任务复制的调度算法比无任务复制调度算法具有更好的性能。TDS算法就是典型的任务复制算法。

由于多处理器上任务之间的执行已经实现了真正意义上的并发,因此传统测试方法很难满足多处理器任务调度验证的需要。与测试不同,形式化验证采用数学的方法,在对软件系统进行规约的基础上分析系统是否具有所期望的性质,这种方法已成为最有希望解决多处理器任务调度验证的途径。常见的形式化验证方法主要有两种:定理证明和模型检测。

定理证明是系统及其特性以某种数学逻辑公式表示的技术,它实质上是从系统公理中寻找特性证明的过程。定理证明具有强大的逻辑表达能力,可直接处理无限的状态空间,但它不适合做时态方面的推理且自动化程度低;模型检测是一种基于有限模型并检测该模型的期望性质的一种技术,它的实质就是在状态空间中蛮力搜索,模型的有限性确保了搜索可以终止。模型检测的优点在于:自动化程度高,当系统不具备所期望的性质时,能给出相应的反例路径,其缺点是状态爆炸的问题。

本文为多处理器调度算法TDS建立了一个数学模型,即多处理器环境下有序任务处理自动机;使用线性时序逻辑来描述算法所期望的性质,将自动机模型转换成Kripke结构并应用不动点理论进行验证;还结合具体的实例展示了基于自动机和线性时序逻辑的模型检测方法在验证多处理器任务调度算法上的有效性。

2 TDS算法介绍[1]

TDS算法的输入是DAG图(有序任务图),输出是任务

到稿日期:2012-04-15 返修日期:2012-06-01 本文受国家自然科学基金(60873119,61003061),高校博士学科点专项科研基金(20090202120006),中央高校基本科研业务费(GK200902017,GK201001003)资助。

李召妮(1982—),女,硕士生,主要研究方向为模型检测,E-mail:lizhaoni01@163.com;雷丽晖(1976—),女,博士,副教授,CCF会员,主要研究方向为软件工程、可信理论与技术、网络计算;李永明(1966—),男,博士,教授,博士生导师,CCF会员,主要研究方向为非经典计算理论、量子逻辑、量子信息等。

簇，其算法的主要思想是：将 DAG 图中 join 结点与其友好前驱结点分配到同一处理器，以降低并行执行时间。

(1) DAG 图

将一批有序任务用 DAG 图表示，它是 TDS 算法的输入，即 $G=(V,E,W,C)$，这里：

$V=\{n_i \mid n_i$ 是有序任务，$i=1,2,3,\cdots,v, v=|V|$ 表示任务的数目$\}$；

$E=\{e_{i,j} \mid e_{i,j}$ 表示任务 n_i 到 n_j 的边$\}$；

$W=\{w_i \mid w_i$ 表示任务 n_i 的计算时间开销，$i=1,2,3,\cdots\}$；

$C=\{c_{i,j} \mid c_{i,j}$ 表示任务 n_i 到 n_j 的通信时间开销$\}$；

$pred(n_i)=\{j \mid e_{j,i}\in E\}$，$succ(n_i)=\{j \mid e_{i,j}\in E\}$；

如果 n_i 没有前驱任务，则 n_i 为开始任务，如果 n_i 没有后继任务，则 n_i 为结束任务。

不失一般性，假定 DAG 图仅有一个开始任务和一个结束任务。

(2) 算法调度所需参数的说明

$EST=\{est_i \mid est_i$ 表示第 i 个任务的最早可能开始时间$\}$，

$est(i)=1$ if $pred(i)=\phi$，

$est(i)=\min\{\max_{j\in pred(i),k\in pred(i),k\neq j}\{ect(j),ect(k)+c_{k,i}\}\}$

if $pred(i)\neq\phi$；

$ECT=\{ect_i \mid ect_i$ 表示第 i 个任务的最早可能结束时间$\}$，

$ect(i)=est(i)+w(i)$；

$FPRED=\{fpred_i \mid fpred_i$ 表示第 i 个任务的友好前驱任务对应的任务序号$\}$，

$\forall j\in pred(i),k\in pred(i),k\neq j, fpred(i)=k \mid (ect(j)+c_{j,i})\leqslant(ect(k)+c_{k,i})$；

$LACT=\{lact_i \mid lact_i$ 表示第 i 个任务的最晚可能结束时间$\}$，$lact(i)=ect(i)$ if $succ(i)=\phi$，否则 $lact(i)=\min\{\min_{j\in succ(i),i\neq fpred(j)}(last(j)-c_{i,j}),\min_{j\in succ(i),i=fpred(j)}(last(j))\}$；

$LAST=\{last_i \mid last_i$ 表示第 i 个任务的最晚开始时间$\}$，$last(i)=lact(i)-w(i)$；

$LEVEL=\{level_i \mid level_i$ 表示第 i 个任务到结束任务的最大长度$\}$，

$level(i)=w(i)$ if $succ(i)=\phi$，$level(i)=\max_{k\in succ(i)}(level(k))+w(i)$ if $succ(i)\neq\phi$。

(3) 基于(2)计算出来的参数将任务分簇

输入：DAG$(v,e,\tau,c_{i,j})$，*queue*：按 *level* 值的升序存放了所有的任务，以及 *pred(i)*：任务 i 的前驱任务集合，最后还有 *succ(i)*：任务 i 的后继任务集合；

输出：任务簇，具体的分簇方法请看下面的伪码

```
x=queue 中的第一个元素；
j=1;/* 处理器标号从 1 开始 */
将 x 分配给 pj；
  While(不是所有的任务已分配了处理器){
    y=fpred(x);
    if (已经将 y 分配到另外的处理器处理){
      if((last(x)-lact(y))>=cy,x) then /* y 对于 x 来说不是关键的 */
        y=x 的另一个还没有分配给处理器的前驱任务
      else 对于 x 的另一前驱任务 z,z≠y;
        if ((ect(y)+cy,x)=(ect(z)+cz,x)并且 z 还没有分配给任何的处理器),then y=z
        endif
      endif
    }
  endif
  将 y 分配给 pj;x=y;
    if (x 是开始任务)
    将 x 分配给 pj;x= queue 中还没有被分配处理器的下一个任务;
    j++;将 x 分配给 pj;
  endif }
```

3 多处理器调度算法 TDS 的模型

3.1 TDS 算法的模型

定义 1 TDS 算法对应的自动机 A 为一个九元组 $A=(AP,L,N,Q,s_0,\delta,F,G,G_a)$，其中：

(1) $AP=\{n_1,n_2,\cdots,n_v\}$ 为任务的集合，表示一段时间里要处理的一些有序任务；

(2) 赋值集 $L=\{0,e,w,r,1\}$，0 表示任务未就绪，e 表示任务正在与前驱任务通信，w 表示任务与自己某个前驱任务通信结束后空闲等待，r 表示任务正在被计算；1 表示任务处理结束；

(3) Q 是有限状态集合，$Q\subset L^{AP}$，即 AP 到 L 的部分映射。$\forall s\in Q, n_i\in AP, l_i\in L, \frac{l_i}{n_i}$ 表示第 i 个任务 n_i 的值为 l_i，记 $s=\frac{l_1}{n_1}+\frac{l_2}{n_2}+\frac{l_3}{n_3}+\cdots+\frac{l_v}{n_v}$，为了方便表示，再记 l_i 为 q_i；

(4) **N** 为自然数的集合，表示系统时间(为了方便，这里使用的是相对时间)；

(5) $s_0\in Q$ 为初始状态，$s_0=\frac{0}{n_1}+\frac{0}{n_2}+\frac{0}{n_3}+\cdots\frac{0}{n_v}$，表示没有一个任务就绪；

(6) $F\subseteq Q$ 为接收状态集，$\forall s\in F, s=\frac{1}{n_1}+\frac{1}{n_2}+\frac{1}{n_3}+\cdots+\frac{1}{n_v}$，表示所有任务处理结束；

(7) 将一组有序任务表示为一个有向无环图(DAG)，G 参考第 2 节中的(1)；

(8) 任务系统的每个任务分配到处理器之后，分配时序情况可以用一个多元组来表示为

$G_a=(P, EST, ECT, FPRED, LACT, LAST, LEVEL, T, STIME)$

其中：

$P=\{p_i \mid p_i$ 表示第 i 个处理器，$i=1,2,3,\cdots,m, m$ 为处理器数目$\}$；

$EST, ECT, FPRED, LACT, LAST, LEVEL$ 的表达式参考第 2 节中的(2)；

$T=\{t_{i,j} \mid t_{i,j}$ 对应的值表示分配到处理器 p_i 的第 j 个任务对应的任务序号$\}$，$t(i)(j)=f(P, FPRED, LEVEL,$

$LAST, LACT, ECT, EST, C)$。

具体的任务分簇方法参见文献[1]，需要说明的是文献[1]中只做了如何把任务分配给每个处理器，所以这里在做的时候只需要同时记录每个任务是处理器上的第几个任务即可；

$STIME=\{stime_i \mid stime_i$ 表示第 i 个任务开始被计算的时刻$\}$

1)如果当前任务 i 是处理器的第一个任务，为了处理方便，默认在时刻 1 开始运行；

2)如果当前任务 i 不是处理器的第一个任务，则 $\forall j \in pred(i)$，且 j 和 i 不在同一个处理器上计算，而且 $\exists k, k$ 是 i 在同一处理器上的前驱任务，则

$$stime(i)=\max\{\max(stime(j)+w(j)+c_{j,i}), (stime(k)+w(k))\}$$

(9) 任务处理函数为映射 $\delta: Q\times N\to Q$，系统时间为 t 时，有 $\delta(s,t)=s'$，其中 $q_i' \in s', q_i \in s$ 则：

1)$q_i'=e$，当 $\forall q_i=0$ 且 $i\neq 1$ 时，如果 $\forall j \in pred(i)$. min $(stime(j)+w(j)) \leqslant t$，这里 i, j 在不同的处理器上，或当 $\exists q_i=w$ 时，如果 $\exists k \in pred(i), k\neq j$, s. t. $stime(k)+w(k)\leqslant t, i, k$ 在不同的处理器上；

2)$q_i'=w$，当 $\exists q_i=e$，且($\exists k, j \in pred(i)$，且 $k\neq j, stime(j)+w(j)+c_{j,i}\leqslant t\leqslant stime(k)+w(k)$)，$i, j, k$ 分别在 3 个不同的处理器上，或者 i, k 在同一处理器上，j 在另一处理器上；

3)$q_i'=r$，当 $q_1=0$，或当 $\exists q_i=e$, s. t. $stime(i)\leqslant t<stime(i)+w(i)$，或当 $\exists q_i=w$ 并存在这样的情况：$\forall j \in pred(i)$ $stime(j)+w(j)+c_{j,i}<t, i, j$ 在不同的处理器上，且 $\exists k \in pred(i), k\neq j, stime(k)+w(k)\leqslant t, i, k$ 在同一处理器上；

4) $q_i'=1$，当 $\forall q_i=r$, s. t. $stime(i)+w(i)\leqslant t$；

5) $s'=\frac{0}{n_1}+\frac{0}{n_2}+\frac{0}{n_3}+\cdots\frac{0}{n_v}$，当 $s=\frac{1}{n_1}+\frac{1}{n_2}+\frac{1}{n_3}+\cdots+\frac{1}{n_v}$，且 $\forall q_i, stime(i)+w(i)<$t 时；

6) 否则，$q_i'=q_i$。

3.2 TDS 算法的模型所具有的性质

(1)根据这个计算模型可以看出这个扩展的 Büchi 自动机是确定性的。输入的字符是系统时间 **N**，只有一个初始状态 s_0 且 $|\delta(s,r)|=1$，其中 $s_0 \in Q, r \in \mathbf{N}$。所以，TDS 算法模型是一个扩展确定性 Büchi 自动机(EDBA)。

(2)可以看出，从 s_0 开始经过状态转移到可接收状态 s 的状态序列是可以被此 EDBA 所接收的，所有满足上述条件的状态序列的集合就构成了此 EDBA 所接收的语言。其中，$s_0=\frac{0}{n_1}+\frac{0}{n_2}+\frac{0}{n_3}+\cdots\frac{0}{n_v}, s=\frac{1}{n_1}+\frac{1}{n_2}+\frac{1}{n_3}+\cdots+\frac{1}{n_v}$。

(3) EDBA 的 AP, L, G, G_a 虽然决定了模型状态的个数(状态的数量是有限的)，但不会影响 EDBA 的性质。所以，EDBA 仍是一类确定性 Büchi 自动机，所有确定性 Büchi 自动机的理论同样适用于 EDBA。

3.3 实例

例 1　下面图 1 中，圆圈中的 $i(i=1,2,3,\cdots,10)$表示任务的结点编号，圆圈旁的数字表示任务的计算时间，带箭头线上的数字表示两个任务间的通信时间。

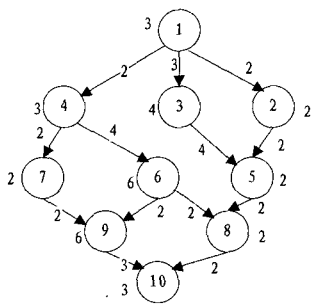

图 1　有序任务的 DAG 图

任务的时序处理器分配情况如表 1 所列。

表 1　任务分配及开始和完成时间表

Nodel	level	est	ect	last	lact	fpred	t	stime
1	21	1	4	1	4	—	$t_{11}, t_{31}, t_{21}, t_{41}$	1
2	9	4	6	7	9	1	t_{32}	4
3	11	4	8	7	11	1	t_{22}	4
4	18	4	7	4	7	1	t_{12}, t_{42}	4
5	7	8	10	11	13	3	t_{23}	8
6	15	7	13	7	13	4	t_{13}	7
7	11	7	9	9	11	4	t_{43}	7
8	5	13	15	15	17	6	t_{24}	15
9	9	13	19	13	19	6	t_{14}	13
10	3	19	22	19	22	9	t_{15}	19

说明：t_{14} 表示任务 9 被分配到了第 1 个处理器的第 4 个任务，其他的以此类推。

处理任务 $n_1, n_2, \cdots, n_{10}$ 的状态转移图如图 2 所示，箭头中间数字就是系统的相对时间。

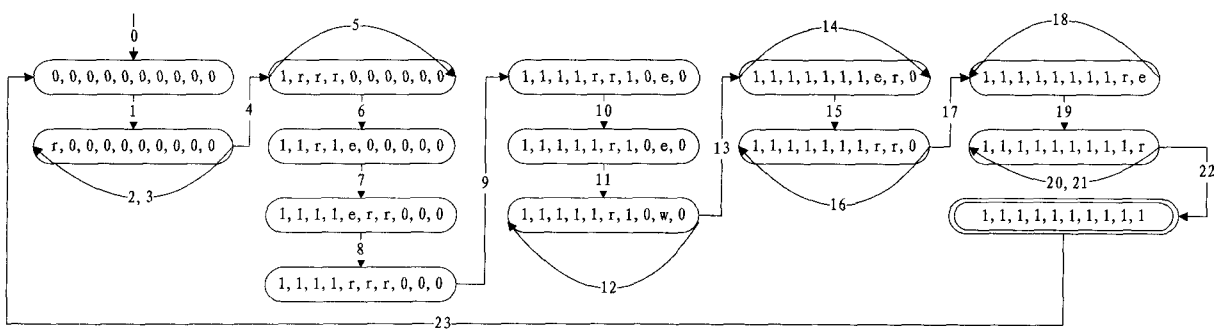

图 2　状态转移图

3.4 性质描述

我们要验证的是 TDS 算法的安全性和活性。安全性是指一个系统模型中坏的事情一定不会发生，可用线性时序逻辑表示为$\square\neg(p)$，这里的 p 描述一件坏事情。活性则是指

在一个系统中好的事情一定能发生，可用时序逻辑表示成$\Diamond q$，q表示好的事情。判定TDS算法是否合理有效，可以参考以下性质是否能被满足：

(1) 如果一个任务的前驱任务没有处理完，那么它不能开始进行计算(安全性)。

LTL描述如下：$\forall j \in pred(i).\ \Box\neg(\neg 1_j \wedge r_i)$，$i,j$是任务序号。

(2) 对于一批有序任务，其中每一个任务都是可以处理结束的(活性)。

LTL描述如下：$\forall i, \Diamond 1_i$，$i=1,2,3\cdots$是任务的序号。

4 对自动机模型进行模型检测

4.1 扩展 Büchi 自动机对应的 Kripke 结构

定义 2 一个扩展Büchi自动机对应一个Kripke结构$K=(AP,S,s_0,R,L_s)$，其中：

AP是一批有序任务的集合，$AP=\{n_1,n_2,\cdots,n_v\}$；

S是有限状态的集合，$S=Q$；

$s_0 \in S$是初始状态；

$R \subseteq S \times S$是状态转移关系，这里要求R是一个全关系，即$\forall s \in S, \exists s' \in S$，使得$(s,s') \in R$；

$L_s: S \to 2^{AP}$为状态标识函数，$\forall s \in S$，

$$L_s(s)=\begin{pmatrix} a_{1,1} & \cdots & a_{1,v} \\ a_{2,1} & \cdots & a_{2,v} \\ a_{3,1} & \cdots & a_{3,v} \\ a_{4,1} & \cdots & a_{4,v} \\ a_{5,1} & \cdots & a_{5,v} \end{pmatrix}$$

$a_{1,i}$表示第i个任务在该状态取值为“0”时的真值(0|1)；

$a_{2,i}$表示第i个任务在该状态取值为“e”时的真值(0|1)；

$a_{3,i}$表示第i个任务在该状态取值为“w”时的真值(0|1)；

$a_{4,i}$表示第i个任务在该状态取值为“r”时的真值(0|1)；

$a_{5,i}$表示第i个任务在该状态取值为“1”时的真值(0|1)。

根据定义2可以将定义1描述的扩展Büchi自动机模型抽象成Kripke结构。对于定义2中的L_s的定义，$\forall s_i \in S$计算$L_s(s_i)$，这样可把多值向量$(0,e,w,r,1)$编码成二值$(0,1)$矩阵。可以看出定义1与定义2所定义的状态空间相同且状态转移一一对应。

4.2 用不动点进行模型检测

由于$\Box\neg f=\neg\Diamond f$，要验证模型能够满足性质$\Box\neg f$，就可以转换成验证模型不能满足$\Diamond f$。又$\Diamond f=A[True \cup f]$，所以要验证的公式如下。

性质 1(安全性的否定式) $\Diamond f_1=A[True \cup f_1]$，其中：$\forall j=pred(i)$，$f_1=\bigvee\limits_{i=2,3,\cdots,v}(a_{4,i} \wedge \neg a_{5,j})$。

$A[True \cup f_1]$表示无约束的在路径上的每一个状态成立直到路径上出现一个状态满足f_1，其中f_1表示至少有一个任务已经开始计算，但是它的前驱任务并非全部处理结束。TDS算法不满足f_1就等价地满足了上述安全性的要求。

性质 2(活性) $A[True \cup f_2]$，其中：$f_2=a_{5,1} \wedge a_{5,2} \wedge \cdots \wedge a_{5,v}$。

$A[True \cup f_2]$表示无约束的在路径上的每一个状态成立直到路径上出现一个状态满足f_2，其中f_2表示所有任务处理结束，TDS算法满足f_2就等价地满足了上述活性的要求。

检测的具体方法[4]：

(1)根据4.1节生成的Kripke结构分别计算$A[True \cup f_1]=\mu Z.\ f_1 \vee (True \wedge AXZ)$，$A[True \cup f_2]=\mu Z.\ f_2 \vee (True \wedge AXZ)$。

(2)如果$S_0 \notin A[True \cup f_1]$且$S_0 \in A[True \cup f_2]$，则算法是合理的，否则不合理。

算法的复杂度[6] 不动点模型检测算法的复杂度为$O(|f| \cdot (|S|+|R|))$，其中，$|f|$是f子表达式个数，$|S|$是状态数，$|R|$是状态迁移数。

例 2 对例1中的模型应用不动点算法来进行检测：

(1) $L_s(S)$的值计算如下：

$$L_s(s_0)=L_s(\langle 0,0,0,0,0,0,0,0,0,0\rangle)=\begin{pmatrix} 1 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

$$L_s(s_1)=L_s(\langle r,0,0,0,0,0,0,0,0,0\rangle)=\begin{pmatrix} 0 & 1 & \cdots & 1 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \end{pmatrix}$$

$$\vdots$$

$$L_s(s_{13})=L_s(\langle 1,1,1,1,1,1,1,1,1,1\rangle)=\begin{pmatrix} 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ 1 & 1 & \cdots & 1 \end{pmatrix}$$

(2) 图2生成的Kripke结构如图3所示。

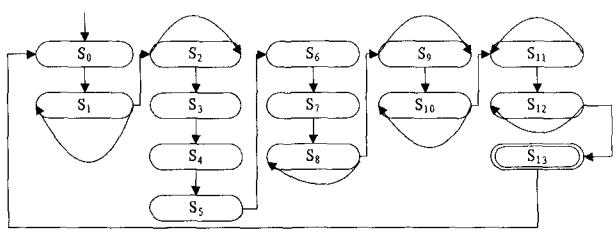

图3 Kripke结构

(3)要验证的性质描述，$A[True \cup f_1]$和$A[True \cup f_2]$，其中：

$f_1=(a_{4,2} \wedge \neg a_{5,1}) \vee (a_{4,3} \wedge \neg a_{5,1}) \vee (a_{4,4} \wedge \neg a_{5,1}) \vee (a_{4,5} \wedge (\neg a_{5,2} \vee \neg a_{5,3})) \vee (a_{4,6} \wedge \neg a_{5,4}) \vee (a_{4,7} \wedge \neg a_{5,4}) \vee (a_{4,8} \wedge (\neg a_{5,5} \vee \neg a_{5,6})) \vee (a_{4,9} \wedge (\neg a_{5,6} \vee \neg a_{5,7})) \vee (a_{4,10} \wedge (\neg a_{5,8} \vee \neg a_{5,9}))$

$f_2=a_{5,1} \wedge a_{5,2} \wedge \cdots \wedge a_{5,10}$

(4) 最小不动点计算，计算过程中公式看作满足它的状态之集；逻辑运算$\vee\ \wedge\ \neg$对应集合运算$\cup\ \cap\ -$。

(a)$\tau(Z)=\mu Z.\ f_1 \vee (True \wedge AXZ)$的最小不动点计算如下：

$\tau^0(\emptyset)=\emptyset$

$\tau^1(\emptyset)=f_1 \vee (True \wedge AX(\tau^0(\emptyset)))=\{\emptyset\} \cup (S \cap AX\{\emptyset\})=\{\emptyset\}$

⋮

可以看出，计算的 τ 值全是 $\emptyset$，所以 $S_0 \notin A[True \cup f_1]$，该模型满足上述定义的安全性要求。

(b)$\tau(Z)=\mu Z.\ f_2 \vee (True \wedge AXZ)$的最小不动点计算如下：

$\tau^0(\emptyset)=\emptyset$

$\tau^1(\emptyset)=f_2 \vee (True \wedge AX(\tau^0(\emptyset)))=\{s_{13}\} \cup (S \cap AX\{\emptyset\})=\{s_{13}\}$

$\tau^2(\emptyset)=f_2 \vee (True \wedge AX(\tau^1(\emptyset)))=\{s_{13}\} \cup (S \cap AX\{s_{13}\})=\{s_{12},s_{13}\}$

⋮

$\tau^{14}(\emptyset)=f_2 \vee (True \wedge AX(\tau^{13}(\emptyset)))$
$=\{s_{13}\} \cup (S \cap AX\{s_1,s_2,s_3,s_4,s_5,s_6,s_7,s_8,s_9,s_{10},s_{11},s_{12},s_{13}\})$
$=\{s_0,s_1,s_2,s_3,s_4,s_5,s_6,s_7,s_8,s_9,s_{10},s_{11},s_{12},s_{13}\}$

$\tau^{15}(\emptyset)=f_2 \vee (True \wedge AX(\tau^{14}(\emptyset)))=\{s_{13}\} \cup (S \cap AX\{S\})=\tau^{14}(\emptyset)$

显然，$\tau^{14}(\emptyset)=S$ 为最小不动点；因为 $s_0 \in S$，该计算模型满足上述定义的活性要求。

由此可以看出，TDS算法是合理有效的。

结束语 本文提出了多处理器任务调度算法 TDS 的自动机模型，并通过不动点理论对其进行了形式化验证，讨论了算法的合理有效性。应用自动机建模，并将所建模型转换成 Kripke 结构，然后在此基础上应用模型检测方法进行验证，本文结合实例进行了系统的说明。今后的研究工作可以进一步探讨目前的扩展 Büchi 自动机模型是否可以应用到其他的系统或者算法建模中，或者探索一下对于目前的模型是否有更简单更好的检测方法。

参 考 文 献

[1] Darbha S, Agrawal D P. Optimal scheduling algorithm for distributed-memory machi-nes[J]. IEEE Transactions on Parallel and Distributed System, 1998, 9(1): 87-95

)[2] Huth M, Ryan M. 面向计算机科学的数理逻辑：系统建模与推理(第二版)[M]. 何伟，樊磊，译. 北京：机械工业出版社，2007：107-112

[3] Baier C, Katoen J-P. Principles of Model Checking [M]. Cambridge: The MIT Press, 2007: 174-188

[4] Clarke E M, Grumberg O, Long D E. Model Checking[M]. Cambridge: The MIT Press, 1999: 61-65

[5] 鱼先锋，雷丽晖，李永明. 单道批处理系统的建模与验证[J]. 计算机科学，2011，38：257-259

[6] 边计年. 数字系统设计自动化(第二版)[M]. 北京：清华大学出版社，2005：351

(上接第 300 页)

[5] Chandra A, Chakrabarty K. Test data compression and test resource partitioning for system-on-chip using frequency-directed run-length(FDR) codes[J]. IEEE Trans. Computers, 2003, 52(8): 1076-1088

[6] Kavousianos X, Kalligeros E, Nikolos D. Multilevel-Huffman test-data compression for IP cores with multiple scan chains[J]. IEEE Trans. Very Large Scale Inter. (VLSI) Syst, 2008, 16(7): 926-931

[7] Gonciari P T, Hashimi B M. Variable-length input Huffman coding for system-on-a-chip test[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2003, 22(6): 783-796

[8] 詹文法，梁华国，时峰，等. 混合定变长码的测试数据压缩方案[J]. 计算机学报，2008，31(10)：1826-1834

[9] Rajski J, Tyszer J, Kassab M, et al. Embedded deterministic test [J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2004, 23(5): 776-792

[10] Dai Gui, You Zhi-qiang, Kuang Ji-shun, et al. DCScan: A Power-Aware Scan Testing Architecture[C]// IEEE Proc. ATS. 2008: 343-348

[11] Donglikar S, Manga M, Chandrasekar M, et al. Fast Circuit Topology Based Method to Configure the Scan Chains in Illinois Scan Architecture[C]// IEEE Proc. ITC. 2009: 18-20

[12] Arai M, Fukumoto S, Iwasaki K. Test Data Compression of 100x for Scan-based BIST[C]// IEEE Proc. ITC. 2006: 23-25

[13] Hosokawa T, Chen Y, Wan L, et al. A Test Pattern Matching Method on BAST Architecture Using Don't Care Identification for Random Pattern Resistant Faults[C]// Proc. IEEE International Symposium on Communications and Information Technologies. 2010: 738-743

[14] Lee H K, Ha D S. HOPE: AN Efficient Parallel Fault Simulator for Synchronous Sequential Circuits[J]. IEEE Transaction on Computer-Aided Design of Interated Circuits and Systems, 1996, 15(9): 1048-1058

[15] Fu Y H, Wang S J. Test Data Compression with Partial LFSR-Reseeding[C]// Proc of Test Symposium. 14th ATS. 2005: 343-347

[16] Wang S M, Wei Wen-long, WANG Zhang-lei. A Low Overhead High Test Compression Technique Using Pattern Clustering With N-Detection Test Support[J]. IEEE Transactions on Very Large Scale Interation Systems, 2010, 18(12): 1672-1685

[17] Maleh E L, A H. Test data compression for system on a chip using extended frequency directed run length code[J]. Computers & Digital Techniques, 2008, 2(3): 155-163

[18] Tseng W D, Lee L J. Test Data Compression Using Multi-dimensional Pattern Run-length Codes[J]. Journal of Electronic Testing, 2010, 26(3): 393-400