基于TRANSPUTER多机系统的研究与设计*

Research and Design of A Transputer-Based Multiprocessor System

王开慧 程退安 郭福顺 廖明宏
Wang Kaihui, Cheng Tuian, Guo Fushun *and* Liao Minghong
(哈尔滨工业大学计算机系)
(Dept. of Computer Science, Harbin Institute of Technology)

摘 要 并行知识库机是利用多机系统的并行性提高知识库系统推理效率的专用机。我们研制的并行知识库机由前后端机组成。其中，前端机是一台486微机，而后端机是基于TRANSPUTER的多机系统。本文重点研究后端机的拓扑结构、与PC机的接口和存储器的设计等问题。

ABSTRACT The parallel knowledge-base machine is a special computer which utilizes the parallelism of a multiprocessor system to enhance the deduction efficiency of the knowledge-base system. The parallel knowledge-base machine we designed consists of a front-end machine and a back-end one. The front-end machine is a PC-486, and the back-end is a transputer-based multiprocessor system. This paper puts emphasis on studying the topological structure of the back-end machine, the interface with PC and the design of the memory, etc.

关键词 TRANSPUTER，拓扑，接口，存储器，多机系统。

KEY WORDS transputer, topology, interface, memory, multiprocessor system.

一、前 言

由英国INMOS公司推出的单片机TRANSPUTER具有独特的优点，它采用了RISC

* 国家自然科学基金资助项目（批准号：69475014）。
收稿日期：1996年1月30日。
作者简介：王开慧，硕士，研究方向为多机系统；程退安，教授，研究方向为计算机体系结构；郭福顺，教授，研究方向为人工智能，并行处理；廖明宏，博士/副教授，研究方向为人工智能，并行处理。
通讯地址：150001 哈尔滨工业大学320信箱

技术，用通信链来高效地实现多处理机之间的互联，从面方便地组成多机系统，正是由于它的这一特点，我们在并行知识库机的研究过程中采用 TRANSPUTER 组成一个多机系统作为后端机。与前端机——PC486微机协调地工作，从而实现产生式系统的并行推理。

本文论述并行知识库机的拓扑结构，前后端机接口设计，以及后端机存储器的设计。

二、拓扑结构的设计

TRANSPUTER 具有四对通信链(LINK)可以进行同步、无缓冲、点到点通信，信息传输率可达20Mbps，通过它的四条 LINK 线可以很方便地实现互联，组成松散耦合的多机系统。

用于高度并行处理系统中的互连拓扑结构种类很多，如环形(Loop)，树形(Tree)和超立方体(Hypercube)等，为了提高多 TRANSPUTER 系统的性能，选择合理的互连拓扑结构十分重要。

我们设计多 TRANSPUTER 系统的拓扑结构遵循两个原则：

第一，较高的互连网络性能；第二，系统扩充时拓扑结构的灵活性。

点对点互联网络的通讯性能可以通过测试从一个点到另一个点的平均传输时间来评价，而平均传输时间又与互连网络中任意两节点间最短路径中步数的最大值有关，故一种好的拓扑结构，应使每两个节点间的最短路径尽量短。

在后端机的结构设计中，我们将接口电路和 Transputer 多机系统做成一个插件板，这样，PC 机将其作为一个外设，通过接口电路和管理程序，与后端机协调工作。而系统结构的灵活性体现在：当多个这种插件板互联，组成规模更大的多机系统时，仍能构成具有高性能的拓扑结构的互连网络。

正出于上述设计原则，以及插件板的面积的限制，我们设计出如图1的单板拓扑结构形式。

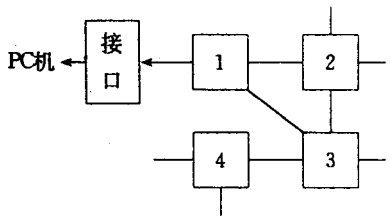

图1　单板拓扑结构

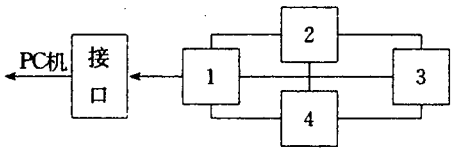

图2　全互连结构

我们采用四片 TRANSPUTER 构成单板多机系统。图中，节点1，2，3，4构成环形，而且，节点1与节点3通过一条 LINK 互联，节点1和 PC 机有一条通道实现多机系统与 PC 机之间的通信，此外，将节点2，3，4剩余的 LINK 线引出，为构成更大规模的多机系统提供方便。

当此四个 TRANSPUTER 多机系统独立工作时，可将节点2，4连接起来，成为图2

这种全互连的结构形式，这样使得每两个节点之间的最短路径长度为1，从而具有最高通讯性能。

当如图1的两块相同的板组成具有八个TRANSPUTER的多机结构系统时，可以构成如图3的拓扑结构形式，这样，除两个主节点与其它节点间的最短路径长度为3外，其余每两个节点间最短路径长度的最大值为2，不仅如此，由于这个多机系统与PC机具有两个通道，所以更进一步提高了系统性能。

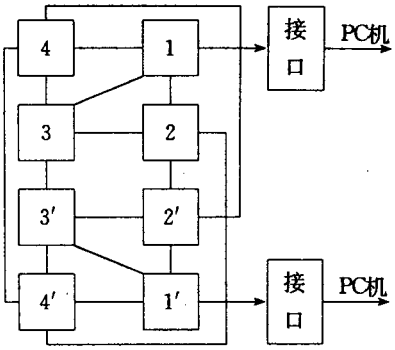

图3　拓扑结构

当三块如图1的板相连组成具有十二个TRANSPUTER的多机系统时可以构成如图4的拓扑结构形式，这样，每两个点之间路径长度最大值为3，而且，此多机系统与PC机之间的通道数目增加至3个，更进一步保证了多机系统与PC机之间的协调工作。

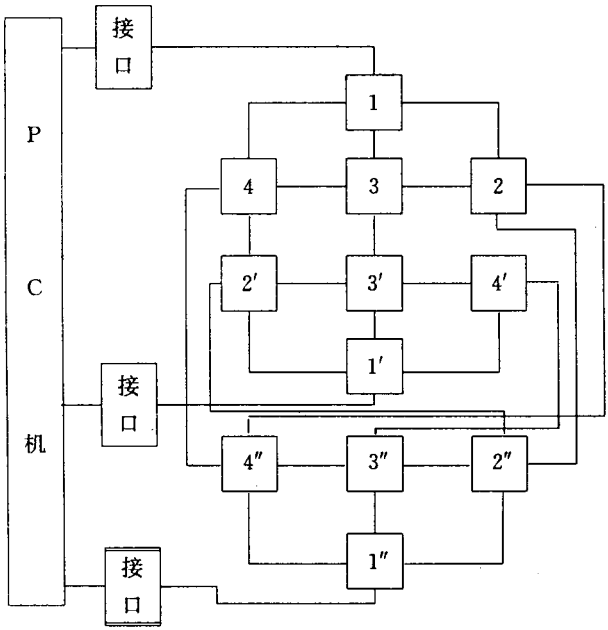

图4　十二个 Transputer 拓扑结构

由于四个TRANSPUTER多机系统的这种灵活的互连方式，使得当多块相同板组成更大规模的多机系统时，都能构成一种比较合理、高效的互连网络拓扑结构。

三、前后端机接口设计

1. 硬件接口设计

在设计前后端机接口时，选用INMOS公司的C012芯片实现PC机并行数据和

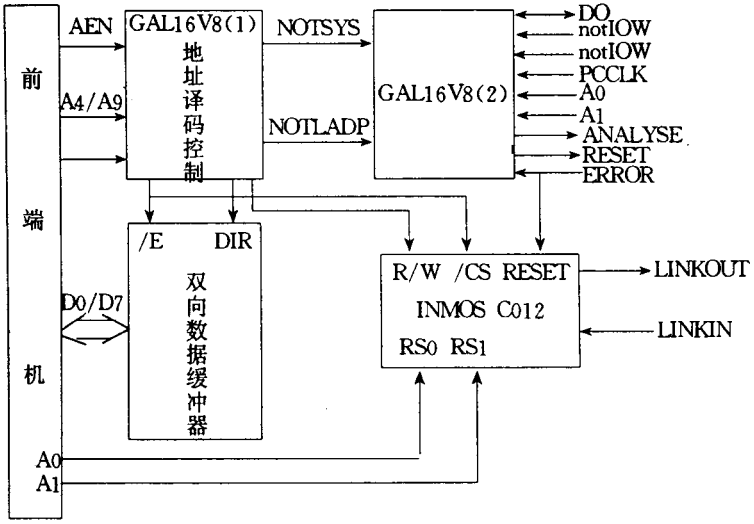

图5　前后端机接口电路

TRANSPUTER 串行数据之间的转换，并用 GAL16V8来实现前后端机信号的逻辑关系。

前后端机接口电路如图5，其主要功能是在前后端机之间建立起一个数据传输的桥梁。

用 GAL 芯片实现地址译码和产生后端机所需的各种控制信号，notSYS 有效时，允许 PC 机对 TRANSPUTER 进行复置、分析，并可查询 TRANSPUTER 的错误，notLADP 有效时，允许 PC 机与 TRANSPUTER 进行数据交换。各信号间的逻辑关系如下：

notSYS=(SA9 & SA8 & SA7′ & SA6′ & SA5′ & SA4 & AEN′)′　　:地址310H

notLADP=(SA9 & SA8 & SA7′ & SA6′ & SA5′ & SA4′ & AEN′)′　:地址300H

STATWR=notIOW……d　　由 PCCLK 锁存

STATR=notIOR……d　　由 PCCLK 锁存

notWR=((notLADP′ & notIOR′)#(notLADP′ & STATWR′))′

notCS=((notLADP′ & notIOW′ & STATWR′)#(notLADP′ & notIOR′))′

IOR245=(START′ & notIOR′)′

EN=notIOR#notSYS#SA0#SA1

D0=EN′??ERROR′

CLK=notSYS′ & SA1′ & notIOW′

RESET=CLK//(SA1′ & D0)

ANALYSE=CLK//(SA0 & D0)

2. 软件接口设计

由于前后端机接口电路中采用了 GAL 芯片，故可以方便地修改后端机的 I/O 端口地址，目前，后端机占 I/O 端口地址为300H—311H。

软件接口程序所处的环境如图6所示，其中，to. filer 和 from. filer 是软件接口程序与后端机程序的通道，软件接口程序在前端机上运行，目前用 C 语言实现，功能如下：

①对后端机的 TRANSPUTER 进行复置、分析、查错；

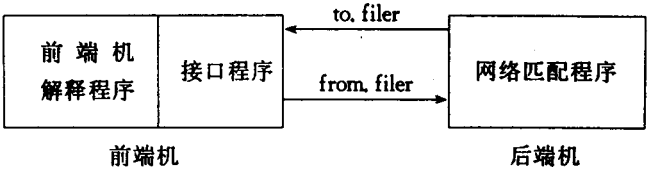

图6 软件接口示意图

②将前端机的程序执行代码加载到 TRANSPUTER 上；

③进行前后端机之间数据格式的转换。

通过软件接口程序，后端机 TRANSPUTER 系统首先进行自举，然后将系统建立好的匹配网络和匹配程序等从某一通道再经与相联的 TRANSPUTER，装载到各 TRANSPUTER，TRANSPUTER 自举过程如图7所示：

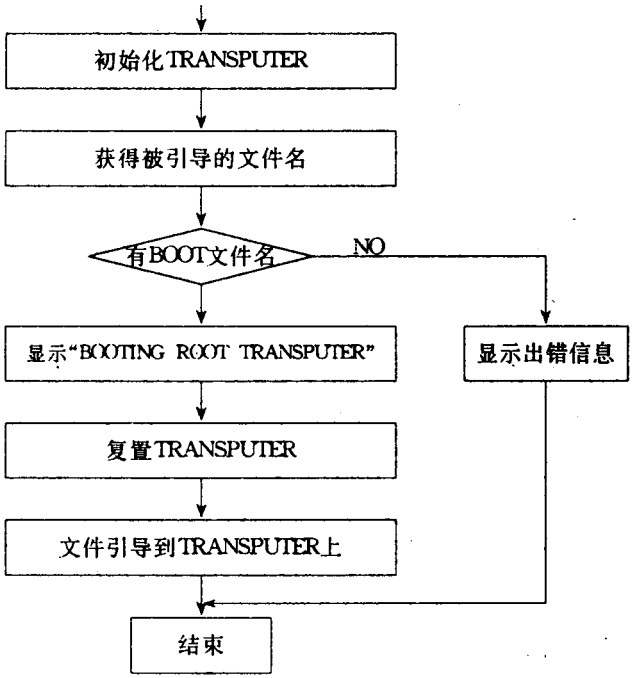

图7 TRANSPUTER 自举过程

四、存储器的设计

后端机每个 TRANSPUTER 节点都带有一定容量的局存，TRANSPUTER 具有32位4GB 线性地址空间，它本身有一个外部存储器接口，可以支持静态存储器(SRAM)。动态存储器(DRAM)和只读存储器(ROM)。

如果选用静态存储器(SRAM)芯片作为 TRANSPUTER 的存储器，则存储器设计十分简单，几乎不需要外加接口电路即可构成，但由于每片静态存储器容量有限，故要构成大容量存储器需要芯片数量多，从而占去大量的 PCB 面积，而另一方面，成本也很高，扩展时更困难。

我们在设计中，采用动态存储器芯片，而且是目前市售PC机通用的内存条，极大地提高了集成度，而且扩展极为简单。由于采用了动态存储器芯片，所以，在存储器设计中必须考虑动态存储器刷新问题，这使得存储器设计变得更为复杂。

TRANSPUTER外部存储器接口为满足不同器件的周期，提供了内部配置和外部配置两种方式，以定义存储器接口的时序。内部配置已定义好十三种配置，它的选择只需将定义的某一个地址线同MemConfig相连即可实现。

当内部配置的时序不能满足要求时，可以选用外部配置来定义所需时序，选择方法是当TRANSPUTER输出#7FFFFF6C到#7FFFFFF8地址时，由MemConfig读1或0，其实现办法，可以用ROM存入已配置好的信息，当系统启动时自动读取，或用其他逻辑电路实现，如用GAL芯片。

由于外部配置需附加电路，考虑到插件板面积的限制，故设计时首先考虑内部配置的要求和存储器性能需要，从而使内部配置时序能满足要求。

我们在存储器的设计中选用存储器周期为120ns的动态存储器，选用内部配置，将MemAD6与MemConfig相连，notMemS0—4功能分配如下：

notMemS0	作地址锁存ALE
notMemS1	作地地址输入RAS
notMemS2	不用
notMemS3	用作行列地址切换
notMemS4	用作列地址输入CAS

存储器接口时见图8：

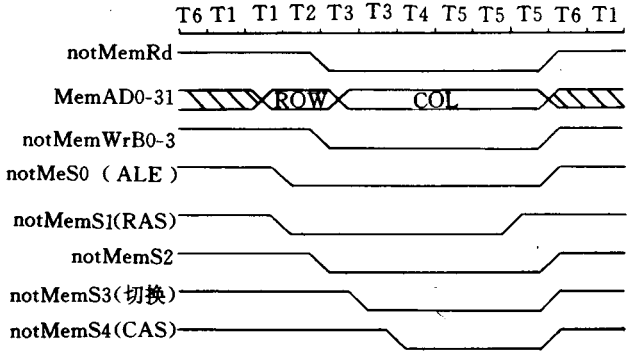

图8 存储器工作时序图（MemConfig=AD6）

存储器设计的另一个问题就是存储器容量的扩展，为使系统具有更广泛应用和灵活性，我们设计的存储器可以方便地实现存储器容量的扩充。

动态存储器内存条将地址信号分作两组，一组作为行地址(ROW)，一组作为列地址(COL)，由于TRANSPUTER提供AD2—AD11作为刷新地址信号，故用AD2—AD11作为行地址(ROW)，假设局存容量为1M字节(由四个256KB内存条组成)，则行地址为AD2—AD10，列地址为AD11—AD19，如果容量为4M字节（由四个1M字节内存条组成)，则行地址为AD2—AD11，列地址为AD12—AD21。采用图9实现冲突地址信号线的

（下转第55页）

上述所有这些弱公平性形式与我们定义1形式在逻辑上是等价的。

下面我们考虑一个简单例子。

程序1

local x, y: integer where x=0, y=0

P1:: [l_0: while true do ‖ P2:: [m_0: while true do

l_1: x: =x+1: l'_0] m_1: y: =y+1: m'_0]

上述程序中有四个转换（除单位转换τ_1），我们用语句标号来表示对应的转换。

l_0：从l_0转到l_1；m_0：从m_0转到m_1；l_1：从l_1转到l_0，同时x加1；m_1：从m_1转到m_0，同时y加1。

考虑下述非终止执行序列σ：

$$<\{l_0, m_0\}, 0, 0> \xrightarrow{l_0} <\{l_1, m_0\}, 0, 0> \xrightarrow{l_1} <\{l_0, m_0\}, 1, 0> \xrightarrow{l_0}$$

$$<\{l_1, m_0\}, 1, 0> \xrightarrow{l_1} \cdots\cdots$$

此执行序列是通过仅仅执行进程$P1$中的语句，而完全忽略$P2$执行所得。如果没有对进程$P1$和$P2$中语句的弱公平性假设，则上述执行序列是程序1中的计算。显然，上述计算并不能由实际执行得到。对转换上的弱公平性假设就排除了这种情况。上述执行序列显然是不满足弱公平性的。

4.2 进程公平性

上一节中的弱公平性假设仅仅当进程中的某一特殊转换一直能行，才保证进程最终被执行。它允许一计算，其中在任何状态，一个进程中总有某一转换能行（也即此进程能行），但此进程始终不能被执行。因此，当一个进程一直能行时，弱公平性并不能保证它最终被执行。考虑下述例子：

程序2

local x, y: boolean where x=F, y=T

P1:: [l_0: when x do y: =F ‖ P2:: [m_0: while y do

or m_1: x: =~x]

l_1: when~x do y: =F]

对执行序列σ：

$$<\{l_0, m_0\}, F, T> \xrightarrow{m_0} <\{l_0, m_1\}, F, T> \xrightarrow{m_1} <\{l_0, m_0\}, T, T> \xrightarrow{m_2}$$

$$<\{l_0, m_1\}, T, T> \xrightarrow{m_1} <\{l_0, m_0\}, F, T> \xrightarrow{m_0} \cdots\cdots$$

此序列是仅仅由进程$P2$中语句m_0和m_1交替执行而得。由于语句m_1交替将x从F变到T，从T变到F，进程$P1$中语句l_1及l_2都有无穷多个状态不能行，因此上述行序列是满足弱公平性的，它是可接受的计算（相对于弱公平性）。虽然在每一状态，进程总有转换能行，但由于其转换并不保持一直能行，因此进程$P1$总不能被执行。

下面我们给出进程公平性概念。

定义2. 我们称一进程P满足进程公平性，如果不会发生进程P在某一位置$k \geqslant 0$以后一直能行，而进程在此位置以后并没有转换被执行的情形。

由定义2知道，上述我们给出的程序2的执行序列σ是不满足进程公平性的，因为在每一状态，或者l_1能行或者l_2能行，因此进程$P1$是一直能行的。然而σ中并无进程$P1$的转换被执行，因此它并不是允许的计算。

进程公平性是一个比弱公平性更强的公平性假设。它已经在一些文献中被作为一个可行的假设。

4.3 强公平性

尽管弱公平性在大多数情形能够足以确保进程的独立执行，然而它并不足以对带有同步和通信语句进程的期望行为进行描述。这样的进程需要与其它进程紧密的协调。本节，我们给出更强的一类公平性概念，称为强公平性。它用于与语言中的通信协调语句相联。

我们首先考虑一个用信号量实现的经典互斥问题的例子（程序3）。我们用抽象语句NC和CR分别表示实际的非临界区语句及临界区语句。它们满足假设：(1) CR语句执行总会终止；(2) NC语句执行不必终止；(3) CR和NC语句执行中不会修改同步变量的值。

程序3

local semaph: integer where semaph=1

P1::	l_0: while true do	‖	P2::	m_0: while true do
	l_1: NC			m_1: NC
	l_2: request (semaph)			m_2: request (semaph)
	l_3: CR			m_3: CR
	l_4: release (semaph)			m_4: release (semaph)

考虑下述周期执行序列σ：

$$<\{l_0, m_0\}, 1> \longrightarrow \cdots\cdots \longrightarrow <\{l_2, m_2\}, 1> \xrightarrow{l_2} <\{l_3, m_2\}, 0> \xrightarrow{l_3}$$

$$<\{l_4, m_2\}, 0> \xrightarrow{l_4} <\{l_0, m_2\}, 1> \xrightarrow{l_0} <\{l_1, m_2\}, 1> \xrightarrow{l_2} <\{l_3, m_2\}, 0> \longrightarrow \cdots\cdots$$

此执行序列显然表示了进程$P2$永远在语句m_2处等待进入临界区，但它永远没有被允许进入。虽然转换m_2永远没有被执行，但它无穷多次不能行。因此上述σ对$P2$是满足弱公平性的。不难看出，σ对$P2$也是满足进程公平性的。然而，上述执行序列并不满足互斥问题的要求。互斥问题要求进程$P1$和$P2$最终都被允许进入其临界区，因此上述σ是不可接受的。

事实上，对信号量的一个合理的实现应该确保对请求语句的公平性。这类公平性显然不能由弱公平性及进程公平性概念来刻划。更强的一类公平性必须给出。对某些转换，特别对于诸如上述信号量语句这样的同步结构，我们必须不仅要求当它们在某一位置以后一直能行的情况下被执行，还必须要求在它们无穷多次能行的情况下也能够被执行。这就是我们下述的强公平性概念。

定义3. 一个计算$\sigma: s_0, s_1, \cdots\cdots$对于转换$\tau \in T$是满足强公平性的，如果不会发生

在某一位置 $k \geq 0$ 以后 τ 无穷多次能行，但它一直没有被执行的情形。一个计算 σ 对于一个转换集合 $T' \subseteq T$ 是满足强公平性的，如果对每一转换 $\tau \in T'$，它是满足强公平性的。

在现有文献中，有许多其它不同的有关强公平性定义形式，如：

·计算 σ 对转换 τ 是满足强公平性的，如果不会发生 τ 无穷多次能行，但仅被执行有限多次。

·计算 σ 对转换 τ 是满足强公平性的，如果或者 τ 只有有限多次能行，或者 τ 被执行无穷多次。

·计算 σ 对转换 τ 是满足强公平性的，如果 τ 无穷多次能行，由 τ 被执行无穷多次。

上述所有这些有关强公平性的形式与我们定义 3 形式在逻辑上是等价的。

五、公平性讨论

本节，我们对上节中所提出的不同公平性类进行具体讨论，特别对它们的实际实现进行讨论。

考虑一个调度程度，其作用是在系统执行的每一步判定所有当前能行的转换中哪一个在下一步应该被执行。一般，在多道程序设计系统中，这样一个任务是由操作系统中的调度程序来完成的。

首先我们考虑用于构造一个弱公平性计算的调度程序问题。此时，最关键的是我们应该采取什么措施保证调度程序达到此目的。任何实际的方案都不可能永远等待去判定一个转换的所有位置都是能行的。实际实现中都是仅仅基于计算的一个有穷前缀，即如果一个转换在足够长的时间中保持能行，则调度程序必须让此转换执行。实际的方案仅仅在它们对“足够长”的解释上会有不同。一般调度算法并不保证产生给定程序的所有可能的弱公平性计算，它仅仅保证产生的是弱公平性计算。

为了进一步讨论弱公平性，我们再考虑程序 1。显然对程序 1，下述不同调度程序所产生的执行序列：

σ_1：进程 $P1$ 执行一步，随后进程 $P2$ 执行一步，反复下去。

σ_{10}：进程 $P1$ 执行十步，随后进程 $P2$ 执行一步，反复下去。

σ_{100}：进程 $P1$ 执行一百步，随后进程 $P2$ 执行一步，反复下去。

它们都是弱公平性计算。由此可知弱公平性并未对公正性进行具体量化，它是一个非常弱的限制。它并未对不同处理器的执行速率进行假设。Dijkstra 指出“不应该对 N 个处理器的相对执行速度作任何假设”，“我们要求每当一个进程被激活，它迟早会得到机会完成其语句执行”。显然，上述弱公平性本质上是 Dijkstra 对交替模型所用的两个前提假设的必然结论。上述执行序列 σ_{10} 及 σ_{100} 说明进行 $P1$ 的处理器比进程 $P2$ 的处理器执行速率要快。

从程序验证的角度，使弱公平性假设尽可能的弱是有其好处的。因为如果一个程序被证明在一个一般的模型上是正确的，则它必然在一个实际的实现中是正确的。因此较弱的限制及较一般的模型就会增加所得验证结果的稳定性。这也是许多分析讨论中尽可能采用弱公平性假设的原因。事实上，由前述讨论可知，任何一个弱公平性的实际实现

都具有比我们定义的弱公平性更强的性质。弱公平性本身的目的并不在于在实际中完全被实现及模拟。弱公平性作为一种有用的抽象，它抽象了许多实际实现。

下面我们考虑进程公平性。由上述进程 2 可知，这一公平性的引入主要是由于使用了 when 语句类型，而 when 语句本身在许多机器指令中并不是一个原子指令。它的实现一般必须通过转换为与其等价的语句来完成，而这种转换最终可以使我们通过考虑弱公平性或强公平性得到解决。因此许多文献中并不考虑进程公平性[5]，它们只考虑弱公平性和强公平性。由我们的转换系统模型及所定义的语言，我们此处不考虑进程公平性。

最后我们考虑强公平性。我们只需要将强公平性调度程序的实现要求与弱公平性调度程序的实现要求进行比较即可。不难看出，与弱公平性调度程度类似，强公平性调度算法并不会产生所有强公平性计算，它只能保证算法产生的计算是强公平性计算。强公平性也可看成是对许多实际实现的一种抽象。由强公平性这种一般性质所得到的结论对其它具体的实际性质都是可适用的。然而，强公平性是一种较弱公平性更强的一类性质。关于强公平性调度算法的实现上有一点需要注意的是，为了确保强公平性调度，我们必须要对转换的能行性进行更加细致的观察。因此用随机选择来模拟弱公平性的方案对于强公平性调度算法不再适用，这是我们实际应用强公平性假设需要特别注意的一点。

六、公平转换系统模型

本节我们给出修改的转换系统模型：公平转换系统模型（FTS）。

一个 FTS 模型 S 为一个六元组$<V, \Sigma, \Theta, T, WF, SF>$，其中，$\cdot V, \Sigma, \Theta, T$ 同前述转换系统模型相应单元。$\cdot WF \subseteq T$：弱公平性转换集。$\cdot SF \subseteq T$：强公平性转换集。

我们称 Σ 中的无穷状态序列 $\sigma: s_0, s_1, \cdots\cdots$ 是公平转换系统 S 的一个计算（Computation），如果 σ 除了满足初始化条件、连续性及勤奋性外，它还满足

$\cdot$ 弱公平性：对每一转换 $\tau \in WF$，不会发生 τ 在 σ 中某一位置以后一直能行，但只有有限多次被执行的情形。

$\cdot$ 强公平性：对每一转换 $\tau \in SF$，不会发生 τ 在 σ 中无穷多次能行，但只有有限多次被执行的情形。

上述并发系统模型是一个最基本的计算模型。它为时序逻辑在并发程序规范及验证中的应用提供了一个基础。由本文公平性分析我们有结论：只要我们对原子转换选择的合适，它满足我们的限制临界引用条件，则 FTS 模型完全可用于模型化并发系统，它是可信的。

七、结 束 语

本文基于转换系统模型，对不同机制的程序设计语言讨论了公平性问题。由本文的分析讨论我们知道弱公平性是对程序中进程的一个极小约束。它仅仅告诉调度程序不能永远忽略一个在某计算位置以后一直能行的转换的执行。而强公平性是对程序中进程的

（下转第 6 页）

		标量算法	标量分类算法		SORTLIB 库分类算法	
			CPU	加速比	CPU	加速比
A	混合网格压力计算	157.28	49.94	3.15	64.54	2.44
	压力计算(总和)	456.55	309.05	1.48	92.13	4.96
B	混合网格压力计算	204.45	34.43	5.94	48.03	4.26
	压力计算(总和)	740.57	447.20	1.66	94.94	7.80

从上面两类计算实例的效能测试与分析可以看出，调用 SORTLIB 库子程序的分类算法，使用方便，计算速度快，应用范围广，是在 YH－2 机上实现分叉计算问题向量化的最优算法。

参 考 资 料

[1]黄清南等，一类分叉函数的向量化计算，数值计算与计算机应用，1993.12。

[2]YH－2 机 FORTRAN 语言用户参考手册，国防科技大学计算机研究所。

[3]YH－2 向量压缩－还原多功能程序库使用说明，国防科技大学计算机研究所。

[4]YH－2 分叉问题中分类计算子程序库使用说明，北京应用物理与计算数学研究所。

(上接第 65 页)一个较大级别的约束。它保证程序整体的执行。对于不同的程度设计语言机制，我们需要不同的公平性假设。在程序的描述及验证中，应该尽可能使用较弱的公平性假设，以得到更一般的结论。

参 考 文 献

[1] Dijkstra E W.，Solution of A Problem in Concurrent Programming Control，Comm. of ACM，1965，8(9)：569

[2] Manna Z，Pnueli A. The Temporal Logic of Reactive and Concurrent Systems：Specification，New York：Spinger-Verlag，1991.

[3] 贾国平，郑国梁，论公平转换系统模型的可信性，计算机科学，1996 年，第 23 卷，第 2 期。

[4] Queille J P，Sifakis J.，Fairness and Related Properties in Transition Systems－a Temporal Logic to Deal with Fairness，Acta Infonmatica，1983(19)：195－220.

[5] Francez N.，Fairness，New York：Springer-Verlag，1986.

[6] Park D.，On the Semantics of Fair Parallelism，In：Abstract Software Specification，Lec. Notes in Comp，Sci，86，Berlin：Springer-Verlag，1980：504－524.

[7] Apt K R，Francez N，Katz S.，Appraising Fairness in Languages for Distributed Programming，Distributed Computing，1988(2)：226－241.