国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第11卷第4期 1989年12月 Vol.11 No.4

多处理机系统中的宏任务并行

廖湘科 陈立杰

（电子计算机系）

摘 要 多处理机系统是巨型机的发展方向。宏任务是在多处理机系统上用来加快程序执行的一条主要途径。文中介绍了宏任务并行，描述了宏任务的实现，讨论了编译器和操作系统的支持。最后解释了宏任务的一个程序。

关键词 多处理，多任务，宏任务，微任务，粒度，逻辑cpu

分类号 TP31

科学技术的发展需要高性能的计算机。由于电子器件的限制，超高速单处理机的开发已接近极限，因而巨型机走向多重处理已是不可避免。

目前新一代的巨型机几乎全是多处理机系统。多处理机系统的一种常见组成方式是由少量高性能的单处理机共享内存来实现。这采取了超高速单处理机与多重处理两者兼有的方式。Cray x-MP，Cray-2，Cray y-MP都属于这种组织方式。

多处理机的引入，使得在多处理机上可以开发5个级别的并行，即实现作业级、作业步级、子程序级、循环级与指令级的全面并行。指令级的并行是靠单处理机采用的流水线技术实现的，而其余4级并行如下：

（1）作业级的并行：在多个不同的处理机上各自运行不同的作业。

（2）作业步级的并行：在多个不同的处理机上同时执行同一个作业的各个作业步。

（3）子程序级的并行：在多个不同的处理机上同时执行同一个程序中的不同子程序调用。

（4）循环级的并行：在多个不同的处理机上同时执行同一个循环的不同迭代。

由此看来，在多处理机系统中，有两种主要的并行处理方式：一种是在各个不同的处理机上运行不同的作业，以提高整个系统的吞吐率，称之为多处理并行（multiprocessing），这是作业级的并行；另一种是在各个不同的处理机上同时运行同一个作业，以缩短该作业运行的墙上时间(wall-time)，称之为多任务并行（multitasking）。作业步级、子程序级、循环级的并行都是多任务。目前的多机系统大多未提供作业步级的并行，而支持子程序级和循环级的并行。

因为子程序级与循环级的任务粒度(granularity)不同，目前的多机系统大多提供了两种工具来支持多任务：一种是宏任务技术（macrotasking），主要针对子程序级的多任

学术论文 1989年3月30日收稿

务；一种是微任务技术（microtasking），主要针对循环级的多任务。下面我们主要讨论宏任务。

1 宏任务

为了支持多任务，必须在语言一级引入并行描述手段。对于子程序级的宏任务，考虑到可移植性、兼容性、工作量大小等因素，目前主要采用在 Fortran 中引入一组支持并行的原语的方式来支持多任务。

为了使操作系统、编译器的变动尽可能小，以及提高实现效率，这些支持并行的原语不采用由操作系统直接支持的方式，而通过一个库中的一组子程序来支持。这样一种供用户在语言级调用，用来开发并行的子程序集就称为多任务库。由多任务库来管理程序中任务的状态变迁，而不通过操作系统，使操作系统的变动很小。系统调用的执行时间一般比较长，而用多任务库来管理任务的开销较之小得多。多任务库可以看作运行在用户区的监督程序(monitor)。

并行程序设计有 4 个基本问题：并行的表示、并行实体之间的同步、互斥以及通讯。为了给用户提供开发并行的手段，多任务库必须提供：并行实体的控制机制、同步机制、互斥机制、与数据通讯机制。

Cray机上的多任务库提供了下列子程序与函数。用户通过调用这些子程序来将程序组织成多任务。

这些子程序可以分为三类：

(1) 任务类子程序：

```
CALL  TSKSTART(task-array,subname[,argument])
CALL  TSKWAIT(task-array)
CALL  TSKVALUE(value)
bollean=TSKTEST(task-array)
CALL  TSKTUNE(keyword1,value1,keyword2,value2,…)
CALL  TSKLIST(dn)
```

其中：

task-array：任务控制数组，为一整型数组，用来存放任务标识、任务值等。

value ：一个整型变量，TSKVALUE将调用任务的任务值回送于value中。

bollean ：一个逻辑变量。当指定任务存在时，回送TRUE，否则回送FALSE.

keywordi ：库调节参数对应的关键字。

valuei ：要赋予的库调节参数值。

dn ：列表输出的数据集名。

(2) 锁类子程序：

```
CALL  LOCKASGN(lock)
CALL  LOCKON(lock)
CALL  LOCKOFF(lock)
CALL  LOCKREL(lock)
```

bollean＝LOCKTEST(lock)

其中：

lock　：锁变量。

bollean：逻辑变量。当指定锁状态为locked时，置TRUE，否则置FALSE.

(3) 事件类子程序

CALL　EVASGN(event)

CALL　EVWAIT(event)

CALL　EVPOST(event)

CALL　EVCLEAR(event)

CALL　EVREL(event)

bollean＝EVTEST(event)

其中：

event　：事件变量。

bollean：逻辑变量。当指定事件为posted状态时，置TRUE，否则置FAUE.

在这三类子程序中，任务类子程序提供并行实体的控制机制，事件类子程序提供同步机制，锁类子程序提供互斥机制。而数据通讯机制则通过Fortran中的 common块以及参数传递来实现。为了更好地支持任务内部的数据通讯，Fortran还引入了一种新的tas common块。

任务类子程序主要处理任务的创建与同步，以及管理与任务有关的一些信息。在这里，任务是一个具有子程序形式（即用子程序来描述）并且可以调度的计算单位。主程序也可以看作根任务。当一个程序创建属于该程序的其它任务时，就出现了多任务的工作方式。一个任务必须从某个子程序的入口处开始执行。

对于每一个用户创建的任务，用一个任务控制数组表示。任务控制数组是一个整型数组，存有任务标识等信息。

一个任务通过调用 TSKSTART 创建。调用 TSKSTART(task-array, subname[, argument])将创建一个任务控制数组为task-array的任务，该任务完成子程序 subname (argument)的功能。任务一经创建，就可以被调度运行。当任务执行到子程序中的end, stop, return语句时，任务终止。

TSKWAIT子程序可以用来完成任务之间的同步。调用 TSKWAIT (task-array) 将挂起调用该子程序的任务，直至由task-array指定的任务终止。

TSKVALUE用来取任务值；TSKTEST用来测试指定任务是否存在；TSKTUNE用来修改库调节参数；TSKLIST用来输出所有任务的信息。

锁类子程序主要用来实现代码临界区与共享数据的保护，提供一种互斥机制。锁有两种状态：locked，unlocked.

调用LOCKON(lock)子程序，若此时锁lock的状态为unlocked，则置为locked状态，返回调用程序。若此时锁 lock已是 locked 状态，则挂起调用LOCKON(lock)的任务，直至该锁lock的状态被另一个任务置为unlocked. 然后再置锁lock的状态为locked，返回调用程序。

调用LOCKOFF(lock)子程序将把锁lock的状态置为 unlocked. 若此时有任务在等待该锁，则激活一个等待的任务，使它恢复执行。

LOCKASGN是创建一个锁，LOCKREL是删除一个锁，LOCKTEST 是测试锁是否在locked状态。

事件类子程序主要用来实现任务同步。事件有两种状态：posted，cleared. 事件与锁的主要区别在于：当一个锁变为 unlocked时，只有一个等待该锁的任务恢复执行，而当一个事件变为posted时(标志某一事件发生)，所有等待该事件的任务都恢复执行。

事件类子程序主要通过 EVPOST，EVWAIT，EVCLEAR 来实现同步功能。调用 EVWAIT(event) 子程序，若指定事件已是 posted 状态，则任务继续执行，否则挂起调用该子程序的任务，直至指定事件被其它任务置为posted状态为止。EVWAIT 调用不改变事件的状态。调用EVPOST(event)将把事件的状态置为 posted，然后返回调用任务。若此时有任务正在等待该事件变为posted 状态，则所有等待该事件的任务都恢复执行。调用EVCLEAR(event)将使事件event的状态变为 cleared.

EVASGN用来创建一个事件变量；EVREL 用来删除一个事件变量；EVTEST 用来测试事件是否在posted状态。

数据通讯机制是通过参数传递，common块，task common块来进行的。common 块中的变量可以被所有子程序所共享，而task common 中的变量只能被在同一个任务中的子程序所共享。

利用多任务库的这些子程序，用户就可以创建任务并进行同步互斥。

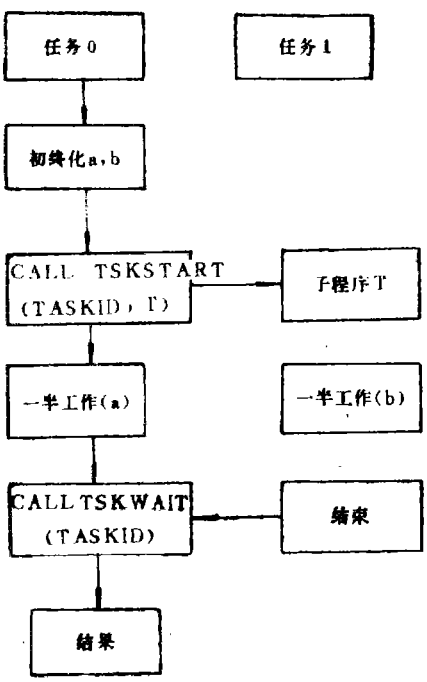

图 1 并行任务的开始和结束

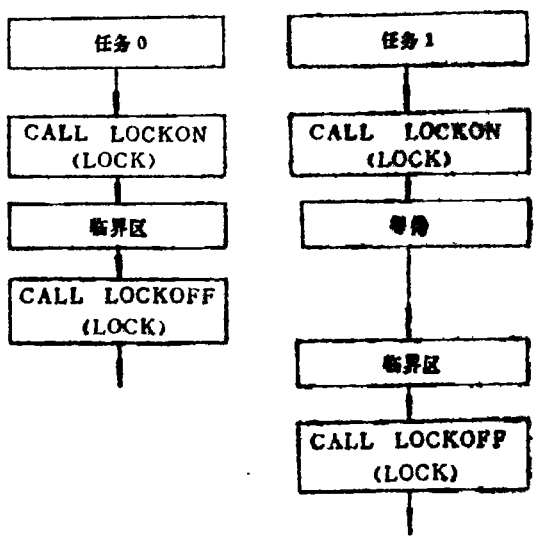

图 2 用锁实现临界区保护

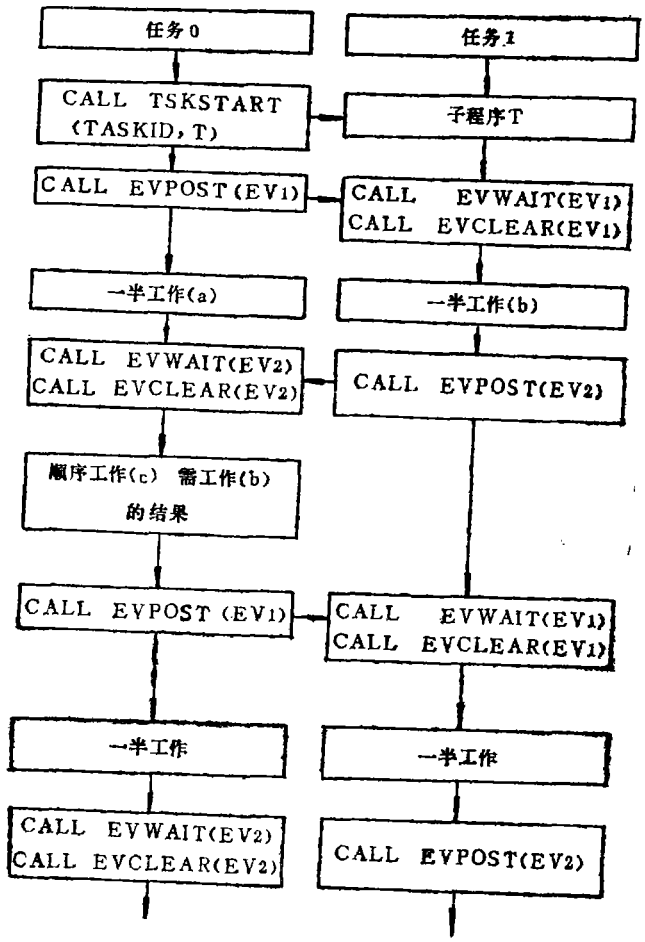

图 3　同事件进行同步

2　多任务库的实现

为了实现宏任务，编译器与操作系统必须提供一定的支持

2.1　编译器对多任务库的支持

静态存贮分配，是 Fortran 的一个重要特点。但由于多任务库的引入，静态存贮分配已不能满足要求。对于每一个子程序，它的环境主要由三部分组成：返回调用程序的地址，传送给被调用子程序的实参地址，被调用子程序的局部变量。Fortran 的静态存贮分配，是对每一个子程序有一个环境，而不是每一个子程序调用有一个环境。这个环境在装入时就分配好了，执行时不再变动。因此，在多任务环境下，当两个任务同时调用同一个子程序时，因为子程序只有一个环境而发生冲突，所以静态存贮分配不能支持多任务。

栈式存贮分配是在每次调用子程序时，分配一个环境。当子程序执行完时就释放环境。这样，它支持了多任务。

由此而知，由于宏任务的引入，Fortran 的存贮分配不宜采用静态存贮分配，而要

引入栈式存贮分配。

栈式存贮分配的采用使Fortran编译器产生很大的改动，尤其是参数传递方式变动，涉及面很广。

2.2 操作系统对多任务库的支持

操作系统对多任务库的支持主要是引入逻辑cpu的概念。操作系统通过对逻辑 cpu的支持来支持多任务库。

所谓的逻辑cpu，就是一个被操作系统调度在物理cpu上执行的实体。从操作系统角度，它对应于一个TCB(任务控制块)和一个TXT(任务执行块)。而从多任务库角度中，它是虚拟的cpu资源。一个任务只要调度在逻辑cpu上执行，就可以认为它在执行了。

这样，一个程序开始执行时，分配一个逻辑 cpu；当有并行部分时，用户程序可以通过多任务库请求新的逻辑cpu，这样就产生了多任务。

库调度器的工作，就是将用户程序中的任务，用最有效的方法调度到逻辑 cpu 上运行。一个逻辑cpu执行一个任务直至该任务等待锁,或等待事件，或等待一个任务完成，或本任务终止。然后，库调度器选择另一个就绪任务，继续在该逻辑 cpu 上运行。这种任务切换的完成不必通过操作系统。操作系统完成逻辑cpu到物理cpu的调度。

因此，由于逻辑 cpu 的引入，实际上形成了一个二级调度：一级是库调度，一级是操作系统调度。库调度针对程序中的任务，操作系统调度针对单任务作业与多任务作业中的逻辑cpu.

库调度与操作系统调度之间的关系如图 4 所示：

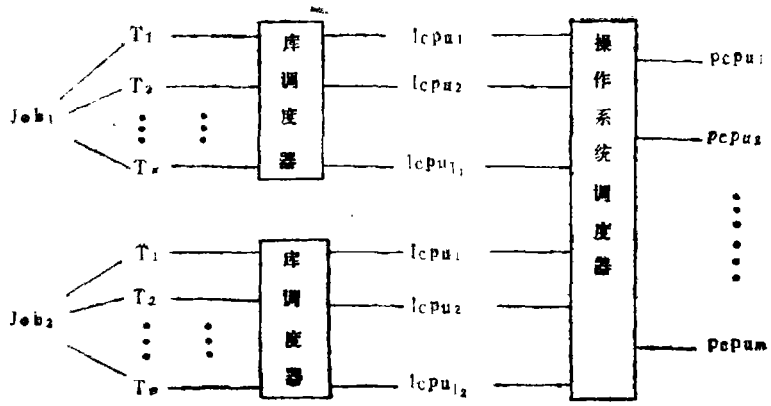

图 4 宏任务时的二级调度

lcpu：逻辑cpu pcpu：物理cpu

用户与操作系统的界面就是系统调用。为了支持多任务库，操作系统提供了两条系统调用：一条是申请逻辑cpu，一条是释放逻辑cpu. 操作系统对于多任务库的支持就是支持这两条系统调用以及完成逻辑cpu到物理cpu的调度。

2.3 多任务库的实现

在多任务库中，任务是一个具有子程序形式并且可以由库调度器调度的计算单位。

它可以处于6种状态，其状态转换图如图5，其中框里字母表示因为任务自身调用该子程序或执行该语句而导致状态转换。非框里字母表示因为其它任务调用该子程序或执行该语句而导致状态转换。

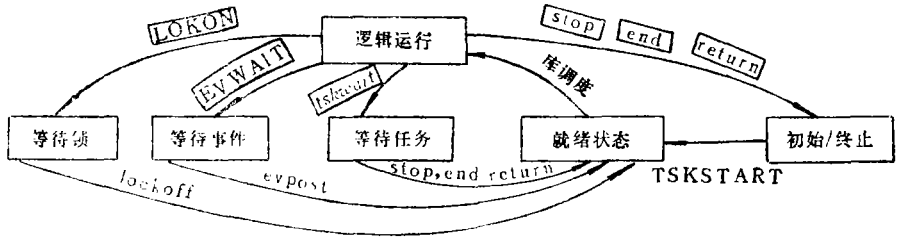

图 5　任务状态转换图

为了管理程序中任务的状态变迁，多任务库设立了几个任务队列。当任务状态改变时，由多任务库来将它们插入到合适的队列中去。

多任务库几个关键子程序的流程如下：

(1) TSKSTART：为该任务建立初始栈段，并置好该任务TIB（任务信息块）的各项值，为task common分配空间，将要传送给任务的参数存放好，完成终止程序的连接，然后将任务插入活动任务队列与就绪队列。若满足申请逻辑cpu的条件，就申请逻辑cpu，返回调用程序。

(2) TSKWAIT：检查指定任务是否在活动任务队中。若不在，表示该任务已经完成，控制返回调用程序；若在，则将调用任务插入等任务队列中，转库调度。

(3) TSKTER：它对应于任务中的stop，return，end语句。它将等待任务队中所有等待该任务完成的任务插入就绪队中，然后将该任务从任务活动队中删除，释放该任务所占全部资源，转库调度。

(4) LOCKON：检查锁状态。若是unlocked，则置为locked返回调用程序。若为locked状态，则将调用任务插入相应的等锁队列，转库调度。

(5) LOCKOFF：将锁的状态置为unlocked．若有任务在相应的等锁队列中，取等锁队的头一个任务，将其加入就绪队，并重置锁为locked；若满足申请逻辑cpu的条件，则申请逻辑cpu，返回调用程序。

(6) EVWAIT：检查事件是否为posted状态。若是，则返回调用程序，否则将任务插入相应的等事件队列，转库调度。

(7) EVPOST：置事件状态为posted．若有任务在相应的等事件队列，将它们全部插入就绪队，若满足申请逻辑cpu的条件，则申请逻辑cpu．返回调用程序。

库调度器的工作是一旦有逻辑cpu空闲，就将就绪任务调度到逻辑cpu上运行。

3　多任务库的使用

多任务是一种用来缩短程序运行墙上时间的工具。但是由于宏任务的开销比较大，当任务粒度太小以及任务紧密耦合时，引入的开销很可能抵消宏任务所带来的好处。因此，并不是所有的程序都是适宜于宏任务的。宏任务主要适合于一类特殊的程序：用

Fortran编程的程序，长时间运行或经常运行的程序，使用大部分或全部内存的程序，要求专用环境的程序。

在宏任务一个程序以前应先将程序向量化。一个程序使用宏任务，其主要步骤如下：

(1) 决定程序是否宏任务。宏任务程序应该有大的任务粒度。宏任务的部分应该是占据了程序大部分运行时间的地方。

(2) 识别可以并行处理的部分。

(3) 通过插入对多任务库子程序的调用，将识别出来的并行部分转换成任务。

(4) 仔细考虑任务的同步通信，共享数据的保护。

通过提供多任务库的方式来支持子程序级的宏任务，给用户提供了一种比较方便的并行描述手段，避免了用户从机器的硬件特性上利用多机系统的并行处理能力。宏任务的实现具有相当大的灵活性。但是由于宏任务的开销较大，不利于小粒度并行的开发，在批处理环境中运行效果也不很理想。此外，宏任务要求用户通过调用多任务库子程序来自己组织并行与同步，用户编码的负担也较大，使用不很方便。因此，宏任务的方法还有待于进一步完善与发展。

参 考 文 献

[1] Gary J B. Reentrancy and Multitasking on Cray Computer. DE85007256

[2] Mohtry G R. A Fortran Multitasking library for use on the Elxsi 6400 and Cray x-MP. DE85016223

[3] Strout R E. Converting Scietific Software to Multiprocessors: A Case study. DE86014751

[4] Hicks H R, Lynch V E. An Introduction to Programming Multiprocessor Computers. DE85010274

[5] Miya E N. Experiments with Cray Multitasking. N86-13916

[6] Larson J L. Multitasking on the Cray x-MP/2 Multiprocessor. Computer, July, 1981: 62～69

Macrotasking on Multiple-processor Computers

Liao Xiangke Chen Lijie

(Department of Computer)

Abstract

The trend of supercomuters is multiprocessors. Macrotask is a major way to speed up the execution of programs on multiprocessors. In this paper a introduction to macrotask is presented, the implementation of macrotask is described, and the supports of compiler and operating system are discussed. Finally to macrotask programs is presented.

Key Words: multiprocessor, granularity; multitask, macrotask, microtask, logical cpu