20-24
计算机科学Vol.20№.1 1993

Ada语言与Occam语言的比较*)

管惠维（上海工业大学计算机科学与工程系 200072上海）
孙永强（上海交通大学计算机科学与工程系 200030上海）

摘 要

One of the most important developments over the last decade in computing language has been the emergence of the Ada programming language and Occam programming language which both have parallel computing ability. This paper discusses and compares the two languages in parallel computing. It is highlighted that the concurrent execution, the mechanism of synchronization and communication, alternative and priority, programming embeded systems, and programming distributed systems in the two languages are analysed.

一、引 言

过去十年在计算语言领域的重要进展之一是出现了具有并行计算能力的Ada语言和Occam语言。Ada语言是具有类似Pascal语言的控制结构，并且能重植入过程和函数的大型语言，其初始设计目标是满足美国国防部所规定的要求，即首先考虑支持适于军事装备的嵌入式编程系统[1-2]。就本质而言，Ada语言是在串行语言的结构上，扩充增加了并发执行的能力。

Occam语言是和Transputer同时设计实现的，它是基于英国计算机科学家C.A.R.Hoare提出的通信顺序进程（CSP）模型所开发的一种新型并行计算语言[3-4]。Occam直接面向并行处理，其特征是通过软通道进行点对点的信息传递，软通道可以直接映射到Transputer上的通信链Linki（i=0，1,2，3），构成信息传递型的分布式多机系统。关于Ada和Occam的细节可参考有关书籍和文献[1,5-7]，本文着重讨论和比较两者在并行计算方面的异同，但值得一提的是Transputer上现在已经可以配置Ada语言了[8]。

*）本文受博士基金、上海市高教科学基金资助。

二、并发执行

在Ada和Occam中，并行执行单元内部是串行进程的组合。Ada中的进程称为任务，是一个包含若干串行语句的结构。Occam中的进程定义还要低级，每个语句都可视为进程。在Ada中一系列动作通常被表达为一串语句，而在Occam中这样的一系列动作用定义给定的顺序执行的进程集合来表示。例如，以两个过程调用proc1和proc2的执行为例，其在Occam和Ada两种语言中的串行形式如下：

```
SEQ     —Occam    begin   —Ada
  proc1             proc1;
  proc2             proc2;
                  end;
```

Occam中的并发执行获得PAR结构的直接支持，上述两个过程的并发执行只需简单地改为下列形式即可。

```
PAR
  proc1
  proc2
```

而在Ada中获得上述并发性需引入两个任务，它们可以写为：

```
declare
    task one;
    task two;
    task body one is
    begin
       proc1;
    end one;
    task body two is
    begin
       proc2;
    end two;
begin
    null;
end;
```

两个任务执行的动作由任务体定义，上述任务one执行对proc1的调用，任务two类似地调用proc2，两个任务的并发执行优先于主程序执行空语句。

Occam语言的基础是进程的概念。一个Occam程序是一个进程的分层。Occam具有五个原始进程：赋值进程、输入进程、输出进程、空操作进程和停止进程。原始进程的组合可以形成高一级的进程。在顶层，可将完整的Occam程序视为一个进程。Ada的任务虽然也可类似地分层，但不能延伸至原始进程。

在Ada中任务需命名。一个任务由两部份组成：规范说明和任务体。一个任务的例子如下：

```
task instance;
task body instance is
 —internal declarations
begin
 —sequence of statements
end instance;
```

Ada中的任务不能被参数化。一个任务可以在程序的任意层加以定义，并在对应该定义的入口后生成。一个任务的实例也可直接由"New"操作符生成。因此，这些功能为构造动态结构的程序创造了条件。而Occam基本上只支持静态结构的并行计算模型。

Ada和Occam程序的正常终止是当它们的进程都已执行完毕。除此之外，Ada任务可在下列情况之一终止：

- 一个异常出现；
- 在当前任务期待终止并且其所有子任务或者已终止或者等待终止时，一个选择语句的终止择一（terminate alternative）被执行；
- 任务夭折。

Occam没有夭折功能和终止择一功能。如果某进程出现错误，则该进程等价于一个停止进程STOP。缺乏夭折功能，使得Occam难以强制终止那些无法控制的进程和死锁进程。然而Ada中的夭折机制是其有争议的特征之一，程序中夭折的出现往往带来不可靠性。

三、同步和通信

Ada中任务可与其作用域内的任意变量通信，各任务之间的通信基于共享变量，其拷贝保持在各个任务的存贮空间中。使用共享变量的突出问题是变量冲突，Ada将避免变量冲突的责任推给了用户，这样在一定程度上增加了程序设计的复杂性。Occam是基于CSP模型的语言，它和Ada截然相反，不使用共享变量。而是借助于通道实现点对点的信息传递型通信。两者在同步和通信方式上的不同详述如下。

1．进程命名

Occam进程不需要命名，而在Ada中，任务一般都需命名。Ada使用直接的、非对称的命名方式。例如：

```
T.E (X) ; —pass the value contained
    in variable X
  to the entry E in task T
accept E (V, in<some type>) do
  —read from any calling task into
  —variable Y the value given on entry call
    Y:=V;
end E;
```

它实现将变量X的值传递给任务T中的入口E，而在接受语句中，从任何调用任务中读入变量Y，其值由入口E的调用给定。Ada任务中的入口在该任务的规范中加以定义，例如：

```
task T is
  entry E (V, in<some type>) ;
  —other entries
end T;
```

2．同步模式

Occam提供一种标准的同步会合点，即使用软通道进行信息的同步传送。同一时刻一个软通道只能用于一个方向上的通信。Ada提供一种扩展的会合点同步模式，需要的话可使数据双向传送。这种模式是远程请求，其结构如下：

```
accept E (X:in  Xtype; Y, out Ytype) do
  —use X
  —construct Y
end E;
```

它不但可进行数据的双向传送，而且可在会合点进行数据加工。

3．通信的信息结构

Ada的入口调用采用和过程调用一样的参数传递方式。任意数目的对象可作为参数传递，每个对象能有任意合法结构，包括先前已定义的和用户定义的类型、可变记录、链表以及多维数组等。Occam虽然也能允许任意对象的集合进行通信，但Ada具有比它丰富得多的数据类型。

四、择一结构和优先级

并发编程语言中的进程间通信，无论是基于信息传递的同步方式（如Occam），还是基于共享变量的远程请求方式（如Ada）都需要允许一个进程有选择地等待有可能通信的若干进程之一的机制。为此，Ada提供了select语句，而Occam提供了择一结构的ALT进程。下列两个程序段分别给出在Ada和Occam中一个进程选择等待两个整数数据之一的通信。

在Ada中：

```
select
  accept C1 (I; integer) do
    value; =I;
  end C1;
or
  accept C2 (I; integer) do
    value; =I;
  end select;
```

在Occam中：

```
INT value:
ALT
  C1? value
  SKIP
C2? value
  SKIP
```

若存在同时对两个或多个分支结构的选择调用，则两种语言都可利用哨警来确定哪个调用被接收。在Ada中的布尔表达式哨警结构为：

```
select
  when<Boolean expression>⇒
    accept E (…) do
      <action>
    end E;
```

在Occam中哨警的表示形式包括布尔表达式和通道，形式简单但一个Occam哨警仅能包含一个输入操作。

除了在若干个入口调用中选择一个入口调用之外，Ada允许在select语句中增加一条else语句，用于处置没有一个入口调用可以接受的情况。其结构为：

```
select
  accept E (…) do
    .
    .
    .
  end E;
else
  <sequence of statements>
end select;
```

在Occam中处理这类情况可使用IF进程或优先级结构。

当存在大量的信息资源可选择时，上述选择结构中的代码势必变得冗长。为此，Ada提供了一种入口族的机制，它用任务规范定义为如下形式：

```
task T is
  entry E (0…N)  (<entry_parameters>) ;
end T;
```

然后，或者用显式的方法写出每一族的入口，例如：

```
select
  accept E(0) (…) do
    …
  end E;
or
  accept E(1) (…) do
    …
  end E;
or
  ……
  accept E(N) (…) do
    …
  end E;
end select;
```

或者结合使用循环语句如下表示：

```
for i:=0 to N loop
  select
```

```
    accept E(i)(…) do
      ……
    end E;
  else
    null;
  end select;
end loop;
```

Occam提供了一种非常精简形式的重复择一结构，对于上述情况的处理可以简单地写成：

```
ALT i=0 FOR N
  ch[i]? X
  —process
```

这一操作允许将一个任意长度的数组通道和择一结构ALT相连，使用十分方便。

Ada和Occam文法不同是显然的，而且它们对于择一结构的处理方式具有重要的语言区别。Occam视所有的选择等价，在ALT结构中是并发选择，且当所有哨警都未满足时则挂起该ALT进程；而在Ada中，对每个被选成员是进行串行检查，轮流操作。显而易见，对于择一情形的处置，Occam比Ada的效率高。

光有择一结构，对于某些算法的支持并不充分，尤其是一些优化算法，往往需要有优先级机制赋予特定的选择。在Ada中，这只能通过count属性来达到，它被赋予特定选择的入口调用。以具有三个分支的选择语句为例，被赋予count属性以确定优先次序可写成如下形式：

```
select
  accept HIGH;
or
  when HIGH'count=0⇒
  accept MID;
or
  when HIGH'count=0 AND MID'count=0⇒
  accept LOW;
end select;
```

然而，Ada的这种描述形式并不能保证满足优先级算法的需求。因为Ada中select语句的实现是采用队列结构，所以往往出现这样的情况：在一个优先级高的任务被接受之前，它可能处于在队列中其哨警被计值且并不夭折，而对该高优先级的任务中的入口调用却不出现（例如该入口调用已被某定时启动的任务所清除。），好就阻塞了其它比它优先级低的任务的可选性。

Occam提供择一的优先级结构ALT PRI，直接支持择一的优先级机制，简洁自然。

五、嵌入式编程和分布式编程

Ada和Occam的并发执行任务都可扩充，使得外部设备可被定义为在其程序范围之外的进程。一个中断可视为外部进程和内部进程之间的同步。若从内部进程的立足点看，这是采用了会合点的形式。在Ada中外部进程的入口调用在中断发生时，是通过在其入口说明中增加一条特别的子句来实现的。在Occam中的中断处理是将通道映射到一个预先定义的存贮器地址。两者的中断处理都含有一个等待选择合适的入口或通道的进程，该进程通常被赋予高优先级，以满足快速中断响应。

Ada和Occam的本质区别之一是两者的通信方式不同。Ada提供低级的输入/输出程序包实现和外部设备直接通信，它是一种基于共享变量的接口形式，适合嵌入式编程。

Occam不允许机器码嵌入，但允许C语言、Pascal、Fortran77语言以及Transputer的汇编语言的嵌入。在和外部设备或其它处理机的通信中，Occam不允许使用共享变量，而是采用基于点对点通信的信息传递方式。多机间通信，直接使用通道映射到Transputer的通信链上，和外部设备的通信，使用端口实现存贮器映射形式的输入/输出。

Occam的分布式编程原理十分清晰，它提供PLACED PAR结构，直接支持将需要并行处理的任务或进程配置到多机网络上。同时它也支持虚结点的分布式处理（例如，在单机上模拟多进程的并发执行。）。一个虚结点是分布式系统中一个物理结点的抽象，并且它同时规定了分布的粒度。虚结点机制是并行处理应用程序调试中必不可少的机制。

Ada的分布式编程原理并不清晰。它虽然可进行分布式编程，但是实现起来比较复

杂和困难。在Ada中，两个主要可被考虑作为虚结点的是：任务和包。其中任务作为虚结点的能力是有限的，因为它不能像包那样处理数据和库单元。包受分离编译、库单元和异常处理功能的支持，其受限形式可被作为一个虚结点，即这个包不允许外部存取它的变量，而且仅有任务规范和类型说明是外部可视的。此外，存取变量不可说明为入口参数。这样，我们能限制包规范仅仅含有过程描述，去实现一个在调用接口与虚结点之间的远程调用过程。在这种情况下，就可在各虚结点间获得任务的同步和并发执行，并且利用远程进程调用机制实现结点间的通信。

综上所述，Ada的嵌入式编程能力强，而Occam的分布式编程直接明了，使用方便。

参考文献

〔1〕 N.H.Cohen, Ada as A Second Language, McGraw-Hill Book Company, New York, 1986

〔2〕 G.Booch, Software Engineering with Ada, The Benjamin Gummings Publishing Company, 1983

〔3〕 C.A.R. Hoare, Communicating Sequential Processes, Prentice Hall International, 1985

〔4〕 INMOS Limited, A Tutorial Introduction to OCCAM Programming, Prentice Hall, Bristol, 1987

〔5〕 管惠维，孙永强，Transputer应用研究的现状，计算机工程，已录用将发表

〔6〕 管惠维，孙永强，基于Transputer的外围控制实现方式，小型微型计算机系统，13：(4)(1992)

〔7〕 管惠维，孙永强，基于Transputer的并行程序设计技术，小型微型计算机系统，已录用将发表

〔8〕 INMOS Limited, The Transputer Development and iq System Databook, Redwood Press Ltd, Melksham, 1991

（上接32页）

对于活性问题的分析也存在类似的问题。如图6中(a)和(b)二个Petri网，前一个可能存在死锁，另一个则不会。但是它们的可达树是完全相同的，如图7所示。

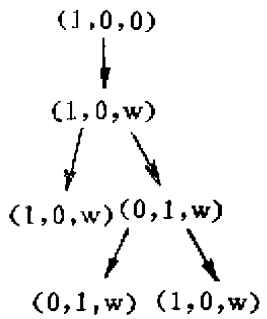

图7 图6(a)和(b)的可达树

虽然可达树分析法不包含足够的信息来解决可达性、活性问题，但是它也有足够的信息来解决很多这样的问题。如标记μ′在可达树中可能出现，则μ′是可达的，否则是不可达的。同样，可达树中有终止结点，则该Petri网不是活的。

五、结束语

从以上本文的讨论表明，Petri网表达对象具有抽象性、准确、精练而无二义性，并且能运用可达树分析法来获得对象多方面的分析性质。目前，Petri网的理论仍处在发展阶段，随着人们对它的研究逐步深入，必将进一步推动它的理论和应用不断向前发展。

参考文献

〔1〕 J.L.Peterson, "Petri Nets", ACM Computing Surveys, Vol.9, No.3 pp223-252, Sept. 1977.

〔2〕 Tadao Murata, "Petri Nets: Properties, Analysis and Application", Vol.77 No.4, pp541-580, Apr. 1989

〔3〕 胡家宝 形式描述技术及其应用 武汉造船 1991,5