程序设计 并行计算模型 GAMMA

计算机科学 1994 Vol. 21 №. 5

20-24

并行计算模型 GAMMA 概述

黄林鹏 童维勤 倪德明 孙永强

(上海交通大学计算机系 上海 200030)

TP311

摘 要 程序设计中的并行性可以区分为逻辑并行性和物理并行性，前者是一种程序构造工具，它通过任务复合来描述程序；后者涉及实现技术，它关心的是任务如何在处理器集合上的分配。本文将介绍一种新的并行计算模型 GAMMA，其使用多重集转换技术，允许在较高的抽象层次上开发逻辑并行性，已被证明是一种有效的并行程序构造方法。

1 引 言

在过程式语言如 Pascal、C 中，有一个很重要的表示顺序复合概念的算符";"，它的引入是由于过程式语言可以看成反映实际计算顺序特征的冯·诺依曼机器的抽象模型。因此，";"的使用在大多数情况下就是为了刻画这种计算的顺序性，而只有一些才是确实反映了程序的逻辑结构，例如在如下的程序段

```
ma; = max-array(a);
mb; = max-array(b);
m; = max(ma, mb);
```

中，(这里，a、b 是两个数组，max-array 是一个用户定义的计算数组中元素最大值的函数)第一个出现的算符";"就是为了反映计算的顺序性而引入的，而第二个出现的";"才是表示了程序段中最后一个语句和前两个语句的逻辑依赖性特征所必需的。

随着各种各样并行计算机体系结构的出现及并行计算模型的深入研究，人们认识到许多并行算法、并行程序中的顺序性有很多是人为引入的，即由于人们的思维、书写习惯、所使用语言的限制及实际运行并行程序的系统结构上的考虑所产生的。因此，为了能充分利用机器的特征及减少设计并行程序的开销，有必要把并行性的开发和讨论分成两个层次：一是物理的，它涉及任务如何在具体的机器系统结构上的分配问题；二是逻辑的，它关心的是程序构造，即如何在语言或模型一级反映实际问题的逻辑结构。从上面顺序计算的例子可以看到，要使得模型或语言能真正反映领域问题的特征，我们应使其不具有人为引入的顺序性。但是，目前的一些模型或语言并不具备这样的性质，下面我们分别从过程式语言和函数式语言来考察这种人为所引入的顺序性的特征。

考虑问题：给定一集合，找出该集合中具最大值的元素。

在过程式语言如 Pascal 中，我们可使用一个数组如 a:array[1 · · n]of integer 来表示集合，这样程序可以写成

```
m; = a[1];
for i; = 2 to n do
    m; = max(m, a[i]);
```

在函数式语言如 Miranda 中，可以使用表来表示集合，这样程序可以写成

```
maxset2(l) = if tail(l) = nil
             then head(l)
             else max(head(l), maxset2(tail))
```

我们可以很清楚地看到，过程式语言在控制结构上人为地引入了一个从左到右的比较程序，函数式与之相反，由于不得不使用递归定义的数据结构来表示集合，也人为地引入了一个从右到左的比较顺序。当然，在这些语言中可以引入显式的并行性，如 maxset2 可以改写成：

```
maxset3(l) = if tail(l) = nil
             then head(l)
             else let(l1, l2) = split(l)in
             max (maxset3(l1), maxset3(l2))
```

这里，split(l)表示将表 l 分成两个长度相等(或差 1)的子表。然而，在这种表示法中，也存在着人为引入的顺序性，如表的第一个元素就不可能和最后一个元素进行比较(除非它们分别是子表 l_1, l_2 中的最大元素)。

事实上，一个集合中的最大元素的计算可以通过对集合中的任意两个元素进行比较而求得，亦即，我们可以使用下述的抽象算法：

```
while 集合中的元素个数≥2
select 任意两个元素
compare 这两个元素
remove 值小的元素。
```

在 GAMMA 中，上述算法思想可以很清楚地表示成下述程序：

maxset4(s)=Γ((R,A))(s)where

$$R(x,y)=x\leqslant y$$

$$A(x,y)=\{y\}$$

这里 s 是一个集合(本文中的集合指的是多重集，即考虑元素在集合中出现的次数)，条件 R 规定了所选择的元素要满足的性质，而这些元素在集合中的出现将被函数 A 应用的结果所替代，要注意的是，在上述程序中并没有对比较的顺序作任何限制，即如果有多个不相交的元素对满足 R，则比较和代换甚至可以并行进行。

一种刻画 GAMMA 模型的直观方法是使用化学术语，多重集被看成是溶液，元素看成是溶液中的分子或离子，而 R 看成欲发生化学反应的分子要满足的条件，A 则描述了化学反应的结果。化学反应若终止于一个状态，即没有任何元素子集可满足 R，则该状态可看成该 GAMMA 程序的一个不动点。下面我们将介绍 GAMMA 语言的形式语法，并通过一些典型例子对 GAMMA 和过程式语言、函数式语言进行比较。

2 GAMMA 的形式描述

GAMMA 模型最早由 Banatre 和 Metayer 在 1990 年提出，GAMMA 是多重集操作一般抽象模型的英文缩写。在 GAMMA 中，最基本的数据结构是多重集，正是采用了多重集才使得不受限制地自然刻画复合数据结构成为可能。而 GAMMA 的控制结构，称之为 Γ 算子，如同我们在前面例子中看到的那样，Γ 反映了数据结构的无层次性和计算的混沌性。GAMMA 上的计算可以如下刻画：

$$\Gamma((R_1,A_1),\cdots,(R_m,A_m))(M)=$$
if $\forall\, i\in[1,m],\forall\, x_1,\cdots,x_n\in M,\neg R_i(x_1,\cdots,x_n)$
then M
else let $x_1,\cdots,x_n\in M$, let $i\in[1,m]$such that
$R_i(x_1,\cdots,x_n)$in
$\Gamma((R_1,A_1),\cdots,(R_m,A_m))((M-\{x_1,\cdots,x_n\})+A_i(x_1,\cdots,x_n))$

多重集上的运算一般有如下几种：

+：一个元素在 M_1+M_2 中出现的次数为其在 M_1 和 M_2 中出现次数之和；

−：一个元素在 M_1-M_2 中出现的次数为其在 M_1 和 M_2 中出现次数之差；

×：M 和 N 的笛卡儿积；

∪：一个元素在 $M_1\cup M_2$ 中出现的次数为其在 M_1、M_2 出现次数的最大者；

∩：一个元素在 $M_1\cap M_2$ 中出现的次数为其在 M_1、M_2 出现次数的最小者；

card(.)：card(M)为 M 中的所有元素的出现次数之和。

在上述定义中，(R_i,A_i) 是一对刻画反应的闭函数对，反应的结果是当 $R_i(x_1,\cdots,x_n)$ 为真时，用 $A_i(x_1,\cdots,x_n)$ 去替换 M 中的子集 $\{x_1,\cdots,x_n\}$；当 M 中没有任何元素满足反应条件(即 $\forall\, i\in[1\cdots m],\forall\, x_1,\cdots,x_n\in M,\neg R_i(x_1,\cdots,x_n)$)时，结果仍为 M，否则用 $((M-\{x_1,\cdots,x_n\})+A_i(x_1,\cdots,x_n))$ 代替 M(设 R_i 为真)并继续计算。要注意的是在定义中 $\forall\, x_1,\cdots,x_n\in M$ 表示 $\forall\, \{x_1,\cdots,x_n\}\subseteq M$，它和 $\forall\, x_1\forall\, x_2\cdots\forall\, x_n$ 不同。前者表示 $x_1,\cdots,x_n$ 是多重集中的不同元素，而后者却无此要求。从计算的定义中我们可以看到若对若干个 M 的子集反应条件都成立，则它们之间的选择是不确定的，特别若这些子集不相交，则反应可以并行独立地进行，这就是为什么 GAMMA 能表示程序的最大并行性的根本原因所在。在 GAMMA 语言中，我们可以使用通常的函数复合来表示一些顺序的概念、特别，R_i 和 A_i 可以通过模式匹配来定义，如 m where{m}=Γ((R,A))(M)表示抽取结果多重集中的单一元素。

3 GAMMA 的程序设计风格和方法学

本节我们首先通过一个例子对过程式语言、函数式语言和 GAMMA 的不同程序设计风格作一比较，然后对 GAMMA 的程序设计方法学进行较深入的讨论。

3.1 GAMMA 的程序设计风格

考虑问题：给定 N>2，求小于 N 的所有素数。

对于过程式语言来说，一个最简单的算法就是筛选法，它基于如下的事实：第 i−1 个素数是大于第 i 个素数且不被前面 i 个素数所整除的那个整数。由此我们可以导出下述性质：大于 $\sqrt{N}$ 而不被小于 $\sqrt{N}$ 所整除的数是整数。下面是用 CSP 类语言书写的一个过程式的并行程序：

```
begin
    sieve[0]:
        print!2;n:integer;n:=3;
        *[n≤N→sieve[1]!n;n:=n+2]
    ||
    sieve[i=1 ⌈√N⌉]::
        p:integer;
        sieve[i-1]?p;
        *[m:integer;sieve[i-1]?m→
          [multiple(m,p)→skip[]
           not(multiple(m,p))→sieve[i+1]!m]
          ]
    ||
```

```
    sieve[⌈√N⌉-1]::
        * [n : integer;sieve[⌈√N⌉]?n→print!n]
end
```

在上述程序中，我们定义了一个进程数组 sieve，其中每个进程在从其前继进程输入一个整数 p 后，即将 p 送给打印进程 print，然后将所有不是 p 的倍数的整数送给其后继进程。由于在 CSP 中，不具备动态创建进程的能力，因此必须确定所需的进程数，事实上，⌈√N⌉-2 个进程是必须的，其中 sieve[0]用于处理偶数，sieve[⌈√N⌉]+1 用于处理大于⌈√N⌉的整数。

对于函数式语言书写的下述求解程序，我们需要两个辅助函数：int-between(m,n)，用来产生大于 m 小于 n 的所有整数，filter(p,x)用于去除表 x 中的所有为 p 的倍数的整数。

```
primes(N)=prime_numbers(int_between(2,N))
int_between(m,n)=if m>n then nil
      else cons(m,int-between(m+1,n))
prime_numbers(l)=if null(l)then nil
      else cons(head(l),prime-numbers(filter(head(l),
      tail(l))))
filter(p,y)=if multiple(head(y),p)then filter(p,tail
(y))
      else cons(head(y),filter(p,tail(y)))
```

在 GAMMA 中，上述问题的求解方法在于从多重集{2,…,N}中移去那些是其它数的倍数的整数，结果的多重集必定包含所有小于 N 的素数，这是因为素数不可能从多重集中移去，而从多重集中移去的必定不是素数，下面就是上述思想的 GAMMA 程序：

```
prime_number(N)=Γ((R,A))({2,…,N})
      where R(x,y)=multiple(x,y)
            A(x,y)={y}
```

从同一问题的三种不同求解风格我们可以看到，GAMMA 语言最简单，但它却反映了问题的本质。对于过程式语言，我们必须考虑进程间的通讯、内存的管理等细节，而对函数式语言来说，由于必须借助递归定义的表来刻画求解的对象，因此必须在求解策略中显式表示表的构造和分解。而 GAMMA 语言把这些细节的考虑推迟到实现阶段(即物理并行性的分析和实现)去考虑，在这里它只是刻画了问题的本质即从集合{2,…,N}中去除所有为其它数倍数的那些整数。

由于没有给出特别的执行的顺序，GAMMA 语言可以很方便地在不同的系统结构上并行实现，当然，并行的过程式语言也可以在并行计算机上高效实现，但和 GAMMA 不同的是，对于后者程序设计者必须负责并行性的管理，即他(她)必须对其所书写的程序将在那种系统结构上的执行有所充分了解，他(她)必须针对特别的系统结构给出并行性在程序中的显式刻画。对于函数式语言来说，也有多种并行计算策略，如对上述例子，可以把 int-between，filter 和 prime-numbers 三个进程作为流水线来考虑，但总的来说，它们都没有 GAMMA 那么自由，因为在 GAMMA 中，任何两个元素间的比较都是不受限制的，也就是说，GAMMA能刻画关于上述问题的一个最大的并行度。

3.2 GAMMA 的程序设计方法学

GAMMMA 模型上的计算提示了一种崭新的程序设计方法。一个 GAMMA 将不是一个对状态进行修改的指令序列，也不是函数对参数的作用，而是对所有数据同时进行多重集转换操作。GAMMA 程序的开发包括数据表示的选择和施用于这些数据的转换类型，下面我们分别进行讨论。

(1)数据分解　GAMMA 中所提供的唯一的数据结构是多重集，多重集中的元素既可以是简单的也可以是复合的(甚至也可以是多重集)，但在 GAMMA 中没有提供递归定义的数据结构。因此，使用 GAMMA 进行程序设计时的首要任务就是寻找一种合适的数据的多重集表示方法，对于那些是基本类型的数据，如整数，则一般说来表示方法不难选择，如上述关于求小于 N 的所有素数的例子，就是把整数 N 分解成多重集{2,…,N}。对于复合数据结构如序列、树、图等，表示方法也是十分直接的，如树可以表示成树中结点和叶加上它们父结点信息组成的多元组的多重集，而一个序列可以表示成对(index,value)的多重集。

对于 GAMMA 来说，由于基本数据结构是多重集，因此，数据结构中的所有成分都可以直接存取。如果成分数据在数据结构中的位置和反应有关的话，则这些位置可在反应条件中指出。

(2)松弛法(Relaxation)　松弛法是数学中通过迭代法求解方程的一种典型方法，它首先假定一个解向量，然后在迭代中不断对解进行修正。在使用 GAMMA 进行程序设计时，我们也可以使用该方法来解决一些问题，即首先给出一个初始多重集作为解的一个估计，然后通过反应不断对多重集进行作修改，直到达到一个稳定状态为止。如上述求所有小于 N 的素数的例子，首先我们用{2,…,N}作为解的一个估计，然后通过去除那些不是素数的整数对该集合进行修正。直到达到一个稳定状态，即所有的数都是素数为止。

(3)数据展开和归约　使用松弛法进行程序设计

的一个限制是我们必须预先知道解的结构，这里我们再介绍两种基本技术：数据展开和数据归约。首先我们来看一个例子。

```
Fibonacci(n)=where
Fibonacci(n)=if n=0 then 1 else
                if n=1 then 1 else
                Fibonacci(n-1)+Fibonacci(n-2)
```

Fibonacci 级数可以用下述的 GAMMA 程序表示：

```
fib(n)=m where
    {m}=Sigma ( gen({n}))
gen(N)=Γ((R1,A1),(R2,A2))(N)where
    R1(n)=n>1
    A1(n)={n-1,n-2}
    R2(0)=true
    A2(0)=1
Sigma(M)=Γ((R,A))(M)where
    R(x,y)=true
    A(x,y)={x+y}
```

这里初始给定的数 n 由 gen 分解成由若干"1"组成的多重集，然后 Sigma 把这些"1"加起来以产生正确的解。由此，我们可以看出，数据展开指的是把值分解成项的集合，这个过程一直到所有的元素都是不可分解时才结束，它所使用的条件和反应函数都是一元的，旨在将值进一步分解。与数据展开相反，数据归约旨在通过反应将多重集归约为一个由单一的元素组成的多重集。

数据展开和数据分解是一对对偶的程序设计技术，前者将数值分解成简单成份组成的类，后者将独立的元素合成复杂的值。从 Fibonacci 级数的函数式语言表示和 GAMMA 表示间的比较我们可以发现，数据展开对应于函数调用，而数据归约对应于调用返回。事实上许多程序在 GAMMA 中的表示都可以使用松驰法、数据展开和归约给予解决。下面我们将通过一些例子揭示 GAMMA 的表示能力。

4 使用 GAMMA 进行程序设计

本节我们将用一些典型例子来说明如何使用上述介绍的三种方法进行程序设计。

4.1 数值计算

数值计算问题对于一个语言的表示能力来说是非常重要的。这里给出一个关于求阶乘的 GAMMA 程序：

```
fact(n)=Γ((R,A))({1,…,n})where
    R(x,y)=true
    A(x,y)={x*y}
```

该程序使用了典型的数据归约技术，要注意的是该程序对任何两个数进行乘法的顺序不作任何限制，这是过程式语言和函数式语言所无法刻画的。

4.2 排序问题

这里我们使用松驰法来表示排序的控制，即初始为对(index,value)组成的多重集，而作用的条件是该对不是已排好序，程序对所有那些未排好序的二元数组对进行交换，直到所有的元素按某种条件都排好序为止。程序可以写成：

```
sort(Array)=Γ((R,A))(Array)where
    R((i,v),(j,w))=(i>j)and (v<w)
    A((i,v),(j,w))={(j,w),(j,v)}
```

4.3 图问题

图问题是程序设计中经常要遇到的问题，这里我们使用一个求所有结点间最短路径的 GAMMA 算法来说明如何进行程序设计的问题。首先我们要选择数据表示的问题，显然对于结点(x,,y)，可以使用一个三元组(x,y,c)来表示在某一状态从 x 到 y 的最短路径为 c，初始时 c 为 x 到 y 的权，若 x 到 y 没有路径，那么 c 为∞。下面是具体的 GAMMA 程序：

```
shortest_path(G)=Γ((R,A))(G)where
    R((v1,v2,c12),(v2,v3,c23),(v1,v3,c13))
      =c13>c12+c23
    A((v1,v2,c12),(v2,v3,c23),(v1,v3,c13))
      ={(v1,v2,c12),(v2,v3,c23),(v1,v3,c12+c23)}
```

上述程序所使用的方法为松驰法。

4.4 进程间的同步

使用 GAMMA 可以非常自然地刻画并行、并发程序设计中的一些问题。但要注意的是，在这种情况下，我们注重的是多重集的状态，而不是最终的结果，换言之，多重集的状态可以被看成进程的状态。当然，使用 GAMMA 来刻画并发问题的基础还依赖于正确的实现。下面我们用一个哲学家问题来说明如何使用 GAMMA 来描述并发性。哲学家问题说的是五个哲学家围绕圆桌而坐，当他们思考问题感到饥饿时可用桌上的五把叉子中的两把饱食空心面。问题是如何找到一个协议来保证无死锁性和无饥饿性。在 GAMMA 中，这样一个并发系统的状态可以由一个包含桌上叉子及进食哲学家的多重集表示。初始状态为{F_0,F_1,F_2,F_3,F_4}，若哲学家 P_1 拿起了叉子 F_1,F_2 进食，则状态演变为{F_0,P_1,F_3,F_4}，若 P_1 进食完毕放回叉子则状态又转化为{F_0,F_1,F_2,F_3,F_4}。下面是描述哲学家问题的 GAMMA 程序：

```
philosophers=Γ((R1,A1),(R2,A2))({F0,F1,F2,F3,
                F4})
  where R1(Fi,Fi⊕1)=True
    A1(Fi,Fi⊕1)={Pi}
    R2(Pi)=true
    A2(Pi)={Fi,Fi⊕1}
```

这里 i⊕1=(i+1)mod 5，可以看到由于基本的同步机制是对项集合元素的原子操作，因此无死锁性是显然的，而无饥饿性的保证是由选择的公平性所保证的，当然这是模型具体实现时要考虑的问题。

人工 神经网络 面向对象 人工智能

计算机科学 1994 Vol. 21 №. 5

24-26

面向对象的人工神经网络

TP18

陆伟民
(同济大学结构工程学院 上海 200092)

摘 要 The artificial neural network models established with object-oriented programming were described in this paper. As a typical sample, the three-layer backpropagation neural network was selected to explain the creation of new classes by using Smalltalk/V。

一、引 言

面向对象编程(OOP)与人类思维和语言在表达模式上的接近而日益受到软件工程界的青睐，人工智能开发上也出现对它明显的倾斜。在诸多面向对象编程工具中，Smalltalk/V的直接编程环境、内设的完整类与方法、弹出式菜单、开发工具包等特色，提供的类封装、继承及多态性这些有力的OOP特性，逐渐成为受用户喜爱的OOP软件。

人工智能的重要分枝：神经网络方法已发展了很多不同的模型，如Adaline、Hopfield、ART、BAM、Hamming、Back-Propagation(B-P)等，其中B-P模型应用较广，成功地用于语言识别、模式分类、数据压缩等方面，为了进一步发挥它的效能，本文探讨与面向对象概念相结合的面向对象的人工神经网络模型建立方法。

陆伟民 教授

二、反向传输网络模型的B-P算法

神经元网络可看作是接收输入产生输出"黑盒"一类的信息处理系统，主要由处理单元(PE)和它们之间加权的联接构成。联接在表示上应包括标记与数值(w_{ij}，表示从PE$_i$联向PE$_j$的权)。权储存信息，其数值通过学习决定，即在学习过程中不断调整。对各个PE执行更新的运算就能使网络回想(recall)信息。

神经网络在组成上包含三个基本方面：如何组织成层和层间联接(拓扑)；信息如何储存(学习)；如何从网络中调出储存的信息(回想)。图1是典型的B-P网络，由一输入层、一输出层和至少一隐层组成，为简单起见，这里取单隐层，每一层与上、下层的所有PE相联，网络的B-P算法包括正向传输和反向传输过程，正向传输即回想时样本输入值从输入层经隐层传

5 结论

使用多重集转换机制可以以很优雅的方式解决许多领域中的问题。GAMMA作为一个并行计算模型，它的特点就是使程序设计者在设计并行程序的时候从无关的细节中解放出来而注重于问题的本质，这个思想和一些其它的并行计算模型如UNITY，Linda都有许多类似之处。特别，Berry和Boudol在GAMMA中引入了"隔膜"的概念产生了所谓的"化学抽象机"(ChAM)。在ChAM中，可以定义代数进程演算的模型、刻画并发λ演算，给出线性逻辑的计算解释。目前，GAMMA语言已成为并行计算模型研究中的一个热点，许多研究者正在探讨在GAMMA上实现函数式语言，研究如何在巨并行计算机上有效地实现GAMMA模型，同时在法国国家并行性研究计划中，GAMMA也占有重要的一席之地。

参考文献

[1]Banatre, J. P. et al., The GAMMA model and its discipline of programming, Sci. Comput. Program. 15 (1990), 55-77

[2]Banatre, J. P. et al., Programming by Multiset Transformation, CACM, Jan. 1993

[3]Berry, G. et al., The Chemical Abstract Machine, TCS, 1992

[4]黄林鹏 孙永强，并发λ演算及在并行抽象机上的实现，《全国第二届系统结构会议论文集》1992，10