函数式语言并行化的方法

古志民　郑守淇
（西安交通大学计算机系　西安710049）

摘　要　In this paper, the method of parallel implementation of functional languages and version of Common LISP is presented by analysis of functional programming for concurrent and distributed computing and some parallel lisp languages such as Multilisp and pailisp , and some implementation considerations are given.

关键词　LISP, Concurrent computing, Distributed computing.

一、引言

Lisp 作为一个表处理语言，有着悠久的历史，而 COMMON LISP 提供了丰富的数据集和多程序设计模式：函数的，强制的和面向对象的[5,6]。我们以 COMMON LISP 为蓝本完成了 XJD-LISP/CLOS，它是为我国曙光一号机配备的，现已通过国家教委组织的鉴定。最近我们正在开发 COMMON LISP 的并行版本，作为863计划资助项目，这个工作很有意义。该并行系统将加速计算，并且还能对 XJD-LISP/CLOS 提供支持。本文就函数式语言并行化的方法进行了探讨。

二、函数式语言并行化方法的探讨

2.1　对并行和分布计算函数程序设计的分析

并行计算的语言可在传统的过程语言和函数语言中实现。过程语言为用户提供了高度的运行控制能力；函数语言特点是易读易写易验证。如果两者特点结合到一起，则会既简单又有效。文[1]中提出了一个小的简单说明符集合，可控制函数语言的运行行为。参数传递可以是值和名传递，也可是惰性求值。在分布式系统里用户可以说明应在当前处理器、任意处理器还是在一个含有特定数据的特殊处理器上工作，当然这些说明符不能影响函数程序的意义[1,2]。

这些说明符的定义如下：

(1)Value：参数不是函数时可传值；

(2)Name：参数是函数时可传名；

(3)Fork/join：一个可并行执行的机构；

(4)Speculation：常用在数据驱动求值时。开始求参数的值但求值未结束将返回一个结果，这种求值基本上是惰性求值的并行版本，若有可用的处理器则允许参数在需要前求值，它保持按名传递语义，不需一个额外存值的"公平"调度，同 Multilisp 的 FUTURE 相似。

(5)Anywhere：在分布系统中指示可在任一处理器中求值；在共享存贮的模式中无意义。

(6)At：说明一个子表达式可在某个带有特定数据的结点工作。

几点说明：

(1)在顺序函数语言里当提供按名和按值传递时惰性求值变得无用[2]。

(2)若有"ANYWHERE"而无"AT"，则只能进行简单的分治式求值。

(3)举例

```
[1]conditional_or(a,name b)(=
    if a=true then true else b
    a 被求值，b 不被求值。
[2]funny_or(x,y)
    where funny_or(a,speculation b)(=
    if a then true else b
```

如 x 真，即使 y 未求出，函数将真。另一方面，如果存在足够的处理器，x 和 y 可并行求值。

```
[3]factorial(n)(=prod(1,n)
    where prod(i,j)(=
      if i=j then i
        else (prod(i,mid)anywhere) * (prod
        (mid+1,j)anywhere)
```

古志民　博士生。郑守淇　教授，博士生导师，主要从事并行处理和智能机的研究。

where mid<=(i+j)div 2

这将带很多分治算法工作。

(4)不确定性。人们已提出一些有关的说明符，象 amb。amb 可加到以上的说明符里，例如，f(name x)(=x=x.x=amb(a.b)，将返回真或假。

2.2 对 Multilisp 的分析

Multilisp 是 Lisp 方言 Scheme 的带有说明符的一个并行版本，允许副作用存在，并行说明符和对抽象数据类型的支持缓和了副作用处理的复杂性，其实现基于共享存贮模型[5]。

它的说明符定义如下：

(1)PCALL：(PCALL F A B C) 当 f 应用到 a,b,c 时，将对表达式 F,A,B,C 并发求值。

(2)FUTURE：(FUTURE X) 立即返回一个 FUTURE 作为表达式 X 的值，且并发求 X，当 X 有值时就替换 FUTURE。

(3)DELAY：(DELAY X) 当其它计算需要 X 值时才求值。

几点说明：

(1)PCALL 是很有用的，因为函数调用很普遍。

(2)FUTURE 最初不确定，计算它的值后便确定了。但很多操作，象参数传递，不需知道它的操作数的值，就可传递了。下面给出一个快速排序程序，FUTURE 的应用揭示了大量的并行性。

```
(defun qsort(l)(qs l nil))
(defun qs(l rest)
(if (null l)
  rest
    (let((parts(partition (car l)(cdr l))))
    (qs (left-part parts)
      (future (cons (car l)(qs(right-part parts)
rest)))))))
(defun partition (elt lst)
  (if (null lst)
    (bundle-parts nil nil)
    (let ((cdrparts (future partition elt (cdr
lst))))
      (if (elt (car lst))
      (bundle-parts(cons car lst)
        (future(left-part cdrparts)))
        (future (right-part cdrparts)))
      (bundle-parts(future(left-part cdrparts))
        (cons(car lst)
        (future(right-part cdrparts))))))))
(defun bundle-parts(x y)(cons x y))
(defun left-part(p)(car p))
defun right-part (p)(cdr p))
```

(3)(defun ints-from(n)

(cons n (delay(ints-from(+ n 1)))))

2.3 Pailsp 的分析

Pailisp 是一个基于 SCHEME 和共享存贮的并行 Lisp 语言，提供 PCALL，PAR_AND，PAR_OR，PCOND，SPAWN，SUSPEND，PAR，FUTURE 和 EX_LAMBDA 等说明符，最重要的和最有趣的一个是 CALL/CC，它是 SCHEME 和 CONTINUATION 到并发的扩展。SPAWN，SUSPEND，CALL/CC，EX_LAMBDA 被加到 Pailisp 的核心上。

2.4 函数式语言并行化的方法

函数语言有简单的语义，惰性求值可以优化顺序的函数求值时间，但不能优化它的空间。若提供名和值传递时，惰性求值是无用的，因为惰性求值不可能放弃和重计算表达式。机会调用是一种并发求值的方法，投机(Speculation)的求值是有用的。ANYWHERE 和 AT 可用在分布系统中，在大多数情况下，采用以上的说明符可实现并行处理。

函数式语言并行化的方法：

(1)可重写一些基本函数，象第三节中(2)。

(2)可加并行说明符象 PCALL，FUTURE，DELAY，PAR-OR，PAR-AND，PAR，PCOND，EX-LAMBDA，AT(它们的意义可扩充)到语言中。

三、对 COMMON LISP 并行版本的探讨

我们如何设计 Common Lisp 的并行版本呢？先考虑以下几点：(1)ANYWHERE 和 AT 只用在分布系统里，共享存贮适于中小粒度，而分布系统适于大粒度的并行。(2)我们需重新定义某些基本函数以适于并行求值(这种方法 MULTILISP 未用)，例如：

[1]cons(a,name b)(=(a b)

[2]car(a,name b)(=a

[3]cdr(a,name b)(=b

现在我们已有 COMMON LISP 的并行版本：包括可重写的基本函数和说明符，象 PCALL，FUTURE，DELAY PAR-OR，PAR-AND，PAR，PCOND，EX-LAMBDA，AT(它们的意义可扩充)。

在共享模型中，一个 COMMON LISP 的程序先被编译成 PCODE 语言形式，PCODE 被一个解释程序 P 解释，P 和栈的 CACHE 放在每个处理器的局存中，共享的数据结构(PCODE 程序)在一个废料收集堆中，该堆分布在存贮模块中。PCODE 的程序可看作共享指向废料堆的指针的任务集合。每个任务有三个指针：程序指针，栈指针和环境指针。PCODE 是一个面向栈的语言，大多操作数可从当前的任务栈弹出，可放操作的结果进栈。当任务挂起时，可完全把栈的内容放在堆中。当用 PCODE FUTURE 时，新的任务被创建，它的环境指针继承了父母的。PCODE 中的数据可分成两类：类型域和值域。

(下转第25页)

应的观点打相同分数的人数。例如对于本课程的必要性，8个同学是一致认同的态度；大部分同学更愿意用英语上此课。

八名学生对课程反映的调查统计情况

	1	2	3	4	5
一、本课程对有效地学习并行程序设计是有必要的					8
二、每次课的内容太多	4	2	1	1	
三、PROJECT 作业对掌握有关内容是有帮助的				1	7
四、所安排的机时不够完成 PROJECT	1		3	1	3
五、小测验太难了	2	1	3	2	
六、最好用汉语上课	4	2		2	

作为任课教师，有如下几点体会：

(1)并行程序设计是一个非常引人入胜的活动。我们不仅要关心程序的正确性，还要在程序的运行效率上投入较大的注意力。通常一个程序8个人写出8个结果：程序都是对的，但加速比可能有很大差别。例如最后一个 QUIZ，有的程序的加速比可以达到10以上，而有的还不能超过3(不论用多少处理器)。

(2)讲课时数还应稍多一些，例如16小时，就可以覆盖关于"计算-汇集-广播模型"以及 REPLICATED WORKER 模型中的程序终止检测等内容。

(3)每周只安排一次课是重要的，否则学生没时间完成作业。

(4)在一个小时里要完成哪怕是很简单的程序也是困难的，从上述 QUIZ 得分情况可以看出这一点。第三个 QUIZ 大家都做得比较好是因为事先将有关内容大致通报给了学生，请他们有所准备。

致谢 谨在此特别感谢加拿大 Concordia 大学的陶立新博士，如果不是他向我们介绍 Dr. Lester 的书，上述工作都将不会发生。

参考文献

[1]Henry Burkhardt, The Smoke is clearing, and parallel processing has won, IEEE Spectrum, Jan. 1994, page 49

[2]Bruce P. Lester. The Art of parallel Programming. Englewood Cliffs, NJ: Prentice Hall Inc., 1993

[3]刘宏伟，李晓明，Multi-Pascal：一个经济有效的并行程序设计研究工具，哈尔滨工业大学计算机系技术报告，PACT-TR-94-025。

(上接第22页)

PCODE 操作要求它的操作数有特定类型(象 ADD)。同步需不同处理：对 FORK/JOIN 和产生/消费型可引入 REPLACE_IF_EQ 原语，FUTURE 需特别处理；对副作用型可引入(WAIT S)，(SIGNAL S)，(SUSPEND F)和(ACTIVE S)原语。任务调度策略是先满足某个任务，其次是别的，这是一个不公平的调度。废料收集的算法是基于复制和 BACKER 增量型废料收集的，在分布系统中它的实现更复杂。

※ ※ ※ ※ ※

鸣谢 借此机会感谢赵银亮博士，党华锐和 Abdulrazaque Memon。

参考文献

[1]Burton F.W., Annotations to control parallelism and reduction order in the distributed evaluation of functional progarams. Transactions on Progress in Language and Systems 6(2)159-174(1984)

[2]Burton F. W., Functional programming for Concurrent and Distributed Computing, The Computer Journal 30(5)437-450(1987)

[3]Robert H. Halstead, JR., MULTILISP: A Language for Concurrent Symbolic Computation, ACM Transactions on Programming Languages and Systems, 7(4)501-538(1985)

[4]Peyton Jones S. L., Parallel Implementations of Functional Programming languages, The Computer Journal 32(2)175-186(1989)

[5]Kenneth H. et al., The Philosophy of Lisp, Commun. ACM 34(9)40-47(1991)

[6]Layer D. K. and Richardson C., Lisp Systems in the 1990s, Commun. ACM 34(9)48-57(1991)

[7]Baker H., Actor Systems for Real-Time Computation, Tech. Rep. Tr-197, Massachesetts Institute of Technology, Laboratory for Computer Science, Cambridge, Mass., Mar. 1978