基于线程的 Java 程序自动并行转换技术*

刘 英[1], 刘 磊[2], 张乃孝[1]

[1](北京大学 数学科学院 信息科学系,北京 100871);

[2](吉林大学 计算机科学系,吉林 长春 130023)

E-mail: liuying@water.pku.edu.cn; liulei@mail.jlu.edu.cn; naixiao@pku.edu.cn

http://www.pku.edu.cn; www.jlu.edu.cn

摘要: Java 程序的并行化研究是一个重要课题.提出一种 Java 程序的自动并行转换技术,并充分利用 Java 语言本身提供的多线程机制,通过操作冲突性检测等方法将串行化的 Java 程序自动转化成并行化程序.使得转化后的并行化程序在多处理机操作系统的支持下,能在共享内存的多处理机系统上运行,从而提高了程序的运行效率.

关键词: Java 程序;自动并行化;多线程;无冲突性操作;有效线程化方法

中图法分类号: TP31. **文献标识码**: A

近年来,随着多处理机系统的成熟与完善,使得程序的并行执行成为可能,也使得人们对自动并行化的研究有了更大的应用价值.程序自动并行化是利用某种并行化技术,对原串行化程序进行全面的分析,自动地将串行化程序转化成并行化程序.程序自动并行化的关键是采用什么样的并行化技术.并行化技术越合理,程序的并行度就越高,从而程序提高的效率就越大.

以前的函数式语言如 Lisp,过程式语言如 Fortran90,对象式语言如 C++等,都有各自的自动并行转换器.Java[1,2]语言自面世以来,因其自身的诸多优点,如简单性、可移植性、健壮性、安全性、多线程机制等而受到了广泛关注.但由于 Java 是解释执行字节码,其运行速度一直是人们比较注意的问题.

Java 本身是一种面向对象式语言,这种语言与现实模型相对应,而现实模型中的对象又往往是并发活动的,因此,面向对象方法具有潜在的并行性.并且,Java 提供的多线程机制使得程序的并行执行成为可能.但对程序员来说,并行化程序设计仍然不是一件简单的事,程序员不仅要熟悉 Java 的线程机制,而且还要充分考虑有关数据的共享.如果有了 Java 程序自动并行转换器,程序员就可以完全不必考虑并行的问题,而只需按照常规的思想编写串行化程序,经过自动并行转换器的分析,可以将原串行化程序转换成用并行化程序,从而提高程序的运行效率.所以,设计 Java 程序自动并行转换器具有重大的意义.

1 多线程计算模型

Java 是为数极少的语言本身提供多线程的语言之一,它是通过多线程机制来实现程序并行化的.为便于理解,让我们先了解一下什么是多线程.

* **收稿日期**: 1999-06-15; **修改日期**: 2000-01-13

基金项目: 国家自然科学基金资助项目(69983001);教育部骨干教师基金资助项目

作者简介: **刘英**(1973-),女,山东五莲人,博士生,主要研究领域为软件理论,软件自动化;**刘磊**(1960-),男,吉林长春人,教授,主要研究领域为程序设计语言,软件自动化;**张乃孝**(1942-),男,江苏镇江人,教授,博士生导师,主要研究领域为软件形式化.

关于线程还没有明确的概念,但我们可以这样理解:线程[3]是进程内的一连串指令的执行,一个进程可以产生多个线程,它们之间共享全部的进程地址空间和进程资源.用户可以按照需要编写拥有多个线程的程序,理论上,用户进程可以创建任意多个线程.这些线程可以在不同的处理机上同时执行.实际上,线程就是程序中的一个控制流程,每个线程都有自己的资源作为执行的情景(如程序记数器和执行堆栈).但是,程序中的所有线程仍然共享许多资源(如内存空间或已打开的文件).因此,线程也称为"轻负荷进程".它与进程或运行程序一样是一个控制流程,但比一个进程更容易生成或销毁,这是因为它所涉及的资源管理更少.

一个多线程的计算在各个线程之间可以存在数据依赖关系,这是一种"生产者/消费者"关系.多线程的计算模型[4]如图 1 所示.

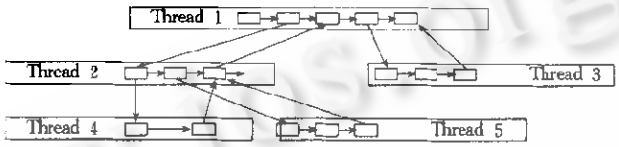

Fig. 1 Multi-Threading computing model

图1 多线程计算模型

其中,"→"代表顺序关系;"↙"、"↘"表示线程生成边;"↖"、"↗"表示数据相关边;"▭"表示任务.每个线程都可以生成多个线程,这样,线程就由生成边连接成一个层次激活树.

Java 语言的多线程机制提供了一个称为 Thread 的类,该类及其派生类的对象均为线程,线程可以与系统中的其他成分并行执行.另外,还提供了一个 Runnable 接口.无论是 Thread 类还是 Runnable 接口,线程中所有活动都是通过 run()方法来实现的,编写多线程程序的关键是如何定义线程控制块 run()方法.这种对象间的并行性属于任务级并行性[5].

2 操作执行顺序无冲突的基本概念

Java 是一种面向对象式语言,其程序是由一个或多个类组成,任意一种方法都必须在某个类中定义,并且,一个对象的行为是通过它所接受的消息来定义的.针对 Java 程序的特点,我们认为 Java 程序的计算都是由操作组成的.

2.1 无冲突性操作的形式化描述

以下我们将给出有关操作以及无冲突性操作的形式化描述.

定义 1(操作). 接收对象 r、点运算符(.)、方法名 m 和实参序列 $P=\{p_1,p_2,\ldots,p_n\}$ 构成一个操作.形如 $r.m(P)$,其中 $m()$ 是在类中定义的方法,方法名 m 标识定义这个操作的所具有行为的方法.

定义 2(对象的状态). 每个对象都用一组实例变量来标识其属性,即对象在某一时刻,其实例变量的值称为此时对象的状态,用符号 σ 表示.

定义 3(操作执行顺序). 假设有两个操作 op_1 和 op_2,如果有计算先执行 op_1,再执行 op_2,则操作 op_1 和 op_2 的执行顺序记为 $op_1 \leqslant op_2$.

约定: 本文中提到的"两个操作执行"是指这两个操作都是原子化执行.

定义 4(程序的状态). 将某一时刻所有对象的状态、全局变量及类变量的值称为此时程序的状态.

定义 5(操作执行函数). 设有操作 $r.m(P)$,操作执行以前程序有状态 σ 存在.此操作执行结

束后对程序状态的影响可用如下函数表示:$MS[\![r.m(P)]\!]\sigma=\sigma'$.其中$\sigma'$表示操作执行结束后的程序状态,则操作执行函数→可表示为$\langle r.m(P),\sigma\rangle\rightarrow\langle\sigma'\rangle\,|\,\sigma'=MS[\![r.m(P)]\!]\sigma$.

定义 6(在状态σ下的无冲突性操作). 设有两个操作$r_1.m_1(P_1)$和$r_2.m_2(P_2)$,在这两个操作均未执行前程序状态为σ.

若操作执行顺序为$r_1.m_1(P_1)\leqslant r_2.m_2(P_2)$,有$\langle r_1.m_1(P_1),\sigma\rangle\rightarrow\langle\sigma_1\rangle\,|\,\sigma_1=MS[\![r_1.m_1(P_1)]\!]\sigma$,且$\langle r_2.m_2(P_2),\sigma_1\rangle\rightarrow\langle\sigma_2\rangle\,|\,\sigma_2=MS[\![r.m(p)]\!]\sigma_1$.

若操作执行顺序为$r_2.m_2(P_2)\leqslant r_1.m_1(P_1)$,有$\langle r_2.m_2(P_2),\sigma\rangle\rightarrow\langle\sigma'_1\rangle\,|\,\sigma'_1=MS[\![r_2.m_2(P_2)]\!]\sigma$,且$\langle r_1.m_1(P_1),\sigma_1'\rangle\rightarrow\langle\sigma'_2\rangle\,|\,\sigma'_2=MS[\![r.m(P)]\!]\sigma'_1$.

如果满足以下条件,我们说操作$r_1.m_1(P_1)$和$r_2.m_2(P_2)$是在状态σ下的执行顺序无冲突操作:$\sigma_2=\sigma'_2$,简称状态σ下的无冲突性操作.

定义 7(无冲突性操作). 设有两个操作$r_1.m_1(P_1)$和$r_2.m_2(P_2)$,如果在任意状态σ下,$r_1.m_1(P_1)$和$r_2.m_2(P_2)$都是无冲突性操作,则操作$r_1.m_1(P_1)$和$r_2.m_2(P_2)$是无冲突性操作.

若某操作集合中任意两个操作都互为无冲突性操作,则这个集合中的操作互为无冲突性操作.

定义 8(独立性操作). 设有操作op_1和op_2.op_1读的变量的集合为RD_1,写的变量集合为WR_1;op_2读的变量集合为RD_2,写的变量集合为WR_2.如果满足如下两个条件,我们说操作op_1和op_2互为独立性操作:(1) $\forall\ x\in RD_1\cup WR_1: x\notin WR_2$;(2) $\forall\ x\in RD_2\cup WR_2: x\notin WR_1$.

若存在一个操作集合中的任意两个操作都互为独立性操作,则这些操作都互为独立性操作.

定理 1. 独立性操作一定是无冲突性操作.

证明:由独立性操作的定义可知,若操作集合中的操作都互为独立性操作,也就是说,这些操作之间没有数据依赖关系.既然操作之间没有数据依赖关系,则操作的执行顺序一定不会对程序的执行结果有任何影响,则这些操作是无冲突性操作.

利用某种方法检测某操作集合中的操作是否为无冲突性操作的技术,称为操作冲突检测技术.

2.2 操作冲突检测算法描述

2.2.1 操作中读写变量集合求解算法

由以上定义我们可以看出,进行操作可交换性检测的主要问题在于找出操作中读写变量的集合[6].

需要说明的是,我们在求解读写变量集合时,还可能遇到别名问题.Java 语言中虽然没有了指针类型,但对象的引用同样会产生别名问题[7].对具体工作而言,找到一个好的别名分析算法是必须的.因此,我们对面向 Java 的别名分析单独进行了详细的研究,在文献[8]中还进行了详细的描述.所以在下面求解读写变量集合的算法中不再考虑别名问题.

求解读写变量集合的算法用函数加以描述为(w表示操作写变量的集合,r表示操作读变量的集合):

◆求方法m写变量集合的函数

方法中的语句只有赋值语句对写变量集合产生影响,其他语句都是空操作,方法的定义如下:

m::=statements　　statements::=statements; statement | statement

statement::=assign | other_statements　　assign::=$v=E$

赋值语句对写变量的影响函数为 assign$\times V->V$　$Ms[\![v=E]\!]w=\{v\}\cup w$

其他语句对写变量影响函数为 otherstatements$\times V->V$　$Ms[\![\text{otherstatements}]\!]w=\text{skip}$

方法 m 对写变量的影响函数为 $M\times V->V$ $Ms[\![m(P)]\!]w=Ms[\![\text{statements}]\!]w$

◆形实参结合函数 Bindings:$OP\times M\times V->V$ $\text{Wbindings}[\![r.m(P)]\!]m(P')w=w[P/P']$

◆接收对象替换 Replace:$OP\times M\times V->V$ $\text{Wreplace}[\![r.m(P)]\!]m(P')w=w[r.iv/\text{this}.iv]$

◆求操作 $r.m(P)$ 写变量集合的函数

操作写变量集合影响函数 Write 为 $M\times OP\times V->V$

$\text{Write}[\![r.m(p)]\!]m(p')w=\text{let } w'=Ms[\![m(P')]\!]w \text{ in let } w''=\text{Wbindings}[\![r.m(P)]\!]m(P')w' \text{ in let } w'''=\text{Wreplace}[\![r.m(P)]\!]m(P')w'' \text{ in } w'''$

◆求方法读变量集合的函数

方法中的所有语句都有可能对读变量集合产生影响.而在赋值语句中,赋值号右边出现的变量都是被读的变量.

首先,定义变量在表达式中出现的判断函数为 IsExistofE(v,E),若变量 v 在表达式 E 中出现,则此函数为真,否则为假;再定义变量在语句中出现的判断函数为 IsExistofS(v,statement),若变量 v 在语句 statement 中出现,则为真,否则为假.

方法的语法描述如下:

m::=statements; statements::=statements;statement|statement;

statement::=assign|otherstates; assign::=v=E;

赋值语句对读变量的影响函数为 assign$\times V->V$

$Ms[\![V=E]\!]r=\text{let } r'=\{v\}\cup r \text{ with IsExistofE}(v,E) \text{ in } r'$

其他语句对读变量的影响函数为 otherstates$\times V->V$

$Ms[\![\text{otherstates}]\!]r=\text{let } r'=\{v\}\cup r \text{ with sExistofS}(v,\text{otherstates}) \text{ in } r'$

方法影响函数为 $M\times V->V$ $Ms[\![m(P)]\!]r=Ms[\![\text{statements}]\!]r$

同理,可定义如下必须的函数:

◆形实参结合函数 Bindings: $OP\times M\times V->V$ $\text{Rbindings}[\![r.m(P)]\!]m(P')r=r[P/P']$

◆接收对象替换 Replace: $OP\times M\times V->V$ $\text{Rreplace}[\![r.m(P)]\!]m(P')r=r[r.iv/\text{this}.iv]$

◆求操作读变量集合的函数

读变量集合影响函数 Read 为 $M\times OP\times V->V$

$Ms[\![r.m(p)]\!]m(p')r=\text{let } r'=Ms[\![m(P')]\!]r \text{ in let } r''=\text{Rbindings}[\![r.m(P)]\!]m(P')r' \text{ in let } r'''=\text{Rreplace}[\![r.m(P)]\!]m(P')r'' \text{ in } r'''$

2.2.2 操作冲突检测

在已知每个操作读写变量集合的基础上,可以对操作是否为无冲突性操作进行检测.根据定理1可知,如果操作是独立性操作,则它们一定是无冲突性操作.判断两个操作是否为独立性操作的检测算法 Independent(op_1,op_2)描述如下:

```
Independent(op1,op2)
    RD1=Read(op1); RD2=Read(op2); WR1=Write(op1); WR2=Write(op2);
    CR1=RD1∩WR2; CR2=RD2∩WR1; CR3=WR1∩WR2;
    if (CR1≠∅) or (CR2≠∅) or (CR3≠∅) return false;
    return true.
```

如果两个操作都不是独立性操作,则将进一步判断这两个操作是否为无冲突性操作.由无冲突性操作定义可知,只有满足程序状态相同这一条件,才能确定这两个操作是无冲突性操作.在判断

这个条件时需要采用符号执行技术[9,10]. 在进行符号执行时,以这两个操作均未执行前的状态作为符号执行的初始状态,操作的符号执行结束后,得到的程序状态都是符号表达式. 通过比较符号表达式,对这个条件作出判断.

无冲突性操作的判断算法 NoConflict(op_1,op_2)描述如下:

```
NoConflict(op1,op2)
    if (independent(op1,op2)) return true;
    ⟨σ1⟩=SymbolicallyExecute(op1,op2);
    ⟨σ2⟩=SymbolicallyExecute(op2,op1);
    if (σ1≠σ2) return false;
    return true;
```

3 自动并行转换技术

Java 的多线程机制的实现不同于 CC++的并行机制,Java 定义了一个线程类和一个 Runnable 接口,可并行的计算在 run()方法中实现,我们关心的是类中的哪个方法应该转换成 run()方法. 下面首先给出有效线程化方法的定义.

定义9(有效线程化方法). 设在某一类 c 中定义了方法 $m()$,若对类 c 的任意实例 r_i 发消息 $m(P)$,则产生一个操作集合 op_set=$\{r_i.m(P)\}$,如果集合 op_set 中的操作都互为无冲突性操作,我们就把这样的方法 $m(P)$称为有效线程化方法.

如果某一方法是有效线程化的,就把此方法中的计算在 run()方法中实现.

下面给出方法 $m(P)$是有效线程化方法的条件.

3.1 有效线程化方法的判定

设在某一类 c 中定义了方法 $m(P)$(P 为形参序列). 若对类 c 的任意实例 r_i 发消息 $m(P')$(P' 为 P 对应的实参序列),则产生一个操作集合 op_set=$\{r_i.m(P')\mid r_i$ 为接收对象,$m(P')$为访问方法$\}$. 集合 op_set 中的元素可归纳为如下几种情况:

情况1.

(1) $m(P')$是静态结合;(2) r_i 代表不同的对象;(3) 方法 $m(P')$不改变类变量和全局变量;(4) op_set 集合中任意对象 r_i 和 r_j 无共享变量;(5) 方法 $m(P')$中没有对对象参数的修改. 如果同时满足以上5个条件,则集合 op_set 中的所有操作都是无冲突性的,此方法是有效线程化方法,并且,此方法转换成 run()方法时可以不加任何同步机制.

情况2.

(1) $m(P')$是静态结合;(2) r_i 可以代表相同对象;(3) $m(P')$只写局部变量. 如果同时满足以上3个条件,同情况1,此方法也是有效线程化方法,可以不加任何同步机制就写成 run()方法.

情况3.

(1) $m(P')$是静态结合;(2) r_i 可以代表相同对象;(3) 对不同对象 r_i 和 r_j 发消息 $m(P')$产生的操作 $r_i.m(P')$和 $r_j.m(P')$,是无冲突性操作. 如果同时满足以上3个条件,此方法也是有效线程化方法,但当将此方法转化成 run()时,需加上适当的同步机制,以保证对共享数据的访问不会产生冲突. 对这种情况进行判断时,我们利用了操作冲突检测技术. 这种技术的应用扩充了可并行的程度,将不可并行的部分通过同步机制来控制,从而实现了部分并行,大大提高了此自动并行转换器的应用价值.

对于情况3,方法 $m()$ 对应的所有操作组成的集合中的所有操作都是无冲突性的,这意味着集合中的操作两两执行顺序无冲突.为了便于分析,我们将所有的操作都归纳为以下4种情况.

(1) $\langle r_i.m(P'),r_i.m(P')\rangle$:接收对象相同参数也相同.

(2) $\langle r_i.m(P'_1),r_i.m(P'_2)\rangle$:接收对象相同但参数不同.

(3) $\langle r_i.m(P'),r_j.m(P')\rangle$:接收对象不同但参数相同.

(4) $\langle r_i.m(P'_1),r_j.m(P'_2)\rangle$:接收对象和参数都不相同.

在进行操作冲突检测时,用符号执行的方法把操作在不同顺序下执行的结果用符号表示出来,然后比较符号表达式是否相等,如果相等,则这两个操作是无冲突性操作.否则,操作是有冲突性的,方法 $m(P)$ 就不可线程化.无冲突性操作的判断算法前面已经给出,在此不再赘述.

需要说明的是,以上3种情况我们都没有考虑方法是动态结合的情况.因为当方法可以是动态结合时,根本无法通过对源程序的静态分析确定其到底是与哪个方法结合,所以也无法确定应该把哪个方法转换成 run()方法.因此,当遇到动态结合的操作时,就认为此操作是不可线程化的.

3.2 问题解决策略

程序并行执行就是为了提高程序的执行效率.而最能提高优化和并行化效率的关键体现在循环上,因此,应该把重点放在循环上.我们对程序控制流图进行模块划分,然后抽取其循环模块,并对这些循环模块进行分析.为便于分析,假定循环内的计算都是操作,若不是一个操作,可以将几个计算抽象成一个操作模块.

如果在一个循环模块内,各语句之间有数据依赖关系,则需要增加等待机制才能保证程序执行结果的正确性,那么,在循环上就不会存在并行了.所以,当循环模块内有数据依赖关系时,我们需要分析增加等待机制会不会提高程序的执行效率以决定方法是否线程化.当存在循环嵌套时,从内往外分析.

另外,由于 *Java* 允许使用转向语句,当对程序进行静态分析时,这种非结构化的程序需要转化成结构化程序,因为这方面的工作已经比较成熟,所以,在此不再详细介绍.

3.3 自动并行转换器实现算法描述

划分模块 SplitModule(G).设程序的控制流图为 $G=(N,E)$,对 G 划分模块 SplitModule(G),然后从中找出循环模块集合 $LOOP=\{loop_i|0\leqslant i\leqslant n,n$ 为循环模块的个数$\}$.

判断是否为有效线程化方法 IsThread(m).　当需要判断某个操作所涉及的方法是否为有效线程化时,利用读写变量的集合判断其属于以上哪种情况,前面已经介绍过,此处不再赘述.

分析每个循环模块中有无数据依赖关系 DataDependence(LOOP).　如果循环模块内的所有操作与模块内的其他计算都没有数据依赖关系,则这些操作对应的方法都可以转化成 run()方法.相反,如果循环模块内的某一操作与模块内的其他计算之间有依赖关系,则此操作对应的方法不能直接转化成 run()方法,而需要进一步分析效率.

另外,我们规定凡是有输入和输出操作的方法都是不可线程化的.因为,并行输入/输出语义与串行输入/输出的语义不相符合.

3.4 程序转换中的关键问题

▲参数的传递

在 Java 语言中,Runnable 接口中的 run()方法的方法说明是:public void run(),即不带参数的,而操作一般都是有参数传递的.所以在由串行到并行转化时,存在消除参数的问题.我们采用给

当前类增加成员变量的办法,通过这些成员变量来传递参数.经实践证明,此方法是可行而有效的.

▲函数的过程化

从以上方法run()的说明可以看出,可线程化的方法都相当于一个过程.所以,当某个有效线程化的方法有返回值类型时,需要将这样的方法转化成没有返回值类型的过程.我们同样采取给类增加成员变量的方法来实现函数到过程的转换.

▲多线程化方法的统一化

Java语言规定,线程的所有活动都是通过run()方法实现的,并且每个类中只能定义一个run()方法.针对Java语言的诸多限制,我们在进行程序转换时不得不考虑采取一些切实可行的办法.而我们的策略是,如果判断出某个方法是可线程化的,就把这个方法改写成run()方法.所以当某一类中存在多个有效线程化方法时,鉴于Java语言自身的特点,我们采用增加类的成员变量作为标号来解决这一问题.

▲两套代码

同一个方法,可能在不同的循环模块中产生不同的操作,而不同的循环模块其数据依赖关系是不同的,从而有些操作可并行化,而对同一方法的另外一些操作就不可并行化.这就需要在最后生成的代码中产生两套不同的代码,一套是线程化的,另一套是串行化的.这样,在不同的情况下就可以调用合适的代码.

3.5 实验结果

我们对串行化程序转化成并行化程序后的效率进行了多次测试,并对转化前后的效率进行了比较.例如,在一个单处理器上运行一个两万次循环计算的程序,等待结果的时间让人难以忍受,你可能要等3分多钟才能看到最终计算结果.而当我们把这样的程序通过自动并行转换器转换成含有线程机制的并行计算时,转换后的程序用两个线程来实现两万次计算,即每个线程执行1万次计算,两个线程在单处理机上执行,运行时间是3分12秒,当然这其中包含了线程切换的微小开销.很明显,如果此并行化程序在多处理机系统上运行,我们估计的运行时间约为1.2分钟,程序运行效率比在单处理机上串行执行提高了60%左右.经实验证明,类似这样的例子,如果并行运行的线程越多,效率就提高得越大.这样,对于一些较复杂的计算,特别是对于那些计算量较大的科学计算,将其可并行的部分用线程的方法来实现并行计算,其效率是可想而知的.因此我们认为,此自动并行转换器的应用价值是很大的.

4 结束语

本文采用了"操作的可交换性检测技术[11]"来确定可并行的计算,突破了传统的只靠"单纯数据依赖分析"确定是否可并行的局限性,程序的并行程度肯定会有很大的提高.

我们对Java程序自动并行化的研究仍然在探索阶段.设计一个自动并行化系统是一件很复杂的工作,特别是基于多线程的自动并行化还有一些需要进一步解决的问题,如动态结合、别名分析等.我们的目标是设计一个高效、易移植、易使用以及有广泛使用价值的Java程序自动并行化系统.

References:

[1] The Java Language Specification. Sun Microsystems, Inc., 1996. http://java.sun.com:80/doc/language_specification/.

[2] The Java Language: a White Paper. Sun Microsystems, Inc., 1996. http://java.sun.com:80/doc/Overviews/java/.

[3] Li, Xin-ming, Li, Yi. Multi-Thread mechanism. Mini-Micro Systems, 1998,19(2):65~72 (in Chinese).
[4] Xiao, Gang, Xu, Ming, Zhou, Xing-ming. Multi-Threading architecture: current state and prospect. Computer Science, 1998,25(4):70~75 (in Chinese).
[5] Du, Jian-cheng, Chen, Dao-xu, Xie, Li. The state and advance of object-oriented parallel programming system. Computer Science, 1997,24(4):25~29 (in Chinese).
[6] Rinard, M. C., Diniz, P. C. Commutativity analysis: a new analysis technique for parallelizing compilers. ACM Transactions on Programming Languages and Systems, 1997,19(6):943~991.
[7] Diwan, A., Mckinley, K. S., Moss, J. E. B. Type-Based alias analysis. ACM Sigplan Notices, 1998,33(5):106~117.
[8] Liu Ying, Liu Lei. A practical alias analysis technique for Java program. Journal of Computer Research and Development, 2000,37(5):595~600 (in Chinese).
[9] King, J. Symbolic execution and program testing. Communications of the ACM., 1976,19(7):385~394.
[10] King, J. Program reduction using symbolic execution. ACM Sigplan Notice, 1981,6(1):9~14.
[11] Liu, Lei, Liu, Ying, Zhang, Xiao-dong. A test technique of commutativity for Java program. ACTA Scientiarum Naturalium Universitatis Jilinensis, 2000,(1):37~42 (in Chinese).

附中文参考文献:

[3] 李新明,李艺.多线程技术.小型微型计算机系统,1998,19(2):65~72.
[4] 肖刚,徐明,周兴铭.多线程体系结构现状及发展.计算机科学,1998,25(4):70~75.
[5] 杜建成,陈道蓄,谢立.OO并行编程系统的研究现状与发展.计算机科学,1997,24(4):25~29.
[8] 刘英,刘磊.面向Java的实用别名分析技术.计算机研究与发展,2000,37(5):595~600.
[11] 刘磊,刘英,张晓东.面向Java操作的可交换性检测技术.吉林大学自然科学学报,2000,(1):37~42.

Thread-Based Automatic Parallel Conversion Technique for Java Program*

LIU Ying[1], LIU Lei[2], ZHANG Nai-xiao[1]

[1](*Department of Informatics, School of Mathematical Sciences, Beijing University, Beijing* 100871, *China*);
[2](*Department of Computer Science, Jilin University, Changchun* 130023, *China*)
E-mail: liuying@water.pku.edu.cn; liulei@mail.jlu.edu.cn; naixiao@pku.edu.cn
http://www.pku.edu.cn; www.jlu.edu.cn

Abstract: The study of parallelism for Java program is one of the most important subjects at present. In this paper, a kind of automatic parallel conversion technique is given. The serial Java program is transformed to parallel program utilizing the multithreading machanism and testing technique of commutativity operations. The parallel program after transforming can run in the supercomputer with multi-CPU under the multi-processor operating system, which will enhance the programs' efficiency.

Key words: Java program; automatic parallelism; multi-threading; conflict-free operation; thread-effective method

* Received June 15, 1999; accepted January 13, 2000
Supported by the National Natural Science Foundation of China under Grant No. 69983001; the Chief Teacher Foundation of the Ministry of Education of China