JAVA 并行化编译器 JAPS-II *

于 勐[1,2], 陈贵海[1,2], 阳雪林[1,2], 谢 立[1,2], 过敏意[3]

[1](南京大学 计算机软件新技术国家重点实验室,江苏 南京 210093);
[2](南京大学 计算机科学与技术系,江苏 南京 210093);
[3](日本国会津大学计算机理工学部,国福导县 会津若松 日本)
E-mail: yumeng@nju.edu.cn
http://cs.nju.edu.cn/~yumeng

摘要: JAPS-II(Java automatic parallelizing system version 2)是一个 Java 源代码重构编译器,用来发现和实现串行 Java 程序中对象内和对象间的并行性.其目标体系结构为基于工作站网络环境的分布式存储器计算机系统.介绍了 JAPS-II 的体系结构和实现 JAPS-II 的关键技术,包括用于对象并行性分析的数据流分析技术、提高对象并行性和减少运行开销的优化技术以及类重构和代码生成技术.测试结果表明,JAPS-II 能够有效地发现循环中和对象内、对象间的并行性,获得加速比.这种技术也可应用于其他面向对象语言的并行化.

关 键 词: 并行编译;并行计算;面向对象语言;对象分布
中图法分类号: TP314 文献标识码: A

鉴于 Java 的可移植、面向对象和可以基于网络进行分布式计算等特性,越来越多的人考虑将 Java 用于高性能计算.

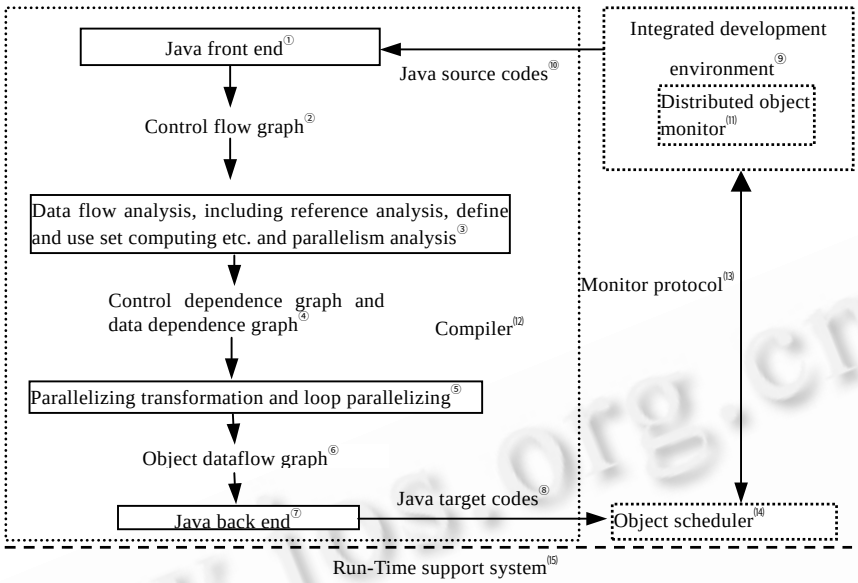
一方面,随着越来越多的 Java 编译器可以生成高效的目标程序代码,目标程序性能不再成为 Java 用于高性能计算的障碍.与此同时,在将传统的 Fortran 或 C 编写的函数库自动转换为 Java 程序方面,也有许多研究工作取得了进展.因此,我们相信 Java 是用于高性能计算的一个新的比较好的选择.另一方面,尽管 C 和 Fortran 等传统语言的自动并行化技术日趋成熟,但是由于面向对象语言的动态绑定、动态类型等特性,以及并行和并发对象执行模型还处于研究阶段,导致许多成熟的程序并行化技术无法直接应用于面向对象语言.将这些成熟技术应用到面向对象语言中,同时探索面向对象语言所特有的并行化技术,是我们设计 JAPS-II 的主要目标.本文将介绍 Java 自动并行化编译器 JAPS-II 的体系结构和关键的实现技术.

1 JAPS-II 的体系结构

JAPS-II 的体系结构如图 1 所示.JAPS-II 共有 4 部分组成:并行化编译器、可视化集成开发环境、支撑平台和调度器.编译器 Java 前端的结构如图 2 所示,它用来将串行 Java 程序转换成为 JAPS-II 的中间表示——控制流图(control flow graph,简称 CFG).

跨过程类分析的目的是分析程序中的对象变量在各个点可能的类型集合.由于面向对象语言的类的继承特性,这个集合中的元素一般不是惟一的.分析此类型集合,就可以确定对象方法调用所有可能的动态绑定.

* 收稿日期: 2000-08-03; 修改日期: 2000-12-04
基金项目: 国家自然科学基金资助项目(69803005);国家 863 高科技发展计划资助项目(863-306-ZT02-03-01)
作者简介: 于勐(1972－),男,辽宁鞍山人,博士,讲师,主要研究领域为分布式并行计算,软件工程;陈贵海(1963－),男,江苏盐城人,博士,教授,博士生导师,主要研究领域为并行计算机体系结构,并行计算;阳雪林(1976－),男,广西临桂人,博士生,讲师,主要研究领域为并行编译,并行计算;谢立(1942－),男,江苏常熟人,博士,教授,博士生导师,主要研究领域为分布式,并行计算;过敏意(1962－),男,江苏无锡人,博士,教授,主要研究领域为并行处理,并行化编译,软件工程.

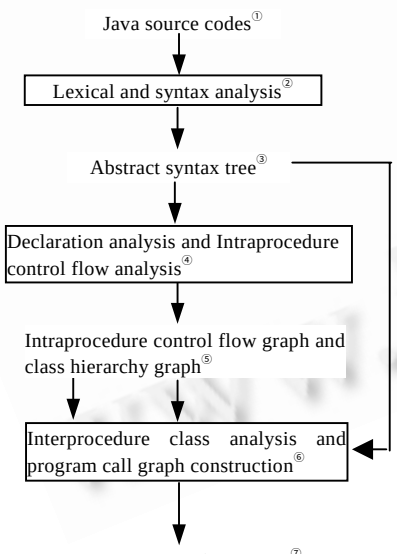


①Java 前端,②控制流图,③数据流分析,包括引用分析,定义和使用集合计算等以及并行性分析,④控制依赖图和数据依赖图,⑤并行转换和循环并行化,⑥对象数据流图,⑦Java 后端,⑧Java 目标代码,⑨集成开发环境,⑩Java 源代码,⑪分布式对象监视器,⑫编译器,⑬监视器协议,⑭对象调度器,⑮运行支持系统.

Fig.1 The architecture of JAPS-II

图 1 JAPS-II 体系结构

数据流分析基于建立的程序控制流程图进行.首先,分析过程和方法调用的部分传递函数(partial transfer function,简称 PTF)[1]进行跨过程的对象引用分析,即计算每个对象引用点可能引用的对象集合.然后,进行定义、使用集合计算、数组引用区域分析、常量传播等常规的数据流分析和优化.最后,根据数据流分析的结果,进行并行性分析,建立程序的数据依赖图(data dependence graph,简称 DDG)和控制依赖图(control dependence graph,简称 CDG).



①Java 源代码,②词法和语法分析,③抽象语法树,④说明部分分析和过程内控制流分析,⑤过程内控制流图和类层次图,⑥跨过程类分析和程序调用图构造,⑦程序控制流图.

Fig.2 The architecture of JAPS-II's front end

图 2 JAPS-II 的前端结构

并行化变换部分根据 DDG 和 CDG 中得到的并行性建立反映对象内和对象间并行性的对象数据流图(object dataflow graph,简称 ODG,).对象数据流图是一种聚类任务图.图的每个结点是对应与 CFG 中的结点,有向边对应于结点之间的数据流向,属于同一个对象的结点构成初始的任务聚类,我们称之为对象聚类.

此时还要对循环进行专门的并行化.循环并行化的任务是根据传统的循环并行化,技术对各个循环结点进行并行化并进行数据分布,将循环结点分裂为对象数据流图中多个并行执行的结点,原来结点的进入和射出的数据流边也要根据数据分布进行相应的分解.JAPS-II 的循环并行化和数据分布采用了过敏意提出的算法[2].

编译器的后端进行与目标体系结构和实现相关的优化以及代码生成工作.JAPS-II 的目标体系结构是分布式存储器系统,因此要进行任务和数据的分布,也就是将对象数据流图中的对象聚类进行再聚类,映射到可用的处理器上,并尽可能地使通信最少,并行度最大.与实现相关的优化包括代码内嵌,方法调用本地化和提前数据发送.代码生

成部分重新生成并行类说明和执行代码,生成 RMI(remote method invocation)远程方法调用接口,并根据数据流分析、常量传播结果生成对象顺序化(serialize)代码.

在支撑平台中,我们实现了对 Java RMI 和对象并行计算模型的支持.由于线程比任务具有更小的调度开销,在 JAPS-II 中我们实现了支撑平台对线程的支持,相应地,JAPS-II 的代码生成部分在可能时将把并行任务包装为线程.

可视化集成开发环境由编辑器和一组内部数据浏览器组成,用于帮助观察源程序的并行性.我们还设计了一个动态的任务监视器以观察目标程序的执行.任务的动态监视是通过在监视器和调度器之间设计协议接口实现的.通过监视器,用户可以收集目标程序的运行数据,为改进源程序的并行性提供参考.

2 关键技术

本节将介绍 JAPS-II 中实现的对象引用分析、对象并行性分析、实现对象并行性的优化、对象分布和重构类代码等关键技术.

2.1 对象引用分析

别名分析和准确的依赖关系分析都依赖于精确的对象引用分析.对象引用本质上是指向对象的指针[3].已经有许多成功的指针分析算法,例如,"points-to"分析[3]和基于 PTF 的分析技术.这些技术主要应用于 C 语言,并设计了相应的低级存储表示,我们将现有技术推广到更高的抽象级别以便它们应用于对象引用分析[1].

我们用 p,q,r 表示引用标量.引用标量是指不含有下标的简单对象变量或对象域.我们用$(p,0)(i)$,$(q,0)(i,j,k)$和$(r,id)(i,j,...,k)$表示 $p[i]$,$q[i].f[j].f[k]$和带有形如 $r.f[i].f[j]. \ldots .f[k].f$ 的下标表达式的引用向量.如果我们不能确定引用向量的下标表达式的值,我们就用区间表示引用向量.表 1 是对象引用表示的例子.

Table 1 Examples of object reference representation

表 1 对象引用表示示例

Expression①	Internal representation②	Description③
Obj	Obj	Simple reference variable④
obj.ref	obj.ref	Single object field⑤
obj[*i*]	(obj,0)(*i*)	One dimension reference vector⑥
obj.*f*[*i*]	(obj,1)(*i*)	id is used to distinguish different fields of different objects. In this example, id is 1⑦
obj[*i*].*f*[*j*].ref	(obj,0)(*I*,*j*)	Multiple dimension vector representation⑧
for (*i*=1;*I*<10;*i*++) obj[*i*]= ...	(obj,0)([1,10])	Region representation of vectors⑨

①表达式,②内部表示,③说明,④简单引用变量,⑤一个对象域,⑥一维引用向量,⑦id 用来区分同名变量的不同域,本例中 id 为 1,⑧多维向量表示,⑨向量的区域表示.

现在我们给出一些引用相关的定义.我们推广指向(points-to)函数的定义到引用(reference-of)函数进行对象引用分析.

假定 R 是引用变量集合,O 是程序中定义的对象集合,则有如下定义.

定义 1. 若函数 f 定义了从 R 到 O 的一个映射,那么我们称 f 为一个引用函数.若映射 f 对应于程序语句 S 的语义,则我们称 f 是语句 S 的引用函数.

例 1:若 f 是一个引用函数,$f(p)=\varnothing$ 表示引用变量 p 未被初始化. $f(p)=\{object\}$ 表示 p 引用对象 $object$.$f(p)=\{object_1,object_2\}$则表示引用 p 可能引用对象 $object_1$ 或 $object_2$.

定义 **2**. 若 f,g 是引用函数,则 $f \vee g(p)=\begin{cases} g(p) & \text{若} f(p) \text{无定义,} \\ S_1 \cup S_2 & \text{若} f(p)=S_1, g(P)=S_2, \\ f(p) & \text{若} g(p) \text{无定义.} \end{cases}$

定义 **3**. 若 f,g 是引用函数,则 $f \wedge g(p)=\begin{cases} S & \text{若} g(p)=S, \\ f(p) & \text{若} g(p) \text{无定义} \end{cases}$

注意运算∧是不可交换的.

定理 **1**. 3 种基本结构的引用函数计算.

若顺序结构中各语句的引用函数分别为 $f_1,f_2,...,f_n$,则该结构的引用函数为 $f_1 \wedge f_2 \wedge ... \wedge f_n$.

若分支结构中各语句的引用函数分别为 $f_1,f_2,...,f_n$,则该结构的引用函数为 $f_1 \vee f_2 \vee ... \vee f_n$.

若循环结构的引用函数为 f,到达循环前的点的引用函数为 g,则该结构的引用函数为 $g \vee^* f$.其中 $\vee^*$ 是关系 $\vee$ 的闭包.

定理 1 可由引用函数的定义 1 直接证明.

定义 **4**. 在过程的入口点,根据参数调用模式,为引用参数建立指向的对象,这些对象并不对应于程序运行中的对象实例,称为扩充参数[1].

扩充参数主要用来根据过程的参数调用模式来分析程序的行为.在程序的入口,用扩充参数到对象的映射来区别不同的调用别名情况.这样,就不用记录所有的调用实参和形参的结合.参考过程式语言的 PTF 函数定义,我们来定义面向对象语言的 PTF 函数.

定义 **5**. 若 f 定义了从过程入口到过程出口的引用函数的部分映射,并且这些引用函数定义的是扩充参数到对象的映射,则我们称映射 f 为部分传递函数.

定义 **6**. 我们定义 PTF 的合并操作 $\vee$ 为

$$f_1 \vee f_2(rof)=f_1(rof) \vee f_2(rof).$$

这里,rof 是一个引用函数.

基于以上定义和定理,我们给出一个 PTF 函数应用于面向对象语言的基本推广.程序中的动态调用点,是指程序中需要运行时刻动态绑定调用方法的调用点.由于大多数情况,面向对象程序的方法调用无法全部进行静态绑定,因此必须考虑动态调用点的处理.我们对 PTF 技术的推广,主要基于下面的定理.

定理 **2**. 在一个动态调用点,若所有可能的被调用者的 PTF 函数分别为 $f_1,f_2,...,f_n$,则此动态调用点的 PTF 函数为 $f_1 \vee f_2 \vee ... \vee f_n$.

证明:若到达此调用点的引用函数为 g,则所有可能的方法调用构成一个分支结构.按照定义 5 中 PTF 函数的定义,此结构中各个分支的引用函数为 $f_1(g),f_2(g),...,f_n(g)$.根据定理 1,此分支结构的引用函数为 $f_1(g) \vee f_2(g) \vee ... \vee f_n(g)$.再由定义 6,经过此调用点的引用函数为 $f_1 \vee f_2 \vee ... \vee f_n(g)$. □

定理 2 说明在动态调用点,我们可以通过分析所有可能调用的方法的 PTF 函数来构造经过该调用点的 PTF 函数.由于仅当调用参数模式没有分析过时,才有必要分析 PTF 函数,所以该算法的效率是比较高的.

基于前面的结果,采用标准的跨过程数据流分析的迭代解法就可以得到对象引用分析的结果.对于递归程序,此分析总会到达一个固定点.

与 R.P.Wilson 以及 M.Emami 的工作[1,3]相比,我们的表示更适合具有强类型系统的语言.

2.2 对象并行性分析

$f()$和 $g()$是对象 Obj 中的两个方法调用,如果 $f()$是可以并行的,我们就称这种并行性为对象的方法内并行性;如果 $f()$和 $g()$是可以并行执行的,我们称这种并行性为对象的方法调用间并行性.对象的方法内并行性和方法调用间并行性都属于对象内并行性.如果 $f()$和 $g()$是属于不同对象的两个方法调用,则 $f()$和 $g()$调用间的并行性称为对象间的并行性.

不失一般性,我们将程序中任意出现的下面形式的两个对象的方法调用作为方法调用模型,并考察它们的并行性.

$$S_1: r_{11}=O_1.f_{11}(p_{111},p_{112},...,p_{11m}); \quad S_2: r_{21}=O_2.f_{21}(p_{211},p_{212},...,p_{21n}).$$

在本文中我们使用下列记号:

$O_1,O_2,...,O_i,...$表示程序中的对象引用变量.$f_{i1},f_{i2},...,f_{ik},...$表示对象 O_i 中的方法.$p_{ik1},p_{ik2},...,p_{ikn},...$表示对象 O_i 的方法 f_k 的参数.$r_{i1},r_{i2},...,r_{ik},...$表示方法 $f_{i1},f_{i2},...,f_{ik},...$的返回值.$class(O_i)$表示返回 O_i 可能的类型(类)集合.$ref(O_i)$表示返回变量 O_i 可能引用的对象集合.$D(S)$表示语句 S 中的定义变量集合(define set).$U(S)$表示语句 S 中的引用变量集合(use set).$mor(O.f)$动态调用点 $O.f$ 处的候选方法集合.

请注意,$r_{i1},r_{i2},...,r_{ik}$ 和 $p_{ik1},p_{ik2},...,p_{ikn}$ 都可能是对象引用变量.

我们知道,两个操作存在数据依赖关系当且仅当两个操作存取同一存储区域并且至少有一个操作是写操作.语句 S_1 和 S_2 可以并行执行当且仅当下面的条件(1)成立.

$$(D(S_1)\cap U(S_2)=\varnothing)\wedge(D(S_2)\cap U(S_1)=\varnothing)\wedge(D(S_1)\cap D(S_2)=\varnothing). \quad (1)$$

要分析对象方法调用间的并行性,就要通过数据流分析计算出方法调用点存取的全部环境变量,包括静态的、动态的以及通过引用存取的全部变量,即计算出上述的定义和使用集合.这也可以通过跨过程的数据流分析技术来实现.

下面给出的是考虑动态调用点的跨过程数据流分析算法.

算法 **1**. 跨过程数据流分析

```
输入:程序控制流图 G.
输出:所有基本结点和过程的定义集合和使用集合.
push(stack, main);
createnv(main);                          /*createnv(proc)建立进程 proc 的分析上下文环境*/
while not empty(stack)                   /*按后序遍历*/{
    currentproc=top(stack);
    if (There is no call node in currentproc)
        if (There is a currentproc in stack under stack top)   /*检测到递归*/
            apply the PTF of currentproc in stack under stack top;
        else if (There is a PTF avaliable)
            match parameter parttern and use PTF mapping summary information;
        else                             /*分析 proc 生成 PTF*/
            regist parameter pattern;
            compute define and use set of currentproc;
            regist PTF mapping;
        endif;
        pop(stack);
        createnv(top(stack));
    else
        for (call node N1,N2,...,Nk in currentproc)
            push(mor(Nk));push(mor(Nk–1));...;push(mor(N1));
            createnv(N1);
        endfor
    endif;
endwhile;
```

在跨过程分析算法中,定义和使用集合的计算是通过对 CFG 按照程序调用树的结构后序遍历计算得到的,对过程执行环境的分析则要通过对过程内的 CFG 按照逆后序(reverse postorder)遍历得到,我们将跨过程的信息收集和环境传播结合到统一的算法中,并在堆栈中对递归执行进行了检测和处理.

前面的跨过程类分析和对象引用分析,已经计算出了 $class(O_i)$和 $rof(O_i)$.其中$|class(O_i)|>1$ 导致动态绑定,$rof(O_i)\cap rof(O_j)\neq\varnothing, i\neq j$ 导致引用别名.在定义和使用变量集合比较时,简单地进行变量链的符号名称比较[4]是不充分的.数据依赖分析依赖于对存储位置访问的分析.确定两个化简的变量表达式是否代表相同的内存单元请参阅下面的算法.

算法 **2**. 访问单元比较

```
输入:O1v11v12...v1k 和 O2v21v22...v2k.
输出:两个输入变量是否会表示同一个存储单元.
if (ref(O1)∩ref(O2)=∅
    return false;
else
```

for i=1,k
if (v_{1i}≠v_{2i})
return false;
endfor
if ($O_1v_{11}v_{12}...v_{1k}$ **and** $O_2v_{21}v_{22}...v_{2k}$ are references)
and ($ref(O_1v_{11}v_{12}...v_{1k})\cap ref(O_2v_{21}v_{22}...v_{2k})=\varnothing$)
return false;
endif
return possible.

2.3 优 化

在建立程序的并行结构,即建立了 ODG 之后,首先要进行各个结点计算代价和各个边的通信代价的评估,然后进行下面的优化工作.

2.3.1 对象分布

对象分布是指将对象分布到系统中的工作站上,并获得尽可能大的并行度和尽可能小的通信开销.对象分布问题是一个图分布问题,该问题已知是 NP 难的.我们提出了一种启发式算法来分布对象.

我们根据 ODG 来分布对象,并且通过 ODG 的划分来平衡工作站之间的计算和通信负载.这将在下面的算法中给出.

对于一个 ODG $G<V,E>$和一个工作站集合 WS,假定 $e\in E$ 的权重为 $CommCost(e)$,$v\in V$ 的计算开销为 $CompCost(v)$,我们用等式(2)来衡量工作站的负载和通信平衡状况.

$$\sum_{e\in E}CommCost(e)\times\sum_{v\in V}\sqrt{(CompCost(v)-\frac{\sum_{v\in V}CommCost(v)}{|WS|})^2}. \quad (2)$$

对象分布算法如下,算法中的对象是指 ODG 中一个聚类包含的所有结点和数据.

算法 3. 对象分布

(1) 对循环中的方法调用进行聚类.如果一个循环中包含有另外一个对象的方法调用,并且该循环存在迭代间依赖关系,则将这两个对象聚类合并为一个对象聚类.

(2) 分配递归循环中的对象.如果在源程序中存在递归调用,则在 ODG 中将出现一个环.这里,我们将所有属于同一递归环的对象聚类合并成为一个新的对象聚类,选择具有最小计算负荷的工作站 $ws\in WS$ 分配该聚类中的对象.更新 ws 的计算负荷,并将 $objs$ 从 V 中删除.

(3) 分配并行对象.假定 C 是一个可并行执行的对象集合.如果$|C|<|WS|$,则将 C 分配到 WS,每个工作站不超过一个繁衍对象.更新 $ws\in WS$ 的计算负荷.否则,我们为每个工作站分配一个对象.对于每个剩下的对象 $obj\in C$,分配 obj 到具有最小计算负荷的工作站 $ws\in WS$,使等式(2)能够获得最小值,重复此步骤直到 $C=\varnothing$为止.最后,更新每个 $ws\in WS$ 的计算负荷.

(4) 分配剩余的对象聚类.对于每一个剩余的对象聚类中的对象 obj 分配到 $ws\in WS$,使等式(2)能够获得最小值.重复此步骤直到 $V=\varnothing$为止.最后,更新每一个 $ws\in WS$ 的计算负荷.

2.3.2 代码内嵌和方法调用本地化

完成对象分布后,对于分布在同一个处理器的比较小的对象方法调用,进行内嵌处理,减小调用开销.对于出现在并行循环中的远程方法调用,进行方法调用的本地化处理.例如,对象 O_1 的方法调用 $O_1.f()$中,在循环中调用不在同一个处理器上的对象 O_2 的方法 $O_2.g()$,当不存在迭代间的循环依赖时,可以将 $O_2.g()$在 O_2 所在的处理器完成,最后汇集调用结果进行一次传输,从而减少远程调用开销.这两种优化的典型例子是 Matrix 类中的乘法代码片段.

例 2:

```
for (int i=0;i<mat1.rows();i++)
    for (int j=0;j<mat2.columns();j++){
    int tmp=0;
    for (int k=0;k<columns;k++)
```

```
        tmp=tmp+mat1.getxy(i,k)*mat2.getxy(k,j);
    matrix[i][j]=tmp;
    };
```

其中 getxy(*x*,*y*)没有迭代间的依赖关系,首先可以将两个对象 *mat*1 和 *mat*2 的 getxy(*x*,*y*)调用本地化,由于 getxy(*x*,*y*)只是返回矩阵 *x* 行 *y* 列的代码,又可以将该部分的实现代码内嵌,从而最终得到高效的对象间并行代码.

2.3.3 提前数据发送

这种情况发生在后继的任务所需要的数据已经计算完成时,可以将数据发送给后继任务,再进行后面的计算.当远程方法调用发生时,在可能的情况下,也可以发送部分返回结果,使通信和后继计算并行进行.

2.4 类重构和代码生成

这部分工作是根据并行变换的结果,构造出符合 Java 语言语法规范的源程序.

在 Java 的 RMI 规范中,只有可以顺序化的对象才可以作为参数和返回值传输.大部分 Java 类库中的类对象都是可以自动顺序化的.对于用户自定义类的对象,由 JAPS-II 编译器来生成顺序化代码.编译器在生成顺序化代码时,需要根据前期数据流分析和常量传播的分析结果,由编译器按照各种数据结构的顺序化代码模板生成用户自定义类的顺序化代码.

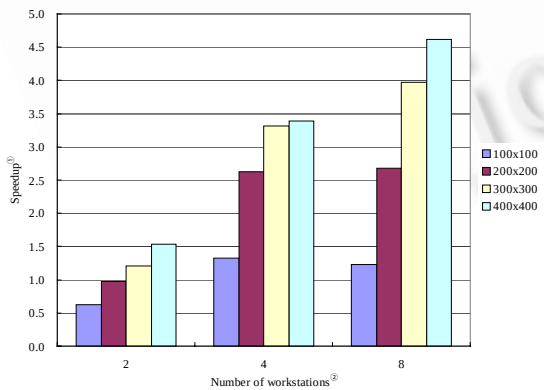
在根据 ODG 中的结点生成相应的类说明和生成远程调用的 RMI 接口之后,还要在各个方法调用的代码中插入适当的通信,最终完成代码生成工作.这部分工作也被称为类重构.

3 测试结果和分析

对 JAPS-II 的测试分为两个方面,一方面对 JAPS-II 并行化面向对象风格的程序的效果进行测试,另一方面,需要对 JAPS-II 和我们的早期工作 JAPS[4]进行比较.

实验采用 SUN SPARC Ultra-30,IBM RS/6000,100M ATM 组成的工作站网络作为分布式试验环境.我们选择 OOPack[5]的 OO Style 部分,用 Java 语言重新编码作为 Benchmark 程序进行测试.OOPack Kernel 中包含 Max,Matrix,Iterate,Complex 这 4 个部分,既可测试对象内的并行性,又可测试对象间的并行性和代码优化.同时,我们也选择了 C,Fortran 并行化编译器所广泛采用的 Linpack 程序的 Java 语言版本对两个系统进行了测试.

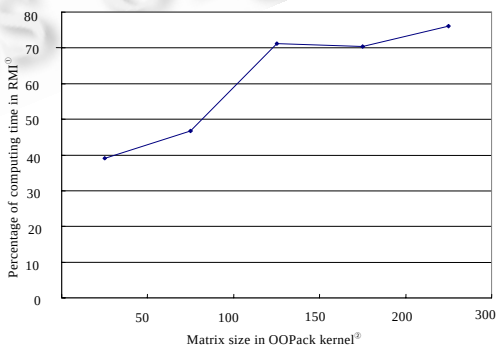
我们设定 OOPack Kernel 中的矩阵为不同大小,按不同的目标机器数量并行化,测试结果如图 3 所示.通过对测试结果的分析可知,矩阵小导致加速比小的原因在于对象的顺序化和 RMI 的开销导致有效计算时间在总计算时间中只有很小的百分比,如图 4 所示.



①加速比,②工作站数量.

Fig.3 Test results of OOPack kernel

图 3 OOPack Kernel 的测试结果



①RMI 调用中计算时间所占的百分比,②OOPack Kernel 中的矩阵大小.

Fig.4 Effective computing cost of target codes

图 4 目标代码的有效计算开销

JAPS 缺乏循环并行化能力,不能对此测试程序并行化.随后,我们用 Java 编写的 Linpack 程序对 JAPS-II 和 JAPS 进行了对比测试,测试的结果与 JAPS 接近.由于 JavaLinpack 是非面向对象风格的程序,其整个算法是在 Linpack 类的一个方法中实现的,因而无法测试对象并行性.

4 相关工作比较

在过去的几十年中,在设计显式的并行编程语言和编程语言的自动并行化领域进行了大量的研究工作.我们将把 JAPS-II 与这些工作以及我们的早期工作分别进行比较.

4.1 并行面向对象语言

Mentat[6,7],HPC++[8],pC++[9]和 CC++[10]都是从 C++繁衍出来的,并且它们的并行计算模型都是基于对象的.例如,Mentat 中特别定义的 Mentat 对象以及 HPC++中的 Collection 等.这些语言所编写的程序的并行性都依赖于语言所提供的特别并行设施.同时,程序的并行性也需要程序员来发现并且用适当的语言设施来表达.

4.2 自动并行化编译器

Stanford 的 SUIF[11],UIUC 的 Polaris[12]及早期的 PTRAN[13],PFC[14],PTOOL[15],PED[16]等编译器都是比较成功的自动并行化编译器,与 JAPS-II 的不同之处在于:这些编译器编译的都是非面向对象的语言,所以不能处理面向对象的语言.

由 Indiana 大学开发的 Javar[17]能够实现 Java 程序中循环的并行化,并能够利用 Java 中的线程机制实现并行.但是,由于指出并行性仍然是程序员的责任,所以该编译器准确的说是半自动化的.这两个自动并行化编译器的研究内容和我们的工作接近,主要差别在于,他们的工作是以共享存储器系统作为目标体系结构,而 JAPS-II 的目标体系结构主要是分布式存储器系统.

4.3 JAPS

我们在早期的工作中开发了一个原型化的系统——Java 自动并行化系统 JAPS(Java automatic parallelizing system)[4],其目标程序采用了 MPMD(multiple programs multiple data flow)模型,能够发现 Java 程序中的任务并行性.与 JAPS 相比,JAPS-II,在以下几个方面进行了关键性的改进:

- 改进了编译器体系结构和中间数据结构,便于将来编译其他面向对象语言,并易于插入新的优化算法.
- 增加了对象引用分析,改进了对象并行性分析算法,使并行性分析更为准确.
- 采用了基于对象的计算模型,与大多数的显式面向对象的并行语言一致.
- 增加了循环并行化、循环数据分布[2]和对象分布.
- 解决了任务的次并行性[4]问题.由于采用了对象数据流图作为程序并行结构的中间表示,在并行变换阶段,编译器能够彻底发现各个层次的任务并行性.在后端的优化阶段,进行了提前数据发送优化,从而进一步提高了目标代码的并行性.
- 并行任务采用线程实现.代码生成部分和支撑环境都增加了对线程的支持,减小了目标代码的启动、调度和运行开销.后端增加了两种优化技术.

5 结论和进一步的工作

JAPS-II 能够有效地发现顺序 Java 程序的对象内和对象间的并行性,并通过并行变换和类重构,将对象分布到工作站上使它们并行执行.由于前端接受的 Java 语法和 Java 1.0 规范一致,从而能够直接处理较大规模的真实 Java 程序.我们将在今后的工作中改进对象分布和类重构算法,并给出更多的测试结果.

References:

[1] Wilson, R.P., Lam, M.S. Efficient context-sensitive pointer analysis for c program. In: Kurihara, K., Chaiken, D., Heinlein, J., eds. Proceedings of the ACM SIGPLAN'95. La Jolla, CA: Springer-Verlag, 1995. 1~12.

[2] Guo, Min-yi. Efficient techniques for data distribution and redistribution in parallelizing compilers [Ph.D. Thesis]. University of Tsukuba, 1998.

[3] Emami, M., Ghiya, R., Hendren, L.J. Context-Sensitive interprocedural points-to analysis in the presence of function pointers. In: Culler, D.E., Jaswinder, P.S., Anoop, G., eds. Proceedings of the ACM SIGPLAN. Oriando, Florida: Springer-Verlag, 1994. 242~256.

[4] Du, Jian-cheng, Chen, Dao-xu, Xie, Li. Japs: an automatic parallelizing system based on java. Science in China, 1999,42(4):396~406 (in Chinese).
[5] The oopack kernels. 1998. http://www.kai.com/oopack/oopack.html
[6] Grimshaw, A.S. Easy-to-Use object-oriented parallel processing with Mentat. IEEE Computer, 1993,26(5):39~51.
[7] Grimshaw, A.S., Weissman, J.B., Strayer, W.T. Portable run-time support for dynamic object-oriented parallel processing. ACM Transactions on Computer Systems, 1996,14(2):139~170.
[8] Diwan, S., Johnson, E., Gannon, D. HPC++ and the Europa call reification model. ACM Applied Computing Review, 1996,4(1):36~69.
[9] Gannon, D., Yang, S.X., Beckman, P. User guide for a portable parallel C++ programming system: pC++. Technical Report, Indiana University, 1994.
[10] The CC++ language definition. 1994. http://globus.isi.edu/ccpp/.
[11] Ramaswamy, S. A framework for exploiting task and data parallelism on distributed memory multicomputers. IEEE Transactions on Parallel and Distributed Systems, 1997,11(8):365~382.
[12] Blume, W. Advanced program restructuring for high-performance computers with Polaris. Technical Report, University of Illinois at Urbana-Champaign, Center for Supercomputing Research & Development, 1996.
[13] Cytron, R., Ferrante, J., Sarkar, V. Experiences Using Control Dependence in PTRAN. Cambridge: The MIT Press, 1990.
[14] Havlak, P., Kennedy, K. An implementation of interprocedural bounded regular section analysis. IEEE Transactions on Parallel and Distributed System, 1991,2(3):351~354.
[15] Allen, J., Baumgartner, D., Kennedy, K., *et al*. PTOOL: a semi-automatic parallel programming assistant. In: Dowek, G., Felty, A., Herbelin, H., eds. Proceedings of the 1986 International Conference on Parallel Processing. Cambridge: Prentice-Hall, 1986. 139~152.
[16] Girker, M.B. Functional parallelism theoretical foundations and implementations. Technical Report, University of Illinois at Urbana-Champaign, 1991.
[17] Bik, A.J.C., Gannon, D.B. Automatic exploiting implicit parallelism in java. Concurrency, Practice and Experience, 1997,9(6):576~619.

附中文参考文献:
[4] 杜建成,陈道蓄,谢立.基于 Java 的自动并行化系统 Japs.中国科学,1999,42(4):396~406.

JAPS-II: a Parallelizing Compiler for Java*

YU Meng[1,2], CHEN Gui-hai[1,2], YANG Xue-lin[1,2], XIE Li[1,2], GUO Min-yi[3]

[1](*State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing* 210093, *China*);
[2](*Department of Computer Science and Technology, Nanjing University, Nanjing* 210093, *China*);
[3](*School of Computer Science and Engineering, Wakamatsu University, Aizu-Wakamatsu City, Fukushima, Japan*)
E-mail: yumeng@nju.edu.cn
http://cs.nju.edu.cn/~yumeng

Abstract: JAPS-II (Java automatic parallelizing system version 2) is a Java parallelizing compiler that exploits and implements intra and interobject parallelism of serial Java programs. Its target architecture is NOW based on distributed memory computer system. In this paper, the infrastructure of JAPS-II and the key techniques to implement JAPS-II which include dataflow analysis for analyzing object parallelism, optimization for improving object parallelism and reducing executing cost, and techniques of class restructure and code generation are introduced. Related experiment results are also included.
Key words: parallelizing compiler; parallel computing; object-oriented language; object distribution

* Received August 3, 2000; accepted December 4, 2000
Supported by the National Natural Science Foundation of China under Grant No.69803005; the National High Technology Development 863 Program of China under Grant No.863-306-ZT02-03-01