JAPS 任务调度协议 并行编译系统

计算机科学 2000Vol. 27№. 12

JAPS 中任务调度协议的设计与实现*)

Design and Implementation of Task Scheduling Protocol in JAPS

李晓明 陈道蓄 谢 立
(南京大学计算机软件新技术国家重点实验室 计算机科学与技术系 南京 210093)

Abstract The automatic parallelizing is mainly to exploit the parallelism hidden in programs and run them in parallel without changing its origin semantic. The runtime task scheduling protocol is important to the parallel computing based on the distributed-memory systems. In this paper, we design and implement a task scheduling protocol for the Automatic Parallelizing System based on JAVA(JAPS). It can satisfy the dependence requirements while reducing the total communication load of the system.

Keywords Task scheduling protocol, Dependence relation, Task scheduling

1 背景和系统概述

在并行编译系统中，主要的目标是分析程序中的依赖关系。通过依赖分析得到的结果将源程序划分成可以并行执行的部分，然后在一定的运行时支撑平台上并行执行。并行执行的基本要求是程序的并行语义和串行语义必须一致。

在以 NOW 基础的并行计算系统中，由于通信开销较大，一般需要发掘在任务层次上的并行性。相对数据并行性来说，开发任务并行性更加繁杂。一般来说，数据并行性是一种比较规则的小粒度的并行性，而任务并行性是一种不大规则的大粒度的并行性。目前很多并行编译系统都是偏重于开发数据并行性，对任务并行性的支持不多。在一些结构复杂、计算量很大的程序中，存在着不少的任务并行性。开发出这类并行性对于缩短整个程序的执行时间有很大的帮助。特别是对于 Java 语言，由于它是一种解释性语言，本身执行效率偏低，因此只有开发任务级的并行性才可能获得较好的加速效果。在任务级的运行环境中，依赖关系的维护一般都需要运行时支撑环境提供。在分布式计算环境下的并行任务执行和共享存储环境下的并行任务执行有较大的不同。在共享存储环境下，任务之间的同步可以利用 semaphore、critical section 等机制来实现；而在分布式计算环境中，由于任务在不同的节点上执行，因此必须使用调度协议来保证任务间的同步。考虑到网络通信开销，任务调度协议的设计对并行任务的执行速度有着很大的影响。

目前基于 NOW 或广域网的并行计算已进行了较多的工作。MIT 的 Sarmenta 等人[2]提出在 Internet 这样的广域网中进行"志愿超级计算"(Volunteer Supercomputing)，它的各个任务被包装成 Applet，通过一个中间协调者保证依赖关系。因此它的调度协议和 NOW 环境下的会有很大的不同。Caltech 的 Brunett 等人[3]提出基于 Java 的多线程超级计算，并且也支持 NOW 环境，但要求运行时环境为共享内存系统。在共享内存系统中依赖关系的控制相对分布内存系统要简单，因此它的任务调度协议对通信因素考虑得不多。U. C. Berkerley 的 Ptolemy 项目[4]提出较好的依赖控制技术，但它的重点是在控制的可靠性，需要较大负载，因此不适于并行计算中对速度的要求。

为了研究基于 Java 这样的面向对象语言的自动并行编译技术，我们提出并实现了一个基于 NOW 的 Java 自动并行编译系统 JAPS(Java Automatic Parallelkizing System)[1]。JAPS 提供了一组编译工具，将串行的 Java 源程序进行预处理，通过分析转换，生成并行 Java 代码，然后在自己开发的基于 NOW 的分布式存储环境下运行。

图 1 给出了 JAPS 的总体模型。在该模型中，既可以进行传统的数据并行性的分析，也可以进行任务并行性的分析。

*)本文所涉及的项目 JAPS 属于国家攀登计划 B 的课题"大规模并行编译和操作系统的某些关键技术"的一项研究内容，同时受到了"863"课题"基于 JAVA 的使用化分布并行工具包"的支持。李晓明 硕士，主要研究方向：并行和分布式系统，计算机网络。陈道蓄 教授，博士生导师，主要研究方向：并行和分布式系统。谢 立 教授，博士生导师，主要研究方向：并行和分布式系统，智能操作系统。

本文工作就是在以上的背景下进行的。主要包括设计并实现一个在分布式计算环境中的任务调度协议。首先给出调度协议必须满足的依赖关系条件和JAPS中任务调度的要求；然后详细讨论协议中的两个关键问题并给出协议；最后给出实验结果并给出结论。

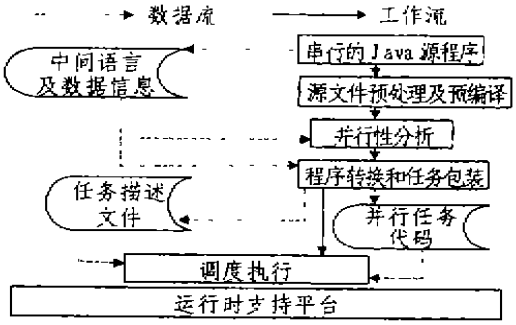

图1

2 JAPS中的任务调度

在并行编译中，核心问题就是分析阶段发掘程序中的依赖关系，并在运行中根据依赖关系保证程序执行的正确性[5]。主要考虑的有两种依赖关系：控制依赖和数据依赖。

控制依赖关系描述控制流程图(CFG)中的节点执行时必须满足的顺序关系。首先给出控制依赖的形式定义：

定义1 后控制关系(Post-dominate)Δ_p：在控制流图中的两个节点x、y，$y\Delta_p x \leftrightarrow$在CFG中，x到结束节点的所有路径都经过y。

定义2 控制依赖(Control Dependence)δ_c：在控制流图中的两个节点x、y，$x\delta_c y \leftrightarrow (y\ \Delta_p x) \wedge$(存在非空路径$p=(x,a,\cdots,y), \forall z: z\in p \wedge z\neq x, y \rightarrow y\Delta_p z$))

从直观上解释，如果一个节点$A\delta_c B$，则A是程序中的一个分支节点，如if，switch语句等；而B是在A的某个分支上。从运行时执行条件分析，为了保证程序并行执行时语义的正确性，控制依赖关系必须被维持。即如果要执行B，必须在A执行结束以后，且A节点的各分支中B对应的分支条件计算为真。

数据依赖关系描述在数据流程图中，各节点由于对数据进行了读写，为了保证程序语义的正确性而在执行时必须满足的先后关系。数据依赖分析的基础是确定节点的数据读写集。一般来说，程序流图中的一个节点N访问的数据可分为如下两种基本集合：

(1)写集记为def(N)：对于一个节点N及一个变量v：$v\in def(N)$表示执行N中改变了v的值。

(2)读集记为use(N)：对于一个节点N及一个变量v：$v\in use(N)$表示执行N中使用了v的值。

根据读写集，就可以得到节点间的数据依赖关系δ[3]。给定两个节点S1、S2，且S2从S1控制流可达。一般考虑下面三种数据依赖关系：

流依赖(δ_f)：节点S1和S2之间存在流依赖的条件是：

$def(S1)\cap use(S2)\neq\varnothing$；

反向依赖(δ_a)：S1和S2之间存在反向依赖的条件是：

$use(S1)\cap def(S2)\neq\varnothing$；

输出依赖(δ_o)：S1和S2之间存在输出依赖的条件是：

$def(S1)\cap def(S2)\neq\varnothing$；

在JAPS中，并行执行的平台是一个分布式存储系统(Distributed-Memory System)。该平台通过存储拷贝的方式消除了反向依赖和输出依赖的影响。因此在我们的工作中只考虑流依赖，这也是一种无法通过其他技术完全消除的依赖关系。为了保证程序并行执行和串行执行的语义相同，必须维持程序的数据依赖关系。如果两个节点$A\delta_f B$，则B必须在A执行结束后才能执行，且A可能向B传送部分数据以使系统状态同步。

JAPS中的任务调度是根据前端分析得到的层次任务图HTG[1-5]，在程序运行时刻接收前导任务发来的消息，将下一层的所有任务调度激活。所使用的调度策略利用编译时刻所得到的全局信息，采用了纵向和横向任务合并，处理节点预分配、集中式调度和层调度相结合等措施。其中一个并行任务由以下四个部分组成：

* receive区：接收前导任务的消息；
* 计算区：任务的主体，完成本任务的所有计算；
* 调度区：向调度器发出消息，指示调度器将后继任务调度激活；
* send区：向其后继任务发送消息。

调度协议主要涉及receive区、调度区和send区，包括任务之间以及任务和调度器之间的协议，以保证程序并行执行和串行执行的语义一致。

在JAPS中采用集中式调度模型，调度器每次调度HTG中的一层中的所有任务，高度和时刻由上一层中最先结束的任务决定。文[6]提到，无论是集中式调度还是分布式调度，都存在着瓶颈问题，但是可以通过仔细的设计来消除这种瓶颈。而一个良好的调度协议可以大大压缩调度通信次数，较好地消除这种瓶颈。

3 任务调度协议

任务调度协议包括两个部分：静态的任务描述文

件和动态的任务及调度器间交互协议。两者共同实现任务的调度。在并行分析的任务包装阶段产生任务描述文件。Scheduler 读入任务描述文件，并按任务依赖关系调度任务的执行。任务有多种类型，它们与 Scheduler 的交互协议也有所不同。并行分析产生的任务类型有如下几种：

(1)简单任务：目前除方法调用以外的所有任务都归于简单任务；

(2)循环任务：目前循环任务和简单任务一样处理；

(3)复合任务，头伤务，尾任务：目前只有方法调用对应的任务类型是复合任务。头任务，尾任务是和复合任务接合在一起的。

一个任务的开始执行条件有如下几条：

(1)该任务的当前状态是一定执行；

(2)该任务的所有数据依赖前驱的状态已经确定(前驱的状态是一定执行或一定不执行)，并且一定执行的前驱已经开始执行(不要求执行结束)。

调度协议中需要重点处理的两个问题是：(1)调度分支任务；(2)调度层任务。下面分别讨论处理这两种任务的调度协议。

3.1 分支任务调度协议

由于控制依赖关系的原因，一个任务在编译时无法完全确定它是否一定会执行，因此在任务描述文件中给出关于控制依赖的部分信息。任务的初始属性可能有 3 种：

(1)一定执行：该任务一定会执行；

(2)一定不执行：该任务一定不会执行。目前可以不处理这样的任务。这个属性是保留为以后进一步分析所用的；

(3)不确定：该任务的执行与否需由其他任务指定。

为了实现控制依赖，需要调度器提供一个功能：某一个正在执行的包含条件判断节点的任务(分支任务)可以向调度器指定某些任务的状态转变。合法的转变类型有：(1)任务状态由不确定变为一定执行；(2)任务状态由不确定变为一定不执行。

由此得到分支任务和调度器的交互协议：

1. 任务被启动；

2. 接收调度发来的调度器的地址的消息；(消息 MSG_{sch_addr})

3. 接收前驱结点的地址；(消息 MSG_{parent_addr})

4. 从前驱结点接收数据；(消息号 MSG_{parent_data})

5. 接收完毕时发送执行开始的状态信息给调度器；(消息 MSG_{begin})

6. 执行本结点的代码；

7. 执行完毕，发送执行完毕的状态信息给调度器；(消息 MSG_{end})

8. 如果本节点为条件分支节点，则向调度器指定哪些任务的状态发生变化；(消息 MSG_{cond})

9 接收后继结点的地址；(消息 MSG_{child_addr})

10. 发送数据给后继结点；(消息 MSG_{child_data})

11. 任务执行完毕。

3.2 层任务调度协议

在 Java 中，方法调用具有动态绑定的特性。在分析时只能获得在某个调用点可能调用那个方法而无法确定一定是调用其中的某一个。为了实现这一特性以支持层调度，给层任务增加动态确定下一层子头任务的属性。所有的复合任务都动态地向调度器指定它的子头任务。调度器根据这一指定来动态启动这一层中的所有任务。对应地，下一层的头任务和尾任务也必须通过协议来传送参数和返回值。由此得到层任务调度协议。

对于可扩充的复合任务，其与调度器的交互过程如下：

1. 任务被启动；

2. 接收调度发来的调度器的地址的消息；(消息 MSG_{sch_addr})

3. 接收前驱结点的地址；(消息 MSG_{parent_addr})

4. 从前驱结点接收数据；(消息 MSG_{parent_data})

5. 接收完毕时发送复合任务执行开始的状态信息给调度器；(消息 MSG_{begin})

6. 执行本结点的代码；

7. 向调度器指定自己的子首任务的 ID；(消息 MSG_{layer})

8. 接收此复合任务的下一层的首任务的地址；(消息 MSG_{head_addr})

9. 发送数据给此复合任务的下一层的首任务；(消息 MSG_{head_data})

10. 接收此复合任务的下一层的尾任务的地址；(消息 MSG_{tail_addr})

11. 接收此复合任务的下一层的尾任务的数据；(消息 MSG_{tail_data})

12. 同一般任务结点调度过程的 7 至 10。

对于每一复合任务的头任务，其与调度器的交互过程为：

1. 同一般任务结点调度过程的 1 至 2；

2. 接收对应层任务的地址；(消息 MSG_{layer_addr})

3. 从层任务接收数据；(消息 MSG_{layer_data})

4. 同一般任务结点调度过程的 5 至 10。

对于每一复合任务的尾任务，其与调度器的交互

过程为：

1. 同一般任务结点调度过程的1至7；

2. 接收对应层任务的地址；(消息 MSG_{layer_addr})

3. 发送数据给层任务；(消息 MSG_{layer_data})

4. 发送尾任务完毕的状态信息给调度器；(消息 MSG_{end})

5. 任务执行完毕.

3.3 任务描述文件

在任务调度协议中，任务描述文件也是一个重要的组成部分。虽然通过任务描述文件实现的功能可以全部在运行时刻实现，但这样就需要在一开始将执行条件得不到满足的任务调度到各个节点上去，浪费大量的存储空间；在程序执行过程中，尽管这些任务被激活时，发现执行条件得不到满足会立即放弃对CPU的占用，但进程间的反复切换开销下容忽视，尤其当任务很多的情况下；特别地，运行时刻体现的是一种和任务数有关的多次开销(通信开销等)，而读入任务描述文件只是一次开销，和任务数基本无关，因此尽可能地使用任务描述文件会降低调度本身的负荷，提高系统效率。

任务描述文件中的信息必须是编译时能够获得的。在运行时刻主要处理编译时下能确定的控制依赖关系，任务描述文件则主要描述编译时可以较精确获得的数据依赖关系。其中的关键信息包括任务类型，任务的数据依赖前驱，数据依赖后继等。JAPS中使用的任务描述文件格式如下：

```
int          type;          //类型包括:简单任务,循环任务,方法调用任务,头任务,尾任务;
int          initproperty;  //任务的初始属性包括:必定执行;必定不执行;不确定状态;
int          tid;           //任务的任务号,全局唯一;
int          layerid;       //任务所在层的层号.层与层之间互相不连通;
String       taskFileName;  //任务文件名
int          time;          //预估的任务执行时间,为调度所用
int          callerid;      //任务的调用节点的任务号

int          depcnt;        //在数据依赖关系中父亲节点的个数;
int          depend[];      //在数据依赖关系中,父亲节点的任务号;
int          childcnt;      //在数据依赖关系中子节点的个数;
int          child[];       //在数据依赖关系中,子节点的任务号;
```

4 协议的实现

本系统的运行硬件环境为由多台个人计算机，多台RS6000工作站及多台Sun Sparc工作站通过EtherNet，ATM网构成的分布式并行计算环境。采用的操作系统包括Microsoft Windows 95，Microsoft Windows NT 4.0，IBM OS/2 3.0，IBM AIX 4.2，Sun OS 5.x。整个系统用Java语言实现，需要JDK1.1以上的版本。

协议的实现涉及到分析程序和调度执行程序两个方面。在分析过程中的任务包装阶段，分析程序根据协议的要求将协议通信代码插入到任务文件中，并根据分析得到的依赖关系信息和任务划分的结果生成任务描述文件。在动态的调度执行阶段，首先需要建立调度器的起始状态。涉及到协议的主要工作是从任务描述文件中获取任务信息，包括任务图的结构、任务的开销、任务的类型等。管理模块得到这些信息后，调用任务结点管理模块与任务图管理模块的构造功能建立任务图与任务结点的起始状态。在任务并行执行阶段，调度器通过协议获得任务状态的变化，调用处理器结点管理模块，任务图管理模块，任务管理模块维持状态的变化，产生调度指令。并进行内部状态管理，如刷新处理器负载等。

图2是实例"矩阵闭包运算的并行化"在运行时系统调度的示意图。实际结果表明，该任务调度协议提供了支持并行任务执行的充分设施，并且大大压缩了运行时系统中的通信次数。使用该协议，一个任务从开始执行到全部运行结束，按照任务类型的不同，总共通信次数为8～9次。在调度过程中，如果HTG有m层，则一共只需要向调度器发出m-1次层调度请求。

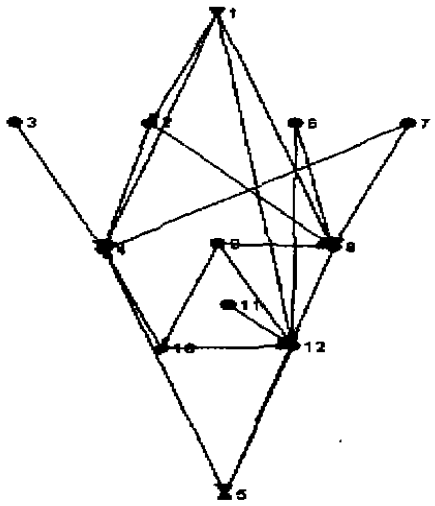

图2 矩阵闭包运算的并行化的调度示意图

结论 任务调度协议是分布式并行计算系统中的一个重要组成部分。在JAPS中我们提出了针对NOW环境的并行任务调度协议。它能充分支撑并行系统对依赖性的要求，而且大大压缩了系统中总的通信次数。较好地支持了系统执行速度的提高。 (下转第6页)

表 2

部件	功能	操作
预处理部件	一般性分析以及对分析结果的处理。	1. 对经过"∩"运算的 S_i、T_i、Q_i 中不相等对应变量取值进行">"、"<"、"⊃"、"⊂"和"--"等各类运算； 2. 根据运算结果给这些变量标识约定的标记； 3. 在各层元组中添加可能有的"差值"变量值。
语义冲突识别部件 这个部件一般要结合谓词推理、"专家系统"或其它人工智能技术设计，还要有人机对话的能力。	分析识别对应变量值间语义不一致性。	1. 对经过"∩"运算的 D_i 中不相等对应变量取值进行"语义矛盾"、"语义交叉"、"语义重复"、"语义含糊"等语义不一致性分析识别； 2. 对经过分析识别的变量取值标识约定的标记。
语义冲突处理部件 这个部件一般要结合谓词推理、"专家系统"或其它人工智能技术设计，还要有人机对话的能力。	处理经过语义冲突识别部件分析识别的对应变量值。	1. 对"语义矛盾"的对应变量值进行处理； 2. 对"语义交叉"的对应变量值进行处理； 3. 对"语义重复"的对应变量值进行处理； 4. 对"语义含糊"的对应变量值进行处理；
综合部件 这个部件一般要结合谓词推理、"专家系统"或其它人工智能技术设计，还要有人机对话的能力。	对经过语义冲突处理部件处理的对应变量值进行综合处理。	1. "∪"运算； 2. "+"运算； 3. 增加变量值中应该增加的内容； 4. 删除变量值中应该删除的内容； 5. 标识变量值中矛盾的内容； 6. 其它。
输出部件	分类输出。	1. 输出 $GW[g_i, g_k]$； 2. g_k。

（上接第 19 页）

参 考 文 献

1 Du Jiancheng, Chen Daoxu, Xie Li. JAPS-An Automatic Parallelizing System Based on JAVA. Science in China, 1999, 3: 279～288

2 Sarmenta L F G, Hirano S, Ward S A. Towards Bayanihan: Building an Extensible Framework for Volunteer Computing Using Java. ACM 1998 Workshop on Java for High-Performance Network Computing, Palo Alto, California, 1998. Concurrency: Practice and Experience, 1998, 10: 1015～1019

3 Thornley B J, Ellenbecher M. An Initial Evaluation of the Tera ultithreaded Architecture and Programming System Using the C3I Parallel Benchmark Suite. SC98, Orlando, Florida, November, 1998. 7～13

4 Edwards A. The Specification and Execution of Heterogeneous Synchronous Reactive Systems: [Ph. D. thesis]. University of California, Berkeley, May 1997

5 Girker M B. Functioal Parallelism Theoretical Foundations and Implementations. University of Illinois at Urbana-Champaign: [CSRD Report]. 1991. 5～19

6 Polychronopodlos C D. Parallel Programming and Compilers. Boston: Kluwer, Academic Pub., 1998. 83～111