境模型中作为未知的部分(谁、时间、地点等)是变项,这种变项的内容范围是有限的,这是一个从一般到特殊的过程。

在情境分析器中,对语义网的处理,是两个过程的相互作用。一是资料驱动加工,是对语义网中的信息的使用,这是一个自底向上的过程。二是用启发知识、策略知识和环境指导对信息的加工,这是一个自顶向下的过程。

一个事件过程属于某一个事件类型,该事件类型包含的情境类型,即是理解该事件过程的背景。在此背景下,能确定概念的具体含义,同时也能确定语义网络中的不确定信息。如从上文中,可确定当前的指示代词及缺省成分等,使该事件过程有一个具体的确切含义。这样的文本理解过程,是一个语义段一个语义段分析的,这是一个自底向上的过程。

假设当前已产生具体的事件过程网为 E-net,一个新的事件过程 e 成立并且可以加入该事件过程网,是指满足下述条件或与其无关:

·与已存在的事件过程间的关系相容。

·与已存在的事件过程相容。

·与当前的情境模型相容。

·与领域知识库相容。

也就是,已存在的事件过程网是判断新的事件过程具体含义的约束条件。这些约束条件对有的理解有约束作用,有的无明显约束作用。

新的事件过程与已存在的事件过程网间的关系的建立,主要是依靠相应情境模型来指导,这是上述的自顶向下过程。

由于人类智能具有非线性、不确定性、不完备性和满意性,所以在计算机中,用文本的内容来填充某个合适的模型产生情景时,也带有一定的模糊性。

结束语 基本的情境模型范畴固然有限,然而由于匹配面涉及太广,在语用理论上不可能有完整表述体系。目前,有关的研究只能在小范围里进行,我们选择的是刑事案件分析这个受限领域。

基于情境理解文本,是我们近些年来一直从事的项目。目前,我们已经完成了一些工作,但仍有许多问题需要探讨。例如,如何利用机器学习来达到情境模型的自我完善,以及如何分析主动对象在整个文本中的活动机制,如在给定的一个情境下,它的自主行为的产生以及对其它对象的影响,而受影响的对象又将如何反应,都是要进一步深入研究的课题。

(下转第15页)

(上接第40页)

了解到这三类结构存在的必要性和合理性。在表中,+代表数据并行语言中包含相应的扩展结构,代表数据并行语言中不包含相应的结构扩展。从表中我们可以看到,除 Vienna Fortran 不提供模板结构而由数组与其它数组直接对准之外,这四个数据并行语言都包含着这几类结构。

表1 数据并行语言及其扩展结构

数据并行语言	HPF	Fortran D	pC++	Vienna Fortran
虚拟处理机结构	+	+	+	+
模板结构	+	+	+	-
对准结构	+	+	+	+
BLOCK 和 CYCLIC 分布	+	+	+	+

参 考 文 献

[1]C. M. Pancake, Do Parallel Languages Respond to the Needs of Scientific Programmers?, IEEE Computer, 23 (12)1990

[2]C. M. Pancake, Software Support for Parallel Computing: Where Are We Headed?, CACM, 34(11)1991

[3]Ian Foster, Designing and Building Parallel Programs, http://www.mcs.anl/dbpp/

[4]G. C. Fox, What have we leant from using real parallel machines to solve real problems?, Caltech report C^3 P-522, Dec. 1989

[5]B. Chapman et al., Vienna Fortran—A Fortran Language Extensioin for Distributed Memory Multiprocessors, ICASE Technical Report 91-72, 1991

[6]G. Fox et al., Fortran D Language Specification, NPAC Technical Report, Syracuse University, 1991

[7]HPFF, High Performanee Fortran Language Specification, Version 1.0, May, 1993

[8]D. Gannon et al., User Guide for a Portable Parallel C^{++} Programing System, pC++, http://www.extreme indiana.edu/sage/pcxx ug/pcxx ug.html

[9]S. Hiranandani et al., Evalution of Compiler Optimizations for Fortran D on Distributed-Memory Machines, In the Proc. of the 6th ACM Intl. Conf. on Supercomputing, July 1992

计算机科学1998 Vol. 25 №. 3

数据并行语言中的扩展结构

Extensional Structures in Data Parallel Languages

韩天舒　胡铭曾　李晓明　方滨兴
（哈尔滨工业大学计算机系　哈尔滨150001）

摘　要　本文通过对并行程序设计的三个步骤及数据并行问题特点的分析，论述了数据并行语言应支持程序员对数据划分和映射的描述，并讨论了为此应对串行语言扩展的几个结构，通过四个成功的数据并行语言，说明了这几个扩展结构存在的必要性和合理性。

关键词　数据并行语言，扩展结构，数据分布，数据对准

1. 并行语言及其结构扩展

直接编写并行程序是非常复杂的，程序员不仅要考虑数据对象及其操作，还要关心并行实体之间的相互作用和时序关系，因此提供并行编程接口，以帮助程序员方便地定义程序的并行特性，对并行的推广和使用都具有重大的意义。并行编程接口按其支持并行的机制可分为三类：全并行语言(Concurrent Languages)、串行语言扩展(Language Extensions)和并行函数库(Parallel Runtime Libraries)[1]。其中，串行语言扩展类并行语言是由串行语言与支持并行的扩展结构——语句或宏构成的，HPF、Fortran D和CC++属于这一类并行语言。由于它具有易于学习，串行程序并行化和并行程序的编写、调试、维护也相对容易的优点，因而成为并行语言发展的主流。文中下面的并行语言指的就是这类并行语言。

在编写并行程序时，程序员通过并行语言的扩展结构描述程序的并行特性。扩展结构决定了程序的哪些并行特性需要由程序员描述，哪些由并行编译器得出。作为程序员和并行编译器的接口，扩展结构决定了他们之间的分工。易于编程和程序的高性能是程序员对语言的根本要求，也是语言设计者的追求。然而，在并行语言扩展结构的设计上二者存在着矛盾。一方面，简略的扩展结构使程序员能简要地定义程序的并行特性，其它的并行特性由并行编译器完成并由它承担更多的性能责任，这往往导致了程序性能的降低[2]；另一方面，全面、详细的扩展结构要求程序员详细地定义程序的并行特性，从而减轻并行编译器的负担并由程序员承担更多的性能责任，给编程带来了更大的困难。

编程和性能之间的矛盾要求并行语言的设计者根据语言的特点确定其均衡策略。一个完整的并行程序需要三个步骤的设计工作：划分、通信和映射[3]。为获得良好的均衡，并行语言应根据其面向问题程序设计三个步骤的特点确定程序员和并行编译器之间的分工，并以此为根据设计语言的扩展结构。数据并行是并行处理的重要研究方向，数据并行语言虽然有着各自的特色，但是它们都有着相似的结构扩展。通过本文，我们可以了解到为什么数据并行语言要提供这些扩展结构，它们的意义和作用是什么，它们是如何被使用的。

2　数据并行问题的程序设计

一个完整的并行程序需要三个设计步骤，划分是第一步，对程序的性能和开发开销都有很重要的影响。一个程序往往有多种划分方案，划分方案最重要的选择是划分方式的选择。在划分时，程序员首先注意问题空间的数据，首先划分数据然后划分计算，这种划分方式被称为数据划分，对数据划分后的子任务进行并行的方式称为数据并行，适于用数据并行方式处理的问题被称为数据并行问题。划分的另一种方法是首先划分问题的计算空间，根据计算划分得出数据划分，这种划分方法被称为功能划分，对功能划分后的子任务进行并行的方式被称为功能并行。数据并行适于处理数据空间易被分解、数据空间上的操作具有一致性的问题，程序的设计、调试和维护相对简单。在并行处理的重要应用领域——科学和工程计算领域中，大多数问题属于数据并行问题。在研究了84个成功的并行应用后，Fox发现其中的83%使用了数据并行[4]。本节主要讨论数据并行问题及其程序设计在划分、通信、映射三方面特点及它们对程序员编程和程序性能的影响，以此作为讨论数据并行语言高级结构的基础。

2.1　数据并行问题的划分

并行程序的划分需要关心两方面：数据划分和计算划分。在设计数据并行程序时，首先进行数据划

分然后根据数据划分确定计算划分，数据划分是整个划分的基础。在编写并行程序时，程序员在脑海中往往有对程序如何并行执行的方案[2]。例如在对模拟某物理现象的三维数组计算中，数据是按片并行执行的。这个方案对程序员是直观的，并且它基于程序员对程序的了解，这是并行编译器无法做到的，因此可以获得较为优化的效果。这个并行方案又往往不够细致，缺乏对并行细节的考虑，在数据并行问题中这个并行方案主要表现为对数据的划分。因此，数据并行问题的数据划分对程序员是直观而恰当的，应该由程序员对此进行描述。另外在许多数据并行问题中，数据空间一般为数组，数据划分表现为数组的划分，程序员可以通过划分数组方便地描述数据划分，在对程序性能的影响上，数据划分是基础，决定了程序的并行度、负载情况和通信需求，是影响性能的关键。

数据并行问题的计算划分主要是由数据划分和数据引用方式决定的。由于数据引用方式的多样性和复杂性，所以即使一个非常简单的数据划分也能引起复杂的计算划分。对于确定的数据划分，计算划分对程序员来说是不直观的、困难的。在对程序性能的影响方面，由于计算划分在很大程度上是由数据划分决定的，确定的数据划分下计算划分的自由度较小，因而对程序性能的影响也较小。

2.2 数据并行问题中的通信

划分后各并行任务不能独立地运行，因为一些任务上的操作需要引用其它任务上的数据，因此需要通信处理任务之间的数据传输。在数据并行问题中，数据划分将具有相同或相似操作的数据空间划分开，严重地割裂了数据之间的联系，因此并行任务间的通信是大量、频繁而复杂的[3]。对于数据并行程序而言，即使一个简单的划分也会产生复杂的通信结构[3]，通信的确定对程序员来说是非常困难的。

在大多数并行系统中，通信开销对程序效率的影响非常显著，但在数据并行问题中通信开销主要是由数据划分所确定的并行结点程序间的通信关系和通信量决定的。在结点程序的优化上已经开发出许多优化方法[9]，并行编译器在无程序员参与的情况下可以得到较好的优化效果。

2.3 数据并行问题中的映射

高级语言程序的一个重要特点是必须具有可移植性，程序员在编写并行程序时不可能了解实际运行并行系统的情况，因此并行程序的映射一般分为两步：由程序员指定的从问题空间到虚拟处理机空间的映射和由并行运行环境完成的从虚拟处理机空间到实际处理机空间的映射。由于数据并行程序一般采用 SPMD 的并行执行方式，因此其映射描述比较简单。映射问题独立于数据划分，对程序员来说是较为直观和简单的。在数据并行程序中映射同数据划分结合构成数据分布，数据分布不仅能更清楚地描述数据如何被分布到各个处理机上，而且能够反映程序划分的粒度，影响并行程序的性能，因此需要程序员的参与。

3 数据并行语言中的结构扩展

数据并行语言由串行语言与支持数据并行问题的扩展结构构成。根据上节的讨论，数据并行问题的数据划分和映射设计对程序员是直观、易于确定和描述的，两者相结合构成数据分布，确定了程序的数据分布方案及划分粒度，对程序的性能有着重要的影响；计算划分和通信的设计决定于数据划分，对于程序员是困难的，对程序性能的影响也较小。因此，数据并行语言应扩展简单、灵活的结构以支持程序员方便地描述他的数据分布方案；而程序的计算划分和通信应由并行编译器确定，无须扩展相应的结构。本节讨论数据并行语言中所应扩展的结构。

为完成数据在虚拟处理机上的分布，虚拟处理机结构是必要的。不同的问题需要一维或多维的数据分布以获得一维或多维的并行，因此虚拟处理机应能定义为一维或多维以方便其相应的分布描述。我们以 HPF 为例说明对虚拟处理机结构的描述。HPF 的虚拟处理机结构为!HPF $ PROCESSORS，确定数目的虚拟处理机可以定义成任意维数。例如，32个虚拟处理机可定义为!HPF $ PROCESSORS pr(32)或!HPF $ PROCESSORS pr(4,8)。

借助于虚拟处理机结构，程序员可以完成数据分布。数据并行问题中的数据分布表现为数组的分布，由于一个问题往往涉及到多个数组，因此需要对多个数组进行分布。这时程序员不仅要关心每个数组是如何被分布的，而且要关心数组间分布的对应关系，后者对数据并行程序的通信关系和通信量有重要的影响，程序员需要根据数组间的引用关系将这些数组"对准"地分布，有两种对准方法：一是将每个数组单独地分布并保证它们的对准关系；二是首先描述这些数组与某个数组或模板的对准关系，然后对这个数组或模板进行分布。第二种方法虽然增加了数据分布的步骤，但是由于能帮助程序员根据数组间的引用关系描述数组间的对准关系，因而程序员能够更方便地表达他对提高程序性能的看法。数据并行语言可以提供模板和对准结构以帮助程序员完成数组的对准[5,7,8]。其中，模板结构定义了一个数组，程序员能够将其它数组在需要的各维上与模板对准；对准结构定义了模板与数组之间的对准关

系。一些数据并行语言不使用模板而用某个数组与其它数组对准[5]。我们仍以 HPF 为例说明。HPF 的模板和对准结构分别为：!HPF $ TEMPLATE 和! HPF $ ALIGN。它的对准方法如下图：

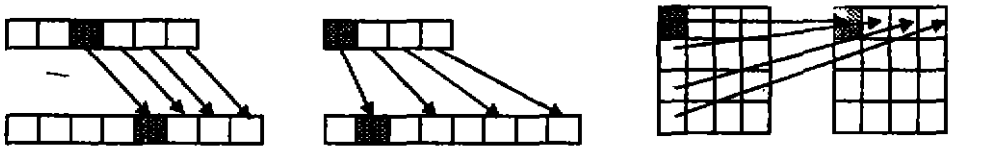

图1.a ALIGN A(I) WITH B(I)　图1.b ALIGN A(I) WITH B(2*I)　图1.c ALIGN A(I,J) WITH B(J,I)

图1　HPF 中 ALIGN 结构的例子（箭头和阴影代表相应对准的数据单元）

分布结构对数据并行语言是必要的，数据并行语言应提供简单、灵活的分布结构以帮助程序员方便地完成某些维上的数据分布并获得较高的性能。设 N 为分布维上数组的数据单元个数，P 为对应维的虚拟处理机个数。对并行程序来说，结点程序间的通信和负载平衡情况是影响程序性能的两个关键因素。由于数据并行问题的数据依赖多发生于相邻的数据之间，因此将数据连续地分布能够减少结点程序间的通信需求。BLOCK 方式将数据按大小相等且连续的块进行分布，每个块的数据量为 N/P，这种分布方式可以减少结点程序间的通信量。由于程序在其数据空间上的计算量具有连续性，因此连续的分布方式容易造成负载的不平衡。CYCLIC 方式将每第 P 个数据分布到相同的虚拟处理机上。由于数据空间上计算量的连续性，这种分布方式易于获得均衡的负载。图2说明了两种分布方式是如何将32个数据分布到4个虚拟处理机上（每个虚拟处理机分得8个数据）的。图中带有阴影的方格代表分布到第一个虚拟处理机上的数据。数据并行语言都提供 BLOCK 和 CYCLIC 分布方式。由于两种分布方式在负载平衡和通信量之间的矛盾，因此有的语言还提供 BLOCK-CYCLIC 分布方式以获得二者的平衡。通过 BLOCK 或 CYCLIC 方式，程序员可以对任意维的数据进行分布。HPF 提供了数据分布结构！HPF DISTRIBUTE $，它的使用方法为！HPF $ DISTRIBUTE A(BLOCK)ONTO pr。

图2.a BLOCK分布方式

图2.b CYCLIC分布方式

图2 HPF中的BLOCK和CYCLIC分布方式

数据的分布方案应适应数据的计算和引用关系以获得高性能。由于程序各个阶段具有不同的数据计算和引用关系，因此为获得全局的性能效果需要在不同的阶段采用不同的数据对准关系和分布方案。数据并行语言应支持动态的数据对准和分布。有两种方法用于实现动态的数据对准和分布：有的数据并行语言直接允许数据对准和分布的重定义；另一些数据并行语言则提供了单独的动态结构。在 HPF 中，DYNAMIC 结构说明一个数据结构是动态的，这样的数据结构可以在程序的不同部分使用 REALIGN 和 REDISTRIBUTE 结构修改数据的对准和分布方案。

综上所述，数据并行语言需要提供三种主要的扩展结构：虚拟处理机结构、模板和数组对准结构、数组分布结构（BLOCK 和 CYCLIC 两种分布方式），并应允许动态的数据对准和分布。图3说明了一个完整的数据分布包含的两个步骤，它还反映了数据并行语言三类结构的作用。

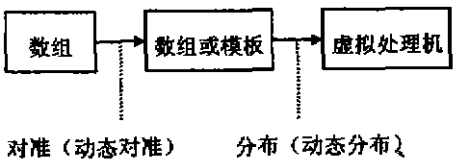

图3　数据并行程序的数据分布

尽管数据并行语言因其基于的串行语言和设计者的想法的不同有着各自的特点，但它们都以各自的方式提供这三类扩展结构。表1描述了四种比较成功的数据并行语言 Vienna Fortran[5]，Fortran D[6]，HPF[7]和 pC++[8]的扩展结构。通过表1，我们可以

（下转第29页）