域:支持并行程序概念设计的一种抽象手段*)

DOMAIN, An Abstraction Means for Parallel Program Conceptual Design

董超群　陆林生

(江南计算技术研究所　无锡214083)

Abstract　DPHL is a Data Parallel High-level modeling Language, with which users can describe problem-solving course in algorithmic level, closer to their perspectives. Domain is a basilic concept introduced in DPHL. This paper discusses the definition of domain and its abstraction method in supporting parallel program conceptual design.

Keywords　Domain, Parallel program, High-level modeling language, Conceptual design

1　引言

长期以来,并行计算机在其潜在高性能和现实性能以及简单易用性之间存在着巨大的鸿沟,原因之一在于并行程序设计方法学的滞后导致大量的并行程序设计实践沿袭着传统串行程序开发的思路,而另一个重要的瓶颈在于并行程序的开发缺乏高效编程环境的支持。

目前,并行程序的构造主要采取以下两种方式:(一)依据并行算法编写并行程序。这种方式构造的并行程序,一般能达到较高的并行效率,但对广大应用领域的用户要求太高。(二)利用并行化工具对串行程序进行并行改造,由于串行计算机上已经积累了大量成熟的应用程序,通过自动并行编译工具将串行程序移植到并行计算机上运行,无疑具有极为重要的现实意义。并行化方法由于对并行编译的对象缺乏高层的全局信息和并行信息,并行粒度较细,并行效率往往不高;另外,由于并行化方法以实验方法作为基本研究途径,即通过对具体实际程序的分析,进而总结归纳出有效的并行化编译技术,针对向量程序和以共享存储(SMP)为主的应用程序,并行化研究取得了较大的进展。而到90年代末期,由于针对MPP应用程序的并行编译对象复杂度增大,并行化的研究开始出现了停滞的现象[1]。

20世纪90年代中期,国际上开始探索并行程序构造方式的新途径,一种基本思路是:首先设计一种面向科学和工程计算应用课题的高层建模语言(high-level modeling language)或元语言(meta-language),从算法层次上描述应用问题的计算步骤(类似于串行程序设计),由并行程序综合器利用应用课题本身具有的高层信息,经并行性分析和优化后,自动或半自动生成适合于目标机器的并行程序(类似于并行编译,但并行程序综合器"掌握"了求解问题的全局信息),我们称这种方法为并行程序的概念设计方法。

利用并行程序概念设计方法构造并行程序,在串行程序设计的简单性和发挥并行计算机性能之间能达到某种平衡,有望切实减少用户的编程负担,使用户把主要精力集中在求解问题的数学描述上,并有望实现较大粒度的并行,达到较高的并行效率。

2　高层建模语言

2.1　与并行编程语言的关系

高层建模语言是实现并行程序概念设计的基础。我们知道,目前广泛使用的并行编程语言主要有以下两种类型:(一)对传统的串行语言进行扩充。这种扩充表现在三个方面:①扩充新的语言结构。例如,Fortran 90在Fortran 77中扩充了聚合数组操作以支持向量并行运算;HPF在Fortran 90中扩充了FORALL结构(语句)以提供灵活的并行描述。②扩充编译指示。例如,HPF在Fortran 90中增加了ALIGN、DISTRIBUTE编译指示,指导编译器在多处理器间进行数据分布,以均匀地划分负载、减少通信开销;增加INDEPENDENT编译指示,以利于编译器优化并行代码等。③扩充并行支持库。例如,C语言配上MPI/PVM库可以支持消息传递方式的并行程序设计。(二)新型并行语言例如Linda和Occam,在这两种类型的

*)本课题得到国家自然科学基金(编号10072077)资助。董超群　博士研究生,工程师,主要研究领域为并行计算环境、CAD/CASE。陆林生　教授,博士生导师,主要研究领域为计算方法、并行计算环境等。

并行编程语言中，新型并行语言可以使用并行的概念或结构来描述程序中的并行性，但由于这类语言往往与现有的串行语言不兼容，用户受惯性驱使短期内难以接受；对串行语言进行扩充而形成的并行语言，与现有大量串行程序继承性较好，并行性由用户显式表达或者由专门的优化编译器实现并行性的自动检测，但这类语言性能易受串行语言的限制，不易充分发挥并行性，而且，描述并行性的层次较低，需要与机器具体的特性紧密结合才能编写出高效率的并行程序，这对广大应用领域的用户来说要求太高。

高层建模语言与上述两种类型的并行编程语言在并行程序构造方式上有较大的差异。首先，高层建模语言与串行程序设计类似，没有定义新的并行成分或结构，用户容易掌握；其次，高层建模语言引入了域及其相关概念，使得应用问题的描述更加简洁，并且在算法描述中提供了关于求解问题的高层信息，支持程序综合器在自动转换成目标机器上的并行程序时能充分开发应用问题本身所蕴含的并行性。

2.2 相关研究

1994年澳大利亚联邦科学和工业研究组织(CSIRO)下属的信息技术分支机构(Division of Information Technology)和大气研究分支机构(Division of Atmospheric Research)联合开发了DPML(Data Parallel Meta Language)语言[2]，该语言引入了数据域(data domains)的概念，用于显式描述课题中的并行性，并定义了数据域上的操作和通信方式，使得应用课题中的部分或全部运算能够用一种甚高层次(a very high level)的方式描述。DPML程序经一种称为DP77的翻译器转换为并行Fortran程序。采用DPML语言编写的气象预报中的浅水模型(shallow water model)程序，在Cray Y-MP、MasPar MP-1等机器上取得了与手写Fortran程序相近的性能。

1997年9月瑞士科学计算中心(CSCS)研制了DMSL(Data Modeling Specification Language)语言[3]，用于描述求解问题的计算步骤，由自动程序综合器结合模板和基于知识库的推理引擎将计算步骤转换成目标平台上的C语言程序。DMSL语言在求解线性代数方程组数值解等问题上验证了其实用性。

表1 DPML、ZPL、DPHL三种语言比较

	DPML	ZPL	DPHL
域	规则域	规则域，可以是稀疏域	可以是不规则域和稀疏域
	DOMAINS Cube=[I,J,K]	Region cube=[I,J,K]	DOMAINS cube=[I,J,K|(J+K)%2=0]
子域	不支持	不支持	支持 SUBDOMAINS Sub(cube)=[I,J,K|K=Kmax]
依赖域	没有明确支持 定义了域上的变换式(TRANSFORMS)	没有明确支持 定义了region上of、in、at、by、>>等运算符	支持 DEPENDENT_DOMAINS cube@WJ=[I,J-1,K];
变量	变量定义不带数学属性 基本变量(可维持历史状态) FUNDEMENTAL VARIABLES U cube real 导出变量(临时或局部变量) DERIVED VARIABLES V grid real	变量定义不带数学属性 变量 var U:[cube]real; 配置变量(可在命令行改变) config var n:integer=512,	变量定义带数学和网络属性 逻辑变量 VAR U COL_VECTOR(5)cube; 存储变量 VAR_STRUCT real U1[L,I,J,K];
复合函数校举变量算子	不支持	不支持	支持
目标语言	Fortran 90/Fortran 77	ANSI C+MPI(PVM)	HPF

代码示例(矩阵乘)	DOMAINS
	R=[1…n,1…n]; FUNDEMENTAL VARIABLES
	A,B,C R real;
	C=A*B;
		Region R=[1…n,1…n];
		var A,B,C:[R]double;
		[R]begin
		C=0.0;
		for i:1 to n do
		C+=(>>[1…n,i]A)*(>>[i,1…n]B);
		end;
			DOMAINS
			R=[1…n,1…n];
			VAR
			A,B,C R;
			C=A*B;

1997年7月美国华盛顿大学计算机科学和工程系开发了 ZPL(Z-level Programming Language)语言[4]，该语言引入了"区域"(region)的概念，定义了 region 上的运算符以实现对数组的整体操作，以区别于传统数组语言(如：Fortran 90)中的"数组片段(slice)"和大量的下标索引。采用 ZPL 编写的程序经编译生成运行于宿主机上的 ANSI C+MPI(或 PVM)程序，在多个并行平台上(Intel Paragon，Cary T3D 等)，通过与 APR(Applied Parallel Research)xHPF 基准程序对比，ZPL 取得了优于 HPF 的并行性能[5]。

1998年5月，我们设计开发了 DPHL(Data Parallel High-level modeling Language)语言[6]，并在此基础上构造了一个包括词法、语法分析、变量相关性分析、Cache 命中率提高、并行程序自动生成等在内的并行编程环境，用于支持并行程序概念设计。

与 DPML、ZPL 等语言类似，DPHL 引入"域"的概念，以支持应用课题的简洁描述，便于进行并行性分析和优化。其中 DPML、ZPL 和 DPHL 三种语言主要语法成分的比较见表1。

从表1可以看出，DPHL 语言与 DPML、ZPL 等语言在较多方面存在着明显的区别。首先，DPHL 将域由稠密域扩展到稀疏域，由规则域扩展到不规则域，而且引入了"子域"的概念，增强了域运算能力。第二，DPHL 将变量的数学属性与定义域分离，实现了符合数据并行模式下基于域的运算。第三，DPHL 的变量定义采用了与数学上一致的定义方式，较好地支持并行性分析和变量依赖关系分析。第四，DPHL 支持复合函数、算子、枚举变量描述，使语言的抽象程度更高，程序更简洁，从而使算法描述更进一步接近于数学形式的表述。

3 基于域的计算

域是高层建模语言中引入的一个基本概念，从应用问题数学描述的角度看，域是求解空间离散化后自变量的定义域；从可编程角度看，域是 n 维整数空间 IR^n 中的一个封闭区域，下面先给出域的定义。

3.1 域

定义1(域) 设四元组 r=(l,h,s,a)代表整数空间的一个序列，r 中的元素用集合 $S(r)=\{x \mid l \leqslant x \leqslant h \text{ and } x \equiv a \pmod s\}$ 表示。则由 d 个这样的整数系列组成的 d 元序偶 $R=(r_1,r_2,\cdots,r_d \mid F(r_1,r_2,\cdots,r_d)=0)$ 称为整数空间上的一个 d 维域，$r_i(1 \leqslant i \leqslant d)$ 称为域的第 i 维，$F(r_1,r_2,\cdots,r_d)=0$ 表示维之间的依赖关系，其中，h,l 代表整数序列的上、下界，s 为整数序列的跨距，a 规定了整数系列的一种排列方式。

定义1涵盖了稀疏域和不规则域。当 $|s|>1$ 时，域为稀疏域。特别地，当 s=1时，域即是通常所讨论的稠密域，这时，a 可为任意整数，通常取维的下限，即 a=1。通过限定整数系列的排列方式，可以定义不规则域。

根据定义1，域 R 中的元素可以表示为整数系列 $r_1,r_2,\cdots,r_d$ 中元素的笛卡尔积，即 $I(R)=S(r_1)\times S(r_2)\times\cdots\times S(r_d)$。例如，一个2维域{(1,6,2,0),(1,6,2,1)}含有下列元素：

I({(1,6,2,0),(1,6,2,1)})
=S(1,6,2,0)×S(1,6,2,1)
={2,4,6}×{1,3,5}
={(2,1),(2,3),(2,5),(4,1),(4,3),(4,5),(6,1),(6,3),(6,5)}

在高层描述语言中，域的基本要素包括整数空间的基，或称整数空间的坐标系，以及域在各坐标轴投影的值域。例如，一个3维域在 DPHL 中定义如下：

```
DOMAINS
    AREA=[0..5,0..100,0..100];
```

其中，AREA 为域的命名。为了表述清晰直观，通常的做法是将各维分离出来，与整数空间的坐标轴对应，采取分别命名的方式予以定义，例如：

```
DIMENSIONS
  I=0,5;
  J=0,100;
  K=1,100;
```

则上述 AREA 可表述为：

```
DOMAINS
    AREA=[I,J,K];
```

规则域为整数空间 IR^n 中的一个超立方体，例如上面定义的域 AREA。为了表达不规则域，在 DPHL 语言中，一个 d 维域更一般的定义方式如下：

```
DOMAINS
  AREA=[I1,I2,…,Id|F(I1,I2,…,Id)=0];
```

$F(I_1,I_2,\cdots,I_d)=0$ 界定维 $I_1,I_2,\cdots,I_d$ 之间的依赖关系。例如：

```
DIMENSIONS
    I=0,5;
    J=0,5;
DOMAINS
    AREA=[I,J|J==I];
```

所表示的计算区域如图1所示：

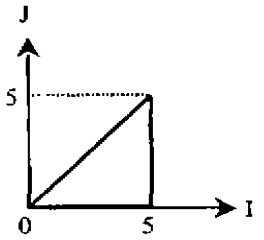

图1 不规则域示意

3.2 域的相关概念

(1)子域 在真实应用问题中,变量的定义域和求解域往往并不一致,变量在整个定义域中的求解往往需要分解成在其多个子域上求解才能完成,在不同子域上的计算公式往往不同,甚至并行特性也有差异。在DPHL语言中,引入子域的概念来描述这种差异。

例如,用红-黑排序、SSOR迭代法求解五点迭代格式椭圆型Possion方程,假设基本域定义为:

AREA=[0:N+1,0:N-1];

红点计算域和黑点计算域可分别定义两个子域:

```
SUB_DOMIANS:
  AREA_R(AREA)=[I,J|(I+J)%2==0,
    I>=1 && I<=N,J>=1 && J<=N];
  AREA_B(AREA)=[I,J|(I+J)%2==1,
    I>=1 && I<=N,J>=1 && J<=N];
```

则在两个子域上可分别实现并行计算。

(2)依赖域 域和子域都是通过对空间区域的描述和构造来支持数据并行性。依赖域也具有与域和子域类似的空间区域特性,但引入依赖域概念更主要的目的在于描述变量在求解空间上的依赖关系。

在许多实际应用问题中,变量之间在空间邻点附近常常存在着简单的依赖关系,例如,二阶中心差分逼近导数的形式为:

$$\left(\frac{\partial^2 u}{\partial x^2}\right)_{i,j}\sim\frac{1}{\Delta x^2}(u_{i+1,j}-2u_{i,j}+u_{i-1,j})$$

$$i,j=1,\cdots,N \qquad (1)$$

如图2所示,依赖点是(i,j)附近相邻的三个点:P(i-1,j)、P(i,j)、P(i+1,j)。

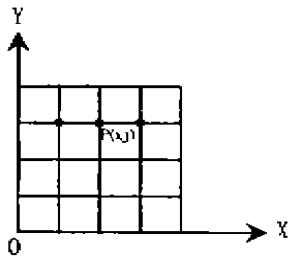

图2 空间邻点依赖

在DPHL语言中,用依赖域来描述这种简单的空间依赖关系,例如:

```
DEPENDENT_DOMAINS
   AREA@WI=[I-1,J];
   AREA@EI=[I+1,J];
```

则公式(1)用DPHL语言可以表达为:

```
V=1.0/(deltax * deltax) * (U@WI-2 * U+U@EI);
```

由此可见,依赖域主要针对域上某一维的邻点依赖关系加以描述。引入依赖域可以使DPHL以接近数学上的方式描述计算公式,而且可以使变量的依赖关系分析变得较为容易。

3.3 基于域计算的讨论

(1)支持对求解问题的简洁描述 引入域、子域和依赖域后,变量定义在域上,可以在域、子域和依赖域上引用。对定义在域上的变量可以实施整体计算,虽然传统的数组语言(例如Fortran 90),也可以对数组的全体元素进行引用,但数组的内部元素和边界元素常常需要分别施加不同的处理。实例表明,使用高层建模语言(例如DPHL),能有效地简化对求解问题的描述,消除程序中的代码冗余,使得求解相同问题时程序规模通常只有一般并行程序设计语言的1/3~1/4[7,8]。

由于域、子域和依赖域可以分别命名,程序员可以对计算域指定确定的意义,如同一般程序语言中对变量给出标识符进行命名而不是直接使用没有任何意义的内存地址一样,避免了传统数组语言中繁琐的下标索引,使得高层建模语言描述的程序可读性好、不易出错。

域、子域和依赖域的使用突出了应用问题中变量在求解空间上的共性和差异。例如,对某一计算式,用Fortran 90描述为:

```
Temp(1:m,1:n)=A(0:m-1,1:n)+
        A(2:m+1,1:n)+A(1:m,2:n+1)+
        A(1:m,0:n-1)
```

而用DPHL描述为:

```
Temp[area]=A@WM+A@EM+A@NN+A@SN;
```

(2)支持代码重用 高层建模语言中将定义域与在域上的计算分离开来,使得程序代码更加通用。程序中当求解域发生变化时,例如从2维域扩展到3维域,使用基于域计算的高层建模语言编程,程序代码几乎不用修改,而对于数组式语言(例如Fortran 90),则程序中每一维的下标索引都要进行相应的修改。

(3)支持并行性能分析 基于域计算的最重要的特色在于高层建模语言在程序的说明部分以域(子域、依赖域)的形式定义了变量的求解域,并显式定义了变量的数学属性、状态特性和存储分布,支持程序综合器进行并行性分析和优化。例如,对作用在相同域上的语句段,程序综合器经过分析,结合定义域和数学属性,可以寻找出程序段的最佳并行方向,并对相邻的语句进行合并和优化,以组织成更大粒度的并行计算。

依赖域的定义突出表达了变量在空间点上的依赖关系,使得程序综合器在生成目标机器上的并行程序时掌握了足够的信息以进行依赖关系分析和边缘通信优化,避免了类似并行化工具那样需要对数组下标作复杂的符号分析。

3.4 域与数组之间的关系

将高层建模语言所描述的求解问题计算步骤直接编译成并行计算机上的可执行代码，一方面工作量太大，另一方面可移植性差。目前，DPML、DMSL、ZPL、DPHL 等语言一般选取某种可编译的并行语言，例如 HPF 或 C+MPI，作为转换成并行计算机上运行的目标语言。

高层建模语言（下面以 DPHL 为例）中的域不等同于传统程序设计语言（下面以 HPF 为例）中的数组，虽然在转化成目标语言时通常用数组来具体实现域的定义和基于域的计算，但二者在本质上是不同的。

首先，相对于数组而言，域的抽象层次更高。DPHL 中将应用问题的求解空间抽象出来，以域的形式独立进行定义，使得描述问题更简洁，可读性好，程序的描述过程更接近于数学上的表达。

其次，域更利于进行并行性分析。我们知道，在真实应用问题中，变量通常具有空间性和数学属性，并且变量在二者之间所呈现的并行特性往往是不同的。例如，对于作用在3维空间[I,J,K]的变量 U，U 的每一个元素为2维矩阵[M,N]，HPF 通常将 U 定义为5维数组参与运算，使得变量的空间性和数学属性湮没在程序代码中，不利于并行化工具对空间性和数学属性分别实施并行性分析。DPHL 语言将变量的空间特性与变量的数学属性分开定义（例如：U MATRIX(M,N) area），使得程序综合器在生成目标程序时能根据不同的并行特性，组织成较为优化的并行代码。

最后，高层建模语言中的域与所生成的目标语言中的数组不一定存在一一对应的关系，例如上面3.2-(1)中的子域 AREA_B 可以用数组加条件判断语句加以实现。

另外，基于域的计算表达方式更为灵活，以 DPHL 语言中的域赋值语句为例，例如：

U=(U@EI-U@WI)/(X-X@WI);

其中变量 U 定义在3维域上，X 定义在1维域上，U 和 X 不同形；而 HPF 语言中的数组赋值语句要求语句中参与计算的数组必须同形。

结束语 本文提出了并行程序的一种新的构造途径——并行程序概念设计方法。并行程序概念设计方法以高层建模语言为基础，在程序编写方面具有串行程序的简单性，高层建模语言描述的计算步骤经并行程序综合器优化和分析，自动生成可在并行计算机上编译运行的并行程序。本文重点介绍了高层建模语言的核心概念——域的定义及其支持并行程序概念设计的抽象方法。

目前，我们在 DPHL 语言的基础上构造了一个并行程序自动生成系统，并且用"显式差分法求解三维 N-S 方程"、"三维湍流直接数值模拟"、"红-黑排序、SSOR 迭代法求解五点迭代格式椭圆型 Possion 方程"等实例课题进行了验证，所生成的并行程序在"神威-II"并行计算机上取得了较好的运行性能[7]。

参考文献

1 臧斌宇. 实验方法-并行化编译研究的基础. 见：计算机与微电子科学技术国际研讨会论文集. 北京：北京大学出版社，1998. 268～270

2 Francis R S, et al. A Data Parallel Modeling Language. Journal of Parallel and Distributed Computing, 1994, 21, 46～60

3 Decker K M, Wylie B J N. Software Tools for Scalable Multilevel Application Engineering. The International Journal of Supercomputing Applications and High Performance Computing, 1997, 11(3): 236～250

4 The ZPL Parallel Programming Language. http, www. cs. washington. edu/research/zpl.

5 Lin C, Snyder L. A Portable Implementation of SIMPLE. International Journal of Parallel Programming, 1991, 20(5): 363～401

6 陆林生，等. DPHL 语言设计方案：[江南计算技术研究所内部研制报告] 1998.5

7 董超群. 基于 DPHL 语言的并行程序自动生成技术 [工学博士论文] 2000.11

（上接第42页）

5 Shekhar S. Spatial Databases—Accomplishments and Research Needs. IEEE Tran. on Knowledge and Data Engineering, 1999, 11(1)

6 Johnsen J A, et al. Concept mapping in mental health: uses and adaptations. Evaluation and Program Planning, 2000, 23, 67～75

7 Chin Chiung-Hui, et al. The evaluation and influence of interaction in network supported collaborative concept mapping. Computers & Education, 2000, 34: 17～25