国防科技大学学报
JOURNAL OF NATIONAL UNIVERSITY OF DEFENSE TECHNOLOGY
第 19 卷第 3 期　1997 年 6 月　Vol. 19 No. 3

一个基于面向对象模型的并行系统*

肖　侬　胡守仁　宋　辉　韩　冰

(国防科技大学并行与分布处理国家重点实验室　长沙　410073)

摘　要　为了开发分布式系统中的计算资源，我们设计了一个基于面向对象大粒度数据流模型的并行 C++ 系统 OOCPCS. 该系统的底层计算模型是面向对象范式和数据流模型的结合体。它将状态对象引入到数据流模型中，并且支持面向对象的特性。本文讨论了此模型在 OOCPCS 系统中的实现；并行化编译器；POC 程序设计语言；面向对象网络文件 I/O 和虚拟 OOLGDFM 机。

关键词　状态对象，数据流，actor forward-list

分类号　TP301

A Parallel System Based on an Object-oriented Model

Xiao Nong　Hu Shouren　Song Hui　Han Bing

(Parallel and Distributed Processing National Lab, NUDT, Changsha, 410073)

Abstract　A system based on object-oriented large-grain data flow model, called OOCPCS, is designed to facilitate parallelism in distributed system. Its underlying model of computation is an integrated object-oriented paradigm and large-grain data flow model, called OOLGDFM. It introduces persistent state object and object-oriented features into large- grain data flow model. Its graphs are constructed dynamically. The paper discusses the realization of the model in OOCPCS, the use of forward __list, parallelizing compiler, POC programming language, object-oriented network file I/O, and the virtual OOLGDFM machine.

Key words　state object, data flow, actor, forward __list

目前，解决科学计算问题有两种发展方向。一种是大规模并行处理机的开发(MPP)；另一种是分布式计算。基于网络计算环境的并行计算已进入实际使用阶段。

* 1996 年 9 月 29 日收稿

面向对象技术[1]是目前广为接受的计算机科学中的新概念、新技术,面向对象范式很自然地反映了问题空间,面向对象程序设计以继承、数据抽象实现了信息隐蔽,增强了系统模块化、可扩展性、可维护性和可理解性,实现了代码共享和重用。这样一种技术和范式为大型软件程序设计和软件工程提供了一种有力的支持手段。

数据流模型[2]为并行处理提供诱人的特性,按照 U-Interpreter[3],数据流图揭示了程序中所有的并行性,并消除了显式管理并行执行的需要,数据流模型对于高性能计算来说,优于传统的控制流模型,大粒度数据流模型[4]在分布式系统下,保持了通讯开销相对较低。

本文描述了一个基于面向对象大粒度数据流模型(OOLGDFM)的并行系统 OOCPCS. OOLGDFM 是基于消息传递而设计的,它与传统的大粒度数据流模型有几个方面的不同:

(1) OOLGDFM 不仅通过弧传递信息,还引入了状态对象,以便于程序间的信息共享。

(2) 支持面向对象特性,例如继承、多重继承、动态联编等特性。

(3) OOLGDFM 的数据流图是动态构造的。

1 面向对象大粒度数据流模型

传统的大粒度数据流模型不太适合分布式的应用,主要原因是它没有为程序间提供一个共享数据的机制,并且,所有信息传送只在有连接弧的结点间进行。这些特性对于科学计算较为适合,然而,对于其它多数应用来说,程序的基本目标是交换数据和共享数据,例如:数据库、AI 等领域,大粒度数据流模型很难模拟这些系统。因此,我们开发了 OOL-GDFM,以支持分布计算。

在详细描述这个模型之前,首先给出一些定义和说明。

对象集合 OBJSET:所有对象组成的集合;类集合 CASET:所有类组成的集合,类间关系仅为继承关系。在 OOLGDFM 中,认为类定义了对象的公共属性和公共操作,它为对象提供了共享的属性(域)和操作,但不提供对象间属性值的共享。

对象关系分为消息关系 MRS、聚合关系 CLN、创建关系 CRT 和共享数据 SHD 四种。

OOLGDFM 支持对象的并行性。我们可以用六元组来表示 OOLGDFM.

MODEL= < PO, SO, PP, SP, T, G>

PO 和 SO 是类的实例对象;

PO= $\{PO_1, PO_2, \ldots, PO_n\}$;处理机对象(processor objects)集合;

其中 PO_i= < ID, MD, DS, LSP, ≅ > ;≅ 表示处理机对象是无状态的对象,它的内部状态信息在各个请求执行之间不予保留。

SO= $\{SO_1, SO_2, \ldots, SO_m\}$;状态对象集合;

其中 SO_i= < ID, MD, DS, LSP, Σ> 且 Σ≠≅:具有记忆操作效果的状态对象集合。状态对象是一个有持久内部状态的对象,它的内部状态在本次请求服务执行后,维持至下次请求服务执行时,供各程序间共享信息。

PO ∪SO⊆ OBJSET, 任意两个对象 $O_i, O_j \in PO \cup SO \quad i \neq j$, 则 $O_i \sim MRS\ O_j$, $O_i \sim CLN\ O_j$, $O_i \sim CRT\ O_j$, $O_j \sim MRS\ O_i$, $O_j \sim CLN\ O_i$, $O_j \sim CRT\ O_i$, $O_i \sim SHD\ O_j$.

PP: 处理机对象的消息处理计算体(INPE)集合。当向一个处理机对象发送一个消息时, 异步形成一个对应处理机对象的消息处理计算体, 在 OOLGDFM 流图中称为 pactor. 一个处理机对象的所有 pactors 是等价的。pactors 的输入数据齐备即可计算, 并产生输出 TOKENS.

SP: 状态对象的消息处理计算体(INPE)集合。当向一个状态对象发送一个消息时, 异步形成一个对应状态对象的消息处理计算体, 在 OOLGDFM 流图中称为 sactor. 它操作计算, 产生输出结果 TOKENS, 并使 ID_i 对象的状态从 Σk 到达 Σj.

$T \subseteq (SP \cup PP) \times (PP \cup SP)$; 它表示计算体间的数据流的连接集合。

$G = < ND, T >$: G 是程序执行数据流图, $ND \subseteq PP \cup SP$ 是图中的结点, $T \subseteq T$ 表示数据流连接。

传统的数据流图可以由编译器静态产生, 程序流图是静态的。OOLGDFM 在静态时很难确定, 主要是因为状态对象的引入。

此外, 一个对象内部可以含有(创建)一些处理机对象和状态对象, 我们把它们称为子对象。一个对象内部也同样可以按 OOLGDFM 来实现。

OOLGDFM 通过引入了状态对象和面向对象的特性, 将大粒度数据流模型的能力进一步增强, 弧不再是信息传递的唯一方式。它引入两类对象, 并没有丢失数据流模型在并行性的开发上所拥有的优势。处理机对象的 pactors 和传统数据流图中的结点完全等价, 实现了数据并行。状态对象实现了信息共享, 不同状态对象的 sactors 之间可以并行执行, 并且其内部也可并行执行。

2 OOLGDFM 在 OOCPCS 中的实现

OOCPCS 基于 OOLGDFM 模型, 为网络并行处理提供易于编程的环境。并行模型中的 actors 在 OOCPCS 中由并行对象的外部操作实现, 这一部分我们讨论模型的实现。

(1) 对象, Actors 和 Tokens

在我们的系统中, 存在着三类对象: 普通 C++ 对象, 处理机对象, 状态对象。处理机对象和状态对象同并行模型中所定义的对象是相同的, 它们统称为并行对象。每一个并行对象只有一个独立的控制线程。一个对象根据输入值及本身值操作计算, 产生某个其它对象作为输出。对象操作的真正实现可以频繁利用其它对象及操作, 但是这些细节对对象外部是不可见的。

每一个 actor 由某一个对象的一个外部操作实现。处理机对象是一个不访问其它对象、拥有一个或多个操作的并行对象。一个状态对象是一个可能访问其它并行对象的并行对象。Pactors 和传统数据流图中的结点一样。Sactors 实现此并行模型的数据流子图。

一个并行对象名有三个组成成份: Objct - class, perssrstent 和 ID. Object - class 定义对象的功能。Persrstent 是一个布尔型的域, 它表示此命名对象是处理机对象还是状态对象。ID 在状态对象名中是全局唯一的、有别于其它对象的标识。处理机对象名不包含 ID 域。

一个 actor 的名字是一个 tuple< object-name, operation, computation# > . Operation 说明了名为 object-name 的对象的一个外部可见操作; 对于 pactor, computation# 是一个特殊 Pactor 的全局系统唯一标识; 对于 Sactor, Computation# 定义了此 actor 作为其对应状态对象的子计算的执行序列数。它定义了流图中标志结点的唯一计算实体, 没有它, 就很难保证针对于一个对象的不同计算实体的输入不相混淆。

一个弧名也是一个 tuple< actor-name, argument# > . Argument# 域表示此命名 actor 所需 tokens 将要匹配输入弧中的某一条弧。

(2) 程序流图

我们已经孤立地讨论了对象, 操作和 actor. 现在我们要说明 actors 之间的连接如何表示, 以及数据流图和程序子图的形成。OOCPCS 提供一个 Forward-List 结构描述动态构造的数据流图。Forward-list= {future-forward, sub-forward}

在程序流图中每个 actor 结点有一个 Forward-list 结构, 它用来表示 actor 自已及连接的信息。此结构由 Future-forward 和 sub-forward 两部分组成。Future-forward 是一个等待接收此结点输出 Tokens 的目标结点对应弧的集合, 因此, 这些目标结点都依赖于此 actor 的计算结果。Sub-forward 表示了 actor 本身的计算子图。事实上, 一个 actor 本身可能也构造新的、自己的 forward __ list 结构, 并且启动这个带有构造的 forword-lists 结构的新子图。这个子图完全对对象的使用者透明。它由一个或多个 forward-list 组成。

Forward-list 的使用允许图控制被完全分散化。每个 actor 接收到足够的相关联的程序图信息便可以继续计算。组合单独的、分开的子图执行是没有必要的, 它们的计算完全可以独立处理。

3 OOCPCS 系统

OOCPCS 采用了并行编译技术识别 C+ + 程序中的可并行对象。并行编译器抽取处理机对象和状态对象两种并行对象。由于类抽取了对象的公共属性和操作, 因此, 编译器通过分析类来识别并行对象。首先, 编译器分析每个类的成员变量和成员方法的特性, 接着, 它根据类的属性及类粒度, 来判断一个类是否是并行类。在面向对象技术中, 对象封装了数据及操作, 我们认为对象内所使用的数据块大小可以近似地表示其计算复杂度。因此, 在 OOCPCS 中, 对象粒度被近似估算为对象使用的数据块大小, 并行编译器采用了基于对象的程序划分算法, 将程序划分成相互合作, 独立并行计算的对象。

此外, OOCPCS 提供一个并行 C+ + 语言 POC, 以支持程序设计者编写高效率的并行程序。POC 语言在四个方面扩展了 C+ + 语言[6].

(1) 在类描述中增加了关键字 POC-PRO. POC-STAT;

(2) 支持并行类间继承性。POC 也支持多继承、动态联编和重载等特性;

(3) 在类定义中的预定义函数 main( );

(4) 隐式的子图的产生。

这些扩展使用户易于利用数据驱动计算模型的能力, 进行并行处理。一个预处理器将 POC 程序翻译成带有系统提供的运行库的 C+ + 程序。

预处理器为每个并行类创造二个标准 C+ + 类。一个是代理类, 另一个是服务器类。

代理类在原始程序中使用,由一套“虚假”(dummy)过程实现。这些过程管理 actors 的产生。它们产生 Foward-list 结构,负责和服务类通讯。预处理器将每个并行类的实现定义作为一个独立程序看待。每一个并行对象使用一个进程模拟实现。一个程序员可能为状态类定义一个 main 过程。一旦此类的对象初始化,对象的 main() 过程就被启动执行。

预处理的一个主要用途是检测 actors 间的数据流。当一个 actor 的结果作为另一个 actor 的输入或者激发另一个 actor 时,则此两个 actors 之间存在着数据流动。预处理器检测数据流,并构造 forward-list 结构以表示数据流图。

另外,OOCPCS 提供一个面向对象的文件 I/O 子系统,该子系统具有面向对象高层次接口,高性能及易于扩展、维护的文件 I/O 对象,并且具有易操作的网络文件 I/O 对象。用户可以提供启发式信息以便于 I/O 系统以较好的方式存储或预取数据而获得高性能的 I/O. 程序员也可以使用申请一释放策略操作远程文件对象。系统为网络并行处理提供了三类可操作的网络文件对象(详述见[7]). OOCPCS 采用了网络控制机制设计与面向对象文件 I/O 系统设计分离化技术,以使它们相互独立,增强了面向对象文件 I/O 系统独立性,移植性和可重用性。

4 OOLGDFM 抽象机

执行面向对象大粒度数据流模型 OOLGDFM 抽象机已经在局部网络上实现了。在这个网络上的每一台计算机都执行一个 OOLGDFM 抽象机(见图 1)

匹配单元的功能有两个方面。一个方面要确定哪一个 actor 就绪执行,并确保一旦 actor 被允许执行,它们应当被点火,同时,将所需的 tokens 传递给它们;另一方面,当一个新消息到达一个 actor 时,匹配单元根据这个 actor 的 computation# 确定是否那个到达的消息可以允许此 actor 点火执行。一旦 actor 可以点火,匹配单元将它放入计算单元中的就绪队列,并将其对应对象 P/V 域置为阻塞状态,以确保一个对象的两个 actor 不会同时在就绪队列中。

Token 存储单元负责确保 tokens 被传递到正确的 actor,它接收、存储并传扩 tokens,它和匹配单元紧密合作。

对象存储单元管理对象存储。所有对象由线程或进程实现。在 OOCPCS 中,用进程模拟实现并行对象,每一个对象有一个相连的消息队列。

更新单元负责向前传递计算结果到 forward-list 结构定义的 actors 中去,它沿着 futureforward 定义的那些弧发送结果消息。

计算单元负责执行已经允许执行并在就绪队列中的 actors. 它检查就绪队列,挑选一个 actor 执行。计算单元从就绪队列中接受一个 actor 的地址及 actor 所需数据 tokens,接着执行此 actor,直至结束。一旦 actor 完成执行,计算单元通知更新单元,并重新检查就绪队列。计算单元在一时刻可能执行多个 actors.

5 总结

本文描述了一个面向对象的并行模型和基于此模型的并行 C++ 系统。该系统采用并行编译器技术,提供并行 C++ 语言 POC 支持用户编写高效的并行程序。采用面向对

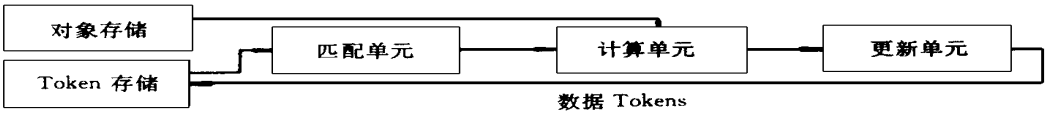

图 1　虚拟的面向对象大粒度数据流模型

象技术设计了一个面向对象高层次、高性能、易于扩展的文件 I/O 对象, 以及易于操作的网络文件 I/O 对象机制。最后, 提出了 OOLGDFM 抽象机。在体系结构方面支持 OOCPCS 的程序执行。有关系统的测试结果请见[8]。

参 考 文 献

1　Wegner P. Dimensions of Object __Based Languages Design. In: ACM OOPSLA´87 Proceeding, USA. Oct, 1987: 21～35

2　Davis A L, Keller R M. Data Flow Program Graphs. IEEE Computer, 1982, 15(2): 31～41

3　Arvind, Gostelow K P. The U-Interpreter. IEEE Computer, 1982, 15(2): 42～50

4　Robert G. Parallel Processing with Large Grain Data Flow Techniques. IEEE Computer, 1984, 17(7): 55～61

5　Grimshaw A S, Liu J W S. MENTAT: an Object-Oriented Macro Data flow System. In: OOPSLA´87 Proceedings. 1987

6　Stroustrup B. The C++ Programming Language. Addison. Westey, 1986

7　肖侬, 胡守仁. 网络的面向对象 I/O 系统开放式分布系统, 南京, 1995

8　肖侬. 基于网络环境的面向对象语言 C++ 的并行化技术的研究与实现: [博士论文]. 长沙: 国防科学技术大学计算机系, 1996

(上接第 43 页)

参 考 文 献

1　张世平. 普通话全音节识别系统: [博士论文]. 清华大学, 1988

2　易克初. 普通话全音节识别系统及语音识别与合成若干新问题的研究: [博士论文]. 清华大学, 1989. 3

3　王跃科. 机器人语音通信的理论与技术研究: [博士论文]. 华中理工大学, 1989

4　易克初, 王跃科. 有序码书矢量量化及其在语音识别中的应用, 语音图像通讯信号处理学术会议, 1989

5　陈尚勤, 罗承烈, 杨雪. 近代语音识别. 电子科技出版社, 1991

6　Tokura Y A. Weighted Cepstrum Distance Measure for Speech Recognition. IEEE Trans. 1987, ASSP-35(10)

7　Yi K H, Wang Y K. Sequential VQ and its Application to Chinese Dictation Recognizer. ICASSP-90

(责任编辑　潘　生)