分解变换 并行化编译 阅象重建 应用程序

计算机科学1998 Vol. 25№. 1

一种基于分解变换的并行化编译新技术*)

A New Technology of Split-based Transformation in Parallelization Compile

陈清萍 李晓峰 郑世荣

(中国科学技术大学计算机系 合肥230027)

摘 要 This paper discusses a new split-based transformation and analyzes an example testifying the advantage in application.

关键词 Parallelization compile, Loop-based transformation, Split-based transformation

1 引言

并行变换是并行化编译过程中的重要组成部分,它对源程序进行等价重构,使其获得更多并行机会。传统的并行变换技术主要侧重于循环并行性的开发,利用各种循环并行技术来将循环重构为具有更多并行机会的形式。常见的循环变换技术有:循环交换,循环分布,循环熔合(loop fusion),循环剥离(loop peeling)等等。由于循环变换仅针对循环进行,因而忽略了存在于循环之外的潜在并行性的开发。当处理机个数较少时,仅利用循环迭代间的并行性就可以获得运行性能很好的目标码,即使进一步开发潜在并行性,其性能改善也不显著。但当处理机个数较多时,仅利用循环并行无法使所有处理机处于"忙"状态,处理机的利用率不高。

为了进一步开发循环外的并行性,本文讨论了一种基于分解策略的并行变换新技术,不仅可以用来将循环重构为具有更多并行性的形式,而且可以用于程序中任一对有依赖关系的结点,暴露出结点操作间的并发性。实验表明,采用分解变换技术优化后可以获得较高的并行效益。

2 分解变换技术

2.1 设计思想

假设有两个计算结点B和A,其中A依赖于B,在未进行分解变换之前,其执行顺序是确定的,即B→A串行执行。利用分解变换可以将A中子计算划分为三个集合:A_I,A_D,A_M。A_I中的计算与B数据无关;除了与集合A_I中子计算相关的计算外,A中其他计算都属于A_D;未被A_I和A_D包括的计算属于A_M,此外为了保持原有的语义而添加的计算也放在A_M中。这样,原来的串行代码就可变成部分并行的形式。A_I中的计算可以与B并行执行,如图1所示。

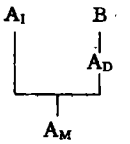

图1 利用分解变换得到的并行形式

对于迭代间相关的循环进行分解变换时,可将第i−1次迭代看作计算结点B,将第i次迭代看作计算结点A,变换的结果可将循环体中不需要依赖上次迭代计算结果的部分分离出来,这部分计算无需等待前一个迭代完成,可以与其并行执行。

2.2 算法

分解变换算法首先将结点A中的计算划分为若干基本计算,基本计算的选择决定了分解的粒度。根据这些基本计算与结点B中计算的相关性,将其分为三类:

(1)Bound:与B数据相关。

(2)Linked:与B数据无关,但与Bound类或其他Linked类基本计算相关。Linked类计算又可细分为:①Needsbound:与Bound类或其他Needsbound类基本计算流相关。②Generatelinked:与Bound类、Needsbound类或其他Generatelinked类基本计算流相关。③Readlinked:其他Linked类计算,与Generatelinked计算流相关。

(3)Free:其他。

在判断了子计算的类别后,可将其分别放入如上所述三个集合A_I,A_D,A_M中。由定义可知,Free类的计算与B完全无关,可以与B并行执行,故将其

*)本课题得到国家自然科学基金的资助

放入 A_I 中。由于 Bound 类或 Needsbound 类基本计算必须等待 B 中计算和 Generatelinked 类计算执行完后方可执行，如果将 Generatelinked 类计算放入 A_I 中，则应将 Bound 类或 Needsbound 类基本计算放入 A_M 中，此时 Readlinked 类计算放入 A_I 或 A_M 中均可；如果在 A_I 和 A_D 中各放入 Generatelinked 类计算的一个副本，此时可将 Bound 类或 needs-bound 类基本计算放入 A_D 中。在实际应用时应根据具体的情况决定何种方法的效率更高。

2.3 实例分析

为说明分解变换技术在实际应用中的效果，以下分析一个实例 PSIRRFAN，它是 UC/Berkeley 大学开发的用于重建二维图象的应用程序。为了构造物体的图象，首先利用射线发射器和探测器获得重建图象所需的数据，射线发射器从不同角度发出的射线穿过物体后，由探测器计量信号强弱。然而有些物体的某些角度不可能被探测到，例如物体的一部分被嵌入到其它物体中。此时获得的重建图象的数据是不完全的。PSIRRFAN 正是用于缺失数据情况下的图象重构。程序的结构化分解如下：

```
Reconstruction
    init
    outer reconstruction integal (for missing data column)
      init
      inner reconstruction integal
      Fourier filter
      data update
Backproject
```

程序中主要包括两个计算步：Reconstruction 利用已有数据计算出缺少的数据值，Backproject 利用数据构造图象。Reconstruction 是一个迭代算法，每个迭代步修改原先数据集中缺少的数据，直至这些值收敛为止。该算法的三个并行化版本是：(1)外层和内层 integal，Fourier filter 以及 Backproject 都可变换为可以并行执行的循环。这时程序中的循环并行性已被完全开发，如图2。现在利用分解变换技术可进一步开发程序中循环之外的并行性。(2)在版本1的基础上，利用 Recounstruction 与 Backproject 间的关系对 Backproject 进行分解。因为缺少的数据毕竟只是一小部分，故 Backproject 算法所需的大部分数据在 Reconstruction 算法执行前即可得。将 Back-projection 分解为两部分，一部分操作依赖缺失的数据，必须等待 Reconstruction 执行结束，另一部分不依赖缺失的数据可以与 Reconstruction 算法并发执行。如图3所示。(3)进一步对 Reconstruction 循环的循环体进行分解，将独立于前次迭代的计算分离出来。当 Reconstruction 算法的第 I 次迭代仍在执行时，就可开始执行下次迭代中的独立计算。如图4所示。

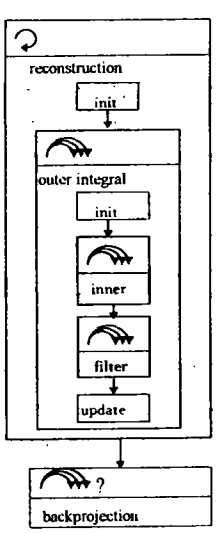

图2

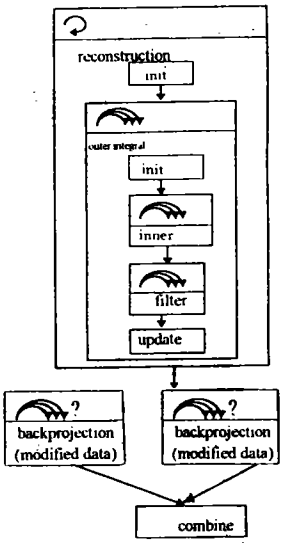

图3

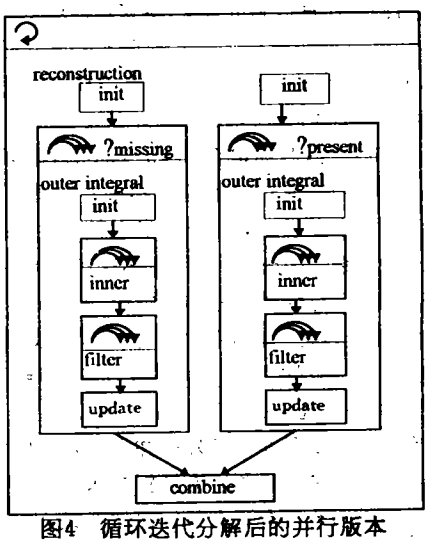

图4 循环迭代分解后的并行版本

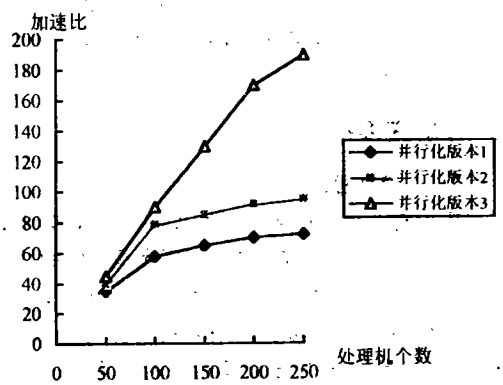

图5 采用分解变换优化前后的并行效益比较

软件体系结构 软件体系结构 软件开发

计算机科学1998 Vol. 25№.1

83-86

软件体系结构概念

The Concepts of Software Architecture

汪 琼

(北京大学计算机系 100871)

TP311.52

摘 要 Software architecture as a method of software reusability has been gotten more attentions now. To identify the definition and the research contents will help us do more research in this area.

关键词 Software architecture, Software reusability

在软件领域关于体系结构的提法可以追溯到1969年Fred Brooks提出的概念结构。他和Ken Iverson称体系结构是"程序员看到的一个计算机系统的概念结构",是对若干实际系统的概括。几年后Brooks定义体系结构为"用户界面完整详细的说明,对于计算机,这是程序设计手册,对于编译器,这是语言手册,对于整个系统,则是用户完成整个工作所用到的所有手册"。在此,体系结构和实现是两个不同的概念,体系结构告知什么事发生,而实现告知是如何发生的。这种区别至今仍旧存在,在面向对象的程序设计中更是如此。

尽管体系结构一词现在还有人是作为"一个系统的用户视图"来使用的,但是软件体系结构决不是一个系统的用户视图,而是那些对用户来说相对隐蔽的系统结构。"体系结构是一类系统的共同描述"仍旧是这个概念的中心。

软件体系结构是软件结构研究的继续。1968年,Edsger Dijkstra指出,为了产生正确的结果,相对于简单编程,应该更加注意软件是如何分解,如何结构化的。当时Dijkstra正在写一个操作系统,他提出层次结构的概念,即程序分层组织,一层中的程序只能与其相邻层的程序通讯。Dijkstra认为这样的组织所展示的优雅的概念完整性可以使开发和维护更为方便。

David Parnas发展了软件结构研究,他提出了信息隐蔽模块概念[1972]、软件结构概念[1974]和

图5显示了在256个处理机Ncube-2上重构一个二维图象时,上述三个并行化版本的性能。由图5可见,当处理机的个数小于50时,是否采用分解变换技术对加速比的影响不大,这是由于较少的处理机个数无法充分利用分解变换所开发出的额外并行机会。但当处理机的个数增加时,仅利用循环并行所获得加速比的增长幅度很小,这是由于循环并行无法使所有的处理机处于忙状态,处理机的利用率不高。在对程序使用了分解变换技术后,获得了进一步的性能改进。

结束语 在串行代码的并行化过程中,传统采用的各种循环变换技术旨在开发循环并行,因而忽略了许多可利用的并行机会,在其基础上进一步采用分解变换技术,可以获得更大并行效益。

参考文献

[1] Sharp O. J., Transforming for parallelism using symbolic summarization. PHD thesis, Department of Computer Science, University of California at Berkeley, 1995

[2] Allen J. R., Kennedy K., Automatic loop interchange. In: Proceedings of the SIGPLAN Symposium on Compiler Construction, 1984

[3] Ted G. L., Hesham E. R., Introduction to parallel computing. Prentice-Hall, Inc, 1992

[4] Zima H, et al., SUPERB: A tool for semi-automatic MIMD/SIMD parallelization. Parallel Computing, 1988, 6

[5] Hwang K., Advanced computer architecture: parallelism, scalability, progrommability. McGraw-Hill, inc, 1993

[6] Callahan D, et al., ParaScope: A parallel programming environment. Int. J. Supercomputer, Appl, 2 (4), 1988