共享内存结构 OpenMP 并行程序的自动生成*)

张 平 赵荣彩 李清宝 董春丽
(解放军信息工程大学信息工程学院 郑州450002)

摘 要 有效的程序自动并行化系统能帮助用户充分利用并行计算机的硬件资源和提高并行程序设计的效率。OpenMP 作为共享内存结构的编程标准,具有良好的性能和可移植性。本文介绍了基于 SUIF 的 OpenMP 并行程序自动生成工具 OAGT 的设计和实现,重点讨论了其中所涉及的几个主要技术问题:循环分析、流水并行、归约操作、同步优化等。

关键词 程序并行化,OpenMP,循环分析,流水并行,归约操作

Automatic Generation of OpenMP-Based Parallel Program on Shared Memory Architecture

ZHANG Ping ZHAO Rong-Cai LI Qing-Bao DONG Chun-Li
(The PLA Information and Engineering University, Zhengzhou 450002)

Abstract An automatic parallizer is helpful for users to exploit the resources of parallel computers and to improve the efficiency of parallel programming. OpenMP as the industrial standard for shared-memory programming has well performance and portability. This paper introduces the design and implementation of the SUIF-based OpenMP Automatic Generation Toolkit-OAGT and discusses some main techniques.

Keywords Program parallizer, OpenMP, Loop analysis, Pipeline parallel, Reduction

1 引言

并行程序往往要针对特定的并行计算机体系结构,只有在并行程序中充分体现机器的硬件结构特点,才能提高并行程序的执行效率,充分发挥并行计算机的性能。因此,并行程序设计要求程序员熟悉并行计算机的硬件结构,相对于串行程序设计而言,并行程序设计需要相当的经验,耗时长,难度大,易出错。

并行计算机可分为共享内存结构和分布内存结构两大类。近年来,成熟先进的技术使共享内存并行系统逐渐占据了并行计算领域的主导地位,如 SGI2000中应用的 cc-NUMA 结构,已经可使系统中的处理机数扩展至上百个。共享内存结构具有全局统一的地址空间,其上的并行程序设计无需考虑数据划分,通常采用编译指示来表示代码的并行,通过硬件缓存一致性来实现可扩展性[5]。以往,不同厂商的计算机定义了各自不同的编译指示,使得并行程序的可移植性很差,OpenMP[2]的出现改变了这一状况,作为共享内存编程的工业标准,OpenMP 在不降低性能的情况下克服了机器专用编译指示所带来的移植性问题,得到了各界的认可和接受。

OpenMP 标准提供了一种增量式程序并行化方法,在串行程序中适当地插入编译指示和运行时库函数调用可以在不改变原程序结构的情况下,将串行程序转变为并行程序。然而,OpenMP 标准并不保证并行程序语义的正确性,所以,这种转化并不一定会提高程序的性能,甚至在最坏的情况下,会带来错误的结果。要保证正确插入 OpenMP 编译指示需要依据应用程序的相关关系信息。为实现自动插入 OpenMP 编译指示将串行程序变为并行程序,我们设计了基于斯坦福大学的并行化编译器 SUIF(Stanford University Intermediate Format)[4]的 OpenMP 并行程序自动生成工具 (OpenMP Automatic Generation Toolkit,OAGT)。OAGT 利用 SUIF 的程序相关性分析功能和循环变换功能,来减少编译指示人工插入可能带来的问题,在无需用户干预的情况下,在 C 串行源程序中自动插入 OpenMP 编译指示,实现源—源重构,将 C 串行程序转变为 OpenMP 并行程序。OpenMP 并行程序可利用支持 OpenMP 的本地编译器编译生成不同机器上的可执行代码。本文主要讨论自动生成 OpenMP 并行程序的实现方法,即如何在 SUIF 相关性分析的基础上在串行程序中插入 OpenMP 编译指示生成语义正确的并行程序。

2 OpenMP 编程模型

OpenMP 是共享内存程序设计的工业标准,其目标是为 SMP 系统提供可移植、可扩展的开发接口。OpenMP 是对现有串行编程语言的扩展,通过编译指示和运行时库函数对 C、C++和 Fortran 语言进行扩展来支持共享内存并行。

OpenMP 使用编译指示与编译器通信说明并行结构、并行方式及数据的共享/私有等属性信息;OpenMP 实现的是线程级并行,线程间通信是隐含的,通过共享变量来实现。

OpenMP 遵循 fork-and-join 执行模式。程序初始化为一个轻量级进程(主线程)开始执行,当遇到并行结构(omp parallel),主线程创建一组线程并发执行并行结构中的语句。当遇到工作共享结构(如 omp for),工作负荷由线程组中的各个线程分担;OpenMP 提供各种同步指示(如 omp barrier,omp flush 等)用于表示线程间通信;OpenMP 的数据环境结构说明了并行区中的数据对线程的共享或私有信息;另外,归约操

*)基金项目:国防重点科研项目资助。张 平 博士研究生,研究方向为并行识别,并行编译。赵荣彩 教授、博士生导师,研究方向为并行编译,二进制转换,网络安全等。

作(如+，*等)可用 omp reduction 子句来说明。并行区结束,线程组中的线程同步,主线程继续执行。在程序执行过程中,可多次执行 fork-and-join 过程[1]。

OpenMP 具有良好的可移植性,其实现相对简单,充分利用了共享内存体系结构的特点,既可用于直接编写并行程序,更方便了串行程序的并行化,利用 OpenMP 将串行程序转变为并行程序无需对源代码做大的改动。

3 OpenMP 并行程序自动生成工具 OAGT

OpenMP 并行程序自动生成工具 OAGT 的设计目标是避免用户手工并行化程序的繁琐劳动和可能引起的错误,通过计算机处理自动生成具有良好可移植性的 OpenMP 并行程序。串行程序并行化的关键是检测程序的并行性,而检测并行性的关键是获得程序中精确的相关信息,因此,OAGT 的设计基于 SUIF 并行化编译器,系统结构如图1。OAGT 接受串行的 C 程序作为输入,调用 SUIF 的相关性分析模块获得精确的数据相关信息,利用循环变换模块对循环作适当的变换来增加可并行化循环,然后进行并行识别,通过循环分析和变量使用情况分析,构建和优化并行区,插入 OpenMP 编译指示,生成并行程序。

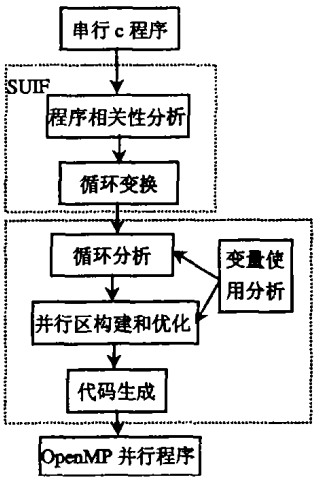

图1 OAGT 结构

自动插入 OpenMP 编译指示要完成的主要工作包括:

1. 遍历程序结构流图,进行循环分析,识别并行循环;

2. 遍历程序的数据流图,分析变量的使用情况,确定变量的共享和私有属性(shared,privated);

3. 基于并行循环构建并行区,并对并行区进行优化减少由 fork_and_join 和同步带来的开销;

4. 根据 OpenMP 标准转换代码、插入相应的编译指示,生成 OpenMP 并行程序。

4 自动生成 OpenMP 编译指示的主要技术

4.1 循环分析

OAGT 实现的是循环级的并行化,即 OpenMP 指示的并行是针对循环的。循环分析的过程主要依据 SUIF 程序相关性分析模块的分析结果对循环进行分类。SUIF 的相关性分析模块对源程序进行各种数据相关性测试,以相关距离向量和方向向量的形式给出了每一循环的相关信息,为循环分析提供了基础。循环分析将循环分为:并行循环(parallel)、串行循环(serial)和流水并行循环(pipeline parallel)。

*并行循环*指循环中不存在跨迭代相关(loop_carried dependence)且循环体中没有转入/转出语句(如 return, goto 等)和输入/输出(I/O)操作的循环。并行循环的各迭代具有并行性,可以被并行化。另外,并行循环还包括存在由于重复使用同一内存空间而带来的具有跨迭代输出相关的循环,这种情况可以通过变量私有化来消除伪相关,转变为并行循环。

含有归约操作(reduction)的循环被当作特殊的并行循环。因为归约操作可以通过循环划分,并发计算部分结果,然后自动更新最终结果来实现。

流水并行循环指存在跨迭代流相关(flow dependence)且相关距离为非0常数的循环,这种循环可用来形成流水并行结构,在 OpenMP 可以通过点到点同步来实现。

串行循环指循环中存在跨迭代相关(除流水循环和规约操作外),不可并行化,只能串行执行。

4.2 建立和优化并行区

建立并行区的目标是增大并行的粒度,主要完成:

·自上而下遍历程序的结构流图,静态评测并行循环的计算量,对达到粒度基准的并行循环构建并行区;未达到基准的循环标注为串行循环,不并行化。对于循环嵌套我们仅考虑最外层并行循环,不支持嵌套并行。在同一级没有并行循环的情况下考虑流水并行循环,构造流水并行结构。

·扩展并行区,将并行区前面满足内存访问冲突测试且没有包含在其他并行区中的代码块合并到并行区中。

·合并相邻两个并行区,形成一个较大的并行区,增大并行粒度,减少由 fork_and_join 带来的开销。

4.3 流水并行实现

流水并行循环可以通过分析循环的相关向量来确定:如果循环存在跨迭代相关关系且对应的相关距离向量为常数,则将该循环确定为流水并行循环。流水并行循环可以通过点到点同步来实现流水并行,我们利用 OpenMP 提供了编译指示(omp flush)和库函数用来表示流水并行。这些编译指示保证只有流水化代码段正确执行所需的数据可用,线程才执行相应的代码段。软件流水并行要求调度方式为 STATIC 和 ORDERED。图2给出了一个流水并行循环的例子,图3给出相应的 OpenMP 流水并行实现。为减少同步的频率,将循环迭代分块,这里设分块的大小为4。

```
for(i=1;i<100;i++)
        for(j=1;j<100;j++)
                a[i][j]=a[i-1][j]+a[i][j-1];
```

图2 流水并行循环的串行代码

```
int sync[NUMBER_OF_THREADS],
int counter,iam;
float a[100][100];
# pragma omp parallel private (iam, neighbor, counter) shared (a,
sync)
{
   counter=0;
   iam=omp_get_thread_num();
   sync[iam]=0;
   #pragma omp barrier
   neighbor=(iam>0?iam:omp_get_num_threads())-1;
   for(j_tile=1;j_tile<100;j_tile+=4)
   {
        if((iam>0)&&(iam<omp_get_num_threads()))
        {
          while(sync[iam-1]<counter);
          counter+=1;
        }
      #pragma omp for schedule(static)nowait
           {
              for(i=1;i<100;i++)
              {
                 tmp2=max(1,j_tile);
```

```
            tmp3=min(100,3+j_tile);
            for(j=tmp2;j<tmp3;j++)
            {
                a[i][j]=a[i-1][j]+a[i][j-1];
            }
        }
        #pragma omp flush(a);
        sync[iam]+=1;
        #pragma omp flush(sync);
    }
}
```

图3 流水并行的 OpenMP 实现

4.4 同步优化

OpenMP 同步用来保证并行结构执行结束后程序执行的正确性，在默认情况下，在每个并行结构的结束都应有一个隐含的同步(barrier)，因此，我们在每个并行循环的结束处增加一个同步。但在一些情况下，两个并行循环间同步可以消除以减少开销。

两个并行循环间的同步可消除，即两个并行循环能够异步并行，必须满足下面的条件：执行一个循环的任一线程不会读写由执行另一循环的任一线程所读写的数据。因此，对于非私有化数组，须验证第一个线程的写操作数组下标集与第二个线程读写同一数组下标集不相交，即两个循环要满足 Bernstein 条件[6]。如果 Bernstein 条件可证明为真，这两个并行循环可异步执行，循环间的同步可消除，如果不能证明为真，则两个循环间的同步不能消除。

为减少复杂性，我们假设执行两个并行循环的线程数相同，线程数大于1，且在循环中有数组读写操作，测试过程分两步：

·检查循环的迭代数：假设循环采用相同的调度方式，线程数可以是任意的，因此每个循环迭代数必须相同。如果不能证明两个循环的迭代数相同，则将 Bernstein 条件置为假。

·数组下标比较：对于一个循环中对一个非私有化数组的左值引用和在另一个循环中对同一数组的每一引用进行数组下标比较。如果不能证明至少有一维数组下标不同，则 Bernstein 条件为假。

4.5 变量使用情况分析

变量使用情况分析的目的是确定循环中的变量相对于每个循环迭代是私有的还是共享的，确定变量的共享/私有属性对于提高程序并行性和并行程序的正确执行起重要作用。私有变量的判定通过分析循环中变量的相关性来确定。如果循环中存在由于内存访问冲突而引起的跨迭代输出相关，则可以通过将变量私有化来消除这种伪相关，使得并行执行时每个线程有该变量的一个本地拷贝。不能私有化的变量称为共享变量，循环中的代码只能串行执行(除规约操作)。

在 OpenMP 中私有变量通过并行结构或共享结构中的 omp private 子句来声明。如果私有变量需要从并行区外获得初始值，则用 omp firstprivate 子句指示获得初始拷贝；如果私有变量在并行区结束后还要使用，可用 omp lastprivate 来更新共享值。

4.6 归约操作处理

归约是计算中经常遇到的一种操作，归约的定义如下[7]：

假设 S 为语句，a 为变量(标量或数组元素)，e 为表达式，$\oplus$为运算符，若下列条件成立：

1. $\oplus$为具有幺元且满足结合律和交换律的运算符；

2. S 具有或经变换后具有形式：$a=a\oplus e$。

则称语句 S 具有归约形式，$\oplus$为归约运算符，a 为归约变量，e 为归约表达式。

归约定义了程序中大量出现的一个基本现象——归约操作：不同的循环实例对某一存储单元进行一系列可结合且可交换的数据操作，操作的结合性和交换性保证了循环迭代可乱序执行，归约变量的最终结果不变，但循环不可并行执行。因此可以通过设置临界区获得一定的并行性，但其并行度小于可完全并行的循环。

目前 OAGT 可识别标量和数组的+，* 等归约操作。标量的归约操作在 OpenMP 中直接利用 reduction 子句表示；因为 OpenMP 不支持数组归约操作，我们通过循环划分、设置临界区来实现。图4、5给出了 OpenMP 实现数组归约的一个例子。

```
……
for(j=0;j<N;j++)
    s[i]=s[i]+a[i][j];
……
```

图4 可实现数组归约的串行代码片断

```
……
#pragma omp parallel private(local)
{
#pragma omp for
  for(j=0;j<N;j++)
    local=local+a[i][j];
#pragma omp critical
  s[i]=s[i]+local;
}
……
```

图5 用 OpenMP 实现数组归约操作

下一步工作 目前，我们已实现了 OAGT 的主要功能，对输入的串行 C 程序进行分析处理，识别并行循环，自动插入 OpenMP 编译指示，生成并行源程序。生成的 OpenMP 并行程序已通过 OpenMP 编译器编译，生成可执行程序。为了获得更好的并行性能，下一步我们打算实现一个可视化用户界面，向用户提供系统执行过程中的各种分析和转换结果：依赖关系分析结果、并行化转换的结果、编译指示的插入情况等，可以使用户集中精力于可以改进性能的区域，如消除伪相关，用户可以通过可视化界面和系统交互提供辅助信息帮助系统实现更高程度的并行化和 OpenMP 指示的插入。

参 考 文 献

1 OpenMP C Application Program Interface Version 2.0. Nov. 2000. http://www.openmp.org/

2 OpenMP. The OpenMP ARB. http://www.openmp.org/

3 Allen R, Kennedy K. Optimizing compliers for Modern Architectures. Morgan Kaufmann, 2002

4 Wilson R, French R, Wilson C, et al. SUIF: An Infrastructure for Research on Parallelizing and Optimizing Compilers. ACM SIGPLAN Notices, 1994, 29: 31～37

5 [美] Hwang K 著. 王鼎兴，等译. 高性能计算机系统结构——并行性 可扩展性 可编程性. 北京：清华大学出版社，1997

6 Koelbel C H, et al. The High Performance Fortran Handbook. MIT Press, 1994. 193

7 李剑慧，等. 归约识别及其单模变换. 计算机学报，1998(1)