软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2012,23(10):2695−2704 [doi: 10.3724/SP.J.1001.2012.04178]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

基于嵌套循环分类的并行识别技术*

赵 捷+, 赵荣彩, 丁 锐, 黄品丰

(解放军信息工程大学 信息工程学院,河南 郑州 450002)

Parallelism Recognition Technology Based on Nested Loops Classifying

ZHAO Jie+, ZHAO Rong-Cai, DING Rui, HUANG Pin-Feng

(Institute of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

+ Corresponding author: E-mail: zjbc2005@163.com

Zhao J, Zhao RC, Ding R, Huang PF. Parallelism recognition technology based on nested loops classifying. *Journal of Software*, 2012,23(10):2695−2704 (in Chinese). http://www.jos.org.cn/1000-9825/ 4178.htm

Abstract: Existing distributed memory parallelizing compiler systems are mostly developed based on shared systems. The parallelism recognition technologies of shared memory parallelizing compiler systems are suitable for OpenMP code generation. Their implementation is used to recognize all nested loops by the same technology, so that the parallelism cannot be efficiently explored when applying them to distributed memory parallelizing compiler systems. Thus, this paper proposes some parallelism recognition technologies suitable for the MPI code generation for distributed memory parallelizing compiler systems by classifying the nested loops according to their structures. To solve these problems, a new classification method of nested loops is proposed, according to the structure of nested loops and characteristics of MPI parallel program. Corresponding parallelism recognition technologies for different nested loops are also presented, respectively. The experimental results show that compared with the distributed memory parallelizing compiler systems that used existing parallelism recognition technologies, the compiler systems, which use the proposed classification method and the corresponding recognition technologies, can more efficiently recognize parallel nested loops in the benchmark programs, and the performance speedup of the MPI codes automatically increased to more than 20%.

Key words: parallelizing compiler; parallelism recognition; nested loops; model algorithm; traverse algorithm; interaction algorithm

摘 要: 传统的分布存储并行编译系统大多是在共享存储并行编译系统的基础上开发的.共享存储并行编译系统的并行识别技术适合 OpenMP 代码生成,实现方式是将所有嵌套循环都按照相同的识别方法进行处理,用于分布存储并行编译系统必然会导致无法高效发掘程序的并行性.分布存储并行编译系统应根据嵌套循环结构的特点进行分类处理,提出适合 MPI 代码生成的并行识别技术.为解决上述问题,根据嵌套循环的结构和 MPI 并行程序的特点,提出了一种新的嵌套循环分类方法,并针对不同的嵌套循环分别提出了相应的并行识别技术.实验结果表明,与采用传统并行识别技术的分布存储并行编译系统相比,按照所提方法对嵌套循环进行分类,采用相应并行识别技术的编译系统能够更高效地识别基准程序中的并行循环,自动生成的 MPI 并行代码其性能加速比提高了 20%以上.

关键词: 并行编译;并行识别;嵌套循环;模型法;遍历法;交互法

* 基金项目: “核高基”国家科技重大专项(2009ZX01036-001-001-2)
收稿时间: 2011-07-26; 修改时间: 2011-11-02; 定稿时间: 2012-01-16

中图法分类号: TP311 文献标识码: A

当今世界上主流的高性能计算机系统大多具备节点间、节点内及 SIMD 短向量部件等多层次并行计算资源,如何使用并行编译系统自动发掘应用程序中蕴含的不同层次、不同粒度的并行性[1],已成为提高高性能计算机系统应用性能的一项关键技术.近年来,针对串行程序 SIMD 向量化和节点内的并行开展了大量的工作,相应的技术也已经发展得比较成熟,如:Stanford 大学研制的自动并行化编译器 SUIF[2],能够很好地识别串行程序中的完美嵌套循环,但是对于非完美嵌套循环的并行识别能力仍然有限;中国科学院和 Delaware 大学维护开发的 Open64[3]编译器,在识别并行循环之前通过大部分循环优化将循环转换为完美嵌套循环,由此提高识别并行循环的能力.

而并行编译在分布存储结构计算机上的发展却相对较慢,针对节点间的并行编译技术也没有取得太大的进展.目前,节点间并行化编译系统大多是在节点内并行化编译系统的基础上开发的,如:Kwon 等人基于 SUIF 编译器实现了面向 MPI 并行的后端[4],依靠局部插桩技术[5]来分析原来 SUIF 后端生成的代码的信息,但是没有改变 SUIF 中识别并行循环的工作;Ferner 的 Paraguin 工具[6]基于 SUIF 编译器实现了在已知计算划分前提下 MPI 通信的求解算法,但 Paraguin 工具也没有把工作重点放在并行识别上,而是在后端通过符号不等式系统完成了并行代码的自动生成.

上述分布存储并行编译系统的效率都不是十分理想,主要原因有以下两个方面:首先,节点内并行化编译系统实现的是面向共享存储结构的代码生成,其并行识别技术主要针对适合 OpenMP 并行的循环.将这些并行识别技术用于面向 MPI 代码生成的分布存储并行编译系统,在处理部分非完美嵌套循环时不能很好地挖掘程序并行性;其次,传统的并行识别技术本身也存在弊端,其实现方式是将所有嵌套循环都按照相同方式处理,导致识别出的并行循环不够精确,并行程序效率有所下降.

为解决上述问题,根据嵌套循环结构的特点,针对传统并行识别技术存在的不足,提出了一种新的嵌套循环分类方法,将嵌套循环分为完美嵌套循环、简单嵌套循环和一般嵌套循环,并针对这 3 种嵌套循环分别提出了相应的并行识别技术.实验测试将这些并行识别技术应用于 Open64 编译器,并与 Kwon 的工具和 Paraguin 工具相比,结果表明,这种分类方法能够更有效地识别程序中的并行循环,利用相应识别技术生成的 MPI 并行代码的效率可提高 20%以上.

1 一种嵌套循环的分类方法

应用程序中的嵌套循环形式多种多样,其结构决定了是否适合并行执行.传统的并行识别方式很少将嵌套循环分类处理,导致并行性无法得到有效的发掘.实际上,针对不同类型的嵌套循环采用不同的处理方法,可能会更高效地发掘程序的并行性.

循环并行性的发掘,实质上就是测试循环中是否存在依赖.依赖可以分为循环无关依赖和循环携带[7]依赖,循环中数组下标的引用构成依赖的方向矩阵[7]是循环交换等优化的基础.由于向量化的基本单位是循环中的单条语句,SIMD 向量化需同时考虑循环无关和循环携带依赖,所以,SIMD 向量化的优化,是将嵌套循环中可以向量化的循环尽量移动到最内层位置.而并行化的基本单位是循环,只需要考察循环携带依赖,因此,其方法是将嵌套循环中可以并行化的循环尽量移动到最外层位置.嵌套循环的结构决定了循环交换之后是否引起语义的改变,因此可以按照不同嵌套循环的结构将嵌套循环分为一般嵌套循环、简单嵌套循环和完美嵌套循环.本文能够识别的嵌套循环称为一般嵌套循环.在定义一般嵌套循环之前,首先引入非规则数组区域访问的定义.

定义 1. 非规则数组区域访问[8]是指无法在编译阶段确定数组元素存储地址的访问.

例如,以数组作为下标的数组访问就属于非规则数组区域访问.目前的依赖测试方法只能测试线性数组下标,无法处理非规则数组区域访问的非线性数据下标.

定义 2. 对于 N 层嵌套 for 循环的第 $k(0<k\leqslant N)$层循环,如果循环体内不存在非规则数组区域访问,那么该层

循环就是一般嵌套循环(ordinary nested loop,简称 ONL).

如果一般嵌套循环的循环体内没有导致在循环迭代结束之前就使程序跳出循环结构的语句,那么这个一般嵌套循环就是简单嵌套循环.

定义 3. N 层嵌套 for 循环的第 $k(0<k\leqslant N)$层循环是简单嵌套循环(simple nested loop,简称 SNL),当且仅当该循环的循环体满足:(1) 不存在非规则数组区域访问;(2) 无跳转、过程调用和过程返回语句.

传统的嵌套循环的分类方法是将嵌套循环分为完美嵌套循环和非完美嵌套循环,可以通过在简单嵌套循环的定义上限定更严格的条件来获得完美嵌套循环.

定义 4. 如果一个 N 层嵌套 for 循环只有最内层循环的循环体由一条或多条非循环语句构成,并且这些非循环语句满足:(1) 不存在非规则数组区域访问;(2) 无跳转、过程调用和过程返回语句.其他所有循环的循环体只由一条 for 循环语句构成,那么该嵌套循环的每层循环都是完美嵌套循环(perfect nested loop,简称 PNL).

目前,编译器中的识别方法还无法直接处理 while 和 do while 等类型的循环.对于这些循环,编译器的识别处理方式是将其规范化为 for 循环,然后再进行并行识别.

下面分别介绍针对不同嵌套循环采用的不同并行识别方法.

2 完美嵌套循环的并行识别

例 1 所示是一段完美嵌套循环的代码,从例 1 可以看出,完美嵌套循环的结构非常严格和紧凑,因此也有学者称其为紧嵌套循环.

例 1:

```
for (i=1; i<=N; i++)
  for (j=1; j<=M; j++)
    for (k=1; k<=L; k++){
      A[i+1,j,k]=A[i,j,k]+x1;
      B[i,j,k+1]=B[i,j,k]+x2;
      C[i+1,j+1,k+1]=C[i,j,k]+x3;
    }
```

Cohen 等人提出了广泛应用于并行编译循环优化中的 Polyhedral 模型[9],这种模型把程序中的循环边界等信息抽象成空间多面体模型,通过多面体的变换实现循环的优化.

识别程序中的完美嵌套循环可以借助 Cohen 等人提出的 Polyhedral 模型,他们在静态控制流嵌套(static control nests)[10]基础上扩展定义了 SCoP(static control part).SCoP 是构建 Polyhedral 模型的基础.

Cohen 等人将程序中的循环分为两种类型:第 1 种循环迭代从 0 开始,到某种边界表达式结束,且步长为整数,这种循环称作 do 型循环;第 2 种循环称作 while 型循环,是指所有不满足上述条件的循环.

定义 5. 循环内除 while 型循环语句外构成的最大连续语句集合,称为该循环的 SCoP.

当 SCoP 中至少含有一个非空 do 型循环(循环体内有语句)时,称该 SCoP 是饱满的(rich).图 1 是一段代码中的 SCoP 示例.

模型法:Polyhedral 模型是目前较为流行的高级编译器实现循环优化的理论,本文选择 Open64 编译器实现识别完美嵌套循环的方法.虽然到目前为止,在官方发布的 Open64 编译器中仍没有 Polyhedral 模型的代码,但已有一个比较稳定的基于 Open64 的 Polyhedral 模型工具 WRaP-IT(Whirl represented as Polyhedral)[9,11].因此,可以借助 WRaP-IT 工具识别程序中的完美嵌套循环.由于该算法借助 Polyhedral 模型,因此称为模型法.算法过程如图 2 所示.

这样,经过上述过程可以识别出程序中的完美嵌套循环,并且 Polyhedral 模型可以将一些非完美嵌套循环转换为完美嵌套循环.仍考虑例 1,如果仅仅依靠简单的分析识别其中的并行循环是比较困难的,通过模型法分析后,在串行执行 i 循环后,将其中的 j 循环识别为并行循环,从而迫使 k 循环串行执行.

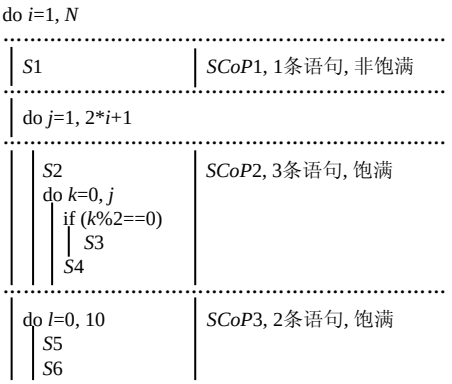


Fig.1 Example of ScoP

图 1 SCoP 示例

```
1  //功能:基于Polyhedral模型识别完美嵌套循环
2  //输入:需要进行并行识别的嵌套循环L;
3  //输出:parallel;当parallel为TRUE时表示该循环为可并行循环,为FALSE时表示为串行循环.
4  procedure Polyhedral_Model_Algorithm(L)
5    获取嵌套循环L的层数N;
6    识别嵌套循环L中的SCoP;                          // SCoP是构建Polyhedral的基础
7    记SCoP的个数为num,并设i=1;
8    parallel=FALSE;
9    if num!=0 then begin                             // 无SCoP
10     while i<=num do begin
11       构建与该SCoPi相对应的Polyhedral模型;
12       if 构建Polyhedral模型成功 then begin
13         利用WRaP-IT分析和转换生成的Polyhedral模型;   // WRaP-IT是一个比较稳定的基于
14                                                     // Open64的用于分析和转换
15                                                     // Polyhedral模型的工具
16         从Polyhedral模型再生成Open64的中间代码WHIRL;
17         标记SCoPi对应层循环的类型为pnl;             // pnl代表完美嵌套循环
18       end
19       i++;
20     end
21   end
22   if 嵌套循环L的类型为pnl then begin
23     设flag=Dependence_Test(L);                     // Dependence_Test依赖测试函数
24     if flag==TRUE then                             // 有循环携带依赖
25       parallel=FALSE;
26     else                                           // 无循环携带依赖
27       parallel=TRUE;
28   end
29   return parallel;
30 end Polyhedral_Model_Algorithm
```

Fig.2 Model algorithm

图 2 模型法

3 简单嵌套循环的并行识别

完美嵌套循环是最理想的嵌套循环,而非完美嵌套循环在应用程序中更常见.模型法可以用来识别完美嵌套循环和那些能够被转换为完美嵌套的循环;但是对于无法转换为完美嵌套的循环,模型法并不适用.如,例 2 所示是 NPB 测试集中 EP(embarrassingly parallel,密集并行)程序代码片段:

例 2:

```
do 140 i=1, nk
  x1=2.d0*x(2*i–1)–1.d0
  x2=2.d0*x(2*i)–1.d0
  t1=x1**2+x2**2
  if (t1.le.1.d0) then
    t2=sqrt(–2.d0*log(t1)/t1)
    t3=(x1*t2)
    t4=(x2*t2)
    l=max(abs(t3),abs(t4))
    q(l)=q(l)+1.d0
    sx=sx+t3
    sy=sy+t4
  end if
140 continue
```

从例 2 中的循环结构特点不难看出,*i* 循环为简单嵌套循环,模型法对于这种循环的识别并不适用.

简单嵌套循环在一定条件下可以转换为完美嵌套循环.这里引入一个夹层代码的定义.

定义 6. 当嵌套循环的两层循环进行置换时,引起循环依赖关系发生改变的语句集合称为夹层代码(sandwiched code).

定理 1. 在已知嵌套循置换条件的情况下,对于 *N* 层简单嵌套循环,置换其中任意两层循环时没有夹层代码,则这个简单嵌套循环就是完美嵌套循环.

证明:由定义 3、定义 4 可知,简单嵌套循环比完美嵌套循环多出的语句是那些引起循环携带依赖的语句,这些语句的上层循环和下层循环之间在进行置换时会改变语义.由定义 5 可知,这些语句就是夹层代码. □

夹层代码为循环转换提供了方便,在确定循环类型时可以直接查看有无夹层代码,而无需通过繁琐的循环置换来判断.

模型法可以很好地解决完美嵌套循环的问题,但对于简单嵌套循环却不使用这种方法,原因有以下两点:

- 首先,使用模型法意味着在编译器中引入新的中间语言.Open64 使用了多层次的中间语言 WHIRL,目的是在不同的语言层次实行不同阶段的优化,这样可以将各个优化阶段模块化,方便控制和代码调试;如果再引入一种新的中间语言,在编译时需要执行 WHIRL-Polyhedral 模型 IR-WHIRL 的转换过程,WRaP-IT 也提供了相应的 CLOOG 代码生成模块.这不仅会延长编译时间,而且多次代码的转换可能会降低 Open64 后端 whirl2c/f 阶段代码语义的精确性;
- 其次,从上一节的分析我们可以得出,Polyhedral 模型的工作建立在 SCoP 基础上,而且 WRaP-IT 只分析饱满的 ScoP.当简单嵌套循环无法转换为完美嵌套循环时,夹层代码所在的 SCoP 必定是非饱满的.因此,模型法不适合、也不能用于简单嵌套循环的并行识别.

遍历法:针对上述问题,本文提出了类似于在 Open64 中实现识别 OpenMP 并行循环的方法.由于该方法通过遍历实现,因此称为遍历法(traverse algorithm),算法过程如图 3 所示.

遍历法在并行识别的过程中,循环类型分为 3 种:*invalid*,*snl* 和 *onl*.其中,*invalid* 是默认值或非循环,*snl* 为简单嵌套循环,*onl* 是不满足简单嵌套条件的循环.

表 1 是使用前两节方法对 NPB3.2.1 中 EP 程序中并行识别的结果.

表 1 中第 3 行表项就是例 2 中 EP 程序的 *i* 循环.如果遍历法标记的循环类型为 *onl*,那么将这类循环按照一般嵌套循环来处理.

```
1 //功能:基于遍历识别简单嵌套循环
2 //输入:需要进行并行识别的嵌套循环L;
3 //输出:parallel;当parallel为TRUE时表示该循环为可并行循环,为FALSE时表示为串行循环.
4 procedure Traverse_Algorithm(L)
5    设num=0;                                          // 用于标记该嵌套循环内的非snl个数
6    获取嵌套循环L的第1条语句S;
7    设 p=S;
8    while p!=NULL && Is_SCF(p) do begin               // Is_SCF表示是否为结构化控制流语句
9       if p是循环 && p的类型为invalid then begin      // invalid代表该循环类型还未知
10         设p内跳出循环的出口个数为exit=0;
11         for each p内的语句statement do begin
12            if statement==跳转语句||跳出语句||过程调用语句 then begin
13               exit++;
14               break;
15            end
16            if exit=0 then
17               标记循环p的类型为snl;                 // snl代表简单嵌套循环
18            else begin
19               标记循环p的类型为onl;                 // onl代表一般嵌套循环
20               num++;
21            end
22         end
23         p=p→next;
24      end
25   end
26   if num==0 then begin
27      设flag=Dependence_Test(L);                     // Dependence_Test依赖测试函数
28      if flag==TRUE then                             // 有循环携带依赖
29           parallel=FALSE;
30      else                                           // 无循环携带依赖
31           parallel=TRUE;
32   end
33   return parallel;
34 end Traverse_Algorithm
```

Fig.3 Traverse algorithm

图 3 遍历法

Table 1 Parallelism recognition of EP

表 1 EP 程序并行识别

循环体内代码	循环类型	识别方式
$x(i)=-1.d99$	*pnl*	模型法
$q(i)=0.d0$	*pnl*	模型法
$sx+=t3$	*snl*	遍历法

4 一般嵌套循环的并行识别

一般嵌套循环在面向对象的程序设计语言中非常常见.本文处理的嵌套循环都属于一般嵌套循环,在特定情况下,一般嵌套循环可以转换为简单嵌套循环,而简单嵌套循环又可以转变为完美嵌套循环.

由定义 2 和定义 3 可知,如果一般嵌套循环无跳转、过程调用和过程返回语句,那么就是简单嵌套循环.但是遍历法不能用于一般嵌套循环的并行识别,主要原因在于遍历法主要依靠依赖关系分析对循环进行并行识别,依赖关系分析无法处理非结构化控制流语句;而跳转和过程调用等出口分支语句是非结构化控制流语句,因此无法应用遍历法进行并行识别.

由于跳转语句和过程返回语句不会返回,可能会破坏循环的结构,因此本文不讨论包含跳转语句和过程返回的循环.对于过程调用,可以借助过程间分析可以很好地解决一般嵌套循环的识别问题.过程间分析问题可以分为流敏感问题和流不敏感问题[12],虽然流敏感问题还没有成熟的技术,但是可以通过使用近似和扩展的方法得到近似解.如,通过求解流不敏感问题逼近流敏感问题的解.

交互法:过程间分析可以分析出过程调用的副作用问题,因此借助 Open64 强大的 IPA 模块可以对一般嵌套循环进行识别.由于 IPA 阶段与其他优化阶段相互独立,而且分析和优化的编译方式相对独特,需要通过交互方式传递 IPA 信息,因此称这种并行识别方法为交互法(interaction algorithm),算法过程如图 4 所示.

```
1  //功能:基于交互文件识别一般嵌套循环
2  //输入:需要进行并行识别的嵌套循环L;
3  //输出:parallel;当parallel为TRUE时表示该循环为可并行循环,为FALSE时表示为串行循环.
4  procedure Interaction_Algorithm(L)
5     设num=0;                                  // 用于标记无法确定副作用问题个数
6     获取嵌套循环L的第1条语句S和最后一条语句T;
7     设 p=S;
8     while p!=NULL do begin                    // 中间语言为树结构
9        if p是循环 && p的类型为onl then begin  // onl代表一般嵌套循环
10          对p进行IPA分析;                     // IPA代表过程间分析
11          if p的副作用问题未确定 then begin   // 指过程间的副作用问题
12             num++;
13             break;
14          end
15          else
16             p=p→next;
17       end
18    end
19    if num==0 then
20       向交互文件file写入L的信息;             // file是一个xml文件
21    else begin
22       parallel=FALSE;
23       return parallel;
24    end
25    以交互界面方式向用户反馈交互文件file;
26    设flag=Dependence_Test(L);                // Dependence_Test依赖测试函数
27       if flag==TRUE then                     // 有循环携带依赖
28          parallel=FALSE;
29       else                                   // 无循环携带依赖
30          parallel=TRUE;
31    return parallel;
32 end Interaction_Algorithm
```

Fig.4 Interaction algorithm

图 4 遍历法

通过交互界面确认并行循环的方式,不仅可以确保一般嵌套循环的并行识别得到用户的确认,而且可以将那些已识别为并行但用户不希望成为并行的循环强制串行,能够更好地提升并行代码的效率.

以例 3 所示代码段为例,对于最外层循环(*i* 层循环),该循环即为一般嵌套循环.由于通过 IPA 分析可以明确副作用,因此可以通过交互法将其并行.

例 3:

```
for (i=1; i<N; i++){
   t1=s; t2=an;
   for (j=1; j<M; j++){
      flag=j/2;
      if (flag!=0) t3=func(t1,t2);
   }
   if (timers_enabled) timer_start(3);
   rank(t1,t2,t3);
}
```

交互法是一种需要与用户交互的方法,因此,如果过多使用这种方法的话,会接近手工并行方式的工作量.

从交互法分析过程中不难看出,对于用户不希望并行执行的循环,在交互法过程中可以通过强制串行的方式将该循环串行执行.在自动并行化过程中,对这部分工作的分析属于代价分析模块.在 Open64 中,从并行识别到实现循环的自动并行化变换过程是先进行循环并行识别,然后再读取标记为并行的循环,通过代价分析对值得并行分析的循环进行并行变换.可以将代价分析模块提前到并行识别阶段,在进行并行识别的同时,对循环进行代价分析,从而消除一些不必要的并行循环.目前,正在开展对这部分工作的研究.

为抑制交互法在编译过程中所占比重过大这一现象,现给出一种高效使用交互法的手段.对于采用 MPI 函数调用的分布存储并行化编译,MPI 函数的通信开销较大,如果识别出的并行循环过多,调用的 MPI 函数数量就会随之增长,并行程序的效率就会降低.因此,应该适当控制用于 MPI 自动并行化的并行循环.对于非专业人员来讲,如果模型法和遍历法已经识别出足够的并行循环(如一个嵌套循环内至多允许识别出适合 MPI 并行的循环个数为 *n*,*n* 可以预先进行设定,而模型法和遍历法已经识别出 *n* 个并行循环),则无需再调用交互法,因为再进行并行识别可能会带来过多的通信开销,导致并行效率降低;否则,再调用交互法进行半自动的并行识别.对于专业人员,因对程序结构和算法特点十分熟悉,可以直接利用交互法进行调优识别,从而获得更高的并行效益.

5 实验分析

5.1 应用举例与加速比测试

为验证本文提出的并行识别方法,以基准测试程序 NPB3.2.1 中的 EP 程序为例来说明本文描述的并行识别方法.该程序用于计算高斯伪随机数,因为它几乎不要求处理器之间相互通信,所以很适合于并行计算,而所测结果往往可以作为一个特定并行系统浮点计算可能达到的上限.

表 2 列出了使用 Open64 原来的并行识别方法、本文并行识别方法及 NPB3.2.1-MPI 手工并行版本中并行循环的对比情况.EP 程序中共有 8 个循环,其中编号为 6 和 7 的循环嵌入在 5 号循环内.

Table 2 Comparison of parallelism recognition methods

表 2 并行识别方式比较

循环编号	Open64 识别	本文识别	手工并行	循环编号	Open64 识别	本文识别	手工并行
1	并行	串行	串行	5	串行	并行	并行
2	并行	并行	并行	6	串行	串行	串行
3	串行	串行	串行	7	并行	串行	串行
4	并行	串行	串行	8	并行	串行	串行

表 2 的结果说明,本文方法的分析结果基本上比 Open64 的传统并行识别效果要显著,与手工并行识别的结果一致.

我们用 NPB3.2.1 中的 CG(conjugate gradient,共轭梯度)程序来测试自动并行化生成的 MPI 代码的加速比.CG 程序用于求解大型稀疏对称正定矩阵的最小特征值的近似值,程序算法是利用反幂法和共轭梯度,不断近似所求结果.并行识别后,*W* 规模下的加速比效果如图 5 所示.

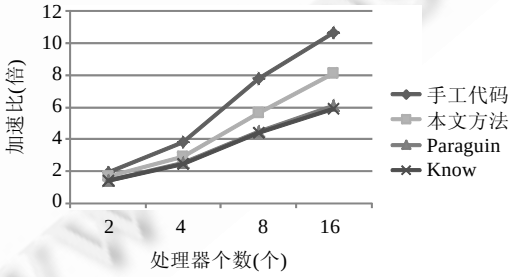


Fig.5 Speedup of CG

图 5 CG 程序加速比

从图 5 中可以看出:应用本文提出的并行识别方法生成 MPI 代码效率,比使用 Paraguin 和 Know 工具的效

率提高了至少 20%;与手工版本的 MPI 程序相比,在不同处理器个数条件下达到了手工 MPI 程序的 72%以上.

5.2 NPB3.2.1核心程序的分析与测试

基准测试集 NPB3.2.1 是美国 Numerical Aerodynamic Simulation 项目开发的并行基准测试程序,目的是用来比较各种并行机的性能,共有 11 个程序,其中,CG,EP,FT,IS 和 MG 为核心程序.将本文提出的并行识别方法分别用于分析这 5 个核心程序中的并行循环,结果见表 3.

Table 3 Results of the parallelism recognition methods used in NPB

表 3 本文并行识别方法分析 NPB 程序的结果

程序	循环总数	*PNL*	*SNL*	*ONL*	*Invalid*	本文识别总数	传统识别总数
CG	37	9	7	17	4	33	13
EP	8	3	1	4	0	8	4
FT	39	6	10	16	7	32	12
IS	17	5	3	8	1	16	8
MG	72	14	33	12	13	59	34

表 3 中,*Invalid* 项不仅包含本文不能进行并行处理的循环,还包括那些包含跳转和过程返回语句的一般嵌套循环.将表 3 分析结果应用于后端生成 MPI 代码,测试生成的 MPI 代码效率.测试平台建立在 SunWay 集群系统上,该集群由 20 个节点组成,每个节点配置 4 个主频 2.8GHz 的处理器.图 6 为 *W* 规模下 4 个进程时的加速比.

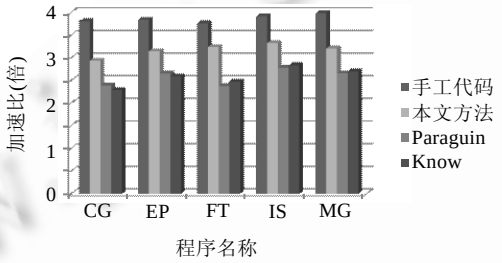


Fig.6 Speedup test for NPB3.2.1 main programs

图 6 NPB3.2.1 核心程序加速比测试

从表 3 和图 6 可以看出,与采用传统识别技术的编译系统 Paraguin 工具和 Kwon 的工具相比,采用本文提出的嵌套循环分类方法和相应的识别技术,能够更有效地识别并行循环,并行代码效率也有明显提升.表 3 中,*Invalid* 列中的循环个数即本文方法与手工识别方式的差别.图 6 中还列出了 NPB3.2.1 中 MPI 并行版本代码的执行加速比,从中可以看出,通过采用本文方法编译生成的被测程序的并行程序代码,平均加速比达到了手工 MPI 程序加速比的 70%以上.

6 结束语

分布存储体系结构并行编译的主要困难之一在于对程序并行循环的识别.以往的并行识别方式对单层循环和结构相对简单的嵌套循环识别比较有效,但是很难处理结构相对复杂的嵌套循环和过程调用.本文提出的并行识别机制有效解决了这个问题,并在各种测试中获得较好的效果.目前的识别机制大多只针对 for 循环,因此下一步工作将是对 while,do while 等循环的并行识别.另外,由于 MPI 并行带来的通信开销较大,导致采用 MPI 函数调用的自动并行化适合于更高粒度的并行,因此,开发粒度更粗的并行也将成为我们的下一步研究内容.对于除循环以外的结构的并行识别,课题组在工程实现和研究中初步形成的想法主要有两种:一种是将并行粒度从循环级提高到任务级,通过对程序不同函数之间的分析识别并行性;另一种是受到 NPB3.2.1 测试集手工 MPI 并行程序的启发,设想从算法优化角度提升并行粒度.

致谢 在此,向对本文研究工作提供基金支持的单位和评阅本文的审稿专家表示衷心的感谢,向为本文研究工作提供基础和研究平台的前辈致敬.

References:

[1] Zou DQ, He LG, Jin H, Chen XG. CRBAC: Imposing multi-grained constraints on the RBAC model in the multi-application environment. Journal of Network and Computer Applications, 2009,32(2):402−411. [doi: 10.1016/j.jnca.2008.02.015]

[2] Hall MW, Amarasinghe SP, Murphy BR, Liao S, Lam MS. Interprocedural parallelization analysis in SUIF. ACM Trans. on Programming Languages and Systems, 2005,27(4):662−731. [doi: 10.1145/1075382.1075385]

[3] Lin M, Yu ZY, Zhang D, Zhu YM, Wang SY, Dong Y. Retargeting the Open64 compiler to PowerPC processor. In: Proc. of the Embedded Software and Systems Symposia. San Francisco: IEEE Computer Society Press, 2008. 152−157. http://doi.ieeecomputersociety.org/10.1109/ICESS.Symposia.2008.69 [doi: 10.1109/ICESS.Symposia.2008.69]

[4] Kwon D, Han S, Kim H. MPI backend for an automatic parallelizing compiler. In: Proc. of the 14th Int'l Symp. on Parallel Architectures, Algorithms and Networks. San Francisco: IEEE Computer Society Press, 1999. 152−157. http://doi.ieeecomputersociety.org/10.1109/ISPAN.1999.778932 [doi: 10.1109/ISPAN.1999.778932]

[5] Ding Y, Kandemir M, Irwin MJ, Raghavan P. Adapting application mapping to systematic within-die process variations on chip multiprocessors. Lecture Notes in Computer Science, 2009,5409(1):231−247. [doi: 10.1007/978-3-540-92990-1_18]

[6] Ferner CS. The paraguin compiler—Message-Passing code generation using SUIF. In: Proc. of the IEEE SoutheastCon 2002. Piscataway: IEEE Press, 2002. 1−6. [doi: 10.1109/.2002.995545]

[7] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures: A Dependence-Based Approach. Morgan Kaufmann Publishers, 2001.

[8] Hu CJ, Li J, Wang J, Yao GL, Li YH, Ding L, Li JJ. Communication set generation for a special case of irregular parallel applications. Chinese Journal of Computers, 2008,31(1):120−126 (in Chinese with English abstract).

[9] BastoulC, Cohen A, Girbal S, Sharma S, Temam O. Putting polyhedral loop transformations to work. In: Rauchwerger L, ed. Proc. of the 16th Int'l Workshop on Languages and Compilers for Parallel Computing. Berlin: Springer-Verlag, 2004. 209−225. [doi: 10.1007/978-3-540-24644-2_14]

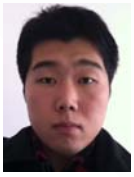
[10] Hoefler T, Lumsdaine A, Dongarra J. Towards efficient map reduce using MPI. In: Ropo M, Westerholm J, Dongarra J, eds. Proc. of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface. Berlin: Springer-Verlag, 2009. 240−249. [doi: 10.1007/978-3-642-03770-2_30]

[11] Girbal S, Vasilache N, Bastoul C, Cohen A, Sigler DPM, Temam O. Semi-Automatic composition of loop transformations for deep parallelism and memory hierarchies. Int'l Journal of Parallel Programming, 2006,34(3):261−317. [doi: 10.1007/s10766-006-0012-3]

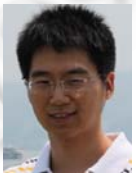
[12] Roy S, Srikant YN. Partial flow sensitivity. Lecture Notes in Computer Science, 2007,4873(1):245−256. [doi: 10.1007/978-3-540-77220-0_25]

附中文参考文献:

[8] 胡长军,李静,王珏,姚广利,李永红,丁良,李建江.一类非规则并行应用问题的通信集生成算法.计算机学报,2008,31(1):120−126.



赵捷(1987－),男,内蒙古通辽人,硕士生,主要研究领域为先进编译技术.



丁锐(1984－),男,博士生,主要研究领域为先进编译技术.

赵荣彩(1957－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为高性能计算,先进编译技术.

黄品丰(1987－),男,硕士生,主要研究领域为先进编译技术.