第 34 卷 第 9 期 计 算 机 工 程 2008 年 5 月
Vol.34 No.9 Computer Engineering May 2008
·软件技术与数据库· 文章编号：1000—3428(2008)09—0061—03 文献标识码：A 中图分类号：TP393

基于融合程序控制流的动态分解算法

韩 林，赵荣彩，姚 远
（解放军信息工程大学信息工程学院，郑州 450002）

摘 要：计算和数据分解是分布主存系统中并行编译的关键，在并行优化编译器的并行识别过程中，许多串行代码无法找到全局一致的分解结果。针对这种情况，该文提出一种融合程序控制流的动态分解算法，增加控制流对分解的影响，使生成的分解结果更适合于后端自动生成的并行代码。实验分析结果表明了该方法的有效性。

关键词：并行编译；动态分解；数据重分布；控制流

Dynamic Decomposition Algorithm Based on Merging Control Flow Analysis

HAN Lin, ZHAO Rong-cai, YAO Yuan
(School of Information and Engineering, PLA Information and Engineering University, Zhengzhou 450002)

【Abstract】 Computation and data decompositions are key factors of parallel compiler on distributed memory parallel computers. In the process of parallel recognition, many serial codes have no way to find a uniform decomposition result. This paper presents a dynamic decomposition algorithm based on merging control flow analysis. This method, based on the former researches, enhances the influence of control flow to the decomposition and makes the result more suitable for the execution of parallel codes generated by end. Experimental analysis shows the method is effective.

【Key words】 parallel compiler; dynamic decomposition; data reorganization; control flow

任何并行程序最终运行在具体的并行计算机上。分布存储多处理机作为高端科学计算最流行的并行计算机之一，具有可扩展性好、计算峰值高等优点，但没有统一的全局地址空间，不易编程。在分布存储计算环境下，计算节点访问本地存储器的速度远远快于通过网络访问异地存储器的速度，因此，为了使并行程序能够在分布存储多处理机中高效地执行，并行编译器必须给出有效的任务划分和数据分布，使计算时需要访问的数据尽量保存在本地存储器，从而减少通信开销。

1 相关工作

计算和数据分解以循环嵌套和循环嵌套中引用的数组为研究对象，以开发计算并行性和提高数据局部性为根本目的。

如何寻找有效的分解是并行编译器需要解决的关键问题之一，Kremer等学者指出，寻找最优的数据分解问题是一个NP难问题[1]。在并行研究领域中，人们从理论上提出仿射计算划分算法[2]、基于约束分析的计算和分解算法[3]、基于超平面的划分算法[4]以及基于数据空间融合的划分算法[5]等，这些算法对如何进行静态分解进行了比较详细的描述。文献[3]提出了一种动态分解算法，该算法基于对循环间数组相关性的分析，完成了静态分解子集的划分。

然而该算法没有考虑程序的控制流问题，试验表明这样的分解结果对于指导后端生成 SPMD(Single-Program Multiple-Data)并行程序会产生较多的数据重分布。

本文提出了一种融合程序控制流分析的动态分解算法，该算法依据程序的执行流程逐步将并行循环聚合为不相交的静态分解子集，在每个子集内调用静态分解算法，集合间允许数据重分布。该方法考虑到了划分结果在此后生成的SPMD 程序中的执行问题，能够有效地减少数据重分布次数。

2 问题描述

对循环L_1,L_2,L_3进行并行性分解，比较基于数组相关性分析的分解算法和本文的分解算法，并且分别给出两种分解的结果和对应的数据重分布情况。在此后的分析中，使用矩阵$\boldsymbol{C}$描述循环的划分，用矩阵$\boldsymbol{D}$描述数组的分布。示例代码段如下：

```
(L1)forall i1:=0 to N do
        for i2:=0 to N do
                X[i1,i2]:=f1(X[i1,g1(i2)]);
                Y[i1,i2]:=f2(Y[i1,g2(i2)]);
(L2)forall i1:=0 to N do
        for i2:=0 to N do
                Y[i2,i1]:=f3(Y[g3(i2),i1]);
(L3)forall i1:=0 to N do
        forall i2:=0 to N do
                X[i1,i2]:=f4(X[i1,i2],Y[i1,i2]);
                Y[i1,i2]:=f5(X[i1,i2],Y[i1,i2]);
```

基金项目：国家部委基金资助项目；河南省杰出人才创新基金资助项目(0521000200)
作者简介：韩 林(1978－)，男，博士研究生，主研方向：并行编译与并行识别；赵荣彩，教授、博士生导师；姚 远，副教授
收稿日期：2007-07-16 **E-mail：**stroller_hl@163.com

根据数组相关性动态分解算法，并行循环L_1与循环L_3之间共同引用数组X和数组Y，这 2 个循环的数组相关性最大，因此，将其合并到一个划分子集中，给出的分解结果为：$C_{1,3}=(1\quad 0)$，$D_{XY}=(1\quad 0)$，表明这两个循环都在第 1 层并行，所引用X和Y的第 1 维并行。并行循环L_2独自在一个划分子集中，分解结果为：$C_2=(1\quad 0)$，$D_Y=(0\quad 1)$。

结果表明：循环L_2在第 1 层是并行的，数组Y在第 2 维是并行的，2 个划分子集间对Y数组存在数据重分布。

可见在基于数组相关性分解的算法中也考虑到了数据的重分布问题，为了减少数据重分布，将 3 个循环划分为 2 个静态分解子集。

考察该算法给出的分解结果，当并行代码结束循环L_1的执行后，因为循环L_1与循环L_2中的数据分布方式不相同，所以在循环L_2之前需要为Y数组准备数据，此时发生数据重分布，当循环L_2执行完成后，循环L_3面临同样的问题，再次发生数据重分布。

该算法考虑到了程序的执行流对分解的影响，对于这个例子给出的分解结果为：$C_1=(1\quad 0)$，$D_{XY}=(1\quad 0)$，$C_{2,3}=(1\quad 0)$，$D_{XY}=(0\quad 1)$。这种分解结果仅在循环L_1执行完成后做一次数据重分布，原分解算法仍然需要改进。

3 算法设计

3.1 通信图

使用通信图$G_C=(V,E)$来描述程序中的每一个过程。如果过程内的某个循环具有一级或多级并行，则将该循环作为节点加入到通信图的节点集中。

每个节点维护着一张记录迭代次数的表，该表有$l+1$个值，此处l为该循环的深度。对于循环中的每个可并行循环层$i\in(1,2,\cdots,l)$，表中的第i项记录了该层的迭代次数，如果该层无并行，则标记迭代次数值为空，表中的最后一项记录整个循环在单处理器中执行的总迭代次数。

通信图中的有向边表明了过程内可能存在数据重分布的位置。如果循环u执行结束后，下一个执行的循环是v，即循环u在控制流图中可达的最近循环是v，则两节点之间存在从u到v的有向边(u,v)，边的权值$w(u,v)$表示两个循环之间发生数据重分布的最大可能通信开销，表示为

$$w(u,v)=\sum_{x\in A(u,v)}\psi(u,v,x)\times t(|x|)$$

其中，$A(u,v)$表示从节点u到v的所有数组集合；$\psi(u,v,x)$表示数组x从u到v可能被重分布的次数；$t(k)$表示在目标机器上移动k个数据元素的时间；$|x|$表示数组x的大小。如果数组在边(u,v)中存在通信，则说明该数组在循环u和循环v中有着不同的分布方式，两个循环间发生了数据重分布。

为了在通信图中表示非完美循环，引入了虚分解节点。如果一个可并行的完美循环u被另一个非完美循环v所嵌套，则对循环v建立虚分解节点，循环u节点位于循环v节点的下一层，加入虚分解节点能够有效地分析程序中的层次结构。

对于选择结构内的节点，可以忽略选择分枝，将节点直接加入到通信图中，这种方法简单有效，但可能会损失掉某些循环的部分并行性。

3.2 动态分解算法

动态分解算法流程如下：

(1)算法扫描串行程序，在每个过程内对可并行循环建立分解节点，将其加入到节点集V中，并且在分析循环时计算出可并行层的迭代次数和循环串行执行时总的迭代次数。根据程序控制流添加有向边，分析循环中的数组引用信息，根据循环间数组引用相关性设置有向边的权值，将其加入到边集E中，为程序建立通信图$G_C=(V,E)$。

(2)分别将每个循环节点初始化到各自的静态分解子集中并且计算此时程序执行开销，开销包括计算开销和通信开销两部分。

(3)自下而上遍历层结构通信图。

(4)在同一层内根据控制流信息，从起始循环开始，对存在有向边的节点做试探性合并，如果合并后数据重分布开销小于合并前，则提交合并操作，否则撤销该操作。当该层节点求解结束后，进入上一层重复分解过程，直至所有节点被加入到相应的静态分解集合。

(5)在每个静态分解子集内调用静态分解算法，求解最终的计算划分和数据分布结果。

3.3 实例分析

通过以下代码演示融合控制流的动态分解算法的求解过程。这段代码中的最外层有 2 个循环，其中 loop1 是完美的，loop2 是非完美的，在 loop2 中有 3 段完美循环，分别为 loop2.1、loop2.2、loop2.3。

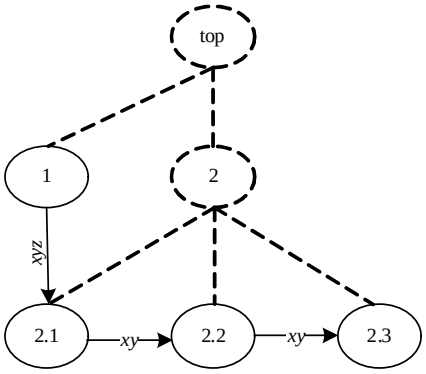
该算法给出了这段代码在并行识别过程中的计算划分和数据分布结果。

示例代码段如下：

```
/* loop nest1*/
forall i1=1 to N do
  forall i2-1 to N do
        x[i1,i2]=f1(i1,i2);
        y[i1,i2]=f2(i1,i2);
        z[i1,i2]=f3(i1,i2);
  end for
end for
/* loop2*/
for time =1 to T do
  /*loop nest2.1*/
  forall i1=1 to N do
        forall i2=1 to N do
              x[i1,i2]=x[i1,i2]+y[i1,i2];
              y[i1,i2]=y[i1,i2]+z[i1,i2];
        end for
  end for
  /*loop nest2.2*/
  for i1=1 to N do
        forall i2=1 to N do
          y[i1,i2]=y[N-i1+1,i2]+x[i1,i2];
        end for
  end for
  /*loop nest2.3*/
  forall i1=1 to N do
        for i2=1 to N do
          x[i1,i2]=x[i1,i2]+x[i1,i2-1];
```

```
            y[i1,i2]=x[i1,i2]+y[i1,i2]+z[i1,i2];
        end for
    end for
end for
```

对该段代码建立相应的通信图。对代码段中的完美循环分别建立节点，同时分析每个节点，设置其并行层的迭代数和串行迭代总数。对非完美循环建立虚分解节点，用以描述通信图中的层次信息。依据程序控制流添加节点间的有向边，最终的通信见图 1。



$\Gamma_1 = (N,\ N,\ N^2)$　　$\Gamma_{2.1} = (N, N, N^2T)$

$\Gamma_{2.2} = (\Lambda,\ N, N^2T)$　　$\Gamma_{2.3} = (N, \Lambda, N^2T)$

图 1　示例代码对应的通信

在图 1 中，向量 Γ 记录循环中各层的执行迭代数及循环在单处理器上串行执行时的迭代总数。算法从通信图的最内层节点开始合并，按照控制流首先对节点 2.1 和节点 2.2 进行试探性合并。合并前开销为

$$M_{old} = \Gamma_{2.1}(3)/\Gamma_{2.1}(1)/\Gamma_{2.1}(2)+\Gamma_{2.2}(3)/\Gamma_{2.2}(2)+w(2.1,2.2) =$$
$$T+NT+\psi(2.1,2.2,x)\times w(|x|)+\psi(2.1,2.2,y)\times w(|y|) =$$
$$(N+1)T+T\times w(N^2)+T\times w(N^2) =$$
$$(N+1)T+2N^2T$$

合并后，循环节点 2.1 和节点 2.2 在第 2 层是并行的，引用的数组 x, y, z 按列分布，开销为

$$M_{\text{new}} = \Gamma_{2.1}(3)/\Gamma_{2.1}(2)+\Gamma_{2.2}(3)/\Gamma_{2.2}(2) = NT+NT = 2NT$$

合并后开销小于合并前开销，即 $M_{\text{new}} \prec M_{\text{old}}$，提交合并操作。继续对“节点 2.1∩节点 2.2、节点 2.3”进行试探性合并，可以发现，合并后的 3 个节点只能串行执行，撤回合并操作。

此时程序进入上一层继续执行，对“节点 1 和节点 2.1∩节点 2.2”进行试探性合并，最终分解结果将节点 1、节点 2.1 和节点 2.2 划分到一个静态分解子集，节点 2.3 在另一个静态分解子集。

对 2 个子集分别调用静态分解算法，最终的分解结果为：

(1)静态分解子集 1 中，有

$C_{1,2.1,2.2} = (1\ \ 0)$, $D_{xyz} = (1\ 0)$

(2)静态分解子集 2 中，有

$C_{2.3} = (1\ \ 0)$, $D_{xyz} = (0\ 1)$

4　试验分析

选取 NAS Parallel Benchmark(NPB) 并行测试程序集中 SP 程序的 *exact_rhs*() 函数作为考察对象，在该函数中，使用 Stanford University Intermediate Format(SUIF)提供的并行性分析前端共识别出 20 个可并行循环。

针对这 20 个循环进行计算和数据分解，比较了使用基于数组相关性的动态分解方法和该融合程序控制流的动态分解方法，数据的重分布次数见图 2。

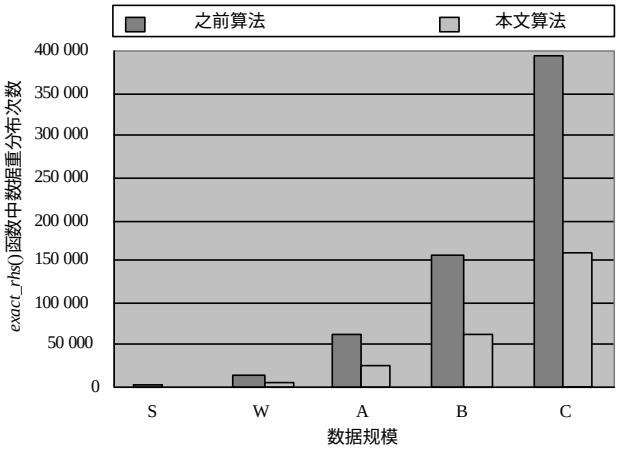


图 2　NPB-SP 数据规模

试验结果表明，融合程序控制流的动态分解方法能够有效地减少数据重分布的次数，对于以分布存储多处理机为目标的并行代码生成来说，能够显著改善通信性能。

5　结束语

在分布存储环境下，计算与数据分解的选择是重点，也是并行编译优化研究需要解决的问题之一。本文以控制流为依据，划分程序内的循环节点到不同的静态分解子集，尽可能地减少数据重分布次数。在分解算法中，控制流对后端生成 SPMD 程序有着现实的意义。并行分解问题中的适用性，开发程序中的粗粒度并行，是下一步研究的方向。

参考文献

[1] Kennedy K. Automatic Data Layout for High Performance Fortran[C]//Proc. of Supper-computer Conference. San Diego, Calif., USA: [s. n.], 1995.

[2] Lim A W, Cheong G I, Lam M S. An Affine Partitioning Algorithm to Maximize Parallelism and Minimize Communication[C]// Proceedings of the 13th ACM SIGARCH International Conference on Supercomputing. [S. l.]: ACM Press, 1999: 228-237.

[3] Anderson J M, Lam M S. Global Optimizations for Parallelism and Locality on Scalable Parallel Machines[C]//Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation. [S. l.]: ACM Press, 1993: 112-125.

[4] Sadayappan P. Communication-free Hyperplane Partitioning of Nested Loops[J]. Journal of Parallel and Distributed Computing, 1993, 19(2): 90-102.

[5] Xia Jun, Yang Xuejun. A Data Space Fusion Based Approach for Global Computation and Data Decompositions[J]. Journal of Software, 2004, 15(9): 1311-1327.