MPI自动并行化编译系统中消息传递代码生成算法

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

摘 要 传统MPI自动并行化编译系统从数据重分布的角度,生成面向分布式存储系统的消息传递程序,但是大量数据重分布通信的额外开销导致其加速比低。为了解决此问题,在基于Open64的MPI自动并行化编译系统后端,提出了一种消息传递代码生成算法。该算法以统一数据分布为中心,根据给定的并行化循环集和通信数组集,通过修改WHIRL表示的串行代码语法结构树,生成更精确的消息传递代码。实验结果表明,该算法能够较大程度地降低消息传递程序的通信开销,并且明显提升其加速比。

关键词 MPI,自动并行化编译,分布式存储系统,消息传递代码,Open64,加速比
中图法分类号 TP311 **文献标识码** A

Message-passing Code Generation Algorithm in the MPI Automatic Parallelizing Compilation System

CHEN Da-zhi ZHAO Rong-cai YAO Yuan HAN Lin
(School of Information Engineering, PLA Information Engineering University, Zhengzhou 450002, China)

Abstract From the perspective of data redistribution, traditional MPI automatic parallelizing compilation systems generate message-passing programs for distributed-memory systems, but a large number of data redistribution communication overheads result in their low speedups. Aiming at the problem, this paper proposed a message-passing code generation algorithm in the back-end of the MPI automatic parallelizing compilation system based on Open64. With the centre of uniform data distribution, the algorithm generates more accurate message-passing codes, according to the given sets of parallel loops and communication arrays, by modifying the WHIRL syntax trees of serial codes. Experimental results show that the algorithm can reduce communication overheads of message-passing programs to a large extent and improve their speedups significantly.

Keywords MPI, Automatic parallelizing compilation, Distributed-memory system, Message-passing code, Open64, Speedup

1 概述

MPI自动并行化编译系统是一种面向分布式存储系统的源源变换软件,它能将串行程序正确地变换为使用消息传递接口(Message Passing Interface, MPI)的单程序多数据(Single Program Multiple Data, SPMD)并行程序[1,2]。研究这种软件,一方面是为了使程序员从繁琐和容易出错的手工并行化过程中解脱出来,另一方面是为了充分利用分布式存储系统中昂贵的硬件资源[3]。

典型的面向分布式存储系统MPI自动并行化编译系统是由美国北卡罗来纳大学的Paraguin[4]提出的。其消息传递代码生成算法是以计算划分为中心,通过依赖关系测试,确定并行化循环集,以生成仅包含广播和收集通信的消息传递代码。当并行程序执行时,每个处理器拥有一份整个程序的数据拷贝,执行不相同的并行化循环代码段。为了维持所有数据拷贝的一致性,在每个并行化循环代码段两端、处理器之间使用广播和收集通信。不过,大量的广播和收集通信会带来巨大的额外开销,导致消息传递程序加速比几乎都为负。文献[5]在Paraguin基础上引入了数据分布概念,要求每个并行化循环拥有自己的数据分布,不同数据分布的并行化循环之间需要数据重分布。在此概念上,文献[6-8]的消息传递代码生成算法是以数据分布为中心,通过依赖关系测试,确定并行化循环集,建立数据重分布通信集,根据计算拥有者原则,确定了通信数组集,并生成了消息传递代码。虽然消息传递程序减少了广播和收集通信开销,但是由于数据重分布的散发和收集通信代码的存在,制约了消息传递程序加速比的提高。另一方面,Paraguin是以美国斯坦福大学的自动并行化编译系统SUIF[9]作为开发平台,但是SUIF是面向共享式存储系统,已停止更新,导致Paraguin难以作为面向分布式存储系统的自动并行化编译系统扩展。

因此,本文在基于Open64的MPI自动并行化编译系统(Open64MPI)后端,提出了一种以统一数据分布为中心、根据

到稿日期:2011-07-17 返修日期:2011-11-28 本文受核高基重大专项(2009ZX01036-001-001-2)资助。
陈达智(1984-),男,硕士生,主要研究方向为先进编译技术,E-mail:dazhi_chen_scholarship@hotmail.com;赵荣彩(1957-),男,教授,博士生导师,主要研究方向为先进编译技术、高性能计算等;姚 远(1974-),男,副教授,硕士生导师,主要研究方向为先进编译技术、高性能计算等;韩 林(1978-),男,讲师,主要研究方向为先进编译技术、高性能计算等。

更多通信类型的通信数组集，生成更精确代码的消息传递代码生成算法，以较大程度地消除由数据重分布导致的散发和收集通信代码，解决实际应用中加速比低的问题。Open64 是设计结构好、分析优化全面的开源工业级优化编译系统。许多知名研究机构和厂商对 Open64 各种分析优化模块进行持续更新[10]，使其具有更广的应用范围，但是没有涉及 MPI 自动并行化编译领域。

2 基于 Open64 的 MPI 自动并行化编译系统

2.1 基本概念

定义 1(并行化循环) 指按某种划分顺序将其迭代分配到各个处理器且能正确运行的循环。并行化循环集指由所有并行化循环组成的集合。

面向分布式存储系统的 MPI 自动并行化编译是一种粗粒度并行化，其对象是循环。一个串行程序可有多种并行化循环集。为了统一数据分布，本文算法要求并行化循环集中所有元素都需满足 3 个条件：(1)包含串行程序中最关键数组；(2)循环迭代空间尽量大；(3)循环划分层的上界值和下界值都一样。

定义 2(通信数组) 指处理器使用非本地或者维护本地数据产生传递的数据集合。通信数组集指由所有通信数组组成的集合。

本文算法所采用的通信类型有 mpi_send/mpi_recv、mpi_reduce、mpi_allreduce 以及 mpi_alltoallv 等。只要满足以下 3 个条件之一，就能发生通信：(1)主进程将输入数据广播给其余进程；(2)引用非本地数据；(3)使用归约操作；(4)主进程将收集所有进程的输出数据。

2.2 基础架构

如图 1 所示，Open64MPI 采用模块化设计方法，不同模块之间通过取名为 WHIRL 的中间表示来传递编译信息[11]。本文算法是对 Open64MPI 后端中消息传递代码生成模块的具体实现。

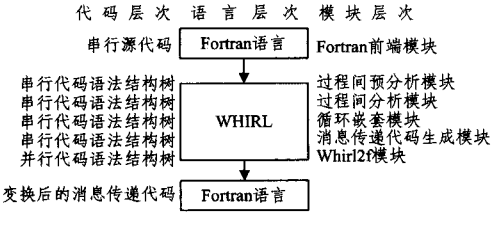

图 1 Open64MPI 的基础架构

由于绝大多数高性能科学计算程序使用 Fortran 语言进行编写，因此系统采用 Fortran 前端模块解析串行源代码的词法和语法，将结果以 WHIRL 表示的串行代码语法结构树传递给后端。以统一数据分布为中心，过程间预分析、过程间分析以及循环嵌套模块依次在串行代码语法结构树上求解并行化循环集和通信数组集。消息传递代码生成模块根据这些信息，按照 Open64 提供的以 WHIRL 表示的语法结构树的操作规则，将串行代码语法结构树变换为并行代码语法结构树，并调用 Open64 提供的 Whirl2f 模块产生变换后的消息传递 Fortran 代码。

2.3 以 WHIRL 表示的代码语法结构树

在基于 Open64 的 MPI 自动并行化编译系统中，使用树来描述整个代码的语法结构。Fortran 代码里的任何一个过程对应一棵树，过程里任何一条语句对应一棵子树，语句里的任何一个表达式也对应一棵子树。树由一系列结点构成，结点由 WN 类型的数据结构描述，分别简称为 WN 树和 WN 结点。

图 2 的代码语法结构树概括地描述了以 WHIRL 表示的一个过程。过程里所有语句属于 FUNC_ENTRY 类型 WN 结点的第三个 BLOCK 类型 WN 结点的子树，WN 结点之间的关系以父子链和兄弟链来表示。过程里的所有变量均由 ST 类型的数据结构唯一地描述，简称为 ST 表。

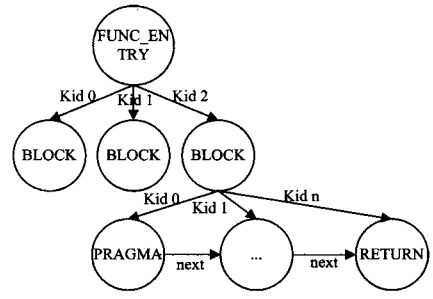

图 2 过程的 WHIRL 代码语法结构树

图 3 的代码语法结构树概括地描述了以 WHIRL 表示的 for 循环语句。DO_LOOP 类型 WN 结点唯一标志 for 循环语句，且只能是 BLOCK 类型 WN 结点的孩子。IDNAME 类型 WN 结点表示循环索引变量名，其右边的 STID 类型 WN 结点是循环索引变量初始化表达式对应的 WN 树的根结点。LE 类型 WN 结点是循环索引变量判断表达式对应的 WN 树的根结点，其右边的 STID 类型 WN 结点是循环索引变量增量表达式对应的 WN 树的根结点。for 循环语句里所有语句都属于 BLOCK 类型 WN 结点的子树。

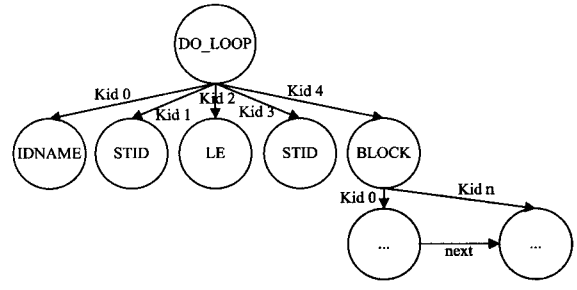

图 3 for 循环语句的 WHIRL 代码语法结构树

3 消息传递代码生成算法

3.1 形式化表示

3.1.1 并行化循环

对于一个 Fortran 过程，并行化循环可以用三元组形式化表示：$l=(wn\_loop, level, offset)$，其中 wn_loop 表示该循环的 WN 子树根结点的首地址，$level$ 表示划分层次号，$offset$ 表示划分偏移量。所有并行化循环组成的集合是并行化循环集：$L=\bigcup_{i=1}^{n}\{l_i\}$，其中 l_i 表示第 i 个并行化循环。

3.1.2 通信数组

对于一个 Fortran 过程，通信数组可以用五元组形式化表示：$a=(st\_array, kind, dim, direct, number)$，其中 st_array 表示通信数组 ST 表的首地址，$kind$ 表示通信类型，dim 表示通信数据划分维号，$direct$ 表示通信方向，$number$ 表示数据划

分维传递的组数目。所有通信数组组成的集合是通信数组集：$A=\bigcup_{i=1}^{n}\{a_i\}$，其中 a_i 表示第 i 个通信数组。

3.1.3 循环划分层

按划分循环的方式将其分为 3 种：环方式、块方式以及环-块方式。为了尽量减少发送数据不连续所导致的通信开销，文中使用块方式划分并行化循环，则循环划分层下界的表达式为_max(*oldlow*,*mypid* * *block_size*+*offset*)，上界的表达式为_min(*oldup*,(*mypid*+1) * *block_size*−1+*offset*)。其中，*oldlow* 表示该层循环的原下界，*oldup* 表示其原上界。

3.1.4 消息传递代码中关键变量

定义 3(变量 *mypid*) 表示当前处理器编号。该变量值通过调用基本 MPI 函数 mpi_comm_rank(MPI_COMM_WORLD,*mypid*,*ierr*)获得。

定义 4(变量 *numprocs*) 表示总处理器数目。该变量值通过调用基本 MPI 函数 mpi_comm_size(MPI_COMM_WORLD,*numprocs*,*ierr*)获得。

定义 5(变量 *block_size*) 表示以块方式划分并行化循环的块尺寸。为了满足统一数据分布，本文算法根据某个并行化循环的上界和下界值，只在所有并行化循环之前对变量 *block_size* 求解一次。设该循环划分层上界为变量 *upper*，下界为变量 *lower*，则通过判断表达式(*upper*−*lower*+1)% *numprocs* 是否为零，计算变量 *block_size* 的值。

3.2 源源变换

以串行程序中抽取出的核心代码段为例，详细描述消息传递代码生成算法如何将串行源代码变换为消息传递代码，见例 1。

例 1
```
program main
… …
do i=0,grid_points(1)−1 ·····loop1
    do j=0,grid_points(2)−1
        u(j,i)=forcing(j,i)
    enddo
enddo
… …
do i=2,grid_points(1)−1 ·····loop2
    do j=2,grid_points(2)−1
        rhs(j,i)=u(j−1,i−1) * dt
    enddo
enddo
……
end
```

3.2.1 并行化循环和通信数组指定

在例 1 中，编号为 loop1 和 loop2 的循环是并行化循环，形式化表示为 $l_1=(wn\_loop_1,0,0)$ 和 $l_2=(wn\_loop_2,0,1)$。其中，wn_loop_1 和 wn_loop_2 值在系统运行时获得，*level*=0 表示划分从外到内的第 0 层，*offset*=1 表示向右偏移 1 个数据，*offset*=0 表示没有偏移数据，则并行化循环集为 $L=\{l_1,l_2\}$。

由于 loop1 和 loop2 并行化循环使二维数组 u 被所有处理器互不相交地本地化，以及 loop1 最外层决定 block_size，导致每个处理器在执行 loop2 并行化循环时，要引用数组 u 中某个非本地元素。因此数组 u 中所有待传递元素组成的集合是通信数组，形式化表示为 $a=(st\_array,"send",2,"left",1)$。其中，*st_array* 值在系统运行时获得，字符串 *send* 表示使用 mpi_isend/mpi_irecv 类型通信，*dim*=2 表示第 2 维为数据划分维，字符串 *left* 表示最左侧数据向左边处理器传递，*number*=1 表示第 2 维传递 1 组数据，则通信数组集为 $A=\{a_1\}$。

3.2.2 消息传递代码生成

第一步，在 main 过程的最前面插入 mpi_init()、mpi_comm_rank()、mpi_comm_size() 3 个 MPI 初始化函数，在最后面插入 mpi_finalize()MPI 结束函数。第二步，采用 loop1 嵌套循环的上界值、下界值以及变量 *numprocs* 求解变量 *block_size*，并将其插于 loop1 嵌套循环之前。第三步，使用变量 *mypid* 和变量 *block_size* 修改 loop1 嵌套循环的最外层循环的上界和下界。第四步，采用 loop2 嵌套循环的上界值、下界值以及变量 *numprocs* 求解变量 *block_size*，并将其插于 loop2 嵌套循环之前。第五步，使用变量 *mypid* 和变量 *block_size* 修改 loop2 嵌套循环的最外层循环的上界和下界。第六步，将点到点通信函数 mpi_isend() 和 mpi_irecv() 分别插于 loop1 和 loop2 循环之间。例 2 是最终生成变换后的消息传递代码。

例 2
```
program main
… …
call mpi_init(…)
call mpi_comm_rank(…,mypid,…)
call mpi_comm_size(…,numprocs,…)
… …
if((grid_points(1)−1+1)%numprocs. eq. 0) then
  block_size=(grid_points(1)−1+1)/numprocs
else
  block_size=(grid_points(1)−1+1)/numprocs+1
endif
low=_max(0,mypid * block_size)
up=_min(grid_points(1)−1,(mypid+1) * block_size−1)
do i= low,up
    do j=0,grid_points(2)−1
        u(j,i)=forcing(j,i)
    enddo
enddo
call mpi_isend(… …)
… …
call mpi_irecv(… …)
if((grid_points(1)−1−2+1)%numprocs. eq. 0) then
  block_size=(grid_points(1)−1−2+1)/numprocs
else
  block_size=(grid_points(1)−1−2+1)/numprocs+1
endif
low=_max(1,mypid * block_size+2)
up=_min(grid_points(1)−1,(mypid+1) * block_size+1)
do i= low,up
    do j=2,grid_points(2)−1
        rhs(j,i)=u(j−1,i−1) * dt
    enddo
enddo
… …
call mpi_finalize(…)
end
```

3.3 算法描述

本文消息传递代码生成算法的基本思想是以统一数据分布为中心，根据给定的并行化循环集和通信数组集，通过修改 WHIRL 表示的串行代码语法结构树，输出更精确的消息传

递 Fortran 代码。图 4 是算法的具体实现过程。

```
procedure CodeGeneration(wn_root,L,A)
  //wn_root 是整个串行 Fortran 源代码的 WN 树根结点的首地址
  //L 是并行化循环集,即 L=⋃_{i=1}^{n}{l_i}
  //A 是通信数组集,即 A=⋃_{j=1}^{m}{a_j}
  wn_block3:= wn_root 的第三个 BLOCK 类型的孩子;
  为变量 mypid 创建和设置 ST 表;
  为变量 numprocs 创建和设置 ST 表;
  为变量 block_size 创建和设置 ST 表;
  为 mpi_init()创建 WN 树,并设为 wn_block3 的第一棵子树;
  为 mpi_comm_rank()创建 WN 树,并设为 wn_block3 的第二棵子树;
  为 mpi_comm_size()创建 WN 树,并设为 wn_block3 的第三棵子树;
  为 mpi_finalize()创建 WN 树,并设为 wn_block3 的倒数第二棵子树;
  if L≠∅ then begin
    for each l_i∈L do begin
      为变量 block_size 的求解创建 WN 树;
      将其设置为并行化循环 l_i 的左兄弟;
      为并行化循环 l_i 修改循环划分层索引的上界和下界;
    end
    for each a_j∈A do begin
      为通信数组 a_j 创建若干棵 MPI 通信函数的 WN 树;
      根据产生通信的并行化循环,设为 wn_block3 的子树;
    end
  end
  调用 Whirl2f 模块;
end CodeGeneration
```

图 4　消息传递代码生成算法

4　性能测试与分析

实验用例是美国国家航空航天局开发的并行基准测试集(NAS Parallel Benchmarks,NPB)。实验平台是 Sunway Ⅱ 分布式存储系统,由 20 个计算结点组成。每个结点配置 4 个主频为 2.8GHz 的 Xeon(TM)处理器,结点之间通过百兆交换机互连。操作系统为 Red Hat Linux 7.2,MPI 编译器为 MPICH1.2.7。

为验证本文算法的效果,首先使用本文算法和改进 Paraguin 分别对 NPB 中串行 Fortran 程序进行 MPI 自动并行化编译。其中,本文算法所需的并行化循环集和通信数组集是通过手工方式给出的;然后使用 MPICH1.2.7 在 W 规模下编译它们变换后的消息传递程序;接着在 Sunway Ⅱ 上运行由本文算法、改进 Paraguin 所产生的两个可执行消息传递程序;最后记录每个程序的运行时间,并计算所有的加速比。

图 5—图 7 显示了对 NPB 中源于计算流体动力学的 BT、CG、EP、FT、LU、MG 以及 SP 用例,在不同处理器个数的运行环境下,两种消息传递程序所产生的加速比情况。由于存在较多的散发和收集通信代码,大量的数据重分布通信开销导致 Paraguin 生成的消息传递程序产生较低的加速比,制约其加速度随着处理器个数扩大而变大的趋势。针对该问题,本文算法生成包含精确 MPI 通信函数的消息传递代码,其并行程序的加速比明显提升,并且随着处理器个数扩大而变大。但是在群通信时可能会引入一些额外开销,造成部分实验用例不够理想。

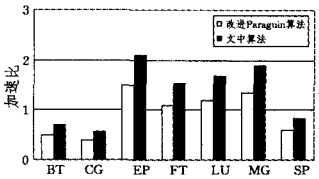

图 5　处理器个数为 4 时的加速比

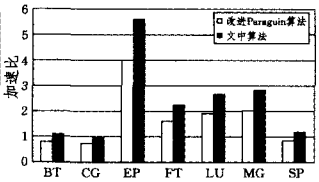

图 6　处理器个数为 16 时的加速比

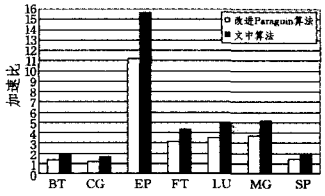

图 7　处理器个数为 64 时的加速比

结束语　分析了传统 MPI 自动并行化编译系统只能生成低加速比的面向分布式存储系统的消息传递程序问题,从统一数据分布的角度,在基于 Open64 的 MPI 自动并行化编译器后端,提出了一种消息传递代码生成算法。算法在给定的并行化循环集和通信数组集情况下,修改 WHIRL 表示的串行代码语法结构树,使串行代码自动生成更精确的消息传递代码。实验结果表明,本文算法能较大程度地减少由数据重分布引起的散发和收集通信代码生成,并且变换后的消息传递程序的加速比有了明显提升,从而适合应用于面向分布式存储系统的 MPI 自动并行化编译。随着主从异构体系结构出现,我们以后的工作是研究 MPI+OpenMP 混合自动并行化编译中并行代码的生成问题,从而进一步挖掘串行程序并行性,更好地发挥新一代高性能并行计算机的作用。

参 考 文 献

[1] Allen R, Kennedy K. Optimizing Compilers for Modern Architectures[M]. San Francisco: Morgan Kaufmann, 2001

[2] Rauber T, Gudula Runger. Parallel Programming: for Multicore and Cluster Systems[M]. New York: Springer, 2010

[3] 沈志宇,胡子昂. 并行编译方法[M]. 北京:国防工业出版社,2000

[4] Ferner C S. The Paraguin Compiler: Message-passing Code Generation Using SUIF[C]//Proceedings of the IEEE SoutheastCon 2002. Columbia SC: ETATS-UNIS, 2002: 1-6

[5] 韩林,赵荣彩,董春丽,等. 一种基于线性代数的计算和数据自动分解算法[J]. 计算机科学,2007,34(1):278-280

[6] 龚雪容,生拥宏,沈亚楠. 串行程序并行化中计算代码与同步通信代码的自动生成[J]. 计算机应用与软件,2008,25(1):91-92

[7] 张平,李清宝,赵荣彩. 消息传递并行程序的自动生成[J]. 计算机工程与应用,2007,43(8):74-77

[8] 杜澎,赵荣彩,董春丽. MPI 通信代码自动生成算法[J]. 计算机应用,2007,27(3):759-761

[9] Hall M W, Anderson J M, Amarasinghe S P, et al. Maximizing Multiprocessor Performance with the SUIF Compiler[J]. IEEE Computer, 1996, 29(12): 84-89

[10] Overview of the Open64 Compiler Infrastructure[EB/OL]. http://www2.cs.uh.edu/~dragon/Documents/open64-doc.pdf

[11] WHIRL Intermediate Language Specification[EB/OL]. http://cdnetworks-kr-1.dl.sourceforge.net/project/open64/open64/Documentation/whirl.pdf