基于重排序变换和循环分布的通信优化算法

陈达智 赵荣彩 韩 林 丁 锐 赵 捷
(解放军信息工程大学信息工程学院 郑州 450002)

摘 要 针对现有通信优化算法无法使MPI自动并行化编译器生成加速比理想的消息传递程序问题,提出了一种基于重排序变换和循环分布的通信优化算法。该算法根据给出的过程间副作用集合和基于mpi_wait/mpi_irecv移动的重排序变换规则,有序地采用重排序变换和循环分布,尽可能安全地扩大点到点非阻塞通信中通信与计算的重叠窗口,使MPI自动并行化编译器生成具有更多计算重叠通信的消息传递代码。实验结果表明,该算法能够隐藏更多的点到点非阻塞通信开销,并且明显提升消息传递程序的加速比。

关键词 重排序变换,循环分布,通信优化,并行化编译,点到点非阻塞通信,通信与计算重叠

中图法分类号 TP314 **文献标识码** A

Communication Optimization Algorithm Using Reordering Transformation and Loop Distribution

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

Abstract Aiming at the problem that existing communication optimization algorithms can't make MPI automatic parallelizing compilers to generate message-passing programs of ideal speedups, this paper proposed a communication optimization algorithm using reordering transformation and loop distribution. According to the interprocedural side effect sets and reordering transformation rule based on mpi_wait/mpi_irecv movements, this algorithm makes orderly use of reordering transformation and loop distribution, and then expands communication-computation overlap windows of the point-to-point non-blocking communication as safely as possible, so that MPI automatic parallelizing compilers can generate message-passing codes of overlapping communication with more computation. Experimental results show that this algorithm can hide more point-to-point non-blocking communication overheads than others and improve their speedups significantly.

Keywords Reordering transformation, Loop distribution, Communication optimization, Parallelizing compilation, Point-to-point non-blocking communication, Communication-computation overlap

1 概述

科学与工程计算中重大挑战性问题的存在,大力推动着计算机体系结构的迅速发展[1]。随着机群系统的普及,基于消息传递接口(Message-Passing Interface)的并行程序已成为提供其性能的有效途径,但是企业或者个人开发实用的消息传递程序,尤其并行化经典串行程序,需要投入大量的人力和财力。因此,MPI自动并行化编译器就成为较好解决并行编程和软件继承问题的重要手段。

MPI自动并行化编译器通常包括语法解析、过程间分析、任务划分、数据分布、通信数组求解、代码生成、通信优化等过程。通信优化是MPI自动并行化编译器中能有效提高转换后消息传递程序性能的最直接办法。大多数通信优化算法致力于直接优化并减少通信,其典型算法是由Amarasinghe[2]提出的。该算法在求解通信数组过程中,采用基于任务划分、数据分布、数据流信息统一的线性不等式系统,聚合了消息和删除了冗余数据。在Amarasinghe基础上,Ferner算法[3]通过合并虚拟机处理器到物理处理器映射,减少了消息总量;董春丽算法[4]通过分析循环嵌套能够进行边界冗余的只读数组,重新构造数据划分不等式,进行数据分布,减少了数据收发的通信量;刘磊算法[5]通过对计算划分不等式和依赖关系不等式进行傅立叶消元,消除了并行化代码中的冗余通信。除此以外,文献[6,7]也对基于减少通信的优化算法进行了深入研究。如果通信开销无法避免,则基于隐藏通信的优化算法能减少通信开销给并行程序性能带来的负面影响。Amarasinghe算法[2]通过对mpi_send、mpi_recv采用"生产即发送,消费才接收"策略来发送短消息,使mpi_send具有非阻塞性质,实现了隐藏通信。张平算法[8]在Amarasinghe基础上,通过数据流依赖分析的通信优化框架,尽量将发送代码前

到稿日期:2011-11-28 返修日期:2012-02-20 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和高性能计算等;韩 林(1978—),男,讲师,主要研究方向为先进编译技术和高性能计算等;丁 锐(1984—),男,博士生,主要研究方向为先进编译技术;赵 捷(1987—),男,硕士生,主要研究方向为先进编译技术。

移和外提，实现了跨 do 循环的隐藏通信。但是由于其使用阻塞通信函数，制约了更多计算重叠通信的机会。Athanasaki 算法[9]通过对包含 mpi_isend、mpi_irecv、mpi_wait 的循环嵌套，采用循环铺砌，实现了流水线方式的隐藏通信，不过，其只能在循环嵌套内隐藏通信。相对于基于减少通信的优化算法，近些年基于隐藏通信的优化算法发展较慢，重要原因是编译器不知道 mpi_isend、mpi_irecv、mpi_wait 过程间副作用，保守地对待包含它们的 do 循环，无法对之采用重排序变换和循环分布，实现跨 do 循环的隐藏通信。

因此，本文提出了一种根据 mpi_isend、mpi_irecv、mpi_wait 过程间的副作用集合和基于 mpi_wait/mpi_irecv 移动的重排序变换规则，有序地采用重排序变换和循环分布，让更多计算重叠通信的通信优化算法，以较大程度地隐藏无法避免的通信开销，解决实际应用中低加速比的问题。

2 安全性分析

由于本文算法将对包含 mpi_isend、mpi_irecv、mpi_wait 的 do 循环采用重排序变换和循环分布，而 MPI 自动并行化编译器不能通过数据依赖分析得到它们的过程间副作用影响，导致无法针对它们进行重排序变换和循环分布，因此，本节先给出保证该两种程序变换都维持原程序数据依赖关系的过程间副作用集合。又由于 mpi_isend、mpi_irecv、mpi_wait 的重排序变换存在通信的安全问题[10]（比如死锁），因此，本节再给出保证重排序变换后通信仍然是安全的基于 mpi_wait/mpi_irecv 移动的重排序变换规则。

2.1 过程间副作用集合

用 3 个从指令映射到变量集合的函数来刻画它们过程间副作用。这些函数是：

DEF, MOD, USE：Procedure→set of Variable

具体如下：

$DEF(p)$＝由函数 p 肯定赋值的变量集合。

$MOD(p)$＝由函数 p 可能赋值的变量集合。

$USE(p)$＝由函数 p 在被其重新赋值之前可能引用的变量集合。

表 1 以 Fortran 语言形式给出了 MPI-2 标准[11]中 mpi_isend、mpi_irecv、mpi_wait 调用接口。mpi_isend 的功能是启动一个标准的非阻塞发送操作，即先告诉后端，将由 sbuf 中连续 count 个 datatype 类型元素组成的 tag 号消息发送到 comm 域的 dst 目标进程，再设置 request 和 ierr，立即返回。其调用返回并不代表消息已经成功发送，只表示该消息可以被发送。mpi_irecv 的功能是启动一个标准的非阻塞接收操作，即先告诉后端，将来自 comm 域的 src 源进程的 tag 号消息存入 rbuf 里的连续 count 个 datatype 类型元素中，再设置 request 和 ierr，立即返回。其调用返回也不代表已经接收到了相应的消息，只表示符合要求的消息可以被接收。mpi_wait 是非阻塞通信检查函数，一直等到与它的 request 相应的非阻塞通信完成后才返回，同时释放 request，然后将与 request 完成有关的信息放在返回的状态参数 status 中，最后设置 ierr。所以，具有相同 request 的 mpi_isend 与 mpi_wait 组成一个发送操作，两条语句之间的区域称为发送重叠窗口；具有相同 request 的 mpi_irecv 与 mpi_wait 组成一个接收操作，两条语句之间的区域称为接收重叠窗口；一个消息传递由一个发送操作和一个接收操作组成。

表 1 mpi_isend、mpi_irecv、mpi_wait 调用接口

mpi_isend(sbuf,count,datatype,dst,tag,comm,request,ierr)
mpi_irecv(rbuf,count,datatype,src,tag,comm,request,ierr)
mpi_wait(request,status,ierr)

综上所述，表 2 给出了 mpi_isend、mpi_irecv、mpi_wait 的过程间副作用集合。其中，除了 MOD(mpi_wait)＝{rbuf}，其它都容易理解。因为 mpi_isend 不修改 sbuf 且接收操作可能在调用 mpi_wait 时才成功完成，所以 MOD(mpi_wait)只包含 rbuf。将这些过程间副作用集合添加到原有的过程间数据流分析信息中，形成扩展的过程间数据流分析信息，从而保证在维持原程序数据依赖关系的情况下，重排序变换和循环分布让更多计算重叠通信。

表 2 过程间副作用集合

DEF(mpi_isend)＝{request,ierr}
USE(mpi_isend)＝{sbuf,count,datatype,dest,tag,comm}
DEF(mpi_irecv)＝{request,ierr}
MOD(mpi_irecv)＝{rbuf}
USE(mpi_irecv)＝{count,datatype,src,tag,comm}
DEF(mpi_wait)＝{request,status,ierr}
MOD(mpi_wait)＝{rbuf}
USE(mpi_wait)＝{request}

2.2 基于 mpi_wait 和 mpi_irecv 移动的重排序变换规则

由于本文算法试图通过向后移动发送操作 mpi_wait 语句，向前移动 mpi_irecv 语句，使更多计算重叠通信，因此采用基于 mpi_wait 和 mpi_irecv 移动的重排序变换。所谓基于 mpi_wait 和 mpi_irecv 移动的重排序变换，是指仅通过改变 mpi_wait 和 mpi_irecv 语句位置，来仅改变代码的执行序，而不增加或取消任何语句的任何执行的程序变换。本文算法不向前移动 mpi_isend 语句，也不向后移动接收操作 mpi_wait 语句，是因为算法输入代码遵循“生产即发送，消费才接收”策略，即 mpi_isend 语句的前一条语句是发送缓冲区设置语句，接收操作 mpi_wait 语句的后一条语句是接收缓冲区设置语句。

如果 MPI 自动并行化编译器将发送操作 mpi_wait 语句或 mpi_irecv 语句移动到原来不包含该语句的执行路径上，或将其从原来包含该语句的执行路径上删除，则重排序变换后消息传递程序会发生通信死锁。所以，基于 mpi_wait 和 mpi_irecv 移动的重排序变换必须遵守以下规则：对发送操作 mpi_wait 语句和 mpi_irecv 语句的移动，不能进入原来不包含自己的复合语句，也不能从原来包含自己的复合语句中出来。该规则保证了每次算法将 mpi_wait 语句向后移动一步后的通信仍然是安全的，即如果位于 mpi_wait 语句初始位置的语句执行，则移动后 mpi_wait 语句必须执行；如果位于 mpi_wait 语句初始位置的语句不执行，则移动后 mpi_wait 语句必须不

执行。其也保证了每次算法将 mpi_irecv 语句向前移动一步后的通信仍然是安全的，即如果位于 mpi_irecv 语句初始位置的语句执行，则移动后 mpi_irecv 语句必须执行；如果位于 mpi_irecv 语句初始位置的语句不执行，则移动后 mpi_irecv 语句必须不执行。

以图 1 为例，解释基于 mpi_wait 和 mpi_irecv 移动的重排序变换规则。需要说明，为了方便描述本文算法处理的中间代码，所有例子都以 Fortran 语言描述并行中间代码。图 1(a)是 mpi_wait 移动前代码，显然，可以通过后移 S3 语句，对其后面计算重叠接收通信。如果遵守重排序变换规则，则可得到如图 1(b)所示的移动后通信安全的代码；否则根据 mpi_wait 过程间副作用集合，得到如图 1(c)所示的移动后通信不安全的代码，即 S3 语句只有在 req 相应的非阻塞通信完成后才返回，导致非 0 号进程永远处于等待状态。

```
S1 if(mypid .eq. 0) then
S2
mpi_isend(sbuf,1,...,req,ierr)
S3 mpi_wait(req,st)
   S4
S5 endif
```

(a)

```
S1 if(mypid .eq. 0) then
S2
mpi_isend(sbuf,1,...,req,ierr)
   S4
S3 mpi_wait(req,st)
S5 endif
```

(b)

```
S1 if(mypid .eq. 0) then
S2
mpi_isend(sbuf,1,...,req,ierr)
   S4
S5 endif
S3 mpi_wait(req,st)
```

(c)

图 1　一个基于 mpi_wait 移动重排序变换的实例

3　基于循环变换的通信优化算法

本节首先用形式化方式来定义算法中的一些概念，然后详细描述基于循环变换的通信优化算法。

3.1　问题描述

定义 1(消息传递四元组 t)　一个消息传递四元组描述组成该消息传递的 4 个非阻塞通信函数语句，记作 $t=(isend,swait,irecv,rwait)$，其中，*isend* 指向 mpi_isend 语句，*swait* 指向发送操作的 mpi_wait 语句，*irecv* 指向 mpi_irecv 语句，*rwait* 指向接收操作的 mpi_wait 语句。

定义 2(消息传递四元组集合 T)　由消息传递四元组组成的集合，记作 $T=\{t_1,t_2,\cdots,t_m\}$。

文本算法的输入代码不仅是并行中间代码，而且满足：(a)每个发送操作(或接收操作)的 mpi_isend(或 mpi_irecv)语句与 mpi_wait 语句紧紧相连，即两条语句之间没有任何其它语句；(b)发送缓冲区赋值语句之后紧跟 mpi_isend 语句，mpi_irecv语句之后紧跟接收缓冲区赋值语句。

3.2　算法描述

基于循环变换的通信优化算法根据过程间副作用集合和基于 mpi_wait 和 mpi_irecv 移动的重排序变换规则，有序地采用重排序变换、循环分布，尽量扩大所有重叠窗口，从而隐藏更多的通信开销。

3.2.1　CommOptUsingProTrans

CommOptUsingProTrans()是本文算法入口，形参 p 指向一个基于并行中间代码的过程，T 是该过程的消息传递四元组集合，ExIPAInf 是扩展的过程间数据流分析信息。它从全局范围角度扩大所有重叠窗口，尽可能多地隐藏通信开销。如图 2 所示，CommOptUsingProTrans 的基本思想是：第一步，调用 ExpandOverlapWindow()，在采用基于 mpi_wait 和 mpi_irecv 移动的重排序变换的情况下，将 p 中所有的重叠窗口尽量扩大。第二步，调用 ExpandOverlapWindowUsingLoopDistribution()，在第一步基础上，根据 ExIPAInf，采用循环分布，对 p 中所有能从循环分布获得收益的包含 mpi_isend、mpi_irecv、mpi_wait 语句的 do 循环分别进行循环分布，尽量扩大所有分布后得到的重叠窗口。

```
procedure CommOptUsingProTrans(p,T,ExIPAInf)
  ExpandOverlapWindow(p,T,ExIPAInf);
  ExpandOverlapWindowUsingLoopDistribution(p,T,ExIPAInf);
end CommOptUsingProTrans
```

图 2　CommOptUsingProTrans()实现

从 CommOptUsingProTrans()实现可知，对于过程中任何一个优化前的重叠窗口，ExpandOverlapWindow()、ExpandOverlapWindowUsingLoopDistribution()依次进行扩大操作或者不操作，所以算法是单调地扩大所有重叠窗口。

3.2.2　ExpandOverlapWindow

作为算法第一步的 ExpandOverlapWindow()，在采用基于 mpi_wait 和 mpi_irecv 移动的重排序变换的情况下，扩大一个过程内的所有重叠窗口。如图 3 所示，ExpandOverlapWindow 的基本思想是：先令 m 为 T 的总元素个数，如果 m 小于 1，则结束；否则通过一个 i 从 1 到 m 的 for 循环，每次处理 t_i 和 t_{m-i}。在循环中，首先尝试向后移动 $t_{m-i}.swait$。当下一条语句不是空语句，与 $t_{m-i}.swait$ 没有数据依赖关系，也不是任何其它发送操作的 mpi_wait 语句时，再向后检查，重复此过程，直到条件不成立；之后，让 $t_{m-i}.swait$ 成为 location 的下一条语句。然后尝试向前移动 $t_i.irecv$。当上一条语句不是空语句，也不是任何其它的 mpi_irecv 语句时，再向前检查，重复此过程，直到条件不成立；之后，让 $t_i.irecv$ 成为 location 的上一条语句。

```
procedure ExpandOverlapWindow(p,T,ExIPAInf)
  m:=T 的总元素个数;
  for each i∈{1,2,…,m} do begin
    next:=t_{m-i}.swait 的下一条语句;
    location:= t_{m-i}.swait;
    while next≠NULL and next 与 t_{m-i}.swait 没有数据依赖关系
          and next≠任何其它发送操作的 mpi_wait 语句 do begin
    location:=next;
    next:=next 的下一条语句;
  end
  在 p 中,将 t_{m-i}.swait 从其位置上删除,然后放在 location 之后,成为 location 的下一条语句;
  prev:=t_i.irecv 的上一条语句;
```

```
location:= t_i.irecv;
while prev≠NULL and prev 与 t_i.irecv 没有数据依赖关系 and
      prev≠任何其它 mpi_irecv 调用语句 do begin
  location:= prev;
  prev:= prev 的上一条语句;
end
在 p 中,将 t_i.irecv 从其位置上删除,然后放在 location 之前,成为 location 的上一条语句;
end
end ExpandOverlapWindow
```

图 3 ExpandOverlapWindow()实现

现在举例说明 ExpandOverlapWindow 实现过程,图 4(a)是输入代码。它先将 mpi_wait(req0,st0,ierr)连续向后移动,直至遇到空语句为止,并将其设为最后一条语句(见图 4(b));再将 mpi_irecv(rbuf,…,req1,ierr)连续向前移动,直至遇到空语句为止,并将其设为数组 $a()$ 赋值循环的前一条语句(见图 4(c))。显然,该应用说明 ExpandOverlapWindow 能扩大非复合语句内通信与计算的重叠窗口,实现隐藏通信的优化效果。如果图 4(a)是复合语句里的全部代码,则 ExpandOverlapWindow 优化复合语句内并行代码,最终复合语句内代码与图 4(c)一样,这说明 ExpandOverlapWindow 也能扩大复合语句内通信与计算的重叠窗口,实现隐藏通信的优化效果。

```
do i = 1, n
  a(i) = i
enddo
do i = 1, n
  sbuf(i) = a(i)
enddo
mpi_isend(sbuf,...,req0,ierr)
mpi_wait(req0,st0,ierr)
do i = 1, n
  b(i) = i
enddo
mpi_irecv(rbuf,...,req1,ierr)
mpi_wait(req1,st1,ierr)
do i = 1, n
  a(i) = rbuf(i)
enddo
```

(a)

```
do i = 1, n
  a(i) = i
enddo
do i = 1, n
  sbuf(i) = a(i)
enddo
mpi_isend(sbuf,...,req0,ierr)
do i = 1, n
  b(i) = i
enddo
mpi_irecv(rbuf,...,req1,ierr)
mpi_wait(req1,st1,ierr)
do i = 1, n
  a(i) = rbuf(i)
enddo
mpi_wait(req0,st0,ierr)
```

(b)

```
mpi_irecv(rbuf,...,req1,ierr)
do i = 1, n
  a(i) = i
enddo
do i = 1, n
  sbuf(i) = a(i)
enddo
mpi_isend(sbuf,...,req0,ierr)
mpi_wait(req0,st0,ierr)
do i = 1, n
  b(i) = i
enddo
mpi_wait(req1,st1,ierr)
do i = 1, n
  a(i) = rbuf(i)
enddo
```

(c)

图 4 ExpandOverlapWindow 应用

3.2.3 ExpandOverlapWindowUsingLoopDistribution

作为算法第二步的 ExpandOverlapWindowUsingLoopDistribution(),在采用循环分布情况下,将 do 循环内重叠窗口扩大到 do 循环外。如图 5 所示,ExpandOverlapWindowUsingLoopDistribution 的基本思想是:先令 m 为 T 的总元素个数,如果 m 小于 1,则结束;否则通过一个 i 从 1 到 m 的 for 循环,每次处理 T 中的 t_{m-i} 和 t_i。在循环中,首先处理 t_{m-i}。如果 $t_{m-i}.swait$ 是所在循环中的最后一条语句,则算法对 t_{m-i} 作进一步处理;否则,不对 t_{m-i} 作进一步处理。接下来,算法要判断 $t_{m-i}.swait$ 所在循环后面是否存在语句,并且那些语句是否与 $t_{m-i}.swait$ 没有数据依赖关系。如果成立,则算法根据 ExIPAInf 在 p 中使用循环分布,将 $t_{m-i}.swait$ 从循环中分布出去,然后不断向后移动,直至遇到与之有数据依赖关系的语句为止。处理完 t_{m-i},算法开始处理 t_i。如果 $t_i.irecv$ 是所在循环中第一条语句,则算法对 t_i 作进一步处理;否则,不对 t_i 作进一步处理。接下来,算法判断 $t_i.irecv$ 所在循环前面是否存在语句,并且那些语句是否与 $t_i.irecv$ 没有数据依赖关系。如果成立,算法根据 ExIPAInf 在 p 中使用循环分布,将 $t_i.irecv$ 从循环中分布出去,然后不断向前移动,直至遇到与之有数据依赖关系的语句为止。

```
procedure ExpandOverlapWindowUsingLoopDistribution(p, T, ExIPAInf)
m:=T 的总元素个数;
for each i∈{1,2,…,m} do begin
  sloop:=t_{m-i}.swait 所在循环;
  if t_{m-i}.swait 是 sloop 的最后一条语句 then begin
    if sloop 后面存在语句 and 紧跟于 sloop 后面的语句与 t_{m-i}.
      swait 语句无数据依赖关系 then begin
      在p 中,根据 ExIPAInf,使用循环分布,将 t_{m-i}.swait 从
        sloop 分布出去,紧跟于 sloop 之后,生成只包含 t_{m-i}.
        swait 的循环;
      swaitloop:=t_{m-i}.swait 所在循环;
      next:= swaitloop 的下一条语句;
      location:=NULL;
      while next≠NULL and next 与 swaitloop 之间无数据依赖
          关系 and swaitloop 不包含 mpi_wait 语句 do begin
        location:=next;
        next:=next 的下一条语句;
      end
      if location≠NULL then begin
        在p 中,将 swaitloop 从其位置上删除,然后放在 location
          之后,成为 location 的下一条语句;
      end
    end
  end
  rloop:=t_i.irecv 所在循环;
  if rloop 是所在循环中的第一条语句 then begin
    if rloop 前面存在语句 and t_i.irecv 与紧跟于 rloop 前面的语句
      无依赖关系 then begin
      在p 中,根据 ExIPAInf,使用循环分布,将 t_i.irecv 从 rloop
```

```
        分布出去，紧跟于 rloop 之前，生成只包含 t_i. irecv 的循
        环；
      irecvloop:=t_i. irecv 所在循环；
      prev:= irecvloop 的上一条语句；
      location:=NULL;
      while prev≠NULL and prev 与 irecvloop 之间无数据依赖关
          系 and irecvloop 不包含 mpi_irecv 调用语句 do begin
        location:= prev;
        prev:= prev 的上一条语句；
      end
      if location≠NULL then begin
      在p 中，将 irecvloop 从其位置上删除，然后放在 location
        之前，成为 location 的上一条语句；
        end
    end
  end
end ExpandOverlapWindowUsingLoopDistribution
```

图 5　ExpandOverlapWindowUsingLoopDistribution()实现

```
do i = 1, 100
  sbuf(i) = sbuf(i)+x(i)*z(i)
enddo
do i = 1, 4
  mpi_irecv(rbuf,...,i,...,req1,ierr)
  mpi_isend(sbuf,...,i,...,rcq0,ierr)
  mpi_wait(req1,st1,ierr)
  mpi_wait(req0,st0,ierr)
  sbuf(0) = sbuf(0)+rbuf(0)
enddo
```

(a)

```
do i = 1, 100
  sbuf(i) = sbuf(i)+x(i)*z(i)
enddo
do i = 1, 4
  mpi_irecv(rbuf,...,i,...,req1,ierr)
enddo
do i = 1, 4
  mpi_isend(sbuf,...,i,...,req0,ierr)
  mpi_wait(req1,st1,ierr)
  mpi_wait(req0,st0,ierr)
  sbuf(0) = sbuf(0)+rbuf(0)
enddo
```

(b)

```
do i = 1, 4
  mpi_irecv(rbuf,...,i,...,req1,ierr)
enddo
do i = 1, 100
  sbuf(i) = sbuf(i)+x(i)*z(i)
enddo
do i = 1, 4
  mpi_isend(sbuf,...,i,...,req0,ierr)
  mpi_wait(req1,st1,ierr)
  mpi_wait(req0,st0,ierr)
  sbuf(0) = sbuf(0)+rbuf(0)
enddo
```

(c)

图 6　ExpandOverlapWindowUsingLoopDistribution 应用

现在举例说明 ExpandOverlapWindow -UsingLoopDistribution 的实现过程，图 6(a)是由 ExpandOverlapWindow 优化过的输入代码。它首先向后移动 mpi_wait(req0,st0,ierr)，但是没有成功。然后，向前移动 mpi_irecv(recvbuf,…,i,…,req1,ierr)。经过判断，算法将 mpi_irecv(recvbuf,…,i,…,req1,ierr)从 do 循环中分布出来，如图 4(b)所示。接着，算法又经过判断，将包含 mpi_irecv(recvbuf,…,i,…,req1,ierr)的 do 循环向前移动，得到图 4(c)。显然，该应用说明 ExpandOverlapWindowUsingLoopDistribution 能够将 do 循环内通信与计算的重叠窗口扩大到 do 循环外，实现跨循环隐藏通信的优化效果。如果图 6(a)中的第一个 do 循环是其它复合语句(比如 if 语句)，且相互没有数据依赖关系，则 ExpandOverlapWindowUsingLoopDistribution 通过向前移动包含 mpi_irecv(recvbuf,…,i,…,req1,ierr)的 do 循环，来实现隐藏更多通信的优化效果。

4　实验对比与性能分析

实验用例由美国国家航空航天局开发，版本序号是3.2.1的高性能计算基准测试程序集(NAS Parallel Benchmarks，简称 NPB)。实验平台是 Sunway Ⅱ 机群系统，其由 24 个计算结点组成，每个结点包含 4 个主频是 2.8GHz 的 Xeon(TM)处理器，结点之间通过百兆以太网互联。操作系统是 RedHat Linux7.2，串行编译器是 gcc3.4.2，MPI 运行环境是 MPICH-2.1。

为验证本文算法优化效果优于 Athanasaki 算法[9]和张平算法[8]，在开源 MPI 自动并行化编译器 Paraguin[12]中实现了该算法。实验首先采用 Paragin 将串行 Fortran 用例分别进行 MPI 自动并行化编译(W 规模)；然后将 Athanasaki 算法和张平算法分别在这些生成的并行程序上手工实现，使用 MPICH-2.1 对优化后的代码进行编译，通过在Sunway Ⅱ上运行可执行消息传递程序，记录每个程序的运行时间；接着采用基于本文算法的 Paragin 将串行 Fortran 用例进行 MPI 自动并行化编译(W 规模)，使用 MPICH-2.1 对优化后代码进行编译，通过在 Sunway Ⅱ 上运行可执行消息传递程序，记录每个程序的运行时间；最后采用 gcc 将串行 Fortran 用例进行编译(W 规模)，通过在 Sunway Ⅱ 上运行可执行程序，记录每个程序的运行时间，并计算所有的加速比。

图 7—图 9 显示了在 NPB 中源于计算流体动力学的 FT、BT、MG、LU 和 SP 用例，在不同处理器个数的运行环境下，4 种消息传递程序所产生的加速比情况。对由 Paraguin 生成的消息传递代码而言，张平算法的优化效果比 Athanasaki 算法好，因为前者是基于跨 do 循环的隐藏通信，而后者重叠窗口扩大范围受 do 循环限制，但是与本文算法相比，张平算法的优化效果明显较差。因为本文算法不但具有 Athanasaki 算法使用 mpi_isend、mpi_irecv、mpi_wait 语句的优点，而且具有张平算法实现跨 do 循环隐藏通信的优点，这使得在由基于本文算法的 Paraguin 生成的消息传递代码中有更多的计算重叠和更多的通信开销。与 Athanasaki 算法相比，基于本文算法的并行程序的加速比平均提高了 27%；与张平算法相比，其加速比平均提高了 22%；并且随着处理器个数的扩大而变大。但是在由 Paraguin 生成的并行程序的数据重分布时可能会引入一些额外开销，以造成部分实验用例不够理想。

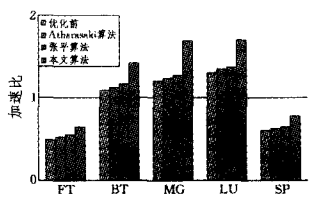

图 7　处理器个数为 4 时的加速比

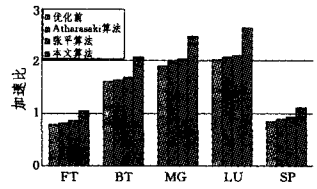

图 8　处理器个数为 16 时的加速比

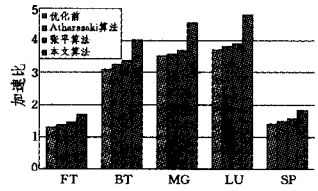

图 9　处理器个数为 64 时的加速比

结束语　本文分析了现有通信优化算法无法使 MPI 自动并行化编译器生成理想加速比的消息传递程序问题，从 mpi_isend/mpi_irecv/mpi_wait 行为角度，提出了一种基于循环变换的通信优化算法。该算法根据给出的过程间副作用集合和基于 mpi_wait/mpi_irecv 移动的重排序变换规则，有序地采用重排序变换和循环分布，尽可能安全地扩大并行中间代码里通信与计算的重叠窗口，使由 MPI 自动并行化编译器生成的消息传递代码中有更多计算重叠通信。实验结果表明，本文算法能够隐藏更多的点到点非阻塞通信开销，并且优化后消息传递程序的加速比有了明显提升，从而适合应用于机群系统的 MPI 自动并行化编译。随着主从异构体系结构系统的出现，我们以后的工作是研究 MPI＋OpenMP＋SIMD 混合自动并行化编译中的通信优化问题，从而更好地发挥新一代高性能并行计算机的性能。

参 考 文 献

[1] Hall M, Padua D, et al. Compiler research: the next 50 years[J]. Communication of the ACM, 2009, 52(2): 60-67

[2] Amarasinghe S P, Lam M S. Communication optimization and code generation for distributed memory machines[C]// Proceedings of the ACM SIGPLAN 1993 Conference on Programming Language Design and Implementation. New York, NY, USA: ACM, 1993, 28(6): 126-138

[3] Ferner C. Revisiting communication code generation algorithms for message-passing systems[J]. International Journal of Parallel, Emergent and Distributed Systems, 2006, 26(5): 323-344

[4] 董春丽，赵荣彩，等. 基于线性不等式的数据划分方法的优化[J]. 计算机应用，2007，27(5)：1251-1253

[5] 刘磊，赵荣彩，等. 基于线形不等式消元实现的通信优化[J]. 计算机工程，2008，34(7)：59-63

[6] 孙彤，李三立，等. 并行化编译中的一种集成优化方法[J]. 软件学报，1996，7(12)：705-713

[7] Martin P J. Suppressing independent loops in packing/unpacking loop nests to reduce message size for message-passing code[D]. Wilmington NC, USA: University of North Carolina Wilmington, 2010

[8] 张平. 并行化编译器中并行程序自动并行化和性能优化技术研究[D]. 郑州：解放军信息工程大学，2006

[9] Athanasaki M, Sotiropoulos A, Tsoukalas G, et al. Minimizing completing time for loop tiling with computation and communication overlapping[C]// Proceedings of the 15th International Parallel and Distributed Processing Symposium. Washington, DC, USA: IEEE Computer Society, 2001

[10] 都志辉. 高性能计算并行编程技术-MPI 并行程序设计[M]. 北京：清华大学出版社，2001

[11] Rauber T. Message-Passing Programming[M]. Springer, 2010

[12] 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

(上接第 291 页)

算法，且在搜索精度、时间开销方面其也要强于对比算法，在实际应用中具有高效性和一定应用价值，在图像处理领域将有着更广泛的应用前景。

参 考 文 献

[1] 胡欣，唐硕. 一种基于灰度级连通性的红外图像分割方法[J]. 计算机科学，2007，34(7)：238-240

[2] 郝颖明，朱枫. 2 维 Otsu 自适应阈值的快速算法[J]. 中国图象图形学报，2005，10(4)：484-488

[3] 刘京南，陈从颜，余玲玲，等. 一种快速二维熵阈值分割算法[J]. 计算机应用研究，2002，19(1)：67-70

[4] 张超，张家树，贾东立. 基于混沌遗传算法的图像阈值分割[J]. 计算机工程与应用，2006，42(2)：45-47

[5] 王春柏，赵宝军，何佩琨. 基于免疫遗传算法的自适应图像分割方法[J]. 红外与激光，2004，33(2)：178-181

[6] 朱峰，宋余庆，金华，等. 改良遗传算法在图像多阈值分割中的应用[J]. 江苏大学学报：自然科学版，2003，24(6)：66-69

[7] 赵娜，王希常，刘江. 自适应蚁群算法优化红外图像分割[J]. 计算机应用研究，2009，26(11)：4375-4377

[8] 郭娟，杨为民. 基于微粒群算法的二维最大熵图像分割方法[J]. 计算机仿真，2005，22(11)：94-97

[9] 孟红记，郑鹏，梅国晖，等. 基于混沌序列的粒子群优化算法[J]. 控制与决策，2006，21(3)：263-266

[10] 王文渊，王芳梅. 改进的最大熵算法在图像分割中的应用[J]. 计算机仿真，2010，28(8)：291-294

[11] 李太白. 基于混沌粒子群的 SVM 参数优化算法[J]. 重庆文理学院学报：自然科学版，2011，30(4)：81-84