一种HPF编译系统的研究与实现*

都志辉 丁文魁 郑耿斌 李晓明 许卓群

(北京大学计算机科学与技术系 北京 100871)

E-mail: duzh@bigfoot.com

摘要 HPF(high performance Fortran)是一种典型的数据并行语言,HPF编译系统的实现是并行计算研究领域的一个难点.文章介绍了一个HPF编译系统的研究与实现情况,在对该系统的主要组成进行了简要介绍之后,着重讨论了系统实现中的若干关键技术,并列出了部分HPF源程序及其编译器生成的相应代码.最后给出了对该编译器的一些性能测试结果和有关问题的讨论.

关键词 HPF(high performance Fortran),并行编译器,运行库,数据并行,群通信.

中图法分类号 TP314

HPF(high performance Fortran)[1]自从1993年被推出以来,在学术界和工业界引起了很大的重视,它集中体现了近十多年来人们研究并行计算的成果,并已成为工业界事实上的数据并行语言标准.迄今为止,不少公司,如PGI,IBM及DEC等都推出了各自的HPF编译器,SUN及其他一些主要计算机公司也有其HPF编译器研发组.尽管如此,人们所期望的,在分布存储并行系统上普遍高效实用的HPF编译器并未完全实现.这里的原因有多种:除了社会背景之外(Web的出现席卷了整个计算机界,计算机工业界的注意力有了很大的转移),对HPF这种复杂语言编译器的构造,在技术上还有许多未知数.因此,对HPF编译器的研究具有理论和实际的意义.

在分布存储系统上,串行程序(例如FORTRAN 77)的完全自动化并行编译难度很大.HPF鼓励程序员明确给出数据在各个处理器上的分布情况,这对于编译器来说是很难做到且不易做好的,但对程序员来说则相对容易得多.由于在程序中有数据分布的显式指导,减轻了编译器的负担,使编译器的主要工作集中于通信的检测和代码的生成上.在这方面有两种基本的学术思想:第1种是以编译优化为主线,开发各种局部和全局优化算法,鼓励在生成的结点程序中使用较低级的"点-点"通信原语,以创造较多的优化机会,这种思想贯彻的一个必然的结果就是编译器需要完成大量的符号操作;第2种思想是让编译器的工作依托在一个较强的(层次较高的)库之上,由该库完成分布数据的管理、全局和局部下标之间的转换以及常用的通信和计算功能.尽管这种库的一些功能不一定在运行时都使用,人们也一般地将它称作"runtime".这种思想实施的结果是,大量的编译时的符号计算体现为运行时的值计算,在所生成的结点程序中不鼓励使用"点-点"通信,而以所谓"群通信"代之.这种思想的基点是指望对HPF这样复杂的语言,能较容易地实现可运行的编译系统,同时对大部分数据并行问题保持较高的处理效率,我们的工作采用的就是这种思想.就目前的情况来看,它是有效的.

下面依次对我们编译系统的主要组成、关键技术、程序的分析与变换及代码的产生、性能测试等进行介绍.

---

* 本文研究得到国家重大基础研究攀登计划基金和国家863高科技项目基金资助.作者都志辉,1970年生,博士,主要研究领域为并行计算,科学计算可视化,地理信息系统.丁文魁,1946年生,副教授,主要研究领域为并行编译,系统软件.郑耿斌,1972年生,硕士生,主要研究领域为科学计算可视化,并行编译.李晓明,1957年生,博士,教授,博士生导师,主要研究领域为并行处理,分布式计算,计算机体系结构.许卓群,1936年生,教授,博士生导师,主要研究领域为科学计算可视化,空间数据分析与智能决策,并行计算.

本文通讯联系人:都志辉,北京 100084,清华大学计算机科学与技术系微机教研室

本文1997-11-14收到原稿,1998-01-19收到修改稿

1 系统主要组成

我们的编译系统主要由 4 部分组成(如图 1 所示)，它们是：支持 HPF1.0 全集的编译前端、各种变换模块、运行时支持类库及可视化分析工具集. 下面对它们分别进行简单介绍.

(1) 编译前端　编译前端[2]对 HPF 程序进行语法分析和语义分析，生成的中间表示是一棵程序树(Program Tree)，它由抽象语法树(Abstract Syntax Tree)、符号表(Symbol Table)和类型表(Type Table)组成. 我们的编译前端支持 HPF1.0 全集(现已支持 HPF2.0 全集)，在实现 HPF 编译前端的同时，我们还对印地安那大学开发的 Sage++类库[3]进行了详细分析并加以改造，使它可以支持 HPF，并增加了各种变换操作功能.

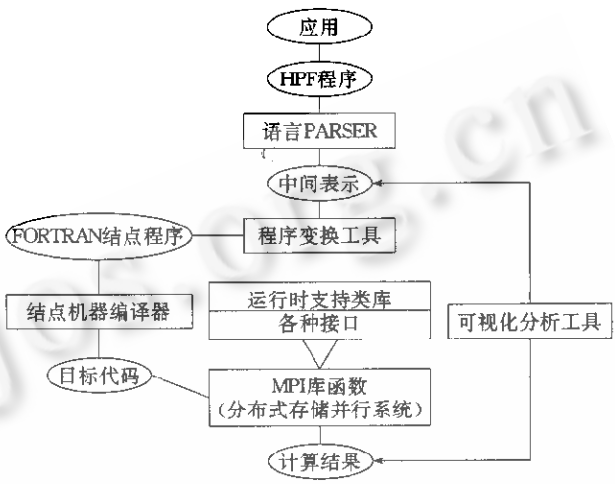

图1　编译系统结构图

(2) 变换模块集　整个编译过程可以分为 3 个阶段，即程序规范化、程序分析和程序变换. 程序规范化即把各种复杂的语法现象归整为语义等价的"规范"形式；程序分析阶段通过扫描分析已规范化的程序，产生变换阶段所需要的信息；变换阶段根据分析的结果，将原来的 HPF 程序转换为等价的结点程序，即编译输出.

(3) 运行时支持类库　运行时支持类库以锡拉丘兹大学开发的 Adlib 为核心. 该类库主要提供分布数据的管理、数据通信和基于分布数组的运算三方面的功能支持. 它是一个用 C++实现的类库，提供支持 Fortran 的接口.[4]

(4) 可视化分析工具集　可视化分析工具提供的功能有：中间表示结构及其与源程序对应关系的可视化、数据分布的可视化、通信方向及通信量的可视化、计算的可视化、运行时信息的统计分析几方面. 这一工具集是我们的编译系统的重要组成部分，我们正在进一步完善该工具集，以提供对编译系统更多、更全面的工具支持.

2 关键技术

这里介绍我们的编译系统中用到的主要技术 5 种：DAD(distributed array descriptor)分布数组描述子[5]、运行时支持类库、通信分类与检测、数据映射标准型、各种并行结构向 FORALL 语句的语义等价变换. 这些技术的应用对整个编译器的构造和编译器的性能有重要的影响.

2.1 DAD：分布数组描述子

一个 HPF 程序的有效运行分为编译和运行两个阶段，编译时和运行时从不同的角度看待它们所处理的程序，编译时看到的往往是一些从全局数据操作向局部数据操作的转换，而运行时看到的却是一些局部数据操作，这些局部操作结合起来形成全局操作的效果. 为了使编译时和运行时能协同工作，需要定义编译和运行时的接口协议. 编译时需要提供给运行时全局数据在局部存储单元的分布信息，同时运行时应该能够从编译时获得这些信息. 那么编译时和运行时的接口应该如何定义呢？一般认为它应满足如下要求：(1) 该接口必须精确地反映数组的分布信息；(2) 编译时可以方便地产生这些信息；(3) 运行时可以有效地使用这些信息. 据此，设计编译和运行时的接口——分布数组描述子 DAD，用来描述数组及其分布信息的数据结构，如图 2 所示.

```
struct DAD: public Frame{
  ... // 施加在
      // DAD 上的操作
  Range rngs[8]; // 数组各维
  Stride strs[8]; // 信息
  int elementLen; // 数组元素信息
  int elementType;
  int size;
  void * data; // 数据
  int grpHandle; // 处理器句柄
  ...
};
```

```
struct Frame{
  ... // 施加在
      // Frame 上的操作
  const int rank; // 处理器定位
  Group group; // 组信息
  ...
};
```

图2 DAD 的组成

从我们编译器的运行状况看,该 DAD 能有效地将编译时和运行时结合起来,使两者协同工作.实践证明,它是一个比较简洁而有效的接口.

2.2 运行时支持类库

我们的运行时支持系统是一个用 C++编写的类库,它直接调用 MPI 的通信原语.追求高效性是该类库的一个重要目标,为此,大量地使用内联(Inline)函数,以尽量避免使用动态内存申请,避免不必要的数据拷贝和移动、过程调用及虚函数等手段.

图3展示了我们运行时支持类库的基本结构.其顶部是运行时向外提供的4种编译接口,即两个 Fortran 接口(针对不同的 HPF 编译器)、一个用户级的 C++接口 adㅣㅣ和一个 Java 接口,它们构成了运行时支持类库对外接口的核心,在此基础上可以直接而方便地构造其他语言或编译器的接口.

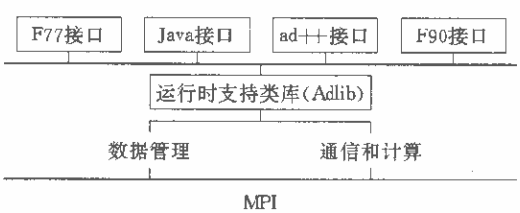

图3 运行时支持类库结构

分布数组上的算术操作和通信是核心类库的重要功能之一,如图3所示.通信操作包括 HPF 的内部函数如 CSHIFT,还有 pcrc_write_halo(用于实现优化 SHIFT 通信),remap(类似于任意形式分布数组间的相互拷贝)以及各种 gather 和 scatter(允许不规则形式的数据存取)操作等.算术操作包括各种数组归约与矩阵算术运算,对 HPF 标准库的支持正在开发之中.

规则的数据移动操作是通过一种块(Block)机制实现的,块包括源块和目标块,用于局部数据拷贝或发送和接收操作.块描述一个多维局部数组片段,包括偏移量(Offset)、维展(Extent)、步长(Stride)等参数.

运行时支持类库同时提供数据移动操作的基础控制设施,它支持 DAD 及对分布数据的遍历.DAD 中的重要结构是"范围"(Range)对象,它描述了数组全局索引在处理器各维的分布,还有"组"(Group)对象,它描述数组及其片断在处理器组中的嵌入情况.

迄今为止,运行时支持类库已在编译系统中发挥作用并已日趋成熟.

2.3 通信分类与检测

HPF 指导语句减轻了编译器数据划分方面的负担,"拥有者计算"的原则在一定程度上减轻了编译器计算划分的负担.那么,剩下的两个重要的工作便是通信检测和结点程序的产生.对于下面的例子:

```
...
REAL X(16),Y(16)
...
X(1:15)=Y(2:16)
```

是否需要通信取决于赋值操作中对应的元素是否在同一个处理器上,由于 HPF 是通过两级映射将数据分布到处理器上的,因此,通信的判断并不是一件显而易见的事情.我们将通信分为3类:无通信、SHIFT 通信及

REMAP 通信，并且设计了一种检测这3种通信的算法.

考虑下面的例子，如图4所示.

```
      REAL X(16),Y(16)
!HPF$ TEMPLATE     T(48)
!HPF$ PROCESSORS   P(4)
!HPF$ DISTRIBUTE   T(BLOCK)  ONTO  P
!HPF$ ALIGN  X(I)  WITH      T(3*I-1)
!HPF$ ALIGN  Y(I)  WITH      T(2*I+1)
      ...
      X(1:9:2)=Y(2:14:3)
```

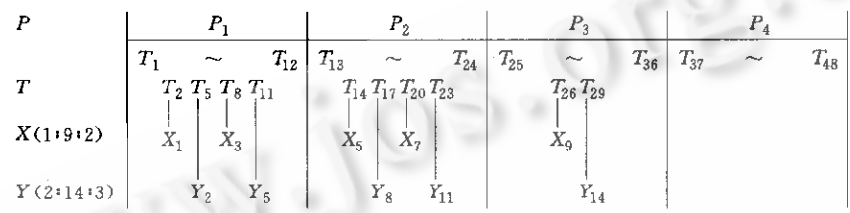

图4　无通信实例

本例子不需通信. 在我们的策略中，无通信作为一种特殊的 SHIFT 通信，即 SHIFT 的位移量为0. 从概念上说，SHIFT 通信只需将数组沿着模板进行一定量的平移就可以使相应的元素在同一个处理器上. 若将上例 BLOCK 形式的分布改为 CYCLIC 分布，即如图5所示，则需要 SHIFT 通信.

```
      REAL X(16),Y(16)
!HPF$ TEMPLATE     T(48)
!HPF$ PROCESSORS   P(4)
!HPF$ DISTRIBUTE   T(CYCLIC)  ONTO  P
!HPF$ ALIGN  X(I)  WITH       T(3*I-1)
!HPF$ ALIGN  Y(I)  WITH       T(2*I+1)
      ...
      X(1:9:2)=Y(2:14:3)
```

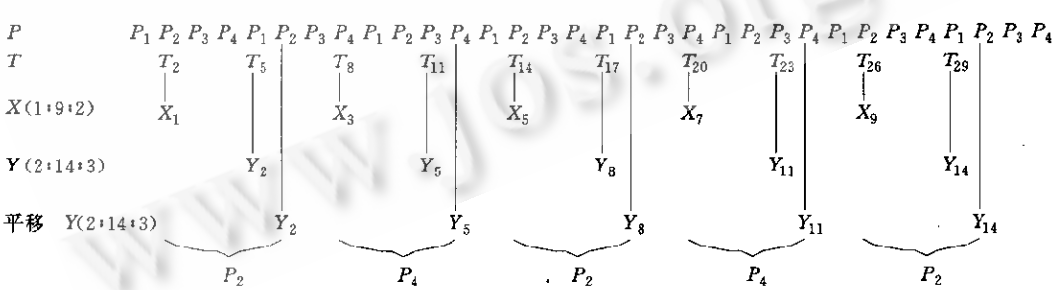

图5　SHIFT通信

SHIFT 通信的判断[6]是基于 SHIFT 同态的概念. 以图6为例，设 t_x，t_y 已规格化为 p 的倍数，则我们说例子中的数组赋值涉及到的数组片段是 SHIFT 同态的，只需满足如下条件：

$$\frac{a_x \times x_s}{a_y \times y_s} = \frac{t_x}{t_y}.$$

(对模板都是 CYCLIC 分布的情形，条件与上述不同，详见文献[6]).

```
!HPF$  PROCESSORS  P(p)
!HPF$  TEMPLATE    TX(tx),TY(ty)
!HPF$  DISTRIBUTE  TX(BLOCK)      ONTO  P
!HPF$  DISTRIBUTE  TY(BLOCK)      ONTO  P
!HPF$  DIMENSION   X(M),Y(N)
!HPF$  ALIGN  X(I)  WITH          TX(ax*I+bx)
!HPF$  ALIGN  Y(I)  WITH          TY(ay*I+by)
       ...
       X(x1:xu:xs)=Y(y1:yu:ys)
```

图6 通用的数组片段赋值

如果该条件成立,则数组片段赋值先通过将数组 *Y* 沿模板 *TY* 平移,然后进行局部的数据拷贝就可以完成.这里略去其证明及 SHIFT 位移量的计算公式.[6]

REMAP 通信泛指上述两种形式之外的所有通信形式.

在运行时支持类库中提供了对以上通信形式的支持,如 *pcrc_write_halo*()函数用于有效地支持 SHIFT 通信,*pcrc_remap*()函数用于支持 REMAP 通信.

2.4 数据映射标准型

由于 HPF 的数据映射形式比较多,这就为变换时的分析增加了困难,因此,我们定义了能与各种数据映射建立等价关系的标准数据映射形式,称之为 HPF 数据映射标准型.[7]首先将 HPF 的数据映射部分转换为与之等价的标准形式,然后再对标准形式进行处理.

通过引入 HPF 数据映射标准型,既简化了分析,又完全保留了原数据映射的含义,这也是我们对数据映射处理有特色的地方.我们的编译器能处理所有形式的数据映射.

2.5 各种并行结构向 FORALL 语句的等价语义变换

并行结构包括数组赋值语句、WHERE 条件数组赋值语句与结构、FORALL 语句与结构等.

FORALL 语句是最通用的描述块赋值的语句,而数组赋值与条件数组赋值都可以看作是它的特例.因此,将数组赋值及条件数组赋值转换成 FORALL 语句是一种完全等价的变换,一点都不丧失并行性,以后再统一对 FORALL 语句进行处理,可以使处理的情形得以化简,减轻生成结点程序的负担,我们所采取的处理方法就是这样的.FORALL 结构在多数情况下可以转换为 FORALL 语句来处理而不影响原来的并行性与语义.

我们的编译器对并行结构向 FORALL 语句转换的处理过程是这样的:

数组赋值语句→FORALL 语句

WHERE 结构→WHERE 语句→FORALL 语句

FORALL 结构→FORALL 语句

各种并行结构向 FORALL 语句的转换算法详见文献[8].

3 程序分析与变换

我们的编译器将 HPF 程序转换为 FORTRAN 77结点程序,并且集中精力于显式并行结构的处理.这一节对程序的分析与变换功能进行介绍,并给出部分编译器产生的代码.

3.1 程序分析

通过对程序的分析,可以得到以下信息供下一阶段使用:

(1) 处理器信息,包括处理器维数及每一维的大小;
(2) 模板信息,包括模板维数及每一维的大小;
(3) 每一模板的分布信息;
(4) 每一分布数组的对准信息;
(5) 每一 FORALL 语句中的变量引用情况;
(6) 过程参数中的数组哑元信息.

前4项可以从 PROCESSORS,TEMPLATE,DISTRIBUTE 和 ALIGN 数据分布指导语句中分别得到,对它们转换后产生相应数组的 DAD 及有关信息.

3.2 程序变换

从实现的角度看，程序变换分为两个阶段，即规范化和结点程序的产生. 规范化阶段在保持语义的情况下，对源程序的形式进行变换，使变换得到的结果更便于进一步的处理. 如，数组赋值向 FORALL 语句的变换. 这样，规范化后的语法现象将会大大缩小，后一阶段只需对缩小的语法集进行处理. 由于规范化操作保持语义，因此可以将不同的规范化操作分离开来，形成独立的模块，进行独立地实现和测试. 实际上，我们的编译器已采取了这种方法，实践证明，它是一种有效的方法.

规范化之后，通过全局到局部的程序变换产生结点程序，下面通过实例讨论一下对不同的语言成分的转换. 为了简单地说明问题，这里仅举一维数组的例子，但这里介绍的方法同样适用于多维数组和多维数组片段.

(1) 内存管理及地址的转换　我们的编译器有两种内存分配策略：①为每一个右手部 REMAP 通信数组动态分配临时空间，②为右手部需 SHIFT 通信的小移位量数组分配一定的阴影区(Ghost Area). 第1种方法用于处理 REMAP 通信的情况，在调用 *perc_remap*()通信函数时，先分配一个临时数组，该数组和左手部有相同的对准和分布形式，然后将右手部的数据拷贝到临时数组中；第2种方法用于有效地处理 SHIFT 通信，若编译器检测到有 SHIFT 通信，那么就在右手部数组加上阴影区，通过 *perc_write_halo*()将边界数据传送到阴影区中，这样就可以避免数组元素在内存中的不必要的拷贝.

除了内存分配、结点程序的生成，还有全局下标和局部下标相互转换的问题，运行时提供不同的函数完成这种转换. 对于多维数组，结点程序将下标计算线性化. 线性化下标和 DAD 查询函数，对于实现 HPF 过程的“抄写”(Transcriptive)参数特性(如 INHERIT 指导语句)是非常重要的，它可以避免在过程调用前后对参数的拷入和拷出操作.

(2) 分布数组描述子 DAD 的产生　编译器产生生成 DAD 的代码并对它进行初始化. 对于如下的 HPF 程序：

```
      REAL X(1:205),Y(-12:161)                    数组声明
!HPF$ PROCRSSORS P(2)                             处理器阵列声明
!HPF$ TEMPLATE TX(-2:205),TY(-17:190)             模板声明
!HPF$ DISTRIBUTE TX(BLOCK) ONTO P                 模板分布形式声明
!HPF$ DISTRIBUTE TY(BLOCK) ONTO P
!HPF$ ALIGN X(I) WITH TX(I)                       数组与模板的对齐
!HPF$ ALIGN Y(I) WITH TY(I--12)                   形式声明
```

将被转换为：

```
perc_shp_P(1)=2                                                          处理器数赋值
perc_grp_P=perc_new_group_grid(1,perc_shp_P))                            创建表示处理器阵列的 grp
perc_rng_TY(1)=perc_new_range_distribute((-17),190,1,perc_grp_P,1)       创建表示模板分布信息的 rng
perc_rng_TX(1)=perc_new_range_distribute((-2),205,1,perc_grp_P,1)
perc_dad_Y=perc_new_array_data(Y,perc_real,perc_size_real,1,perc_grp_P)  创建数组 DAD
call perc_set_array_align(perc_dad_Y,1,(-12),161,1,(-12),0,0,perc_rng_TY(1))  设置 DAD 有关参数
perc_dad_X=perc_new_array_data(X,perc_real,perc_size_real,1,perc_grp_P)
call perc_set_array_align(perc_dad_X,1,1,205,1,0,0,0,perc_rng_TX(1))
   ...
call perc_delete_array(perc_dad_X)                                       删除 DAD
call perc_delete_array(perc_dad_Y)
call perc_delete_range(perc_rng_TX(1))                                   删除 rng
call perc_delete_range(perc_rng_TY(1))
call perc_delete_group(perc_grp_P                                        删除 rng
```

对每一个处理器阵列产生一个相应的 grp，对每一个模板的每一维，创建一个 rng，记录其分布信息，对每一个分布数组，创建一个 DAD，记录其分布信息，它们都是整数句柄，指向运行时的相应对象，在程序结束时释放这些对象.

(3) 表达式及赋值语句的处理　由于在规范化阶段已做了许多工作，剩下的主要任务是 FORALL 语句的处理和标量赋值. 对于上例中的程序头，下面的 FORALL 语句

FORALL (I=8:112) X(I)=Y(I-1)

需要SHIFT通信,局部数组Y将增加相应的空间来接收边界移位数据,*perc_write_halo*()函数将边界数据送到下一个处理器的适当位置,转换后的结果为:

```
perc_irg0=perc_new_range_loop(8,112,1,1,0,perc_range(perc_dad_X,1))   创建表示数组分布的 rng
call perc_loop_bounds(perc_irg0,perc_lil_I,perc_liu_I,perc_lis_I)     计算循环边界及步长
perc_gtl_Y(1)=0                                                       设置模板参数
perc_gtu_Y(1)=2
call perc_write_halo(perc_dad_Y,perc_gtl_Y,perc_gtu_Y)                更新阴影区的数据
perc_sdd0=perc_new_array_section(1,perc_dad_X)                        创建数组片段分布描述信息
call perc_set_array_triplet(perc_sdd0,1,8,112,1,perc_dad_X,1)
call perc_coef(perc_dad_X,1,8,112,1,1,0,0,perc_u00,perc_v00)          计算循环控制变量的系数
perc_sdx0=perc_v00
call perc_coef(perc_dad_Y,1,8,112,1,1,(-1),2,perc_u10,perc_v10)       计算循环控制变量的系数
perc_sdx1=perc_v10
if (perc_on(perc_group(perc_sdd0))) then                              数据是否在本处理器上
   do I=0,(perc_liu_I-perc_lil_I)/perc_lis_I,1                        规范化后的循环头
      X(perc_sdx0)=Y(perc_sdx1)                                       循环体
      perc_sdx1=perc_sdx1+perc_u10                                    对下标索引变量的更新
      perc_sdx0=perc_sdx0+perc_u00
   enddo
endif
call perc_delete_array(perc_sdd0)                                     删除表示数组分布的 rng
call perc_delete_array(perc_irg0)                                     删除数组片段分布描述信息的对象
```

其中循环界及步长lil,liu,lis是通过计算FORALL索引空间和左手部DAD得到的,调用coef函数得到计算局部下标的系数,if语句保证赋值仅在拥有左部数据的处理器上进行,最后释放掉用于描述左手部数组片段和FORALL range的临时对象.

4 性能测试结果

图7、8分别是对LAPLACE方程例子进行不同的测试所得的结果.本文提供的测试结果数据所用的测试环境为:4台POWER PC通过HUB连接起来,使用的操作系统为AIX版本4,Fortran编译器为IBM的xlf.数组大小分别为256×256,512×512,1024×1024,2048×2048,图8将我们编译器产生的结点程序与直接用MPI编写的程序进行了对比,测试结果是编译器产生结点程序的执行效率为手写结点程序的80%左右.我们的编译器已实现了一个实用的HPF子集[9],测试表明,它对一大类问题的解决具有高效性.关于测试数据和结果,详见文献[10,11].

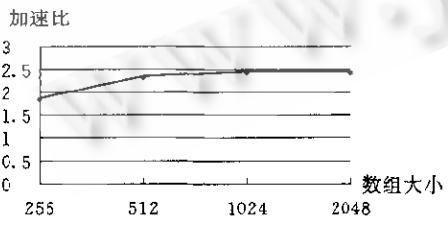

图7 Laplace方程加速比的测试

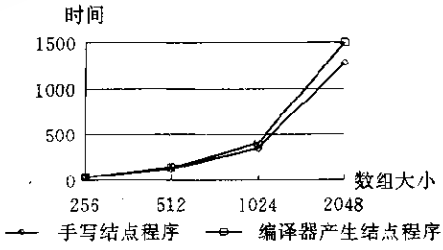

图8 编译器产生结点程序与手写结点程序的对比

5 结 论

以上介绍的HPF编译系统已部分实现.实践表明,基于运行时支持构造编译器是一种可行的方法,对于并行语言编译器的研究、开发和教学都有意义,它可以用于迅速地开发一个比较实用的并行编译器.

对于文献[12]中的编译器构造策略,运行时支持比较弱,需要编译器直接产生发送和接收原语来完成通信

操作，这样虽然有产生更有效代码的潜力，但编译器的构造会变得很复杂.

最近见到的关于 HPF 编译器的文章是文献[13]. 它通过枚举的方法来得到局部循环迭代集和通信集，相对而言，我们认为，通过运行时支持可以得到更直接、更有效的好处，特别是对于规则的数组存取更是这样.

在编译系统构造中强调运行时支持，从本质上说是分而治之思想的体现，它将编译系统分成很明确的两大部分——编译时和运行时，这对于实现一种新语言的编译系统有着现实的意义. 而后，可通过"inline"等方式逐步将运行时常用的函数直接嵌入到结点程序中来，以降低函数调用的开销，进而向高级的编译优化过渡.

致谢　本 HPF 编译系统是由3个单位联合开发的，另外两个单位是哈尔滨工业大学和美国的 Syracuse 大学. 在此，谨向这两个单位的同仁表示感谢.

参考文献

1 High Performance Fortran Forum. High Performance Fortran Language Specification. Version 1.0. Rice University, Houston Texas, 1993

2 Li Xiao-ming. HPFfe: a front-end for HPF. Technical Report, SCCS-771. Northeast Parallel Architectures Center, 1996

3 Gannon D. A class library for building Fortran 90 and C++ restructuring tools. Nov. 1993. http://www.extreme.indiana.edu/sage/index.html

4 Carpenter B, Fox G, Leskiw D *et al*. PCRC runtime interface. Technical Report, SCCS-799. Northeast Parallel Architectures Center, 1996

5 Carpenter B, Cowie J, Leskiw D *et al*. The distributed array descriptor for a PCRC HPF compiler, Version 2.0. Technical Report, SCCS-770d. Northeast Parallel Architectures Center, 1997

6 Li Xiao-ming. Runtime Oriented HPF Compilation. Technical Report, CRPC-TR97694. Center for Research on Parallel Computation, 1997

7 郑耿斌. PACT HPF 标准数据映射语句的定义. 技术报告，1996
(Zheng Geng-bin. The definition of PACT standard HPF data mapping statement. Technical Report, Beijing University, 1996. http://pact0.cs.pku.edu.cn/pact/TR/index.html)

8 都志辉. HPF 并行语句及结构的标量化算法. 技术报告，1996
(Du Zhi-hui. The algorithm of how to scalarize the HPF parallel statement or construct. Technical Report, Beijing University, 1996. http://pact0.cs.pku.edu.cn/pact/TR/index.html)

9 都志辉，汪剑平. PACT HPF 子集的定义，技术报告，1997
(Du Zhi-hui, Wang Jian-ping. The definition of PACT HPF subset. Technical Report, Beijing University, 1997. http://pact0.cs.pku.edu.cn/pact/TR/index.html)

10 都志辉，汪剑平，向华. Laplace 例子的测试结果. 技术报告，1997
(Du Zhi-hui, Wang Jian-ping, Xiang Hua. The testing results of Laplace resolver. Technical Report, Beijing University, 1997. http://pact0.cs.pku.edu.cn/pact/TR/index.html)

11 都志辉. HPF 程序与手写结点程序的加速比. 技术报告，1997
(Du Zhi-hui. The speedup of HPF program and handcoded MPI program. Technical Report, Beijing University, 1997. http://pact0.cs.pku.edu.cn/pact/TR/index.html)

12 Tseng Chau-wen. An optimizing Fortran D compiler for MIMD distributed memory machines [Ph. D. Dissertation]. Rice University, 1993

13 Kees van Reeuwijk, Denissen W, Sips H J *et al*. Paalvast, an implementation framework for HPF distributed arrays on message-passing parallel computer systems. IEEE Transactions on Parallel and Distributed System, 1996, 7(9): 897～914

Research and Implementation of an HPF Compilation System

DU Zhi-hui　DING Wen-kui　ZHENG Geng-bin　LI Xiao-ming　XU Zhuo-qun

(*Department of Computer Science and Technology　Beijing University　Beijing*　100871)

Abstract　An operational HPF (high performance Fortran) compilation system is introduced in this paper. An overview of the system organization and the key techniques employed in implementing the system, such as communication detection, runtime supporting, are described in detail. To serve as concrete examples, some code fragments generated by the compiler are also included for given HPF source. Finally, some performance data are displayed, followed by a conclusion.

Key words　HPF (high performance Fortran), parallel compiler, runtime, data parallel, collective communication.