第33卷 第23期 计 算 机 工 程 2007年12月
Vol.33 No.23 Computer Engineering December 2007
·软件技术与数据库· 文章编号：1000—3428(2007)23—0084—03 文献标识码：A 中图分类号：TP391

Co-array Fortran 编译器的设计与实现

唐沛蓉，黄 春，杨学军，王桂彬
(国防科技大学计算机学院，长沙 410073)

摘 要：基于 GUN Fortran 编译器，设计并实现了 co-array Fortran(CAF)编译器。通过源到源的转换将 CAF 代码转换为带有运行库调用的 Fortran 90 程序。典型用例的测试表明，CAF 具有较好的可编程性，且 CAF 程序通过对数据分布的显式控制可获得比 OpenMP 程序更为高效的执行性能。

关键词：co-array Fortran；映像；源到源转换

Design and Implementation of Co-array Fortran Compiler

TANG Pei-rong, HUANG Chun, YANG Xun-jun, WANG Gui-bin
(School of Computer Science, National University of Defense Technology, Changsha 410073)

【Abstract】 This paper presents the design and implementation of the co-array Fortran(CAF)compiler based on GNU Fortran compiler. The CAF compiler translates CAF into Fortran 90 plus calls to runtime library. Typical tests are performed and the results show that CAF has better scalability. CAF gains better performance than OpenMP due to the explicit control on shared data distribution.

【Key words】 co-array Fortran(CAF); image; source-to-source translation

在目前流行的并行计算编程模型中，OpenMP和MPI应用最为广泛。但这两种模型仍存在缺陷[1]：OpenMP编程简单，却只适合于一致性访存的共享存储系统，可扩展性受限；MPI实现的SPMD的并行处理适合于大规模的分布存储系统，可扩展性高，但编程复杂。

新一代的并行程序设计接口CAF[2]既适合共享存储系统又适合分布式存储系统。它将MPI中复杂的通信进行抽象，通过使用赋值语句对全局编址的存储空间直接进行存取实现数据的隐式通信。这种比库调用更为清晰的通信表达方式，大大降低了编程复杂度。为了使CAF获得实际应用，本文在GNU Fortran 编译器的基础上设计实现了CAF编译器并通过程序实例对其进行了性能测试。

1 Co-array Fortran

CAF在标准Fortran上进行了最小的语法扩展以支持并行编程。它采用分割全局地址空间的SPMD编程模型[3]。CAF程序在执行时，代码被复制成*n*份，一个副本为一个映像，各个映像将全局地址空间分割，每个映像拥有自己的独立的地址空间，存放各自的数据。CAF提供了两极存储模型：数据显式地分为本地数据和远程数据。这样，CAF程序可以通过对数据和计算的显式控制更好地开发程序的局部性。

CAF 引入了一种全局可见的共享数据——co-array 的新对象。co-array 的定义方式为普通的变量定义后加上方括号。例如，语句：real::x(n)[*]声明了 co-array 数组 *x*，它由每个映像上的一个同名局部对象组成，每个局部对象都是一个大小为 *n* 的一维实型数组，拥有独立的存储空间，可以具有不同的值。方括号中的维度信息称为 co-dimension，方括号定义的数组的维数称为 co-rank。程序通过 co-array 对象方括号中的索引来区分访问不同映像上同一 co-array 的局部对象，即进行远程数据访问。如果访问没有方括号的 co-array，则为引用 co-array 在当前映像上的本地对象。例如，程序中有如下语句：x(:)=y(:)[1]则该语句会被所有的映像执行，并且每个映像都将 1 号映像上的数组 *y* 的值复制到本地数组 *x*。co-array 方括号中的索引也可为多维，如语句：real::x(n)[k,*]将映像逻辑上组织成 *k* 行(提交给 j3 组织的新规范禁止了这一点)。

此外，CAF 引入了同步控制等固有过程，为 SPMD 编程模型下的并行程序开发提供有力支持。

2 Co-array Fortran 编译器的实现

本文的 CAF 编译器采用源到源转换的方式，将 CAF 代码翻译为带有运行库调用的扩展的 Fortran90 代码。源到源的代码产生策略使得用户可以用商用 Fortran 编译器对本地代码进行最有效的优化。

2.1 编译器结构

本文的 CAF 编译器在 GNU Fortran 编译器上实现，对 GNU Fortran 编译器前端进行了适当修改使它识别和处理 CAF 语法，如图 1。该修改过程是：(1)对语法分析器进行扩充，使其能够正确识别 CAF 语法，生成扩展的 Fortran 抽象语法树；(2)遍历扩展的抽象语法树，由 caf2f 转换器对含有 CAF 语法的语句节点进行转换，得到标准 Fortran 抽象语法树，并据此生成带有运行库调用的标准 Fortran 中间文件；(3)交给编译器后端(或其他标准 Fortran 串行编译器)进行优化；(4)链接运行库，得到可执行代码。源到源转换后得到的带运行库调用的标准 Fortran 文件，可以作为目前已有的各种标准 Fortran 编译器的输入文件。这就可以利用已有的串行编译器对本地代码进行最有效的优化。

作者简介：唐沛蓉(1982－)，女，硕士研究生，主研方向：并行编译；黄 春，博士研究生；杨学军，教授；王桂彬，硕士研究生

收稿日期：2006-12-25 **E-mail：**serein_0719@yahoo.com.cn

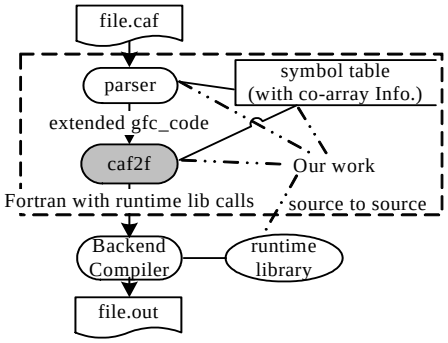

图 1　CAF 编译器结构

2.2　Co-array 数据的表示

2.2.1　Co-array 描述符

为了处理跨映像的数据访问和实现固有函数，需要对 co-array 进行描述。描述 co-array 的数据结构称为 co-array 描述符，其类型为自定义的导出类型 co_array：

每个 co-array 对应一个描述符，记录它的 co-dimension 信息。其中，dims 记录 co-rank 信息；lower, upper, stride 分别记录每一维的上界、下界及跳步。CAF 编译器在源到源转换时插入必要的代码初始化这些描述符。

2.2.2　Co-array 数据声明

在对输入文件的数据声明部分进行扫描时，需要词法语法分析器正确识别 co-array 数据声明，同时收集并记录这些数据的符号及维度信息。本文对 GNU Fortran 编译器的符号表结构进行扩充，为 co-array 符号添加记录其维度信息的结构成员。

2.2.3　Co-array 数据访问

对 co-array 数据的访问有 2 种方式：对本地 co-array 数据的访问和对远程 co-array 数据的访问。本地 co-array 数据即不带方括号的 co-array 数据，存放在各个映像的本地存储空间中，因此，各个映像可以直接对自己本地的 co-array 进行操作；对远程数据即带有中括号的数据进行访问时，会发生跨映像的数据访问，编译器需要生成适当的指令进行映像之间的通信。

对远程 co-array 数据进行访问时通信库调用实际转换成 Fortran 90 的形式，但是由于远程存储器可能使用的是不同的地址空间，因此这个转换并不能直接进行。为了表示出数据在各映像上的运转，编译器要产生一些通信库调用来对远程数据进行存取，同时，还需要在本地申请临时空间对存储这些数据。例如，对语句 a(:)=c(:)[p]+b 进行处理的步骤如图 2(a)所示：(1)分配一个临时空间 temp 来存放从 p 号映像上取来的 c 数组的数据；(2)调用运行库进行通信，读取远程数据；(3)将语句改写为 a(:)=temp+b，进行计算；(4)完成计算并释放临时空间。向远程数据进行写操作也是类似的，如对语句 c(:)[p]=a(:)+b 的处理如图 2(b)：(1)申请临时空间 temp，改写原语句为 temp=a(:)+b；(2)计算后通过运行库调用对 p 号映像上的 c 数组执行写操作；(3)释放临时空间。

```
parameter (MAX_DIMENSION=7)
type co_array
integer :: dims
integer :: lower(MAX_DIMENSION)
integer :: upper(MAX_DIMENSION)
integer :: stride(MAX_DIMENSION)
end type
```

```
a(:)=c(:)[p]+b  ⟶  allocate (temp(n))
                   call coarray_get(temp,n,c,n,p)
                   a(:) = temp + b
                   deallocate (temp)
```

(a)读远程数据

```
c(:)[p]=a(:)+b  ⟶  allocate (temp(n))
                   temp = a(:)+b
                   call coarray_put(temp,n,c,n,p)
                   deallocate (temp)
```

(b)写远程数据

图 2　Co-array 数据访问代码转换

为了减少临时空间的申请释放带来的开销，应尽可能地重用已有的临时缓存区，在不必要的情况下避免使用临时缓存区。本编译器在处理远程数组访问时，若编译时不可确定数组的大小(如 a(1:n)[p])，则选用动态数组来做临时缓存区；若编译时可以确定数组的大小(如：a(1:3), b)，则在源到源转换时生成静态数组作为缓冲区，减少临时空间的申请、释放。

2.3　参数传递

CAF允许将co-array作为子程序和函数的参数。在CAF程序中co-array有 2 种参数传递方式：传值和传co-array[4]。

如果过程的形参的 co-rank 为 0，则使用传值的方式来进行参数传递。过程调用语句中，实参为带有中括号的 co-array，如 call foo (a(1:n, k)[p])。这种情况下，CAF 编译器首先在本地申请临时空间 temp，读取远程 co-array 数据片段 a(1:n, k)[p]；然后将 temp 作为实参调用过程 foo；最后，过程返回，释放临时空间。

如果过程的形参的 co-rank 不为 0，在过程调用时则使用传 co-array 的参数传递方式。过程调用语句形式如 call foo(c(i, k))。传 co-array 有着和传统 Fortran 的传引用参数传递方式类似的语义：仅将 c(I, k)的本地地址传给了子例程。过程通过显式的接口定义形参的属性，每个 co-array 形参都显式声明了形状以便进行语法检查。实参必需是 co-array 名或者不带方括号、向量值下标或指针成员的 co-array 子对象。在传 co-array 的参数传递方式中传递一个 co-array 片段(如 c(i, k)[n])是不合法的，因为这可能会导致 copy-in-copy-out 语义，扰乱过程调用中的存储一致性。

2.4　多维支持

CAF支持多维co-array。程序员可以类似声明传统的多维Fortran数组一样声明多维co-array数组。在调用运行库对远程数据进行访问时，需要将方括号中的多维索引转换成其对应的映像号。假设一个co-array数据的远程索引为$[i_1,\cdots,i_n, i_{n+1}]$，根据这个co-array数据定义时的维度信息$[l_1: u_1,\cdots, l_n: u_n, l_{n+1}: u_{n+1}]$，可以计算出这个索引对应的映像号，进而对目标映像上的相关数据进行存取操作。计算映像号的公式为

$$\sum_{j=1}^{n+1}(i_j - L_j) * m_j$$

其中，$m_1=1$，$m_j=\prod_{k=1}^{j-1}(u_k-l_k+1)$，$2\le j\le n+1$。

多维的 co-array 能够非常自然、清晰地映射出数据的空间分布情况。对于要将处理器空间映射到网格的这类问题，使用多维 co-array 非常方便、有效。程序员可以使数组 co-dimension 上的各维与处理器的逻辑网络的各维相对应，这样就可以直观、灵活地通过索引实现对各远程映像上数据的组织与计算。远程数据方括号中的多维索引到映像号的映射由编译器自动完成，从而降低了编程的复杂性。

2.5 同步

同步控制成为了 CAF 语言的一部分，如 sync_all 和 sync_team。程序员可以通过指定 sync_all(wait)中的 wait 参数与指定的映像同步，也可以使用过程 sync_team 实现映像子集间的同步。在全局同步不能由硬件直接支持的平台上，这种局部同步的方式可以带来一定的性能提升。这些同步控制都是通过运行库调用通信库操作实现的。

2.6 单边通信

CAF 中对远程数据存取需要选用单边通信的方式实现，单边通信方式不需要通信双方的协作，数据传递操作的各个参数均由通信的一方确定，并由其完成对远程数据的存取。单边通信将数据的运转从同步控制中分离出来，避免了消息传递方式中的双边握手过程，提高了程序执行性能。

使用双线程的方式实现单边通信。每个映像包括两个线程，一个计算线程，负责执行程序，完成本映像上除通信外的任务操作；另一个通信线程，负责处理通信事务。程序启动后，这两个线程被创建，计算线程执行程序，通信线程则进入睡眠状态直到发生通信事件。如图 3 所示，映像 1 执行赋值语句 B=A[2]，需要读取映像 2 上的 A 数据。因此，映像 1 的计算线程唤醒本地的和映像 2 的通信线程；映像 2 的通信线程被唤醒后，组织数据，将数据 A 放入共享空间，然后置标志位；在此期间，映像 1 的通信线程反复查询标志位，一旦标志位被置上，立即从共享空间中取出数据存储到本地临时缓冲区，操作完毕后返回。此过程中，映像 2 上的计算线程无需中断执行，此次通信对映像 2 的计算线程是透明的，从而实现了单边通信。

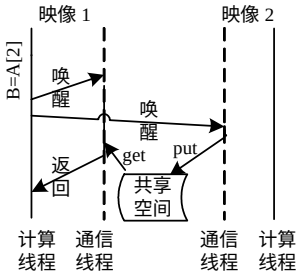

图 3　单边通信的实现

3　测试

本文结合数值计算中两个常见用例测试 CAF 程序的实际执行性能。选用 Intel 编译器作为后端编译器，在有两个 CPU 的安腾芯片上对 CAF 和 OpenMP 版本的辛普森求积及正方体网格上的有限差分程序进行测试，比较程序获得的加速比。

3.1 辛普森求积

本文使用辛普森求积公式对初等函数 $f(x)=6x^5$ 进行积分求值计算。辛普森公式为

$$\int_{x_0}^{x_2} f(x)\mathrm{d}x \approx \frac{h}{3}(f_0+4f_1+f_2)$$

其中，$h=(x_2-x_0)/2$。

分别用 CAF 和 OpenMP Fortran 编制程序，图 4 给出了这两个程序取得的加速比。可以看到，二者取得了非常近似的加速比，并且，随着迭代次数的增长 CAF 程序获得的加速比逐渐增大。

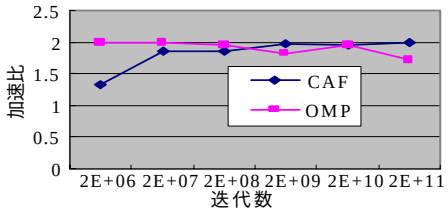

图 4　安腾上的辛普森性能

3.2 正方体网格上的有限差分

假设有一个边界条件周期性变化的矩形网格，在网格的每一个点上使用 5-点近似拉普拉斯算子(将所有相邻点的值加起来，再减去本点值的 4 倍)计算。假定网格的大小为 $r*c$，在 CAF 编写的程序中，用 co-array 数组 $u(r)[c]$来记录网格各节点的数据。将网格按列划分给 c 个映像，第 i 列元素存储在 i 号映像的数据区，并由 i 号映像完成此列数据的计算更新。

在用OpenMP编写的程序中，数组u声明为全局共享的二维数组，各线程仍按列分工进行计算。在$r*2$ 的网格上进行计算，取得这两种程序的加速比，如图 5。可以看出，当行数为 10^8时，CAF程序的加速比陡增。这是因为OpenMP程序中，各线程是对二维共享数组进行操作，当数据量超出了内存容量时，由于需要将数据交换到外存，从而引入大量开销。而CAF程序对数据进行了划分，各映像仅需对存储在本地的co-array局部对象进行操作。各映像需处理的局部数据并未超出其所在节点的内存容量，因而，避免了访问外存的操作，使得CAF程序获得了可观的加速比。

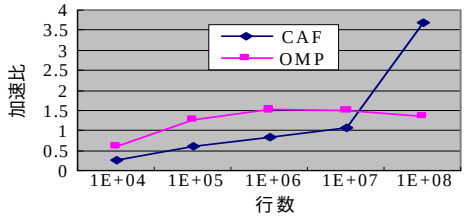

图 5　安腾上的 Laplace 性能

4　结束语

CAF 编程简单、可扩展性高，是一种既适合共享主存系统，又适合分布主存系统的并行编程接口。通过提供对数据的存放、计算以及通信的控制，可以用 CAF 来手工编写局部敏感的并行程序。这样编写的程序拥有比 OpenMP 更好的局部控制，也使得 CAF 程序的执行存在更高的性能潜能。

参考文献

1 唐沛蓉, 黄　春, 杨学军. Co-array Fortran & OpenMP Fortran 研究与比较[C]//计算机体系结构学术年会, 2006.

2 Numrich R W, Reid J. Co-array Fortran for Parallel Programming[J]. ACM Fortran Forum, 1998, 17(2): 1-31.

3 Coarfa C, Dotsenko Y, Mellor-Crummey J. An Evaluation of Global Address Space Languages: Co-array Fortran and Unified Parallel[C]//Proc. of the 10th ACM SIGPLAN Sym. on Principles and Practice of Parallel Programming. 2005: 15-17.

4 Dotsenko Y. Coarfa C, Mellor-Crummey J. A Multiplatform Co-array Fortran Compiler[C]//Proceedings of the 13th Int'l Conference of Parallel Architectures and Compilation Techniques, Antibes Juan-les-Pins, France. 2004.